

iOS内核研究：定位内核gadget符号地址

wooy0ung / 2019-08-01 09:30:00 / 浏览数 3681 [安全技术](#) [二进制安全](#) [顶\(0\)](#) [踩\(0\)](#)

原文地址：https://medium.com/@cji_hunting-for-ios-kernel-symbols-e48a446bb00

0x001 前言

上周，Google Project Zero的Ian Beer在[Twitter](#)上发文，他将通过task_for_pid_0或tfp0提供对内核内存的读、写访问，以帮助研究人员更深入研究iOS 11内核安全性。

这消息一出，可能人们更多的关注会放在新系统的越狱上面，但我想借此机会再研究一下移动安全。

Ian Beer昨日已将poc代码放在Project Zero的主页上[Issue 1417: iOS/macOS kernel double free due to IOSurfaceRootUserClient not respecting MIG ownership rules](#)，包含有CVE-2016-7612、CVE-2016-7633两个漏洞。正当我准备好开始我的工作时，便遇到一个问题。Ian Beer放出来的poc代码中为几个设备添加了内核符号，但这几个设备我手上并没有。运气比较好，他提供了有关如何找到符号的详细说明。

本文旨在介绍查找iOS符号的过程，因为我没有找到关于如何执行此操作的任何其他文档，因此我想为其他人（以及将来的我自己）记录此过程，希望对你添加对设备的支持。

0x002 查找内核符号地址

查找符号的第一步是获取目标iOS版本的kernelcache。我是通过访问[ipsw.me](#)并下载了我的iPad Mini 2的11.1.2固件，同样也可以下载任何已经在symbols.c中定义了符号的设备。

在解压缩ipsw文件之后，我下载一份[joker](#)工具。这是一个命令行工具，在阅读该工具的帮助页以后，设定好-j和-m选项并处理kernelcache文件，转储所有可用的符号。输出

```
KSYMBOL_OSARRAY_GET_META_CLASS
KSYMBOL_IOUSERCLIENT_GET_META_CLASS
KSYMBOL_IOUSERCLIENT_GET_TARGET_AND_TRAP_FOR_INDEX
KSYMBOL_CSBLOB_GET_CD_HASH
KSYMBOL_KALLOC_EXTERNAL
KSYMBOL_KFREE
KSYMBOL_OSSERIALIZER_SERIALIZE
KSYMBOL_KPRINTF
KSYMBOL_UUID_COPY
```

在用IDA Pro加载kernelcache文件之前，我们还需要解密kernelcache文件。详细的方法可以参考这里：[getios10beta1kernelcache.sh](#)

```
* open kernelcache in a hex editor and look for 0xFFCFFAEDFE, note the offset (435)
* wget -q http://nah6.com/%7Eitsme/cvs-xddevtools/iphone/tools/lzssdec.cpp
* g++ -o lzssdec lzssdec.cpp
* ./lzssdec -o 435 < kernelcache >kernelcache.dec # 435 is offset byte count to 0xFFCFFAEDFE header
```

首先，IDA加载解密好的kernelcache文件，kernelcache带有一些已知内核符号，我们可以通过反汇编的结果来大致了解一下iOS内核。

最先找到的是KSYMBOL_RET，跳转到已知符号的地址，这是从_kalloc_external函数返回的RET指令。利用joker转储的符号地址信息很容易在IDA中跳转到对应的地址并

接下来是KSYMBOL_CPU_DATA_ENTRIES，其中提示称数据段为0x6000。在IDA中，选择Jump to Segment并转到_data的开头，将该地址加上0x6000最终得到了我们所需的地址。

接下来的两个地址是KSYMBOL_EL1_HW_BP_INFINITE_LOOP和KSYMBOL_SLEH_SYNC_EPILOG，这是ksymbol列表最后两个地址，具体可以看到Ian Beer利用代码里的symbols.h。在IDA中打开了String窗口(Shift + F12)并搜索字符串，双击查看引用。

```
text:FFFFFFFF0071A90B4 loc_FFFFFFFF0071A90B4 ; CODE XREF: __TEXT_EXEC:__text:FFFFFFFF0071A8FAC;j
text:FFFFFFFF0071A90B4 ; DATA XREF: __TEXT_EXEC:__text:FFFFFFFF0071AA4C8;j
text:FFFFFFFF0071A90B4 AND W8, W20, #0x3F ; jumtable FFFFFFFF0071A8FAC case 49
text:FFFFFFFF0071A90B4 CMP W8, #0x22
text:FFFFFFFF0071A90BC B.NE loc_FFFFFFFF0071A9A84
text:FFFFFFFF0071A90C0 loc_FFFFFFFF0071A90C0 ; CODE XREF: __TEXT_EXEC:__text:loc_FFFFFFFF0071A90C0;j
text:FFFFFFFF0071A90C0 B loc_FFFFFFFF0071A90C0
text:FFFFFFFF0071A90C4 ; -----
```

