

Exploit Development – An illustration of a buffer overflow Vulnerability

Michael Sowemimo 1505134

White Paper

Abertay University

BSc Ethical Hacking & Countermeasures

2016/17

Contents

[2 Abstract 3](#_Toc481516312)

[3 Introduction 4](#_Toc481516313)

[4 Procedure 6](#_Toc481516314)

[4.1 Description of the Vu Player 2.49 Vulnerability 6](#_Toc481516315)

[4.2 DEP (Data Execution Prevention ) Enabled 6](#_Toc481516316)

[4.2.1 Basic Buffer Overflows 7](#_Toc481516317)

[4.2.2 Verifying the Flaw 7](#_Toc481516318)

[4.2.3 Return Orientated Programming- Calculator 10](#_Toc481516319)

[4.2.4 ROP Gadgets 10](#_Toc481516320)

[4.2.5 ROP-Bind Shell 15](#_Toc481516321)

[4.2.6 ROP – VNC Reverse \_tcp 16](#_Toc481516322)

[4.3 NO DEP 17](#_Toc481516323)

[4.3.1 Local Buffer Over Flow 17](#_Toc481516324)

[4.3.2 Verifying the Flaw 20](#_Toc481516325)

[4.3.3 Buffer Over Flow Calculator 21](#_Toc481516326)

[4.3.4 Buffer Over Flow Remote Shell 22](#_Toc481516327)

[5 Discussion And Results 24](#_Toc481516328)

[6 References 24](#_Toc481516329)

[7 Appendix 24](#_Toc481516330)

# Abstract

The Buffer Overflow vulnerability has been in existence since 1988. The famous Morris Worm was responsible for dictionary attack which had embedded list of popular passwords. Though the Buffer Overflow vulnerability occurs when an adjacent buffer in memory is overridden by large amount of data than what is expected, this leaves a program vulnerable to attack implementing large data such as a malicious code. This paper is documented to examine Buffer Over Flow attacks with the assistance of a vulnerable MP3 application called VU Player 2.49. The Application itself is vulnerable to specific Buffer Overflow vulnerability.

The paper will illustrate a simple proof of concept to create exploits in Perl that will be used to create .M3U files that will contain an embedded payload. The exploits will permit a bypass in the Data execution policy on a Windows XP SP3 machine. The Data execution prevention is used to prevent malicious executables from running on any Windows Operating system. Also, this paper will describe how to bypass the Data execution policy by skilfully crafting ROP chains that will allow the execution of shellcodes. The shellcode will be a simple calculator payload. Also ,being able to obtain a target desktop by using VNC inject.

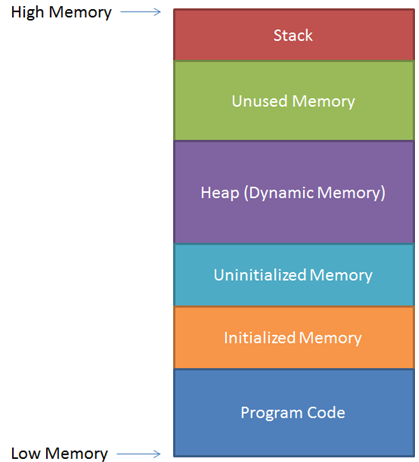
# Introduction

It is quite common for developers to negate secure coding procedures during the initial stage of developing applications. As a result, the applications are subjected to an arbitrary of exploits that allow complete access to remote machines. Nevertheless, it detrimental that all developers utilize software testing at every phase of the software development life cycle. At times developers may use low level languages such as; C++ to compile commercial applications. C++ is known to have no built-in protection against accessing and overwriting data in memory, plus the capability of manipulating common programming constructs (Depual, 2012).

It apparent that software developers should address the errors as soon as there encountered, moreover the software vulnerabilities can be critical in nature and used to manipulate memory usage on a Computerized system. Attacks such as buffer overflows ,are targeted at poorly designed applications that do not contain bounds checking (OWASP, 2016 ). A buffer overflow attack, is an attack were the malicious user takes advantage of program that is waiting on users input. Moreover, the buffer overflow attack overwrites memory that is an adjacent to a buffer that should not be modified.

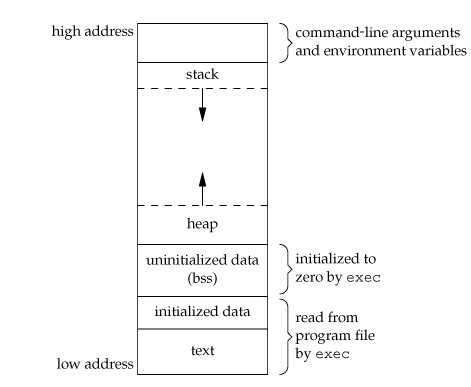
Furthermore, once the buffer overflow occurs in a specific program, it will crash the program or make it unstable. There are Two main types of buffer Overflow attacks which are as followed:

1. **Stack based:** The stack is an area in memory that used to organise data that relates to function calls and the parameters, function local variables and instruction pointers (CWE, Stack based Buffer Overflow , 2017). However, the stack based overflow is one of the most common vulnerability (**Figure 1.1**).



**Figure 1.1 The stack**

1. **Heap based:** A heap based overflow, is like a stack based Overflow but the difference is that the attacker targets the heap. The heap is an area in memory used for dynamic memory allocation (CWE, Heap Based Buffer Overflow , 2017). Also, the heap is used to process and store data in a variable that won’t be known until the program is running. The heap buffer Overflow attack comprises sensitive data by overwriting filenames and other variables on the heap and the logical program flow (**Figure 1.2**).



**Figure 1.2**

The objective of this report is to provide a tutorial of a Stack based Overflow attack. The attacks will be conducted in a virtual environment while using Windows XP SP3 Virtual environment. To test the buffer Overflow attack, an MP3 Application such as VU player will be used a test guide (Symantec, 2017). According to Symantec, the VU player is subjected to stack-based Overflow vulnerability. Since the Vu Player fails to determine the size of the user-supplied data before copying it into an inadequately sized process buffer. Exploiting the vulnerability allows a malicious attack to execute a vast amount of machine code.

The report will illustrate the procedures used for a buffer flow attack. Plus, a discussion relating to bypassing intrusion detection system.

# Procedure

## Description of the Vu Player 2.49 Vulnerability

Vu Player is a freeware multi-format audio player for Windows Operating Systems. Meanwhile, the Vu player contains a Buffer Overflow vulnerability. The Buffer Overflow vulnerability, allows an attacker to overwrite a computer system register to extinguish the local software processes. An attacker can manipulate the EIP register to implement the next instruction of their choosing. As well as executing an arbitrary of codes such as; a large malformed string to overwrite the EIP register of the process. Once the attacker overwrites an active process, the attacker can compromise the entire computer system.

