least privilege

a typical program I run on my desktop is allowed to...

make network connections to anywhere

upload all my files to the Internet

delete all my files

record all my keystrokes

...

but it probably doesn't need to...

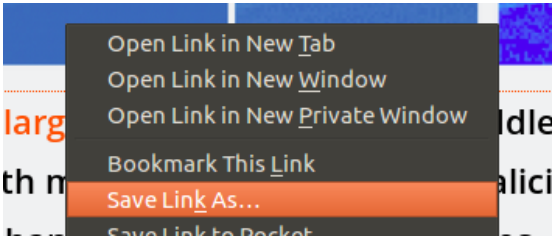
ideally: if typical program was compromised/malicious, it still wouldn't be able to do most of these things

things applications need?

what things should browser be able to do?

what things should word processor be able to do?

things broswers need



save files

have your webmail password

...

multi-user OSs

```
cr4bd@labunix01:~$ cp myprogram.exe /bin/ls
cp: cannot create regular file ‘/bin/’ls: Permission denied
```

programs have *limited privileges*

permission enforcement

```
struct Process {
    int user_id;
    ...
};
int handle_open_system_call(char *filename, ...) {
    Process* currentProcess = GetCurrentProcess();
    File* file = GetFileByFilename(filename);
    if (!file->UserCanAccess(currentProcess->user_id)) {
        return ERROR_PERMISSION_DENIED;
    }
    ...
}
```

multi-user OSs

```
cr4bd@labunix01:~$ cp myprogram.exe /bin/ls
cp: cannot create regular file ‘/bin/’ls: Permission denied
```

programs have *limited privileges*

OS tracks "user" of running every program

result: malware I installed shouldn't be able to effect other users

idea 1: reuse this support for web browsers

- webpage should run as "different user"
- malware should only affect web browser?

the privilege separation idea

can't make whole browser run as "different user"

- still need to save files, read password, etc.

how about just the parts that are "dangerous"?

- part that runs scripts, parses HTML

simple privilege separation

simple example: want to show videos

video decoding library is tens of thousands of lines of code

- often buggy, includes hard-to-check hand-written assembly

what does video decoding library do?

- read video file as input
- output images as output

simple privilege seperation

setup: create new user

start video decoder as new user

communicate via “pipes”

- like terminal to be used by program

simple privilege seperation

```
/* dangerous video decoder to isolate */
int main() {
    /* switch to right user */
    SetUserTo("user-without-privileges"));
    while (fread(videoData, sizeof(videoData), 1, stdin) > 0) {
        doDangerousVideoDecoding(videoData, imageData);
        fwrite(imageData, sizeof(imageData), 1, stdout);
    }
}

/* code that uses it */
    FILE *fh = RunProgramAndGetFileHandle("./video-decoder");
    for (;;) {
        fwrite(getNextVideoData(), SIZE, 1, fh);
        fread(image, sizeof(image), 1, fh);
        displayImage(image);
    }
```

issues with privilege separation (1)

"other user" can still do too much

read unprotected files

most of them?

write temporary files?

open network connections

use all your memory

...

issues with privilege separation (2)

awkward to do

switching users requires special permissions

seperate user for *each* video decoder, audio decoder, web page renderer?

- users can debug processes from same user

slowdown — extra copying

program to OS interface

primary way application talks to OS: system calls

function calls that request OS do something

typically: how program can interact with rest of system

- files
- other programs
- the network
- devices
- ...

controlling program behavior: control what system calls

“sandboxing”

result of filtering operations called a “sandbox”

idea: attacker can play in sandbox as much as they want

can’t do anything “harmful”

other possible implementations:

- e.g. virtual machine

Linux system call filtering API

privilege seperation support: system call filtering

simple API: `prctl(SECCOMP_SET_MODE_STRICT, 0, 0)`

"The only system calls the calling thread is permitted to make are `read`, `write`, `_exit`, and `sigreturn`. Other system calls [kill the program]."

read/write only work on *already open files*

later: what if we want to be finer-grained?

strace hello_world (1)

```
#include <stdio.h>
int main() { puts("Hello, World!"); }
```

when statically linked:

```
execve("./hello_world", ["./hello_world"], 0x7ffeb4127f70 /* 28 vars */)
                                        = 0
brk(NULL)                               = 0x22f8000
brk(0x22f91c0)                          = 0x22f91c0
arch_prctl(ARCH_SET_FS, 0x22f8880)      = 0
uname({sysname="Linux", nodename="reiss-t3620", ...}) = 0
readlink("/proc/self/exe", "/u/cr4bd/spring2023/cs3130/slide"..., 4096)
                                        = 57
brk(0x231a1c0)                          = 0x231a1c0
brk(0x231b000)                          = 0x231b000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
                                                     directory)
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 4), ...}) = 0
write(1, "Hello, World!\n", 14)         = 14
exit_group(0)                           = ?
```

aside: what are those syscalls?

execve: run program

brk: allocate heap space

arch_prctl(ARCH_SET_FS, ...): thread local storage pointer

may make more sense when we cover concurrency/parallelism later

uname: get system information

readlink of /proc/self/exe: get name of this program

access: can we access this file [in this case, a config file]?

fstat: get information about open file

exit_group: variant of exit

only after starting? (1)

okay, but that's only after starting up, right...?

surely simpler if we limit system calls after startup

yes, but...

only after starting? (2)

```c
#include <stdio.h>
int main() {
    FILE *fh = fopen("output.txt", "w");
    fprintf(fh, "example");
    fclose(fh);
}
```

```
$ strace ...
... [startup stuff, not shown] ...
openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
newfstatat(3, "", {st_mode=S_IFREG|0664, st_size=0, ...}, AT_EMPTY_PATH) = 0
write(3, "example", 7)                    = 7
close(3)                                  = 0
```

only after starting? (2)

```
#include <curl/curl.h>
int main() {
    CURL *handle = curl_easy_init();
    curl_easy_setopt(handle, CURLOPT_URL, "https://www.cs.virginia.edu/~cr4bd/test.txt");
    curl_easy_perform(handle);
    ...
}
```

```
$ strace ...
... [startup stuff, not shown] ...
futex(0x73f0bd640ba4, FUTEX_WAKE_PRIVATE, 2147483647) = 0
...
openat(AT_FDCWD, "/usr/lib/ssl/openssl.cnf", O_RDONLY) = 3
...
sysinfo({...}) = 0
...
socket(AF_INET6, SOCK_DGRAM, IPPROTO_IP) = 3
close(3)                                = 0
socketpair(AF_UNIX, SOCK_STREAM, 0, [3, 4]) = 0
fcntl(3, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
...
rt_sigaction(SIGPIPE, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
...
```

Linux system call filtering: detailed

Linux supports more fine-grained system call filtering

using BPF (Berkeley Packet Filter) programming language

- compiled in the kernel to assembly to check system calls

can check system call argument values, but...

