绕过多杀软的公式编辑器漏洞变种样本惊现

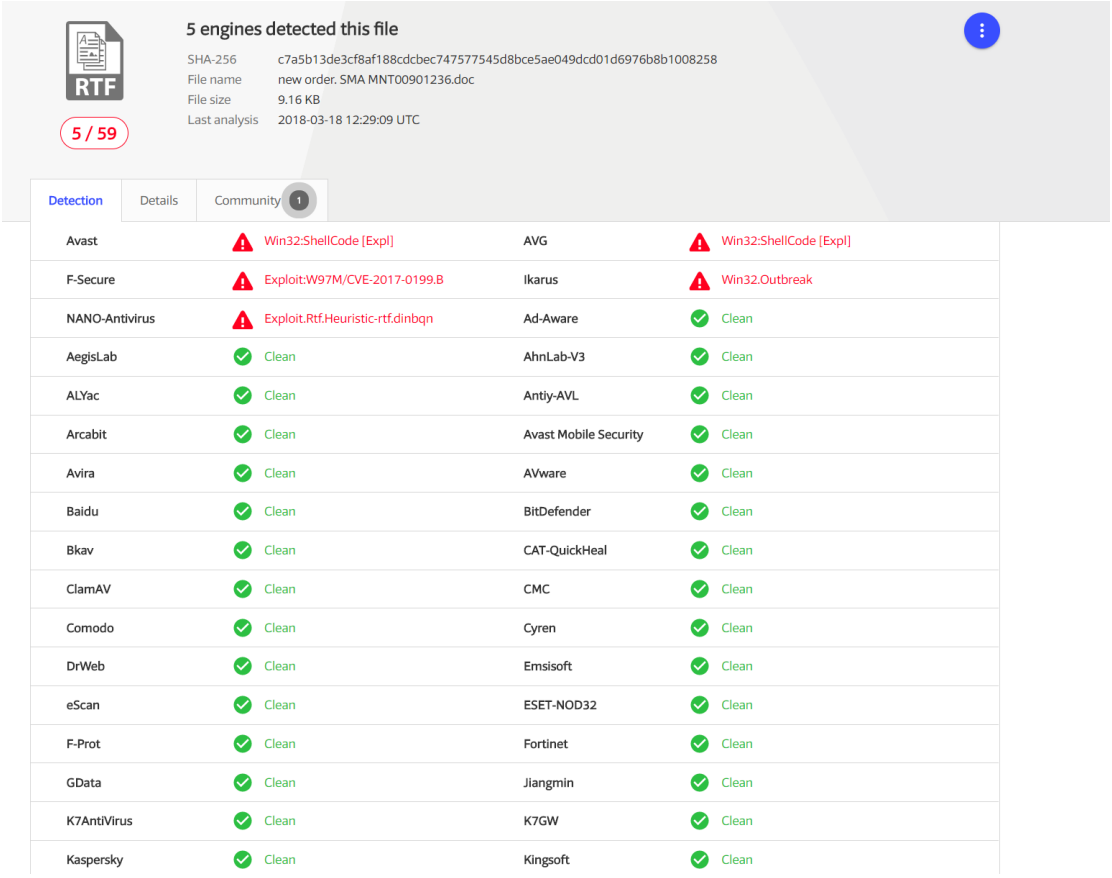
恐将导致新一波 Office 漏洞攻击

概述

此前，微软发布的月度安全更新中修复了多个最新的 Office 漏洞(CVE-2017-11882，CVE-2018-0798 和 CVE-2018-0802)。该漏洞为 Office 公式编辑器的最新漏洞，微软对此的处理方式是将该模块删掉。（参见金睛安全研究团队对 CVE-2018-0798 漏洞的分析报告：http://venustech.com.cn/NewsInfo/4/49366.Html；金睛安全研究团队对 CVE-2017-11882 漏洞的分析报告：http://venustech.com.cn/NewsInfo/4/48671.Html）。

然而即使模块在最新版本中被删掉，仍然有大量攻击者在利用 CVE-2017-11882+CVE-2018-0802 的漏洞文档进行攻击。

近日，金睛安全研究团队捕获到一些奇怪的 RTF 样本。这批样本相较于 CVE-2018-0802 传统利用方式而言，采用了一种新的利用方式，能够绕过市面上大多数杀软，并且即使部分杀软能查杀，也无法正确识别漏洞。

