# MacOS SUHelper Root Privilege Escalation Vulnerability: A Deep Dive Into CVE-2022-22639

We discovered a now-patched vulnerability in macOS SUHelper, designated as CVE-2022-22639. If exploited, the vulnerability could allow malicious actors to gain root privilege escalation.

April 04, 2022

By: Mickey Jin Read time: 4 min (1138 words)

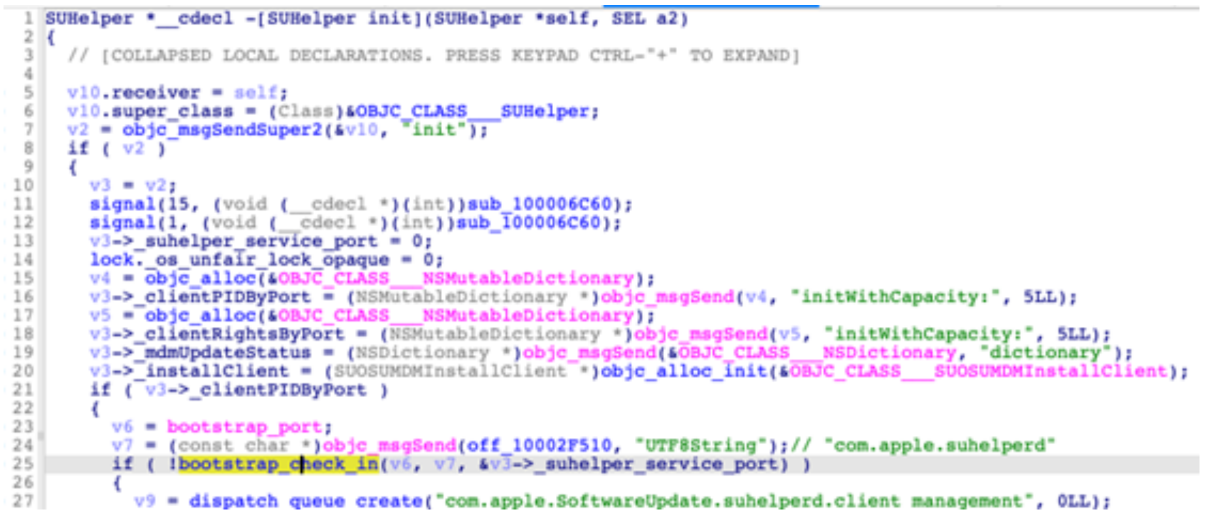
We discovered a vulnerability in suhelperd, a helper daemon process for Software Update in macOS. A class inside suhelperd, SUHelper, provides an essential system service through the inter-process communication (IPC) mechanism. The process runs as root and is signed with special entitlements, such as com.apple.rootless.install, which grants the process permission to bypass System Integrity Protection (SIP) restrictions. This combination of functionalities presents an attractive opportunity for malicious actors to exploit the vulnerability.

Designated as CVE-2022-22639, the vulnerability could allow root privilege escalation if successfully exploited. After discovering the flaw, we reported it to Apple, hence the release of a patch through the macOS Monterey 12.3 security update

This report dives into the daemon process, enumerates all the services it provides, and discusses the vulnerabilities found therein.

## The IPC service

The core logic of the daemon process is to register an IPC service by API bootstrap_check_in, named as com.apple.suhelperd.

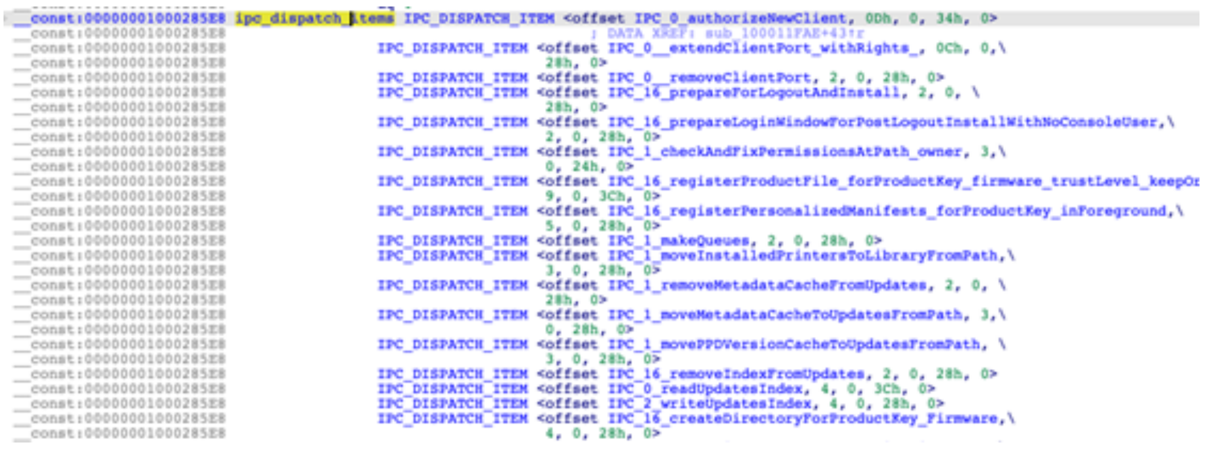


```
SUHelper *__cdecl -[SUHelper init](SUHelper *self, SEL a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v10.receiver = self;
  v10.super_class = (Class)&OBJC_CLASS___SUHelper;
  v2 = objc_msgSendSuper2(&v10, "init");
  if ( v2 )
  {
    v3 = v2;
    signal(15, (void (__cdecl *)(int))sub_100006C60);
    signal(1, (void (__cdecl *)(int))sub_100006C60);
    v3->_suhelper_service_port = 0;
    lock._os_unfair_lock_opaque = 0;
    v4 = objc_alloc(&OBJC_CLASS___NSMutableDictionary);
    v3->_clientPIDByPort = (NSMutableDictionary *)objc_msgSend(v4, "initWithCapacity:", 5LL);
    v5 = objc_alloc(&OBJC_CLASS___NSMutableDictionary);
    v3->_clientRightsByPort = (NSMutableDictionary *)objc_msgSend(v5, "initWithCapacity:", 5LL);
    v3->_mdmUpdateStatus = (NSDictionary *)objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionary");
    v3->_installClient = (SUOSUMDMInstallClient *)objc_alloc_init(&OBJC_CLASS___SUOSUMDMInstallClient);
    if ( v3->_clientPIDByPort )
    {
      v6 = bootstrap_port;
      v7 = (const char *)objc_msgSend(off_10002F510, "UTF8String");// "com.apple.suhelperd"
      if ( !bootstrap_check_in(v6, v7, &v3->_suhelper_service_port) )
      {
        v9 = dispatch queue create("com.apple.SoftwareUpdate.suhelperd.client management", 0LL);
```

Figure 1. SUHelper server initialization

The client process can find the service with names through API bootstrap_look_up, and then request the service routines through the IPC mechanism. (The IPC mechanism is discussed at length in chapter 11 of the book "MacOS and iOS Internals, Volume I: User Mode.")

The IPC server provides 45 service routines, some of which are shown in the following figure. I renamed all the routines using the format IPC_NUMBER_XXX, according to their functions and the corresponding rights, for easy reference.

