EADS

seccomp-nurse

Nicolas Bareil

EADS

Executive summary

Environnement de sandboxing

- Sans recompilation
- Sans modification du noyau

EADS

CONFIG_SECCOMP

SEC*ure* **COMP***uting*

- Spécifique Linux (ni POSIX, ni SUS, ni BSD)
- Introduit dans le 2.6.23 (2005)
- Par Andrea Arcanlegi

EADS

CONFIG_SECCOMP

- Entrée volontaire via `prctl()`
- Quatre appels systèmes autorisés
  - `read()`
  - `write()`
  - `sigreturn()`
  - `_exit()`

EADS

CONFIG_SECCOMP

- Entrée volontaire via `prctl()`
- Quatre appels systèmes autorisés
  - `read()`
  - `write()`
  - `sigreturn()`
  - `_exit()`

Tout écart mène au SIGKILL

EADS

Sandbox

Cible : *grid computing*

- Fait tourner des logiciels arbitraires
- Ne nécessite pas de recompilation
- Sûre

EADS

SECCOMP + Sandbox = seccomp-nurse

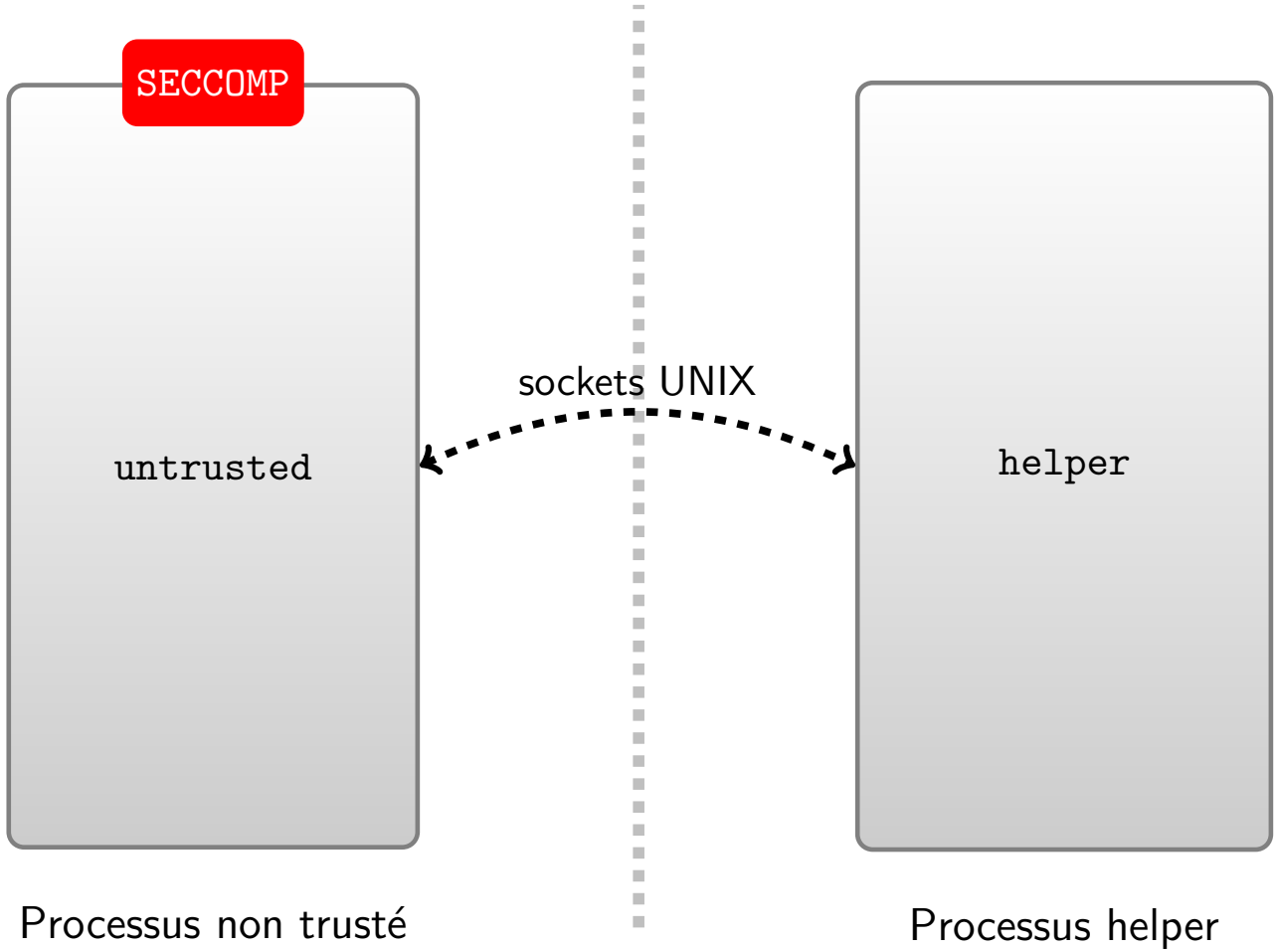
- Fail safe
- Surface d'attaque **très** limitée
- Tellement sûr qu'inutile...

