Seccomp

The secure computing framework

Brendan Guevel

November 29, 2018

Summary

History

HISTORY

History

Back in 2005

Andrea Arcangeli has an idea

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky
- One needs an assurance that foreign code will not harm its computer

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky
- One needs an assurance that foreign code will not harm its computer

Solution : Seccomp

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky
- One needs an assurance that foreign code will not harm its computer

Solution : Seccomp

- Restrict the system calls that a process may make

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky
- One needs an assurance that foreign code will not harm its computer

Solution : Seccomp

- Restrict the system calls that a process may make
- Sandbox the process

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky
- One needs an assurance that foreign code will not harm its computer

Solution : Seccomp

- Restrict the system calls that a process may make
- Sandbox the process
- First version : only read(), write(), exit() and sigreturn()

History

Back in 2005

Andrea Arcangeli has an idea

- Enable owners of Linux systems to rent out their CPUs
- But execute strangers code inside our machine is risky
- One needs an assurance that foreign code will not harm its computer

Solution : Seccomp

- Restrict the system calls that a process may make
- Sandbox the process
- First version : only read(), write(), exit() and sigreturn()
- Any other syscall -> Sigkill !

History

And then

Seccomp integrated in linux kernel (2.6.12)

History

And then

Seccomp integrated in linux kernel (2.6.12)

- Andrea's CPU sharing idea never took off...

History

And then

Seccomp integrated in linux kernel (2.6.12)

- Andrea's CPU sharing idea never took off...
- ... but introduced a nice sandboxing feature

History

And then

Seccomp integrated in linux kernel (2.6.12)

- Andrea's CPU sharing idea never took off...
- ... but introduced a nice sandboxing feature

What happens next ?

History

And then

Seccomp integrated in linux kernel (2.6.12)

- Andrea's CPU sharing idea never took off...
- ... but introduced a nice sandboxing feature

What happens next ?

- Security hole in 2009 in seccomp's code

History

And then

Seccomp integrated in linux kernel (2.6.12)

- Andrea's CPU sharing idea never took off...
- ... but introduced a nice sandboxing feature

What happens next ?

- Security hole in 2009 in seccomp's code
- No worries, but Linux Torvald asks if anybody actually uses seccomp

History

One prospective user : Google

History

One prospective user : Google

- The company wants to sandbox google chrome plugins

History

One prospective user : Google

- The company wants to sandbox google chrome plugins
- -> uses seccomp in order to do it

History

One prospective user : Google

- The company wants to sandbox google chrome plugins
- -> uses seccomp in order to do it

One main problem

History

One prospective user : Google

- The company wants to sandbox google chrome plugins
- -> uses seccomp in order to do it

One main problem

- Being limited to 4 syscalls is not handy...

History

One prospective user : Google

- The company wants to sandbox google chrome plugins
- -> uses seccomp in order to do it

One main problem

- Being limited to 4 syscalls is not handy...
- Introduce a monitor process

History

One prospective user : Google

- The company wants to sandbox google chrome plugins
- -> uses seccomp in order to do it

One main problem

- Being limited to 4 syscalls is not handy...
- Introduce a monitor process
- Still not very convenient...

History

2012, new seccomp feature : filter mode (Linux 3.5)

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore
- The programmer decides which syscall he forbids/allows for his program

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore
- The programmer decides which syscall he forbids/allows for his program
- Better !

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore
- The programmer decides which syscall he forbids/allows for his program
- Better !

Use of BPF programs

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore
- The programmer decides which syscall he forbids/allows for his program
- Better !

Use of BPF programs

- Berkely packet filter (BPF) is a small assembly-like program

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore
- The programmer decides which syscall he forbids/allows for his program
- Better !

Use of BPF programs

- Berkely packet filter (BPF) is a small assembly-like program
- It helps the programmer to clearly states what to allow

History

2012, new seccomp feature : filter mode (Linux 3.5)

- Not restricted to 4 syscalls anymore
- The programmer decides which syscall he forbids/allows for his program
- Better !

Use of BPF programs

- Berkely packet filter (BPF) is a small assembly-like program
- It helps the programmer to clearly states what to allow
- For instance, only allow open() in O_RDONLY mode

Seccomp strict mode

STRICT MODE

How to use it

Strict mode is enabled via prctl :

```
1 prctl(PR_SET_SECCOMP, SECCOMP_MODE_STRICT);
```

How to use it

Strict mode is enabled via prctl :

```
1 prctl(PR_SET_SECCOMP, SECCOMP_MODE_STRICT);
```

That's it !

