The Linux Intrusion Detection System (LIDS) ROOTING OUT ATTACKS

DAVID SPREEN

When an attacker gains Linux root privileges, simple tools won't fend them off. LIDS offers methods to deny the intruder sight of and access to important parts of the system.

On Linux systems, root privileges are necessary for many actions. Unfortunately, there are no further restrictions for root, so an intruder can simply read everything, write everything, delete everything. When, instead of the system administrator, an intruder gains your root privileges, he can also do everything: read private e-mails, swap programs, install back doors, wipe away his own tracks and obtain all and any information from the system.

With LIDS, files and directories can also be protected from root, so his omnipotence is at an end. You can define the access rights to files more precisely (ACLs, Access Control Lists). Also, via capabilities, you can allow individual programs to do things which really require root privileges. For example a process for opening a port under 1024 really requires root privileges. With the appropriate capability this can also be done without root.

LIDS is just its name

You may be wondering why the whole thing is called Intrusion Detection. There were certainly some long discussions about this name, but after all, the primary objective of LIDS is not to spot break-ins, but to protect the system after a break-in. Nevertheless there are a few mechanisms aimed at recognition. When a LIDS rule is violated, LIDS closes not only the shell, from which the rule was violated: LIDS can also inform the administrator, say via e-mail. Also, LIDS offers a portscan detector in the kernel, although this is far from being as refined as Port Sentry.

Issuing access rights with ACLs

Being able to protect files from root is certainly a useful thing. But for your Linux to continue to function as it ought to, you can give back individual programs writing and reading rights. For example LIDS hides the file */etc/shadow* from all users, but explicitly allows */bin/login* to read this file.

The first step makes sense, since passwords can be cracked with the aid of the shadow file. But without any access to this file, no user can log in now, as */bin/login* can no longer check the password entered.

In fact, you ought to have made an exception: Without write access to */etc/shadow* no user can change his password.

At first glance, this appears perfectly simple, you could indeed give the program */usr/bin/passwd* write access to the shadow file. In this case, though, root could again change all the passwords. But since in LIDS we are assuming there is an attacker, we ought to exclude this possibility.

LIDS also has a solution for this, the LIDS free session (LFS). An LFS applies to the current terminal and all programs started from it. In an LFS, the LIDS restrictions do not apply. Since LFS only starts after you enter the *lidsadm* password, only the sysadmin could change the passwords. If there are no users on the system apart from the admin, this is a more feasible way.

New versions of LIDS

While working on this article, more recent versions of LIDS have come into being. But their syntax is incompatible with the older ones and nor are they adequately documented, but on the other hand it is more consistent per se. This article is mainly concerned with the old versions: 0.9.13 for kernel 2.2.18 and 1.0.5 for kernel 2.4.1. But by reading the source code and with the aid of the LIDS mailing lists I have still been able to make some amendments to the text in time.

But if several people have access, you will have to choose between comfort for the user and reinforced security.

For which systems does LIDS actually make sense?

LIDS is designed to prevent the overwriting of important parts of a system. It therefore makes little sense to use LIDS on a system which is changed daily, unlike production systems with individual services, which are installed once, then run in a stable environment. Playing with a bugfix or an update from time to time is not a problem, but automatic upgrades like Apt-get from Debian will not get along with LIDS.

A Linux with a 2.2.x kernel is recommended. There is also a LIDS for the Linux kernel 2.4, but as a rule the use of a kernel under x.x.10 is not advisable. Also, LIDS is still in development, so you should keep an eye open for updates at regular intervals.

Installation of LIDS

A kernel patch and the administration programs are available as *.tar.gz* archives at *http://www.lids.org*. Once the packet for the corresponding kernel version has been downloaded and unpacked, change to the directory */usr/src/linux*. The patch is applied from here, with

```
patch -p1  .../lids-version.patch
```

after which the kernel is configured.

To obtain the LIDS options as a selection the following items must be selected:

```
[*] Prompt for development and/or
   incomplete code/drivers
[*] Sysctl Support
```

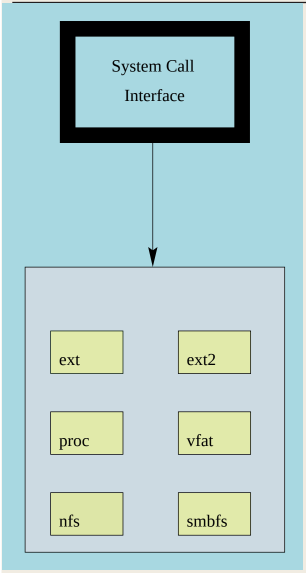
Now you can configure and compile your LIDS kernel.

Next the *lidsadm* tool needs to be compiled. You will find the sub-directory *lidsadm-version* in the LIDS package. Before you type *make* in there, however, in version 0.9.12 a minor change needs to be made to the Makefile: Complete the line *CFLAGS=...* with the entry *-DLIDS_CONFIG*. Now call up *make && make install*, to compile and install the program.

