C# Workshop

Wild West Hackin Fest 2018

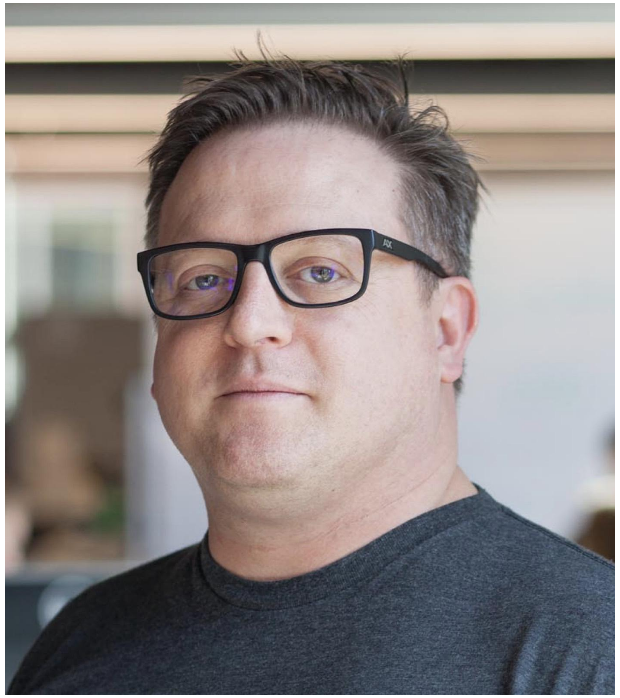
Zac Brown

~~Casey Smith~~

Joe Moles

RIP subTee
Goodnight, sweet prince.

Just kidding. Seriously.



Introductions

Zac Brown, Principal Engineer
@zacbrown on the twitters

Joe Moles, VP of Customer SecOps
@FlyingMonkey127 on the twitters

Housekeeping

- Timing / Flow Of Workshop
- Outline
- How to get help
- Resources
- Ask questions!

C# Workshop Learning Objectives

1. Build and compile using only built in tools - csc.exe
2. Understand the high level of the CLR
3. Understand C# Program Layout and Terms
4. Understand Platform Invoke (P/Invoke) Architecture and how to use

C# Workshop Caveats

1. The topics discussed are greatly condensed. We are leaving out some important concepts for the sake of conceptual clarity.

2. We won’t be doing much direct programming due to the limited time. We **will** be compiling code and playing around with it.

3. If you had issues downloading the repo, AV may have flagged Example 4.

Why the interest? Advantages/Disadvantages

+ Built in, Default
+ Many Bypass Harnesses (InstallUtil, MsBuild, RegAsm, DotNetToJscript)
+ Easy to Use, Portable, Reuse Code
+ Assembly.Load(byte[]) - Loading From A Memory location instead of disk

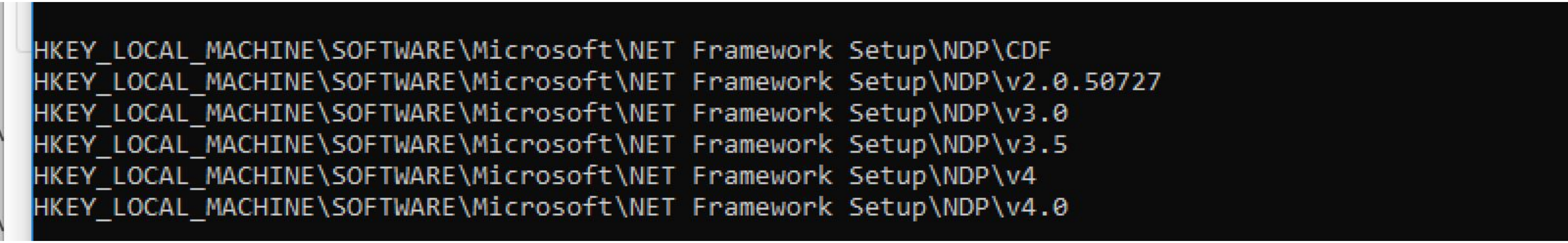
- Binary Analysis - Reversing Payloads, so DnSpy, or ILSpy to get to source
- Artifacts Left Behind - Did you clean up your binary?
- Many ways to collect Telemetry - Fusion Logging, ETW

.NET Framework Ships With Windows Since Vista

It is installed by default.

It powers PowerShell

reg.exe query "HKLM\SOFTWARE\Microsoft\NET Framework Setup\NDP\



```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\CDF
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v2.0.50727
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v3.0
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v3.5
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4.0
```

Managed vs. Unmanaged

Why this terminology matters…

Unmanaged Code

Platform specific - you compile for a target CPU architecture

Requires no virtual machine or runtime (e.g. .NET, JVM)

Typically written in C, C++, Rust, Pascal, etc.

Managed vs. Unmanaged

Why this terminology matters…

- **Managed Code**
    - Platform independent
    - Requires a virtual machine or runtime (e.g. .NET, JVM)
        - **But…** you can sometimes remove the need for this.
    - Can be interpreted (e.g. PowerShell) or compiled (e.g. C#)
        - **Interpreted** - code is evaluated at runtime
        - **Compiled** - code is compiled to intermediate assembly format

.NET History and Background

Originates in late 90's and early 2000's. First full version released in 2002.
Once called the "COM+ Runtime" - lots of roots in COM and COM+.

Goals

- Unify the runtimes of the OS (cscript/jscript, MFC, VB)
- Create a development platform ready for the next generation of web services
- Create a new language & APIs to enable the next generation of development:
  - Windows
  - Web
  - Mobile

Common Language Specification - CLS

C#, VB.NET, F#, etc all compile to same format

  Common **I**ntermediate **L**anguage

All can interchange/reuse modules

  For our purposes, a module is a CLR assembly but it’s more nuanced

Fundamental abstraction

  CLS defines what CLR implemented. Mono is another implementation.

JIT, CIL, MSIL - Just In Time, Bytecode

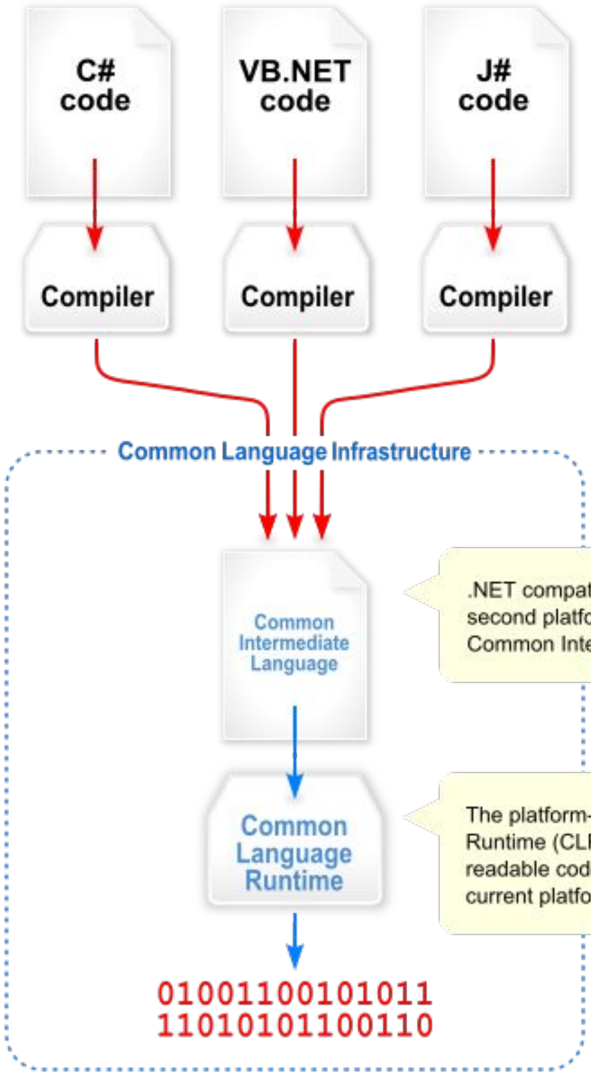
CIL/MSIL

CIL == MSIL
CIL - Common Intermediate Language (new)
MSIL - Microsoft Intermediate Language (old)

