Securing Linux Systems With Host-Based Firewalls

Implemented With Linux `iptables`

Gé Weijers, Linux Security Group

Sun BluePrints™ OnLine—November 2003

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-4403-10
Revision 1.0, 2/2/04
Edition: November 2003

Please

Securing Linux Systems With Host-Based Firewalls

The goal of this article is to provide the reader with a template for constructing a host-based firewall that provides a useful layer of protection against the risks of exposing a system to internal and/or external users. Additionally, the reader can gain an understanding of construction methods for host-based firewalls in general and Linux-based firewalls in particular.

This article is targeted for use with RedHat Advanced Server 2.1 and SuSE Enterprise Server 8, but most of the material applies to distributions based on Linux kernel version 2.4 and newer.

We assume that the reader is capable of creating basic Bourne shell scripts and can perform basic system administration tasks.

This article contains the following topics:

- "Introduction" on page 2
- "Why Use a Host-Based Firewall?" on page 3
- "`iptables`" on page 4
- "Creating a Host-Based Firewall Script" on page 8
- "Installing the Firewall Rule Set" on page 16
- "Removing the Firewall Rule Set" on page 16
- "Verifying Firewall Services" on page 17
- "Firewall Script Sample" on page 19
- "About the Author" on page 22
- "Related Resources" on page 22
- "Ordering Sun Documents" on page 23
- "Accessing Sun Documentation Online" on page 23

Introduction

In September 2001, Sun BluePrints™ OnLine published Martin Englund's article titled "Securing Systems with Host-Based Firewalls – Implemented With SunScreen™ Lite 3.1 Software." Since then, Sun Microsystems has introduced new products that are capable of running Linux in addition to Solaris™ X86. These new products prompted the writing of this article.

The character of this article is different from Martin Englund's article, because the tools that are offered under Linux and Solaris are quite different. SunScreen™ is a mature product that is relatively easy to configure. The Linux `iptables` packet filter is not nearly as sophisticated, and more knowledge is required to implement firewalls that function correctly.

This document provides only part of the solution for securing Linux hosts. For more information about securing Linux hosts, refer to the following resources:

- For information about system hardening, refer to the Sun BluePrints OnLine articles titled "Securing Sun Linux Systems: Part I, Local Access and File Systems" and "Securing Sun Linux Systems: Part II, Network Security."
- For in-depth information about firewalls and protocols, refer to *Building Internet Firewalls*.

No security tool by itself is sufficient to defend a host against compromise. A good defense strategy consists of multiple layers. The application and the operating system it runs on have to be configured to run as securely as possible. The operating system should provide only services and software that are needed; access privileges should be configured as tightly as possible; and an application should be configured to run with the least amount of privileges possible. Additional tools such as Intrusion Detection Systems provide another layer of security, and are an indispensable part of a security engineer's toolbox.

Security tools are effective only if they are used by well-trained and qualified people. Security requires a thorough risk analysis and clearly documented policies and processes, which have to be kept up to date. The use of tools to automate security processes is highly recommended, because it lowers cost and improves reliability.

This article includes an example host-based firewall setup that is designed to protect a single system more effectively than a one-size-fits-all corporate firewall system. The firewall is of modest complexity, and is easier to verify.

Why Use a Host-Based Firewall?

Enterprises commonly use perimeter-based firewall systems to protect systems on internal and service networks. Practical experience shows that this approach to protecting networks is often insufficient, for various reasons. Corporate firewalls installed on the perimeter of the network often have to handle a large number of protocols and services, and are difficult to configure correctly. Also, they do not protect against threats such as malicious mobile code (for example, viruses and worms) that somehow make it onto internal networks. Consequentially, many corporate networks are plagued by the latest crop of computer viruses, even though these networks are protected by firewalls and virus filters. Lastly, perimeter-based security does not protect against threats from inside a corporate network.

Host-based firewalls offer improved protection against the previously mentioned threats, and software is widely available for many systems. Linux systems support a kernel-based packet filter that is a suitable tool for constructing host-based firewalls. However, constructing a good set of rules that adequately protects a host is not trivial.

