VUPEN security

VUPEN Security – Binary Analysis & Exploits Service

In-Depth Analysis of Microsoft VBA and Office Stack Memory Corruption Vulnerability (MS10-031 / CVE-2010-0815)

Table of Contents

Introduction

A vulnerability exists in Microsoft Visual Basic for Applications (VBA) when searching for ActiveX controls, which could potentially allow attackers to compromise a vulnerable system.

Tested Versions

The vulnerability was analyzed on Windows XP SP3 with Microsoft Office PowerPoint XP SP3 (VBE.dll version 6.5.10.24).

Fixed Versions

The vulnerability was fixed with the MS10-031 security update.

Technical Details

Visual Basic for Applications (VBA) is an implementation of Microsoft's event-driven programming language Visual Basic 6, and associated integrated development environment (IDE), which is built into most Microsoft Office applications. VBA enables developers to build user defined functions, automate processes, and access Win32 and other low level functionality through DLLs.

Microsoft Office documents can contain objects that link to external resources (e.g. linked audio, linked video, embedded and linked OLE objects, and hyperlinks).

When embedding an external object into a PowerPoint document, an "ExObjListContainer" container is created. This type of container record specifies a list of external objects in the document.

Each child container (except for the very first one) specifies an "ExControlContainer" i.e. an external object.

In the example that follows we used an object with a "progID" (Program ID) set to the following string:

- MDACVer.Version

When such a document is opened, the code flow goes to the following code:

```
;
; In function starting at 0x30272BDA – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                              Comments
30272BEE    MOV EAX,DWORD PTR DS:[ESI+30]        ; eax = "MDACVer.Version"
30272BF1    PUSH ECX                             ; /Arg2 => GUID output
30272BF2    PUSH EAX                             ; |Arg1, progID
30272BF3    CALL 301226B4                        ; \POWERPNT.301226B4
```

In the above code snippet, the first argument of the function is the "progID" of the external object while the second argument is an output parameter.

When the function returns, the second parameter is filled with the GUID (class ID / CLSID), in binary, corresponding to the "progID".

Confirmation is brought by the inner code of the function which is merely a wrapper around the "CLSIDFromProgID()" function in the "Ole32.dll" module:

```
;
; In function starting at 0x301226B4 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                                 Comments
301226DD    PUSH DWORD PTR SS:[ARG.2]               ; GUID output buffer
;[…]
301226E1    PUSH DWORD PTR SS:[LOCAL.131]           ; string (MDACVER.Version)
;[…]
301226E9    CALL DWORD PTR DS:[<&OLE32.CLSIDFromProgID>]
```

As its name implies, this function translates a "progID" to its corresponding CLSID. In our test sample, the "MDACVer.Version" progID is translated to the following CLSID:

- 54AF9350-1923-11D3-9CA4-00C04F72C514

This is confirmed in the registry:

Hence the second parameter is filled with the following CLSID, in binary:

```
CPU Dump
Address     Hex dump
00138A80    50 93 AF 54|23 19 D3 11|9C A4 00 C0|4F 72 C5 14|
```

After converting the "progID" to its CLSID in binary format, the code flow reaches this portion of code :

```
;
; In function starting at 0x30272BDA – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
30272D29    LEA EDX,[EBP-38]
30272D31    PUSH DWORD PTR SS:[EBP-84]       ; PPT Version (string)
30272D37    PUSH EDX                         ; CLSID (GUID) of external object (HEX)
30272D38    PUSH EAX
30272D39    CALL DWORD PTR DS:[ECX+14]       ; 0x65117F64 (VBE6.dll)
```

The 2nd argument of the function is the CLSID that was previously obtained.

