

# FOCUS on Linux: Securing Linux Part Two

Dale Coddington 2000-04-25

## Securing Linux Part II

Securing Linux for the Net - Part II

Dale Coddington <[dalec@eEye.com](mailto:dalec@eEye.com)>

### I. Introduction

Part I of this article focused on basic methods to secure a default Linux installation. Aside from SSH, no additional software was installed on the machine to increase security. This article will examine some additional tools that can be installed to increase the overall security posture of a Linux system.

Most of the tools that will be discussed in this paper include excellent documentation. Therefore, this paper will serve as an overview of these tools and not as a "how to".

Disclaimer: Properly securing a machine can be a daunting task, especially with the amount of new exploits surfacing on an almost daily basis. This paper is not meant to be a total security solution, but should instead be regarded as a guide to some of the security measures that can be taken to protect a machine from intruders.

### II. Kernel Modifications - <http://www.openwall.com/linux/>

Many kernel level modifications are available to help increase system security. Even if an attacker was to gain root access to a machine it would be difficult, if not impossible, to circumvent some of these security measures. Naturally, in order to take advantage of these tools you must be familiar with rebuilding a kernel. Typically the modifications come in the form of a patch to apply to the Linux source code. After applying the patch, a fresh kernel is then built. Kernel patches tend to favor more recent kernels.

One of the most popular and widely used set of patches is from the Openwall Project, written by Solar Designer. This patch adds a series of security related features to the Linux kernel that can be configured in the "security options" section during kernel configuration. Some of the features of Solar Designers' patch include:

- o Non-executable user stack area

One of the most popular and widely used exploits today is the common buffer overflow. A buffer overflow occurs when excess data is stuffed into a buffer and the function return address is altered to point at shell code which will be executed, typically spawning a shell with the access level the particular program runs at. This patch will generally defeat buffer overflows by disallowing the execution of code on the stack.

- o Restricted links in /tmp

A popular method for local users to gain elevated system privileges, or just to wreak havoc, is through what is known as a "/tmp symlink" attack. On Unix systems, the /tmp directory is world writable and is used to store temporary files generated by various programs. For example, when root runs the program "foo" it might create a temporary file called foo.tmp in /tmp. A malicious local user could create a symlink to /kernel from /tmp/foo.tmp. When foo is run and creates the temporary file, the link may be followed and /kernel overwritten. Another form of this attack is if the user creates a link from /tmp/foo.tmp to /etc/shadow. They can then cat /tmp/foo.tmp which blindly follows the link to /etc/shadow and they can then view the contents of the shadow password file, or potentially add entries to the shadow file. This patch defeats this by not allowing users to create hard or soft links to files they do not own from a directory containing the sticky bit (+t).

- o Restricted /proc

By default a regular user on a Linux system can view running processes that are owned by other users. This is potentially valuable information. A regular user really has no need to know what other processes are running on the system in question. This patch restricts users to viewing only processes owned by them.

Obviously adding kernel patches does require some extra work on the part of the administrator. The patches must be applied to the original kernel source and a fresh kernel must be built to enable the new functionality offered by the patch. The extra work required is, for some, worth the peace of mind the added security will offer.

### III. Additional Logging Measures

By default Linux does not log all TCP connections, but rather only connections to "well-known" ports, or those ports listed in the /etc/services file. In this day and age this is woefully inadequate. Linux does not include any means to log any additional ports besides those listed in /etc/services. However, there are tools that can be added to log any TCP connection to any port. Although extended logging mechanisms can make it easier to determine if a system is under attack, it can also make it more difficult at the same time by vastly increasing the amount of logs generated, and could even in some cases lead to a denial of service by overfilling the drive where logging takes place.

