Securing Linux

Through various ins and outs I came to worry about the vulnerability of Linux servers. My worst case scenario means a server under my control is hacked and a rootkit installed. The hacker can then use the server to send spam, attack other computes and so on. So I've been checking out SELinux, LIDS, OSSEC, Snort, TripWire and AppArmor. It dawned on me that I couldn't be sure they worked unless I attacked the system. More specifically, I had to find a way of hacking a regular Linux installation and then verify that a given addition thwarted that attack.

Damn Vulnerable Linux seemed an ideal choice but turned out to be more about finding exploits than using them. I found no neat guide showing me how to start and then hack the web server that comes with the distribution. I tried Metasploit from Backtrack 4 to no avail. After some searching I found a known exploit in a OpenDcHub on exploit-db.com. It took me something like two hours before I could carry out a successful attack on the running hub: I could create two empty directories!

It became apparent that I couldn't find any useful exploits that could be used to test the efficacy of a security solution. Surely someone must have written a dummy application that is designed to execute arbitrary code, for exactly this purpose? No, or at least I couldn't find it. So I wrote my own. I mainly copied code from http://webpython.codepoint.net/cgi_file_upload and added things like shellcode execution by using code from http://zeltser.com/reverse-malware/convert-shellcode.html. Changing UID, removing path sanitation(thus allowing the "hacker" to create files anywhere on the server) and allowing the execution of arbitrary shell commands were added as well.

While this process was instructive, the outcome was meager, bordering on pathetic. Nothing entered as shellcode or plaintext commands got me close to root access or even created an annoyance to the system as a whole. True, I could have erased lots of stuff pertaining to the web server, but that is hardly undetectable and so can be fixed. Setuid didn't work(there's a surprise...) and I got nowhere.

Perhaps it would have been better to have used Fedora Core 3 instead of CentOS 5.4 but I really put my shoulder into this. No firewall and no SELinux! As the root user I cobbled together a Python CGI script with the explicit intention of making it easy for someone(me) to gain access to the system via the web server, and it didn't work! In my defense, I've circumvented protection mechanisms on games and demos in just a few hours - for my amusement only - so it's not that I can't hack. Linux is just too robust to allow people other than professional hackers to reliably gain root access.

After some more searching I found an exploit that would allow privilege escalation. It compiled nicely but didn't work. This was not entirely surprising since the exploit was fixed in the Linux kernel in 2009 but it turns out it also relies on the system running PulseAudio. What server runs that? I ran this stuff on a desktop installation of CentOS and *it* didn't have PulseAudio installed. As my patience ran dry I went back to the previous script. It would allow a user to enter any command in a form field and send it for execution. Using this I could create files in /tmp and even change the permissions. Of course, they all had the user permissions of apache, which were not extensive.

It turns out that Python has a call os.setuid() - giving the script a different user privilege - which was added to the cgi-script. More specifically it was "os.setuid(0)" to give it root privileges. This of course broke the script because the Python interpreter is not allowed to perform the setuid call, even though it is a file owned by root. By manually setting the Python binary's setuid-bit to 1 I was able to enter the command "chmod 777 /etc/passwd" into the webpage and have the password file made readable to everyone on the computer. Because of the necessity to purposefully circumvent these safeguards I at one point figured I might as well say that we were working under the assumption that a hacker has performed privilege escalation and obtained a root shell. In the end however there is a difference between the two. The ludicrous security hole I created did after all obey certain constrants due to its origin. Placing the web server in a sandbox for instance means that root privileges obtained this way aren't much good.

Installing PulseAudio on CentOS was pretty easy but didn't work. The most stable release of the Linux Intrusion Detection System is for kernel 2.6.21 and it turns out Fedora 7 has that kernel as its default. After installing Fedora 7 in a virtual machine I ran the privilege escalation exploit and it succeeded! It didn't matter if SELinux was enabled or not, the hack - Mr. Magorium's Wunderbar Emporium - succeeded to give a non-privileged user root. The next two or three days went into trying to compile a kernel in Fedora. Actually compiling a kernel was easy enough - I have installed Gentoo Linux more than once - but getting it to run was less straight forward. For some reason the hard drive in the virtual machine were not readable no matter what drivers I compiled into the kernel and I didn't know how to make an init-ramdisk. Compiling the kernel Fedora-style made it easy to make such an initrd image but the process isn't entirely straight forward.

