Abysssec Research

1) Advisory information

Title	**: Novell Netware NWFTPD RMD/RNFR/DELE Argument Parsing Buffer overflow**
Version	**: NWFTPD.NLM 5.09.02 (Netware 6.5 – SP8)**
Analysis	**: http://www.abysssec.com**
Vendor	**: http://www.Novell.com**
Impact	**: Critical**
Contact	**: shahin [at] abysssec.com , info [at] abysssec.com**
Twitter	**: @abysssec**

2) Vulnerability Information

Class

1- Stack overflow

Impact

Successfully exploiting this issue allows remote attackers to execute arbitrary code in the context of vulnerable application or cause denial-of-service conditions in failed exploitation. For the purpose of the attack, remote attackers need authentication, or anonymous user should be available on the ftp server.

Remotely Exploitable

Yes

Locally Exploitable

Yes

3) Vulnerabilities detail

Stack overflow.

A Buffer overflow attack can be implemented in the way ftp server parse path of file or directories when processing MKD, RNFR, RMD and DELE commands. The vulnerability exists in some kind of parsing function that does not properly check length of strings when using string handling functions NWLstrbcpy and NWsprintf.

The vulnerable function is sub_9038 and we've found three implement of improper use of NWLstrbcpy and NWsprintf. The function takes the path of file or directory and does special tasks for different format of the path. In such tasks it copy the path string to a fixed length buffer.

```
.bss:00009105          push    62h ; 'b'
.bss:00009107          lea     eax, [edi+0A90h]
.bss:0000910D          push    eax
.bss:0000910E          push    ebp
.bss:0000910F          call    NWLstrbcpy
.bss:00009114          add     esp, 0Ch
.bss:00009117          push    esi
.bss:00009118          lea     eax, [edi+0E23h]
.bss:0000911E          push    eax
.bss:0000911F          add     edi, 0E01h
```

The above code shows the first flaw which there is no bound checking before using NWLstrbcpy and it directly copies string offset eax to offset ebp. The above code in the patch version is replaced like this:

```
.bss:00009143          push    202h
.bss:00009148          lea     eax, [edi+0E01h]
.bss:0000914E          push    eax
.bss:0000914F          mov     [esp+6C0h+var_18], eax
.bss:00009156          call    NWLmblen
.bss:0000915B          add     esp, 8
.bss:0000915E          push    202h
.bss:00009163          mov     ebx, eax
.bss:00009165          lea     eax, [edi+0E23h]
.bss:0000916B          push    eax
.bss:0000916C          mov     [esp+6C0h+var_1C], eax
.bss:00009173          call    NWLmblen
.bss:00009178          add     esp, 8
.bss:0000917B          push    202h
.bss:00009180          push    esi
```

```
.bss:00009181          add     ebx, eax
.bss:00009183          call    NWLmblen
.bss:00009188          add     eax, ebx
.bss:0000918A          add     eax, 4
.bss:0000918D          add     esp, 8
.bss:00009190          cmp     eax, 202h
.bss:00009195          jg      loc_922D
.bss:0000919B          push    62h ; 'b'
.bss:0000919D          add     edi, 0A90h
.bss:000091A3          push    edi
.bss:000091A4          push    ebp
.bss:000091A5          call    NWLstrbcpy
.bss:000091AA          add     esp, 0Ch
.bss:000091AD          push    esi
.bss:000091AE          mov     ebx, [esp+6BCh+var_1C]
.bss:000091B5          push    ebx
.bss:000091B6          mov     esi, [esp+6C0h+var_18]
```

The above code uses NWLmblen function to check the length of the string and if the string is not valid bound the conditional jump in address 0009195 changes the execution flow to the code below and the function returns with the return value of 0FFFFFFF2h.

```
.bss:0000922D          mov     eax, 0FFFFFFF2h
.bss:00009232          add     esp, 6A8h
.bss:00009238          pop     ebp
.bss:00009239          pop     edi
.bss:0000923A          pop     esi
.bss:0000923B          pop     ebx
.bss:0000923C          retn
```

