VUPEN security

VUPEN Security – Private Exploits & PoC Service

In-Depth Analysis of Microsoft Office PowerPoint Viewer TextCharsAtom Stack Overflow Vulnerability (MS10-004 / CVE-2010-0034)

Table of Contents

Introduction

A vulnerability exists in Microsoft PowerPoint Viewer, in the way it processes malformed PPT files. Exploitation of this vulnerability could lead to arbitrary code execution when opening a specially crafted PPT file.

Tested Versions

The vulnerability was analysed on Windows Vista SP1 with PowerPoint Viewer 2003 SP3 (PPVIEW.exe version 11.0.8305.0).

Fixed Versions

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

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 stack overflow vulnerability exists in the PowerPoint document viewer because of an invalid parsing of TextCharsAtom atoms (opcode 4000 or 0FA0h).

The program incorrectly processes the length of such records which leads to a stack overflow.

This occurs in sub_300FA0DA:

```
.text:300FA0DA          mov     eax, offset loc_30176718
.text:300FA0DF          call    __EH_prolog
.text:300FA0E4          sub     esp, 1CCh
.text:300FA0EA          mov     eax, ds:dword_301DB000
.text:300FA0EF          push    ebx
.text:300FA0F0          push    esi
.text:300FA0F1          mov     esi, ecx
.text:300FA0F3          movzx   ecx, [ebp+arg_6]          //ecx is the current atom
.text:300FA0F7          xor     ebx, ebx
.text:300FA0F9          mov     [ebp+var_10], eax
.text:300FA0FC          mov     eax, 0FDFh
.text:300FA101          inc     ebx
.text:300FA102          cmp     ecx, eax                  //beginning of the switch case
.text:300FA104          push    edi
.text:300FA105          jg      loc_300FA56E
.text:300FA10B          jz      loc_300FA4B0
.text:300FA111          sub     ecx, 0FA0h
.text:300FA117          jz      loc_300FA45B              //case TextCharsAtom (0FA0h)
```

When the program encounters such atom, it first put its size in ebx and tests if it is greater than 0:

```
.text:300FA45B loc_300FA45B:
.text:300FA45B              mov     ebx, [ebp+arg_8]            //ebx = size
.text:300FA45E              jmp     short loc_300FA4A7
…
.text:300FA4A7 loc_300FA4A7:
.text:300FA4A7              test    ebx, ebx
.text:300FA4A9              ja      short loc_300FA460          //jump if size > 0
.text:300FA4AB              jmp     loc_300FA2A2
```

It then compares this size with a signed value FEh, and eventually copies data from the file to a stack buffer:

```
.text:300FA460 loc_300FA460:
.text:300FA460              mov     edi, 0FEh
.text:300FA465              cmp     ebx, edi                    //edi = ebx if ebx < edi
.text:300FA467              jge     short loc_300FA46B
.text:300FA469              mov     edi, ebx
.text:300FA46B
.text:300FA46B loc_300FA46B:
.text:300FA46B              push    edi
.text:300FA46C              lea     eax, [ebp+var_194]          //stack buffer
.text:300FA472              push    eax
.text:300FA473              mov     ecx, esi
.text:300FA475              sub     ebx, edi
.text:300FA477              call    sub_300F17E0                //read and copy data in OLE32.dll
```

The problem lies in the previous comparison. If a TextCharsAtom has a size lower than 0, then the program tries to read up to size bytes from the file. This leads to a stack overflow which can then be turned to execute arbitrary code.

Exploitation

Exploitation of such vulnerability is easy when ASLR is not activated as this program is not compiled with a Safe-SEH option.

The provided exploit consists in inserting a huge TextCharsAtom in a document (offset 0x3FBD in the provided file, size = 0x80000005) which then triggers the exploitable condition.

Due to the SEH redirection, execution flow is next redirected to 0x30091DDC in PPVIEW.EXE:

```
.text:30091DDC              pop     edi
.text:30091DDD              pop     esi
.text:30091DDE              retn    4                           //return to the stack buffer
```

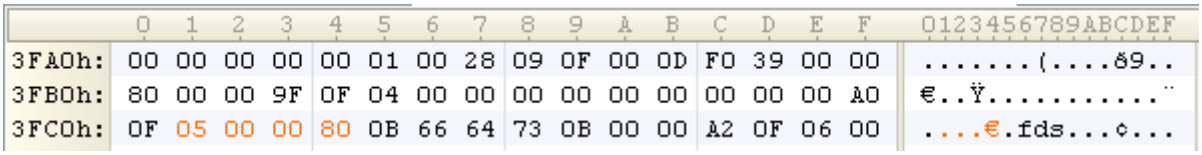
which allows data from the file to be executed. Note that this exploits only succeeds on Windows Vista, since the function which copies data in OLE32.dll behaves differently on Windows XP. Due to a call to IsBadHugeWritePrt(), an error is returned which prevents this vulnerability to be exploited on this system.

Detection

Attempts to trigger this vulnerability can be detected by inspecting TextCharsAtom atoms (opcode 4000 or 0FA0h) embedded in a PowerPoint document.

If at least one TextCharsAtom has a size larger than 0x7FFFFFFF, consider the file malicious.

Figure 1 illustrates a malicious atom:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
3FA0h:	00	00	00	00	00	01	00	28	09	0F	00	0D	F0	39	00	00	.....(....ð9..
3FB0h:	80	00	00	9F	0F	04	00	00	00	00	00	00	00	00	00	A0	€..Ÿ..........¨
3FC0h:	0F	05	00	00	80	0B	66	64	73	0B	00	00	A2	0F	06	00	....€.fds...¢...

Figure 1 – Malicious TextCharsAtom

On Figure 1, a malicious TextCharsAtom (offset 0x3FBD) is embedded in an msofbtClientTextbox container (opcode F00Dh, at offset 0x3FA9). As one can see, the TextCharsAtom has its size greater than 0x7FFFFFFF, so the document is likely to be 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

2010-02-17: Initial release