MOAUB

Abysssec Research

1) Advisory information

Title	: Novell iPrint Client Browser Plugin call-back-url stack overflow
Version	: iPrint Client plugin v5.42 (XP SP3)
Analysis	: http://www.abysssec.com
Vendor	: http://www.novell.com
Impact	: Critical
Contact	: shahin [at] abysssec.com , info [at] abysssec.com
Twitter	: @abysssec
CVE	: CVE-2010-1527

2) Vulnerable version

Novell iPrint Client 5.42
Novell iPrint Client 5.32
Novell iPrint Client 5.30
Novell iPrint Client 5.08
Novell iPrint Client 5.06
Novell iPrint Client 5.04

3) Vulnerability information

Class

1- Stack overflow

Impact

Successfully exploiting this issue allows remote attackers to execute arbitrary code or cause denial-of-service conditions on vulnerable version.
User interaction is required in order to open a malformed page.

Remotely Exploitable

Yes

Locally Exploitable

Yes

4) Vulnerabilities detail

By sending varius parameters to the active control by using <param> tags we can instruct the plugin client for varius tasks. The vulnerability occurs when using 'op-client-interface-version' as operation and 'url' as result-type parameters.

Because result-type is url there is a function that makes a longer string by combining "op-client-interface-version", the result-type string url and the string indicating version of the client plugin.

Here is part of the sub_10001710 function which is responsible for generating the result string.

```
.text:100017DA
.text:100017DA loc_100017DA:                         ; CODE XREF: sub_10001710+97j
.text:100017DA                 cmp     eax, 5
.text:100017DD                 jnz     short loc_10001819
.text:100017DF                 push    0
.text:100017E1                 lea     ecx, [esp+218h+var_210]
.text:100017E5                 call    sub_1003EEC2
.text:100017EA                 mov     edx, [esi+10Ch]
.text:100017F0                 push    eax
.text:100017F1                 lea     ecx, [esp+218h+var_20C]
.text:100017F5                 mov     eax, off_100620F8[edx*4]
.text:100017FC                 push    eax
.text:100017FD                 push    offset aSuccessSInterf ; "Success{%s}interfaceVersion:%s"
.text:10001802                 push    ecx             ; LPSTR
.text:10001803                 call    ds:wsprintfA
.text:10001809                 push    offset aIppclientpingR ; "ippClientPing=reply"
.text:1000180E                 push    esi
.text:1000180F                 call    sub_100076D0
.text:10001814                 add     esp, 18h
.text:10001817                 jmp     short loc_1000186D
.text:10001819 ; ---------------------------------------------------------------------------
.text:10001819
.text:10001819 loc_10001819:                         ; CODE XREF: sub_10001710+CDj
```

```
.text:10001819              cmp     eax, 6
.text:1000181C              jnz     short loc_1000186D
.text:1000181E              push    0
.text:10001820              lea     ecx, [esp+218h+var_210]
.text:10001824              call    sub_1003EEC2
.text:10001829              mov     edx, [esi+10Ch]
.text:1000182F              mov     ecx, [esi+1ACh]
.text:10001835              push    eax
.text:10001836              mov     eax, off_100620F8[edx*4]
.text:1000183D              lea     edx, [esp+218h+var_20C]
.text:10001841              push    eax
.text:10001842              push    ecx
.text:10001843              push    offset aS?successSInte ;
"%s?Success{%s}interfaceVersion:%s"
.text:10001848              push    edx           ; LPSTR
.text:10001849              call    ds:wsprintfA
```

As demonstrated in the above code in address loc_10001819, the function checks value of eax register against 6. There are also other part of the functions that this register is checked against 7,5,2 which in the moment this value represent the result-type. This code is set earlier in function sub_100020BA and, 6 is for result-type of url. So after checking this value the program transfer to our vulnerable part of the program.

The flaw here is wsprintfA copies 3 string with the format string "%s?Success{%s}interfaceVersion:%s" to a fixed length buffer and there is no bound checking on the final string copied to the buffer and of course the first parameter of format string is our url which indicated by call-back-url parameter. In case of long url it can cause a buffer overflow and simply overwrite EIP regisrer.

Here is a simple html example that can trigger this vulnerability:

```
<object ID='target' classid='clsid:36723f97-7aa0-11d4-8919-ff2d71d0d32c'>
<param name='operation' value='op-client-interface-version' />
<param name='result-type' value='url' />
<param name='call-back-url' value='AAAAAAAAAAAAAA…'  /> <!—1000*A long buffer  -->
</object>
```

Patch analysis:

In the patched version before using vulnerable wsprintfA a new block is added which by using repne scasb instructions checks length of the strings. Here is a simple implementation of repne scasb instruction for calculating length of a string:

```
         xor     ecx, ecx
         xor     eax,eax
         not     ecx
repne    scasb
         not     ecx
         dec     ecx
```

The above code return length of a string represented by edi register in ecx register.

