


Unexpected, Unreasonable, Unfixable: Filesystem Attacks on macOS

Gergely Kalman

OBTS v6, 2023



Who am I

- my name is **Gergely** (), but you can **call me Greg**
- lifetime computer nerd:
 - **hacking, linux, networks, coding (C, Python)**
- my job is:
 - `sysadmin`
 - `programmer`
 - `entrepreneur`
 - `consultant`
 - **independent bug hunter**
- my views are mine and mine alone



Intro

- we will attack **file operations** on **macOS**
- this is the **condensed** version
 - more info on my blog:
 - <https://gergelykalman.com>
 - or **bottom right corner**
 - two bugs (that were cut) are already up
 - the rest will follow
- my twitter: **@gergely_kalman**



Why attack file operations?

- they're **simple** (to find and exploit)
 - ubiquitous and often a result of **bad design** (→ hard to fix)
 - a **failed exploit has no downside**
- they're **dangerous**
 - a treasure trove of “classic” **LPEs**
 - **TCC + entitlements** made a lot of useless bugs bounty-eligible
 - by rugpulling **POSIX** and **30 years of legacy code**
- **Nobody is paying attention!**
 - and that's good, because I can't learn all the **CFI / PPL / PaC BS**



Terminology

- LPE is **user** → **root**
- TCC bypass is **user** → **user with FDA**
 - **FDA** → location, camera, contacts, etc...



How hard are file operations?

- well...



How hard are file operations?

no copy() syscall

“Everything is a file”

legacy filesystems

symlinks are hard to prevent

in-band signaling in file operations

. and .. are special

network filesystems

race conditions everywhere

POSIX permissions are incredibly complex

mountpoints can move

CWD is unintuitive



How hard are file operations?

no copy() syscall

case-insensitivity

reliance on extended attributes

“Everything is a file”

legacy filesystems

user sets fs options

user mounting

/.file, /.vol/

symlinks are hard to prevent

in-band signaling in file operations

firmlinks

sandbox_exec

. and .. are special

union mounts

network filesystems

user can change mount options at runtime

noowners

race conditions everywhere

applesingle/appledouble

in-band signaling: /..namedfork/rsrc

POSIX permissions are incredibly complex

(force) unmount

hardlinked directories

mountpoints can move

CWD is unintuitive



My bag of dirty tricks

- races: **TOCTOUs**, **rename()**s
- reshape the **fs** graph while in-use
- set **CWD** to a nonexistent directory
- hardlink a directory, hardlink a symlink
- use inheriting **ACLs**
- modify **xattrs** by editing `applesingle` / `appledouble` files
- `rugpull` programs
 - by force unmounting
 - by moving the mountpoint



My bag of dirty tricks

- mount (as a user):
 - a network volume with 5s latency
 - use mount options with **noowners**, **union**, **etc...**
 - update the mountpoint or remount in place
 - change filesystem encoding
 - use filesystems that don't support **xattrs**
- corrupt the filesystem image:
 - create a directory loop
 - hardlink directories at the top level
 - make “..” point not to the parent
 - create structures that normally would not be possible



File operations ARE hard

- an **unprivileged user** can do any/all of this
- So is this the end of the World?
 - No, but **Apple** is in a tough spot...
 - **without total FS isolation, TCC will always be problematic**
 - but to be fair: **TCC is better than nothing**
- **Apple** doesn't isolate apps with **uids** like **Android**
 - IDK why, but if you do → **DM** me :)



Our focus

- we will focus on **TCC bypass** and **LPE**
- lots of good syscalls, but the best are:
 - **open()** and **rename()**:
 - they're everywhere
 - they're easy to mess up
 - they're useful for **LPEs** and **TCC bypasses**
 - these are promising, but I don't have time:
 - **unlink()**, **rmdir()**, **mkdir()**: **Ubiquitous**, but **tricky to exploit**
 - honorable mentions:
 - **chmod**, **chown**, **setxattrs**, **umask**, **chflags**, **clonefile**, **readlink**, **link**, **symlink**, etc...
 - **Rare** and usually **only good for LPEs**



The obstacles

1) file **path control**

2) file **content control**

- **more control → higher severity**
 - partial control over each is only good for **LPEs**
 - for **TCC bypass** you need **full path and content control**
 - if I missed smth → **DMs** are open



The allies

- lots of large entitled apps
- sudo
- bad POSIX APIs: `O_NOFOLLOW`, no symlink prevention, etc...
- atomic rename → `renamex_np()` / `renameatx_np()` + `RENAME_SWAP`
- user mounting



The allies

- **string truncation bugs**
 - can help you get **full path control** → common and deadly
- **rename bugs:**
 - `rename()` always follows symlinks
 - `rename("./a", "./b")` is a race-condition
 - “a” and “b” are looked up **separately**, and **CWD** is implicit
 - if I control any path component in **CWD** I can turn this into
 - `rename(“anything/a”, “somethingelse/b”)`
 - write a file called “b” anywhere, with **fully controlled contents** :)



Things I look for

- insecure **open()**:
 - bad path, bad/missing flags
 - classic `access()` / `open()` races
 - file “copy”
 - file “recreation”
- insecure **rename()**:
 - bad path
 - **dangerous renames**



The bugs

librarian (CVE-2023-38571) - TCC bypass - [check blog](#)

unnamed app sandbox escape (CVE-2023-32364) - app sandbox escape - [check blog](#)

1)lateralus (CVE-2023-32407) - TCC bypass

2)sqlol (CVE-2023-32422) - TCC bypass

3)batsignal (no CVE) - LPE

4)alfred (CVE-2023-40443) - LPE

5)badmalloc - (CVE-2023-32428) - LPE



Cut bugs

- **librarian** - TCC bypass
 - fully controlled `rename()` in **Music**
- **unnamed sbx escape** - app sbx escape
 - **sandbox escape** by preventing **quarantine xattr** placement using **devfs** and **symlinks**



The bugs

~~librarian (CVE-2023-38571) - TCC bypass - check blog~~

~~unnamed app sandbox escape (CVE-2023-32364) - app
sandbox escape - check blog~~

1)lateralus (CVE-2023-32407) - TCC bypass

2)sqlol (CVE-2023-32422) - TCC bypass

3)batsignal (no CVE) - LPE

4)alfred (CVE-2023-40443) - LPE

5)badmalloc - (CVE-2023-32428) - LPE



Bugs: #1 Lateralus

- **lateralus** (CVE-2023-32407) - TCC bypass
- insecure file write in the Metal library
 - `MTL_DUMP_PIPELINES_TO_JSON_FILE="path/name"`
- Foundation's `NSFileManager` `createFilePath` is used:
 - `open()`s new tempfile: `"path/.dat.nosyncXXXX.XXXXXX"` (X is random)
 - writes the contents
 - calls `rename("path/.dat.nosyncXXXX.XXXXXX", "path/name")`
- dangerous rename with full control over the path



Bugs: #1 Lateralus

- how do we get **content control**?
 - impossible if we use “~/Library/Application Support/com.apple.TCC/” directly
- but we can use a **controlled location**
 - wait for the temp file and **open()** it
 - race the **rename()**
 - in a loop: atomically swap the directory with a symlink
- → **full control over path and contents**



Bugs : #1 Lateralus

<https://www.youtube.com/watch?v=JPrCwUFYPkw>



The bugs

~~librarian (CVE-2023-38571) - TCC bypass - check blog~~

~~unnamed app sandbox escape (CVE-2023-32364) - app
sandbox escape - check blog~~

1) ~~lateralus (CVE-2023-32407) - TCC bypass~~

2) **sqlol (CVE-2023-32422) - TCC bypass**

3) ~~batsignal (no CVE) - LPE~~

4) ~~alfred (CVE-2023-40443) - LPE~~

5) ~~badmalloc - (CVE-2023-32428) - LPE~~



Bugs: #2 sqlol

- **sqlol** (CVE-2023-32422) - **TCC bypass**
- insecure file write in **libsqlite** (only on **macOS**)
 - debug functionality in production (compiled with `SQLITE_ENABLE_SQLLOG`)
 - `SQLITE_SQLLOG_DIR="whatever"` means:
 - copy the opened DBs to **whatever**
 - write a query log and index file as well
 - files are created with `open()`, which:
 - **follows symlinks**
 - **overwrites files**
- a trivial infoleak, but I want to overwrite **TCC.db**



Bugs: #2 sqlol

- **controlling the filename:** use a **symlink**
- **controlling content** is tricky
 - I can overwrite files, but only with debug files: the **DB**, the **statement log**, the **index**
 - this stumped me a bit...



Bugs: #2 sqlolol

- until I realised:
 - a sqlite DB can have multiple tables in it
 - **TCC.db** is a sqlite DB
- we can “smuggle” the **TCC.db**’s tables into any other sqlite DB:
 - **Music** has **FDA**, and a writable DB (**Cache.db**)
 - I can add the **TCC** tables to it
 - **Cache.db** can now replace and function as **TCC.db** :)
 - we don’t even need to race



Bugs: #2 sqlol

<https://www.youtube.com/watch?v=rfGcd0YrbTM>



The bugs

~~librarian (CVE-2023-38571) - TCC bypass - check blog~~

~~unnamed app sandbox escape (CVE-2023-32364) - app
sandbox escape - check blog~~