JIT - Just In Time Compilation

Generate native machine code (i.e. x86 assembly) on the fly

Only do it when code is called

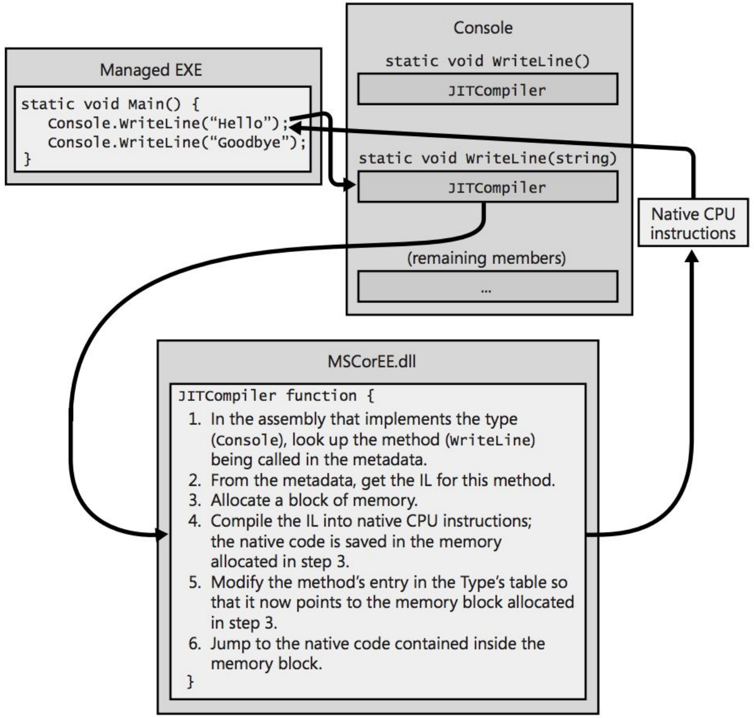


.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).

The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.

A Piece of JIT - Just-In-Time Compilation

Figure 1-4 shows what happens the first time a method is called.

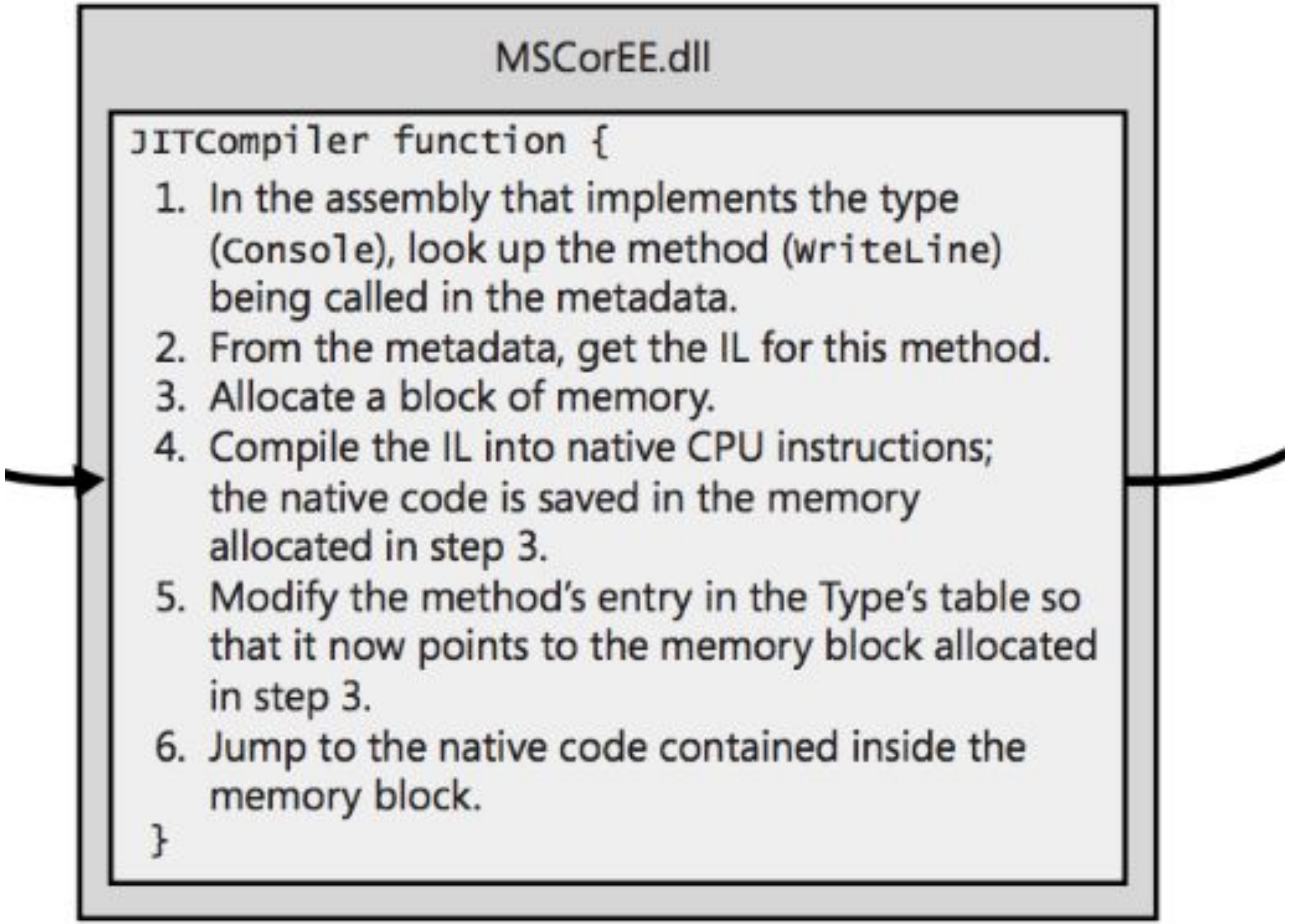


```
static void Main() {
   Console.WriteLine("Hello");
   Console.WriteLine("Goodbye");
}
```

```
JITCompiler function {
  1. In the assembly that implements the type
     (Console), look up the method (WriteLine)
     being called in the metadata.
  2. From the metadata, get the IL for this method.
  3. Allocate a block of memory.
  4. Compile the IL into native CPU instructions;
     the native code is saved in the memory
     allocated in step 3.
  5. Modify the method's entry in the Type's table so
     that it now points to the memory block allocated
     in step 3.
  6. Jump to the native code contained inside the
     memory block.
}
```

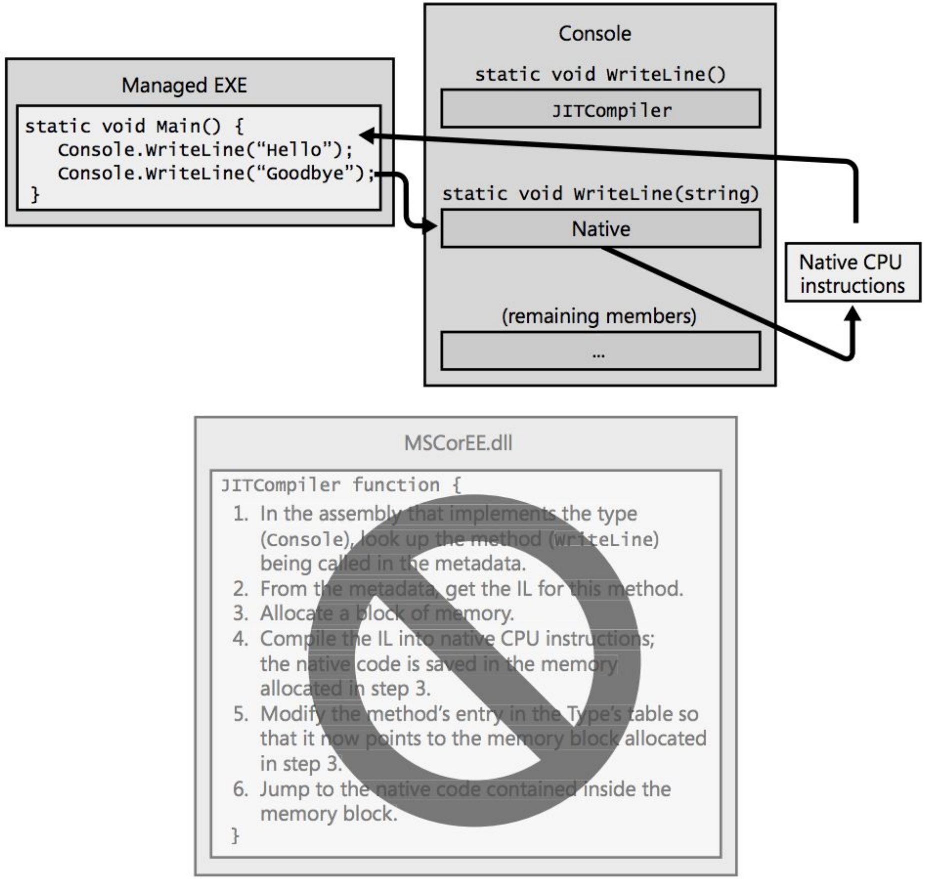
FIGURE 1-4 Calling a method for the first time.

