KCon

KCon 2019 無界

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

macOS 从运行库劫持到内核提权

周智

About

- Senior Security Engineer of AntFinancial (Alipay) LightYear Security Labs
- Product security and offensive security research
- Acknowledged by Microsoft, Apple, Adobe and VMware for reporting security vulnerabilities
- Conference speaking:
  - BlackHat USA 2017
  - HITB 2019
  - TyphoonCon 2019

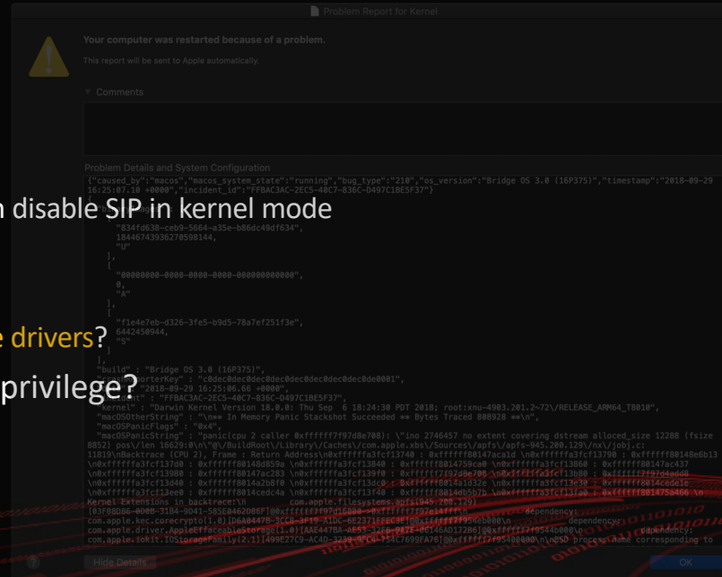
CONTENTS

Why kernel exploit

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

Motivation

- 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?



KCon

PART 01
An Attack Surface

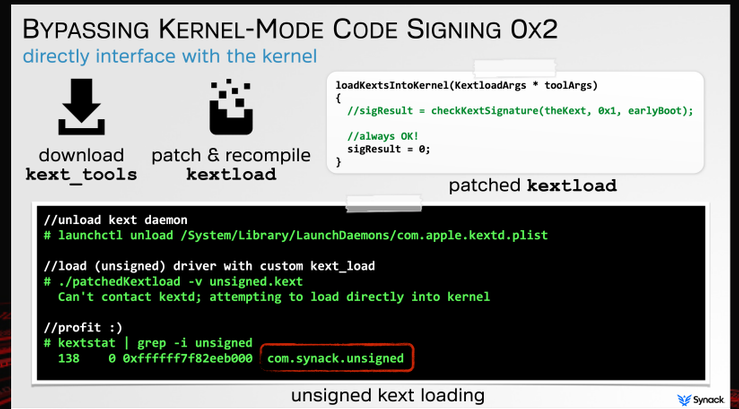
Old days with kext_tools

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.

patch kextd
(@osxreverser, Nov 2013)



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

- 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)

