CVE-2018-8423：Jet数据库引擎漏洞利用

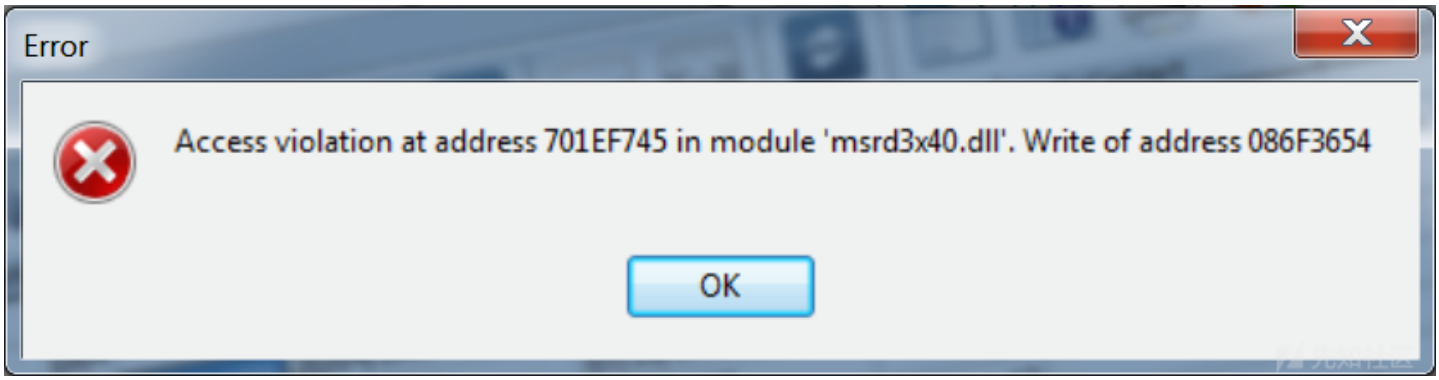
angel010 / 2019-08-03 09:04:00 / 浏览数 3663 安全技术 漏洞分析 顶(0) 踩(0)

2018年9月，Zero Day Initiative发布了微软Jet Database
Engine漏洞CVE-2018-8423的PoC，并于2018年10月发布了补丁。研究人员在分析该漏洞和补丁时发现该漏洞的补丁会引发另一个漏洞CVE-2019-0576。
该漏洞利用了许多微软应用中都使用的Jet Database
Engine的漏洞，其中就包括Access。攻击者利用该漏洞可以执行代码来进行权限提升或下载恶意软件。无从得知该漏洞是否已经被用于攻击活动中，但是PoC代码是随处可

概览

为了利用该漏洞，攻击者需要利用社工技术使受害者打开一个JS文件，该JS文件使用ADODB
`connection`对象来访问恶意Jet数据库文件。一旦访问恶意Jet数据库文件，就会调用`msrd3x40.dll`中有漏洞的函数，最终导致该漏洞被成功利用。

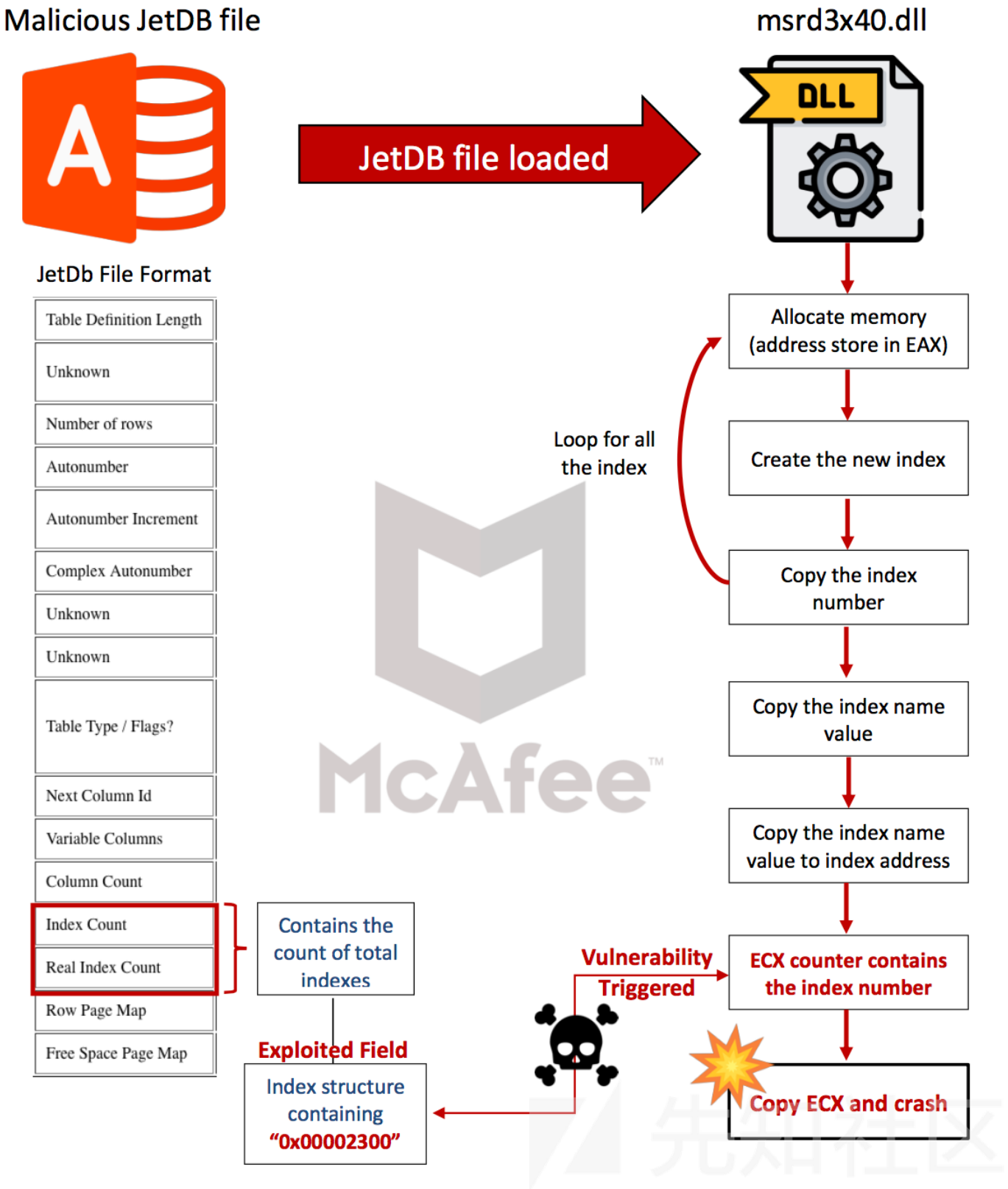
虽然PoC是导致`wscript.exe`奔溃，但实际上所有使用该DLL的应用都可能会受到该攻击的影响。
下面的错误信息表明该漏洞会被成功触发：



