26 - 28 NOVEMBER 2024
RIYADH, SAUDI ARABIA

.NET ALCHEMY
DISCOVERING PRIMITIVES FOR OFFENSIVE TOOLING

KYLE AVERY

ORGANISED BY:

IN ASSOCIATION WITH:

PRESENTATION SPONSOR:

WHOAMI

Kyle Avery

- Offensive Developer @ Outflank
- Red team background / .NET and C development
- Lone US Outflank member

AGENDA

Memory Allocation

- Indirect creation of executable memory

Deserialization Bugs

- Exploiting “intentional” vulnerabilities

Unmanaged Control

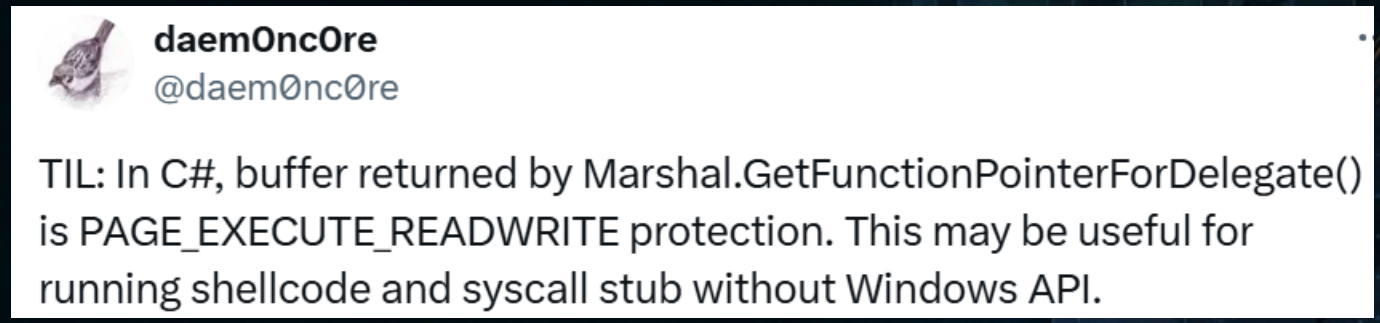
- Resolving and patching managed functions