Source : CLR via C# (4th Edition) (Developer Reference)



```
JITCompiler function {
  1. In the assembly that implements the type
     (Console), look up the method (WriteLine)
     being called in the metadata.
  2. From the metadata, get the IL for this method.
  3. Allocate a block of memory.
  4. Compile the IL into native CPU instructions;
     the native code is saved in the memory
     allocated in step 3.
  5. Modify the method's entry in the Type's table so
     that it now points to the memory block allocated
     in step 3.
  6. Jump to the native code contained inside the
     memory block.
}
```

a method for the first time



```
static void Main() {
   Console.WriteLine("Hello");
   Console.WriteLine("Goodbye");
}
```

```
JITCompiler function {
  1. In the assembly that implements the type
     (Console), look up the method (WriteLine)
     being called in the metadata.
  2. From the metadata, get the IL for this method.
  3. Allocate a block of memory.
  4. Compile the IL into native CPU instructions;
     the native code is saved in the memory
     allocated in step 3.
  5. Modify the method's entry in the Type's table so
     that it now points to the memory block allocated
     in step 3.
  6. Jump to the native code contained inside the
     memory block.
}
```

FIGURE 1-5 Calling a method for the second time.

Source : CLR via C# (4th Edition) (Developer Reference)

NGEN - No MORE JIT

Generally for improving assembly performance:

startup - make startup faster by pre-generating native assembly
memory - shared code pages reduce memory footprint in big applications

Considerations:

Architecture specific - leverages SSE, AVX, etc when available
Ngen.exe binaries are NOT true native binaries
You generally ngen.exe binaries as part of setup - replaces JIT.

Most importantly, NGEN binaries are installed in native image cache.

CLR Hosting

CLR Hosting

Unmanaged processes can load the CLR to run .NET code.

Unmanaged process interacts with CLR types using COM.

Be wary of processes that do this. Mainly Microsoft binaries.

Detection Ideas…

If for example notepad.exe loads mscoree.dll.

Understand that this is not detection, just telemetry.

This is the ingredients, and recipe, not the finished product.

Real Quick - Assembly Resolution

Global Assembly Cache

Used to share common libraries across the operating system.

Speeds up resolution of dependencies system-wide.

Native Image Cache

Remember NGEN? This is where NGEN’ed binaries go.

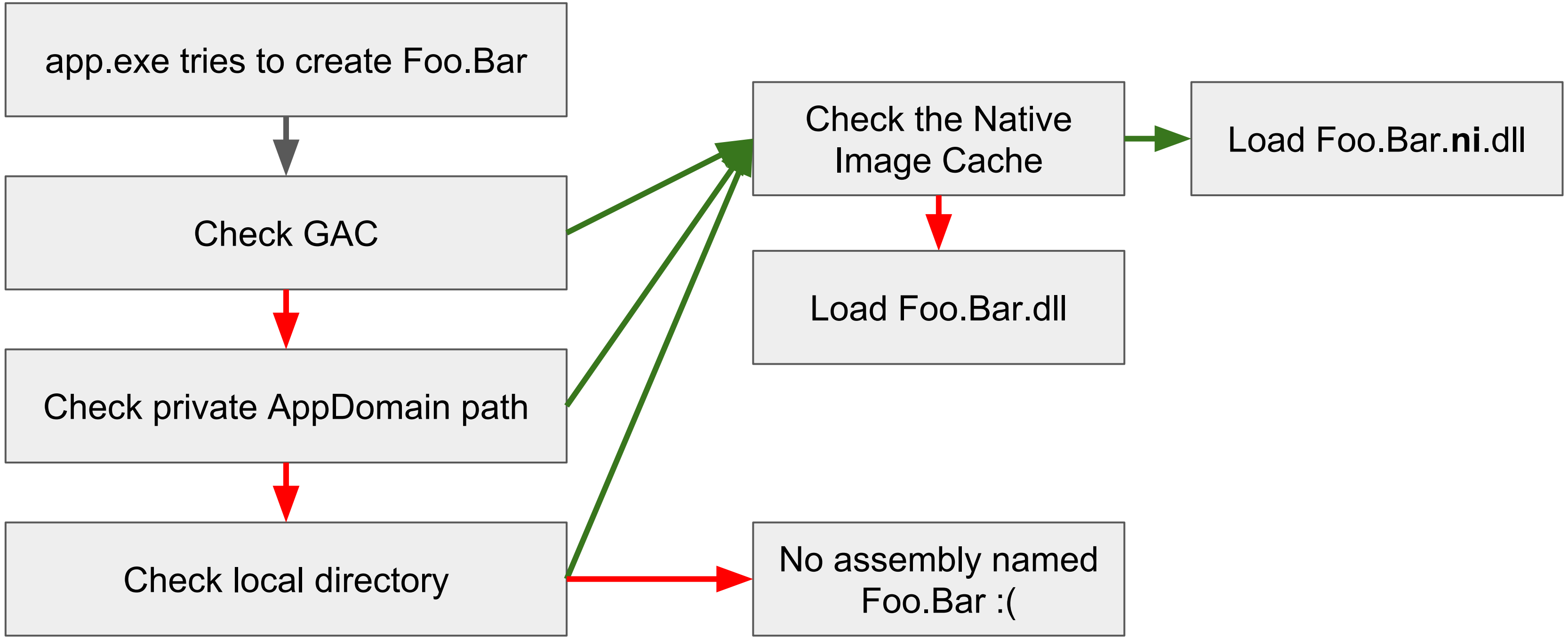
System.Potato.dll ⇒ System.Potato.ni.dll

Assembly Binding

1. App.exe wants to load type Foo.Bar
2. CLR checks Global Assembly Cache (GAC) for Foo.Bar
3. CLR checks private paths added to the current AppDomain
4. CLR checks local app directory for Foo.Bar
5. If CLR finds Foo.Bar, check Native Image Cache (NIC)
6. If CLR:
   a. finds Foo.Bar in NIC, use that
   b. doesn’t find Foo.Bar in NIC, use the original

You can observe a lot of this in `fuslogvw.exe`.
CAVEAT: This is vastly simplified.

Assembly Binding in pictures...


app.exe tries to create Foo.Bar
Check GAC
Check private AppDomain path
Check local directory
Check the Native Image Cache
Load Foo.Bar.**ni**.dll
Load Foo.Bar.dll
No assembly named Foo.Bar :(

Ideas And Thoughts on JIT

1. .NET CLR Injection: Modify IL Code during Run-time

https://www.codeproject.com/Articles/463508/NET-CLR-Injection-Modify-IL-Code-during-Run-time

2. Fuzzing the .NET JIT Compiler

http://mattwarren.org/2018/08/28/Fuzzing-the-.NET-JIT-Compiler/

Let’s write some code...

csc.exe location and versions

csc.exe

Locations:

```
x86 - C:\Windows\Microsoft.NET\Framework
x64 - C:\Windows\Microsoft.NET\Framework64
```

Versions:

v2.0.50727

v4.0.30319 (**most likely, this is the only one)**

Setup your path...

```
set DotNet64=C:\Windows\Microsoft.NET\Framework64

set PATH=%PATH%;%DotNet64%\v4.0.30319
```

OR

Run **setup_path.cmd** in the repo root.