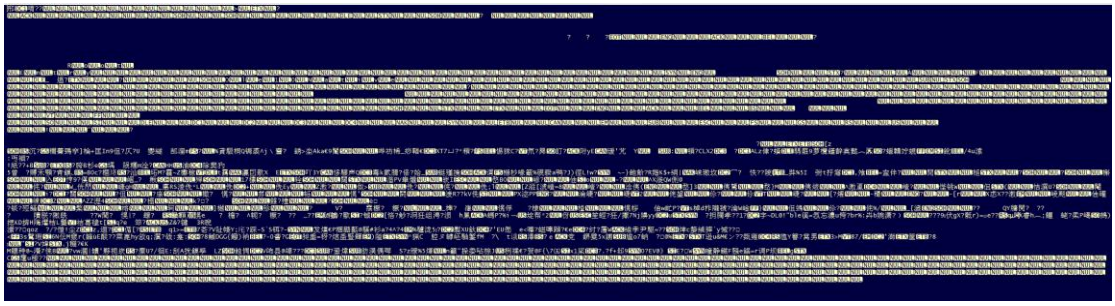


技术细节

经过分析，我们发现这批样本有一个通用特征：都内嵌了一个 CFBF(Compound File Binary Format)对象，将该文件提取后发现只有一个"\x010le10Native"流。

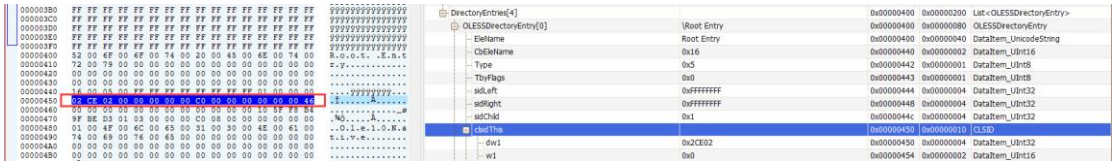
OLESSDirectoryEntry[0]	\Root Entry	0x00000400	0x00000080	OLESSDirectoryEntry
OLESSDirectoryEntry[1]	\Root Entry\□□le10Native	0x00000480	0x00000080	OLESSDirectoryEntry
OLESSDirectoryEntry[2]	\	0x00000500	0x00000080	OLESSDirectoryEntry
OLESSDirectoryEntry[3]	\	0x00000580	0x00000080	OLESSDirectoryEntry

在文本编辑器中浏览该文件也未发现明文的 URL，文件路径或是命令等信息。

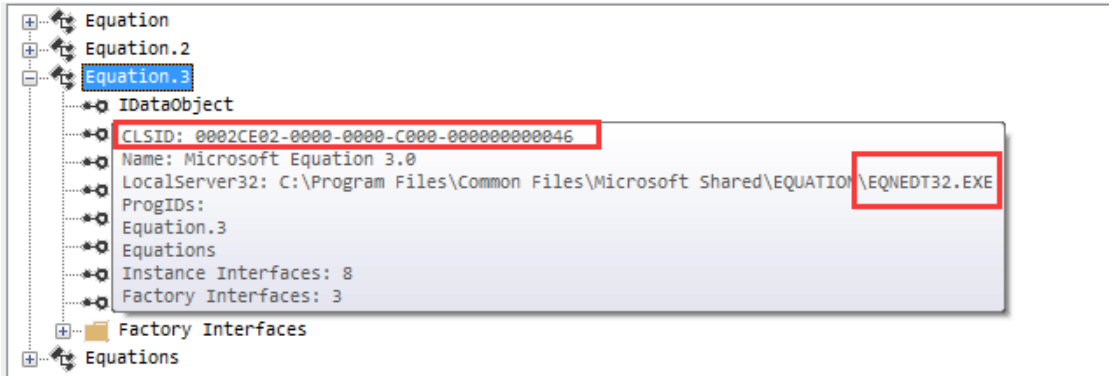


经过分析，我们发现这批样本是公式编辑器系列漏洞的新利用方式。可以用于构造和 CVE-2017-11882，CVE-2018-0802 相同触发机制，但静态特征更少的恶意样本。

