Network Firewalls

Computer security is a hard problem. Security on networked computers is much harder. Firewalls (barriers between two networks), when used properly, can provide a significant increase in computer security.

Steven M. Bellovin and William R. Cheswick

Computer security is a hard problem. Security on networked computers is much harder. The administrator of a single host can — with a great deal of care and attention to details, luck in the choice of vendor software, and a careful and educated user community — probably do an adequate job of keeping the machine secure. But if the machine is connected to a network, the situation is much difficult.

First, many more entry points to the host than a simple `login` prompt must be secured. The mailer, the networked file system, and the database servers are all potential sources of danger. Furthermore, the authentication used by some protocols may be inadequate. Nevertheless, they must be run, to provide adequate service to local users.

Second, there are now many more points from which an attack can be launched. If a computer's users are confined to a single building, it is difficult for an outsider to try to penetrate system security. A network-connected computer, on the other hand, can be reached from any point on the network — and the Internet reaches tens of millions of users in every part of the globe.

Finally, networks expose computers to the problem of transitive trust. Your computers may be secure, but you may have users who connect from other machines that are less secure. This connection — even if duly authorized and immune to direct attack — may nevertheless be the vehicle for a successful penetration of your machines, if the source of the connection has been compromised.

The usual solution to all of these problems is a firewall: a barrier that restricts the free flow of data between the inside and the outside. Used properly, a firewall can provide a significant increase in computer security.

Stance

A key decision when developing a security policy is the stance of the firewall design. The stance is the attitude of the designers. It is determined by the cost of failure of the firewall and the designers' estimate of that likelihood. It is also based on the designers' opinions of their own abilities. At one end of the scale is a philosophy that says, "we'll run it unless you can show me that it's broken." People at the other end say, "show me that it's both safe and necessary; otherwise, we won't run it." Those who are completely off the scale prefer to pull the plug on the network, rather than take any risks at all. Such a move is too extreme, but understandable. Why would a company risk losing its secrets for the benefits of network connection?

We do not advocate disconnection for most sites. Our philosophy is simple: there are no absolutes. One cannot have complete safety; to pursue that chimera is to ignore the costs of the pursuit. Networks and internetworks have advantages; to disconnect from a network is to deny oneself those advantages. When all is said and done, disconnection may be the right choice, but it is a decision that can only be made by weighing the risks against the benefits.

We advocate caution, not hysteria. For reasons that are spelled out below, we feel that firewalls are an important tool that can minimize the danger, while providing most — but not necessarily all — of the benefits of a network connection. However, a paranoid stance is necessary for many sites when setting up a firewall.

Most computing professionals realize that most large software systems are buggy. If the system is security-sensitive — that is, if it provides any sort of network service at all — one runs the risk that the bugs will manifest themselves as security holes. The most practical solution is to run as few programs as possible, and to make sure that these are as small and simple as possible. A firewall can do this. It is not constrained to offer general computing services to a general user population. It need not run networked file systems, distributed user name databases, etc. The very act of eliminating such programs automatically makes a firewall more secure than the average host.

We also feel that any program, no matter how innocuous it seems, can harbor security holes. (Who would have guessed that on some machines, integer divide exceptions could lead to system penetrations?) We thus have a firm belief that everything is guilty until proven innocent. Consequently, we configure our firewalls to reject everything, unless we have explicitly made the choice — and accepted the risk — to permit it. Taking the opposite tack, of blocking only known offenders, strikes us as extremely dangerous.

STEVEN M. BELLOVIN works at AT&T Bell Laboratories, where he does research in networks and security

WILLIAM R. CHESWICK serves as an assistant programmer trainee and member of the technical staff at Bell Laboratories.

Much of this article was taken from "Firewalls and Internet Security: Repelling the Wiley Hacker" by William R. Cheswick and Steven M. Bellovin, AddisonWesley Publishing Company, ISBN 0-201-63357-4,

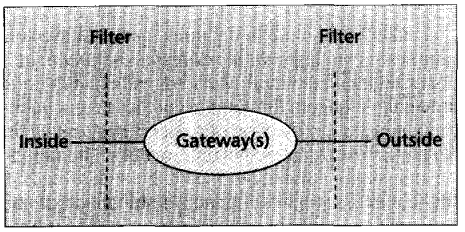

Figure 1. *Schematic of a firewall.*

Furthermore, whether or not a security policy is formally spelled out, one always exists. If nothing else is said or implemented, the default policy is "anything goes." Needless to say, this stance is rarely acceptable in a security-conscious environment. If one does not make explicit decisions, one will have made the default decision to allow almost anything.