MEMORY ALLOCATION

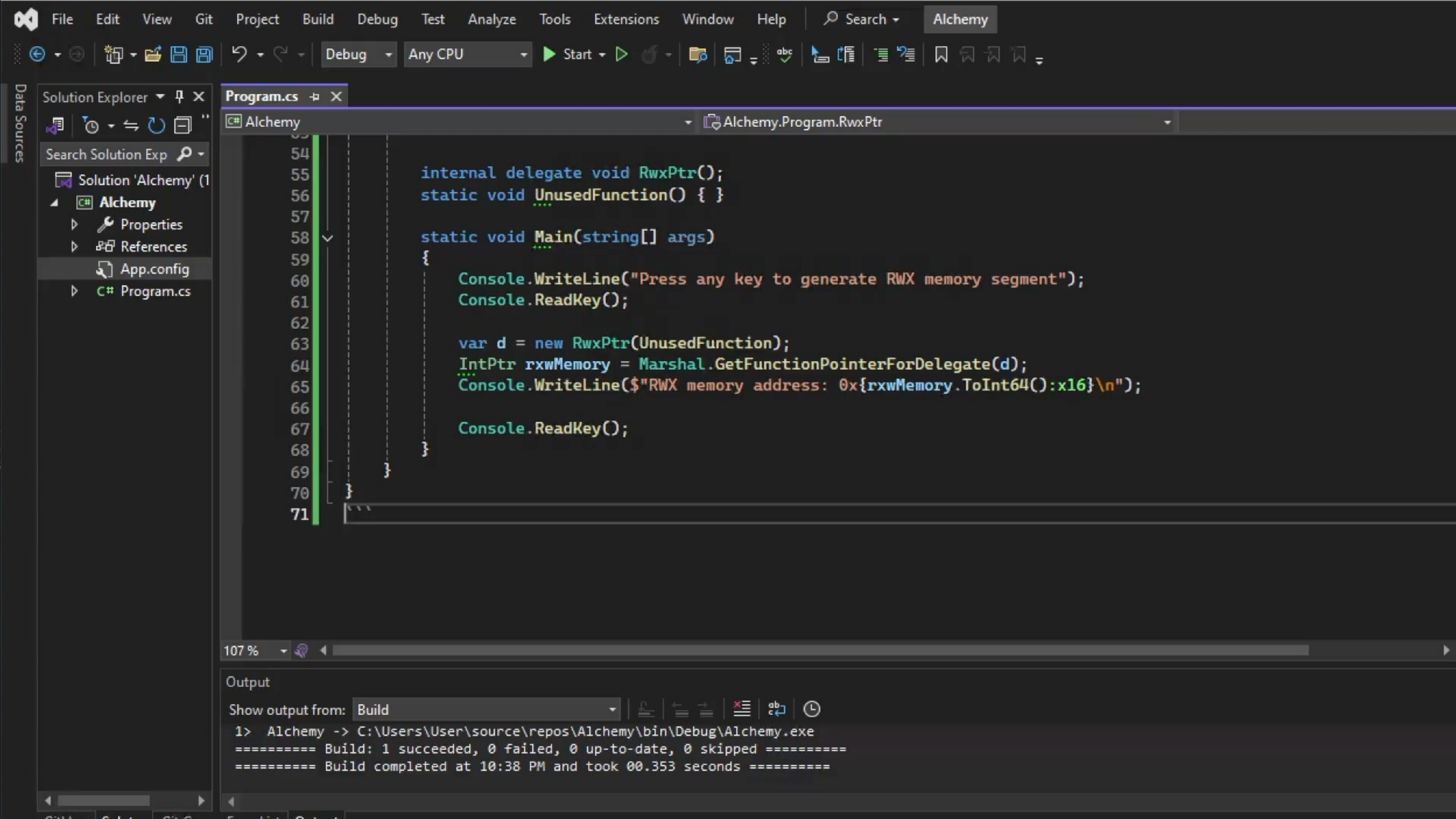
INDIRECT CREATION OF EXECUTABLE MEMORY

INDIRECT ALLOCATION

- Calling **`GetFunctionPointerForDelegate`** creates 5-6KB of RWX memory
  - Does not work reliably for x86
  - Too small for most shellcode




```
internal delegate void RWXPtr();
void UnusedFunction() {}

var d = new RWXPtr(UnusedFunction);
IntPtr rxwMemory = Marshal.GetFunctionPointerForDelegate(d);
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Alchemy
Debug
Any CPU
Start
Solution Explorer
Search Solution Exp
Solution 'Alchemy' (1
Alchemy
Properties
References
App.config
Program.cs
Data Sources
Program.cs
Alchemy
Alchemy.Program.RwxPtr
internal delegate void RwxPtr();
static void UnusedFunction() { }
static void Main(string[] args)
{
Console.WriteLine("Press any key to generate RWX memory segment");
Console.ReadKey();
var d = new RwxPtr(UnusedFunction);
IntPtr rxwMemory = Marshal.GetFunctionPointerForDelegate(d);
Console.WriteLine($"RWX memory address: 0x{rxwMemory.ToInt64():x16}\n");
Console.ReadKey();
}
}
}
107 %
Output
Show output from: Build
1> Alchemy -> C:\Users\User\source\repos\Alchemy\bin\Debug\Alchemy.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 10:38 PM and took 00.353 seconds ==========

INDIRECT ALLOCATION

- Alternative from SharpHellsGate:

```
static UInt32 Gate() {
    return new UInt32();
}

bool GenerateRWXMemorySegment() {
    BindingFlags gateFlags = BindingFlags_Public | BindingFlags_Instance;
    MethodInfo method = GetMethod(nameof(Gate), gateFlags);

    RuntimeHelpers.PrepareMethod(method.MethodHandle);
    IntPtr pMethod = method.MethodHandle.GetFunctionPointer();
}
```

INDIRECT ALLOCATION

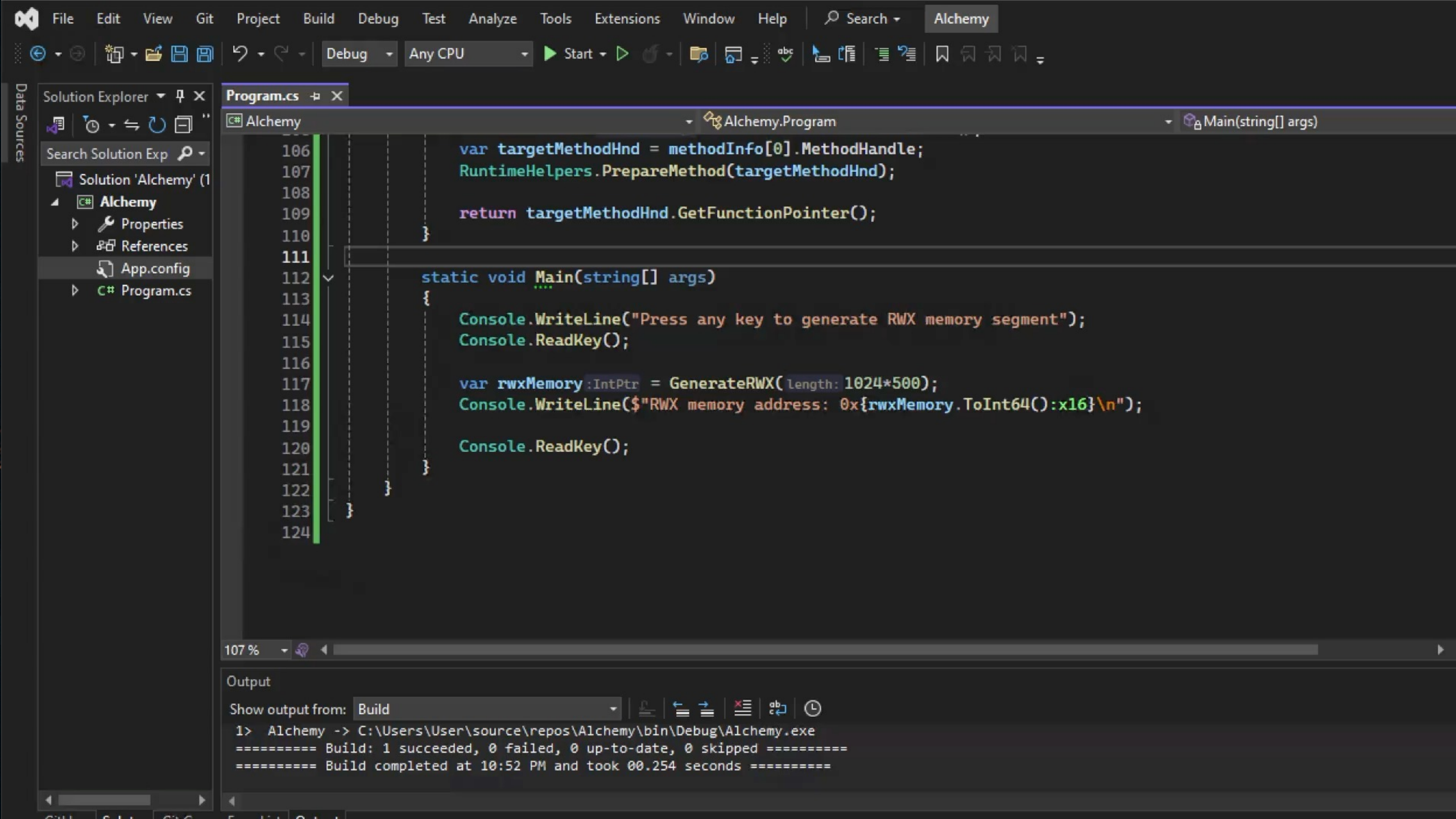
- Can we increase the size of JIT memory?
  - Yes! **Make the method larger** – crude but functional

