

EX 1008

Paranoid Penguin

Designing and Using DMZ Networks to Protect Internet Servers

Mick explains how to care for services that come into contact with untrusted networks.

by Mick Bauer

One of the most useful tools in firewall engineering today is the DMZ, or DeMilitarized Zone, a network where all publicly accessible services are placed so they can be more closely watched and, also, isolated from one's internal network. DMZs, bastion servers and Linux make a particularly good combination.

But what, really, is a DMZ? Is there more than one correct way to design one? Does everyone who hosts internet services need a DMZ network? These are issues I really haven't addressed yet, so this month we're going to take a higher-level look at DMZ security.

By the way, you may decide that your current DMZ-less firewall system is reasonable for your needs. I hope you keep reading, regardless: any host or service (whether on a DMZ or not) that has direct contact with untrusted networks demands particular care, and many of the techniques and considerations discussed in this article apply to both non-DMZ and DMZ environments.

Some Terminology

Let's get some definitions cleared up before we proceed. These may not be the same definitions you're used to or prefer, but they're the ones I use in this article:

- *DMZ (DeMilitarized Zone)*: a network containing publicly accessible servers that is isolated from the "internal" network proper but not necessarily from the outside world.
- *Internal Network*: that which we're trying to protect: end-user systems, servers containing private data and all other systems with which we do not wish the outside world to initiate connection. Also called the protected network.
- *Firewall*: a system or network that isolates one network from another. This can be a router, a computer running special software in addition to or instead of its standard operating system, a dedicated hardware device (although these tend to be prepackaged routers or computers), or any other device or network of devices that performs some combination of packet filtering, application-layer proxying and other access control. In this article the term will generally refer to a single multihomed host.
- *Multihomed Host*: any computer having more than one network interface.
- *Bastion Host*: a system that runs publicly accessible services but is not itself a firewall. Bastion Hosts are what we put on DMZs (although they can be put anywhere). The term implies that a certain amount of OS-hardening has been done, but this (sadly) is not always the case.
- *Packet Filtering*: inspecting the IP headers of packets and passing or dropping them based on some combination of their Source IP Address, Destination IP Address, Source Port (Service) and Destination Port (Service). Application data is not considered, i.e., intentionally malformed packets are not necessarily noticed, assuming their IP headers can be read. Packet filtering is part of nearly all firewalls' functionality but is not considered, in and of itself, to be sufficient protection against any but the most straightforward attacks. Most routers (and many low-end firewalls) are limited to packet

filtering when it comes to network security.

- *Proxying*: to act as an intermediary in all interactions of a given service type (FTP, HTTP, etc.) between internal hosts and untrusted/external hosts. This implies, but does not guarantee, sophisticated inspection of Application-Layer data (i.e., more than simple packet filtering). Some firewalls possess, and are even built around, Application-Layer Proxies. Each service to be proxied must be explicitly supported (i.e., ``coded in"); firewalls that rely on Application-Layer Proxies tend to use packet filtering or rewriting for services they don't support by default.
- *Stateful Inspection*: at its simplest, this refers to the tracking of the three-way handshake (host1:SYN, host2:SYNACK, host1:ACK) that occurs when each session for a given TCP service is initiated. At its most sophisticated, it refers to the tracking of this and subsequent (including application-layer) state information for each session being inspected. The latter is far less common than the former.

That's a mouthful of jargon, but it's useful jargon (useful enough, in fact, to make sense of the majority of firewall-vendors' propaganda). Now we're ready to dig into DMZ architecture.

Types of Firewall and DMZ Architectures

In the world of expensive commercial firewalls (the world in which I earn my living), the term firewall nearly always denotes a single computer or dedicated hardware device with multiple network interfaces. Actually, this definition can apply to much lower-end solutions as well: network interface cards are cheap, as are PCs in general.

Regardless, this is different from the old days when a single computer typically couldn't keep up with the processor overhead required to inspect all ingoing and outgoing packets for a large network. In other words, routers, not computers, used to be the first line of defense against network attacks.