对于前者，向下阅读代码找到了这段switch case 49语句，并且得到该地址。

```
53
54 enum ksymbol {
55     KSMBOL_OSARRAY_GET_META_CLASS,
56     KSMBOL_IOUSERCLIENT_GET_META_CLASS,
57     KSMBOL_IOUSERCLIENT_GET_TARGET_AND_TRAP_FOR_INDEX,
58     KSMBOL_CSLOB_GET_CD_HASH,
59     KSMBOL_KALLOC_EXTERNAL,
60     KSMBOL_KFREE,
61     KSMBOL_RET,
62     KSMBOL_OSSERIALIZER_SERIALIZE,
63     KSMBOL_KPRINTF,
64     KSMBOL_UUID_COPY,
65     KSMBOL_CPU_DATA_ENTRIES,
66     KSMBOL_VALID_LINK_REGISTER,
67     KSMBOL_X21_JOP_GADGET,
68     KSMBOL_EXCEPTION_RETURN,
69     KSMBOL_THREAD_EXCEPTION_RETURN,
70     KSMBOL_SET_MDSCR_EL1_GADGET,
71     KSMBOL_WRITE_SYSCALL_ENTRYPOINT,
72     KSMBOL_EL1_HW_BP_INFINITE_LOOP,
73     KSMBOL_SLEH_SYNC_EPILOG
74 };
```



