نویسنده: *Lord_Viper*

Dynamic Injection

پروسه چیست؟ پروسه یک task میباشد که توسط سیستم عامل اجرا و مدیریت می شود .برای اینکه با پروسه ها بیشتر اشنا شوید دکمه های CTRL+ALT+DEL را با هم فشار دهید برنامه WindowsTaskManager باز خواهد شد به سربرگ processes بروید اینجا لیست کلیه پروسه هایی که بر روی سیستم در حال اجرا میباشند را نشان می دهد که image name نام پروسه و username نام کاربری ویندوز که برنامه را اجرا نموده می باشد

تعریف :

در ادبيات امنيت نرم افزار ، Process Injection به تزريق كد باينري به فضاي آدرسي پروسه هاي ديگه گفته ميشود . سيستمهاي عامل مدرني كه از مدل Protected Mode استفاده ميكنن ، براي هر پروسه با استفاده از تركيب حافظه حقيقي و مجازي ، فضاي آدرسي مستقلي رو تعريف ميكنن كه اجزاء يك پروسه (توابع ، متغيرها ، اشاره گر ها ، رفرنسها ، كتابخانه هاي اشتراكي و ...) داخل اون فضا آدرس دهي ميشوند . پردازنده ، به نوبت ، كد ماشين رو بصورت جداگانه از هر فضاي آدرسي دريافت و پردازش ميكنه . اگر شما كد ماشين يا متغير يا ساير موجوديتهاي باينري مورد نظرتان را ، بدون اينكه بطور مستقيم متعلق به يك پروسه باشن به فضاي آدرسي اون پروسه تزريق كنيد ، يك Process Injection انجام داده ايد.

مثال : شما برنامه A.exe رو نوشته ايد . يك برنامه به نام explorer.exe هم روي ويندوز وجود دارد . شما به هر دليل مايليد يكي از عناصر موجود در فضاي آدرسي explorer.exe رو توسط A.exe تغيير بديد . Process Injection يعني تزريق كد مورد نظر از فضاي A.exe به فضاي explorer.exe .

كاربرد:

كاربرد اصلي و عام Process Injection ، توسعهء برنامه هاي مرتبط با Access Control است . برنامه هاي مثبت ، با استفاده از اين تكنيك و با تزريق يك كد محافظ به فضاي آدرس پروسه هاي سرور يا سرويس هاى (inetinfo.exe يا svchost.exe و ...) سعي ميكنن اين پروسه ها رو از گزند تلاشهاي مخرب و كدهاي مخرب (Exploit) ايمن نگه دارن . برنامه هاي منفي مثل كرم ها و تروجان ها با استفاده از Process Injection به مقاصد شون مثل

عبور از فايروالهاي شخصي و فريب دادن آنتي ويروسها خواهند رسيد .

مثال مثبت : زون آلارم(ZoneAlarm) ، با تزريق كد كه اجراي كد باينري از طريق بخش Data ي Stack رو ممنوع ميكنه ، به داخل فضاي پروسه هائي مثل Services.exe از اونها نگهداري ميكنه . اين يه نمونه كاربرد مثبت Injection Process است .

مثال منفي : اغلب فايروالهاي شخصي ، Access Control رو مبتني بر پروسهء فراخوان كد انجام ميدهند . يعني اگر فراخواني كدي از طريق پروسهء قابل اعتماد IExplorer.exe (مرورگر ويندوز) انجام شده باشه ، فايروال اجازه ايجاد اتصال شبكه رو ميده و اگر نه ، خير . يك تروجان ، براي اتصال به شبكه ، با وجود يك فايروال شخصي توفيق چنداني نخواهد داشت ، اما با استفاده از يك Process Injection ساده و تزريق كد باينري مورد نظر براي ايجاد اتصال شبكه اي ، به فضاي پروسهء IExplorer.exe ميتونه فايروال رو دور بزنه .

