The forgotten art of filesytem magic

Gergely Kalman
Alligatorcon, 2024

REALLY STUPID
RS
CONTENT RATED BY
ESRB

Who am I

- my name is **Gergely** (**call me Greg**)
- did a bunch of stuff
  - no one cares
- currently
  - Full Time hunter in the **Apple Security Bounty** (**ASB**)
- I have no affiliation with anyone

Intro

- we will attack **file operations** and **filesystems**
- more info:
    - **bottom right corner**
        - **https://gergelykalman.com** →
        - or my OBTS v6 presentation
- twitter: **@gergely_kalman**

A “quick” riddle

- on an **HFS+** volume on **macOS**
- in a directory called **/Volume/ours** owned by the **attacker** user
- we can trigger a **file creation**
  - by a system daemon running as **root**
  - **/Volume/ours**/**secret** can be created as **root:wheel**, perms **“rwx------”**
    - a **POSIX “read” extended ACL** will be created for **attacker**
    - and an **extended attribute** called “**com.apple.quarantine**” will be placed by the system
  - content will be written to the file by the daemon
- **Question**: can **attacker** read the contents of “**secret**”?

If you think **“How the *@!# should I know?”**

You are not alone

The question can’t be answered.

Why not?

- long answer:
  - https://gergelykalman.com/the-missing-guide-to-the-security-of-filesystems-and-file-apis.html

The “short” answer

- short answer:
  - users can mount
  - mountpoints can move
  - permissions are insanely complicated
  - file operations are racy
  - tons of magic at every level

File ops are shockingly hard

- **there are many layers that take care of access control**
  - their possible interactions are often exponential
- **lots of magic**
  - **OS magic: SIP** → policies are hidden
  - **FS magic: FS** attributes → can override decisions
- **path resolution is really unintuitive**
- **the VFS and FS drivers can have surprising bugs/features**
- **filesystems are racy**
  - pretty much everything can be turned into a race condition
    - **provided that you have control** - this is very important

Lesson learned

- **it’s impossible to secure file ops in attacker-controlled locations**
    - ie: if attacker controls a path fully or partially
        - **one path component might be enough** if you can symlink
    - The **only** secure way to handle file operations is to do it in a **completely separate silo**
        - not always feasible
            - think /tmp/, IPC sockets, etc…

Good bugs are hard to find

- it’s rare to find easy FS bugs, **since they are taken seriously**
    - ex: arbitrary file **rename()**
    - ex: arbitrary file write
    - ex: arbitrary **chmod()/chown()**
- these are **obviously bad**

“Garbage” bugs are everywhere

- **devs don’t care about “garbage” bugs**
  - how bad is an arbitrary **unlink()**?
  - if a bug can’t be exploited, it won’t get fixed
    - and “can be exploited” usually means:
      - **is there a widely-known (easy) way to exploitation?**
        - **NO** → not a security issue
  - from this it follows that:
    - **bugs without widely-known security implications won’t get patched**

Let me say that again

Bugs without widely-known security implications won’t get patched.

Let’s go dumpster diving!

- **every system is full of bugs that were deemed unexploitable**
  - “**deemed**” is the key here
  - you win if you know more than the devs
    - which is not difficult since this is pretty obscure stuff

One of my “garbage” bugs

- **root daemon in /.../test/**
  - creates file “./a” without following any symlinks
  - calls rename(“./a”, “./b”)
  - attacker owns **/.../test**
- **is this secure?**
  - **show of hands**

One of my “garbage” bugs

- is this secure? - **NO!**
- this can be used to rename a file from “a” to “b”
  - but that’s pretty useless
- **this is a typical “garbage” bug**
- **is there more to this?**

The rename() trick

- **is there more to this? – YES!**
- don’t let the “.” deceive you:
  - **rename**(“./a”, “./b”) → **rename**(“/tmp/test/a”, “/tmp/test/b”)
- in **rename(src, dst) src** and **dst** are looked up **separately**
  - the path lookup doesn’t (can’t) know that the files are in the same directory

The rename() trick

- **rename**(“/tmp/test/a”, “/tmp/test/b”) will run:
  - **GET_PATH(“/tmp/test/a”)**
  - **GET_PATH(“/tmp/test/b”)**
- **this is a race condition!**
  - if I can **switch out “test” to be a symlink**
    - **after** the **src** lookup
    - but **before** the **dst** lookup
  - I can end up moving the file to **anywhere**/**b**
    - write a file called “**b**” anywhere, with arbitrary contents...

The rename() trick

- **This works both on macOS and Linux**
    - I didn’t know about this
    - nobody I asked did either
- So this just became really interesting

The rename() trick

- How did I find it?
  - I was browsing the **xnu** (**macOS**) kernel source for unrelated reasons and some weird logic stood out to me
    - But this is the exception
- Usually I just **write dumb tests**
  - I write code that **tries hard** to do **obviously stupid things**
    - mimic the conditions the best I can
  - **5% of my dumb tests succeed in doing the obviously dumb thing**
    - so I either learn something new
    - or I find a really cool new trick

Back to my “garbage” bug

- in this case, I have uncovered something really cool
- **this is a brand new vector to exploiting rename()s**
    - it’s **everywhere**
    - and it **can’t be fixed easily**
        - **POSIX** is not at fault
        - **rename()** works as intended
        - so the real culprit is the **userspace program**

Back to my “garbage” bug

- it shouldn’t have been **writing to** a location **where others can write**
  - which is easy to say now
  - but until I investigated it I also thought it was fine:
    - **fishy, but seems okay**
      - symlinks weren’t followed in **open()**
      - I had no control over the destination file path or file name
      - **at best** this would be an overwrite of the fixed file path
        - which was pretty useless
    - **probably this is what the developers thought**

Back to my “garbage” bug

- in the end I exploited the bug with **sudo**
    - dumped a file in **/etc/sudoers.d/**
- **sudo is great**
    - filename doesn’t matter in **/etc/sudoers.d/**
    - only permissions are checked
    - sudo is okay with binary garbage → partial content control is enough

This is cutting edge in 2024

Which is absolutely insane

POSIX has been around for 40 years...

The bugs

1) **librarian** (CVE-2023-38571) - **TCC bypass**
2) **lateralus** (CVE-2023-32407) - **TCC bypass**
3) **sqlol** (CVE-2023-32422) - **TCC bypass**
4) **batsignal** (no CVE) - **LPE**
5) **alfred** (CVE-2023-40443) - **LPE**
6) **badmalloc** (CVE-2023-32428) - **LPE**
7) **jetson** (CVE-2023-41986) - **TCC bypass**