Long story made slightly shorter, two kernels with initrd images were compiled and tested successfully. One was a standard 2.6.21.2 kernel with Fedora patches and the other was a kernel with the same patches with the addition of LIDS 2.2.3rc1. To make LIDS work I had to turn off SECURITY_SELINUX and SECURITY_CAPABILITIES in the configuration file for the kernel before compiling. Running LIDS in itself proved less impressive than I had hoped. Once I could log on to the LIDS-hardened system I could run the hack as a normal user and gain root. I guess I had misunderstood the nature of LIDS' locking down so it was probably more fair to go back and use LIDS to place Apache in a sandbox and limit the damages made possible by a successful privilege escalation.

Compiling kernels Fedora style

Making the story longer however involves explaining how to compile a kernel on Fedora 7.

Start by installing the necessary tools, as root of course. I included all the development tools I could during the install of the operating system so maybe a different install will need more of these.

```
# cd /root
# yum install rpmdevtools.noarch
# yum install yum-utils
# cp /mnt/Public/Hardening/Kernel_stuff/kernel-2.6.21-1.3194.fc7.src.rpm /root
# yum-builddep kernel-2.6.21-1.3194.fc7.src.rpm
# rpmdev-setuptree
```

Now you will have a directory /root/rpmbuild/ with a couple of empty directories.

```
# rpm -Uvh kernel-2.6.21-1.3194.fc7.src.rpm
```

This will place the source in the right place.

```
# cd /root/rpmbuild/SPECS
# rpmbuild -bp kernel-2.6.spec
```

This is for a regular Fedora kernel, no LIDS this time. Otherwise that last step would have been preceded by modifications.

```
# cd /root/rpmbuild/BUILD/kernel-2.6.21/linux-2.6.21.i386/
# make rpm
```

Now the kernel has been built and it has been placed with a bunch of modules in an rpm file.

```
# cd /root/rpmbuild/RPMS/i386/
# rpm -ivh kernel-2.6.21prep-1.i386.rpm
# cd /boot
# mkinitrd /boot/initrd-2.6.21-prep.img 2.6.21-prep
# nano grub/menu.lst
```

Now you add a boot entry for the new kernel and initrd image. You don't have to use nano but I like it. Reboot the computer and hope for the best. If this worked as intended, now you can do the process all over again, with a few modifications. I'll repeat the whole thing for clarity, marking new sections in gray.

```
# cd /root
# yum install rpmdevtools.noarch
# yum install yum-utils
# cp /mnt/Public/Hardening/Kernel_stuff/kernel-2.6.21-1.3194.fc7.src.rpm /root
# yum-builddep kernel-2.6.21-1.3194.fc7.src.rpm
# rpmdev-setuptree
# rpm -Uvh kernel-2.6.21-1.3194.fc7.src.rpm
```

```
# tar -xvzf lidstools-2.2.7.2.tar.gz
# cp lids-2.2.3rc1-2.6.21.patch /root/rpmbuild/SOURCES
# nano /root/rpmbuild/SPECS/kernel-2.6.spec
```

Find the line that says something like "Patch17: DRM.patch" and add after that a new line saying "Patch18: lids-2.2.3rc1-2.6.21.patch". After that, find the line that says "%patch17 -p1" and add the line %patch18 -p1" on the next line. Ctrl+O writes the file to disk and Ctrl+X exits from nano.

```
# cd /root/rpmbuild/SPECS
# rpmbuild -bp kernel-2.6.spec
# cd /root/rpmbuild/BUILD/kernel-2.6.21/linux-2.6.21.i386/
```

```
# make menuconfig
```

The following configurations need to be met for LIDS to be available in the Security sub-menu.

