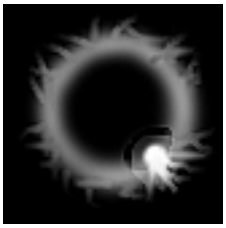


Zeroday Emergency Response Team (ZERT)

Analysis of CVE-2006-4868 and Patch Description

by Michael Hale Ligh with thanks to (member list).
2006-10-04

Synopsis: This document provides a brief, quasi-technical analysis of CVE-2006-4868. The purpose is to disseminate information on the particular vulnerability and to assist with understanding of how the ZERT patch was designed to mitigate the flaw.

Introduction

The vulnerability in Microsoft's Vector Graphics Rendering Engine (vgx.dll) exists due to an overzealous for() loop that copies data from a large, dynamically allocated buffer into an inadequate, fixed-size buffer on the stack. The data being copied in this routine is user-supplied as a Vector Markup Language (VML) "fill method" attribute. Legitimate values for the attribute include "none", "any", "linear," and "sigma.", [Specification draft for VML]. A vulnerable version of the library will copy the user-supplied string without checking its size, allowing a malicious document containing an overly-long fill method string to cause data to be written outside of the destination buffer's boundaries.

The ZERT patch for this vulnerability adds a check to the code before it can begin execution of the described loop. If the length of the user-supplied fill method string is greater than 512 bytes (size of destination buffer), the loop is avoided by making a jump to the function's cleanup instructions, and subsequently returns null. The function could return null for several reasons; such as if the fill method string does not contain any characters. By handling the overly-long string condition in this manner, we can be relatively certain that nothing out of the ordinary will happen. After patching, the library function will react to overly-long fill method strings with the same behavior as it exhibits when the fill method string is null.

If the added conditional determines that the string is not greater than 512 bytes, it will allow the library function to enter the copy loop and proceed with normal execution of the program.

Technical Description

To further describe the vulnerability and patch mechanism, an affected version of the library was disassembled and reverse engineered. It was found that the code declares a class similar to the one below:

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 license.

```
class TOKENS {
  public:
    WCHAR *Ptok(void);
  private:
    LPWSTR szInput;        // pointer to input string on heap
    int nSize;             // length of input string (in WCHARs)
    int idxInput;          // index used within the for()loop
    WCHAR szOutput[256];   // output buffer for string
};
```

The length of the supplied fill method string is determined by counting the number of 2-byte values at the location pointed to by szInput. The number is saved in the nSize variable. As shown in the following pseudo-code, nSize is then used as the criteria for terminating the loop that copies values from the source buffer containing the user-supplied fill method string into the 512 byte szOutput buffer, which is stored on the stack.

The length-checking conditional inserted by the ZERT patch is commented, just above the for() loop. Please note that this code is meant to describe the behavior of the vulnerable function, it is not an exact duplicate of the original source code.

```
WCHAR *TOKENS::Ptok(void){
      register int idxCurr;

      if (szInput==NULL)
            return(NULL);

      /* Added by the ZERT patch
      if (nSize >= 256) {
            return(NULL);
      }*/

      for(idxCurr=0; idxInput < nSize && szInput[idxInput] != '\0';
      idxInput++)
      {
            if (szInput[idxInput] == ' '){
                  if (idxCurr){      // Encountered non-leading space
                        break;
                  }
                  else{              // Encountered leading space
                        continue;
                  }
            }
            szOutput[idxCurr]=szInput[idxInput];  // Copy the WCHAR
            idxCurr++;
      }
      if (idxCurr > 0){
            szOutput[idxCurr]='\0';   // NULL terminate
            return(szOutput);
      }
      return(NULL);
}
```

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 license.

It should now be clear how the ZERT patch fixes the vulnerability, from a high-level language perspective. To understand the fix at a lower level, please see the following disassembly views of the function containing the vulnerability. The samples were prepared using IDA Pro and selected instructions are commented with lines from the pseudo-source code as accurately as possible.

```
mov     edx, [ecx]
push    ebx
push    esi
xor     esi, esi
cmp     edx, esi
push    edi
jz      short loc_5DEDED3B      ; if (szInput==NULL)
mov     eax, [ecx+8]
cmp     eax, [ecx+4]
jge     short loc_5DEDED3B      ; if (idxInput >= nSize)
cmp     [edx+eax*2], si
jz      short loc_5DEDED3B
lea     eax, [ecx+0Ch]          ; szOutput
mov     edi, eax
top_for_loop:
  mov     edx, [ecx+8]
  mov     ebx, [ecx]
  mov     dx, [ebx+edx*2]
  test    dx, dx
  jz      short loc_5DEDED2F    ; if (szInput[idxInput]==’\0’)
  cmp     dx, 20h
  jnz     short loc_5DEDED1E    ; if (szInput[idxInput]!=0x20)
  test    esi, esi
  jg      short loc_5DEDED33    ; if (idxCurr)
  jmp     short loc_5DEDED24
loc_5DEDED1E:
  mov     [edi], dx             ; szOutput[idxCurr]= szInput[idxInput];
  inc     esi
  inc     edi                   ; idxCurr++;
  inc     edi
loc_5DEDED24:
  inc     dword ptr [ecx+8]     ; idxInput++;
  mov     edx, [ecx+8]
  cmp     edx, [ecx+4]
  jl      short top_for_loop    ; if (idxInput < nSize)
loc_5DEDED2F:
  test    esi, esi
  jle     short loc_5DEDED3B
loc_5DEDED33:
  and     word ptr [ecx+esi*2+0Ch], 0  ; szOutput[idxCurr]=0;
  jmp     short loc_5DEDED3D
loc_5DEDED3B:
  xor     eax, eax              ; return(NULL);
```

