

# [NT] Windows Embedded Open Type (EOT) Font Heap Overflow

---

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2006-01/msg00038.html>

---

- *From:* SecuriTeam <[support@xxxxxxxxxxxxxxxx](mailto:support@xxxxxxxxxxxxxxxx)>
  - *Date:* 15 Jan 2006 19:18:54 +0200
- 

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>  
-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.  
<http://www.securiteam.com/maillinglist.html>

-----

## Windows Embedded Open Type (EOT) Font Heap Overflow

---

### SUMMARY

Microsoft Embedded OpenType Font Engine "t2embed" is "designed to create the ability of using specific fonts that might not be available on your local system".

A vulnerability in the way that Windows uncompresses Embedded Open Type fonts allow the author of a malicious web page to execute arbitrary code on the system of a user who visits the site, at the privilege level of that user.

### DETAILS

Vulnerable Systems:

- \* Windows ME
- \* Windows 98
- \* Windows NT
- \* Windows 2000
- \* Windows XP SP1
- \* Windows XP SP2
- \* Windows Server 2003

## [NT] Windows Embedded Open Type (EOT) Font Heap Overflow

\* Windows Server 2003SP1

Embedded Open Type fonts are referenced through the use of style data, as the following snippet illustrates:

```
@font-face {
font-family: Abysmal;
font-style: normal;
font-weight: normal;
src: url(evil.eot);
```

Although these fonts typically have .eot file extensions, it should be noted that any extension may be used in order to exploit this vulnerability.

A heap overflow vulnerability exists in T2EMBED.DLL, which Internet Explorer invokes to process EOT fonts. The data within an EOT file is compressed in Agfa MicroType Express format, which hosts an LZ-compressed stream that includes a 24-bit allocation size. This size + 1C00h is allocated within the function MTX\_LZCOMP\_UnPackMemory, but the resulting allocation size is not validated before data is copied into the block, allowing a malformed EOT file to cause an essentially arbitrary-length heap buffer overflow with binary data.

When Microsoft Embedded OpenType Font Engine is parsing a .EOT file, it allocates the heap memory block with size calculated from the .EOT file.

Note: all the "values" came from special case of a POC code

```
text:73C993EA call _MTX_BITIO_ReadValue@8 ; MTX_BITIO_ReadValue(x,x)
text:73C993EF mov [esi+FONT_STRUCT.mem_size], eax
```

The \_MTX\_BITIO\_ReadValue@8 (in this case) loops 18 times the following code:

```
text:73C9BBF8 give_input_bit: ; CODE XREF: MTX_BITIO_ReadValue(x,x)+24#j
text:73C9BBF8 push [esp+8+arg_0]
text:73C9BBFC shl esi, 1 ; (*)
text:73C9BBFE call _MTX_BITIO_input_bit@4 ; MTX_BITIO_input_bit(x)
text:73C9BC03 test ax, ax
text:73C9BC06 jz short max_io_ret_0
text:73C9BC08 or esi, 1 ; (*)
```

Note the bit operations on ESI register marked as (\*), first one simply multiplies the value stored in ESI register with  $2^1$ . The second one powers on the LSB (Less Significant Bit).

The \_MTX\_BITIO\_input\_bit@4 (in this case) does the following:

```
text:73C9BA53 mov eax, [esp+read_io] ; some struct ptr
text:73C9BA57 xor ecx, ecx ; ECX = 0
text:73C9BA59 mov cx, [eax+read_io.size?]; CX = 7 (counter)
```

## [NT] Windows Embedded Open Type (EOT) Font Heap Overflow

```
text:73C9BA5D test cx, cx ; is it 0?
text:73C9BA60 lea edx, [ecx-1] ; EDX = 6
text:73C9BA63 mov [eax+read_io.size?], dx ; read_io.size? - 1
text:73C9BA67 jnz short _MTX_BITIO_end ; until != 0 => take the jump
text:73C9BA69 mov ecx, [eax+read_io.licznik_do_fdata] ; ecx = 1
text:73C9BA6C mov edx, [eax] ; EDX = ptr to font data
text:73C9BA6E movzx dx, byte ptr [ecx+edx] ; give on byte from [ecx+edx]
text:73C9BA73 inc ecx ; ECX++
text:73C9BA74 cmp ecx, [eax+8] ; is ECX == 0x3B8 ?
text:73C9BA77 mov [eax+read_io.zwracana], dx ; set the return value
text:73C9BA7B mov [eax+read_io.licznik_do_fdata], ecx ; store the counter
text:73C9BA7E jle short loc_73C9BA91
```

...

```
text:73C9BA91 loc_73C9BA91: ; CODE XREF: MTX_BITIO_input_bit(x)+2B#j
text:73C9BA91 inc dword ptr [eax+10h]
text:73C9BA94 mov [eax+read_io.size?], 7 ; reset the io size to 7
text:73C9BA9A
text:73C9BA9A _MTX_BITIO_end: ; CODE XREF: MTX_BITIO_input_bit(x)+14#j
text:73C9BA9A shl [eax+read_io.zwracana], 1 ; multiple via 2 ->
{100,200,...}
text:73C9BA9E movzx eax, [eax+read_io.zwracana] ; eax = what we will
return
text:73C9BAA2 and ax, 100h ; and the bits, the return values are {0,100h}
text:73C9BAA6 retn 4
```