Experimental -> Yes
Sysctl -> Yes
Security -> Yes
Security_seclvl -> No
Security_rootplug -> No
Security_selinux -> No
Security_capabilities -> No

In practice you only need to change SECURITY_SELINUX and SECURITY_CAPABILITIES from Yes to No. The other settings are already made by default in the Fedora kernel at least. Exit the configuration process and save the .config-file.

```
# nano Makefile
```

I recommend changing the EXTRA_VERSION variable from -prep to -LIDS.

```
# make rpm
# cd /root/rpmbuild/RPMS/i386/


# rpm -ivh --oldpackage kernel-2.6.21LIDS-1.i386.rpm

# cd /boot
# mkinitrd /boot/initrd-2.6.21-LIDS.img 2.6.21-LIDS
# nano grub/menu.lst


# cd /root/lidstools-2.2.3rc1
# run ./configure KERNEL_DIR=/root/rpmbuild/BUILD/kernel-2.6.21LIDS/
# make
# make install
```

During make install you have to set an LIDS* password which should probably be different than your root password in live environments. For testing of course the bar is lower.

* I think of LIDS as being pronounced El-Ai-Dee-Ess, so it's "**an** LIDS password".

Reboot and hope for the best.

Testing LIDS

Two things stood out during the initial test of LIDS.

1. It fails to prevent root escalation.

2. It's like trying to play volleyball while wading in molasses.

To LIDS credit, I could boot and login on the first attempt after compiling the LIDS kernel and userland tools. It's not a vanilla kernel but one rife with Fedora-specific patches. None the less. The ACL system is terse and prevent login via SSH(fair enough) and sometimes from the terminal(weird).

Developing a unified hacking suite

After some testing back and forth I decided to try to combine the privilege escalation exploit and the only backdoor I had been able to put into effect on a Linux system. With some copy-paste magic I ended up with a binary "exploit" which could be uploaded through my deliberately exploitable cgi-script. If the command to be run is "./files/exploit" the binary is run and I can connect to the remote host on port 1337 using the rat program. Reasonably one would use IPtables to only allow sockets to listen on ports in normal use but I can't be bothered to figure out how to do a reverse shell-thing so this is a substitute for that.

SELinux - Permissive

Having checked that this works on a Fedora 7 installation without SELinux, IPtables or any other protection software, the first step was see how SELinux performed in the weakest mode - permissive. Once the relabeling of files was completed Apache was started again and the whole procedure repeated. Under permissive rules the exploit worked, which isn't entirely strange. Violations are permitted a few times and logged before finally being banned. SELinux labels make the root shell have a type of httpd_sys_script_exec_t and while I didn't circumvent that I see no problems doing that as the root user has extensive power under Fedora's SELinux rules.

SELinux - Enforcing

At this level SELinux not only prevents me from executing the exploit uploaded via the web server, it doesn't actually allow me to upload the file. Manually changing the label on the files-directory to httpd_sys_script_rw_t allowed me to both upload and execute the exploit. Interestingly enough the exploit still doesn't work. That was precisely the kind of result I had been looking for! It turns out in the SELinux audit log that the file "exploit" tried to carry out a memory-modification that isn't allowed by SELinux. I remember that rule popping up earlier as it prevented Firefox from running on a CentOS installation with SELinux in enforcing mode, which made me turn it off. In Fedora this isn't an issue but even in the case of CentOS it seems far fetched that a server hardened with SELinux would run a graphical desktop environment. That leads us back to the issue of the privilege escalation hack requiring PulseAudio to be installed and capable of running(it needs ALSA and other subsystems to function or it doesn't run long enough to be useful for the exploit). Fedora doesn't have a policy to prohibit loading of kernel modules but this can be enabled by compiling the kernel yourself with appropriate settings.

I tried to change the settings of SELinux so I could at least get a root shell through my exploit. That way I could see if SELinux restricted the root shell on account of it being created by Apache/Python in some combination. In the end this seemed to require absurd dismantling of the SELinux rules for the web server and was not pursued. When you are looking to see if a security system works and it works so well that you can't dismantle it enough to see how it would operate in a crippled state, it might be for the best to see the writing on the wall. To use another idiom: don't look a gift horse in the mouth.

OSSEC

Some information came from a book called "OSSEC Host-Based Intrusion Detection Guide" by Andrew Hay, Daniel Cid, Rory Bray, which was available on Google Books.

