
FOURTH EDITION
LINUX
FIREWALLS
ENHANCING SECURITY WITH NFTABLES AND BEYOND
STEVE SUEHRING
FREE SAMPLE CHAPTER
SHARE WITH OTHERS

Linux® Firewalls

Fourth Edition

This page intentionally left blank

Linux® Firewalls

Enhancing Security with nftables and Beyond

Fourth Edition

Steve Suehring

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

Suehring, Steve.
Linux firewalls : enhancing security with nftables and beyond.—Fourth edition / Steve Suehring.
pages cm
Earlier ed. authored by Robert L. Ziegler.
Includes bibliographical references and index.
ISBN 978-0-13-400002-2 (pbk. : alk. paper)—ISBN 0-13-400002-1 (pbk. : alk. paper)
1. Computers—Access control. 2. Firewalls (Computer security) 3. Linux. 4. Operating systems (Computers) I. Title.
QA76.9.A25Z54 2015
005.8—dc2

2014043643

ISBN-13: 978-0-13-400002-2
ISBN-10: 0-13-400002-1
Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing, January 2015

❖

This book is dedicated to Jim Leu,
without whom I couldn't have written a book on Linux.

❖

This page intentionally left blank

Contents at a Glance

Contents

This page intentionally left blank

Preface

Welcome to the fourth edition of *Linux® Firewalls*. The book looks at what it takes to build a firewall using a computer running Linux. The material covered includes some basics of networking, IP, and security before jumping into `iptables` and `nftables`, the latest firewall software in Linux.

A reader of this book should be running a Linux computer, whether standalone or as a firewall or Internet gateway. The book shows how to build a firewall for a single client computer such as a desktop and also shows how to build a firewall behind which multiple computers can be hosted on a local network.

The final part of the book shows aspects of computer and network security beyond `iptables` and `nftables`. This includes intrusion detection, filesystem monitoring, and listening to network traffic. The book is largely Linux agnostic, meaning that just about any popular flavor of Linux will work with the material with little or no adaptation.

Acknowledgments

I'd like to thank my wife, family, and friends for their unending support. Thanks also to Robert P.J. Day and Andrew Prowant for reviewing the manuscript.

This page intentionally left blank

About the Author

Steve Suehring is a technology architect specializing in Linux and Windows systems and development. Steve has written several books and magazine articles on a wide range of technologies. During his tenure as an editor at *LinuxWorld* magazine, Steve wrote and edited articles and reviews on Linux security and advocacy including a feature story on the use of Linux in Formula One auto racing.

This page intentionally left blank

This page intentionally left blank

2

Packet-Filtering Concepts

What is a firewall? Over the years, the term has changed in meaning. According to RFC 2647, "Benchmarking Terminology for Firewall Performance," a firewall is "a device or group of devices that enforces an access control policy between networks." This definition is very broad, purposefully so in fact. A firewall can encompass many layers of the OSI model and may refer to a device that does packet filtering, performs packet inspection and filtering, implements a policy on an application at a higher layer, or does any of these and more.

A nonstateful, or stateless, firewall usually performs some packet filtering based solely on the IP layer (Layer 3) of the OSI model, though sometimes higher-layer protocols are involved in this type of firewall. An example of this type of device might include a border router that sits at the edge of a network and implements one or more access lists to prevent various types of malicious traffic from entering the network. Some might argue that this type of device isn't a firewall at all. However, it certainly appears to fit within the RFC definition.

A border router access list might implement many different policies depending on which interface the packet was received on. It's typical to filter certain packets at the edge of the network connecting to the Internet. These packets are discussed later in this chapter.

As opposed to a stateless firewall, a stateful firewall is one that keeps track of the packets previously seen within a given session and applies the access policy to packets based on what has already been seen for the given connection. A stateful firewall implies the basic packet-filtering capabilities of a stateless firewall as well. A stateful firewall will, for example, keep track of the stages of the TCP three-way handshake and reject packets that appear out of sequence for that handshake. Being connectionless, UDP is somewhat trickier to handle for a stateful firewall because there's no state to speak of. However, a stateful firewall tracks recent UDP exchanges to ensure that a packet that has been received relates to a recent outgoing packet.

An *Application-level gateway* (ALG), sometimes referred to an as an *Application-layer gateway*, is yet another form of firewall. Unlike the stateless firewall, which has knowledge of the Network and possibly Transport layers, an ALG primarily handles Layer 7, the Application layer of the OSI model. ALGs typically have deep knowledge of the application data

being passed and can thus look for any deviation from the normal traffic for the application in question.

An ALG will typically reside between the client and the real server and will, for all intents and purposes, mimic the behavior of the real server to the client. In effect, local traffic never leaves the LAN, and remote traffic never enters the LAN.

ALG sometimes also refers to a module, or piece of software that assists another firewall. Many firewalls come with an FTP ALG to support FTP's port mode data channel, where the client tells the server what local port to connect to so that it can open the data channel. The server initiates the incoming data channel connection (whereas, usually, the client initiates all connections). ALGs are frequently required to pass multimedia protocols through a firewall because multimedia sessions often use multiple connections initiated from both ends and generally use TCP and UDP together.

ALG is a proxy. Another form of proxy is a *circuit-level proxy*. Circuit-level proxies don't usually have application-specific knowledge, but they can enforce access and authorization policies, and they serve as termination points in what would otherwise be an end-to-end connection. SOCKS is an example of a circuit-level proxy. The proxy server acts as a termination point for both sides of the connection, but the server doesn't have any application-specific knowledge.

In each of these cases, the firewall's purpose is to enforce the access control or security policies that you define. Security policies are essentially about access control—who is and is not allowed to perform which actions on the servers and networks under your control.

Though not necessarily specific to a firewall, firewalls many times find themselves performing additional tasks, some of which might include Network Address Translation (NAT), antivirus checking, event notification, URL filtering, user authentication, and Network-layer encryption.

This book covers the ideas behind a packet-filtering firewall, both static and dynamic, or stateless and stateful. Each of the approaches mentioned controls which services can be accessed and by whom. Each approach has its strengths and advantages based on the differing information available at the various OSI reference model layers.

Chapter 1, "Preliminary Concepts Underlying Packet-Filtering Firewalls," introduced the concepts and information a firewall is based on. This chapter introduces how this information is used to implement firewall rules.

A Packet-Filtering Firewall

At its most basic level, a packet-filtering firewall consists of a list of acceptance and denial rules. These rules explicitly define which packets will and will not be allowed through the network interface. The firewall rules use the packet header fields described in Chapter 1 to decide whether to forward a packet to its destination, to silently throw away the packet, or to block the packet and return an error condition to the sending machine. These rules can be based on a wide array of factors, including the source or destination IP addresses, the source and (more commonly) destination ports, and portions of individual packets such as the TCP header flags, the types of protocol, the MAC address, and more.

MAC address filtering is not common on Internet-connected firewalls. Using MAC filtering, the firewall blocks or allows only certain MAC addresses. However, in all likelihood you see only one MAC address, the one from the router just upstream from your firewall. This means that every host on the Internet will appear to have the same MAC address as far as your firewall can see. A common error among new firewall administrators is to attempt to use MAC filtering on an Internet firewall.

Using a hybrid of the TCP/IP reference model, a packet-filtering firewall functions at the Network and Transport layers, as shown in Figure 2.1.

The overall idea is that you need to very carefully control what passes between the Internet and the machine that you have connected directly to the Internet. On the external interface to the Internet, you individually filter what's coming in from the outside and what's going out from the machine as exactly and explicitly as possible.

For a single-machine setup, it might be helpful to think of the network interface as an I/O pair. The firewall independently filters what comes in and what goes out through the interface. The input filtering and the output filtering can, and likely do, have completely different rules. Figure 2.2 depicts processing for rules in a flowchart.

This sounds pretty powerful, and it is; but it isn't a surefire security mechanism. It's only part of the story, just one layer in the multilayered approach to data security. Not all application communication protocols lend themselves to packet filtering. This type of

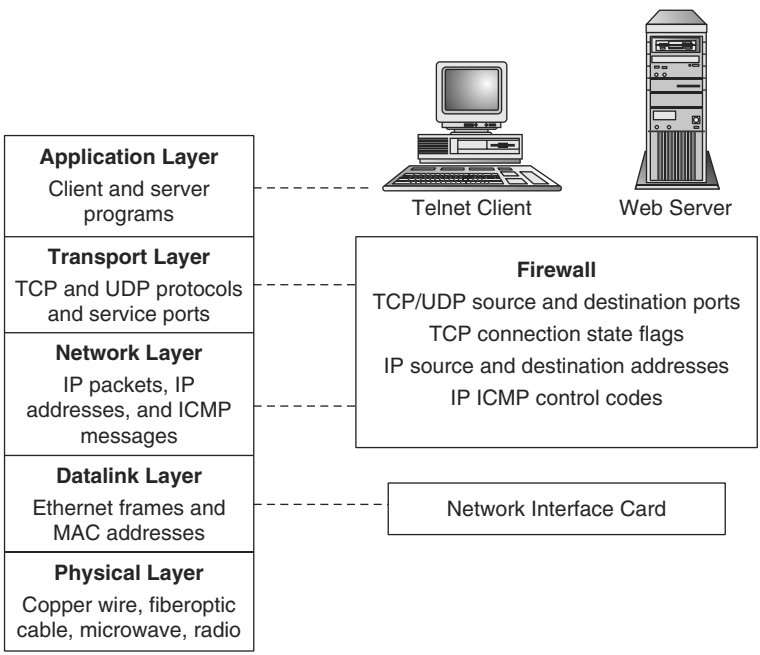

Figure 2.1 Firewall placement in the TCP/IP reference model

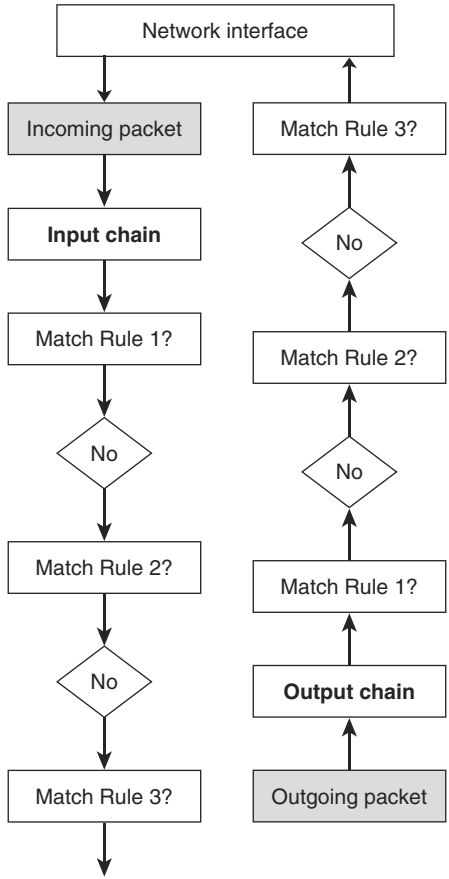

Figure 2.2 Input and output flowchart

filtering is too low-level to allow fine-grained authentication and access control. These security services must be furnished at higher levels. IP doesn't have the capability to verify that the sender is who he or she claims to be. The only identifying information available at this level is the source address in the IP packet header. The source address can be modified with little difficulty. One level up, neither the Network layer nor the Transport layer can verify that the application data is correct. Nevertheless, the packet level allows greater, simpler control over direct port access, packet contents, and correct communication protocols than can easily or conveniently be done at higher levels.

Without packet-level filtering, higher-level filtering and proxy security measures are either crippled or potentially ineffective. To some extent, at least, they must rely on the correctness of the underlying communication protocol. Each layer in the security protocol stack adds another piece that other layers can't easily provide.

Choosing a Default Packet-Filtering Policy

As stated earlier in this chapter, a firewall is a device to implement an access control policy. A large part of this policy is the decision on a default firewall policy.

There are two basic approaches to a default firewall policy:

- Deny everything by default, and explicitly allow selected packets through.
- Accept everything by default, and explicitly deny selected packets from passing through.

Without question, the deny-everything policy is the recommended approach. This approach makes it easier to set up a secure firewall, but each service and related protocol transaction that you want must be enabled explicitly (see Figure 2.3). This means that you

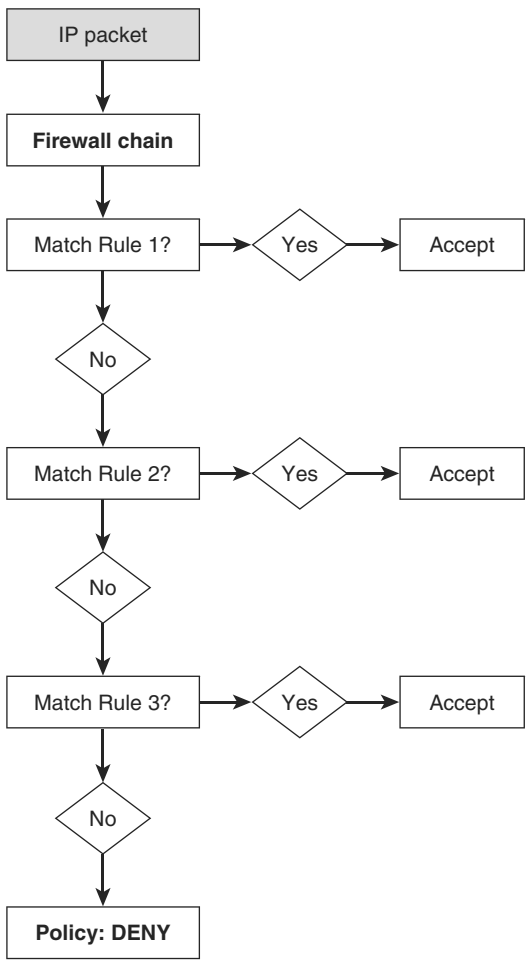

Figure 2.3 The deny-everything-by-default policy

must understand the communication protocol for each service you enable. The deny-everything approach requires more work up front to enable Internet access. Some commercial firewall products support only the deny-everything policy.

The accept-everything policy makes it much easier to get up and running right away, but it forces you to anticipate every conceivable access type that you might want to disable (see Figure 2.4). The danger is that you won't anticipate a dangerous access type until it's too late, or you'll later enable an insecure service without first blocking external access to it. In the end, developing a secure accept-everything firewall is much more work, much more difficult, almost always much less secure, and, therefore, much more error-prone.