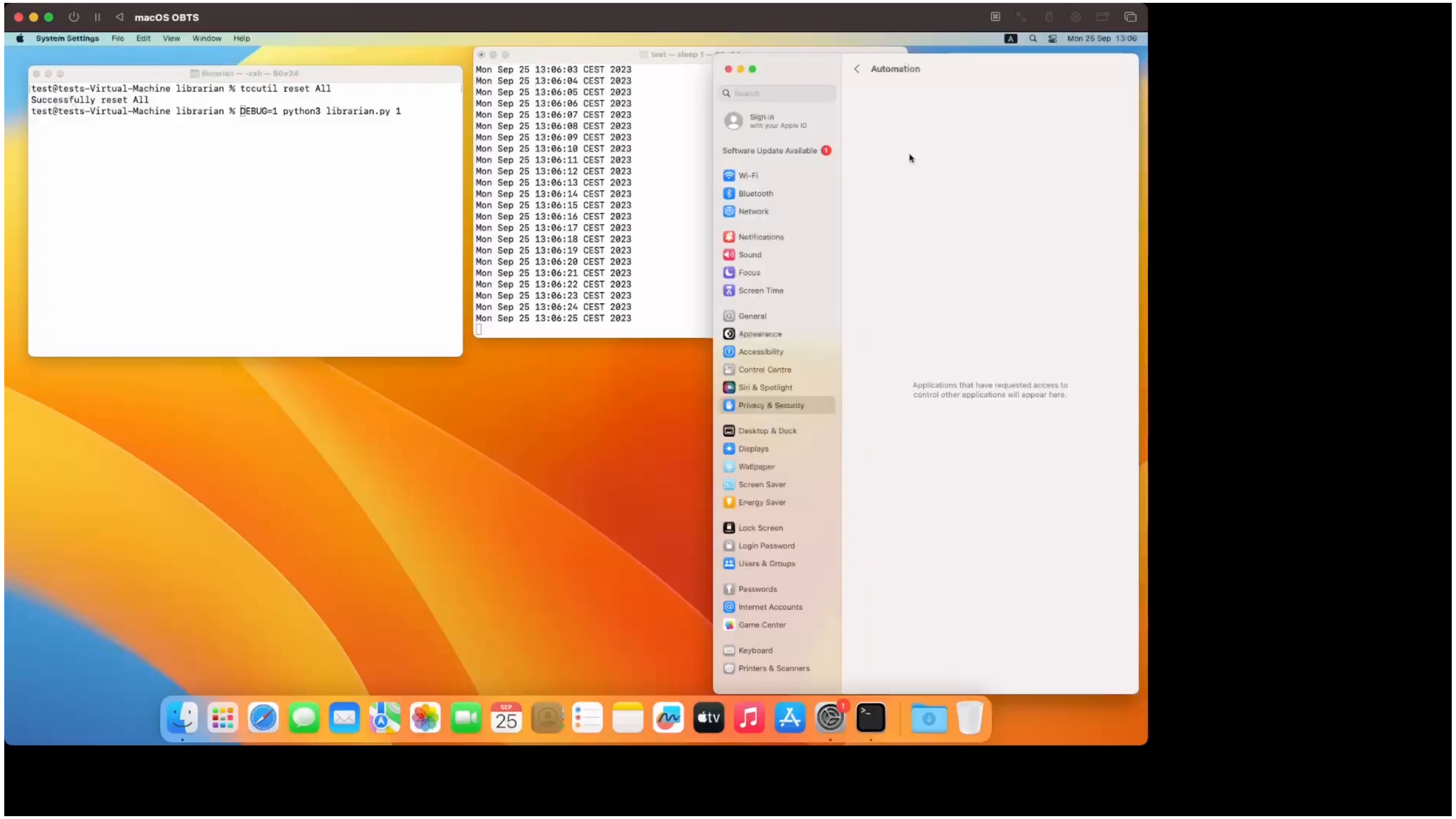
Bugs: #1 Librarian

- **rename()** bug in **Music** on **macOS**
  - any file dumped here:
    - ~/Music/Music/Media.localized/Automatically Add to Music.localized/myfile.mp3
  - will be moved here:
    - "~/Music/Music/Media.localized/Automatically Add to Music.localized/Not Added.localized/2023-09-25 11.06.28/myfile.mp3
  - **a best-case rename() bug**
    - **dst** filename is fully controlled
    - **src** is fully controlled

Bugs: #1 Librarian

- for a successful exploit:
  - we have to replace the date directory with a symlink
    - **a really easy race, no tricks necessary**
- what did we get?
  - **Music** (at that time) had **FDA** access
    - so we could use this to overwrite the user's **TCC.db**
      - which grants us access to all **TCC**-protected data
        - aka a "**FULL TCC bypass**"

Bugs: #1 Librarian



```
test@tests-Virtual-Machine librarian % tccutil reset All
Successfully reset All
test@tests-Virtual-Machine librarian % DEBUG=1 python3 librarian.py 1
```

```
Mon Sep 25 13:06:03 CEST 2023
Mon Sep 25 13:06:04 CEST 2023
Mon Sep 25 13:06:05 CEST 2023
Mon Sep 25 13:06:06 CEST 2023
Mon Sep 25 13:06:07 CEST 2023
Mon Sep 25 13:06:08 CEST 2023
Mon Sep 25 13:06:09 CEST 2023
Mon Sep 25 13:06:10 CEST 2023
Mon Sep 25 13:06:11 CEST 2023
Mon Sep 25 13:06:12 CEST 2023
Mon Sep 25 13:06:13 CEST 2023
Mon Sep 25 13:06:14 CEST 2023
Mon Sep 25 13:06:15 CEST 2023
Mon Sep 25 13:06:16 CEST 2023
Mon Sep 25 13:06:17 CEST 2023
Mon Sep 25 13:06:18 CEST 2023
Mon Sep 25 13:06:19 CEST 2023
Mon Sep 25 13:06:20 CEST 2023
Mon Sep 25 13:06:21 CEST 2023
Mon Sep 25 13:06:22 CEST 2023
Mon Sep 25 13:06:23 CEST 2023
Mon Sep 25 13:06:24 CEST 2023
Mon Sep 25 13:06:25 CEST 2023
```

Automation

Applications that have requested access to control other applications will appear here.

Bugs: #1 Librarian

- a FULL TCC bypass on macOS is worth $30,500

Bugs: #2 Lateralus

- insecure file write in the **Metal** library
  - used by **Music** (among others)
- triggered from an env var:
  - **MTL_DUMP_PIPELINES_TO_JSON_FILE = ”path/name”**
- tempfile creation + rename
  - using **createFileAtPath**

Bugs: #2 Lateralus

- **createFileAtPath**:
    - in **Foundation** framework’s **NSFileManager**
    - **this is THE core Apple framework** everyone relies on
        - **so it should be secure, right?**

Bugs: #2 Lateralus

- **createFileAtPath("path/name", ...):**
    - **open()** creates temp file:
      "path/.dat.nosyncXXXX.XXXXXX" (**X** is random)
    - **write()**s the contents
    - calls **rename("path/.dat.nosyncXXXX.XXXXXX", "path/name")**
- **what do you think: how secure is this?**

Bugs: #2 Lateralus

this is an arbitrary file overwrite primitive

Bugs: #2 Lateralus

this is an arbitrary file overwrite primitive

(in a core system framework)

Bugs: #2 Lateralus

- **at first glance content control seems** impossible

```
MTL_DUMP_PIPELINES_TO_JSON_FILE = ”path/name”
open(“path/.dat.nosyncXXXX.XXXXXX”, ...)
rename(“path/.dat.nosyncXXXX.XXXXXX”, “path/name”)
```

Bugs: #2 Lateralus

- but we **know rename()** is racy:
    - we can have the **tempfile** dumped anywhere we can write
    - then race the “in-place” **rename()** to change the **dst** path

```
MTL_DUMP_PIPELINES_TO_JSON_FILE = ”path/name”
open(“path/.dat.nosyncXXXX.XXXXXX”, ...)
rename(“path/.dat.nosyncXXXX.XXXXXX”, “path/name”)
```