- problems with pointer arguments
- too many system calls

Linux system call: open

```
open("foo.txt", O_RDONLY);
```

parameters:

- system call number: 2 (“open”)
- argument 1: 0x7fffe983 (address of string “foo.txt”)
- argument 2: 0 (value of “O_RDONLY”)

very problematic to filter using BPF interface

can deal with using ‘ptrace’ — Linux debugging interface

- BPF can trigger something like a debugger breakpoint
- breakpoint wakes up monitor program (attached like debugger)
- ‘monitor’ program can perform system call on program’s behalf

BPF filter example (1)

showing syntax for producing machine code from C macros / non-extended BPF

```
// memory[offset of "nr"] --> accumulator
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
// if (accumulator == SYS_write) PC += 1
BPF_STMT(BPF_JMP | BPF_JEQ, BPF_K, SYS_write, 1, 0),
// return "kill process"
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL_PROCESS),
// return "allow"
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
```

BPF filter example (2)

```
// memory[offset of "nr"] --> accumulator
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, nr))),
// if (accumulator == SYS_write) PC += 1 else PC += 0
BPF_STMT(BPF_JMP | BPF_JEQ, BPF_K, SYS_write, 1, 0),
// return "kill process"
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL_PROCESS),
// memory[offset of args[0]] --> accumulator
BPF_STMT(BPF_LD | BPF_W | BPF_ABS, (offsetof(struct seccomp_data, args[0])
// if (accumulator == 2) PC += 1 else PC += 0
BPF_STMT(BPF_JMP | BPF_JEQ, BPF_K, 2, 1, 0),
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_KILL_PROCESS),
BPF_STMT(BPF_RET | BPF_K, SECCOMP_RET_ALLOW),
```

other BPF operations

arithmetic (add, or, xor, ...)

in eBPF (extended BPF): 10 additional registers

- not just accumulator

running BPF fast/safely

idea: can verify in advance that...

there are no loops

there are no out-of-bounds accesses

convert to assembly function to run very fast

libseccomp

wrapper for writing BPF programs

specify list of rules re: system call identifiers/arguments

it generates BPF program with LDs, JMPs, etc.

```
#define CHECK(x) if (!(x)) handle_error();
...
scmp_filter_ctx filter = seccomp_init(SCMP_ACT_KILL_PROCESS);
CHECK(seccomp_rule_add(filter, SCMP_ACT_ALLOW, SCMP_SYS(read)
CHECK(seccomp_rule_add(filter, SCMP_ACT_ALLOW, SCMP_SYS(write
CHECK(seccomp_load(filter) == 0);
```

Linux system call: open

```
open("foo.txt", O_RDONLY);
```

parameters:

- system call number: 2 (“open”)
- argument 1: 0x7fffe983 (address of string “foo.txt”)
- argument 2: 0 (value of “O_RDONLY”)

very problematic to filter using BPF interface

can deal with using ‘ptrace’ — Linux debugging interface

- BPF can trigger something like a debugger breakpoint
- breakpoint wakes up monitor program (attached like debugger)
- ‘monitor’ program can perform system call on program’s behalf

lots of ways to open (1)

let's say we want to allow/disallow (but not 'normal' files):

```
open("/dev/keyboard", O_RDONLY);
```

problem 1: some other ways of doing that?

```
chdir("/dev");
open("keyboard", O_RDONLY);

open("../../../../../../dev/keyboard", O_RDONLY);

symlink("/dev", "/tmp/foo");
open("/tmp/foo/keyboard", O_RDONLY);
```

lots of ways to open (2)

let's say we want to allow/disallow:

```
open("/dev/keyboard", O_RDONLY);
```

problem 2: filter language doesn't allow reading pointers

- string is passed via pointer

problem 3: string can be changed from another core

- between when filter runs and when syscall runs

lots of ways to open (3)

let's say we want to disallow:

```
open("/dev/keyboard", O_RDONLY);
```

problem 4: several other syscalls (that might be used innocently)

- openat, open_by_handle_at
- would need to write additional filter rules
- ...or break programs that aren't trying to violate rule

Linux system calls

x86-64 linux: *313 system calls*

opening a file:

- open (number 2)
- openat (number 257)
- open_by_handle_at (number 304)

coordinating between threads (for using multiple cores):

- rt_sigaction (number 13)
- rt_sigprocmask (number 14)
- rt_sigreturn (number 15)
- tkill (number 200)
- futex (number 202)
- set_robust_list (number 273)
- get_robust_list (number 274)

shared services?

often programs do operations by talking to "server" program

- example: GUI management on Linux (X11 or Wayland), OS X (WindowServer)
- example: mixing sound from multiple applications
- ...

whole extra set of calls to sanitize

- when to allow "get keyboard input" for GUI
- when to allow "get microphone input" for sound manager
- making sure one isn't manipulating wrong program's windows?

also, server programs might have security problems

- common "sandbox escape"

Chrome architecture

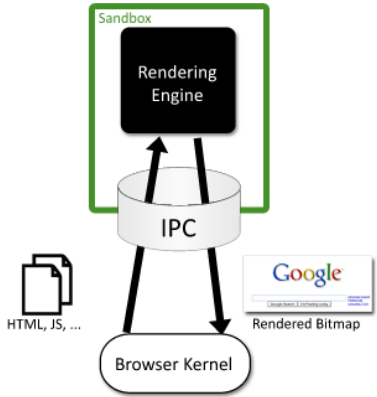


Figure 1: The browser kernel treats the rendering engine as a black box that parses web content and emits bitmaps of the rendered document.