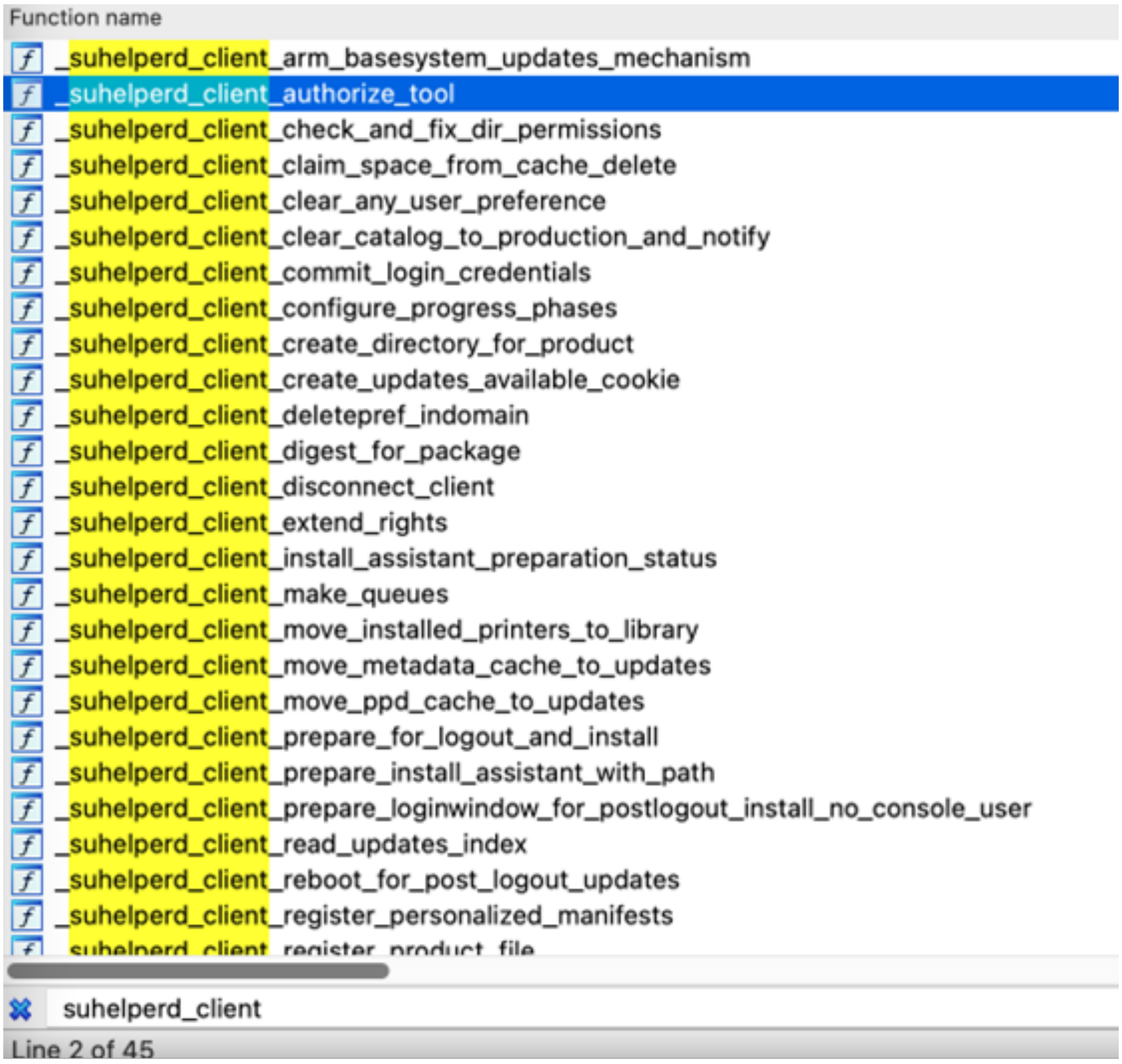


```
__const:00000001000285E8 ipc_dispatch_items IPC_DISPATCH_ITEM <offset IPC_0_authorizeNewClient, 0Dh, 0, 34h, 0>
__const:00000001000285E8                                        ; DATA XREF: sub_100011FAE+43↑r
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_0__extendClientPort_withRights_, 0Ch, 0,\
__const:00000001000285E8                                        28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_0__removeClientPort, 2, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_16_prepareForLogoutAndInstall, 2, 0, \
__const:00000001000285E8                                        28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_16_prepareLoginWindowForPostLogoutInstallWithNoConsoleUser,\
__const:00000001000285E8                                        2, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_1_checkAndFixPermissionsAtPath_owner, 3,\
__const:00000001000285E8                                        0, 24h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_16_registerProductFile_forProductKey_firmware_trustLevel_keepOr
__const:00000001000285E8                                        9, 0, 3Ch, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_16_registerPersonalizedManifests_forProductKey_inForeground,\
__const:00000001000285E8                                        5, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_1_makeQueues, 2, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_1_moveInstalledPrintersToLibraryFromPath,\
__const:00000001000285E8                                        3, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_1_removeMetadataCacheFromUpdates, 2, 0, \
__const:00000001000285E8                                        28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_1_moveMetadataCacheToUpdatesFromPath, 3,\
__const:00000001000285E8                                        0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_1_movePPDVersionCacheToUpdatesFromPath, \
__const:00000001000285E8                                        3, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_16_removeIndexFromUpdates, 2, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_0_readUpdatesIndex, 4, 0, 3Ch, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_2_writeUpdatesIndex, 4, 0, 28h, 0>
__const:00000001000285E8                    IPC_DISPATCH_ITEM <offset IPC_16_createDirectoryForProductKey_Firmware,\
__const:00000001000285E8                                        4, 0, 28h, 0>
```

Figure 2. Some IPC service routines

The IPC client is already implemented in the private

SoftwareUpdate.framework. There are 45 exported functions with a one-to-one correspondence to their respective service routines.



```
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
```

Figure 3. IPC client interfaces

Instead of reinventing the wheel, one can reuse the code from the framework. Fortunately, there is an Objective-C class named SUHelperProxy, which encapsulates all the IPC client interfaces that one can directly use.

The following is an example of a service routine handling flow.