Before first booting with the new kernel you must set a LIDS password and synchronize the configuration with your computer. The LIDS password is set using *lidsadm -P*. It will be needed later to deactivate LIDS or to change to an LFS. If you should start with the LIDS kernel, without first having set a password, the attempt will end with a kernel panic.

We had already addressed the ACLs. These are stored in the file */etc/ lids/lids.conf*; but in addition to the file and directory names, there are also Inode numbers entered there. The Inode numbers next to the file names are comparable to an address for the files.

Since file system accesses can occur, not via the file names, but also via Inode numbers, LIDS obviously has to know the numbers of the files to be protected. For the ACLs which were preinstalled with the LIDS package to fit into your system,



LIDS and VFS

LIDS sits on the VFS layer. Linux is designed so as to be able to work with many different file systems. The VFS layer is a general interface for all file systems supported in the kernel. If there is now access to the file system, the system-call interface does not access the individual file system directly, but forwards this query — or put in simplified terms — to the VFS. In reality, the device driver subsystem also plays a role in this. Figure 1 shows the principle; details on this can be found at http://plg.uwaterloo.ca/ ~itbowman/CS746G/a2

Figure 1: The system-call interface (the file system interface available for user processes) does not access the individual file system, but its common interface, the VFS.

update the Inode numbers with *lidsadm -U*.

The ACLs, which relate to files and directories, are active as soon as the kernel loads the LIDS system. On the other hand the ACLs which deal with the capabilities only becomes active the first time when *lidsadm -I* is executed. So you must ensure, too, that this occurs immediately on booting.

Decide precisely when you want the capability rules to be made active: For example it makes sense only to make these take hold after activating the firewall, if the firewall makes use of capabilities which you want to block.

You must complete the configuration before you boot the system with the new kernel, so that the machine is actually still usable when it next starts. If your computer ever becomes unusable as the result of LIDS rules, when booting, assign LILO the parameter *security=0*.

Configuration

New rules are added with *lidsadm*. For a file rule the *lidsadm* command looks like this:

```
lidsadm -s /path/program -o⮐
 /path/file,directory -j RULE
```

The syntax is a bit like that of Ipchains. The option *-s* assigns the subject of the rule, the option *-o* the object. If the subject (here, a program) wants to access the object (file, directory or capability), the ACL (or to be more precise: by means of the rule assigned with *-j*) governs whether and how the subject can gain access.

The following call up allows */bin/login* read access to */etc/shadow*:

```
/sbin/lidsadm -s /bin/login -o /etc/shadow⮐
 -j READ
```

To make */etc/shadow* unreadable for all, the following call up would be necessary:

```
/sbin/lidsadm -o /etc/shadow -j DENY
```

Further rules can be found on the Lidsadm manpage. Why LIDS works for any number of file systems, not just for Ext2, is explained in the box “LIDS and VFS”.

Capabilities

Since kernel 2.2.x the Linux kernel has had a sort of access control of its own — capabilities. For certain actions, tasks need more privileges than a normal user has. Instead of the all-or-nothing division between root and normal user, there are more finely subdivided authorisations. For example

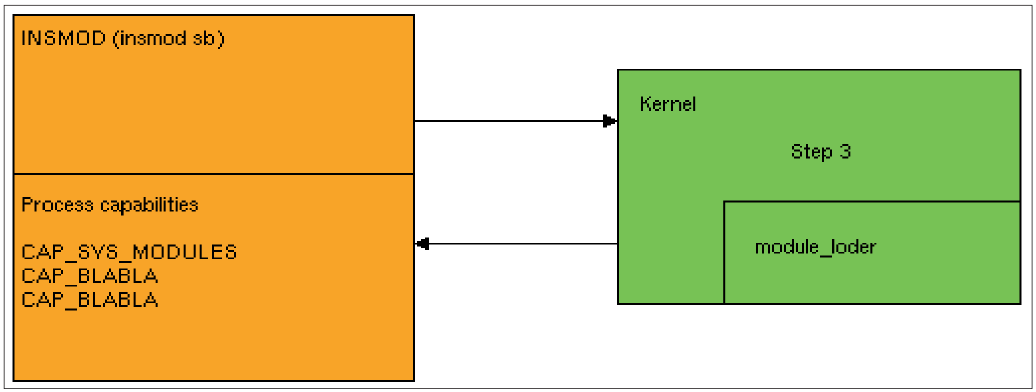


Figure 2: In step 1 Insmod wants to access the Module_loader. Firstly the kernel checks, in step 2, whether Insmod has the capability CAP_SYS_MODULES, and only then, in step 3, allows the sub-module to be loaded.

the loading and removal of modules requires the capability *CAP_SYS_ MODULES*, *insmod* thus needs precisely this capability (see Figure 2).