talking to the sandbox

browser kernel sends commands to sandbox

sandbox sends commands to browser kernel

idea: commands only allow necessary things

original Chrome sandbox interface

sandbox to browser “kernel”

- show this image on screen
  - (using shared memory for speed)
- make request for this URL
- download files to local FS
- upload user requested files

browser “kernel” to sandbox

- send user input

original Chrome sandbox interface

sandbox to browser “kernel”

- show this image on screen
  - (using shared memory for speed)
- *make request for this URL*
- download files to local FS
- upload user requested files

browser “kernel” to sandbox

- send

needs filtering — at least no `file:` (local file) URLs

original Chrome sandbox interface

sandbox to browser “kernel”

- show this image on screen
  - (using shared memory for speed)
- *make request for this URL*
- download files to local FS
- upload user requested files

browser “kernel” to sa

- send user input

can still read any website!
still sends normal cookies!

original Chrome sandbox interface

sandbox to browser “kernel”

- show this image on screen
  - (using shared memory for speed)
- make request for this URL
- *download files to local FS*
- upload user requested files

browser “kernel” t

- send user input

files go to download directory only
can’t choose arbitrary filenames

original Chrome sandbox interface

sandbox to browser "kernel"

- show this image on screen
  - (using shared memory for speed)
- make request for this URL
- download files to local FS
- *upload user requested files*

browser "kernel" to sandbox

- send user input

browser kernel displays file choser
only permits files selected by user

Site Isolation

Chrome since version 67 (desktop)/77 (Mobile) has process per site

site $\approx$ registered domain name (example.com, example.co.uk, etc.)

- slightly different than same origin policy

complicated to implement:

- single web page can embed content from multiple other sites
  - and those other sites can embed content from yet more sites
- web page can call services on other websites with "permission" of other website
- clicking link may or may not requiring switching to new process

same separation now present in Firefox

OpenSSH privilege seperation

OpenSSH uses privilege seperation for its SSH server

what runs on the lab machines when you log into them

separate network processing code from authentication code

seperate process per connection — users don't share

developed before system call filtering was widely available

- uses separate user + chroot (we'll talk later) to isolate

OpenSSH privsep protocol

sandboxed process tells “monitor” to:

perform *cryptographic operations*

- long-term keys never in sandboxed process
- commands to ask for cryptographic messages they need

ask to switch to user — if given user password, etc.

- *monitor process verifies* login information

after authentication: new process running as logged-in user

- (normally) no issues with special privileges

privilege seperation overall

large application changes

- OpenSSH: 3k lines of code for communication/etc. added
- OpenSSH: 2% of existing code (950 of 44k lines) changed
  (but most changes simple)

lots of application knowledge

- what is a meaningful separation of ‘privileged’ and ‘unprivileged’?

better application design anyways?

privilege separation for

let's say we wanted to add sandboxing/privilege separation to an (standalone) mail program

exercise 1: where would be concerned about security problems?

exercise 2: propose a way of dividing up the program

changing what programs can name

seccomp, separate users: program tries to access X, checks if allowed

alternate idea: changing what Xs program can name

aside: capabilities/ambient authority (1)

user permissions — authority tied to each running program

"access control lists" for resources

sometimes called "ambient authority"

alternate model: "capabilities"

running program has list of things it can access/how

aside: capabilities/ambient authority (2)

capabilities = program has list of things it can access

most common thing with design: open files

used as basis of some operating system designs

- not desktop OSes, but...
- Unix/Linux has many things with 'flavor' of capabilities

in "fully" capability-based OSes also...

- capabilities for accessing non-regular-file resources (processes, directories, network ports, ...)
- way of transferring capabilities between programs (instead of, e.g., filenames/PIDs/etc.)
- OS doesn't track user IDs/etc. (though maybe system services do)

Unix filesystems and mounting

my Linux desktop has two disks:

- `/` — an SSD
- `/mnt/extradisk` — a hard drive

hard drive appears as *subdirectory* of SSD

subdirectory called a *mount point*

per-process root

on Unix: each process tracks its own root directory (/)

can be changed with chroot() system call

- command-line tool to access: `chroot`

usage: can isolate program from other files on system

- example: limit what public file server can access?

chroot ls

```
# mkdir /tmp/example
# cp /bin/ls /tmp/example/ls
# chroot /tmp/example /ls
chroot: failed to run command ‘/’ls: No such file or directory
# cp -r /lib64 /tmp/example/lib64
# mkdir -p /tmp/example/lib
# cp -r /lib/x86_64-linux-gnu /tmp/example/lib/x86_64-linux-gnu
# chroot /tmp/example /ls
/ls: error while loading shared libraries: libpcre2-8.so.0: cannot
# cp /usr/lib/x86_64-linux-gnu/libpcre2-8* /tmp/example/lib/x86_64-
# chroot /tmp/example /ls /
lib  lib64  ls
# chroot /tmp/example /ls /..
lib  lib64  ls
#
```

duplicating OS?

seems like we need second copy of system files

modern Linux has better solution (more detail later):

"bind mounts"

make alias to part of normal filesystem in temporary directory

accessing outside the root (1)

can still have open files outside the root

example: `chroot /tmp/example /ls >tmp.txt`

‘ls’ running from chroot writes to ‘tmp.txt’

accessing outside the root (2)

can still have open directories outside the root

```
int dir_fd = open("/tmp/other", O_PATH);
// change root to /tmp/example, cd to /
if (0 != chroot("/tmp/example")) handle_error();
if (0 != chdir("/")) handle_error();
// access /tmp/other/other.txt through old open directory
int other_txt_fd = openat(dir_fd, "other.txt", O_RDONLY);
read(other_txt_fd, ...);
// access /tmp/outside.txt through old open directory
int outside_fd = openat(dir_fd, "../outside.txt", O_RDONLY);
read(other_fd, ...);
```

oops.

chroot escapes

chroot prevents accessing files outside the new /

but root (system adminstrator) user in chroot can access disks, etc.

typical usage: combine chroot with extra user

chroot impracticality

some things make chroot impractical in general:

seems like one needs extra copies of most of the system

hard to communicate between separate roots

requires administrator permissions to configure

- dangerous to let normal users configure b/c they could confuse priviliged (set-user-ID) programs like `sudo`

exercise

what scenarios does chroot make most/least sense for?