- o tcplogd - <http://www.kalug.lug.net/tcplogd/>

tcplogd is a daemon that monitors and logs any tcp connection to a machine, via the syslog facility. tcplog will detect most stealth SYN scans in use by popular scanners like nmap and queso. The tcplogd installation includes a configuration file, tcplogd.cf. This files allows the system administrator to configure what packets tcplogd will actually log to the system log files. As an example, you can exclude the logging of TCP connections from a trusted host. It may be desirable to disable logging of the machine used to scan machines for vulnerabilities on your network. This would greatly reduce

the amount of traffic generated in the log files. For example, if you run the following command against the machine running tcplogd:

```
$ nmap -p1-1024 foo.com
```

1024 entries would appear in /var/log/messages, 1 entry for every single connection attempt.

- o icmpinfo - <ftp://ftp.sunet.se/pub/network/monitoring/icmpinfo/>

icmpinfo is a useful tool, included with the Slackware distribution, that logs all icmp traffic via the syslog facility. Many options are available to the icmpinfo program to determine the verbosity of messages. As an example, icmpinfo can be used to generate an ascii dump of any icmp traffic received:

```
regret:~# /usr/sbin/icmpinfo -vvv
```

```
icmpinfo: Icmp monitoring in progress...
Apr 18 08:51:51 ICMP_Echo < 216.32.49.18 sz=64(+20)
0000 : 4500 0054 19A2 0000 FF01 8F9D D820 3112 E..T..... 1.
0010 : D820 3116 0800 0614 F215 0000 CF73 FD38 . 1.....s.8
0020 : 4226 0600 0809 0A0B 0C0D 0E0F 1011 1213 B&.....
0030 : 1415 1617 1819 1A1B 1C1D 1E1F 2021 2223 ..... !"#
```

### Important Note

A program exists on the internet called tcplog.c that basically works the same way as tcplogd . There exists a possible buffer overflow in this package and it's use should be avoided.

- o The Abacus Project - <http://www.psonic.com/abacus/>

The Abacus Project is a suite of tools, designed by Craig Rowland, that includes portscan detection (Portsentry) and log monitoring (Logcheck). These tools are both very powerful and easy to configure and compliment each other perfectly.

- Portsentry

Portsentry is a portscan detection tool that can be run on a host as an additional security measure. Portsentry listens on ports read from a configuration file and can take action, as defined in the configuration file. The response Portsentry takes may be as simple as adding the the IP Address of the source of the scan to the /etc/hosts.deny file or even dropping the actual route back to the originating computer. To the attacker, this would make it appear as if the target machine dropped its connection. Portsentry can also be configured to ignore connections from trusted hosts. One thing to keep in mind is that it is rather trivial for an attacker to spoof an attack so it appears to be coming from a trusted host. It is really up to

the system administrator to determine if any hosts should be ignored or not. Also, if Portsentry is being run on a remotely administered machine, and no hosts are specified in the portsentry.ignore file it is possible to lock yourself out of the machine by simply running a portscan against it if Portsentry is configured to drop the route of an "attacker".

Portsentry only listens on those ports the administrator wishes it to. Typically this can be set to listen to commonly trojaned ports, or ports that run known-vulnerable services.

