Caught you - reveal and exploit IPC logic bugs inside Apple

Zhipeng Huo, Yuebin Sun, Chuanda Ding
of
Tencent Security Xuanwu Lab

Who are We?

- Zhipeng Huo (@R3dF09)
  - Senior security researcher
  - Member of EcoSec Team at Tencent Security Xuanwu Lab
  - macOS, iOS and Windows platform security
  - Speaker of Black Hat Europe 2018, DEF CON 28

Who are We?

- Yuebin Sun (@yuebinsun2020)
  - Senior security researcher
  - Member of EcoSec Team at Tencent Security Xuanwu Lab
  - macOS, iOS platform security

Tencent 腾讯

腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

Who are We?

- Chuanda Ding (@FlowerCode_)
  - Senior security researcher
  - Leads EcoSec Team at Tencent Security Xuanwu Lab
  - Windows platform security
  - Speaker of Black Hat Europe 2018, DEF CON China 2018, DEF CON 28

Tencent 腾讯

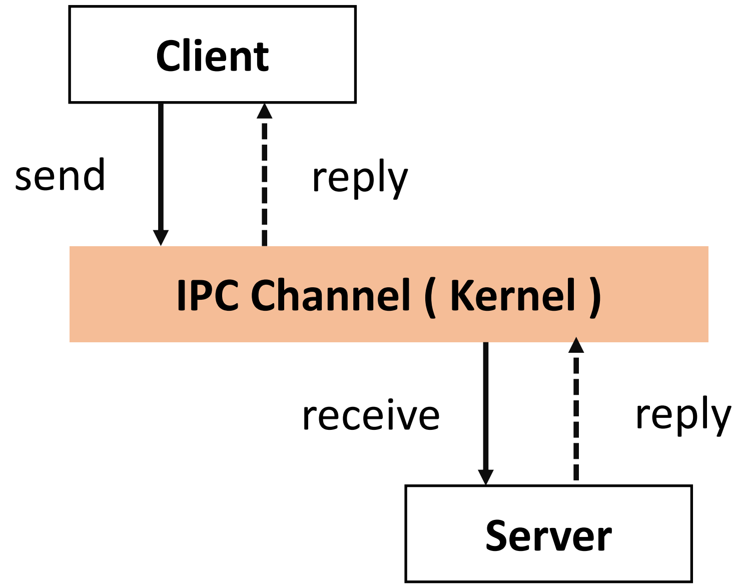
腾讯安全玄武实验室
TENCENT SECURITY XUANWU LAB

Agenda

Introduction

Inter-Process Communication

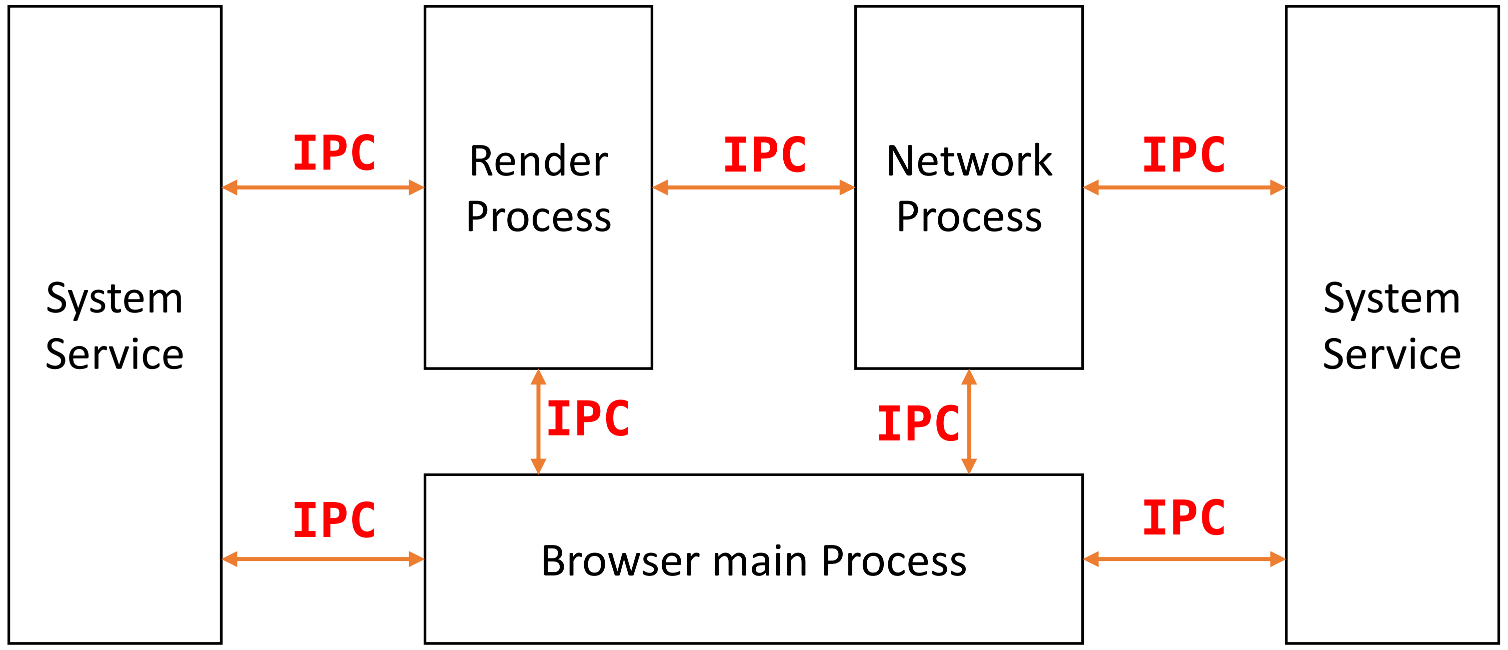
- Inter-Process Communication (IPC) is the set of techniques provided by the operating system to allow processes to communicate with each other
- Roles - Client/Server
  - Client requests Server
  - Server responds to Client if needed
- IPC Channel



Why Needs IPC ?

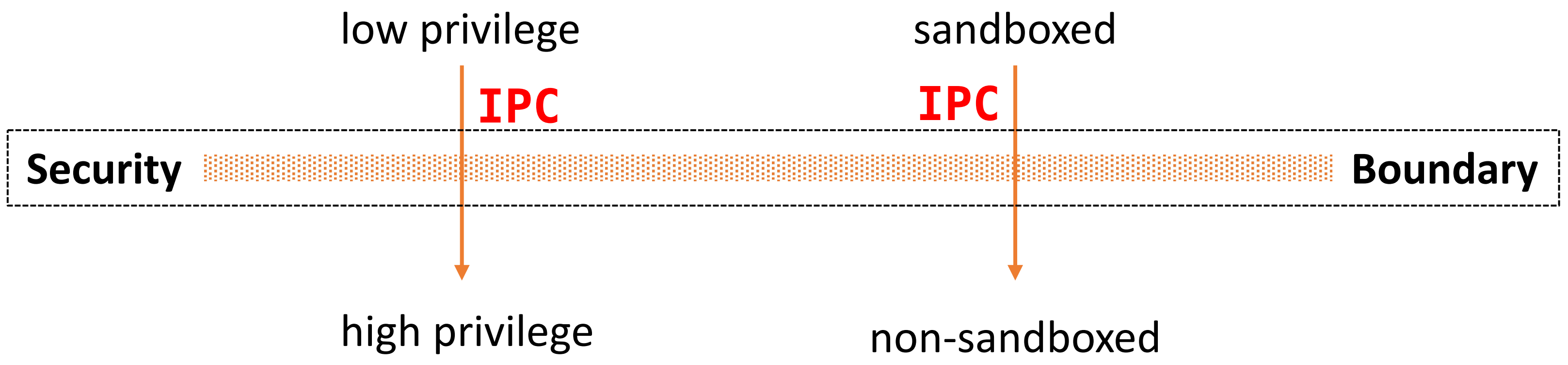
- Modularity
  - Divide complex system to separated modules
  - Don’t Repeat Yourself
- Stability
  - Module crash would not crash entire system
- Privilege separation
  - Isolate sensitive operations into separated processes
  - Principle of Least Privilege
  - Module attacked would not compromise entire system

An IPC Example: Browser Architecture



IPC Breaks Security Boundary

- Different process, different privilege
- IPC is the “Window” between different privilege
  - IPC vulnerability is “Key” to the high privilege



Logic Vulnerability

- Not memory corruption vulnerabilities
  - Boring for us
- Kinds of logic flaws
  - Design flaw
  - Implementation flaw
- Combine “features” to compromise system

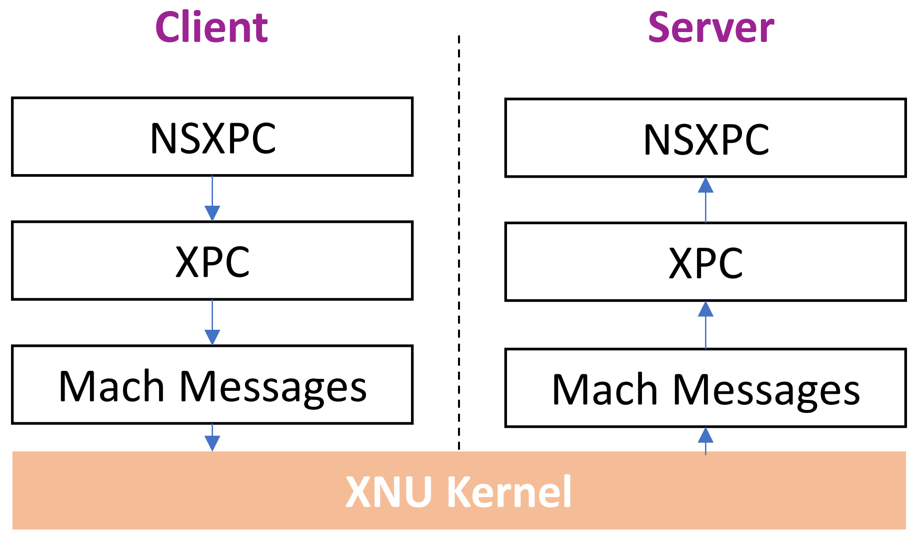
New Challenge - Apple M1

- New Chip, New Security Features
  - System Integrity
  - Data Protection
  - **Pointer Authentication Code (PAC)**
- Hardware-level security mechanism against memory bug
  - Memory game became harder!
- So, spring of logic vulnerability is coming ?