1)lateralus (CVE-2023-32407) - TCC bypass

2)sqlol (CVE-2023-32422) - TCC bypass

3)batsignal (no CVE) - LPE

4)alfred (CVE-2023-40443) - LPE

5)badmalloc - (CVE-2023-32428) - LPE



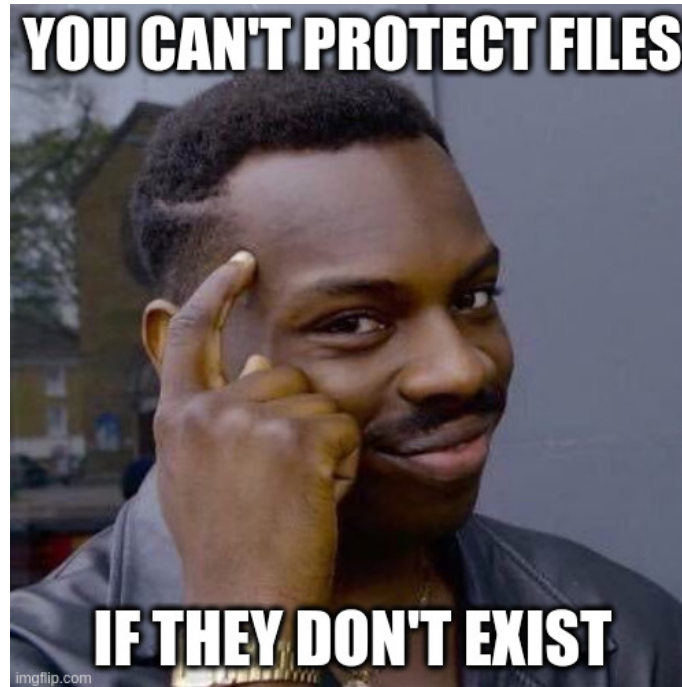
Bugs: #3 batsignal

- **batsignal** (no CVE) – LPE
- collision with **Joshua Mason's** CVE-2022-32801
 - + a couple bypasses
 - no credit, just a small bounty...
- **Spotlight** performs file operations on **user-mounted volumes**
 - daemons **mds** and **mds_stores** run as **root** (mds even has FDA)
 - they use a **SIP-protected** directory on the volume:
 - **“/.Spotlight-V100”**



Bugs: #3 batsignal

- v1: Exploiting Spotlight for the first time



Bugs: #3 batsignal

- **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**
- **HFS+** is fine with this
- **Spotlight** won't care
- **SIP** won't notice



Bugs: #3 batsignal

- (one of the) **bug(s)**:
 - **Spotlight** writes cache files insecurely with **open()**
- to exploit:
 - **symlink** a file in the Caches directory
 - **Spotlight** will truncate and overwrite existing files
- the cache file has:
 - **attacker-controlled content**
 - a **known filename** → **X.txt**, where **X** is the inode number
- **the fix: Spotlight no longer likes symlinks :(**



Bugs: #3 batsignal

- v2: Exploiting Spotlight for the second time



Bugs: #3 batsignal

- the protection is **still** a regex engine
 - 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**" volumes?
 - **macOS** does allow crazy things...



Bugs : #3 batsignal

- **Yes**, we can use **unions**
 - **these complicate EVERYTHING**
 - but not for us :)
- **union 101:**
 - two volumes mounted over each other
 - **top** and **bottom**
 - lookups start in **top**
 - fall back to **bottom**
- they can also nest
 - nest
 - nest



Bugs: #3 batsignal

- to exploit:
 - mirror Spotlight's directory structure 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: #3 batsignal

- the fix: Apple now disallows union mounts using SIP :(
- At least I got a bounty. After more than a year. Still no credit though
- conclusion:
 - allowing users to mount disk images is **crazy**:
 - **attacker has all the leverage**
 - Apple does pay bounties
 - but it's complicated...



Bugs : #3 batsignal v1

<https://www.youtube.com/watch?v=Xvb9peOSys0>



The bugs

~~librarian (CVE-2023-38571) - TCC bypass - check blog~~

~~unnamed app sandbox escape (CVE-2023-32364) - app
sandbox escape - check blog~~

1)lateralus (CVE-2023-32407) - TCC bypass

2)sqlol (CVE-2023-32422) - TCC bypass

3)batsignal (no CVE) - LPE

4)alfred (CVE-2023-40443) - LPE

5)badmalloc - (CVE-2023-32428) - LPE



Bugs: #4 alfred

- **alfred** (CVE-2023-40443) - LPE
- really it's **batsignal v3**
- to recap:
 - **Spotlight** does insecure writes on user-provided volumes
 - now we can't use **symlinks**, or **union-mounts** :(
- **What now?**



Bugs: #4 alfred

- move the mountpoint :)
 - by moving the parent
- `rugpull mds` → write to system volume
- Apple did a **good job** of restricting mds, with two exceptions:
 - (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/"`
 - we'll call this `^^^ tmpdir`



