

Linux Intrusion Detection System

By Paul Wouters <paul@xtdnet.nl>
Xtended Internet, The Netherlands

What is LIDS?

- Limit the rights of "root" on its own server
- Using Linux 2.2 capabilities to restrict root
- Protect files, processes and network
- Protect low-level access (IO, memory, network)
- Protect processes against signals
- Hide stuff
- Warn (mail, syslog, inbuilt mailer)

The result of offering these services is that we have a diverse network with various operating systems and trust relationships to external systems. Naturally, we have an internal network with Windows desktops as well, secured behind a Linux firewall using Network Address Translation (masquerading).

We host some high profile websites, such as the computer magazine c't, at <http://www.ct.nl/> as well as some small but controversial sites. For example an anti-Scientology site (<http://xenu.xtdnet.nl/>) and a site with court documents about Belgian police affairs (<http://www.klotenknijper.com/>). Some of these sites might be responsible for the regular probes on our network (roughly 1-15 probe sessions per day).

Problems of LIDS

- Difficult to install, requires advanced knowledge of Linux systems, compiling kernels, general knowledge of system startup
- Even more difficult to configure correctly (use a scrap server, trust me :)
- Some system limitations depending on the restrictions you put in place.
- If you've truly messed up, boot through LILO with "security=0"

There are various people called "Paul Wouters". I'm most notably not Paul Wouters of the University of Amsterdam (pwouters@xs4all.nl)

I am not a full-time security professional. I am a security-minded network administrator. Hopefully I am a good example of the average administrator who needs to keep his systems running, but who also needs to put security concerns against other company interests, such as pleasing the customers, leaving certain tasks undone when there is no time, or even take a break from it all in order to be able to cope with next week's mess.

(Or be half a world away to give a lecture on Linux security)

Lids: Installation

- Assumes clean fresh uncompromised system
- Install latest 2.2 kernel source
- Patch the kernel with lids patch, run "make config"
- Compile and install lidsadm
- Generate MD160 password
- Rerun "make config" and fill in MD160 passwd
- Compile the kernel
- Setup the rc.lids file
- Install new kernel (don't forget to run lilo :)
- Reboot

No one can deny that disconnecting the system is the most safe thing to do. Also, to gather as much information as possible regarding the hackers, widely informing organisations and people around you would be wise. Thereafter, one should quietly spend a few days investigating the compromised machine and refining the already optimal security.

LIDS: Some kernel options

- Hang up console when raising security alert
- Don't execute unprotected programs before sealing
- Enable init children lock feature (carefull!)
- Allow switching LIDS security on/off with MD160 password
- Allow remote users to switch LIDs
- Enable portscanner in kernel (no promisc. Mode)
- Sent security alerts over network (mail, syslog, internal mailer)
- Allow hardcode list of files specific access (eg Xserver and /dev/kmem)

Once the system has been restored and the fix to the vulnerability has been applied we are almost ready to go back online.

Of course, our secret credentials could have been stolen, so to prevent abuse, we need to renew all of those and revoke the old ones. Especially SSL certificates and SSH keys are vulnerable, because they tend to be used without a protective passphrase, since that would restrict their usefulness for automated tasks.

Finally, we can put the system back online.

Rc.lids

- Note: Documentation is outdated :)

```
#clear protection list
/sbin/lidsadm -Z
#make essential files immutable/readonly
/sbin/lidsadm -A -o /boot -j READ
/sbin/lidsadm -A -o /bin -j READ
#Alternative for /usr is to mount from a readonly exporting NFS
server.
```

Unfortunately, the previous scenario is not something that most companies can afford. Networks and servers cannot normally be pulled off the network for careful and slow investigations. They need to be up and running, generating revenue. Downtimes are considered bad for image. So is admitting that you have been hacked. So often a system administrator will be forced to cover up and keep things running, even if this means operating a compromised host on the network. Not to mention the fact that these events tend to happen when you can least afford it. There is never enough time for a thorough investigation.