Bugs: #2 Lateralus

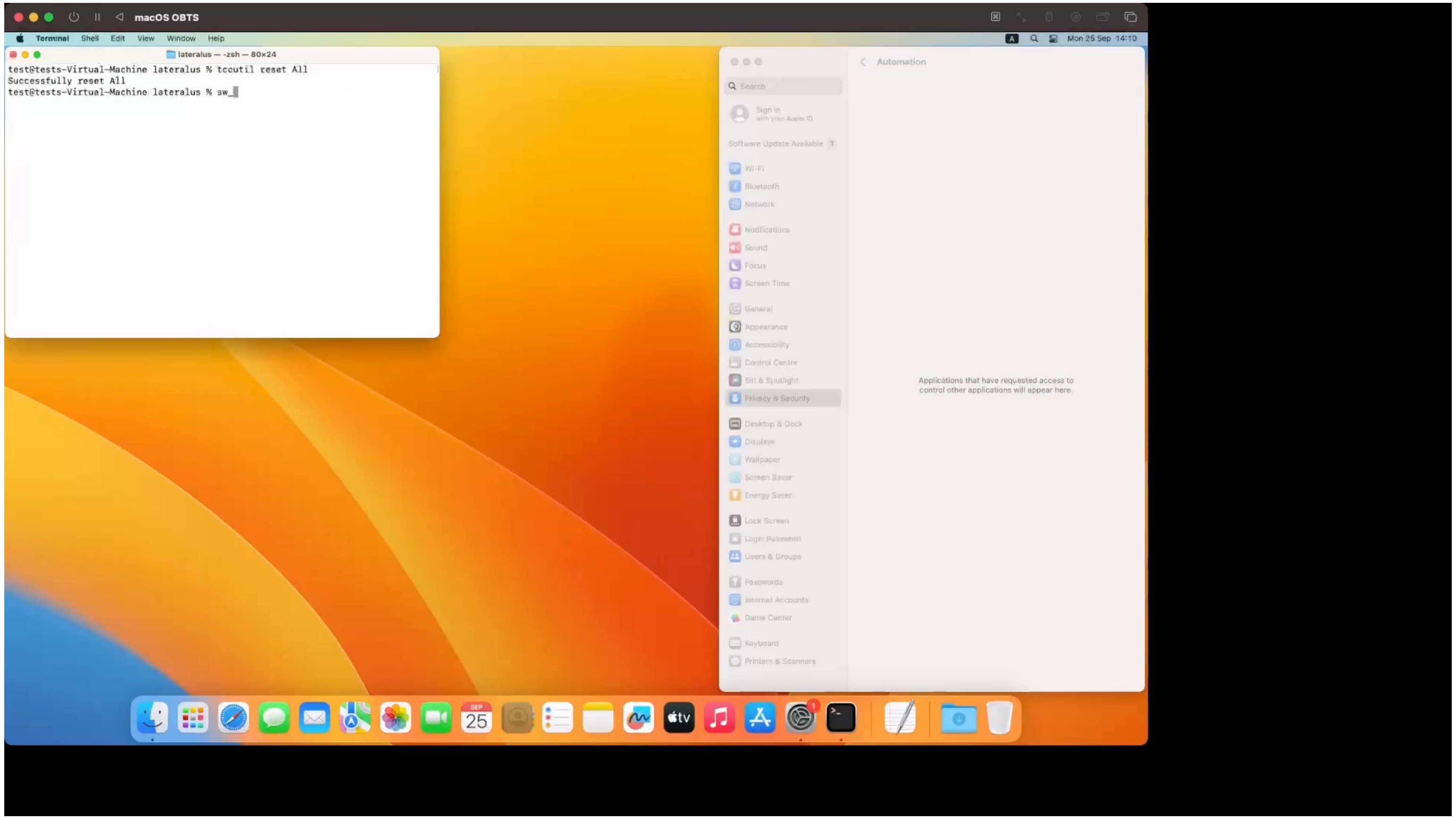
- **did you get that?**
- there are three path lookups:
  - 1) for **open()**
  - 2) for **src** in **rename()**
  - 3) for **dst** in **rename()**
- **we need to win two races:**
  - between **open()** and **rename()** to switch the temp file
  - between **src** and **dst** in **rename()** to switch the **path** dir

```
open("path/.dat.nosyncXXXX.XXXXXX", ...)
rename("path/.dat.nosyncXXXX.XXXXXX", "path/name")
```

Bugs: #2 Lateralus

- what do we get?
    - **full control over path and contents**
- **if rename() worked as most people assume, this won’t be too bad**
    - we could create a new file anywhere, without content control
        - typical “informational” garbage bug
- **instead of that**
    - I overwrote **TCC.db** for another **$30,500**

Bugs: #2 Lateralus



```
test@tests-Virtual-Machine lateralus % tccutil reset All
Successfully reset All
test@tests-Virtual-Machine lateralus % sw_
```

Applications that have requested access to control other applications will appear here.

Bugs: #3 sqlol

- **libsqlite** was compiled with **debug ON** on **macOS**
  - **SQLITE_SQLLOG_DIR** = "**whatever**" means:
    - **copy every opened sqlite DB to “whatever”**
    - and write a query log and index file

Bugs: #3 sqlol

- **libsqlite** was compiled with **debug ON** on **macOS**
  - **SQLITE_SQLLOG_DIR** = "**whatever**" means:
    - **copy every opened sqlite DB to "whatever"**
    - and write a query log and index file

Bugs: #3 sqlol

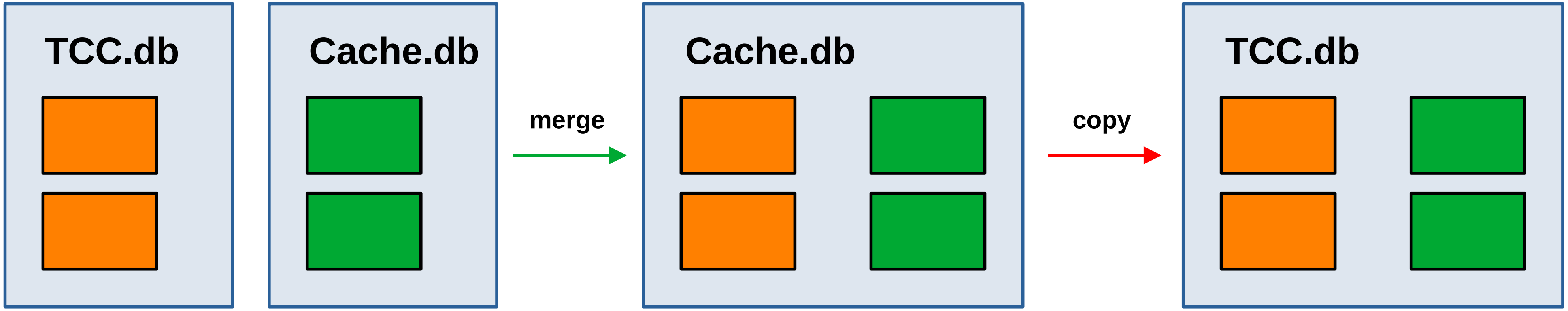
- **libsqlite** is used by apps with **TCC-bypass privileges**
  - **Music** uses it (among many others)
- **already a horrible infoleak**
  - but can we do even more?

Bugs: #3 sqlol

- good:
  - **open() follows all symlinks**
  - **open() overwrites existing files**
- bad:
  - **lack of content control**
  - I can overwrite files, but only with:
    - **sqlite DB, statement log, index file**

