SYNACKTIV

Finding and exploiting an old XNU logic bug

Hexacon 2023

Whoami

SYNACKTIV

- **Eloi Benoist-Vanderbeken**
  - @elvanderb
  - Reverse Engineering team tech lead
  - iOS / macOS
- **Past presentations**
  - An Apple a day keeps the exploiter away (SSTIC 2022)
  - macOS: how to gain root with CVE-2018-4193 in < 10s (OffensiveCon 2019)
  - Heapple Pie: macOS and iOS default heap (Sthack 2018)
- **Synacktiv**
  - Hexacon organisers!
  - Offensive security
  - 140 experts
  - Pentest, Reverse Engineering, Development, Incident Response
- **Reverse Engineering team**
  - 47 reversers
  - Low level researches, reverse engineering, vulnerability research, exploit development, etc.

Pwn2own 2023

SYNACKTIV

- **New target !**
  - LPE on a MacBook Pro
    - MUST use a kernel bug
  - With an M-series SOC
    - PAC!
  - $40,000
    - Not much but better than nothing :)

- **Time to find some bugs…**

Which bugs?

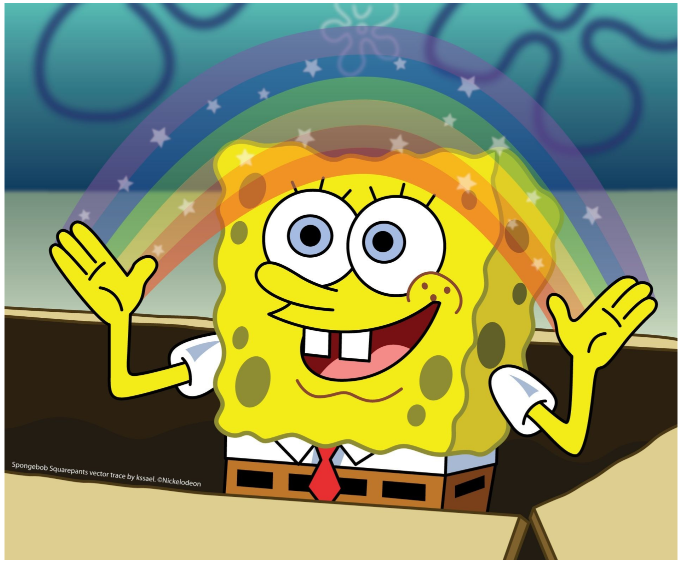
- **No more cheap bugs!**
  - No iOS bug
  - No PAC bypass
  - No ninja exploit techniques
- **Actually not that easy…**
  - No memory corruption
    - Or very specific ones
  - Not a lot of surface
- **Other constraints…**
  - Want to work on my M1 MacBook Air
  - No company tools
    - IDA > Ghidra…
    - No KEXTs

Which bugs?

- **No more c** **.**
  - No iOS ee time
  - No PAC ny tools
  - No ninj
- **Actually n**
  - No me
    - Or
  - Not a l

File system

- **Large non iOS attack surface**
  - Can mount / unmount things on macOS
  - SUID binaries
  - Almost no sandbox
- **Source of logic bugs/exploits**
  - SUID binaries
  - Turns UAF into arb. file write
  - etc.
- **Lots of code in XNU**
  - No need to get our hand dirty with Ghidra



