VUPEN security

VUPEN Security – Private Exploits & PoC Service

In-Depth Analysis of Microsoft Office PowerPoint OEPlaceHolderAtom Use-after-free Vulnerability (MS10-004 / CVE-2010-0032)

Table of Contents

Introduction

A vulnerability exists in Microsoft Office PowerPoin when processing malformed PPT files, which could lead to arbitrary code execution.

Tested Versions

The vulnerability was analysed on Windows XP SP2 with Microsoft Office PowerPoint 2003 SP3 (Powerpnt.exe version 11.0.8307.0).

Fixed Versions

The vulnerability was fixed with the MS10-004 security update.

Technical Details

A Powerpoint document may embed containers like Handout, MainMaster, Notes or Slides to record data used in the different parts of the presentation. Each of these four containers can contain several atoms, some of them being optional.

A use-after-free vulnerability exists in Powerpoint triggered by a malicious use of the OEPlaceholderAtom atom (opcode 3011 or 0BC3h). This atom contains usually 8 bytes and has the following organization:

Offset	Field	Size
0	placementId	4
+4	placeholderId	1
+5	Size	1
+6	Unused	2

The first time Powerpoint encounters such atom, it allocates an object o1 whose reference is saved in a second object. If a second atom is found in the same container, o1 is first deleted and the program parses the rest of the atom. If this atom is corrupted, parsing is skipped and a reference to a deleted object stays in memory.

All of this occurs in sub_3009C4CB, which parses several atoms:

```
.text:3009C4D8          mov     ebx, [ebp+0Ch]
.text:3009C4DB          mov     ax, [ebx+2]
.text:3009C4DF          cmp     ax, 0BC3h               //case OEPlaceholderAtom
.text:3009C4E3          push    esi
.text:3009C4E4          push    edi
.text:3009C4E5          mov     edi, ecx
.text:3009C4E7          jnz     loc_300D98F8
.text:3009C4ED          mov     eax, [edi+4]
.text:3009C4F0          mov     esi, [eax+24h]
.text:3009C4F3          test    esi, esi
```

The first time Powerpoint encounters this atom, ESI is null so the next jump is not taken:

```
.text:3009C4F5          jnz     loc_301CA9A0
.text:3009C4FB
.text:3009C4FB loc_3009C4FB:
.text:3009C4FB          mov     ecx, [ebp+10h]
.text:3009C4FE          push    8
```

```
.text:3009C500          call    sub_3009D869          //read 8 bytes from the file
.text:3009C505          mov     esi, eax
.text:3009C507          mov     al, [esi+4]           //test placeholderId
.text:3009C50A          cmp     al, 1
.text:3009C50C          jb      short loc_3009C549
.text:3009C50E          cmp     al, 18h
.text:3009C510          ja      short loc_3009C549
```

Here, the program clearly expects "placeholderId" to be between 1 and 18h. Otherwise the function returns.

```
.text:3009C512          push    18h
.text:3009C514          call    sub_30004D75          //allocate o1, which is 18h bytes long
.text:3009C519          mov     [ebp+10h], eax
.text:3009C51C          push    dword ptr [edi+4]
.text:3009C51F          and     dword ptr [ebp-4], 0
.text:3009C523          mov     ecx, eax
.text:3009C525          call    sub_3009DF7B          //initialize o1
.text:3009C52A          mov     ecx, [edi+4]
.text:3009C52D          or      dword ptr [ebp-4], 0FFFFFFFFh
.text:3009C531          mov     [ecx+24h], eax        //save a reference to o1 to an internal structure
.text:3009C534          mov     al, [esi+5]
.text:3009C537          push    eax
.text:3009C538          mov     al, [esi+4]
.text:3009C53B          push    dword ptr [esi]
.text:3009C53D          push    eax
.text:3009C53E          mov     eax, [edi+4]
.text:3009C541          mov     ecx, [eax+24h]
.text:3009C544          call    sub_3009DFA9          //write other values to o1
```

The second time an "OEPlaceholderAtom" is found, ESI points to o1:

```
.text:3009C4ED          mov     eax, [edi+4]
.text:3009C4F0          mov     esi, [eax+24h]
.text:3009C4F3          test    esi, esi
.text:3009C4F5          jnz     loc_301CA9A0
```

This results in deleting o1 by the following functions:

```
.text:301CA9A0 loc_301CA9A0:
.text:301CA9A0          mov     ecx, esi
.text:301CA9A2          call    sub_3004E279          //delete references contained in o1
.text:301CA9A7          push    esi
.text:301CA9A8          call    sub_300072ED          //lead to calling MsoFreePv()
.text:301CA9AD          jmp     loc_3009C4FB
```

At this point, the program resumes to loc_3009C4FB and tries to parse the atom. However, if this second atom has placeholderId out of the range (1,18h), the function returns and a reference to o1 remains in memory. This may lead to various crashes later when the program tries to access o1 again.