Hardware is generally cheaper than downtime. Having reserve machines to restore the systems on and then take the time to carefully analyse the hacked machine(s) can be a very good solution to this dilemma.

Rc.lids

```
# some files need to be written on protected directories
/sbin/lidsadm -A -o /etc -j READ
/sbin/lidsadm -A -o /etc/mtab -j IGNORE
#some files can be safely append only
/sbin/lidsadm -A -o /var/log -j APPEND
#limit some more file access
/sbin/lidsadm -A -s /usr/sbin/httpd -o /etc/httpd -j READ
#example limit writes
/sbin/lidsadm -A -s /usr/bin/passwd -o /etc/shadow -j WRITE
```

The main concern is that no single file on the host can be trusted. Binaries can log passwords, copy e-mail offsite, log keystrokes, harvest information, or attempt automated attacks on other hosts. Another hacker might enter the system using the same vulnerability and cause more, perhaps fatal, damage. Or a hacker might simply panic once he realises that he has been detected and yet “allowed” to be on the system. Such hackers might resort to destroying the entire compromised host.

There is also a risk to other systems. The compromised host might contain a network sniffer to harvest information about servers in the same network segment. A very careful check of the trust relationships with the compromised host should be done if the host is left running.

The longer a system is left running in this state, the more information will be gathered. Every time a user logs on, a password might be obtained. Thus, one of the important (yet practically impossible) tasks is to keep the system from leaking the gathered information to the outside world.

Keep in mind that the machine might self-destroy any second. Don't keep the host running without having its backup tape restored to another host.

You don't want to find out, after the hacker destroyed the compromised system, that your only backup tape has become unreadable over time.

Rc.lids

```
# seal the current system
# any firewall scripts should now have been run, eg rc.firewall or
# rc.local or /etc/rc.d/init.d/bastille
/sbin/lidsadm -I -- -CAP_KILL -CAP_LINUX_IMMUTABLE
-CAP_NET_BIND_SERVICE -CAP_NET_ADMIN
-CAP_NET_RAW -CAP_SYS_ADMIN +CAP_SYS_BOOT
+CAP_SYS_MODULE -CAP_SYS_RESOURCE
-CAP_SYS_TIME -CAP_SYS_TTY_CONFIG
-INIT_CHILDREN_LOCK
```

The dilemma you are faced with here, is the choice between either getting the network going as soon as possible or collecting evidence. If the hackers want to make sure that they don't leave a trace, they can do a lot to hide what has happened. Imagine for instance a binary running that has been deleted. As soon as you kill the process, the binary's inode on disk is cleared and finding it will be next to impossible, especially on a fully running production server. You need to make an educated guess in harvesting information and then getting the systems back as soon as possible.

CAP_CHOWN

Restrict changing owner and group permissions on files

Fairly safe to restrict

- Might cause problems for console logins, Xwindows, floppy, cdrom access etc.

One easy trick is comparing “ps” output with “pstree” output. The “ps” binary is almost always trojanned for a version that hides the attackers processes, yet “pstree” is almost always left alone. Recompiling a new “ps” won’t hurt either, assuming that no one has put Ken Thompson’s idea about the C-compiler compromise in practise yet. With “pstree -p” you can easily find the rogue processes that have been given other names. The “lsof” tool (<ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/>) is also a great tool to discover what file handles or sockets a process is using.

The output of “netstat -an” is valuable to see what ports are being used as backdoors. These ports can then be added to the firewall to give you a warning when the hackers try to get back onto your system.

Once the gathering of runtime information is complete, kill the processes with the SIGKILL (-9) signal, so they get no chance to do anything before being terminated.

CAP_SETGID/CAP_SETUID

Restrict setgid/setgroup/setuid

Dangerous to use

- Might cause problems for servers that run as root and setuid to users/groups such as ftp, mail, apache (suexec)

At first we checked files that are often a target. In principle there is not much to lose, as the rogue binaries are already running on the system. Files we found changed were /etc/rc.d/rc.local, /etc/profile, /etc/bashrc, as well as in /root/.??*.