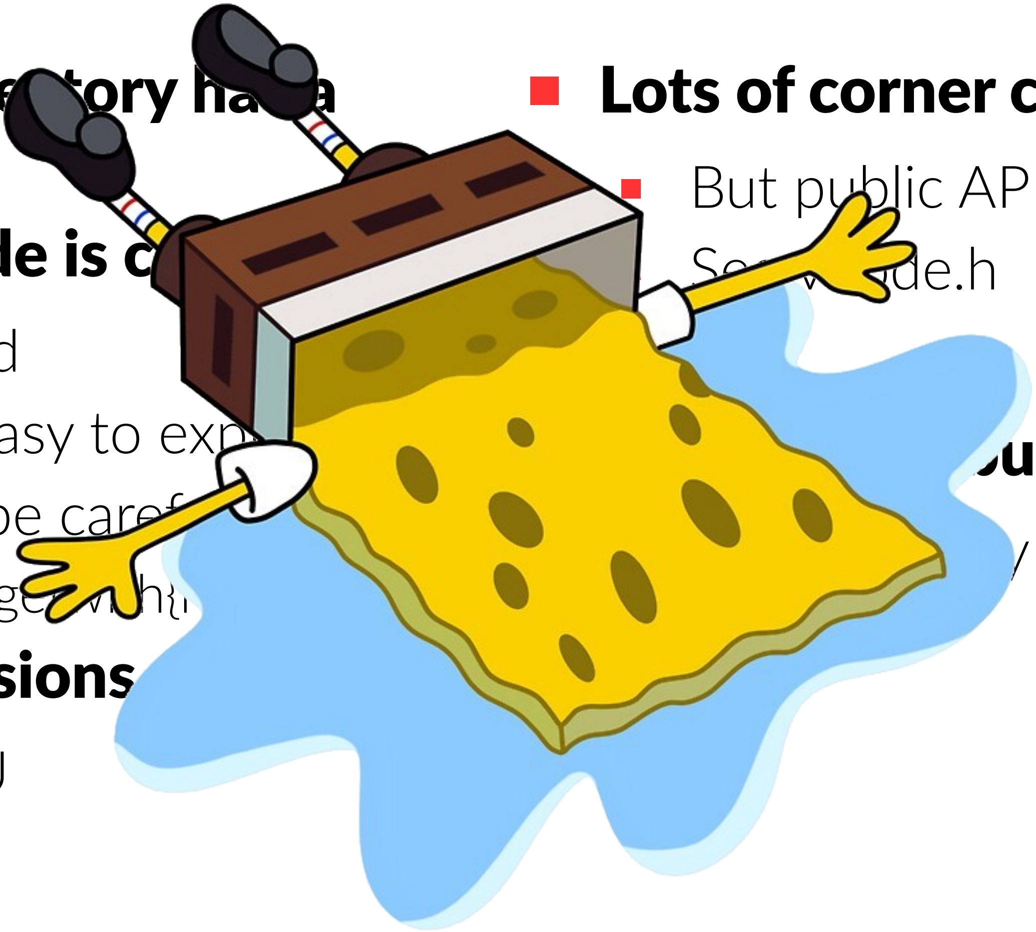
vnodes

- **Each file/directory has a vnode**
- **Path ↔ vnode is cached**
  - Lazily freed
  - Not that easy to exploit UAF
  - Needs to be careful
    - vnode_getwith{ref/vid}
- **Unix permissions are cached**
  - Saves CPU
- **Lots of corner cases**
  - But public API
  - See vnode.h
- **Found some bugs…**
  - Not that easy to exploit :'(

vnodes

- **Each file/dire tory ha a vnode**
- **Path ↔ vnode is c**
  - Lazily freed
  - Not that easy to exp
  - Needs to be caref
    - vnode_ge h
- **Unix permissions**
  - Saves CPU

- **Lots of corner cases**
  - But public API
  - Se de.h
- **ugs…**
  - to exploit :'(



10 days before the dead line...

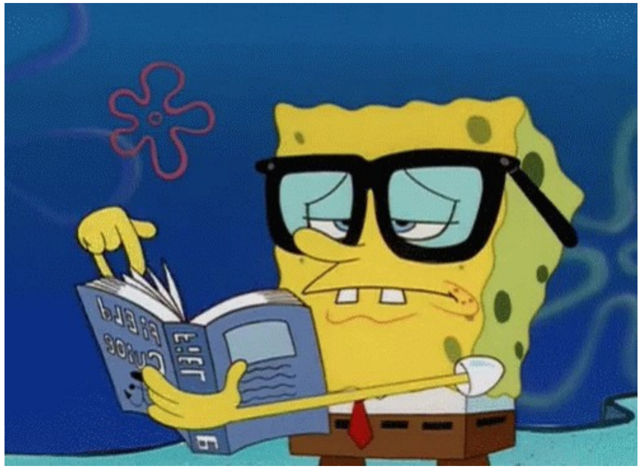
2 days after saying that I gave up...

