The Linux Instrusion Detection System (LIDS)

Febuary 6, 2001

Contents

1 Introduction to LIDS

With the rapid pace of development and open source nature of Linux, programs are often evaluated for security vulnerabilities. Between the time the known security vulnerabilities are found, additional protection is available to provide an extra layer of security, until the system can be updated.

Since Linux is an art of the open source community, security holes may be found more easily but can also be patched just as quickly and easily. But when the hole is disclosed to the public, and the administrator is unable to patch the hole, it could potentially compromise your system. With the typical Linux systems, a cracker has absolute control if superuser access is gained. With the added protection of LIDS, this and many other potential problems can be reduced.

LIDS provides the ability to control all access to files, processes, binaries, memory, raw devices, drives, etc. One of the main features of LIDS is protection from the superuser, known on a Linux system as the root user.

NOTE: LIDS requires advanced administration skills to manage properly and therefore should not be modified by inexperienced users.

The root user has control over every single aspect of the system. They can mount and unmount drives, delete and create files, remove users, access the database, edit the Web page, shutdown the system, etc. So you can see the possible security hazard here. If someone managed to gain root access, the entire system could be put into the crackers control. Here is a number of security enhancements LIDS has to protect the system from this threat.

- Every single file can be protected. Giving each file its own set of read, write, or append rules that even the root user must obey. For example, if you set your log files to append only, no one could go in and delete any trace of themselves on the system. You can set the login binary as read-only and it can not be replaced. Even if there was a possible way to overwrite the file LIDS would know it's not the same file because it indexes the files by their inodes, not their file names.
- Files can also be completely hidden from view and only be accessible by specific programs. For example, if you want to protect your Apache SSL server key from everyone including root, you can hide the file so to every user, including root, it doesn't exist, but at the same time it allows Apache to have full access to the file so it can get the information it needs from it.
- LIDS can also protect processes from being killed by the root user. This could be used to protect your database server, your Web server, your mail server, etc. from being taken off-line by an intruder.
- You can have full control of the Linux kernel "capabilities". The current Linux capabilities control what a process can and can't do. Changing these capabilities gives you more control over your system. By setting the capabilities to your needs you can prevent all users from rebooting the system, mounting and unmounting disks, changing network settings, `/dev` control, ownership control, loading and unloading of kernel modules, and many others.
- Root has the ability to turn LIDS off locally for just the current session or globally. This can be configured so it can only be done locally, and/or remotely. It also requires a password which is protected by Ripe MD-160 encryption.
- A built in port scanner allows you to disable promiscuous mode and still detect port scans.
- All attempts on the system are logged and if any user tried to break one of the LIDS rules, an e-mail is immediately sent to a predefined e-mail address. (A cell phone or a pager can be configured to be alerted when this happens also so you know when someone is making an attempt on your system.)

Some minor drawbacks to this increased method of security is it could hinder the use of certain programs by denying them access to needed files if configured incorrectly. It also makes it more difficult to administer the system from the console but the included GD WebTool includes enhancements that integrate will with LIDS.

2 Using LIDS

LIDS is always running on your Lockbox. If you will be doing your administration via the GD WebTool you can skip this section, but it's suggested reading anyway.

Minimal maintenance is required to keep LIDS running. Management of LIDS on servers that are co-located with Guardian Digital is included with your support contract.

You may sometimes need to change the configuration or add new packages requiring you to disable LIDS. The GD WebTool will automatically enable and disable LIDS while you administer the system. For adminstration from a shell, a program called `lidsadm` is used to interface with LIDS.

First you have to disable LIDS. After logging in as root type:

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

This will prompt you for your password. After entering your password LIDS is disabled for the current session you are in. This method will still apply all the LIDS resource settings and rules to every other user on the system while you administer the system. Optionally, issuing:

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

will disable LIDS globally. While in this mode no LIDS rules will be applied to any user or resource. Use this with caution. Once you have LIDS turned off you may configure your capabilities, file permissions, resource permissions, etc. If you changed the LIDS configuration while LIDS was turned off you will need to reload the configuration file into LIDS. Before turning LIDS on enter this:

```
/sbin/lidsadm -S -- +RELOAD_CONF
```

This will make sure you have the latest configuration loaded into LIDS. It is suggested you run this command every time you make a change to the LIDS configuration. To turn LIDS protection back on after administration simply issue:

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

or to enable it globally:

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

Your system is now protected again by LIDS. When enabling, disabling and reloading the configuration information with lidsadm you will be prompted for a password every time. You will see the following message:

```
SWITCH

WARNING: Only system administrators should enable/disable LIDS. Disabling LIDS can open your Lockbox
to possible attacks.  Make sure you read the LIDS section in your included manual before manually
changing options in LIDS. Incorrect configurations can have drastic effects.

enter password:
```

At this point you can enter in your password.

