6/15/2020

Mannage M.S.D.S. IT17069564

Buffer Overflow Exploit Workthrough

**Buffer Overflow Exploit Walkthrough - VUPlayer 2.49**

This is an in-depth walkthrough of a Buffer overflow vulnerability which will be exploited in the VUPlayer software in order to obtain root shell access of the victim’s system. The exploit that we are about to perform is similar to the local buffer overflow vulnerability available in Exploit-DB (<https://www.exploit-db.com/exploits/40018>), but in this walkthrough we will have the final goal of gaining root access of the victim’s machine.

**Local Buffer Overflow Vulnerability in VUPlayer**

As mentioned previously, we are exploiting a vulnerability that exists in VUplayer software versions 2.49 and prior. It is a local exploit which means the victim must ultimately run a malicious file through the software in order for the vulnerability to be exploited.

In this case, we will be creating a malicious .m3u file which contains our payload, and we will be sending it to the victim through social engineering means (ex: Downloading from the Internet, Sent through a USB drive, etc.). Once the victim opens this file through their VUPlayer software, the exploit will run, and our system will have their root shell access.

**Setting up the Lab**

The first step in developing this exploit is setting up our necessary machines and software for writing and testing the exploit. We will be having two virtual machines for developing this exploit.

The first will be a Windows XP Service Pack 3 machine, which will act as the victim’s PC. In this machine, we must install the VUPlayer software. For this walkthrough we will be using VUPlayer 2.49 which can be downloaded here:

<https://www.exploit-db.com/apps/39adeb7fa4711cd1cac8702fb163ded5-vuplayersetup.exe>.

After installing VUPlayer 2.49, we can run the program and view its interface as shown in Figure 1 below. The victim will be opening our exploit file through File > Open Playlist.

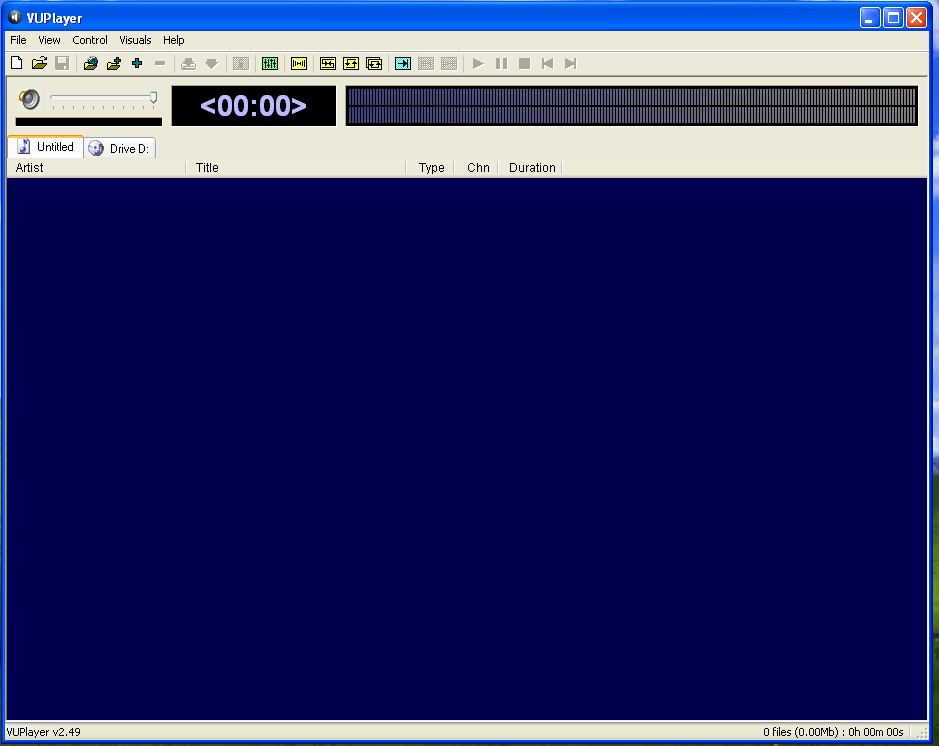


Figure 1 : VUPlayer 2.49 Interface

The machine must also have a debugger installed, which we will be using to analyze the registers and stack for changes to how they react to our exploit files. In this walkthrough, we will be using the Immunity Debugger, which can be downloaded here:

<https://www.immunityinc.com/products/debugger/>

The second machine will be running Kali Linux OS, which will be acting as the attacker’s machine. We will be using this machine to write all of our exploit codes using Kali’s in-built tools.