The 3rd argument is a constant string related to the application: in our example it is the major version of PowerPoint:

- "PPT10.0"

The above call leads to the following code:

```
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
65117FBB    PUSH EAX                         ; /pResult => OFFSET LOCAL.50
65117FBC    PUSH VBE6.65078960               ; |SubKey = "CLSID"
65117FC1    PUSH 80000000                    ; |hKey = HKEY_CLASSES_ROOT
65117FC6    MOV EBX,80040150                 ; |
65117FCB    CALL ESI                         ; \ADVAPI32.RegOpenKeyA
65117FCD    TEST EAX,EAX
65117FCF    JNE 65118164
65117FD5    PUSH 27                          ; /Arg3 = 27
65117FD7    LEA EAX,[LOCAL.11]               ; |
65117FDA    PUSH EAX                         ; |Arg2 => unicode String output
65117FDB    PUSH DWORD PTR SS:[LOCAL.53]     ; |Arg1 => [ARG.2], CLSID (hex)
65117FDE    CALL 6508D519                    ; GUIDToString->WideCharToMultiByte
```

The above code starts by opening the "HKEY_CLASSES_ROOT\CLSID" key. If it succeeds, it converts the CLSID (obtained earlier) to a string, and then this string is converted to Unicode, as shown in the memory dump below:

```
CPU Dump
Address     Hex dump                                        ASCII
001389D8    7B 35 34 41|46 39 33 35|30 2D 31 39|32 33 2D 31| {54AF9350-1923-1
001389E8    31 44 33 2D|39 43 41 34|2D 30 30 43|30 34 46 37| 1D3-9CA4-00C04F7
001389F8    32 43 35 31|34 7D 00 00|                        2C514}..
```

It then opens the key that has the same name as the CLSID, and which is a sub-key of "HKEY_CLASSES_ROOT\CLSID":

- "HKEY_CLASSES_ROOT\CLSID\{54AF9350-1923-11D3-9CA4-00C04F72C514}"

```
CPU Disasm
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
65117FE3    LEA EAX,[LOCAL.49]
65117FE6    PUSH EAX                         ; pResult
65117FE7    LEA EAX,[LOCAL.11]
65117FEA    PUSH EAX                         ; SubKey (CLSID)
65117FEB    PUSH DWORD PTR SS:[LOCAL.50]     ; hKey
65117FEE    CALL ESI                         ; RegOpenKey => open CLSID Key
65117FF0    TEST EAX,EAX
65117FF2    JNE 6511815B
65117FF8    LEA EAX,[LOCAL.51]
65117FFB    PUSH EAX                         ; pResult
65117FFC    PUSH VBE6.65118180               ; ASCII "TypeLib"
65118001    PUSH DWORD PTR SS:[LOCAL.49]     ; hKey
65118004    CALL ESI
```

Once the CLSID key is opened, the above code tries to open the "TypeLib" key, which is a sub-key of the previously named CLSID key:

- Key = "HKEY_CLASSES_ROOT\CLSID\{54AF9350-1923-11D3-9CA4-00C04F72C514}"
  - SubKey = "TypeLib"

Once the "TypeLib" key is opened, the code tries to get the value attributed to the key:

```
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
6511800E    LEA EAX,[LOCAL.48]
65118011    PUSH EAX                         ; /pValueLen => (output length)
65118012    LEA EAX,[LOCAL.11]
65118015    PUSH EAX                         ; |Value => (output)
65118016    PUSH 0                           ; |SubKey = NULL
65118018    PUSH DWORD PTR SS:[LOCAL.51]     ; |hKey => (TypeLib)
;[…]
65118022    CALL DWORD PTR DS:[<&ADVAPI32.RegQueryValueA>]
```

In our example, the output buffer is filled with the following string:

```
CPU Dump
Address   Hex dump                                  ASCII
001389D8  7B 35 34 41|46 39 33 34|33 2D 31 39|32 33 2D 31| {54AF9343-1923-1
001389E8  31 44 33 2D|39 43 41 34|2D 30 30 43|30 34 46 37| 1D3-9CA4-00C04F7
001389F8  32 43 35 31|34 7D 00 00|                         2C514}..
```