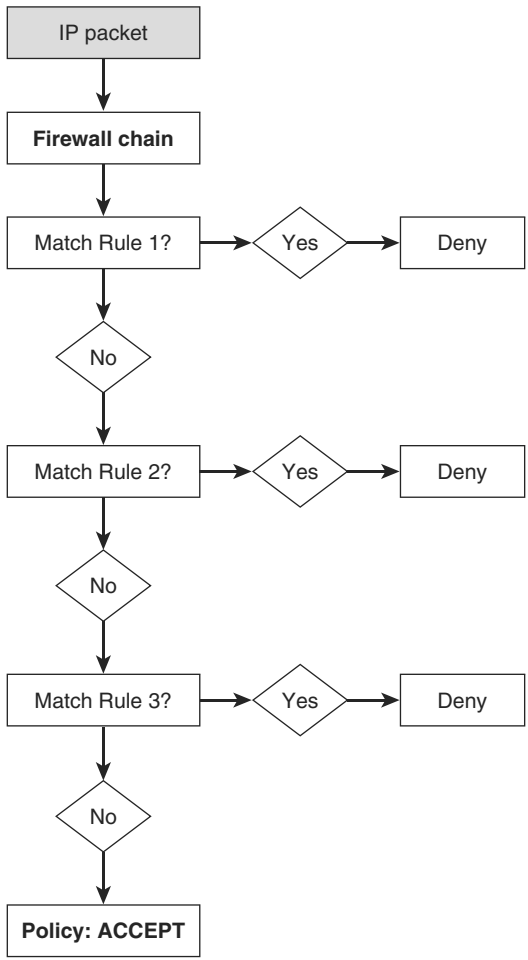

Figure 2.4 The accept-everything-by-default policy

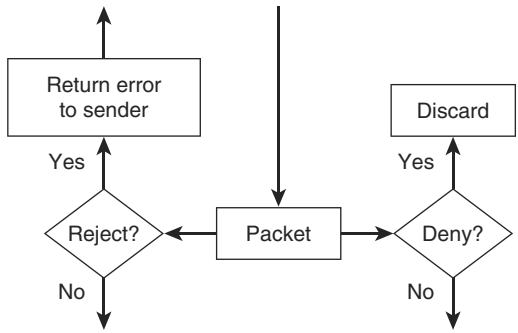

Figure 2.5 Rejecting versus denying a packet

Rejecting versus Denying a Packet

The Netfilter firewall mechanism in `iptables` and `nftables` gives you the option of either rejecting or dropping packets. What's the difference? As shown in Figure 2.5, when a packet is rejected, the packet is thrown away and an ICMP error message is returned to the sender. When a packet is dropped, the packet is simply thrown away without any notification to the sender.

Silently dropping the packet is almost always the better choice, for three reasons. First, sending an error response doubles the network traffic. The majority of dropped packets are dropped because they are malevolent, not because they represent an innocent attempt to access a service you don't happen to offer. Second, a packet that you respond to can be used in a denial-of-service (DoS) attack. Third, any response, even an error message, gives the would-be attacker potentially useful information.

Filtering Incoming Packets

The input side of the external interface I/O pair, the input rule set, is the more interesting in terms of securing your site. As mentioned earlier, you can filter based on source address, destination address, source port, destination port, TCP status flags, and other criteria. You'll learn about all these pieces of information at one point or another in the following sections.

Remote Source Address Filtering

At the packet level, the only means of identifying the IP packet's sender is the source address in the packet header. This fact allows for the possibility of source address spoofing, in which the sender places an incorrect address rather than his or her own address in the source field. The address might be a nonexistent address, or it might be a legitimate address belonging to someone else. This can allow unsavory types to break into your

system by appearing as local, trusted traffic; appearing to be you while attacking other sites; pretending to be someone else while attacking you; keeping your system bogged down responding to nonexistent addresses; or otherwise misleading you as to the source of incoming messages.

It's important to remember that you usually can't detect spoofed addresses. The address might be legitimate and routable but might not belong to the packet's sender. The next section describes the spoofed addresses you can detect.

Source Address Spoofing and Illegal Addresses

There are several major classes of source addresses you should deny on your external interface in all cases. These are incoming packets claiming to be from the following:

- **Your IP address**—You will never see legal incoming packets claiming to be from your machine. Because the source address is the only information available and it can be modified, this is one of the forms of legitimate address spoofing you can detect at the packet-filtering level. Incoming packets claiming to be from your machine are spoofed. You can't be certain whether other incoming packets are coming from where they claim to be. (Note that some operating systems crash if they receive a packet in which both the source and the destination addresses belong to the host's network interface.)
- **Your LAN addresses**—You will rarely see legal incoming packets on the external, Internet interface claiming to be *from* your LAN. It's possible to see such packets if the LAN has multiple access points to the Internet, but it would probably be a sign of a misconfigured local network. In most cases, such a packet would be part of an attempt to gain access to your site by exploiting your local trust relationships.
- **Class A, B, and C private IP addresses**—These three sets of addresses in the historical Class A, B, and C ranges are reserved for use in private LANs. They aren't intended for use on the Internet. As such, these addresses can be used by any site internally without the need to purchase registered IP addresses. Your machine should never see incoming packets from these source addresses:
  - Class A private addresses are assigned the range from `10.0.0.0` to `10.255.255.255`.
  - Class B private addresses are assigned the range from `172.16.0.0` to `172.31.255.255`.
  - Class C private addresses are assigned the range from `192.168.0.0` to `192.168.255.255`.
- **Class D multicast IP addresses**—IP addresses in the Class D range are set aside for use as destination addresses when participating in a multicast network broadcast, such as an audiocast or a videocast. They range from `224.0.0.0` to `239.255.255.255`. Your machine should never see packets from these source addresses.

- **Class E reserved IP addresses**—IP addresses in the Class E range were set aside for future and experimental use and are not assigned publicly. They range from `240.0.0.0` to `247.255.255.255`. Your machine should never see packets from these source addresses—and mostly likely won't. (Because the entire address range is permanently reserved up through `255.255.255.255`, the Class E range can realistically be defined as `240.0.0.0` to `255.255.255.255`. In fact, some sources define the Class E address range to be exactly that.)
- **Loopback interface addresses**—The loopback interface is a private network interface used by the Linux system for local, network-based services. Rather than sending local traffic through the network interface driver, the operating system takes a shortcut through the loopback interface as a performance improvement. By definition, loopback traffic is targeted for the system generating it. It doesn't go out on the network. The loopback address range is `127.0.0.0` to `127.255.255.255`. You'll usually see it referred to as `127.0.0.1`, localhost, or the loopback interface, `lo`.
- **Malformed broadcast addresses**—Broadcast addresses are special addresses applying to all machines on a network. Address `0.0.0.0` is a special broadcast source address. A legitimate broadcast source address will be either `0.0.0.0` or a regular IP address. DHCP clients and servers will see incoming broadcast packets from source address `0.0.0.0`. This is the only legal use of this source address. It is not a legitimate point-to-point, unicast source address. When seen as the source address in a regular, point-to-point, nonbroadcast packet, the address is forged, or the sender isn't fully configured.
- **Class A network 0 addresses**—As suggested previously, any source address in the `0.0.0.0` through `0.255.255.255` range is illegal as a unicast address.
- **Link local network addresses**—DHCP clients sometimes assign themselves a link local address when they can't get an address from a server. These addresses range from `169.254.0.0` to `169.254.255.255`.
- **Carrier-grade NAT**—There are IPs that are marked for use by Internet providers that should never appear on a public network, the public Internet. These addresses can, however, be used in cloud scenarios, and therefore, if your server is hosted at a cloud provider, you may see these addresses. The carrier-grade NAT addresses range from `100.64.0.0` to `100.127.255.255`.
- **TEST-NET addresses**—The address space from `192.0.2.0` to `192.0.2.255` is reserved for test networks.

Blocking Problem Sites

Another common, but less frequently used, source address–filtering scheme is to block all access from a selected machine or, more typically, from an entire network's IP address block. This is how the Internet community tends to deal with problem sites and ISPs that

don't police their users. If a site develops a reputation as a bad Internet neighbor, other sites tend to block it across the board.

On the individual level, blocking all access from selected networks is convenient when individuals in the remote network are habitually making a nuisance of themselves. This has historically been used as a means to fight unsolicited email, with some people going so far as to block an entire country's range of IP addresses.