2.1 Using the lidsadm Utility

The lidsadm utility is a small program you will use to administer your LIDS configuration. It stores all configuration information in `/etc/lids/lids.conf`. If you are using the GD WebTool for administering LIDS you do not need to use lidsadm.

Some basic lidsadm options are as follows:

```
/sbin/lidsadm -A Add a new entry

/sbin/lidsadm -D Delete an entry

/sbin/lidsadm -Z Delete all entries

/sbin/lidsadm -U Update all entries

/sbin/lidsadm -L List current entries, requires LIDS to be turned off

/sbin/lidsadm -P Creates a new password.  It will store the password in Ripe MD-160 encryption

/sbin/lidsadm -S Switch LIDS on/off and capabilities

/sbin/lidsadm -r View current status of LIDS

/sbin/lidsadm -h Help
```

The next section will contain more detailed information about the lidsadm options

2.2 Adding an Entry

Using this option allows you to add a new item to the LIDS config. You have the options to add a single file with an attribute, give a file permission to override another files permissions, and change the capabilities of a file.

```
lidsadm -A [-s subject] -o object [-t] -j TARGET
```

To protect a file enter the filename and path using the *-o* flag, followed by the attribute, READ, WRITE, IGNORE, DENY, or APPEND under the *-j* attribute. If your object is a capability setting you need to use the *-t* flag to tell lidsadm it's a special option. *-s* is used to point the object to a subject. In the case of capabilities you, are pointing a capability to the subject or giving the subject the capability. Same idea with file protections. If you deny access to a file but want the subject to use it, you point to the denied file(*object*) to the file to give access to(*subject*) then tell it what kind of access to give it *-j*. Here's an example of protecting a file:

```
lidsadm -A -o /path/to/protected_file -j DENY
```

Now to give a binary full access to the file that was denied to everyone else:

```
lidsadm -A -s /path/to/binary \
           -o /path/to/protected_file -j WRITE
```

We also want to give the binary the capability to chown, which has been disabled earlier by LIDS:

```
lidsadm -A -s /path/to/binary \
        -t -o CAP_CHOWN -j INHERIT
```

When changing a files capabilities we use INHERIT or NO_INHERIT instead of the READ...APPEND commands. Using INHERIT gives the file access to the capability while the NO_INHERIT turns off the files abilities to use the given capability. In a later section capabilities are explained in more detail. In the next session an example of a package being protected is given.

NOTE: Don't forget to do a *lidsadm -S – +RELOAD_CONF* after changes were made so they take effect when you reload LIDS.

2.3 Deleting an Entry

Deleting an entry is an extremely simple task and there is no need to go into great detail. If there is a file you no longer want to be protected or wish to change protection on, you need to delete the entry from the LIDS config. Simply issue the following command to accomplish this task:

```
lidsadm -D [-s file] [-o file]
```

and the file will be removed from the configuration. You can now enter new attributes for the file, if you like.

2.4 Deleting and Updating All Entries

Lidsadm gives you the ability to delete and update all the file entries in your configuration. Issuing:

```
lidsadm -Z
```

will delete every entry in your LIDS configuration and you will be starting with a clean configuration file. The original configuration shipped on your box is stored in */usr/bin/lids_default_config/* and can be executed to revert LIDS back to it's original configuration.

Updating all the file entries works a little differently. The configuration files are linked to LIDS by their inode number, not their filename. If a file gets deleted and replaced later it may not be protected by lids because of the inode change. By issuing:

```
lidsadm -U
```

lidsadm will go through your configuration and check every file making changes as necessary. This should be ran if you upgrade a package too since it's more than likely one or more of the files will be overwritten and the inode will change.

2.5 Password Creation

LIDS uses a user defined password it stores in encrypted form(Ripe MD-160), in `/etc/lids/lids.pw`. To create a new password simply type:

```
lidsadm -P
```

It will prompt you twice for your new password and then change the password. This will obviously only work if LIDS is turned off. Once you have done this every time you need to reload the configuration and turn LIDS on or off you will have to enter your password in plaintext.

2.6 Viewing LIDS Status

You can use:

```
lidsadm -r
```

to view the current running status of LIDS. This can be useful for writing scripts that need to know if LIDS is turned on or not.

2.7 Viewing the Current LIDS Configuration

You can use the:

```
lidsadm -L
```

option to view a list of all the files and their attributes in the configuration. You must have LIDS disabled to run this command since it requires access to the `/etc/lids/lids.conf` file.