The following code is the block added to the patched version, which calculate length of the strings and after addition of these lengths in case of greater than 511 the function will be returned and wsprintfA would not be executed.

```
.text:1000181D loc_1000181D:                ; CODE XREF: sub_10001710+CEj
.text:1000181D          cmp     eax, 6
.text:10001820          jnz     loc_10001915
.text:10001826          push    esi
.text:10001827          push    edi
.text:10001828          mov     ecx, 8
.text:1000182D          mov     esi, offset aS?successSInte ; "%s?Success{%s}interfaceVersion:%s"
.text:10001832          lea     edi, [esp+24Ch+var_230]
.text:10001836          xor     eax, eax
.text:10001838          rep movsd
.text:1000183A          movsw
.text:1000183C          lea     edi, [esp+24Ch+var_230]
.text:10001840          or      ecx, 0FFFFFFFFh
.text:10001843          repne scasb
.text:10001845          mov     edx, [ebx+10Ch]
.text:1000184B          push    eax
.text:1000184C          not     ecx
.text:1000184E          mov     ebp, off_100630F8[edx*4]
.text:10001855          dec     ecx
.text:10001856          mov     esi, ecx
.text:10001858          lea     ecx, [esp+250h+var_23C]
.text:1000185C          sub     esi, 6
.text:1000185F          call    sub_1003F392
.text:10001864          mov     edi, [ebx+1ACh]
.text:1000186A          mov     edx, eax
.text:1000186C          or      ecx, 0FFFFFFFFh
.text:1000186F          xor     eax, eax
.text:10001871          repne scasb
.text:10001873          not     ecx
.text:10001875          dec     ecx
.text:10001876          mov     edi, edx
.text:10001878          mov     eax, ecx
```

```
.text:1000187A          or      ecx, 0FFFFFFFFh
.text:1000187D          mov     [esp+24Ch+var_234], eax
.text:10001881          xor     eax, eax
.text:10001883          repne scasb
.text:10001885          not     ecx
.text:10001887          dec     ecx
.text:10001888          mov     [esp+24Ch+var_238], edx
.text:1000188C          mov     edx, ecx
.text:1000188E          mov     edi, ebp
.text:10001890          or      ecx, 0FFFFFFFFh
.text:10001893          add     edx, esi
.text:10001895          repne scasb
.text:10001897          mov     eax, [esp+24Ch+var_234]
.text:1000189B          pop     edi
.text:1000189C          not     ecx
.text:1000189E          dec     ecx
.text:1000189F          add     edx, eax
.text:100018A1          add     ecx, edx
.text:100018A3          pop     esi
.text:100018A4          cmp     ecx, 1FFh
.text:100018AA          jbe     short loc_100018DA
.text:100018AC          lea     ecx, [esp+244h+var_23C]
.text:100018B0          mov     [esp+244h+var_4], 0FFFFFFFFh
.text:100018BB          call    sub_1003F03C
.text:100018C0          pop     ebp
.text:100018C1          or      eax, 0FFFFFFFFh
.text:100018C4          pop     ebx
.text:100018C5          mov     ecx, [esp+23Ch+var_C]
.text:100018CC          mov     large fs:0, ecx
.text:100018D3          add     esp, 23Ch
.text:100018D9          retn
.text:100018DA ; ---------------------------------------------------------------------------
.text:100018DA
.text:100018DA loc_100018DA:                      ; CODE XREF: sub_10001710+19Aj
.text:100018DA          mov     ecx, [esp+244h+var_238]
.text:100018DE          mov     eax, [ebx+1ACh]
.text:100018E4          push    ecx
.text:100018E5          push    ebp
.text:100018E6          push    eax
.text:100018E7          lea     edx, [esp+250h+var_230]
.text:100018EB          lea     eax, [esp+250h+var_20C]
.text:100018EF          push    edx             ; LPCSTR
.text:100018F0          push    eax             ; LPSTR
.text:100018F1          call    ds:wsprintfA
.text:100018F7          lea     ecx, [esp+258h+var_20C]
.text:100018FB          push    200h
```

The patch calculate length of url, length of version of activeX string, length of operation and length of the format string – 6 by using repne scasb 4 times. The reason in the last one there is a -6 because the format string characters will be replaced. After that it sum all of these lengths in address 1000189F by using add instruction and store the final length in ecx register. A little later this value is checked against 1FFh(511).

Exploit:

For the purpose of exploitation it is simple to fine the exact location of overwritten EIP and take control of the program but because of possibility of using javascript and allocating dynamic memory it is better to use the more general heap spray method that is more relible.

The point here is using <script> tag before loading the activeX object.