



Sandboxing in Linux with Zero Lines of Code

Ignat Korchagin

@ignatkn

\$ whoami

- Performance and security at Cloudflare
- Passionate about security and crypto
- Enjoy low level programming

Linux system calls and seccomp

Modern operating systems

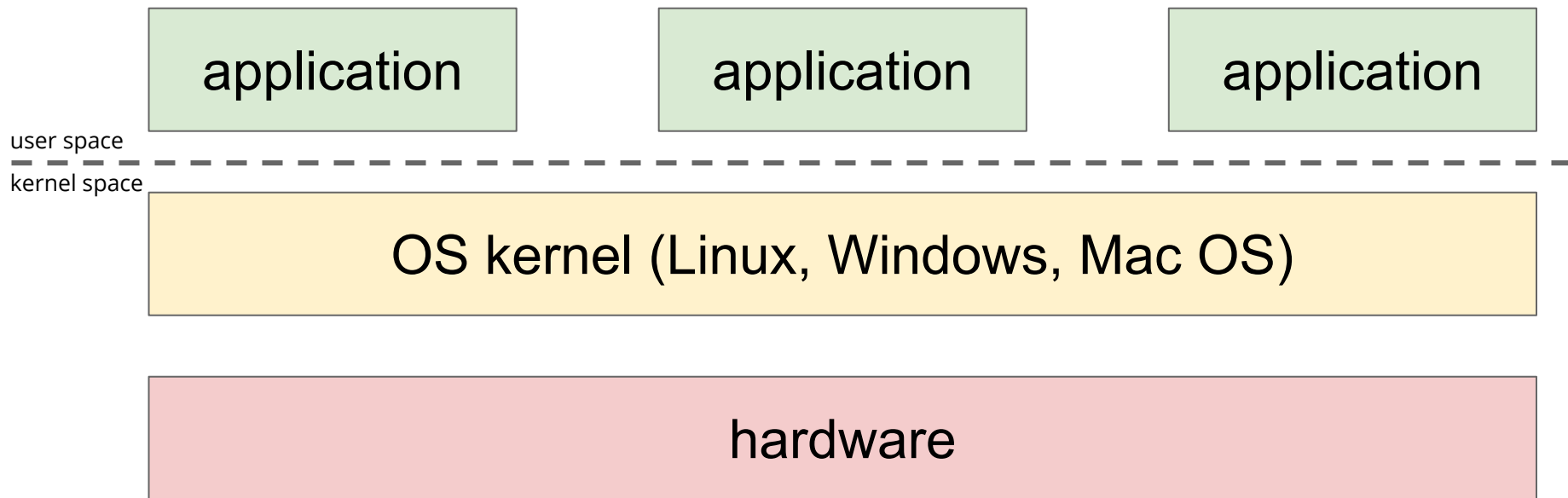
hardware

Modern operating systems

OS kernel (Linux, Windows, Mac OS)

hardware

Modern operating systems



System calls

application

System calls

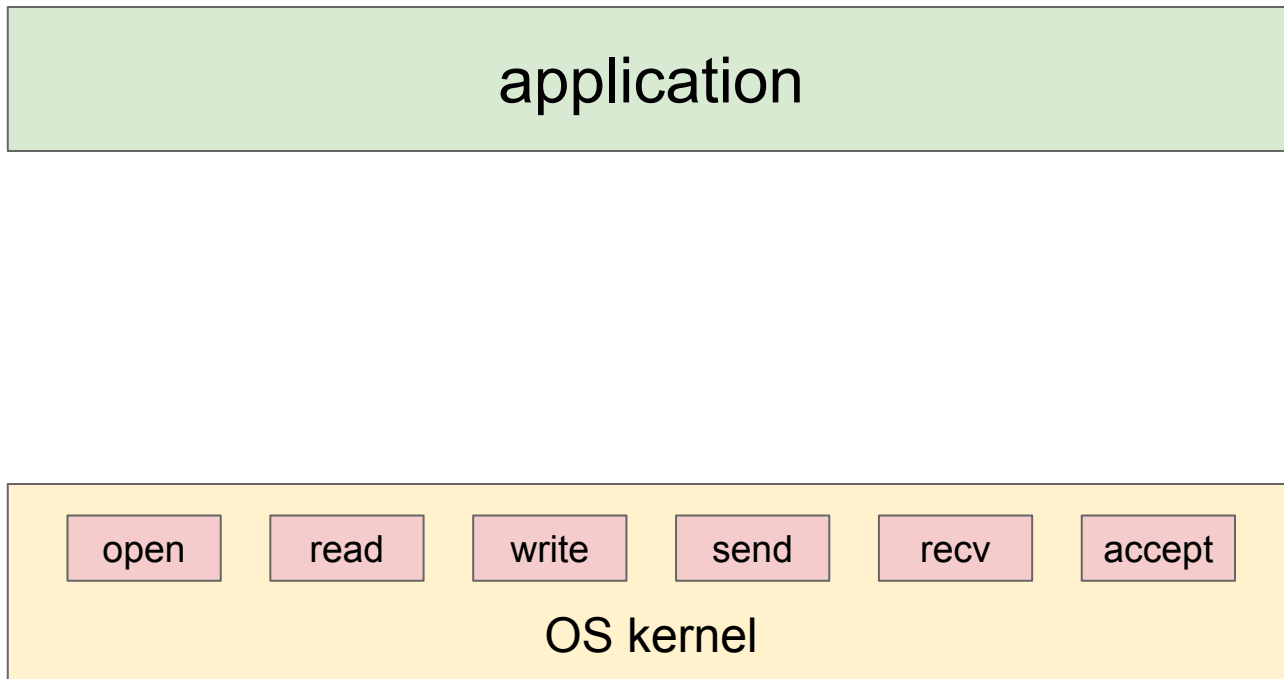


application

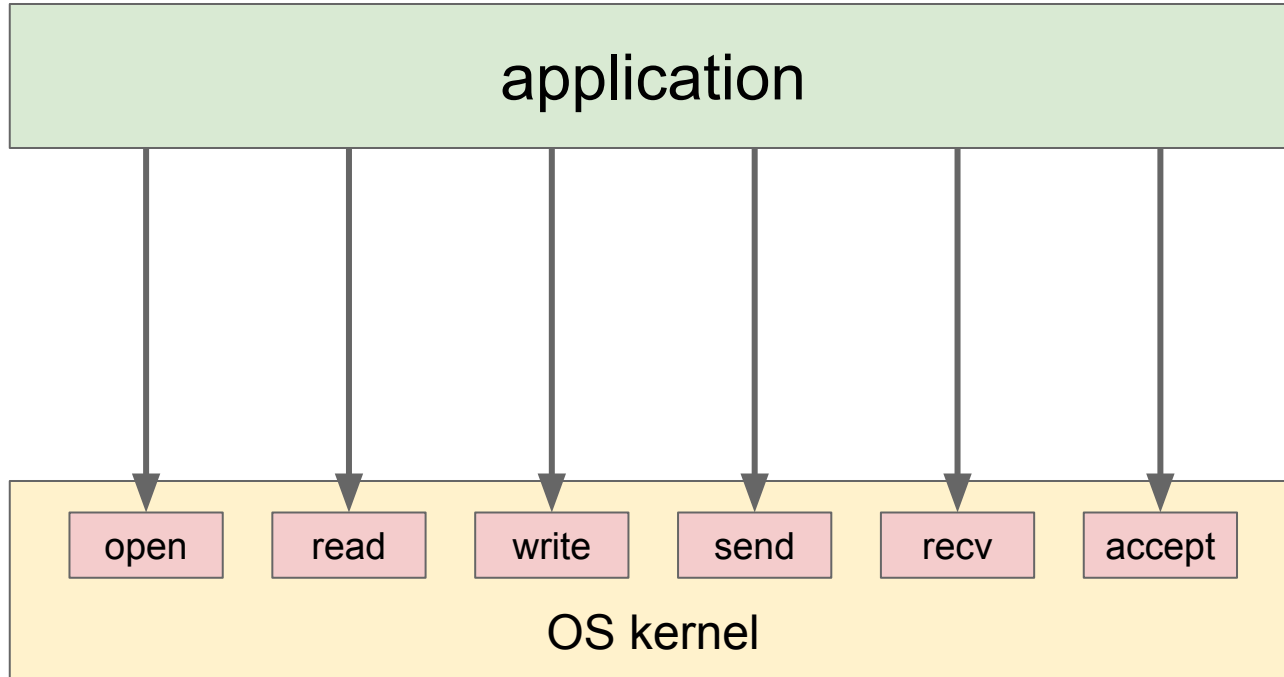
The diagram consists of two horizontal rectangular boxes. The top box is light green and contains the text 'application'. The bottom box is light yellow and contains the text 'OS kernel'. The boxes are separated by a white space, representing the interaction between user space and kernel space.