- Logcheck

Logcheck is a program that is used to help in the processing of Unix log files. The program contains configuration files that can be set up to watch for certain events in the log files, and mail a report of those events to the system administrator. This can make the job of monitoring log files much easier for the administrator. Rather than wading through large quantities of log files looking for suspicious activity, Logcheck can do the same, at set intervals when run from the cron facility. Logcheck is also very flexible in that certain events can be ignored in reports. As an example, there is really no reason to report every time a user on the particular machine sends or receives a piece of mail.

```
-- begin sample logcheck report --
```

```
Active System Attack Alerts
```

```
=====
```

```
Apr 18 09:09:29 regret portsentry[57]: attackalert: Connect from host:
badguys.com/192.168.1.1 to TCP port: 31337
```

```
Apr 18 09:09:29 regret portsentry[57]: attackalert: Host 192.168.1.1 has
been blocked via wrappers with string: "ALL: 192.168.1.1"
```

```
Apr 18 09:09:29 regret portsentry[57]: attackalert: Host 192.168.1.1 has
been blocked via dropped route using command:
"/sbin/ipfwadm -I -i deny -S 192.168.1.1 -o"
```

```
Security Violations
```

```
=====
```

```
Apr 18 09:09:29 regret portsentry[57]: attackalert: Connect from host:
badguys.com/192.168.1.1 to TCP port: 31337
```

```
Apr 18 09:09:29 regret portsentry[57]: attackalert: Host 192.168.1.1 has
been blocked via wrappers with string: "ALL: 192.168.1.1"
```

```
Apr 18 09:09:29 regret portsentry[57]: attackalert: Host 192.168.1.1 has
been blocked via dropped route using command:
"/sbin/ipfwadm -I -i deny -S 192.168.1.1 -o"
```

```
Unusual System Events
```

```

=====
Apr 18 09:09:29 regret tcplog: port 31337 connection attempt from
badguys.com
Apr 18 09:09:29 regret portsentry[57]: attackalert: Connect from host:
badguys.com/192.168.1.1 to TCP port: 31337
Apr 18 09:09:29 regret portsentry[57]: attackalert: Host 192.168.1.1 has
been blocked via wrappers with string: "ALL: 192.168.1.1"
Apr 18 09:09:29 regret portsentry[57]: attackalert: Host 192.168.1.1 has
been blocked via dropped route using command:
"/sbin/ipfwadm -I -i deny -S 192.168.1.1 -o"

-- end sample logcheck report --

```

As we can see from the sample report, an attacker at badguys.com attempted to probe our system for Netbus on TCP port 12345. But since we have Portsentry listening for connections on that port, the connection was detected and the the route back to the attacker at badguys.com was dropped using the ipfwadm command.

Dropping the route to a host for a simple probe such as this may seem a bit fascist. It is entirely up to the system administrator how these types of probes should be handled.

It should be noted that if the box is rebooted, all routes dropped through portsentry will be re-enabled.

#### IV. Firewalling

Although these papers have been focusing on securing a host and not actually setting up a firewall, this does not mean that you cannot take advantage of some of the built-in firewalling capabilities of Linux. As an example, a machine placed on the outside of a firewall may take advantage of these capabilities. In some cases it is desirable to place machines such as web servers or ftp servers outside of a firewall if they contain only non-critical information. If an attacker was to gain unauthorized access to one of these servers, they would still have the firewall to contend with. If this same machine was sitting on the inside of a firewall and an attacker was able to successfully gain unauthorized access, the rest of your network would then be at their fingertips.

Linux kernels 2.1.102 and above implement a packet filtering mechanism called 'ipchains'. Prior kernels used a mechanism called 'ipfwadm' which is not as flexible as ipchains.

Packet filtering is a very simple concept. As each network packet arrives at the firewalled machine, the contents of the packets headers are checked against a set of rules. These rules determine how the packet will be handled. Typically a packet filtering firewall will contain two ethernet cards, one on the "outside" and one on the "inside". That does not mean that a machine with a single ethernet card cannot benefit from ipchains.

Ipchains can be a very powerful and flexible tool if implemented properly. The following is a small sampling of what can be accomplished with ipchains. First use the list (-L) command to see if any rules are currently in place.

```
# ipchains -L
Chain input (policy ACCEPT):
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
```

From the output above it can be noted that there are actually three chains in place. As their names imply the input chain specifies rules for ingress packets, the forward chain specifies rules for forwarded packets (for a machine with two or more ethernet interfaces), and the output chain specifies rules for egress packets. It can also be noted that the default behavior for all three chains, or rule sets, is to accept anything. Basically ipchains are doing absolutely nothing at this point to protect the system.

