

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

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.

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.
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 its 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.

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.

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.

Protecting Your Files

EnGarde Secure Professional 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 your EnGarde system 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.

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_package/. 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 EnGarde 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 -l`). 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.

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 -l
```

sets all the capabilities listed in the `/etc/lids/lids.cap` file. By default, in EnGarde Linux, 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.

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`

`TID` 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/ext3 file system
- * NOTE: ext2/ext3 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

QUICK START GUIDE

This appendix is intended to give an overview of the functions of the Guardian Digital WebTool. After reading this appendix, the reader should be able to perform the steps required to set up a domain to receive mail, configure DNS services, and serve Web pages. If your EnGarde system will not be used to perform all of the functions listed above, it is especially important that you read the User Guide and have a full understanding of each of the services you will be configuring.

Before following the example below, EnGarde should have already undergone initial configuration and be plugged in and operating on a network. Information regarding the initial configuration can be found in Section [*] Installing EnGarde Installing EnGarde.

To obtain a fast and most accurate setup, follow the steps in the described order. Once you have successfully completed each step, proceed in order to the next step. There are four primary steps required to configure EnGarde:

1. Configure the network interface
2. Configure the DNS Server
3. Configure the Mail Server
4. Configure the Web Server to prepare for normal and secure websites

After the initial configuration of your EnGarde Secure Professional system, the basic system and networking functions are operating correctly and is ready to configure a sample store. We will be configuring our example EnGarde system to use the following initial values entered when EnGarde was configured:

- * [Hostname:]myserver
- * [Domain Name:]mydomain.com
- * [IP Address:]192.168.1.70
- * [Netmask:]255.255.255.0
- * [Gateway:]192.168.1.1
- * [Primary DNS Address:]192.168.1.70
- * [Secondary DNS Address:]192.168.1.60

In this example, we will be creating the domain engardelinux.com that will be hosting our DNS, routing mail, and serving web pages.

Network Interfaces

Before any interfaces are created you will need to know the following:

- * Each SSL-based website requires its own IP address. If more SSL-based websites are to be served, then a new interface must be created on another IP address for each website.
- * There can be many normal websites on the same IP address, given a Name Virtual Host defined in the Web server. See the Section [*] Virtual Host Management Tool - Virtual Host Management in the User Guide for more information on Name Virtual Hosts.

Example:

In the WebTool, click on System Management, and then click on Network Configuration. There will already be an interface defined as:

```
\resizebox*{4.4in}{!}{\includegraphics{images/QG1.eps}}
```

We want to set up a separate IP address for `www.engardelinux.com`, since we will be creating a Secure Web Server on it. Click on Add a New Interface to do this. We are now prompted for our information, at which point we enter:

```
* [IP Address:]192.168.1.71
* [Netmask:]255.255.255.0
```

After clicking the Create button the Persistent Interfaces screen will look like:

```
\resizebox*{4.4in}{!}{\includegraphics{images/QG2.eps}}
```

We have now successfully configured our network interface.

DNS Server

The DNS Server is the mechanism that provides name to IP address, and IP address to name mappings. It also provides the information necessary for mail to be properly routed. DNS was created because IP addresses are often hard to remember. DNS is used to map that address to a name, which is much easier to remember.

When typing `http://www.guardiandigital.com` into a Web browser, for example, the DNS server translates the host name (`www.guardiandigital.com`) into the IP address associated with `www.guardiandigital.com`. The browser then sends the request to that IP address and responds with the information available at that address.

DNS contains a number of unique characteristics about each host. Each characteristic forms a 'record' in the database that stores the DNS information. DNS "zones" are regions of IP addresses or names for which a particular organization is responsible.

Address Records

This is a record that provides a host name to be assigned to an IP address. All host names are associated with an IP address.

Name Server Records

This is a record that defines what name servers are responsible for the zone. In most cases, this will be the same as the hostname of the machine. Do not alter these records unless you have an explicit reason to.

Name Alias Records

This is a record which provides an "alias" for a pre-existing host name. There may be multiple aliases for a single host name.

Mail Server Records

This is a record which provides the information necessary to correctly route mail to correctly deliver electronic mail. Multiple e-mail servers may be defined for the same domain, each with a differing priority. Servers defined with a lower number have a higher priority and mail will be delivered to these hosts first.

Example:

Because we are creating a new domain (`engardelinux.com`), we must create a new forward zone for it. Before EnGarde can be configured to provide DNS for this domain, it must have been listed among the list of authoritative name servers for this domain.

>From the System Management menu, select DNS Management. The next step will be to create a new master zone. Click on the Create a New Master Zone link.

Leave the Forward (Names to Addresses) button checked since that is the type of zone to be created. Keep the default value of Master server. The rest the input looks like:

```
* [Domain name:]engardelinux.com
* [Email Address:]administrator@engardelinux.com
```

Leave the Allow transfers from... set to Allow None, and the Allow queries from... set to Allow Any. For more

information on these fields please refer to the full manual.

Click on the Create button to see the new zone in the zone listing. To add the records for our example, click on the engardelinux.com link.

Address Records

* [Hostname:]www.engardelinux.com

* [Address:]192.168.1.71

* [Hostname:]mail.engardelinux.com

* [Address:]192.168.1.71

Name Alias Records

* [Alias:]sales.engardelinux.com

* [Real Name:]www.engardelinux.com

Mail Server Records

* [Mail Server:]mail.engardelinux.com

* [Priority:]10

At this point we have successfully created www.engardelinux.com and mail .engardelinux.com to go to 192.168.1.71.

We have now successfully configured the DNS records for our sample domain.

Mail Server

The mail server provides the mechanism to deliver e-mail to a recipient on the Internet. When an e-mail is sent, the mail server is instructed to deliver the message to the remote mail server responsible for the recipient's domain.