OS kernel

System calls



System calls



System calls

- a well-defined interface between (user space) applications and the OS kernel

System calls

- a well-defined interface between (user space) applications and the OS kernel
 - hardware-independent abstractions
 - basic OS services
 - file I/O
 - network access
 - time, sound etc

System calls

- a well-defined interface between (user space) applications and the OS kernel
 - hardware-independent abstractions
 - basic OS services
 - file I/O
 - network access
 - time, sound etc
- OS kernel enforces ACLs and permissions

System calls

- a well-defined interface between (user space) applications and the OS kernel
 - hardware-independent abstractions
 - basic OS services
 - file I/O
 - network access
 - time, sound etc
- OS kernel enforces ACLs and permissions
- resource management

seccomp

- yet another system call

seccomp

- yet another system call
- provides a way for application to notify the kernel which other system calls it intends to use
 - and which system calls it will definitely not use

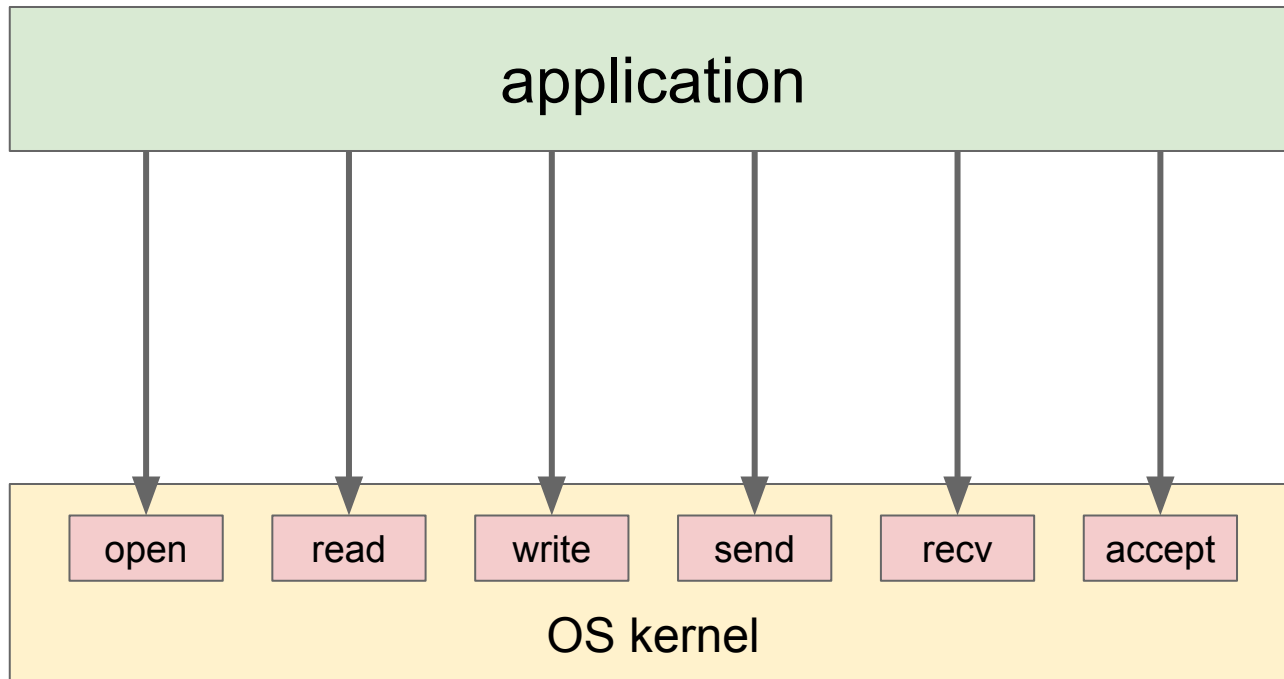
seccomp

- yet another system call
- provides a way for application to notify the kernel which other system calls it intends to use
 - and which system calls it will definitely not use
- the kernel enforces the system call policy provided by the application

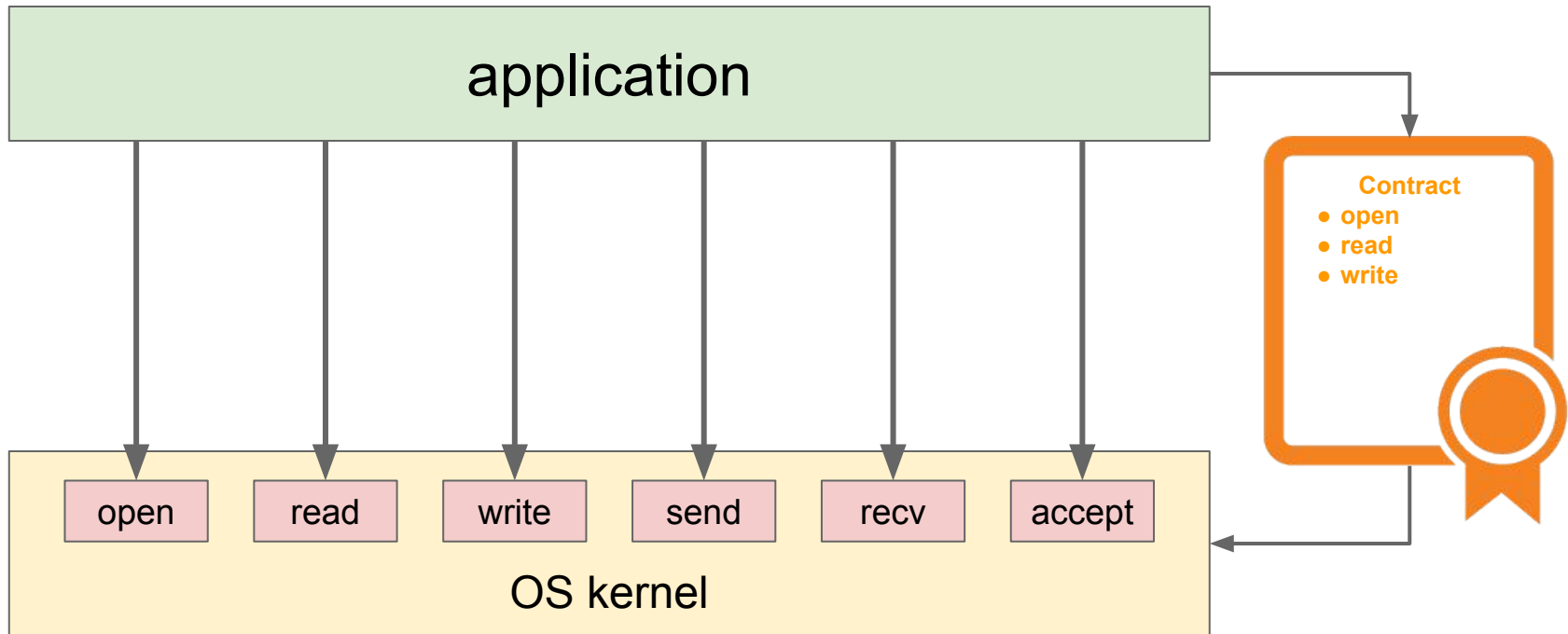
seccomp

- yet another system call
- provides a way for application to notify the kernel which other system calls it intends to use
 - and which system calls it will definitely not use
- the kernel enforces the system call policy provided by the application
- a tool an application can use to confine/sandbox itself

