



seccomp-nurse

Nicolas Bareil

EADS CSC – Innovation Works
France

ekoparty 2010

Executive summary

New sandboxing environment on Linux

- Focusing on headless applications (think: “cloud”!)
- No recompilation
- No kernel patches
- Free Software (GPL)

Why another one?

State of art

- Chroot+capabilities (+ Capsicum)
- ptrace() based
- Role Based Access Control (based on LSM)
- Virtualization

Why another one?

State of art

- Chroot+capabilities (+ Capsicum)
 - Huge attack surface
 - Jail evasion easy without kernel patches
- ptrace() based
- Role Based Access Control (based on LSM)
- Virtualization

Why another one?

State of art

- Chroot+capabilities (+ Capsicum)
- ptrace() based
 - Big attack surface
 - Complex to safely validate a syscall
 - Slow
- Role Based Access Control (based on LSM)
- Virtualization

Why another one?

State of art

- Chroot+capabilities (+ Capsicum)
- ptrace() based
- Role Based Access Control (based on LSM)
 - Huge attack surface
 - Brad Spengler proved his point many times
- Virtualization

Why another one?

State of art

- Chroot+capabilities (+ Capsicum)
- ptrace() based
- Role Based Access Control (based on LSM)
- Virtualization
 - Qubes OS
 - Just a shift of the attack surface

Why another one?

State of art

- Chroot+capabilities (+ Capsicum)
- ptrace() based
- Role Based Access Control (based on LSM)
- Virtualization

And the Google Chrome approach, based on SECCOMP.

CONFIG_SECCOMP

SECure COMputing

- Linux specific (neither POSIX, nor SUS or BSD)
- Introduced in 2.6.23 (2005)
- By Andrea Arcanlegi

CONFIG_SECCOMP

- Voluntary entrance by calling `prctl()`
- Four system calls allowed
 - `read()`
 - `write()`
 - `sigreturn()`
 - `_exit()`

CONFIG_SECCOMP

- Voluntary entrance by calling `prctl()`
- Four system calls allowed
 - `read()`
 - `write()`
 - `sigreturn()`
 - `_exit()`

Any deviance drives to SIGKILL



SECCOMP as sandboxing method?

Welcome seccomp-nurse!

- Fail safe
- Kernel's attack surface **really** limited^a
- So limited that...

^aMore details at <http://bit.ly/aoxCEX>

SECCOMP as sandboxing method?

Welcome seccomp-nurse!

- Fail safe
- Kernel's attack surface **really** limited^a
 - On 440 vulnerabilities, only 13 were triggerable from a SECCOMP process
- So limited that...

^aMore details at <http://bit.ly/aoxCEX>

Even that program get killed

```
#include <unistd.h>
#include <sys/prctl.h>

#define S "Hello Ekoparty!\n"

int main(int argc, char **argv) {
    prctl(PR_SET_SECCOMP, 1, 0);
    write(STDOUT_FILENO, S, sizeof S);

    return 0;
}
```

Because "return 0" calls `exit()` and not `_exit()` :-)

Even that program get killed

```
#include <unistd.h>
#include <sys/prctl.h>

#define S "Hello Ekoparty!\n"

int main(int argc, char **argv) {
    prctl(PR_SET_SECCOMP, 1, 0);
    write(STDOUT_FILENO, S, sizeof S);

    return 0;
}
```

Because “return 0” calls `exit()` and not `_exit()` :-)

Problematic

Two problems need to be solved to use SECCOMP as a sandbox:

- How to enter in SECCOMP mode?
- How to prevent applications to make forbidden system calls?

Problem #1: Entering SECCOMP

Constraints

- Without `ptrace()`
- No recompilation
- No instruction rewriting on-the-fly

Solutions

- `LD_PRELOAD` trick (`__libc_start_main`)
- GNU Linker feature: `LD_AUDIT`

Problem #1: Entering SECCOMP

Constraints

- Without `ptrace()`
- No recompilation
- No instruction rewriting on-the-fly

Solutions

- `LD_PRELOAD` trick (~~`--libc_start_main`~~)
- GNU Linker feature: `LD_AUDIT`

Problem #1: Entering SECCOMP

LD_AUDIT

```
$ man rtld-audit
```

The GNU dynamic linker (run-time linker) provides an auditing API that allows an application to be notified when various dynamic linking events occur.