```
static UInt32 Gate() {
    Console.WriteLine("A");
    Console.WriteLine("B");
    Console.WriteLine("C");
    ...

    return new UInt32();
}
```

INDIRECT ALLOCATION

- Can we make the method larger at runtime?
  - Dylan Tran (@d_tranman) solved this by defining a dynamic assembly, module, and method

```
var name = new AssemblyName("FakeAssembly");
var ad = System.AppDomain.CurrentDomain;
var asm = ad.DefineDynamicAssembly(name, AssemblyBuilderAccess.Run);
var bld = asm.DefineDynamicModule("FakeModule");
var mtd = bld.DefineGlobalMethod("FakeMtd", Static, typeof(void), new Type[]{});

ILGenerator il = mtd.GetILGenerator();
il.EmitWriteLine("AAAA");
il.Emit(OpCodes.Ret);
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Alchemy
Debug
Any CPU
Start
Solution Explorer
Search Solution Exp
Solution 'Alchemy' (1
Alchemy
Properties
References
App.config
Program.cs
Program.cs
Alchemy
Alchemy.Program
Main(string[] args)
var targetMethodHnd = methodInfo[0].MethodHandle;
RuntimeHelpers.PrepareMethod(targetMethodHnd);
return targetMethodHnd.GetFunctionPointer();
}
static void Main(string[] args)
{
Console.WriteLine("Press any key to generate RWX memory segment");
Console.ReadKey();
var rwxMemory :IntPtr = GenerateRWX(length: 1024*500);
Console.WriteLine($"RWX memory address: 0x{rwxMemory.ToInt64():x16}\n");
Console.ReadKey();
}
}
}
107 %
Output
Show output from: Build
1> Alchemy -> C:\Users\User\source\repos\Alchemy\bin\Debug\Alchemy.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 10:52 PM and took 00.254 seconds ==========

DESERIALIZATION BUGS

EXPLOITING “INTENTIONAL” VULNERABILITIES

SERIALIZATION INTRODUCTION

- **Serialization** – Convert object in memory to a format that can be saved or transferred

- **Formatters** – .NET classes that serialize/deserialize objects:
  - `BinaryFormatter`
  - `SoapFormatter`
  - `XMLSerializer`
  - `JavaScriptSerialzier`
  - `NetDataContractSerializer`
  - …

SERIALIZATION INTRODUCTION

- **Deserialization** – Object is decoded and then its constructor is executed

- **Vulnerabilities** – Unsanitized data is passed to deserializer
  - Introduced by James Forshaw at Black Hat USA 2012
  - RCE methods discussed at Black Hat USA 2017

EXPLOITING “INTENTIONAL BUGS”

DotNetToJScript – Generate VBScript/JScript that loads a .NET assembly

- Uses **`BinaryFormatter`** to deserialize a **`delegate`**
- Works for VBScript/JScript because there are no CAS restrictions

Alternative: **GadgetToJScript**

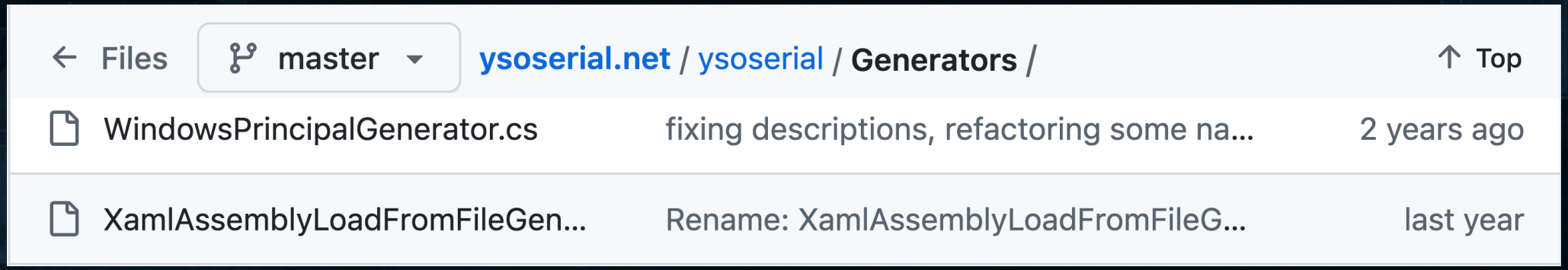
- Uses **`BinaryFormatter`** and **`ActivitySurrogateSelector`** gadget
- Must use an additional gadget to disable type protections in .NET 4.8