A. the rendering part of web browser

B. a web server

C. a media player

D. a network time server (for other machines to set their clocks)

Linux namespaces (1)

Linux: alternate sandboxing features

"namespaces" for other resources

chroot: each process has own idea of root directory

- change to OS: look up root directory in process, not global variable

can apply this to other resources:

- what filesystems (disks) are available
- what network devices are available
- what user identifier numbers are
- ...

Linux namespaces (2)

user namespace:

can run programs with new view of users:

inside namespace: running as root

outside namespace: root translated to innocent user ID

allows running programs that expect different users

...without changes, but without giving special permissions

mechanism: reassign user ID numbers in kernel

figuring out what user ID means — always apply current process mapping

aside: Linux clone(), unshare() syscalls

Linux clone system call: start new process (or thread)

flags to specify environment of new process

these flags can include "make a new namespace of a type"

```
int id = clone(start_function, ..., CLONE_NEWUSER | other-flags);
```

above option: new user namespace for new process

alternative: for changing current process's namespace:

```
unshare(CLONE_NEWUSER);
```

user namespaces API

Linux: users identified by numerical *user IDs* (UIDs)

with user namespaces:

control file `/proc/PROCESS-ID/UID_MAP` contains lines like:

- `0 1000 2` — UID 0–1 maps to UID 1000–1001
- `1000 2000 100` — UID 1000-1100 maps to UID 2000–2100

can write to that file to reconfigure (if enough permissions)

Linux namesapces (3)

mount namespaces:

- Unix: mounting disk = making the contents of the disk available as directories+files

different idea of what filesystems are available

can be setup with *bind mounts* to "real FS"

- but otherwise: no access to directories outside mount namespace
- normally requires root — but special case with user namespaces

mount namespaces API

from command line:

```
    # runs shell (/bin/sh) in new mount namesapce
shell1$ unshare --mount /bin/sh

    # setup directories in /tmp/workdir and make them aliases of th
    # these aliases will only exist for processes in mount namespac
shell2$ mkdir -p /tmp/workdir/bin
shell2$ mkdir -p /tmp/workdir/lib
shell2$ mkdir -p /tmp/workdir/usr
shell2$ mkdir -p /tmp/workdir/current
shell2$ mount -o bind,ro /bin /tmp/workdir/bin
shell2$ mount -o bind,ro /lib /tmp/workdir/lib
shell2$ mount -o bind,ro /usr /tmp/workdir/usr
shell2$ mount -o bind /home/someuser /tmp/workdir/current

    # start new shell with the root directory being /tmp/workdir
shell2$ chroot /tmp/workdir /bin/sh
shell3$ cd /
```

Linux namespaces (3)

user namespace and mount namespace together:

run program in new user namespace

map regular root (in namespace) to regular user

- "opts out" of programs like sudo

move to new mount namespace

setup bind mounts + chroot

- special case: allowed because root in user namespce
- can't get "real" root (administrator) privileges ever

run program with subset of available files

Linux namespaces (4)

other resources with namespaces

network — common usage: virtual network device for set processes

- different "what is my IP address?" answer for different processes

hostname ("UTS")

process identifiers

control groups (resource limits for memory, CPU usage, disk I/O, etc.)

Linux control groups

control groups — tied to namespaces

primarily: CPU/memory/IO performance restrictions

- primarily intended for 'friendly sharing' (containers, etc.)
- important for preventing denial-of-service/etc.
- not as big a security conern as file/user/etc. access

also mechanism for adding IO device restrictions

also mechanism to start/stop a bunch of processes together

Linux sandboxing programs, generally

docker, lxc, lxd, containerd

- use namespaces to create "container" with own copy of OS libraries, services
- but containers share OS 'kernel' and potentially files with host unlike VM
- (might also have options to use other ways of getting this functionality — VM's, etc.)

bubblewrap, firejail

- use Linux namespace tools + "bind mounts" to give programs only subset of files, etc.
- firejail has option of running a "proxy" windowing system server

SELinux's sandbox

- uses Security Enhanced Linux's mandatory access controls instead of

containers

Linux's seccomp + namespaces + SELinux commonly used to implement containers

(plus cgroups (control groups) for performance isolation)

usual goal: looks like virtual machine, but much lower overhead

examples: Docker, Kubernetes

(note: these may also support other ways of creating 'lightweight VMs')

runc bug

2019 bug in Docker, other container implementations (CVE-2019-5736)

- blog post for vulnerability finders:
  `https://blog.dragonsector.pl/2019/02/cve-2019-5736-escape-from-docker-and.html`

bug setup:

- user starts malicious container X
- user tells docker to start a new command in malicious container X
- malicious container X hijacks the "new command" starting program
- hijacked program used to access stuff outside container

part of problem: Docker and others weren't using user namespaces at the time

- compatability problems

runc bug

2019 bug in Docker, other container implementations (CVE-2019-5736)

- blog post for vulnerability finders:
  `https://blog.dragonsector.pl/2019/02/cve-2019-5736-escape-from-docker-and.html`

bug setup:

- user starts malicious container X
- user tells docker to start a new command in malicious container X
- *malicious container X hijacks the "new command" starting program*
- hijacked program used to access stuff outside container

part of problem: Docker and others weren't using user namespaces at the time

- compatability problems

setup: /proc/PID

Linux provides /proc directory to access info about programs

used for implementing process list utils, debugging

- needed to make a functional container

subdirectory for each process in current container

- process ID PID has /proc/PID subdirectory
- /proc/self is alias for current process's subdirectory

included is /proc/PID/exe file — alias for executable file

running a command in existing container

to run command X in existing container:

step 1: switch current process to that container

step 2: execute command X

running a command in existing container

to run command X in existing container:

step 1: switch current process to that container

code in container can access /proc here?