Exploitation

Once o1 is deleted, it is easy to allocate a new block with controlled data at its location. This can be achieved for example by inserting a SlideTextProp11Atom (opcode 4022 or 0FB6h) containing controlled data.

```
.text:301CAABB              cmp     ax, 0FB6h               //case SlideTextProp11Atom
.text:301CAABF              jnz     short loc_301CAB2B
...
.text:301CAAE6 loc_301CAAE6:
.text:301CAAE6              push    dword ptr [ebx+4]
.text:301CAAE9              call    sub_30004D75            //allocate a new block
.text:301CAAEE              mov     ecx, [ebp+10h]
.text:301CAAF1              add     edi, 0D0h
.text:301CAAF7              mov     [edi], eax
.text:301CAAF9              mov     eax, [ebx+4]
.text:301CAAFC              push    eax
.text:301CAAFD              mov     [ebp+8], eax
.text:301CAB00              call    sub_3009D869            //fill it with controlled data
.text:301CAB05              mov     ecx, [ebp+8]
.text:301CAB08              mov     edi, [edi]
.text:301CAB0A              mov     esi, eax
.text:301CAB0C              mov     eax, ecx
.text:301CAB0E              shr     ecx, 2
.text:301CAB11              rep movsd                       //memcpy to the new block
```

This is not satisfying enough as o1 does not embed any function or object pointers. Consequently, even if fully controlled, this object does not lead to dereferencing an arbitrary pointer or calling a virtual function.

Theoretically, the following sequence of actions might be successful:
-allocate o1
-delete o1
-allocate a second object o2 where o1 was
-delete o2
-allocate a third block with controlled data where o1 was

If successful, this sequence of actions should lead to controlling data in o2 which may turn into controlling function or object pointers. Unfortunately when the program deletes o1, it performs several checks on the second dword pointed contained in o1. This is done in sub_3004E279, where ECX points to o1:

```
.text:3004E279 sub_3004E279
.text:3004E279              mov     eax, [ecx+4]            //dereference a pointer to eax
.text:3004E27C              push    ecx
.text:3004E27D              mov     ecx, [eax+10h]          //dereference a second pointer to ecx
.text:3004E280              add     ecx, 68h
.text:3004E283              call    sub_3004E289
.text:3004E288              retn
```

Then the code in sub_3004E289 compares [ECX] with 0. If [ECX] = 0, the function returns and o1 is deleted. Otherwise the program loops and tries to find a pointer that equals EAX. During our tests, we did not find an object o2 allocated such that:

p1 = [o2+4],
p2 = [p1+10h]
and [p2 + 68h] = 0

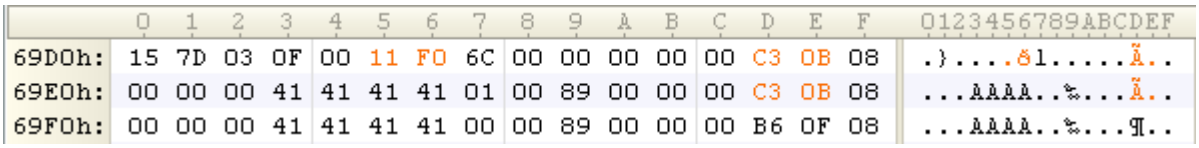
Our tests resulted most of the time in crashing while trying to dereference incorrect values. The provided proof of concept first allocates o1, deletes it and then allocates a text object at its location. When the object is accessed again, a crash occurs at 0x3004E27D while trying to read from an invalid location:

```
.text:3004E27D              mov     ecx, [eax+10h]      //eax controlled
```

Detection

An OEPlaceholderAtom atom normally belongs to a container named msofbtClientData (opcode 0xF011) which can be found in the Powerpoint Document stream.

To detect a malicious PPT file, look for OEPlaceholderAtom atoms in a msofbtClientData containers. Malicious documents (as shown on Figure 1) must have at least one OEPlaceholderAtom atom with placeholderId < 1 or placeholderId > 18h.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
69D0h:	15	7D	03	0F	00	11	F0	6C	00	00	00	00	00	C3	0B	08	.}....ðl.....Ã..
69E0h:	00	00	00	41	41	41	41	01	00	89	00	00	00	C3	0B	08	...AAAA..‰...Ã..
69F0h:	00	00	00	41	41	41	41	00	00	89	00	00	00	B6	0F	08	...AAAA..‰...¶..

Figure 1 – Malicious OEPlaceholderAtom

Figure 2 shows an example of a malicious atom. A msofbtClientData container begins at offset 0x69D3 and embeds at least two OEPlaceholderAtom atoms (offsets 0x69DB and 0x69EB). The first one is correct (placeholderId = 01, offset 0x69E7) while the second has placeholderId set to 0 (offset 0x69F7). Such document should then be considered malicious.

References

VUPEN/ADV-2010-0337:
http://www.vupen.com/english/advisories/2010/0337

MS10-004:
http://www.microsoft.com/technet/security/bulletin/ms10-004.mspx

Changelog

2010-02-12: Initial release