Bugs: #3 sqlol

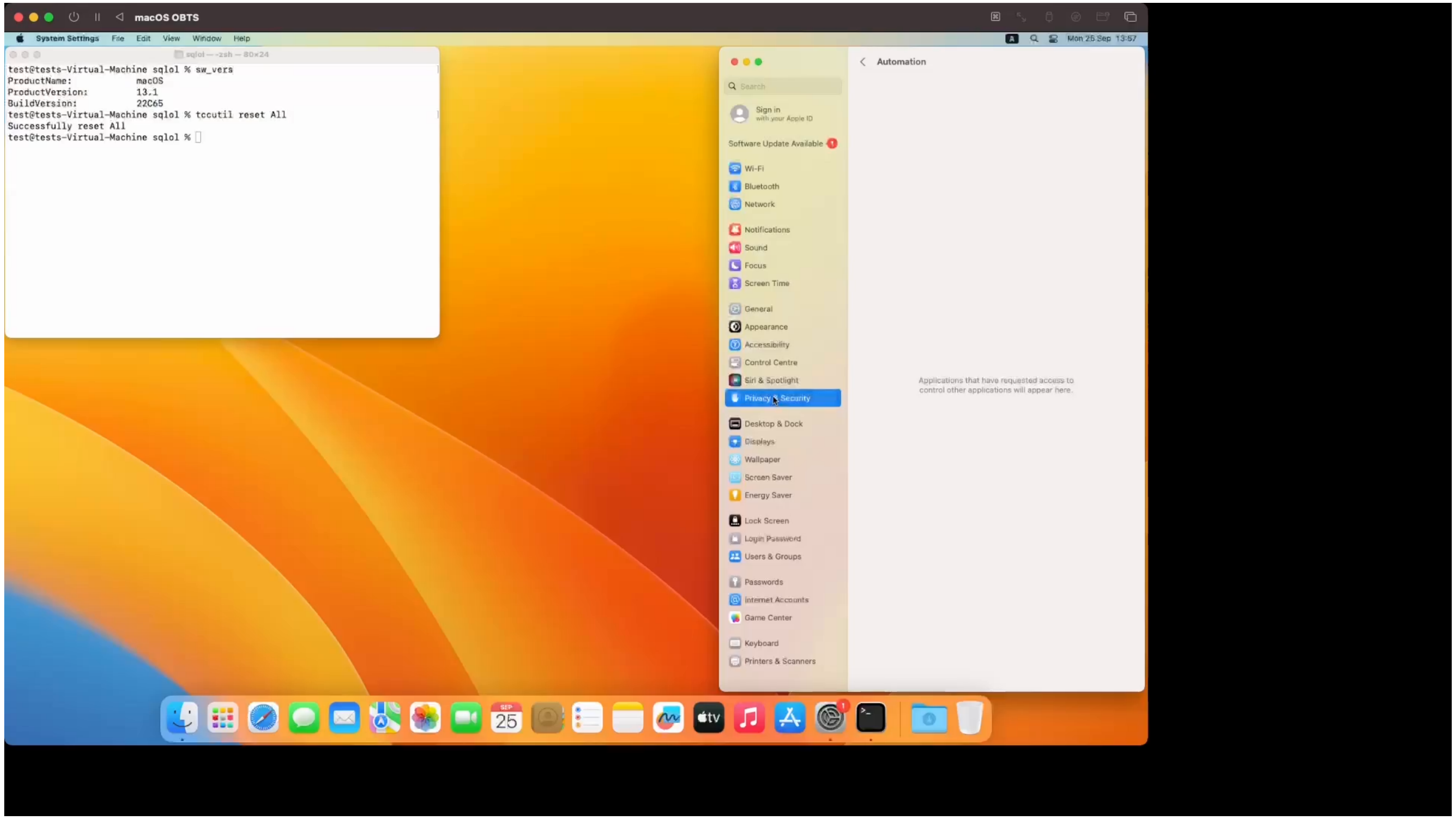
- the big ideas:
  - I can write to some of the source **DBs**
  - sqlite supports multiple tables
  - **TCC.db** is an sqlite DB



Bugs: #3 sqlol

- **sqlite “table smuggling”:**
  - put valid **TCC.db** tables into **Music**’s **Cache.db**
  - upon opening it, **libsqlite** will make a copy
  - with a predictable name, following symlinks
    - → overwrite the real **TCC.db** with one that has our data
  - we don’t even need to race
- **this cost Apple yet another $30,500**

Bugs: #3 sqlol



```
test@tests-Virtual-Machine sqlol % sw_vers
ProductName:            macOS
ProductVersion:         13.1
BuildVersion:           22C65
test@tests-Virtual-Machine sqlol % tccutil reset All
Successfully reset All
test@tests-Virtual-Machine sqlol % 
```

Applications that have requested access to control other applications will appear here.

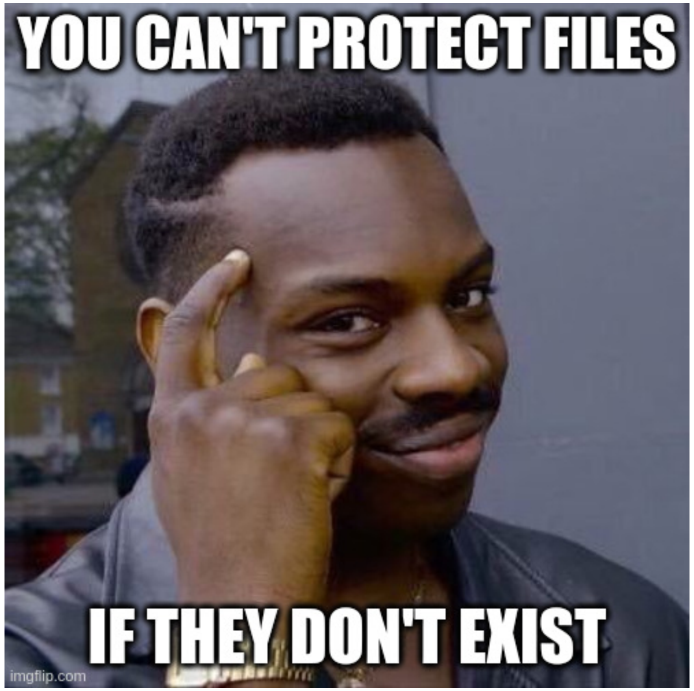
For my next trick...

- let’s stop bullying **Music** for a second
- and see if we can get **root**

Bugs: #4 batsignal

- **Spotlight** performs file operations on **user-mounted volumes**
  - **a truly horrendous idea...**
- two root daemons: **mds**, **mds_stores**
- they operate in a **SIP-protected** directory on the volume:
  - “/mntpoint**/.Spotlight-V100/**”
- **SIP** will use **this** as a regular expression to block access

Bugs: #4 batsignal v1

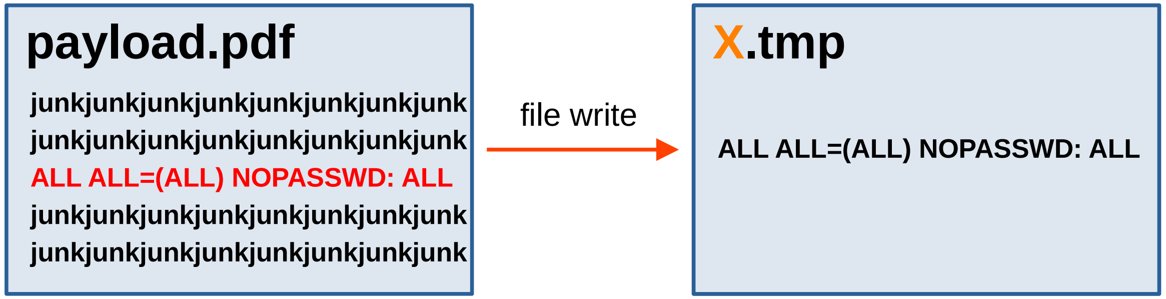


Bugs: #4 batsignal v1

- **umount** the disk and **edit it offline**
  - changing a directory name is easy in **HFS+** :)
    - buf.replace(b’\x31\x00\x30\x00\x30\x00’, b'\x39\x00\x30\x00\x30\x00’)
    - **.Spotlight-V100 → .Spotlight-V900**
- **this allows me to booby trap this directory**

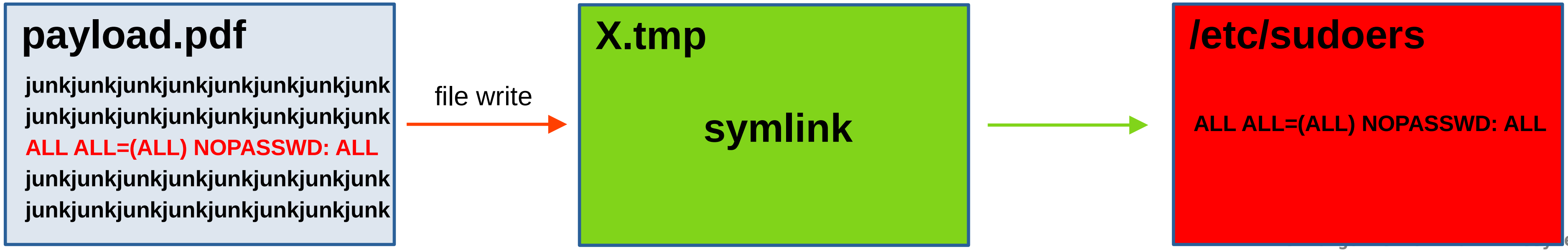
Bugs: #4 batsignal v1

- one of the (**many**) bugs:
  - **Spotlight** writes cache files with **open()**
    - **follows symlinks, overwrites files** with truncation
  - the cache file has:
    - **attacker-controlled content**
    - a **known filename** → **X.tmp**, where **X** is the inode number


payload.pdf
junkjunkjunkjunkjunkjunkjunkjunk
junkjunkjunkjunkjunkjunkjunkjunk
ALL ALL=(ALL) NOPASSWD: ALL
junkjunkjunkjunkjunkjunkjunkjunk
junkjunkjunkjunkjunkjunkjunkjunk
file write
X.tmp
ALL ALL=(ALL) NOPASSWD: ALL