If the "TypeLib" value has been successfully retrieved, the code converts the "TypeLib" string value to Unicode and then obtains the CLSID from the string (i.e. it converts the "TypeLib" string value to bin/hex):

```
CPU Disasm
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
65118055    LEA ECX,[LOCAL.11]
65118058    PUSH ECX                         ; /Arg2 => TypeLib ASCII string
65118059    PUSH EAX                         ; |Arg1, output buffer Unicode TypeLib
6511805A    CALL Mbcs2Unicode                ; \convert ASCII to Unicode
6511805F    LEA ECX,[LOCAL.47]
65118062    PUSH ECX                         ; (output) Binary TypeLib GUID
65118063    PUSH EAX                         ; (input) Unicode string TypeLib GUID
65118064    CALL DWORD PTR DS:[<&ole32.CLSIDFromString>]
```

Once done, the code opens the "Version" sub-key of the current CLSID:

```
CPU Disasm
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
6511806A    LEA EAX,[LOCAL.52]
6511806D    PUSH EAX                         ; pResult
6511806E    PUSH VBE6.650D8CF4               ; ASCII "Version"
65118073    PUSH DWORD PTR SS:[LOCAL.49]     ; hKey
65118076    CALL ESI                         ; RegOpenKey (version subkey)
```

It then gets the value if this Version key:

```
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
65118080    LEA EAX,[LOCAL.48]
65118083    PUSH EAX                         ; /pValueLen => OFFSET LOCAL.48
65118084    LEA EAX,[LOCAL.43]               ; |
65118087    PUSH EAX                         ; |Value => OFFSET LOCAL.43
65118088    PUSH 0                           ; |SubKey = NULL
6511808A    PUSH DWORD PTR SS:[LOCAL.52]     ; |hKey => [LOCAL.52]
;[…]
65118094    CALL DWORD PTR DS:[<&ADVAPI32.RegQueryValueA>]
```

Once the version string has been retrieved, the code checks the string by doing the following:

- [0x651180A2] Checks if the very first character is a dot (“.”). If it is not a dot, it continues:
- [0x651180AB / 0x651180AC] Searches for a dot (“.”) in the bytes following the first character without taking care of the size of string
- [0x651180B5] Once a dot is found, the code replaces it by a NULL character.

```
;
; In function starting at 0x65117F64 – PowerPoint.exe module
; Module Base: 0x30000000 – Module Code Base: 0x30001000
;
Address     Command                          Comments
651180A2    CMP BYTE PTR SS:[LOCAL.43],2E    ; first character == '.' ?
651180A6    LEA ESI,[LOCAL.43]
651180A9    JE SHORT 651180B1                ; skip loop if true
651180AB    /INC ESI                         ; search for '.'
651180AC    |CMP BYTE PTR DS:[ESI],2E
651180AF    \JNE SHORT 651180AB
;[…]
651180B5    MOV BYTE PTR DS:[ESI],0          ; put '\0' if '.' found
```

The vulnerability lies in the code above, precisely in the loop (0x651180AB/0x651180AC).

As the stop condition of this loop is the presence of a dot, if there is no dot in the version string, the loop will go past the normal end of the string simply because there is no check for a NULL byte in the loop.

As soon as a 0x2E byte present on the stack is found, this byte will be overwritten with a NULL byte, potentially leading to arbitrary code execution.

Exploitation

Exploitation of this vulnerability to execute arbitrary code is unlikely to be achieved although it remains theoretically possible if, for any reason, the following conditions are met:

- The external object used in the Office document has a valid ‘progID’ on the target machine.
  - The progID must match a valid CLSID.
- The CLSID string has a “TypeLib” sub-key.
  - the “TypeLib” sub-key must have a value

- The CLSID key has a "Version" sub-key.
  - The "Version" key must have a value.
  - This value string must not contain any dot.

If the all above conditions are met, the first 0x2E byte encountered on the stack (located in and after the stack frame where the vulnerability is triggered) will be replaced by a NULL byte.

Under these conditions, it is unlikely that an attacker can control a pointer that has a 0x2E byte, or a return address with this particular byte.

Although the attacker might not control anything directly, the overwriting of a byte is still a risk.

