Operating Systems Security III

CS 1660: Introduction to Computer Systems Security

So setuid/setgid is dangerous…

setuid/setgid is dangerous...

In modern times: only for programs that really need it

- System programs that changing passwords/users, legacy programs
  - Don't do this yourself!
- Very very bad idea for shell scripts

What else can we do?

When do we actually need setuid/setgid?

Can we do better?

In the shell: su, sudo

- Run as another user (if you have permissions)

```
user@shell:~$ su -c "command" other user
```

- Run commands as root (or another user) based on system config file (/etc/sudoers)
  - Can restrict to specific commands, environment, ...

```
user@shell:~$ sudo whoami
root
```

```
/etc/sudoers:
%wheel ALL=(ALL) NOPASSWD: ALL

. . .
```

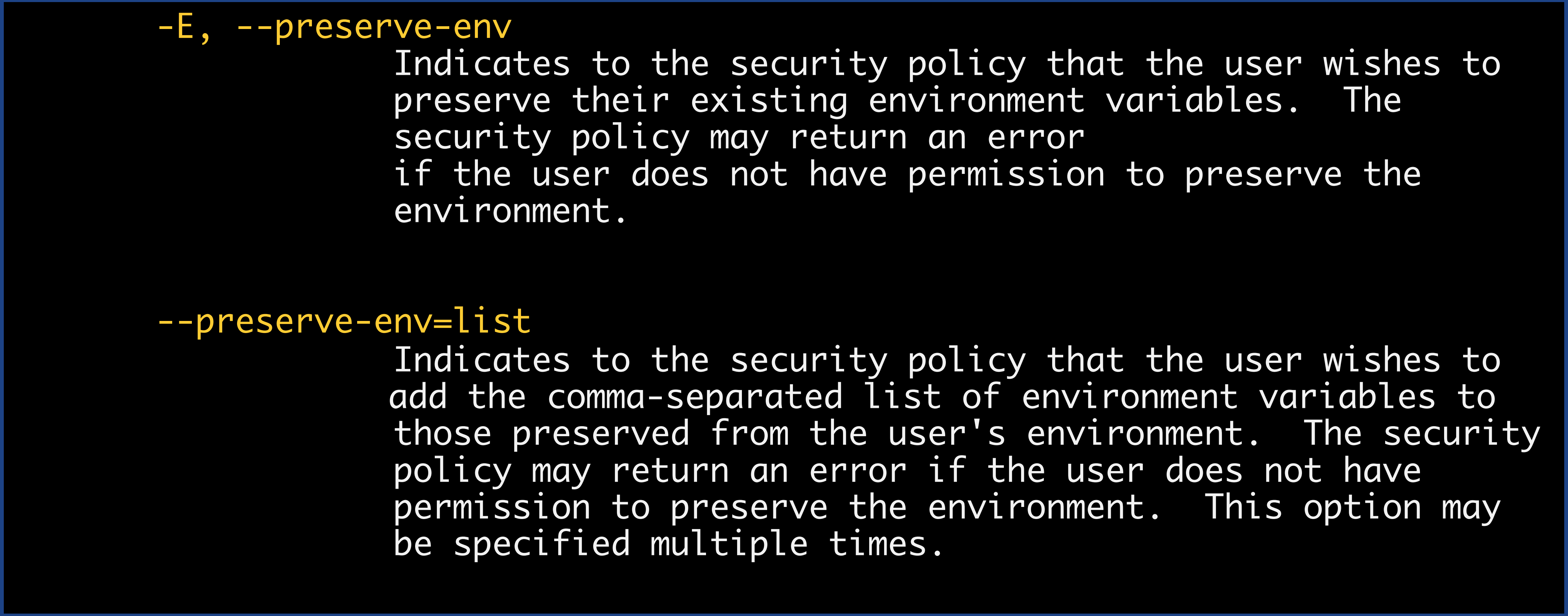
From man page on /etc/sudoers: (aka sudoers(5))

```
ALL             CDROM = NOPASSWD: /sbin/umount /CDROM,\
                /sbin/mount -o nosuid\,nodev /dev/cd0a /CDROM


Any user may mount or unmount a CD-ROM on the machines in the CDROM
Host_Alias (orion, perseus, hercules) without entering a password.
```

sudo has a LOT of features, see
`man sudoers` for details!

Why is this better?