Oh ya, AV is a problem :(

`Run `**`disable_av.cmd`**` in the repo root.`

Relevant csc.exe Flags

/reference:

Reference another library for types used in the current file.

/target:

Most likely values: **`exe, library`**

/platform:

Useful for shellcode runners when you need to target specific process arch.

Most likely values: **`x86, x64, anycpu`**

/out:

Specify the output file name. By default: Foobar.cs ⇒ Foobar.{dll, exe}

Relevant csc.exe Flags (advanced)

/keyfile: (if time allows)

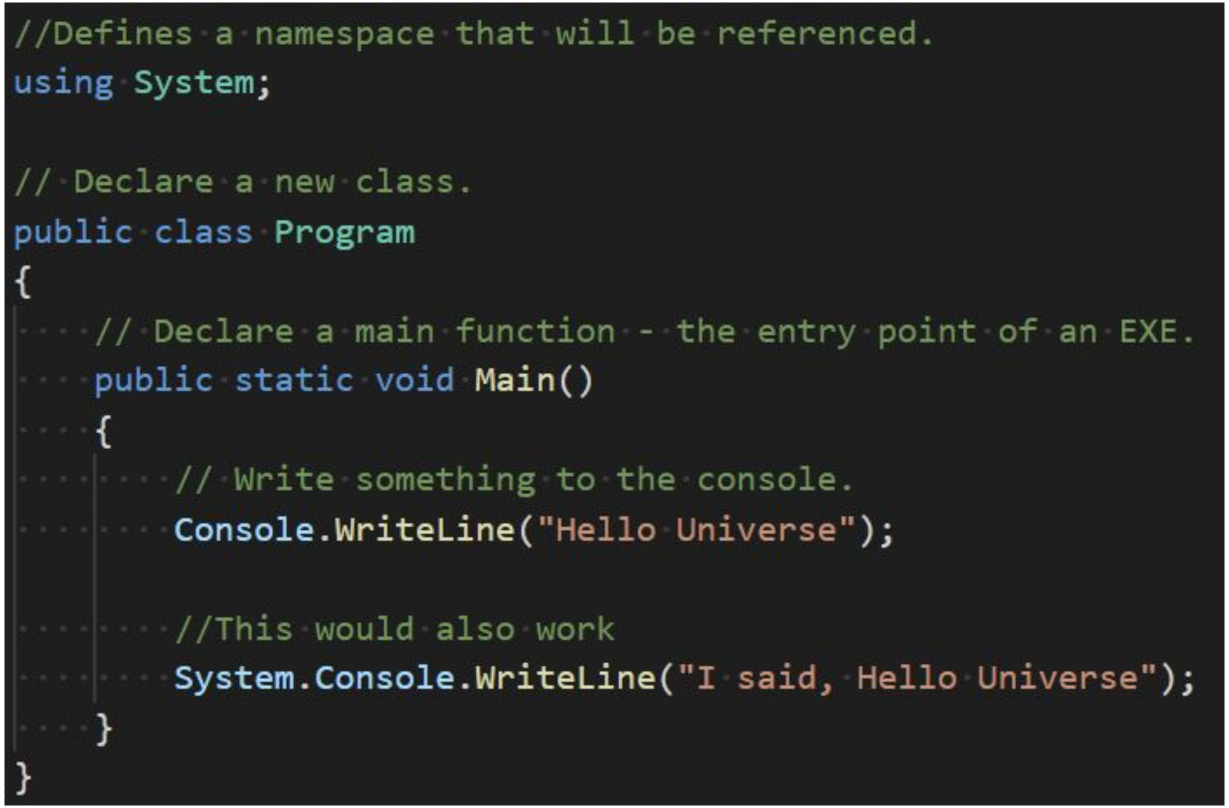
Sometimes assembly needs to be a Strong Named

Regsvcs - bypass but requires SNK

PS: SNK’s are weird. Most of the time it can be any old SNK, **not** an authenticode signature check.

Example: Hello world!
Let’s learn the tooling and terms.

Hello Universe!



```
//Defines a namespace that will be referenced.
using System;

// Declare a new class.
public class Program
{
    // Declare a main function - the entry point of an EXE.
    public static void Main()
    {
        // Write something to the console.
        Console.WriteLine("Hello Universe");

        //This would also work
        System.Console.WriteLine("I said, Hello Universe");
    }
}
```

15 Minute Lab - Exercise 1

Compile and execute HelloUniverse.cs

We want to make sure everyone is on the same page and everything is working before we discuss advanced topics.

Please let a TA or the instructors know if you're having trouble.

Example: Classes
Let’s build a class.

Object Oriented Programming

Terms

- **Class Definition** - blueprints
- **Class Instance** - blueprints constructed into an object
- **Inheritance** - defining relationships between classes
- **Interface** - contract specifying methods a class will implement
- **Encapsulation** - construct for abstracting away implementation details
  - E.g. It’s not necessary for driver to understand car’s engine.

Time To Write Some Code

Basic Building Blocks

- **using statements** - `using System.Diagnostics;`
- **class declarations** - `class Foo {}`
- **methods** - `void TaterSalad() {}`
  - **main function (EXE only)** - `static void Main() {}`
- **properties** - getters & setters of instance variables

Class, Constructors, Fields, Properties, Methods

Constructor - Called when you create the object

Fields - “Internal Variables”

Properties - Expose fields in getter/setter form

Methods - Functions That Can be called.

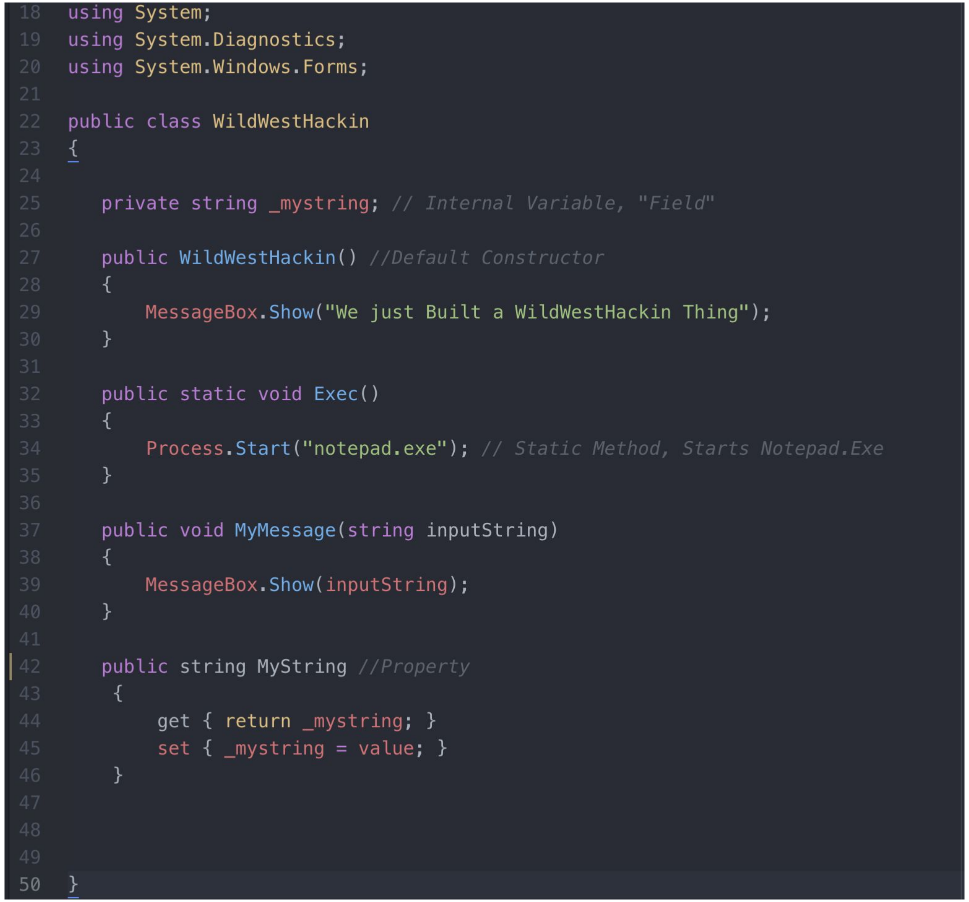
Public, Private, Static

Static can be invoked, WITHOUT creating an object...

