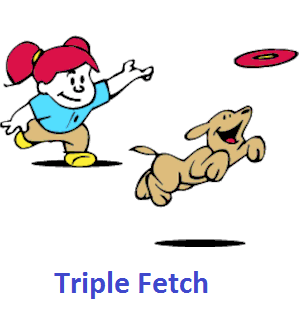
再谈CVE-2017-7047 Triple\_Fetch和另一种用NSXPC过沙盒的姿势

作者：蒸米

0x00 序

Ian Beer@google发布了CVE-2017-7047 Triple\_Fetch的exp和writeup[1]，chenliang@keenlab也发表了关于Triple\_Fetch的分析[2]，但由于这个漏洞和exp有非常多的亮点，所以还剩很多可以深入挖掘的细节。另外这个漏洞以及利用思路和我们去年OverSky私有越狱过沙盒的方法有一定相似性，也是利用NSXPC服务的漏洞，我们把这个漏洞称之为MLSqlite，并在这篇文章中一起分享出来。

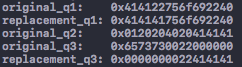
0x01 CVE-2017-7047 Triple\_Fetch漏洞形成的原因



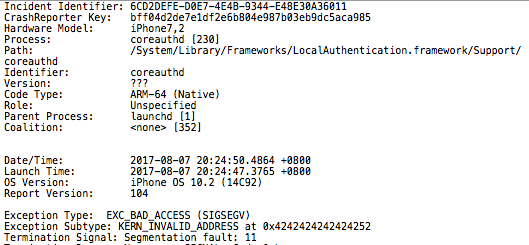
因为chenliang对漏洞成因的分析非常详细，这里我就简单描述一下，因为使用XPC服务传输大块内存的话很影响效率，苹果为了减少传输时间，对大于0x4000的OS\_xpc\_data数据会通过mach\_vm\_map的方式映射这块内存，然后将这块数据的send right以port的方式发送到另一方。但这段内存的共享是基于共享物理页的方式，也就是说发送方和接收方会共享同一块内存，因此我们将数据发送以后再在发送端对数据进行修改，接收方的数据也会发生变化。

因此通过race condition，可以让接收端得到不同的数据（接收端认为是相同的数据），如果接收端没有考虑到这一点的话就可能会出现漏洞。比如我们刚开始让接收端获取的字符串是@”ABCD”（包括@和”），那么接收端会为这个字符串分配7个字节的空间。随后在进行字符串拷贝的时候，我们将字符串变为@"ABCDOVERFLOW\_OVERFLOW\_OVERFLOW"，接收端会一直拷贝到遇到”符号为止，这样就造成了溢出。

Triple\_Fetch攻击所选择的函数是CoreFoundation里的\_\_\_NSMS1()函数，这个函数会对我们构造的恶意字符串进行多次读取操作，如果在读取的间隙快速对字符串进行三次修改，就会让函数读取到不同的字符串，让函数产生判断失误，从而造成溢出并让我们控制pc，这也是为什么把这个漏洞称为Triple\_Fetch的原因。下图就是攻击所使用的三组不同的字符串：



攻击所选择的NSXPC服务是“com.apple.CoreAuthentication.daemon”。对应的二进制文件是/System/Library/Frameworks/LocalAuthentication.framework/Support/coreauthd。原因是这个进程是root权限并且可以调用processor\_set\_tasks() API从而获取系统其他进程的send right[3]。下图是控制了pc后的crash report：

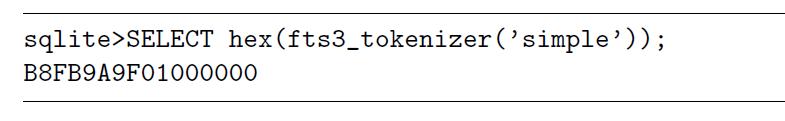


0x02 MLSqlite Sandbox Escape漏洞分析

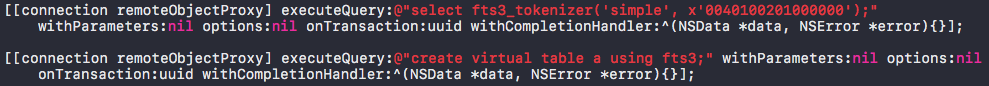
这个漏洞是我们OverSky团队（Cererdlong, Eakerqiu, Min (Spark) Zheng）发现的，并用于我们去年的OverSky私有越狱当中，但因为iOS 10加强了沙盒的规则，导致接口已经无法在沙盒内访问到了，但是这个漏洞在iOS 9.3.5(4s能升级到的最高版本)上依然可以使用。