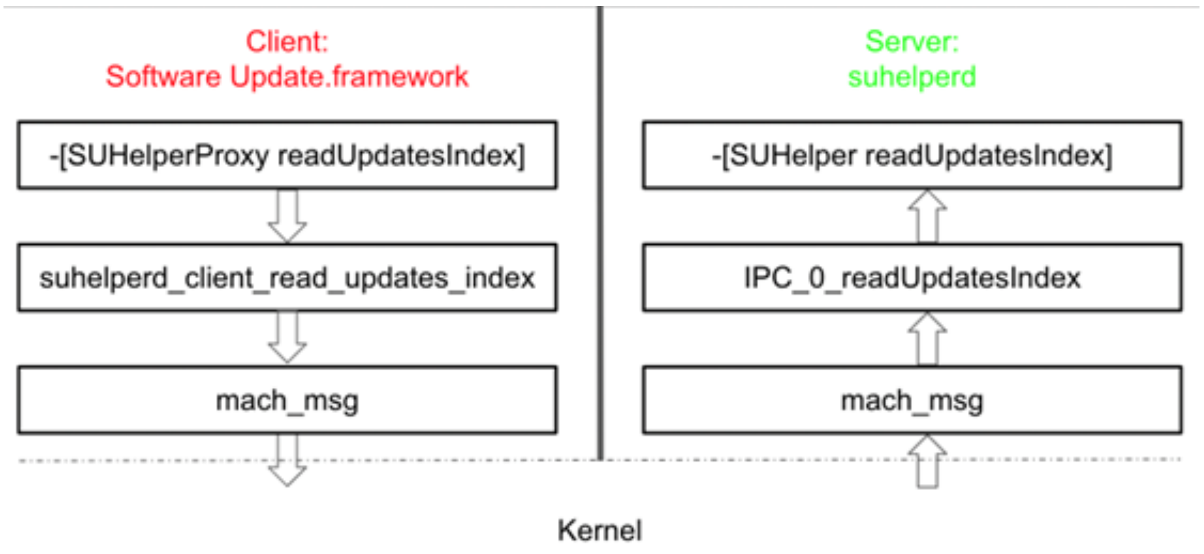
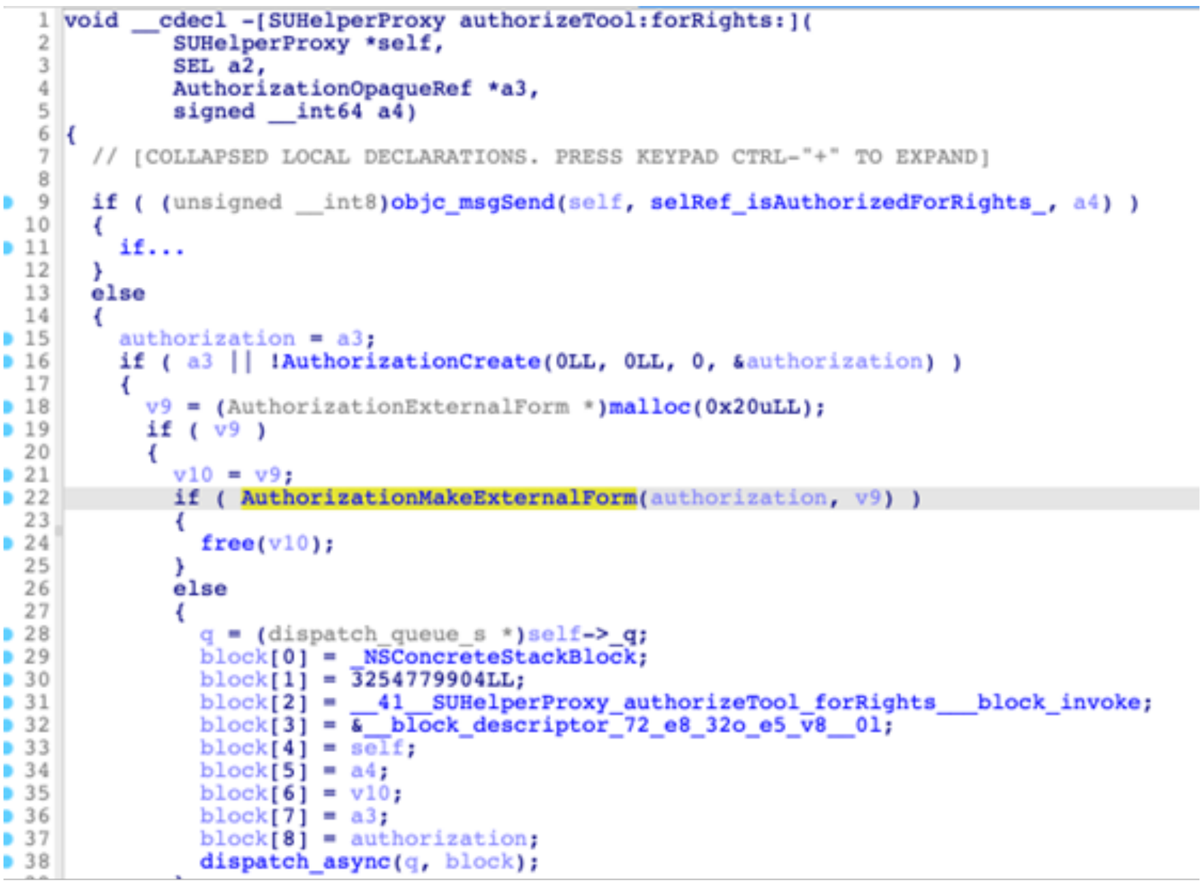

Client:
Software Update.framework
-[SUHelperProxy readUpdatesIndex]
suhelperd_client_read_updates_index
mach_msg
Server:
suhelperd
-[SUHelper readUpdatesIndex]
IPC_0_readUpdatesIndex
mach_msg
Kernel

Figure 4. An IPC handling flow

## Client authorization

It should be noted that not all 45 services are available to unprivileged clients, and that the server has a rights authorization mechanism to verify if a service request is from a legitimate client.

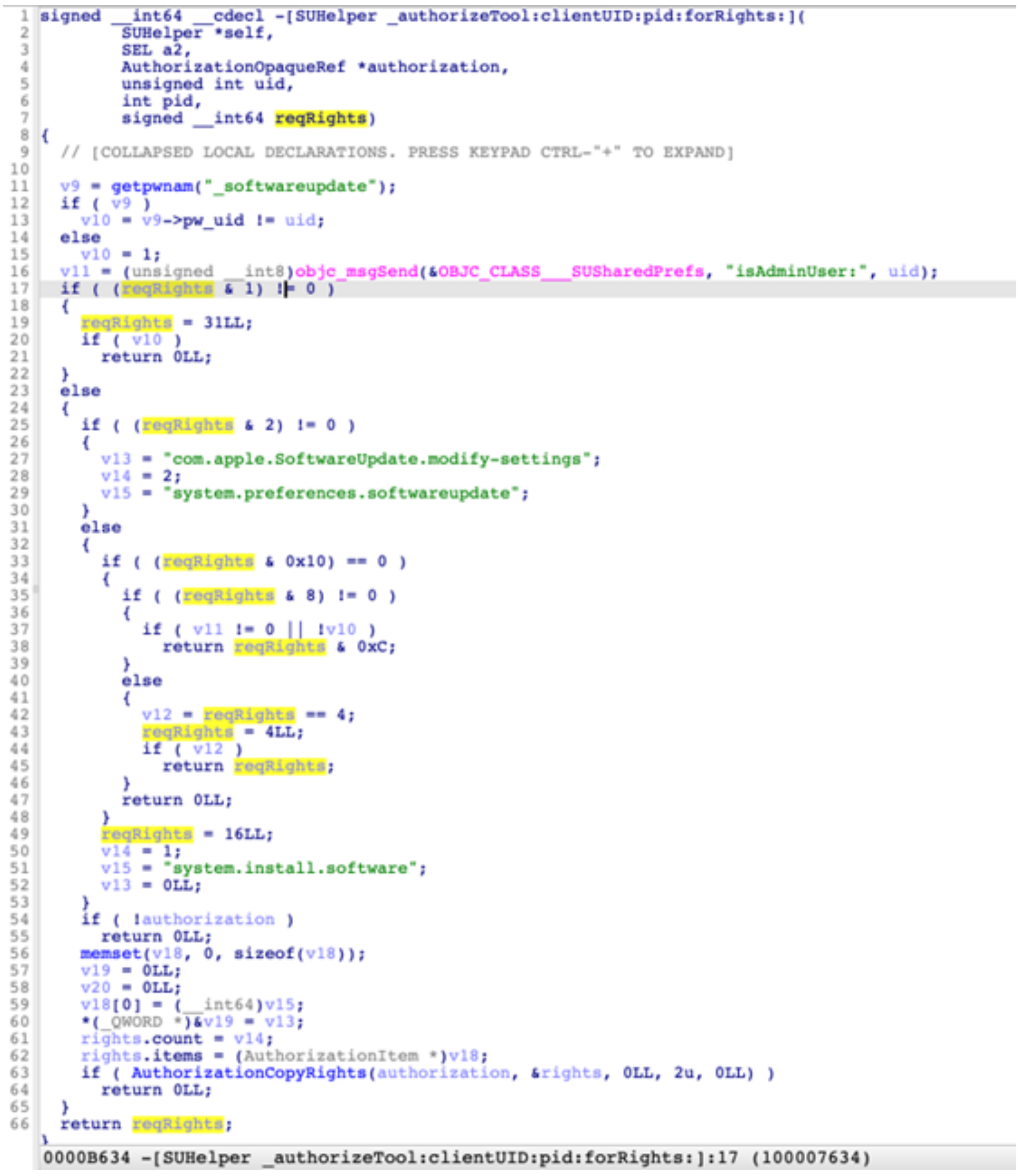
First, the client needs to generate an authorization object by API AuthorizationCreate, and then make it as an external form (32 bytes of data) to transfer the authorization object to the server for verification.



```
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);
```