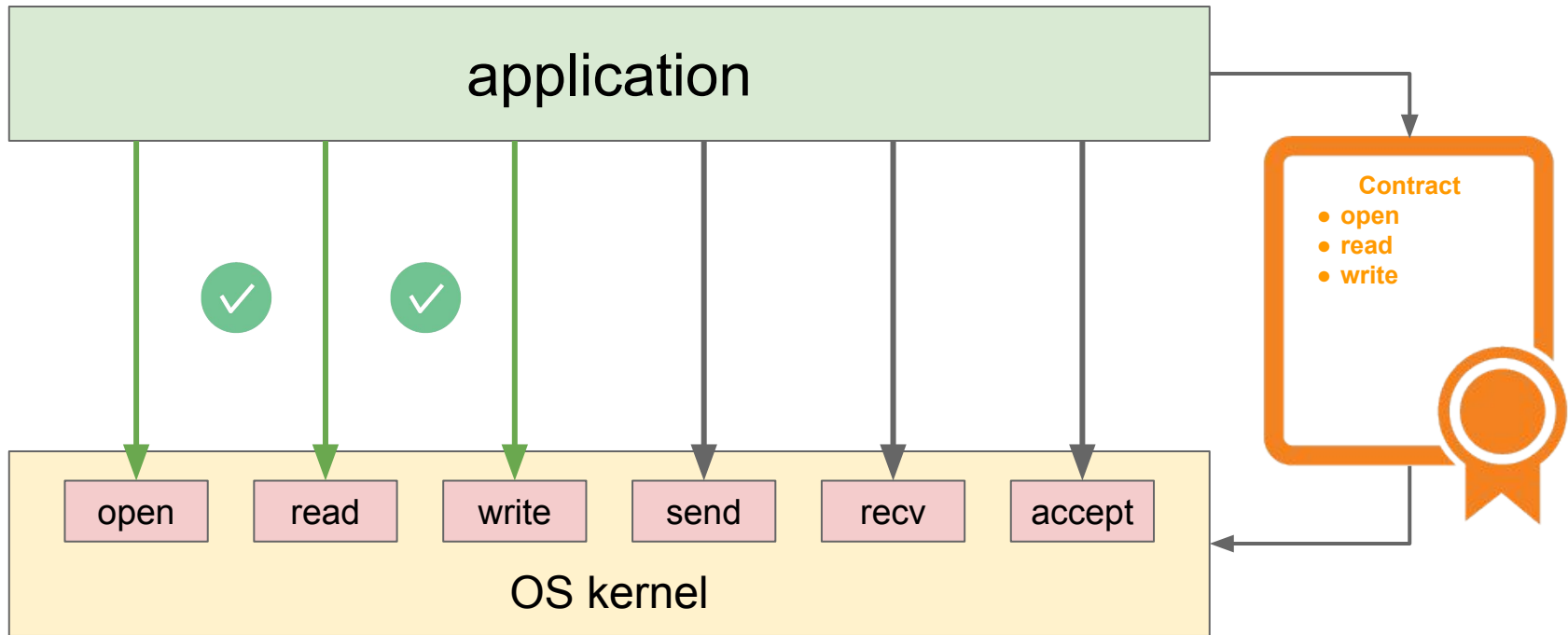
seccomp



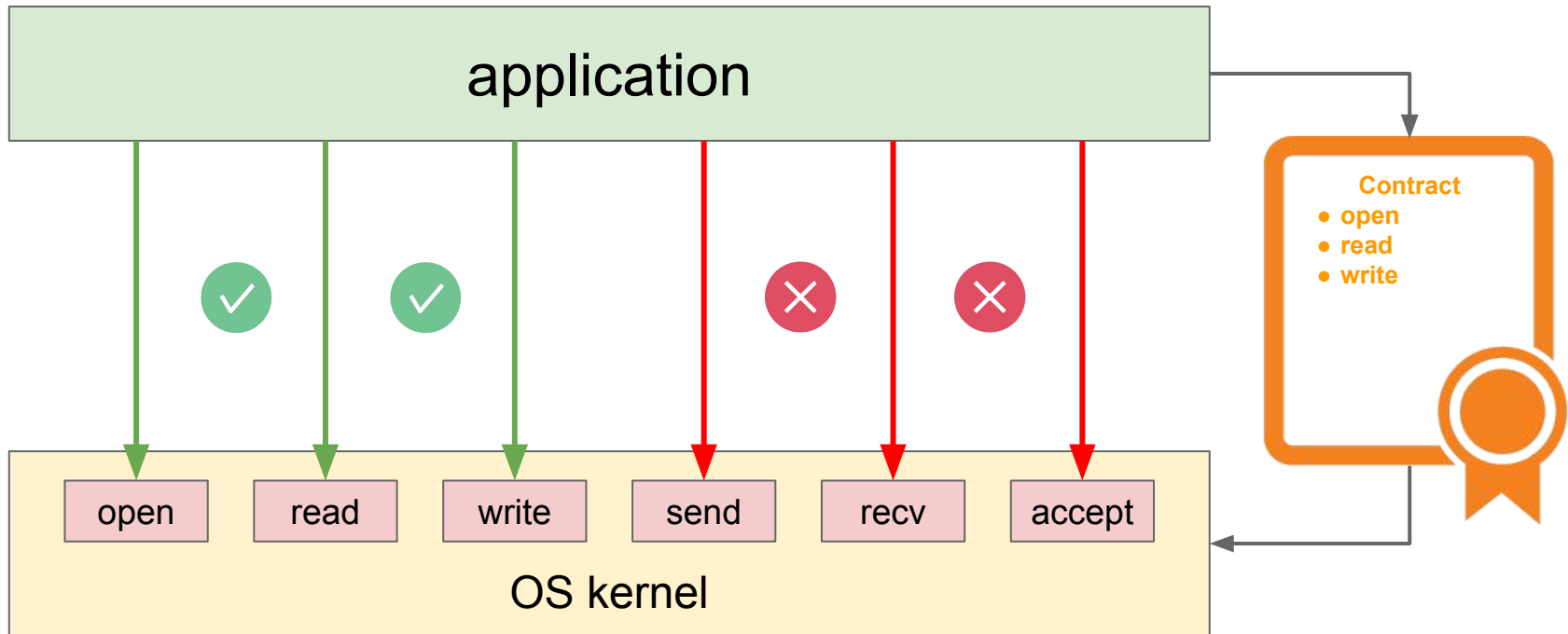
seccomp



seccomp



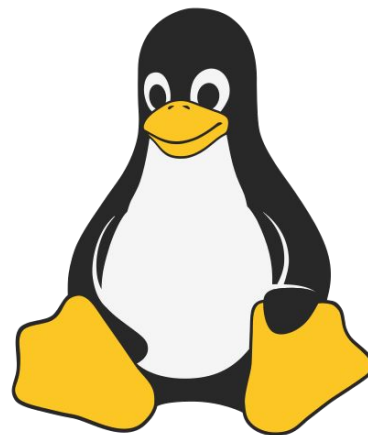
seccomp



seccomp example

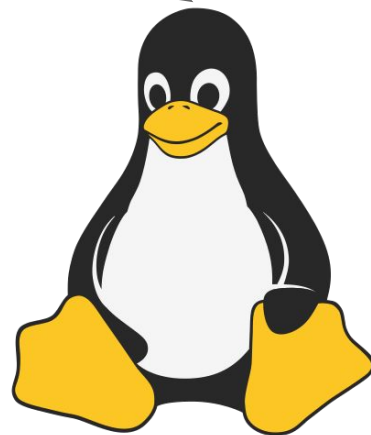


seccomp example



seccomp example

Hi! I'm a clock app. I will only use `gettimeofday`

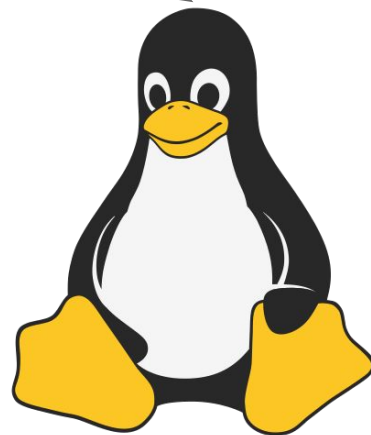


seccomp example

Hi! I'm a clock app. I will only use `gettimeofday`

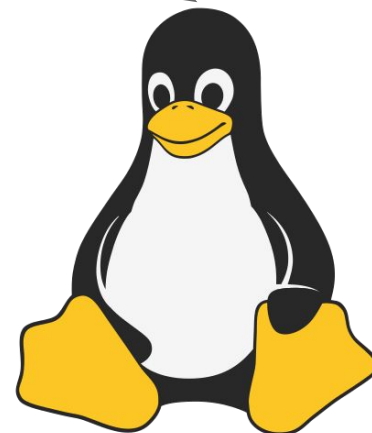
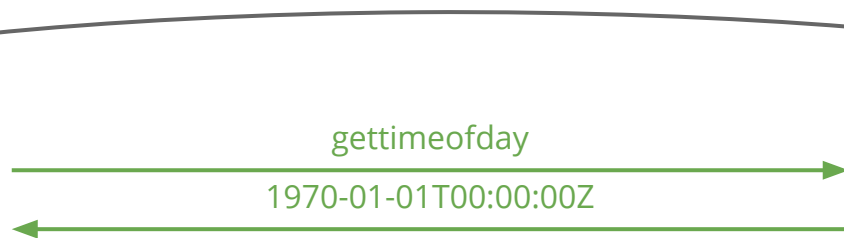