IPC on Apple Platforms

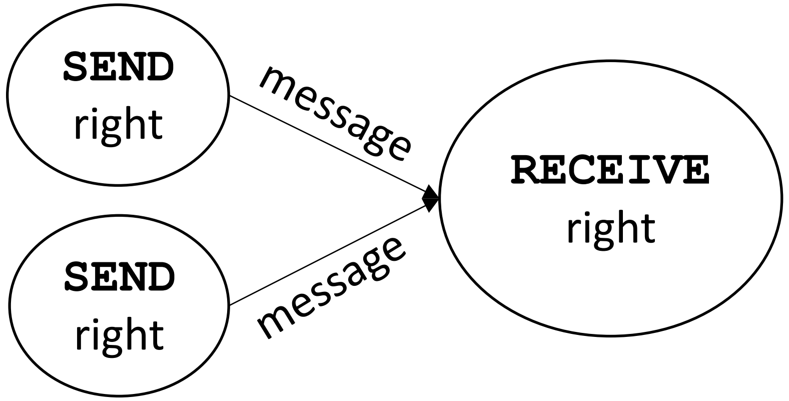
Apple IPC Methods

- Shared File
- Shared Memory
- Sockets
- Apple Events
- Distributed Notifications
- Pasteboard
- **Mach messages**
- **XPC**
- **NSXPC**
- …



Mach Port

- An endpoint of a unidirectional **communication channel**
  - Messages can be sent to or received from
- Port Rights
  - **`RECEIVE`** right – Receive message
  - **`SEND`** right – Send message

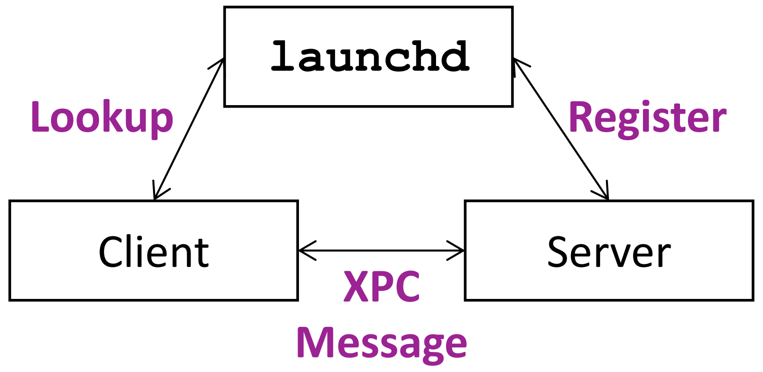


Mach Messages

- Message send and receive through system call
  - **`mach_msg`**
  - **`mach_msg_overwrite`**
- Message structure
  - Header
  - Complex data (optional)
    - Port rights or OOL data
  - Message Buffer
- Pros and Cons
  - Low-level, fundamental, powerful
  - Ancient, poorly documented, hard to use directly

XPC

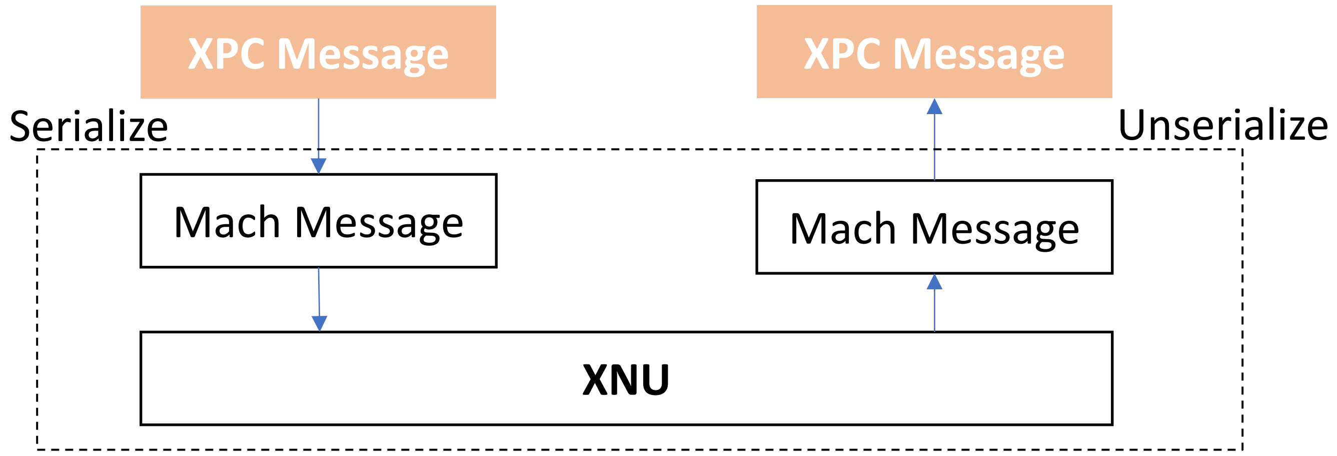
- Latest IPC mechanism on top of Mach messages
- Managed by **`launchd`**
  - Naming server
  - On-demand launch, monitor, terminate server
    - Transparent
- XPC Message
  - Dictionary object
  - Don’t need to play with Mach message structure directly



XPC Message

- **`xpc_dictionary_set_*`**
- **`xpc_dictionary_get_*`**
- …

```
{
    "CFPreferencesUser": "kCFPreferencesCurrentUser",
    "CFPreferencesOperation": 1 ,
    "CFPreferencesShouldWriteSynchronously": true ,
    "CFPreferencesCurrentApplicationDomain": true ,
    "CFPreferencesDomain": "/tmp/xlab.txt" ,
}
```



XPC API

- **`xpc_connection_create_mach_service`**
  - Creates a new connection object that represents a Mach service
  - A peer connection will be returned
  - if **`XPC_CONNECTION_MACH_SERVICE_LISTENER`** flag is set, a listener connection returned
- **`xpc_connection_set_event_handler`**
  - Sets the event handler block for the connection
- **`xpc_connection_send_message`**
  - Sends a message over the connection to the destination service

NSXPC

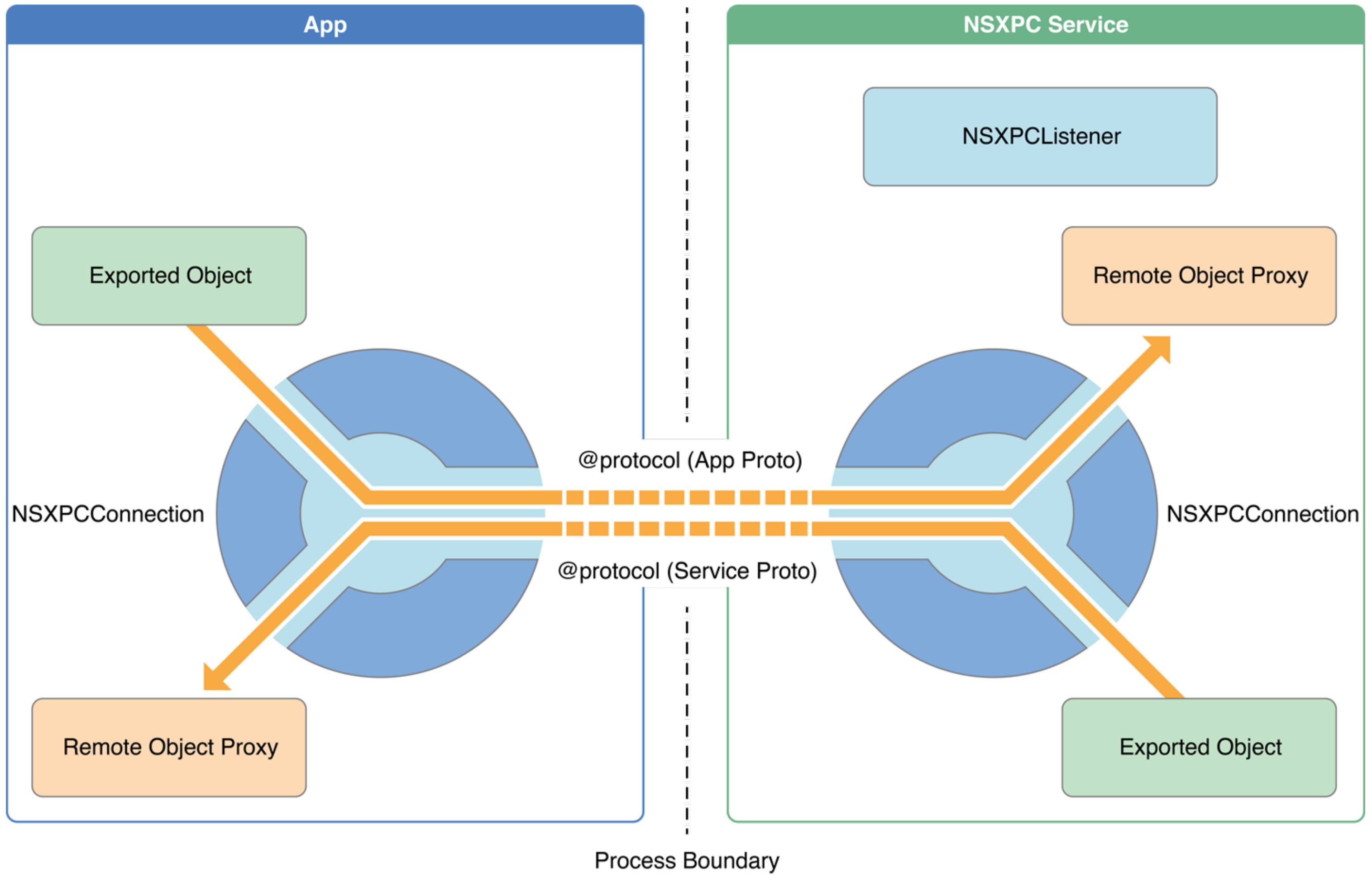
- Object-oriented IPC mechanism on top of XPC
- High-level remote procedure call interface that allows you to call methods on objects in one process from another process
- **`NSXPCConnection`** API
  - **`NSXPCListener`**
  - **`NSXPCListenerDelegate`**
  - **`NSXPCConnection`**

NSXPC Interfaces

- Use Objective-C protocols to define programmatic interface between the calling application and service
- Supported types of interface parameters
  - Arithmetic types, BOOL
  - C Strings, Structures, arrays
  - **Custom Objective-C objects that implement the `NSSecureCoding` protocol**