Binaries that were started through these scripts were replaced with known clean versions.

At this point no strange binaries were left running, but we could of course trigger something at any time without realising it. Also, the machine is still susceptible to the original vulnerability. But we are now ready for the slow and thorough cleaning.

CAP_SETPCAP

Restrict transferring current set of capabilities to other processes

Probably safe to restrict

- Assuming lidsadm doesn't use this feature itself (untested :)

Restoring the machine to a spare disk from DAT tape and then running a comparison on the backup server with the compromised host's read-only mounted NFS filesystem(s) is the best way of testing the integrity of the files. One should not use the compromised system itself for the comparison, because "cmp" or "diff" might have been replaced.

This process takes hours, even on a 100 Mbit network. We learned the hard way that MD5 checksums would have been a much better way to verify the integrity of the files on the compromised host.

Once all the hacker files have been isolated and saved, the original files can be restored onto the running system. We now have a clean – but still vulnerable – host back.

CAP_LINUX_IMMUTABLE

- Disallow changing immutable flag on ext2fs
- You MUST restrict this
- If not restricted, LIDS cannot protect the filesystem

If the compromised host is a server without regular users, the search for a vulnerability becomes a lot easier. Make sure that you are running the latest versions of all daemons, both of those running stand-alone (eg: sendmail, apache) but also those that are started from inetd (eg: ftpd, popd). Be careful to not disregard daemons that are blocked in the firewall (eg syslogd and nfsd, which should be blocked) because the exploit could just as well have come from the inside of the network. If you do have local (untrusted) users, then you will also have to hunt for other vulnerabilities. A good start is to go check all setuid and setgid files on the system. With local users, daemons blocked in the central firewall are still open to local exploits.

If you are still running inetd, consider switching to xinetd instead. It gives you better access control and a superior logging facility. Xinetd can be found at <http://www.synack.net/xinetd/>

Every host should, apart from logging to its own disk, send their logfiles to a remote loghost as well. This way, in case of a hack, you will still have the untampered logfiles. Most hackers don't even check to see if remote logs have been generated, so checking to see what entries in the local logs are deleted can actually give some rather useful information which you normally wouldn't find suspicious at all.

D_SERVICE

ports < 1024

strict

ve started before

What we found were three attempts at buffer overruns in the remote logfiles, which were not present in the local logfiles. We could only assume that a fourth one succeeded.

This left two programs as the suspects. Either the qpopper POP server had been overflowed (and failed attempts had been logged through syslog), or qpopper was merely used to generate a message to overflow the syslog daemon (which can not be contacted directly because we block syslog in our firewall). We were already running the latest version of the syslogd/klogd package, but there were newer versions of the Qualcomm qpopper daemon. Unfortunately, Murphy hit us. All the newer versions (in the stable as well as the development tree) failed to work correctly with mail files on NFS mounted file systems. After many complaints from users, we decided to switch back to the vulnerable version in order to have a functional POP server. In the following week, having had no feedback from Qualcomm regarding either the original (reproducible) overflow nor the NFS failures in their newer versions, we finally switched over to ids-pop (now called gnu-pop3d, available at <http://www.nodomainname.net/software/gnu-pop3d.shtml>).

To collect possible new information about the hackers, we changed the greeting banner of gnu-pop3d to resemble the one from qpopper, but so far, the hackers haven't returned to our POP server.

CAP_NET_BROADCAST

Disallow broadcasting and listening to multicast

Probably safe, but depends on your setup

- eg. Squid uses multicast, Samba if using remote announce feature
- Prevents being abused in smurf attack

We found out that the tools that were used listened to UDP and TCP ports 20746, 9090 and 6462, as well as to ICMP messages by sniffing the ethernet. We added filters for these ports in our firewall in case the hackers decided to try to come back.

We log various types of ICMP packets, which though, without logging the entire packets, is of limited use. Don't be tempted to block all ICMP traffic, it has its valid use. A good candidate for blocking (and logging) though is ICMP Echo Requests sent to your network broadcast address.