## DEP (Data Execution Prevention ) Enabled

The Data Execution prevention is a security feature for Windows Operating System, can help prevent damage of computers from viruses and any other potential security threats (Nakodari, 2008). Additionally, Data Execution prevention can help protect a computer which has a Windows Operating System, by monitoring the programs and making sure that they use system memory is safe. To examine the Buffer Overflow vulnerability, the Vu Player will adhere a series of Buffer Overflow attacks while DEP is enabled. Also, the Buffer Overflow will be examined using the Immunity debugger (sectools.org, 2010). The immunity debugger is implemented to reverse engineer the Vu Player Application, to understand the behaviour of the Buffer Overflow Attack in Memory. The Data Execution prevention can be enabled by following the procedure which is shown below:

* Click start and select Control Panel
* Select Performance and Maintenance tab
* Click System
* Click the Advanced Tab
* Select Performance Options and then click Data Execution Policy Tab
* Under the Data Execution Prevention Tab, click the Turn on DEP all programs and services except those select option
* Click Ok
* A notification will pop up indicating that the computer must restart
* Restart the computer
* On the boot menu select Opt-in

Moreover, the Data Execution policy has 4 global configurations. They can be defined as the following ;

* **Optin :** This is the default configuration on computer systems that can implement hardware enforced DEP.Futhermore, this setting only covers window systems binaries.
* **OptOut :** DEP is enabled by default to all processes. Plus, being able to manually create a list of specific programs that do not have DEP enabled
* **AlwaysOn :** This setting allows full DEP coverage for the whole system. All processes will always run with DEP applied, but with the exception list for DEP protection is not authorized
* **AlwaysOff**: This setting does not allow any DEP protection for any part of the system.

For this tutorial , the Buffer Over Flow examination will be utilized using the Opt-in setting .

### Basic Buffer Overflows

As a proof of concept, the Buffer overflow attack will be implemented by a Perl Scripts which will be compiled to manipulate strings and create .M3U files. The Perl script will be structured as shown below :

Define two files name as stings with the keyword MY i

**my $file1 = “crash1.txt”;**

**$junk1 = “x41” x 1000;**

Create a string containing 1000 ‘A’

**my $eip = pack(‘v’, 0x0013438); # pack “Value”**

**my $junk2 = “BBBB”;**

**my $junk3 = “1ABCDEFGHIJKLMNOPQRSTUVWXYZ”;**

Create the file “Crash1.txt”

**open($FILE , “>$file1”);**

**print $FILE $junk1;**

**close($FILE);**

Create “Crash2.txt” containing 3 strings concatenated

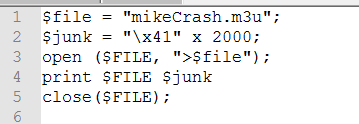
**open($FILE “>$file2”);**

**print $FILE $junk1.$junk2.$junk3;**

**close($FILE)**

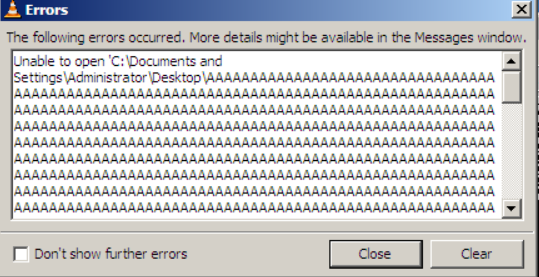
### Verifying the Flaw

During this stage of the examination the objective is to identify the existence of the flaw and then structure the parameters for the buffer Over Flow exploit, plus identifying the size of the stack and space for shellcode. To achieve the Buffer Over Flow exploit, the first procedure is to create a basic Perl program that will generate an MP3 that will contain 2000 x ‘A’ in the form of a playlist and see if it crashes the VU Player Application. The Perl program can be illustrated as shown below ;

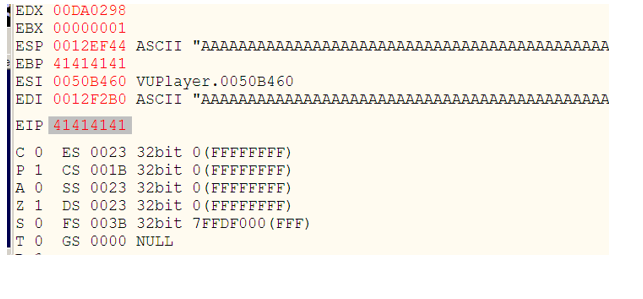


**Figure 1.1 Generating 2000 A’s in the MP3 File**

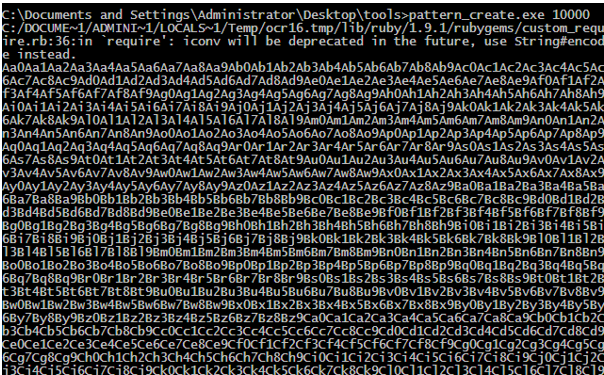
The Screenshot below verifies that the M3U player does contain 2000 x A’s.



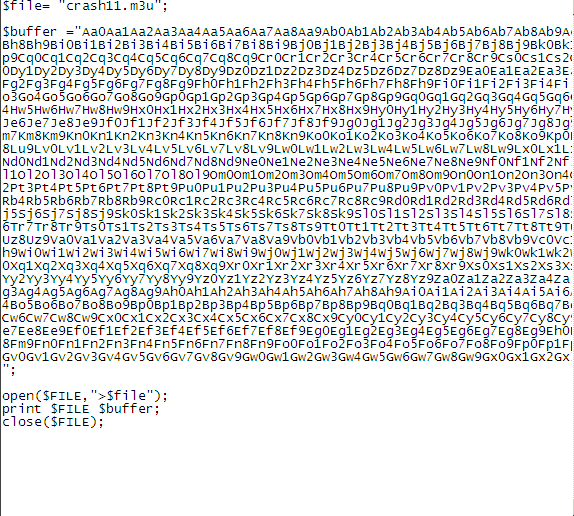
The next step is to load the MP3 File into the VU Player and check if the MP3 Application crashes, to examine the behaviour of the MP3 Application. The MP3 Application will be attached to immunity debugger to test and analyse the Buffer Over Flow Vulnerability. As soon as the MP3 is loaded in the VU player playlist , the program will crash revealing that the Buffer Overflow has occurred. An inspection can be sourced with the immunity debugger. The Immunity Debugger will illustrate that the Instruction pointer has been overwritten with characters 41414141 which is the ascii for AAAAA. Evidence regarding the crash is shown below.