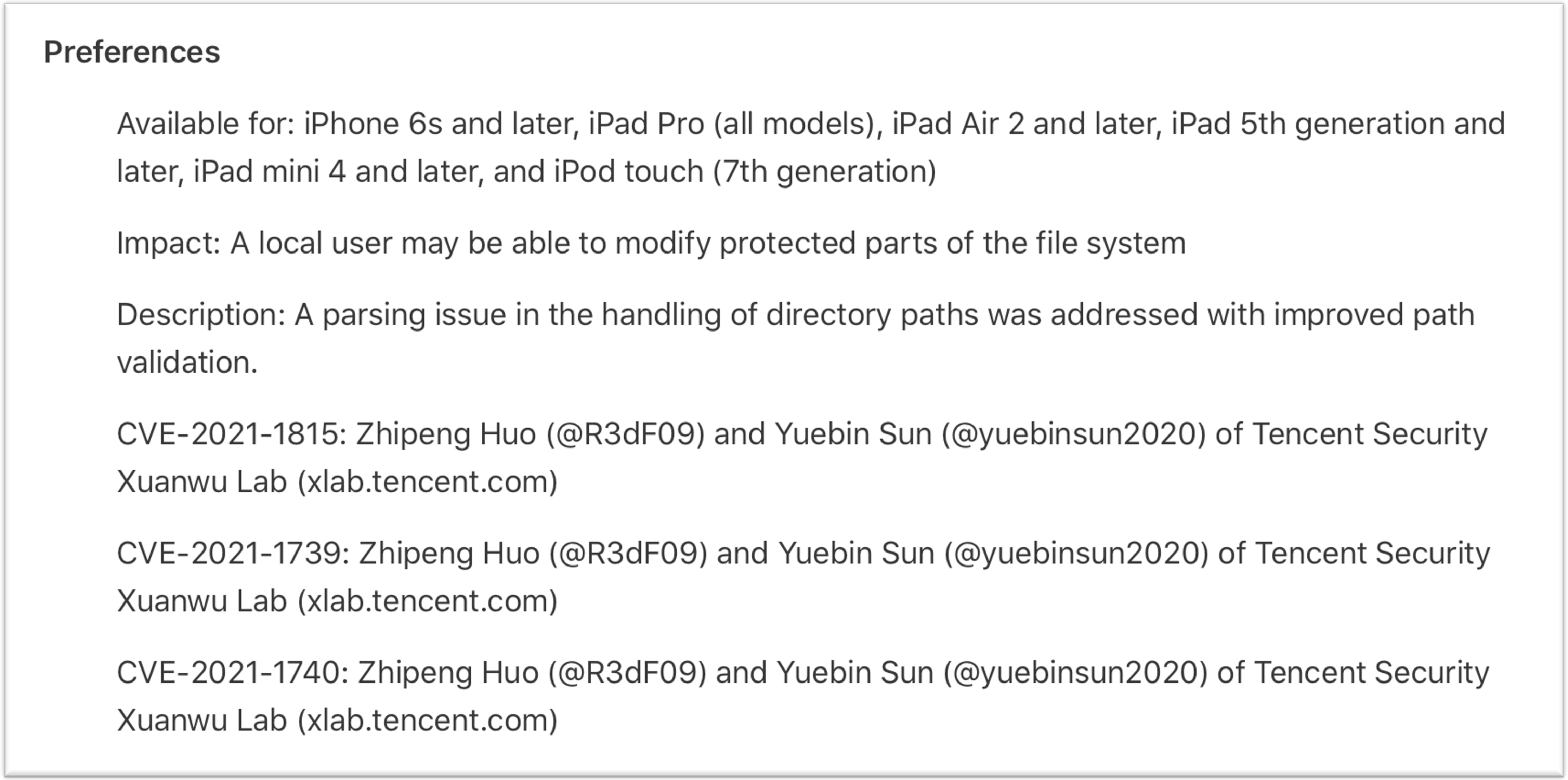
```
@protocol ISDownloadService <NSObject>
- (void)setStoreClient:(ISStoreClient*)storeClient;
- (void)performDownload:(SSDownload*)download
             withOptions:(NSUInteger)options
              replyBlock:(void (^)(NSUInteger,NSError*))reply;
@end
```

NSXPC Architecture



https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingXPCServices.html

IPC Logic Vulnerabilities


Preferences

Available for: iPhone 6s and later, iPad Pro (all models), iPad Air 2 and later, iPad 5th generation and later, iPad mini 4 and later, and iPod touch (7th generation)

Impact: A local user may be able to modify protected parts of the file system

Description: A parsing issue in the handling of directory paths was addressed with improved path validation.

CVE-2021-1815: Zhipeng Huo (@R3dF09) and Yuebin Sun (@yuebinsun2020) of Tencent Security Xuanwu Lab (xlab.tencent.com)

CVE-2021-1739: Zhipeng Huo (@R3dF09) and Yuebin Sun (@yuebinsun2020) of Tencent Security Xuanwu Lab (xlab.tencent.com)

CVE-2021-1740: Zhipeng Huo (@R3dF09) and Yuebin Sun (@yuebinsun2020) of Tencent Security Xuanwu Lab (xlab.tencent.com)

https://support.apple.com/en-us/HT212317

What are Preferences?

- Preferences are user-defined settings
  - Persistent data stored in preferences file
  - Property list – “**`plist`**”
- Service `/usr/sbin/cfprefsd` manages preferences
  - Reads / writes preferences by user requests

How does App Get/Set Preferences Values?

- Foundation API
  - **`NSUserDefaults`**

```
NSUserDefaults* defaults = [NSUserDefaults standardUserDefaults];
[defaults setBool:YES forKey:@"CacheDataAggressively"];
```

- Core Foundation API
  - **`CFPreferencesSetAppValue`**
  - **`CFPreferencesCopyAppValue`**

```
CFStringRef textColorKey = CFSTR("defaultTextColor");
CFStringRef colorBLUE = CFSTR("BLUE");
// Set up the preference.
CFPreferencesSetAppValue(textColorKey, colorBLUE,
        kCFPreferencesCurrentApplication);
// Read the preference.
textColor = (CFStringRef)CFPreferencesCopyAppValue(textColorKey,
        kCFPreferencesCurrentApplication);
```

cfprefsd Handle Requests as a XPC Server

```
service = xpc_connection_create_mach_service(
        "com.apple.cfprefsd.daemon",
        0,
        XPC_CONNECTION_MACH_SERVICE_LISTENER
);
```

```
handler[0] = _NSConcreteStackBlock;
handler[1] = 0xC2000000LL;
handler[2] = &__39__CFPrefsDaemon_initWithRole_testMode___block_invoke_2;
handler[3] = &__block_descriptor_40_e8_32o_e33_v16__0__NSObject_OS_xpc_object__8l;
handler[4] = v7;
xpc_connection_set_event_handler(service, handler);
```

com.apple.cfprefsd.daemon run with root privilege without sandbox

Directly Message **cfprefsd**

```
xpc_connection_t conn = xpc_connection_create_mach_service(
    "com.apple.cfprefsd.daemon", NULL, 0
);

xpc_object_t msg = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_string(msg, "CFPreferencesUser", "kCFPreferencesCurrentUser");
xpc_dictionary_set_int64(msg, "CFPreferencesOperation", 1);
xpc_dictionary_set_string(msg, "Key", "hello");
xpc_dictionary_set_data(msg, "Value", "world", 5);
xpc_dictionary_set_bool(msg, "CFPreferencesCurrentApplicationDomain", true);
xpc_dictionary_set_string(msg, "CFPreferencesDomain", domain);
int fd = open("/tmp/xlab.plist", O_RDWR);
xpc_dictionary_set_fd(msg, "CFPreferencesAccessToken", fd);

xpc_connection_send_message(conn, msg);
```

Where does **`cfprefsd`** Save Preferences Data?

Preferences File Path Construction

- `PreferencesDirectory`
  - **`kCFPreferencesAnyUser`** `“/Library/Preferences”`
  - **`kCFPreferencesCurrentUser`** `“~/Library/Preferences”`
- `PreferencesDomain`
  - XPC Message - `“CFPreferencesDomain”`
- `”CFPreferencesIsByHost”: True`
  - **`PreferencesDirectory`** `+ (`**`PreferencesDomain`** `+ “.” +` **`HostIdentifier`**`) + “.plist”`
- `“CFPreferencesIsByHost”: False` (Default)
  - **`PreferencesDirectory`** `+` **`PreferencesDomain`** `+ “.plist”`

Implementation of Preferences File Path

- **CFStringCreateWithFormat**

```
CFStringRef filePath = CFStringCreateWithFormat(kCFAllocatorDefault,NULL,
    CFSTR("%@.plist"), PreferencesDomain
);
```

- **CFURLCreateWithFileSystemPathRelativeToBase**
  - baseURL - PreferencesDirectory
  - filePath

```
CFURLRef plist_url = CFURLCreateWithFileSystemPathRelativeToBase (
    kCFAllocatorDefault,
    filePath, // fully controllable
    kCFURLPOSIXPathStyle, // pathStyle
    true, // isDirectory
    baseURL // PreferencesDirectory
);
```

Features of **`CFURLCreateWithFileSystemPath-RelativeToBase`**

`filePath` path traversal with "../"

$$\underbrace{\texttt{/Library/Preferences} \quad \texttt{../../tmp/xlab.plist}}_{\texttt{/tmp/xlab.plist}}$$

`baseURL` `filePath`

`filePath` is absolute path

$$\underbrace{\texttt{/Library/Preferences} \quad \texttt{/tmp/xlab.plist}}_{\texttt{/tmp/xlab.plist}}$$

`baseURL` `filePath`

Preferences file path is absolutely controllable

What if Preferences File Path does not Exist?

```
[CFPDSource cacheActualPathCreatingIfNecessary:euid:egid:isWritable:](...)
{
    if ( open(plist_path, O_CREATE, v18) >= 0 ){  ← 1. open preferences file
        return;
    }                                              2. create preferences directory
    PathComponent = CFURLCreateCopyDeletingLastPathComponent(, v15);
    _CFPrefsCreatePreferencesDirectory(PathComponent, a4, a5);

    v16 = open(plist_path, O_CREATE, 0x384); ← 3. open preferences file again
}
```

CFPrefsCreatePreferencesDirectory

```
int _CFPrefsCreatePreferencesDirectory(path, uid, gid) {
    int dirfd = open("/", O_DIRECTORY);
    for (slice in path.split("/")) {
        int fd = openat(dirfd, slice, O_DIRECTORY);
        if (fd == -1 && errno == ENOENT && !mkdirat(dirfd, slice, perm)) {
            fd = openat(dirfd, slice, O_DIRECTORY|O_NOFOLLOW);
            if ( fd == -1 ) return -1;
            fchown(fd, uid, gid);
        }
    }
}
```

Ownership of Preferences Directory

- Default, ownership is the caller of request
  - **`xpc_connection_get_euid`**
  - **`xpc_connection_get_egid`**
- Other, ownership is controllable
  - **`CFPreferencesUseCorrectOwner`** `== True`
  - **`CFPreferencesUser`** `== ‘root’`
  - **`getpwnam`**
    - `pw_uid`
    - `pw_gid`

CVE-2021-1815 Create arbitrary directories with controlled ownership

Exploit of CVE-2021-1815

- Periodic scripts
  - The method mentioned by **Csaba Fitzl**
  - Create directory `/usr/local/etc/periodic/daily` with current user’s privilege
  - Write a script file in `/usr/local/etc/periodic/daily` directory
  - Wait a day
  - The script would run as root

https://www.offensive-security.com/offsec/macos-preferences-priv-escalation/

Patch of CVE-2021-1815

~~cacheActualPathCreatingIfNecessary:euid:egid:isWritable:~~

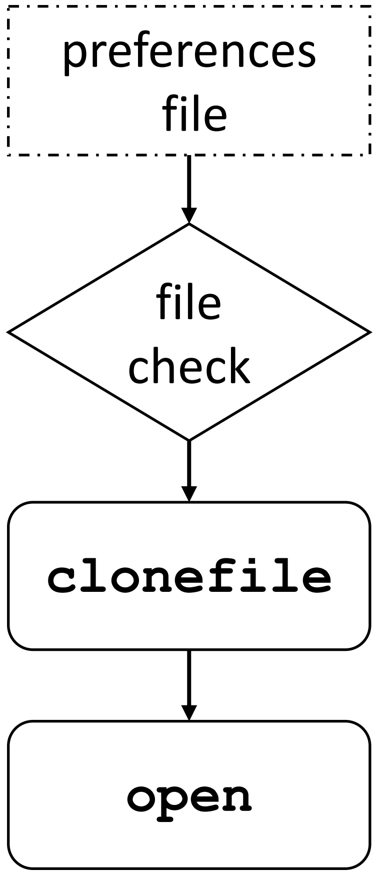
↓

cacheFileInfoForWriting:euid:egid:didCreate:

How does **`cfprefsd`** Read Preferences Data?