**Verifying the Vulnerability**

After we have set up both our machines, the next step is to verify that the VUPlayer actually has a buffer overflow vulnerability. To do this, we can create a .m3u file that consists of a long string of A’s, B’s and C’s.

We can write the following nodejs script to create this file.

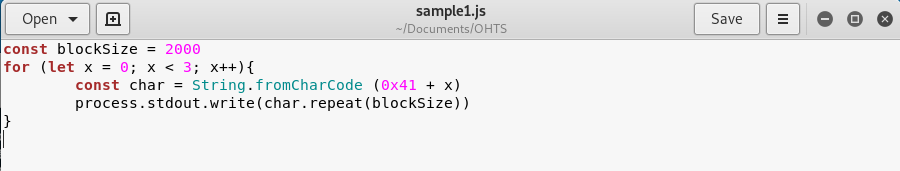


Figure 2 : Script to verify buffer overflow

We can run the script through the terminal and pass the output into the exploit.m3u file as shown in Figure 3 below. The resulting .m3u file contains 2000 A’s, 2000 B’s, and 2000 C’s.

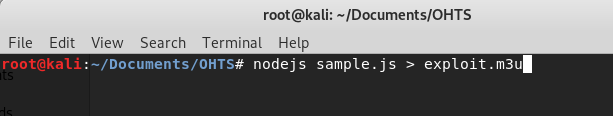


Figure 3 : Write output into exploit.m3u

Now that we have crafted our first exploit.m3u file, we must send it to our Windows XP machine to test it. To test the file, we must first open the Immunity Debugger and open the VUPlayer software through the debugger. After doing so, we will be able to see the assembly code of the VUPlayer program and the current register and stack information.

We must now run the VUPlayer through the debugger (F9 key) and we will see the VUPlayer interface open up. Using the VUPlayer we can now open our exploit.m3u file and observe the output.

