To run the exploit generator, execute **python winamp-exploit.py**. My program will write the exploit string to a file called **mcvcore.maki** that is then ready to be deployed in the scripts folder.

To begin developing this exploit, I downloaded **winampx0.pl**, the sample exploit that was the bare minimum code needed to crash the Winamp program. Once downloaded, I copied the script to a Kali Linux virtual machine and executed **perl winampx0.pl > mcvcore.maki**, sending the output to a deployable file. I then copied this file to the Windows 7 virtual machine on which the Winamp program was installed. Navigating to the Winamp directory, **C:/Program Files/Winamp/**, I replaced the benign **mcvcore.maki** file in the **/Skins/Bento/scripts** directory with the one that I had just produced. Once copied, I opened the **WinDBG** program, selected **File > Open Executable**, and chose the **Winamp.exe** file to start a debugger on the process. Once the debugger was ready, I entered the 'g' command, causing the program to begin and, soon after, crash. With the concept now proven, I began writing my own exploit in Python. To start, I copied the header, types, function, and function length hexadecimal code from the **winampx0.pl**. The function name would be where I place the exploit code. The first phase would be to discover the offset of the **EIP**; to do so, I generated a pattern of **20,000 bytes** using **pattern_create.rb**. Placing this in the program created a **mcvcore.maki** file that I placed in the Winamp directory. This time, when the program crashed in **WinDBG**, the saved **EIP** was a value of **0x376d5636**. When entered into **pattern_offset.rb**, the calculated offset was **16,760 bytes**. This was half of the work complete. Now, I had to find the address of a **POP/POP/RETURN** instruction in one of the **DLL** files used by the program. To accomplish this, I used the **narly** tool (**!load narly**, **!nmod**)in the **WinDBG** program to first list all of the **DLL** files. With these **DLL** files now visible, I could select one of them that was compiled with **/SafeSEH** (not **NO_SEH**), but without ASLR or DEP. These DLLs were **gen_ff.dll**, **in_mp3.dll**, **nscrt.dll**, **out_ds.dll**, **tataki.dll**, and **zlib.dll**. Choosing **nscrt.dll**, I copied the file to Kali Linux where I used **msfpescan -p** to search the file for a **POP/POP/RETURN** instruction. This produced a list of all the matching instructions, from which I chose the first one (**0x7C3410C2 pop ecx; pop ecx; ret**). The near-complete exploit could now be generated. It's final form would be:

1. **16,756 bytes** of junk data (I used A's - **\x41**),
2. a **JUMP + 4** instruction (**\xEB\x04**) coupled with two bytes of junk **(\x41\x41**) that would be executed once the **POP/POP/RETURN** from the **DLL** was finished,
3. the address of this **POP/POP/RETURN** instruction (**\xC2\x19\x34\x7C**) ,
4. and lastly the payload which, instead of the shellcode that would be used to return a shell to the attacker machine, would be a few **\xCC** bytes to stop the debugger and confirm that I had succeeded.

This **mcvcore.maki** file was placed in the Winamp directory and, once executed, resulted in WinDBG stopping at the **\xCC** instruction - success. To finish up, I generated the shellcode needed to return a shell to an attacker listening from a machine at **10.247.49.140** on port **4444** and replaced the **\xCC** instruction with it.