Host Security

To some people, the very notion of a firewall is anathema. In most situations, the network is not the resource at risk; rather, the endpoints of the network are threatened. By analogy, con artists rarely steal phone service per se; instead, they use the phone system as a tool to reach their real victims. So it is, in a sense, with network security. Given that the target of the attackers is the hosts on the network, should they not be suitably configured and armored to resist attack?

The answer is that they should be, but probably cannot. Such attempts are probably futile. There will be bugs, either in the network programs or in the administration of the system. It is this way with computer security: the attacker only has to win once. It does not matter how thick are your walls, nor how lofty your battlements; if an attacker finds one weakness — say, a postern gate, to extend our metaphor — your system will be penetrated. And if one machine falls, its neighbors are likely to follow.

Types of Firewalls

We define a firewall as a collection of components placed between two networks that collectively have the following properties:

- All traffic from inside to outside, and vice-versa, must pass through the firewall.
- Only authorized traffic, as defined by the local security policy, will be allowed to pass.
- The firewall itself is immune to penetration.

We should note that these are design goals; a failure in one aspect does not mean that the collection is not a firewall, simply that it is not a very good one.

That firewalls are desirable follows directly from our earlier statements. Many hosts — and more likely, most hosts — cannot protect themselves against a determined attack. Firewalls have several distinct advantages.

First, of course, a firewall is likely to be more secure than an average host. The biggest single reason for that is simply that it is not a general-purpose machine. Thus, features that are of doubtful security but add greatly to user convenience — Network Information Service (NIS), `rlogin`, etc. — are not necessary. For that matter, many features of unknown security can be omitted if they are irrelevant to the firewall's functionality.

A second benefit comes from having professional administration of the firewall machines. We do not claim that firewall administrators are necessarily more competent than your average system administrator, but they may be more security conscious. However, they are almost certainly better than nonadministrators who must nevertheless tend to their own machines. This category would include physical scientists, professors, etc., who (rightly) prefer to worry about their own areas of responsibility. It may or may not be reasonable to demand more security consciousness from them; nevertheless, it is obviously not their top priority.

Fewer normal users is a help as well. Poorly chosen passwords are a serious risk; if users and their attendant passwords do not exist, this is not a problem. Similarly, one can make more or less arbitrary changes to various program interfaces if that would help security, without annoying a population accustomed to a different way of doing things. One example would be the use of hand-held authenticators for logging in. Many people resent them, or they may be too expensive to be furnished to an entire organization; a gateway machine, however, should have a user community that is restricted enough so that these concerns are negligible.

More subtly, gateway machines need not, and should not, be trusted by any other machines. Thus, even if the gateway machine has been compromised, no others will fall automatically. On the other hand, the gateway machine can, if the user wishes (and decides against using hand-held authenticators), trust other machines, thereby eliminating the need for most passwords on the few accounts it should have. Again, something that is not there cannot be compromised.

Gateway machines have other, nonsecurity advantages as well. They are a central point for mail and FTP administration, for example. Only one machine need be monitored for delayed mail, proper header syntax, return-address rewriting (i.e., to `firstname.lastname@org.domain` format), etc. Outsiders have a single point of contact for mail problems and a single location to search for files being exported.

Our main focus, though, is security. And for all that we have stated about the benefits of a firewall, it should be stressed that we neither advocate nor condone sloppy attitudes toward host security. Even if a firewall were impermeable, and even if the administrators and operators never made any mistakes, the Internet is not the only source of danger. Apart from the risk of insider attacks and in some environments, that is a serious risk — an outsider can gain access by other means. In at least one case, a hacker came in through a modem pool, and attacked the firewall from the inside [7]. Strong host security policies are a necessity, not a luxury. For that matter, internal firewalls are a good idea, to protect very sensitive portions of organizational networks.