Preferences Read Logic

- How does **`cfprefsd`** returns preferences file data ?
  - Put the preferences data in the reply directly
- What if the preferences file is too large ?
  - Open the file and return file descriptor
  - What if the preferences file changes ?
    - Clone file and open
    - Return the file descriptor



Implementation of Preferences File Read

```
char __str[1032];
plist_path = [CFPDSource cacheActualPath];
if (!lstat(plist_path, &stat_buf) && stat_buf.st_size >= 0x100000){
    snprintf(__str, 0x400uLL, "%s.cfp.XXXXXXX", plist_path);
    tmp_plist_path = mktemp(__str);
    if (tmp_plist_path) {
        if (!clonefile(plist_path, tmp_plist_path, 0) ){
            v4 = open(tmp_plist_path, 0);
            ...
        }
    }
}
```

1. get preferences file path

2. judge the file size

3. clone preferences file

```
[CFPDSource cloneAndOpenPropertyListWithoutDrainingPendingChangesOrValidatingPlist]
```

Implementation of File Clone

```
clonefile(plist_path, tmp_path)
```

```
snprintf(__str, 0x400uLL, "%s.cfp.XXXXXXX", plist_path);
dst_path = mktemp(__str);
```

Random file name at same directory of preferences file

Implementation of File Clone

```
clonefile(plist_path, tmp_path)
```

```
snprintf(__str, 0x400uLL, "%s.cfp.XXXXXXX", plist_path);
dst_path = mktemp(__str);
```

Random file name?

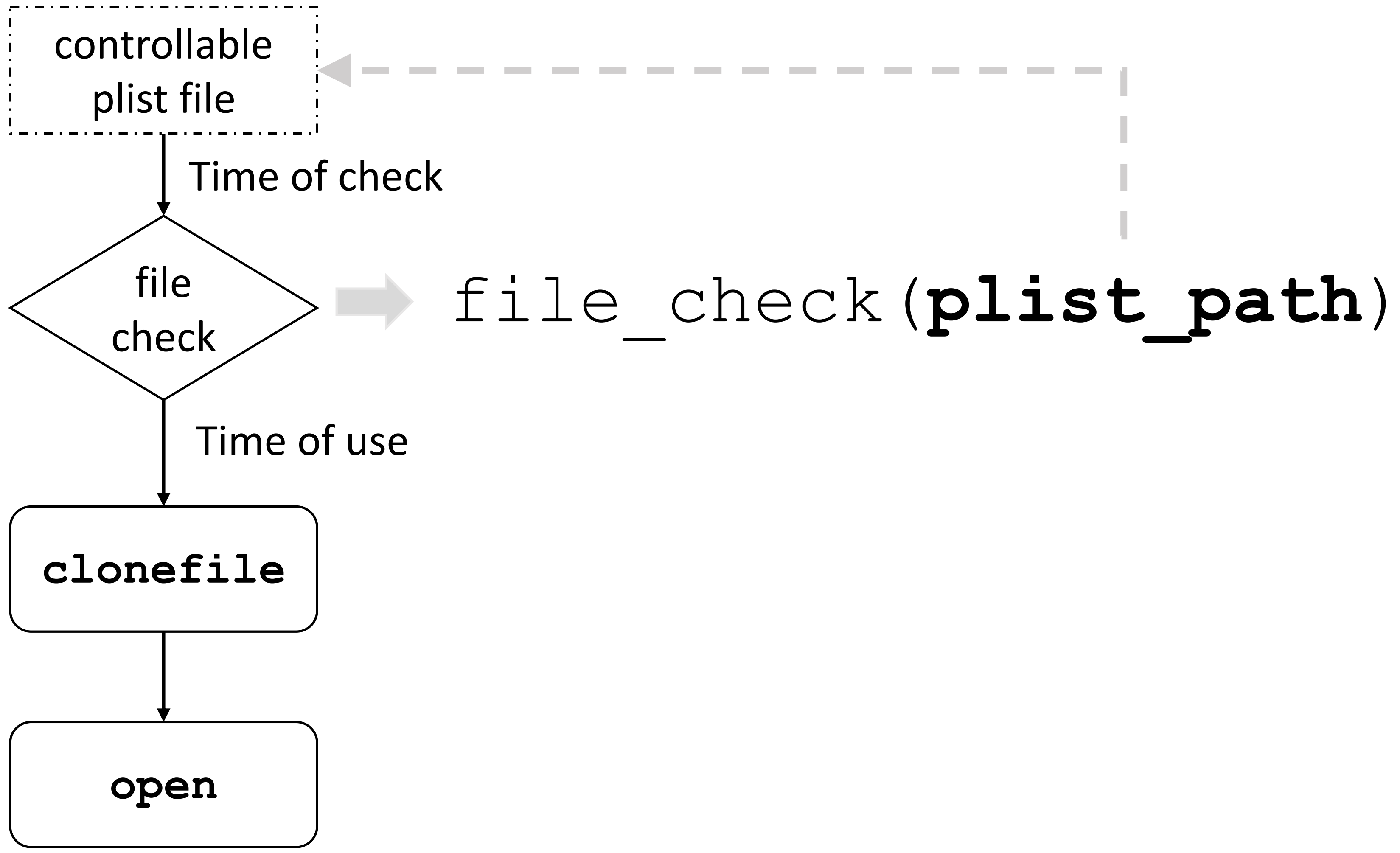
Really Random File Name?

```
snprintf(__str, 0x400uLL, "%s.cfp.XXXXXXX", plist_path);
dst_path = mktemp(__str);
```

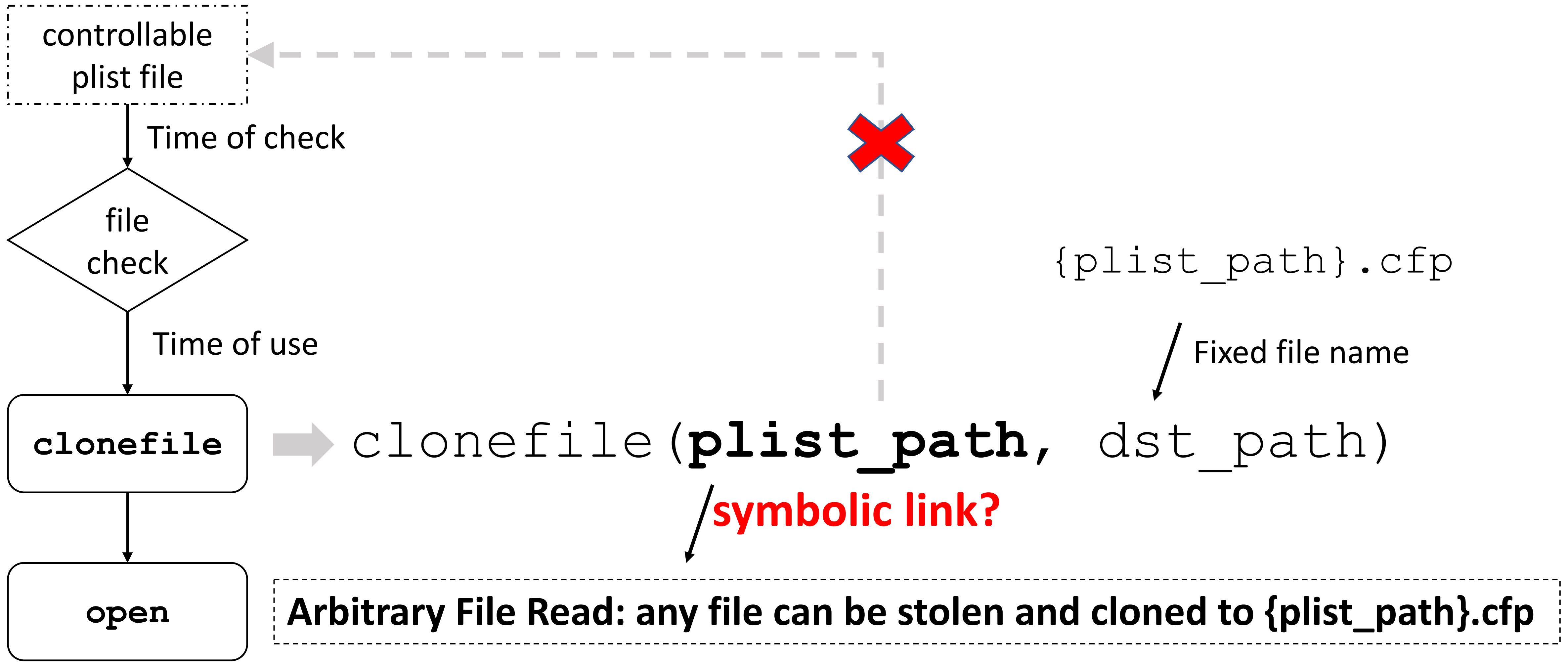
- **`snprintf`**
  - Generate formatted string with its MAX length as `0x400`
- **`mktemp`**
  - “The trailing `X’s are replaced with a unique alphanumeric combination”

- What if **`strlen`**`(plist_path) + `**`strlen`**`(”.cfp”)` is equal to `0x400-1`?
  - **`snprintf`** will overflows and generate ”_str” without ”X”
  - **`mktemp`** will returns a fixed filename without any randomness