From sudo’s man page…



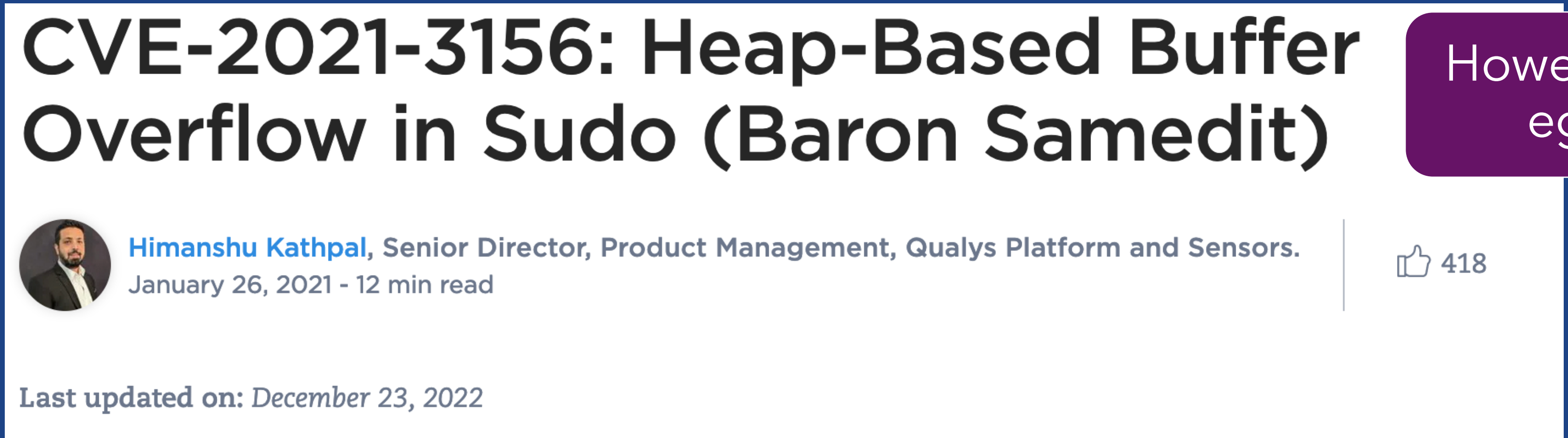
```
       -E, --preserve-env
                   Indicates to the security policy that the user wishes to
                   preserve their existing environment variables.  The
                   security policy may return an error
                   if the user does not have permission to preserve the
                   environment.


       --preserve-env=list
                   Indicates to the security policy that the user wishes to
                   add the comma-separated list of environment variables to
                   those preserved from the user's environment.  The security
                   policy may return an error if the user does not have
                   permission to preserve the environment.  This option may
                   be specified multiple times.
```

Why is this better?

- Leaves the tricky code that deals with privileges to one program (sudo)
  => Maintained by professionals, like with crypto libraries

- Application developers don’t need to decide how to elevate permissions

- One common system to decide how to authenticate and set policies
  => System users/passwords, /etc/sudoers rules