This is no longer the case. Even organizations with high-capacity Internet connections typically use a multihomed firewall (whether commercial or OSS-based) as the primary tool for securing their networks. This is possible thanks to Moore's law, which has provided us with inexpensive CPU power at a faster pace than the market has provided us with inexpensive Internet bandwidth. In other words, it's now feasible for even a relatively slow PC to perform sophisticated checks on a full T1's-worth (1.544MBps) of network traffic.

The most common firewall architecture one tends to see nowadays, therefore, is the one illustrated in Figure 1. In this diagram, we have a packet-filtering router that acts as the initial but not sole line of defense. Directly behind this router is a proper firewall, in this case a Sun SparcStation running, say, Red Hat Linux with IPChains. There is no direct connection from the Internet or the external router to the internal network: all traffic to it or from it must pass through the firewall.

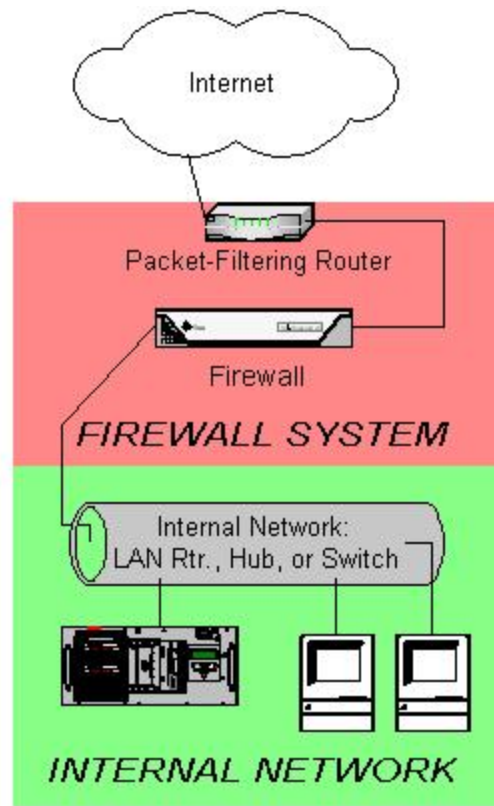


Figure 1: "Multi-Homed Host" Firewall

Figure 1. ``Multihost Firewall''

By the way, in my opinion, all external routers should use some level of packet filtering (aka ``Access Control Lists" in the Cisco lexicon). Even when the next hop inward from such a router is an expensive and/or carefully configured and maintained firewall, it never hurts to have redundant enforcement points. In fact, when several Check Point vulnerabilities were demonstrated at the most recent Black Hat Briefings, no less a personage than a Check Point spokesperson mentioned that it's foolish to rely solely on one's firewall!

What's missing or wrong in Figure 1? (I said this architecture is common, not perfect!) Public services such as SMTP (e-mail), Domain Name Service (DNS) and HTTP (WWW) must either be sent through firewall to internal servers or hosted on the firewall itself.

Passing such traffic doesn't automatically expose other internal hosts to attack, but it does magnify the consequences of such a server being compromised. Hosting public services on the firewall isn't necessarily a bad idea on the face of it, either (what could be a more secure environment than a firewall?), but the performance issue is obvious: the firewall should be allowed to use all its available resources for inspecting and moving packets. (Although there are some possible exceptions that we'll examine shortly.)

Where, then, to put public services so that they don't directly or indirectly expose the internal network and overtax the firewall? In a DMZ network, of course! At its simplest, a DMZ is any network reachable by the public but isolated from one's internal network. Ideally, however, a DMZ is also protected by the firewall. Figure 2 shows my preferred firewall/DMZ architecture.