Time of Check



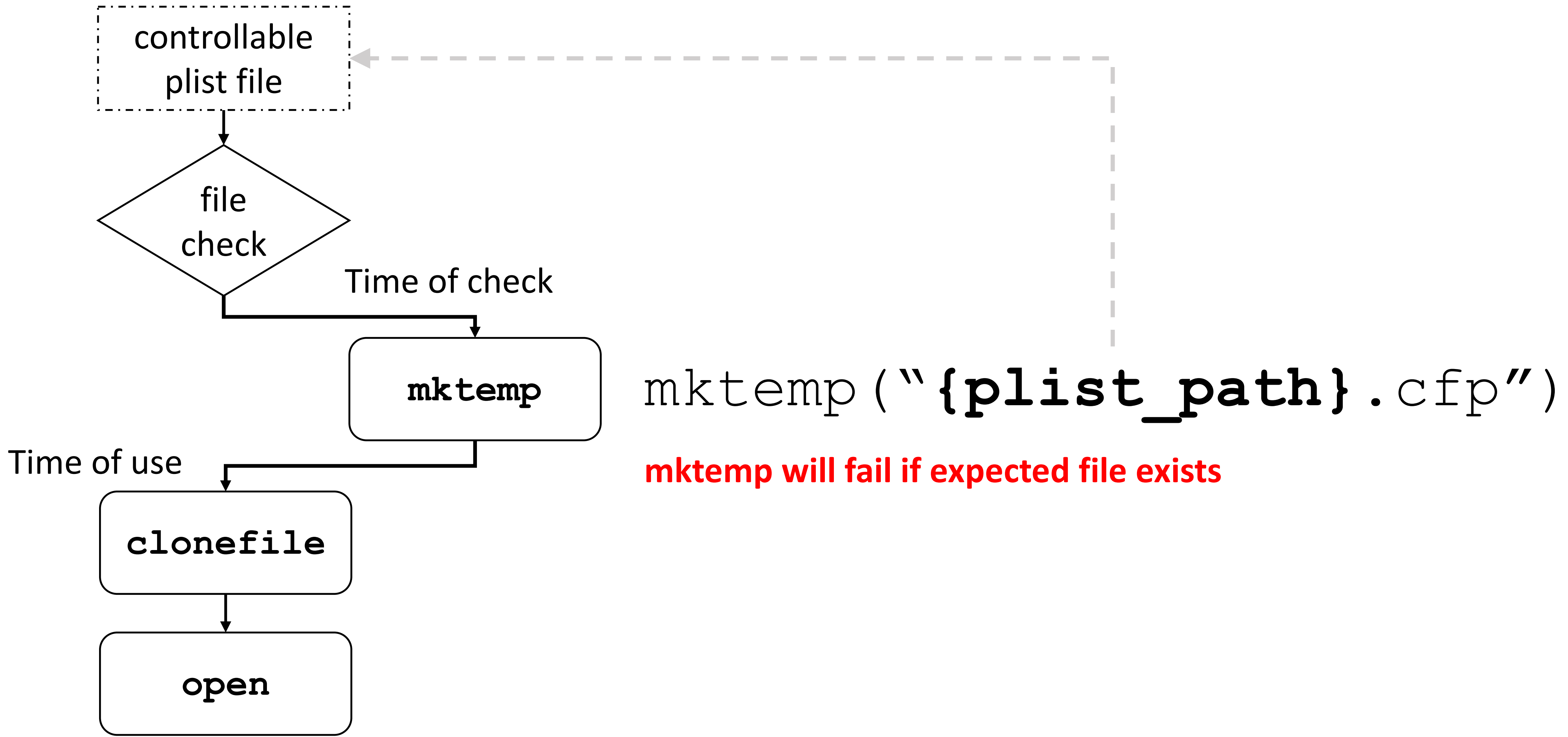
Arbitrary File Read



```
clonefile(plist_path, dst_path)
```

Arbitrary File Read: any file can be stolen and cloned to {plist_path}.cfp

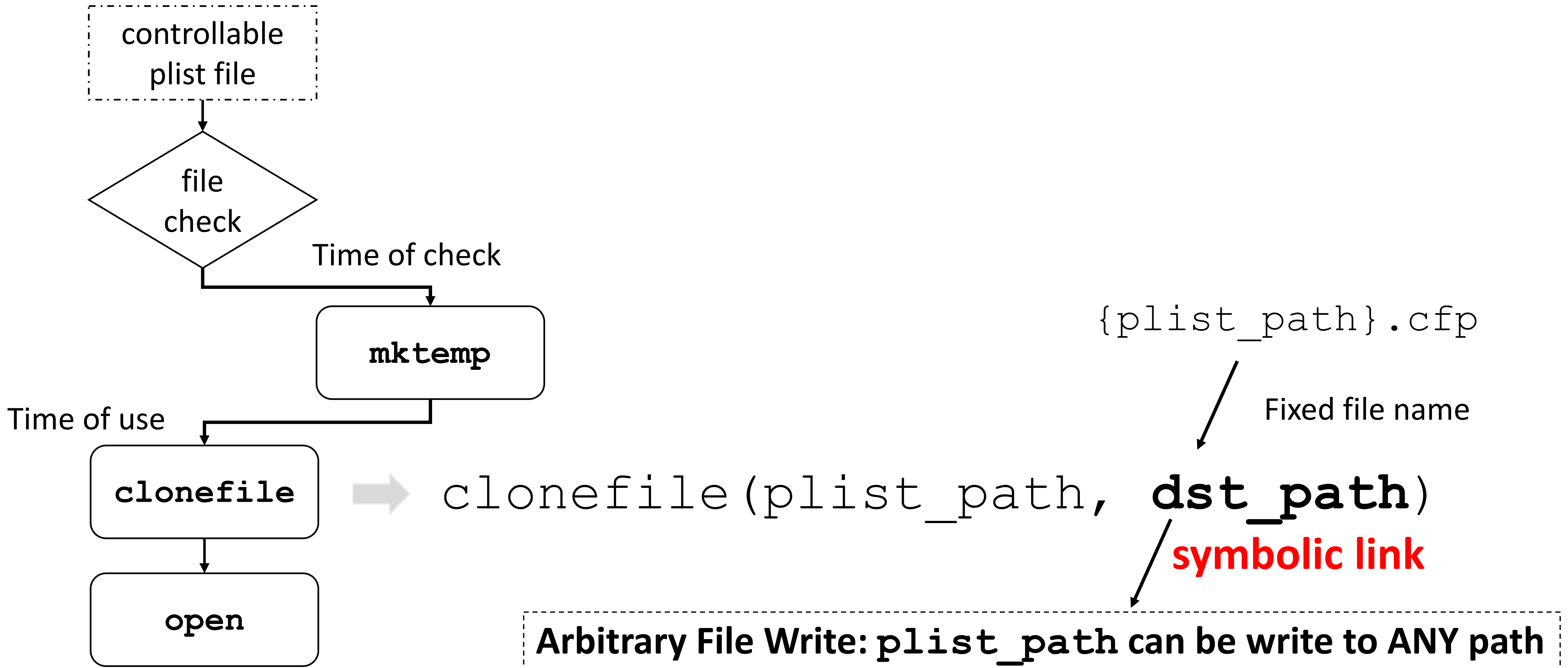
Make Temporary File Name



```
mktemp("{plist_path}.cfp")
```

mktemp will fail if expected file exists

Arbitrary File Write



```
clonefile(plist_path, dst_path)
```

Arbitrary File Write: `plist_path` can be write to ANY path

Patch of CVE-2021-1740

- Replace temp path with symbolic link? No more fixed file
  - Predicted/fixed temp path need to overflow **`snprintf`**
  - Random temp file will not be created unless **`clonefileat`** is called successfully, so we have no time window to replace it

```
int ret = snprintf(__str, 0x400uLL, "%s.cfp.XXXXXXX", plist_path);
if (ret >= 0x400){
    goto FAIL;
}
char *temp_path = mktemp(__str);
if (!clonefileat(dirfd, plist_file, AT_FDCWD, temp_path)){
    int fd = open(temp_path, 0);
    // ...
    return fd;
}
```

How does **`cfprefsd`** Write Preferences Data?

Preferences Write Logic

- Parse key, value from XPC message
- Read data from target preferences file
- Generate new data
- Write data to a temp file
- Rename temp file back to the target preferences file

Client have Write Permission to Preferences File?

The client needs to pass the file descriptor to **`cfprefsd`** to prove that it has write permission to preferences file

```
bool -[CFPDSource validateAccessToken:accessType:]{
    char fd_path[1024];
    xpc_fd = xpc_dictionary_dup_fd(xpc_msg, "CFPreferencesAccessToken");
    if (fcntl(xpc_fd, F_GETPATH, fd_path) != -1){
        // check if path is consistent, plist_path is controllable by client
        if (!strcmp(fd_path, plist_path) &&
           // check if the file is writable by client
            ((fcntl(xpc_fd, F_GETFL, 0LL) & 3) == 2)){
            return true; // check success
        }
    }
    return false; // check failed
}
```

Implementation of Preferences File Write

```
int64 _CFPrefsWritePlistToFDThenClose(){
                                                       1. generate temp file
    tmp_file_fd = _CFPrefsTemporaryFDToWriteTo(v3, v4);

    fcntl(tmp_file_fd, F_GETPATH, tmp_file_path);
    while (...) {
                                              2. write preferences data to temp file
        write(tmp_file_fd, plist_data, plist_size);
    }
    close(tmp_file_fd);
                                         3. rename temp file to target plist file
    rename(tmp_file_path, plist_path);
}
```

Source Path is Symbolic Link?

Created in temp directory

Fully controllable by client

```
rename(tmp_file_path, plist_path)
```

Symbolic link?

No, created in root-owned directory, client has no access to it

Target Path is Symbolic Link?

Created in temp directory

Fully controllable by client

```
rename(tmp_file_path, plist_path)
```

Symbolic link?

`rename` will firstly delete the symbolic link if it exists already

“If the **final component** of target is a symbolic link, the symbolic link is renamed, not the file or directory to which it points.”

Target Path is Symbolic Link?

Created in temp directory

Fully controllable by client

```
rename(tmp_file_path, plist_path)
```

Symbolic link?

`rename` will firstly delete the symbolic link if it exists already

"If the **final component** of target is a symbolic link, the symbolic link is renamed, not the file or directory to which it points."

What if Middle Component of **`plist_path`** is a Symbolic Link?