`gettimeofday`

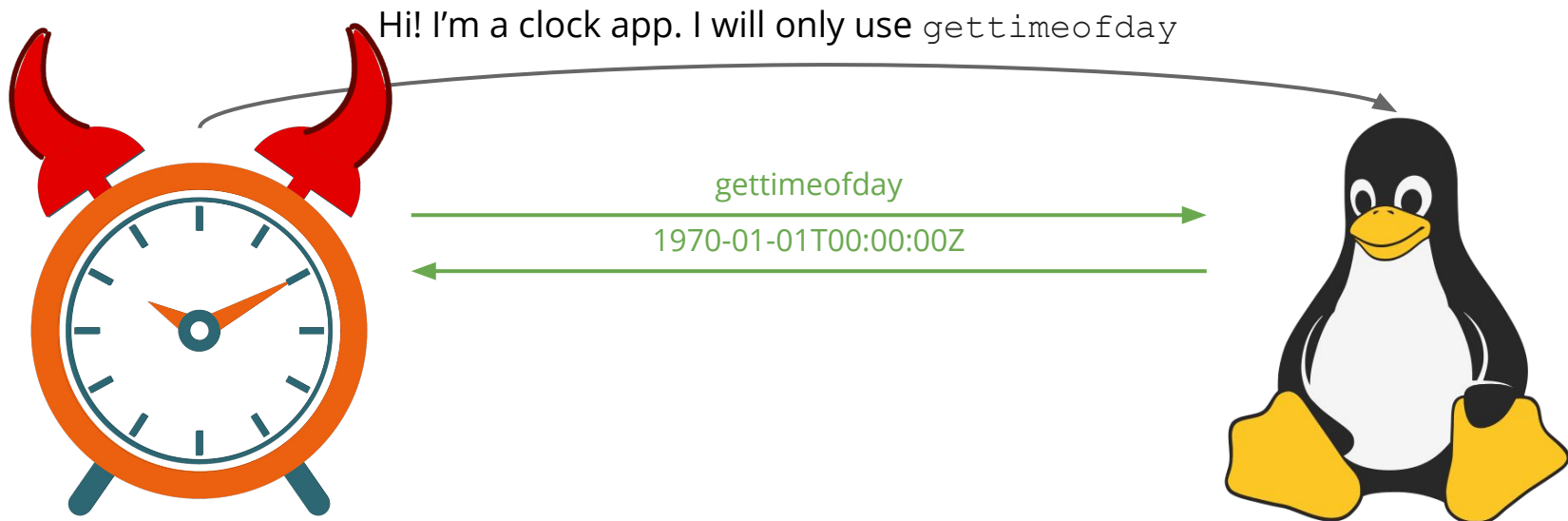


seccomp example

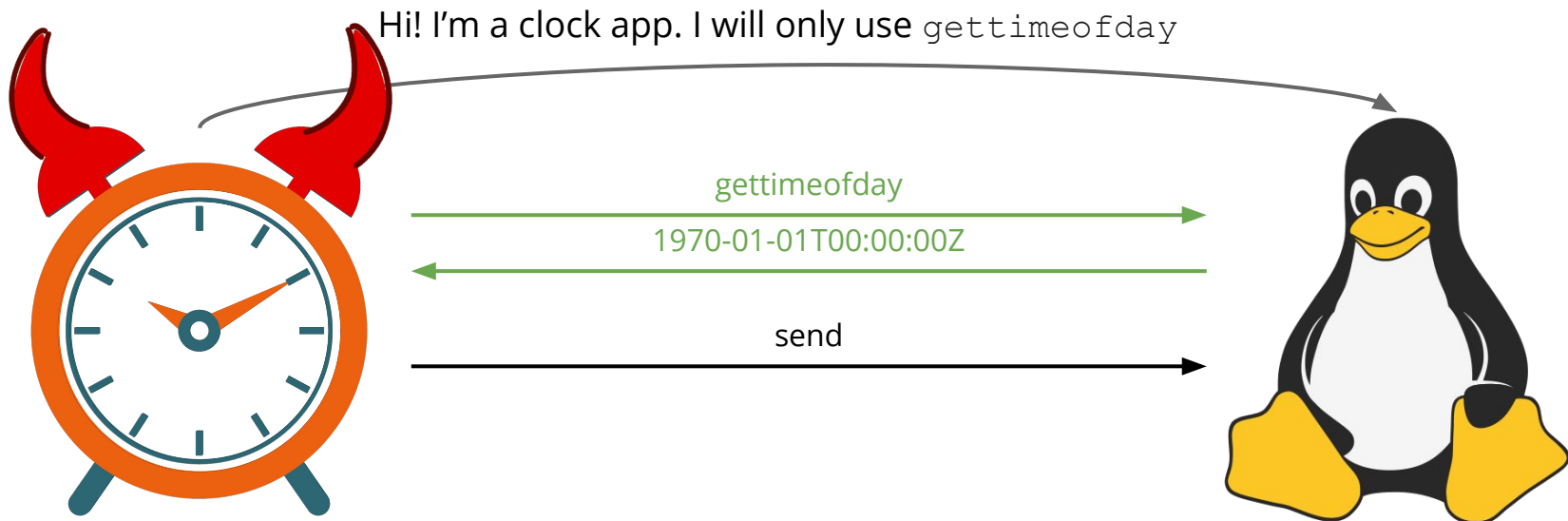
Hi! I'm a clock app. I will only use `gettimeofday`



seccomp example



seccomp example



seccomp example



Using seccomp

A simple “uname-like” toy tool

```
#include <stdio.h>
#include <sys/utsname.h>

int main(void)
{
    struct utsname name;
    if (uname(&name)) {
        perror("uname failed");
        return 1;
    }
    printf("My OS is %s!\n", name.sysname);
    return 0;
}
```


A simple “uname-like” toy tool

```
#include <stdio.h>
#include <sys/utsname.h>

int main(void)
{
    struct utsname name;
    if (uname(&name)) {
        perror("uname failed");
        return 1;
    }
    printf("My OS is %s!\n", name.sysname);
    return 0;
}
```

A simple “uname-like” toy tool

```
$ gcc -o myos myos.c
```

A simple “uname-like” toy tool

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

Sandboxing "myos"

```
#include <stdio.h>
#include <sys/utsname.h>
int main(void)
{
    struct utsname name;

    if (uname(&name)) {
        perror("uname failed");
        return 1;
    }
    printf("My OS is %s!\n", name.sysname);
    return 0;
}
```

Sandboxing "myos"

```
#include <stdio.h>
#include <sys/utsname.h>
int main(void)
{
    struct utsname name;
    sandbox();
    if (uname(&name)) {
        perror("uname failed");
        return 1;
    }
    printf("My OS is %s!\n", name.sysname);
    return 0;
}
```

Sandboxing "myos"

```
static void sandbox(void)
{
    struct sock_filter filter[] = {
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, arch))),
        /* if not x86_64, tell the kernel to kill the process
*/
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, AUDIT_ARCH_X86_64,
0, 4),
        /* get the actual syscall number */
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, nr))),
        /* if "uname", tell the kernel to return ENETDOWN,
otherwise just allow */
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, SYS_uname, 0, 1),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO | (ENETDOWN
& SECCOMP_RET_DATA)),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL),
    };
};
```

```
    struct sock_fprog prog = {
        .len = (unsigned short) (sizeof(filter) /
sizeof(filter[0])),
        .filter = filter,
    };
    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
        perror("PR_SET_NO_NEW_PRIVS failed");
        exit(1);
    };
    /* glibc does not have a wrapper for seccomp(2) */
    /* invoke it via the generic syscall wrapper */
    if (syscall(SYS_seccomp, SECCOMP_SET_MODE_FILTER, 0,
&prog)) {
        perror("seccomp failed");
        exit(1);
    };
}
```

Sandboxing "myos"

```
static void sandbox(void)
{
    struct sock_filter filter[] = {
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, arch))),
        /* if not x86_64, tell the kernel to kill the process
*/
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, AUDIT_ARCH_X86_64,
0, 4),
        /* get the actual syscall number */
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, nr))),
        /* if "uname", tell the kernel to return ENETDOWN,
otherwise just allow */
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, SYS_uname, 0, 1),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO | (ENETDOWN
& SECCOMP_RET_DATA)),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL),
    };
};
```

```
    struct sock_fprog prog = {
        .len = (unsigned short) (sizeof(filter) /
sizeof(filter[0])),
        .filter = filter,
    };
    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
        perror("PR_SET_NO_NEW_PRIVS failed");
        exit(1);
    };
    /* glibc does not have a wrapper for seccomp(2) */
    /* invoke it via the generic syscall wrapper */
    if (syscall(SYS_seccomp, SECCOMP_SET_MODE_FILTER, 0,
&prog)) {
        perror("seccomp failed");
        exit(1);
    };
}
```

Sandboxing "myos"

```
static void sandbox(void)
{
    struct sock_filter filter[] = {
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, arch))),
        /* if not x86_64, tell the kernel to kill the process
*/
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, AUDIT_ARCH_X86_64,
0, 4),
        /* get the actual syscall number */
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, nr))),
        /* if "uname", tell the kernel to return ENETDOWN,
otherwise just allow */
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, SYS_uname, 0, 1),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO | (ENETDOWN
& SECCOMP_RET_DATA)),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL),
    };
};
```

```
    struct sock_fprog prog = {
        .len = (unsigned short) (sizeof(filter) /
sizeof(filter[0])),
        .filter = filter,
    };
    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
        perror("PR_SET_NO_NEW_PRIVS failed");
        exit(1);
    };
    /* glibc does not have a wrapper for seccomp(2) */
    /* invoke it via the generic syscall wrapper */
    if (syscall(SYS_seccomp, SECCOMP_SET_MODE_FILTER, 0,
&prog)) {
        perror("seccomp failed");
        exit(1);
    };
};
```


Sandboxing "myos"

```
static void sandbox(void)
{
    struct sock_filter filter[] = {
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, arch))),
        /* if not x86_64, tell the kernel to kill the process
*/
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, AUDIT_ARCH_X86_64,
0, 4),
        /* get the actual syscall number */
        BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct
seccomp_data, nr))),
        /* if "uname", tell the kernel to return ENETDOWN,
otherwise just allow */
        BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K, SYS_uname, 0, 1),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ERRNO | (ENETDOWN
& SECCOMP_RET_DATA)),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
        BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL),
    };
};
```

```
    struct sock_fprog prog = {
        .len = (unsigned short) (sizeof(filter) /
sizeof(filter[0])),
        .filter = filter,
    };
    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
        perror("PR_SET_NO_NEW_PRIVS failed");
        exit(1);
    };
    /* glibc does not have a wrapper for seccomp(2) */
    /* invoke it via the generic syscall wrapper */
    if (syscall(SYS_seccomp, SECCOMP_SET_MODE_FILTER, 0,
&prog)) {
        perror("seccomp failed");
        exit(1);
    };
};
```

Sandboxing "myos"

```
$ gcc -o myos myos_raw_seccomp.c
```

Sandboxing "myos"

```
$ gcc -o myos myos_raw_seccomp.c  
$ ./myos  
uname failed: Network is down
```

Sandboxing "myos"

```
#include <stdio.h>
#include <sys/utsname.h>
int main(void)
{
    struct utsname name;
    sandbox();
    if (uname(&name)) {
        perror("uname failed");
        return 1;
    }
    printf("My OS is %s!\n", name.sysname);
    return 0;
}
```

Sandboxing "myos"

```
#include <stdio.h>
#include <sys/utsname.h>
int main(void)
{
    struct utsname name;
    sandbox();
    if (uname(&name)) {
        perror("uname failed");
        return 1;
    }
    printf("My OS is %s!\n", name.sysname);
    return 0;
}
```

Writing seccomp by hand

- low level assembly language
 - hard to write
 - hard to review
 - hard to debug
 - hard to update

Writing seccomp by hand

- low level assembly language
 - hard to write
 - hard to review
 - hard to debug
 - hard to update
- need to be aware of low-level details
 - system call numbers
 - architecture
 - some quirks, like setting `no_new_privs` bit

Sandboxing "myos" with libseccomp

```
static void sandbox(void)
{
    /* allow all syscalls by default */
    scmp_filter_ctx seccomp_ctx =
seccomp_init(SCMP_ACT_ALLOW);
    if (!seccomp_ctx)
        err(1, "seccomp_init failed");

    /* kill the process, if it tries to use
"uname" syscall */
    if (seccomp_rule_add_exact(seccomp_ctx,
SCMP_ACT_KILL,
seccomp_syscall_resolve_name("uname"), 0)) {
        perror("seccomp_rule_add_exact
failed");
        exit(1);
    }
}
```

```
/* apply the composed filter */
if (seccomp_load(seccomp_ctx)) {
    perror("seccomp_load failed");
    exit(1);
}

/* release allocated context */
seccomp_release(seccomp_ctx);
}
```