including overwriting /proc/self/exe!

which is a program run as root!

step 2: execute command X

partial fix

can disable access to /proc/PID/exe (and related things)

system call: `prctl(PR_SET_DUMPABLE, 0)`

but...the run-in-container tool did this for a while

partial fix

can disable access to /proc/PID/exe (and related things)

system call: `prctl(PR_SET_DUMPABLE, 0)`

but...the run-in-container tool did this for a while

problem: this gets reset on executing a new program

and attacker could make the new program be /proc/PID/exe

- one mechanism: symbolic links (file aliases)

but change dynamic linking setup to run attacker code

...which accesses /proc/self/exe

full fix

make single-use copy of start-in-container tool each time command run

- in-memory file

...so modifying it doesn't change anything

- (but it's also protected from modification)

other solutions:

- make executable non-writable (e.g. SELinux, don't run container as root)

Chrome sandbox escape (1)

recall: renderer communicates with ‘browser kernel’

‘browser kernel’ might have bugs in this interface

- example: `https://project-zero.issues.chromium.org/issues/42451090` — missing access check for ‘wrong’ website
- example: `https://chromereleases.googleblog.com/2021/09/stable-channel-update-for-desktop.html` `https://starlabs.sg/blog/2022/01-the-cat-escaped-from-the-chrome-sandbox/` — use after free in API available from renderer

Chrome sandbox escape (2)

Chrome Windows sandbox

based on “restricted access tokens”

- on Windows, programs have ‘access tokens’ representing their permissions
- can create ‘restricted’ version to limit access

Chrome sandbox escape (3)

https://googleprojectzero.blogspot.com/2020/04/you-wont-believe-what-this-one-line.html:

problem: Windows erronously allowed starting new processes with a more unrestricted token

- supposed to have check where the token being used came from, but incomplete

...but needed to get 'handle' to that token

solution: ask to duplicate another running process

Android sandbox

Android — Linux based OS for phones/tablets

`https://source.android.com/security/app-sandbox`

current version: SELinux + seccomp (system call filter)

OS X sandboxing

OS X (tries to) implement system call filtering

main challenge: what about files?

- user can open a file anywhere — we expect that to work

OS X sandboxing

OS X (tries to) implement system call filtering

main challenge: what about files?

- user can open a file anywhere — we expect that to work

OS X solution: OS service displays file-open dialog

- OS knows user really choose a file

application can ask to remember file was chosen previously

not chosen/remembered — can't access

- requires changes to how applications open files

another sandboxing OS: Qubes

Qubes: heavily sandboxed OS

runs *seperate VMs* instead of filtering syscalls

UI that clearly shows what VM each window is from

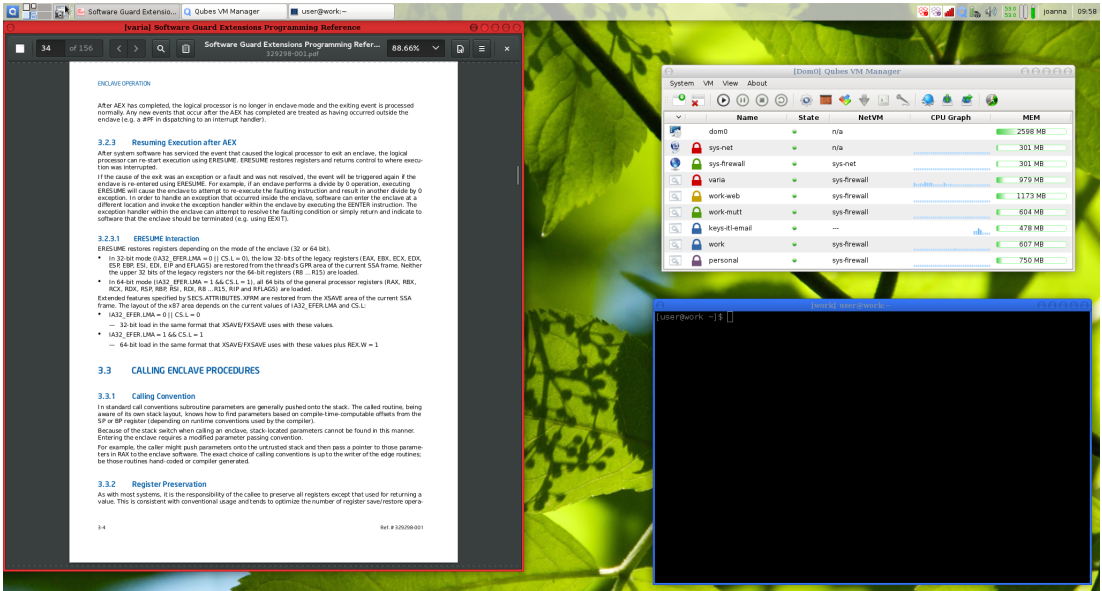
advantage: easier to gaurentee isolation

- many, many more bugs in system call filtering than VMs

disadvantage: harder to share between VMs

disadvantage: much more runtime overhead

Qubes screenshot



which sandboxing?

which whole-application sandboxing technique seems better for

security, performance, usability, handling unchanged applications

(full answer: could mix techniques + probably depends on details of app)

A. chroot + system call filtering

B. chroot + mount and user namespaces

C. virtual machine dedicated to application

D. SELinux-like mandatory access control

sandboxing without OS support

so far: relying on OS features for sandboxing

good reasons:

- primarily want to filter system calls
- hardware-assisted, strong protection

but problems with relying on OS:

- sending information in/out of sandbox relatively slow
- requires heavily OS-specific code

sandboxing without OS ideas

'dynamic' language virtual machine, like Java VM, .Net CLR

- hard to use with code intended to compile to native machine code

virtual machine targetted for C/C++-like code, like WebAssembly

assembly-to-assembly conversion

- example: Wahbe, Lucco, Anderson, and Graham, "Efficient Software-Based Fault Isolation" (1993)
- example: Ford and Cox, "Vx32: Lightweight User-level Sandboxing on the x86" (2008)

WebAssembly

WebAssembly: language virtual machine specification intended...