- 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

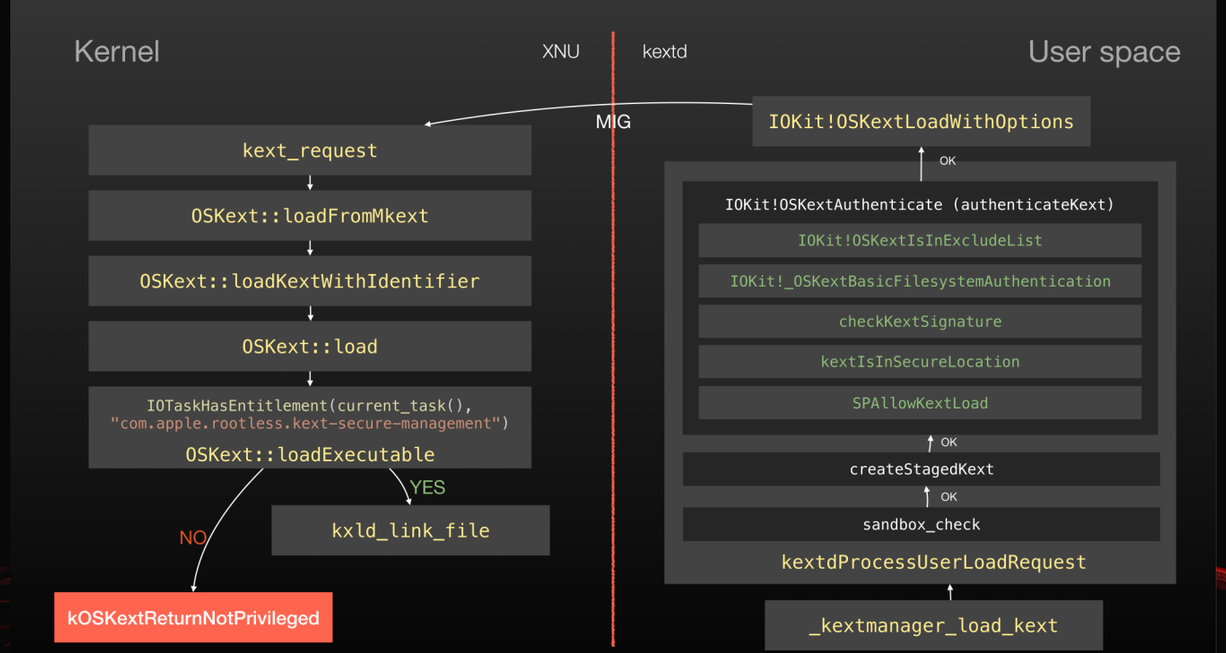
kextd_double_port_deallocate.zip
9.7 KB Download

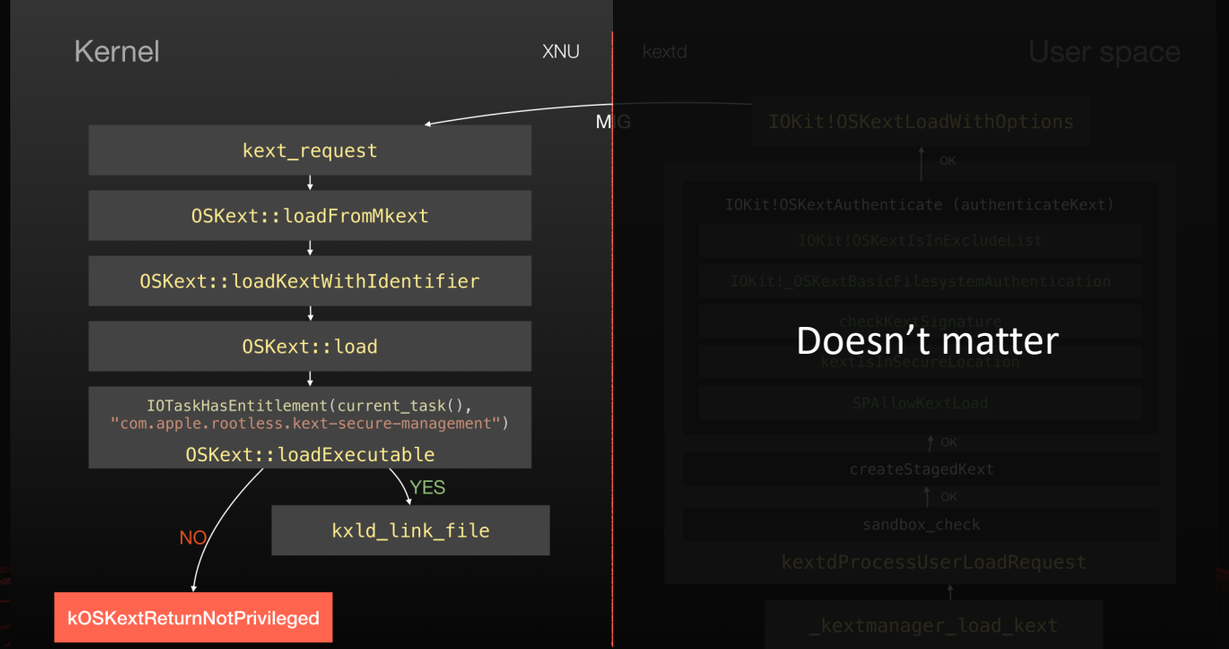
Comment 1 by ianbeer@google.com on Thu, Jan 25, 2018, 12:21 AM GMT+8