1 Creation of an audit library which:

- Allocate some pages for our code/variable
- Intercept syscalls
- Enter into SECCOMP

```
2 /lib/ld-linux.so.2 --audit ./sandbox.so /bin/ls
```

Problem #1: Entering SECCOMP

LD_AUDIT

- 1 Creation of an audit library which:
 - Allocate some pages for our code/variable
 - Intercept syscalls
 - Enter into SECCOMP
- 2 `/lib/ld-linux.so.2 --audit ./sandbox.so /bin/ls`

Barely used feature

- Only one known application using it (latrace^a)
- Thread Local Storage not behaving normally

^a<http://people.redhat.com/jolsa/latrace/>



Problem #2: SIGKILL

Preventing application from doing forbidden syscalls

Problem #2: SIGKILL

Preventing application from doing forbidden syscalls

- Without `ptrace()`
- Without library overloading
- Without recompilation

Problem #2 : SIGKILL

How do syscalls work?

```

kill :
  mov    %ebx,%edx
  mov    0x8(%esp),%ecx
  mov    0x4(%esp),%ebx
  mov    $0x25,%eax
  call   *%gs:0x10
  mov    %edx,%ebx
  cmp    $0xffff001,%eax
  jae    2aa30 <kill+0x20>
  ret
  
```

- On x86, the libc no longer inlines “int 0x80”
- Instead, it calls an indirect function stored in VDSO

call *%gs:\$0x10

(This is what the mysterious libc6-686 package provides)

Problem #2 : SIGKILL

Hijacking VDSO

```
call *%gs:$0x10
```

- %gs is overwritten to point into our handler
- From now on, every (legit) syscalls are intercepted

Well, syscalls are intercepted... and now?

The syscall handler still runs into SECCOMP...

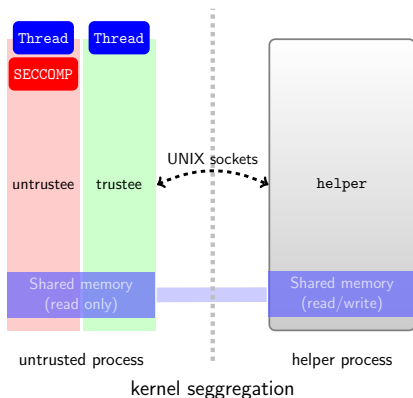
Well, syscalls are intercepted... and now?

The syscall handler still runs into SECCOMP...

It needs to be assisted by another process, the helper

- 1 The untrusted application makes a syscall
- 2 The handler intercepts it and notifies the helper
- 3 The helper does something
- 4 The helper pass a return value to the handler
- 5 The handler gives the return value to the untrusted application

Well, syscalls are intercepted... and now?



- Two processes
- In the untrusted process, two threads:
 - Trustee
 - Untrustee, under SECCOMP
- The original program and our handler runs in untrustee

Good fences make good neighbors

- When a syscall is made in the untrustee, our handler kicks in and notifies the helper
- The helper is written in Python
 - Implementing access control (policy engine)
 - Delegating syscall execution to the trustee
- The trustee is a tiny assembly routine
 - Executing orders from the helper

Thread magic^Wwoodoo

Programming 101

Threads share **everything** except CPU registers

- File descriptors
- Address space
- Locks

Any action in a thread is propagated in the other.

Thread magic^Wvoodoo

Programming 101

Threads share **everything** except CPU registers
Any action in a thread is propagated in the other.

Cool!

The trustee will perform syscalls on behalf of the untrustee

Thread magic^Wwoodoo

Programming 101

Threads share **everything** except CPU registers

Any action in a thread is propagated in the other.

Cool!

The trustee will perform syscalls on behalf of the untrustee

WARNING

So the trustee code runs in a hostile environment.

It can only use CPU registers!

Helper: policy engine

When a syscall is intercepted and then sent to the helper, the security policy is checked. Like:

```
class Security:
    def open(self, filename, perms, mode):
        path = os.path.realpath(filename) # Bug #990669
        for authorized_path in self.fs_whitelist:
            if path.startswith(authorized_path):
                return True
        return False

    def access(self, filename, mode):
        return True # XXX
```

Demo time!

Hope it will work (or cheat-sheet in appendix 6)...

Status

What is implemented?

- Core system
- Trustee “self-protection”

What is **not** implemented?

Most of the policy engine :(

Status

What is implemented?

- Core system
- Trustee “self-protection”

What is **not** implemented?

Most of the policy engine :(

Is it usable?

Yes!

You can already run most of classic “conversion” tools:

- Image manipulation: no more libpng harm!
- PDF transformation: do not be afraid of opening PDF!
- Python interpreter: think Google App Engine!

Limitations

Sorry...

- Linux specific
- Needs a “recent” GNU Libc (> 2005) compiled for 686
- Will (most likely) never support:
 - `clone()`
 - `execve()`
- `dlopen()` not yet implemented



Future

- Implementation of more syscalls
- Use Google Chrome's `libseccomp-sandbox`
- Get peer reviews!

Future

- Implementation of more syscalls
- Use Google Chrome's `libseccomp-sandbox`
- Get peer reviews!

HIT ME!



seccomp-nurse: sandboxing environment

└ Project' status

Thank you!

<http://chdir.org/~nico/seccomp-nurse/>

```
foobar@debian32:~/python-2.6$ sandbox -- ./python
Python 2.6.5+ (release26-maint:82382M, Jul 6 2010, 15:41:57) [GCC 4.4.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> fd=open('/etc/resolv.conf')
>>> for line in fd:
...     print line,
...
nameserver 192.168.9.2
domain vmlab
search vmlab
>>> import os
>>> os.getpid()
27997
>>> open('/secret/password')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 1] Operation not permitted: '/secret/password'
>>> os.access('/secret/password', os.R_OK)
True
```