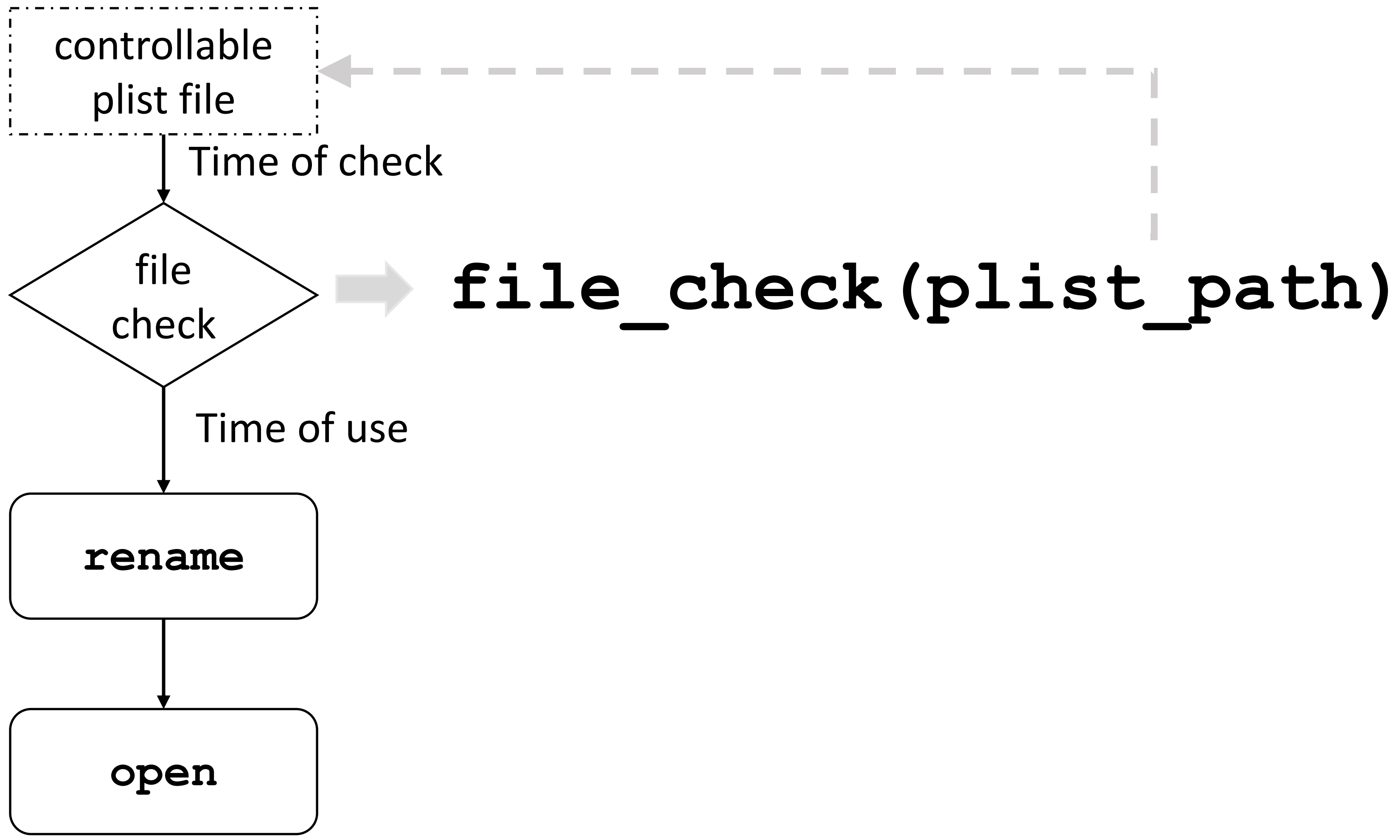
```
rename(tmp_file_path, plist_path)
```

```
/tmp/test/hello.plist
```

middle component **final component**

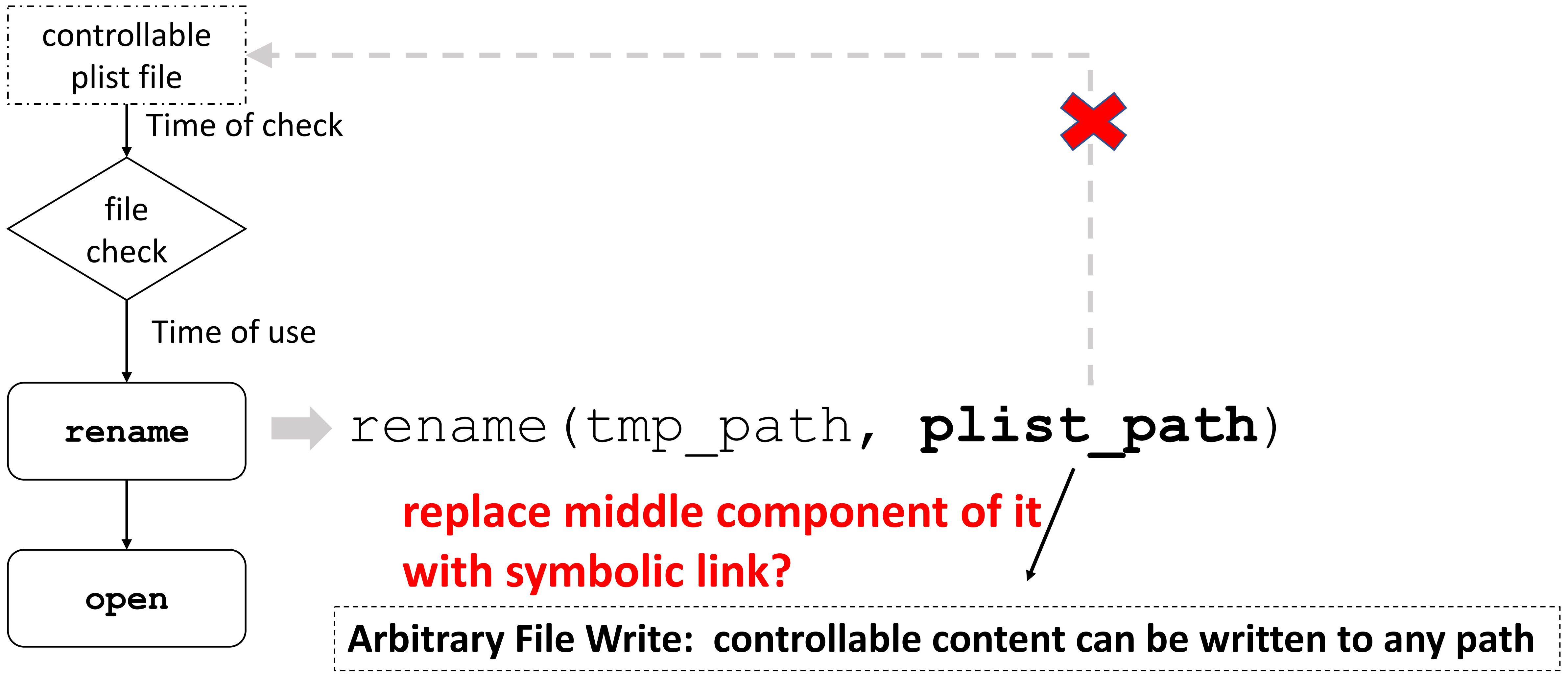
```
➜  ln -s /Library/LaunchDaemons /tmp/test
➜  ls -l /tmp/
lrwxr-xr-x  1 xuanwulab  wheel   22  6 29 18:07 test -> /Library/LaunchDaemons
```

Time of Check



```
file_check(plist_path)
```

Arbitrary File Rename



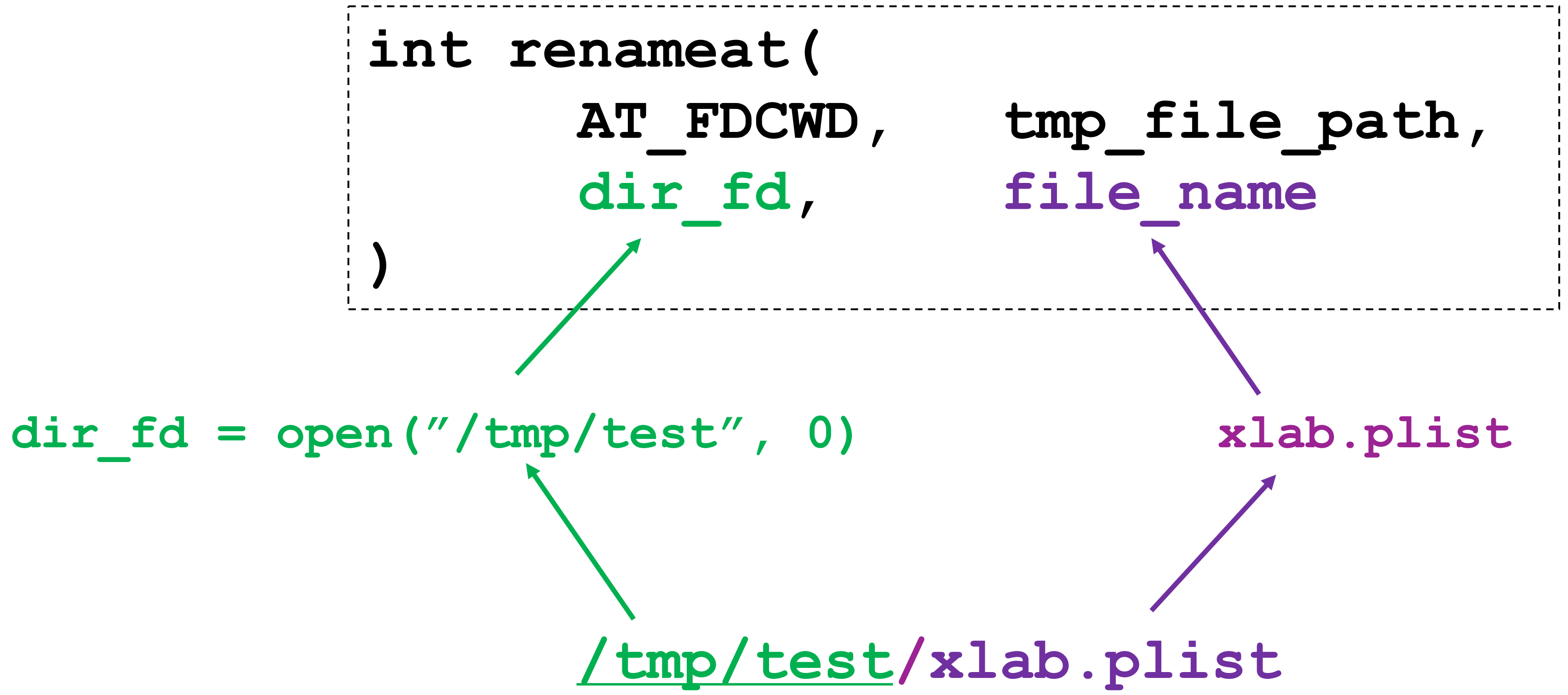
```
rename(tmp_path, plist_path)
```

replace middle component of it with symbolic link?