- similar idea to Java VM

to be compiled to from C/C++

- support by Clang/LLVM

to be easy to just-in-time compile to native machine code

to be run in web browsers (fast web apps)

WebAssembly memory management

WebAssembly 'modules' have a single "linear memory"

starts at index 0, goes to some maximum

load/store instructions take index into current memory

observation 1: close to memory model "normal" C/C++ code expects

observation 2: only goal is to prevent sandbox (WebAssembly) code from interfering with outside code

...so no need to check array bound or similar

WebAssembly validation

WebAssembly virtual machine code designed to be *validated* before running

allows for efficient interpreters or conversion to assembly

- validation ensures that you can safely skip certain type checks, etc.

language specification very explicit about what needs to be checked at runtime

example WebAssembly validation

check that instructions have right number of operands available

- WebAssembly instructions use stack (compile 2 + 2 into 2 2 +)

check operands that can be checked (constants)

check the calls go to only functions listed in table

- should make it easier to do just-in-time compilation to machine code?

check the branches go to only locations listed in table, and only within one function

- should make it easier to do just-in-time compilation to machine code?

example WebAssembly instruction specification

return

1. Let F be the current frame.
2. Let n be the arity of F.
3. Assert: due to validation, there are at least n values on the top of the stack.
4. Pop the results val^n from the stack.
5. Assert: due to validation, the stack contains at least one frame.
6. While the top of the stack is not a frame, do:
    a. Pop the top element from the stack.
7. Assert: the top of the stack is the frame F.
8. Pop the frame from the stack.
9. Push val^n to the stack.
10. Jump to the instruction after the original call that pushed the frame.

WebAssembly as sandboxing

can compile existing C/C++ library using WebAssembly...

then call using language virtual machine

RLBox

saw interfaces for using sandboxes from user perspective?

what about for privilege separation?

- recall: like Chrome separate renderer process idea
- need to navigate OS sandboxing API + create interface for sandboxed part?

some reusable tools have appeared for this (but no clear winner)

one example: RLBox (published in Usenix Security 2020)

- Shravan Narayan and Craig Disselkoen, UC San Diego; Tal Garfinkel, Stanford University; Nathan Froyd and Eric Rahm, Mozilla; Sorin Lerner, UC San Diego; Hovav Shacham, UT Austin; Deian Stefan, UC San Diego

RLBox usage

part of example from author's presentation:

goal: invoke JPEG parser in sandbox

```
autosandbox = rlbox::create_sandbox<wasm>();
tainted<jpeg_decompress_struct*> p_jpeg_img = sandbox.malloc_in_sandbox<jpeg_decompress_stru
tainted<jpeg_source_mgr*> p_jpeg_input_source_mgr = sandbox.malloc_in_sandbox<jpeg_source_mg
sandbox.invoke(jpeg_create_decompress, p_jpeg_img);
p_jpeg_img->src = p_jpeg_input_source_mgr;
p_jpeg_img->src->fill_input_buffer = ...;
sandbox.invoke(jpeg_read_header,p_jpeg_img/*...*/);
```

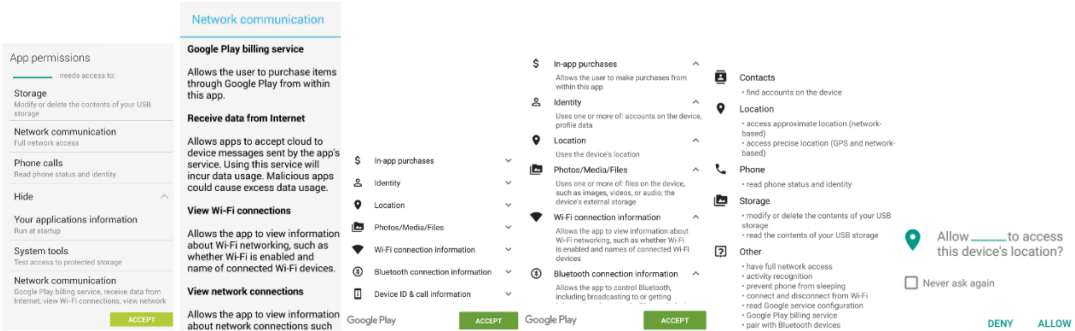
tool handles running 'jpeg_create_decompress', 'jpeg_read_header' in sandbox

values shared with sandbox marked as "tainted"

C++ (template) class

this example: using WebAssembly-based sandbox

some Android prompts



(a) Explicit (b) Explicit(II) (c) Grouped (d) Grouped(II) (e) Grouped(III) (f) Request

Figure 1: The permissions displays under consideration. From left to right: explicit permissions model (Explicit) prior to Play Store 4.8.20, expanded explicit permissions (Explicit(II)) for "Network Communication", grouped permissions (Grouped) after Play Store 4.8.20, expanded grouped permissions (Grouped(II)) for all displayed categories, detailed group permissions (Grouped(III)) for the app on the Play Store, and a permission request (Request) for Location in Android M.

from Clark et al, "No Time At All: Opportunity Cost of Android Permissions" (HotWireless'16)

UI problems with application permissions

do applications request sensible permissions?

do users pay attention to permission requests?

do users understand what permissions mean?

are permissions fine-grained enough?

are permissions coarse-grained enough?

UI problems with application permissions

do applications request sensible permissions?

do users pay attention to permission requests?

do users understand what permissions mean?

are permissions fine-grained enough?

are permissions coarse-grained enough?

right permissions?