چند نمونه از كاربرد هاى مختلف آن:

ويندوز ۲۰۰۳ سرور به عنوان يكي از معدود سيستمهاي عاملي كه روي IA32 ميتونه تا حدود ٤ گيگ آدرس دهي بكنه ، هم از Process Injection براي افزايش كيفيت و كارائي اش استفاده كرده . ويژگي HotPatch موجود در ويندوز ۲۰۰۳ دقيقا" از همين قابليت استفاده ميكنه . با تشكر از اين تكنيك ، در صورت كشف يك نقطه ضعف امنيتي روي يك سرويس خاص (مثلا" IIS 6) شما براي نصب Patch اون لازم نيست سرويس رو متوقف كنين . Patch با استفاده از سرويس HotPatch يك نسخه از خودش رو روي Image سرويس در هارد ديسك بازنويسي ميكنه و يك نسخه از تصحيحات لازم رو بصورت زنده (Process Injection) روي سرويس در حال اجرا و سرويسدهي اعمال ميكنه . به اين ترتيب دفعه بعدي كه سرويس اجرا ميشه ، از يك نسخه امن استفاده ميكنه مضاف بر اينكه ، همين حالا هم با اعمال زندهء تغييرات روي نسخهء در حال اجراي سرويس در فضاي آدرسي مخصوص به خودش ، نقطه ضعف امنيتي بر طرف شده .

مايكروسافت براي كساني كه ميخوان از اين ويژگي بصورت جدي استفاده كنن يك Calss Library مبتني بر COM به رايگان منتشر كرده كه با جستجو روي سايت مايكروسافت ميتوان انرا يافت. dynamic Injection هنگامی که پروسه هدف در حالت اجرا میباشد انجام میشود بر خلاف static injection که بر روی فایل هدف انجام میشد .هنگامی که یک برنامه اجرا می شود یک پروسه توسط سیستم عامل ساخته می شود هنگامی که نفوظ گر سعی در بارگزاری کد خود درون فضای ادرسی پروسه دیگر کند نفوظ گر Dynamic injection انجام داده است

Dynamic injection بیشتر توسط تروجانها مورد استفاده قرار می گیرد و شامل ۲ بخش می باشد DllInjection و CodeInjection که در حالت اول کد مورد نظر درون یک dll قرار گرفته و به پروسه هدف تزریق می شود و در حالت دوم کد باینری درون برنامه بصورت مستقیم درون پروسه هدف تزریق می شود.مزیت dll injection به code injection در این میباشد که dll injection بسیار استتر می باشد زیرا هنگامی که یک dll در حافظه بارگزاری می شود تمام توابع مورد نیاز برای اجرای ان نیز توسط loader در حافظه بارگزاری می شود که این حالت در code injection صدق نمیکند و اگر ما از دستورات یا توابعی که درون

پروسه هدف وجود ندارد استفاده کرده باشیم باعث ایجاد exception و در نتیجه بسته شدن برنامه هدف می شود که البته برای جلوگیری از این مشکل نیز راه هایی وجود دارد
حال ما شروع به تزریق یک dll به فضای ادرسی پروسه دیگرمثلا Explorer میکنیم برای شروع نیاز به برنامهDiamondCS APM (Advanced Process Manipulation) داریم که می توانید از لینک زیر دریافت کنید

http://diamondcs.com.au/index.php?page=apm

برنامه APM را نصب کنید و اجرا کنید خواهید دید که لیستی از پروسه های در حال اجرا به همراه PID (Process Id Number)شان به شما نشان خواهد داد این برنامه لیستی از dll هایی که توسط برنامه ها فراخوانی شده و در حال اجرا میباشد نیز به شما نشان خواهد داد و میتوانید هر کدام را که مایل بودید از حافظه خارج کنید Microsoft's Platform SDK
تعدادی توابع برای کار با پروسه ها قرار داده که ما به برسی تعدادی که مهمترو برای انجام این کار به انها نیاز داریم می پردازیم پس به خوبی انها را یاد بگیرید

OpenProcess : برای بدست اوردن هندل یک شیی پروسه در حال اجرا
LoadLibrary : بارگزاری یک dll یا یک module اجرایی درون فضای ادرسی پروسه فراخوان تابع
VirtualAllocEx : ایجاد یک فضای خالی به اندازه دلخواه در فضای ادرسی پروسه دیگر
WriteProcessMemory : نوشتن اطلاعات درمحدوده فضای ادرسی پروسه دیگر
CreateRemoteThread : ایجاد یک ریسمان و اجرای ان در فضای ادرسی پروسه دیگر

اکنون به برسی دقیق تر این توابع می پردازیم

```
HANDLE OpenProcess(

   DWORD dwDesiredAccess,
   BOOL bInheritHandle,
   DWORD dwProcessId
  );
```

dwDesiredAccess:مشخص کننده نوع دسترسی به شیی پروسه میباشد. که می تواند یکی از موارد زیر یا ترکیبی از انها باشد