The read bytes (read from .EOT data file), of course the .EOT data was firstly xored by the engine, like the code here shows:

```
text:73C770AD loc_73C770AD: ; CODE XREF: sub_73C770A1+16#j
text:73C770AD xor dword ptr [edx+ecx*4], 50505050h
text:73C770B4 inc ecx
text:73C770B5 cmp ecx, eax
text:73C770B7 jb short loc_73C770AD
```

The end of \_MTX\_BITIO\_ReadValue@8 comes with following code:

```
text:73C9BC0B max_io_ret_0: ; CODE XREF: MTX_BITIO_ReadValue(x,x)+1E#j
text:73C9BC0B dec edi ; if AX was 0
text:73C9BC0C jnz short give_input_bit ; until != 0, repeat
text:73C9BC0E pop edi
text:73C9BC0F
text:73C9BC0F loc_73C9BC0F: ; CODE XREF: MTX_BITIO_ReadValue(x,x)+A#j
text:73C9BC0F mov eax, esi ; return value is computed from ESI!!!
text:73C9BC11 pop esi ;
text:73C9BC12 retn 8
```

The returned mem\_size is then used with creating the heap memory block:

```
text:73C993FB mov eax, [esi+FONT_STRUCT.mem_size]
text:73C99401 add eax, 1C00h
```

...

```
text:73C99410 push eax ; *our mem_size value + 1c00h*
text:73C99411 push dword ptr [esi+5Ch]
```

## [NT] Windows Embedded Open Type (EOT) Font Heap Overflow

text:73C99414 call \_MTX\_mem\_malloc@8 ; MTX\_mem\_malloc(x,x)

By looking the Microsoft Embedded OpenType Font Engine heap allocations we have:

K: 23 -> [\*] HeapAlloc(0x150000,0x8,0x2139(8505))=0x154820 end at: 0x156959 (\*)

K: 24 -> [\*] HeapAlloc(0x150000,0x8,0x2C(44))=0x156978 end at: 0x1569A4

K: 25 -> [\*] HeapAlloc(0x150000,0x8,0x246(582))=0x1569C0 end at: 0x156C06 (O)

K: 26 -> [\*] HeapAlloc(0x150000,0x8,0x1B48(6984))=0x156C28 end at: 0x158770 (C)

K: 27 -> [\*] HeapAlloc(0x150000,0x8,0x539(1337))=0x158790 end at: 0x158CC9 (\*)

First one is computed together with adding 1c00h value, the last one is plain mem\_size (1337). Such "faking" mem\_size value has bad influence of future EOT file decompression. While the data unpacking in the routine stored at 73C98E9Fh, the memory block marked as "O" is overwritten, and the block marked as "C" is corrupted. The block corruption is done here:

text:73C98FBB mov al, [eax+esi] ; read byte from other heap block

text:73C98FBE inc [ebp+var\_14]

text:73C98FC1 cmp [ebp+var\_1C], 0

text:73C98FC5 mov byte ptr [ebp+var\_C], al

text:73C98FC8 mov [ebx+ecx], al ; STORE !!!

This gives an heap overflow condition, illustrated here:

77F83905 8901 MOV DWORD PTR DS:[ECX],EAX ; EAX = 0D100C10h

77F83907 8948 04 MOV DWORD PTR DS:[EAX+4],ECX ; EBX = 0F100E10h

The vulnerability may lead to remote code execution when specially crafted file is being parsed, however the exploitation is hard due to the fact attacker doesn't control directly the data which will overwrite the heap block. However, it doesn't mean it can't be done.

CVE Information:

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0010>>

CVE-2006-0010

### ADDITIONAL INFORMATION

The information has been provided by <<mailto:Advisories@xxxxxxxx>> eEye and <<mailto:bania.piotr@xxxxxxxx>> Piotr Bania.

The original article can be found at:

<<http://eeye.com/html/research/advisories/AD20060110.html>>

<http://eeye.com/html/research/advisories/AD20060110.html>,

<<http://www.piotrbania.com/all/adv/MS06-002-adv.txt>>

<http://www.piotrbania.com/all/adv/MS06-002-adv.txt>

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@xxxxxxxxxxxxxxxx

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@xxxxxxxxxxxxxxxx

=====  
=====

**DISCLAIMER:**

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

- 
- Prev by Date: [\*\[NT\] Vulnerability in TNEF Decoding in Microsoft Outlook and Microsoft Exchange Allow Code Execution \(MS06-003\)\*](#)
  - Next by Date: [\*\[NEWS\] Cisco MARS Default Administrative Password\*](#)
  - Previous by thread: [\*\[NT\] Vulnerability in TNEF Decoding in Microsoft Outlook and Microsoft Exchange Allow Code Execution \(MS06-003\)\*](#)
  - Next by thread: [\*\[NEWS\] Cisco MARS Default Administrative Password\*](#)
  - Index(es):
    - ◆ [\*Date\*](#)
    - ◆ [\*Thread\*](#)