该消息表明在有漏洞的DLL中引发了访问违反。该漏洞是一个越界写漏洞，可以通过OLE
DB触发，该API是许多微软应用中用来访问数据。该类漏洞表明数据可以在目标缓冲区外写数据导致奔溃。奔溃的原因是由于恶意伪造的JET数据库文件。利用Jet数据库文件

下图介绍了该漏洞利用的过程：

CVE-2018-8423



漏洞利用

PoC代码中含有一个JS文件(`poc.js`)，会调用第二个文件(`group 1`)。这是一个Jet数据库文件。通过`wscript.exe`运行`poc.js`就会触发该奔溃。

```
0:000> g
(bf0.aa8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=01dcae60 ebx=06fb427c ecx=00002300 edx=01dca868 esi=01dccbf0 edi=00000001
eip=68f5f745 esp=001fcff8 ebp=06fb4258 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
msrd3x40!TblPage::CreateIndexes+0x175:
68f5f745 89b48a74050000  mov     dword ptr [edx+ecx*4+574h],esi ds:0023:01dd39dc=????????
```

从图中可以看出，引发该函数奔溃的是msrd3x40!TblPage::CreateIndexes。还可以确定该程序在尝试写数据，但是失败了。该程序用esi寄存器来在位置[edx+ecx*

Debug信息表明寄存器ecx中含有值0x00002300。Edx是一个指向内存的指针。最后，将这些和offset 0x574添加到一起来引用内存位置。根据这些信息可以猜测这里保存的数据类型。应该是一个数组，每个变量是4字节长，从位置edx+574h处开始。在追踪给程序时，研究1。

```
1200h: FF 00 FF FF 00 FF FF 00 FF FF 00 FF FF 00 FF FF  ÿ.ÿÿ.ÿÿ.ÿÿ.ÿÿ.ÿÿ
1210h: 00 FF FF 00 00 08 00 00 09 00 00 00 01 00 00 01  .ÿÿ.............
1220h: FF FF 00 FF FF 00 FF FF 00 FF FF 00 FF FF 00 FF  ÿÿ.ÿÿ.ÿÿ.ÿÿ.ÿÿ.ÿ
1230h: FF 00 FF FF 00 FF FF 00 FF FF 00 00 0A 00 00 0B  ÿ.ÿÿ.ÿÿ.ÿÿ......
1240h: 00 00 00 01 01 00 00 00 01 00 00 00 00 FF FF FF  .............ÿÿÿ
1250h: FF 00 00 00 00 04 04 01 00 23 00 00 00 00 00 00  ÿ.......#......
1260h: 00 FF FF FF FF 00 00 00 00 04 04 00 02 49 64 0C  .ÿÿÿÿ........Id.
1270h: 50 61 72 65 6E 74 49 64 4E 61 6D 65 09 00 04 06  ParentIdName....
```

从上面我们了解到该程序会尝试越界写，并且确定了越界写发生的位置。下面分析为什么程序会尝试在该位置写。再此之前，需要了解Jet数据库，并根据用户提供的0x000

分析Jet数据库文件

许多研究人员都分析过Jet数据库文件结构。其中包括：

- Jabakobob.net http://jabakobob.net/mdb/
- Brian B GitHub https://github.com/brianb/mdbtools/blob/master/HACKING