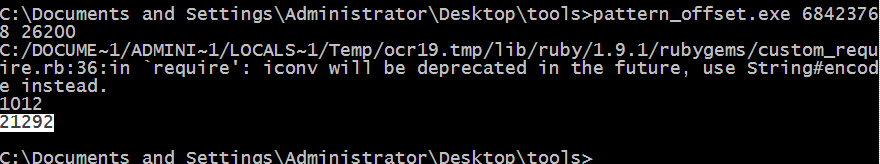
Once the vulnerability is confirmed, the next procedure is to finalize the precise distance to the EIP, getting the distance to the EIP is important. The reason is that it is possible to use the full scope of the EIP to run more malicious code. To get the distance to the EIP, A Metasploit pattern\_create tool can be used to generate a predictable pattern.



Immediately when the predictable pattern is generated, the predicted pattern is placed within the $junk variable of the Perl script and then attached to Immunity debugger for analysis.



The results indicate that the EIP and the stack has been overwritten with the predictable pattern, by using the value held in the EIP and using the offset, it is possible to get the exact distance to the EIP. In this case, the exact distance is 1012 but the screenshot below indicates a range between 1012 - 26200.



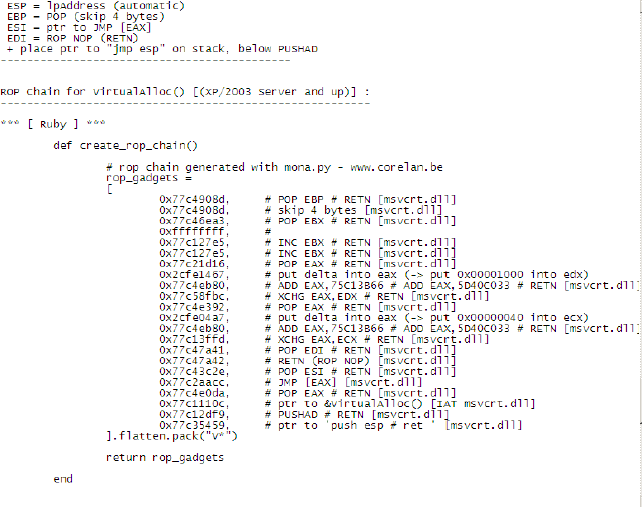
### Return Orientated Programming- Calculator

As mentioned earlier, the DEP prevents the attackers from running malicious scripts in memory. Especially the shellcode, but the DEP does not prevent access us from controlling the EIP. Which means It is possible to direct the Vu Player to another memory location to run shellcode. This can be accomplished by Return Orientated Programming(ROP). ROP allows the stack to jump to an alternative memory location by using a ROP chain.

### ROP Gadgets

The immunity debugger has mona.py which is a plugin that is used to generate the ROP gadgets. The command **“!mona rop -m msvcrt.dll -cpb ‘/x00/xoa/xod’**” was used to search for ROP gadget that is embedded with the msvcrt.dll and does not have any bad character references.

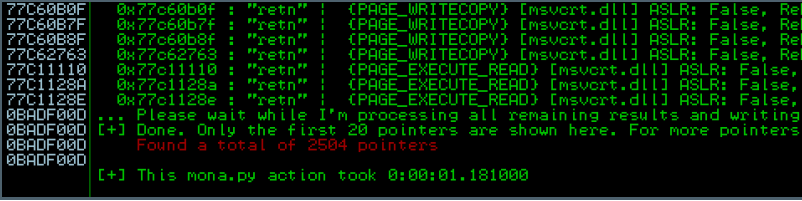
The text file below illustrates a ROP.txt file which has an extensive list of ROP gadgets that can used as a proof of concept. The Screenshot below is a ROP chain for Virtual allocation that can be used to bypass DEP.



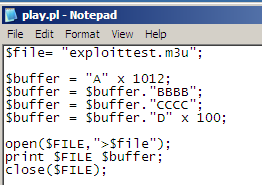
The next procedure is to allocate an Return address(RET) to initiate the ROP chain, this can be done by using the following command;

**mona find -type instr -s "retn" -m msvcrt.dll -cpb '\x00\x0a\x0d**

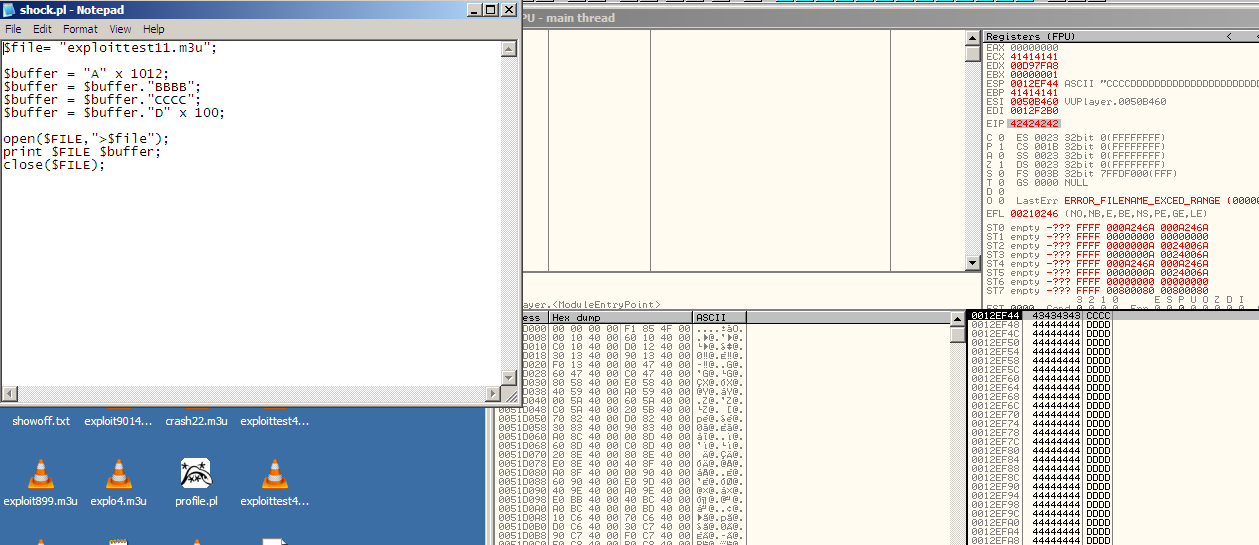
The command above will output some RET addresses that is located within the msvcrt.dll, it is also vital to note that only a small portion of the addresses are functional. The aim of this exercise is to bypass DEP, so that we can launch the shellcode, the RET addresses that are marked page\_execute\_read cannot be used for the examination. Although the address 0x77c110 is suitable for the shellcode exploit.