Figure 5. The client generating the authorization object

Second, when the server receives the authorization object, it determines whether specific rights can be granted to the client. At this stage, the server checks the client’s authorization object and 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)
```

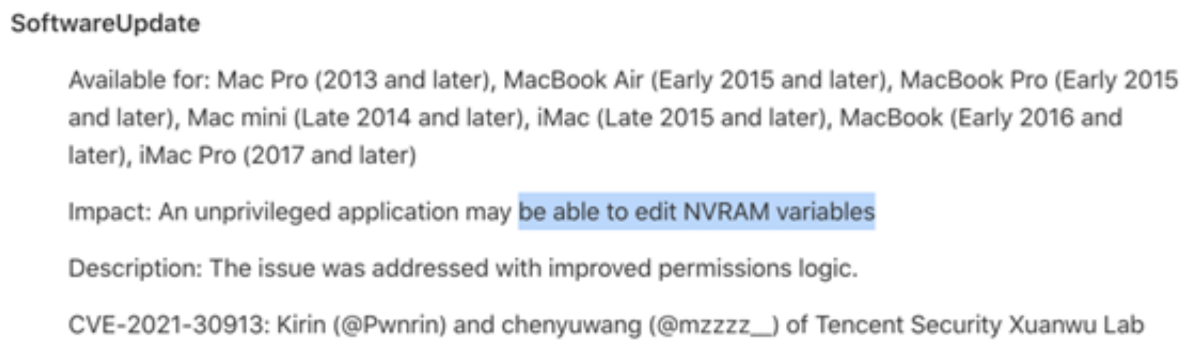
Figure 6. The server verifying the client’s authorization object and uid

Third, when the client requests a special service routine, the server checks whether the specific rights were previously granted to the client, otherwise it denies the request.

## Old vulnerabilities

As mentioned earlier, not all the service routines are allowed because of the requisite client authorization. However, there were some essential routines that were left unprotected because the server did not validate the rights at the third step.

Here are two old vulnerabilities, for example, which were discovered by researchers at Xuanwu Lab. CVE-2021-30913 could allow malicious actors to edit NVRAM variables.



**SoftwareUpdate**

Available for: Mac Pro (2013 and later), MacBook Air (Early 2015 and later), MacBook Pro (Early 2015 and later), Mac mini (Late 2014 and later), iMac (Late 2015 and later), MacBook (Early 2016 and later), iMac Pro (2017 and later)

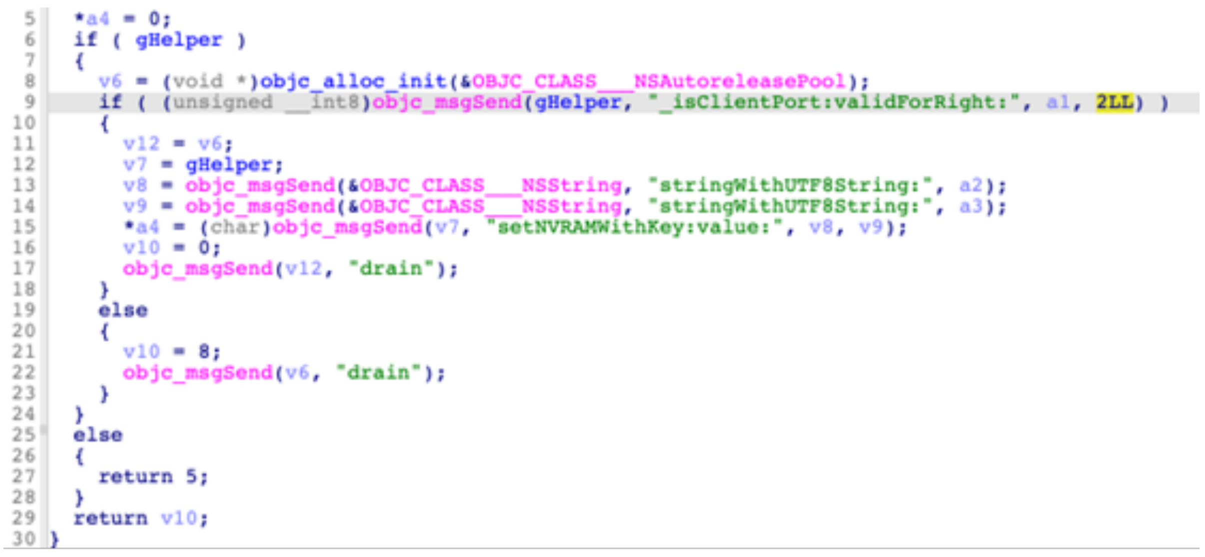
Impact: An unprivileged application may be able to edit NVRAM variables

Description: The issue was addressed with improved permissions logic.

CVE-2021-30913: Kirin (@Pwnrin) and chenyuwang (@mzzzz__) of Tencent Security Xuanwu Lab

Figure 7. CVE-2021-30913 details

The vulnerability exists in the caller function of the function "-[SUHelper setNVRAMWithKey:value:]". Its patch adds the validation code at line 9.

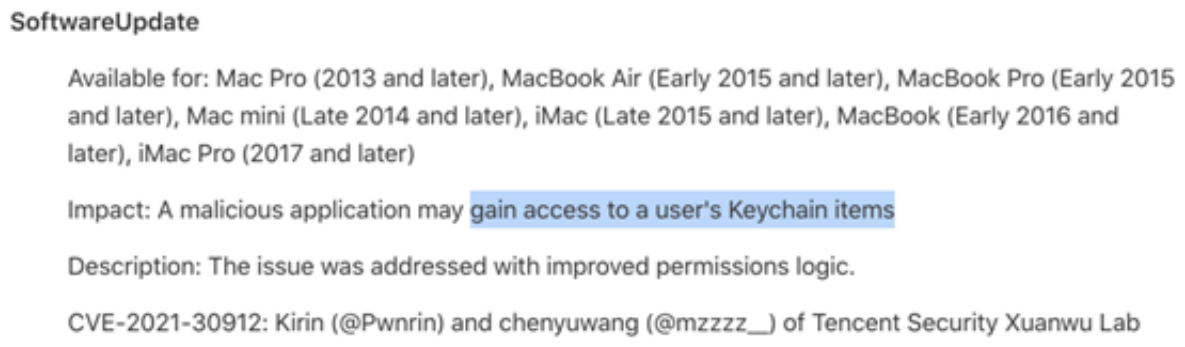