Arbitrary File Write: controllable content can be written to any path

Patch of CVE-2021-1739

- **rename** -> **renameat**
  - The target file of **renameat** will be created based on dir_fd
  - Symbolic link (**middle component** of the target path) will not be followed



```
int renameat(
        AT_FDCWD,    tmp_file_path,
        dir_fd,      file_name
)
```

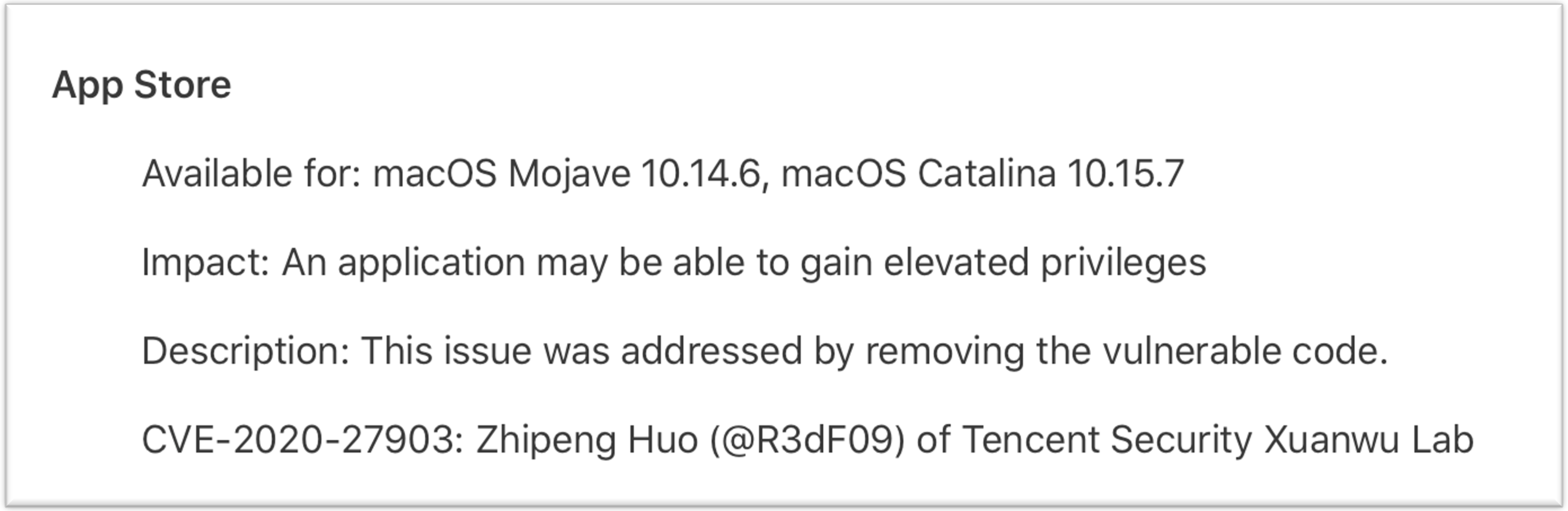
```
dir_fd = open("/tmp/test", 0)
```

```
xlab.plist
```

```
/tmp/test/xlab.plist
```

Demo of Preferences Vulnerabilities

https://www.youtube.com/watch?v=Kh6sEcdGruU


App Store

Available for: macOS Mojave 10.14.6, macOS Catalina 10.15.7

Impact: An application may be able to gain elevated privileges

Description: This issue was addressed by removing the vulnerable code.

CVE-2020-27903: Zhipeng Huo (@R3dF09) of Tencent Security Xuanwu Lab

https://support.apple.com/en-us/HT212011

NSXPC Server

```
[NSXPCListener initWithMachServiceName: @"com.apple.storedownloadd.daemon”];
```

- **`com.apple.storedownloadd.daemon`**
- `/System/Library/PrivateFrameworks/CommerceKit.framework/Versions/A/Resources/storedownloadd`
- Root privilege, but sandboxed
  - Sandbox Profile `/System/Library/Sandbox/Profiles/com.apple.storedownloadd.sb`
  - It is allowed to write many sensitive paths such as `/Applications`, `/Library/Keychains/`

`storedownloadd`’s Interfaces

```
@protocol ISDownloadService <NSObject>
    - (void)setStoreClient:(ISStoreClient*)storeClient;
    - (void)performDownload:(SSDownload*)download
                          withOptions:(NSUInteger)options
                   replyBlock:(void (^)(NSUInteger, NSError*))reply;
@end
```

What to download ? What is **`SSDownload`** ?

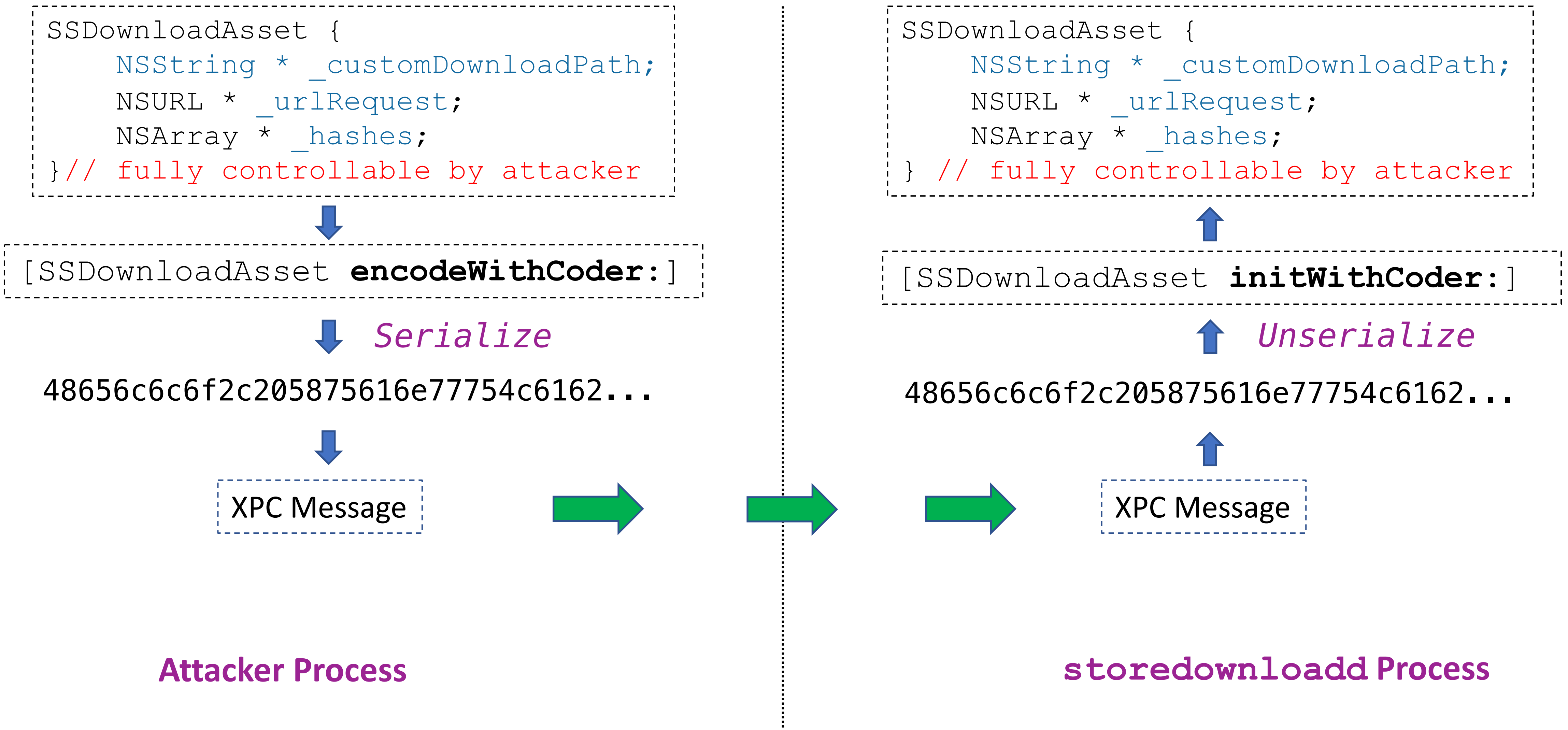
SSDownload

```
@interface SSDownload : NSObject<NSSecureCoding>
@property(copy, nonatomic) NSArray *_assets;
@end
@implementation SSDownload
+ (BOOL)supportsSecureCoding{
return YES;
}
- (void)encodeWithCoder:(nonnull NSCoder *)coder {
[coder encodeObject:self._assets forKey:@"_assets"];
}
- (nullable instancetype)initWithCoder:(nonnull NSCoder *)coder {
return self;
}
@end
```

SSDownloadAsset

```
@interface SSDownloadAsset : NSObject<NSSecureCoding>
    @property NSString * _customDownloadPath;
    @property NSURL * _urlRequest;
    @property NSArray * _hashes;
@end
@implementation SSDownloadAsset
    + (BOOL)supportsSecureCoding{
        return YES;
    }
    - (void)encodeWithCoder:(nonnull NSCoder *)coder {
        [coder encodeObject:self._customDownloadPath forKey:@"download-path"];
        [coder encodeObject:self._urlRequest forKey:@"url"];
        [coder encodeObject:self._hashes forKey:@"hashes"];
    }
    - (nullable instancetype)initWithCoder:(nonnull NSCoder *)coder {
        return self;
    }
@end
```

Serialization and Unserialization



```
SSDownloadAsset {
    NSString * _customDownloadPath;
    NSURL * _urlRequest;
    NSArray * _hashes;
}// fully controllable by attacker
```

```
[SSDownloadAsset encodeWithCoder:]
```

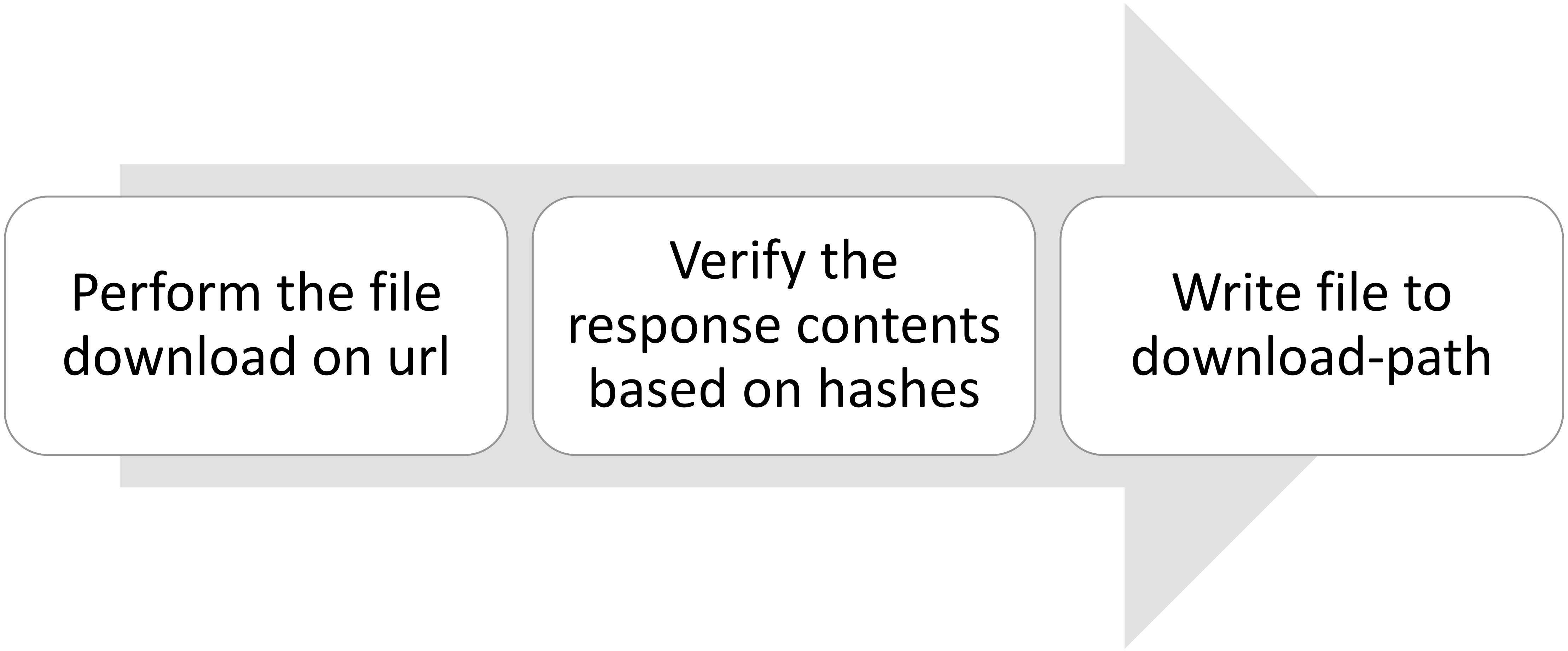
```
48656c6c6f2c205875616e77754c6162...
```

```
SSDownloadAsset {
    NSString * _customDownloadPath;
    NSURL * _urlRequest;
    NSArray * _hashes;
} // fully controllable by attacker
```

```
[SSDownloadAsset initWithCoder:]
```

```
48656c6c6f2c205875616e77754c6162...
```

Download Logic


Perform the file download on url
Verify the response contents based on hashes
Write file to download-path

Hash Verification

```
-[HashedDownloadProvider
_verifyStreamedBytesWithHashes:]
```

- Calculate hash of response contents
- Compare with input hash

It’s just a data integrity check !!!