Sandboxing "myos" with libseccomp

```
static void sandbox(void)
{
    /* allow all syscalls by default */
    scmp_filter_ctx seccomp_ctx =
seccomp_init(SCMP_ACT_ALLOW);
    if (!seccomp_ctx)
        err(1, "seccomp_init failed");
    /* kill the process, if it tries to use
"uname" syscall */
    if (seccomp_rule_add_exact(seccomp_ctx,
SCMP_ACT_KILL,
seccomp_syscall_resolve_name("uname"), 0)) {
        perror("seccomp_rule_add_exact
failed");
        exit(1);
    }
}
```

```
/* apply the composed filter */
if (seccomp_load(seccomp_ctx)) {
    perror("seccomp_load failed");
    exit(1);
}

/* release allocated context */
seccomp_release(seccomp_ctx);
}
```

Sandboxing "myos" with libseccomp

```
static void sandbox(void)
{
    /* allow all syscalls by default */
    scmp_filter_ctx seccomp_ctx =
seccomp_init(SCMP_ACT_ALLOW);
    if (!seccomp_ctx)
        err(1, "seccomp_init failed");
    /* kill the process, if it tries to use
"uname" syscall */
    if (seccomp_rule_add_exact(seccomp_ctx,
SCMP_ACT_KILL,
seccomp_syscall_resolve_name("uname"), 0)) {
        perror("seccomp_rule_add_exact
failed");
        exit(1);
    }
}
```

```
/* apply the composed filter */
if (seccomp_load(seccomp_ctx)) {
    perror("seccomp_load failed");
    exit(1);
}

/* release allocated context */
seccomp_release(seccomp_ctx);
}
```

Sandboxing "myos" with libseccomp

```
static void sandbox(void)
{
    /* allow all syscalls by default */
    scmp_filter_ctx seccomp_ctx =
seccomp_init(SCMP_ACT_ALLOW);
    if (!seccomp_ctx)
        err(1, "seccomp_init failed");
    /* kill the process, if it tries to use
"uname" syscall */
    if (seccomp_rule_add_exact(seccomp_ctx,
SCMP_ACT_KILL,
seccomp_syscall_resolve_name("uname"), 0)) {
        perror("seccomp_rule_add_exact
failed");
        exit(1);
    }
}
```

```
/* apply the composed filter */
if (seccomp_load(seccomp_ctx)) {
    perror("seccomp_load failed");
    exit(1);
}

/* release allocated context */
seccomp_release(seccomp_ctx);
}
```

Sandboxing “myos” with libseccomp

```
$ gcc -o myos myos_libseccomp.c -lseccomp
```

Sandboxing “myos” with libseccomp

```
$ gcc -o myos myos_libseccomp.c -lseccomp
```

Sandboxing “myos” with libseccomp

```
$ gcc -o myos myos_libseccomp.c -lseccomp
$ ./myos
Bad system call
```

No code seccomp

Question?

How many developers start a project with a mindset:

Question?

How many developers start a project with a mindset:

“Most likely I will write bugs and security vulnerabilities. I should sandbox my code!”



Question?



Problems with seccomp

- seccomp applies the rules/policy to the current process
 - it is expected developers will sandbox their code themselves
 - no external interface to apply seccomp rules to a running process

Problems with seccomp

- seccomp applies the rules/policy to the current process
 - it is expected developers will sandbox their code themselves
 - no external interface to apply seccomp rules to a running process
- operators/sysadmins have almost no control over the seccomp policy

Problems with seccomp

- seccomp applies the rules/policy to the current process
 - it is expected developers will sandbox their code themselves
 - no external interface to apply seccomp rules to a running process
- operators/sysadmins have almost no control over the seccomp policy
- no way to sandbox third-party proprietary applications

Mind the gap!



Sandboxing with systemd

`SystemCallFilter=`

- Takes a space-separated list of system call names. If this setting is used, all system calls executed by the unit processes except for the listed ones will result in immediate process termination with the `SIGSYS` signal...

Sandboxing with systemd

`SystemCallFilter=`

- Takes a space-separated list of system call names. If this setting is used, all system calls executed by the unit processes except for the listed ones will result in immediate process termination with the `SIGSYS` signal...

<https://www.freedesktop.org/software/systemd/man/systemd.exec.html#SystemCallFilter=>

Sandboxing with systemd

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

Sandboxing with systemd

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec ./myos
```

```
Running as unit: run-u16.service
```

```
Press ^] three times within 1s to disconnect TTY.
```

```
My OS is Linux!
```

```
Finished with result: success
```

```
Main processes terminated with: code=exited/status=0
```

```
Service runtime: 3ms
```

Sandboxing with systemd

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec ./myos
```

```
Running as unit: run-u16.service
```

```
Press ^] three times within 1s to disconnect TTY.
```

```
My OS is Linux!
```

```
Finished with result: success
```

```
Main processes terminated with: code=exited/status=0
```

```
Service runtime: 3ms
```

Sandboxing with systemd

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" ./myos
```

Running as unit: run-u17.service

Press ^] three times within 1s to disconnect TTY.

Finished with result: signal

Main processes terminated with: code=killed/status=SYS

Service runtime: 6ms

Sandboxing with systemd

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" ./myos
```

Running as unit: run-u17.service

Press ^] three times within 1s to disconnect TTY.

Finished with result: signal

Main processes terminated with: code=killed/status=SYS

Service runtime: 6ms

Sandboxing with systemd

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" ./myos
```

Running as unit: run-u17.service

Press ^] three times within 1s to disconnect TTY.

Finished with result: signal

Main processes terminated with: **code=killed/status=SYS**

Service runtime: 6ms

Sandboxing with systemd

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" ./myos
```

Running as unit: run-u17.service

Press ^] three times within 1s to disconnect TTY.

Finished with result: signal

Main processes terminated with: code=killed/status=SYS

Service runtime: 6ms

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" --property="SystemCallErrorNumber=ENETDOWN" ./myos
```

Running as unit: run-u18.service

Press ^] three times within 1s to disconnect TTY.

uname failed: Network is down

Finished with result: exit-code

Main processes terminated with: code=exited/status=1

Service runtime: 6ms

Sandboxing with systemd

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" ./myos
```

Running as unit: run-u17.service

Press ^] three times within 1s to disconnect TTY.

Finished with result: signal

Main processes terminated with: code=killed/status=SYS

Service runtime: 6ms

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" --property="SystemCallErrorNumber=ENETDOWN" ./myos
```

Running as unit: run-u18.service

Press ^] three times within 1s to disconnect TTY.

uname failed: Network is down

Finished with result: exit-code

Main processes terminated with: code=exited/status=1

Service runtime: 6ms

Sandboxing with systemd

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" ./myos
```

Running as unit: run-u17.service

Press ^] three times within 1s to disconnect TTY.

Finished with result: signal

Main processes terminated with: code=killed/status=SYS

Service runtime: 6ms

```
$ systemd-run --user --pty --same-dir --wait --collect --service-type=exec  
--property="SystemCallFilter=~uname" --property="SystemCallErrorNumber=ENETDOWN" ./myos
```

Running as unit: run-u18.service

Press ^] three times within 1s to disconnect TTY.

uname failed: Network is down

Finished with result: exit-code

Main processes terminated with: code=exited/status=1

Service runtime: 6ms

Sandboxing with systemd

The small print:

Sandboxing with systemd

The small print:

“Note that the `execve`, `exit`, `exit_group`, `getrlimit`, `rt_sigreturn`, `sigreturn` system calls and the system calls for querying time and sleeping are implicitly whitelisted and do not need to be listed explicitly”

<https://www.freedesktop.org/software/systemd/man/systemd.exec.html#SystemCallFilter=>

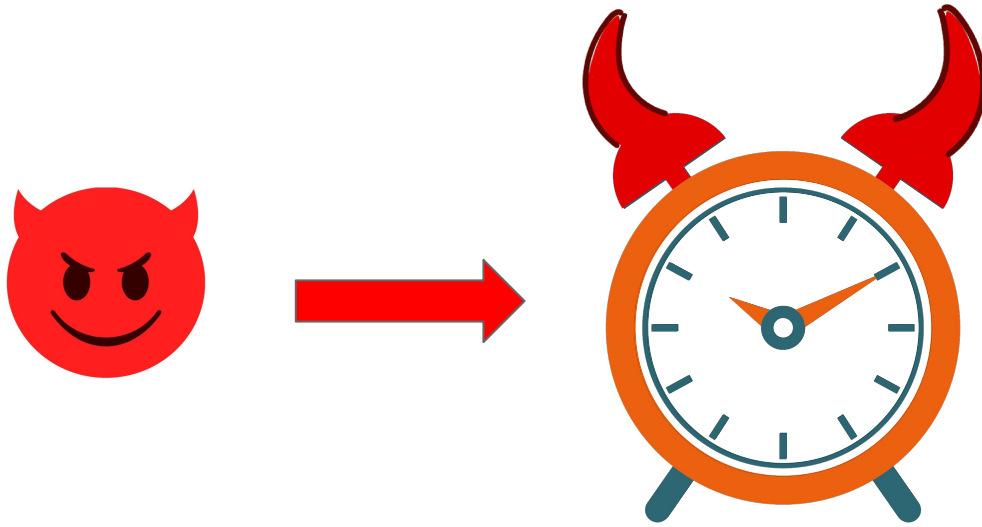
Why “execve” is bad?



Why “execve” is bad?