Host-based firewalls have the following advantages:

- ***Protection Against Firewall Failure*** – Adding another firewall of different design is helpful in case the primary firewall fails, because most likely the attack or problem that causes the primary firewall failure will not affect the host-based firewall similarly. Multiple firewalls do not offer increased protection against attacks directed at vulnerabilities in applications or operating systems.
- ***Simplicity*** – Configuring a host-based firewall is usually far simpler than configuring a perimeter firewall, because the host usually requires support for just a few protocols in order to function. Simplicity makes verification of the rule set simpler as well. (Complexity is the enemy of security.)
- ***Protection Against a Wider Number of Threats*** – The host-based firewall can protect against threats originating from within a corporate network, and can help mitigate the risks of badly configured software on a host.
- ***Specificity*** – A host-based firewall can be tuned to support a single set of applications and to block everything else. Perimeter firewalls are usually configured with a rule set designed to support many applications, and consequentially are much more likely to have exploitable weaknesses.

Note – We acknowledge one disadvantage of host-based firewalls is that they often require specific configurations, depending on the application programs hosted. It is time consuming to configure host-based firewalls on many different servers. In some cases, it may not be practical to provide individualized configurations for every host.

---

iptables

The packet filter/firewall available in current versions of the Linux kernel is `iptables`. It was introduced in the 2.3.x development kernels, and is part of the upcoming 2.6.x kernels.

Another type of firewall is implemented as a set of programs that intercept network traffic. These programs are commonly known as proxies. They typically are not part of the operating system kernel. Proxy-based firewalls can handle very complicated protocols, but they are slower than packet filters. Some firewalls combine features of both packet filters and proxies.

Unlike the previous version of the kernel packet filter (known as `ipfwadm`), `iptables` can keep track of open connections by creating rules dynamically. This feature is important because it improves the effectiveness of the filters we create.

Using `iptables` allows the definition of rule sets, which are named lists of rules that are searched sequentially until a rule is found that terminates the search. Three rule sets are predefined: `INPUT`, `OUTPUT`, and `FORWARD`. Others can be defined by the user, and rule sets can be linked together.

Many of the features provided by `iptables` are contained in kernel modules that can be loaded on an as-needed basis. Note that `iptables` has many features, such as Network Address Translation, which we do not describe in this document. Refer to the documents listed in the “Related Resources” on page 22 for more information.

Packet Flow

The `iptables` firewall uses one of three predefined rule sets to decide whether a packet should be discarded. First, packets are classified into one of three categories:

- Outgoing packets: packets originating on the host, destined for other computers
- Incoming packets: packets originating elsewhere, destined for the host
- Forwarded packets: packets that are forwarded by the host

FIGURE 1 illustrates how incoming and outgoing packets are processed.

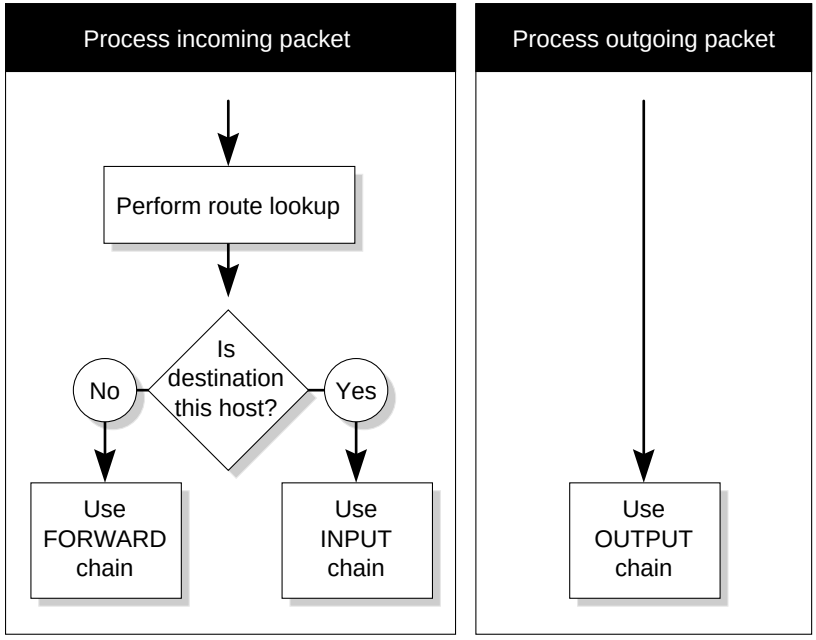

FIGURE 1 Processing Incoming and Outgoing Packets

Outgoing packets are passed to the `OUTPUT` rule set. Incoming packets are passed to the `INPUT` rule set. If the host is configured to act as a router, the forwarded packets are passed to the `FORWARD` rule set. We do not use the `FORWARD` rule set in our example because using Linux `iptables` to build a routing firewall is outside the scope of this article.