A firewall, in general, consists of several different components (Fig. 1). The "filters" (sometimes called "screens") block transmission of certain classes of traffic. A gateway is a machine or a set of machines that provides relay services to compensate for the effects of the filter. The network inhabited by the gateway is often called the demilitarized zone (DMZ). A gateway in the DMZ is sometimes assisted by an internal gateway. Typically, the two gateways will have more open communication through the inside filter than the outside gateway has to other internal hosts. Either filter, or for that matter the gateway itself, may be omitted; the details will vary from firewall to firewall. In general, the outside filter can be used to protect the gateway from attack, while the inside filter is used

Everything is guilty until proven innocent. Thus, we configure our firewalls to reject everything, unless we have explicitly made the choice — and accepted the risk — to permit it.

Even authorized users should pass through a security gateway when crossing the firewall; otherwise, if their home machines are compromised, the equipment on the inside could be next.

to guard against the consequences of a compromised gateway. Either or both filters can protect the internal network from assaults. An exposed gateway machine is often called a bastion host.

We classify firewalls into three main categories: packet filtering, circuit gateways, and application gateways. Commonly, more than one of these is used at the same time. As noted earlier, mail is often routed through a gateway even when no security firewall is used.

Our examples and discussion unabashedly relate to UNIX systems and programs. The majority of multiuser machines on the Internet run some version of the UNIX operating system. Most application-level gateways are implemented in UNIX. This is not to say that other operating systems are more secure; however, there are fewer of them on the Internet, and they are less popular as targets for that reason. But the principles and philosophy apply to network gateways built on other operating systems as well.

Our focus is on the TCP/IP protocol suite, especially as used on the Internet. Again, this is not because TCP/IP has more security problems than other protocol stacks (we doubt that very much), rather, it is a commentary on the success of TCP/IP. By far, it is the heterogeneous networking protocol of choice — not only on workstations, for which it is the native tongue — but on virtually all machines, ranging from desktop personal computers to the largest supercomputers. Many internal corporate networks are based on TCP/IP; some — but not all — of these are connected to the Internet. And the Internet links most major universities in the United States (and many others around the world), research labs, many government agencies, and even a fair number of businesses. We believe, though, that our advice is applicable to any network with similar characteristics. We have read of serious attacks on computers attached to public X.25 data networks. Firewalls are useful there, too, although naturally they would differ in detail.

Traditionally, firewalls are placed between an organization and the outside world. But a large organization may need internal firewalls as well to isolate security domains (also known as administrative domains). A security domain is a set of machines under common administrative control, with a common security policy and security level.

There are many good reasons to erect internal firewalls. In many large companies, most employees are not (or should not be) privy to all information. In other companies, the cash business (like the factory, or a phone company's telephone switches) needs to be accessible to developers or support personnel, but not to the general corporate population. Even authorized users should pass through a security gateway when crossing the firewall; otherwise, if their home machines, which live outside of the firewall, are compromised, the sensitive equipment on the inside could be next. The firewall controls the access and the trust in a carefully predictable way.

Packet-Filtering Gateways

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. Since 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, you will probably find that they will install any filters you wish.

Packet filters work by dropping packets based on their source or destination addresses or service (i.e., port number). In general, no context is kept; decisions are made only from the contents of the current packet. Depending on the type of router, filtering may be done at input time, at output time, 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.

Most security policies require finer control than this; they need to define access to specific services for hosts that are otherwise untrusted. 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 disadvantages of packet filters: if you get these tables wrong you may inadvertently let in the Bad Guys [5]. But even with a perfectly implemented filter, some compromises can be dangerous. We discuss these in a section to follow.

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. Next, 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 was to allow inbound mail (SMTP, port 25), but only to your gateway machine. However, mail from some particular site SPIGOT is to be blocked, because of their penchant for trying to mail several gigabytes of data at a time. A filter that implemented such a ruleset might look like ruleset A in the text box on the following page.

The rules are applied in order from top to bottom. The "*" in a field matches anything. Packets not explicitly allowed by a filter rule are rejected, i.e., every ruleset is followed by an implicit rule reading like ruleset B in the textbox above. This fits with our general philosophy: all that is not expressly permitted is prohibited.

Note carefully the distinction between ruleset A and ruleset C, which is intended to implement the policy "any inside host can send mail to the outside."

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 call from port 25 on the outside machine.

A better rule would be to permit outgoing calls to port 25, i.e., 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

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

B

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

C

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

D

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

E

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

A TCP conversation consists of packets flowing in two directions. Even if all the data is flowing one way, acknowledgment packets and control packets must flow the other way.