```
 5    *a4 = 0;
 6    if ( gHelper )
 7    {
 8      v6 = (void *)objc_alloc_init(&OBJC_CLASS___NSAutoreleasePool);
 9      if ( (unsigned __int8)objc_msgSend(gHelper, "_isClientPort:validForRight:", a1, 2LL) )
10      {
11        v12 = v6;
12        v7 = gHelper;
13        v8 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", a2);
14        v9 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", a3);
15        *a4 = (char)objc_msgSend(v7, "setNVRAMWithKey:value:", v8, v9);
16        v10 = 0;
17        objc_msgSend(v12, "drain");
18      }
19      else
20      {
21        v10 = 8;
22        objc_msgSend(v6, "drain");
23      }
24    }
25    else
26    {
27      return 5;
28    }
29    return v10;
30  }
```

Figure 8. The patch of CVE-2021-30913

It validates the client rights with value 2, so I renamed the caller function as IPC_2_setNVRAMWithKey_value to mark the needed rights.

Next is CVE-2021-30912, a vulnerability that could grant malicious actors access to a user's Keychain items.



**SoftwareUpdate**

Available for: Mac Pro (2013 and later), MacBook Air (Early 2015 and later), MacBook Pro (Early 2015 and later), Mac mini (Late 2014 and later), iMac (Late 2015 and later), MacBook (Early 2016 and later), iMac Pro (2017 and later)

Impact: A malicious application may gain access to a user's Keychain items

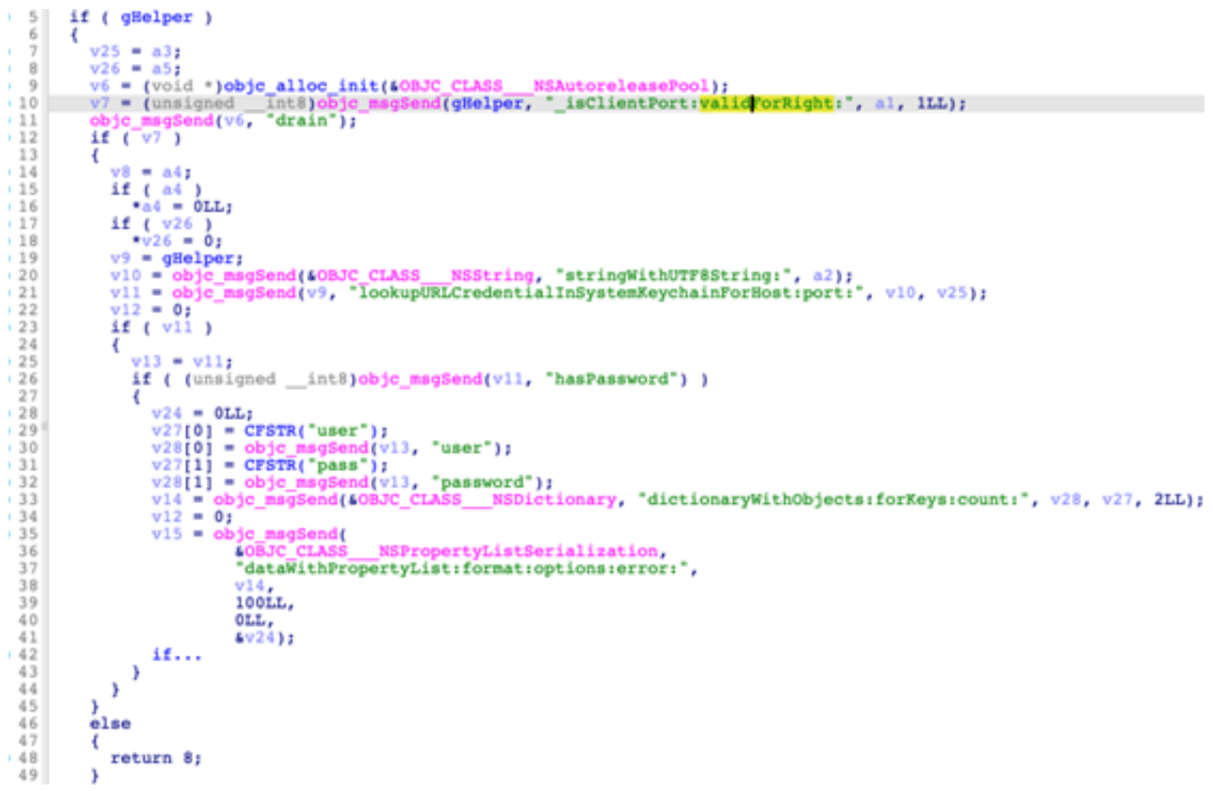
Description: The issue was addressed with improved permissions logic.

CVE-2021-30912: Kirin (@Pwnrin) and chenyuwang (@mzzzz__) of Tencent Security Xuanwu Lab

Figure 9. CVE-2021-30912 details

The vulnerability exists in the caller function of the function "-[SUHelper lookupURLCredentialInSystemKeychainForHost:port:]".

Its patch adds the validation code at line 10.

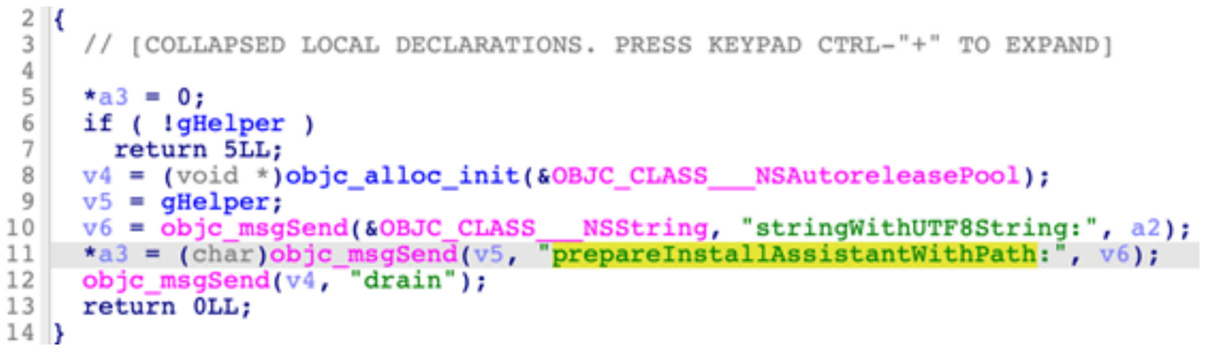