总结一下就是，Jet数据库文件是以页的形式组织，如下所示：

header page含有与文件相关的不同信息：

Name	Offset	Length	Type	Description
Magic Number	0x00	4 bytes	UINT 32 LE	0x100
File format ID	0x04	16 bytes	CHAR	A zero-terminated string identifying the file format • MDB format (Access 97-2003): "Standard Jet DB" • ACCDB format (Access 2007-2010): "Standard ACE DB"
Jet Version	0x14	4 bytes	UINT 32 LE	JET file format version • 0 Access 97 (Jet 3) • 1 Access 2000, 2002/2003 (Jet 4) • 2 Access 2007 • 0x103 Access 2010

Header之后是含有key 0x6b39dac7的RC4加密的126字节，其中key对每个JetDB文件都是一样的。与PoC文件key值对比，可以发现group 1是Jet Version 3文件。

更多页pages定义参见http://jabakobob.net/mdb/table-page.html。

Table的定义数据有不同的域，其中包括Index Count和Real Index Count。

Table Definition data

Table Definition Length	4 bytes	UINT 32 LE	The total length of the table definition.
Unknown	4 bytes Jet 4 only	???	Unknown field, Jet 4 and later only
Number of rows	4 bytes	UINT 32 LE	The total number of rows in the table
Autonumber	4 bytes	UINT 32 LE	The next value for the autonumber field
Autonumber Increment	4 bytes Jet 4 only	UINT 32 LE	Jet 4 only, *probably* the amount that the autonumber is increased everytime (I haven't tested this)
Complex Autonumber	4 bytes	UINT 32 LE	Jet 4 only. On Access 2007 and later, this contains the Autonumber for complex fields (shared across all complex fields). Unknown for earlier versions.
Unknown	4 bytes	UINT 32 LE	Jet 4 only
Unknown	4 bytes	UINT 32 LE	Jet 4 only
Table Type / Flags?	1 bytes	UINT 8	The type of the table, or maybe some table flags? Known values: • 0x4e user table • 0x53 system table
Next Column Id	2 bytes	UINT 16 LE	The Column Id that the next column to be created will have. Incremented every time a column is created, never decremented. Equal to the total number of columns in the table including deleted columns.
Variable Columns	2 bytes	UINT 16 LE	The number of variable length columns in the table
Column Count	2 bytes	UINT 16 LE	The number of columns in the table
Index Count	4 bytes	UINT 32 LE	The total number of indexes in the table, including those that aren't real indices.
Real Index Count	4 bytes	UINT 32 LE	The number of real indices in the table
Row Page Map	4 bytes	UINT 32 LE	A record pointer to a page bitmap of all pages that contain rows in this table (excluding LVAL pages)
Free Space Page Map	4 bytes	UINT 32 LE	A record pointer to a page bitmap for pages containing free space (for inserting rows).

可以确定PoC文件中的这些值。检查group 1文件研究人员发现：

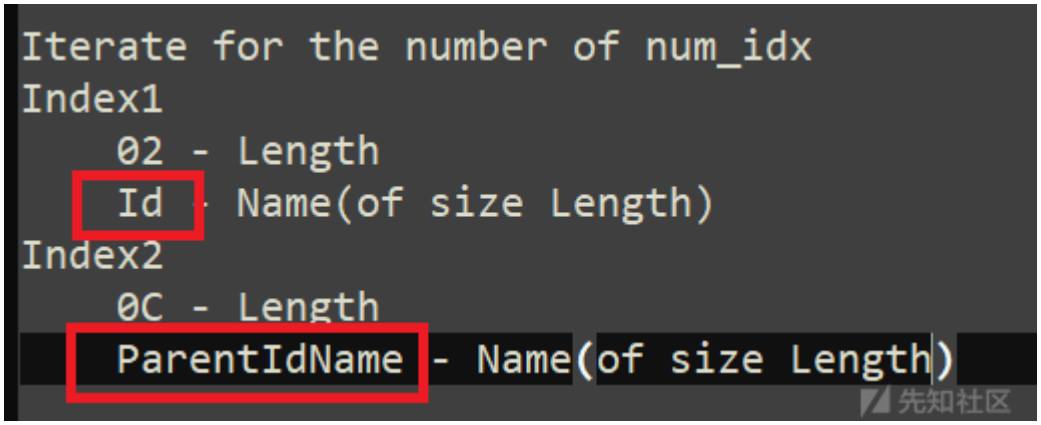
```
02 01 - table definition page identifier.
56 43 - VC
00 00 00 00 - next page
C4 02 00 00 - Table Definition Length
10 00 00 00 - Number of rows
00 00 00 00 - Autonumber
53 - Table Type / Flags? ==> system table
11 00 - Next Column Id
0B 00 - Variable Columns
11 00 - Column Count
02 00 00 00 - Index Count
02 00 00 00 - Real Index Count
00 06 00 00 - Row Page Map
01 06 00 00 - Free Space Page Map
```