Bugs: #4 batsignal v1

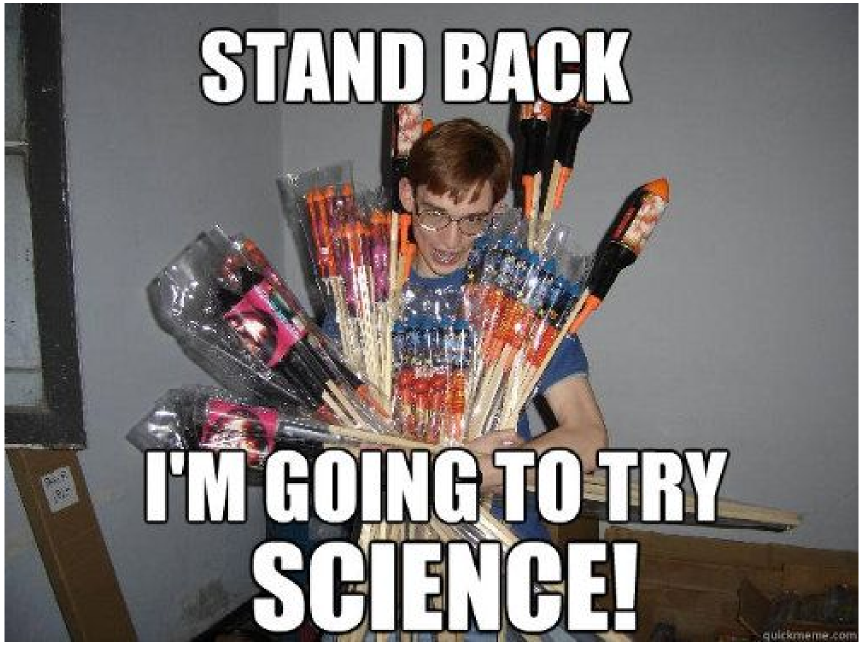
- **to exploit:**
  - symlink the desired cache file to **/etc/sudoers**
  - place a pdf with arbitrary text on the volume
- content from the pdf will be written to **/etc/sudoers**


payload.pdf
junkjunkjunkjunkjunkjunkjunkjunk
junkjunkjunkjunkjunkjunkjunkjunk
ALL ALL=(ALL) NOPASSWD: ALL
junkjunkjunkjunkjunkjunkjunkjunk
junkjunkjunkjunkjunkjunkjunkjunk
file write
X.tmp
symlink
/etc/sudoers
ALL ALL=(ALL) NOPASSWD: ALL

Bugs: #4 batsignal v1

- bounty: $0 (collision)
- the fix: Spotlight no longer likes symlinks :(

Bugs: #4 batsignal v2



Bugs: #4 batsignal v2

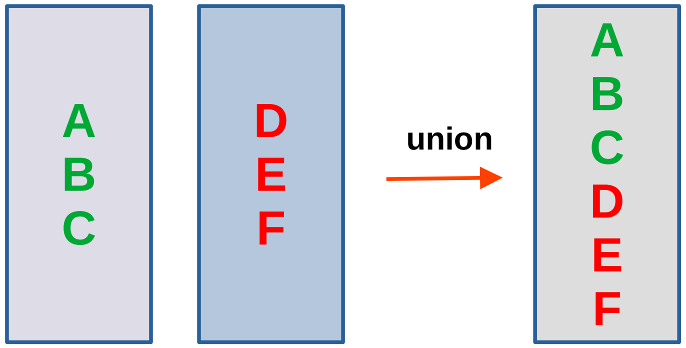
- we can’t use symlinks now :(
- the **/.Spotlight-V100** protection is **still a regex match**
    - it’s not filesystem-aware
    - no idea about **mountpoints**, **symlinks**, etc…
- **hardlinks** are sort of like **symlinks**
    - if they’re on the same volume
- can we **“merge”** two different volumes somehow?

Bugs: #4 batsignal v2

- **Yes: we can use unions!**
- **union 101**:
  - **top** and **bottom** filesystems
  - mounted over each other
  - lookups start in **top**
  - if name is missing
    - fall back to **bottom**

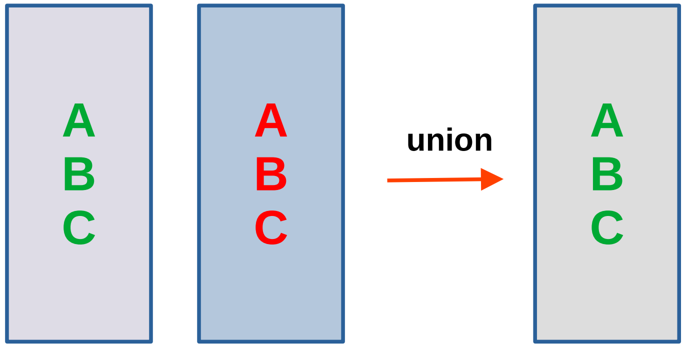
Bugs: #4 batsignal v2

- **unions:**



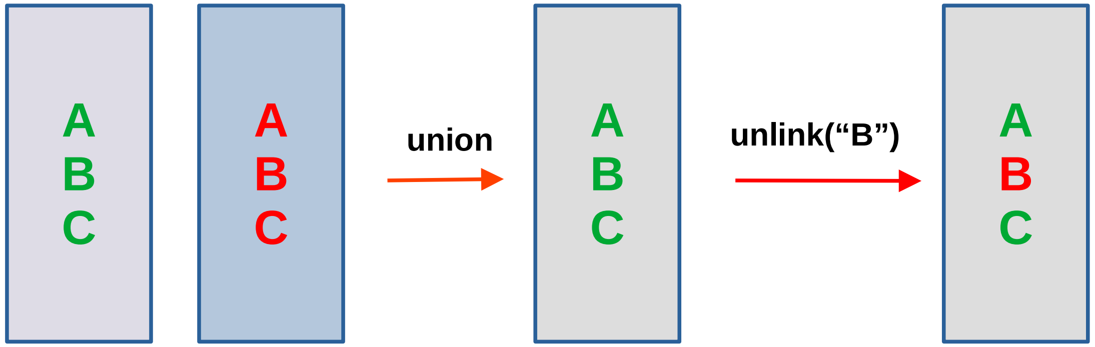
Bugs: #4 batsignal v2

- **unions:**



Bugs: #4 batsignal v2

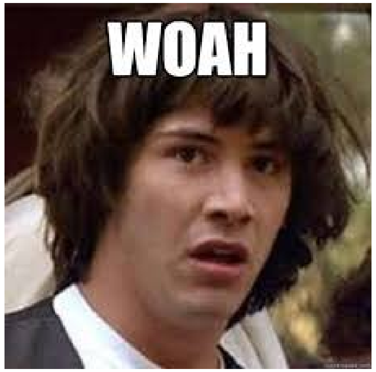
- unions:



Bugs: #4 batsignal v2

- **the file write bug is exploitable again**
    - mirror Spotlight’s directories on the system disk (**bottom**)
    - mount the volume over it with union (**top**)
    - delete the target file from **top,** so it’s used from **bottom**
    - where it’s hardlinked to **/etc/sudoers**

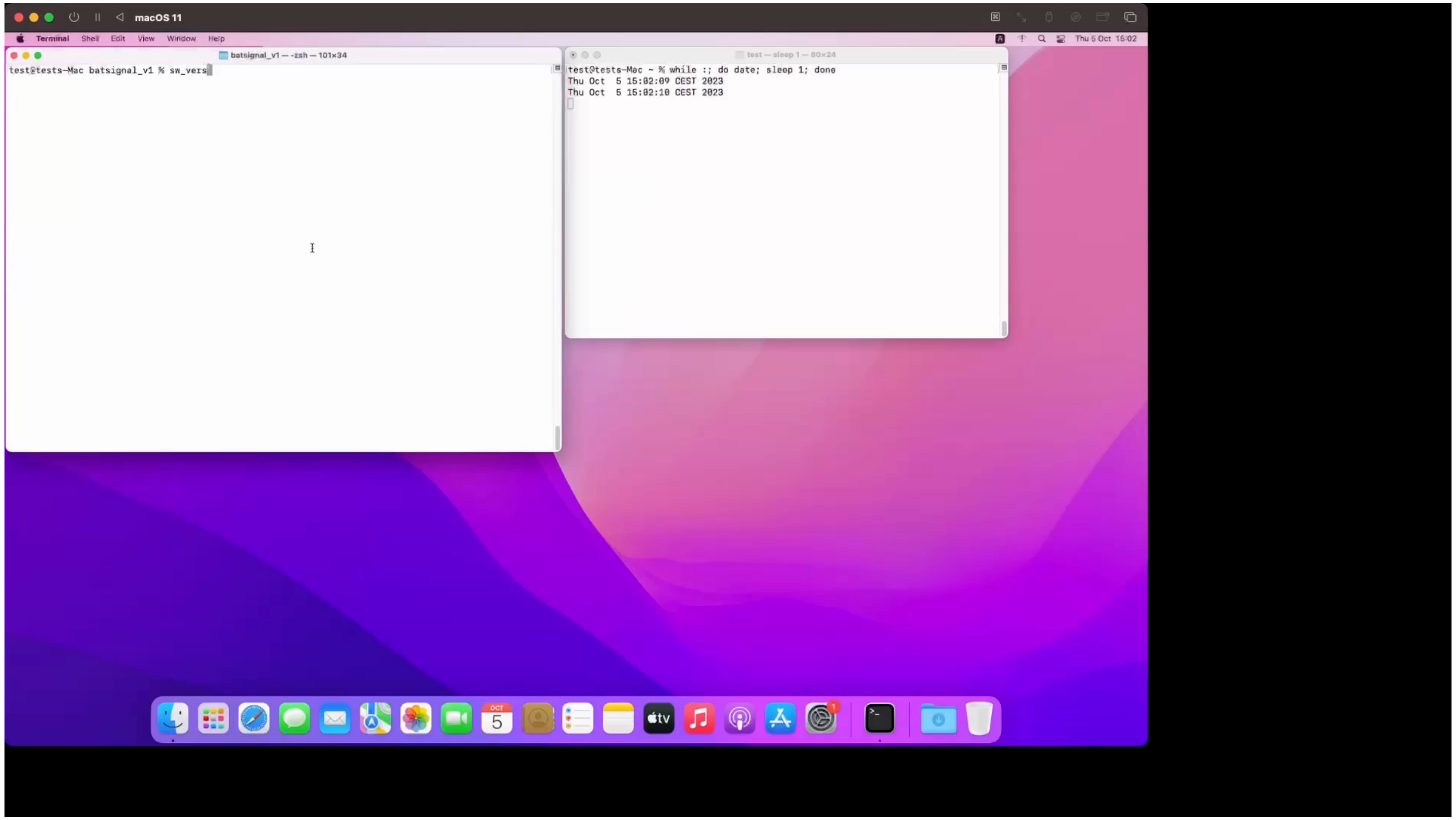
This is how you symlink without symlinks!



Bugs: #4 batsignal v2

- **Apple finally paid me $17,000**
  - after more than a year, without any credit
- **Lessons learned:**
  - **Allowing users to mount their own images is absolutely insane**
  - **Apple does pay for bugs**
- **the fix: Apple disallows unions for Spotlight**

Bugs: #4 batsignal v1


macOS 11
Terminal Shell Edit View Window Help
Thu 5 Oct 15:02
batsignal_v1 — -zsh — 101×34
test — sleep 1 — 80×24

```
test@tests-Mac batsignal_v1 % sw_vers
```

```
test@tests-Mac ~ % while :; do date; sleep 1; done
Thu Oct  5 15:02:09 CEST 2023
Thu Oct  5 15:02:10 CEST 2023
```

Bugs: #5 alfred

- **alfred** is actually **batsignal v3**
- to recap:
  - **Spotlight** does insecure writes on user-provided volumes
  - we can’t use **symlinks**
  - we can’t use **union-mounts :(**
- **Surely we are done, right?**

Bugs: #5 alfred

- **Nope, we can just move the mountpoint**
- rugpull **mds** → write to system volume
- **Apple did a good job** of restricting **mds**, except:
  - `(regex #"^/private/var/folders/[^/]+/[^/]+/C/com.apple.metadata.mdworker($|/)")`
  - `(regex #"^/private/var/folders/[^/]+/[^/]+/T/com.apple.metadata.mdworker($|/)")`
    - **"/var/folders/RANDOM/RANDOM/T/com.apple.metadata.mdworker/"**
      - doesn't exist, but can be created by us
      - is not protected by **SIP**
      - we'll call this **tmpdir**

Bugs: #5 alfred

- to exploit this:
  - prepare the **Spotlight** directories in **tmpdir**
  - create a hardlink to **/etc/sudoers** in **tmpdir**
  - **swap the mountpoint with tmpdir in a loop**
- when the race is won:
  - **mds** will operate under **tmpdir**
    - this is allowed in the policy
- **but how do we control the content?**