The `iptables` Command

The kernel packet filter is configured using the `iptables` command. This command has many options, and we do not use all of them in this article. The command must be run under the `root` account, because the system calls used to change filter settings in the kernel are privileged. Add firewall rules one rule at a time, using multiple `iptables` commands. For a comprehensive list of options, refer to the `iptables`(8) manual page.

The `iptables` command accepts positional parameters and flags. The command syntax can appear quite cluttered, and the resulting rule sets are far from easy to read. To give a few examples:

```
# iptables –flush        # deletes are filter rules in all
chains
# iptables –delete-chain    # deletes all user-defined chains
# iptables –append INPUT -p tcp --dport 80 -j ACCEPT # allow
incoming TCP packets to port 80
```

Dynamic Packet Filtering

Traditional packet filters do not retain any state of packets that pass through them (except for logging information). Each packet is handled independently. It is very difficult to create a secure firewall using this type of packet filter. For instance, it is impossible to block Transport Control Protocol (TCP) port scanning of hosts protected by such a firewall; therefore, we consider a packet filter to be of limited use in protecting hosts.

The current generation of Linux packet filters is able to maintain state about TCP and User Datagram Protocol (UDP) sessions, making it much easier to create a selective set of rules that block port scanning and that correctly handle difficult cases such as dynamic naming service (DNS) queries and file transfer protocol (FTP) sessions.

Maintaining session state for every connection comes at a cost: It requires system resources to maintain this state, and it leaves the system open to denial-of-service (DOS) attacks that try to overload the filtering code. The rule set design presented later in this article attempts to mitigate this problem by only maintaining state for those sessions for which it is really necessary: outbound sessions from ephemeral ports and inbound FTP sessions. This approach should make it much harder to perform denial of service (DOS) attacks against the host.

Disabling Routing

Unless you use a host as a router, make sure that the host does not forward packets. On Linux systems, this technique is achieved by making sure the following line is included in the file `/etc/sysctl.conf`:

```
net.ipv4.ip_forward = 0
```

Use the following command to verify that routing is disabled:

```
$ sysctl net.ipv4.ip_forward
```

For more information, refer to the Sun BluePrints OnLine articles titled “Securing Sun Linux Systems, Part I, Local Access and File Systems and Securing Sun Linux Systems, Part II, Network Security.”

Logging

The `iptables` firewall is somewhat lacking in its support of logging. When it generates a log message, this message is passed through the standard interface for kernel messages, which is not designed to handle a large volume of messages. The logging daemon `syslog` then writes the message to a log file. The `syslog` daemon uses a synchronous write that flushes every message to disk, a slow operation that causes message loss when the firewall rule set generates a lot of messages.

The rule set design we present in this article is designed to limit the rate at which log messages are produced. For the approach used, this is the most suitable method.

Creating a Host-Based Firewall Script

Now we are ready to start designing a rule set that protects a single host. We will write the rule set in the form of a shell script, attempting to keep the script readable by parameterizing it using shell variables.

Note – We assume that the reader is familiar with basic Bourne shell scripting.

We store the firewall script in the file named `/etc/sun_fw/fw.sh`. An additional script is needed to automatically run the firewall script at boot time.

Specify Firewall Script Parameters

We start the script with definitions that supply the script with the network parameters we will need later.

```
INTERFACE="eth0"
IPADDR="192.168.0.2"
BCASTADDR="192.168.0.255"
```

The next set of variables specifies which types of protocol sessions we want to allow inbound and outbound. In this example, we allow inbound access to SSH, HTTP, and FTP, and we allow outbound access only to DNS. This configuration reflects the necessary rules for a web and FTP server.