在`Index Count`中一共有2个`index`。分析这个index研究人员值是很熟悉的`0x00002300`：

```
for every index (including those that aren't real):
index1:
01 00 00 00 - Index Number
01 00 00 00 - Index Column Number
00 - type of the other table in this fk
FF FF FF FF - index number of other index in fk
00 00 00 00 - page number of other table in fk
04 - flag indicating if updates are cascaded
04 - flag indicating if deletes are cascaded
01 - index_type

index2:
00 23 00 00 - Index Number
00 00 00 00 - Index Column Number
00 - type of the other table in this fk
FF FF FF FF - index number of other index in fk
00 00 00 00 - page number of other table in fk
04 - flag indicating if updates are cascaded
04 - flag indicating if deletes are cascaded
00 - index_type
```


`0x00230000`是table中`index2`的`index`值。Index看似非常大，而且与crash有关。
为什么会使程序奔溃呢？通过分析文件，研究人员发现了两个index的名：



Debug

通过debugger，研究人员发现第一个程序调用了函数`msrd3x40!operator new`。这会为在`eax`中存储内存指针地址来分配内存：

Offset: @$scopeip

```
6924f63a 83c701          add     edi,1
6924f63d 83c214          add     edx,14h
6924f640 3bf8            cmp     edi,eax
6924f642 7cec            jl      msrd3x40!TblPage::CreateIndexes+0x60 (6924f630)
6924f644 8d0c80          lea     ecx,[eax+eax*4]
6924f647 33ff            xor     edi,edi
6924f649 85c0            test    eax,eax
6924f64b 8d5c8d00        lea     ebx,[ebp+ecx*4]
6924f64f 0f8eb5010000    jle     msrd3x40!TblPage::CreateIndexes+0x23a (6924f80a)
6924f655 8b542414        mov     edx,dword ptr [esp+14h]
6924f659 81c2f4050000    add     edx,5F4h
6924f65f 8954241c        mov     dword ptr [esp+1Ch],edx
6924f663 68c4000000      push    0C4h
6924f668 e8ba220000      call    msrd3x40!operator new (69251927)
6924f66d 83c404          add     esp,4
6924f672 0f8403010000    je      msrd3x40!TblPage::CreateIndexes+0x1ab (6924f77b)
6924f678 56              push    esi
6924f679 8bc8            mov     ecx,eax
6924f67b e87021feff      call    msrd3x40!Index::Index (692317f0)
6924f680 8bf0            mov     esi,eax
6924f682 85f6            test    esi,esi
6924f684 0f84ed000000    je      msrd3x40!TblPage::CreateIndexes+0x1a7 (6924f777)
6924f68a 8b442410        mov     eax,dword ptr [esp+10h]
6924f68e f60008          test    byte ptr [eax],8
6924f691 0f85ff000000    jne     msrd3x40!TblPage::CreateIndexes+0x1c6 (6924f796)
6924f697 55              push    ebp
6924f698 8bce            mov     ecx,esi
6924f69a e83123feff      call    msrd3x40!Index::Restore (692319d0)
```

Command

```
6924f659 81c2f4050000    add     edx,5F4h
0:000>
eax=00000002 ebx=071f426c ecx=0000000a edx=017dae5c esi=0015d4ac edi=00000000
eip=6924f65f esp=0015d320 ebp=071f4244 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
msrd3x40!TblPage::CreateIndexes+0x8f:
6924f65f 8954241c        mov     dword ptr [esp+1Ch],edx ss:0023:0015d33c=071f403b
0:000>
eax=00000002 ebx=071f426c ecx=0000000a edx=017dae5c esi=0015d4ac edi=00000000
eip=6924f663 esp=0015d320 ebp=071f4244 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
msrd3x40!TblPage::CreateIndexes+0x93:
6924f663 68c4000000      push    0C4h
0:000>
eax=00000002 ebx=071f426c ecx=0000000a edx=017dae5c esi=0015d4ac edi=00000000
eip=6924f668 esp=0015d31c ebp=071f4244 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
msrd3x40!TblPage::CreateIndexes+0x98:
6924f668 e8ba220000      call    msrd3x40!operator new (69251927)
0:000>
eax=017dcae0 ebx=071f426c ecx=000000c4 edx=017dcadb esi=0015d4ac edi=00000000
eip=6924f66d esp=0015d31c ebp=071f4244 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
msrd3x40!TblPage::CreateIndexes+0x9d:
6924f66d 83c404          add     esp,4
```