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

Arbitrary code execution in kextd == kernel code execution

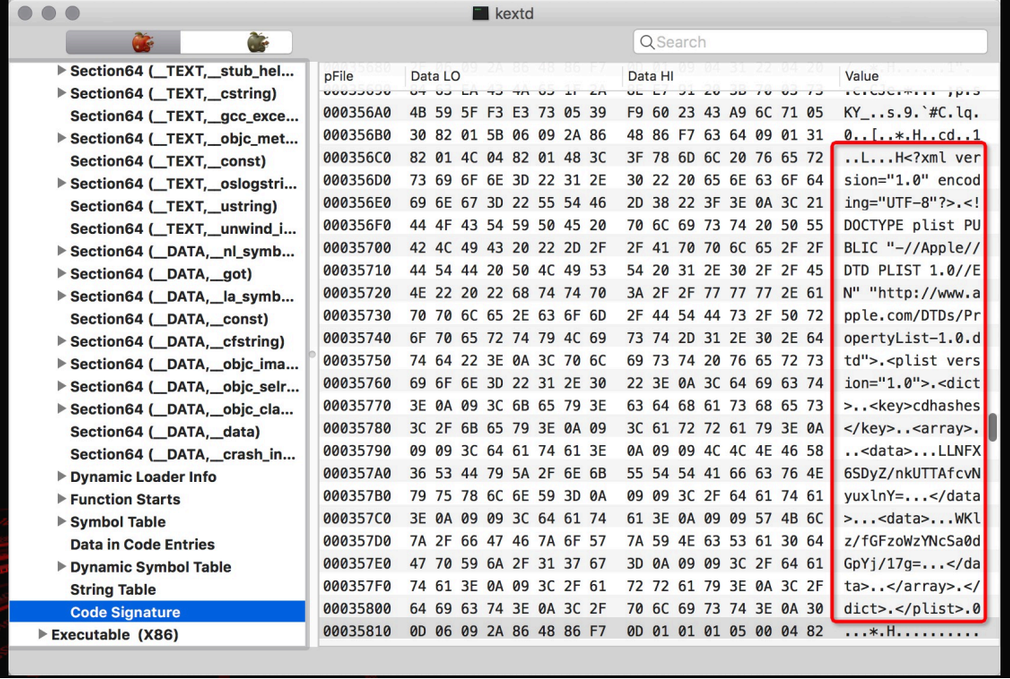

kextload
kextd
XNU


Kernel
XNU
kextd
User space
MIG
kext_request
OSKext::loadFromMkext
OSKext::loadKextWithIdentifier
OSKext::load
IOTaskHasEntitlement(current_task(), "com.apple.rootless.kext-secure-management")
OSKext::loadExecutable
YES
NO
kxld_link_file
kOSKextReturnNotPrivileged
IOKit!OSKextLoadWithOptions
OK
IOKit!OSKextAuthenticate (authenticateKext)
IOKit!OSKextIsInExcludeList
IOKit!_OSKextBasicFilesystemAuthentication
checkKextSignature
kextIsInSecureLocation
SPAllowKextLoad
OK
createStagedKext
OK
sandbox_check
kextdProcessUserLoadRequest
_kextmanager_load_kext


Kernel
XNU
kextd
User space
MIG
kext_request
OSKext::loadFromMkext
OSKext::loadKextWithIdentifier
OSKext::load
IOTaskHasEntitlement(current_task(), "com.apple.rootless.kext-secure-management")
OSKext::loadExecutable
YES
NO
kxld_link_file
kOSKextReturnNotPrivileged
IOKit!OSKextLoadWithOptions
OK
IOKit!OSKextAuthenticate (authenticateKext)
IOKit!OSKextIsInExcludeList
IOKit!_OSKextBasicFilesystemAuthentication
checkKextSignature
SPAllowKextLoad
OK
createStagedKext
OK
sandbox_check
kextdProcessUserLoadRequest
_kextmanager_load_kext
Doesn’t matter

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"



```
➜  ~ 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>
```

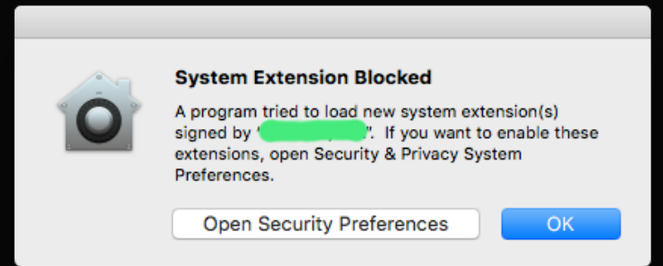
- Entitled to call kext_request
- Permission to write /Library/StagedExtensions

