



In-Depth Analysis of Microsoft Office PowerPoint Viewer TextBytesAtom Stack Overflow Vulnerability (MS10-004 / CVE-2010-0033)

Table of Contents

Introduction	2
Tested Versions	2
Fixed Versions	2
Technical Details	2
Exploitation	3
Detection	3
References	4

This Binary Analysis and Exploit or Proof-of-concept codes are under the copyrights of VUPEN Security. Copying or reproducing the document, exploit or proof-of-concept codes is prohibited, unless such reproduction or redistribution is permitted by the VUPEN Exploits & PoCs Service license agreement. Use of the Binary Analysis, Exploit or Proof-of-concept codes is subject to the VUPEN Exploits & PoCs Service license terms.

Introduction

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

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.

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 "TextBytesAtom" atoms (opcode 4008 or 0FA8h). 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
.text:300FA11D      dec     ecx
.text:300FA11E      jz     loc_300FA317
.text:300FA124      dec     ecx
.text:300FA125      jz     loc_300FA2BD
.text:300FA12B      push   6
.text:300FA12D      pop     edi
.text:300FA12E      sub     ecx, edi
.text:300FA130      jz     loc_300FA23A             //case TextBytesAtom (0FA8h)
```

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

```
.text:300FA23A loc_300FA23A:
.text:300FA23A      mov     ebx, [ebp+arg_8]           //ebx = size
.text:300FA23D      jmp     short loc_300FA29E
...
.text:300FA29E loc_300FA29E:
.text:300FA29E      test   ebx, ebx
.text:300FA2A0      ja     short loc_300FA23F        //jump if size > 0
```

It then compares this size with a signed value 7Fh, and eventually copies data from the file to a stack buffer:

```
.text:300FA23F loc_300FA23F:
.text:300FA23F      push   7Fh
.text:300FA241      pop    edi
.text:300FA242      cmp    ebx, edi
.text:300FA244      jge   short loc_300FA248
.text:300FA246      mov    edi, ebx                  //edi = ebx if ebx < edi
.text:300FA248
.text:300FA248 loc_300FA248:
.text:300FA248      push   edi
.text:300FA249      lea   eax, [ebp+var_90]         //stack buffer
.text:300FA24F      push   eax
.text:300FA250      mov    ecx, esi
.text:300FA252      sub    ebx, edi
.text:300FA254      call  sub_300F17E0              //read and copy data in OLE32.dll
```

The problem lies in the previous comparison. If a TextBytesAtom has a size lower than 0, then the program tries to read up to size bytes from the file. This leads to an exploitable stack overflow condition.

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 "TextBytesAtom" 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 "TextBytesAtom" atoms (opcode 4008 or 0FA8h) embedded in a PowerPoint document.

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

Figure 1 illustrates a malicious atom:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
3FA0h:	00	00	00	00	00	01	00	28	09	0F	00	0D	F0	39	00	00(.....89..															
3FB0h:	80	00	00	9F	0F	04	00	00	00	00	00	00	00	00	A8	€..ÿ....."																
3FC0h:	0F	05	00	00	80	0B	66	64	73	0B	00	00	A2	0F	06	00€.fds...c...															

Figure 1 – Malicious TextBytesAtom

On Figure 1, a malicious TextBytesAtom (offset 0x3FBD) is embedded in an msosftClientTextbox container (opcode F00Dh, at offset 0x3FA9).

As one can see, the TextBytesAtom 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.msp>

Changelog

2010-02-17: Initial release