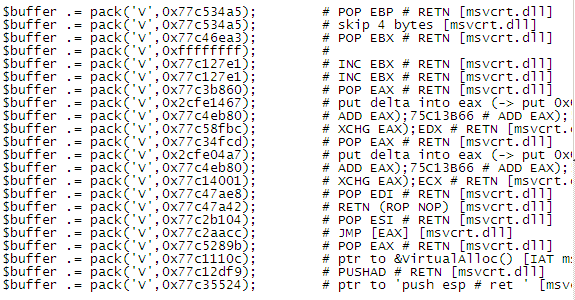
Since we have an appropriate address, we can commence with the examination with a PERL script which can be shown below;



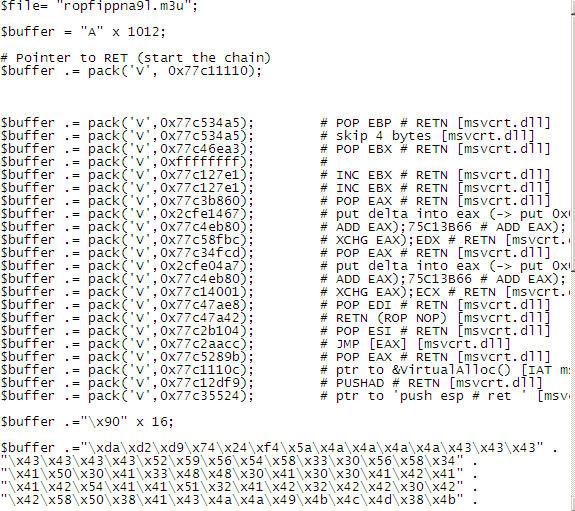
As soon as the script created the malicious .m3u file, the next procedure is to load the VU Player into the immunity debugger and set a breakpoint to the return address (0x77c1110) and then run the VU Player Application with the malicious .m3u file.

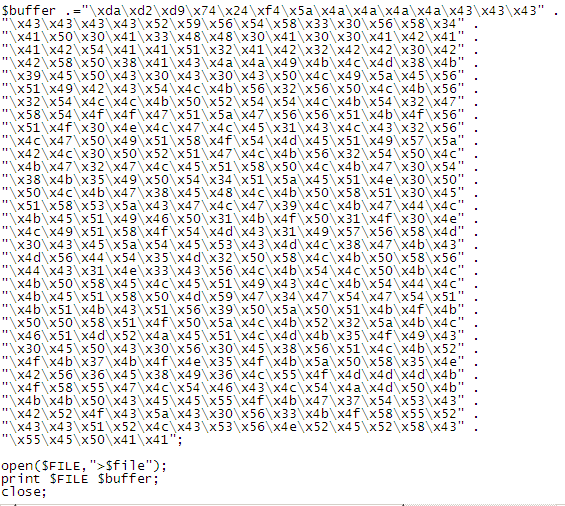


The screenshot above indicates that the stack is retains the value BBBB, now we need four bytes as a vector to permit the shellcode to run. This should be enough to allow the exploit to run successfully. The only issue is that mona.py cannot generate a ROP Chain in Perl, in this case we can use a tool called rop2perl, which is used to compile a ROP chain in Perl.

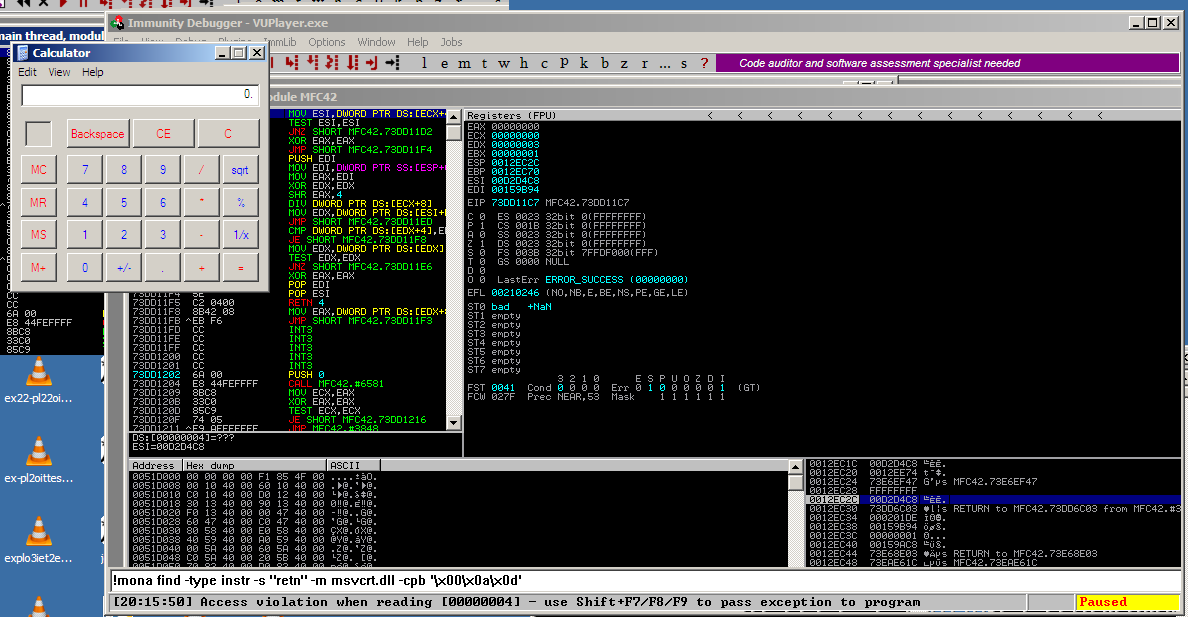


Now that the Perl ROP chain is generated, it is possible to test the exploit by adding the Calculator shellcode after the NOPS.



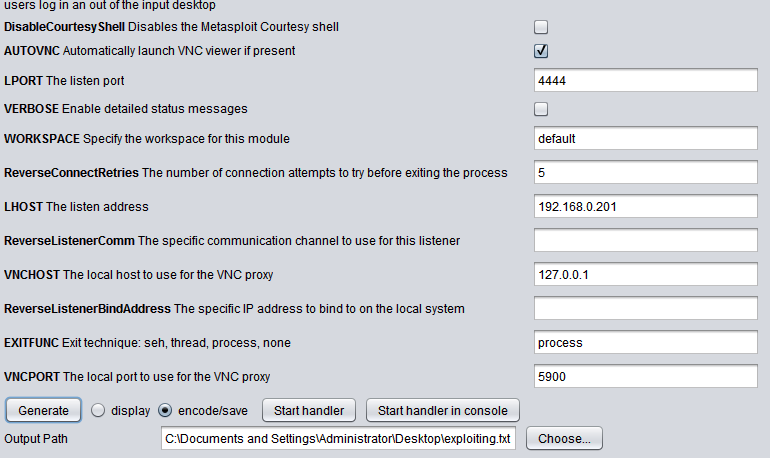


Subsequently, all is required is to compile the Perl Script and place the .M3U player within VU Player and loading the Application inside the Immunity debugger. The Calculator will work as soon as the .M3U is opened. The screenshot below indicated that the calculator is running.