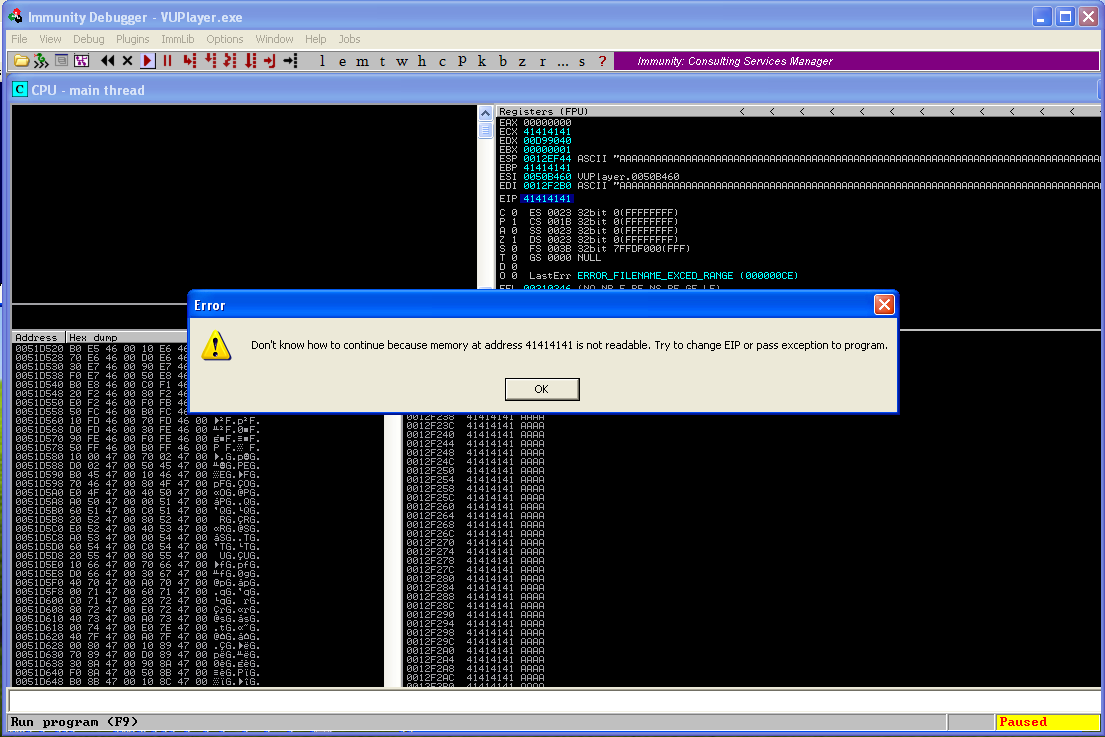


Figure 4 : Program crashes as expected

As shown in Figure 4 above, after opening our exploit.m3u file, VUPlayer will crash with the error message saying that EIP address 41414141 is not readable. 41 is the ASCII value of the character “A”, which means that we have successfully overflowed the buffer with our large string of characters.

A screenshot of a computer

Description automatically generated

Figure 5 : Inspecting the overflowed stack

Figure 5 above shows the EIP register overwritten with 41414141, and below we can see many addresses of the stack, all overwritten with multiple A’s. With this, we are able to verify the buffer overflow in the VUPlayer software.

**Finding the exact locations of EIP and ESP**

Now that we have verified the buffer overflow in the program, the next step is to find the exact locations of the EIP (Instruction Pointer) and ESP (Stack Pointer) registers. Once we find these locations, we can input our payload into the ESP, and have the address of ESP in the EIP value so that our payload will execute.

To find the exact locations of the registers, we can try using smaller chunks of bytes to improve our precision. Let’s try using 500 bytes long strings as shown in Figure 6.

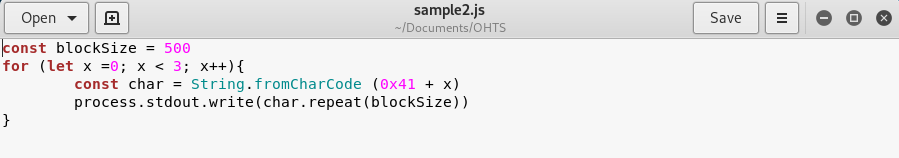


Figure 6 : Script to create string of 500-byte characters

Repeat the same process as in the previous step to generate the .m3u file, send it to the Windows XP Machine, and run it through the debugger. If we analyze the stack (Figure 7), we can see the ESP as 0x0012EF44 is filled with 43’s (C’s). We can also see that the C’s start 4 addresses before the ESP. This means there are 16 bytes of C’s in the buffer before we reach ESP.