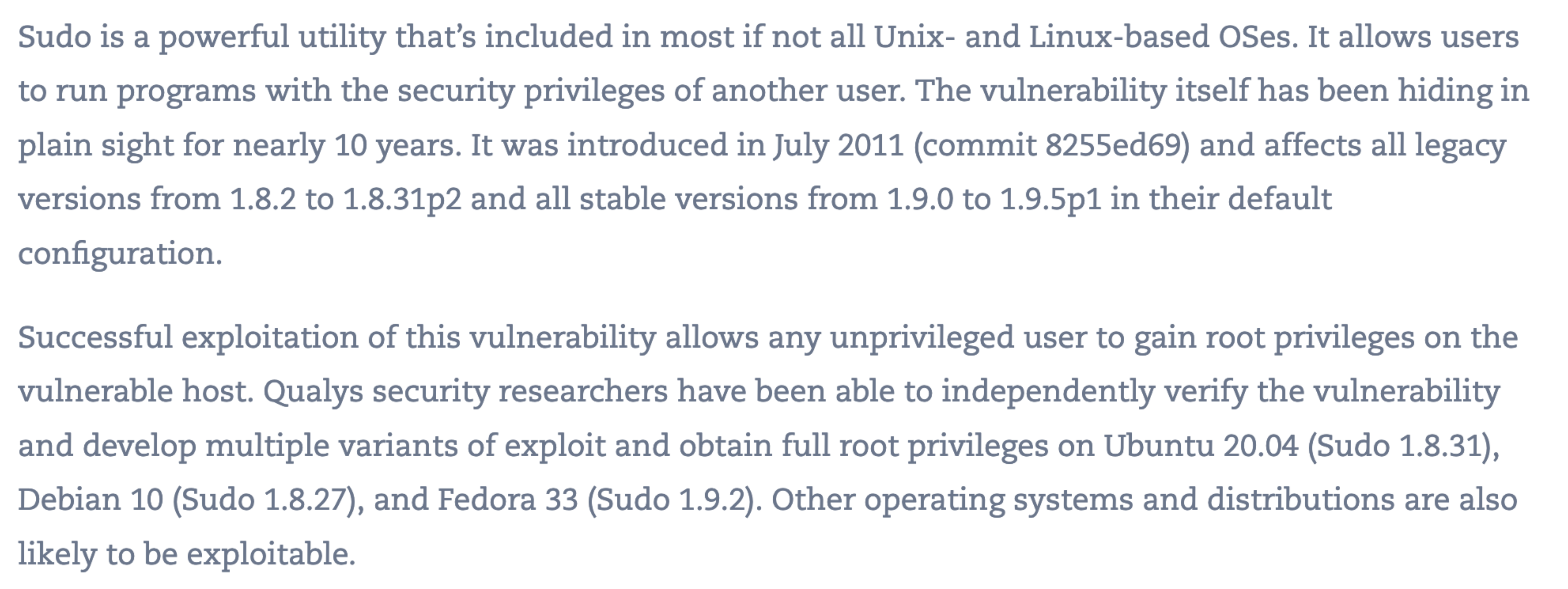
CVE-2021-3156: Heap-Based Buffer Overflow in Sudo (Baron Samedit)

Himanshu Kathpal, Senior Director, Product Management, Qualys Platform and Sensors.
January 26, 2021 - 12 min read

418

Last updated on: *December 23, 2022*

However, there can still be problems… eg. CVE-2021-3156 (more info)



Sudo is a powerful utility that's included in most if not all Unix- and Linux-based OSes. It allows users to run programs with the security privileges of another user. The vulnerability itself has been hiding in plain sight for nearly 10 years. It was introduced in July 2011 (commit 8255ed69) and affects all legacy versions from 1.8.2 to 1.8.31p2 and all stable versions from 1.9.0 to 1.9.5p1 in their default configuration.

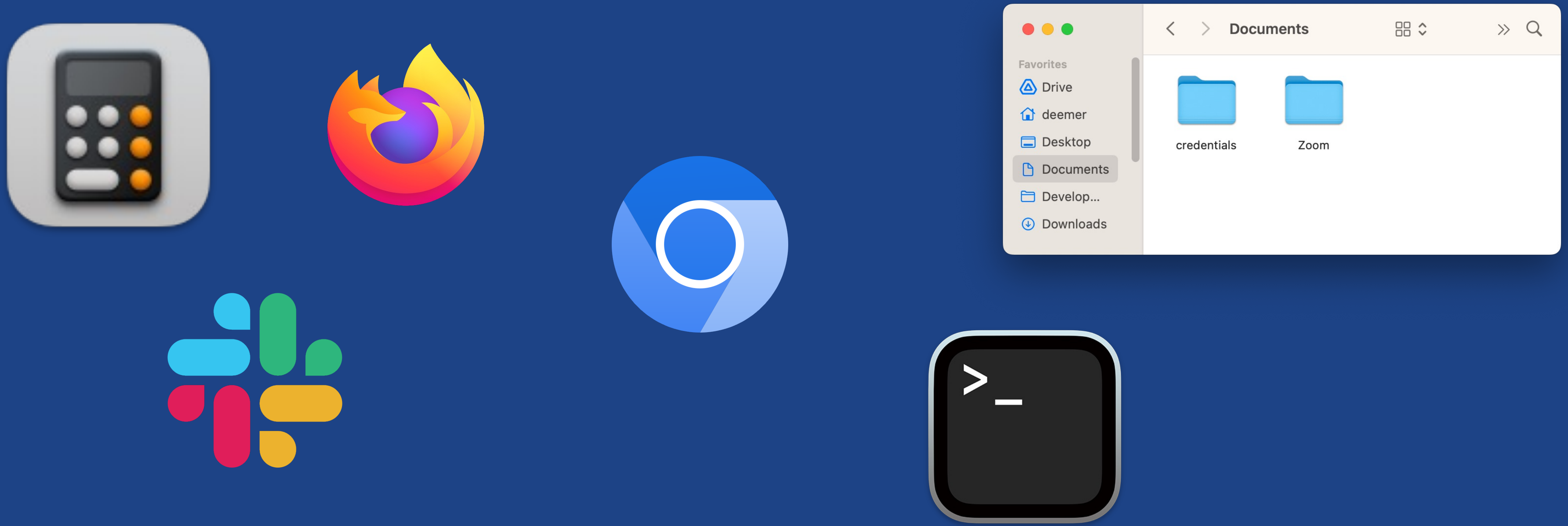
Successful exploitation of this vulnerability allows any unprivileged user to gain root privileges on the vulnerable host. Qualys security researchers have been able to independently verify the vulnerability and develop multiple variants of exploit and obtain full root privileges on Ubuntu 20.04 (Sudo 1.8.31), Debian 10 (Sudo 1.8.27), and Fedora 33 (Sudo 1.9.2). Other operating systems and distributions are also likely to be exploitable.