Checks by 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 will be 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
➜  ~ sudo sqlite3 /var/db/SystemPolicyConfiguration/KextPolicy

SQLite version 3.24.0 2018-06-04 14:10:15
Enter ".help" for usage hints.
sqlite> .tables
kext_load_history_v3  kext_policy_mdm
kext_policy           settings
sqlite> .header on
sqlite> select * from kext_policy;
team_id|bundle_id|allowed|developer_name|flags
9PTGMPNXZ2|com.symantec.kext.SymAPComm|1|Symantec|8
9PTGMPNXZ2|com.symantec.kext.ndcengine|1|Symantec|8
9PTGMPNXZ2|com.symantec.kext.internetSecurity|1|Symantec|8
9PTGMPNXZ2|com.symantec.kext.ips|1|Symantec|8
Z3L495V9L4|com.intel.kext.intelhaxm|1|Intel Corporation Apps|1
VB5E2TV963|org.virtualbox.kext.VBoxDrv|1|Oracle America, Inc.|1
```

kextd

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

XPC

syspolicyd

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

Prompt / Reject / Pass based on SQLite database rules

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

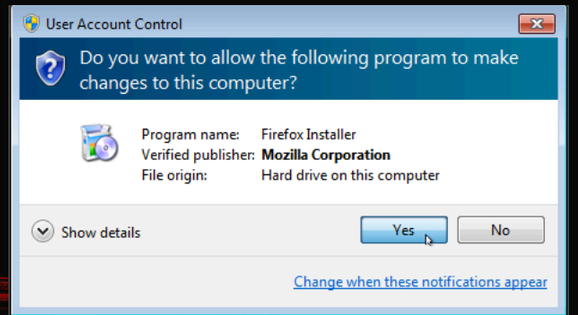
- Neither the signature nor file permission is checked by kernel
- It accepts kext_request as long as the user space process has com.apple.rootless.kext-secure-management entitlement
- User space process kextd / kextutil / kextload are responsible to perform the signature and other validation
- Once 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)

PART 02

Hijack the Entitlement

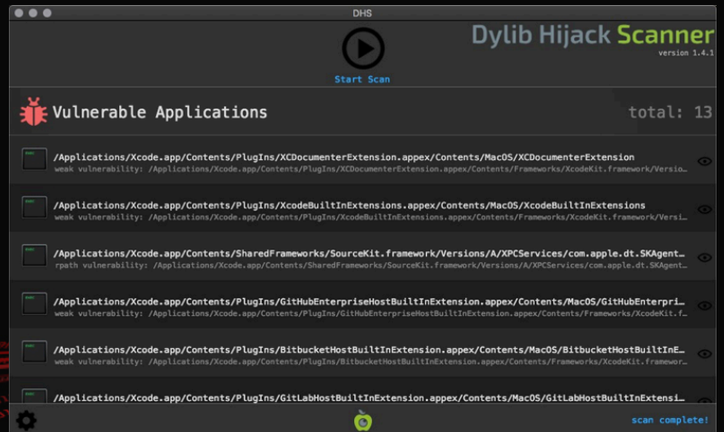
DLL Hijack on Windows

- Trick the target application to load malicious library
  - Abuse DLL search order
  - Abuse runtime (LoadLibrary)
  - …
- DLL hijacking on trusted application to bypass UAC prompt
- Is there anything similar on macOS?



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)



```
VM Regions Near 0xdeadbf57:
-->
    __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: 0x00000000deadbeef  rcx: 0x00000001171ee66c  rdx: 0x0000000000000001
```

Symbolication

The bug

- The CoreSymbolication framework provides private APIs for symbolicating and other diagnostic information (/System/Library/PrivateFrameworks/CoreSymbolication.framework)
- Under certain circumstances it will try to load a dynamic library from a controllable path
  - `When trying to demangle Swift symbols:`
    - `CoreSymbolication!call_external_demangle(char const*)`
  - More specifically, it will try to load a Swift runtime library, libswiftDemangle.dylib

```
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");
```

insecure dlopen
(dylib hijack)

xcselect.dylib!xcselect_get_developer_dir_path

