Mobile Software Security Research

March 13, 2015

iOS Kernel Exploitation: Fuzzing on IOKit

iOS has many similarities as Mac OSX on kernel components and functions. One way to study their kernel mechanism is to look into XNU open source project(http://www.opensource.apple.com/source/xnu/), which is FreeBSD kernel and shares many features with iOS/MacOS because of Darwin Unix history.

In the kernel there are three important components:

- Mach: Low level abstraction of kernel
- BSD: High level abstraction of kernel
- IOKit: Apple kernel extension framework

where IOKit framework was developed by Apple in C++ subset implementation (See Fig. 1).

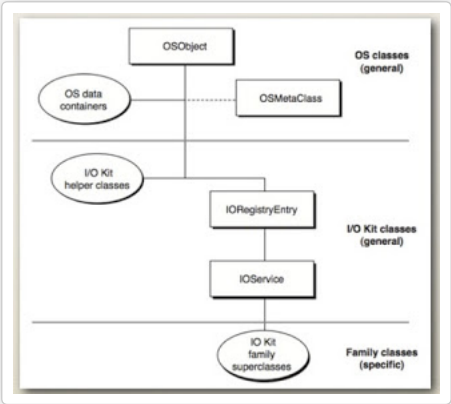


Fig. 1 OOP structure of IOKit.

All the classes have a root object, called OS Object. OS Object mainly overwrite new operator to allocate memory, and declare init method to initialize the object self. The OSMetaClass is a runtime object type checker, and also provides safe type conversion functionality. Another important class is IOService. It defines an interface for most kernel extensions, and implemented basic methods like `init/start/stop/attach/detach/probe`. You can download a tool called **ioreg** to list all the attached IOService. Suppose we are going to write a driver extension by ourselves, the above information would be very helpful. Since we need to implement our own server, our driver must provide service to upper layer. We can inherit IOService simply to cast some parameters or get some information. After that, we need to modify plist, so that system knows which driver it is for.

Apple stores all the kernel modules into one cache file called Kernelcache, and iBoot will load the whole kernel cache and jump to entry so that the kernel loading speed is increased greatly. The location of encrypted kernelcache is here:

```
/System/Library/Caches/com.apple.kernelcaches/kernelcache
```

For older devices (A4 chip or older), it is easy to decrypt original cache with IV+KEY using xpwntool, with the help of bootrom vulnerability of A4 chip. However for A5 above chip device, there is no IV and KEY available to use, hence cannot do reverse engineering on kernelcache. But we can still get it by dumping from kernel memory. kernelcache is combined with lots of mach-o files, and IDA Pro 6.2 could identify each of them. After that, we need to do reverse engineering on IOKit. The steps of creating an IOKit constructor are similar: firstly, use `OSObject::new` to allocate memory, then initialize `IOService`, and at last init `OSMetaClass`.

However, debugging iOS kernel is much more difficult. A useful tool called `SerialDPProxy` can be used here to perform proxy between serial and UDP. And we need to make our own cable to implement the serial communication between USB and Dock connector. For A4 and older CPU devices, we can use redsn0w to `set boot-args -a "-v debug=0x09"`. However, for A5 CPU devices, there is no way to set boot-arg, hence we need a kernel exploit to cheat kernel with boot-arg & debug enable. More details could be seen from "***iOS5 An Exploitation Nightmare***" from *Stefan Esser*.

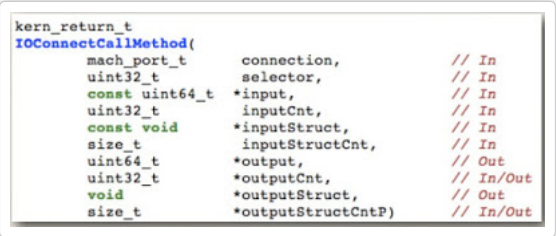
Fuzz

Passive Fuzz

Basically we can hook iOS IOConnectCallMethod, relying on MSHookFunction/MSHookMessage provided by MobileSubstrate Cydia for C/Object Method hook. TheOS/Tweak is also a good choice for hook framework, which is based on MobileSubstrate but more user-friendly. We can also using interpose(dyld function) to redirect the functions in the import table with no libmobilesubstrate required. Be reminded that struct object could be Data/Plist, and pay more work here; and scalar object are integer values only, make it random enough to find interesting stuffs. We expected to get some results like: NULL pointer deference, Kernel Use-after-free and handled panic exception errors.

Active Fuzz

Since passive fuzz only covers small amount of IOKit Interfaces, and needs user interaction on iPhone, it is not a efficient way for kernel exploitation. Most will turn to active fuzz, which covers most of IOKit interfaces, and could be done automatically and efficiently. The general steps are to find all IOKit drivers with IOUserClient, and identify all external methods of the driver, then test all those methods. The external methods are used by IOKit to provide function to user space application. Application call IOConnectCallMethod to control driver.



```
kern_return_t
IOConnectCallMethod(
        mach_port_t      connection,              // In
        uint32_t         selector,                // In
        const uint64_t  *input,                   // In
        uint32_t         inputCnt,                // In
        const void      *inputStruct,             // In
        size_t           inputStructCnt,          // In
        uint64_t        *output,                  // Out
        uint32_t        *outputCnt,               // In/Out
        void            *outputStruct,           // Out
        size_t          *outputStructCntP)       // In/Out
```

Fig. 2 IOConnectCallMethod function

We then can make use of it to dispatch the kernel, then overwrite the externalMethod.

In Summary

Two methods come up for fuzzing hook: one is to get IOExternelMethodDispatch sMethods[]; another is to get IOExternalMethod methodTemplate[]. Since the IOKit drivers are closed source, we need to reverse the KernelCache with symbol problem resolved, and list all the IOKit drivers interface. By using IDA Pro, we can have the interface names and address, but there are just bytes in the method dispatch table and IDA pro currently not handle it properly. Although crash could be easily generated by our fuzzer, it is a hard job to analyzing it, since there is no code or symbols for most IOKit drivers.

Reference

[1] Find Your Own iOS Kernel Bug
http://www.powerofcommunity.net/poc2012/hao.pdf
[2] iOS Kernel Vulnerability -fuzz & Mining Code audit, Pangu team on xKungFoo2015
[3] iOS 6 exploitation - 280 days later, Stefan Esser, CanSecWest 2013
http://www.slideshare.net/i0n1c/csw2013-stefan-esserios6exploitation280dayslater

Posted by nlog2n at 11:19 AM

Labels: iOS Security

Newer Post Home Older Post