3 Protecting Your Files

The Linux Lockbox comes with a default configuration for protecting your files based on your configuration options and installed packages. If packages are removed, or added LIDS will have to be updated. Most of this can be easily accomplished using the GD WebTool application.

If you wish to do administration of LIDS from the console you will need to use the lidsadm program. Using the commands described in the previous section we will remove, add and update files on the Lockbox. Before any administration can be done you must first turn off LIDS. Turn LIDS off only on your session. Unless you are working in multiple sessions and feel safe leaving your system unprotected for the time.

```
lidsadm -S -- -LIDS
```

Now with LIDS disabled you can proceed with your work.

3.1 An Example: Protecting a Freshly Installed Package

For this example we added a package called my_package.rpm. my_package.rpm has a configuration file in `/etc`, a binary in `/sbin`, a log is kept
`/var/log/my_package.log` and stores user data in `/var/lib/my_pack`
`age/`. `my_package.rpm` also requires *setuid* and *setgid* access. Without reconfiguring LIDS this application won't function properly. Here is what needs to be done to add this package to your LIDS configuration. Issuing the following command will give you a list of the files an RPM uses. Though it won't tell you if it needs, read, write and/or append access to them.

```
rpm -qpl package_name.rpm
```

The first thing we want to do now is protect the configuration file. The configuration file never needs to be changed by the program so we can give it READ access only. If you want to make changes in the future simply disable LIDS, make your changes and enable LIDS. Here is how to protect our config file for READ only access:

```
lidsadm -A -o /etc/my_package.conf -j READ
```

Now the file is in the LIDS configuration file and set as read only. We used the *-A* option to ADD a new object. The *-o* object is the file my_package.conf and it's *-j* attribute is READ. Valid attributes are READ, WRITE, APPEND, DENY, and IGNORE.

NOTE: These are case sensitive and therefore must be written in all upper case letters.

We have successfully protected the configuration file. Next we will tackle the log file. The log file is simply a file that maintains a list of program events. The file never changes previous information and therefore can be set to APPEND only. So we issue a similar command as the one used for the configuration file:

```
lidsadm -A -o /var/log/my_package.log \
           -j APPEND
```

This command is almost the same as above except we set the log file to APPEND. Next we want to protect the user data. We want to be able to read and write to the user data, but we don't want root to have the ability to view the data, since it could be private information. This is also a secure method of protecting sensitive data from an intruder, if they gain root access. First we have to deny everybody access from the user data. There could be a slight problem if the user data directory contains dozens, maybe hundreds of files. This could be quite cumbersome typing in each file name into lidsadm. Well the lidsadm program allows you to protect a directory and everything under it. So now lets protect the directory:

```
lidsadm -A -o /var/lib/my_package/ -j DENY
```

Now everyone is denied access to that directory and everything in it. In fact, if you get a directory listing of `/var/lib the my_package/` directory will not even be visible. So now it's safe. Too safe now actually. You have to give your my_package binary access to the data for it to run properly. To give the binary, and only the binary, access to the data, we can issue this command:

```
lidsadm -A -s /sbin/my_package_binary \
        -o /var/lib/my_package -j IGNORE
```

Once that is issued it gives `/sbin/my_package_binary` full access to everything in the `/var/lib/my_package` directory. In the example above we *-A* added a new *-o* object but this time linked it to a *-s* subject. So now the user data is completely protected and is not hindering the usage of the my_package application.

Finally we need to protect the binary from being deleted. So we can simply set it as read only. We can use the same command that we used for the config file:

```
lidsadm -A -o /sbin/my_package_binary -j READ
```

When initially securing the system the entire `/sbin` directory was protected. To add `/sbin/my_package_binary` separately you can do what was done above or you can update all the items in the LIDS config. Doing this will add the `/sbin/my_package_binary` to the config

```
lidsadm -U
```

We are now left with one last problem. The my_package_binary needs *setuid* and *setgid* permissions to run properly. By default the setuid and setgid capabilities are disabled by LIDS (more concerning capabilities will be explained in the following sections). Using lidsadm you can assign capabilities to a specific file. The lidsadm command is similar to adding a file:

```
lidsadm -A -s /sbin/my_package_binary -t \
        -o CAP_SETUID -j INHERIT
lidsadm -A -s /sbin/my_package_binary -t \
        -o CAP_SETGID -j INHERIT
```

Now the `/sbin/my_package_binary` will inherit the setuid and setgid capabilities in the kernel giving it permission to use. The -t flag is used to tell lidsadm the object is special, or not a file in this case.