Limiting Incoming Packets to Selected Remote Hosts

You might want to accept certain kinds of incoming packets from only specific external sites or individuals. In these cases, the firewall rules will define either specific IP addresses or a limited range of IP source addresses that these packets will be accepted from.

The first class of incoming packets is from remote servers responding to your requests. Although some services, such as web or FTP services, can be expected to be coming from anywhere, other services will legitimately be coming from only your ISP or specially chosen trusted hosts. Examples of servers that are probably offered only through your ISP are POP mail service, Domain Name Service (DNS) name server responses, and possible DHCP or dynamic IP address assignments.

The second class of incoming packets is from remote clients accessing services offered from your site. Again, although some incoming service connections, such as connections to your web server, can be expected to be coming from anywhere, other local services will be offered to only a few trusted remote users or friends. Examples of restricted local services might be `ssh` and `ping`.

Local Destination Address Filtering

Filtering incoming packets based on the destination address is not much of an issue. Under normal operation, your network interface card ignores regular packets that aren't addressed to it. The exception is broadcast packets, which are broadcast to all hosts on the network.

The IPv4 address `255.255.255.255` is the general broadcast destination address. It refers to all hosts on the immediate physical network segment, and it is called a *limited broadcast*. A broadcast address can be defined more explicitly as the highest address in a given subnet of IP addresses. For example, if your ISP's network address is `192.168.10.0` with a 24-bit subnet mask (`255.255.255.0`) and your IP address is `192.168.10.30`, you would see broadcast packets addressed to `192.168.10.255` from your ISP. On the other hand, if you have a smaller range of IP addresses, say a `/30` (`255.255.255.252`), then you have a total of four addresses: one network, two for hosts, and the broadcast. For example, consider the network `10.3.7.4/30`. In this network, `10.3.7.4` is the network address, the two hosts would be `10.3.7.5` and `10.3.7.6`, and the broadcast address would be `10.3.7.7`. This `/30` subnet configuration type is typically used between routers, though the actual addresses themselves may vary. The only way to know what the broadcast address will be for a given subnet is to know both an IP address within the subnet and the subnet mask. These types of broadcasts are called *directed subnet broadcasts* and are delivered to all hosts on that network.

Broadcast-to-destination address `0.0.0.0` is similar to the situation of point-to-point packets claiming to be from the broadcast source address mentioned earlier, in the section "Source Address Spoofing and Illegal Addresses." Here, broadcast packets are directed to source address `0.0.0.0` rather than to the destination address, `255.255.255.255`. In this case, there is little question about the packet's intent. This is an attempt to identify your system as a Linux machine. For historical reasons, networking code derived from BSD UNIX returns an ICMP Type 3 error message in response to `0.0.0.0` being used as the broadcast destination address. Other operating systems silently discard the packet. As such, this is a good example of why dropping versus rejecting a packet makes a difference. In this case, the error message itself is what the probe is looking for.

Remote Source Port Filtering

Incoming requests and connections from remote clients to your local servers will have a source port in the unprivileged range. If you are hosting a web server, all incoming connections to your web server should have a source port between `1024` and `65535`. (That the server port identifies the service is the intention but not the guarantee. You cannot be certain that the server you expect is running at the port you expect.)

Incoming responses and connections from remote servers that you contacted will have the source port that is assigned to the particular service. If you connect to a remote website, all incoming messages from the remote server will have the source port set to `80` (or whatever port the local client specified), the `http` service port number.

Local Destination Port Filtering

The destination port in incoming packets identifies the program or service on your computer that the packet is intended for. As with the source port, all incoming requests from remote clients to your services generally follow the same pattern, and all incoming responses from remote services to your local clients follow a different pattern.

Incoming requests and connections from remote clients to your local servers will set the destination port to the service number that you assigned to the particular service. For example, an incoming packet destined for your local web server would normally have the destination port set to `80`, the `http` service port number.

Incoming responses from remote servers that you contacted will have a destination port in the unprivileged range. If you connect to a remote website, all incoming messages from the remote server will have a destination port between `1024` and `65535`.

Incoming TCP Connection State Filtering

Incoming TCP packet acceptance rules can make use of the connection state flags associated with TCP connections. All TCP connections adhere to the same set of connection states. These states differ between client and server because of the three-way handshake during connection establishment. As such, the firewall can distinguish between incoming traffic from remote clients and incoming traffic from remote servers.

Incoming TCP packets from remote clients will have the SYN flag set in the first packet received as part of the three-way connection establishment handshake. The first connection request will have the SYN flag set, but not the ACK flag.

Incoming packets from remote servers will always be responses to the initial connection request initiated from your local client program. Every TCP packet received from a remote server will have the ACK flag set. Your local client firewall rules will require all incoming packets from remote servers to have the ACK flag set. Servers do not normally attempt to initiate connections to client programs.

Probes and Scans

A *probe* is an attempt to connect to or get a response from an individual service port. A *scan* is a series of probes to a set of different service ports. Scans are often automated.

Unfortunately, probes and scans are rarely innocent anymore. They are most likely the initial information-gathering phase, looking for interesting vulnerabilities before launching an attack. Automated scan tools are widespread, and coordinated efforts by groups of hackers are common. The security, or lack thereof, of many hosts on the Internet, along with the proliferation of worms, viruses, and zombied machines, makes scans a constant issue on the Internet.

General Port Scans

General port scans are indiscriminate probes across a large block of service ports, possibly the entire range (see Figure 2.6). These scans are somewhat less frequent—or, at least, less obvious—as more sophisticated, targeted stealth tools become available.

Targeted Port Scans

Targeted port scans look for specific vulnerabilities (see Figure 2.7). The newer, more sophisticated tools attempt to identify the hardware, operating system, and software versions. These tools are designed to identify targets that might be prone to a specific vulnerability.

Common Service Port Targets

Common targets often are individually probed as well as scanned. The attacker might be looking for a specific vulnerability, such as an insecure mail server, an unpatched web server, or an open remote procedure call (RPC) `portmap` daemon.

A more extensive list of ports can be found at http://www.iana.org/assignments/port-numbers. Only a few common ports are mentioned here, to give you the idea:

- Incoming packets from reserved port `0` are always bogus. This port isn't used legitimately.
- Probes of TCP ports `0` to `5` are a signature of the sscan program.
- `ssh` (`22/tcp`), `smtp` (`25/tcp`), `dns` (`53/tcp/udp`), `pop-3` (`110/tcp`), `imap` (`143/tcp`), and `snmp` (`161/udp`), are favorite target ports. They represent some of the most potentially vulnerable openings to a system, whether intrinsically, due to