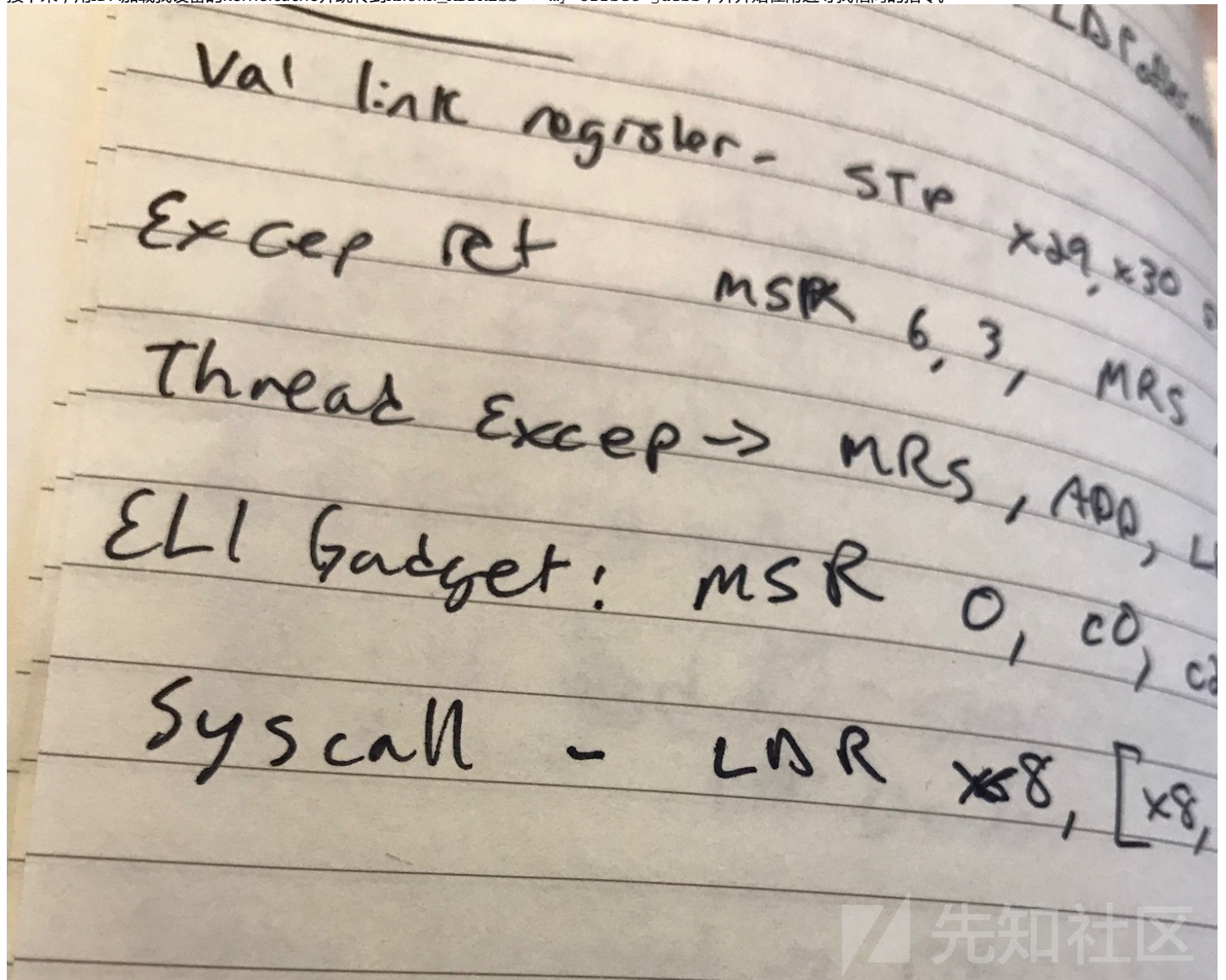
对于后者，是字符串引用地址下面几个LDP指令中的第一个。

最后一个就是KSMBOL_X21_JOP_GADGET，在看到所需的指令是MOV X21, #0后，我在IDA中进行了搜索，以找到我的iPad设备的Gadget。

还剩下的5条内核符号地址是最棘手的，在IDA中搜索并没有找到什么有用的信息，所以我开始查看我找到的地址以及它们与已知地址的比较。再然后，通过对比symbols.h

```
0xFFFFFFFF00709818C, // KSMBOL_VALID_LINK_REGISTER
0xFFFFFFFF007098164, // KSMBOL_X21_JOP_GADGET
0xFFFFFFFF007098434, // KSMBOL_EXCEPTION_RETURN
```

接下来，用IDA加载我设备的kernelcache并跳转到KNOWN_ADDRESS + my offset guess，并开始在附近寻找相同的指令。



当找到所有所需内核符号地址，需在poc代码中再添加一个if路径来支持我的设备的。

```
} else if (strstr(u.machine, "iPad4,4")) {  
    printf("this is iPad Mini 2 WiFi, should work!\n");  
    symbols = ksymbols_ipad_mini_2_wifi_15b202;  
    have_syms = 1;  
} else {
```

先知社区

添加一个printf()打印出信息，编译运行poc


```
void sys_write_breakpoint_handler(arm_context_t* state) {

    printf("=== in break-point ===\n");
    // we will have to skip it one instruction ahead because single step won't work...
    state->ss.ss_64.pc += 4;

    // this means emulating what that instruction did:
    // LDR      X8, [X8,#0x388]
    uint64_t val = rk64(state->ss.ss_64.x[8] + 0x388);
    state->ss.ss_64.x[8] = val;

    uint64_t uap = state->ss.ss_64.x[1];
    char* replacer_string = strdup("a different string!\n");
    wk64(uap+8, (uint64_t)replacer_string);
    wk64(uap+0x10, strlen(replacer_string));
}

char* hello_wrlld_str = "hellowrld!\n";
void test_kdbg() {
    run_syscall_with_breakpoint(ksym(KSYMBOL_WRITE_SYSCALL_ENTRYPOINT), // breakpoint address
                                sys_write_breakpoint_handler,           // breakpoint hit handler
                                4,                                       // SYS_write
                                3,                                       // 3 arguments
                                1,                                       // stdout
                                (uint64_t)hello_wrlld_str,              // "hellowrld!\n"
                                strlen(hello_wrlld_str));                // 11
}
```



```
0000000000000000
ffffffff115a71c48
ffffffff115a71d10
0000000000000004
ffffffff00082bc80
ffffffff008f99d38
ffffffff00082bb60
ffffffff0091f6094
ffffffff120000104
000000016f1270f8
0000002cc6000022
000000016f09ca00
0000008400000017
000000008e99fd4
=== in break-point ===
a different string!
return val 1007
syscall returned
monitor exited
```

最后漏洞在我的设备上成功触发。这是复现iOS设备内核漏洞的一般步骤，因为网上贴出来的poc、exp用到的内核符号地址一般与我们手头上的设备不一致，这样往往造成

点击收藏 | 0 关注 | 1

[上一篇：一次真实axis2渗透测试](#) [下一篇：深入理解Apk加固之Dex保护](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟贴

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)