Having worked - however briefly - with SELinux and LIDS I became more and more convinced that detection is the last line of defense. If a server is compromised the attacker can hide his intrusion but if an e-mail is sent every time someone gains root privileges it will be quite difficult to keep me out of the loop. Keep in mind the system for sending such messages would only be possible to shut down from the root account. It seems fairly clear that it wouldn't be possible to hide one's intrusion if such a system was in place. Well that's not entirely true... If an attacker is able to block the server from connecting to the destination of the e-mails or logs he could continue but that's no small precondition. So where does one turn for this kind of detection? Tripwire seems old and frail while AIDE seemed frail. OSSEC sports a more impressive and at first glance more robust functionality. Before you start, see if prelink is enabled on your system. If it is, turn it off. At http://www.atomicorp.com/forums/viewtopic.php?f=3&t=1750 you can read more about it, but the gist is that prelink modifies binaries for performance reasons which makes OSSEC flip its wig. Of course OSSEC should do that because one of its duties is to report modifications of important files.

The story of how I learned to use OSSEC consists large blocks of XML, console output and an assortment of C-code, with brief notes sprinkled on top. I will try to clean it up. Installing OSSEC is straight forward on CentOS and Fedora. Download the source code and run the install.sh script. This requires that you have GCC installed but that's an easy thing to accomplish with Yum. The virtual machine running OSSEC also runs Cacti and the latter spooked OSSEC with warnings of an unknown failure. Well, the Cacti poller does use the word "error" in the command it runs via Cron but OSSEC looks for those words whenever it is unable to determine the source of the message. So I added a new rule which states that this is a rule to be considered whenever rule 1002 has triggered(the rule for unknown error). It states that if the log entry alerting us to this unknown error contains the term "/poller.php", it is to be ignored. This is to be added to the file /var/ossec/rules/local_rules.xml

```
<rule id="100001" level="0">
  <if_sid>1002</if_sid>
  <match>/poller.php</match>
  <description>Cacti system polling.</description>
</rule>
```

For rules with frequency/timeframe attributes I figured out that frequency="6" timeframe="120" means the rule activates if the rule or its parent triggered 6 times in 120 seconds. You don't need timeframe but I don't know what the default timeframe is. A rule with an attribute ignore="60" is ignored by OSSEC for 60 seconds after triggering, the intention being to avoid flooding. I guess the default timeframe doesn't exist, or if you want to get technical it's "infinity". It keeps counting up and it never resets. I used the sshd rule warning about multiple failed login attempts to try out OSSEC's active-response feature. It works well but the rules containing frequency are a bit sluggish. Perhaps we just don't count the same way. To me, frequency="3" means that the rule triggers after 3 violations, not that it accepts 3 violations and triggers on the 4th.

The following entry in ossec.conf makes the main OSSEC server order a client that has received numerous failed ssh logins from one IP address to block connections from that host, where the only difference from the default is the field rules_id:

```
<active-response>
    <!-- Firewall Drop response. Block the IP for
       - 600 seconds on the firewall (iptables,
       - ipfilter, etc).
      -->
    <command>firewall-drop</command>
    <location>local</location>
    <level>6</level>
    <rules_id>5720,</rules_id>
    <timeout>600</timeout>
  </active-response>
```

I wanted OSSEC to be able to recognize messages from a kernel modification called GrSecurity(it's in the next section). A note deserves to be included as it paints a colorful picture of the process involved in making a new decoder.

Pardon me for getting informal for a minute: "I think that's straight now... Two hours, it's taken me to panel-beat my head back into shape! Two #### hours!" A quote by the android Kryten on the space ship Red Dwarf after he got into an argument with Cat. Of course it has taken me more like six hours to finally get a nice setup in OSSEC for handling GrSec events. Not that I had to panel-beat my head back into shape. So maybe we can call it a draw. Some problems:

1. Entries starting with "Jul 17 16:53:03 hostname kernel" will - somewhat counter-intuitively - match against the regular expression "^kernel", which defines quite explicitly that the first six letters in a matching string must be "kernel".