GARY

Let’s have a look to /dev/fd

man fd

```
FD(4)                    Device Drivers Manual                    FD(4)

NAME
     fd, stdin, stdout, stderr – file descriptor files

DESCRIPTION
     The files /dev/fd/0 through /dev/fd/# refer to file descriptors which can
     be accessed through the file system.  If the file descriptor is open and
     the mode the file is being opened with is a subset of the mode of the
     existing descriptor, the call:

           fd = open("/dev/fd/0", mode);

     and the call:

           fd = fcntl(0, F_DUPFD, 0);

     are equivalent.
```



Ugly hack

- **Saw the code during my review**
- **Ugly hack in open**
  - */dev/fd* open func returns *ENODEV*…
    - And set *bsdthread_info→uu_dupfd* = *vnode→fd_fd*
  - … which is handled by the *open* syscall…
  - … by calling *dupfdopen(bsdthread_info→uu_dupfd)*
- **Fun but not interesting…**
  - Almost exact same thing than dup…
  - Used to use the same */dev/fd vnodes* for every process

Sometimes all you need is vnode

SYNACKTIV

- **This ugly hack doesn't always work**
  - Other syscalls manipulate paths
- **What happens when you call chmod("/dev/fd/3", 777)?**
  1. get "*/dev/fd/3*" vnode
     - */dev/fd* special vnode
     - Mostly only hold the *fd* number
  2. check if the *chmod* operation is authorized
     - Call the MAC hooks
     - Call *vnode_getattr* to get the *vnode* mode bits / owner etc.
  3. change the mode bits
     - Call *vnode_setattr* on the *vnode*

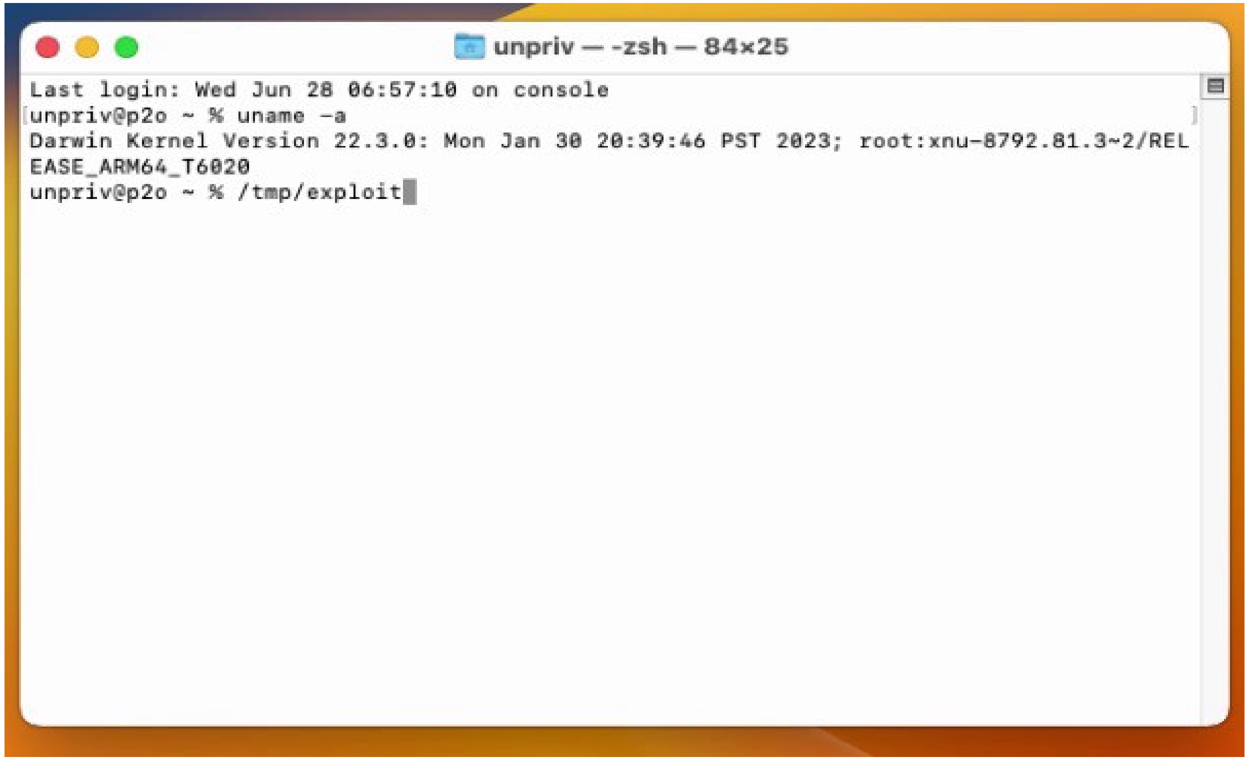
Got it?

- **vnode_getattr / vnode_setattr**
  - Call the */dev/fd* functions *fdesc_getattr* / *fdesc_setattr*
  - Lookup the *fd* in the current context with *fp_lookup*
  - Call *vnode_getattr* / *vnode_setattr* on the underlying *vnode*

Got it?