To make certain everything in your LIDS configuration is set properly issuing a:

```
lidsadm -L
```

will present you with a list of all the items in the configuration and their attributes. You must have lidsadm turned off to use this option. Now the entire package is done. Reload the config into LIDS and finally enable LIDS again:

```
lidsadm -S -- +RELOAD_CONF
lidsadm -S -- +LIDS
```

Now you are ready to go.

When LIDS is initially configured for your Lockbox a script was created that contains all file attributes. This script can be run at any time to reset you back to the system defaults. Additionally you can create your own script file for any additions you make. This makes it much easier if you make a mistake and have to start over from scratch. A simple command to launch your script will put you back where you were instead of typing everything back in. If you are using the GD WebTool this is already done for you. The script can be something basic, here is a sample script using the example above:

```
#!/bin/bash
#
### LIDS configuration - 9/13/00
#
#### Configuration for my_package.rpm
#
 lidsadm -A -o /etc/my_package.conf -j READ
 lidsadm -A -o /var/log/my_package.log -j APPEND
 lidsadm -A -o /var/lib/my_package/ -j DENY
 lidsadm -A -s /sbin/my_package_binary \
           -o /var/lib/my_package -j IGNORE
 lidsadm -A -o /sbin/my_package_binary -j READ
 lidsadm -A -s /sbin/my_package_binary -o CAP_SETUID \
           -j INHERIT
 lidsadm -A -s /sbin/my_package_binary -o CAP_SETGID \
           -j INHERIT
#
#### End my_package.rpm configuration
```

You can even add this to your */etc/rc3.d/* (*/etc/rc.d/rc3.d/ for RedHat systems*)so the LIDS configuration is freshened on every boot up. Just make sure it's done before the kernel is sealed (*lidsadm -I*). More information about sealing the kernel is explained in later sections.

If this package is ever removed you will have to delete the entries. Using the script method above, delete out all the entries then *lidsadm -Z* and run all the scripts again. Otherwise you can issue a *lidsadm -D* for each file entry you have. For files with multiple entries, you only need enter it in once. Lidsadm will delete all entries for that file.

4 Kernel Capabilities

When a process is created it is given a set of capabilities from the kernel. These capabilities tell the process what it can and can not do. LIDS gives you the ability to alter these capabilities in the kernel. You can set the capabilities to apply to all processes or only specific processes. We saw how to apply capabilities to only specific processes previously in the *Adding an Entry* section and in the above example.

The default capabilities set that LIDS used is defined in the `/etc/lids/lids.cap` file. This file contains a list of the capabilities by name, with a number and a + or - symbol before it. A + enables the listed capability following it and a - disables it. Before each capability is a description of what the capability does. We suggest you keep the default capabilities. You can also find a list of all the capabilities and definitions at the end of this section and by just typing `lidsadm` or `lidsadm -h`. Issuing:

```
lidsadm -I
```

sets all the capabilities listed in the `/etc/lids/lids.cap` file. By default, in the Lockbox, the command is entered into the `/etc/rc.local` file so the kernel is sealed during boot up. When LIDS is disabled the capabilities return to their original settings and when you enable the kernel again they return to their previous state.

Earlier we set capabilities to a binary. We were actually linking a capability a process the binary creates:

```
lidsadm -A -s /path/to/binary -t -o CAP_NAME
```

All processes, however are protected from being killed by anyone but the owner of the process. This too can be avoided with the above process.

5 Capability Names and Descriptions

Here is a list of all the capabilities supported by LIDS and what their function is.

CAP_CHOWN In a system with the _POSIX_CHOWN_RESTRICTED option defined, this overrides the restriction of changing file ownership and group ownership.

CAP_DAC_OVERRIDE Override all DAC access, including ACL execute access if _POSIX_ACL is defined. Excluding DAC access covered by CAP_LINUX_IMMUTABLE.

CAP_DAC_READ_SEARCH Overrides all DAC restrictions regarding read and search on files and directories, including ACL restrictions if _POSIX_ACL is defined. Excluding DAC access covered by CAP_LINUX_IMMUTABLE.

CAP_FOWNER Overrides all restrictions concerning allowed operations on files, where the file owner ID must be equal to the user ID, except where CAP_FSETID is applicable. It doesn't override MAC and DAC restrictions.

CAP_FSETID Overrides the following restrictions that the effective user ID shall match the file owner ID when setting the S_ISUID and S_ISGID bits on that file; that the effective group ID (or one of the supplementary group IDs) shall match the file owner ID when setting the S_ISGID bit on that file; that the S_ISUID and S_ISGID bits are cleared on successful return from chown(2) (not implemented).