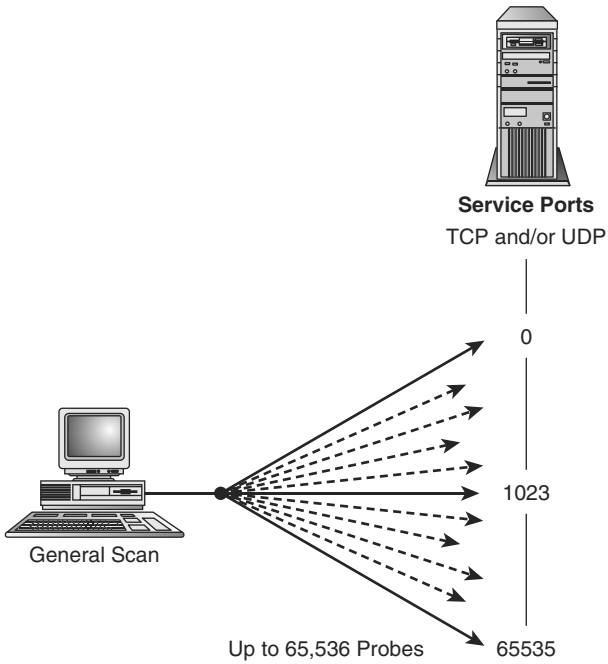

Figure 2.6 A general port scan

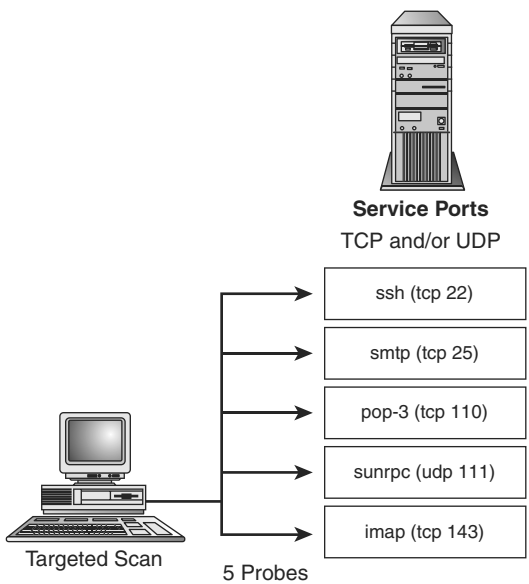

Figure 2.7 A targeted port scan

common configuration errors, or due to known flaws in the software. Because these services are so common, they are good examples of why you want to either not offer them to the outside world, or very carefully offer them with controlled outside access. NetBIOS (`137-139/tcp/udp`) and Server Message Block (SMB) on Windows (`445/tcp`) probes are tediously common. They typically pose no threat to a Linux system unless Samba is used on the system. The typical target is a Windows system, in this case, but the scans are all too common.

Stealth Scans

Stealth port scans, by definition, aren't meant to be detectable. They are based on how the TCP protocol stack responds to unexpected packets, or packets with illegal state flag combinations. For example, consider an incoming packet that has the `ACK` flag set but has no related connection. If the `ACK` were sent to a port with a listening server attached, the TCP stack wouldn't find a related connection and would return a TCP `RST` message to tell the sender to reset the connection. If the `ACK` were sent to an unused port, the system would simply return a TCP `RST` message as an error indication, just as the firewall might return an ICMP error message by default.

The issue is further complicated because some firewalls test only for the `SYN` flag or the `ACK` flag. If neither is set, or if the packet contains some other combination of flags, the firewall implementation might pass the packet up to the TCP code. Depending on the TCP state flag combination and the operating system receiving the packet, the system will respond with an `RST` or with silence. This mechanism can be used to help identify the operating system that the target system is running. In any of these cases, the receiving system isn't likely to log the event.

Inducing a target host to generate an `RST` packet in this manner also can be used to map a network, determining the IP addresses of systems listening on the network. This is especially helpful if the target system isn't a server and its firewall has been set to silently drop unwanted packets.

Avoiding Paranoia: Responding to Port Scans

Firewall logs normally show all kinds of failed connection attempts. Probes are the most common thing you'll see reported in your logs.

Are people probing your system this often? Yes, they are. Is your system compromised? No, it isn't. Well, not necessarily. The ports are blocked. The firewall is doing its job. These are failed connection attempts that the firewall denied.

At what point do you personally decide to report a probe? At what point is it important enough to take the time to report it? At what point do you say that enough is enough and get on with your life, or should you be writing `abuse@some.system` each time? There are no "right" answers. How you respond is a personal judgment call and depends in part on the resources available to you, how sensitive the information at your site is, and how critical the Internet connection is to your site. For obvious probes and scans, there is no clear-cut answer. It depends on your own personality and comfort level how you personally define a serious probe, and your social conscience.

With that in mind, these are some workable guidelines.

The most common attempts are a combination of automated probing, mistakes, legitimate attempts based on the history of the Internet, ignorance, curiosity, and misbehaving software.

You can almost always safely ignore individual, isolated, single connection attempts to `telnet`, `ssh`, `ftp`, `finger`, or any other port for a common service that you're not providing. Probes and scans are a fact of life on the Internet, are all too frequent, and usually don't pose a risk. They are kind of like door-to-door salespeople, commercial phone calls, wrong phone numbers, and junk postal mail. For me, at least, there isn't enough time in the day to respond to each one.

On the other hand, some probers are more persistent. You might decide to add firewall rules to block them completely, or possibly even their entire IP address space.

Scans of a subset of the ports known to be potential security holes are typically the precursor to an attack if an open port is found. More inclusive scans are usually part of a broader scan for openings throughout a domain or subnet. Current hacking tools probe a subset of these ports one after the other.

Occasionally, you'll see serious hacking attempts. This is unquestionably a time to take action. Write them. Report them. Double-check your security. Observe what they're doing. Block them. Block their IP address block.

Some system administrators take every occurrence seriously because, even if *their* machine is secure, other people's machines might not be. The next person might not even have the capability of knowing that he or she is being probed. Reporting probes is the socially responsible thing to do, for everyone's sake.

How should you respond to port scans? If you write these people, their postmaster, their uplink service provider network operations center (NOC), or the network address block coordinator, try to be polite. Give them the benefit of the doubt. Overreactions are misplaced more often than not. What might appear as a serious hacking attempt to you is often a curious kid playing with a new program. A polite word to the abuser, `root`, or postmaster can sometimes take care of the problem. More people need to be educated about Netiquette than need their network accounts rescinded. And they might be innocent of anything. Just as often, the person's system is compromised and that person has no idea what's going on and will be grateful for the information.

Probes aren't the only hostile traffic you'll see, however. Although probes are harmless in and of themselves, DoS attacks are not.

Denial-of-Service Attacks

DoS attacks are based on the idea of flooding your system with packets to disrupt or seriously degrade your Internet connection, tying up local servers to the extent that legitimate requests can't be honored or, in the worst case, crashing your system altogether. The two most common results are keeping the system too busy to do anything useful and tying up critical system resources.

You can't protect against DoS attacks completely. They can take as many different forms as the attacker's imagination allows. Anything that results in a response from your system, anything that results in your system allocating resources (including logging of the attack), anything that induces a remote site to stop communicating with you—all can be used in a DoS attack.

> **More on Denial-of-Service Attacks**
> For further information on DoS attacks, see the "Denial of Service" paper available at http://www.cert.org.

These attacks usually involve one of several classic patterns, including TCP `SYN` flooding, `ping` flooding, UDP flooding, fragmentation bombs, buffer overflows, and ICMP routing redirect bombs.

TCP `SYN` Flooding

A TCP `SYN` flood attack consumes your system resources until no more incoming TCP connections are possible (see Figure 2.8). The attack makes use of the basic TCP three-way handshaking protocol during connection establishment, in conjunction with IP source address spoofing.

The attacker spoofs his or her source address as a private address and initiates a connection to one of your TCP-based services. Appearing to be a client attempting to open a TCP connection, the attacker sends you an artificially generated `SYN` message. Your machine responds by sending an acknowledgment, a `SYN-ACK`. However, in this case, the

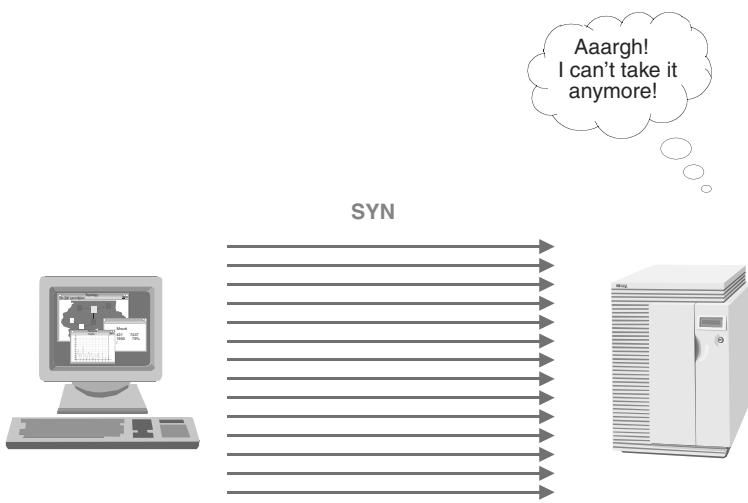

Figure 2.8 A TCP `SYN` flood

address that you're replying to isn't the attacker's address. In fact, because the address is private, there is no one out there to respond. The spoofed host won't return an RST message to tear down the half-opened connection.