首先可以观察到，这个 CFBF 对象的 Root Entry 带有一个 CLSID {0002CE02-0000-0000-C000-000000000046}。



而这个 CLSID 对应的对象是 Microsoft Equation 3.0，即该对象是一个公式 3.0 编辑器对象。



而在[MS-OLEDS]中可以查到"\x010le10Native"流的结构。前四个字节表示数据长度，后面的字节流均为对应的数据。但是可以发现，这段数据的含义并不显然。也许是 shellcode，但是 shellcode 的头部并不在偏移量为 0 的位置。

我们来回顾一下之前在 CVE-2017-11882 中，公式数据被加载和读取的过程。

通过 Unicode 字符串"Equation Native"可以定位到数据流加载的过程。如下图所示。

```
26  v11 = (*(int (__stdcall **)(int, wchar_t *, _DWORD, signed int, _DWORD, int *))(*(_DWORD *)a2 + 16))(
27          a2,
28          s_EquationNative,
29          0,
30          16,
31          0,
32          &v10);
33  if ( v11 )
34  {
35    v11 = (*(int (__stdcall **)(int, void *, _DWORD, signed int, _DWORD, int *))(*(_DWORD *)a2 + 16))(
36            a2,
37            &s_Ole10Native,
38            0,
39            16,
40            0,
41            &v10);
```

注意到这里可能是对虚函数表的调用，28 行的 s_EquationNative 是流名，则 v10 这个唯一的 out 变量应该对应于 IStream，那么 a2 应该对应于 IStorage。

进行标注后得到以下结果。

```
26  v11 = root->lpVtbl->OpenStream(root, s_EquationNative, 0, STGM_SHARE_EXCLUSIVE, 0, &v10);
27  if ( v11 )
28  {
29    v11 = root->lpVtbl->OpenStream(root, (const OLECHAR *)&s_Ole10Native, 0, STGM_SHARE_EXCLUSIVE, 0, &v10);
30    flag = 1;                                   // flag -> "\x01Ole10Native" stream
31    *(_DWORD *)(*(_DWORD *)(v8 + 4) + 210) = 1;
32  }
33  if ( !v11 )
34  {
35    v11 = flag ? v10->lpVtbl->Read(v10, &v9, 4, (ULONG *)&size) : v10->lpVtbl->Read(v10, &v13, 0x1C, (ULONG *)&size);
36    if ( !v11 )
37    {
38      if ( flag || (unsigned __int16)v13 == size && *(unsigned int *)((char *)&v13 + 2) <= 0x20000 )
39      {
40        if ( flag )
41          v2 = GlobalAlloc(2u, v9);
42        else
43          v2 = GlobalAlloc(2u, v14);
44        hMem = v2;
45        if ( v2 )
46        {
47          v7 = GlobalLock(hMem);
48          if ( v7 )
49          {
50            if ( flag )
51              v3 = v10->lpVtbl->Read(v10, v7, v9, (ULONG *)&size);
52            else
53              v3 = v10->lpVtbl->Read(v10, v7, v14, (ULONG *)&size);//
54                                                  // v13 -> [esp + 0x34]
55                                                  // v14 -> [esp + 0x3C] = [esp + 0x34 + 0x8]
56            v11 = v3;
57            if ( v3 )
58            {
59              v11 = E_FAIL;
60            }
61            else
62            {
63              GlobalUnlock(hMem);
64              v7 = 0;
65              sub_403A20(v8);
66              sub_42F8FF((char *)hMem);         // data processed here
67              sub_4032D1(v8, 0);
```

我们来梳理一下图中这一段的逻辑：

在第 26~29 行，公式编辑器首先尝试从 CFBF 对象中读取名为"Equation Native"的流。如果失败(v11 不为 0)，则尝试读取名为"\x01Ole10Native"的流。

在第 35 行，公式编辑器根据流的类型，从流中读取不同大小的数据到一个变量中。如果流的类型是"Equation Native"，则读取 0x1C 个字节；如果流的类型是"\x01Ole10Native"，则读取 4 个字节。根据之前对于 CVE-2017-11882 等漏洞的分析，可以得知这里是在处理公式对象的 OLE 头。