Taking a step back…

Is this enough?

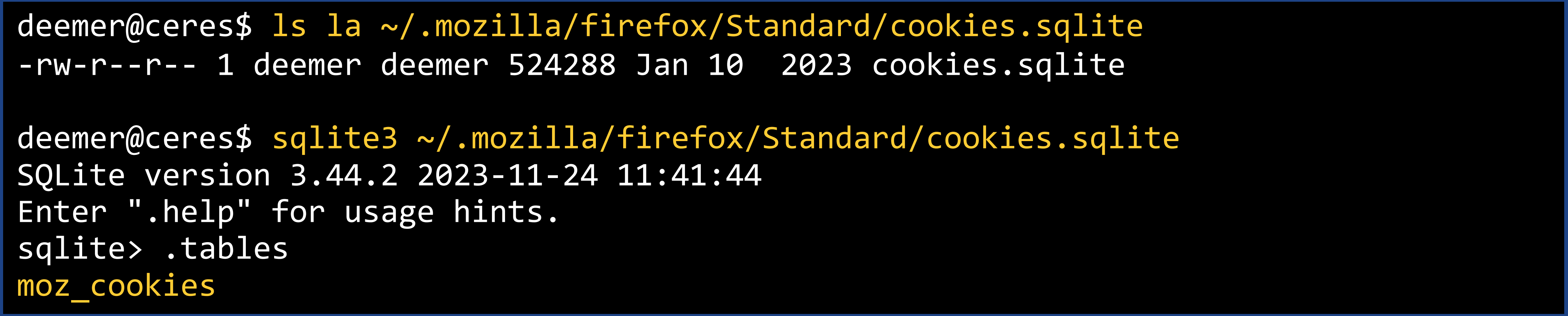
Linux Default: Discretionary Access Control

- Owner of a resource decides on how it can be used
- Privileges depend on current user (and some groups)
- To elevate: admin user (root) vs. other users



=> How many of these can read your browser history?

.... all of them?!?!



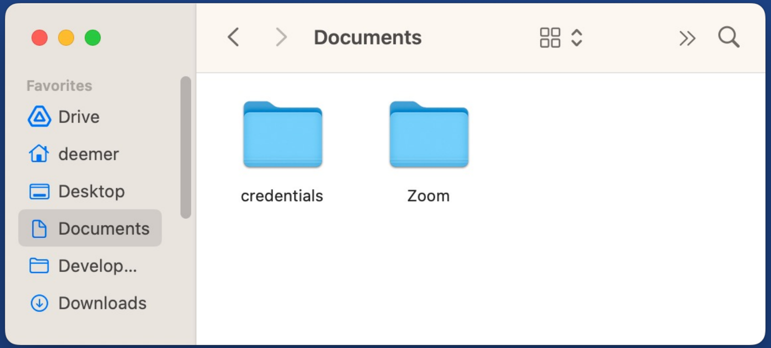
```
deemer@ceres$ ls la ~/.mozilla/firefox/Standard/cookies.sqlite
-rw-r--r-- 1 deemer deemer 524288 Jan 10  2023 cookies.sqlite

deemer@ceres$ sqlite3 ~/.mozilla/firefox/Standard/cookies.sqlite
SQLite version 3.44.2 2023-11-24 11:41:44
Enter ".help" for usage hints.
sqlite> .tables
moz_cookies
```