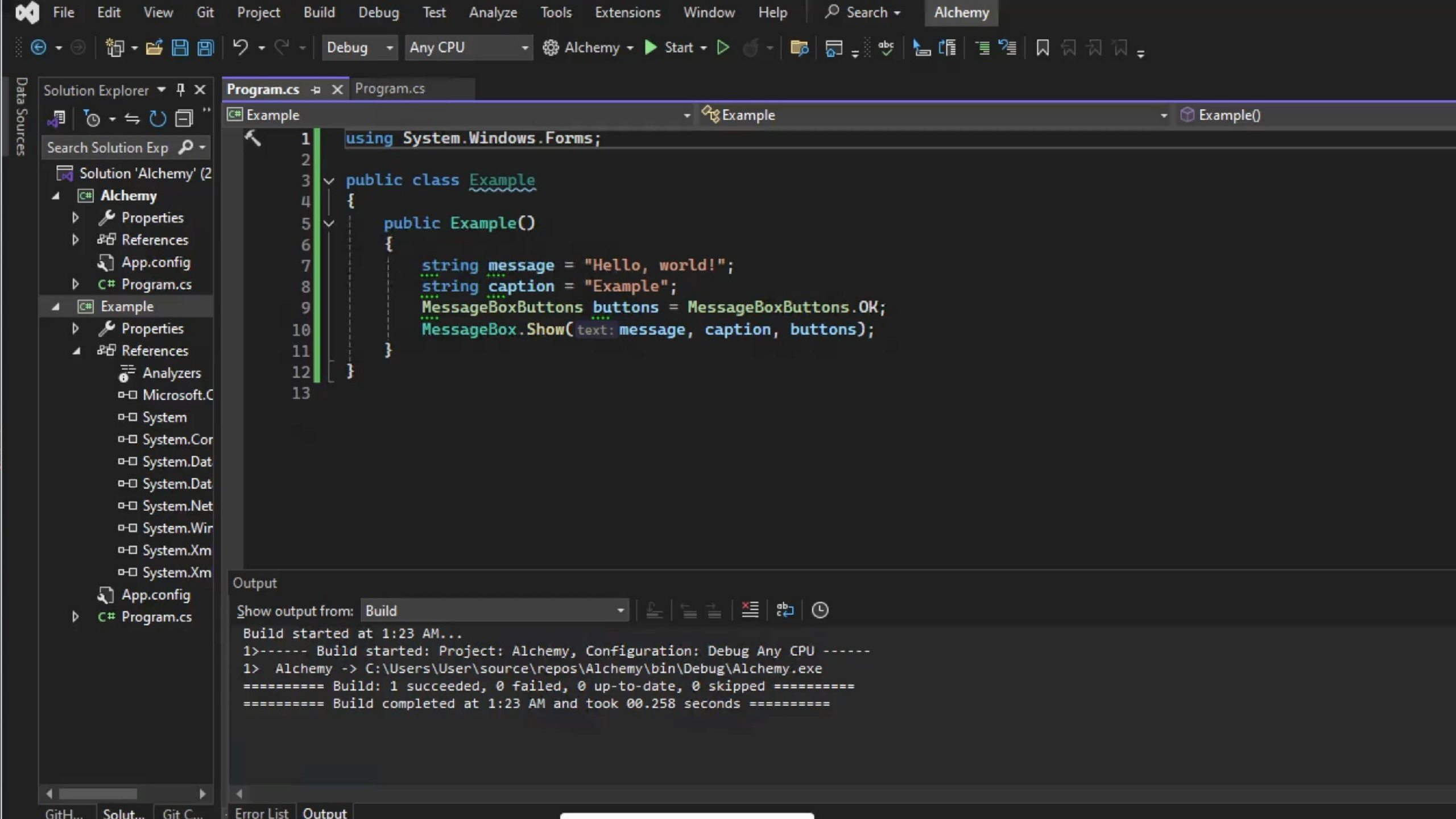
DESERIALIZATION GADGETS

- **Gadgets** – Serializable .NET classes that wrap unserializable classes
  - The **`Process`** and **`Assembly`** classes are not serializable – How can we execute code?
  - Serializable classes like **`ActivitySurrogateSelector`** can execute other .NET methods
- **YSoSerial.Net** – RCE payloads with support for **44 gadgets!**

ALTERNATIVE DESERIALIZATION GADGETS

- **New gadget – `XamlAssemblyLoadFromFile`**
  - Supports multiple formatters: **`Binary`**, **`Soap`**, **`Los`**, …
  - Example code already loads a .NET assembly