在第 38 行，判断了数据的头部是否正确。如果流的类型是"\x01Ole10Native"，则不作任何检查；如果流的类型是"Equation Native"，则判断 OLE 头的前两个字节是不是 0x001C(小端序)，及随后的四个字节是不是 0x00020000(小端序)。

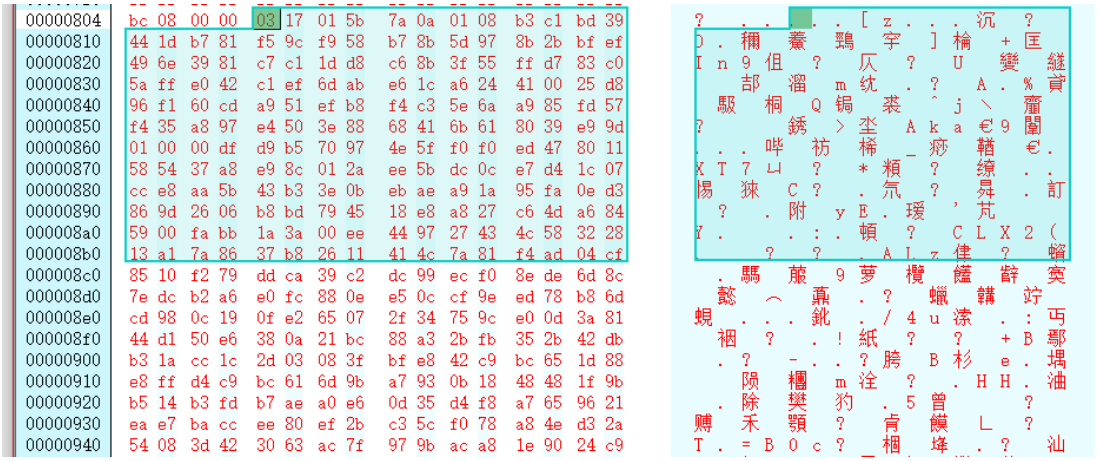
在第 40~43 行，根据流的类型为接下来的公式数据分配内存空间。如果流的类型是"\x01Ole10Native"，则流的前 4 个字节([v9+0])就是数据大小；如果流的类型是"Equation Native"，则流的第 8 个字节处的 DWORD 就是数据大小，而图中的 v14([esp+0x3C])正好是 v13([esp+0x34])偏移量为 8 的位置。

在第 50~53 行，根据流的类型来读入公式数据。对两种流的处理实际是相同的。而在

第 66 行的函数调用中，程序对读入的公式数据开始进行处理。

也就是说，两种不同的流，对应的数据，区别仅仅是数据头："Equation Native"流的数据头是 0x1C 个字节，而"\x01Ole10Native"流的数据头仅仅是 4 个字节。

我们继续回到样本数据处进行查看。



前面的 5 个字节是公式头数据。按理说应该是如下格式。

MTEF header (version 2 and later):

The version 2 header consists of a 5-byte header:

byte	description	value
0	MTEF version	3
1	generating platform	0 for Macintosh, 1 for Windows
2	generating product	0 for MathType, 1 for Equation Editor
3	product version	3
4	product subversion	0

但是实际上，上图中的第 1、3、4 三个字段是被公式编辑器的处理函数忽略掉了的。

```
signed int __cdecl sub_43A988(int a1, RECORD *a2)
{
  unsigned __int8 v2; // al
  unsigned __int8 v3; // al

  g_context = a1;
  *(a1 + 12) = 0;
  v2 = ReadByte();                              // This also increases the index.
                                                //
                                                // index = 1
  MTEFVersion = v2;
  if ( v2 >= 100 )
    MTEFVersion -= 100;
  if ( MTEFVersion == 1 )
  {
    error(45, 145);
  }
  else
  {
    if ( MTEFVersion < 2 || MTEFVersion > 3 )
    {
      error(43, 145);
      return 0;
    }
    ++g_context->index;                         // index = 2, skip "generating platform" field
    if ( ReadByte() != 1 )                      // index = 3, read "generating product" field
      error(45, 145);
    ++g_context->index;                         // index = 4, skip "product version" field
    ++g_context->index;                         // index = 5, skip "product subversion" field
  }
  v3 = ReadByte();
  ProcessSizeRecord(v3, &logical_size, &delta_size);
  dword_45B24C = sub_438DB8() || a2->logical_size == logical_size && a2->delta_size == delta_size;
  return 1;
}
```