There are other flaws in other part of the function which exactly the same. At the address 000092D0 (as you see below) the block does an unconditional jump to 0000912B which in turn there is an improper use of NWsprintf.

```
.bss:000092D0          push    esi
.bss:000092D1          lea     eax, [edi+0E23h]
.bss:000092D7          push    eax
.bss:000092D8          add     edi, 0E01h
.bss:000092DE          push    edi
.bss:000092DF          push    offset aSSS_4   ; "%s:%s%s"
.bss:000092E4          jmp     loc_912B
```

Jump to :

```
.bss:0000912B          lea     eax, [esp+6C0h+var_2A8]
.bss:00009132          push    eax
```

```
.bss:00009133          call    NWsprintf
.bss:00009138          add     esp, 14h
```

In the patched version the function first jump to the code below and check for proper length if it is proper conditional jump is implemented to the NWsprintf block and it banned us from overflowing the stack.

Here is the bound check block:

```
.bss:0000942B          push    202h
.bss:00009430          lea     ebx, [edi+0E01h]
.bss:00009436          push    ebx
.bss:00009437          call    NWLmblen
.bss:0000943C          mov     [esp+6C0h+var_20], eax
.bss:00009443          add     esp, 8
.bss:00009446          push    202h
.bss:0000944B          add     edi, 0E23h
.bss:00009451          push    edi
.bss:00009452          call    NWLmblen
.bss:00009457          add     esp, 8
.bss:0000945A          mov     edx, [esp+6B8h+var_20]
.bss:00009461          push    202h
.bss:00009466          add     edx, eax
.bss:00009468          push    esi
.bss:00009469          mov     [esp+6C0h+var_20], edx
.bss:00009470          call    NWLmblen
.bss:00009475          add     esp, 8
.bss:00009478          add     eax, [esp+6B8h+var_20]
.bss:0000947F          add     eax, 2
.bss:00009482          cmp     eax, 202h
.bss:00009487          jle     loc_9551
.bss:0000948D          mov     eax, 0FFFFFFF2h
.bss:00009492          add     esp, 6A8h
.bss:00009498          pop     ebp
.bss:00009499          pop     edi
.bss:0000949A          pop     esi
.bss:0000949B          pop     ebx
.bss:0000949C          retn
```

The above bound checking block like the last one returns 0FFFFFFF2h when the length is not proper. There is another similar flaw that patched in the same way in our parsing function that we no cover because of similarity.

Now it is time to see what happened when the function return 0FFFFFFF2h error code.

After the program has patched and the buffer overflow omitted when the parse function is called with long buffer it return and error code. The caller function check return value and send a error message to the client.

Here is the way it checks the return value:

```
.bss:00009474          push    ebx
.bss:00009475          push    esi
.bss:00009476          push    edi
.bss:00009477          sub     esp, 818h
.bss:0000947D          mov     ebx, [esp+824h+arg_0]
.bss:00009484          lea     esi, [ebx+61h]
.bss:00009487          mov     al, [ebx+60h]
.bss:0000948A          lea     edi, [ebx+0E01h]
.bss:00009490          mov     [esp+824h+var_10], al
.bss:00009497          mov     ax, [ebx+682h]
.bss:0000949E          push    ebx
.bss:0000949F          mov     [esp+828h+var_14], ax
.bss:000094A7          call    sub_9038
.bss:000094AC          add     esp, 4
.bss:000094AF          test    eax, eax
.bss:000094B1          jnz     short loc_94F2
.bss:000094B3          mov     ah, byte ptr [esp+824h+var_14]
.bss:000094BA          test    ah, 2
.bss:000094BD          jz      short loc_950C
```

It checks if the return code is 0 or not and if it is not 0 jump to loc_94F2. And here is the 000094f2:

```
.bss:000094F2          push    offset aInvalidPath ; "Invalid Path"
.bss:000094F7          push    226h
.bss:000094FC          push    ebx
.bss:000094FD          call    sub_1CE0
.bss:00009502          mov     eax, 0FFFFFFF2h
.bss:00009507          add     esp, 0Ch
.bss:0000950A          jmp     short loc_94E8
```

And the above code is the error message that is sent to the client. The function sub_1CE0 is responsible for sending various text messages to the client and is used in many part of the program and here simply warn the user for invalid path.

Exploitation

There is a proof of concept on the exploit database - http://www.exploit-db.com/exploits/14928/