For desktop systems, we would need to allow outbound protocols, such as HTTP, HTTP/SSL (http), and FTP.

Also, we specify which Internet Control Message Protocol (ICMP) types can pass the firewall. Note that ICMP redirects are not allowed in this configuration. It might be necessary to add them, depending on your network configuration.

```
TCP_IN="ssh http ftp"
TCP_OUT="domain ssh http https 1024:65535"

UDP_IN=""
UDP_OUT="domain ntp"

ICMP_IN="destination-unreachable source-quench echo-request
time-exceeded parameter-problem"
ICMP_OUT="destination-unreachable source-quench echo-request
time-exceeded parameter-problem"
```

Lastly, we add shell variables that make our script more readable. `FW` is an abbreviation for the `iptables` command, and the most common command (`iptables –append`) receives its own abbreviation, `NEW`.

```
FW="/sbin/iptables"
NEW="${FW} --append"
```

Load Helper Modules

The `iptables` firewall is modular, and its functionality can be extended by loading additional modules into the kernel. A module that is commonly used is `ip_conntrack_ftp`, which inspects FTP control connections and can be used to associate FTP data connections with existing control connections. Without this support for FTP, it would be necessary to open up a range of ports for use with inbound passive FTP connections, which would limit the usefulness of the firewall.

The following lines load the module `ip_conntrack_ftp` into the kernel. The `MODPROBE` command loads Linux kernel modules, including firewall helper modules.

```
MODPROBE="/sbin/modprobe"
$MODPROBE ip_conntrack_ftp
```

Prepare the Firewall

First remove the previous firewall rules, then delete all user-defined chains. These tasks must be performed in this order, because user-defined rule chains can only be deleted if there are no references to them. By clearing all the chains first, we ensure that this is the case.

```
$FW --flush

$FW --delete-chain
```

The predefined chains (`INPUT`, `OUTPUT`, and `FORWARD`) have a default policy, which decides what to do when none of the filter rules match. We set the default policy to `DROP`, which silently discards packets. Although not strictly necessary, we set the policy for `FORWARD` as well, which ensures that the host does not act as a router.

```
for ch in INPUT OUTPUT FORWARD; do

  $FW -P $ch DROP

done
```

The kernel firewall is now in a known state, and we can start adding rules.

Establish Logging Rules

It is usually not a good idea to log every single packet that is rejected by firewall rules. Most IP networks tend to be noisy environments, and systems receive all kinds of unsolicited packets that can usually be ignored without problems. Examples of these types of packets are NetBIOS broadcasts, NTP broadcast and multicast packets, and Routing Information Protocol (RIP) broadcast or multicast packets.

On Linux, we have the additional problem that a large volume of log entries negatively impacts the performance of the system. We devise a strategy to lower the volume of packets that are logged if they are rejected. We do not currently log packets that are allowed through the firewall.

We implement a separate rule chain named `discard` to process all packets that our firewall rules do not pass. Depending on your environment, you might have to augment this rule chain to filter additional log entries.

We decide not to log any broadcast packets. In a controlled environment such as a service or DMZ (demilitarized zone) network, you might want to fine tune this strategy, depending on your policies and the risks you are mitigating.

The logging rule on the fourth line limits the number of log entries to about 10 per minute, to keep the log file manageable during a flooding attack. This rate is relatively low, and it should be adjusted to make sure that you capture as much information as possible without running the risk of overflowing your logging partition.

The fifth line handles a special case: If an attempt is made to connect to the `ident` service, the firewall replies with a TCP RST packet. We notice that delivery of email to certain remote systems is impaired because the remote system is attempting to contact the `ident` server on the test system. The addition of this rule notifies remote systems that no `ident` server exists, and speeds up the delivery of email. Any other packets are silently discarded.

```
$FW -N discard # create new rule
$NEW discard -p udp -d ${BCASTADDR} -j DROP
$NEW discard -p udp -d 255.255.255.255 -j DROP
$NEW discard -m limit --limit 10/minute --limit-burst 20 -j LOG
$NEW discard -p tcp --syn -d ${IPADDR} --dport ident -j REJECT -
-reject-with tcp-reset
$NEW discard -j DROP
```

Add Anti-Spoofing Rules

We are now ready to start adding entries to the `INPUT` and `OUTPUT` chains, which filter the inbound and outbound traffic.

