Microsoft Office Excel 2003 SP2
'Heap Buffer Overflow' vulnérabilité

Fournisseur:
Microsoft

Système affecté :
Microsoft Office Excel 2003 SP2 avec la dernière mise à jour de sécurité (KB943985) v2 ainsi que les versions suivantes.

Références CVE :
CVE-2008-3477

Informations complémentaires :
http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=746
http://www.microsoft.com/technet/security/bulletin/ms08-057.mspx

Description générale :

Un Heap Overflow existe dans la dernière mise à jour de sécurité (last Security Update) (KB943985) v2 pour l'application Excel 2003.
Le problème existe avec un champ non vérifié de tableau d'objet OLE. Ce champ représente le "nombre d'objet". Lorsque ce champ est supérieur à 0, Excel considère que le tableau ne contient qu'un seul objet et réserve la mémoire minimum. Mais Excel copie ensuite autant d'objet que le défini ce champ.

Lorsque ce champ est supérieur à 1, un Heap overflow se produit inévitablement lors du traitement des objets OLE en mémoire.

D'après mes tests, ce problème n'existe qu'avec la dernière mise à jour de sécurité (last Security Update) (KB943985) v2 pour Excel 2003. Cette mise à jour ne peut être installée qu'à partir du SP2 d'Excel.
Excel 2002(XP) et 2007 ne s'emblent pas affectés par ce problème avec les dernières mises à jour de sécurité.

Description technique :

Tous les détails techniques sont décrits avec le module VBE6.DLL version 6.5.10.24 (Excel 2003 SP3).

Les objets responsables de ce problème sont de cette forme :

```
Struct ole_VBA_object
{
    byte[0x10]      ID_VBA_Object;
}
```

Ils sont définis dans un tableau :

```
Struct Array_ole_VBA_object
{
    unsigned short              Nb_Object;
    short                       align;          //(0)
    ole_VBA_object[Nb_Object]   list_ID_vba_object;
    dword                       sizeof_list_ID_vba_object _???; //(0x10)
}
```

Dans le fichier xls, un tableau est toujours immédiatement suivi de cette suite d'octets : 03 00 00 00 05

Un champ Nb_Object à 0 indique un tableau vide d'un objet et celui-ci n'est pas traité par Excel.
Si ce champ n'est pas null, alors celui-ci est toujours à 1 dans le fichier original.xls.

Dans le fichier original.xls, le 1er tableau non vide se trouve à l'offset 0x1603 :

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00001600   00 00 00 01 00 00 00 98  02 EE 1B AF 98 98 46 83   .......˜.î.¯˜˜F
00001610   E6 5B 12 60 9A 94 2D 10  00 00 00 03 00 00 00 05   æ[.`š”-.........
```

En totalité, il existe 3 tableaux non vides dans le fichier original.xls.

Pour chaque tableau, Excel alloue juste assez de mémoire pour recevoir un seul ole_VBA_object. Les blocks mémoire créés dans le heap sont de 0x14 octets (sizeof (dword) + (sizeof (object)*1)).
Ce qui donne des blocks de 0x20 octets avec le chunck et l'alignement sur 8.

Voici le code de VBE6.DLL qui traite ce type de tableau :

```
.text:651A0B92     cmp word ptr [edi], 0
.text:651A0B96     jbe short loc_651A0BF3 ;if (Nb_Object==0) then jmp (the array isn't to copy in the heap)
.text:651A0B98 ****************************************
.text:651A0B98 * Treatment of the array of ole_VBA_object
.text:651A0B98 ****************************************
.text:651A0B98     push 14h                            ; sizeof array with one ole_VBA_object
.text:651A0B9A     call Yo_Malloc
.text:651A0B9F     test eax, eax
.text:651A0BA1     jnz short loc_651A0BAD
.text:651A0BA3
.text:651A0BA3 loc_651A0BA3:
.text:651A0BA3     mov eax, 8007000Eh  ; alloc error
.text:651A0BA8     jmp loc_651A0C6C
.text:651A0BAD ; ---------------------------------------------------------------------------
.text:651A0BAD
.text:651A0BAD loc_651A0BAD:
.text:651A0BAD     movzx ecx, word ptr [edi]           ; Nb_Object (in xls file)
.text:651A0BB0     and [ebp+cpt_ole_VBA_object], 0
.text:651A0BB4     shl ecx, 4
.text:651A0BB7     mov [eax], ecx                      ; Total_size = Nb_Object * sizeof(ID_VBA_Object)
.text:651A0BB9     add eax, 4
.text:651A0BBC     cmp word ptr [edi], 0
.text:651A0BC0     mov [esi+0BCh], eax                 ; p_first_ID_VBA_Object_in_heap
.text:651A0BC6     jbe short loc_651A0BF3   ; if (Nb_Object==0) then jmp (array of one empty ID_VBA_object)
.text:651A0BC8     and [ebp+heap_offset_ole_VBA_object], 0

