**How To Execute From Command Line:**

perl winamp_infrastructure.pl > mcvcore.maki

Then, copy the mcvcore.maki file to the Winamp/Skins/Bento/scripts directory and launch Winamp. Make sure to enter nc -lp 4444 on the attacking machine.

**Exploit Results:**

My exploit works locally. It should work in the infrastructure since I generated the corresponding shellcode.

**How I developed my exploit:**

The first thing that I did when developing my exploit was to modify the sample winamp.pl file that was provided. I replaced the HelloWorld string with 20000 A's and generated the corresponding maki file. Then, I ran winamp using windbg and tried out the various debugger commands to make sure that I could see the A's overwriting the eip. I saw the rest of the A's using kb. After this, I used pattern_create.rb with length 20000 to crash the application and used the value that was in eip during the crash to find the offset using pattern_offset.rb with the same length (20000). I found the offset to be 16760. After doing this, I generated the shellcode in msfconsole for it to return a shell to my local machine. I used !load narly and !nmod to find a library. I chose the first one that I could find, which was in_line.dll. I copied it to my Kali machine (using scp after copying it to my mac) and used msfpescan -p in_line.dll to find the jump code (pop pop ret). Then, I structured the rest of the output to represent the following format: nops (or A's)/JMP+4/EIP(pop pop ret)/shellcode. Finally, I added the eb 04 in order to jump to the shellcode from nops as was illustrated on the slides. I tested this out on my local machine. It did not return a shell. So I used !exchain to to view the exception handler chain and saw that the eip was at 14b5eb04 (14b5 was part of the jump code address); I could see the nops were being loaded and causing the invalid exception stack. This meant that I had to modify my nops length because the proper value was not overwriting eip; I was over by two bytes. So I lowered the nops count to 16758 because I had initially forgotten to account for the eb 04. Once I did that, I tried it again. Although I still did not get a shell, this time I could see the right value (04eb9090) when using the !exchain command; this would cause the pop pop ret to be jumped over and my shellcode to execute. I debugged using !exchain, dd, and kb (to view call stack) and went through the slides to ensure that I did not miss anything regarding the structure of the exploit. The exception handler chain looked correct as it was correctly loaded with the content I was expecting. The slides led me to think that noSEH was what I wanted to select when picking a library. However, this is actually a stricter protection type/mitigation method. The update from Daniel helped me realize that my choice of library was not correct and I tried other libraries as Daniel suggested. I tried pop pop rets from a few other libraries including zlib.dll and tataki.dll, but kept running into the same issue; I could not get a shell on my local Kali machine. Next, I tried a pop pop ret from the nscrt.dll. This time when I ran Winamp, I got a shell on my local Kali machine. After verifying that it worked on my local Kali, I generated the new shellcode corresponding to the infrastructure's attack machine and port 4444, which should work.