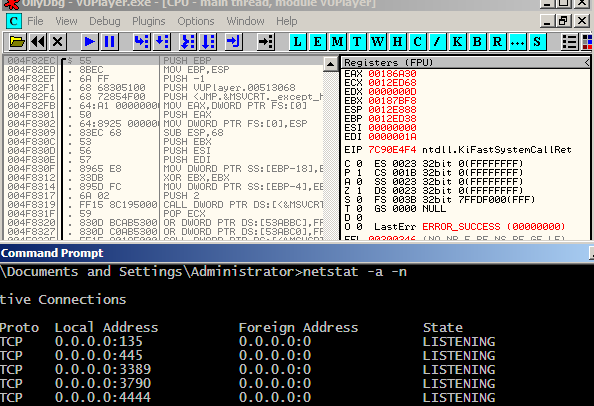


### ROP-Bind Shell

Since the exploit has worked and has been analysed, additional exploits such as a bind shell can be implemented for command execution on a target machine. In this step, all we need is to generate shell code for Bind Shell and copy and paste the payload into the variable that contains the payload for the Calculator in the Perl Script. The payload can be generated using the MSF GUI.

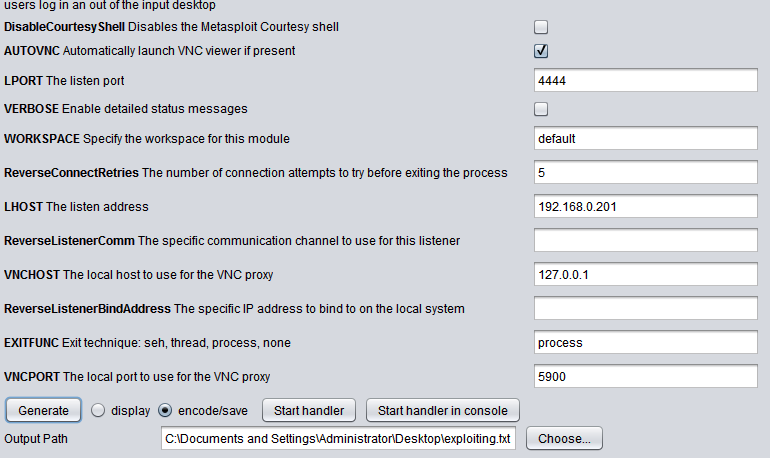


Once the listener is set on port 4444, it is possible to connect to port and activate a command line of the target. By using Kali and using the netcat tool , it is possible to access the command of the target machine. This can be done using the following command ; nc.exe 192.168.0.200 4444.

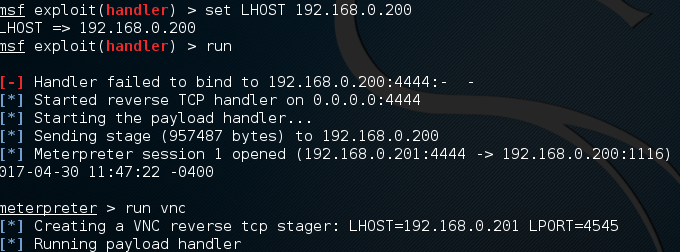


### ROP – VNC Reverse \_tcp

For an additional investigation, this exercise examines a Buffer Over Flow attack with the VNC inject. The VNC inject exploit can be used to inject VNC DLL into memory of a remote process, plus being to connect to the graphical user interface of a target that is using the VNC client. To concept still remains the same, all we have to do is to generate a VNC payload for Perl . An example can be seen in the screen shot below .



Once the VU player opens the .M3U file containing the VNC inject payload. A handler will be implemented in Metasploit which will wait for the target to open the .M3U file.



The VNC injection will allow us to see the Victims Desktop.

## NO DEP

This step is used to Examine the buffer Overflow attack when DEP is not enabled. This step is for a Proof of concept of a basic Buffer Overflow exploit. Disabling DEP can cause vast number of issues such as; an attacker being to run malware and getting restricted access to a target machines memory.

### Local Buffer Over Flow

The first step is the exact same step for bypassing the DEP while it is enabled. All methods are the same except from generating the ROP gadgets.

Define two files name as stings with the keyword MY i

**my $file1 = “crash1.txt”;**

**$junk1 = “x41” x 1000;**

Create a string containing 1000 ‘A’

**my $eip = pack(‘v’, 0x0013438); # pack “Value”**

**my $junk2 = “BBBB”;**

**my $junk3 = “1ABCDEFGHIJKLMNOPQRSTUVWXYZ”;**

Create the file “Crash1.txt”

**open($FILE , “>$file1”);**

**print $FILE $junk1;**

**close($FILE);**

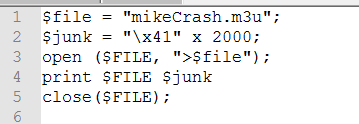
Create “Crash2.txt” containing 3 strings concatenated

**open($FILE “>$file2”);**

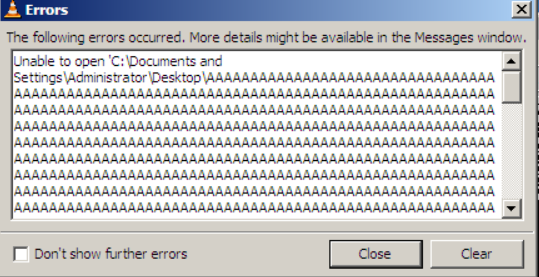
**print $FILE $junk1.$junk2.$junk3;**

**close($FILE)**

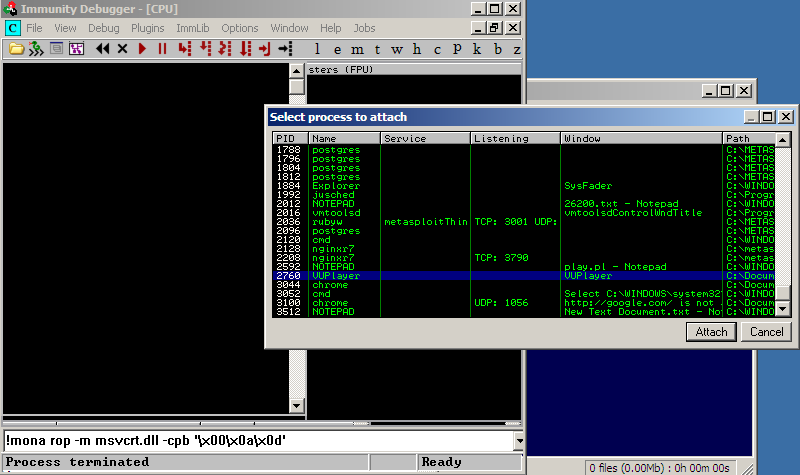
The screen shot below proves that the Buffer OverFlow vulnerability exists.