A TCP conversation consists of packets flowing in two directions [19]. 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 set in the header; all other TCP packets do. 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 seen in ruleset D, keying our rules by the source and destination fields, rather than the more nebulous "OURHOST" and "THEIRHOST":

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 you could specify a group of machines, probably by the network number portion of the IP address.

Filtering FTP Sessions

Some services are not handled well by packet filters. We use the File Transfer Protocol (FTP) [20] as an example here; other problematic protocols include X11 and the Domain Name System (DNS) [12, 16, 17, 23].

For FTP, files are transferred via a secondary connection. If the control channel to a server on THEIRHOST uses the connection

```
(ourhost, ourport, theirhost, 21),
```

file transfers will occur on

```
(ourhost, ourport, theirhost, 20)
```

by default. Furthermore, the server must initiate the file transfer call. We thus have the problem we saw earlier, but without the ability to screen based on the direction of the call.

One idea is to use the range of ourport to make filtering decisions. Most servers, and hence most attack targets, live on low-numbered ports; most outgoing calls tend to use higher numbered ports, typically above 1023. Thus, a sample ruleset might be ruleset E in the text box, where packets are passed under one of three circumstances:

- They originated from one of our machines.
- They are reply packets to a connection initiated by one of our machines.
- They are destined for a high-numbered port on our machines.

Actually, the last two rules apply to all packets, not just packets originating from outside. But any packets from the inside would be accepted by the first rule, and would not be examined by the later rules.

Unfortunately, this ruleset does not accomplish what we really want, which is to block incoming calls to our servers. We said "most servers" live on low-numbered ports, not "all." A number of tempting targets, especially X11, inhabit high-numbered ports. Presumably, one could filter out known dangerous ports; unfortunately, new ones could be added without notice. Thus, a cautious stance dictates that this heuristic not be adopted.

Under certain circumstances, a bypass is available if you have the source code to the FTP client programs. You can modify the programs to issue a PASV command to the server, directing it to do a passive open, and thus permitting an outgoing call through the firewall for the data channel.

This variant is not without its problems. The data channel, though an outgoing call, is to a random port. Such calls are generally barred by sites that wish to restrict outbound data flow. You also have the obvious problem of distributing modified clients to all inside machines. Also, not all servers understand the PASV command, even though they should. The issues are discussed further in [3].

Protocols Without Fixed Addresses

Some services are problematic for packet filters because they can involve random port numbers. On occasion the situation is even worse: a number of services always use random port numbers, and rely on a separate server to supply the current contact information.

Two examples of this are the tcpmux protocol [13] and the portmapper [26] used by SunOS for RPC

Filtering TCP circuits is difficult. Filtering UDP packets while still retaining desired functionality is all but impossible.

[25]. In both cases, client programs contact the mapping program rather than the application. The portmapper also processes registration requests from applications, informing it of their currentport numbers. On the other hand, tcpmux will invoke the application directly, passing it the open connection.

This difference gives rise to different filter-based protection mechanisms. With `tcpmux`, one can block access to either all such services, or none, simply by controlling access to the `tcpmux` port. With the portmapper, each service has its own port number. While one can deny easy access to them by filtering out portmapper requests, an intruder can bypass the portmapper and simply sweep the port number space looking for interesting applications. We have seen evidence of this happening. The only cure is to block access to all possible port numbers used by RPC-based servers — and there is no easy way to know what that range is.

Packet Filters and UDP

Filtering TCP circuits is difficult. Filtering UDP packets [18] 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 illustrates the problem. Suppose an internal host wishes to query the UDP echo server on some outside machine. The originating packet would carry the address

```
(localhost, localport, remotehost, 7),
```

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

```
(remotehost, 7, localhost, localport),
```

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

```
(remotehost, 7, localhost, 2049),
```

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 conservative stance therefore 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 where there is a peer-to-peer relationship. A good example is the Network Time Protocol (NTP) [15]. 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. But 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 should not ask an outsider for the time.)

Typical Configurations

We cannot provide readers with the exact packet filter for a particular site, because we do not know what its policies are. But we can give some reasonable samples that may serve as a starting point.

Universities tend 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. Also, there might be a PC lab in a dorm that has been the source of some trouble, so they do not allow that lab access the Internet. (The users have to go through one of the main systems that require an account, which gives some accountability.) Finally, there is to be no access to the administrative computers except for access to a transcript manager. That service should use strong authentication and encryption.