漏洞出现在com.apple.medialibraryd.xpc这个NSXPC服务中，对应的bin在系统中的位置是：/System/Library/PrivateFrameworks/MusicLibrary.framework/Support/medialibraryd。这个NSXPC有一个接口是与sqlite操作相关的，但因为对调用者没有进行权限的检测，导致在沙盒内就可以对系统上任意的sqlite文件进行增删改查（漏洞一）。首先，你可以通过[[connection remoteObjectProxy] beginTransactionForDatabaseAtPath]方法创建或打开任意的sqlite文件。然后可以通过[[connection remoteObjectProxy] executeQuery]对这个sqlite文件执行sql语句。

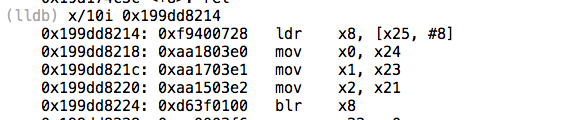
但是仅仅执行sql语句是不够的，我们的目标是控制pc。幸运的是，iOS 9上有一个很经典的SQLite fts3\_tokenizer分词器漏洞没有被修复（漏洞二）。关于这个漏洞，刚好长亭科技在BH2017上有一个相关的演讲[4]，里面讲到了这个漏洞在浏览器中的利用，这次我们讲一下这个漏洞在iOS上用户态过沙盒的用法。首先我们可以通过fts3\_tokenizer('simple')指令来泄露内存地址：



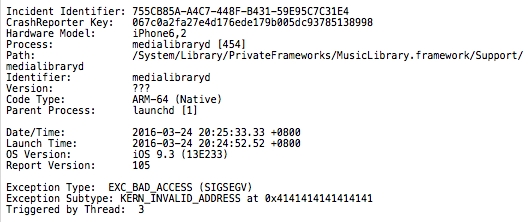
然后可以使用fts3\_tokenizer('simple’, addr)我们可以控制虚表，攻击代码如下：



在汇编代码中，x25的值会在我们的控制之中：



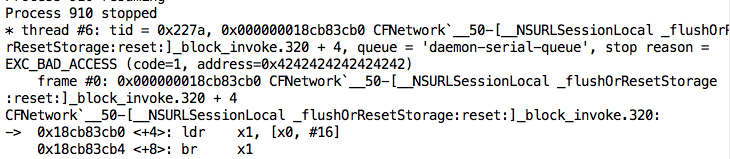
因此我们可以用这种方法控制pc：



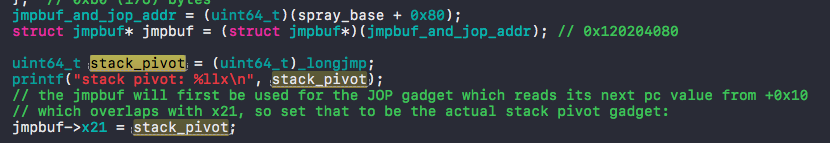
随后我们会介绍如何执行ROP以及任意代码。

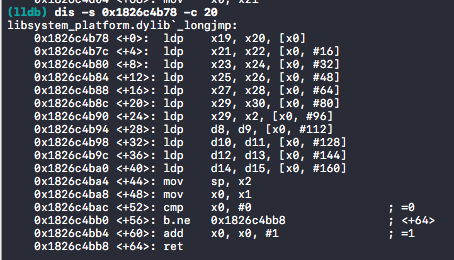
0x03 Triple\_Fetch JOP & ROP & 任意代码执行

利用漏洞Triple\_Fetch虽然可以控制pc，但是还不能控制栈，所以需要先做stack\_pivot，好消息是x0寄存器指向的xpc\_uuid对象是我们可以控制的：

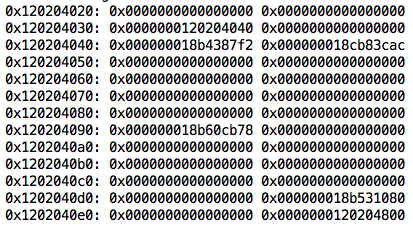


因此我们可以利用JOP跳转到\_longjmp函数作为来进行stack pivot，从而控制stack:

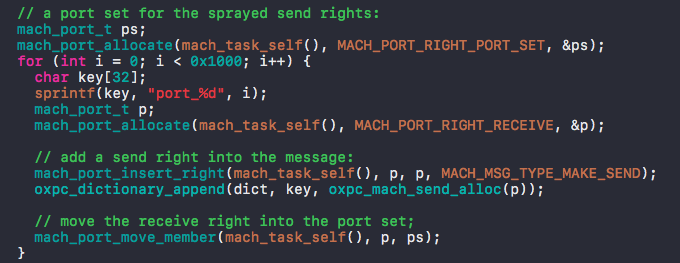




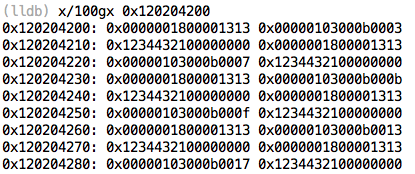
最终发送的用来做JOP的格式伪造的xpc\_uuid对象如下：



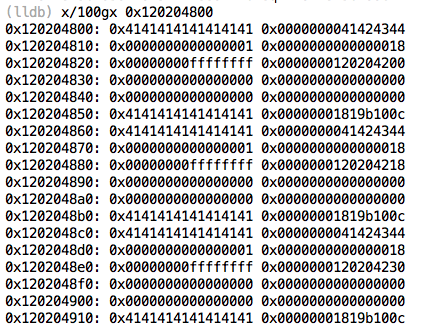
控制了stack就可以很容易的写rop了。但是beer目标不仅仅是执行rop，它还希望获取目标进程的task port并且执行任意二进制文件，因此除了exp，攻击端还用mach msg发送了0x1000个带有send right的port到目标进程中：



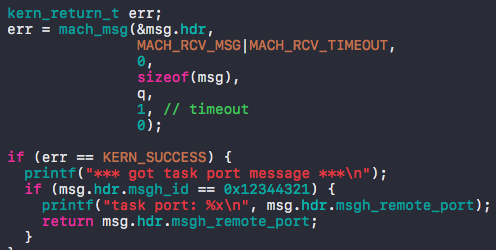
这些port的mach msg在内存中的位置和内容如下（msgh\_id都为0x12344321）：



随后，exp采用rop的方法对这些port进行遍历并且发送回发送端：

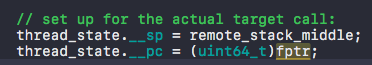


随后，攻击端会接收mach msg，如果获取到的msgh\_id为0x12344321的消息，说明我们成果得到了目标进程的task port：



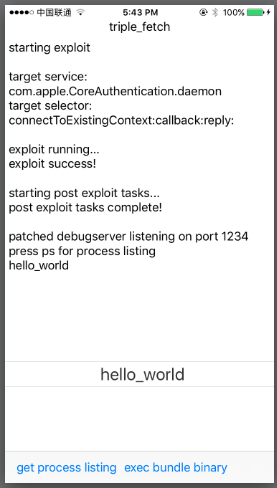
得到了task\_port后，sploit()函数就结束了，开始进入do\_post\_exploit()。do\_post\_exploit()也做了非常多的事情，首先是利用coreauthd的task port以及processor\_set\_tasks()获取所有进程的task port。这是怎么做到的呢？

利用coreauthd的task port我们可以利用mach\_vm\_\* API任意的修改coreauthd的内存以及寄存器，所以我们需要先开辟一段内存作为stack，然后将sp指向这段内存，再将pc指向我们想要执行的函数地址就可以让目标进程执行任意的函数了，具体实现在call\_remote()中：



随后我们控制coreauthd依次执行task\_get\_special\_port(), processor\_set\_default(), host\_processor\_set\_priv(),processor\_set\_tasks()等函数，来获得所有进程的task port并返回给攻击端（具体实现在get\_task\_ports()）中。接着，攻击端会遍历这个列表并筛选出amfid，launchd，installd，springboard这四个进程的task port。然后利用之前patch amfid的技巧，对amfid打补丁。最后再启动debugserver。

