关于恢复kernelcache符号的问题

Author: wnagzihxa1n
Email: wnagzihxa1n@gmail.com

太长不看版：流水账啥结论都没有

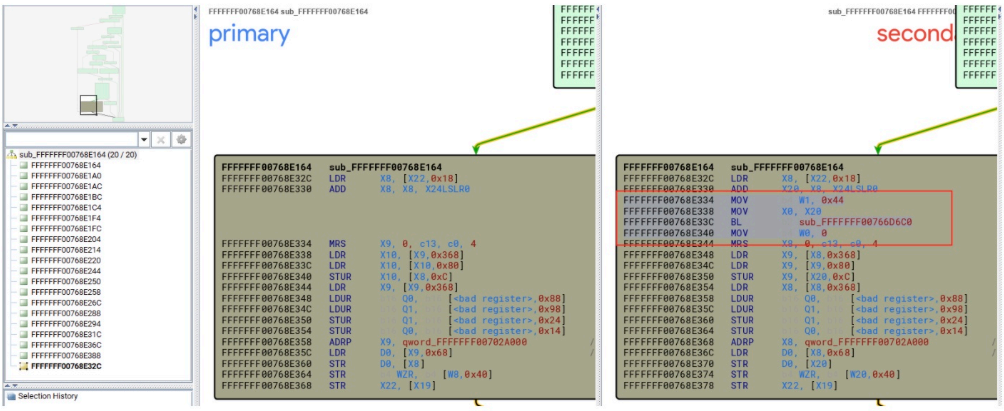
作为一个初级iOS安全研究爱好者，我在近期的漏洞复现实践中遇到了一个问题，就是如何恢复符号

- https://www.synacktiv.com/en/publications/ios-1-day-hunting-uncovering-and-exploiting-cve-2020-27950-kernel-memory-leak.html#

作者先通过bindiff发现了八个函数是有差异的，然后推出其中五个是添加了`bzero`函数的结论，这是怎么看出来的？

接着又说通过joker工具结合XNU源码可以确定几个有差异的函数是下面这几个，我对照着做的时候发现这里并不像作者说的这么简单

> These results are beyond all expectations! There are only 8 functions slightly changing between the two versions, all in the kernel!
>
> Among these 8 results, 2 are actually minor instructions ordering changes. In the 6 remaining ones, 5 of them have an added call to *bzero*, which make them the perfect candidates for a memory leak vulnerability involving a "memory initialization issue" :)



Added bzero call

> iOS kernelcaches usually lack symbols, but some entry points such as mach traps can be easily identified, using e.g. joker tool. Debug strings along with public XNU sources also allow renaming many functions, and we could identify the 5 patched functions as:
>
> - mach_msg_send
> - mach_msg_overwrite
> - ipc_kmsg_get
> - ipc_kmsg_get_from_kernel
> - ipc_kobject_server