PROCESS_ALL_ACCESS
PROCESS_CREATE_PROCESS.
PROCESS_CREATE_THREAD
PROCESS_DUP_HANDLE
PROCESS_QUERY_INFORMATION
PROCESS_SET_INFORMATION

PROCESS_TERMINATE
PROCESS_VM_OPERATION
PROCESS_VM_READ
PROCESS_VM_WRITE
SYNCHRONIZE

bInheritHandle: مشخص میکند که ایا مقداری که توسط این تابع برگشت داده شد میتواند توسط پروسه جدید مورد استفاده قرار گیرد(به ارث برده شود)اگر مقدار true باشد مورد استفاده قرار میگیرد
dwProcessId:شناسه پروسه ای که میخواهیم هندل انرا به دست اوریم (PID پروسه مورد نظر)
مقداری که این تابع بر میگرداند هندل پروسه مورد نظر می باشد

```
HINSTANCE LoadLibrary(

   LPCTSTR lpLibFileName
 );
```

lpLibFileName : ادرس dll مورد نظر برای نگاشت شدن در حافظه برنامه فراخوان
مقداری که این تابع برمیگرداند یک هندل به ماژول مورد نظر می باشد

```
LPVOID VirtualAllocEx(

   HANDLE hProcess,
   LPVOID lpAddress,
   DWORD dwSize,
   DWORD flAllocationType,
   DWORD flProtect
  );
```

hProcess:هندل پروسه هدف که میخواهید محدوده خالی در فضای ادرسی ان ایجاد کنید
lpAddress:یک شاره گر به نقطه شروع محدوده صفحاتی که میخواهید اختصاص دهید اگر مقدار نرا null در نظر بگیرید نقطه شروع محدوده را خود تابع تایین میکند
dwSize:سایزمحدوده ای را که میخواهید اختصاص دهید به مقدار ان به بایت میباشد
flAllocationType: یک glag bit میباشد که نوع حافظه اختصاص داده شده را معیین میکند که میتواند یکی از مقادیر زیر باشد

MEM_COMMIT
MEM_RESERVE

MEM_TOP_DOWN
flProtect:یک bit flag میباشد که نوع دسترسی به محدوده حافظه ای که ایجاد کردید می باشد که میتواند یکی از مقادیر زیر باشد

PAGE_READONLY
PAGE_READWRITE
PAGE_EXECUTE
PAGE_EXECUTE_READ
PAGE_EXECUTE_READWRITE
PAGE_GUARD
PAGE_NOACCESS
PAGE_NOCACHE

مقداری که این تابع برمیگرداند Base Address محدوده حافظه اختصاص داده شده می باشد

```
BOOL WriteProcessMemory(

   HANDLE hProcess,
   LPVOID lpBaseAddress,
   LPVOID lpBuffer,
   DWORD nSize,
   LPDWORD lpNumberOfBytesWritten
  );
```

hProcess:هندل پروسه هدف که میخواهید در محدوده فضای ادرسی ان بنویسید
Base Address:lpBaseAddress محلی که میخواهید از انجا شرع به نوشتن کنید
lpBuffer:اشاره به buffer ی که اطلاعات توسط ان برای نوشتن منتقل می شود
nSize:مشخص کننده سایز اطلاعاتی که باید نوشته شوند . سایز به بایت میباشد
lpNumberOfBytesWritten:شاره به مقدار بایت نوشته شده در پروسه هدف این مقدار اختیاری می اشد
مقدار این تابع در حالت اجرای درست مقدار غیر ۰ میباشد

```
HANDLE CreateRemoteThread(

   HANDLE hProcess,
   LPSECURITY_ATTRIBUTES lpThreadAttributes,
   DWORD dwStackSize,
   LPTHREAD_START_ROUTINE lpStartAddress,
   LPVOID lpParameter,
   DWORD dwCreationFlags,
```

```
  LPDWORD lpThreadId
  );
```