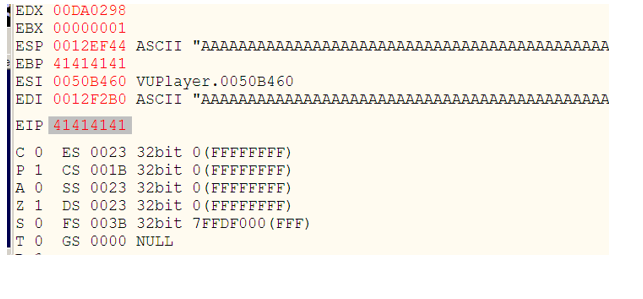
A quick look to reveal if the .M3U contains 2000 X A’s.



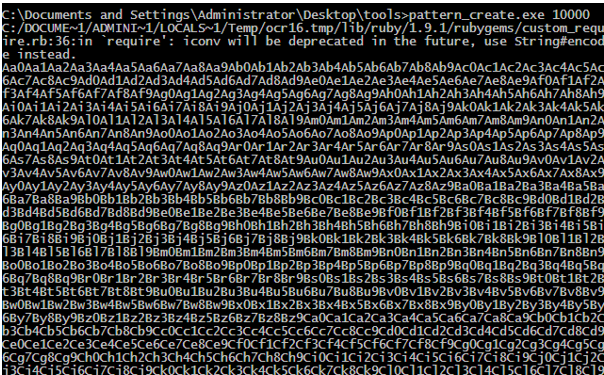
The next step is to attach the VU Player to the Immunity Debugger and examine how the program crashes.



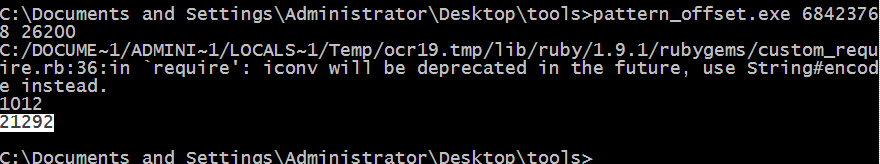
As soon as the crafted perl Script generates the malformed .M3U file, the VU Player should crash indicating that the EIP holds characters 41414141 which is the ascii for AAAA.



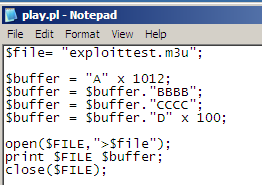
As soon as the vulnerability is identified, the next procedure is the exact replica of the Buffer Overflow when DEP is Enabled.



The same distance was applied in this step which is 1012.



The next step involves adding more characters, for this exercise we will write 1012 x A’s, 4 X b’s and 4 x c’s. This will let us understand where the shellcode will be saved.



### Verifying the Flaw

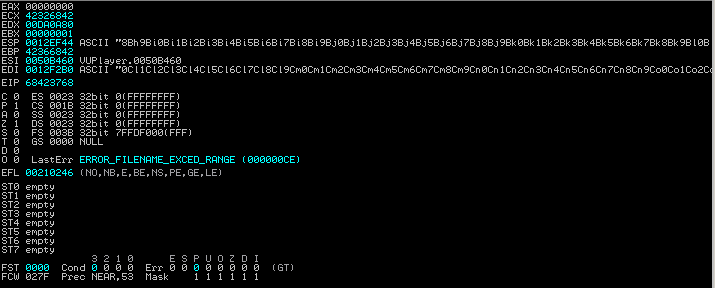
|  |
| --- |
| CCCC(The Shell Code Will Run here) |
| BBBB (EIP) |
| AAAA (Stack)  AAAA  AAAA |

Now we know that the code is on top of the stack, the exploit can commence from here. In theory, we want to substitute our BBBB with a value that will execute the Shellcode, which will then execute the next ret command. The ret command is the command that is used to tell the program to return to replaced instruction pointer with our own memory address, when the program exits.

Then we need to replace BBBB with a jump ESP to the address where the shellcode starts. The address holding CCCC is 0012ED50 ;



The next step is calculate the amount of space that the stack has for the Shellcode. By using the Create\_pattern .exe it is possible to determine how much room is on the heap to for us to use for the shellcode.



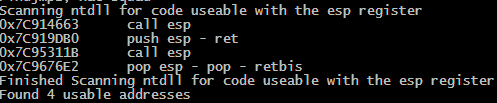
As a result the pattern has not overwritten any of values in the stack and the EIP still contains a series of BBBB. This means that we have over 1012 of space, which a plentiful amount to run the shellcode.

However, once the shellcode is executed it used system calls. The system calls will alter the functionality of our code. To counterattack this issue, we must generate some NOPS(NO operation). This will allow the shellcode to auto-increment are shellcode, since there is enough room for the shellcode, we will add 24 NOPS to the Perl Script. Using many NOPS is referred to as a NOP slide.

Furthermore, a stack jitter is the result of code being run or a program executing in several events. It is quite common for programmers to make this mistake and leads to the ESP being handled at a different memory Address. Although a JMP ESP command can be used to jump straight to the top of the stack, so that the EIP is overwritten.

The JMP ESP is a native command that is used in many programs and .dlls, it also works similarly to the Call ESP command. The comparison is that the CALL command will place values on the stack. This will mean a larger NOP slide.

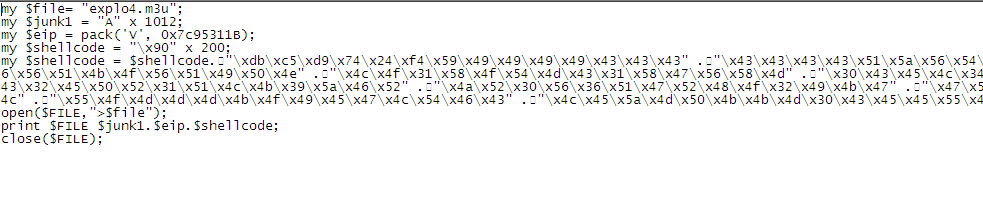
To look for a jump command we will need the find Jump tool, the find jump tool looks for addresses relating to JMP ESP /CALL ESP addresses.



