Bitdefender杀毒软件整数溢出漏洞分析 Part 1：漏洞本身

mss**** / 2018-06-24 09:41:43 / 浏览数 4964 技术文章 技术文章 顶(0) 踩(0)

原文：https://www.zerodayinitiative.com/blog/2018/6/19/analyzing-an-integer-overflow-in-bitdefender-av-part-1-the-vulnerability

在众多的软件缺陷中，安全软件中出现的漏洞被认为比其他漏洞尤为严重。这是因为，安全软件是帮助用户抵御攻击者的靠山，所以，如果这些软件中出现安全问题，不仅会
Internet
Security中的安全漏洞的原因之一，如果该漏洞被利用，这款安全防护软件就会被攻击者用于远程执行代码。实际上，这个漏洞是由整数溢出引起的，尽管现在已经修复，但

当前，尽管系统已经提供了DEP和ASLR等防御机制，但是，人们仍然可以对现代软件中的整数溢出进行有效的利用。在本文中，我们将对这个运行攻击者在目标系统上面远

Bitdefender概述

Bitdefender提供了多种具有防病毒（AV）功能的产品，并且所有这些AV产品都安装了SYSTEM服务vsserv.exe。该服务的作用是处理AV扫描请求。就像各种AV引擎一样，
Files\Bitdefender\Bitdefender Threat
Scanner\Antivirus_XXXX\Plugins\目录中。除此之外，该引擎还有一个鲜为人知的功能，即模拟遇到的可执行代码，以此检测未知或经过混淆处理的病毒。

遇到x86可移植可执行文件（PE）时，Bitdefender实际上是通过保存在cevakrnl.xmd和ceva_emu.cvd文件中高度复杂的代码仿真器来虚拟执行PE的。这些仿真器会创建一
API的实现，并为频繁执行的地址创建相应的JIT（Just-In-Time，JIT）代码。

杀毒原理简介

当仿真器检测到函数调用时，就会调用cevakrnl.xmd!sub_366F4D0()。该函数处理被仿真函数中的第一条指令，实际上就是在一个常量表中搜索匹配项。在处理后面的指令

```
cevakrnl.xmd:0366F4D0 sub_366F4D0     proc near               ; CODE XREF: sub_363F400+Fp
cevakrnl.xmd:0366F4D0
...
cevakrnl.xmd:0366F530                 mov     edi, [edx+esi*4] ; offset
cevakrnl.xmd:0366F533                 mov     bl, [ebx+eax]   ; first byte in code
cevakrnl.xmd:0366F536                 add     edi, ecx
cevakrnl.xmd:0366F538                 inc     eax
cevakrnl.xmd:0366F539                 mov     [ebp+var_218], eax
cevakrnl.xmd:0366F53F                 movzx   eax, byte ptr [edi]
cevakrnl.xmd:0366F542                 and     eax, 0FFFFFF7Fh
cevakrnl.xmd:0366F547                 dec     eax             ; interpreter state
cevakrnl.xmd:0366F548                 cmp     eax, 3          ; switch 4 cases
cevakrnl.xmd:0366F54B                 ja      loc_366F5E2     ; jumptable 0366F551 default case
cevakrnl.xmd:0366F551                 jmp     ds:off_366F7F4[eax*4] ; switch jump
...
cevakrnl.xmd:0366F58F                 movzx   edx, byte ptr [edi+1] ; search distance
cevakrnl.xmd:0366F593                 xor     esi, esi
cevakrnl.xmd:0366F595                 test    edx, edx
cevakrnl.xmd:0366F597                 jz      @out
cevakrnl.xmd:0366F59D                 lea     ecx, [ecx+0]
cevakrnl.xmd:0366F5A0
cevakrnl.xmd:0366F5A0 @locatelopcode:                         ; CODE XREF: sub_366F4D0+EBj
cevakrnl.xmd:0366F5A0                 lea     eax, [edx+esi]
cevakrnl.xmd:0366F5A3                 shr     eax, 1
cevakrnl.xmd:0366F5A5                 lea     ecx, [edi+eax*2] ; edi + eax*3 + 2 = opcode
cevakrnl.xmd:0366F5A8                 mov     cl, [eax+ecx+2]
cevakrnl.xmd:0366F5AC                 cmp     bl, cl
cevakrnl.xmd:0366F5AE                 jnb     short loc_366F5B4
cevakrnl.xmd:0366F5B0                 mov     edx, eax
cevakrnl.xmd:0366F5B2                 jmp     short loc_366F5B9
cevakrnl.xmd:0366F5B4 ;
---------------------------------------------------------------------------
cevakrnl.xmd:0366F5B4
cevakrnl.xmd:0366F5B4 loc_366F5B4:                            ; CODE XREF: sub_366F4D0+DEj
cevakrnl.xmd:0366F5B4                 jbe     short @opcodematchfound
cevakrnl.xmd:0366F5B6                 lea     esi, [eax+1]
cevakrnl.xmd:0366F5B9
cevakrnl.xmd:0366F5B9 loc_366F5B9:                            ; CODE XREF: sub_366F4D0+E2j
cevakrnl.xmd:0366F5B9                 cmp     esi, edx
cevakrnl.xmd:0366F5BB                 jb      short @locatelopcode
cevakrnl.xmd:0366F5BD                 jmp     @out
cevakrnl.xmd:0366F5C2 ;
---------------------------------------------------------------------------
```