```
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
```

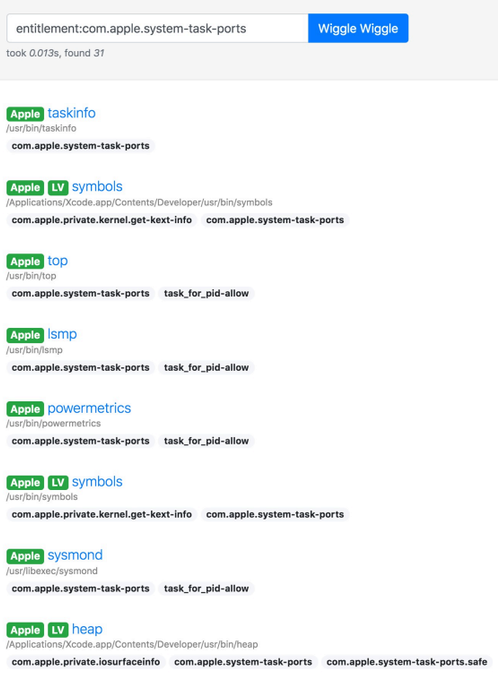
Trigging the bug

- 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
- *以子之盾 攻子之盾*

```
(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")
)
```

Find an entitled 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 other processes (even those restricted), and gain arbitrary entitlement



com.apple.SamplingTools

```
➜  ~ ls
/usr/bin/{filtercalltree,heap32,stringdups32,leaks32,heap,atos,vmmap32,sample,malloc_history32,symbols,vmmap
,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/vmmap
/usr/bin/heap32            /usr/bin/sample           /usr/bin/vmmap32
/usr/bin/leaks             /usr/bin/stringdups


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

com.apple.SamplingTools

- *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
- SIP exception, entitled to debug any process, including restricted

```
➜  ~ jtool --ent `which vmmap`
<?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

- Target app is written in Swift
- Use symbols to inspect the target app
- Use -printDemangling flag to trigger dylib hijack
- `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
```

Problem: Library Validation

- Library Validation is a protection that prohibits a process to load dynamic libraries without a digital signature issued by same team id
- 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, 0x000000010d745000
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           000000010d70a000-000000010d745000 [  236K] rw-/rwx SM=PRV
--> mapped file            000000010d745000-000000010d746000 [    4K] r-x/r-x SM=PRV  Object_id=2929ab85
    mapped file            000000010d748000-000000010d762000 [  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
```

El Capitan

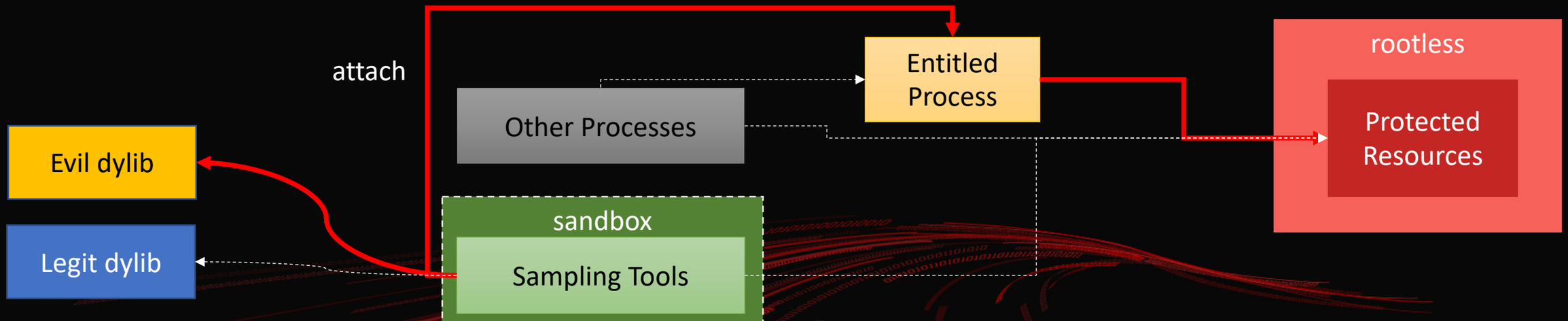
```
→  bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=812 flags=0x0(none) hashes=32+5
location=embedded
Platform identifier=1
Hash type=sha1 size=20
```

An old binary grabbed from previous OS X does not have this flag!

Exploit

- Craft the Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib
- Invoke sandbox_init_with_parameters to drop access to the legit swift libraries
- Set the DEVELOPER_DIR environment variable to redirect access to our payload
- 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


attach
Entitled Process
rootless
Protected Resources
Other Processes
Evil dylib
sandbox
Legit dylib
Sampling Tools

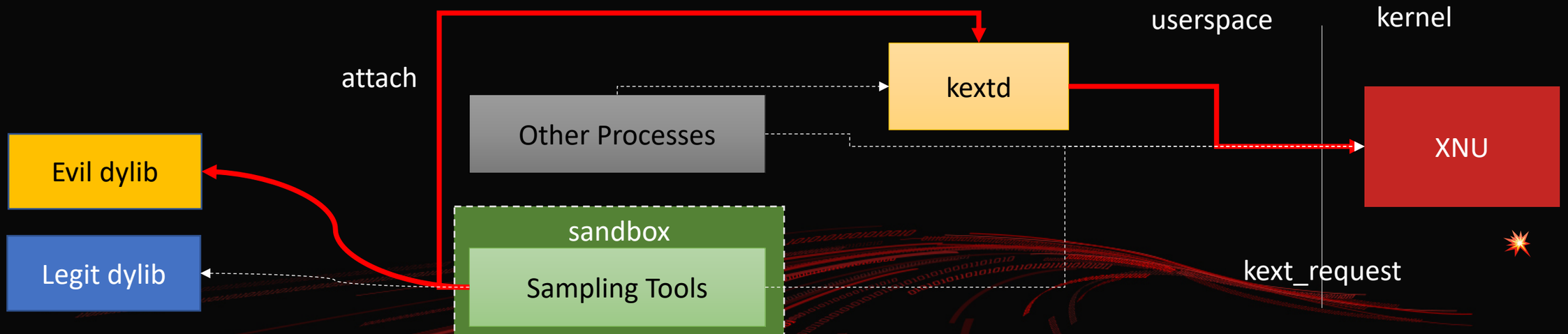
PART 03

To the Kernel

Rule the 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

Kernel Code Execution without actually touching XNU



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 45  ........8...__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                                               .
```