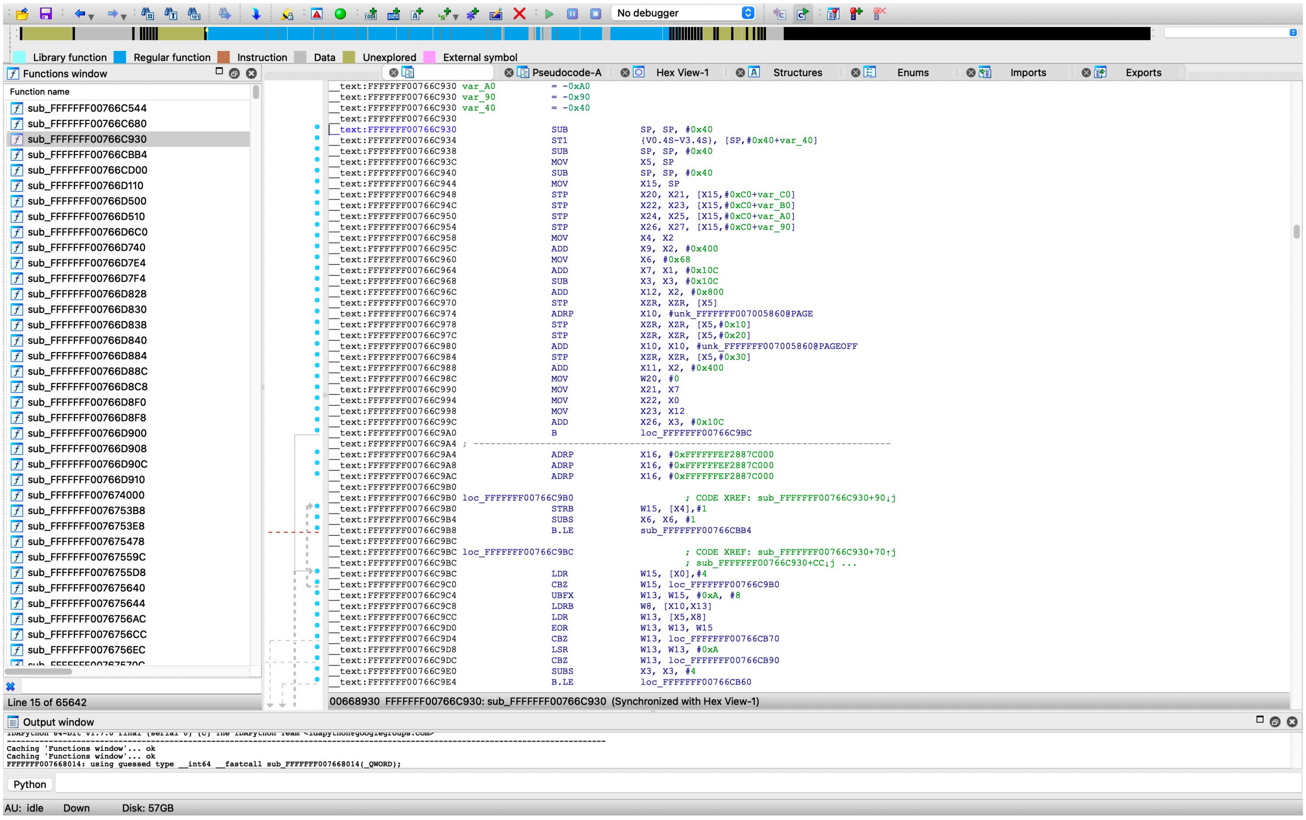
我按照时间线的形式来记录我在遇到这个问题时是如何思考与解决的

我下载了12.4.8和12.4.9两个版本的固件，其中12.4.8是漏洞版本，12.4.9是补丁版本

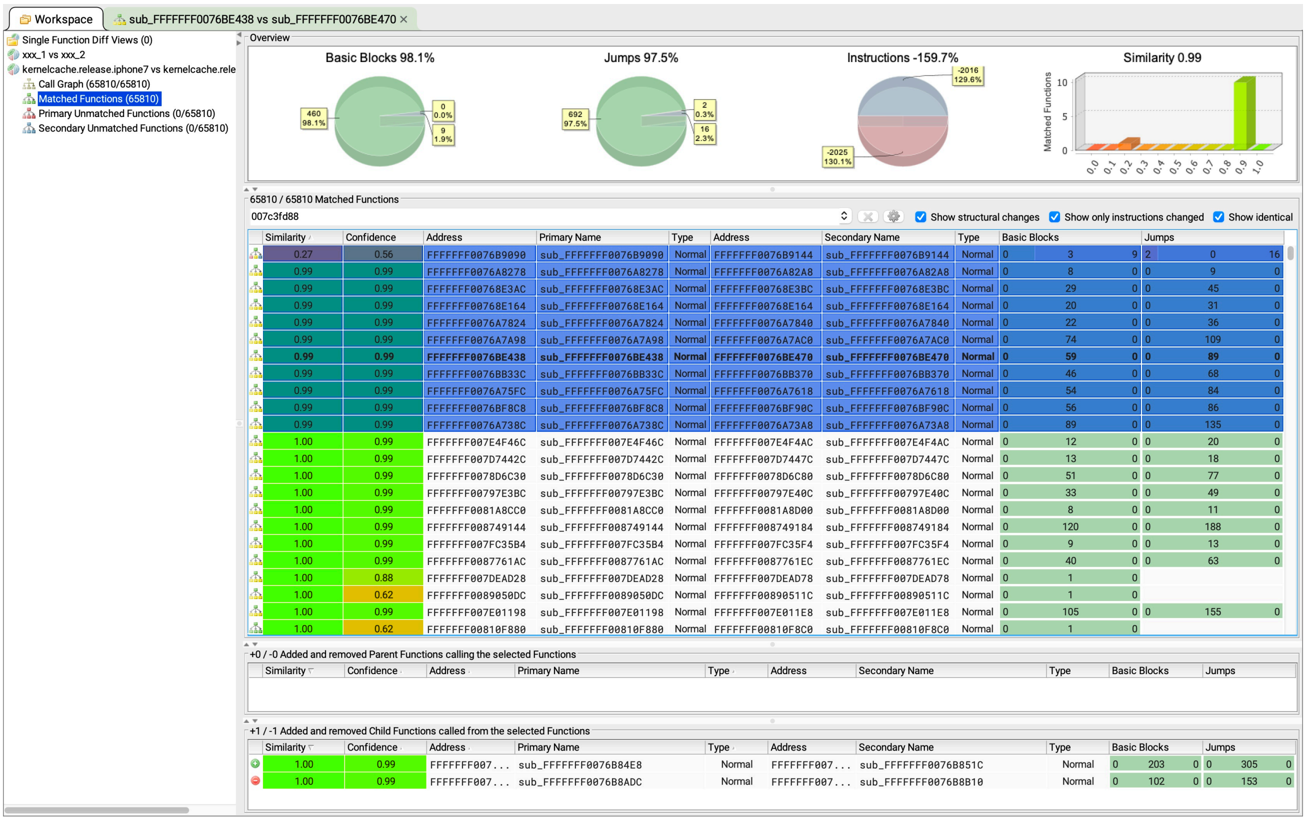
解压缩ipsw固件包，获取压缩后的kernelcache文件，再使用lzssdec进行解压缩kernelcache，获取到可反编译kernelcache文件

以下将12.4.8的kernelcache记为`kc_12.4.8`，将12.4.9的kernelcache记为`kc_12.4.9`

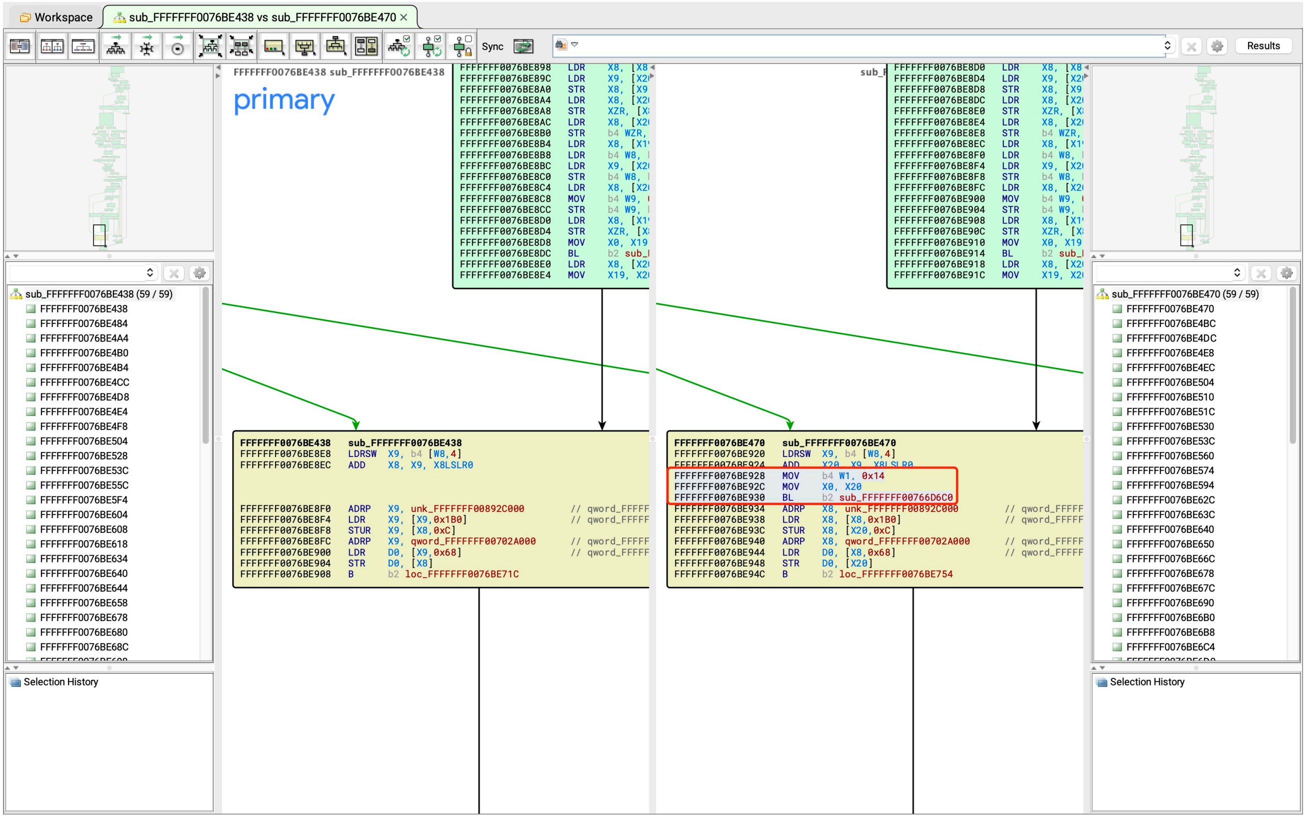
当我在macOS平台用IDA 7.0反编译`kc_12.4.8`的时候，我发现了符号的问题，真的是一丁点都没有，`kc_12.4.9`也是如此



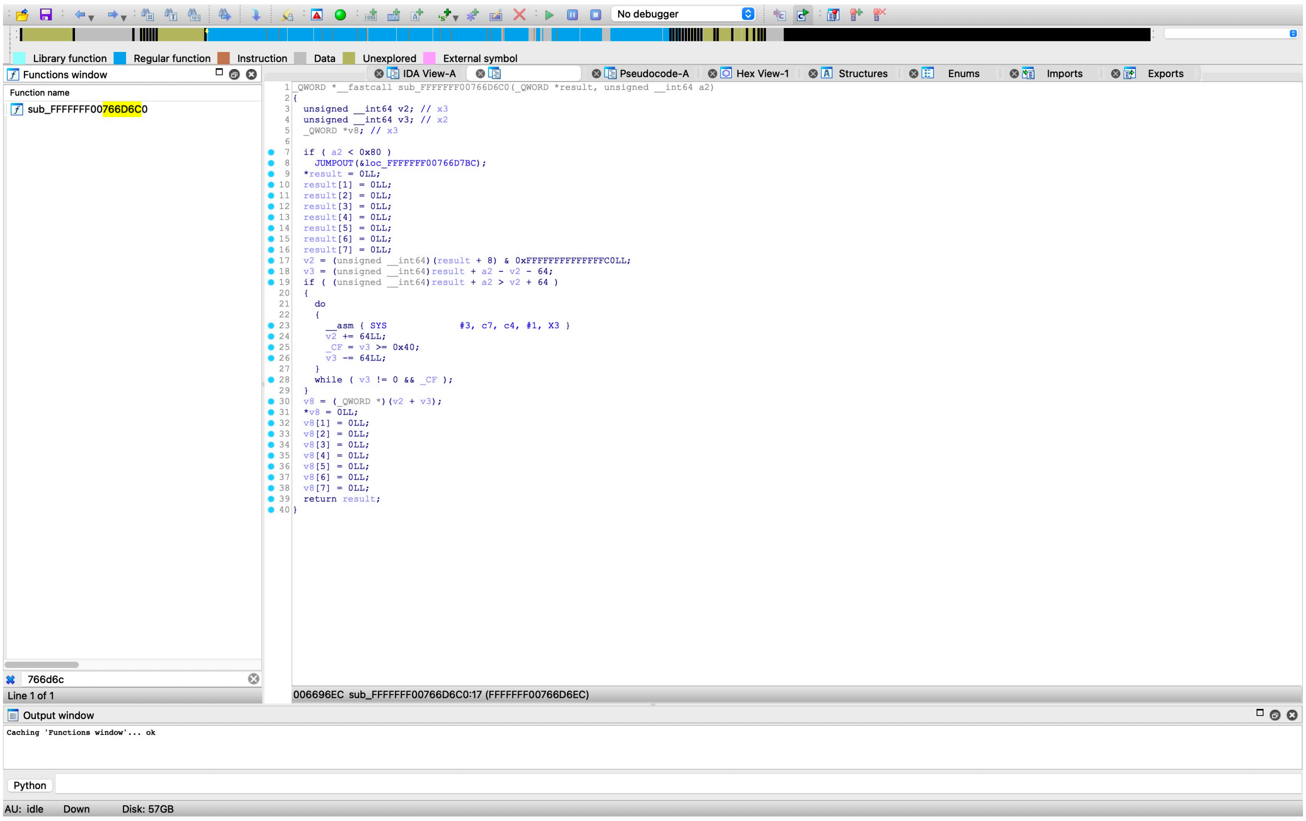
使用bindiff进行比对，发现比对结果与作者有所差别，这里猜测是因为我使用的是IDA 7.0，对应的bindiff是5导致的，IDA 7.5可以使用bindiff 6