hProcess:هندل پروسه ای که ریسمان در ان ایجاد می شود
lpThreadAttributes:اشاره کری به SECURITY_ATTRIBUTES که مشخص کننده security ریسمان میباشد اگر مقدار ان null باشد ریسمان از مقدار SECURITY_ATTRIBUTES پیش فرض استفاده میکند
dwStackSize:مشخص کننده اندازه stack ریسمان جدید این اندازه به بایت میباشد مقدار پیش فرض ان به اندازه ریسمان اولیه برنامه میباشد
lpStartAddress:اشاره به ادرس شروع ریسمان میباشد
lpParameter:یک مقدار ۳۲بیتی می باشد که به ریسمان ارسال می شود
dwCreationFlags:یک flag برای کنترل روند ساخت ریسمان می باشد اگر مقدار ان CREATE_SUSPENDED باشد یعنی ریسمان در حالت suspend ساخته شده و تا تابع ResumeThread اجرا نشود ریسمان اجرا نخواهد شد
lpThreadId:اشاره به یک متغییر ۳۲ بیتی که شناسه ریسمان به ان ارسال خواهد شد مقداری که این تابع باز میگرداند هندل ریسمان ایجاد شده می باشد

چگونه یک dll به پروسه دیگر تزریق کنیم

یک برنامه هنگامی که اجرا می شود میتواند با طی کردن مراحل زیر یک dll را درون فضای ادرسی پروسه دیگر بارگزاری کند
باز کردن پروسه بوسیله OpenProcess با استفاده از PID پروسه مورد نظرهندل پروسه هدف را به دست می اوریمبعد از ان نوبت به تخصیص حافظه توسط تابع VirtualAllocE در فضای ادرسی پروسه ای که با استفاده از OpenProcess هندل انرا بدست اوردیم و سپس نوشتن اطلاعات مورد نظر در فضای خالی تخصیص داده در فضای ادرسی پروسه مورد نظر با استفاده از تابع WriteProcessMemory و دادن ادرس فضای تخصیص داده و یک اشاره گر به dll یا کد مورد نظرکه میخواهیم در ان بارگزاری کنیم و سپس یک ریسمان جدید ایجاد میکنیم تا تابع یا dll ما را اجرا کند که ادرس تابع مورد نظر همان ادرس تابع LoadLibrary ادرس فضای تخصیص داده شده در پروسه هدف را به ریسان میدهیم سپس با اجرای ریسمان کد ما در پروسه هدف اجرا می شود

حال به کد زیر توجه کنید

کد زیر را درون یک dll قرار دهید

```
library CommandDll;

uses
  Windows,
  SysUtils,
```

```
  Classes;

{$R *.RES}

procedure EntryPointProc(reason: integer);
var
  str: pChar;
begin
  case reason of
    DLL_PROCESS_ATTACH:
      begin
        str := GetCommandLine;
        MessageBox(0, 'This is a DynamicDllInjection Sample', 'Lord_Viper',
0);
        MessageBox(0, 'WwW.xexample.CoM', 'Lord_Viper', 0);
        MessageBox(0, str, 'Lord_Viper', 0);
      end;
    DLL_PROCESS_DETACH:
      begin
      end;
    DLL_THREAD_ATTACH:
      begin
      end;
    DLL_THREAD_DETACH:
      begin
      end;
  end;
end;

begin
  DLLProc := @EntryPointProc;
  EntryPointProc(DLL_PROCESS_ATTACH);
end.
```

حال اولین قدم برای تزریق کد بدست اوردن PID پروسه هدف می باشد که با استفاده از توابع موجود در کتابخانه TLhelp32 می توان به دست اورد به کد زیر توجه کنید

```
Uses
tlhelp32
```

```
function PidProcesso(NomeProcesso: string): LongWord;
var
  pe: TProcessEntry32;
  hSnap: THandle;
begin
  hSnap := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

  pe.dwSize := sizeof(TProcessEntry32);

  //Prelevo informazioni sul primo processo nello snapshot di sistema
  Process32First(hSnap, pe);
  repeat  //loop sui processi
    Result := pe.th32ProcessID;
    if (pe.szExeFile = NomeProcesso) then
      begin
        break;
      end;
  until (not (Process32Next(hSnap, pe)) ) ;

  CloseHandle(hSnap);

end;
```