当函数的前16个字节被仿真器找到相应的匹配项时，或者当指令序列是未知的指令序列时，初始搜索就宣告结束。

常量表中最后的匹配项会提供函数起始位置之前一些已知字节的内容，以及函数起始位置后16字节的内容。

```
[X bytes before]
FUNCTIONSTART:
16 bytes already matched
```

```
[Y bytes after]
```

然后，该引擎会读取并检查这些字节的内容，以确定是否与已知的AV签名相匹配。如果匹配，则调用检测到的代码签名对应的处理程序做进一步处理。

漏洞分析

遇到Themida加壳软件代码序列时，该杀毒软件就会调用sub_36906D0()函数以便对匹配的序列进行代码解释。

```
cevakrnl.xmd:0368BAA0                   push    0A69h
cevakrnl.xmd:0368BAA5                   push    97Fh
cevakrnl.xmd:0368BAAA                   push    16Fh
cevakrnl.xmd:0368BAAF                   push    931h
cevakrnl.xmd:0368BAB4                   push    ebx
cevakrnl.xmd:0368BAB5                   push    edi
cevakrnl.xmd:0368BAB6                   call    sub_36906D0
```

被解释的代码中，唯一的变量为ebp偏移量"X"，并且ebx的值就是通过它读取的。该函数从代码流中提取偏移量，并使用它从仿真堆栈中获取ebx的值。

```
cevakrnl.xmd:036906D0 sub_36906D0       proc near
...
cevakrnl.xmd:036906D0                   push    ebp
cevakrnl.xmd:036906D1                   mov     ebp, esp
cevakrnl.xmd:036906D3                   sub     esp, 84h
cevakrnl.xmd:036906D9                   mov     eax, ds:dword_369A5C4
cevakrnl.xmd:036906DE                   xor     eax, ebp
cevakrnl.xmd:036906E0                   mov     [ebp+var_4], eax
cevakrnl.xmd:036906E3                   push    ebx
cevakrnl.xmd:036906E4                   mov     ebx, [ebp+arg_4]
cevakrnl.xmd:036906E7                   push    edi
cevakrnl.xmd:036906E8                   mov     edi, [ebp+arg_0]
cevakrnl.xmd:036906EB                   push    0               ; int
cevakrnl.xmd:036906ED                   mov     eax, [ebx+88h]  ; virtual function address
cevakrnl.xmd:036906F3                   sub     eax, 0Fh
cevakrnl.xmd:036906F6                   mov     [ebp+var_60], ebx
cevakrnl.xmd:036906F9                   push    eax             ; addr
cevakrnl.xmd:036906FA                   lea     eax, [ebp+put]  ; mov ebx, [ebp+0xdeadbeef]
cevakrnl.xmd:036906FD                   push    eax             ; output
cevakrnl.xmd:036906FE                   push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:03690701                   call    get_virtual_dword ; read ebp offset X
cevakrnl.xmd:03690706                   add     esp, 10h
cevakrnl.xmd:03690709                   test    eax, eax
cevakrnl.xmd:0369070B                   jnz     loc_3690C64
cevakrnl.xmd:03690711                   mov     eax, [ebx+28h]  ; 0x12ffc0 (ebp)
cevakrnl.xmd:03690711                                           ; put has to be dataaddr-0x12ffc0
cevakrnl.xmd:03690711                                           ; dataaddr holds first dword after
standard code
cevakrnl.xmd:03690711                                           ; which can be controlled
cevakrnl.xmd:03690711                                           ; functionstart+0x35C = dataaddr
cevakrnl.xmd:03690714                   add     eax, [ebp+put]  ; X => [ebp+X]
cevakrnl.xmd:03690717                   push    0               ; int
cevakrnl.xmd:03690719                   push    eax             ; addr
cevakrnl.xmd:0369071A                   lea     eax, [ebp+readebx] ; ebx
cevakrnl.xmd:0369071D                   push    eax             ; output
cevakrnl.xmd:0369071E                   push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:03690721                   call    get_virtual_dword ; read ebx from [ebp+X]
```