- **vnode_getattr / vnode_setattr**
  - Call the */dev/fd* functions *fdesc_getattr* / *fdesc_setattr*
  - Lookup the *fd* in the current context with *fp_lookup*
  - Call *vnode_getattr* / *vnode_setattr* on the underlying *vnode*
- **Obvious TOCTOU**
  - You can change the *fd* between the calls
    - Just close the *fd* and reopen anything
  - Can be used to *chmod* all the files we can get a *fd* on
    - Trivial to get root (just modify a root file and make it *suid*)
  - Less than 1 day to find and exploit the vulnerability

Making animated ASCII arts is hard



```
Last login: Wed Jun 28 06:57:10 on console
unpriv@p2o ~ % uname -a
Darwin Kernel Version 22.3.0: Mon Jan 30 20:39:46 PST 2023; root:xnu-8792.81.3~2/REL
EASE_ARM64_T6020
unpriv@p2o ~ % /tmp/exploit
```

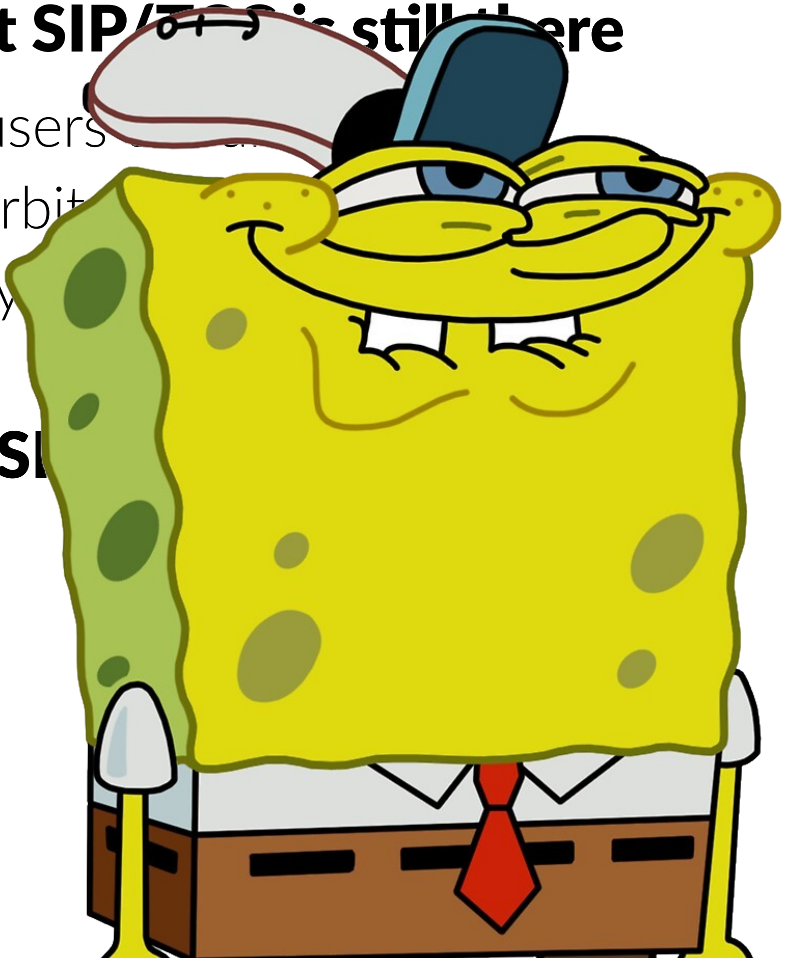
Can we do more?

SYNACKTIV

- **root is great but SIP/TCC is still there**
  - Cannot read users documents
  - Cannot load kexts
  - Cannot modify all the files
- **Can we bypass SIP with the same bug?**

Can we do more?

- **root is great but SIP/TCC is still there**
  - Cannot read users
  - Cannot load arbit
  - Cannot modify
- **Can we bypass SI**



SIP

- **Protects system files against arbitrary modifications**
  - Among other things
- **Used to enforce other security mechanisms**
  - Notably the kext related files
    - restrictions / MDM configuration / user consent / etc.
  - Protected with the “restricted” flag

```
% ls -aOl /var/db/SystemPolicyConfiguration/KextPolicy
-rw-------  1 root  wheel  restricted 4096 Nov 15  2022 KextPolicy
```

Ooops

- **Remember few slides back…**
  - MAC hooks are called with the */dev/fd vnode*
  - The sandbox only sees this *vnode*

- **The vulnerability**
  - SIP has no way to know what's the "real" underlying vnode
  - It could call *vnode_getattr* to check the *restricted* flag
    - But it would still be exploitable with a race
  - But it actually don't even bother!
    - Path based rule?