EADS

SECCOMP + Sandbox = seccomp-nurse

- Fail safe
- Surface d'attaque **très** limitée
- Tellement sûr qu'inutile...

...sauf s'il est aidé !


SECCOMP
untrusted
sockets UNIX
helper
Processus non trusté
Processus helper
Ségrégation kernel

EADS

Problème #1

Rentrer dans SECCOMP

- Sans `ptrace()`
- Sans recompilation

EADS

Problème #1 : Rentrer dans SECCOMP

Solution #1

Binaire compilé avec la GNU Libc

- ELF entry point : routine `_start`
- `call __libc_start_main@plt`
- `main()`

EADS

Problème #1 : Rentrer dans SECCOMP

Solution #1

Binaire compilé avec la GNU Libc

- ELF entry point : routine `_start`
- `call __libc_start_main@plt`
- `main()`

`LD_PRELOAD` permet de surcharger des symboles

EADS

Problème #1 : Rentrer dans SECCOMP

Solution #1 : Problèmes

- Ne supporte pas les binaires statiques
- Ne supporte que les binaires compilés avec la GNU Libc

Hypothèse d'usage : *grid computing*

Nous ne ciblons pas l'exécution de programmes malveillants.

EADS

Problème #1 : Rentrer dans SECCOMP

Solution #2 : 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. Création d'une bibliothèque d'audit
2. `/lib/ld-linux.so.2 --audit ./sandbox.so /bin/ls`

EADS

Problème #1 : Rentrer dans SECCOMP

Solution #2 : LD_AUDIT

```
$ man rtld-audit
```

1. Création d'une bibliothèque d'audit
2. `/lib/ld-linux.so.2 --audit ./sandbox.so /bin/ls`

Fonctionnalité inconnue

- Un seul client connu de l'interface
- Gestion délicate du *Thread Local Storage*

EADS

Problème #2 : SIGKILL

Éviter les appels système interdits

EADS

Problème #2 : SIGKILL

Éviter les appels système interdits

- Sans `ptrace()`
- Sans surcharge de toutes les fonctions la Libc
- Sans recompilation

EADS

Problème #2 : SIGKILL

Comment fonctionnent les appels systèmes ?

- Sur x86, les appels systèmes ne sont plus fait à base de « `int 0x80` » inlinés
- Il faut désormais appeller une fonction disponible dans la VDSO qui utilisera la meilleur implémentation

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

(C'est ce qu'apporte le paquet `libc6-686`)

EADS

Interception des syscalls

- Tous les registres sont envoyés au helper
- Aucune intelligence, il ne fait que suivre les ordres :
  - Écrire/Lire à une adresse mémoire
  - Émuler la valeur de retour du syscall
  - Se terminer proprement (via `_exit()`)