In our example with the 'MDACVer.Version' object the version string of this CLSID is "2,81,1117,0" (note that the delimiters are commas and not dots). The parsing of the string to replace the first dot by a NULL byte starts here:

```
CPU Stack
Address     Value        ASCII  Comments
00138958    31382C32     2,81   ; start searching for a '.'
0013895C    3131312C     ,111
00138960    00302C37     7,0.
00138964    00310031     1.1.
00138968    002C0037     7.,.
0013896C    00000030     0...
;[…]
001389D8    4134357B     {54A   ; MDACVer.Version Typelib value string
001389DC    34333946     F934
001389E0    39312D33     3-19
001389E4    312D3332     23-1
001389E8    2D334431     1D3-
001389EC    34414339     9CA4
001389F0    4330302D     -00C
001389F4    37463430     04F7
001389F8    31354332     2C51
001389FC    00007D34     4}..
00138A00    2A9D24EE     $*
00138A04    00138AB8     .
00138A08    30272D3C     <-'0   ;RETURN to POWERPNT.30272D3C
;[…]
00138A40    00500050     P.P.   ; "PPT10.0" string
00138A44    00310054     T.1.
00138A48    002E0030     0...   ; 0x2E = '.' => first dot encountered in the stack
00138A4C    00000030     0...
```

In the above example, the first 0x2E character found on the stack is the one from the "PPT10.0" string, which is far from the start of the version string. In this example, this dot character is not even in the same stack frame (see return address at 0x00138A08).

Note that, although the vulnerability is triggered, the overwriting of the first 0x2E byte found in the stack, in our tests, always overwrites the '.' of the "PPT10.0" string which will NOT cause PowerPoint to crash (silent trigger).

Detection

It is not possible to reliably detect attempts to exploit this vulnerability without reading the registry hive of the target computer where the document is opened.

PowerPoint document

Parse binary PowerPoint documents using the official specifications [MS-PPT].

❖ Common:

This is the common detection base for the 'Base Detection' and the 'Full Detection'

- Look for a record named **RT_Document** (Type: 0x03E8 - Specifies a DocumentContainer record).
- Look in the children of the above record for a record named **RT_ExternalObjectList** (Type: 0x0409 - Specifies an ExObjListContainer.).
- The above record may have one or more records named **RT_ExternalOleControl** (Type: 0x0FEE - Specifies an ExControlContainer.).

❖ Base Detection:

The base detection detects a document that would try to exploit the flaw. It cannot detect if the flaw will be effectively triggered.

- One of the children of the RT_ExternalOleControl should be a record named **RT_ExternalOleObjectAtom** (Type: 0x0FC3 - Specifies an ExOleObjAtom).
- If you find a RT_ExternalOleControl record, look for the value of its '**persistIdRef**' field.
- If the value of the 'persistIdRef' field is set to **0**, mark the document as malicious.

❖ Full Detection (local):

The full detection can be applied only to the local computer that received a potentially dangerous PowerPoint document.

To apply the full detection, first you must apply the base detection. If the base detection has marked the document as malicious, you may continue:

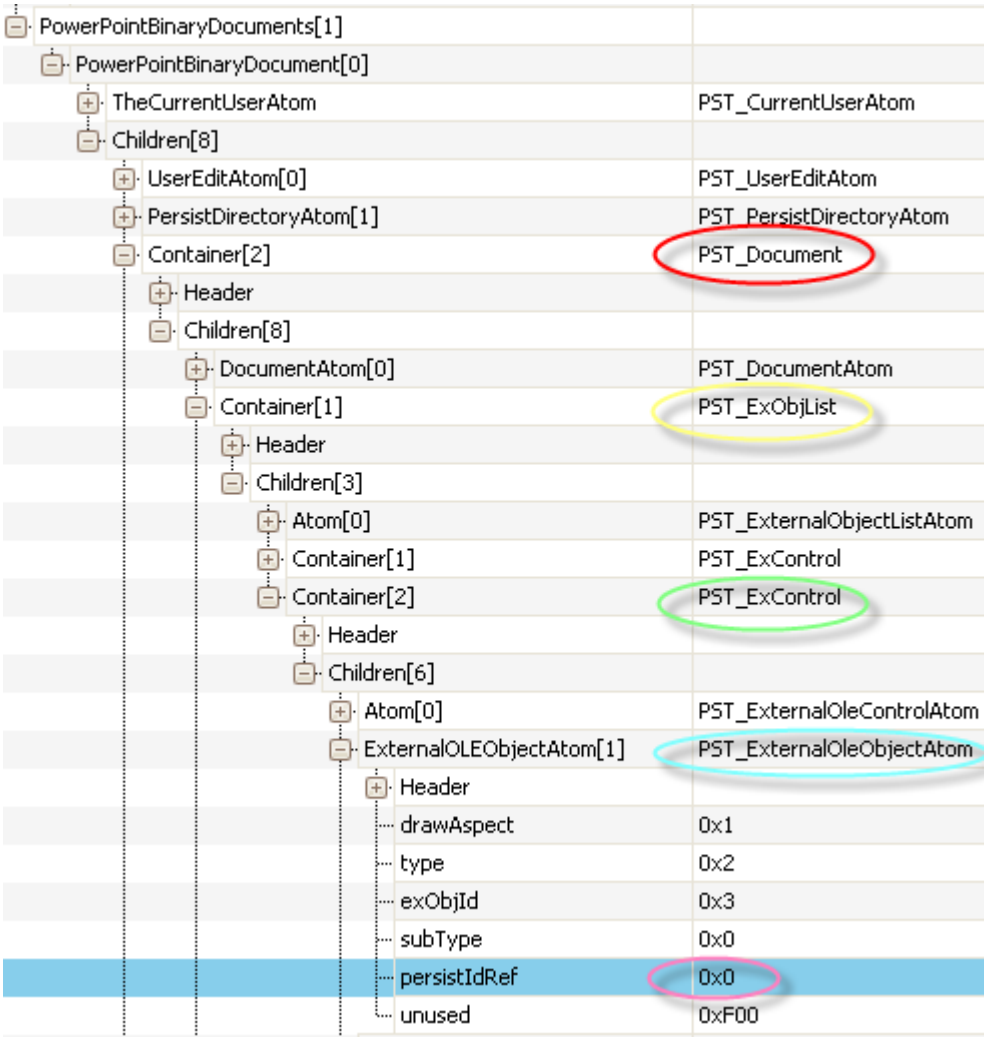
- Parse the RT_ExternalOleControl to find all of its children. For each child which is a **RT_CString** (Type: 0x0FBA), you should:
- Use the string as a 'ProgID' or an 'AppID'.
- Check the corresponding class ID (CLSID) in the registry, using the **CLSIDFromProgID()** function available in the OLE32 library.
- Open the corresponding key under the "HKEY_CLASSES_ROOT\CLSID" key in the registry.
- If the key has a "Version" sub key.
    - Get the value of the "Version" key. If the "Version" string value has no dot, the document is malicious.

Example 1 (Base Detection)

In our example, the PowerPoint document has a **RT_Document** record (circled in red).

This record has 8 children form which a **RT_ExternalObjectList** (circled in yellow). The previous record has 3 children. The child at index 2 is a **RT_ExternalOleControl** (circled in green). The previous container has 6 children from which (at index 1) is a **RT_ExternalOleObjectAtom** (circled in blue).

The '**persistIdRef**' field, circled in pink, is set to 0. Therefore, this document can be marked as malicious.



Example 2 (Full Detection)

From the base detection, we parse all children of the RT_ExternalOleControl record, searching for RT_CString records. In the capture shown here, two RT_CString records can be seen.