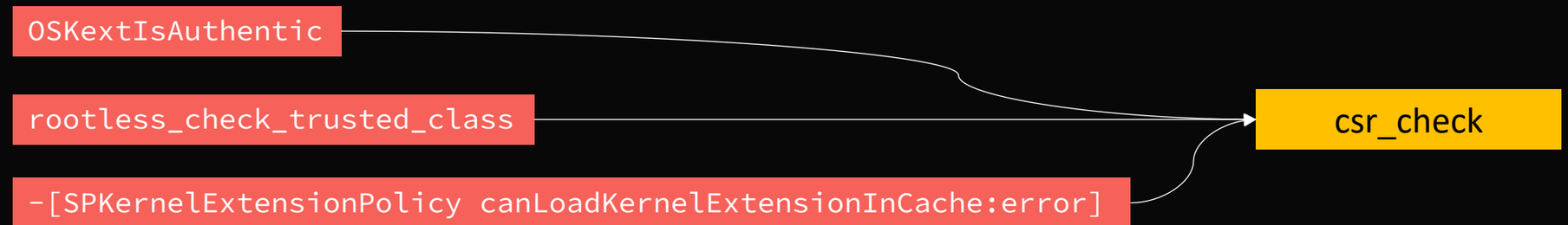
mkext2_header | mkext2_file_entry | mkext2_file_entry | ... | plist