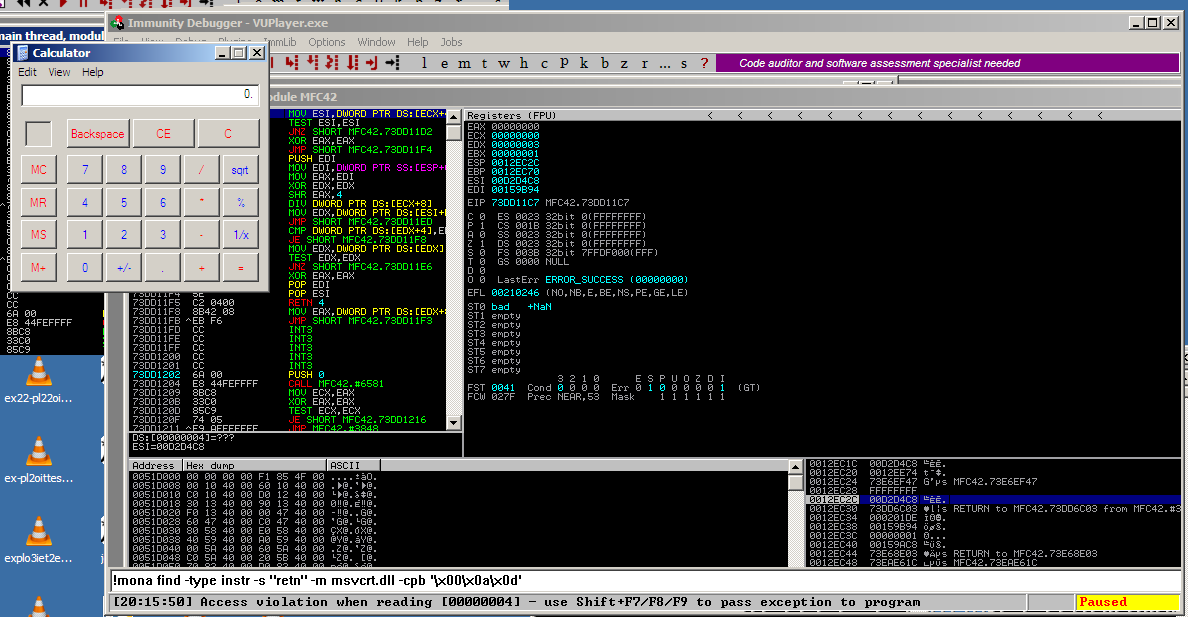
The screen above illustrates that there is no JMP addresses, although we can use the CALL ESP memory address(0x7C914663). Plus we will need the 200 NOPS after the value of the EIP .

### Buffer Over Flow Calculator

To run the calculator, all we need is the shell code which can be generated using Metasploit. Once the shellcode is generated it can be placed after the NOPS.

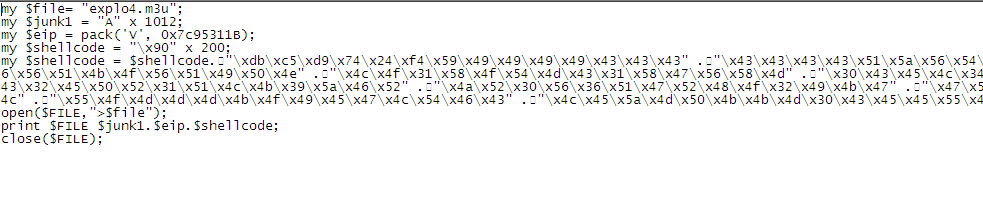


Once the .m3u is loaded it should execute the Calculator.

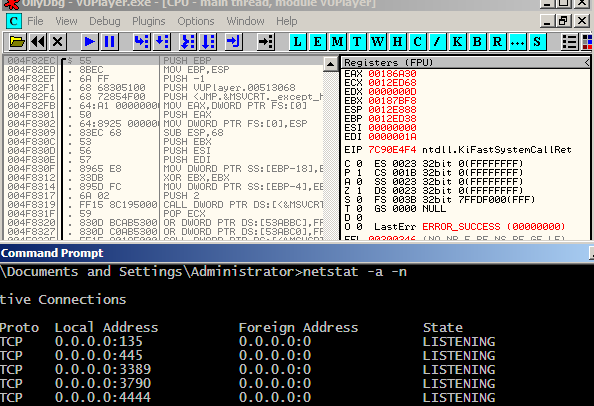


### Buffer Over Flow Remote Shell

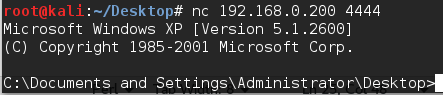
Similarly, the remote shell can be implemented by following the exact procedure that was used to execute the calculator exploit. To set up the remote shell exploit, a listener must be created on port 4444 and then then will permit access to the target command prompt once the connection is made. This attack allows the intruder to gain full access to the target machine. The attack can be launched by generating the payload for the remote shell using Metaspolit and adding to the perl script. In this case the Script needed 200 NOPS.



Once the playlist has been inserted to the VU Player , the program will crash allowing us to have control of the target machine. For verification , the command Netstat -a -n was used to consolidate that the port 4444 is open and listening.



To connect to the target machine open Kali Linux and make sure that the Machine is on the same network and run the following command; nc 192.168.0.200 4444 , this allow access to the target’s machine.



# Discussion And Results

This report was conducted to illustrate a Buffer Over Flow vulnerability in the VU Player. As stated earlier a Buffer Over Flow is the ability to manipulate a program or process to write more data into a fixed block of memory. As illustrated in the procedures, the buffer over flow attack can cause an arbitrary of issues in terms of memory manipulation. The steps shown are identical to a real-life Buffer Overflow attack. Developers should take extreme caution when developing in-house programs. It appears that the cause of the Buffer Overflow attack is due to not having bounds checking enforced.

Also, all Buffer Overflow attacks were successful in the Windows XP Service Pack 3 regardless of DEP being enabled or Disabled. Organisations should be discouraged from using this operating system, since it can cause many obstruction in terms of malicious sattacks against a target system. In the real world , these attacks would be difficult especially if the targeted system has a Intrusion Detective System. Although evading the Intrusion detective system is possible. Moreover, an intrusion detective system in spilt into two categories which is signature detection and anomaly detection.

The signature detection is a well-known signature that used for network traffic to detect potential malicious traffic. While the anomaly based detection is an approach to examine previous network traffic to search for patterns that are abnormal. Although most commercial intrusion detective systems use the signature based detection because of the speed and the ease of use by the analyst, plus a reduced number of false positives. However, the signature based detection riles on a string matching tactics which depends on string matching a specific traffic. Nevertheless, the IDS system looks for a known signature in the exact place in data payload of the packet .

The downfall of this, is an attacker can perform several manoeuvres to evade the detection by the IDS. A tool called ADMutate, can be used to obfuscate any buffer overflow attack against any network service. The main objective of this tool is to change an exploit signature every time that is executed, which results in polymorphic shell code. This proves that there is no limit to what a Buffer overflow is executed on a target system. Users should pay great attention to the files that they download from the web broswer, and implement the appropriate security techniques to avoid an attack such as a Buffer Overflow.