Bugs: #4 alfred

- rinse and repeat...
 - prepare the directory structure like before in **tmpdir**
 - swap the mountpoint between **tmpdir** and the original mountpoint in a loop
- when the race is won **mds** will overwrite one of our files
 - that is a hardlink to **/etc/sudoers**
- **how do we control the content?**



Bugs: #4 alfred

- we need a file that:
 - we can smuggle our payload into
 - gets recreated
- the most obvious target is **VolumeConfiguration.plist**
 - we can smuggle our payload in as a bogus file exclusion path:
 - “\n\nroot ALL=(ALL:ALL) ALL\n\n”
- **Spotlight** will
 - remember this after a remount
 - recreate the file if it's missing



Bugs : #4 a1fred

<https://www.youtube.com/watch?v=YfJdzqqqQFo>



The bugs

~~librarian (CVE-2023-38571) - TCC bypass - check blog~~

~~unnamed app sandbox escape (CVE-2023-32364) - app
sandbox escape - check blog~~

1)lateralus (CVE-2023-32407) - TCC bypass

2)sqlol (CVE-2023-32422) - TCC bypass

3)batsignal (no CVE) - LPE

4)alfred (CVE-2023-40443) - LPE

5)badmalloc - (CVE-2023-32428) - LPE



Bugs : #5 badmalloc

- **badmalloc** - (CVE-2023-32428) - LPE
- **MallocStackLogging**(.framework) on macOS / iOS performs **insecure file writes**
 - if **MallocStack*** env vars are set dyld force-loads **MallocStackLogging** into **any binary**
 - this is in macOS since at least 2005 (!) (phrack #63)
 - **MallocStackLogging** writes a file at an attacker-provided path
- we can make **any app** (== "host") do this :)



Bugs : #5 badmalloc

- to trigger
 - `MallocStackLogging=1`
 - `MallocStackLoggingDirectory="whatever"`
- `MallocStackLogging` writes debug files to `whatever`
- `Apple's` not stupid though, so there are `defenses`



Bugs : #5 badmalloc

- defenses:
 - **whatever** is checked with `access()` first
 - `open()` will be used to create the file:
 - won't overwrite files
 - and won't follow a symlink
 - permissions are restricted (no `umask()` trickery)
 - the filename is randomized
- **Pretty secure, right?**



Bugs: #5 badmalloc

- defenses:

- ~~whatever~~ is checked with `access()` first
 - `access()` / `open()` is classic TOCTOU
 - we can race it
- ~~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` (!)
- permissions are restricted (no `umask()` trickery)
 - this actually helps us...
- the filename is randomized
 - `sudo` will gobble up any file from `/etc/sudoers.d/ :`
 - and the random generator was hilariously broken...



Bugs: #5 badmalloc

- only minimal content control :(
- this stumped me for quite a long time...



Bugs: #5 badmalloc

- until I realised that:
 - **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: #5 badmalloc

- **Yes, crontab!**
- **crontab is suid and executes our editor**
 - it does not expect a **force-loaded library to open a file**
 - most programs wouldn't...



Bugs : #5 badmalloc

- to exploit we can call `crontab` with
 - `EDITOR=ourscript.py`
 - `MallocStackLogging=1`
 - `MallocStackLoggingDirectory="whatever"`
- we race the `access()/open()` by swapping `whatever` with a symlink to `/etc/sudoers.d/`



Bugs: #5 badmalloc

- the race is won in a couple tries
- our **EDITOR** gets executed:
 - with an open **fd** to a random file under **/etc/sudoers.d/**
 - writes payload: **“root ALL=(ALL:ALL) ALL”**
 - **sudo bash**



Bugs : #5 badmalloc

<https://www.youtube.com/watch?v=iNfeo9vkhK0>



We're done :)

~~librarian (CVE-2023-38571) - TCC bypass - check blog~~

~~unnamed app sandbox escape (CVE-2023-32364) - app
sandbox escape - check blog~~

1) ~~lateralus (CVE-2023-32407) - TCC bypass~~

2) ~~sqlol (CVE-2023-32422) - TCC bypass~~

3) ~~batsignal (no CVE) - LPE~~

4) ~~alfred (CVE-2023-40443) - LPE~~

5) ~~badmalloc - (CVE-2023-32428) - LPE~~



Thank you friendly hackers!

- **Special thanks to these folks**
 - Csaba Fitzl (@theevilbit)
 - Wojciech Reguła (@_r3ggi)
 - Joshua Mason
 - Buherator
 - Zoltan Padanyi aka max
 - Tamas Kozak
 - Dora
- **among many others**



Thanks Apple!



Thank you!

Gergely Kalman
@gergely_kalman



Talk to me if you have questions

Find me in the hallways or Twitter

Gergely Kalman
@gergely_kalman