These two rules pass all traffic that does not pass through the protected interface, including traffic going through the loopback interface. Note that the exclamation mark (`!`) is used by `iptables` to invert the meaning of a condition, for example, `-i ! eth0` matches all packets *not* coming in through interface `eth0`.

```
$NEW INPUT -i '!' ${INTERFACE} -j ACCEPT

$NEW OUTPUT -o '!' ${INTERFACE} -j ACCEPT
```

It’s relatively simple to forge IP addresses. One of the tasks of a firewall is to verify whether the address information contained in an IP packet is consistent with its knowledge of the network. For a host-based firewall, this knowledge is usually quite limited.

The first two rules verify that all incoming packets are actually intended for this host and do not appear to be sent from this host. The third rule disallows traffic that uses the address range assigned to the loopback interface. Similar rules can be added to filter out other address ranges, such as those defined in RFC 1918. We do not implement any outbound anti-spoofing rules.

```
$NEW INPUT -s ${IPADDR} -j discard
$NEW INPUT -d '!' ${IPADDR} -j discard
$NEW INPUT -s 127.0.0.0/8 -j discard
```

Add Dynamic Rules

The following two rules handle packets that belong to sessions for which the `iptables` firewall maintains state. When the firewall passes the initial packet of a session, it stores information about the session that enables it later to match packets to this session.

A packet matches the `ESTABLISHED` criterion if it is part of an existing TCP connection or UDP session. It matches the `RELATED` criterion if it is associated with an existing connection. For the purpose of this rule set, `RELATED` matches FTP data connections associated with existing FTP control connections. Also, `RELATED` matches certain ICMP packets that carry information about individual sessions.

```
$NEW INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

$NEW OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Note – The capability to associate certain ICMP packets is often important because it allows Path MTU discovery to work correctly.

Manage Inbound Sessions

Stateful packet filtering uses a certain amount of memory per active connection. To limit the memory impact, we use stateful packet filtering only where necessary. This strategy limits the impact during a flooding or DOS attack. For simple inbound services like HTTP and secure shell (SSH), we do not need to use stateful packet filtering. The FTP protocol is a different matter, because passive mode FTP does not use a fixed port number for the data connection.

The firewall rule for the FTP case creates a new entry in the stateful packet filtering table maintained in the kernel. The rules matching `ESTABLISHED` and `RELATED` traffic process the remaining packets of a legitimate FTP session.

For all other protocols, we use static rules. Note that we do not limit the source port for these rules to unprivileged port numbers, as is commonly done for a simple packet filter, because there is no need for that. One disadvantage of using static rules for inbound is that the ports associated with services can be probed using stealth scans. This disadvantage is usually not a problem, because we are not trying to keep the existence of these services a secret.

For most protocols, we have to add one static rule for incoming traffic and one for outgoing traffic. In the case of FTP, we add only one rule, because the stateful filtering automatically takes care of the remaining traffic. The individual rules are generated by iterating over all the protocols we want to allow in.

```
for port in ${TCP_IN}; do
  case "${port}" in
  ftp) $NEW INPUT -p tcp --dport ${port} --syn -m state --state
NEW -j ACCEPT
       ;;
  *)   $NEW INPUT -p tcp --dport ${port} -j ACCEPT
       $NEW OUTPUT -p tcp '!' --syn --sport ${port} -j ACCEPT
       ;;
  esac
done
```

We do not recommend the use of standard FTP, except for anonymous access, because the protocol exchanges authentication information as plain text. We recommend OpenSSH as an alternative.

The rules for inbound UDP sessions are exactly analogous to the ones for inbound TCP. Again, we do not use stateful packet filtering.

```
for port in ${UDP_IN}; do
  $NEW INPUT -p udp --dport ${port} -j ACCEPT
  $NEW OUTPUT -p udp --sport ${port} -j ACCEPT
done
```

Manage Outbound Sessions

For outbound TCP sessions, we use stateful packet filtering because the security gain is considerable: Attackers cannot probe for open ports, and we can actually support normal FTP without resorting to special tricks.