File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Alchemy
Debug
Any CPU
Alchemy
Start
Solution Explorer
Search Solution Exp
Solution 'Alchemy' (2
Alchemy
Properties
References
App.config
Program.cs
Example
Properties
References
Analyzers
Microsoft.C
System
System.Cor
System.Dat
System.Dat
System.Net
System.Wir
System.Xm
System.Xm
App.config
Program.cs
Program.cs
Program.cs
Example
Example
Example()
1 using System.Windows.Forms;
2
3 public class Example
4 {
5 public Example()
6 {
7 string message = "Hello, world!";
8 string caption = "Example";
9 MessageBoxButtons buttons = MessageBoxButtons.OK;
10 MessageBox.Show(text: message, caption, buttons);
11 }
12 }
13
Output
Show output from: Build
Build started at 1:23 AM...
1>------ Build started: Project: Alchemy, Configuration: Debug Any CPU ------
1> Alchemy -> C:\Users\User\source\repos\Alchemy\bin\Debug\Alchemy.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 1:23 AM and took 00.258 seconds ==========
Error List
Output

FORTRA

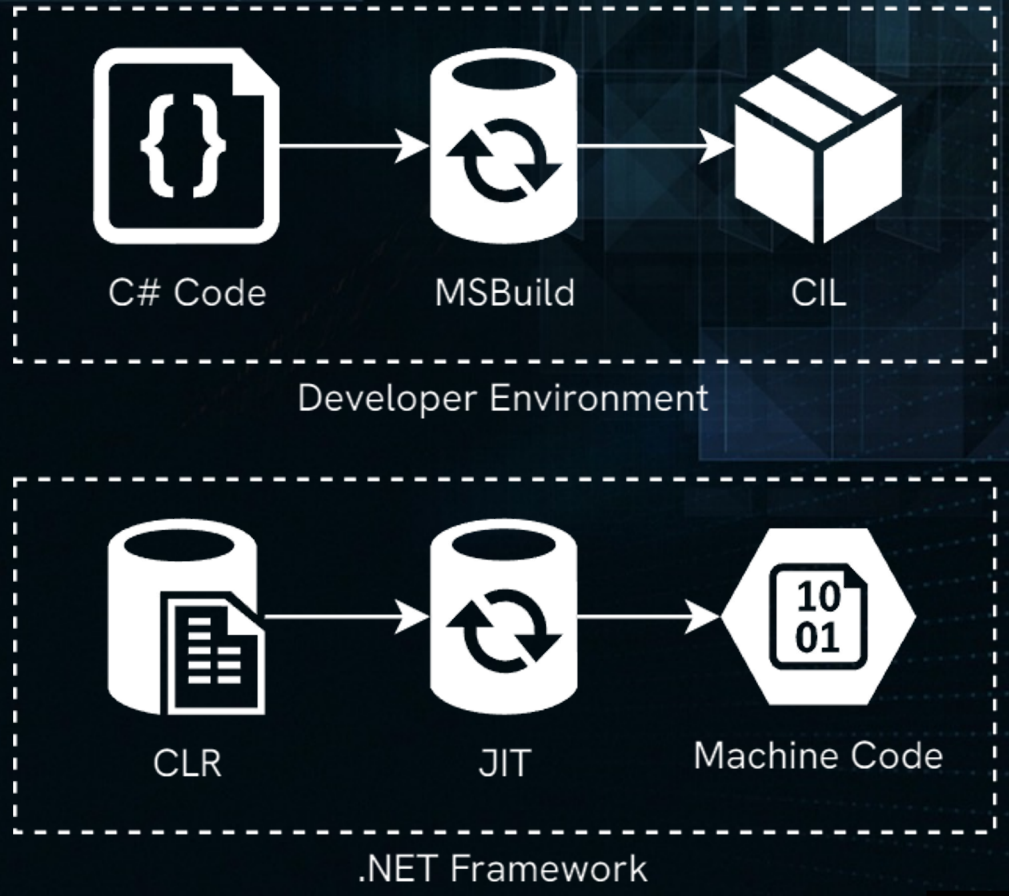
UNMANAGED CONTROL

RESOLVING AND PATCHING MANAGED FUNCTIONS

UNMANAGED CLR HOSTING API

- Common Language Runtime (CLR)
  - Executes “compiled” .NET projects

- Loading the .NET Runtime:
  1. Load the CLR
  2. Create an AppDomain (optional)
  3. Load a .NET assembly into an AppDomain