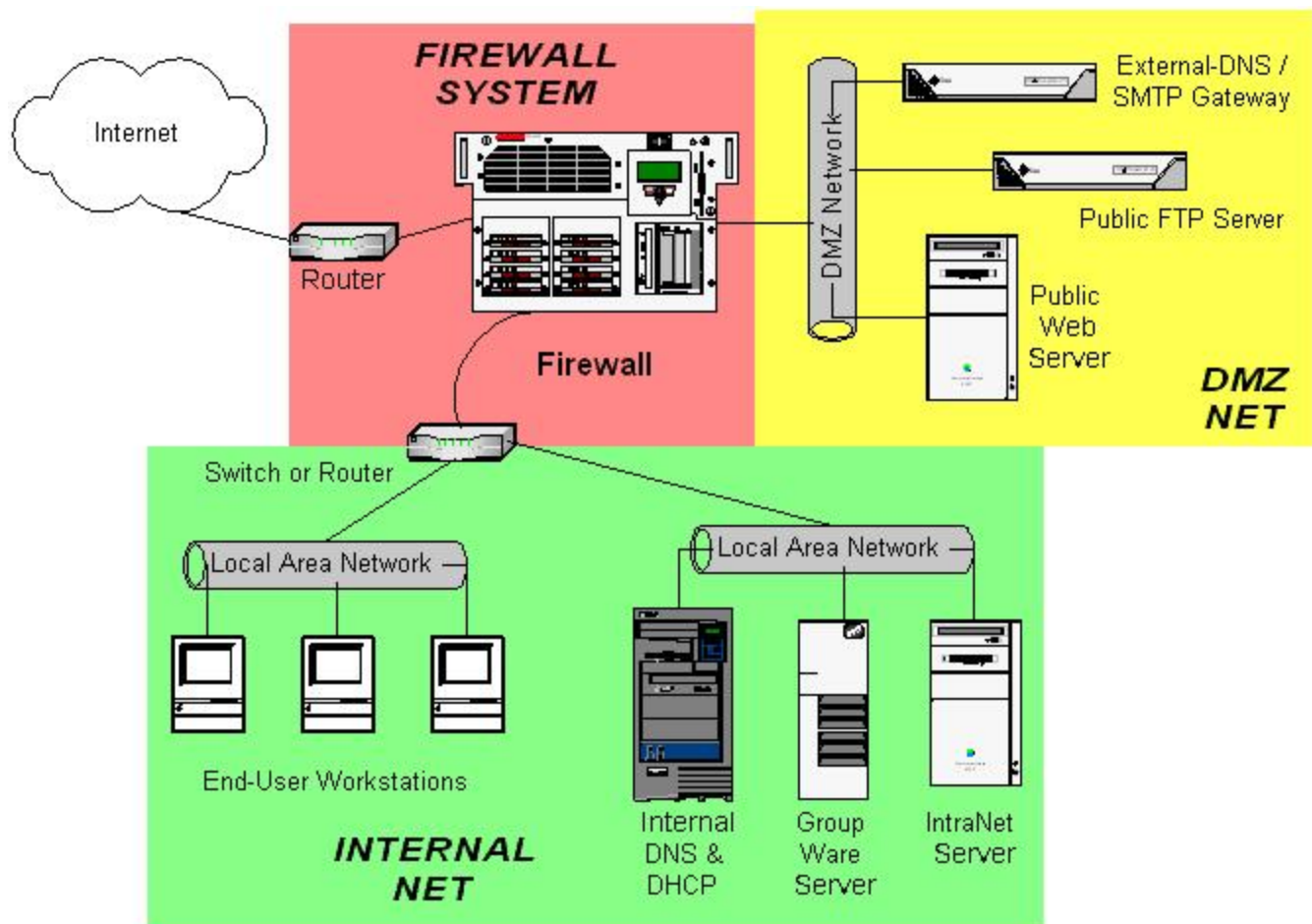


Figure 2: "Multi-Homed Host" Firewall With DMZ

Figure 2. "Multihoned Host" Firewall with DMZ

In Figure 2 we have a *three*-homed host as our firewall, placed so that hosts providing publicly accessible services are in *their own* network with a dedicated connection to the firewall, with the rest of the corporate network facing a different firewall interface. If configured properly, the firewall uses different rules in evaluating traffic from the Internet to the DMZ, from the DMZ to the Internet, from the Internet to the internal network, from the internal network to the Internet, from the DMZ to the internal network and from the internal network to the DMZ.