On the other hand, a small company 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. Only access to that machine, and to the necessary services, should be permitted.

Application-Level Gateways

An application-level gateway represents the opposite extreme in firewall design. 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. Although this seems wasteful, it is likely to be far more secure than any of the alternatives. One need not worry about interactions among different sets of filter rules, nor about holes in thousands of hosts offering nominally secure services to the outside. Only a chosen few programs need to be scrutinized.

Application gateways have another advantage that in some environments is quite critical: it is easy to log and control all incoming and outgoing traffic. The SEAL package [21] from Digital Equipment Corporation takes advantage of this. Outbound FTP traffic is restricted to authorized individuals, and the effective bandwidth is limited. The intent is to prevent theft of valuable company programs and data. While of limited utility against insiders, who could easily dump the desired files to tapes or floppies, it is a powerful weapon against electronic intruders who lack physical access.

Electronic mail is often 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. The gateway machines also worry about mail header formats and logging (mail logging is a postmaster's friend) and provide a centralized point for monitoring the behavior of the electronic mail system.

It is equally valuable to route incoming mail

through a gateway. One person can be aware of all internal connectivity problems, rather than leaving it to hundreds of random system administrators around the net. Reasonably constant mail addresses can be accepted and processed. Different technologies, such as uucp, can be used to deliver mail internally. Indeed, the need for incoming mail gateways is so obvious that the DNS has a special feature — MX records — defined to support them. No other application has a defined mechanism for indirect access.

These features are even more valuable from a security perspective. Internal machine names can be stripped off, hiding possibly valuable data. Traffic analysis and even content analysis and recording can be performed to look for information leaks. But these abilities should be used with the utmost reluctance, for both legal and ethical reasons.

Application gateways are often used in conjunction with the other gateway designs, packet filters and circuit-level relays. An application gateway can be used to pass X11 through a firewall with reasonable security [27]. The semantic knowledge inherent in the design of an application gateway can be used in more sophisticated fashions. Gopher servers [1] can specify that a file is in the format used by the uuencode program. But that format includes a file name and mode. A clever gateway could examine or even rewrite this line, thus blocking attempts to force the installation of bogus .rhosts files or shells with the setuid bit turned on.

The type of filtering used depends on local needs and customs. A location with many PC users might wish to scan incoming files for viruses.

We note that the mechanisms described herein are intended to guard against attack from the outside. A clever insider who wanted to retrieve such 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. This may not be entirely bad — again, programs that you do not run cannot hurt you — but it does make it harder to adopt newer technologies. Also, use of such gateways is easiest with applications that make provision for redirection, such as mail and X11. Otherwise, new client programs must be provided.

Circuit-Level Gateways

The third type of gateway — our preference for outgoing connections — is circuit level. Circuit gateways relay TCP connections. The caller connects to a TCP port on the gateway, which connects to some destination on the other side of the gateway. During the call the gateway's relay program(s) copy the bytes back and forth: the gateway acts as a wire.

In some cases a circuit connection is made automatically. For example, we have a host outside our gateway that needs to use an internal printer. We have told that host to connect to the print service on the gateway. Our gateway is configured to relay that particular connection to the printer port on an internal machine. We use an access control mechanism to ensure that only that one external host can connect to the gateway's printer service. We are also confident that this particular connection will not provide a useful entry hole should the external host be compromised.

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. In our implementation, called proxy, the destination is a host name. In socks (discussed later), it is the numeric IP address. If the connection is successful, the protocol ends and the real bytes start flowing. These services require modifications to the calling program or its library.

In general, these relay services do not examine the bytes as they flow through. Our services do log the number of bytes and the TCP destination. 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.

The outgoing proxy TCP service provides most of the external connectivity our internal users need. As noted, though, protocols such as FTP and X11 require incoming calls. But it is too much of a security risk to permit the gateway to make an uncontrolled call to the inside.

Any general solution is going to involve the gateway machine listening on some port. Ths approach demonstrates 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, say, 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.

Clearly, some sorts of 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.

The other big problem with circuit relays is the need to provide new client programs. Although the code changes are generally not onerous, they are a nuisance. Issues include availability of application source code for various platforms, version control, distribution, and the headache to users of having to know about two subtly different programs.