内存分配后，程序会创建一个新的index：

Disassembly

Offset: @$scopeip

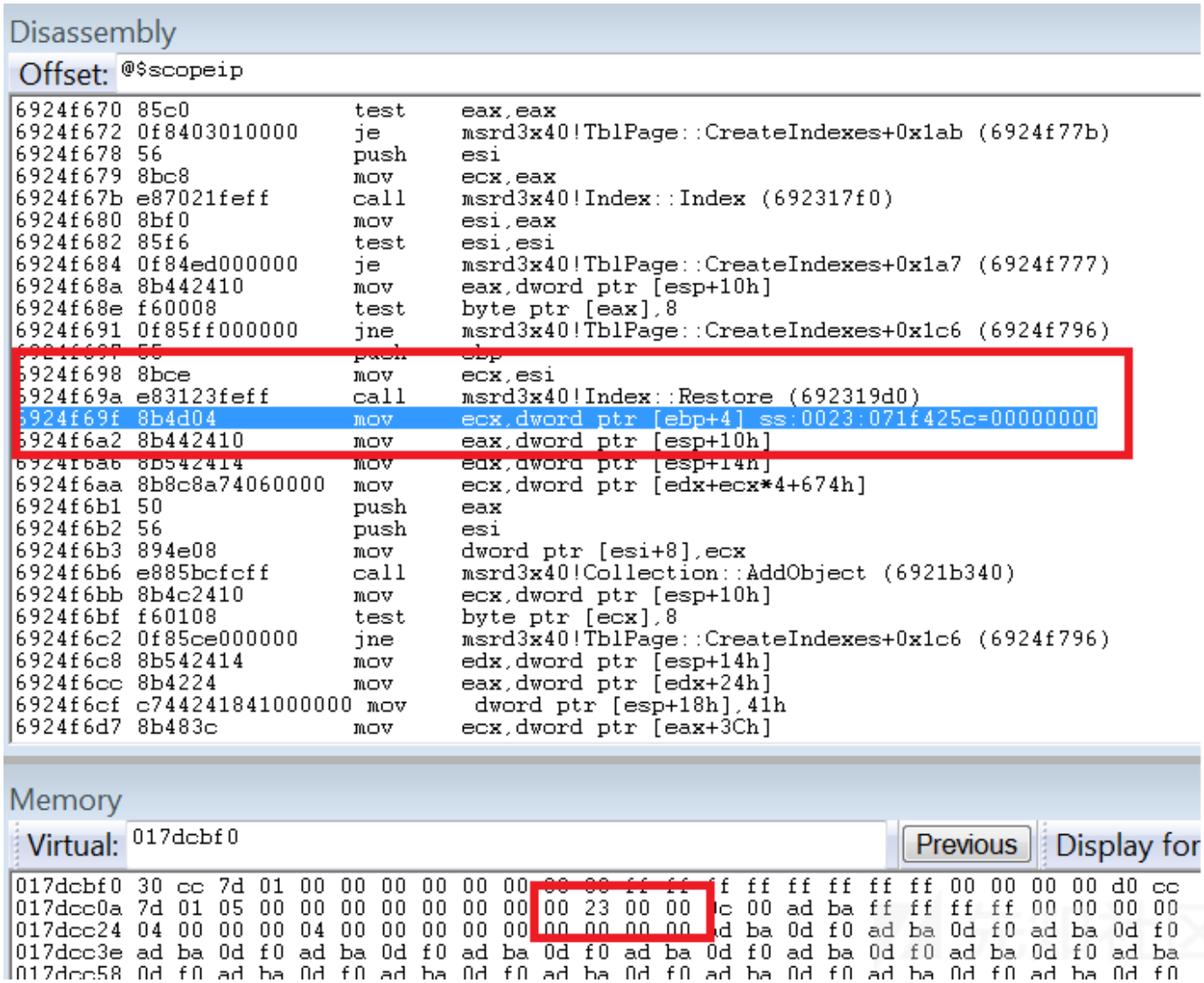
```
6924f649 85c0            test    eax,eax
6924f64b 8d5c8d00        lea     ebx,[ebp+ecx*4]
6924f64f 0f8eb5010000    jle     msrd3x40!TblPage::CreateIndexes+0x23a (6924f80a)
6924f655 8b542414        mov     edx,dword ptr [esp+14h]
6924f659 81c2f4050000    add     edx,5F4h
6924f65f 8954241c        mov     dword ptr [esp+1Ch],edx
6924f663 68c4000000      push    0C4h
6924f668 e8ba220000      call    msrd3x40!operator new (69251927)
6924f66d 83c404          add     esp,4
6924f670 85c0            test    eax,eax
6924f672 0f8403010000    je      msrd3x40!TblPage::CreateIndexes+0x1ab (6924f77b)
6924f678 56              push    esi
6924f679 8bc8            mov     ecx,eax
6924f67b e87021feff      call    msrd3x40!Index::Index (692317f0)
6924f680 8bf0            mov     esi,eax
6924f682 85f6            test    esi,esi
6924f684 0f84ed000000    je      msrd3x40!TblPage::CreateIndexes+0x1a7 (6924f777)
6924f68a 8b442410        mov     eax,dword ptr [esp+10h]
6924f68e f60008          test    byte ptr [eax],8
6924f691 0f85ff000000    jne     msrd3x40!TblPage::CreateIndexes+0x1c6 (6924f796)
6924f697 55              push    ebp
6924f698 8bce            mov     ecx,esi
6924f69a e83123feff      call    msrd3x40!Index::Restore (692319d0)
6924f69f 8b4d04          mov     ecx,dword ptr [ebp+4]
6924f6a2 8b442410        mov     eax,dword ptr [esp+10h]
6924f6a6 8b542414        mov     edx,dword ptr [esp+14h]
6924f6aa 8b8c8a74060000  mov     ecx,dword ptr [edx+ecx*4+674h]
6924f6b1 50              push    eax
6924f6b2 56              push    esi
```

