MOAUB

Abysssec Research

1) Advisory information

Title	: Trend Micro Internet Security Pro 2010 ActiveX extSetOwner Remote Code Execution
Version	: UfPBCtrl.DLL 17.50.0.1366 (XP SP3)
Analysis	: http://www.abysssec.com
Vendor	: http://www.trendmicro.com
Impact	: Critical
Contact	: shahin [at] abysssec.com , info [at] abysssec.com
Twitter	: @abysssec

2) Vulnerable version

Trend Micro Internet Security Pro 2010 and prior version may also be affected

3) Vulnerability information

Class

1- Uninitialized pointer code execution

Impact

Successfully exploiting this issue allows remote attackers to execute arbitrary code or cause denial-of-service conditions.

Remotely Exploitable

Yes

Locally Exploitable

Yes

4) Vulnerabilities detail

The extSetOwner function of UfPBCtrl.dll activeX takes a pointer as its only argument. The pointer is not initialized before using and allows the attacker to transfer the control of the program to arbitrary address that may contain shellcode.

Here is the vulnerable sub_51602160 function which has some role in processing the extSetOwner function and takes extSetOwner argument as one of its argument:

```
.text:51602160 sub_51602160    proc near             ; DATA XREF: .rdata:51605B08o
.text:51602160                                       ; .rdata:516064D0o
.text:51602160
.text:51602160 arg_0          = dword ptr  4
.text:51602160 arg_C          = dword ptr  10h
.text:51602160
.text:51602160                push    edi
.text:51602161                mov     edi, [esp+4+arg_C]
.text:51602165                test    edi, edi
.text:51602167                jnz     short loc_51602172
.text:51602169                mov     eax, 80004003h
.text:5160216E                pop     edi
.text:5160216F                retn    14h
.text:51602172 ; ---------------------------------------------------------------------------
.text:51602172
.text:51602172 loc_51602172:                         ; CODE XREF: sub_51602160+7j
.text:51602172                push    esi
.text:51602173                mov     esi, [esp+8+arg_0]
.text:51602177                mov     eax, [esi+0ACh]
.text:5160217D                test    eax, eax
.text:5160217F                jz      short loc_51602193
.text:51602181                mov     ecx, [eax]
.text:51602183                mov     edx, [ecx+8]
.text:51602186                push    eax
.text:51602187                call    edx
.text:51602189                mov     dword ptr [esi+0ACh], 0
.text:51602193
.text:51602193 loc_51602193:                         ; CODE XREF: sub_51602160+1Fj
.text:51602193                mov     [esi+0ACh], edi
.text:51602199                mov     eax, [edi]
.text:5160219B                mov     ecx, [eax+4]
.text:5160219E                push    edi
.text:5160219F                call    ecx
.text:516021A1                pop     esi
```

```
.text:516021A2                xor     eax, eax
.text:516021A4                pop     edi
.text:516021A5                retn    14h
.text:516021A5 sub_51602160    endp
```

As the above code demonstrate in the beginning of function the address which is sent to the function is stored in edi register and checked if zero or not. If the address is zero or null the function returns, but in case of not being a null address the conditional jump at address 51602167 is taken. If you follow the code you will notice that at address 51602199 the function store the content of edi register to eax and after incrementing eax by 4 and storing the contents of resulted pointer to ecx register, it call ecx without any previous check.

So if we redirect edi register to a valid pointer, it is possible to transfer the program flow to our arbitrary shellcode.

Exploit:

To exploit this vulnerability it is possible to use heap spray method to load our shellcode to memory and after allocating heap and initializing it to nop slides and shellcode the attacker can transfer the control to allocated heap by using proper pointer address to vulnerable extSetOwner function.

The point here is before using edi the function references its contents and after adding 4 to the pointer, there is another reference. So for heap range of example 0x0a0a0a0a an attacker should find a valid address in memory that contains value of 0x0a0a0a06 (0x0a0a0a0a – 0x4) ,so after referencing this address and adding 4 ,the call ecx instruction will call 0x0a0a0a0a that is the address in range of our allocated heap.

Here is an example that we have used a valid address containing our heap spray range address – 4 in mshtml.dll.

```
target.extSetOwner(unescape('%ua5de%u3da6'));             //mshtml.dll [0x3DA6A5DE] = 0A0A0A06
```