```
for port in ${TCP_OUT}; do
  $NEW OUTPUT -p tcp --dport ${port} --syn -m state --state NEW
-j ACCEPT
done
```

For UDP, the security gains are even more substantial. Because the filter matches requests with responses, it blocks unsolicited packets used for overflow attacks on DNS and Network Transport Protocol (NTP) servers that only serve internal networks.

```
for port in ${UDP_OUT}; do
  $NEW OUTPUT -p udp --dport ${port} -m state --state NEW -j
ACCEPT
done
```

Manage ICMP

Certain types of ICMP packets are necessary for the correct functioning of TCP and UDP. We pass only the types required. The `iptables` stateful packet filtering automatically passes ICMP packets that are related to existing TCP or UDP sessions (use the `RELATED` rule when matching packets). Because we choose not to use stateful filtering for all traffic, we cannot rely on this mechanism to pass all the ICMP packets we need, so we are required to pass them explicitly. If the choice were made to use stateful inspection for all traffic, this would be unnecessary.

```
for t in ${ICMP_IN}; do
  case "${t}" in
  echo-request)
    $NEW INPUT -p icmp --icmp-type echo-request -j ACCEPT
    $NEW OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
    ;;
  *)
    $NEW INPUT -p icmp --icmp-type ${t} -j ACCEPT
    ;;
  esac
done

for t in ${ICMP_OUT}; do
  case "${t}" in
  echo-request)
    $NEW OUTPUT -p icmp --icmp-type ${t} -m state --state NEW -j
ACCEPT
    ;;
  *)
    $NEW OUTPUT -p icmp --icmp-type ${t} -j ACCEPT
    ;;
  esac
done
```

Discard Other Traffic

All other incoming packets are invalid, so we jump to the discard chain, which takes care of logging.

```
$NEW INPUT -j discard
```

For outgoing packets that we discard, we do not create a separate logging chain. Instead of silently discarding the packets, we have the firewall send either a TCP, RST, or ICMP destination-unreachable packet. This allows applications on our host to quickly determine that the traffic is not allowed through the firewall.

```
$NEW OUTPUT -m limit --limit 10/minute --limit-burst 20 -j LOG
$NEW OUTPUT -p tcp -j REJECT --reject-with tcp-reset
$NEW OUTPUT -j REJECT
```

Installing the Firewall Rule Set

You can install the firewall rule set manually by running it as a shell script. After changing the firewall rules, reinstall them by simply running the script again. It might be a good idea to preserve old versions of your firewall script, so you can reverse changes.

To verify which firewall rules are currently active, enter the following:

```
$ iptables -L
```

Refer to the `iptables`(8) manual page for display options.

With the release of this article, Sun released packages (RPM files) for the Linux platforms supported on Sun hardware. Sun keeps these packages current. The packages contain the sample firewall script described in this document and a system dependent script that installs this firewall rule set at boot time, before any network services are configured.

Consult the `README` file included with the package for details on installation and activation.

Removing the Firewall Rule Set

The following script is useful when you want to remove the firewall rule set from the kernel:

```
#!/bin/sh
FW="/sbin/iptables"
$FW –flush  # clear all firewall chains
$FW –delete-chain  # delete all user-defined chains

# set the default action of the built-in chains to ACCEPT, the
default at boot time
for ch in INPUT OUTPUT FORWARD; do
  $FW -P $ch ACCEPT
done
```

We suggest storing this script in `/etc/sun_fw/removefw.sh`.

Verifying Firewall Services

After installing a host, it is prudent security practice to verify that only those services that are intended to be exposed to the network are exposed. Most Linux distributions provide the open source utility `nmap`, which is a general-purpose network scanner. The `nmap` utility lists ports that are open. The utility has many options, but for our purposes only two are necessary: `-sT` and `-sU`, which scan for open TCP and UDP ports, respectively.

Scanning for open TCP ports is simple. Performing a TCP port scan using `nmap` shows the following output.