CAP_KILL Overrides the restriction that the real or effective user ID of a process sending a signal must match the real or effective user ID of the process receiving the signal.

CAP_SETGID

- Allows setgid(2) manipulation
- Allows setgroups(2)
- Allows forged gids on socket credentials passing.

CAP_SETUID

- Allows set*uid(2) manipulation (including fsuid).
- Allows forged pids on socket credentials passing.

CATP_SETPCAP Transfer any capability in your permitted set to any pid, remove any capability in your permitted set from any pid.

CAP_LINUX_IMMUTABLE Allow modification of S_IMMUTABLE and S_APPEND file attributes.

CAP_NET_BIND_SERVICE Allows binding to TCP/UDP sockets below 1024.

CAP_NET_BROADCAST Allow read/write of device-specific registers

CAP_NET_ADMIN

- Allow broadcasting, listen to multicast.
- Allow interface configuration
- Allow administration of IP firewall, masquerading and accounting
- Allow setting debug option on sockets
- Allow modification of routing tables
- Allow setting arbitrary process / process group ownership on sockets
- Allow binding to any address for transparent proxying
- Allow setting TOS (type of service)
- Allow setting promiscuous mode
- Allow clearing driver statistics
- Allow multicasting

CAP_NET_RAW

- Allow use of RAW sockets
- Allow use of PACKET sockets

CAP_IPC_LOCK

- Allow locking of shared memory segments
- Allow mlock and mlockall (which doesn't really have anything to do with IPC).

CAP_IPC_OWNER Override IPC ownership checks.

CAP_SYS_MODULE Insert and remove kernel modules.

CAP_SYS_RAWIO

- Allow `ioperm/iopl` and `/dev/port` access
- Allow `/dev/mem` and `/dev/kmem` access
- Allow raw block devices (`/dev/[sh]d??`) access

CAP_SYS_CHROOT Allow use of `chroot()`

CAP_SYS_PTRACE Allow `ptrace()` of any process

CAP_SYS_PACCT Allow configuration of process accounting

CAP_SYS_ADMIN

- Allow configuration of the secure attention key
- Allow administration of the random device
- Allow device administration (`mknod`)
- Allow examination and configuration of disk quotas
- Allow configuring the kernel's syslog (printk behavior domain name)
- Allow setting the domain name
- Allow setting the host name
- Allow calling `bdflush()`
- Allow `mount()` and `umount()`, setting up new smb connection
- Allow some autofs root ioctls
- Allow nfsservctl Allow `VM86_REQUEST_IRQ`
- Allow to read/write pci config on alpha
- Allow irix_prctl on mips (`setstacksize`)
- Allow flushing all cache on m68k (`sys_cacheflush`)
- Allow removing semaphores
- Used instead of *CAP_CHOWN* to chown IPC message queues, semaphores and share memory
- Allow locking/unlocking of shared memory segment
- Allow turning swap on/off Allow forged pids on socket credentials passing
- Allow setting read-ahead and flushing buffers on block devices
- Allow setting geometry in floppy driver
- Allow turning DMA on/off in xd driver
- Allow administration of md devices (mostly the above, but some extra ioctls)
- Allow tuning the ide driver Allow access to the nvram device
- Allow administration of apm_bios, serial and bttv (TV) device

- Allow manufacturer commands in isdn CAPI support driver
- Allow reading non-standardized portions of pci configuration space
- Allow DDI debug ioctl on sbpcd driver
- Allow setting up serial ports
- Allow sending raw qic-117 commands
- Allow enabling/disabling tagged queuing on SCSI controllers and sending arbitrary SCSI commands
- Allow setting encryption key on loopback file system

CAP_SYS_BOOT Allow use of `reboot()`

CAP_SYS_NICE

- Allow raising priority and setting priority on other (different UID) processes
- Allow use of FIFO and round-robin (realtime) scheduling on own processes and setting the scheduling algorithm used by another process.

CAP_SYS_RESOURCE

- Override resource limits. Set resource limits.
- Override quota limits.
- Override reserved space on ext2 file system
- NOTE: ext2 honors fsuid when checking for resource overrides, so you can override using fsuid too
- Override size restrictions on IPC message queues
- Allow more than 64hz interrupts from the real-time clock
- Override max number of consoles on console allocation
- Override max number of keymaps

CAP_SYS_TIME

- Allow manipulation of system clock
- Allow irix_stime on mips
- Allow setting the real-time clock

CAP_SYS_TTY_CONFIG

- Allow configuration of tty devices
- Allow `vhangup()` of tty