Hardened*OS exploitation techniques

Sebastian Krahmer

krahmer@cs.uni-potsdam.de

Abstract: To overcome the common 'find and fix' technique on todays operating systems, more and more people install special hardening patches or complete RBAC systems. One goal of such systems is to stop intruders from taking control over the whole system and to make exploitation of particular application bugs hard, if not impossible. This paper explores the possibilities intruders have nevertheless.

Contents

*I wanted to write 'Trusted OS' until I realized that there is a strong definition for this which is not met by the systems I looked at.

1 Introduction

In this paper I write down some ideas I had during analyzation of various 'hardening patches' for the Linux Kernel. Most of these systems add very little additional security such as LIDS [4]. On the other hand, there are some patches which offer complete ACL and/or RBAC systems, trusted path execution and other nice things to make life for intruders as hard as possible. Two of these systems are *SELinux* and *grsecurity*. I will focus on *grsecurity* and the weaknesses I have found during my research, but most things apply to other hardening patches as well.

2 Exploiting, the common way

Observing the privilege escalation exploits for *UNIX* systems over many years, experience shows that they can be divided into two classes:

- passive exploits
  Passive exploits require no additional actions by the user which the attacker tries to compromise. Passive exploits directly drop you in a shell with the elevated privileges. An example for this is the traceroute [6] buffer overflow exploit which drops the attacker into a root shell. No action by root is required for the attack to succeed, except to forget to install the proper update.
- active exploits
  Active exploiting requires any kind of action by the user being attacked. This ranges from clicking to an evil link in an email to invoking certain programs which contain `/tmp` race conditions but are not setuid or to Trojan fi les he is executing.

This said, and looking back for some years, we see that local passive exploits caused much more trouble in past compared to local active exploits. Obviously they are much easier to launch and the possibility to succeed is higher because the user-factor is erased. However imagine a bug where it would be possible for an attacker to replace the `/bin/su` binary. As a result the attacker could steal the passwords of the system, namely the root password.

3 User-scoped Trojan horses

Unfortunally in view of the attacker, the `/bin/su` binary cannot be replaced on most systems because users usually do not have the permissions to do so. In case the user-account was broken into and is used to *su* to the root account, the attacker has an additional option: He can create a pty wrapper for the *su* program or setting aliases to *su* which actually execute Trojan horses. An example session is shown here, utilizing the *loga* screen-capturing program:

method	example	affects
shell aliasing	`alias su=id; su`	all command-line tools as shown in pty sniff example
shell hashing	`hash -p /usr/bin/id su; su`	same as shell aliasing
shell functions	`su(){ id; }; su`	same as shell aliasing
pre-loading	`LD_PRELOAD='pwd'/evil.so ssh`	all non-setuid command-line tools

Figure 1: Some methods to place user-scoped Trojans to a system.

```
stealth@linux:~> ls -la proxy
-rwxr-xr-x    1 stealth  users       23017 2003-12-03 09:37 proxy
stealth@linux:~> alias /bin/su='pwd'/proxy
stealth@linux:~> /bin/su
Password:
linux:/home/stealth # id
uid=0(root) gid=0(root) groups=0(root)
linux:/home/stealth # exit
exit
stealth@linux:~> ./proxy decrypt /tmp/mcX192YZ5002224
Password: secret
linux:/home/stealth # iidd
uid=0(root) gid=0(root) groups=0(root)
linux:/home/stealth # eexxiitt
exit
stealth@linux:~>
```

The attacker sets an alias to `/bin/su` which indeed executes a Trojan horse which records the session along with the password encrypted to `/tmp/mcX192YZ5002224`. The *loga* program is part of the MSS [7] suite.
Other nice targets for user-scoped Trojans are *ssh*, *telnet* or *passwd*. Figure 1 shows how attackers could place Trojans to a system that only affect a certain user.

The LIDS [4] protection for example can be disabled with the shell-function trick. Usually the `/etc/rc.d/halt` script holds the `CAP_SYS_ADMIN` capability to be able to unmount the filesystems during shutdown. However by exporting apropriate shell functions, the script is executing code of the attacker instead of internal shell commands.
Trojan horses can be thought as a special subset of active exploits. Weak configuration or operating systems without proper user separation suffer very much on Trojan horses. On such systems (imagine, write permissions to `/lib` or `/root`) a Trojan horse would be an active exploit for the attacker to escalate the privileges. The action required to succeed is the invocation by the attacked user.
As we will see later these attacks are very practical on RBAC and ACL systems once the attacker compromised the root account but is still restricted by ACLs.