```
[root@scl1 root]# nmap -sT lsg-parity4

Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Interesting ports on lsg-parity4 (192.168.0):
(The 1526 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
37/tcp     open        time
111/tcp    open        sunrpc
1014/tcp   open        unknown
6000/tcp   open        X11
22289/tcp  open        wnn6_Cn
32770/tcp  open        sometimes-rpc3

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

It is clear that this host exposes too many services to the network. On a production system, most of these services are unnecessary and should be turned off, and the software that implements them removed.

After installing the firewall configuration, we have only two services exposed:

```
[root@scl1 root]# nmap -sT lsg-parity4

Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Interesting ports on lsg-parity4 (192.168.0):
(The 1538 ports scanned but not shown below are in state: filtered)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
80/tcp     closed      http
113/tcp    closed      auth

Nmap run completed -- 1 IP address (1 host up) scanned in 139
seconds
```

The host does not run a web server, but our firewall allows web server traffic (`http`) to pass through, so `nmap` is able to get a response from the TCP stack. No configuration changes were made to the host, except for the installation of the firewall.

For more information about `nmap`, refer to the `nmap`(1) manual page or the web site at `http://www.insecure.org/nmap/`.

If the behavior of the firewall is not what you are expecting, the following might be helpful. Enter the following command:

```
# iptables -vL
```

This command produces a list of firewall rules that includes the number of times each rule has matched. This list can be useful when determining which rule in the firewall is causing unexpected behavior.

The `iptables` firewall we created passes log entries to the system log daemon, which by default writes them to the file `/var/log/messages`. The `nmap` probe causes the firewall to generate lines of the form:

```
Oct 28 11:36:42 testhost kernel: IN=eth0 OUT=
MAC=00:c0:f0:3e:28:b9:00:03:47:31:89:41:08:00 SRC=192.168.44.12
DST=192.168.44.105 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=57606 DF
PROTO=TCP SPT=53360 DPT=374 WINDOW=5840 RES=0x00 SYN URGP=0
Oct 28 11:36:48 testhost kernel: IN=eth0 OUT=
MAC=00:c0:f0:3e:28:b9:00:03:47:31:89:41:08:00 SRC=192.168.44.12
DST=192.168.44.105 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=29197 DF
PROTO=TCP SPT=53569 DPT=7009 WINDOW=5840 RES=0x00 SYN URGP=0
Oct 28 11:36:54 testhost kernel: IN=eth0 OUT=
MAC=00:c0:f0:3e:28:b9:00:03:47:31:89:41:08:00 SRC=192.168.44.12
DST=192.168.44.105 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=36024 DF
PROTO=TCP SPT=53786 DPT=1414 WINDOW=5840 RES=0x00 SYN URGP=0
```

Information like this can be invaluable. Ideally, this information should be logged to a secure host, and it should be monitored either automatically or periodically, as determined by your security policies and procedures.

Firewall Script Sample

Throughout this article, we use excerpts of the following firewall script. This section provides the complete script.

CODE EXAMPLE 1 Firewall Script

```
INTERFACE="eth0"
IPADDR="192.168.0.2"
BCASTADDR="192.168.0.255"

TCP_IN="ssh http ftp"
TCP_OUT="domain ssh http https 1024:65535"
UDP_IN=""
UDP_OUT="domain ntp"
ICMP_IN="destination-unreachable source-quench echo-request time-
exceeded parameter-problem"
ICMP_OUT="destination-unreachable source-quench echo-request
time-exceeded parameter-problem"

FW="/sbin/iptables"
```