This may sound like more administrative overhead than with internally-hosted or firewall-hosted services, but actually, it's potentially much simpler because the DMZ can be treated as a single entity. In the case of internally hosted services, each host must be considered individually unless they're all located on a single IP network otherwise isolated from the rest of the internal network.

Other architectures are sometimes used, and Figure 3 illustrates two of them. The Screened Subnet architecture is completely dependent on the security of both the external and internal routers. There is a direct physical path from the outside to the inside, a path controlled by nothing more sophisticated than the router's packet-filtering rules.

Figure 3. "Screened Subnet" and "Flapping in the Breeze" DMZ Architectures

The right-hand illustration in Figure 3 shows what I call the "Flapping in the Breeze" DMZ architecture, in

which there *is* a full-featured firewall between the Internet and the internal network but *not* between the Internet and the DMZ, which is placed *outside* of the firewall and is protected only by a single packet-filtering router.

Both the Screened Subnet and Flapping in the Breeze architectures still show up in firewall textbooks (albeit with different names), but in my opinion, they both place too much trust in routers. Such trust is problematic for several reasons: first, in many organizations routers are under a different person's control than the firewall is, and this person many insist that the router have a weak administrative password, weak access-control lists or even a modem attached so that the router's vendor can maintain it; second, routers are considerably more hackable than well-configured computers (for example, by default they nearly always support remote-administration via Telnet, a highly insecure service); and third, packet filtering is a crude and incomplete means of regulating network traffic.

Even an OSS/freeware-based firewall can support IPSEC, application-layer proxies, stateful inspection, RADIUS authentication and a variety of other sophisticated controls unavailable on most routers. When all is said and done, routers are designed to route, not to protect.

What about Cisco PIX? The PIX firewall *is* a router but with a hardened and security-focused version of the Cisco IOS operating system. Although it relies heavily on simple packet filtering, it supports enough additional features to be a good firewall if properly configured. When I question the viability of routers as firewalls, I'm referring to nonhardened, general-purpose routers.

In summary, what one's DMZ architecture looks like depends on what one's firewall architecture looks like. A firewall design built around a multihomed host lends itself to the DMZ architecture I recommend (see Figure 2), in which the DMZ is connected to its own interface on the firewall host and, thus, is isolated from both the Internet and one's internal network.

Deciding What Should Reside on the DMZ

Once you've decided where to put the DMZ, you need to decide precisely what's going to reside there. In a nutshell, my advice is to put *all* publicly accessible services in the DMZ.

Too often I encounter organizations in which one or more crucial services are passed through the firewall to an internal host despite an otherwise strict DMZ policy. Frequently, the exception is made for MS-Exchange or some other application not necessarily designed for Internet-strength security to begin with and which hasn't been hardened to the extent it could be.

But, the one application passed through in this way becomes the dreaded single point of failure: all it takes is one buffer-overflow vulnerability in that application for it to be possible for an unwanted visitor to gain access to all hosts reachable by that host. Far better for that list of hosts to be a short one, i.e., DMZ hosts, than a long one (and a sensitive one), i.e., all hosts on the internal network. This point can't be stressed enough: the real value of a DMZ is that it allows us to better manage and contain the risk that comes with Internet connectivity.

Furthermore, the person who manages the passed-through service may be different from the one who manages the firewall and DMZ servers and may not be quite as security-minded. If for no other reason, all public services should go on a DMZ so that they fall under the jurisdiction of an organization's most paranoid system administrators, namely, its firewall administrators.

Does this mean that the corporate e-mail, DNS and other crucial servers should all be moved from the inside to the DMZ? Absolutely not! They should instead be split into internal and external services (see Figure 2).