Memory

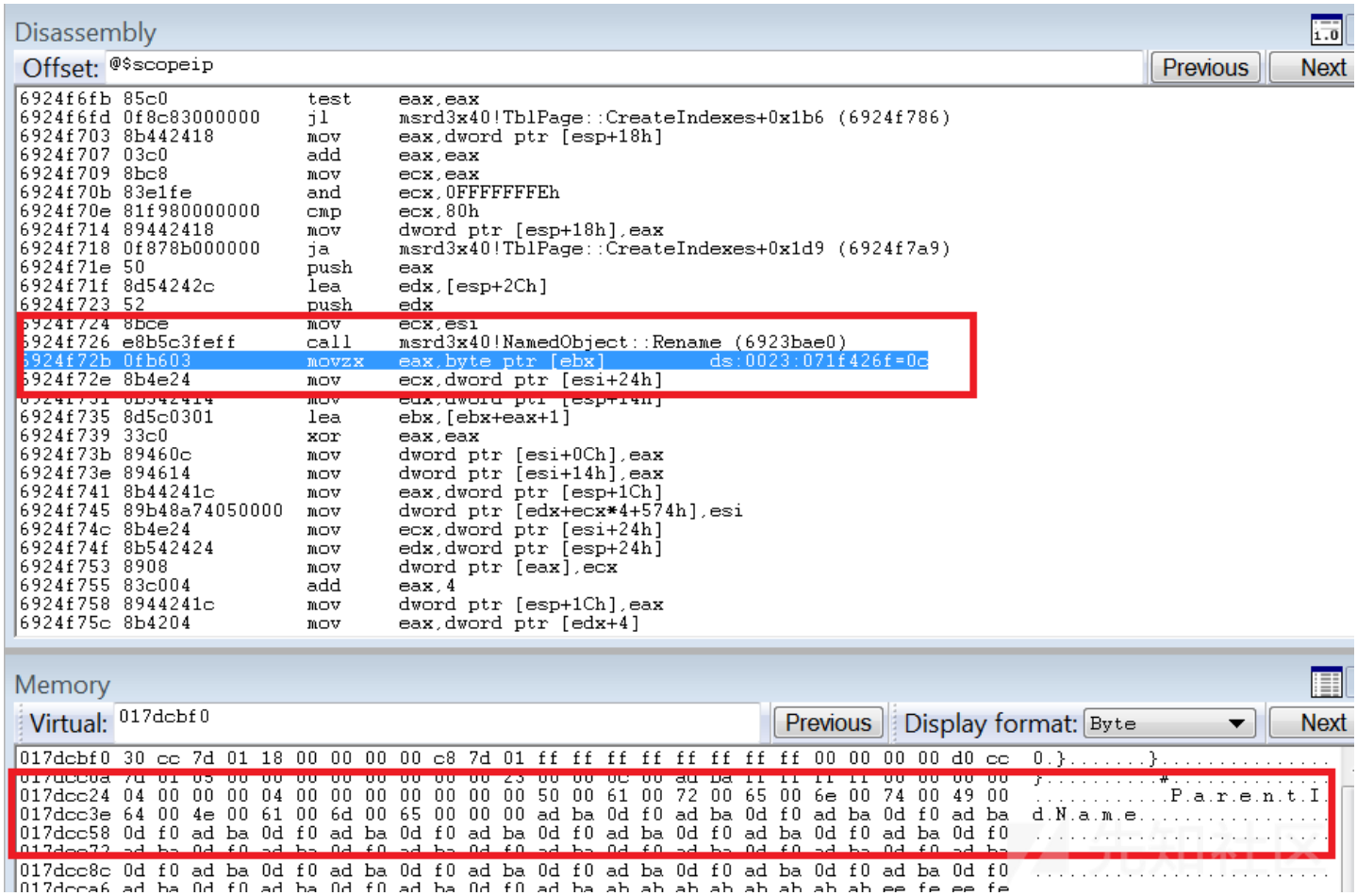
Virtual: 017dcae0 Previous Display format: Byte

```
017dcae0 20 cb 7d 01 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff 00 00 00 00 c0 cb  .}.....
017dcaf8 7d 01 05 00 00 00 00 00 00 00 00 ff ff ff ff 0c 00 ad ba ff ff ff ff 00 00 00 00  }.....
017dcb14 04 00 00 00 04 00 00 00 00 00 00 00 00 00 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0  .....
017dcb2e ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba  .....
017dcb48 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0  .....
017dcb62 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba  .....
017dcb7c 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0  .....
017dcb96 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba ab ab ab ab ab ab ab ab ab ab fe fe  .....
017dcbb0 00 00 00 00 00 00 00 00 0b 03 9b 69 82 fe 00 18 0d f0 ad ba 0d f0 ad ba 0d f0  .....
017dcbca ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba ab ab ab ab ab ab ab ab ab 00 00 00 00
```