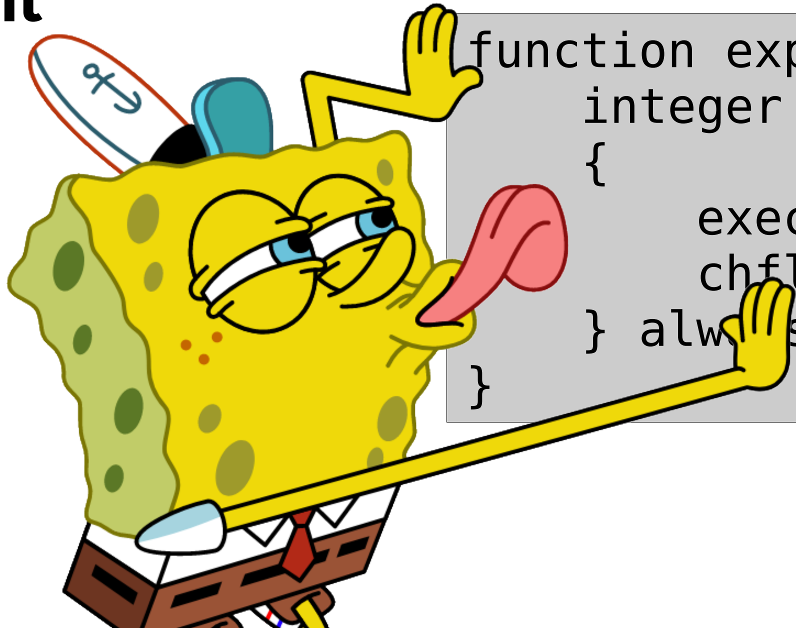
31337 exploit

- Open a file read only
- Change the flags on the */dev/fd/XXX* alias
- …
- Profit

31337 exploit

- Open a file read only
- Change the flags on the */dev/fd/XXX* alias
- ...
- Profit

```
function exploit() {
    integer i
    {
        exec {i}<"$1"
        chflags norestricted "/dev/fd/$i"
    } alw[illegible]s { exec {i}>&- }
}
```



But how to get kernel code exec?

SYNACKTIV

- **Easy to bypass user consent**
  - Just edit the *KextPolicy* database
- **Easy to bypass deprecated function detection**
  - Just rm *KextClassification.plist*
- **Not that easy to load unsigned kexts**
  - It may be possible, I didn't spent too much time on it
  - Ping me if you know how to do it :)
- **Sufficient to load a correctly signed kext**
  - Don't forget to kill *syspolicyd*

How has it been fixed?

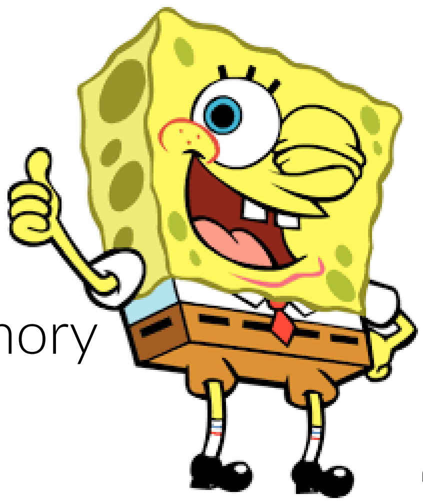
SYNACKTIV

- **Apple just added some checks in the *****/dev/fd***** code**
  - Get the underlying *vnode*
  - Re-do the checks done in *chmod/chflags*

- **Fixed in macOS 12.6.6 and iOS 16.5**
  - CVE-2023-32413
  - iOS shouldn't be impacted
    - */dev/fd* is not even compiled in the release kernels…
    - … but it was in the accidentally released 15.x dev kernels
      - Please Apple, release more of them

Conclusion

- **No /dev/fd on iOS**
  - Even if…
  - Sandbox, no SUID, mandatory code signature, no interpreter, etc…
- **Still a lot easier to get root on macOS**
  - Even with PAC
- **Logic bugs won’t save us all**
  - But “classic” memory corruptions neither
  - Probably why we see so much reports in virtual memory
    - But for how long…



SYNACKTIV

www.linkedin.com/company/synacktiv

www.twitter.com/synacktiv

www.synacktiv.com