CAP_NET_ADMIN

Restricts a lot of abusable features

You **MUST** restrict this

- Stops manipulating network interface, routing table, firewall settings, promisc. Mode
- Can't run sniffers anymore
- Prevents certain audit tools (tcpdump)

When contacting the system administrators of the intended target of the programs installed on our compromised host, the destination IP address in the spoofed packets that the hacker tried to send out from our systems, they realised that they had suffered a distributed attack on one of their customers' servers. This customer had received threats from one of his (disgruntled) customers. The attack had only caused minor damage. The ISP's name servers briefly died, but restarted automatically and their internet connection had been big enough to cope with the extra bandwidth that the attack consumed.

It is very unfortunate that in these days a lot of network administrators don't configure their firewalls to only allow valid outgoing IP packets. As can be seen in our case, our desire not to "litter" the Internet gave us an extremely fast response time to having been compromised.

Small networks should really enable outgoing filters next to their incoming filters.

Backbones unfortunately cannot enable filtering, simply because their backbone routers cannot handle the extra load.

CAP_NET_RAW

Restrict raw packets and PACKET socket

Safe to restrict

- Might hamper traceroute/ping
- Can't sent custom made packets ("xmas packets")

CAP_SYS_MODULE

Restrict loading/unloading kernel modules

Essential to restrict

- Any module can redefine a kernel syscall, even the `lids()` calls itself. Must restrict!
- Load all modules before sealing kernel. Don't run `kmod/kerneld`
- Is possible (kernel not too big) don't use modules

This customer called us in a panic. The systems had already been down for two days and he couldn't find the cause. None of his users could log in, not even root. They didn't have enough knowledge to figure out what was wrong.

Entering the system in single user mode and testing the login binary seemed to give normal results. It opened `/etc/shadow`, checked the password and then identified it as wrong and refused access. The `ppp` daemon and the `pop` daemon behaved slightly different. They checked `/etc/passwd` and then denied access.

Conclusion: The system was confused on whether it was a regular password system or a shadow-based password system. Someone had played around with root access.

CAP_SYS_RAWIO

Restricts access to /dev/mem /dev/kmem

Absolutely needed. File integrity is useless without memory integrity

- Option; hardcoded list of binaries can be compiled in kernel which are allowed to bypass this setting (Xserver)

CAP_SYS_CHROOT

Restricts use of chroot()

Tricky, depends on server setup

- Don't use it on anonymous ftp servers, or environment using "jailed" programs (sendmail or named often run in chroot() jail).

Since there was no backup and no MD5 checksums, we could neither restore the system nor validate the existing files. We decided to properly install the rootkit, since irreversible damage had already been done. We changed the "configure" script to not use shadow passwords, changed the specific rootkit's backdoor passwords and reinstalled the hacker software. From that point on, most of the system was back up running "correctly" again. Some programs still failed, (the rootkit was a lot newer than the system it was installed on, a Linux/a.out 2.0.13 Slackware 2 based system, so there were various complications).

A quick script to convert shadow passwords to regular passwords was written and used, and the "passwd" command was disabled to prevent users from changing their password and thus locking themselves out again. We left with the explicit instructions that the entire network should be completely upgraded and the compromised host should be replaced as soon as possible. This advise was ignored and the system was again compromised. A downtime of over a week resulted before a new system was finally installed. The ISP has since changed their focus to small business web hosting.

CAP_SYS_PTRACE

Restrict tracing binaries

Safe to use

- A production server shouldn't need this to begin with.

We thought it was very strange to receive probes from the name servers of such a big ISP. Surely they had dedicated name servers which didn't need to give shell access to regular users (e.g. an old PC with extra serial ports running Linux). We assumed they had been hacked and sent them a warning.

We were quite surprised at their response, which clearly indicated that they were indeed probing our systems for some reason. We threatened to block the entire ISP (meaning blocking a few large cities, something we weren't even sure if we could justify to our own customers).

CAP_SYS_ADMIN

Restricts half the kernel

A MUST !!