Several strategies are available for making the necessary changes. The best known is the socks package [8]. It consists of a set of almost-compatible replacements for various system calls: socket, connect, bind, etc. Converting an application is as simple as replacing the vanilla calls with the

The principal disadvantage of application-level gateways is the need for a specialized user program or variant user interface for most services provided.

Regardless of the firewall design, it is generally necessary to support various incoming services. Naturally, access to any of these must be blessed by the filter and the gateway.

`socks` equivalents. A version of it has been implemented via a replacement shared library, similar to that used in securelib [11] and 3-D FS [10]. This would permit existing applications to run unchanged. But such libraries are not portable, and it may not be possible to include certain of the security features mentioned earlier.

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.

Supporting Inbound Services

Regardless of the firewall design, it is generally necessary to support various incoming services. These include things like e-mail, FTP, logins, and possibly site-specific services. Naturally, access to any of these must be blessed by the filter and the gateway.

The most straightforward way to do this is to provide these services on the gateway itself. This is the obvious solution for mail and FTP. For incoming logins, we provide a security server; users must have one-time password devices to gain access to inside machines. If they pass that test, the gateway program will connect them to an inside machine, using some sort of preauthenticated connection mechanism such as rlogin.

Ganesan has implemented a gateway that uses Kerberos [4, 9, 14, 24] to authenticate calls [6]. Once the gateway has satisfied itself about the identity of the caller, it will pass the connection on to the desired internal server. With this design, the Kerberos server should be run by the same group that administers the firewalls, since the party that controls the server controls the authenication, and hence the ability to make calls through the firewall.

Regardless of the scheme used, all incoming calls carry some risk. The telnet call that was authenticated via a strong mechanism could be the product of a booby-trapped command. Consider, for example, a version that, after a few hundred bytes, displays "`Destination Unreachable`" on the console and exits — but before doing that, it forks, and retains the open session to your inside machine. Similarly, a legitimate user who connects for the purpose of reading mail takes the risk that some of those messages contain sensitive information, information that can now be read by anyone monitoring the unprotected, untrustworthy outside network.

Tunnels Good and Bad

Although firewalls offer strong protection, tunnels can be used to bypass them. As with most technologies, tunnels can be used in good or bad ways.

Tunneling refers to the practice of encapsulating a message from one protocol in another, and using the facilities of the second protocol to traverse some number of network hops. At the destination point, the encapsulation is stripped off, and the original message is reinjected into the network. In a sense, the packet burrows under the intervening network nodes, and never actually sees them. There are many uses for such a facility, such as encrypting links and supporting mobile hosts. More are described in [2].

In some cases, a protocol may be encapsulated within itself. That is, IP may be buried within either IP or some part of its own protocol suite, such as TCP or UDP. That is the situation we are concerned about here. If a firewall permits user packets to be sent, a tunnel can be used to bypass the firewall. The implications of this are profound.

Suppose that an internal user with a friend on the outside dislikes the firewall, and wishes to bypass it. The two of them can construct (dig?) a tunnel between an inside host and an outside host, thereby allowing the free flow of packets. This is far worse than a simple outgoing call, since incoming calls are permitted as well.

Almost any sort of mechanism can be used to build a tunnel. At least one vendor of a Point-to-Point Protocol (PPP) package [22] supports TCP tunneling. There are reports of telnet connections and even DNS messages being used to carry IP packets. Almost any gateway that supports anything more powerful than mail relays can be abused in this fashion (yet see RFC 1149 [28]). Even pairs of FTP file transfer connections can provide a bidirectional data path.

The extent of the damage done by a tunnel depends on how routing information is propagated. Suppression of routing information is almost as effective as full isolation. If the tunnel does not leak your routes to the outside, the damage is less than might be feared at first glance. On the other hand, routing filters are difficult to deploy in complex topologies; if the moles choose to pass connectivity information, it is hard to block them. In the Internet, the backbone routers do, in fact, perform filtering. Thus, if your internal networks are not administratively authorized for connection to the Internet, routes to them will not propagate past that point. Even so, you are exposed to anyone using the same network provider as the tunnel exit.

