Limiting syscall access for a Linux application

Asked 11 years, 10 months ago Active 1 year, 5 months ago Viewed 2k times

6

2

Assume a Linux binary `foobar` which has two different modes of operation:

- Mode A: A well-behaved mode in which syscalls `a`, `b` and `c` are used.
- Mode B: A things-gone-wrong mode in which syscalls `a`, `b`, `c` and `d` are used.

Syscalls `a`, `b` and `c` are harmless, whereas syscall `d` is potentially dangerous and could cause instability to the machine.

Assume further that which of the two modes the application runs is random: the application runs in mode A with probability 95 % and in mode B with probability 5 %. The application comes without source code so it cannot be modified, only run as-is.

I want to make sure that the application cannot execute syscall `d`. When executing syscall `d` the result should be either a NOOP or an immediate termination of the application.

How do I achieve that in a Linux environment?

linux security linux-kernel hook system-calls

Share Improve this question Follow

edited Jan 27 '10 at 10:38

asked Jan 27 '10 at 10:30

knorv

46.4k ● 72 ● 205 ● 290

Could you perhaps clarify the meaning of "probability" in your question in case there is something to misunderstand there? – Pascal Cuoq Jan 27 '10 at 10:35

Pascal: Sure! Post edited with a clarification. For the purpose of this questions the "mode choice" is random. – knorv Jan 27 '10 at 10:39

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies Customize settings

Report this ad

4 Answers

Active | Oldest | Votes

8

Is the application linked statically?

If not, you may override some symbols, for example, let's redefine `socket`:

```
int socket(int domain, int type, int protocol)
{
        write(1,"Error\n",6);
        return -1;
}
```

Then build a shared library:

```
gcc -fPIC -shared test.c -o libtest.so
```

Let's run:

```
nc -l -p 6000
```

Ok.

And now:

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies | Customize settings

Share Improve this answer Follow

edited Jun 15 '20 at 9:14
Community Bot
1 • 1

answered Jan 27 '10 at 10:45
Artyom
29.5k • 20 • 122 • 210

This actually does not fulfil the security requirements? It is fairly easy to get code to load an appropriate syscall number in `eax` or whatever the CPU conventions is and then transfer control to the tail end of 'a', 'b', or 'c' syscalls. Altering a syscall table per process and loader changes (like SELinux) are the only way to stop user space corruption. See for instance ROP at wikipedia. At least this answer assumes several things in the system are beyond compromise. – artless noise Nov 3 '15 at 15:39

7

It seems that systrace does exactly what you need. From the Wikipedia page:

> An application is allowed to make only those system calls specified as permitted in the policy. If the application attempts to execute a system call that is not explicitly permitted an alarm gets raised.

Share Improve this answer Follow

answered Jan 27 '10 at 10:57
Torsten Marek
76.8k • 20 • 89 • 96

4

This is one possible application of sandboxing (specifically, Rule-based Execution). One popular implementation is SELinux.

You will have to write the policy that corresponds to what you want to allow the process to do.

Share Improve this answer Follow

edited Dec 17 '19 at 15:27
Burkart
460 • 4 • 9

answered Jan 27 '10 at 10:33
Pascal Cuoq
76.4k • 6 • 148 • 268

This is certainly the use case for SELinux. Other sandboxing technologies are available. – stsquad Jan 27 '10 at 15:06

@stsquad I incorporated your comment. You were perhaps reacting partly to the "claims" in the previous version… I phrased it this way because of having heard some people SELinux is not so usable in practice, precisely because of the need for adequate policies. Not having tried it, I do not have an opinion one way or the other, so perhaps the new version is better from this point of view. – Pascal Cuoq Jan 27 '10 at 15:27

Your privacy
By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies Customize settings

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings