CVE-2019-0697：通过DHCP漏洞发现其余两个关键漏洞

Pinging / 2019-06-05 09:12:00 / 浏览数 4613 安全技术 漏洞分析 顶(0) 踩(0)

在前文中我们已经讨论过关于CVE-2019-0726的相关内容。我们知道，有时候在搜索某一个已知漏洞的时候便会偶然发现新的漏洞，有时这种漏洞还不止一个。

本文讨论了dhcpcore.dll的两个函数：在传递消息时使用的UpdateDomainSearchOption，，以及由第一个函数连续调用的DecodeDomainSearchListData。我们在

这正是这次发生的事情。在研究负责处理来自服务器的DHCP响应中的所有选项的DhcpExtractFullOptions函数时，特别是调用UpdateDomainSearchOption的选项

```
53  DHCPOptionPointers *dhcp_pointers_; // [esp+40h] [ebp-A44h]@1
54  DHCPOptions *dhcp_options_; // [esp+44h] [ebp-A40h]@1
55  int unknown_tags[256]; // [esp+48h] [ebp-A3Ch]@1
56  int all_tags[256]; // [esp+448h] [ebp-63Ch]@1
57  char v62; // [esp+848h] [ebp-23Ch]@71
58  char *v63; // [esp+868h] [ebp-21Ch]@71
59  int v64; // [esp+86Ch] [ebp-218h]@71
```

没有任何检查限制这些数组的迭代器值的迹象。 当时我们正在分析一个不同的漏洞，因此这些信息无关紧要。
因此，我们所能做的就是记住这部分代码以供日后使用。

分析

几周后，我们回想起之前引起我们注意的DhcpExtractFullOptions函数。
我们把它放在一个反汇编程序中，计算出那些未完全解析的代码片段，并试图找出这两个静态数组的用途。

当函数执行开始时，数组及其迭代器被清零：

```
111
112  dhcp_pointers_ = dhcp_pointers;
113  dhcp_options__ = dhcp_options;
114  dhcp_options_ = dhcp_options;
115  options_stream = options_stream;
116  memset(all_tags, 0, 0x400u);
117  memset(unknown_tags, 0, 0x400u);
118  all_tags_index = 0;
119  unknown_tag_index = 0;
120  is_android_metered = 0;
121  is_known_option_tag = 1;
```

该函数解析从DHCP服务器接收数据包中的所有选项，从中收集信息并对其进行处理。
此外，根据解析结果，它会在ETW（Windows事件跟踪）服务中记录相应的事件。 记录事件正是我们查看的缓冲区位置。
除了大量数据外，还会将这些数据传递给EtwEventWriteTransfer。
准备所有数据以进行日志记录需要大量工作。然而与我们正在讨论的漏洞无关，因此我们将跳过这些示例。

这里我们看看这些缓冲区是如何填充的。 填充是选项解析周期的一部分。 首先，为接收进行处理的当前选项调用具有自解释名称ParseDhcpv4Option的函数。
它使用接收的数据填充dhcp_pointers对象中的字段，或者如果遇到没有处理程序的选项标识符，则记下未知选项。

```
248  res_ = ParseDhcpv4Option(cur_pos_, option_size, 0, dhcp_pointers____, &is_known_option_tag);
249  if ( !res_ )
250  {
251    option_tag_ = *cur_pos_;
252    all_tags_index_ = (unsigned __int16)all_tags_index;
253    is_unknown_option_tag = is_known_option_tag == 0;
254    ++all_tags_index;
255    all_tags[all_tags_index_] = option_tag_;
256    if ( is_unknown_option_tag )
257    {
258      unknown_tag = *cur_pos_;
259      unknown_tag_index_ = (unsigned __int16)unknown_tag_index++;
260      unknown_tags[unknown_tag_index_] = unknown_tag;
261    }
262    if ( *cur_pos_ == DHCPTAG_VENDOR_INFO
```

从ParseDhcpv4Option返回后，当前选项option_tag的标识符值将写入all_tags数组的下一个元素，即我们正在查看的第一个数组。如果函数遇到未知选项，因此未设置is_

因此，all_tags数组存储来自接收消息选项的标记，而unknown_tags数组仅包含解析器未知的选项标记，除此之外，它根本没有检查数组的索引。因此，这些索引的值可