Why “execve” is bad?



Why “execve” is bad?



Cloudflare sandbox tool

A toolkit to inject seccomp rules into almost any process

Cloudflare sandbox tool

A toolkit to inject seccomp rules into almost any process

- follows the systemd approach one step further
 - a shared library for dynamically linked applications
 - a “launcher” for statically linked applications

Cloudflare sandbox tool

A toolkit to inject seccomp rules into almost any process

- follows the systemd approach one step further
 - a shared library for dynamically linked applications
 - a “launcher” for statically linked applications
- can block any system call, including `execve`

Cloudflare sandbox tool

A toolkit to inject seccomp rules into almost any process

- follows the systemd approach one step further
 - a shared library for dynamically linked applications
 - a “launcher” for statically linked applications
- can block any system call, including `execve`
- works on any binary written in any language, even proprietary ones

Cloudflare sandbox tool

A toolkit to inject seccomp rules into almost any process

- follows the systemd approach one step further
 - a shared library for dynamically linked applications
 - a “launcher” for statically linked applications
- can block any system call, including `execve`
- works on any binary written in any language, even proprietary ones

<https://github.com/cloudflare/sandbox>

Cloudflare sandbox tool



Cloudflare sandbox tool

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

Cloudflare sandbox tool

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so
```

```
SECCOMP_SYSCALL_DENY=uname ./myos
```

```
adding uname to the process seccomp filter
```

```
Bad system call
```

Cloudflare sandbox tool

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so  
SECCOMP_SYSCALL_DENY=uname ./myos
```

```
adding uname to the process seccomp filter
```

```
Bad system call
```


Cloudflare sandbox tool

```
$ gcc -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so  
SECCOMP_SYSCALL_DENY=uname ./myos
```

```
adding uname to the process seccomp filter
```

```
Bad system call
```

Cloudflare sandbox tool

```
$ patchelf --add-needed /usr/lib/x86_64-linux-gnu/libsandbox.so ./myos
```

Cloudflare sandbox tool

```
$ patchelf --add-needed /usr/lib/x86_64-linux-gnu/libsandbox.so ./myos
$ ldd ./myos
    linux-vdso.so.1 (0x00007ffc35dfb000)
    /usr/lib/x86_64-linux-gnu/libsandbox.so (0x00007f2f7c232000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2f7c065000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f2f7c287000)
```

Cloudflare sandbox tool

```
$ patchelf --add-needed /usr/lib/x86_64-linux-gnu/libsandbox.so ./myos
$ ldd ./myos
    linux-vdso.so.1 (0x00007ffc35dfb000)
    /usr/lib/x86_64-linux-gnu/libsandbox.so (0x00007f2f7c232000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2f7c065000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f2f7c287000)
```

Cloudflare sandbox tool

```
$ patchelf --add-needed /usr/lib/x86_64-linux-gnu/libsandbox.so ./myos
$ ldd ./myos
    linux-vdso.so.1 (0x00007ffc35dfb000)
    /usr/lib/x86_64-linux-gnu/libsandbox.so (0x00007f2f7c232000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2f7c065000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f2f7c287000)
$ SECCOMP_SYSCALL_DENY=uname ./myos
adding uname to the process seccomp filter
Bad system call
```

Cloudflare sandbox tool (static binaries)

```
$ gcc -static -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

Cloudflare sandbox tool (static binaries)

```
$ gcc -static -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

Cloudflare sandbox tool (static binaries)

```
$ gcc -static -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so  
SECCOMP_SYSCALL_DENY=uname ./myos
```

```
My OS is Linux!
```


Cloudflare sandbox tool (static binaries)

```
$ gcc -static -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so  
SECCOMP_SYSCALL_DENY=uname ./myos
```

```
My OS is Linux!
```

```
$ SECCOMP_SYSCALL_DENY=uname sandboxify ./myos
```

```
adding uname to the process seccomp filter
```

Cloudflare sandbox tool (static binaries)

```
$ gcc -static -o myos myos.c
```

```
$ ./myos
```

```
My OS is Linux!
```

```
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so  
SECCOMP_SYSCALL_DENY=uname ./myos
```

```
My OS is Linux!
```

```
$ SECCOMP_SYSCALL_DENY=uname sandboxify ./myos
```

```
adding uname to the process seccomp filter
```

Cloudflare sandbox tool

- configuration via environment variables

Cloudflare sandbox tool

- configuration via environment variables
 - `SECCOMP_SYSCALL_ALLOW`
 - anything not in the list is not allowed

Cloudflare sandbox tool

- configuration via environment variables
 - `SECCOMP_SYSCALL_ALLOW`
 - anything not in the list is not allowed
 - `SECCOMP_SYSCALL_DENY`
 - anything in the list is not allowed, everything else is

Cloudflare sandbox tool

- configuration via environment variables
 - SECCOMP_SYSCALL_ALLOW
 - anything not in the list is not allowed
 - SECCOMP_SYSCALL_DENY
 - anything in the list is not allowed, everything else is
 - SECCOMP_DEFAULT_ACTION
 - do not kill the process on violation, just log the system call

Cloudflare sandbox tool

libsandbox.so

Cloudflare sandbox tool

libsandbox.so

- dynamically linked executables only

Cloudflare sandbox tool

libsandbox.so

- dynamically linked executables only
- need to preload the library into the process
 - LD_PRELOAD
 - patch the ELF binary

Cloudflare sandbox tool

libsandbox.so

- dynamically linked executables only
- need to preload the library into the process
 - LD_PRELOAD
 - patch the ELF binary

sandboxify

Cloudflare sandbox tool

libsandbox.so

- dynamically linked executables only
- need to preload the library into the process
 - LD_PRELOAD
 - patch the ELF binary

sandboxify

- statically and dynamically linked executables

Cloudflare sandbox tool

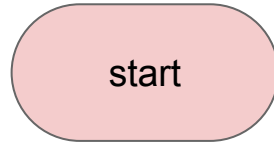
libsandbox.so

- dynamically linked executables only
- need to preload the library into the process
 - LD_PRELOAD
 - patch the ELF binary

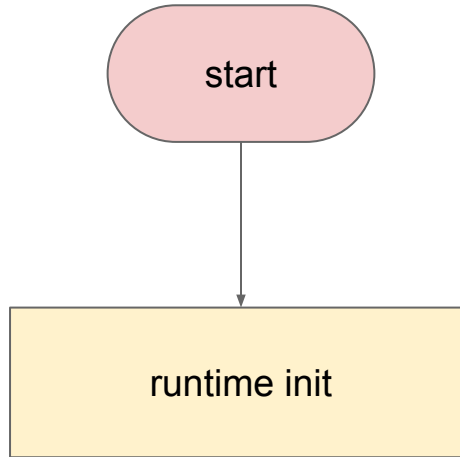
sandboxify

- statically and dynamically linked executables
- need to launch the executable through the launcher