我们可以注意到，该样本中的前五个字节中，在第 1、3、4 个字段有意地使用了错误的数值，使得静态特征更难被提取。

接下来的"0a"，"01"数据分别对应初始 SIZE 记录和 LINE 记录，然后"08 b3 c1"对应的是 FONT 表记录，即 CVE-2017-11882/CVE-2018-0802 漏洞。

随后的部分就是 shellcode 了。这里可以确定是 CVE-2018-0802 漏洞。

```
seg000:00000000                         ; Segment type: Pure code
seg000:00000000                         seg000          segment byte public 'CODE' use32
seg000:00000000                                         assume cs:seg000
seg000:00000000                                         assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000 BC 08 00 00                             dd 8BCh
seg000:00000004 03                                      db    3
seg000:00000005 17                                      db 17h
seg000:00000006 01                                      db 1
seg000:00000007 5B                                      db  5Bh ; [
seg000:00000008 7A                                      db  7Ah ; z
seg000:00000009 0A                                      db 0Ah
seg000:0000000A 01                                      db    1
seg000:0000000B 08                                      db 8
seg000:0000000C B3                                      db 0B3h
seg000:0000000D C1                                      db 0C1h
seg000:0000000E                         ; ---------------------------------------------------------------------------
seg000:0000000E BD 39 44 1D B7                          mov     ebp, 0B71D4439h
seg000:00000013 81 F5 9C F9 58 B7                       xor     ebp, 0B758F99Ch
seg000:00000019 8B 5D 97                                mov     ebx, [ebp-69h]
seg000:0000001C 8B 2B                                   mov     ebp, [ebx]
seg000:0000001E BF EF 49 6E 39                          mov     edi, 396E49EFh
seg000:00000023 81 C7 C1 1D D8 C6                       add     edi, 0C6D81DC1h
seg000:00000029 8B 3F                                   mov     edi, [edi]
seg000:0000002B 55                                      push    ebp
seg000:0000002C FF D7                                   call    edi
seg000:0000002E 83 C0 5A                                add     eax, 5Ah ; 'Z'
seg000:00000031 FF E0                                   jmp     eax
seg000:00000031                         ; ---------------------------------------------------------------------------
```