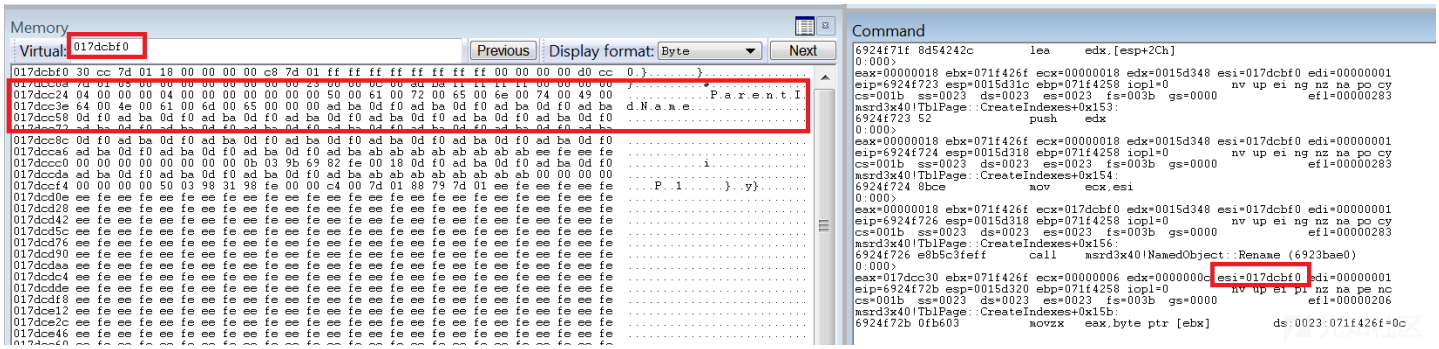
该index值用于后面的执行过程。函数msrd3x40!Index::Restore会复制该index数到index■■+24h。这一过程会在所对所有index循环。首先调用分配内存的new，然



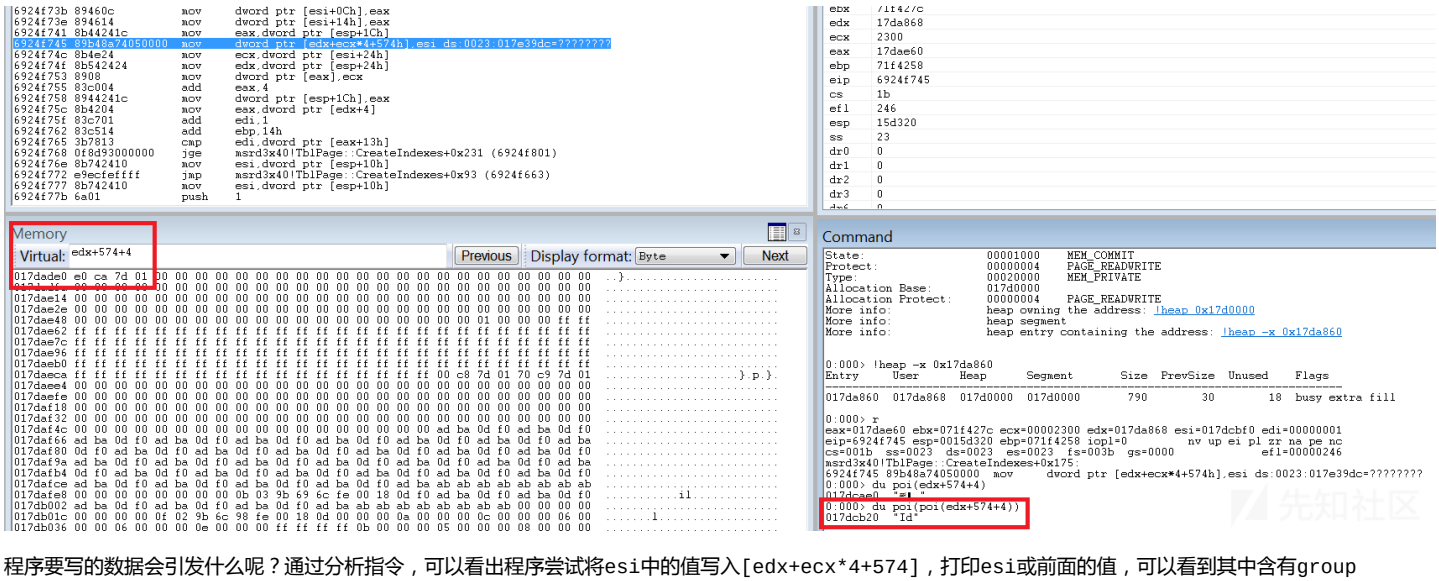
成功移动后，函数msrd3x40!NamedObject::Rename会被调用并复制index name值到index■■+40h：