The final stage of TCP connection establishment, receiving an ACK in response, will never happen. Consequently, finite network connection resources are consumed. The connection remains in a half-opened state until the connection attempt times out. The attacker floods your port with connection request after connection request, faster than the TCP timeouts release the resources. If this continues, all resources will be in use and no more incoming connection requests can be accepted. This applies not only to the service being probed, but to all new connections as well.

Several aids are available to Linux users. The first is source address filtering, described previously. This filters out the most commonly used spoofed source addresses, but there is no guarantee that the spoofed address falls within the categories you can anticipate and filter against.

The second is to enable your kernel's SYN cookie module, a specific retardant to the resource starvation caused by SYN flooding. When the connection queue begins to get full, the system starts responding to SYN requests with SYN cookies rather than SYN-ACKs, and it frees the queue slot. Thus, the queue never fills completely. The cookie has a short timeout; the client must respond to it within a short period before the serving host will respond with the expected SYN-ACK. The cookie is a sequence number that is generated based on the original sequence number in the SYN, the source and destination addresses and ports, and a secret value. If the response to the cookie matches the result of the hashing algorithm, the server is reasonably well assured that the SYN is valid.

Depending on the particular release, you may or may not need to enable the SYN cookie protection within the kernel by using the command `echo 1 > /proc/sys/net/ipv4/tcp_syncookies`. Some distributions and kernel versions require you to explicitly configure the option into the kernel using `make config`, `make menuconfig`, or `make xconfig` and then recompile and install the new kernel.

> **SYN Flooding and IP Spoofing**
> For more information on SYN flooding and IP spoofing, see CERT Advisory CA-96.21, "TCP SYN Flooding and IP Spoofing Attacks," at http://www.cert.org.

ping Flooding

Any message that elicits a response from your machine can be used to degrade your network connection by forcing the system to spend most of its time responding. The ICMP echo request message sent by ping is a common culprit. An attack called *Smurf*, and its variants, forces a system to expend its resources processing echo replies. One method of accomplishing this is to spoof the victim's source address and broadcast an echo request to an entire network of hosts. A single spoofed request message can result in hundreds or thousands of resulting replies being sent to the victim. Another way of accomplishing a similar result is to install trojans on compromised hosts across the Internet and time them

to each send echo requests to the same host simultaneously. Finally, a simple `ping` flood in which the attacker sends more echo requests and floods the data connection is another method for a DoS, though it's becoming less common. A typical `ping` flood is shown in Figure 2.9.

Ping of Death

An older exploit called the *Ping of Death* involved sending very large `ping` packets. Vulnerable systems could crash as a result. Linux is not vulnerable to this exploit, nor are many other current UNIX operating systems. If your firewall is protecting older systems or personal computers, those systems could be vulnerable.

The Ping of Death exploit gives an idea of how the simplest protocols and message interactions can be used by the creative hacker. Not all attacks are attempts to break into your computer. Some are merely destructive. In this case, the goal is to crash the machine. (System crashes also might be an indicator that you need to check your system for installed trojan programs. You might have been duped into loading a trojan program, but the program itself might require a system reboot to activate.)

`ping` is a very useful basic networking tool. You might not want to disable `ping` altogether. In today's Internet environment, conservative folks recommend disabling incoming `ping` or at least severely limiting from whom you accept echo requests. Because of `ping`'s history of involvement in DoS attacks, many sites no longer respond to external `ping` requests from any but selected sources. This has always seemed to be an overreaction to the relatively small threat of a DoS based on ICMP when compared to the more ubiquitous and dangerous threats against applications and other protocols within the stack.

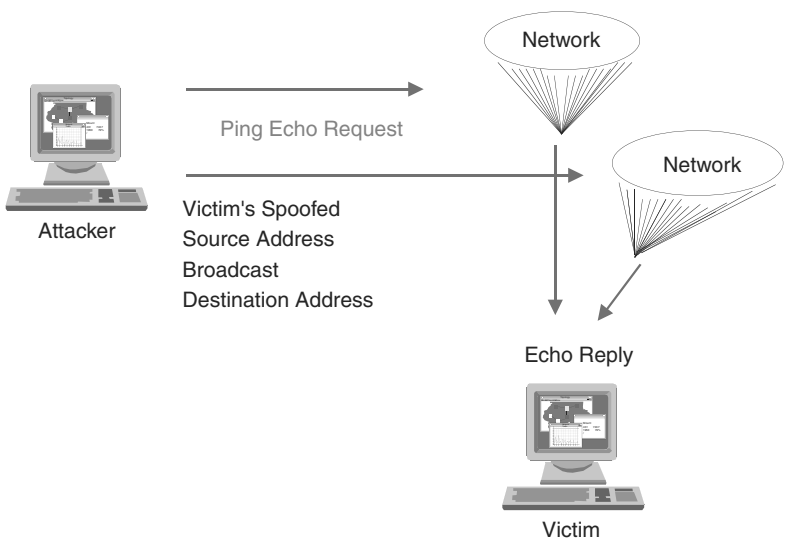

Figure 2.9 A `ping` flood

Dropping `ping` requests isn't a solution for the victim host, however. Regardless of how the recipient of the flood reacts to the packets, the system (or network) can still be overwhelmed in the process of inspecting and dropping a flood of requests.

UDP Flooding

The UDP protocol is especially useful as a DoS tool. Unlike TCP, UDP is stateless. Flow-control mechanisms aren't included. There are no connection state flags. Datagram sequence numbers aren't used. No information is maintained on which packet is expected next. There is not always a way to differentiate client traffic from server traffic based on port numbers. Without state, there is no way to distinguish an expected incoming response from an unsolicited packet arriving unexpectedly. It's relatively easy to keep a system so busy responding to incoming UDP probes that no bandwidth is left for legitimate network traffic.

Because UDP services are susceptible to these types of attacks (as opposed to connection-oriented TCP services), many sites disable all UDP ports that aren't absolutely necessary. As mentioned earlier, almost all common Internet services are TCP based. The firewall we'll build in Chapter 5, "Building and Installing a Standalone Firewall," carefully limits UDP traffic to only those remote hosts providing necessary UDP services.

The classic UDP flood attack either involves two victim machines or works in the same way the Smurf `ping` flood does (see Figure 2.10). A single spoofed packet from the attacker's UDP `echo` port, directed to a host's UDP `chargen` port, can result in an infinite loop of network traffic. The `echo` and `chargen` services are network test services. `chargen` generates an ASCII string. `echo` returns the data sent to the port.