4 If root is not root

Adding an additional hardening patch to the Kernel also adds a new problem. The userspace programs and daemons have been written for a completely different environment. They expect that a UID of 0 means unrestricted access. With ACL systems however *crontab* for example could be invoked two times with UID 0 but with different permissions. Bugs which have not been fixed because the code could only be triggered by root becomes dangerous now.[1] A complete code review of the daemons is again necessary, this time under the view of ACL-systems.
Certain daemons need to be rewritten completely if they are too complex to restrict them with a view ACLs or one label. The *sshd* daemon for example can be abused to read out arbitrary ASCII files by placing appropriate `Banner` options to the configuration file. If you allow *sshd* to read hidden files from `/etc/ssh/` such as *grsecurity* does, everyone could abuse the `Banner` option to read out these hidden files. It looks as if the impact is very limited, but on ACL systems the security is based on the access restrictions and the given example leads to a complete compromise of the ACL system eventually.

5 A grsecurity example

The *grsecurity* patch for the Linux kernel offers a lot of measurements to make exploitation of active and passive vulnerabilities harder or to prevent them at all. Among a lot of other protections this includes

- PaX
  By removing execute permissions of pages belonging to a process' `.data` segment and the process' stack, PaX makes exploitation of buffer overflows much harder if not impossible.
- randomized mmap()
  By randomizing the base address of the libraries mapped to the process' address space and by randomizing even the base address of the executable mapped to memory its almost impossible to get programs vulnerable to buffer overflows to execute code that drops the attacker into a shell or something similar.
- ACLs
  By setting appropriate restrictions to system paths such as removing write permissions from `/etc, /lib, /bin` and so on, attackers have no chance to place Trojan horses there as long as the protection is enabled.
- fine grained privileges
  Grsecurity allows to grant certain binaries special capabilities such as the capability to bind to a privileged port. This way, the program does not longer need the setuid-root bit and therefore a potential hole is closed.

[1] such bugs exist indeed in *crontab*, which gets a own label on *SELinux*

The first two items directly impact the behavior of passive overflow exploits. Given that the information where the libraries are mapped to do not leak to the attacker, he has no way to know where his evil input is located at [2] and to execute it at all. His input is data, and as a general rule here, data cannot be 'executed' as long as it is located outside the `.text` segment which is read-only.

The third item 'ACL' affects the active exploits. Once an attacker got root, somehow, he wont be able to place Trojan horses to the system because the important places are all write-protected. Note, the active exploit *grsecurity* protects you against, is an exploit that escalates the privileges from a normal root account to an account with special privileges to bypass ACL checks (admin role) or an account from which *grsecurity* can be disabled.

To summarize, an attacker has two choices of 'hacking':

- least privilege
  Once a user-account was compromised, do not try to get root or to disable *grsecurity*. Just install Trojan horses in a user-local scope to catch SSH passwords for example by installing special `LD_PRELOAD`able libraries or by installing the *loga* program and setting certain aliases. There's not much an Hardened OS can do about this. The users homedir needs to be writable and executable in most cases and this suffices for this kind of attack.
- highest privilege possible
  Once a user-account was compromised, try to get root and try to disable *grsecurity* or try to reach a admin role.

The first case is not to underestimate, but this paper concentrates on disabling the Hardened OS protection measurements, so we only look at the second case. In most cases it will not be enough to execute an passive root-exploit since it may drop you into a root-shell if you figure out how to exploit buffer overflows in *PaX* but you still do not have full permissions. So a two-level exploit is needed

- user to root elevation
- root to admin role [3] elevation

6 Exploits for grsecurity

While user-to-root elevation gets much harder on systems like *grsecurity*, they do not differ much from active or passive exploits for 'normal' operating systems. Attackers still need to look for holes in suid programs and daemons etc. However the way of exploitation

[2] which he needs to know to get it executed
[3] Or any other level which is needed to disable *grsecurity*

becomes different because, for example, code on the stack cannot be executed or symlinks are not followed. Once a normal root exploit succeeded, its likely that a second exploit is needed to disable *grsecurity*.[4] How such a second exploit could look like is explained now.