Often, such a situation can be detected. If you are using an application- or circuit-level gateway, and an external router knows a path to any internal network except the gateway's, something is leaking. This argues strongly that a gateway net should not be a subnet of an internal net. Rather, it should have its own, separate, Class C address. Standard network management tools may be able to hunt down the source, at which time standard people management tools should be able to deal with the root cause. Unauthorized tunnels are, in the final analysis, a management problem, not a technical one. If insiders will not accept the need for information security, firewalls and gateways are likely to be futile. (Of course, it is worth asking if one's protective measures are too stringent. Certainly, that happens as well.) Once suspected or spotted, the gateway logging tools should be able to pick out the tunnels.

Tunnels have their good side as well. When properly employed, they can be used to bypass the limitations of a topology. For example, a tunnel could link two separate sites that are connected only via a commercial network provider. Firewalls at each location would provide protection from the outside, while the tunnel provides connectivity. If the tunnel traffic is encrypted, the risks are low and the benefits high.

What Firewalls Cannot Do

Firewalls are a powerful tool for network security. However, there are things they cannot do. It is important to understand their limitations as well as their benefits.

Consider the usual network protocol layer cake. By its nature, a firewall is a very strong defense against attacks at a lower level of the protocol stack. For example, hosts behind a circuit-level relay are more or less immune to network-level attacks, such as IP address-spoofing. The forged packets cannot reach them; the gateway will only pass particular TCP connections that have been properly set up.

In contrast, firewalls provide almost no protection against problems with higher level protocols, except by peeking. The best TCP relay in the world is no protection if the code that uses it is buggy and insecure. You only get protection at this level if your gateway refuses to connect you to certain services (i.e., `X11`), and even that decision is applying application-layer knowledge to make that decision. (If you think of the standard protocol stack as an onion rather than as a layer cake, peering up through the layers may be referred to as "looking through a glass onion.")

The most interesting question is what degree of protection a firewall can provide against threats at its own level. The answer turns entirely on how carefully the gateway code — the permissive part — is written. Thus, a mail gateway, which runs at the application level, must be exceedingly careful to implement all of the mail protocols, and all of the other mail delivery functions, absolutely correctly. To the extent that it is insecurely written — `sendmail` comes to mind — it cannot serve as an adequate firewall component.

The problems, however, do not stop there. Any information that passes inside can trigger problems, if a sensitive component should lay hands (or silicon) on it. We have seen files that, when transferred over a communications link, effectively brought down that link, because of bit pattern sensitivity in some network elements. Were that deliberate, we would label it a denial-of-service attack.

A recent `sendmail` bug provides a sterling example. Problems with certain mail header lines could tickle bugs in delivery agents. Our firewall, and many others, paid almost no attention to headers, believing that they were strictly a matter for mail readers and composers (known as user agents in the e-mail business). But that meant that the firewalls provided no protection against this problem, because under certain circumstances, `sendmail` — which is run on many internal machines here — does look at the headers, and certain entries made it do evil things.

Furthermore, even if we had implemented defenses against the known flaws, we would still be vulnerable to next year's. If someone invented a new header line that was implemented poorly — and this particular problem did involve a nonstandard header — we would still be vulnerable. We could have protected ourselves if and only if we had refused to pass anything but the minimal subset of headers we did know of, and even then there might have been danger if some aspect of processing a legitimate, syntactically correct header was implemented poorly. At best, a firewall provides a convenient single place to apply a corrective filter.

Conclusions

Networks are very powerful tools. Like all tools, they can be misused. Firewalls, though not perfect, provide a strong measure of protection for computers connected to networks. There are a number of firewall technologies to choose from, each with its own advantages. Regardless of which is selected, careful configuration is necessary. But if one have a good security policy, and a correct implementation of it, one can enjoy most of the benefits of networking, while minimizing the risks.