Build And Reuse A Class Library

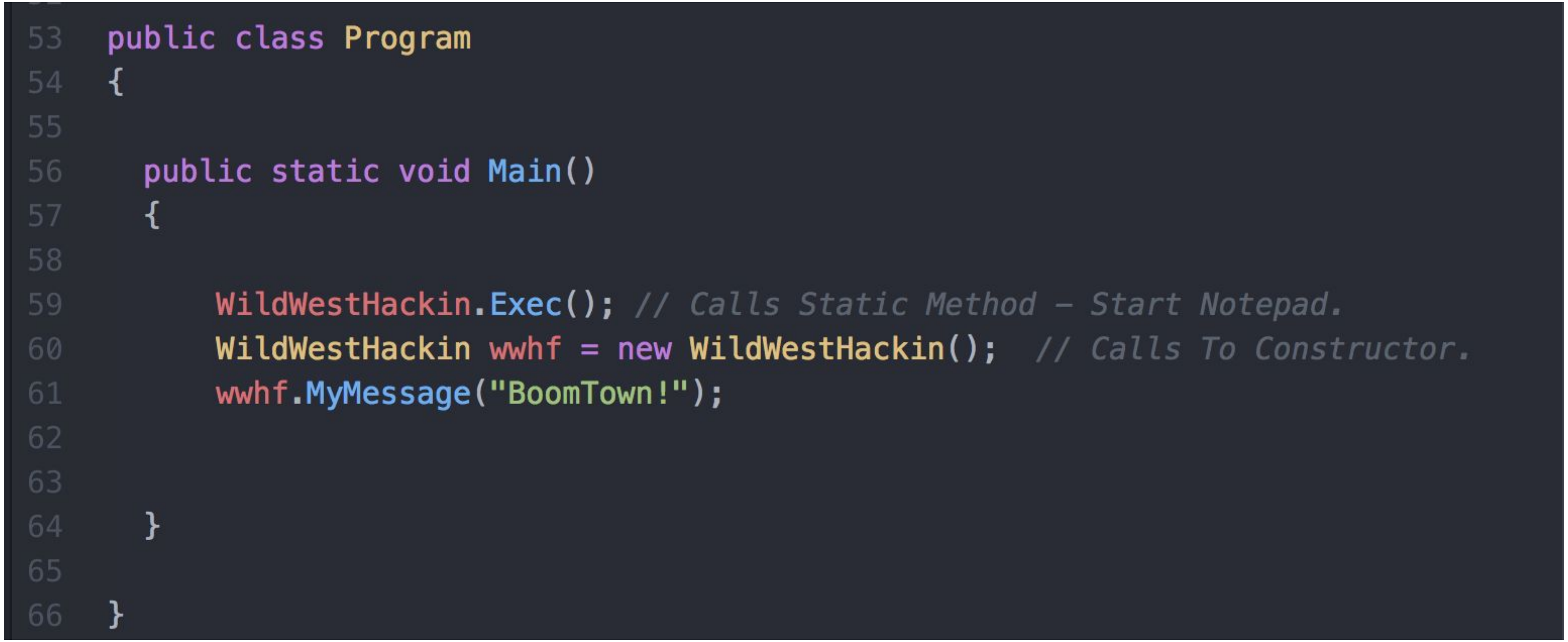
1. Can be included in same file OR
2. Can be in a binary dll, and referenced at load time

Code Time - Basic Class



```csharp
using System;
using System.Diagnostics;
using System.Windows.Forms;

public class WildWestHackin
{

   private string _mystring; // Internal Variable, "Field"

   public WildWestHackin() //Default Constructor
   {
       MessageBox.Show("We just Built a WildWestHackin Thing");
   }

   public static void Exec()
   {
       Process.Start("notepad.exe"); // Static Method, Starts Notepad.Exe
   }

   public void MyMessage(string inputString)
   {
       MessageBox.Show(inputString);
   }

   public string MyString //Property
    {
        get { return _mystring; }
        set { _mystring = value; }
    }



}
```

Basic Class Instantiation



```
public class Program
{

  public static void Main()
  {

      WildWestHackin.Exec(); // Calls Static Method - Start Notepad.
      WildWestHackin wwhf = new WildWestHackin();  // Calls To Constructor.
      wwhf.MyMessage("BoomTown!");


  }

}
```

15 Minute Lab - Exercise 2

Build your assembly and reference that library.

Compile the class library (WildWestHackin) as DLL: csc.exe /target:library

Compile the main file (Main) as EXE: csc.exe /reference:<DLL> /target:exe

Example: P/Invoke

Platform Invoke (P/Invoke)

C# can do ANYTHING, C++ can…

With fewer established detection or prevention strategies.

- Focus on behavioral versus signature detections, similar to C/C++.

Many more opportunities to be scripted. For example:

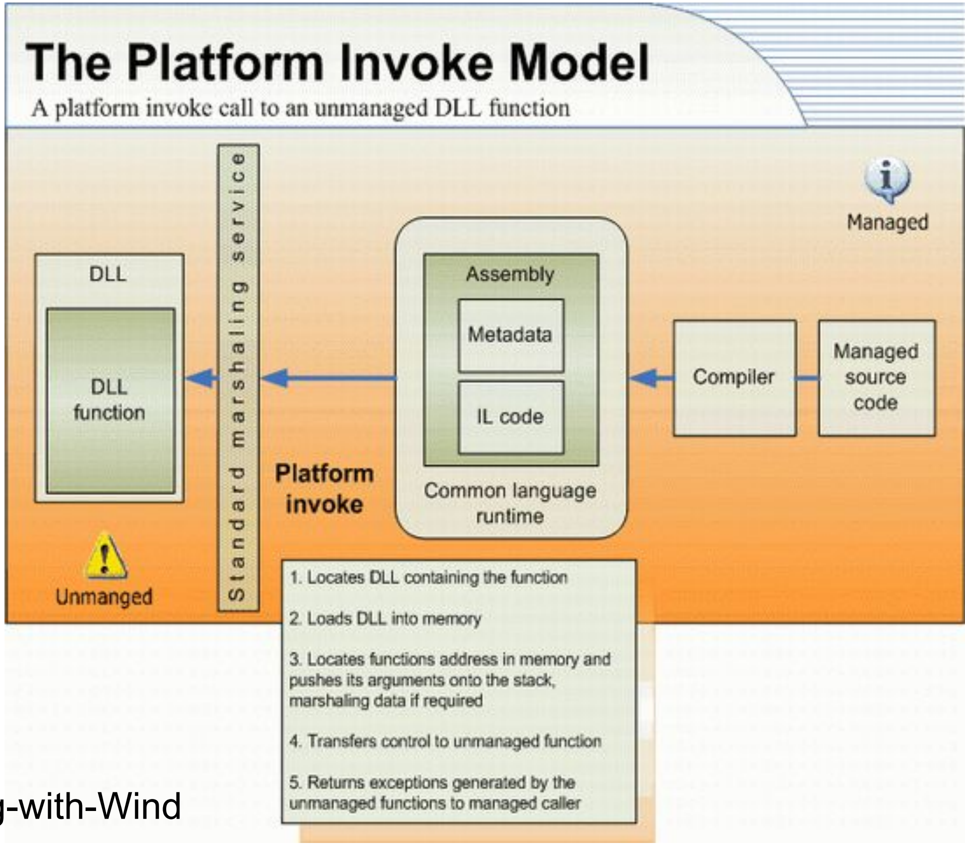
- MSBuild.exe
- InstallUtil.exe

Platform Invoke (P/Invoke)

Allows .NET assemblies to call Win32 APIs.

Any Win32 API can be called if you get the P/Invoke signature correct.

P/Invoke **does not** handle deallocating resources from the Win32 layer.