Felt, Chin, Hanna, Song and Wagner, "Android Permissions Demystified" (CCS 2011)

used static analysis to compare requested permissions to what applications did

- at the time: permissions requested at installation

sample of 900 applications

estimate approx 200 over-privileged

- (estimate because using false positive rate from manual checking)

why extra permissions?

selected from Felt et al's analysis:

developers confused similar permissions

- `ACCESS_NETWORK_STATE` versus `ACCESS_WIFI_STATE`

developers thought permissions were needed for delegated tasks

- `CALL_PHONE` not needed to invoke phone app
- `INSTALL_APPLICATION` not needed to open app store install dialog

developers thought permissions needed for all methods of class

- `WRITE_SETTINGS` when using (no-permission) read-settings operations

copy-and-paste

UI problems with application permissions

do applications request sensible permissions?

do users pay attention to permission requests?

do users understand what permissions mean?

are permissions fine-grained enough?

are permissions coarse-grained enough?

a user study (2012)

Felt, Ha, Egelman, Haney, Chin, Wagner, "Android Permissions: User Attention, Comprehension, and Behavior"

performed lab study; task: find + install coupon app

at the time: Android prompted for permissions on installation

a user study (2012)

Felt, Ha, Egelman, Haney, Chin, Wagner, "Android Permissions: User Attention, Comprehension, and Behavior"

performed lab study; task: find + install coupon app

at the time: Android prompted for permissions on installation

17% looked at app permissions detail

42% aware of permissions

42% unaware of permissions

versus: 88% read reviews

a user survey (2012)

same paper did survey about what permissions meant

three multiple choice questions

selected from bank of 11

302 respondents; 3 fully correct

average 21%

example survey question

'Read phone state and identity' allows which of these?

Read your phone number

See who you have called

Track you across applications

Load adverisements

survey questions (1)

INTERNET Category: Network communication Label: Full Internet access	109	✔ Send information to the application's server	45	41.3%
		✔ Load advertisements	30	27.5%
		✗ None of these	16	14.7%
		✗ Read your text messages	13	11.9%
		✗ Read your list of phone contacts	11	10.1%
		I don't know	36	33.0%
READ_PHONE_STATE Category: Phone calls Label: Read phone state and identity	85	✔ Read your phone number	41	47.7%
		✗ See who you have called	37	43.0%
		✔ Track you across applications	20	23.3%
		✗ Load advertisements	11	12.8%
		✗ None of these	10	11.6%
		I don't know	15	17.4%
CALL_PHONE Category: Services that cost you money Label: Directly call phone numbers	83	✔ Place phone calls	30	35.3%
		✗ Charge purchases to your credit card	27	31.8%
		✗ None of these	16	18.8%
		✗ See who you have made calls to	14	16.5%
		✗ Send text messages	11	12.9%
		I don't know	16	18.8%
WRITE_EXTERNAL_STORAGE Category: Storage Label: Modify/delete SD card contents	92	✔ Read other applications' files on the SD card	41	44.6%
		✔ Change other applications' files on the SD card	39	42.4%
		✗ None of these	16	17.4%
		✗ See who you have made phone calls to	15	16.3%
		✗ Send text messages	11	12.0%
		I don't know	15	16.3%

survey questions (2)

WRITE_EXTERNAL_STORAGE Category: Storage Label: Modify/delete SD card contents	92	✔ Read other applications' files on the SD card	41	44.6%
		✔ Change other applications' files on the SD card	39	42.4%
		✗ None of these	16	17.4%
		✗ See who you have made phone calls to	15	16.3%
		✗ Send text messages	11	12.0%
		I don't know	15	16.3%
WAKE_LOCK Category: System tools Label: Prevent phone from sleeping	81	✔ Keep your phone's screen on all the time	49	60.5%
		✔ Drain your phone's battery	37	45.7%
		✗ None of these	7	8.6%
		✗ Send text messages	4	4.9%
		✗ Delete your list of contacts	4	4.9%
		I don't know	13	16.0%
CHANGE_NETWORK_STATE Category: System tools Label: Change network connectivity	66	✔ Turn your WiFi on or off	36	52.9%
		✗ Send information to the application's server	13	19.1%
		✗ Read your calendar	7	10.3%
		✗ None of these	7	10.3%
		✗ See who you have made calls to	5	7.4%
		I don't know	17	25.0%

survey questions (3)

READ_SMS$_2$ Category: Your messages Label: Read SMS or MMS	54	✔ Read text messages you've sent	30	54.5%
		✔ Read text messages you've received	25	45.5%
		✗ Send text messages	10	18.2%
		✗ Read your phone's unique ID	6	10.9%
		✗ None of these	4	7.3%
		I don't know	11	20.0%
READ_SMS$_1$ Category: Your messages Label: Read SMS or MMS	77	✔ Read text messages you've received	44	56.4%
		✗ Read e-mail messages you've received	30	38.5%
		✗ Read your call history	13	16.7%
		✗ None of these	8	10.3%
		✗ Access your voicemail	8	10.3%
		I don't know	13	16.7%
READ_CALENDAR Category: Your personal information Label: Read calendar events	101	✔ Read your calendar	56	53.3%
		✗ None of these	18	17.1%
		✗ Add new events to your calendar	12	11.4%
		✗ Send text messages	12	11.4%
		✗ Place phone calls	9	8.6%
		I don't know	19	18.1%

survey questions (4)

READ_CONTACTS Category: Your personal information Label: Read contact data	86	✔ Read your list of contacts ✔ Read your call history ✗ None of these ✗ Delete your list of contacts ✗ Place phone calls *I don't know*	52 19 14 9 5 14	60.5% 22.1% 16.3% 10.5% 5.8% 16.3%
CAMERA Category: Hardware controls Label: Take pictures	72	✔ Take pictures when you press the button ✔ Take pictures at any time ✔ See pictures taken by other applications ✔ Delete pictures taken by other apps ✗ None of these *I don't know*	27 27 16 13 13 17	37.0% 37.0% 21.9% 17.8% 17.8% 23.3%

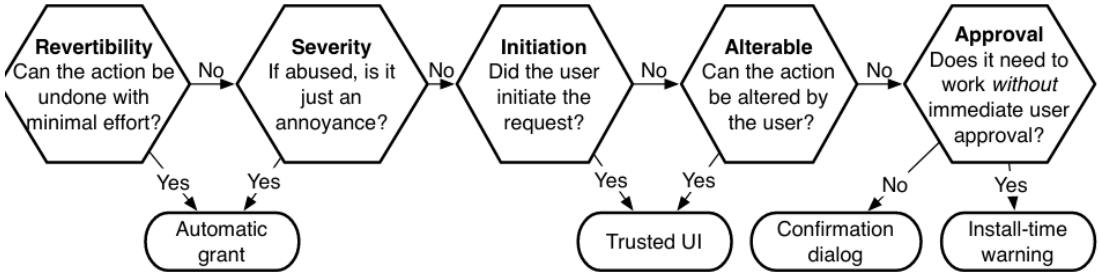


Figure 1: A guide to selecting between the different permission-granting mechanisms.