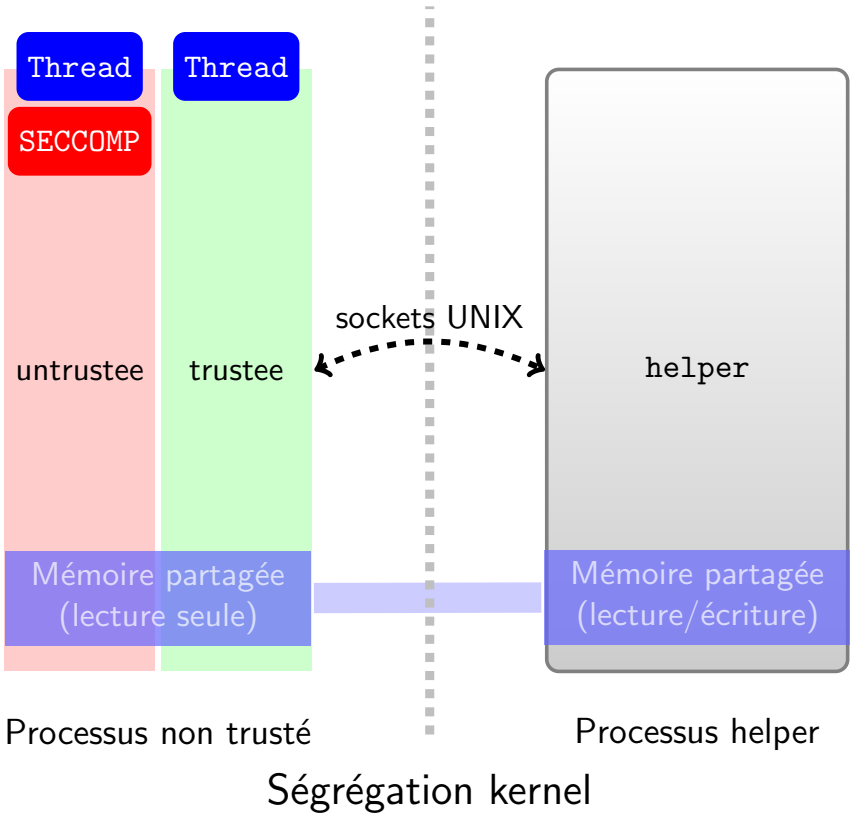
EADS

Version initiale

- Tout était émulé dans l'autre processus
- Lourd avec la sensation de réécrire le noyau en Python :)
- Performance : Double copie des données lors de `read()`/`write()`

EADS

seccomp-nurse 2.0



- Toujours deux processus
- Dans le processus non-trusté, deux threads :
  - Un trusté
  - Un non-trusté, sous `SECCOMP`

EADS

Partage de la mémoire

Problématique

Les threads partagent **tout** !
⇒ Toute action d'un thread impacte l'autre

Le thread trusté n'a qu'une seule tâche : exécuter les syscalls que le processus helper lui ordonne de faire.

EADS

```
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 sys
>>> sys.path.append('/usr/lib/python2.6/')
>>> 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
>>> open('/var/log/../../../../../../secret/password')
Traceback (most recent call last) :
  File "<stdin>", line 1, in <module>
IOError : [Errno 1] Operation not permitted : '/var/log/../../../../../../secret/password'
```

EADS

Ce qui est implémenté

- open
- close
- access
- getcwd
- getpgrp
- getpid
- gettimeofday
- exit
- lseek
- llseek
- readlink
- stat64
- fstat64
- mmap2
- mmap
- brk
- ugetrlimit
- munmap
- time
- times

EADS

Ce qu'il reste à faire

Toutes les vérifications de sécurité

```
class Security:
    def open(self, filename, perms, mode):
        path = os.path.realpath(filename) # Bug #990669
        if not path.startswith(self.chroot):
            return False
        ...
        return True
```

EADS

Ce qu'il sera difficile d'implémenter

Tous les appels systèmes démultiplexés

(sockets)

EADS

Ce qu'il sera difficile d'implémenter

Tous les appels systèmes démultiplexés

(sockets)

Et `dlopen()` ?!?

EADS

Limitations

- Spécifique à Linux
- glibc récente ($>$ 2005) compilée pour 686
- Ne supportera jamais ces syscalls :
  - `clone()` : thread ou fork impossible
  - `execve()`

EADS

Avez-vous des questions ?