进入到作者所说的函数，添加的是`sub_FFFFFFF00766D6C0`



在IDA里找到这个函数，想不明白作者是如何通过这种代码判断出来这是一个`bzero()`函数的



随着疑惑越来越多，第一反应当然是谷歌搜索，看看有没有前辈们的经验可以借鉴学习的

最先找到的是jtool2，它有一个`--analyze`选项可以获取到kernelcache文件里的符号

- http://newosxbook.com/tools/jtool.html

使用`--analyze`生成符号文件

```
➜  jtool2 ./jtool2 --analyze kernelcache.release.iphone7
```

生成的符号文件格式有如下七种类型，可能有更多，我这里只关注到这几个，其中又以`_func_xxxxxxxxxxxxxxxx`最多，数量还是太少

```
0xfffffff00767b9c8|_func_fffffff00767b9c8|
0xfffffff00767b9f4|_func_fffffff00767b9f4|

0xfffffff00767bfc0|_strlen|Rule type 0 #1
0xfffffff00767c050|_getsectbynamefromheader|Rule type 1 #4

0xfffffff007694be0|_ipc_object_translate|Flow for _mk_timer_destroy_trap, call 0
0xfffffff0076ab374|_ipc_object_translate|Flow for __Xmach_port_guard, call 1

0xfffffff0074fe4c8|__ZTV35IOAccessoryPowerSourceItemUSBDevice|
0xfffffff0074fe610|__ZTV34IOAccessoryPowerSourceItemExternal|
0xfffffff0074fe758|__ZTV42IOAccessoryPowerSourceItemUSB_ChargingPort|
0xfffffff0074fe880|__ZTV47IOAccessoryPowerSourceItemUSB_DataContactDetect|
0xfffffff0074fe9c8|__ZTV43IOAccessoryPowerSourceItemUSB_TypeC_Current|
0xfffffff0074feb10|__ZTV33IOAccessoryPowerSourceItemBrickID|

0xfffffff008713f4c|IOAudioCodecsUserClient method 0|IOAudioCodecsUserClient method 0
0xfffffff008713f58|IOAudioCodecsUserClient method 1|IOAudioCodecsUserClient method 1
0xfffffff008713f64|IOAudioCodecsUserClient method 2|IOAudioCodecsUserClient method 2
0xfffffff008713f78|IOAudioCodecsUserClient method 3|IOAudioCodecsUserClient method 3
0xfffffff008713f84|IOAudioCodecsUserClient method 4|IOAudioCodecsUserClient method 4
0xfffffff008713f90|IOAudioCodecsUserClient method 5|IOAudioCodecsUserClient method 5

0xfffffff008941278|redirect|
0xfffffff00894127c|hlim|
0xfffffff008941280|defmcasthlim|
0xfffffff008941284|accept_rtadv|

0xffffffffffffff80|skywalk.mem|zone
0xffffffffffffffa0|necp.clientfd|zone
```

本来想着jtool2生成的符号会很漂亮，我就写了一个脚本来自动化恢复符号，现在看来要吃灰了

思路就是获取所有的函数，然后获取地址跟函数名，匹配符号文件的数据，再把符号替换掉函数名

这里会有几个问题，第一个问题就是上面的符号其实并不能直接用，需要二次处理，比如把空格替换成下划线，还有重名函数的问题等等

那么这个方法到这里就暂时放一边了

我又继续搜，发现了一个`ida_kernelcache`的项目，不过这个三年前的项目年久失修，我没跑起来，这个脚本应该是要好好研究一下的，学习作者的解析思路

- https://github.com/bazad/ida_kernelcache

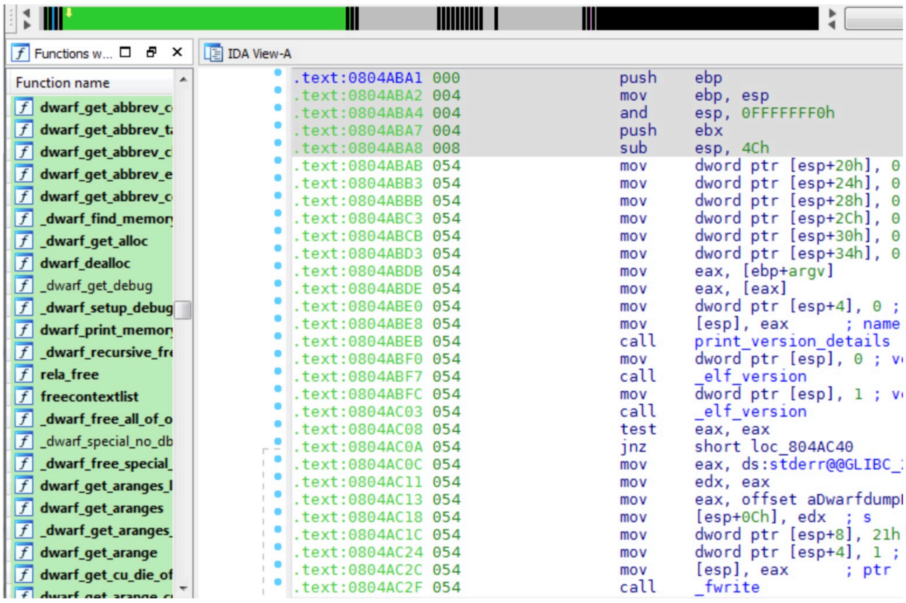
在自己研究无果后，我选择直接找大佬问解决方法

一问才发现原来这么些年我竟然没几个认识的搞iOS系统安全的朋友

不过总归还是有收获的，一位师傅建议我看一下IDA的Lumina功能，另一位师傅跟我说iOS的一个beta版本包含有调试符号的kernelcache

首先来讲Lumina，这是从IDA 7.2开始引进的一个实验性功能，它的作用就是动态从Lumina服务器获取函数的数据，比如我正在分析一个静态编译的固件，然后使用Lumina，它可以将函数的哈希发送到Lumina服务器匹配再返回对应的数据，相当于一个动态的FLIRT

- Finally, with IDA v7.2 we introduce a new (experimental) Lumina server. Think of it as about a public storage of popular function patterns. The lumina server holds metadata about functions (like names, comments, prototypes, operand representation, etc). IDA can ask Lumina to recognize functions in the current database and apply the corresponding metadata. It is like FLIRT but it is dynamic, stores more information, and can recognize functions that FLIRT cannot.



(yes, you guessed it right, green means lumina 😉)

因为众所周知的原因，Windows的IDA目前有最新的7.5，macOS只有7.0，而Lumina是从IDA 7.2开始引进的，所以我们切换到Windows的IDA 7.5

又是众所周知的原因，我们手上的IDA不能访问Lumina服务器，有位大佬搭建了一个私服

- https://github.com/naim94a/lumen

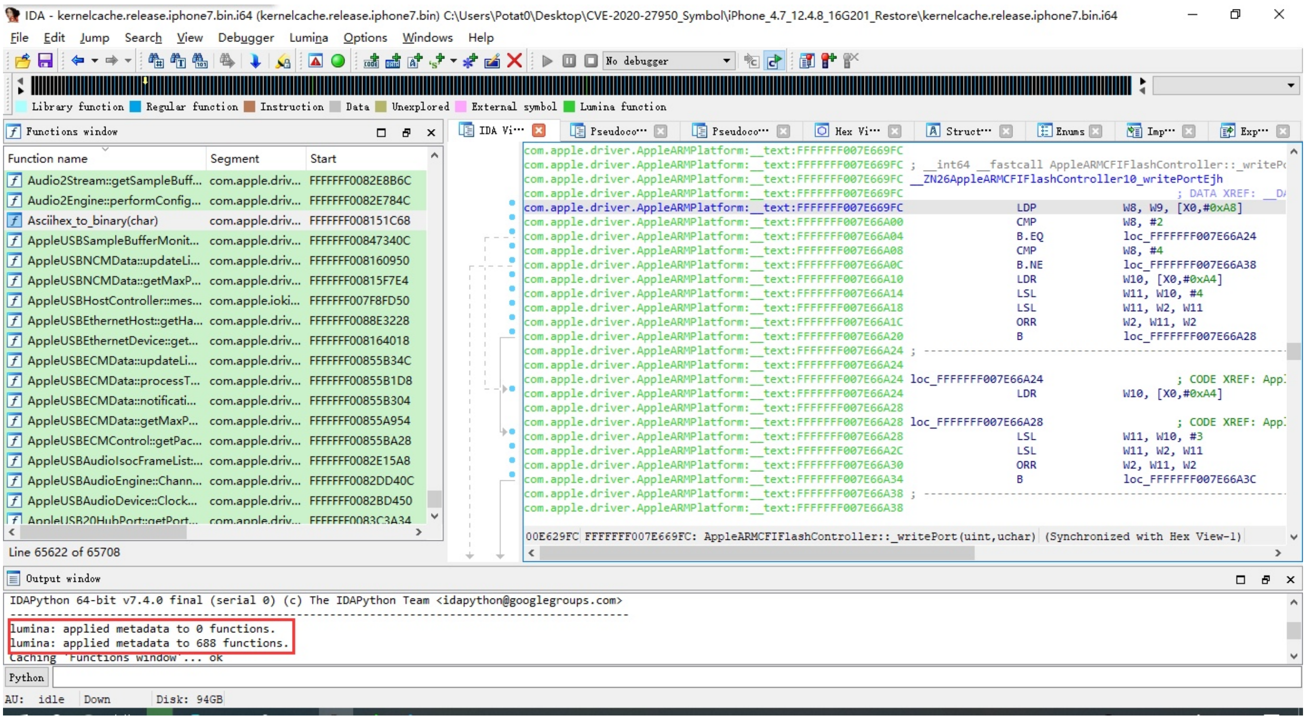
修改`{IDA_HOME}\cfg\ida.cfg`

```
LUMINA_HOST = "lumen.abda.nl";
LUMINA_PORT = 1234
```

下载`hexrays.crt`放到IDA根目录，重启IDA即可

- https://lumen.abda.nl/cert

我这里尝试了一下，恢复了一小部分符号，距离舒舒服服的分析还是有差距



另一个是师傅和我说的一个拥有调试符号的kernelcache

- https://twitter.com/tihmstar/status/1295814618242318337

从这个地址下载

- https://updates.cdn-apple.com/2020SummerSeed/fullrestores/001-32635/423F68EA-D37F-11EA-BB8E-D1AE39EBB63D/iPhone11,8,iPhone12,1_14.0_18A5342e_Restore.ipsw

按照上面的步骤解压缩ipsw固件获得一个`kernelcache.research.iphone12b`，对其进行解析

```
→ jtool2 ./jtool2 -dec kernelcache.research.iphone12b
```

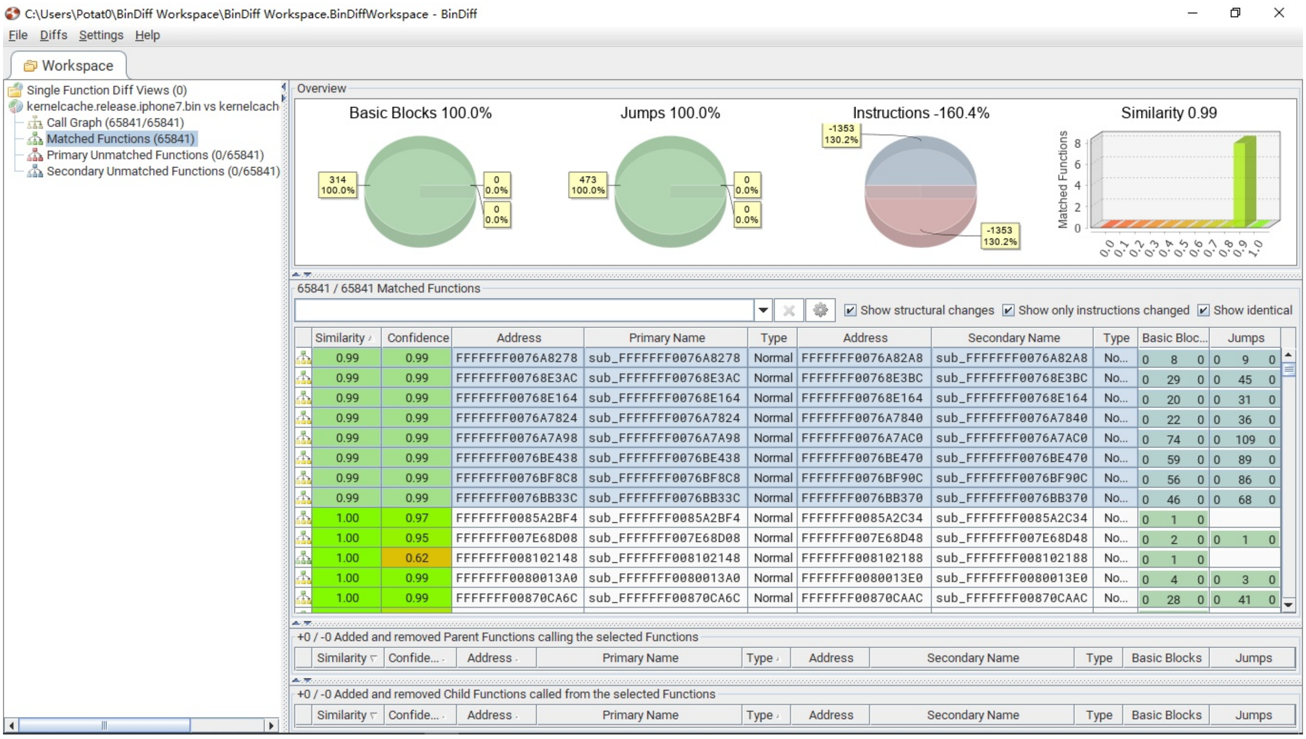
解析完成后会生成文件`/tmp/kernel`，拷贝重命名为`kernelcache.research.iphone12b.bin`

尝试分析提取符号，发现有15万+的符号，惊喜！以下将其记为`kc_symbols`

```
→ jtool2 ./jtool2 --analyze kernelcache.research.iphone12b.bin
...
opened companion file ./kernelcache.research.iphone12b.bin.ARM64.CCA1C472-EE81-32F2-8AB8-2ADD55281591
Dumping symbol cache to file
Symbolicated 150904 symbols and 47 functions
```

那么此时我们拥有两个一丁点符号都没有的`kc_12.4.8`和`kc_12.4.9`，以及一个全是符号的`kc_symbols`

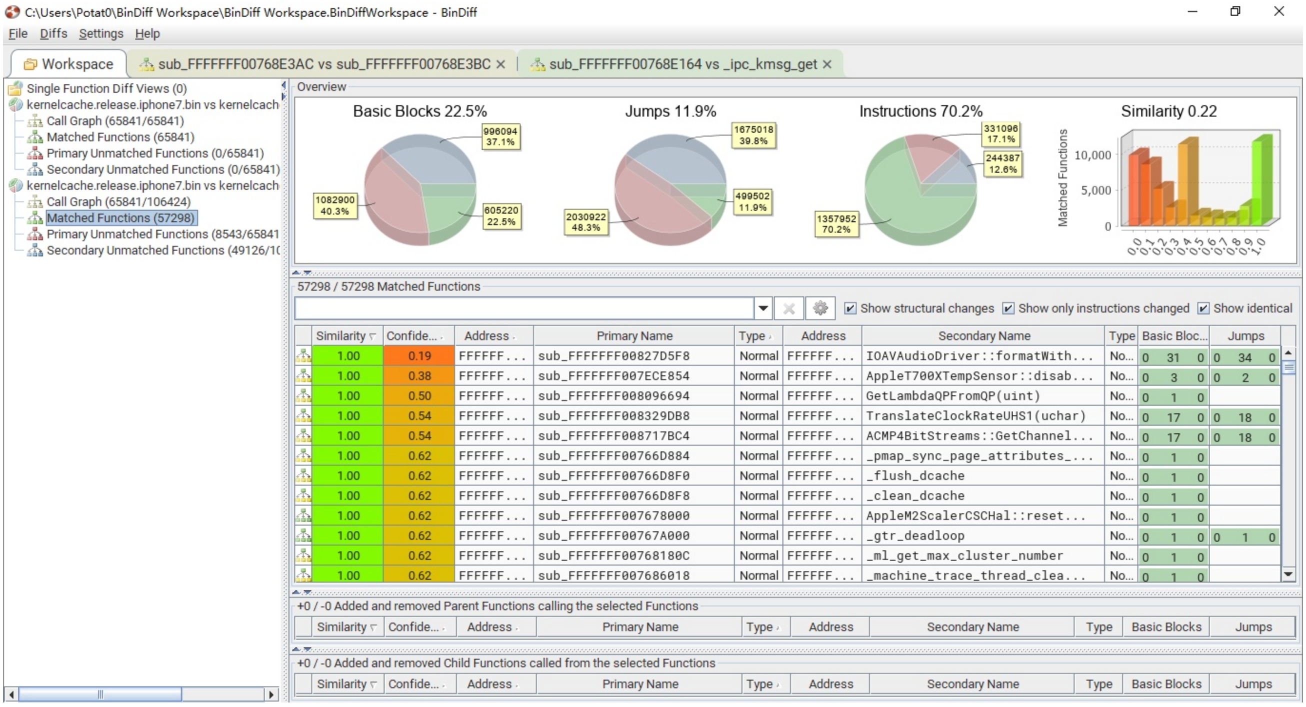
首先我们需要对比出补丁版本修改后的函数，为了结果更加准确，这里使用Windows平台的IDA 7.5，有差异的函数也是八个



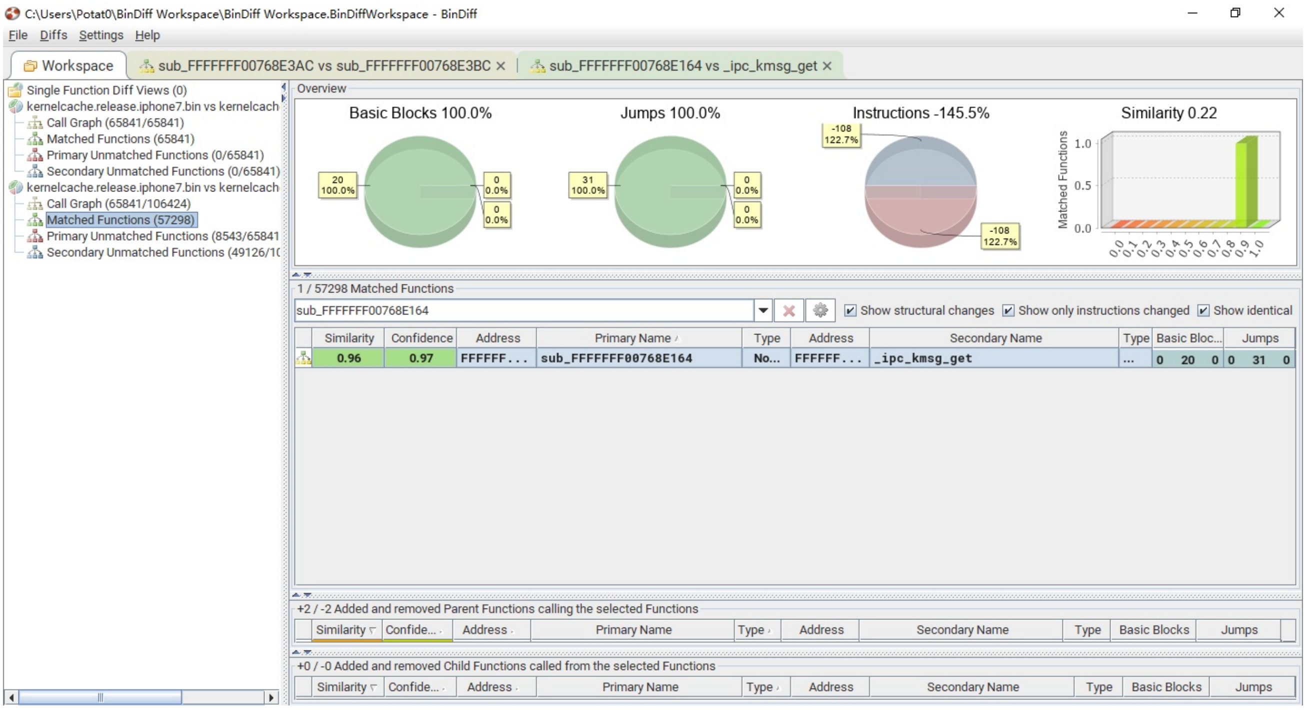
对diff结果做一个记录，方便后面搜索

kc_12.4.8_address	kc_12.4.8_func_name	kc_12.4.9_address	kc_12.4.9_func_name
FFFFFFF0076A8278	sub_FFFFFFF0076A8278	FFFFFFF0076A82A8	sub_FFFFFFF0076A82A8
FFFFFFF00768E3AC	sub_FFFFFFF00768E3AC	FFFFFFF00768E3BC	sub_FFFFFFF00768E3BC
FFFFFFF00768E164	sub_FFFFFFF00768E164	FFFFFFF00768E164	sub_FFFFFFF00768E164
FFFFFFF0076A7824	sub_FFFFFFF0076A7824	FFFFFFF0076A7840	sub_FFFFFFF0076A7840
FFFFFFF0076A7A98	sub_FFFFFFF0076A7A98	FFFFFFF0076A7AC0	sub_FFFFFFF0076A7AC0
FFFFFFF0076BE438	sub_FFFFFFF0076BE438	FFFFFFF0076BE470	sub_FFFFFFF0076BE470
FFFFFFF0076BF8C8	sub_FFFFFFF0076BF8C8	FFFFFFF0076BF90C	sub_FFFFFFF0076BF90C
FFFFFFF0076BB33C	sub_FFFFFFF0076BB33C	FFFFFFF0076BB370	sub_FFFFFFF0076BB370

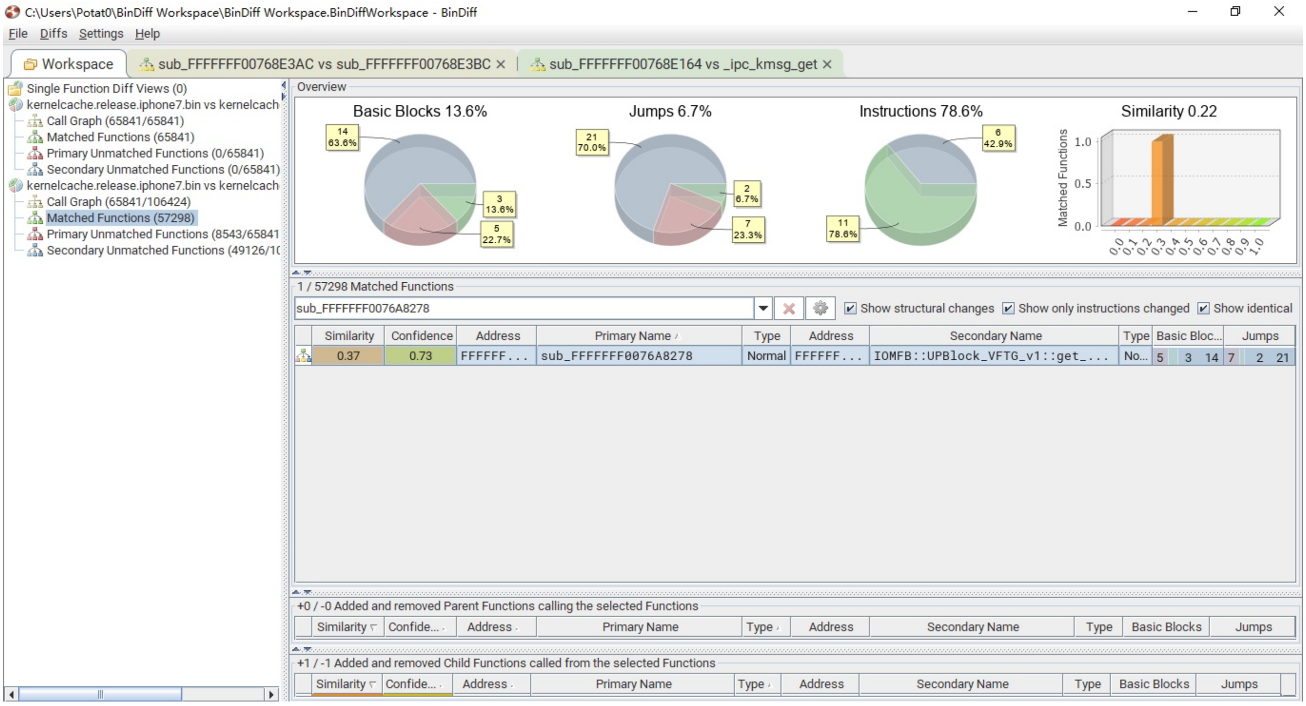
我们再对`kc_12.4.8`和`kc_symbols`进行比对



有识别效果好的



也有识别效果可能有错的



八个差异函数对比结果记录如下

kc_12.4.8_address	kc_12.4.8_func_name	kc_12.4.9_address	kc_12.4.9_func_name	bindiff_symbol
FFFFFFF0076A8278	sub_FFFFFFF0076A8278	FFFFFFF0076A82A8	sub_FFFFFFF0076A82A8	IOMFB::UPBlock_VFTG_v1::get_expected_timings(IOMFB::UPBlock_VFTG::Timings *)
FFFFFFF00768E3AC	sub_FFFFFFF00768E3AC	FFFFFFF00768E3BC	sub_FFFFFFF00768E3BC	_ipc_kmsg_get_from_kernel
FFFFFFF00768E164	sub_FFFFFFF00768E164	FFFFFFF00768E164	sub_FFFFFFF00768E164	_ipc_kmsg_get
FFFFFFF0076A7824	sub_FFFFFFF0076A7824	FFFFFFF0076A7840	sub_FFFFFFF0076A7840	_mach_gss_accept_sec_context_v2
FFFFFFF0076A7A98	sub_FFFFFFF0076A7A98	FFFFFFF0076A7AC0	sub_FFFFFFF0076A7AC0	AppleCS46L21IDPT::_startTransfer(AppleCS46L21IDPT::IDIO_Cmd_Packet *,ulong long,ulong long)
FFFFFFF0076BE438	sub_FFFFFFF0076BE438	FFFFFFF0076BE470	sub_FFFFFFF0076BE470	_ipc_port_send_turnstile_prepare
FFFFFFF0076BF8C8	sub_FFFFFFF0076BF8C8	FFFFFFF0076BF90C	sub_FFFFFFF0076BF90C	_ptmx_get_ioctl
FFFFFFF0076BB33C	sub_FFFFFFF0076BB33C	FFFFFFF0076BB370	sub_FFFFFFF0076BB370	_vm_compressor_pager_reap_pages

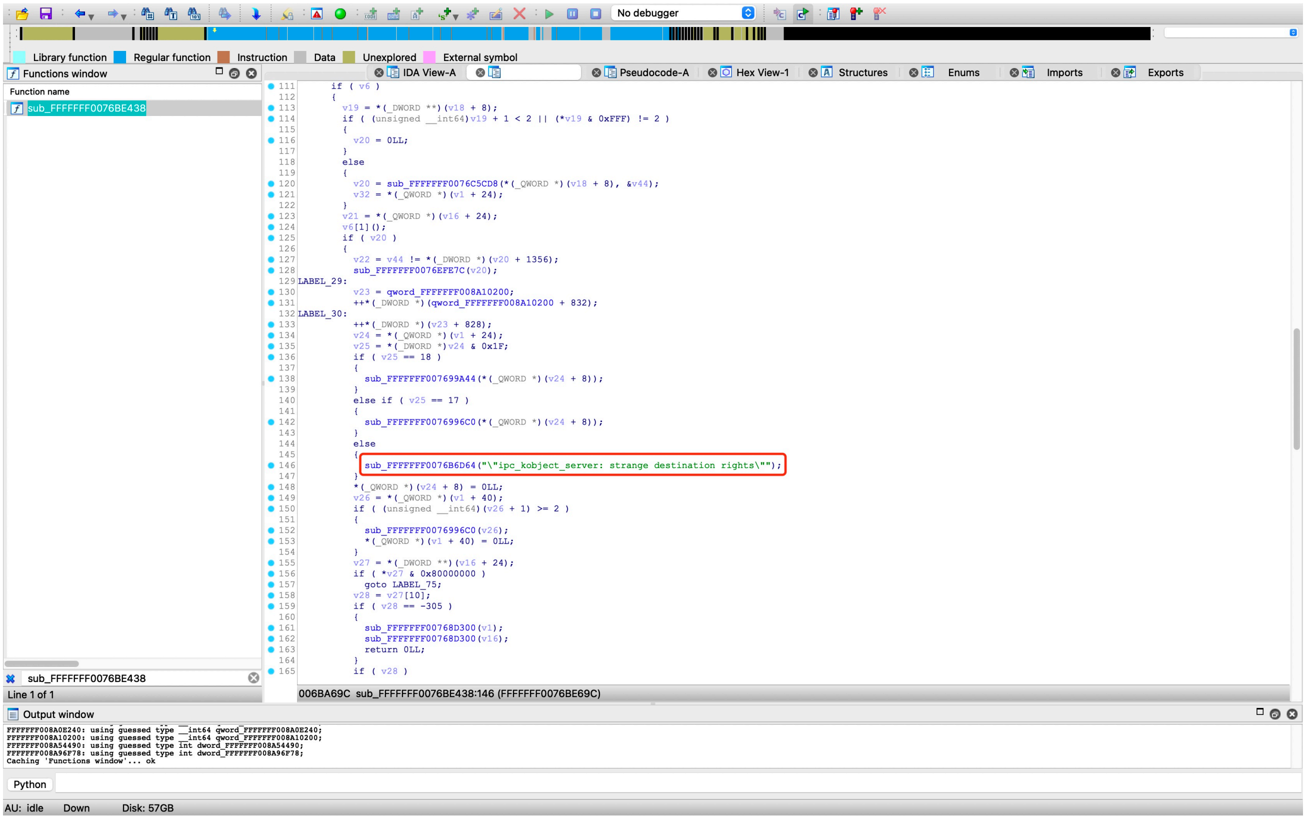
表格太大，做一下精简，提取出添加了`bzero()`的五个函数，有两个相似度超过百分之五十的能够正确匹配，有三个函数相似度都在百分之三十以下，并没有正确匹配上

kc_12.4.8_func_name	kc_12.4.9_func_name	Similarity	bindiff_symbol	true_symbol
sub_FFFFFFF00768E3AC	sub_FFFFFFF00768E3BC	0.58	_ipc_kmsg_get_from_kernel	ipc_kmsg_get_from_kernel
sub_FFFFFFF00768E164	sub_FFFFFFF00768E164	0.96	_ipc_kmsg_get	ipc_kmsg_get
sub_FFFFFFF0076A7824	sub_FFFFFFF0076A7840	0.13	_mach_gss_accept_sec_context_v2	xxxxx
sub_FFFFFFF0076BE438	sub_FFFFFFF0076BE470	0.11	_ipc_port_send_turnstile_prepare	xxxxx
sub_FFFFFFF0076BF8C8	sub_FFFFFFF0076BF90C	0.28	_ptmx_get_ioctl	xxxxx

现在还剩下三个函数没有匹配到`mach_msg_send`，`mach_msg_overwrite`，`ipc_kobject_server`

再一次陷入僵局，突然想到作者说的XNU源码字符串也可以找一下

反编译三个没有匹配到符号的函数，发现函数`sub_FFFFFFF0076BE438`存在字符串



通过源码找到一个函数

```
default:
    panic("ipc_kobject_server: strange destination rights");
}
```

现在还剩两个函数没有匹配到符号，又没有符号

kc_12.4.8_func_name	kc_12.4.9_func_name	Similarity	bindiff_symbol	true_symbol
sub_FFFFFFF00768E3AC	sub_FFFFFFF00768E3BC	0.58	_ipc_kmsg_get_from_kernel	ipc_kmsg_get_from_kernel
sub_FFFFFFF00768E164	sub_FFFFFFF00768E164	0.96	_ipc_kmsg_get	ipc_kmsg_get
sub_FFFFFFF0076A7824	sub_FFFFFFF0076A7840	0.13	_mach_gss_accept_sec_context_v2	xxxxx
sub_FFFFFFF0076BE438	sub_FFFFFFF0076BE470	0.11	_ipc_port_send_turnstile_prepare	ipc_kobject_server
sub_FFFFFFF0076BF8C8	sub_FFFFFFF0076BF90C	0.28	_ptmx_get_ioctl	xxxxx

这就有点难为小王了

我又找到了一篇文章

- https://supergithuber.github.io/ios/exportiOSSystemSymbol.html

这里的原理我暂时还没有弄明白，我这里只做一下记录

前面所有的步骤我们都是将ipsw固件解压缩之后，直接获取解压缩根目录下的`kernel.release.iphonexxx`文件进行处理

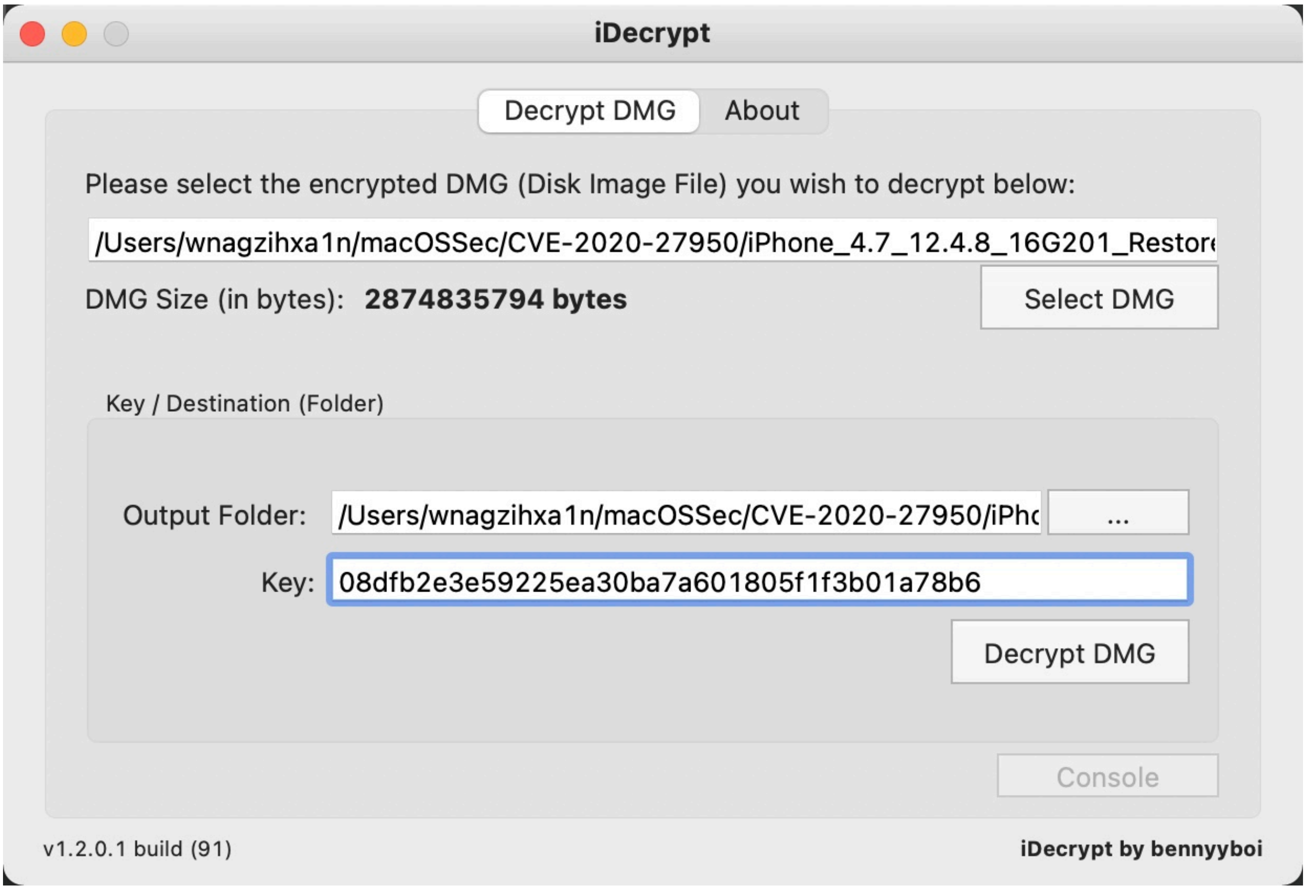
这篇文章里的方式是使用解压缩的其它文件来获取系统符号，以12.4.8的固件为例，解压后有下面这些文件

```
➜  iPhone_4.7_12.4.8_16G201_Restore ls -al
drwxr-xr-x@ 19 wnagzihxa1n  staff          608 Jan  5 15:26 .
drwxr-xr-x  10 wnagzihxa1n  staff          320 Jan  5 15:39 ..
-rw-r--r--@  1 wnagzihxa1n  staff  2874835794 Jan  9  2007 038-60223-004.dmg
-rw-r--r--@  1 wnagzihxa1n  staff    93846555 Jan  9  2007 038-60285-004.dmg
-rw-r--r--@  1 wnagzihxa1n  staff    91602971 Jan  9  2007 038-60305-004.dmg
-rw-r--r--@  1 wnagzihxa1n  staff      128367 Jan  9  2007 BuildManifest.plist
drwxr-xr-x@ 10 wnagzihxa1n  staff          320 Jan  9  2007 Firmware
-rw-r--r--@  1 wnagzihxa1n  staff          985 Jan  9  2007 Restore.plist
-rw-r--r--@  1 wnagzihxa1n  staff    14061377 Jan  9  2007 kernelcache.release.iphone7
```