…. all of them?!?!

```
deemer@ceres$ ls la ~/.mozilla/firefox/Standard/cookies.sqlite
-rw-r--r-- 1 deemer deemer 524288 Jan 10  2023 cookies.sqlite

deemer@ceres$ sqlite3 ~/.mozilla/firefox/Standard/cookies.sqlite
SQLite version 3.44.2 2023-11-24 11:41:44
Enter ".help" for usage hints.
sqlite> .tables
moz_cookies
```

=> Just a syscall! Works as long as permissions check out 😮

```
deemer@ceres:~$ strace -- sqlite3 cookies.sqlite
. . .
access("cookies.sqlite", F_OK)          = 0
openat(AT_FDCWD, "cookies.sqlite", O_RDONLY) = 3
. . .
```



How many of these <u>should</u> be able to read your browser history?

Why?

Why?

- File permissions are very coarse
- Apps might not be trusted
- Apps might get compromised

Why?

- File permissions are very coarse
- Apps might not be trusted
- Apps might get compromised

=> Would like a more secure design: restrict application privileges so they can only access what they need

✨ ✨ ✨ ***Principle of Least Privilege*** ✨ ✨ ✨

An application should only be able to perform the operations necessary for its intended purpose

How? Depends on the context

Affects design of different systems/abstractions

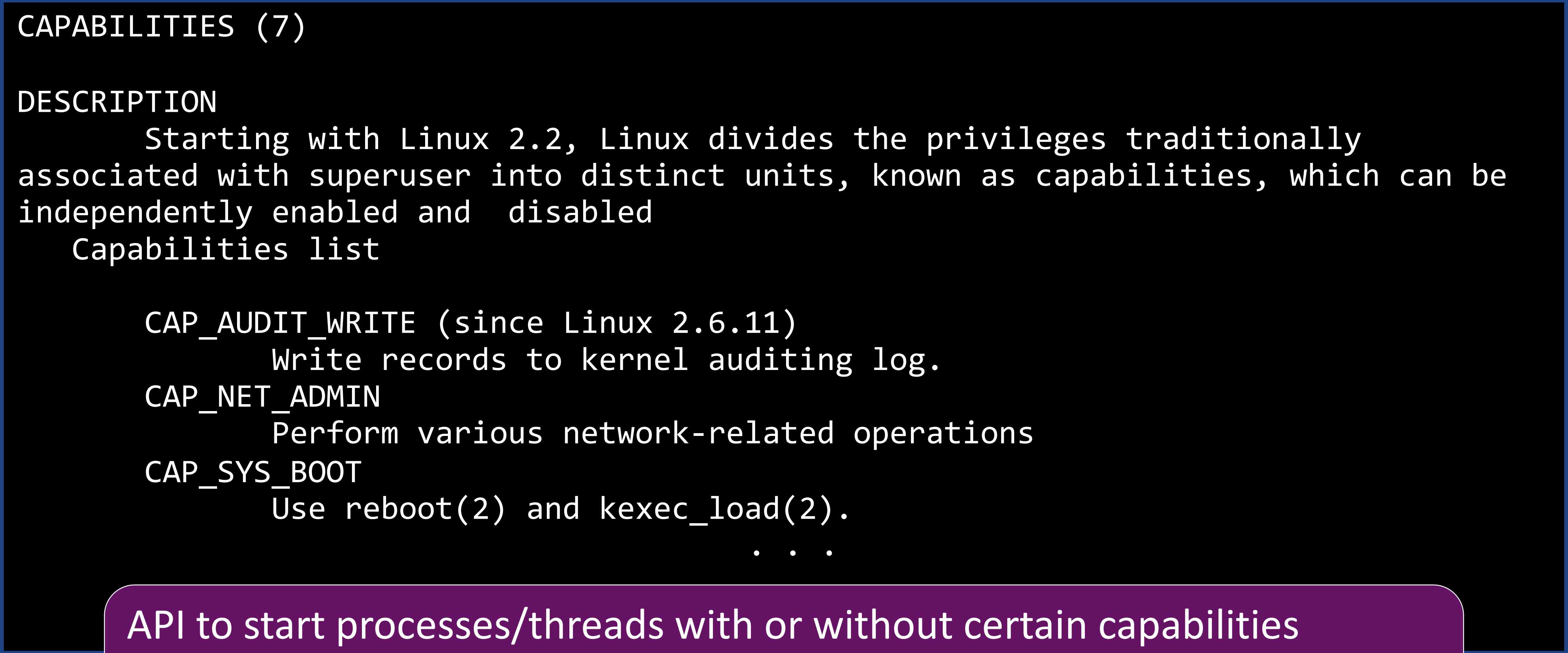
One way: finer-grained permissions

Linux: can we do better than just root vs. non-root?