```
 5  if ( gHelper )
 6  {
 7    v25 = a3;
 8    v26 = a5;
 9    v6 = (void *)objc_alloc_init(&OBJC_CLASS___NSAutoreleasePool);
10    v7 = (unsigned __int8)objc_msgSend(gHelper, "_isClientPort:validForRight:", a1, 1LL);
11    objc_msgSend(v6, "drain");
12    if ( v7 )
13    {
14      v8 = a4;
15      if ( a4 )
16        *a4 = 0LL;
17      if ( v26 )
18        *v26 = 0;
19      v9 = gHelper;
20      v10 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", a2);
21      v11 = objc_msgSend(v9, "lookupURLCredentialInSystemKeychainForHost:port:", v10, v25);
22      v12 = 0;
23      if ( v11 )
24      {
25        v13 = v11;
26        if ( (unsigned __int8)objc_msgSend(v11, "hasPassword") )
27        {
28          v24 = 0LL;
29          v27[0] = CFSTR("user");
30          v28[0] = objc_msgSend(v13, "user");
31          v27[1] = CFSTR("pass");
32          v28[1] = objc_msgSend(v13, "password");
33          v14 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithObjects:forKeys:count:", v28, v27, 2LL);
34          v12 = 0;
35          v15 = objc_msgSend(
36                  &OBJC_CLASS___NSPropertyListSerialization,
37                  "dataWithPropertyList:format:options:error:",
38                  v14,
39                  100LL,
40                  0LL,
41                  &v24);
42          if...
43        }
44      }
45    }
46    else
47    {
48      return 8;
49    }
```

Figure 10. The patch of CVE-2021-30912

## New finding: CVE-2022-22639

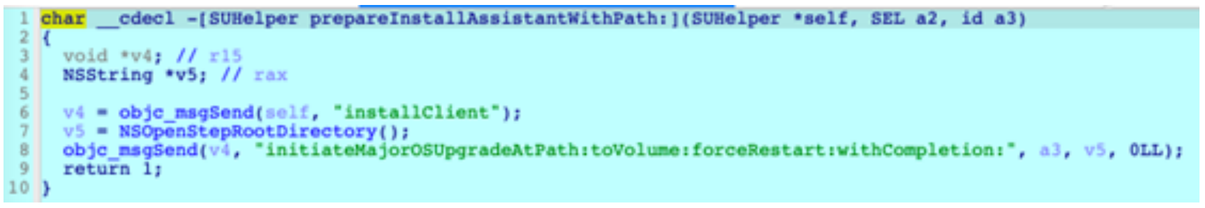
After reviewing the 45 service routines, I filtered out those with validation codes and found a few that had names starting with "IPC_0_". A close inspection of these routines revealed that the function "-[SUHelper prepareInstallAssistantWithPath:(NSString *) path]" was exploitable. The caller function IPC_0_prepareInstallAssistantWithPath did not validate the client's rights and called the real routine directly.



```
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  *a3 = 0;
  if ( !gHelper )
    return 5LL;
  v4 = (void *)objc_alloc_init(&OBJC_CLASS___NSAutoreleasePool);
  v5 = gHelper;
  v6 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", a2);
  *a3 = (char)objc_msgSend(v5, "prepareInstallAssistantWithPath:", v6);
  objc_msgSend(v4, "drain");
  return 0LL;
}
```

Figure 11. A vulnerable IPC service routine

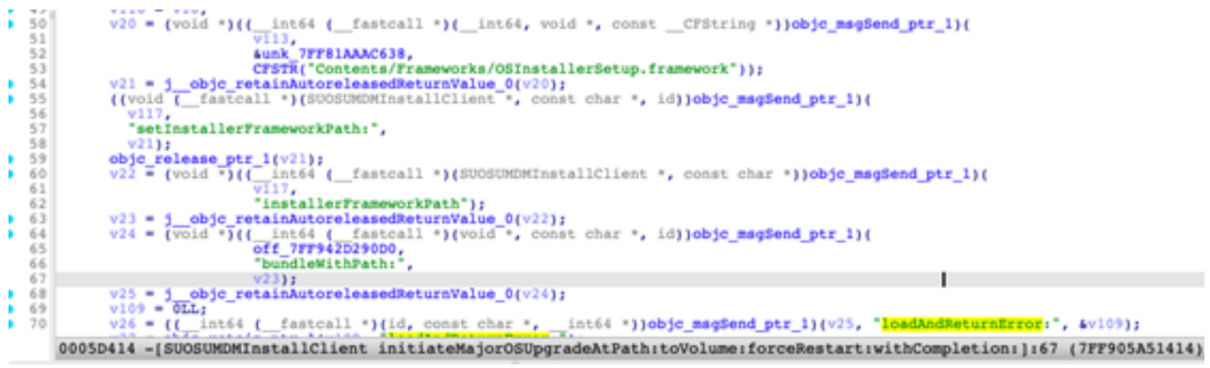
The implementation of the function is as follows, with the third parameter (NSString *) path that is passed from the client.



```
char __cdecl -[SUHelper prepareInstallAssistantWithPath:](SUHelper *self, SEL a2, id a3)
{
  void *v4; // r15
  NSString *v5; // rax

  v4 = objc_msgSend(self, "installClient");
  v5 = NSOpenStepRootDirectory();
  objc_msgSend(v4, "initiateMajorOSUpgradeAtPath:toVolume:forceRestart:withCompletion:", a3, v5, 0LL);
  return 1;
}
```

Figure 12. The implementation of the function "–[SUHelper prepareInstallAssitantWithPath:]"

A look at the internal function reveals that it loads a bundle at line 70.