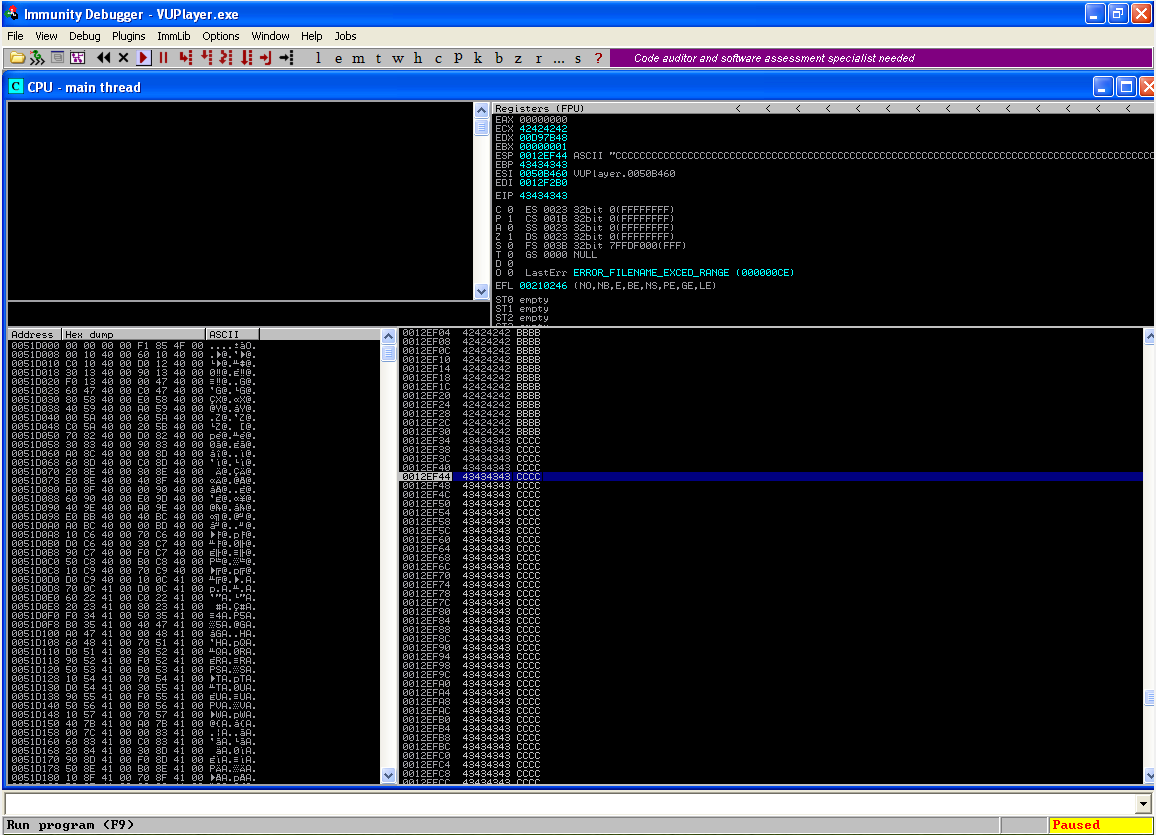


Figure 7 : Analyzing the 500 bytes overflow

To match this, let us now make our next .m3u file contain 1016 A’s, 1016 B’s, and 1016 C’s. Figure 8 below shows the modified script.

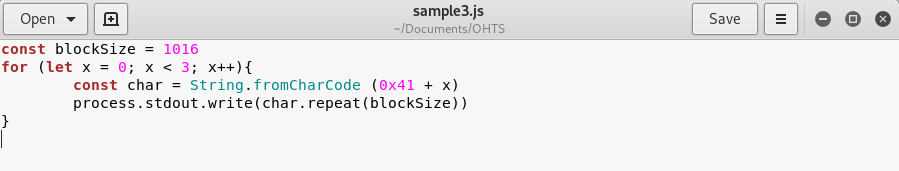


Figure 8 : Script with 1016-byte character strings

If we run the created .m3u file through the debugger, we are able to analyze the stack as follows (Figure 9).