下载工具iDecrypt

- https://supergithuber.github.io/Resources/iDecrypt-Mac-build91-bennyyboi.zip

选择解压缩后的最大的那个dmg文件



解压缩的key从下面这个网站搜索

- https://www.theiphonewiki.com/wiki/Firmware/iPhone/12.x

由于iOS 10之后不再加密，所以这里其实可以直接双击`038-60223-004.dmg`挂载读取文件，emmmmmm

在如下目录找到一个`dyld_shared_cache_arm64`文件

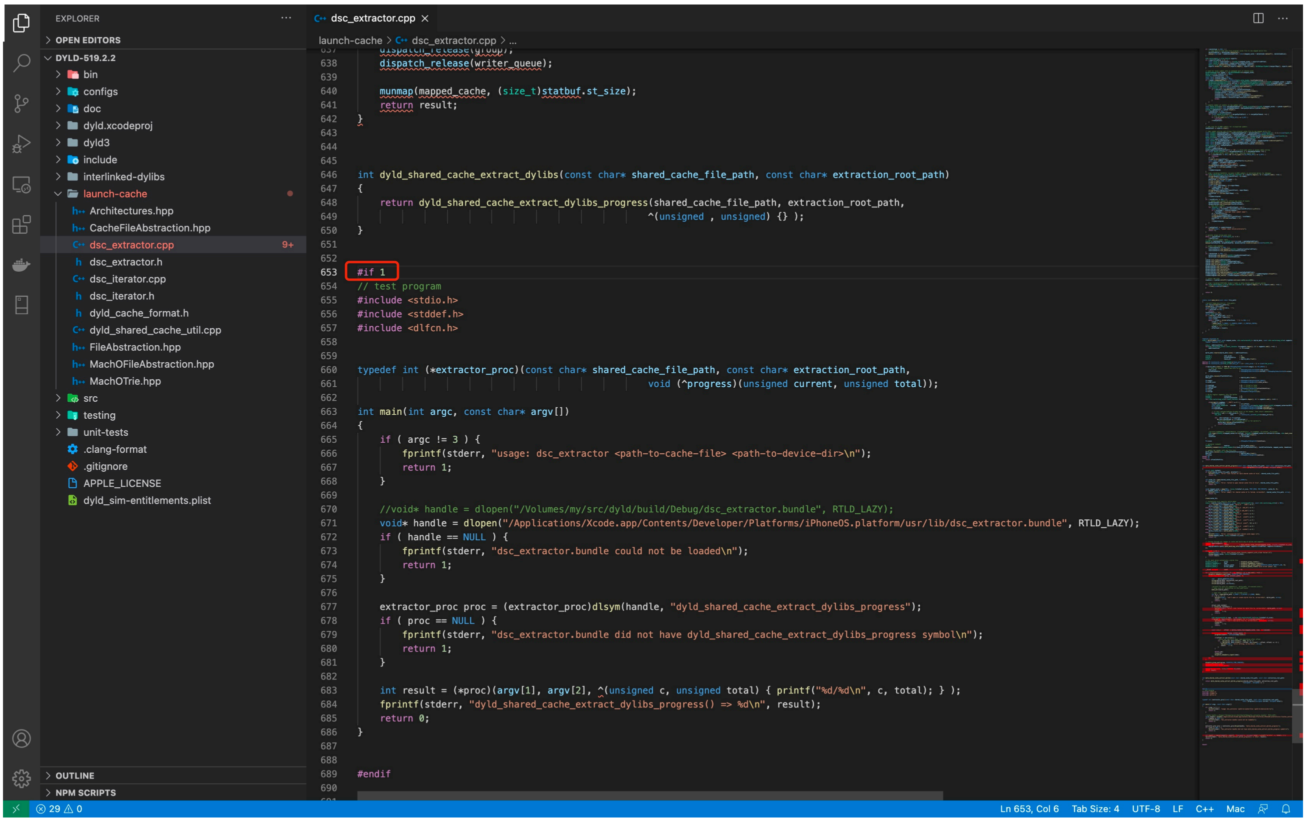
```
/System/Library/Caches/com.apple.dyld/
```

提取出来做解析的准备

接下来编译解析工具dyld，我使用的是`dyld-519.2.2`，高版本编译会提示找不到头文件

- https://opensource.apple.com/tarballs/dyld/
- https://opensource.apple.com/tarballs/dyld/dyld-519.2.2.tar.gz

如图修改`dyld-519.2.2/launch-cache/dsc_extractor.cpp`，将`0`改为`1`



使用clang编译

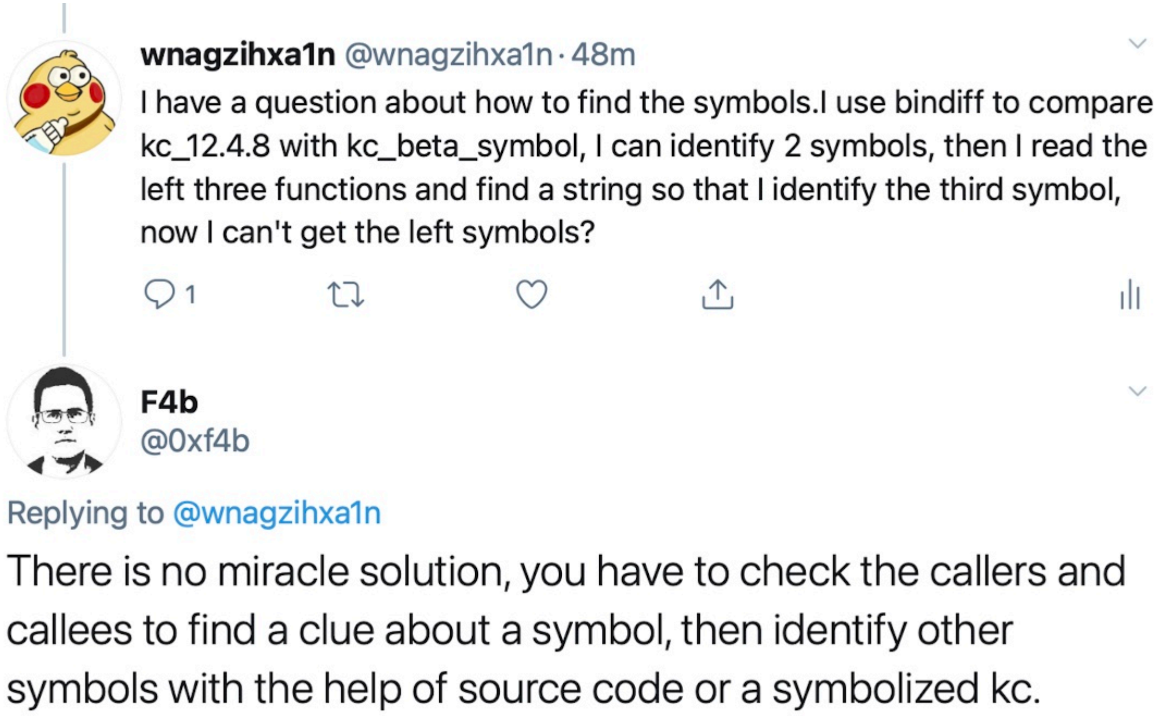
```
➜  launch-cache clang++ -o dsc_extractor ./dsc_extractor.cpp dsc_iterator.cpp
```

使用编译的`dsc_extractor`提取`dyld_shared_cache_arm64`的符号

```
➜  iPhone_4.7_12.4.8_16G201_Restore ./dsc_extractor dyld_shared_cache_arm64 dyld_shared_cache_arm64_symbol
```

然后我拿到了一堆动态库，喵喵喵，不是说好的提取符号吗？？？

实在是没辙了，我决定问这篇文章的作者，一番搜索找到了作者的推特，我猜这种文章一般作者都会转发一下，果不其然，我就留言描述了我的问题，作者回复的也很迅速，大概意思就是没啥特别的好办法，从调用关系，字符串，或者有符号的kernelcache上手慢慢找



我最后跟brightiupzl师傅请教了一下这个问题，大概意思就是：我这个diff没有符号固件来找漏洞的操作，对于新手来说难度有点大

那iOS固件符号恢复这事到这里就暂时告一段落了，等我对iOS系统有更进一步的学习成果后我再回过头来思考这个问题，看是否有更好一点的解决方案，起码`bzero()`这样的得自动识别出来