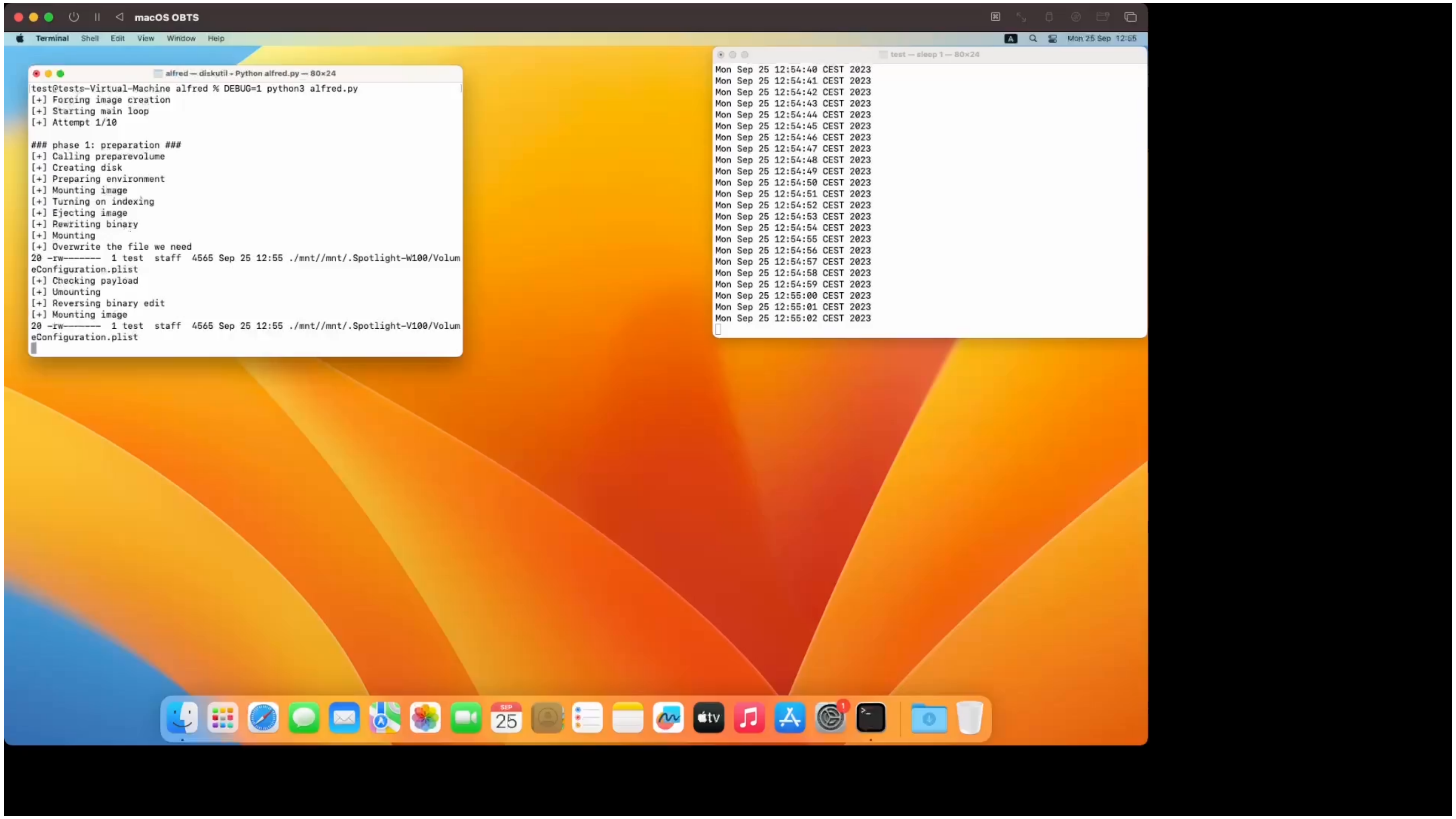
Bugs: #5 alfred

- many bugs to pick from
- so I looked for a file create that
  - we can smuggle our payload into
  - recreates a file
- didn’t have to look too long:
  - **VolumeConfiguration.plist** contains configuration options for Spotlight
    - has **user-provided file exclusion paths**
    - **gets re-created** with the contents it remembered

Bugs: #5 alfred

- we can **partially control data** now
  - which works well for **sudo**
    - reads /etc/sudoers.d/*
    - parses what it can
    - ignores everything else
- with a valid sudo entry we **escalate to root**
- **Apple** paid **$22,500** for this one

Bugs: #5 alfred


macOS OBTS
Terminal Shell Edit View Window Help
Mon 25 Sep 12:55
alfred — diskutil ◂ Python alfred.py — 80×24
test — sleep 1 — 80×24

```
test@tests-Virtual-Machine alfred % DEBUG=1 python3 alfred.py
[+] Forcing image creation
[+] Starting main loop
[+] Attempt 1/10

### phase 1: preparation ###
[+] Calling preparevolume
[+] Creating disk
[+] Preparing environment
[+] Mounting image
[+] Turning on indexing
[+] Ejecting image
[+] Rewriting binary
[+] Mounting
[+] Overwrite the file we need
20 -rw-------  1 test  staff  4565 Sep 25 12:55 ./mnt//mnt/.Spotlight-W100/Volum
eConfiguration.plist
[+] Checking payload
[+] Umounting
[+] Reversing binary edit
[+] Mounting image
20 -rw-------  1 test  staff  4565 Sep 25 12:55 ./mnt//mnt/.Spotlight-V100/Volum
eConfiguration.plist
```

```
Mon Sep 25 12:54:40 CEST 2023
Mon Sep 25 12:54:41 CEST 2023
Mon Sep 25 12:54:42 CEST 2023
Mon Sep 25 12:54:43 CEST 2023
Mon Sep 25 12:54:44 CEST 2023
Mon Sep 25 12:54:45 CEST 2023
Mon Sep 25 12:54:46 CEST 2023
Mon Sep 25 12:54:47 CEST 2023
Mon Sep 25 12:54:48 CEST 2023
Mon Sep 25 12:54:49 CEST 2023
Mon Sep 25 12:54:50 CEST 2023
Mon Sep 25 12:54:51 CEST 2023
Mon Sep 25 12:54:52 CEST 2023
Mon Sep 25 12:54:53 CEST 2023
Mon Sep 25 12:54:54 CEST 2023
Mon Sep 25 12:54:55 CEST 2023
Mon Sep 25 12:54:56 CEST 2023
Mon Sep 25 12:54:57 CEST 2023
Mon Sep 25 12:54:58 CEST 2023
Mon Sep 25 12:54:59 CEST 2023
Mon Sep 25 12:55:00 CEST 2023
Mon Sep 25 12:55:01 CEST 2023
Mon Sep 25 12:55:02 CEST 2023
```

Bugs: #6 badmalloc

- an at least 20 year old bug
- **in macOS since at least 2005** (phrack #63)

Bugs: #6 badmalloc

- if the dynamic loader (**dyld**) sees **MallocStack*** env vars:
  - it **force-loads MallocStackLogging.framework**
- **MallocStackLoggingDirectory=pwned**:
  - **framework writes a debug file in pwned**
- this happens in **ALL** processes, **including suids**

Bugs: #6 badmalloc

- if the dynamic loader (**dyld**) sees **MallocStack*** env vars:
    - it **force-loads MallocStackLogging.framework**
- **MallocStackLoggingDirectory=pwned**:
    - **framework writes a debug file in pwned**
- this happens **in ALL** processes, **including suids**

Bugs: #6 badmalloc

- **defenses:**
  - **1. whatever is checked with access() first**
  - **2. open()** will be used to create the file:
    - **won’t overwrite files**
    - **and won’t follow symlinks**
  - **3. permissions are restricted** (no umask() trickery)
  - **4. the filename is randomized**
- **Pretty secure, right?**

Bugs: #6 badmalloc

- ~~1. **whatever** is checked with **access()** first~~
  - **access() / open()** is classic **TOCTOU**
- ~~2. **open()** will be used to create the file:~~
  - ~~won’t overwrite files~~
  - ~~and **won’t** follow a symlink~~
    - **O_NOFOLLOW** is used, not **O_NOFOLLOW_ANY (!)**
- 3. ~~permissions are restricted (no umask() trickery)~~
  - this actually helps
- ~~4. the filename is randomized~~
  - **sudo** doesn’t care

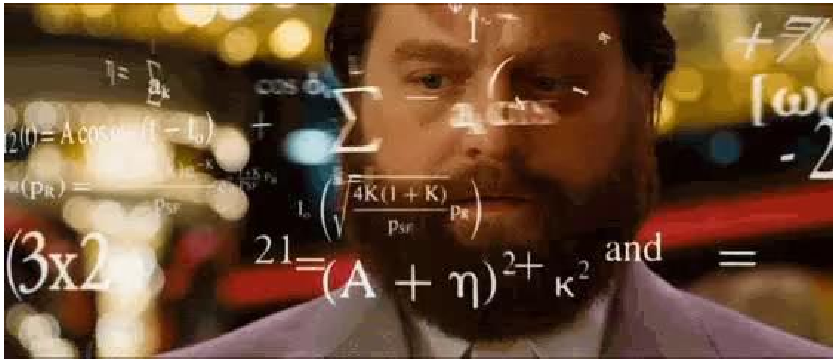
BONUS: the random generator was **hilariously broken...**

Bugs: #6 badmalloc

- one problem remains...
- we have **negligible content control :(**

Bugs: #6 badmalloc

- however:
  - **every application** is affected
  - **“host” app has no idea** about the **open()**
  - **open()** does **not** set **O_CLOEXEC**
- **can we have a suid leak this fd?**



