# 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,* 

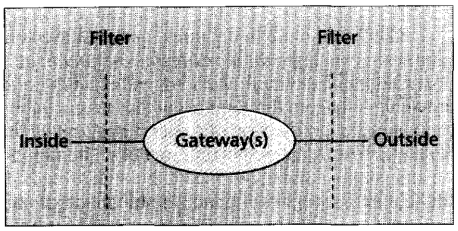


**Figure 1.** *Schematic of a firewall.*

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

DOCKET ALARM

# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

**LAW FIRMS**
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

**FINANCIAL INSTITUTIONS**
Litigation and bankruptcy checks for companies and debtors.

**E-DISCOVERY AND LEGAL VENDORS**
Sync your system to PACER to automate legal marketing.