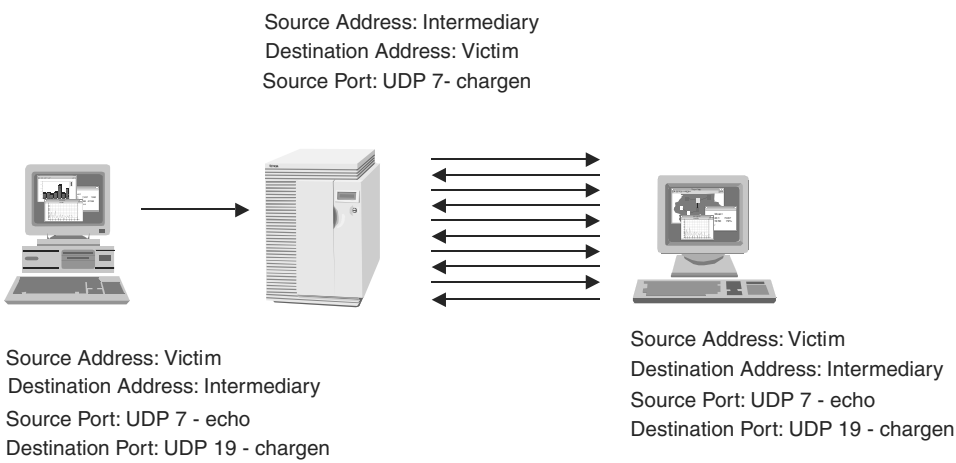

Figure 2.10 A UDP flood

> **UDP Port Denial-of-Service Attacks**
> For a fuller description of a DoS exploit using these UDP services, see CERT Advisory CA-96.01, "UDP Port Denial-of-Service Attack," at http://www.cert.org.

Fragmentation Bombs

Different underlying network technologies (such as Ethernet, Asynchronous Transfer Mode [ATM], and token ring) define different limits on the size of the Layer 2 frame. As a packet is passed on from one router to the next along the path from the source machine to the destination machine, network gateway routers might need to cut the packet into smaller pieces, called *fragments*, before passing them on to the next network. In a legitimate fragmentation, the first fragment contains the usual source and destination port numbers contained in the UDP or TCP transport header. The following fragments do not.

For example, although the maximum theoretical packet length is 65,535 bytes, the maximum Ethernet frame size (Maximum Transmission Unit, or MTU) is 1500 bytes.

When a packet is fragmented, intermediate routers do not reassemble the packet. The packets are reassembled either at the destination host or by its adjacent router.

Because intermediate fragmentation is ultimately more costly than sending smaller, nonfragmented packets, current systems often do MTU discovery with the target host at the beginning of a connection. This is done by sending a packet with the `Don't Fragment` option set in the IP header options field (the only generally legitimate current use of the IP options field). If an intermediate router must fragment the packet, it drops the packet and returns an ICMP 3 error, `fragmentation-required`.

One type of fragmentation attack involves artificially constructing very small packets. One-byte packets crash some operating systems. Current operating systems usually test for this condition.

Another use of small fragments is constructing the initial fragment so that the UDP or TCP source and destination ports are contained in the second fragment. (All networks' MTU sizes are large enough to carry a standard 40-byte IP and transport header.) Packet-filtering firewalls often allow these fragments through because the information that they filter on is not present. This form of attack is useful to get packets through the firewall that would not otherwise be allowed.

The Ping of Death exploit mentioned earlier is an example of using fragmentation to carry an illegally large ICMP message. When the `ping` request is reconstructed, the entire packet size is larger than 65,535 bytes, causing some systems to crash.

A classic example of a fragmentation exploit is the Teardrop attack. The method can be used to bypass a firewall or to crash a system. The first fragment is constructed to go to an allowed service. (Many firewalls don't inspect fragments after the first packet.) If it is allowed, the subsequent fragments will be passed through and reassembled by the target host. If the first packet is dropped, the subsequent packets will pass through the firewall, but the end host will have nothing to reconstruct and eventually will discard the partial packet.

The data offset fields in the subsequent fragments can be altered to overwrite the port information in the first fragment to access a disallowed service. The offset also can be altered so that offsets used in packet reassembly turn out to be negative numbers. Because kernel byte-copy routines usually use unsigned numbers, the negative value is treated as a very large positive number; the resulting copy trashes kernel memory and the system crashes.

Firewall machines and machines that do NAT for other local hosts should be configured to reassemble the packets before delivering them to the local target. Some of the `iptables` features require the system to reassemble packets before forwarding the packet to the destination host, and reassembly is done automatically.

Buffer Overflows

Buffer overflow exploits can't be protected against by a filtering firewall. The exploits fall into two main categories. The first is simply to cause a system or server to crash by overwriting its data space or runtime stack. The second requires technical expertise and knowledge of the hardware and system software or server version being attacked. The purpose of the overflow is to overwrite the program's runtime stack so that the call return stack contains a program and a jump to it. This program usually starts up a shell with `root` privilege.

Many of the current vulnerabilities in servers are a result of buffer overflows. It's important to install and keep up-to-date all the newest patches and software revisions.

ICMP Redirect Bombs

ICMP redirect message Type 5 tells the target system to change its in-memory routing tables in favor of a shorter route. Redirects are sent by routers to their adjacent hosts. Their intention is to inform the host that a shorter path is available (that is, the host and new router are on the same network, and the new router is the router that the original would route the packet to as its next hop).

Redirects arrive on an almost-daily basis. They rarely originate from the adjacent router. For residential or business sites connected to an ISP, it's very unlikely that your adjacent router will generate a redirect message.

If your host uses static routing and honors redirect messages, it's possible for someone to fool your system into thinking that a remote machine is one of your local machines or one of your ISP's machines, or even to fool your system into forwarding all traffic to some other remote host.

Denial-of-Service Attacks and Other System Resources

Network connectivity isn't the only concern in DoS attacks. Here are some examples of other areas to keep in mind while configuring your system:

- Your filesystem can overflow if your system is forced to write enormous numbers of messages to the error logs, or if your system is flooded with many copies of large email messages. You might want to configure resource limits and set up a separate partition for rapidly growing or changing filesystems.

Email Denial-of-Service Exploits
For a description of a DoS exploit using email, see "Email Bombing and Spamming" at http://www.cert.org.

- System memory, process table slots, CPU cycles, and other resources can be exhausted by repeated, rapid invocations of network services. You can do little about this other than setting any configurable limits for each individual service, enabling `SYN` cookies, and denying rather than rejecting packets sent to unsupported service ports.

Source-Routed Packets

Source-routed packets employ a rarely used IP option that allows the originator to define the route taken between two machines, rather than letting the intermediate routers determine the path. As with ICMP redirects, this feature can allow someone to fool your system into thinking that it's talking to a local machine, an ISP machine, or some other trusted host, or to create the necessary packet flow for a man-in-the-middle attack.

Source routing has few legitimate uses in current networks. Some routers ignore the option. Some firewalls discard packets containing the option.

Filtering Outgoing Packets

If your environment represents a trusted environment, filtering outgoing packets might not appear to be as critical as filtering incoming packets. Your system won't respond to incoming messages that the firewall doesn't pass through. Residential sites often take this approach. Nevertheless, even for residential sites, symmetric filtering is important, particularly if the firewall protects Microsoft Windows machines. For commercial sites, outgoing filtering is inarguably important.

If your firewall protects a LAN of Microsoft Windows systems, controlling outgoing traffic becomes much more important. Compromised Windows machines have historically been (and continue to be) used in coordinated DoS attacks and other outbound attacks. For this reason especially, it's important to filter what leaves your network.

Filtering outgoing messages also allows you to run LAN services without leaking into the Internet, where these packets don't belong. It's not only a question of disallowing external access to local LAN services. It's also a question of not broadcasting local system information onto the Internet. Examples of this would be if you were running a local DHCPD, NTP, SMB, or other server for internal use. Other obnoxious services might be broadcasting `wall` or `syslogd` messages.

A related source is some of the personal computer software, which sometimes ignores the Internet service port protocols and reserved assignments. This is the personal computer equivalent of running a program designed for LAN use on an Internet-connected machine.

A final reason is simply to keep local traffic local that isn't intended to leave the LAN but that conceivably could. Keeping local traffic local is a good idea from a security standpoint but also as a means for bandwidth conservation.