=> Capabilities: more precise permissions for certain actions, can be bestowed per-process

```
CAPABILITIES (7)

DESCRIPTION
       Starting with Linux 2.2, Linux divides the privileges traditionally
associated with superuser into distinct units, known as capabilities, which can be
independently enabled and  disabled
   Capabilities list

       CAP_AUDIT_WRITE (since Linux 2.6.11)
              Write records to kernel auditing log.
       CAP_NET_ADMIN
              Perform various network-related operations
       CAP_SYS_BOOT
              Use reboot(2) and kexec_load(2).
                                        . . .
```

API to start processes/threads with or without certain capabilities
=> Possible to “drop” permissions for unsafe operations
=> One you drop permissions, process can’t get them back



```
CAPABILITIES (7)

DESCRIPTION
       Starting with Linux 2.2, Linux divides the privileges traditionally
associated with superuser into distinct units, known as capabilities, which can be
independently enabled and  disabled
   Capabilities list

       CAP_AUDIT_WRITE (since Linux 2.6.11)
              Write records to kernel auditing log.
       CAP_NET_ADMIN
              Perform various network-related operations
       CAP_SYS_BOOT
              Use reboot(2) and kexec_load(2).
                                            . . .
```

API to start processes/threads with or without certain capabilities
=> Possible to “drop” permissions for unsafe operations
=> One you drop permissions, process can’t get them back

Examples: webservers, sshd, etc.
=> Servers that operate on untrusted inputs

Another way: Process separation

- System service runs as privileged user
- Client program run by unprivileged users

Separation of processes

- System service runs as privileged user
- Client program run by unprivileged users
- Some API for how these programs communicate
  - Local network connection
  - Unix socket
  - dbus or other IPC mechanism
  - …

One way: Separation of processes

- System service runs as privileged user
- Client program run by unprivileged users
- Some API for how these programs communicate
  - Local network connection
  - Unix socket
  - dbus or other IPC mechanism
  - …

=> Better control over *how* privileged code runs
=> Interface between privileged/unprivileged defined more clearly

Example: docker

```
[root@ceres run]# ls -la /run/docker.sock
srw-rw---- 1 root docker 0 Jan  4 07:26 /run/docker.sock


deemer@ceres$ id
uid=1000(deemer) gid=1000(deemer) groups=1000(deemer),...,966(docker),...
```

Example: docker

```
[root@ceres run]# ls -la /run/docker.sock
srw-rw---- 1 root docker 0 Jan  4 07:26 /run/docker.sock


deemer@ceres$ id
uid=1000(deemer) gid=1000(deemer) groups=1000(deemer),...,966(docker),...
```

```
[root@ceres run]# ps aux | grep docker
root        1417  0.0  0.1 4350944 80252 ?        Ssl  Jan04  87:22
         /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
. . .

deemer    309604  0.0  0.0  12300   512 ?        S+   Feb26   0:00 /bin/bash
         /home/deemer/cs1660/env/run-container
```

One way: Isolation within OS

Linux namespaces (+ related features): give processes/users separate views of userspace components

Example: chroot (1980s)

- "Change root"
- Run command with separate root directory
- All child processes inherit this root directory

Demo: chroot

Example: chroot (1980s)

- "Change root"
- Run command with separate root directory
- All child processes inherit this root directory

- Implications?

If you need to do this in practice: look up "schroot"

One way: Isolation within OS

Linux namespaces (+ related features): give processes/users separate views of userspace components

- chroot (separate filesystem trees)
- Processes trees
- UIDs/GIDs
- cgroups (Resource limits/quotas)
- Network connections
- Time
- . . .

One way: Isolation within OS

Linux namespaces (+ related features): give processes/users separate views of userspace components

- chroot (separate filesystem trees)
- Processes trees
- UIDs/GIDs
- cgroups (Resource limits/quotas)
- Network connections
- Time
- . . .

Not a security feature *per se*, but can help...

Containers (ie, Docker) [ON LINUX]

Automated way to run applications

- Leverages lots of Linux namespaces at once
- Super great for deploying software!!

Example: course container

What do we notice?