# References

CWE. (2017, January 18). *Heap Based Buffer Overflow* . Retrieved from https://cwe.mitre.org/data/definitions/122.html

CWE. (2017, January 18). *Stack based Buffer Overflow* . Retrieved from https://cwe.mitre.org/data/definitions/121.html

Depual, N. (2012, 10 April). *Veracode*. Retrieved from What is a buffer Overflow ? : https://www.veracode.com/blog/2012/04/what-is-a-buffer-overflow-learn-about-buffer-overrun-vulnerabilities-exploits-attacks

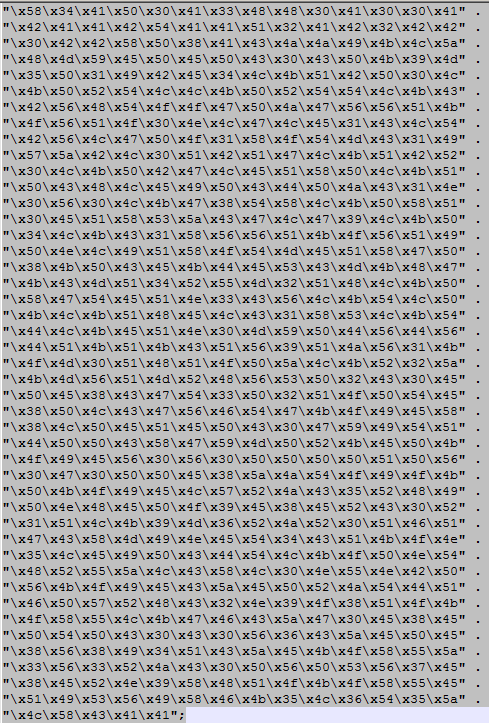
Nakodari. (2008, October 28). Retrieved from http://www.addictivetips.com/windows-tips/what-is-data-execution-prevention-dep-and-how-to-disable-it-in-windows-vista/

OWASP. (2016 , 06). *Buffer Overflows* . Retrieved from https://www.owasp.org/index.php/Buffer\_Overflow

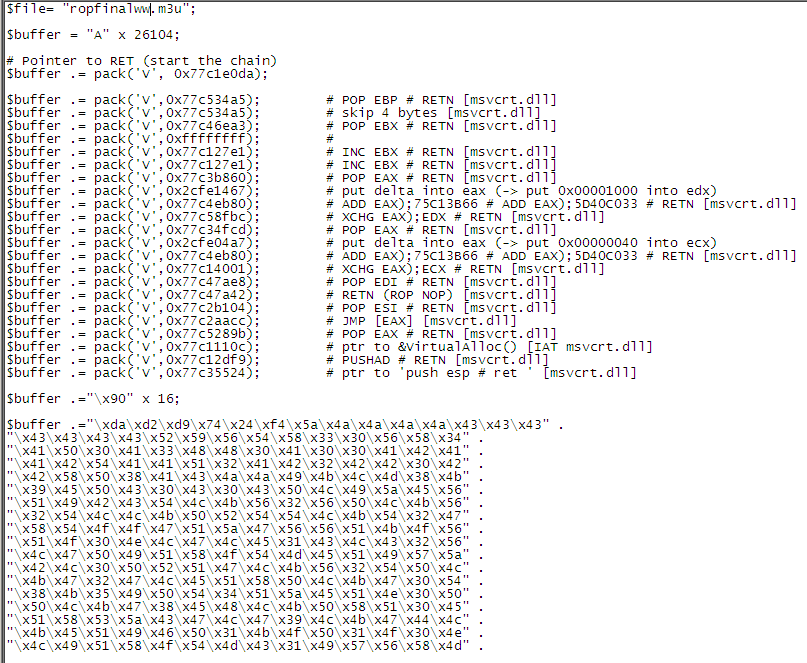
sectools.org. (2010, Decemeber 6 ). Retrieved from Immunity Debugger : http://sectools.org/tool/immunitydebugger/

Symantec. (2017). Retrieved from HTTP VUPlayer M3U UNC Buffer Overflow: https://www.symantec.com/security\_response/attacksignatures/detail.jsp?asid=23276

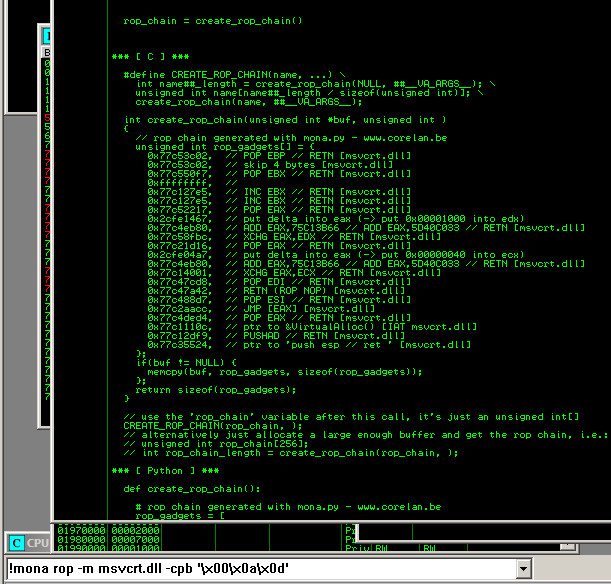
# Appendix



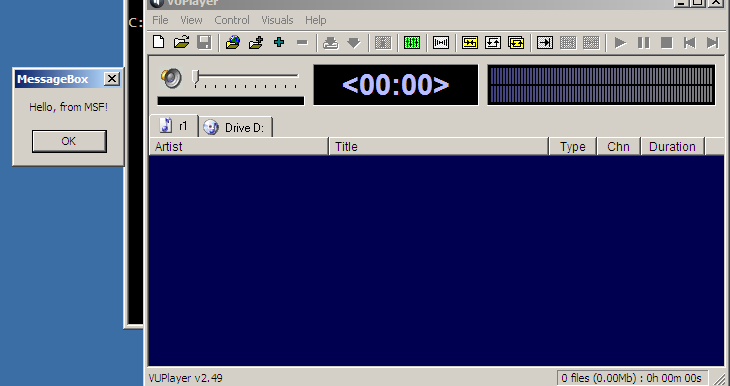
**Figure 1.1 Shellcode for the Remote Shell**



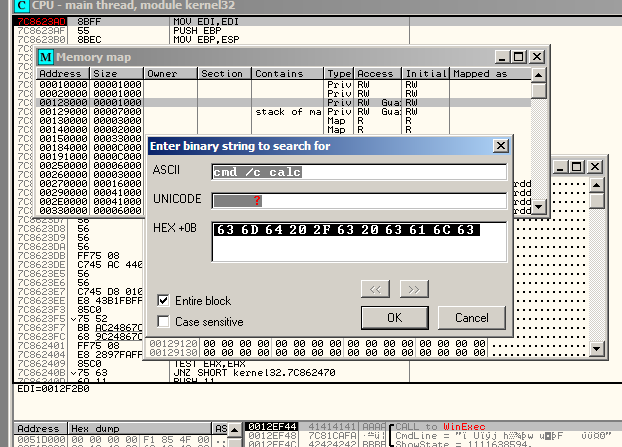
**Figure 1.2 Remote Shell**



**Figure 1.3 Mona.py**



**Figure 1.4 Message Box**



**Figure 1.4 Set Breakpoint for Return to Lib**