DNS, for example, should be split into external DNS and internal DNS. The external DNS zone information, which is propagated out to the Internet, should contain only information about publicly accessible hosts. Information about other, nonpublic hosts should be kept on separate internal DNS zone lists that can't be transferred to or seen by external hosts.

Similarly, internal e-mail (i.e., mail from internal hosts to other internal hosts) should be handled strictly by internal hosts, and all Internet-bound or Internet-originated mail should be handled by a DMZ host, usually called an SMTP Gateway. (For more specific information on Split-DNS servers and SMTP Gateways and how to use Linux to create secure ones, see the October 2000 issue of *Linux Journal*.)

Thus, almost any service that has both private and public roles can and should be split in this fashion. While it may seem like a lot of added work, it need not be, and in fact, it's liberating: it allows you to construct your internal services based primarily on features or other management- and end-user-friendly factors while designing your public (DMZ) services based primarily on security and performance factors. It's also a convenient opportunity to integrate Linux, OpenBSD and other open-source software into otherwise proprietary-software-intensive environments!

Needless to say, any service that is strictly public (i.e., not used in a different or more sensitive way by internal users than by the general public) should reside solely in the DMZ. In summary: all public services, including the public components of services that are also used on the inside, should be split if applicable and hosted in the DMZ, without exception.

Allocating Resources in the DMZ

Okay, so everything public goes in the DMZ. But does each service need its own host? Can any of the services be hosted on the firewall itself? And should one use a hub or a switch on the DMZ?

The last question is the easiest: with the price of switched ports decreasing every year, switches are preferable on any LAN and especially so in DMZs. Switches are superior in two ways. From a security standpoint, they're better because it's impossible to sniff or eavesdrop traffic not delivered to one's own switch-port. Because one of our assumptions about DMZ hosts is they are more likely to be attacked than internal hosts, this is important. We need to think not only about how to prevent these hosts from being compromised but also about what the consequences might be if they are--being used to sniff other traffic on the DMZ is one possible consequence.

The other way switches are better than hubs is, of course, their performance. Most of the time each port has its own chunk of bandwidth rather than sharing one big chunk with all other ports. Note, however, that each switch has a backplane that describes the actual volume of packets the switch can handle; a 10-port 100Mbps switch can't really process 1000Mbps if it has an 800Mbps backplane. Even so, low-end switches disproportionately outperform comparable hubs.

Still, a hub may yet suffice for your DMZ depending on the number of hosts on your DMZ, the degree to which you're worried about a compromised DMZ host being used to attack other DMZ hosts and the amount of traffic to and from the DMZ.

The other two questions can usually be determined by nonsecurity-driven factors (cost, expected load, efficiency, etc.) *provided* that all DMZ hosts are thoroughly secured and monitored. Also, firewall rules (packet filters, etc.) governing traffic to and from the DMZ need to be as paranoid as possible.

Guidelines for Securing DMZ Hosts

It would seem to be common sense that each host on a DMZ must be thoroughly nailed down. But sure

enough, one commonly encounters organizations paranoid (prudent) enough to have a DMZ but not quite paranoid enough to secure their DMZ properly. The good news is that with a little time and a properly suspicious attitude, you can significantly lower any system's exposure to compromise by script kiddies.

Always run the latest stable version of your operating system, software and kernel, and keep current with security patches as they are released.

If everyone followed this simple and obvious tenet, the ``Rumors from the Underground" list of hacked web pages on [www.hackernews.com](http://www.hackernews.com) would be a lot shorter. As we discussed last month in ``Securing DNS", the vast majority of DNS-based hacks don't apply to the most recent versions of BIND; the same can safely be said for most other Linux network software packages. We all know this, but we don't always find the time to follow through.

Disable all unnecessary services and daemons.

A program you don't use for anything important is a program you've got little reason to maintain properly and is, therefore, an obvious target for attackers. This is an even easier thing to fix than old software. At setup time, simply delete or rename all unneeded links in the appropriate runlevel directory in `/etc/rc.d/`.