```
50     v20 = (void *)((__int64 (__fastcall *)(__int64, void *, const __CFString *))objc_msgSend_ptr_1)(
51                v113,
52                &unk_7FF81AAAC638,
53                CFSTR("Contents/Frameworks/OSInstallerSetup.framework"));
54     v21 = j__objc_retainAutoreleasedReturnValue_0(v20);
55     ((void (__fastcall *)(SUOSUMDMInstallClient *, const char *, id))objc_msgSend_ptr_1)(
56       v117,
57       "setInstallerFrameworkPath:",
58       v21);
59     objc_release_ptr_1(v21);
60     v22 = (void *)((__int64 (__fastcall *)(SUOSUMDMInstallClient *, const char *))objc_msgSend_ptr_1)(
61                v117,
62                "installerFrameworkPath");
63     v23 = j__objc_retainAutoreleasedReturnValue_0(v22);
64     v24 = (void *)((__int64 (__fastcall *)(void *, const char *, id))objc_msgSend_ptr_1)(
65                off_7FF942D290D0,
66                "bundleWithPath:",
67                v23);
68     v25 = j__objc_retainAutoreleasedReturnValue_0(v24);
69     v109 = 0LL;
70     v26 = ((__int64 (__fastcall *)(id, const char *, __int64 *))objc_msgSend_ptr_1)(v25, "loadAndReturnError:", &v109);
0005D414 -[SUOSUMDMInstallClient initiateMajorOSUpgradeAtPath:toVolume:forceRestart:withCompletion:]:67 (7FF905A51414)
```

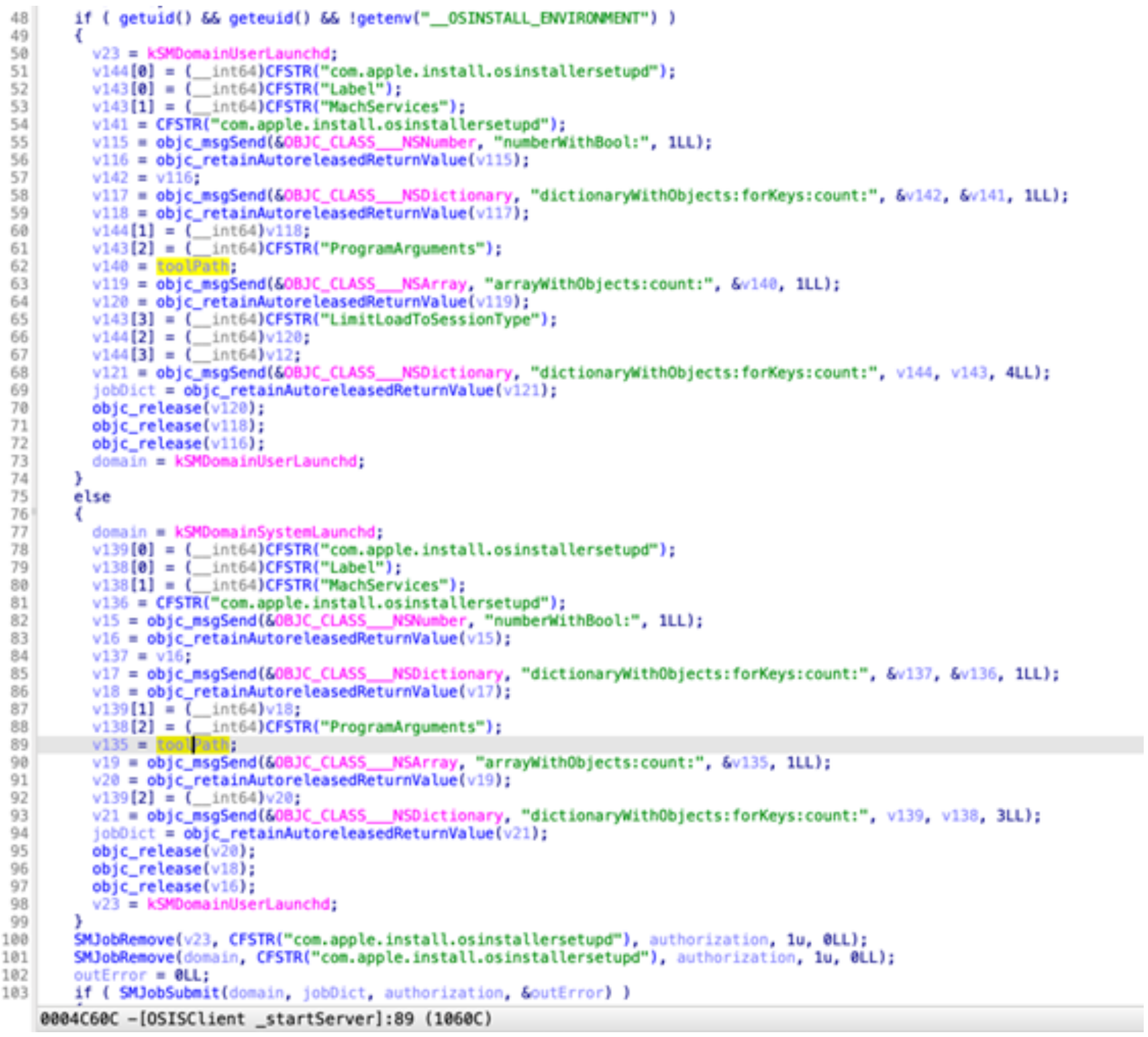
Figure 13. The internal function implementation

I debugged and found the bundle path as ${Assistant.app}/Contents/Frameworks/OSInstallerSetup.framework. An important finding is that the ${Assistant.app} is actually the third parameter (NSString *) path, which can be completely controlled by the client.

In a normal scenario, the ${Assistant.app} should be the real path to "Install macOS XXX.app". It is extracted from InstallAssistant.pkg, which is downloaded from the Apple server. However, I discovered that a user could fake the path and contents of the ${Assistant.app} by exploiting this vulnerability.

It seems that I found a primitive to load any dylib into the target process to get the root privilege and the special entitlements. However, I failed to load a self-signed dylib directly because I found that hardened runtime is enabled by default for system processes when SIP is on, even though it is not signed with runtime flags. But I could load arbitrary Apple-signed dylib into it even if it was an old, vulnerable dylib.

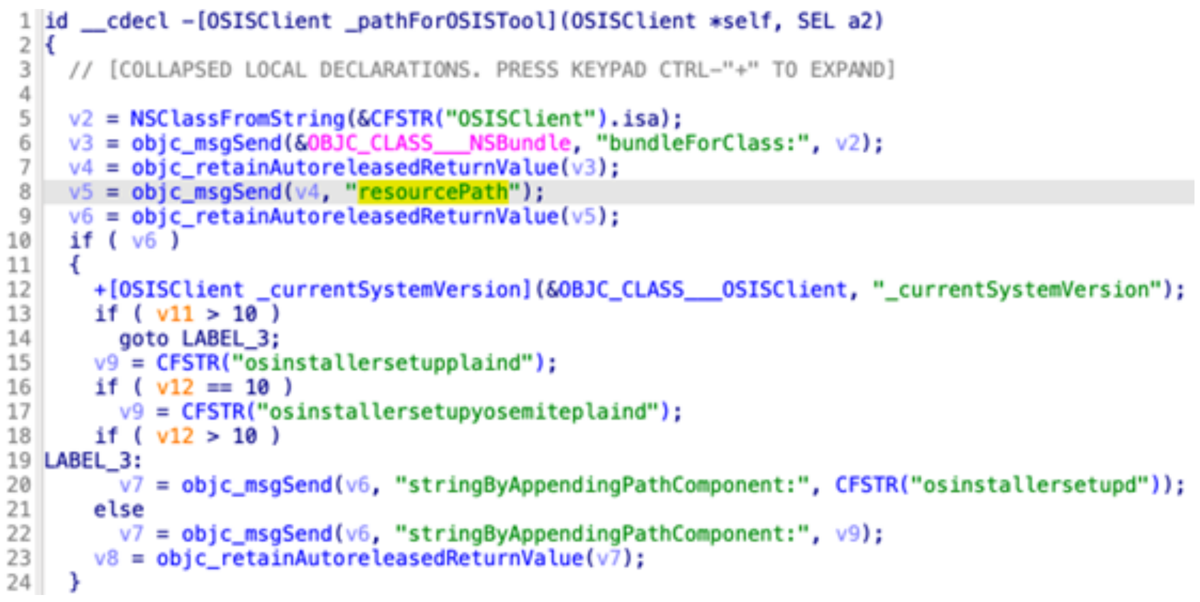
Perhaps there are other methods to exploit the issue. Here, I let it load the original OSInstallerSetup.framework. Once the OSInstallerSetup.framework is loaded, it calls the function "-[OSISClient _startServer]". At line 103, it launches another IPC service, com.apple.install.osinstallersetupd, by API SMJobSubmit. From line 48, it can be seen that if the current process is running as root, the newly submitted job runs at system domain with root privileges too.