接下来，会解释mov ecx, [ebx]指令，ecx的值是从保存在ebx中的模拟地址中提取的。

```
cevakrnl.xmd:036906D0 sub_36906D0     proc near
...
cevakrnl.xmd:036906D0                 push    ebp
cevakrnl.xmd:036906D1                 mov     ebp, esp
cevakrnl.xmd:036906D3                 sub     esp, 84h
cevakrnl.xmd:036906D9                 mov     eax, ds:dword_369A5C4
cevakrnl.xmd:036906DE                 xor     eax, ebp
cevakrnl.xmd:036906E0                 mov     [ebp+var_4], eax
cevakrnl.xmd:036906E3                 push    ebx
cevakrnl.xmd:036906E4                 mov     ebx, [ebp+arg_4]
cevakrnl.xmd:036906E7                 push    edi
cevakrnl.xmd:036906E8                 mov     edi, [ebp+arg_0]
cevakrnl.xmd:036906EB                 push    0               ; int
cevakrnl.xmd:036906ED                 mov     eax, [ebx+88h]  ; virtual function address
cevakrnl.xmd:036906F3                 sub     eax, 0Fh
cevakrnl.xmd:036906F6                 mov     [ebp+var_60], ebx
cevakrnl.xmd:036906F9                 push    eax             ; addr
cevakrnl.xmd:036906FA                 lea     eax, [ebp+put]  ; mov ebx, [ebp+0xdeadbeef]
cevakrnl.xmd:036906FD                 push    eax             ; output
cevakrnl.xmd:036906FE                 push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:03690701                 call    get_virtual_dword ; read ebp offset X
cevakrnl.xmd:03690706                 add     esp, 10h
cevakrnl.xmd:03690709                 test    eax, eax
cevakrnl.xmd:0369070B                 jnz     loc_3690C64
cevakrnl.xmd:03690711                 mov     eax, [ebx+28h]  ; 0x12ffc0 (ebp)
cevakrnl.xmd:03690711                                         ; put has to be dataaddr-0x12ffc0
cevakrnl.xmd:03690711                                         ; dataaddr holds first dword after
standard code
cevakrnl.xmd:03690711                                         ; which can be controlled
cevakrnl.xmd:03690711                                         ; functionstart+0x35C = dataaddr
cevakrnl.xmd:03690714                 add     eax, [ebp+put]  ; X => [ebp+X]
cevakrnl.xmd:03690717                 push    0               ; int
cevakrnl.xmd:03690719                 push    eax             ; addr
cevakrnl.xmd:0369071A                 lea     eax, [ebp+readebx] ; ebx
cevakrnl.xmd:0369071D                 push    eax             ; output
cevakrnl.xmd:0369071E                 push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:03690721                 call    get_virtual_dword ; read ebx from [ebp+X]
```

接下来解释下面的代码序列：

```
cevakrnl.xmd:03690806                 push    [ebp+readebx]   ; addr
cevakrnl.xmd:03690809                 mov     [ebp+var_68], eax
cevakrnl.xmd:0369080C                 lea     eax, [ebp+readecx]
cevakrnl.xmd:0369080F                 push    eax             ; output
cevakrnl.xmd:03690810                 push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:03690813                 call    get_virtual_dword
```

这里，会根据ecx + dword[ecx + 0x78 + word[ecx+0x3c]]计算出仿真esi的值。

```
.data:000000A5                 push    eax
.data:000000A6                 push    ecx
.data:000000A7                 pusha
.data:000000A8                 xor     eax, eax
.data:000000AA                 mov     [ebp+0], eax
.data:000000B0                 mov     esi, 3Ch
.data:000000B5 BYTESAFTER:
.data:000000B5                 add     esi, [esp+20h] ; saved ecx offset 0x3C
.data:000000B9                 lodsw
.data:000000BB                 add     eax, [esp+20h] ; saved ecx
.data:000000BF                 mov     esi, [eax+78h] ; dword [ecx + word [ecx+0x3C]+0x78]
.data:000000C2                 add     esi, [esp+20h] ; ecx + dword [ecx + word [ecx+0x3C]+0x78]
```