References

[1] F. Anklesaria *et al.*, The Internet gopher protocol (A distributed document search and retrieval protocol). RFC 1436, March 1993.
[2] S. M. Bellovin, Pseudo-network drivers and virtual networks. In USENIX Conf. Proc., pp. 229-244, Washington, D.C., Jan. 22-26,1990.
[3] S. M. Bellovin, Firewall-friendly FTP. RFC 1579, Feb. 1994.
[4] B. Bryant, Designing an authentication system: A dialogue in four scenes, Feb. 8, 1988, Draft.
[5] D. B. Chapman, Network (in)security through IP packet filtering. In Proceedings of the Third Usenix UNIX Security Symposium, pp. 63-76, Baltimore, MD, Sept. 1992.
[6] R. Ganesan, BAfirewall: A modern design, Proc. of the Internet Society Symposium on Network an d Distributed System Security, San Diego, CA, Feb. 3, 1994.
[7] K. Hafner and J. Markoff, Cyberpunk: Outlaws and Hackers on the Computer Frontier (Simon & Schuster, 1991).
[8] D. Koblas and M. R. Koblas, Socks, UNIX Security III Symposium, pp. 77-83, Baltimore, MD, Sept. 14-17,1992. USENIX.
[9] J. Kohl and C. Neuman, The Kerberos network authentication service (V5), RFC 1510, Sept. 1993.
[10] D. G. Korn and E. Krell, The 3-d file system, USENIX Conf. Proc., pp. 147-156, Baltimore, MD, Summer 1989.
[11] W. LeFebvre, Restricting network access to system daemons under SunOS, UNIX Security III Symposium, pp. 93-103, Baltimore, MD, Sept. 14-17,1992. USENIX.
[12] M. Lottor, Domain administrators operations guide. RFC 1033, Nov. 1987.
[13] M. Lottor, TCP port service multiplexer (TCPMUX), RFC 1078, Nov. 1988.
[14] S. P. Miller *et al.*, Kerberos authentication and authorization system, Project Athena Technical Plan, MIT, Dec. 1987, Section E.2.1.
[15] D. Mills, Network time protocol (version 3) specification, implementation and analysis. RFC 1305, March 1992.
[16] P. Mockapetris, Domain names — concepts and facilities, RFC 1034, Nov. 1987.
[17] P. Mockapetris, Domain names — implementation and specification, RFC 1035, Nov. 1987.
[18] J. Postel, User datagram protocol, RFC 768, Aug. 28, 1980.
[19] J. Postel, Transmission control protocol, RFC 793, Sept.1981.
[20] J. Postel and J. Reynolds, File transfer protocol. RFC 959, Oct. 1985.
[21] M. J. Ranum, A network firewall, Proc. World Conference on System Administration and Security, Washington, D.C., July 1992.
[22] William Simpson, The point-to-point protocol (PPP) for the transmission of multi-protocol datagrams over point-to-point links. RFC 1331, May 1992.
[23] M. Stahl, Domain administrators guide, RFC 1032, Nov. 1987.
[24] J. Steiner, B. Clifford Neuman, and J. I. Schiller, Kerberos: An authentication service for open network systems, Proc. Winter USENIX Conf., pp. 191-202, Dallas, TX, 1988.
[25] Sun Microsystems, RPC: Remote procedure call protocol specification: Version 2. RFC 1057, June 1988.
[26] Sun Microsystems, Network Interfaces Programmer's Guide, Mountain View, CA, March, 1990. SunOS 4.1.
[27] W. Treese and A. Wolman, X through the firewall, and other application relays, USENIX Conf. Proc., pp. 87-99, Cincinnati, OH, June 1993.
[28] D. Waitzman, Standard for the transmission of IP datagrams on avian carriers. RFC 1149, April 1, 1990.

Biographies

STEVEN M. BELLOVIN received a B.A. from Columbia University, and M.S. and Ph.D. degrees in computer science from the University of North Carolina at Chapel Hill. While a graduate student, he wrote the original version of `pathalias` and helped create `netnews`. However, the former is not an indictable offense, and the statute of limitations on the latter has expired. Since 1982 he has been at AT&T Bell Laboratories, Murray Hill, New Jersey, where he does research in networks and security, and why the two don't get along. He is also the co-author of the recent book *Firewalls and Internet Security: Repelling the Wily Hacker*. His e-mail address is smb@research.att.com.

WILLIAM R. CHESWICK was graduated from Lehigh University in the mid-1970s with a computer science degree. He pursued a random career of technical support, teaching, and system administration at a variety of universities in the northeast. For the past seven years he has served as an assistant programmer trainee and member of the technical staff at Bell Laboratories in Murray Hill, New Jersey. He recently coauthored, with Steve Bellovin, *Firewalls and Internet Security: Repelling the Wily Hacker*. His e-mail address is ches@research.att.com.

What degree of protection can a firewall provide against threats at its own level? The answer turns entirely on how carefully the gateway code — the permissive part — is written.