- Separate filesystem
- Separate UIDs/GIDs
  - Can be root in the container => does it matter?
- Separate network interfaces, etc.

- When running the container, we decide what resources are shared with the host (files, network, etc)

Isolation mediated by Docker, OS kernel

What does this mean?

- Easy to "punch holes" depending on configuration
  - Shared directories, "privileged containers", ...
- Namespaces are growing all the time
- Docker has lots of permissions levels for what privileges containers can use

A lot of "knobs"...

- What if the configuration is incorrect?
- What if the kernel has a bug?

Problems?

But…

What if the container config is incorrect?

What if the kernel has a bug?

What if you don’t trust the software you’re running?

Another way: Virtual Machines (VMs)

Isolated way to run an entire system (hardware, kernel, ...)

Another way: Virtual Machines (VMs)

Isolated way to run an entire system (hardware, kernel, ...)

- A whole OS could run as a program
- Modern systems: hardware support for isolating memory, page tables, etc. and preserving performance
  - Curious? Take CS1670.
- Virtual hardware/drivers to interact with host

Another way: Virtual Machines (VMs)

Isolated way to run an entire system (hardware, kernel, ...)

- A whole OS could run as a program
- Modern systems: hardware support for isolating memory, page tables, etc. and preserving performance
  - Curious? Take CS1670.
- Virtual hardware/drivers to interact with host

=> "Stronger" isolation, possibly more overhead for configuration/performance vs. containers

Example: A VM

So where should we run our untrusted code?

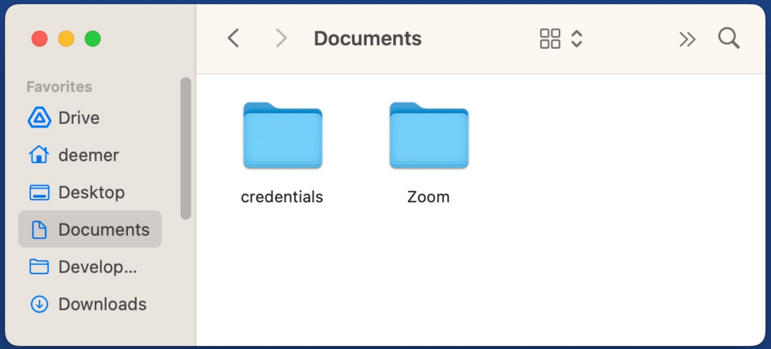
- Functionality: What privileges should the code (or the user) have?
- Threat model: What are the attacker's capabilities?

Docker on Windows, Mac?

Windows/Mac don't have Linux namespaces...

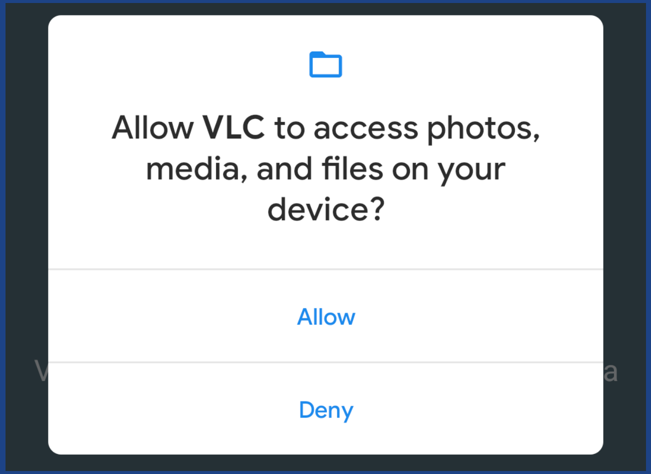
Comparing isolation mechanisms

Mechanism	"Interface" to privileged operations
setuid/setgid application	Application code
Process isolation (client/server process)	API between client program and service (network protocol, socket file, IPC calls, ...)
Container	OS kernel (+ any host features turned on by container author)
VM	Virtualization Platform (hypervisor, virtual device drivers, shared folders, ...)