The Platform Invoke Model
A platform invoke call to an unmanaged DLL function
Managed
DLL
DLL function
Standard marshaling service
Platform invoke
Assembly
Metadata
IL code
Common language runtime
Compiler
Managed source code
Unmanged
1. Locates DLL containing the function
2. Loads DLL into memory
3. Locates functions address in memory and pushes its arguments onto the stack, marshaling data if required
4. Transfers control to unmanaged function
5. Returns exceptions generated by the unmanaged functions to managed caller

Source: https://www.codeproject.com/Articles/6042/Interoperating-with-Windows-Media-Player-using-P-I

Platform Invoke (P/Invoke)

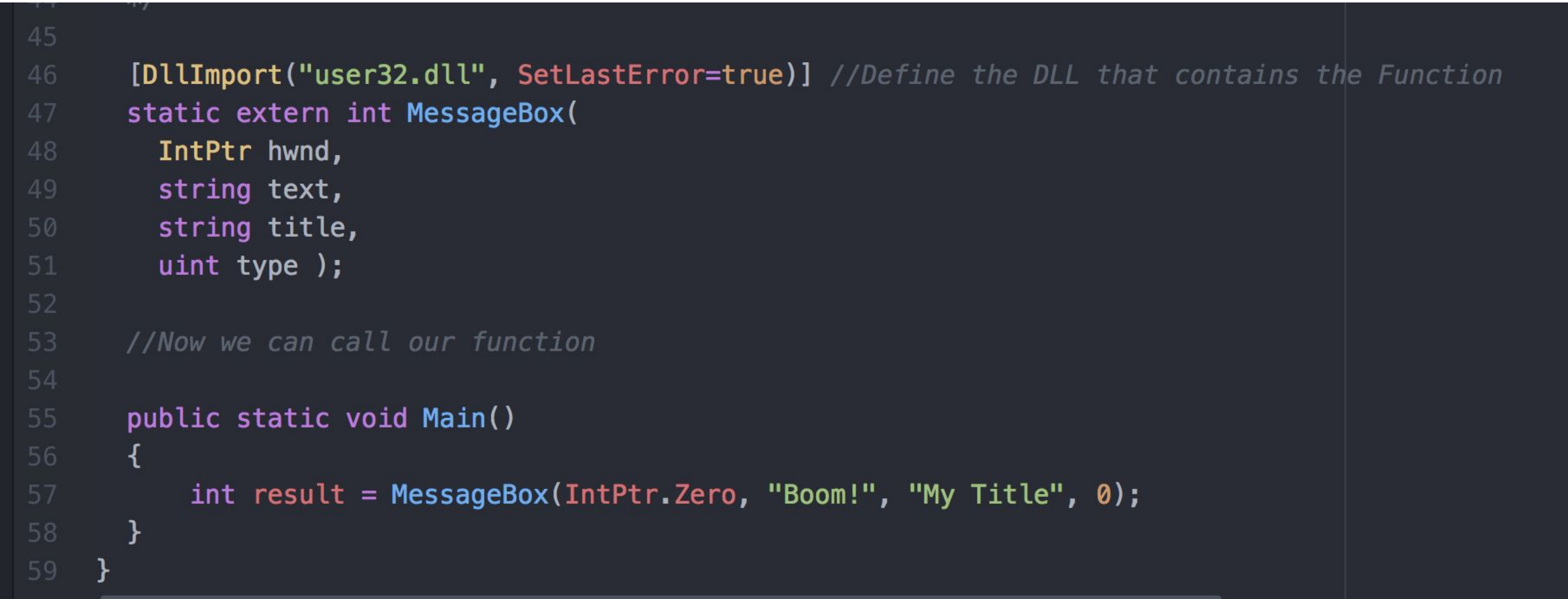
Teach me how to signature:

Pinvoke.net - website that has many common P/Invoke signatures

PInvoker - converts Win32 DLL to P/Invoke signatures in .NET assembly

If you’re not sure how to write the P/Invoke signature, consult one of those tools.

Introductory Example - MessageBox



```
  [DllImport("user32.dll", SetLastError=true)] //Define the DLL that contains the Function
  static extern int MessageBox(
    IntPtr hwnd,
    string text,
    string title,
    uint type );

  //Now we can call our function

  public static void Main()
  {
      int result = MessageBox(IntPtr.Zero, "Boom!", "My Title", 0);
  }
}
```

15 Minute Lab: Exercise 3

Let’s launch a message box!

Compile the main file (PlatformInvoke) as EXE: csc.exe /target:exe

Example: Advanced Persistent Topics

Inheritance, Override

Advanced Topics

Interfaces

Class Inheritance

Method Override

Platform Invoke (P/Invoke)

Interfaces

Basic Concepts

Interfaces are contracts - “I agree to implement these methods on my class.”

Interfaces are for abstraction - allows you to only expose partial functionality.

By convention, always start with an **`I`**. e.g. `ITaterSalad`

Interfaces

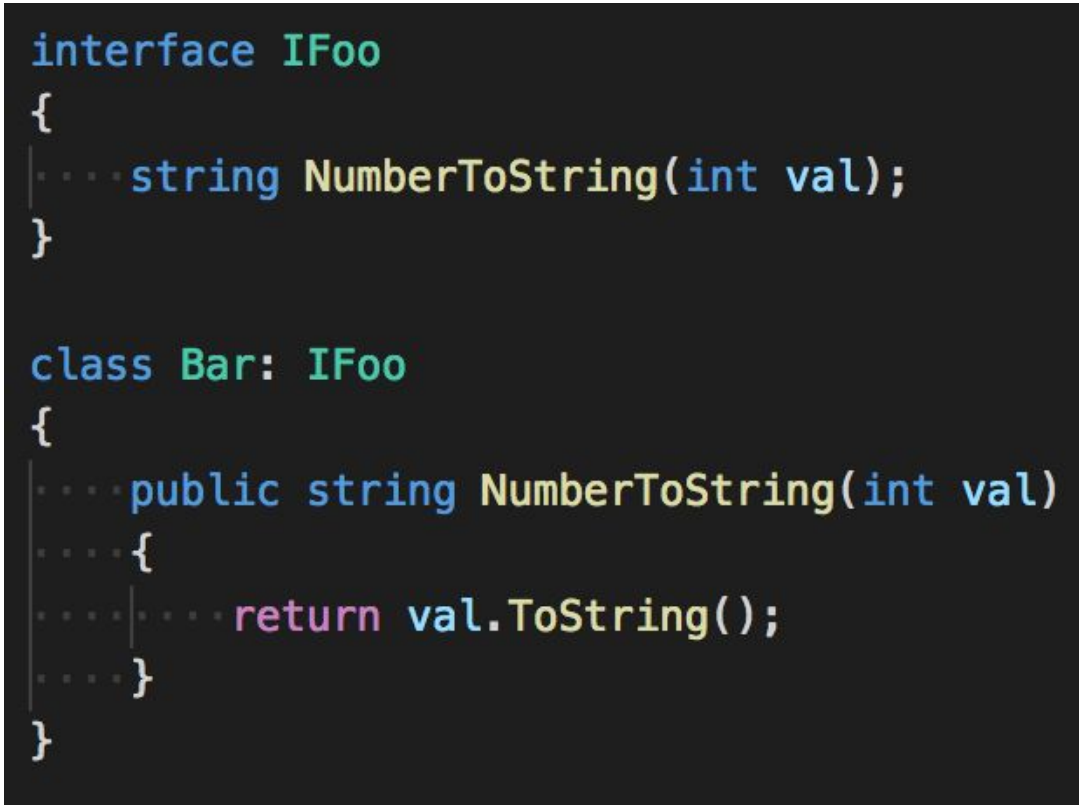
Takeaways

Interfaces are specifications, not implementation.

C# classes can implement multiple interfaces.

At least initially, not crucial as you learn C#. Long term, will be useful.

Interfaces Example



```
interface IFoo
{
    string NumberToString(int val);
}

class Bar: IFoo
{
    public string NumberToString(int val)
    {
        return val.ToString();
    }
}
```

Inheritance

Basic Concepts

Enables code reuse. Base class implements common functionality.

C# classes can only inherit from a **single** class.

Inheritance is useful for composition. “I extend the functionality of my base.”