For example, if you're configuring a web server that doesn't need to be its own DNS server, you would enter something like the following:

```
mv /etc/rc.d/rc2.d/S30named /etc/rc.d/rc2.d/disabled_S30named
```

(Note that your named startup script may have a different name and may exist in different or additional subdirectories of `/etc/rc.d/`.)

While any unneeded service should be disabled, the following deserve particular attention:

- **RPCservices:** Sun's Remote Procedure Control protocol (which is included nowadays on virtually all flavors of UNIX) lets you execute commands on a remote system via **rsh**, **rcp**, **rlogin**, **nfs**, etc. Unfortunately, it isn't a very secure protocol, especially for use on DMZ hosts. You shouldn't be offering these services to the outside world--if you need their functionality, use **ssh** (the Secure Shell), which was specifically designed as a replacement for rpc services. Disable (rename) the `nfsd` and `nfsclntd` scripts in all subdirectories of `/etc/rc.d` in which they appear, and comment out the lines corresponding to any `r-`commands in `/etc/inetd.conf`. (Warning: local processes sometimes require the RPC portmapper, aka `rpcbind`--disable this with care, and try re-enabling it if other things stop working).
- **inetd:** The Internet Dæmon is a handy way to use a single process (i.e., `inetd`) to listen on multiple ports and to invoke the services on whose behalf it's listening on an as-needed basis. Its useful life, however, is drawing to a close: even with TCP Wrappers (which allow one to turn on very granular logging for each `inetd` service), it isn't as secure as simply running each service as a dæmon. (An FTP server really has no reason not to be running `FTPD` processes all the time.) Furthermore, most of the services enabled by default in `inetd.conf` are unnecessary, insecure or both. If you must use `inetd`, edit `/etc/inetd.conf` to disable all services you don't need (or never heard of). Note: many rpc services are started in `inetd.conf`.
- **linuxconfd:** While there aren't any known exploitable bugs in the current version of `linuxconf` (a system administration tool that can be accessed remotely), CERT reports that this service is commonly scanned for and may be used by attackers to identify systems with other vulnerabilities (CERT Current Activity page 7/31/2000, http://www.cert.org/current/current_activity.html).
- **sendmail:** Many people think that `sendmail`, which is enabled by default on most versions of UNIX, is necessary even on hosts that send e-mail only to themselves (e.g., administrative messages such as

Ctrl+tab output sent to root by the crontab daemon). This is not so: sendmail (or postfix, qmail, etc.) is needed only on hosts that must deliver mail to or receive mail from other hosts. sendmail is usually started in /etc/rc.d/rc2.d or /etc/rc.d/rc3.d.

- **Telnet, FTP and POP:** These three protocols have one very nasty characteristic in common: they require users to enter a username and password that are sent in clear text over the network. Telnet and FTP are easily replaced with ssh and its file-transfer utility **scp**; e-mail can either be automatically forwarded to a different host, left on the DMZ host and read through a ssh session or downloaded via POP using a "local forward" to ssh (i.e., piped through an encrypted Secure Shell session). All three of these services are usually invoked by inetd. These daemons are usually started by inetd.

Run services "chrooted" whenever possible.

Some daemons, such as named, have explicit support for being run in a "chroot jail" (i.e., such that to the chrooted process, "/" is actually some other directory that can't be navigated out of). This is a valuable security feature; if a chrooted process is hijacked or exploited somehow, the attacker will not be able to access files outside of the chroot jail.

On Linux, even processes that don't have built-in chroot support can be run chrooted: simply type `chroot chroot-jail path command string`. For example, to run the imaginary command `bubba -v plop` chrooted to `/var/bubba`, you would type:

```
chroot /var/bubba /usr/local/bin/bubba -v plop
```

Note, however, that any system files a chrooted process needs in order to run must be copied to appropriate subdirectories of the chroot jail. If our imaginary process `bubba` needs to parse `/etc/passwd`, we need to put a copy of the `passwd` file in `/var/bubba/etc`. The copy need not, however, contain any more information than the chrooted process needs; to extend our example still further, if `bubba` is a server that only anonymous users may access, `/var/bubba/etc/passwd` probably only needs one line (e.g., `nobody::50:50:Anonymous user::/bin/noshell`).