Bugs: #6 badmalloc

- **Of course!**
- **crontab** is **suid** and **executes our $EDITOR**
  - it does not expect a force-loaded library to open a file

Bugs: #6 badmalloc

- **Of course!**
- **crontab** is **suid** and **executes our $EDITOR**
    - it does not expect a force-loaded library to open a file
        - **why would it? that’d be F@#%ING INSANE!**

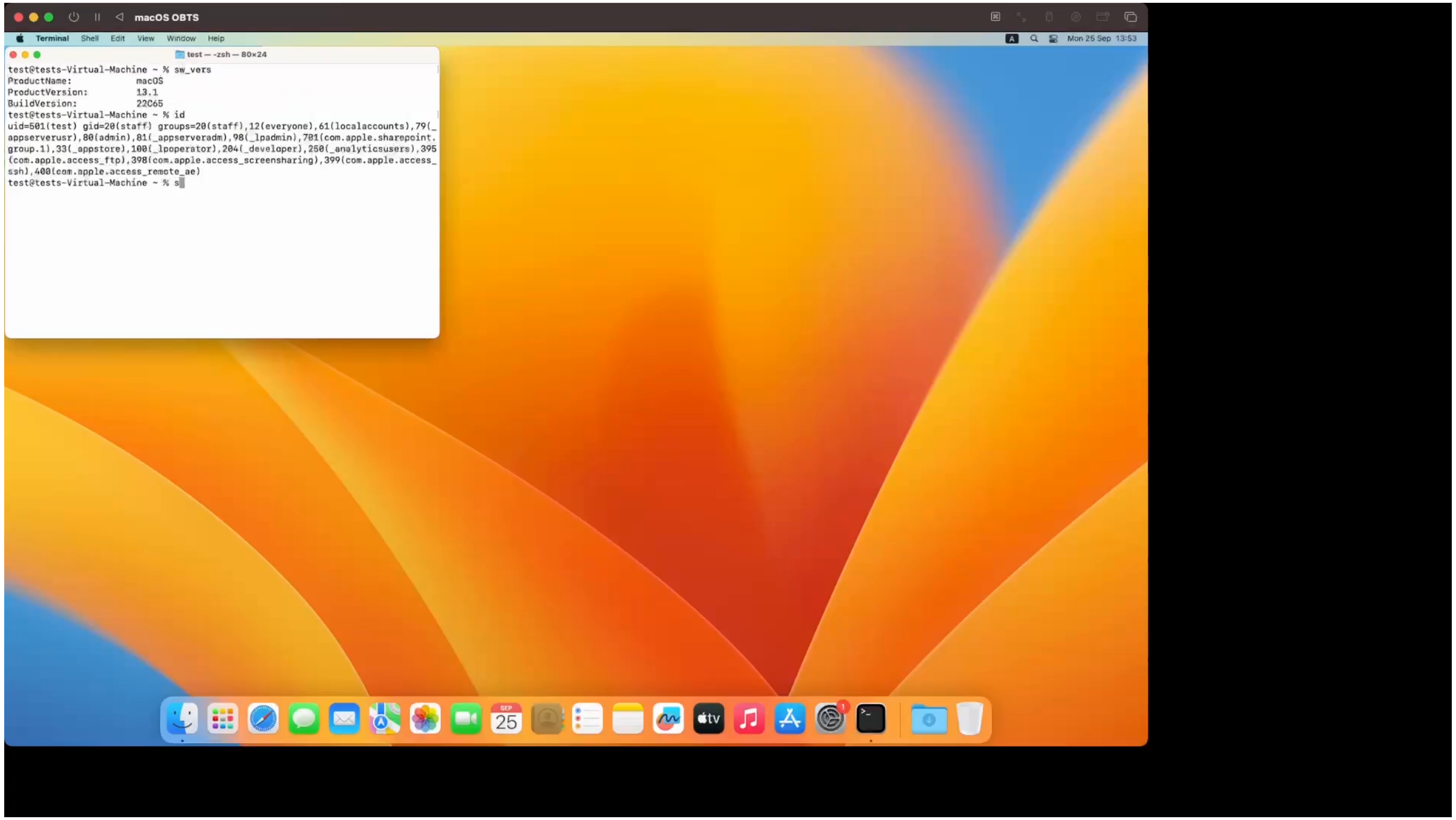
Bugs: #6 badmalloc

- the exploit is trivial
  - one **access() / open()** race, easy to win

Bugs: #6 badmalloc

- this cost **Apple** another **$22,500**
- again, a (minimum) 20 year old bug

Bugs: #6 badmalloc



```
test@tests-Virtual-Machine ~ % sw_vers
ProductName:		macOS
ProductVersion:		13.1
BuildVersion:		22C65
test@tests-Virtual-Machine ~ % id
uid=501(test) gid=20(staff) groups=20(staff),12(everyone),61(localaccounts),79(_
appserverusr),80(admin),81(_appserveradm),98(_lpadmin),701(com.apple.sharepoint.
group.1),33(_appstore),100(_lpoperator),204(_developer),250(_analyticsusers),395
(com.apple.access_ftp),398(com.apple.access_screensharing),399(com.apple.access_
ssh),400(com.apple.access_remote_ae)
test@tests-Virtual-Machine ~ % s
```

Bugs: #7 jetson

- a bug in the handling of **JetPack** files
  - **JetPack** is a custom Apple archive format
  - took me 30min to reverse
  - container of containers
  - supports tar, brotli, etc...
  - supports encryption
    - turned off by default...

Bugs: #7 jetson

- **archive can be tampered with after download**
- files are extracted with **open()** with **O_NOFOLLOW** and **O_EXCL**
- **to a directory I can write to**

Bugs: #7 jetson

- **archive can be tampered with after download**
- files are extracted with **open()** with **O_NOFOLLOW** and **O_EXCL**
- **to a directory I can write to**

Bugs: #7 jetson

- I can replace the archive with my own!
- include a malicious **TCC.db** 100 times :)
    - yes, **you can include the same file 100 times in a tar**

      → **100x** chance to win the **open()** race
- **full path and content control**

Bugs: #7 jetson

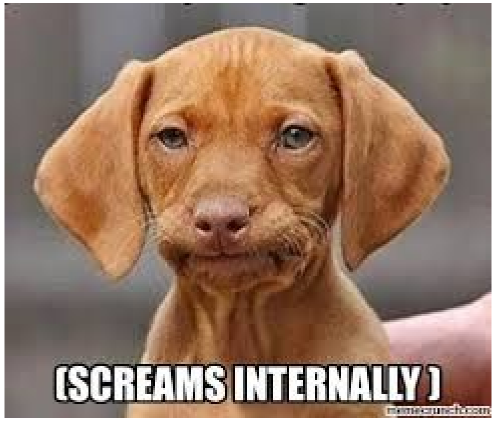
- but what about **O_EXCL**?
    - **JetPack** is friendly and even **unlink()**s the file if it exists
- **lesson:**
    - **it’s not obvious whether any single file operation is secure**
        - because they depend on each other

Bugs: #7 jetson

- anyway, I abused **Music** again
  - to gain **FDA**

Bugs: #7 jetson

- **The fix: Music FINALLY** no longer has **FDA**
- **bounty: $0**
  - **Apple** said this is **not eligible**
    - they were already working on turning on encryption...



Bugs: #7 jetson

- **but they changed their minds!**
  - no idea why
- **bounty: $30,500**

Bugs: #7 jetson

Video will come out with the blogpost

Phew

- I hope you’re still awake... **and that you learned something**
- in case you are wondering:

  total was $153,500

  - **so far...**

This is only the start

- for more:
  - **https://gergelykalman.com**
  - “The missing guide to the security of filesystems and file APIs”

I WANT YOU

TO CONTINUE THIS RESEARCH

Thank you!

Gergely Kalman
@gergely_kalman