2. Entries starting with a date containing fewer than two digits - such as today's date: Jul 1 in syslog shorthand - will not match anything. Only by adding some more code to the cleanevent.c file can this be avoided.

3. There exists a decoder for all syslog entries starting with "kernel", and it is called iptables.

4. You can't nest decoders more than once.

5. A rule for entries decoded as kernel messages already exists, and it's rule 5100 in syslog_rules.xml.

Don't get me wrong, OSSEC is great but the documentation could be a bit more extensive considering the peculiarities of the system. Getting rid of the timestamp is great, but if you don't tell users this they will be confused when the regular expressions seemingly have no relevance to how text is matched. The same thing with the colon after kernel and grsec. How am I supposed to know "^grsec" matches both "grsec:" and "grsec: "? By trial-and-error obviously...

The modification mentioned for cleanevent.c yields the following if-statement, starting from line 92:

```
if(
        (
        (loglen > 17) &&
        (pieces[3] == ' ') &&
        (
        (
        (pieces[6] == ' ') &&
        (pieces[9] == ':') &&
        (pieces[12] == ':') &&
        (pieces[15] == ' ') && (lf->log+=16)
        )
```

```
        ||
        (
        (pieces[5] == ' ') &&
        (pieces[8] == ':') &&
        (pieces[11] == ':') &&
        (pieces[14] == ' ') && (lf->log+=15)
        )
        )
        )
        ||
        (
        (loglen > 33) &&
        (pieces[4] == '-') &&
        (pieces[7] == '-') &&
        (pieces[10] == 'T') &&
        (pieces[13] == ':') &&
        (pieces[16] == ':') &&

        (
         ((pieces[22] == ':') &&
          (pieces[25] == ' ') && (lf->log+=26)) ||

         ((pieces[19] == '.') &&
          (pieces[29] == ':') && (lf->log+=32))
        )

        )
      )
```

I don't quite understand why the lf->log variable is incremented but it seems to be a measurement of how many characters is in it so it stands to reason that we should add 15 instead of 16 since the only difference is that the numerical date value contains one fewer character. If I were feeling malicious I would suggest that the previously verbose and painfully clear explanation would serve the OSSEC documentation well. The decoder added to /var/ossec/etc/decoders.xml was:

```
<decoder name="grsec-user">
    <parent>kernel</parent>
    <regex>(grsec): \((\w+):\w:(\.+)\)</regex>
    <order>id, user, action</order>
</decoder>
```

Note that I have renamed the rule "iptables" and made the appropriate modifications to the actual iptables rules so that the rule matching kernel messages is called "kernel". In the end the decoder worked and with the following two rules added to local_rules.xml I got a rudimentary recognition of GrSecurity messages in OSSEC:

```
  <rule id="100002" level="2">
    <if_sid>5100</if_sid>
    <match>grsec</match>
    <description>GrSec event.</description>
  </rule>

  <rule id="100003" level="4">
    <if_sid>100002</if_sid>
    <match>denied</match>
    <description>GrSec access denied.</description>
  </rule>
```

GrSecurity

OSSEC was nice but as you have figured out by now I was looking for more. The problem with OSSEC is that root can shut it down, modify and so on. One can create new OSSEC binaries that don't alert when files have changed. What about adding a cryptographic salt in the source code used to make the binaries? Then an attacker would be in trouble because he won't know what salt to use for his own version! That isn't very secure either. The attacker could just make the syscheck binary run in a chroot jail with the monitored files copied into it. That way the hacker can modify the system to his heart's content without OSSEC sending an alert. In reality it's not entirely easy to do this without it entering logs on another server but I want more assurances that that. That's where GrSecurity comes in. It can lock running processes so that not even root can stop them and it can even hide their very existence. So it seemed like a good system for shoring up OSSEC.