接下来从仿真的esi中总共提取0x28字节，并读取偏移0x18处的dword（对应仿真代码的mov edi，[esi + 18h]指令）。

```
cevakrnl.xmd:0369082E                   add     eax, 3Ch
cevakrnl.xmd:03690831                   mov     [ebp+var_30], 0
cevakrnl.xmd:03690838                   push    0
cevakrnl.xmd:0369083A                   push    eax             ; ecx+0x3c
cevakrnl.xmd:0369083B                   lea     eax, [ebp+pout]
cevakrnl.xmd:0369083E                   push    eax
cevakrnl.xmd:0369083F                   push    dword ptr [edi+0Ch]
cevakrnl.xmd:03690842                   call    get_virtual_word ; word ptr [ecx+0x3c]
cevakrnl.xmd:03690847                   add     esp, 10h
cevakrnl.xmd:0369084A                   test    eax, eax
cevakrnl.xmd:0369084C                   jnz     loc_3690C63
cevakrnl.xmd:03690852                   mov     eax, [ebp+readecx]
cevakrnl.xmd:03690855                   movzx   ecx, [ebp+pout] ; word[ecx+0x3c]
cevakrnl.xmd:03690859                   add     eax, 78h
cevakrnl.xmd:0369085C                   add     eax, ecx        ; ecx + 0x78 + word[ecx+0x3c]
cevakrnl.xmd:0369085E                   push    0               ; int
cevakrnl.xmd:03690860                   push    eax             ; addr
cevakrnl.xmd:03690861                   lea     eax, [ebp+output]
cevakrnl.xmd:03690864                   push    eax             ; output
cevakrnl.xmd:03690865                   push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:03690868                   call    get_virtual_dword
cevakrnl.xmd:0369086D                   add     esp, 10h
cevakrnl.xmd:03690870                   test    eax, eax
cevakrnl.xmd:03690872                   jnz     loc_3690C63
cevakrnl.xmd:03690878                   mov     edx, [ebp+output] ; dword[ecx + 0x78 + word[ecx+0x3c]]
cevakrnl.xmd:0369087B                   lea     esi, [ebp+var_2C]
cevakrnl.xmd:0369087E                   mov     ecx, [edi+0Ch]
cevakrnl.xmd:03690881                   add     edx, [ebp+readecx] ; ecx + dword[ecx + 0x78 +
word[ecx+0x3c]] =
cevakrnl.xmd:03690881                                           ; emulated esi
```

对dword（N）乘以4的时候，没有进行整数边界检查，然后调用了malloc()。这样的话，将导致分配的缓冲区过小。

```
cevakrnl.xmd:03690884                   push    esi
cevakrnl.xmd:03690885                   push    0
cevakrnl.xmd:03690887                   push    28h
cevakrnl.xmd:03690889                   mov     [ebp+output], edx
cevakrnl.xmd:0369088C                   mov     eax, [ecx]
cevakrnl.xmd:0369088E                   push    0
cevakrnl.xmd:03690890                   push    edx
cevakrnl.xmd:03690891                   call    dword ptr [eax+0C0h]
cevakrnl.xmd:03690897                   test    eax, eax
cevakrnl.xmd:03690899                   jnz     loc_3690C63
cevakrnl.xmd:0369089F                   mov     eax, dword ptr [ebp+structure+18h] ; at offset 0x18
```

接下来进入一个循环，在该循环中，将会把通过位于仿真堆栈中的偏移量找到的N个字符串的CRC32校验和填充到这个缓冲区中。当字符串的偏移量过大的时候，循环将会中

```
cevakrnl.xmd:036908A6                   shl     eax, 2          ; integer overflow
cevakrnl.xmd:036908A6                                           ; 0x40 00 00 00
cevakrnl.xmd:036908A9                   push    eax
cevakrnl.xmd:036908AA                   call    near ptr 15D08D0h ; malloc
cevakrnl.xmd:036908AF                   mov     ebx, eax
cevakrnl.xmd:036908B1                   add     esp, 4
cevakrnl.xmd:036908B4                   mov     [ebp+buffer], ebx
```