查看esi寄存器可以看到它指向index的地址。Ecx寄存器的值为[esi+24h]，这也就是index数：

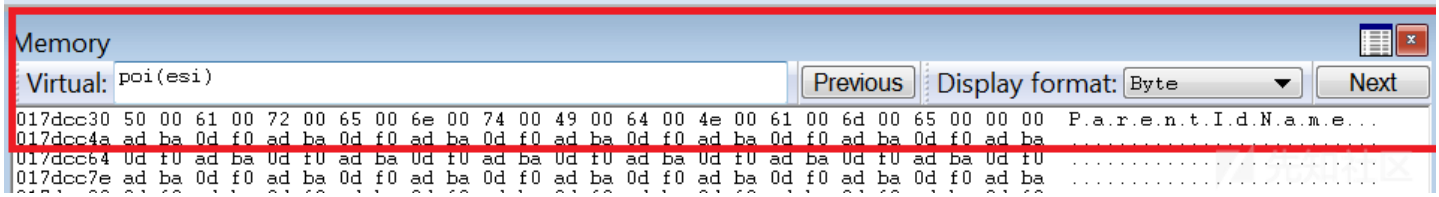


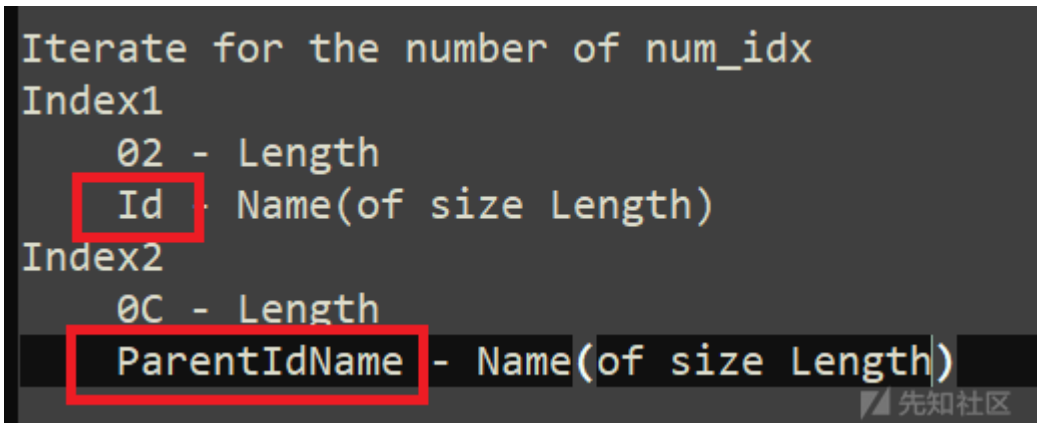
在一些指令操作后，可以看到引发奔溃的指令。Edx指向内存地址，ecx中含有来自文件group 1的非常大的数。该程序尝试访问[edx+ecx*4+574h]处的内存，这会引发越界写和程序奔溃：



程序要写的数据会引发什么呢？通过分析指令，可以看出程序尝试将esi中的值写入[edx+ecx*4+574]，打印esi或前面的值，可以看到其中含有group 1中出现过的index name ParentIdName：

```
6924f73e 894614          mov     dword ptr [esi+14h],eax
6924f741 8b44241c        mov     eax,dword ptr [esp+1Ch]
6924f745 89b48a74050000  mov     dword ptr [edx+ecx*4+574h],esi ds:0023:017e39dc=????????
6924f74c 8b4e24          mov     ecx,dword ptr [esi+24h]
6924f74f 8b542424        mov     edx,dword ptr [esp+24h]
6924f753 8908            mov     dword ptr [eax],ecx
6924f755 83c004          add     eax,4
6924f758 8944241c        mov     dword ptr [esp+1Ch],eax
6924f75c 8b4204          mov     eax,dword ptr [edx+4]
6924f75f 83c701          add     edi,1
6924f762 83c514          add     ebp,14h
6924f765 3b7813          cmp     edi,dword ptr [eax+13h]
6924f768 0f8d93000000    jge     msrd3x40!TblPage::CreateIndexes+0x231 (6924f801)
6924f76e 8b742410        mov     esi,dword ptr [esp+10h]
6924f772 e9ecfeffff      jmp     msrd3x40!TblPage::CreateIndexes+0x93 (6924f663)
6924f777 8b742410        mov     esi,dword ptr [esp+10h]
6924f77b 6a01            push    1
```





最后，程序在尝试用很大的index数处理ParentIDName时就会奔溃。逻辑是：

- 分配内存并获取内存开始位置的指针。
- 从■■■■+674h开始，程序会用文件中提到的index数乘4来保存指向index name的指针。

如果index数很大，又没有有效性验证，程序就会尝试越界写，并奔溃。

结论

这是一个逻辑错误，而此类错误一般很难发现。许多开发者都会采取额外的预防措施来避免代码中出现类似bug。如果出现了类似的问题，研究人员建议尽快应用厂商提供的

微软提供的补丁参见：
CVE-2018-8423
https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8423
CVE-2019-0576
https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0576

参考：
• https://github.com/brianb/mdbtools/blob/master/HACKING
• http://jabakobob.net/mdb/table-page.html

本文翻译自：https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/jet-database-engine-flaw-may-lead-to-exploitation-analyzing-cve-2018-8423/

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