برای اینکه بتوانیم در فضای ادرسی ایجاد شده در پروسه هدف کد خود را بنویسیم باید مجوز لازم برای این کار را داشته باشیم در غیر این صورت عملیات تزریق با شکست مواجه خواهد شد که برای افزایش دسترسی از توابع مربوط به Privilage استفاده میکنیم

```
procedure ModificaPrivilegio(szPrivilege: pChar; fEnable: Boolean);
var
  NewState: TTokenPrivileges;
  luid: TLargeInteger;
  hToken: THandle;
  ReturnLength: DWord;

begin
  OpenProcessToken(
                GetCurrentProcess(),
                TOKEN_ADJUST_PRIVILEGES,
                hToken);
```

```
  LookupPrivilegeValue(nil, szPrivilege, luid);
  NewState.PrivilegeCount := 1;
  NewState.Privileges[0].Luid := luid;
  if fEnable then
    NewState.Privileges[0].Attributes := SE_PRIVILEGE_ENABLED
  else
    NewState.Privileges[0].Attributes := 0;
  AdjustTokenPrivileges(hToken, FALSE, NewState,
                        sizeof(NewState), nil, ReturnLength);
  CloseHandle(hToken);

end;
```

حالا باید تابع اصلی خود را بنویسیم که dll مورد نظر را در فضای ادرسی پروسه مورد نظر مان اجرا کند تابع اصلی ما اینگونه خواهد بود

```
procedure InjectDll(PID: dword;  DLL: pChar);
var
  BytesWritten, Process, Thread, ThreadId: dword;
  Paramaters: pointer;
begin

  ModificaPrivilegio('SeDebugPrivilege', TRUE);

  Process := OpenProcess(PROCESS_CREATE_THREAD +
                         PROCESS_QUERY_INFORMATION +
                         PROCESS_VM_OPERATION +
                         PROCESS_VM_WRITE +
                         PROCESS_VM_READ, False, PID);

  Paramaters := VirtualAllocEx(Process, nil, Length(DLL),
MEM_COMMIT, PAGE_READWRITE);

  WriteProcessMemory(Process, Paramaters, Pointer(DLL),
Length(DLL), BytesWritten);

  Thread := CreateRemoteThread(Process, nil, 0,
GetProcAddress(GetModuleHandle('KERNEL32.DLL'), 'LoadLibraryA'),
                             Paramaters, 0, ThreadId);
```

```
  WaitForSingleObject(Thread, INFINITE);

  VirtualFreeEx(Process, Paramaters, 0, MEM_RELEASE);

  CloseHandle(Thread);
  CloseHandle(Process);

end;
```

با اجرای برنامه و تزریق dll به پروسه هدف ۳ مسیج باکس نمایش داده خواهد شد که نماینده موفقیت امیز بودن روند تزریق می باشد

چگونه کد برنامه مان را به پروسه دیگر تزریق کنیم

این کار هم همانند تزریق dll می باشد اما با کمی تفاوت که در اینجا تابع مورد نظر درون dll قرار نمیگیرد بلکه درون خود برنامه نوشته می شود و به جای dll ادری تایع به ریسمان مورد نظر برای اجرا ارسال می شود
Get module handle این تابع هندل ماژول مورد نظر ما را برمیگرداند

```
HMODULE GetModuleHandle(

   LPCTSTR lpModuleName
  );
```

lpModuleName:اشاره به یک ارایه کاراکتری(pchar) میباشد که مشخص کننده نام ماژول میباشد(dll یا exe) اگر نام ماژول را null در نظر بگیرید هندل ماژول پروسه ما را میدهد(پروسه ای که این تابع را فراخوانی کرده است)
خب تابع اصلی ما برای تزریق به صورت زیر خواهد بود

```
procedure injectcode();
begin
      MessageBox(0, 'This is a DynamicCodeInjection Sample',
'Lord_Viper', 0);
      MessageBox(0, 'WwW.xexample.CoM', 'Lord_Viper', 0);
end;
```

حالا به پیاده سازی تابع اصلی برای تزریق می رسیم که به صورت زیر خواهد بود

```
procedure Inject(ProcessHandle: longword; EntryPoint: pointer);
var
  Module, NewModule: Pointer;
  Size, BytesWritten, TID: longword;
begin
  Module := Pointer(GetModuleHandle(nil));
  Size := PImageOptionalHeader(Pointer(integer(Module) +
PImageDosHeader(Module)._lfanew + SizeOf(dword) +
SizeOf(TImageFileHeader))).SizeOfImage;
  VirtualFreeEx(ProcessHandle, Module, 0, MEM_RELEASE);
  NewModule := VirtualAllocEx(ProcessHandle, Module, Size,
MEM_COMMIT or MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
  WriteProcessMemory(ProcessHandle, NewModule, Module, Size,
BytesWritten);
  CreateRemoteThread(ProcessHandle, nil, 0, EntryPoint, Module, 0, TID);
end;
```