Run services with unprivileged UIDs and GIDs whenever possible.

While some daemons will only work if run by root (the default UID of processes invoked at startup time), nowadays many programs can be set to run as unprivileged users. For example, Postfix, Wietse Venema's sendmail replacement, usually runs with a special, unprivileged account named `postfix`.

This has a similar effect as `chroot` (and in fact the two often go together). Should the process become hijacked or otherwise compromised, the attacker will gain a level of access privileges lower than root's (hopefully much lower). Be careful, however, to make sure that such an unprivileged account still has enough privilege to do its job.

Delete or disable unnecessary user accounts.

Some Linux distributions have, by default, lengthy `/etc/passwd` files that contain accounts even for use by software packages that haven't been installed. My laptop computer, for example, which runs SuSE Linux, has 22 unnecessary entries in `/etc/passwd`. Commenting out or deleting such entries, especially the ones that include executable shells, is important.

Configure logging and check logs regularly.

This is another thing we all know we should do but often fail to follow through on. You can't check logs that

don't exist, and you can't learn anything from logs you don't read. Make sure your important services are logging at an appropriate level, know where those logs are stored and whether/how they're rotated when they get large, and get in the habit of checking the current logs for anomalies.

grep is your friend here: using **cat** alone tends to overwhelm people. You can automate some of this log-parsing with shell scripts; scripts are also handy for running **diff** against your system's configuration files to monitor changes (i.e., by comparing current versions to cached copies).

If you have a number of DMZ hosts, you may wish to consider using syslogd's ability to consolidate logs from several hosts on one system. You may not realize it, but the syslog daemon can be configured to listen not only for log data from other processes on the local system, but on data from remote hosts as well. For example, if you have two DMZ hosts (bobo and rollo) but wish to be able to view both machines' logs in a single place, you could change bobo's /etc/syslogd.conf to contain only this line:

```
*.* @rollo
```

This will cause syslogd on bobo to send all log entries not to its own /var/log/messages file but to rollo's.

While handy, be aware that this technique has its own security ramifications: if rollo is compromised, bobo's logs can also be tampered with. Furthermore, rollo's attacker may learn valuable information about bobo that they can subsequently use to attack bobo. This may or may not be of concern to you, but you should definitely think about whether the benefit justifies the exposure (especially given that the benefit may be that you can more effectively prevent your DMZ hosts from being compromised in the first place).

We'll close with the guideline that makes DMZs worthwhile in the first place.

Use your firewall's security policy and anti-IP-spoofing features.

Naturally, you want to carefully restrict traffic from the outside world to the DMZ. But it's equally important to carefully restrict traffic from the DMZ to your internal network (to protect it in the event that a DMZ host is compromised) and from the DMZ to the outside world (to prevent a compromised DMZ host from being used to attack other networks).

It goes without saying that you'll probably want to block all traffic from the Internet to internal hosts. (You may or may not feel a need to restrict traffic from the internal network to the DMZ, depending on what type of access internal users really need to DMZ hosts and how much you trust internal users.) In any event, your firewall-security policy will be much more effective if your firewall can distinguish between legitimate and phony source-IP addresses. Otherwise, it might be possible for an external user to slip packets through the firewall by forging internal source IPs.

By default, most firewalls don't have this functionality enabled (the feature is usually called something like anti-IP-spoofing. Even if your firewall supports it, you'll probably have to configure and start it yourself. It's well worth the effort, though.



Mick Bauer (mick@visi.com) is security practice lead at the Minneapolis bureau of ENRGI, a network engineering and consulting firm. He's been a Linux devotee since 1995 and an OpenBSD zealot since 1997, taking particular pleasure in getting these cutting-edge operating systems to run on obsolete junk. Mick welcomes questions, comments and greetings.