PATCHING MANAGED FUNCTIONS

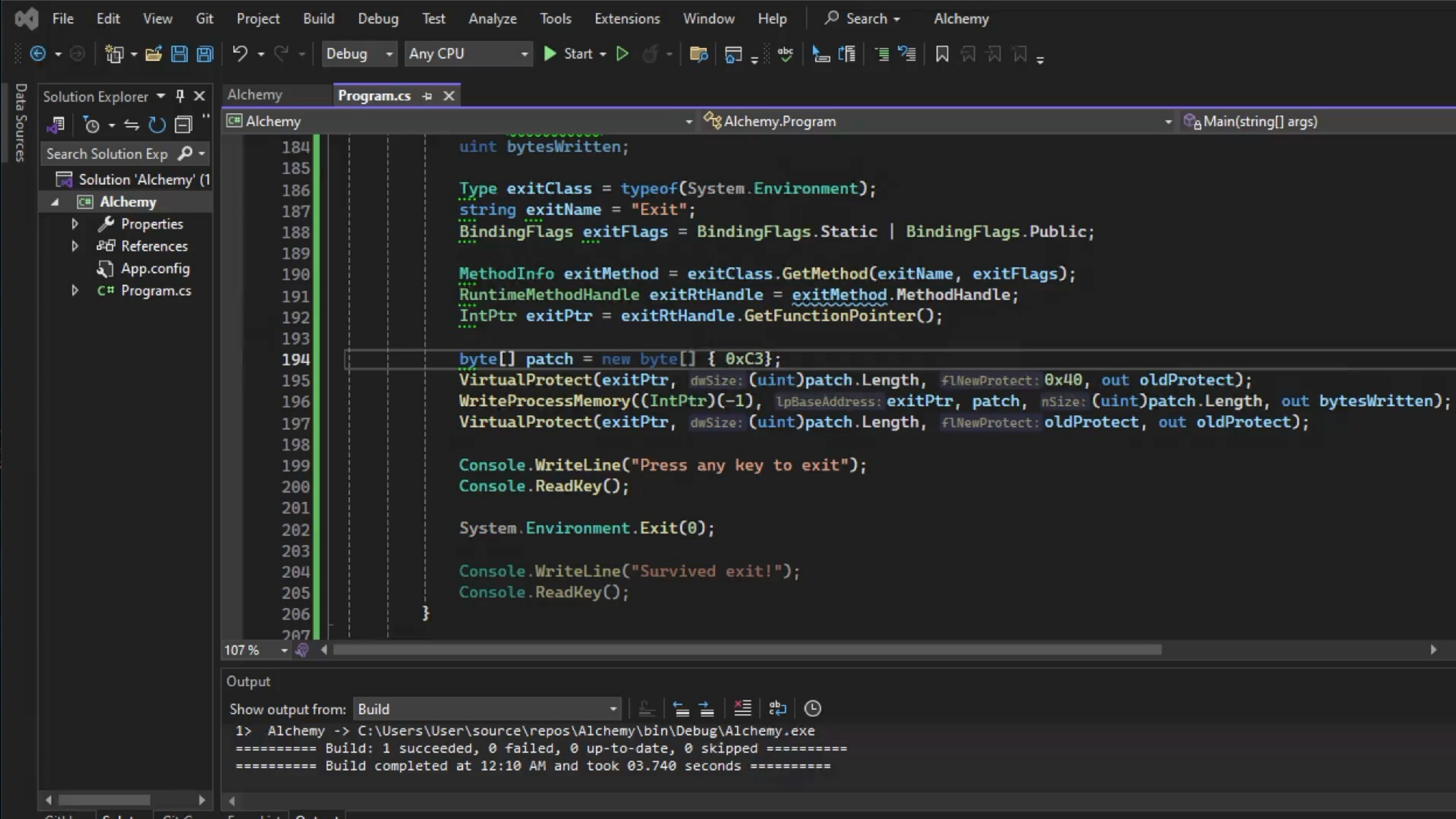
- Some .NET functions cause issues or risk implant stability
- Resolve .NET functions reflectively:
  1. Define+resolve the managed method
  2. Use the method handle to get its unmanaged function pointer

```
namespace EnvExitPatch
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("About to exit...");
            Environment.Exit(0);
            Console.WriteLine("Survived exit!");
        }
```

```
Type exitClass = typeof(System.Environment);
string exitName = "Exit";
BindingFlags exitBinding = BindingFlags.Static | BindingFlags.Public;

MethodInfo exitInfo = exitClass.GetMethod(exitName, exitBinding);
RuntimeMethodHandle exitRtHandle = exitInfo.MethodHandle;
IntPtr exitPtr = exitRtHandle.GetFunctionPointer();
```

MDSec

Massaging your CLR


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
Alchemy
Debug
Any CPU
Start
Solution Explorer
Search Solution Exp
Solution 'Alchemy' (1
Alchemy
Properties
References
App.config
Program.cs
Alchemy
Program.cs
Alchemy
Alchemy.Program
Main(string[] args)
uint bytesWritten;
Type exitClass = typeof(System.Environment);
string exitName = "Exit";
BindingFlags exitFlags = BindingFlags.Static | BindingFlags.Public;
MethodInfo exitMethod = exitClass.GetMethod(exitName, exitFlags);
RuntimeMethodHandle exitRtHandle = exitMethod.MethodHandle;
IntPtr exitPtr = exitRtHandle.GetFunctionPointer();
byte[] patch = new byte[] { 0xC3};
VirtualProtect(exitPtr, dwSize: (uint)patch.Length, flNewProtect: 0x40, out oldProtect);
WriteProcessMemory((IntPtr)(-1), lpBaseAddress: exitPtr, patch, nSize: (uint)patch.Length, out bytesWritten);
VirtualProtect(exitPtr, dwSize: (uint)patch.Length, flNewProtect: oldProtect, out oldProtect);
Console.WriteLine("Press any key to exit");
Console.ReadKey();
System.Environment.Exit(0);
Console.WriteLine("Survived exit!");
Console.ReadKey();
}
107 %
Output
Show output from: Build
1> Alchemy -> C:\Users\User\source\repos\Alchemy\bin\Debug\Alchemy.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 12:10 AM and took 03.740 seconds ==========