و برای استفاده از کد فوق به صورت زیر عمل میکنیم

```
procedure TForm1.Button1Click(Sender: TObject);
var
  StartupInfo: TStartupInfo;
  ProcessInfo: TProcessInformation;
begin
  FillMemory(@StartupInfo,SizeOf(StartupInfo),0);
  StartupInfo.cb:=SizeOf(StartupInfo);
  CreateProcess(nil, 'notepad.exe', nil, nil, False, 0, nil, nil, StartupInfo,
ProcessInfo);
  Inject(ProcessInfo.hProcess, @ injectcode);
end;
```

با اجرای برنامه و کلیک روی button برنامه notepad.exe اجرا شده و مسیج های مورد نظر را نشان میدهد

بخش زیر برای ایجاد پروسه مورد نظر در صورت نبود در حافظه استفاده شده است و شما میتوانید به جای notepad.exe اسم پروسه مورد نظر خود یا ادرس انرا بنویسید

```
CreateProcess(nil, 'notepad.exe', nil, nil, False, 0, nil, nil, StartupInfo,
ProcessInfo);
```

درمورد پروسه های فعال نظیر Explorer یا Svchost یا Winlogon که همیشه در حال اجرا هستند نیازی به کد فوق نیست و در صورت لزوم می تواید از تابع زیر که قبلا گفته شده برای بدست اوردن هندل انها استفاده کنید

```
function PidProcesso(NomeProcesso: string): LongWord;
```

همچنین مقداری که InjectSize بر میگرداند اندازه کد ما برای تزریق می باشد
گاهی اوقات در تزریق مستقیم کد ممکن است با مشکل برخورد کنیم و ان هم نبود توابع استفاده شده در کدمان در لیست توابع Import شده پروسه مقصد می باشد زیرا در تزریق dll تمامی توابع لازم برای اجرای dll توسط سیستم عامل بارگزاری می شود اما در تزریق مستقیم کد چنین نیست در این حالت برای جلوگیری از ایجاد exception به دلیل نبود توابع میتوان توابع را به صورت dynamic فراخوانی کرد یا کتابخانه های مورد نظر را در صورت عدم وجود داخل کد در زمان اجرا بارگزاری کرد با استفاده از تابع LoadLibrary و GetProcAddress
خروجی تابع GetProcAddress ادرس تابع export شده مورد نظر در یک dll می باشد

تا کنون در مورد روشهای تزریق صحبت کردیم حال چگونه می توان پروسه ای که مورد حمله تزریق قرار گرفته را پاک سازی کرد
میتوان با چک کردن ماژولهای بارگزاری شده درون پروسه و یافتن ماژول تزریق شده انرا از فضای ادرسی پروسه هدف پاک نمود به کد زیر توجه کنید

```
Uses
Tlhelp32

procedure UninjectDll(dllname : String; PID : Integer);
var
hProc : Cardinal;
hSnap : Cardinal;
module : moduleEntry32;
modstring : String;
hModule : PByte;
hremThread : Cardinal;
begin
hSnap := CreateToolHelp32Snapshot(TH32CS_SNAPMODULE,PID);
module.dwSize := sizeof(MODULEENTRY32);
If Module32First(hSnap,module) = True then begin
```

```
  While Module32Next(hSnap,module) do begin
     If module.szModule = dllname Then begin
       hModule := module.modBaseAddr;
     end;
  end;
end;
hProc := OPenProcess(PROCESS_ALL_ACCESS,false,PID);
CreateRemoteThread(hProc,nil,0,
                GetProcAddress(GetModuleHandle('Kernel32.dll'),
'FreeLibrary'),hModule,0,hremThread);
CloseHandle(hProc);
CloseHAndle(hSnap);
end;
```

در کد فوق ابتدا با استفاده از توابع api نظیرModule32First و Module32Next لیست ماژولهای پروسه هدف چک می شوند سپس با استفاده از modBaseAddr ادرس ماژول در فضای ادرسی پروسه هدف بدست میاید و با استفاده از یک ریسمان راه دور و استفاده از freelibrary به عنوان تابع و پاس دادن ادرس ماژول به عنوان پارامتر(hModule) به پروسه مقصد dll تزریق شده از حافظه پروسه مقصد خارج خواهد شد

پایان

WwW.xexample.CoM

Ghoghnoose.dana@Gmail.CoM