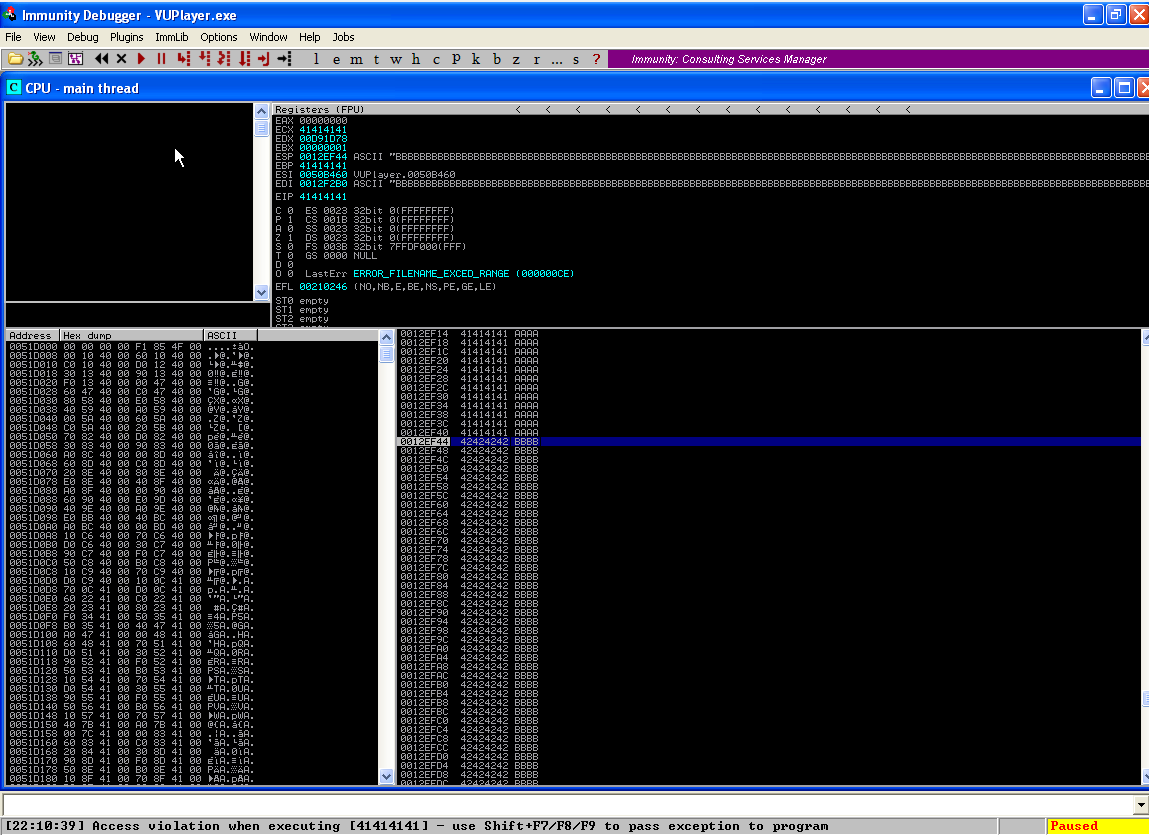


Figure 9 : Analyzing the 1016 bytes overflow

As shown in the above debugger output, when we changed the blocksize to 1016 bytes, we successfully hit ESP as expected by our calculations, as the ESP has been overwritten with 4242424242 (“BBBB”).

**Pointing the EIP to the address of ESP**

Now that we have identified the exact locations of EIP and ESP, we must now overwrite EIP with the address of ESP. But we cannot simply add the 0x0012EF44 address to EIP because the address starts with 0x00, which is a null character. If the program detects the null character it will stop the execution of the program.

To mitigate this issue, we must find an indirect method to jump to the ESP address. We can do this by finding an existing “JMP ESP” command in the program and overwrite EIP with that command’s address.

We can easily find the JMP ESP addresses used by the program in Immunity debugger, by using the command: “!searchcode jmp esp” as shown in Figure 10 below.