The second one (index 3) has a Data field at offset 0x1B34 in the file (row highlighted in blue):

Container[2]	PST_ExControl	0x00001ae0
Header		0x00001ae0
Children[6]		0x00001ae8
Atom[0]	PST_ExternalOleControlAtom	0x00001ae8
ExternalOLEObjectAtom[1]	PST_ExternalOleObjectAtom	0x00001af4
Header		0x00001af4
drawAspect	0x1	0x00001afc
type	0x2	0x00001b00
exObjId	0x3	0x00001b04
subType	0x0	0x00001b08
persistIdRef	0x2	0x00001b0c
unused	0xF00	0x00001b10
Atom[2]	PST_CString	0x00001b14
Atom[3]	PST_CString	0x00001b2c
Header		0x00001b2c
Version	0x0	0x00001b2c
Instance	0x2	0x00001b2c
Type	0xFBA	0x00001b2e
Length	0x1E	0x00001b30
Data	4D 00 44 00 41 00 43 00 56 ...	0x00001b34

The data at 0x1B34 in the PowerPoint file reads "MDACVer.Version" as shown in the capture below:

```
00001B20   65 00 65 00 43 00 74 00 72 00 6C 00 20 00 BA 0F   e.e.C.t.r.l. .°.
00001B30   1E 00 00 00 4D 00 44 00 41 00 43 00 56 00 65 00   ....M.D.A.C.V.e.
00001B40   72 00 2E 00 56 00 65 00 72 00 73 00 69 00 6F 00   r...V.e.r.s.i.o.
00001B50   6E 00 01 00 41 41 06 00 00 00 31 00 00 00 00 00   n...AA....1.....
```

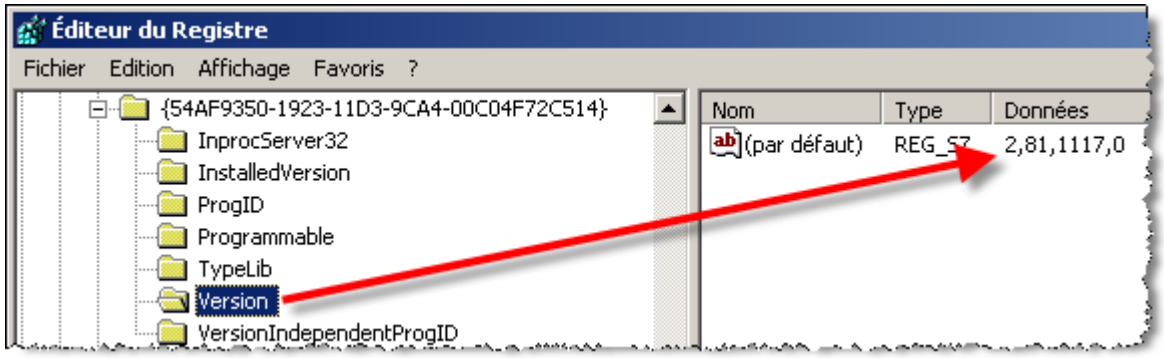
Open the local registry and use the **CLSIDFromProgID()** from OLE32 library (or parse all the keys). This returns the following CLSID:

- 54AF9350-1923-11D3-9CA4-00C04F72C514

Open the following key by prefixing the CLSID with "HKEY_CLASSES_ROOT\CLSID\" (don't forget brackets around the CLSID):

- HKEY_CLASSES_ROOT\CLSID\{54AF9350-1923-11D3-9CA4-00C04F72C514}

The CLSID has a "Version" sub key. The sub key value is "2,81,1117,0" which does not contain any dot:



The PowerPoint document can be marked as malicious.

References

VUPEN/ADV-2010-1121:
http://www.vupen.com/english/advisories/2010/1121

Ms10-031:
http://www.microsoft.com/technet/security/bulletin/ms10-031.mspx

[MS-PPT]: PowerPoint Binary File Format (.ppt) Structure
http://msdn.microsoft.com/en-us/library/cc313106.aspx

[MSDN1] CLSIDFromProgID Function
http://msdn.microsoft.com/en-us/library/ms688386(v=VS.85).aspx

Changelog

2010-05-17: Initial release