.text:651A0BCC LOOP:
.text:651A0BCC     mov eax, [esi+0BCh]                 ; p_first_ID_VBA_Object_in_heap
.text:651A0BD2     add eax, [ebp+heap_offset_ole_VBA_object]
.text:651A0BD5 ********************************************
.text:651A0BD5 * As of the second passage in the loop, there
.text:651A0BD5 * is heap overflow
.text:651A0BD5 ********************************************
.text:651A0BD5     push eax
.text:651A0BD6     push ebx
.text:651A0BD7     call Copy_ID_VBA_Object_To_Heap
.text:651A0BDC     test eax, eax
.text:651A0BDE     jl  loc_651A0C6C
.text:651A0BE4     movzx eax, word ptr [edi]
.text:651A0BE7     inc [ebp+cpt_ole_VBA_object]
.text:651A0BEA     add [ebp+heap_offset_ole_VBA_object], 10h   ; heap_offset_ole_VBA_object
                                                               ;            += sizeof (ID_VBA_object)
.text:651A0BEE     cmp [ebp+cpt_ole_VBA_object], eax
.text:651A0BF1     jb  short LOOP
```

Avec Nb_Object > 1, le heap owerflow se produit dans la routine **Copy_ID_VBA_Object_To_Heap**.

Les 3 tableaux valides du fichier xls original permet de générer 3 heap owerflow avec des contextes mémoire différents.
Les 3 POC fournis avec ce texte permettent de voir ces 3 cas de figures.

Chaque tableau est corrompu comme ceci :

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

0000XX00   00 00 00 FF FF 00 00 YY  YY YY YY YY YY YY YY YY
0000XX10   YY YY YY YY YY YY YY 10  00 00 00 41 41 42 42 43
0000XX20   43 44 44
```

Le nombre de ID_VBA_Object est mis à sa valeur maximum (0xFFFF).
Et la suite d'octet 41 41 42 42 43 43 44 44 écrase le chunck du block suivant dans le heap.

Le champ Nb_Object à 0xFFFFF permet d'écraser 1048536 octets ((0xFFFF*0x10)-0x18 = 0xFFFD8) avec des données arbitraire.

Le code qui génère la corruption du heap :

```
.text:65002E27     mov eax, [esi+ecx*4-0Ch]
.text:65002E2B     mov [edi+ecx*4-0Ch], eax    <--- crushing of chunck of block
.text:65002E2F     jmp loc_65002CBC
```

Le breakpoint (windbg) qui permet de voir le heap owerflow avec les POC :

```
bp 65002E2B "j(@eax==0x42424141) ''; 'gc'"
OR
bu VBE6+0x2E2B "j(@eax==0x42424141) ''; 'gc'"
```

Les POC1 et POC2 génèrent le heap owerflow dés leurs ouvertures avec Excel.
Le POC3 affecte un objet VBA qui est traité lorsque l'utilisateur accède au projet VBA. Après l'ouverture du POC3, les manipulations suivantes génèrent le heap overflow :
- Tools->Macro->Visual Basic Editor
OR
- Alt+F11
OR
- clic-droit sur un sheet -> View Code

Merci d'avoir lu ce papier jusqu'au bout. :)

That's all folks !
14 octobre 2008
Lionel d'Hauenens - www.laboskopia.com –