攻击过程

现在让我们试着在实践中测试我们的理论结论。 首先，选项标记的大小为一个字节，而数组元素的类型为int，这意味着元素大小为四个字节。
因此，我们有一个溢出，我们控制每个第四个字节，其余的在覆盖时归零。

```
000180017EA8 loc_180017EA8:                          ; CODE XREF: DhcpExtractFullOptions+1CD↑j
000180017EA8                                         ; DhcpExtractFullOptions+1E1↑j
000180017EA8   lea     rax, [rsp+0B50h+is_known_option_tag]
000180017EAD   mov     r9, rdi
000180017EB0   xor     r8d, r8d        ; r8 = 0
000180017EB3   mov     [rsp+0B50h+var_B30], rax
000180017EB8   mov     edx, r12d
000180017EBB   mov     rcx, r13
000180017EBE   call    ParseDhcpv4Option
000180017EC3   mov     esi, eax
000180017EC5   test    eax, eax
000180017EC7   jnz     loc_1800181FA
000180017ECD   movzx   eax, [rsp+0B50h+all_tags_index]
000180017ED2   lea     r8d, [rsi+1]    ; r8 = 1
000180017ED6   movzx   edx, byte ptr [r13+0] ; edx = option_tag
000180017EDB   mov     [rbp+rax*4+0A50h+all_tags], edx ; all_tags[all_tags_index] = option_tag
000180017EE2   add     ax, r8w
000180017EE6   mov     [rsp+0B50h+all_tags_index], ax ; all_tags_index++
000180017EEB   cmp     [rsp+0B50h+is_known_option_tag], esi
000180017EEF   jnz     short loc_180017F0B
000180017EF1   movzx   eax, [rsp+0B50h+unknown_tags_index]
000180017EF6   movzx   edx, byte ptr [r13+0]
000180017EFB   mov     [rbp+rax*4+0A50h+unknown_tags], edx ; unknown_tag[unknown_tags_index] = option_tag
000180017F02   add     ax, r8w
000180017F06   mov     [rsp+0B50h+unknown_tags_index], ax ; unknown_tags_index++
000180017F0B
```

测试漏洞的最简单方法是覆盖存储在堆栈中的函数的安全cookie，这将导致安全检查相关的异常。
让我们模拟DHCP服务器并发送足够数量的选项以导致覆盖的情况。 假设有0x1a0选项，标识符为0xaa，大小为零。
因此每个选项的大小是两个字节，包含所有标头的数据包的总大小将是1100-1200字节。
此值在以太网的MTU限制范围内，因此我们有理由相信该消息不会被分散执行，这将有助于我们避免任何复杂情况。

我们发送以这种方式形成的数据包以响应来自DHCP客户端的请求，并且在客户端的计算机上，我们在相应的svchost.exe进程中捕获异常：

```
0:015> k
 # Child-SP          RetAddr           Call Site
00 000000c3`658fcac8 00007ffc`438f6099 ntdll!NtWaitForMultipleObjects+0x14
01 000000c3`658fcad0 00007ffc`438f5f8e KERNELBASE!WaitForMultipleObjectsEx+0xf9
02 000000c3`658fcdd0 00007ffc`43e670bb KERNELBASE!WaitForMultipleObjects+0xe
03 000000c3`658fce10 00007ffc`43e66b6c kernel32!WerpReportFaultInternal+0x51b
04 000000c3`658fcf30 00007ffc`4399bebb kernel32!WerpReportFault+0xac
05 000000c3`658fcf70 00007ffc`3d892cba KERNELBASE!UnhandledExceptionFilter+0x35b
06 000000c3`658fd080 00007ffc`3d892e49 dhcpcore!_raise_securityfailure+0x1a
07 000000c3`658fd0b0 00007ffc`3d8a756b dhcpcore!_report_gsfailure+0x169
08 000000c3`658fd140 000000aa`000000aa dhcpcore!DhcpExtractFullOptions+0x77b
09 000000c3`658fdc60 000000aa`000000aa 0x000000aa`000000aa
0a 000000c3`658fdc68 000000aa`000000aa 0x000000aa`000000aa
```

正如我们从堆栈跟踪中看到的那样，来自我们数据包的选项标识符覆盖了堆栈cookie和函数的返回地址。

当然，创建类似可用的漏洞需要攻击者付出巨大努力。在系统上由于所有现代保护机制，缓冲区堆栈溢出是一个复杂且难以利用的漏洞。另一方面，我们不要忘记所有这些机
DhcpExtractFullOptions函数包含该范围内的几个潜在危险变量。

我们再次写信给微软，告知我们发现的错误。在一些通信之后，在分析了持续一周左右的请求后，我们得到了一个回复，说明正在准备此漏洞的CVE标识符，计划在3月发布

3月有一个修复故障的补丁，现在确定为CVE-2019-0697。 此前报告此漏洞的研究人员是Mitch
Adair，同样是微软的员工，他发现了1月份修复的DHCP漏洞CVE-2019-0547。

■■■■■■■■■■■■http://blog.ptsecurity.com/2019/05/how-analyzing-one-critical-dhcp.html(http://blog.ptsecurity.com/2019/05/how

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板