PATCHING MANAGED FUNCTIONS

- **Potential drawback** – Must load a .NET assembly before anything else
  - Can this code be replicated from an unmanaged context?

```
Type exitClass = typeof(System.Environment);
string exitName = "Exit";
BindingFlags exitBinding = BindingFlags.Static | BindingFlags.Public;

MethodInfo exitInfo = exitClass.GetMethod(exitName, exitBinding);
RuntimeMethodHandle exitRtHandle = exitInfo.MethodHandle;
IntPtr exitPtr = exitRtHandle.GetFunctionPointer();
```

UNMANAGED FUNCTION RESOLUTION

1. Resolve mscorlib in default AppDomain with **Load**:

```
IUnknownPtr appDomainUnk;
corRtHost->GetDefaultDomain(&appDomainUnk);

_AppDomain* appDomain;
appDomainUnk->QueryInterface(IID_PPV_ARGS(&appDomain));

_Assembly* mscorlib;
appDomain->Load_2(SysAllocString(L"mscorlib, Version=4.0.0.0"), &mscorlib);
```

UNMANAGED FUNCTION RESOLUTION

2. Resolve target class and method with **GetType** and **GetMethod**:

```
Type exitClass = typeof(System.Environment);
string exitName = "Exit";
BindingFlags exitBinding = BindingFlags.Static | BindingFlags.Public;

MethodInfo exitInfo = exitClass.GetMethod(exitName, exitBinding);
```

```
_Type* exitClass;
mscorlib->GetType_2(SysAllocString(L"System.Environment"), &exitClass);

_MethodInfo* exitInfo;
BindingFlags exitFlags = BindingFlags_Public | BindingFlags_Static;
exitClass->GetMethod_2(SysAllocString(L"Exit"), exitFlags, &exitInfo);
```

UNMANAGED FUNCTION RESOLUTION

3. Resolve **MethodHandle** property:

```
RuntimeMethodHandle exitRtHandle = exitInfo.MethodHandle;
```

```
_Type* methodInfoClass;
BSTR methodInfoStr = SysAllocString(L"System.Reflection.MethodInfo");
mscorlib->GetType_2(methodInfoStr, &methodInfoClass);

_PropertyInfo* methodHndProperty;
BindingFlags methodHndFlags = BindingFlags_Instance | BindingFlags_Public;
BSTR methodHndStr = SysAllocString(L"MethodHandle");
methodInfoClass->GetProperty(methodHndStr, methodHndFlags, &methodHndProperty);
```

UNMANAGED FUNCTION RESOLUTION

4. Get value of **MethodHandle** property:

```
RuntimeMethodHandle exitRtHandle = exitInfo.MethodHandle;
```

```
VARIANT methodHndPtr = { 0 };
methodHndPtr.vt = VT_UNKNOWN;
methodHndPtr.punkVal = exitInfo;

SAFEARRAY* methodHndArgs = SafeArrayCreateVector(VT_EMPTY, 0, 0);
VARIANT methodHndVal = { 0 };
methodHndProperty->GetValue(methodHndPtr, methodHndArgs, &methodHndVal);
```