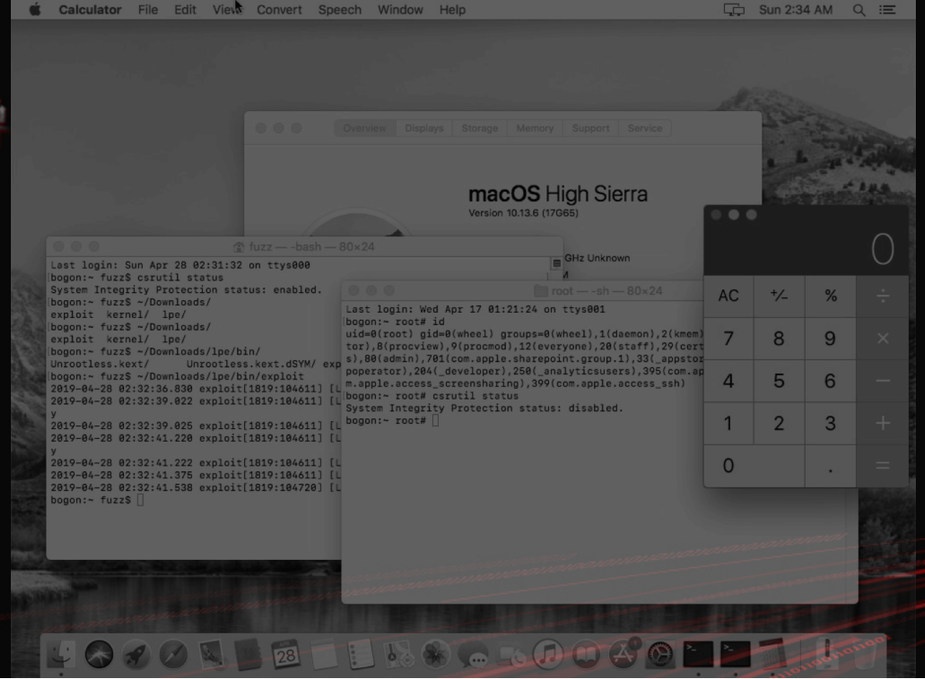
```
#define MKEXT_MAGIC 0x4D4B5854 /* 'MKXT' */
#define MKEXT_SIGN  0x4D4F5358 /* 'MOSX' */

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-O
    cpu_subtype_t cpusubtype;  // same as Mach-O

    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;
```

The Kill-Switch

- KEXT Validations
  - Code Signature
  - KEXT Staging
  - SKEL





Load completely unsigned kext on macOS 10.13.6 (17G65) (chained with CVE-2019-8565 Apple Feedback Assistant local root privilege escalation)

PoC

- You can grab the source code here
  https://github.com/ChiChou/sploits/tree/master/ModJack

PART 04

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
```

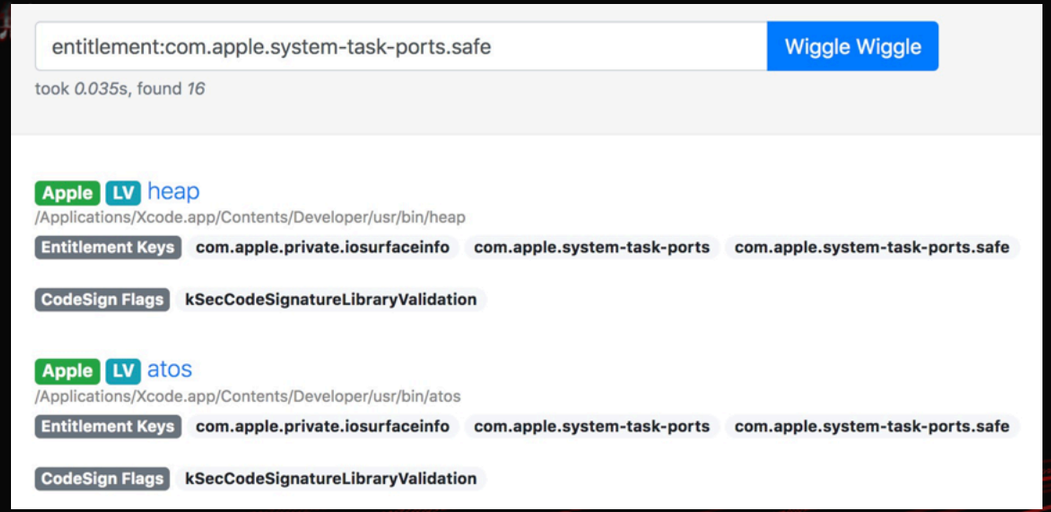
```
BOOL __cdecl -[VMUObjectIdentifier _dlopenLibSwiftRemoteMirrorFromDir:](VMUObjectIdentifier *self, SEL a2, NSString* directory) {
  if (!directory)
    return NO;


  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);
  ...
```

Another dylib Hijack

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

Mitigation

- The variant doesn’t work anymore on macOS Mojave
- Hardened Runtime has been applied
  - 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.vmmap), and have a new entitlement com.apple.system-task-ports.safe




```
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");
```

KCon

Thanks

@CodeColorist