```
if ( getuid() && geteuid() && !getenv("__OSINSTALL_ENVIRONMENT") )
{
  v23 = kSMDomainUserLaunchd;
  v144[0] = (__int64)CFSTR("com.apple.install.osinstallersetupd");
  v143[0] = (__int64)CFSTR("Label");
  v143[1] = (__int64)CFSTR("MachServices");
  v141 = CFSTR("com.apple.install.osinstallersetupd");
  v115 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithBool:", 1LL);
  v116 = objc_retainAutoreleasedReturnValue(v115);
  v142 = v116;
  v117 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithObjects:forKeys:count:", &v142, &v141, 1LL);
  v118 = objc_retainAutoreleasedReturnValue(v117);
  v144[1] = (__int64)v118;
  v143[2] = (__int64)CFSTR("ProgramArguments");
  v140 = toolPath;
  v119 = objc_msgSend(&OBJC_CLASS___NSArray, "arrayWithObjects:count:", &v140, 1LL);
  v120 = objc_retainAutoreleasedReturnValue(v119);
  v143[3] = (__int64)CFSTR("LimitLoadToSessionType");
  v144[2] = (__int64)v120;
  v144[3] = (__int64)v12;
  v121 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithObjects:forKeys:count:", v144, v143, 4LL);
  jobDict = objc_retainAutoreleasedReturnValue(v121);
  objc_release(v120);
  objc_release(v118);
  objc_release(v116);
  domain = kSMDomainUserLaunchd;
}
else
{
  domain = kSMDomainSystemLaunchd;
  v139[0] = (__int64)CFSTR("com.apple.install.osinstallersetupd");
  v138[0] = (__int64)CFSTR("Label");
  v138[1] = (__int64)CFSTR("MachServices");
  v136 = CFSTR("com.apple.install.osinstallersetupd");
  v15 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithBool:", 1LL);
  v16 = objc_retainAutoreleasedReturnValue(v15);
  v137 = v16;
  v17 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithObjects:forKeys:count:", &v137, &v136, 1LL);
  v18 = objc_retainAutoreleasedReturnValue(v17);
  v139[1] = (__int64)v18;
  v138[2] = (__int64)CFSTR("ProgramArguments");
  v135 = toolPath;
  v19 = objc_msgSend(&OBJC_CLASS___NSArray, "arrayWithObjects:count:", &v135, 1LL);
  v20 = objc_retainAutoreleasedReturnValue(v19);
  v139[2] = (__int64)v20;
  v21 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithObjects:forKeys:count:", v139, v138, 3LL);
  jobDict = objc_retainAutoreleasedReturnValue(v21);
  objc_release(v20);
  objc_release(v18);
  objc_release(v16);
  v23 = kSMDomainUserLaunchd;
}
SMJobRemove(v23, CFSTR("com.apple.install.osinstallersetupd"), authorization, 1u, 0LL);
SMJobRemove(domain, CFSTR("com.apple.install.osinstallersetupd"), authorization, 1u, 0LL);
outError = 0LL;
if ( SMJobSubmit(domain, jobDict, authorization, &outError) )
0004C60C -[OSISClient _startServer]:89 (1060C)
```

Figure 14. The implementation of the function "–[OSISClient _startServer]"

Now, the current process is suhelperd, running as root, and the job executable path is toolPath, which is inside the bundle ${Assistant.app}/Contents/Frameworks/OSInstallerSetup.framework/Resources/osinstallersetupd. A malicious actor could put the payload in toolPath directly to attain root privilege escalation.



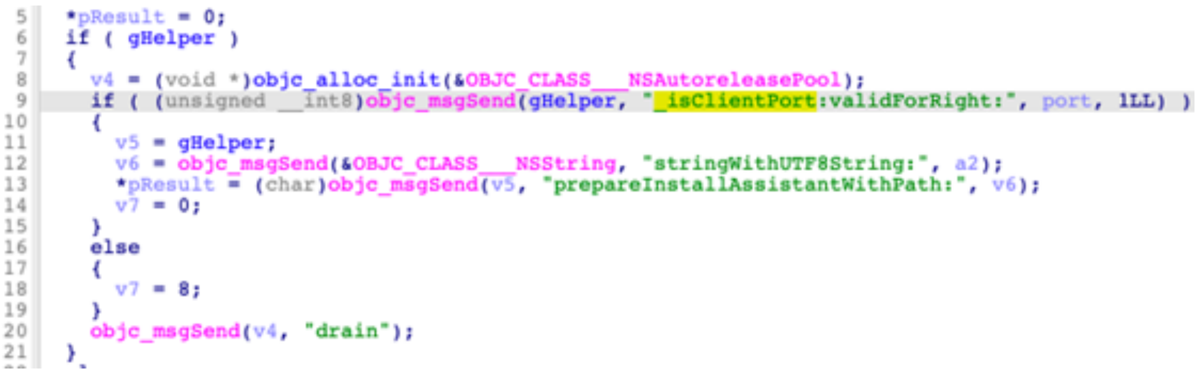
```
id __cdecl -[OSISClient _pathForOSISTool](OSISClient *self, SEL a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = NSClassFromString(&CFSTR("OSISClient").isa);
  v3 = objc_msgSend(&OBJC_CLASS___NSBundle, "bundleForClass:", v2);
  v4 = objc_retainAutoreleasedReturnValue(v3);
  v5 = objc_msgSend(v4, "resourcePath");
  v6 = objc_retainAutoreleasedReturnValue(v5);
  if ( v6 )
  {
    +[OSISClient _currentSystemVersion](&OBJC_CLASS___OSISClient, "_currentSystemVersion");
    if ( v11 > 10 )
      goto LABEL_3;
    v9 = CFSTR("osinstallersetupplaind");
    if ( v12 == 10 )
      v9 = CFSTR("osinstallersetupyosemiteplaind");
    if ( v12 > 10 )
LABEL_3:
      v7 = objc_msgSend(v6, "stringByAppendingPathComponent:", CFSTR("osinstallersetupd"));
    else
      v7 = objc_msgSend(v6, "stringByAppendingPathComponent:", v9);
    v8 = objc_retainAutoreleasedReturnValue(v7);
  }
```

Figure 15. How to get the toolPath

The full proof of concept can be found here and a video demonstration can be viewed here.

## Patch

As mentioned earlier, Apple has addressed the CVE-2022-22639 issue through the macOS Monterey 12.3 security update. This patch now adds the validation code at line 9.



```
 5  *pResult = 0;
 6  if ( gHelper )
 7  {
 8    v4 = (void *)objc_alloc_init(&OBJC_CLASS___NSAutoreleasePool);
 9    if ( (unsigned __int8)objc_msgSend(gHelper, "_isClientPort:validForRight:", port, 1LL) )
10    {
11      v5 = gHelper;
12      v6 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", a2);
13      *pResult = (char)objc_msgSend(v5, "prepareInstallAssistantWithPath:", v6);
14      v7 = 0;
15    }
16    else
17    {
18      v7 = 8;
19    }
20    objc_msgSend(v4, "drain");
21  }
```

Figure 16. The patch of CVE-2022-22639

## Security recommendations

End-users can mitigate the risks by regularly updating systems and applications with the latest patches to ensure that security flaws cannot be exploited for malicious activities.

Learn about Trend Micro™ Maximum Security for Mac so you can enjoy your digital life safely. It blocks viruses, spyware, ransomware, and other malicious software for your peace of mind.

## Authors

Mickey Jin

Threats Analyst

Contact Us
Subscribe