- Restricts syslogd, printk, (u)mount, nfs*, swap, dma,pci raid nvram settings.
- CaveAt: How do you shutdown the machine nicely?
You don't

It is obvious these people made several mistakes. Instead of opening a conversation with us, they decided to actually play detective themselves and even ended up threatening to go to the police. Apart from the fact that this was most likely not sanctioned by their superiors, they were in fact breaking the law (the Dutch Law on computer criminality, "Wet computercriminaliteit") by trying to hack our systems.

Being accused, I was no longer willing to start a dialogue nor had I any reason to further assist them.

CAP_SYS_BOOT

Restricts rebooting

Solves previous problem :)

- Obviously has its pro's and con's. Clean shutdown becomes impossible

Again these administrators made various mistakes.

The small ISP that was abused in the attack was a customer of us. I assisted their system administrator and was therefor frequently logged on. And there were reasons for that, the system needed various security upgrades, and had been abused a few times.

I do indeed have an account at XS4ALL (Formerly known as HackTic, the first Dutch commercial ISP), but they couldn't know this. The person they found at XS4ALL (pwouters@xs4all.nl) was indeed "Paul Wouters" but Paul Wouters of the University of Amsterdam.

If you do a search on "Paul Wouters" on AltaVista, you find several people. Indeed, you will find me in many Linux and Unix security related material. Does that make me a hacker (Or rather a cracker)?

These people were lucky I didn't file any complaints or charges against them, but instead decided to defuse the situation.

CAP_SYS_TIME

Restricts changing the clock

Again tricky

- Will prevent weird times on syslog packets
- You can't use NTP to synchronize

CAP_SYS_TTY_CONFIG

Restricts tty device reconfiguration

Needed for integrity

It can be valuable to determine the type of intruder you have on your system. Perhaps he can and is willing to assist you, or give you information about the vulnerabilities on your system.

First, the professional industrial espionage person is not someone you will be able to talk to. They are most likely more clever than you, and it is highly advisable to get help. Contact law enforcement agencies and/or Internet authorities for help. You are going to need it.

Second, the typical hacker is usually just a “geek” experimenting a bit with new knowledge gained. Most likely he hasn’t intended to harm. It could be valuable to talk to this person. He could even might become your new system administrator! It can make sense to give him room to play. For instance, the Dutch ISP XS4ALL has a policy that hacking the system at root level and then telling them how you did it would get you an apple pie and a free subscription for a year.

The last type you can encounter are the Script Kids. These people are hard to talk to and you tend to gain practically nothing from talking to them that you cannot find out yourself on Bugtraq or Rootshell. Talking to them only gives them an additional kick. They will be unreasonable (“It’s not my responsibility, you should secure your system better”) and they won’t care about your worries.

INIT_CHILDREN_LOCK

Prevents signals being sent to processes that have
/sbin/init as parent process.

Asking for trouble, but recommended :)

- Prevents daemons re-reading config files, being HUPed KILL'ed (see CAP_KILL as well)
- If you thought rebooting nicely was difficult before...

The mailing list is a semi-private list on which many technical and political people are subscribed. System administrators from big companies, “freedom fighters”, people of various EFF groups, former HackTic people.

What I posted was a false message claiming that I had a dilemma between finding this issue too minor for any serious complaint on one hand, and on the other hand being coerced into co-operating with the police for such minor, but officially criminal, offence. I expressed my wishes for these script kids not to end up with a criminal record over such a minor offence. The worst part being that I had of course identified the intruders (which was true). The list received a lot of responses. Some quotes:

Now that they have been tracked down, and the police is aware of that, you have no choice but to report what you know. Otherwise you end up having a legal problem, and an economical problem; your business reputation is at stake.

I don't believe it would be the right thing if you would cover them, because it ends up damaging yourself. You have responsibilities as a witness that cannot be escaped.

You shouldn't have to accept the consequences of their foolishness. If there's no other way to avoid it, you're going to have to testify.

If a [computer related] crime has been committed, you are forced to co-operate with the investigation..

I managed to perfectly get my point across to the script kids, without actually threatening them.