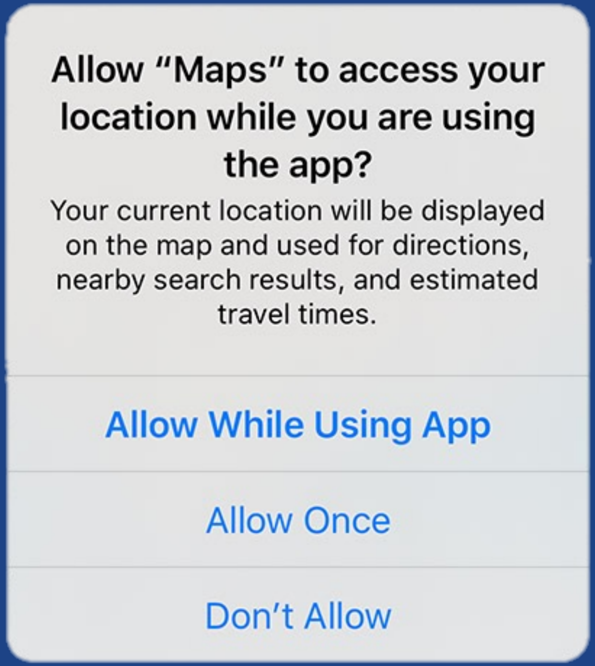


How many of these should be able to read your browser history?

```
access("cookies.sqlite", F_OK)                = 0
openat(AT_FDCWD, "cookies.sqlite", O_RDONLY) = 3
```



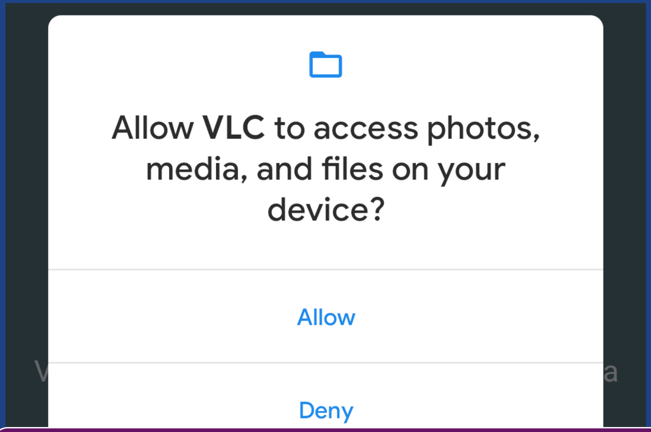
Allow **VLC** to access photos, media, and files on your device?



Allow “Maps” to access your location while you are using the app?

Your current location will be displayed on the map and used for directions, nearby search results, and estimated travel times.

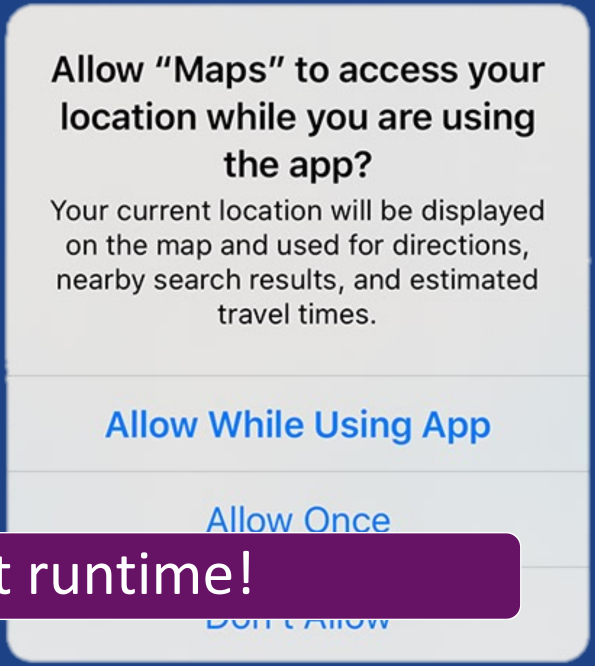
```
access("cookies.sqlite", F_OK)               = 0
openat(AT_FDCWD, "cookies.sqlite", O_RDONLY) = 3
```



Allow **VLC** to access photos, media, and files on your device?

Allow

Deny



Allow “Maps” to access your location while you are using the app?

Your current location will be displayed on the map and used for directions, nearby search results, and estimated travel times.

Allow While Using App

Allow Once

Don't Allow

=> Fine-grained permissions at runtime!

…at compile time?

Other ways?

- What does it mean for the user to be "unprivileged"?
- What does it mean for code run by a user to be "unprivileged"?

- What do we want that code to be able to do?

  => How much do we trust the user? The code?

Other ways?

- What does it mean for the user to be "unprivileged"?
- What does it mean for code run by a user to be "unprivileged"?

- What do we want that code to be able to do?

  => How much do we trust the user? The code?

- sudo is pretty coarse-grained…