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 license.

The first 0x23 bytes of this function provide an accurate signature for locating it within the library, and is consistent across all 32-bit versions of the library available to the team for testing. In other words, the sequence of bytes occurs exactly once. During execution, the patch loads a copy of the vulnerable library into memory, locates the signature, and applies the patch code. It then writes the patched library to disk (patchedvgx.dll), un-registers the original DLL, and registers the new one. The original DLL is not modified, so the patch can be easily reversed by using the patch's rollback function or by manually re-registering the original. Patching vgx.dll for AMD64 was also supported; the technical details are the same as for the 32bits patch, using a different signature.

At this stage it should be noted that there was not enough space to apply the patch. ZERT volunteer Gil Dabah re-wrote the vulnerable function, removing several instructions in order to make room for the bounds check, while maintaining most of the funtionality (as described below). He compiled the new function using the Yasm assembler, and hard coded the compiled result into the binary.

The next disassembly shows the function code after applying the ZERT patch.

```
mov     edx, [ecx]
push    ebx
push    esi
xor     esi, esi
cmp     edx, esi
push    edi
jz      short loc_5DEDED38
cmp     dword ptr [ecx+4], 100h
jnb     short loc_5DEDED38        ; if (nSize >= 256)
mov     eax, [ecx+8]
cmp     eax, [ecx+4]
jge     short loc_5DEDED38
cmp     [edx+eax*2], si
jz      short loc_5DEDED38
lea     eax, [ecx+0Ch]
mov     edi, eax
top_for_loop:
  mov     edx, [ecx+8]
  mov     ebx, [ecx]
  mov     dx, [ebx+edx*2]
  test    dx, dx
  jz      short loc_5DEDED33
  cmp     dx, 20h
  jnz     short loc_5DEDED23
  jmp     short loc_5DEDED28
loc_5DEDED23:
  mov     [edi], dx
  inc     edi
  inc     edi
loc_5DEDED28:
  inc     dword ptr [ecx+8]
  mov     edx, [ecx+8]
  cmp     edx, [ecx+4]
  jl      short top_for_loop
```

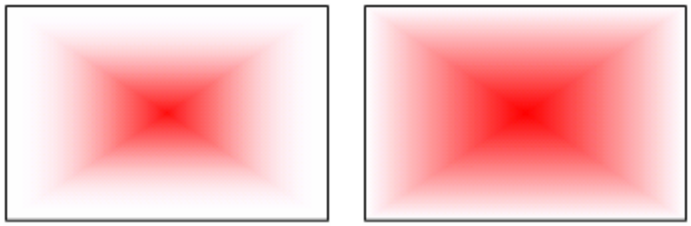
This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 license.

```
loc_5DEDED33:
  and     [edi], si
  jmp     short loc_5DEDED3A
loc_5DEDED38:
  xor     eax, eax
loc_5DEDED3A:
  pop     edi
  pop     esi
  pop     ebx
  retn
```

In order to make room for the length checking conditional (without extending the size of the original function, which would encroach upon instructions belonging to surrounding functions and cause unpredictable reactions), some modifications were made to the handling of space characters in the fill method string. This will produce a minor difference in the visual effect of VML objects rendered with the patched binary if the object has a fill method value consisting of more than one property. For example, the following VML is legitimate, and therefore could be used in some documents:

```
<v:rect fill method="linear sigma">
```

Below there are two screenshots provided to display the potential difference in appearance. One is produced with a fill method value of "linear sigma" (left) and the other has only "sigma" (right).



Microsoft's Official Patch

The official patch provided by Microsoft on September 26, 2006 mitigates the vulnerability by adding a size check within the for() loop. It only copies the first 254 WCHARs into the destination buffer. The critical section of this adjustment it shown below:

```
size_check:
  cmp     esi, 0FEh
  jnb     short skip_copy
  mov     [edi], dx
  inc     esi
  inc     edi
  inc     edi
```

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 license.

The ZERT patch was a joint work of the ZERT team, and specifically, from a technical stand-point it was created by Gil Dabah with help from Joe Stewart and Michael Hale Ligh (the author of this document). We would also like to thank Gadi Evron for reviewing.

This paper can be found at:
http://isotf.org/zert/papers/vml-details-20061004.pdf

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 license.