6.1 Active exploits for grsecurity

Lets now consider two active attacks which allows attackers to steal the password which is needed to disable *grsecurity*. Much as with any kind of software, a Hardened OS may contain bugs and of course needs to be configured correctly.
The configuration of *grsecurity* 1.99 contained a flaw in the default config which allowed attackers to trick root into executing Trojan horses or to sniff the `gradm` password. Usually you won't be able to place any Trojan horses for root, as the complete `$PATH` and `/root` is write protected. However, attackers who already gained root permissions but are still restricted by ACLs may kill the `sshd` daemon and replace it by a own daemon which logs plaintext traffic to a file under attackers control or which allows of setting a new `$PATH` variable. This has been fixed in newer versions of *grsecurity* by making the ssh host-keys not readable once ACL's are enabled. This prevents attackers from restarting a patched SSH daemon and still prompting the correct host-key to the legitimate admin. However, remember that *sshd* can be abused to read out the ssh host-keys with the `Banner` option which again opens the same hole. This should show that it can be very hard to have no holes within the ACL configuration.

Another active vulnerability is shown below. *grsecurity* 2.0-rc3 does not remove the `CAP_SYS_ADMIN` capability by default, allowing attackers to play tricks with the `mount` command:

```
[root@lalawesi root]# cd /tmp/
[root@lalawesi tmp]# gradm2 -E
[root@lalawesi tmp]# dmesg|tail -3
eth0: no IPv6 routers present
grsec: From 192.168.0.2: Loaded grsecurity 2.0
[root@lalawesi tmp]# mkdir sbin
[root@lalawesi tmp]# cd sbin
[root@lalawesi sbin]# cat>gradm2.c
int main() { printf("Hello from trojan.\n"); return 0;}
^C
[root@lalawesi sbin]# cc gradm2.c -o gradm2
[root@lalawesi sbin]# mount -n --bind /tmp/sbin/ /sbin/
[root@lalawesi sbin]# ls -la /sbin/
total 24
drwxr-xr-x    2 root     root         4096 Dec 15 17:03 .
```

[4] As long as the attacker hasnt hit the jackpot with a kernel bug or alike.

```
drwxr-xr-x   24 root     root         4096 Dec 15 17:00 ..
-rwxr-xr-x    1 root     root        11371 Dec 15 17:03 gradm2
-rw-r--r--    1 root     root           56 Dec 15 17:03 gradm2.c
[root@lalawesi sbin]# gradm2 -D
Hello from trojan.
[root@lalawesi sbin]#
```

The attacker re-binds a directory tree under his control into a write-protected area. Thus he effectively replaces the `gradm` utility and can capture the admin role password even without getting the admin a clue about it. It is however not possible for an attacker to re-bind his own libraries to `/lib` and steal certain capabilities from binaries since the rebound location is not write-protected in the eyes of *grsecurity* and it will therefore deny the mapping.

6.2 Passive exploits for grsecurity

I am not aware of passive exploits against *grsecurity* except a modified version of the brk() exploit.[5]. There is little a hardening patch can do about bugs in software it has to trust. However there are some ideas written down in [3] how the kernel can be protected from such attacks.
Beside bugs inside the kernel which could lead to a full compromise, a lot of other aspects have to be viewed. My research concentrates on the following issues:

- The possibility to leak capabilities from higher privileged programs to the attacker [8]
- The possibility to bypass the gr authentication
- The possibility to load/execute arbitrary code with elevated capabilities by playing with the dynamic linker and/or shell-scripts
- The possibility to bypass the ACL checks on the filesystem

7 Summary

Todays computer systems offer no special protection against skilled attackers. Most software suffers from buffer overflow vulnerabilities which allows remote attackers to gain complete control over the system if properly exploited. To protect against such attacks, special hardening patches need to be installed. The quality of these patches ranges from a null-effect to stable and quite secure RBAC systems. However even on high-quality hardening systems the attacker still has some options to get a feet into the door and where the hardening patch cannot do much about it. Especially weak default configurations allow attackers to circumvent various protection measurements. Additionally it is questionable

whether attackers need to elevate their privilegs to the highest privilege possible in future. They may just install user-scoped Trojan horses to gather important information.

References

[1] grsecurity: http://www.grsecurity.net

[2] PaX: http://pax.grsecurity.net/

[3] PaX-future: http://pax.grsecurity.net/docs/pax-future.txt

[4] LIDS: http://www.lids.org

[5] brk() bug: http://isec.pl/papers/linux_kernel_do_brk.pdf

[6] traceroute exploit: http://stealth.openwall.net/exploits/traceroot.c

[7] pty-sniffer: http://stealth.openwall.net/local/mss-0.33.tgz

[8] LIDS-hack: http://stealth.openwall.net/lids-hack.tgz