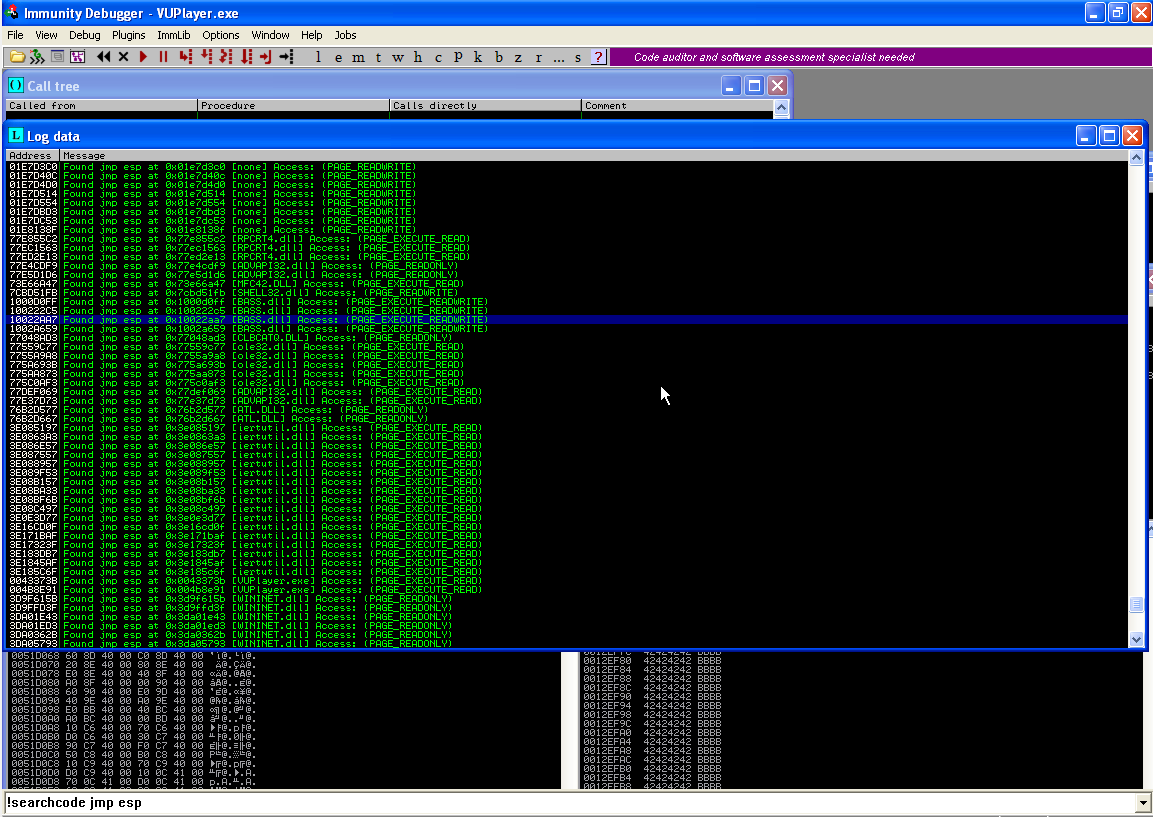


Figure 10 : Searching for JMP ESP commands used by the program

Now what we must do is find a JMP ESP address that does not contain any 0x00 null characters. In this walkthrough, we will be using the 0x10022aa7 address which is a JMP ESP command in the BASS.dll file.

Now let’s modify our script to overwrite EIP with the JMP ESP address. Since the stack works in the Last-In-First-Out order, we must enter the address in the reverse order for it to work accurately (Figure 11). After overwriting EIP, let us follow up with “DDDD” as a placeholder value for ESP. ESP will later be updated with our shellcode payload.



Figure 11 : Script to overwrite EIP with the JMP ESP address

This new script will create exploit5.m3u simply by executing the command “nodejs sample4.js” in the terminal.

Before running this new exploit through the debugger, let us first set a breakpoint at our JMP ESP line by pressing the F2 key so that we can analyze the stack at that point.