Placing grsecurity.repo from Rpm Cormander(http://rpm.cormander.com/repo/grsec/) in /etc/yum.repos.d/ doesn't work too well since there's no repodata/repomd.xml in the kernel directories. Installing the right package directly by downloading them with wget and running "rpm -ivh kernel-grsec-2.6.32.14-104.i686.rpm" works just fine. There's no initrd packaged with the kernel so to make such a ramdisk we have to execute the command:

```
mkinitrd \
--builtin=ehci-hcd \
--builtin=ohci-hcd \
--builtin=uhci-hcd \
--builtin=ata_piix \
/boot/initrd-2.6.32.14-grsec-104.img 2.6.32.14-grsec-104
```

The builtin-bits keeps the mkinitrd binary from complaining that these modules can't be found. I made a quick inspection of the config file for the kernel packaged in the rpm and figured the missing modules were probably compiled in. To my amazement the virtual machine rebooted and worked flawlessly.

I tested the combined Wunderbar/Hole exploit with the same attitude as the Mythbusters when they test whether or not putting an apple inside a pyramid keeps it from decomposing(it doesn't). Keep in mind that GrSecurity is headed by the guy who wrote the Wunderbar Emporium hack, so it would be rather strange if he failed to plug that hole. Not that the 2.6.32-kernel has that hole in it, but even if it had the people at GrSecurity would have patched it.

Procedure

First create two gradm passwords:

```
gradm --passwd
```

[Use tough long password]

```
gradm -P admin
```

[Use another long password]

http://forums.grsecurity.net/viewtopic.php?f=3&t=2065&start=0 tells us that we have to comment out "inherit-learn /etc/rc.d/init.d" in /etc/grsec/learn_config because that can cause a problem on some systems.

Next:

```
gradm -F -L /etc/grsec/learning1.log
```

[Use system but not as root, as this would give root lots of privileges, which we don't want. root should have to go through GrSecurity to get those privileges.]

```
gradm -D
gradm -F -L /etc/grsec/learning1.log -O /etc/grsec/policy
```

Structure of ACLs

Stored in /etc/grsec/policy

Subjects: Processes

Objects: Files, capabilities, resources, IP.

```
<path of subject process> <optional subject modes> {
<file object> <optional object modes>
[+|-]<capability>
<resource name> <soft limit> <hard limit>
      connect {
            <ip>/<netmask>:<low port>-<high port> <type> <proto>
      }
      bind {
            <ip>/<netmask>:<low port>-<high port> <type> <proto>
      }
}
```

Some modifications:

/proc/kallsyms should be set to h for root's and users' subject / definitions.

```
        /proc/kallsyms                  h
        /proc/sys                      h
```

The following entry in /bin/touch under the root role gives it the power to create files in /root(GrSecurity's learning mode is very conservative):

```
            /root                              rwcW
```

It allows read, write, creation and the W makes GrSecurity audit writes(just to be on the safe side).

Drupal under GrSecurity

I wasn't sure GrSecurity could be trusted to learn how to allow valid services. We have all had firewalls that claim to be in learning mode and yet keep asking if process X is allowed to access the internet. The software doesn't even have to change between executions for the same question to be asked over and over again. To my amazement it worked fine. Drupal works nicely though at some point the site was not available, but that may well have been a result of me fiddling with the ACL.

OSSEC under GrSecurity

OSSEC is suspicious of the environment setup by GrSecurity. A first attempt at fixing warnings about rule 510 was to delete lines 177 to 200 in rootcheck/check_rc_pids.c. Then compile OSSEC(or if you've already installed OSSEC: go to syscheckd and run "make syscheckd" which will go back and recompile rootcheck(presumably... I recompiled roothceck manually)). It still does good work and scans files to see if they've been modified, so it's not ideal to turn it off completely. By the way: DON'T compile while running GrSec in learning mode. The log will be 18 MB in size and policy creation takes much longer than usual. It's just a right old mess.

GrSecurity learns about OSSEC very well and gives it appropriate access. I recommend setting all OSSEC subjects as h, so it should be

```
subject /var/ossec/bin/ossec-logcollector oh { etc...
```

Also, set /var/ossec as hidden in default root-rules and for bash and ls. This way it is not evident to an attacker that OSSEC is running or even installed. I mentioned earlier that administrative tasks should be performed when in GrSecurity's administrative mode and not using regular root, so don't start or stop OSSEC with root or any other normal user account when in learning mode.