Inheritance

Takeaways

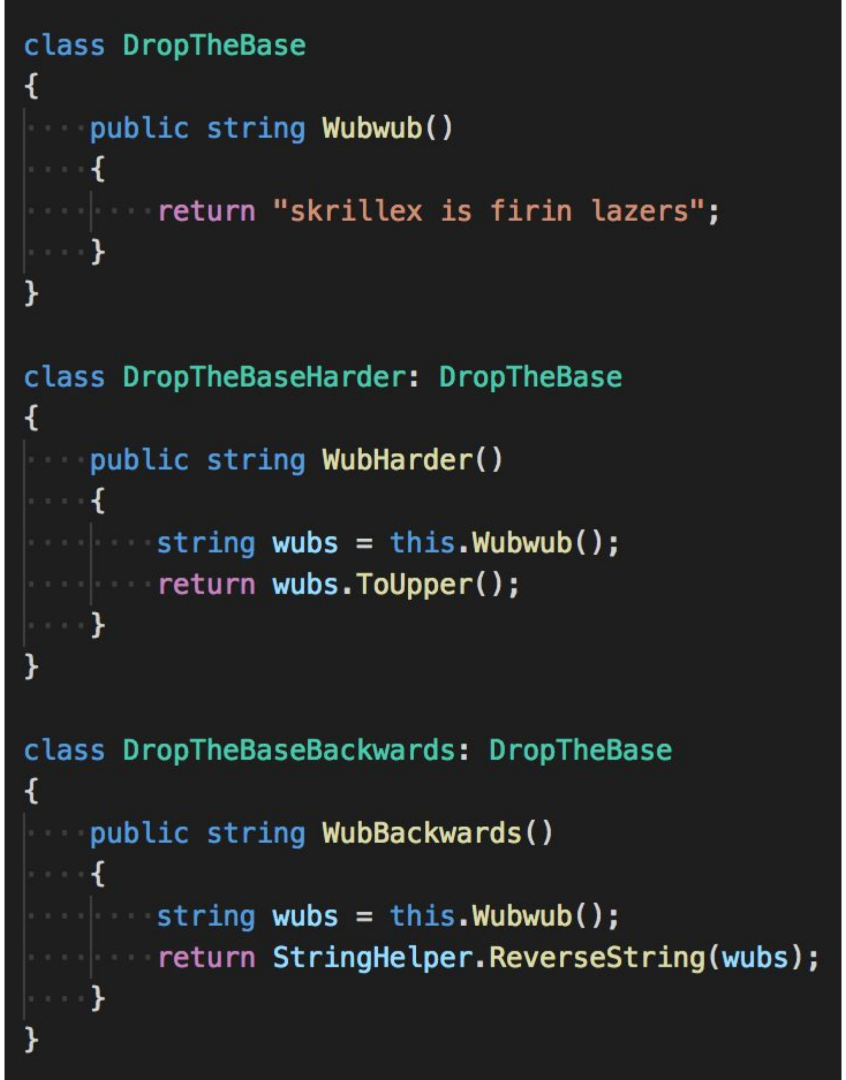
Inheritance should be used judiciously.

Inheritance done sloppily can introduce subtle bugs.

e.g. I augmented the base class’s behaviour in an unexpected way.

Again, like interfaces, not terribly important now but will be later on.

Inheritance Example



```
class DropTheBase
{
    public string Wubwub()
    {
        return "skrillex is firin lazers";
    }
}

class DropTheBaseHarder: DropTheBase
{
    public string WubHarder()
    {
        string wubs = this.Wubwub();
        return wubs.ToUpper();
    }
}

class DropTheBaseBackwards: DropTheBase
{
    public string WubBackwards()
    {
        string wubs = this.Wubwub();
        return StringHelper.ReverseString(wubs);
    }
}
```

Override

Basic Concepts

Technically another concept in inheritance.

Allows you to override a base class’s implementation (vs extend).

Good opportunity to do sketchy things □. (e.g. InstallUtil.exe)

Requires two keywords: `override` and `virtual`

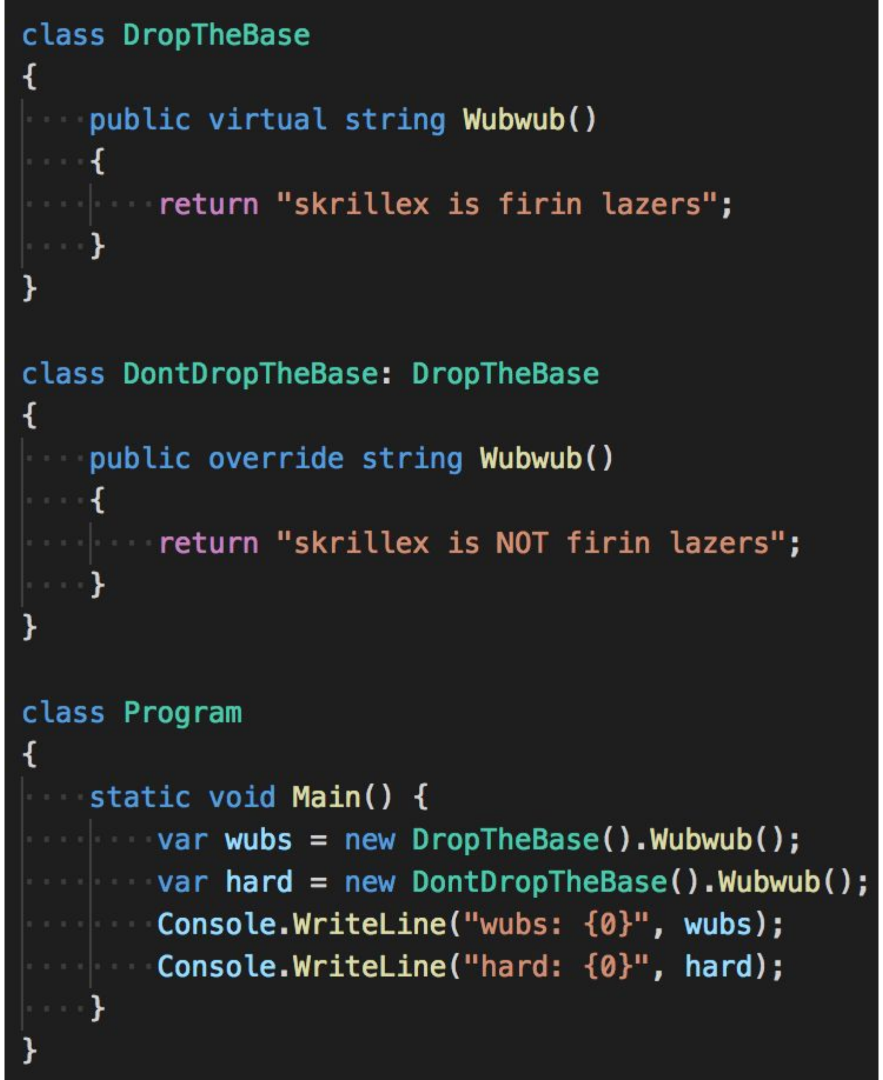
Override

Takeaways

Overrides can confuse code if overused.

Good when you want to reuse a most of the base class code.