CODE EXAMPLE 1 Firewall Script *(Continued)*

```
NEW="${FW} --append"
MODPROBE="/sbin/modprobe"

$MODPROBE ip_conntrack_ftp

$FW --flush
$FW --delete-chain

for ch in INPUT OUTPUT FORWARD; do
  $FW -P $ch DROP
done

$FW -N discard # create new rule
$NEW discard -p udp -d ${BCASTADDR} -j DROP
$NEW discard -p udp -d 255.255.255.255 -j DROP
$NEW discard -m limit --limit 10/minute --limit-burst 20 -j LOG
$NEW discard -p tcp --syn -d ${IPADDR} --dport ident -j REJECT --
reject-with tcp-reset
$NEW discard -j DROP

$NEW INPUT -i '!' ${INTERFACE} -j ACCEPT
$NEW INPUT -s 127.0.0.0/8  -j ACCEPT
$NEW OUTPUT -o '!' ${INTERFACE} -j ACCEPT

$NEW INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
$NEW OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

for port in ${TCP_IN}; do
        case "${port}" in
        ftp) $NEW INPUT -p tcp --dport ${port} --syn -m state --
state NEW -j ACCEPT
             ;;
        *)   $NEW INPUT -p tcp --dport ${port} -j ACCEPT
             $NEW OUTPUT -p tcp ! --syn --sport ${port} -j ACCEPT
             ;;
        esac
done
for port in ${UDP_IN}; do
        $NEW INPUT -p udp --dport ${port} -j ACCEPT
        $NEW OUTPUT -p udp --sport ${port} -j ACCEPT
done
for port in ${TCP_OUT}; do
       $NEW OUTPUT -p tcp --dport ${port} --syn -m state --state
NEW -j ACCEPT
done
```

CODE EXAMPLE 1 Firewall Script *(Continued)*

```
for port in ${UDP_OUT}; do
        $NEW OUTPUT -p udp --dport ${port} -m state --state NEW -
j ACCEPT
done
for t in ${ICMP_IN}; do
        case "${t}" in
        echo-request)
            $NEW INPUT -p icmp --icmp-type echo-request -j ACCEPT
            $NEW OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
            ;;
        *)
            $NEW INPUT -p icmp --icmp-type ${t} -j ACCEPT
            ;;
        esac
done
for t in ${ICMP_OUT}; do
        case "${t}" in
        echo-request)
                $NEW OUTPUT -p icmp --icmp-type ${t} -m state --
state NEW -j ACCEPT
                ;;
        *)
                $NEW OUTPUT -p icmp --icmp-type ${t} -j ACCEPT
                ;;
        esac
done

$NEW INPUT -j discard

$NEW OUTPUT -m limit --limit 10/minute --limit-burst 20 -j LOG
$NEW OUTPUT -p tcp -j REJECT --reject-with tcp-reset
$NEW OUTPUT -j REJECT
```

About the Author

Gé Weijers is a staff engineer working for the Sun Linux Security Group in Columbus, Ohio. He specializes in Security Engineering, the construction of systems that perform their intended function in a dependable way.

Recently, Gé worked on the design and delivery of software that reduces the vulnerability of Linux-based Sun products to network-based attacks, and he worked on the development of new initiatives within Sun to improve the security of Sun products.

Prior to Sun, Gé worked for Progressive Systems, Inc. of Columbus, Ohio. The main product was the Phoenix Adaptive Firewall Appliance. He designed the cryptographic tools used to allow secure remote configuration of firewalls using a user interface based on Java™ technology.

Some of Gé's professional interests are security engineering, cryptography, protocol design, and provable security.

Related Resources

Publications

- Anderson, Ross J. *Security Engineering, A Guide to Building Dependable Distributed Systems*, Wiley 2001, ISBN 0-471-38922-6.
- Brunette, Glenn, Hullhorst, Michael, and Weijers, Gé. “Securing Sun Linux Systems, Part I, Local Access and File Systems,” Sun BluePrints OnLine, July 2003, `http://www.sun.com/solutions/blueprints/0703/817-3420.pdf`.
- Brunette, Glenn, Hullhorst, Michael, and Weijers, Gé. “Securing Sun Linux Systems, Part II, Network Security,” Sun BluePrints OnLine, July 2003, `http://www.sun.com/solutions/blueprints/0703/817-3421.pdf`.
- Englund, Martin. “Securing Systems with Host-based Firewalls - Implemented With SunScreen™ Lite 3.1 Software,” Sun BluePrints OnLine, September 2001, `http://www.sun.com/solutions/blueprints/0901/sunscreenlite.pdf`.
- Rekter, Y., et. al. *Address Allocation for Private Internets*, RFC 1918.
- Zwicky, Elizabeth D., Cooper, Simon, and Chapman, Brent D. *Building Internet Firewalls*, O'Reilly 2000, ISBN 1-56592-871-7.

Web Sites

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

- Netfilter web site: `http://netfilter.org/`
- NMAP, Nmap web site: `http://www.insecure.org/nmap/`

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`