后面的分析与各厂商的 CVE-2018-0802 漏洞分析报告大同小异，就不再做具体说明了，欢迎进行讨论交流。

解决方案：

1、下载微软最新补丁。

(1)https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-0802

(2)开启 Windows 自动更新。微软最新版本已经去掉了该模块微软将 EQNEDT32.EXE 从产品中删除，并不再支持旧版本的公式格式。

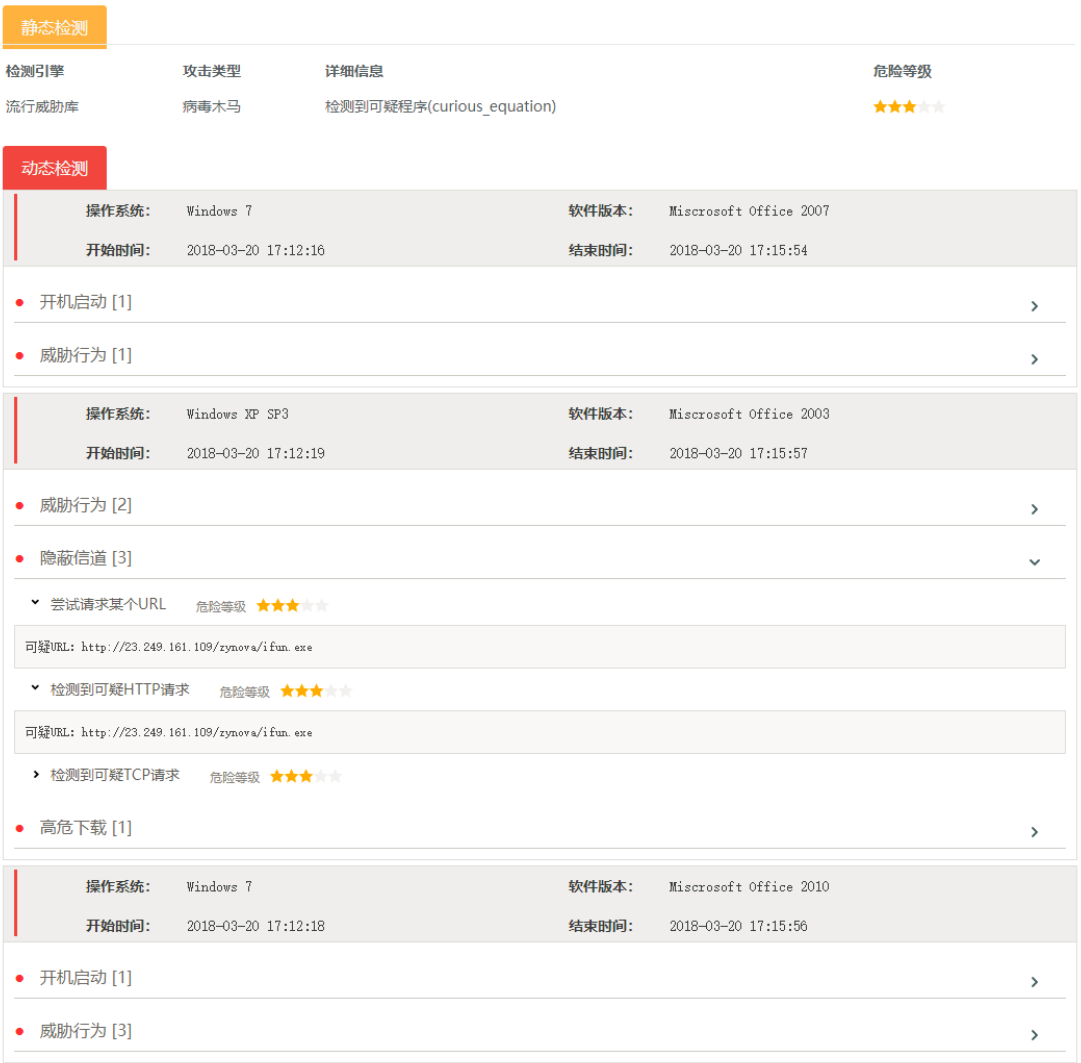
```
Reg add
“ HKLM\SOFTWARE\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-000000000046}” /v “Compatibility Flags” /t REG_DWORD /d 0x400
```

2、在注册表中取消该模块的注册。

```
reg add “HKLM\SOFTWARE\Wow6432Node\Microsoft\Office\Common\COM Compatibility\{0002CE02-0000-0000-C000-000000000046}” /v “Compatibility Flags” /t REG_DWORD /d 0x400
```

对于在 64 位操作系统上的 32 位的 Office，执行下列操作

3、天阗高级持续性威胁检测与管理系统，无需升级即可有效检测变种公式编辑器漏洞样本。



4、使用 VenusEye 威胁情报中心可以查询与该样本相关的威胁情报信息。

其他相关 IOC：

www[.]pensandwoodworking[.]com
www[.]euroasianbridge[.]com
www[.]dlpestscontrol[.]com
www[.]salihatasarim[.]com
www[.]datascienceactuaries[.]net
www[.]wisataanambas[.]com
www[.]5nesglaz[.]online
www[.]tewyt[.]info
www[.]rahafim[.]com
www[.]gardenshades[.]com
www[.]beb-sef-sufficient[.]com
www[.]femmefeline[.]com
www[.]edosensei[.]com
www[.]outdoormerk[.]com
www[.]prfitvxnfe[.]info

www[.]bloggingsays[.]com
www[.]rahafim[.]com

关于金睛安全研究团队

金睛安全研究团队是启明星辰集团检测产品本部从事专业安全分析的技术型团队，主要职责是对现有产品上报的安全事件、样本数据进行挖掘、分析，并向用户提供专业的分析报告。

关于 VenusEye 威胁情报中心

VenusEye 威胁情报中心（www.venuseye.vip）是启明星辰倾力打造的集威胁情报收集、分析、处理、发布和应用为一体的威胁情报云服务平台，提供威胁情报数据、系统、技术和专业能力的输出。