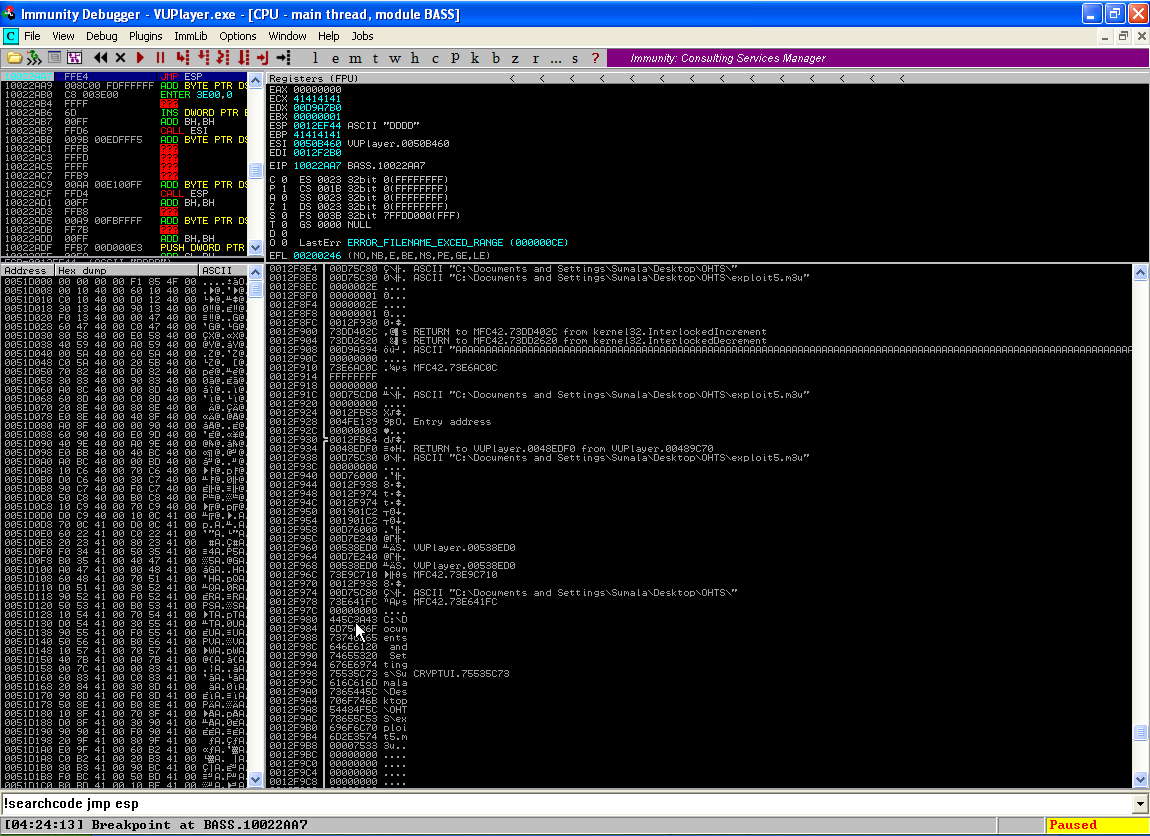


Figure 12 : Stack at the JMP ESP Breakpoint

After running the exploit, the program will pause at the breakpoint as shown in Figure 12 above. Here we can analyze the new register values. We can see that EIP is successfully overwritten with 10022AA7, which is our JMP ESP address, and ESP contains our placeholder value “DDDD”.

**Generating the Payload**

Now that we have set the EIP to point to the ESP address, the final step is to replace the placeholder value with our actual payload. To do this we can use the msfvenom command in Kali.

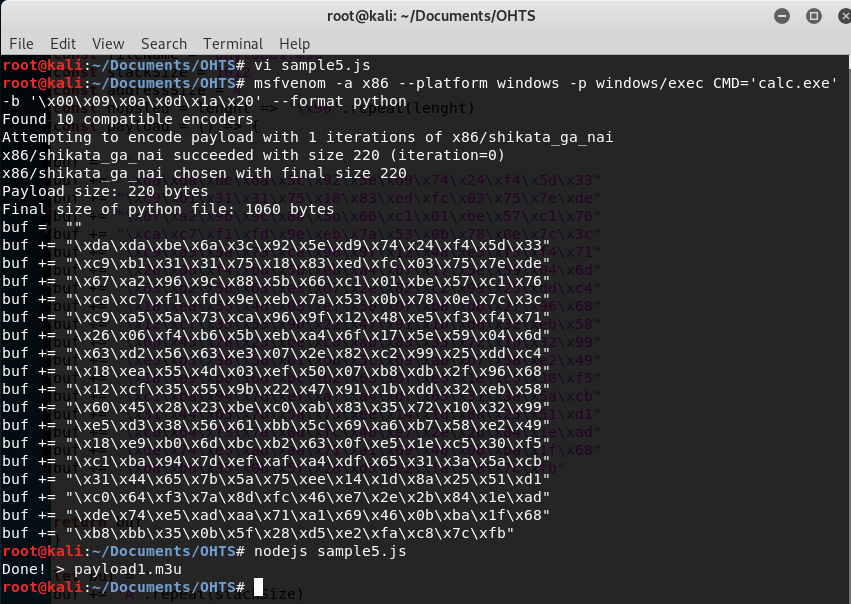


Figure 13 : Msfvenom command to generate payload (calculator)

Figure 13 above shows the commands used to generate the shellcode. In this example we will first generate a shellcode that executes the calculator application. We must use -a x86 to define the architecture of our target system which is a 32-bit Windows XP system. -p defines the type of payload, which in this example executes the calc.exe program. We must also use -b to omit all bad characters from the payload being generated. Running this command will generate the shellcode as shown above. Let us copy this shellcode and add it to our script (Figure 14).