In the Linux standard kernel, it is not possible to assign a program file any capabilities, as there is no support in the file system. Capabilities can only be given to or taken from a current process. The kernel does, however, check the capabilities; this is resolved by root processes automatically receiving all capabilities. To take a capability away from a root program, it has to be removed globally from the system.

LIDS, on the other hand, offers the option of giving to and taking from each program capabilities individually, regardless of whether it has root privileges or not. */etc/lids/lids.cap* defines which capabilities are available by default and which are only issued by special regulations in the ACLs. So for example the Apache web server can be assigned the capability *CAP_NET_BIND_ SERVICE*, so that it can occupy a port smaller than 1024, although this capability was globally deactivated. This example requires the following command:

```
/sbin/lidsadm -s /usr/sbin/httpd -t -o
 CAP_NET_BIND_SERVICE -j INHERIT
```

The option *-t* says that the object of this rule is a capability. The rule *INHERIT* determines that the capability is also inherited by child processes of */usr/sbin/httpd*. The opposite would be*NO_INHERIT*. An overview of the capabilities relevant to LIDS can be found in the Lidsadm-manpage.

From version 0.9.14-2.2.18 or 1.0.6-2.4.2 respectively, the syntax of *lidsadm* has changed slightly. Here the call would look like this:

```
/sbin/lidsadm -s /usr/sbin/httpd -i -1 -o
 CAP_NET_BIND_SERVICE -j GRANT
```

The options *-s* and *-o* still assign subject and object to the ACL. The option *-t* is dropped and the rules *INHERIT* and *NO_INHERIT* also cease to apply. Instead of these, in capabilities the rules *GRANT* is assigned so as to allocate it to the subject. The option *-i -1* means that the child processes of the httpd also receive the capability.

What's new is the option of defining the depth of the inheritance. While *-i -1* means infinitely deep inheritance, with *-i 1* the capability would still be inherited by the child processes of the httpd, but would no longer by its children (thus the grandchildren). To prevent inheritance completely, specify *-i 0* or leave out the option *-i* completely.

Hidden Processes

With the methods proposed so far, you can certainly make life difficult for an intruder. In the real world, though, one would leave him tapping in the dark for as long as possible. A firewall can effect this very well outwardly, for example by concealing the inner network structure.

Info

Brian Ward & Peter Sütterlin: Linux kernel HOWTO: http://www.tu-harburg.de/dlhp/HOWTO/DE-kernel-HOWTO.html
Ivan Bowman, Saheem Siddiqi & Meyer C. Tanuan: Concrete Architecture of the Linux kernel: http://plg.uwaterloo.ca/~itbowman/CS746G/a2
Steve Bremer: LIDS FAQ: http://www.clublinux.org/lids/
Port Sentry: http://www.psionic.com/abacus/portsentry/

■

LIDS causes the same effects inside a computer: Processes are hidden, if the program is given the capability *CAP_HIDDEN*:

```
/sbin/lidsadm -s /usr/sbin/popper -t -o
 CAP_HIDDEN -j INHERIT
```

The result of this example, as you have surely already discovered, is that */usr /sbin/popper* can no longer be seen in process lists such as *ps* and *top*.

Here, too, the call in the newer versions of *lidsadm* looks somewhat different:

```
/sbin/lidsadm -s /usr/sbin/popper -i -1 -o
 CAP_HIDDEN -j GRANT
```

Switching LIDS

Hiding processes, granting or refusing capabilities, restricting file accesses even for root — all with the aim of making life harder for an intruder. Unfortunately, however, this also makes one's own work harder: Sometimes one has to change something, as legitimate admin, simply and quickly, a route or a gateway, or changes need making to the firewall. For these cases there is the LFS (LIDS Free Session), in which you can again work as normal. With the aid of the LIDS password a terminal can be released from the LIDS controls. The command for this reads:

```
/sbin/lidsadm -S –- -LIDS
```

But if a service is to be started from new or even a restart is imminent, then even the LFS is no longer adequate. At this point the actions are no longer under the control of the one released shell, so we must first completely deactivate LIDS. The following command does this:

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

To load in an altered LIDS configuration from new while LIDS is active, call up the following:

```
/sbin/lidsadm -S –- -RELOAD_CONF
```

LIDS should serve the majority of its time without needing much changing. We hope this article has given you a basic look at the methods and operation of LIDS. Nevertheless, you should certainly look for advice on configuration in both the Lidsadm-manpage and the LIDS-FAQ, so that your Linux does not suddenly regard you as the enemy and deny you entry. ■

The author

David Spreen is the Debian maintainer of the LIDS packages. Apart from studying, he also works for NetUSE AG, an ISP in Kiel. And spends most of the rest of his time on his Linuxbox or programming. He would like to thank, among others, Benjamin Traube and Eugene A. Brin..