from Felt et al, "How To Ask For Permission" (HotSec'12)

principles

Felt et al list "principles":

"Conserve user attention, utilizaing it for only permissions that have severe consquences"

> too many security warnings means users won't pay attention

"When possible, avoid interrupting the user's primary task with explicit security decisions"

> users will dismiss warnings because they get in the way of work

Cloak and Dagger

Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop

Yanick Fratantonio
UC Santa Barbara
yanick@cs.ucsb.edu

Chenxiong Qian, Simon P. Chung, Wenke Lee
Georgia Tech
qchenxiong3@gatech.edu
pchung34@mail.gatech.edu
wenke.lee@gmail.com

cloak and dagger permissions

the two permissions:

SYSTEM_ALERT_WINDOW:
draw windows on top of screen
(at time: enabled by default)
BIND_ACCESSIBILITY_SERVICE:
“Observe your actions”
“Retrieve window content”

can hide window content while user interacts with it

...and stealthy get user to do more things

also, a clickjacking attack

at the time, could draw overlay window over permissions dialog

...convince user to press where "OK" button is

countermeasure: permissions dialog would detect this, ignore clicks

problem: wouldn't detect if overlay didn't cover enough of button

privacy and permissions

50 Ways to Leak Your Data:
An Exploration of Apps' Circumvention of the Android Permissions System

Joel Reardon
University of Calgary
AppCensus, Inc.

Álvaro Feal
IMDEA Networks Institute
Universidad Carlos III de Madrid

Primal Wijesekera
U.C. Berkeley / ICSI

Amit Elazari Bar On
U.C. Berkeley

Narseo Vallina-Rodriguez
IMDEA Networks Institute / ICSI
AppCensus, Inc.

Serge Egelman
U.C. Berkeley / ICSI
AppCensus, Inc.

2019 paper

many mobile application permissions related to privacy

getting phone ID, email address, location, ...

but applications (especially ad libraries) find workarounds

permissions being insufficient

permissions check limited API calls for getting private info,...

...but there were alternative, unfiltered system calls for

getting MAC address (effectively phone ID)

- Linux `ioctl` system call on socket

WiFi base station address

- ARP cache (recently seen machines on network, to know where to send packets)

location

- geolocation tag on recent photos

covert channels

advertising libraries would store phone ID/account info in a file

...when they had permissions to retrieve it

and would read phone ID/account info from a file

...when they did not

backup slides

backup slides

SELinux

Security **E**nhanced Linux

"Mandatory Access Control" system for the Linux

- mandatory: can be configured to require enumeration of files programs can access
  (versus normally: specify what files programs can't access)

not necessairily run in mandatory control mode

programs run in particular "domain"

objects (files, port numbers, other programs, etc.) can be assigned labels

rules about what labels programs are allowed to access

viewing/assigning labels (1)

```
$ ls -Z /var/log/lastlog
-rw-r--r--. root root system_u:object_r:lastlog_t:s0   /var/l
```

above: default Red Hat Linux/CentOS configuration

system user

object role

lastlog type

```
$ chcon --type=newtype_t some_file
```

assigning labels (2)

labels via: “file context mapping”

```
$ semanage fcontext --add --type web_files_t '/var/www/html(/
$ restorecon -R -v /var/www/html
```

pattern matching rules set *default* labels

restorecon — switch to default labels, applying rules

assigning rules

subset of default rules for Apache httpd (webserver):

```
define(`read_files_pattern',`
  allow $1 $2:dir search_dir_perms;
  allow $1 $3:file read_file_perms;
')
...
define(`read_lnk_files_pattern',`
  allow $1 $2:dir search_dir_perms;
  allow $1 $3:lnk_file read_lnk_file_perms;
')
...
allow httpd_t httpd_config_t:dir list_dir_perms;
read_files_pattern(httpd_t, httpd_config_t, httpd_config_t)
read_lnk_files_pattern(httpd_t, httpd_config_t, httpd_config_
```

httpd_t: 'type' for webserver process

application confinement

confining whole browsers was hard

- we trust them to do a lot of things — e.g. write arbitrary files

but maybe we can do this for simpler applications?

idea 1: applications send system calls to OS

- *limit syscalls* like we limited browser kernel commands
- constructing command language "in reverse"

filtering system calls?

example: video player VLC playing a local file on my laptop

uses *73 unique kinds of system calls*

opens many files that *are not the video file*

- libraries
- fonts
- configuration files
- translations of messages

can I limit the files my video player can read?

how do I come up with a useful filter?

exercise: app confinement options

sandboxed applications want to access display server

which option seems best for security/performance?

A. proxy for protocol display server supports natively that filters display calls

B. custom protocol that sends bitmaps + receives inputs, plus copy of display server runs with application

C. divide application into UI and non-UI part, sandbox just the non-UI part

D. have application take over screen when running, give its own display server

SELinux escape

```
When executing a program via the SELinux sandbox, the nonpriv session
can escape to the parent session by using the TIOCSTI ioctl to push
characters into the terminal's input buffer, allowing an attacker to
escape the sandbox.

$ cat test.c
#include <unistd.h>
#include <sys/ioctl.h>

int main()
{
     char *cmd = "id\n";
     while(*cmd)
      ioctl(0, TIOCSTI, cmd++);
     execlp("/bin/id", "id", NULL);
}

$ gcc test.c -o test
$ /bin/sandbox ./test
id
uid=1000 gid=1000 groups=1000
context=unconfined_u:unconfined_r:sandbox_t:s0:c47,c176
$ id    <------ did not type this
uid=1000(saken) gid=1000(saken) groups=1000(saken)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```