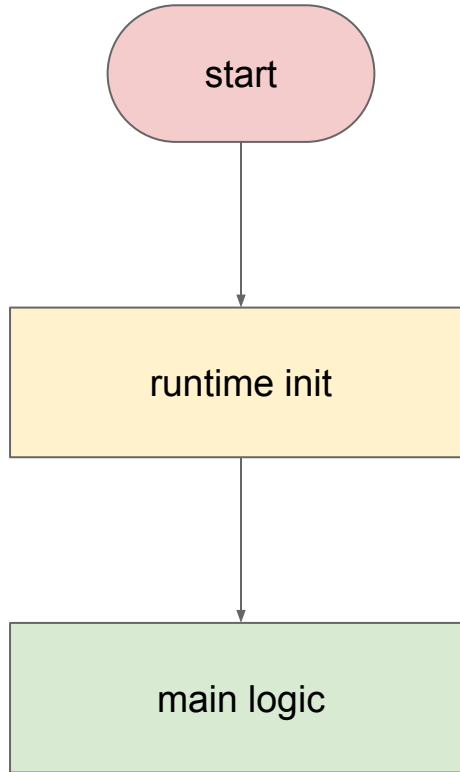
Process startup



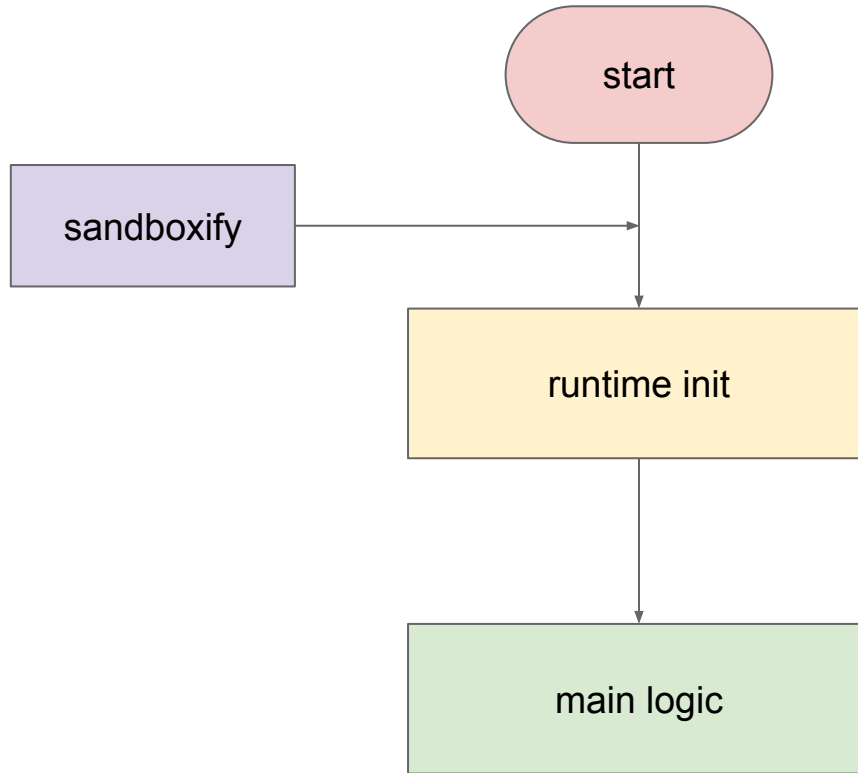
Process startup



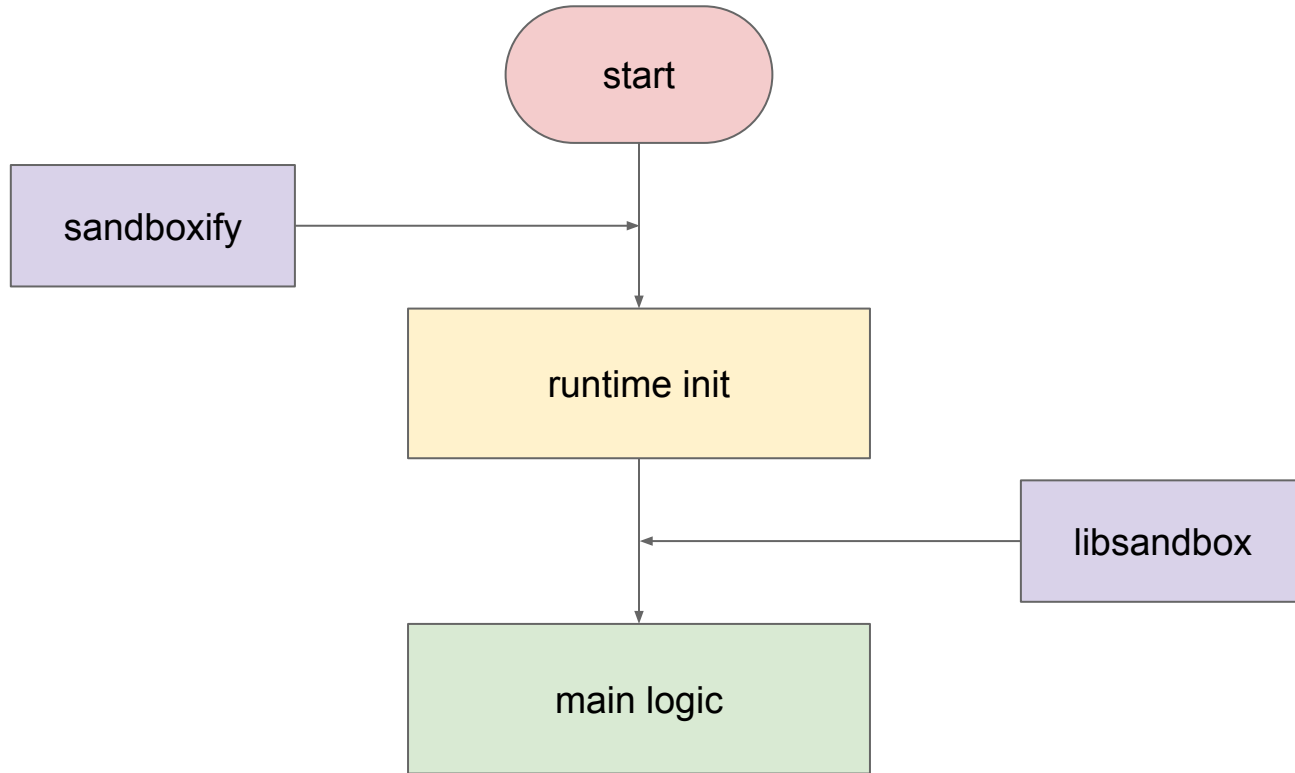
Process startup



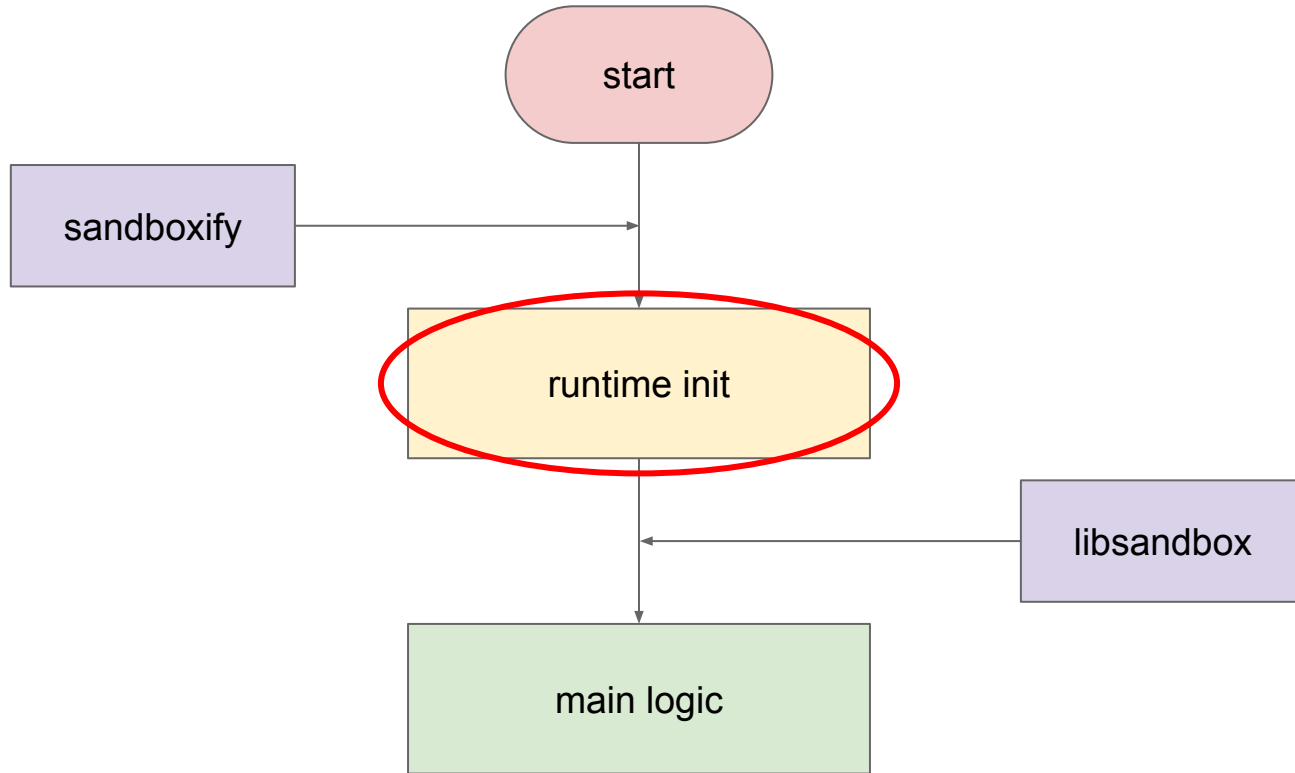
Process startup



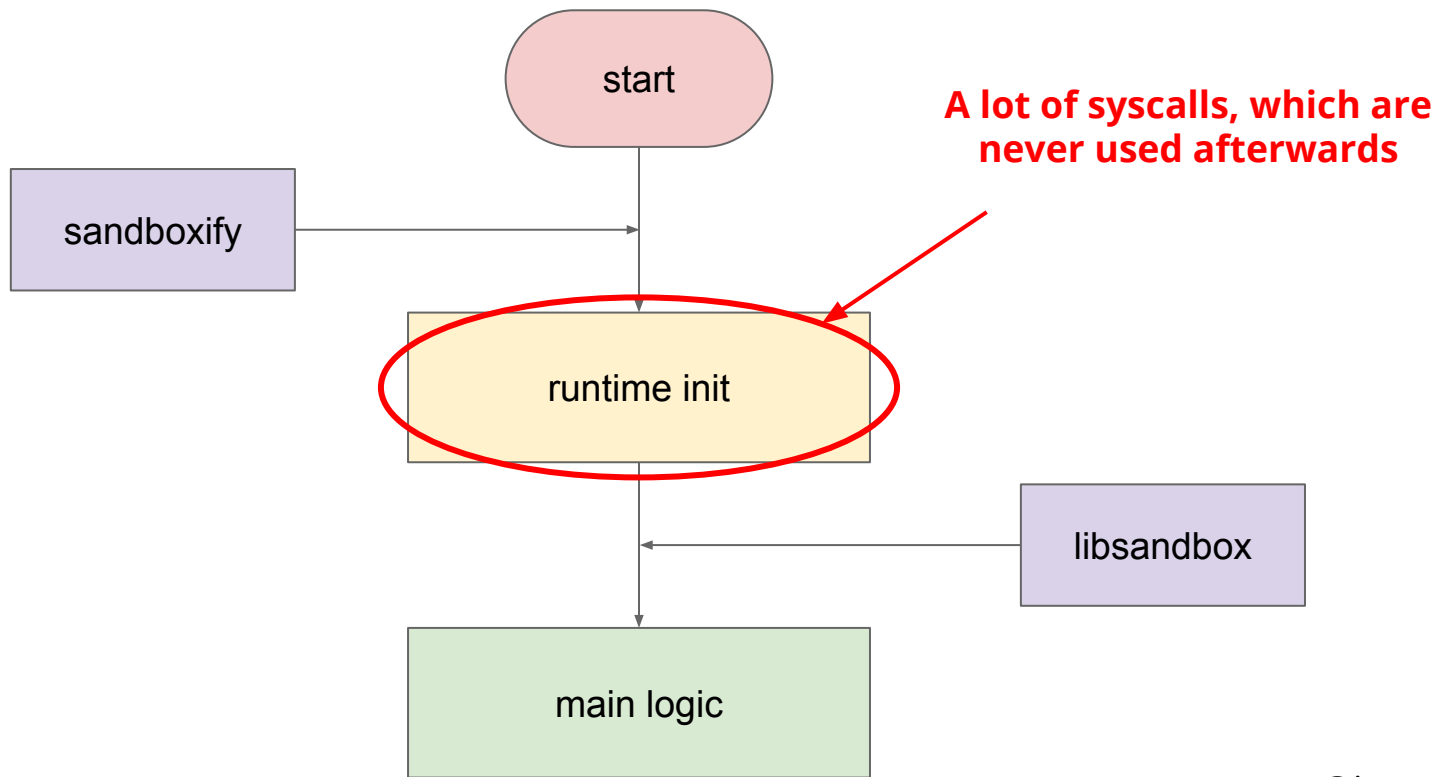
Process startup



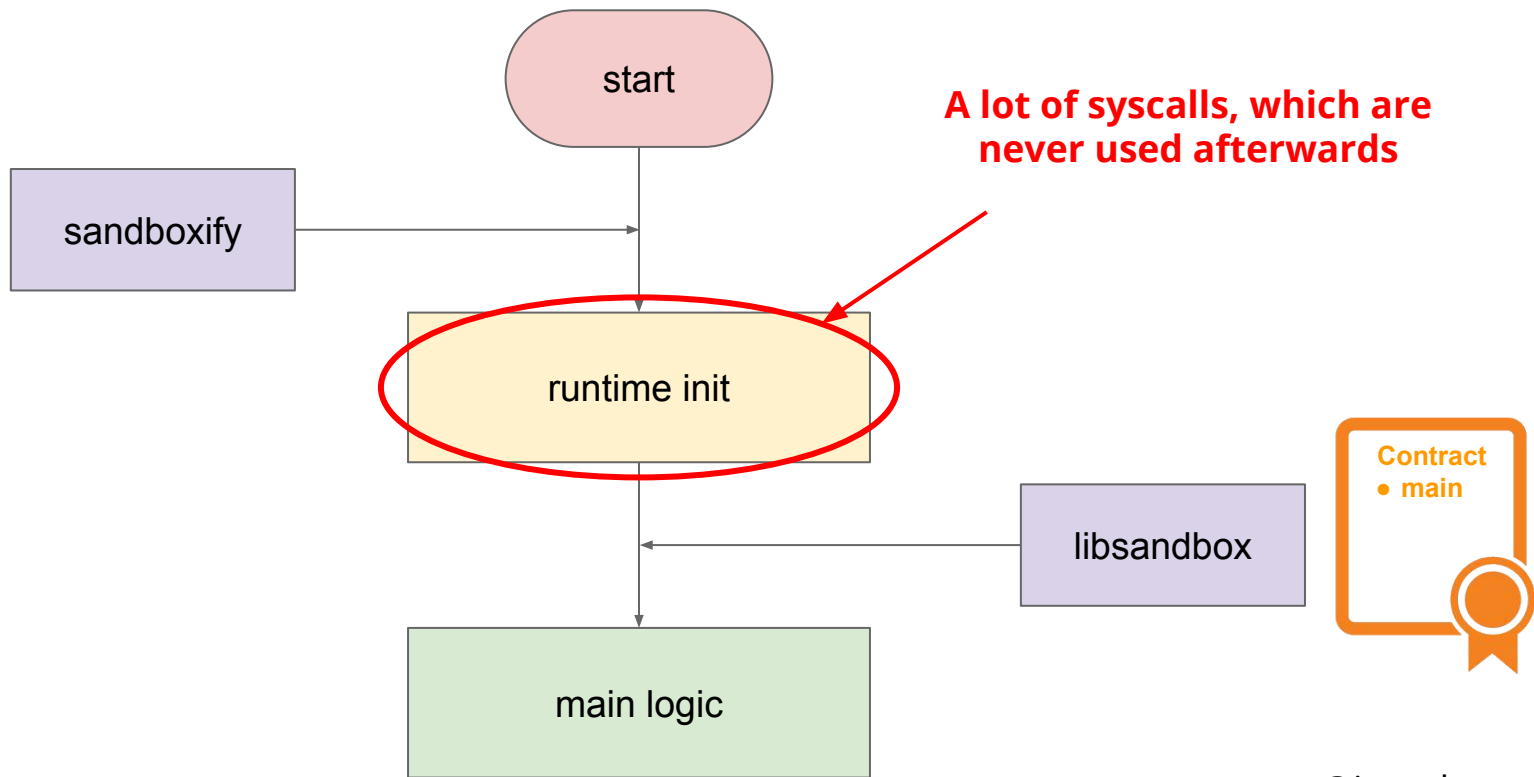
Process startup



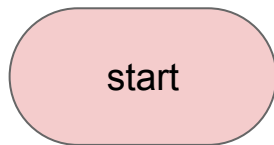
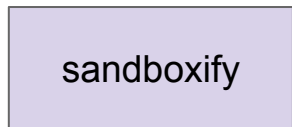
Process startup



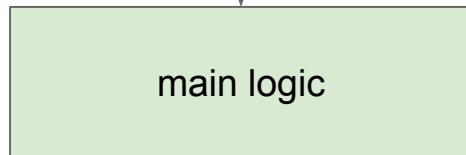
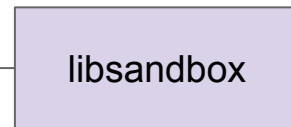
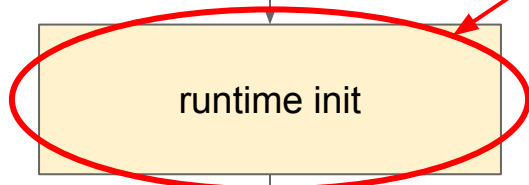
Process startup



Process startup



A lot of syscalls, which are never used afterwards



sandboxify vs libsandbox

```
$ gcc -o myos myos.c
```

sandboxify vs libsandbox

```
$ gcc -o myos myos.c
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so
SECCOMP_SYSCALL_ALLOW=exit_group:fstat:uname:write ./myos
adding exit_group to the process seccomp filter
adding fstat to the process seccomp filter
adding uname to the process seccomp filter
adding write to the process seccomp filter
My OS is Linux!
```

sandboxify vs libsandbox

```
$ gcc -o myos myos.c
$ LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libsandbox.so
SECCOMP_SYSCALL_ALLOW=exit_group:fstat:uname:write ./myos
adding exit_group to the process seccomp filter
adding fstat to the process seccomp filter
adding uname to the process seccomp filter
adding write to the process seccomp filter
My OS is Linux!
```


sandboxify vs libsandbox

```
$ SECCOMP_SYSCALL_ALLOW=access:arch_prctl:brk:close:exit_group:fstat:mmap:mprotect:munmap:openat:read:uname:write  
sandboxify ./myos
```

```
adding access to the process seccomp filter  
adding arch_prctl to the process seccomp filter  
adding brk to the process seccomp filter  
adding close to the process seccomp filter  
adding exit_group to the process seccomp filter  
adding fstat to the process seccomp filter  
adding mmap to the process seccomp filter  
adding mprotect to the process seccomp filter  
adding munmap to the process seccomp filter  
adding openat to the process seccomp filter  