For this example we have determined we do not want anyone to be able to send an icmp echo request, or ping, to our machine. The following rule would reject all incoming echo requests:

```
# ipchains -A input -p icmp --icmp-type echo-request -i eth0 -j REJECT -l
      (1) (2)   (3)           (4)           (5)   (6)   (7)
```

#### ipchains flags explained

1. (A)ppend a new rule
2. The chain to apply the rule to, in this case the rule will apply to ingress packets
3. Protocol to apply the rule to
4. ICMP Type, in this case all icmp echo requests will be blocked
5. Interface name
6. Target, or what should actually be done with the packet in question
7. Log all packets matching the rules criteria to system log file

Now if we run the list command once again:

```
# ipchains -L
Chain input (policy ACCEPT):
target      prot opt      source                destination            ports
REJECT      icmp ----l- anywhere             anywhere
echo-request
Chain forward (policy ACCEPT):
Chain output (policy ACCEPT):
```

As you can see the input chain now has a new rule added that will reject all incoming icmp echo requests and log the request, regardless of the source of the request.

Now if someone were to ping the host on which this rule was in place, the host would not generate an icmp echo reply and would make a log entry. The log entry has been split up for easier reading:

```
Apr 19 18:03:33 foo kernel: Packet log: input REJECT eth0 PROTO=1
      (1)      (2) (3)      (4)      (5)  (6)  (7)  (8)

208.225.201.200:8 216.32.49.22:0 L=84 S=0x00 I=9994 F=0x0000 T=55
      (9)      (10)      (11)  (12)  (13)  (14)  (15)
```

ipchains log entry explained

1. System timestamp
2. Hostname
3. Logging facility
4. Indicates message was generated from ipchains
5. The name of the chain in which the rule matched
6. The target of the matching rule
7. The interface the packet arrived on
8. The protocol number of the packet (see /etc/protocols)
9. Source IP Address and source port
10. Destination IP Address and destination port
11. Length of the packet
12. TOS (Type of service)
13. IP Identification
14. Fragment offset (used to reassemble fragmented packets)
15. TTL (Time to live) of the packet

Now if we change our minds, and decide that we want to allow icmp echo requests, the rule could be removed with:

```
# ipchains -F input
```

This will flush (delete) all the rules for the specified chain. If we had more than one rule in place on the input chain we could have also deleted just the first rule:

```
# ipchains -D input 1
```

This is only an extremely small sample of what ipchains can do to help defend a system from attackers. It is highly recommended that one reviews the ipchains man page before putting any rules into place, especially on a remotely administered machine, as it is very easy to lock yourself out if implementing a rule incorrectly.

## V. Staying Current

Although it can be a time consuming task, when responsible for maintaining the security of many servers, it is extremely important to ensure that each system is patched for current vulnerabilities. One thing to bear in mind is that what is secure today is not necessarily secure tomorrow. Be sure to follow the proactive philosophy of security and stay current with new vulnerabilities and attack methods.

It is always a good idea to have a system set up in a closed laboratory environment to try new attacks against. Knowing what the "fingerprints" of attacks actually look like will make it much easier to determine what is happening on your "live" systems.

Be sure to subscribe to any security related mailing lists for whatever flavor(s) of Operating Systems you are responsible for protecting.

## VI. Closing

Over the course of two papers many methods have been discussed to secure a default Linux installation. One thing to bear in mind is there is no "fail-safe" checklist of measures that can be used to guarantee the security of a system, whether it be Linux, Solaris, FreeBSD, or any other operating systems.

Typically those responsible for securing a system will develop their own checklists and methods. The more you learn about security, and the methods attackers use to circumvent that security, the better you will be able to secure a machine to a comfortable level. Remember, the only machine that is 100% secure is that which is disconnected from the network and locked in a steel vault with an armed guard.

### Relevant Links

#### [OpenWall Project](#)

*OpenWall*

#### [Abacus Project](#)

*Craig Rowland*

#### [tcplogd](#)

*tcplogd*

[Privacy Statement](#)

Copyright 2006, SecurityFocus