Seccomp filter mode

FILTER MODE

BPF program

Berkeley packet filter (BPF)

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump
- Allow packets to be filtered in the kernel (more efficient than userland)

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump
- Allow packets to be filtered in the kernel (more efficient than userland)
- BPF now supports syscall filtering ! (well, since 2012)

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump
- Allow packets to be filtered in the kernel (more efficient than userland)
- BPF now supports syscall filtering ! (well, since 2012)

BPF characteristics

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump
- Allow packets to be filtered in the kernel (more efficient than userland)
- BPF now supports syscall filtering ! (well, since 2012)

BPF characteristics

- Small instruction set (instructions of same size)

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump
- Allow packets to be filtered in the kernel (more efficient than userland)
- BPF now supports syscall filtering ! (well, since 2012)

BPF characteristics

- Small instruction set (instructions of same size)
- Only branch-forward (no loop)

BPF program

Berkeley packet filter (BPF)

- Originally used for packet filtering with tcpdump
- Allow packets to be filtered in the kernel (more efficient than userland)
- BPF now supports syscall filtering ! (well, since 2012)

BPF characteristics

- Small instruction set (instructions of same size)
- Only branch-forward (no loop)
- One accumulator register

BPF program

An example of a BPF program :

BPF program

An example of a BPF program :

```
01001110101100011010001001010011101010001100101
10111011000011101100100100111010100011010010100011
00011000010001111111100101000101001010010011101010 1
01010010101010010101010101000111010101001010100010 11
00010001001001010101010101010101101000110010100011 10
10101001000111000101001000011001010101000100101001
01001000100010010110001010010010101001000100100101
```

BPF instruction

An instruction is a 64 bits number

BPF instruction

An instruction is a 64 bits number

```
struct sock_filter {    /* Filter block */
    __u16   code;       /* Actual filter code */
    __u8    jt;         /* Jump true */
    __u8    jf;         /* Jump false */
    __u32   k;          /* Generic multiuse field */
};
```

BPF program

BPF instruction set

BPF program

BPF instruction set

- Load instructions
- Store instructions
- Jump instructions
- Arithmetic instructions
- Return instructions

BPF program

BPF instruction set

- Load instructions
- Store instructions
- Jump instructions
- Arithmetic instructions
- Return instructions

Data section the program has access to :

BPF program

BPF instruction set

- Load instructions
- Store instructions
- Jump instructions
- Arithmetic instructions
- Return instructions

Data section the program has access to :

```
struct seccomp_data {
    int nr;
    __u32 arch;
    __u64 instruction_pointer;
    __u64 args[6];
};
```

BPF program

Several macros help the programmer code BPF programs

BPF program

Several macros help the programmer code BPF programs

```
#define BPF_JUMP(code, k, jt, jf) { (unsigned short)(code),
                                    jt, jf, k }
```


```
#define BPF_JUMP(code, k, jt, jf) { (unsigned short)(code),
                                    jt, jf, k }
```

BPF program

Example of some BPF instructions

BPF program

Example of some BPF instructions

```
1 BPF_STMT(BPF_LD | BPF_W | BPF_ABS,
          (offsetof(struct seccomp_data, arch)))
3 BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, AUDIT_ARCH_X86_64, 1, 0)
  BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL)
5 /* ... */
```

Libseccomp API

LIBSECCOMP

Libseccomp API

Make seccomp simplier to use

- No need to code BPF program by hand anymore

Libseccomp API

Make seccomp simplier to use

- No need to code BPF program by hand anymore
- Architecture independent

Libseccomp API

Make seccomp simplier to use

- No need to code BPF program by hand anymore
- Architecture independent

How it works

Libseccomp API

Make seccomp simplier to use

- No need to code BPF program by hand anymore
- Architecture independent

How it works

- Set a default behaviour (whitelist/blacklist)

Libseccomp API

Make seccomp simplier to use

- No need to code BPF program by hand anymore
- Architecture independent

How it works

- Set a default behaviour (whitelist/blacklist)
- Allow/forbid syscall based on name (architecture independent)

Libseccomp API

Make seccomp simplier to use

- No need to code BPF program by hand anymore
- Architecture independent

How it works

- Set a default behaviour (whitelist/blacklist)
- Allow/forbid syscall based on name (architecture independent)
- Basic filtering on parameters can be used

How to use it

Set default behaviour

How to use it

Set default behaviour

```
ctx = seccomp_init(SCMP_ACT_KILL);
```

How to use it

Set default behaviour

```
ctx = seccomp_init(SCMP_ACT_KILL);
```

Add some rules

How to use it

Set default behaviour

```
ctx = seccomp_init(SCMP_ACT_KILL);
```

Add some rules

```
1 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
```

How to use it

Set default behaviour

```
ctx = seccomp_init(SCMP_ACT_KILL);
```

Add some rules

```
1 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
```

Load filters in kernel memory

How to use it

Set default behaviour

```
ctx = seccomp_init(SCMP_ACT_KILL);
```

Add some rules

```
1 seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
  seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
```

Load filters in kernel memory

```
  seccomp_load(ctx);
2 seccomp_release(ctx);
```

Sources

- Michael Kerrisk slides : http ://man7.org/conf/lpc2015/limiting_kernel_attack_surface_with_seccomp-LPC_2015-Kerrisk.pdf
- Seccomp and sandboxing (2009) : https ://lwn.net/Articles/332974/
- A seccomp overview (2015) : https ://lwn.net/Articles/656307/
- Libseccomp : https ://github.com/seccomp/libseccomp

Questions

Questions ?