Local Source Address Filtering

Filtering outgoing packets based on the source address is easy. For a small site or a single computer connected to the Internet, the source address is always your computer's IP address during normal operation. There is no reason to allow an outgoing packet to have any other source address, and the firewall should enforce this.

For people whose IP address is dynamically assigned by their ISP, a brief exception exists during address assignment. This exception is specific to DHCP and is the one case in which a host broadcasts messages using `0.0.0.0` as its source address.

For people with a LAN whose firewall machine has a dynamically assigned IP address, limiting outgoing packets to contain the source address of the firewall machine's IP address is mandatory. It protects you from several fairly common configuration mistakes that appear as cases of source address spoofing or illegal source addresses to remote hosts.

If your users or their software aren't 100% trustworthy, it's important to ensure that local traffic contains legitimate, local addresses only, to avoid participating in DoS attacks using source address spoofing.

This last point is especially important. RFC 2827, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing" (updated by RFC 3704, "Ingress Filtering for Multihomed Networks"), is a current "best practices" document speaking to exactly this point. Ideally, every router should filter out the obvious illegal source addresses and ensure that traffic leaving the local network contains only routable source addresses belonging to that network.

Remote Destination Address Filtering

As with incoming packets, you might want to allow certain kinds of outgoing packets to be addressed only to specific remote networks or individual machines. In these cases, the firewall rules will define either specific IP addresses or a limited range of IP destination addresses to which these packets will be allowed.

The first class of outgoing packets to filter by destination address is packets destined to remote servers that you've contacted. Although some packets, such as those going to web or FTP servers, can be expected to be destined to anywhere on the Internet, other remote services will legitimately be offered from only your ISP or specially chosen trusted hosts. Examples of services that are probably offered only through your ISP are mail services such as SMTP or POP3, DNS services, DHCP dynamic IP address assignment, and the Usenet news service.

The second class of outgoing packets to filter by destination address is packets destined to remote clients who are accessing a service offered from your site. Again, although some outgoing service connections, such as responses from your local web server, can be

expected to be going anywhere, other local services will be offered to only a few trusted remote sites or friends. Examples of restricted local services might be telnet, SSH, Samba-based services, and RPC services accessed via `portmap`. Not only will the firewall rules deny general incoming connections to these services, but the rules also won't allow outgoing responses from these services to just anyone.

Local Source Port Filtering

Explicitly defining which service ports on your network can be used for outgoing connections serves two purposes—one for your client programs and one for your server programs. Specifying the source ports allowed for your outgoing connections helps ensure that your programs are behaving correctly, and it protects other people from any local network traffic that doesn't belong on the Internet.

Outgoing connections from your local clients will almost always originate from an unprivileged source port. Limiting your clients to the unprivileged ports in the firewall rules helps protect other people from potential mistakes on your end by ensuring that your client programs are behaving as expected.

Outgoing packets from your local server programs will always originate from their assigned service port and will be in response to a request received. Limiting your servers to their assigned ports at the firewall level ensures that your server programs are functioning correctly at the protocol level. More important, it helps protect any private, local network services that you might be running from outside access. It also helps protect remote sites from being bothered by network traffic that should remain confined to your local systems.

Remote Destination Port Filtering

Your local client programs are designed to connect to network servers offering their services from their assigned service ports. From this perspective, limiting your local clients to connect only to their associated server's service port ensures protocol correctness. Limiting your client connections to specific destination ports serves a couple of other purposes as well. First, it helps guard against local, private network client programs inadvertently attempting to access servers on the Internet. Second, it does much to disallow outgoing mistakes, port scans, and other mischief potentially originating from your site.

Your local server programs will almost always participate in connections originating from unprivileged ports. The firewall rules limit your servers' outgoing traffic to only unprivileged destination ports.

Outgoing TCP Connection State Filtering

Outgoing TCP packet acceptance rules can make use of the connection state flags associated with TCP connections, just as the incoming rules do. All TCP connections adhere to the same set of connection states, which differs between client and server.

Outgoing TCP packets from local clients will have the SYN flag set in the first packet sent as part of the three-way connection establishment handshake. The initial connection request will have the SYN flag set, but not the ACK flag. Your local client firewall rules will allow outgoing packets with either the SYN or the ACK flag set.

Outgoing packets from local servers will always be responses to an initial connection request initiated from a remote client program. Every packet sent from your servers will have the ACK flag set. Your local server firewall rules will require all outgoing packets from your servers to have the ACK flag set.

Private versus Public Network Services

One of the easiest ways to inadvertently allow uninvited intrusions is to allow outside access to local services that are designed only for LAN use. Some services, if offered locally, should never cross the boundary between your LAN and the Internet beyond. Some of these services annoy your neighbors, some provide information you'd be better off keeping to yourself, and some represent glaring security holes if they're available outside your LAN.

Some of the earliest network services, the `r-*`-based commands in particular, were designed for local sharing and ease of access across multiple lab machines in a trusted environment. Some of the later services were intended for Internet access, but they were designed at a time when the Internet was basically an extended community of academicians and researchers. The Internet was a relatively open, safe place. As the Internet grew into a global network including general public access, it developed into a completely untrusted environment.

Lots of Linux network services are designed to provide local information about user accounts on the system, which programs are running and which resources are in use, system status, network status, and similar information from other machines connected over the network. Not all of these informational services represent security holes in and of themselves. It's not that someone can use them directly to gain unauthorized access to your system. It's that they provide information about your system and user accounts that can be useful to someone who is looking for known vulnerabilities. They might also supply information such as usernames, addresses, phone numbers, and so forth, which you don't want to be readily available to everyone who asks.

Some of the more dangerous network services are designed to provide LAN access to shared filesystems and devices, such as a networked printer or fax machine.

Some services are difficult to configure correctly and some are difficult to configure securely. Entire books are devoted to configuring some of the more complicated Linux services. Specific service configuration is beyond the scope of this book.

Some services just don't make sense in a home or small-office setting. Some are intended to manage large networks, provide Internet routing service, provide large database informational services, support two-way encryption and authentication, and so forth.

Protecting Nonsecure Local Services

The easiest way to protect yourself is to not offer the service. But what if you need one of these services locally? Not all services can be protected adequately at the packet-filtering level. File-sharing software, instant messaging services, and UDP-based RPC services are notoriously difficult to secure at the packet-filtering level.

One way to safeguard your computer is to not host network services on the firewall machine that you don't intend for public use. If the service isn't available, there's nothing for a remote client to connect to. Let firewalls be firewalls.

A packet-filtering firewall doesn't offer complete security. Some programs require higher-level security measures than can be provided at the packet-filtering level. Some programs are too problematic to risk running on a firewall machine, even on a less secure residential host.

Small sites such as those in the home often won't have a supply of computers available to enforce access security policies by running private services on other machines. Compromises must be made, particularly for required services that are provided solely by Linux. Nevertheless, small sites with a LAN should not be running file-sharing or other private LAN services on the firewall, such as Samba. The machine should not have unnecessary user accounts. Unneeded system software should be removed from the system. The machine should have no function other than that of a security gateway.

Selecting Services to Run

When all is said and done, only you can decide which services you need or want. The first step in securing your system is to decide which services and daemons you intend to run on the firewall machine, as well as behind the firewall in the private LAN. Each service has its own security considerations. When it comes to selecting services to run under Linux or any other operating system, the general rule of thumb is to run only network services that you need and understand. It's important to understand a network service, what it does and who it's intended for, before you run it—especially on a machine connected directly to the Internet.

Summary

Between this and the preceding chapter, the basics of networking and firewalls have been laid out. The next chapter digs deeper into `iptables` itself.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z