Override Example

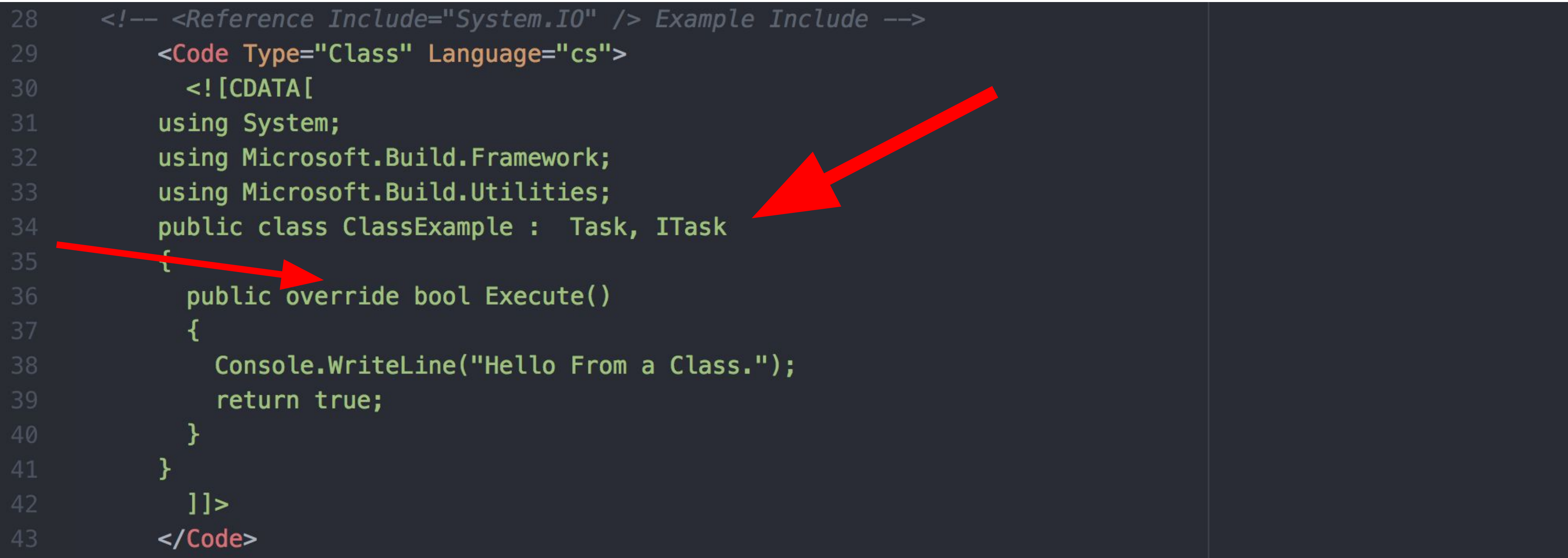


```
class DropTheBase
{
    public virtual string Wubwub()
    {
        return "skrillex is firin lazers";
    }
}

class DontDropTheBase: DropTheBase
{
    public override string Wubwub()
    {
        return "skrillex is NOT firin lazers";
    }
}

class Program
{
    static void Main() {
        var wubs = new DropTheBase().Wubwub();
        var hard = new DontDropTheBase().Wubwub();
        Console.WriteLine("wubs: {0}", wubs);
        Console.WriteLine("hard: {0}", hard);
    }
}
```

Practical Offensive Use of Override

MSBuild Payload:

https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-inline-tasks?view=vs-2017



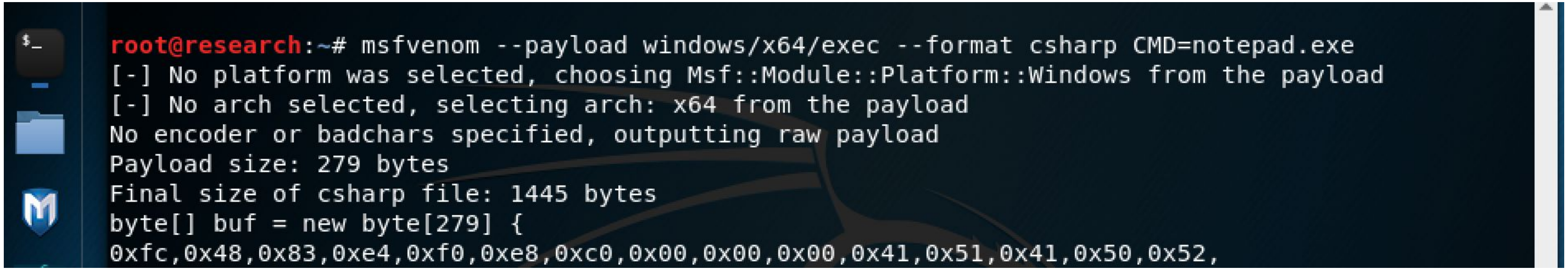
```
  <!-- <Reference Include="System.IO" /> Example Include -->
      <Code Type="Class" Language="cs">
        <![CDATA[
      using System;
      using Microsoft.Build.Framework;
      using Microsoft.Build.Utilities;
      public class ClassExample :  Task, ITask
      {
        public override bool Execute()
        {
          Console.WriteLine("Hello From a Class.");
          return true;
        }
      }
        ]]>
      </Code>
```

Classic Code Injection via C#

Injecting into host process. This is not cross process injection

1. Allocate space in process to stage your shellcode
    a. READ, WRITE, EXECUTE (RWX) Permissions (More on this later)
2. Copy Shellcode to address of allocated memory
3. Trigger execution

Shellcode Generation



```
root@research:~# msfvenom --payload windows/x64/exec --format csharp CMD=notepad.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 279 bytes
Final size of csharp file: 1445 bytes
byte[] buf = new byte[279] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
```

Basic Functions Required To Complete This

1. VirtualAlloc
2. WriteProcessMemory
3. CreateThread

InstallUtil - Application Whitelist Bypass

Let’s build a Shellcode Runner that gets loaded into InstallUtil.exe

First Some Background

InstallUtil.exe is a default app, which is used to setup/register applications

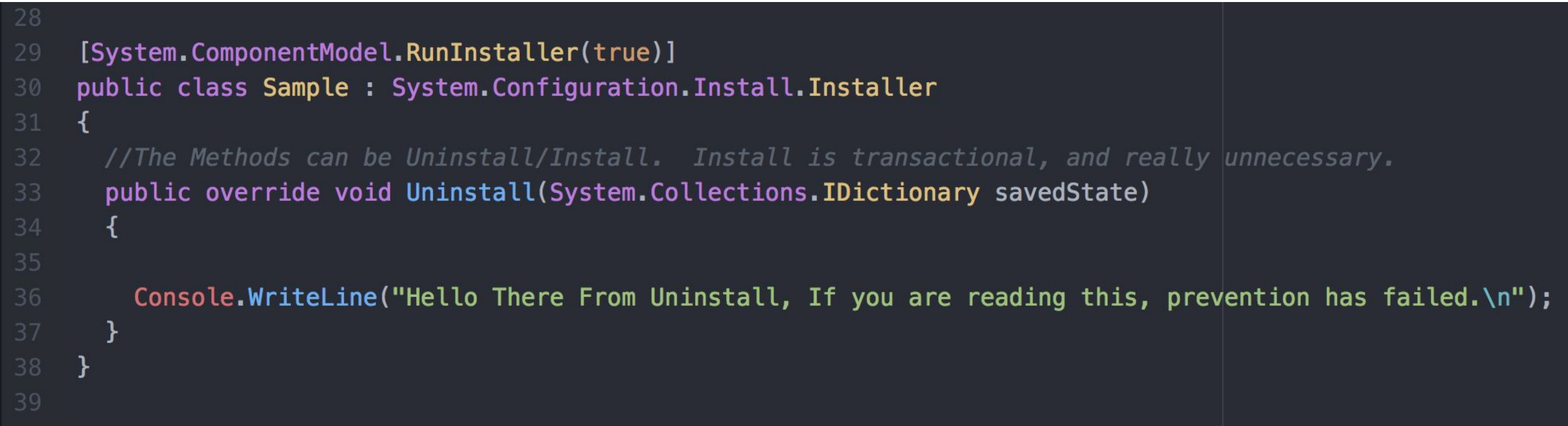
It takes a specially crafted DLL as an input parameter

InstallUtil - EntryPoint, its not Main :)

A class that inherits from *System.Configuration.Install.Installer*

We will then Override the function *Uninstall*

We covered this earlier :)



```
[System.ComponentModel.RunInstaller(true)]
public class Sample : System.Configuration.Install.Installer
{
  //The Methods can be Uninstall/Install.  Install is transactional, and really unnecessary.
  public override void Uninstall(System.Collections.IDictionary savedState)
  {

    Console.WriteLine("Hello There From Uninstall, If you are reading this, prevention has failed.\n");
  }
}

```

15 Minute Lab - Shellcode Runner!

wwhf-2018/Exercises/4/Shellcode.cs

Resources

Workshop Exercises - https://github.com/redcanaryco/wwhf-2018

Matt Warren - https://mattwarren.org

Writes some incredible blog posts about .NET internals.

PInvoke.net - https://pinvoke.net

For all your PInvoke needs.