Exploit MacOS Kernel Vulnerability to Escape Safari Sandbox

06月20, 2017

Backgrounds

文章目录

On Pwn2own 2017, macOS Sierra and Safari 10 from Apple were two of the platforms that have taken the highest number of hits. Though several other teams also successfully compromised or almost compromised the target macOS + Safari, 360Vulcan Team is the very one that exploited the least number of vulnerabilities. It is also the only team that realized sandbox escape through kernel exploit and obtained the system privilege to gain complete control over macOS kernel. In this article, we will share the exploitation techniques of how we successfully found and made use of the kernel vulnerability of macOS.

In order to hack into macOS Sierra + Safari and control the system kernel, 360Vulcan Team used two vulnerabilities: one is a remote code execution vulnerability of Safari (CVE-2017-2544) and the other one is a privilege escalation vulnerability of macOS (CVE-2017-2545). The latter exists in the components of macOS IOGraphic.

Seeing from the historical source codes on the Internet, this vulnerability can be dated back to 1992 and is transplanted from Joe Pasqua's codes. It means the vulnerability has existed in the macOS for more than 25 years and may possibly affect all the versions. In the meanwhile, the vulnerability can bypass sandbox and reach the kernel directly.

After we submitted our wining vulnerability on Pwn2own 2017 to Apple, they fixed it in the version macOS Sierra 10.12.5 released on May 15th.

Hunt down browser addressable kernel driver

Same as Windows system, the browser sandbox of Safari defines the kernel drivers that the processes in the sandbox can access, in order to mitigate the effects of kernel attack surface on sandbox escape. Therefore, our first step is to find the interface of kernel driver that is addressable in browser sandbox.

macOS constraints Safari's allowable operations according to two rules below:

```
/System/ Library/Sandbox/Profiles/system.sb
/System/Library/Frameworks/WebKit.framework/Versions/A/Resources/com.apple.WebProcess.sb
```

Then we analyzed various kernel drivers that are addressable by Safari. In the system.sb file, we found the flowing rule:

(allow iokit-open (iokit-registry-entry-class "IOFramebufferSharedUserClient"))

This rule indicates that Safari can open the driver interface ***IOFramebufferSharedUserClient*** which is provided to user mode by the components of IOGraphic. IOGraphic is the core underlying driver that is responsible for image processing in macOS. The source code package of IOGraphic 10.12.4 above versions can be found here:

https://opensource.apple.com/source/IOGraphics/IOGraphics-514.10/.

The codes are open source, so we did code audit on IOGraphic in our next step.

Attack surface

IOFramebufferSharedUserClient is derived from ***IOUserClient***. By matching with IOService named “IOFramebuff”, user mode can invoke function ***IOServiceOpen*** to access the port of object_**IOFramebufferSharedUserClient.**_

After accessing the port of an IOUserClient obeject, we can trigger the kernel execution of the following ports:

• implement port ***::externalMethod*** through the user mode API ***IOConnectCallMethod***
• implement port ***::clientMemoryForType*** through the user mode API ***IOConnectMapMemory***
• implement port ***::registerNotificationPort*** through the user mode API ***IOConnectSetNotificationPort***

In fact, ***IOFramebufferSharedUserClient*** provides very limited user mode ports. Function ***IOFramebufferSharedUserClient::getNotificationSemaphore*** aroused our attention. In ***IOKit.framework***, there is a function **_io_connect_get_notification_semaphore_** that hasn’t been exported. Through this API, we can trigger the kernel to execute the port ***::getNotificationSemaphore*** of the corresponding ***IOUserClient*** objects.

Vulnerability: getNotificationSemaphore UAF

We referenced the port code of ***IOFramebufferSharedUserClient::getNotificationSemaphore.***
The port is simple and the codes are as below:

IOReturn IOFramebufferSharedUserClient::getNotificationSemaphore(UInt32 interruptType, semaphore_t * semaphore) { return (owner->getNotificationSemaphore(interruptType, semaphore)); }

It shows ***IOFramebufferSharedUserClient::getNotificationSemaphore*** directly invokes the port ***getNotificationSemaphore*** that belongs to its owner (i.e. instance ***IOFramebuffer***).

The code of ***OFramebuffer::getNotificationSemaphore*** are as below:

```
IOReturn IOFramebuffer::getNotificationSemaphore(
    IOSelect interruptType, semaphore_t * semaphore )
{
    kern_return_t kr;
    semaphore_t sema;
    if (interruptType != kIOFBVBLInterruptType)
        return (kIOReturnUnsupported);
   if (!haveVBLService)
        return (kIOReturnNoResources);
    if (MACH_PORT_NULL == vblSemaphore) {
        kr = semaphore_create(kernel_task, &sema, SYNC_POLICY_FIFO, 0);
        if (kr == KERN_SUCCESS)
        vblSemaphore = sema;
    } else {
        kr = KERN_SUCCESS;
    }
    if (kr == KERN_SUCCESS)
        *semaphore = vblSemaphore;
    return (kr);
}
```

We can see from the above codes that ***vblSemaphore*** is a member of global objects and its initialized value is 0. After the first execution, the kernel will invoke function semaphore_create to create a semaphore, and then allocates it to ***vblSemaphore***. When the latter function executes, it will return ***vblSemaphore*** directly.

The question is, when the user mode invokes **_io_connect_get_notification_semaphore_** to get the semaphore, it can also destroy the semaphore simultaneously. But the ***vblSemaphore*** in the kernel still points to that semaphore object that has been destroyed.

If the user mode continues to invoke **_io_connect_get_notification_semaphore_** to get and use vblSemaphore, it will trigger off UAF (use-after-free).

Conclusion

IOUserClient framework provides many ports for user mode programs. However, due to historical reasons, ***IOFramebufferSharedUserClient*** still keeps a very rare port. Though no API is exported from the ***IOKit.framework*** in user mode, this port is still callable. We can transform the UAF issue of***IOFramebuffer::getNotificationSemaphore*** in the kernel to kernel memory leaks and arbitrary code execution, so as to realize successful sandbox escape and privilege escalation of the browser.

本文链接：https://blogs.360.net/post/pwn2own-using-macos-kernel-vuln-escape-from-safari-sandbox-en.html

-- EOF --