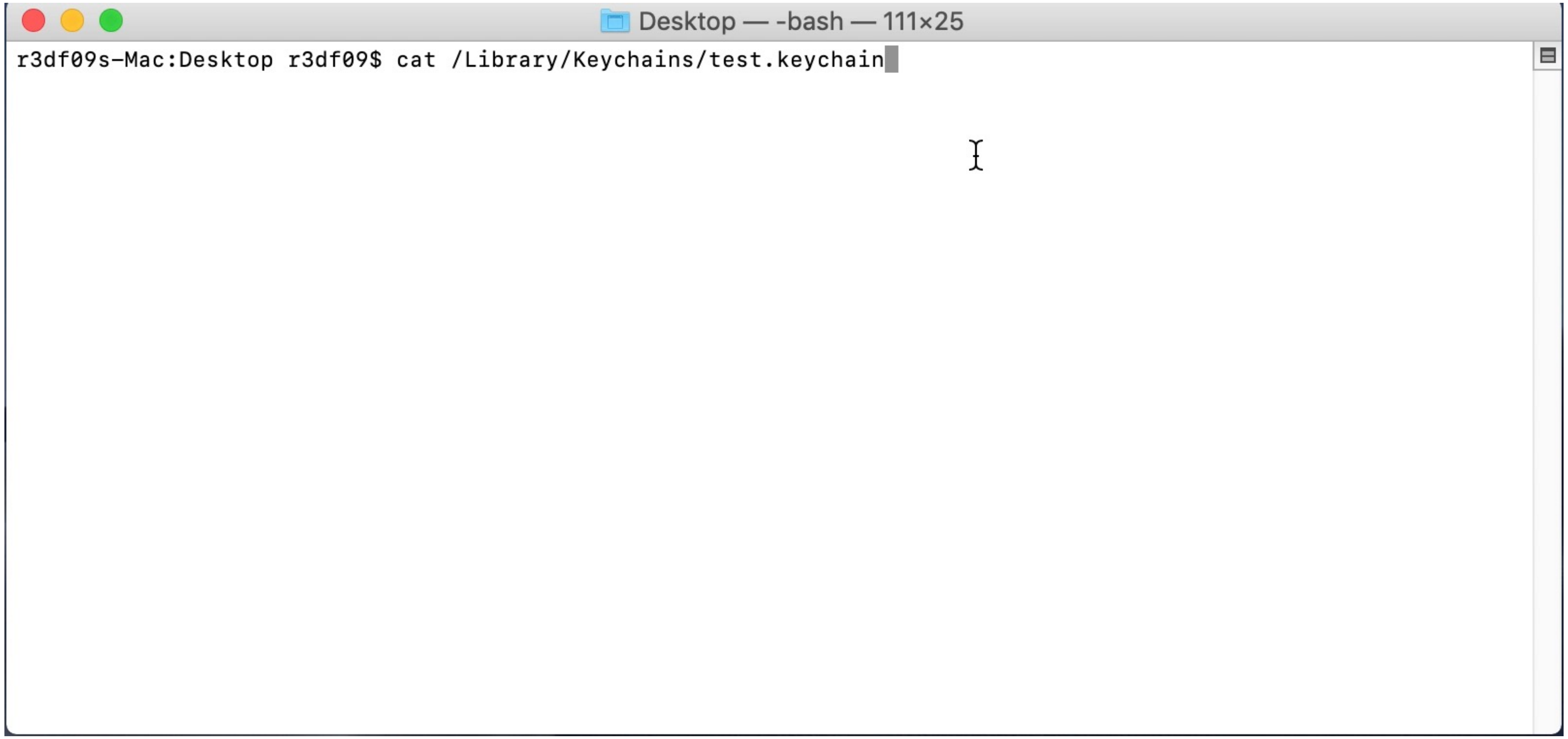
Exploit of CVE-2020-27903

“Hi **`storedownloadd`**, please help me to download a file from this URL path, its hash is balabala ... and then write the contents to this download path, thanks!”

Patch of CVE-2020-27903

- Removing the vulnerable code
- No ~~**`com.apple.storedownloadd.daemon`**~~ any more.
  - RIP

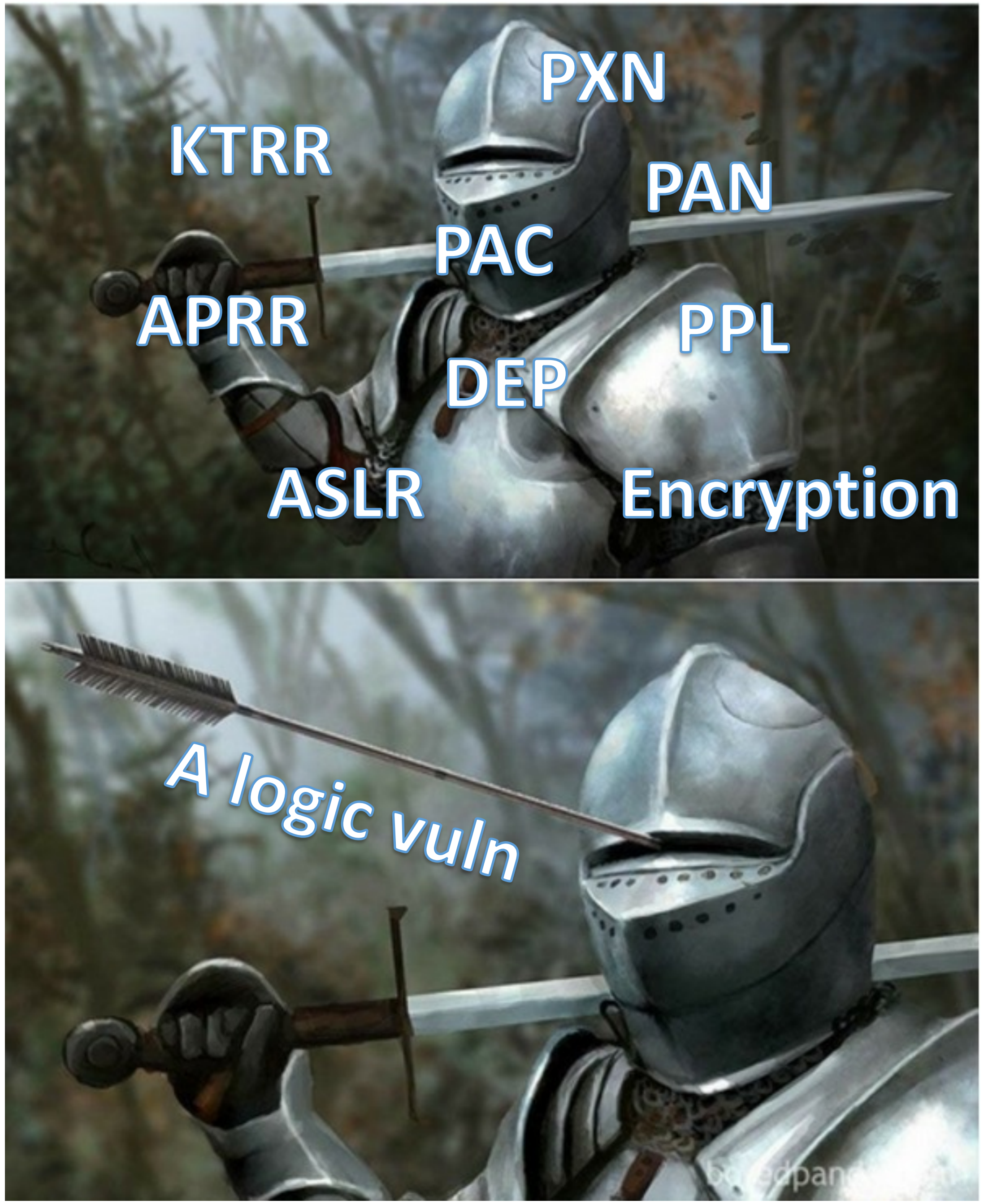
Demo of CVE-2020-27903



```
r3df09s-Mac:Desktop r3df09$ cat /Library/Keychains/test.keychain
```

Other Logic Vulnerabilities

- XPC Service implementation flaw
  - https://xlab.tencent.com/en/2021/01/11/cve-2020-9971-abusing-xpc-service-to-elevate-privilege/
- NSXPC Vulnerabilities in Adobe Acrobat Reader
  - https://rekken.github.io/2020/05/14/Security-Flaws-in-Adobe-Acrobat-Reader-Allow-Malicious-Program-to-Gain-Root-on-macOS-Silently/



https://i.kym-cdn.com/photos/images/newsfeed/001/890/751/e0e.png

Advantage of IPC Logic Vulnerability

- Easy to exploit
- Stable
- One exploit to rule them all

“Logic bugs in core framework like prefrences let us rule all Apple platforms, Intel and Apple Silicon alike, without changing one line of our exploit.”

State of Apple IPC Security

- Reduce the IPC attack surfaces
  - More restricted sandbox rules
  - Delete unnecessary high privilege services
  - Adding more and more private entitlements
    - **`com.apple.private.xxx`**
  - …
- Limit the damage
  - Sandbox IPC Server
  - Rootless
  - …

Conclusion

Latest IPC Mechanisms on Apple Platforms

- XPC, NSXPC

Interesting Apple IPC Logic Vulnerabilities

- Three logic vulnerabilities in Preferences
- One logic vulnerability in App Store

Advantage of IPC Logic Vulnerability

Status of Apple IPC Logic Vulnerability

"Logic bugs are always fun!"

Special Thanks

- Csaba Fitzl (@theevilbit)
- Ian Beer (@i41nbeer)
- Zhi Zhou (@CodeColorist)

Thanks.

Tencent Security Xuanwu Lab

@XuanwuLab

xlab.tencent.com

Tencent 腾讯

腾讯安全玄武实验室

TENCENT SECURITY XUANWU LAB