这提供了执行代码时所需的一切。首先，我们需要能够用自己的内容覆盖任意数量的字节。这就需要用到一些其CRC值符合特定要求的字符串。当然，这里的CRC32算法是

```
cevakrnl.xmd:036908C6 @fillbuffer:                            ; CODE XREF: sub_36906D0+245j
cevakrnl.xmd:036908C6                 mov     eax, dword ptr [ebp+structure+20h] ; at offset 0x18 +
8
cevakrnl.xmd:036908C6                                         ; (8 bytes after large num)
cevakrnl.xmd:036908C6                                         ; offset to a table of 32-bit offsets
cevakrnl.xmd:036908C6                                         ; to data
cevakrnl.xmd:036908C9                 push    0               ; int
cevakrnl.xmd:036908CB                 lea     eax, [eax+esi*4]
cevakrnl.xmd:036908CE                 add     eax, [ebp+readecx]
cevakrnl.xmd:036908D1                 push    eax             ; addr
cevakrnl.xmd:036908D2                 lea     eax, [ebp+dataoffset]
cevakrnl.xmd:036908D5                 push    eax             ; output
cevakrnl.xmd:036908D6                 push    dword ptr [edi+0Ch] ; int
cevakrnl.xmd:036908D9                 call    get_virtual_dword
cevakrnl.xmd:036908DE                 mov     ebx, eax
cevakrnl.xmd:036908E0                 add     esp, 10h
cevakrnl.xmd:036908E3                 test    ebx, ebx
cevakrnl.xmd:036908E5                 jnz     @error          ; in case of error 32-bit zero is
written
cevakrnl.xmd:036908EB                 mov     eax, [ebp+buffer]
cevakrnl.xmd:036908EE                 lea     eax, [eax+esi*4]
cevakrnl.xmd:036908F1                 push    eax
cevakrnl.xmd:036908F2                 mov     eax, [ebp+dataoffset]
cevakrnl.xmd:036908F5                 add     eax, [ebp+readecx]
cevakrnl.xmd:036908F8                 push    0EDB88320h      ; xorval
cevakrnl.xmd:036908FD                 push    eax             ; srcvirt
cevakrnl.xmd:036908FE                 push    edi
cevakrnl.xmd:036908FF                 call    crc32b          ; unsigned int crc32b(unsigned char
*message) {
cevakrnl.xmd:036908FF                                         ;    int i, j;
cevakrnl.xmd:036908FF                                         ;    unsigned int byte, crc, mask;
cevakrnl.xmd:036908FF                                         ;
cevakrnl.xmd:036908FF                                         ;    i = 0;
cevakrnl.xmd:036908FF                                         ;    crc = 0xFFFFFFFF;
cevakrnl.xmd:036908FF                                         ;    while (message[i] != 0) {
cevakrnl.xmd:036908FF                                         ;       byte = message[i];
// Get next byte.
cevakrnl.xmd:036908FF                                         ;       crc = crc ^ byte;
cevakrnl.xmd:036908FF                                         ;       for (j = 7; j >= 0; j--) {
// Do eight times.
cevakrnl.xmd:036908FF                                         ;          mask = -(crc & 1);
cevakrnl.xmd:036908FF                                         ;          crc = (crc >> 1) ^
(0xEDB88320 & mask);
cevakrnl.xmd:036908FF                                         ;       }
cevakrnl.xmd:036908FF                                         ;       i = i + 1;
cevakrnl.xmd:036908FF                                         ;    }
cevakrnl.xmd:036908FF                                         ;    return ~crc;
cevakrnl.xmd:036908FF                                         ; }
cevakrnl.xmd:03690904                 mov     ebx, eax
cevakrnl.xmd:03690906                 add     esp, 10h
cevakrnl.xmd:03690909                 test    ebx, ebx
cevakrnl.xmd:0369090B                 jnz     @error
cevakrnl.xmd:03690911                 inc     esi
cevakrnl.xmd:03690912                 cmp     esi, dword ptr [ebp+structure+18h]
cevakrnl.xmd:03690915                 jb      short @fillbuffer
```

当匹配的函数被调用时，漏洞被触发，即利用任意内容覆盖特定数量的内存空间。

小结

通过深入分析，我们可以看到，攻击者可以含有整数溢出漏洞的Bitdefender版本上执行任意代码。目前，软件供应商已经通过更新修复了该漏洞（以及其他漏洞）。因此，

上一篇：Microsoft Windows... 下一篇：在Windows下利用格式字符串

1. 0 条回复
   - 动动手指，沙发就是你的了！

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板