adding read to the process seccomp filter  
adding uname to the process seccomp filter  
adding write to the process seccomp filter  
My OS is Linux!
```

sandboxify vs libsandbox

```
$ SECCOMP_SYSCALL_ALLOW=access:arch_prctl:brk:close:exit_group:fstat:mmap:mprotect:munmap:openat:read:uname:write  
sandboxify ./myos
```

```
adding access to the process seccomp filter
```

```
adding arch_prctl to the process seccomp filter
```

```
adding brk to the process seccomp filter
```

```
adding close to the process seccomp filter
```

```
adding exit_group to the process seccomp filter
```

```
adding fstat to the process seccomp filter
```

```
adding mmap to the process seccomp filter
```

```
adding mprotect to the process seccomp filter
```

```
adding munmap to the process seccomp filter
```

```
adding openat to the process seccomp filter
```

```
adding read to the process seccomp filter
```

```
adding uname to the process seccomp filter
```

```
adding write to the process seccomp filter
```

```
My OS is Linux!
```

sandboxify vs libsandbox

```
$ SECCOMP_SYSCALL_ALLOW=access:arch_prctl:brk:close:exit_group:fstat:mmap:mprotect:munmap:openat:read:uname:write  
sandboxify ./myos
```

```
adding access to the process seccomp filter  
adding arch_prctl to the process seccomp filter  
adding brk to the process seccomp filter  
adding close to the process seccomp filter  
adding exit_group to the process seccomp filter  
adding fstat to the process seccomp filter  
adding mmap to the process seccomp filter  
adding mprotect to the process seccomp filter  
adding munmap to the process seccomp filter  
adding openat to the process seccomp filter  
adding read to the process seccomp filter  
adding uname to the process seccomp filter  
adding write to the process seccomp filter  
My OS is Linux!
```

4 vs 13 syscalls
in the allowlist

Links

<https://www.man7.org/linux/man-pages/man2/seccomp.2.html>

<https://github.com/seccomp/libseccomp>

<https://www.freedesktop.org/software/systemd/man/systemd.exec.html#SystemCallFilter=>

<https://github.com/cloudflare/sandbox>

Questions?