Figure 14 : Script with calculator shellcode

Executing the above script will generate payload1.m3u which we send to our Windows XP machine. After running the payload through VUPlayer, we will see that the calculator application has been executed by the payload as shown in Figure 15 below.

A screenshot of a computer

Description automatically generated

Figure 15 : calculator application successfully executed when opening payload

With the successful execution of the calculator application, we can confirm that we have successfully created a buffer overflow exploit. Now let us create another shellcode to generate a Reverse TCP payload to obtain the root shell of our target PC. When we generate a Reverse TCP payload in our Kali machine, it will include information about our machine, so that when the Windows PC executes the payload, that PC will give our Kali machine permission to connect with its shell, giving us root privileges to the machine.

We will modify our previous msfvenom command as shown in Figure 16 below to generate the Reverse TCP payload.

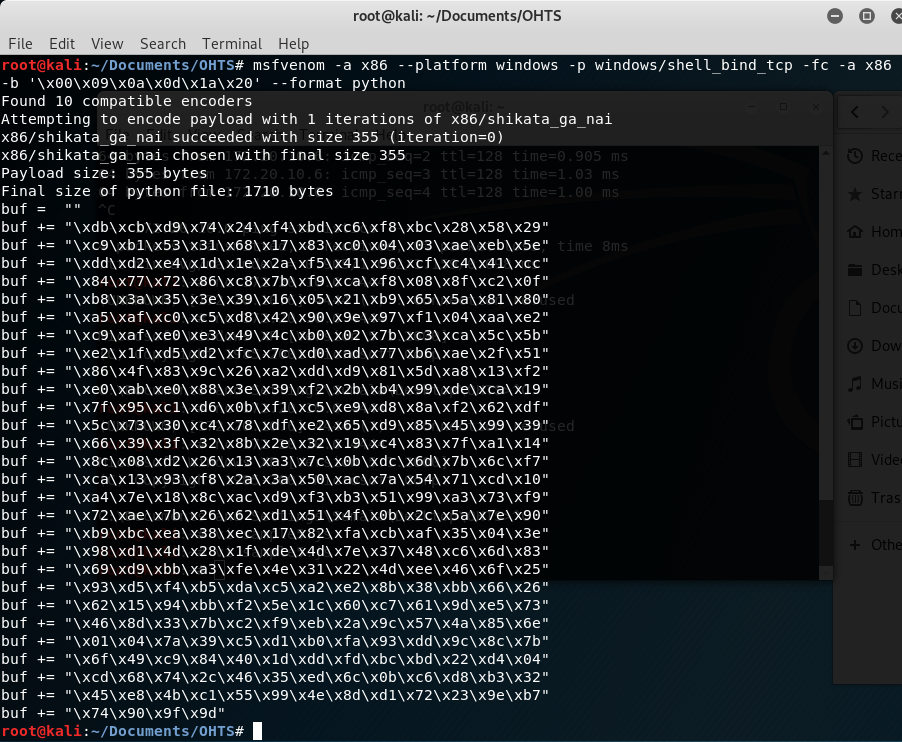


Figure 16 : msfvenom command to generate Reverse TCP payload

Now let’s copy this shellcode and replace our previous script with it (Figure 17)



Figure 17 : Script with reverse TCP payload

Now we execute this script to generate our final payload2.m3u and send it to the Windows XP machine. Before running the exploit, let’s see what happens when we try to connect to the windows machine without the payload.



Figure 18 : Attempting connection before execution of the payload

We attempt to connect to TCP port 4444 of the Windows machine, but the connection is refused as the Windows machine considers us as an untrusted source (Figure 18).

Now let’s run our exploit through VUPlayer and see if we can connect to the Windows machine.

A screenshot of a cell phone

Description automatically generated

Figure 19 : Attempting connection after execution of the payload

As shown in Figure 19 above, after the payload has been executed, the Windows machine has given us permission to connect to its root shell. Now we have successfully created a buffer overflow exploit to obtain the victim’s root shell.