UNMANAGED FUNCTION RESOLUTION

5. Resolve **GetFunctionPointer** method:

```
IntPtr exitPtr = exitRtHandle.GetFunctionPointer();
```

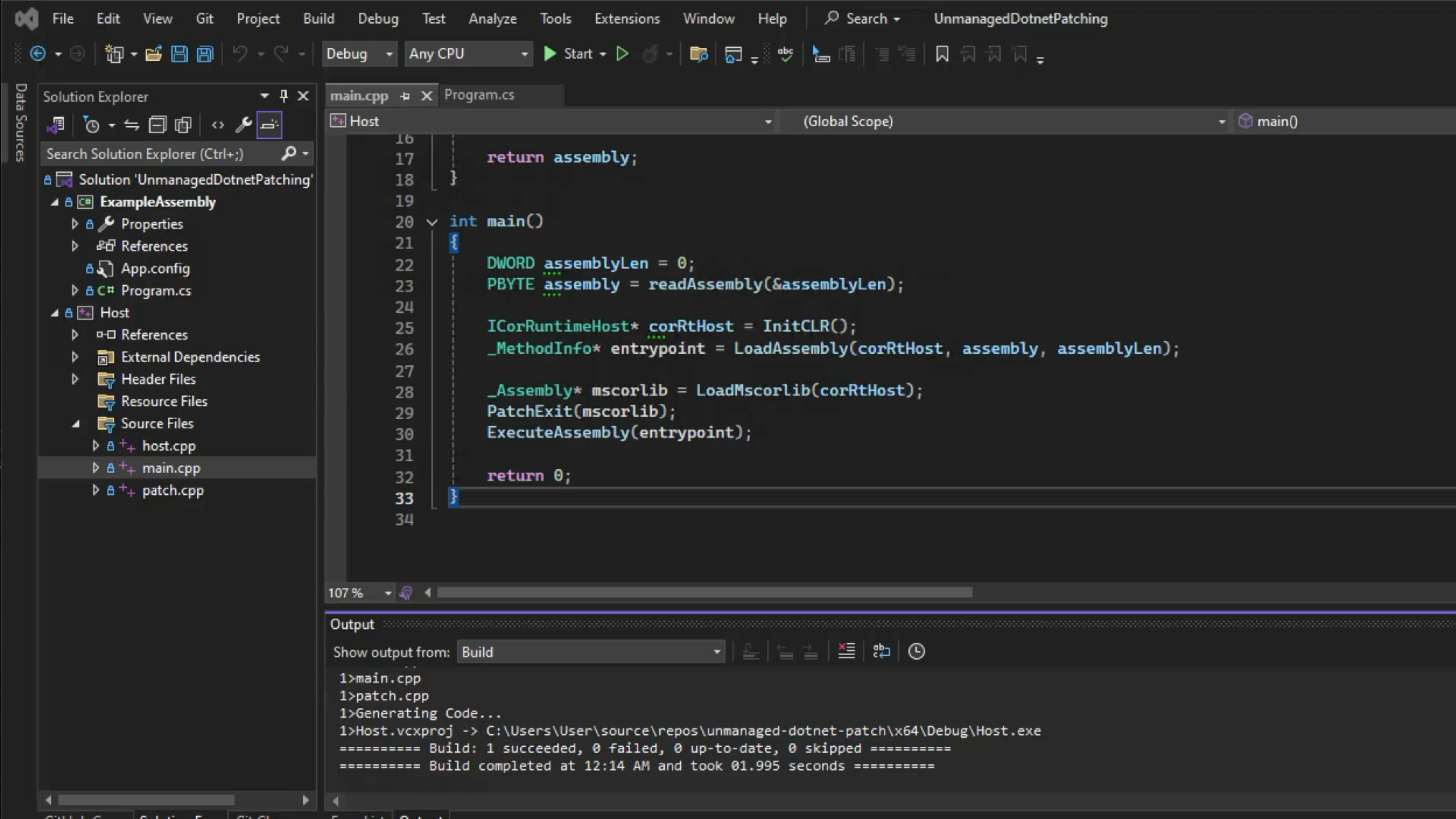
```
_Type* rtMethodHndType;
BSTR rtMethodHndStr = SysAllocString(L"System.RuntimeMethodHandle");
mscorlib->GetType_2(rtMethodHndStr, &rtMethodHndType);

_MethodInfo* getFuncPtrMtdInf;
BindingFlags getFuncPtrFlags = BindingFlags_Public | BindingFlags_Instance;
BSTR getFuncPtrStr = SysAllocString(L"GetFunctionPointer");
rtMethodHndType->GetMethod_2(getFuncPtrStr, getFuncPtrFlags, &getFuncPtrMtdInf);
```

UNMANAGED FUNCTION RESOLUTION

6. Execute **GetFunctionPointer**:

```
IntPtr exitPtr = exitRtHandle.GetFunctionPointer();
```

```
SAFEARRAY* getFuncPtrArgs = SafeArrayCreateVector(VT_EMPTY, 0, 0);
VARIANT exitPtr = { 0 };
getFuncPtrMtdInf->Invoke_3(methodHndVal, getFuncPtrArgs, &exitPtr);

printf("Exit function pointer: 0x%p\n", exitPtr.byref);
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
UnmanagedDotnetPatching
Debug
Any CPU
Start
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'UnmanagedDotnetPatching'
ExampleAssembly
Properties
References
App.config
Program.cs
Host
References
External Dependencies
Header Files
Resource Files
Source Files
host.cpp
main.cpp
patch.cpp
main.cpp
Program.cs
Host
(Global Scope)
main()
return assembly;
}
int main()
{
DWORD assemblyLen = 0;
PBYTE assembly = readAssembly(&assemblyLen);
ICorRuntimeHost* corRtHost = InitCLR();
_MethodInfo* entrypoint = LoadAssembly(corRtHost, assembly, assemblyLen);
_Assembly* mscorlib = LoadMscorlib(corRtHost);
PatchExit(mscorlib);
ExecuteAssembly(entrypoint);
return 0;
}
107 %
Output
Show output from: Build
1>main.cpp
1>patch.cpp
1>Generating Code...
1>Host.vcxproj -> C:\Users\User\source\repos\unmanaged-dotnet-patch\x64\Debug\Host.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 12:14 AM and took 01.995 seconds ==========

EXECUTING MANAGED FUNCTIONS

- The “unmanaged patching” capability is useful, but what primitives does it use?
  1. Resolving managed functions
  2. Executing managed functions

```
Type exitClass = typeof(System.Environment);
string exitName = "Exit";
BindingFlags exitBinding = BindingFlags.Static | BindingFlags.Public;

MethodInfo exitInfo = exitClass.GetMethod(exitName, exitBinding);
RuntimeMethodHandle exitRtHandle = exitInfo.MethodHandle;
IntPtr exitPtr = exitRtHandle.GetFunctionPointer();
```

- Executing arbitrary .NET functions from unmanaged code: **Many possibilities!**

https://github.com/outflanknl/unmanaged-dotnet-patch


black hat
MIDDLE EAST AND AFRICA
FORTRA
OUTFLANK
RED TEAM TOOLS AND EXPERT SERVICES