One issue to consider is that of active response. To allow for this feature one could create a dummy rule that triggers when someone requests a web page called "trigger.html" and then run through all the active responses while the agent is in learning mode. Then GrSecurity would know that OSSEC is allowed to do things like add hosts to /etc/hosts.deny and modify the firewall. One might well be able to create rules such that root can perform these actions using ossec-execd but not directly from the command line.

Future investigations

Snort has always been on the list of security tools to test but it has been removed, to no fault of its own. The systems tested thus far have provided me with the functionality I require and surprisingly the confidence to trust them. There are definitely environments where I would want Snort to monitor the network but that day may not come for another couple of years. Since I don't have a switch with port mirroring I would have to test it by making a Snort device act as a gateway between to hosts which would work well but that isn't a sensible use of my time right now. The problem is exacerbated by my inability to carry out attacks over a network. How am I supposed to test Snort's ability to detect such things? So it is a future area of investigation.

AppArmor is - as I've understood it - like SELinux with a nice learning mode that figures out how various applications are allowed to operate. It bears investigating but SELinux seems to have superceded AppArmor. While SELinux has been mentioned herein I would like to learn more about it. Not least when it comes to data flow control SELinux appears to be the number one player. Nessus is another system that I should look into. One question is whether or not Nessus will trigger a harsh response from OSSEC or Snort, should the latter be used. Scanning for vulnerabilities is sort of red flag for an IDS like Snort. One thing that I may never get the opportunity to check is whether one can compromise a Xen dom0 by compromising a Xen domU. If so, does that apply to both fully and paravirtualized guests? The answer is probably going to be "no, you can't compromise the dom0 that way" because guest kernels aren't running directly on the dom0 kernel but on top of the hypervisor. Perhaps KVM is different though I have the feeling fully virtualized guests are less capable of attacking their hosts than paravirtualized ones and KVM thus far only supports fully virtualized guests.

Conclusions

So what have we learnt? Well, Linux operating systems are very hard to hack. You might be able to hack an application running on it but that doesn't translate well to accessing the underlying system. The best shot you had at compromising Linux servers was autumn 2009, provided you could execute arbitrary code and the server had a certain sound system installed. Should you be the nervous type or run servers that are often attacked or are very critical to the operation of the organization, SELinux as it comes with RHEL, CentOS and Fedora will work well. As described SELinux was not able to protect against the 2009 hack if it was run from a user account but when uploaded via the web server the hack wasn't able to run in the first place. My money is on GrSecurity. The ACLs for it are very lucid and allow you to determine what is and what isn't allowed when the system is in force. I wouldn't know how to construct a proof for a boss how SELinux prevents an attacker from stealing information if the web server were to have a vulnerability. There are so many rules that could allow an exception to make it very hard to issue anything similar to a guarantee. Policy analysis tools like apol can help but GrSecurity ACLs are much more clear. LIDS was too complicated to be useful and seems to be a subset of the functionality of GrSecurity, as the latter also includes a series of hardening techniques that make buffer overrun attacks and stack smashing nigh on impossible.

With OSSEC monitoring the system, attacks will be recorded and lead to automatic bans on the attackers IP address(if active response is enabled). Should an attack succeed it will quite likely show up in the logs stored on the OSSEC server. Experts have implored us to reduce the attack surface of our servers and only include software that is needed and now I agree with them without any hesitation. Don't just reduce the attack surface seen from the network. Pulseaudio was the stepping stone for the Wunderbar Emporium hack and its absence made it possible for CentOS to escape the attack despite my using an old kernel that was supposed to be vulnerable. On a more philosophical note I want to argue against the suggestions that people should stay away from things like SELinux rather than use them inappropriately. Yes, that could lead to a false sense of security but it would add at least some extra hindrance to an attacker. I could spend time learning SELinux and Snort really well, but the time I spent moving from point-and-click user to contributor of rules and features I could have become a point-and-click user of another three types of systems perhaps. So my argument is that the best return on investment in terms of security is to be obtained by learning to install systems and verify that they block a sample of attacks(which you do not need to understand). I intend to learn Snort but if the need should arise tomorrow I would deploy it and verify as best I could without knowing much at all about how it works.

Addendum: The nested parentheses on page 10 are intentional.