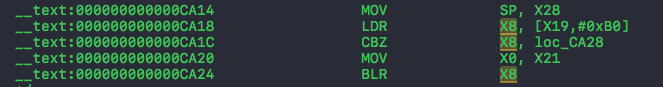
其实这个exp不但可以执行debugserver，还可以用来在沙盒外执行任意的二进制文件。只要把pocs文件夹下的hello\_world二进制文件替换成你自己的想要执行的二进制文件，编译安装后，点击ui中的exec bundle binary即可：



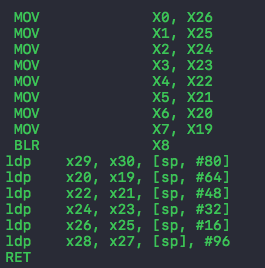
具体怎么做到的呢？秘密在spawn\_bundle\_binary()函数中，先在目标进程中调用chmod将bin改为0777，然后通过一系列的posix\_spawn API（类似fork()）在目标进程中执行该bin文件。

0x04 MLSqlite SBE JOP & ROP & 任意代码执行

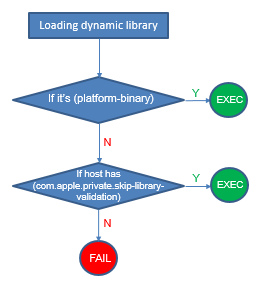
因为之前并没有beer的exp作参考，因此在我们的MLSqlite SBE利用中，我们选用的stack pivot为：



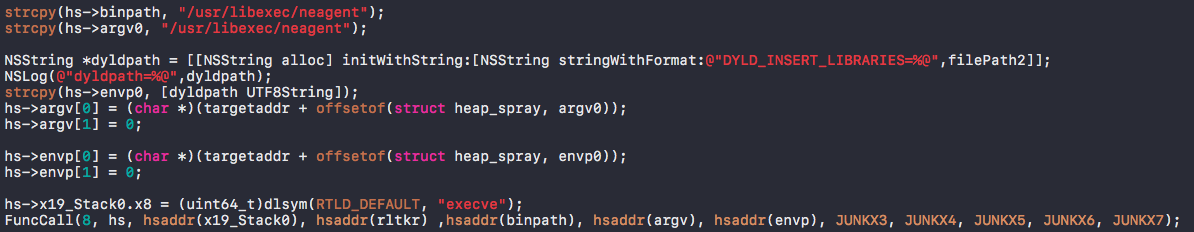
为了给X28赋值，找了一段比较长的JOP（5个gadget），并没有beer的exp简洁。随后利用一段万能gadget做到了ROP上的任意函数调用：



当然使用ROP执行代码又麻烦效率又低，我们当然希望能找到一种执行任意代码的方法，但是iOS并不允许不带”platform-binary” TeamID的二进制文件或动态链接库在沙盒外执行，除非这个加载这个库的bin有”com.apple.private.skip-library-validation” entitlement：



我们搜了一下iOS系统中所有的bin，发现neagent是唯一带有这个entitlement的bin。这个bin的作用是加载第三方的vpn库，这也是为什么它必须拥有这个entitlement的原因。因此，我们采用这个漏洞（漏洞三），让目标进程调用execve()执行/usr/libexec/neagent，同时使用DYLD\_INSERT\_LIBRARIES环境变量来执行我们的第三方库，从而做到了任意代码执行：



0x05 总结

本文介绍了2个NSXPC漏洞，分别是一个beer发现的通用NSXPC漏洞，和我们发现的MLSqlite漏洞。另外，还分析了两种iOS用户态上，用JOP做stack pivot以及利用ROP做到任意代码执行的攻击技术。当然，这些漏洞只是做到了沙盒外的代码执行，想要控制内核还需要一个或两个XNU或者IOKit的漏洞才行，并且苹果已经修复了yalu102越狱用的kpp绕过方法，因此，即使有了Triple\_Fetch漏洞，离完成全部越狱还有很大一段距离。

0x06 参考文献

1. <https://bugs.chromium.org/p/project-zero/issues/detail?id=1247>
2. <http://keenlab.tencent.com/zh/2017/08/02/CVE-2017-7047-Triple-Fetch-bug-and-vulnerability-analysis/>
3. <http://newosxbook.com/articles/PST2.html>
4. <https://www.blackhat.com/docs/us-17/wednesday/us-17-Feng-Many-Birds-One-Stone-Exploiting-A-Single-SQLite-Vulnerability-Across-Multiple-Software.pdf>