EVOLUTION OF .NET

Advancing Post-Exploitation Tactics

Kyle Avery

December 2024

OUTFLANK

clear advice with a hacker mindset

WHOAMI

Kyle Avery – @kyleavery_

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

OUTFLANK

- Outflank Security Tooling (OST)
- Red Teaming Services

AGENDA

Introduction

- Offensive security goals
- What is .NET?

Post-Exploitation History

- PowerShell to C#

Current Research

- Stability improvements
- Output capture
- Memory obfuscation

INTRODUCTION – OFFENSIVE SECURITY GOALS

What is the goal of computer security?

- Ensure confidentiality, integrity, and availability

Relationship between offense and defense

- Challenge prevention and detection assumptions
- Defensive strategies are refined based on offensive tactics.

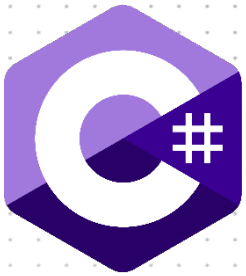
Goals of offensive security

- Identify and exploit vulnerabilities
- Simulate real-world attacks
- Stay ahead of emerging threats

INTRODUCTION – WHAT IS .NET?

The .NET Standard

- Formal specification of APIs required for a programming language



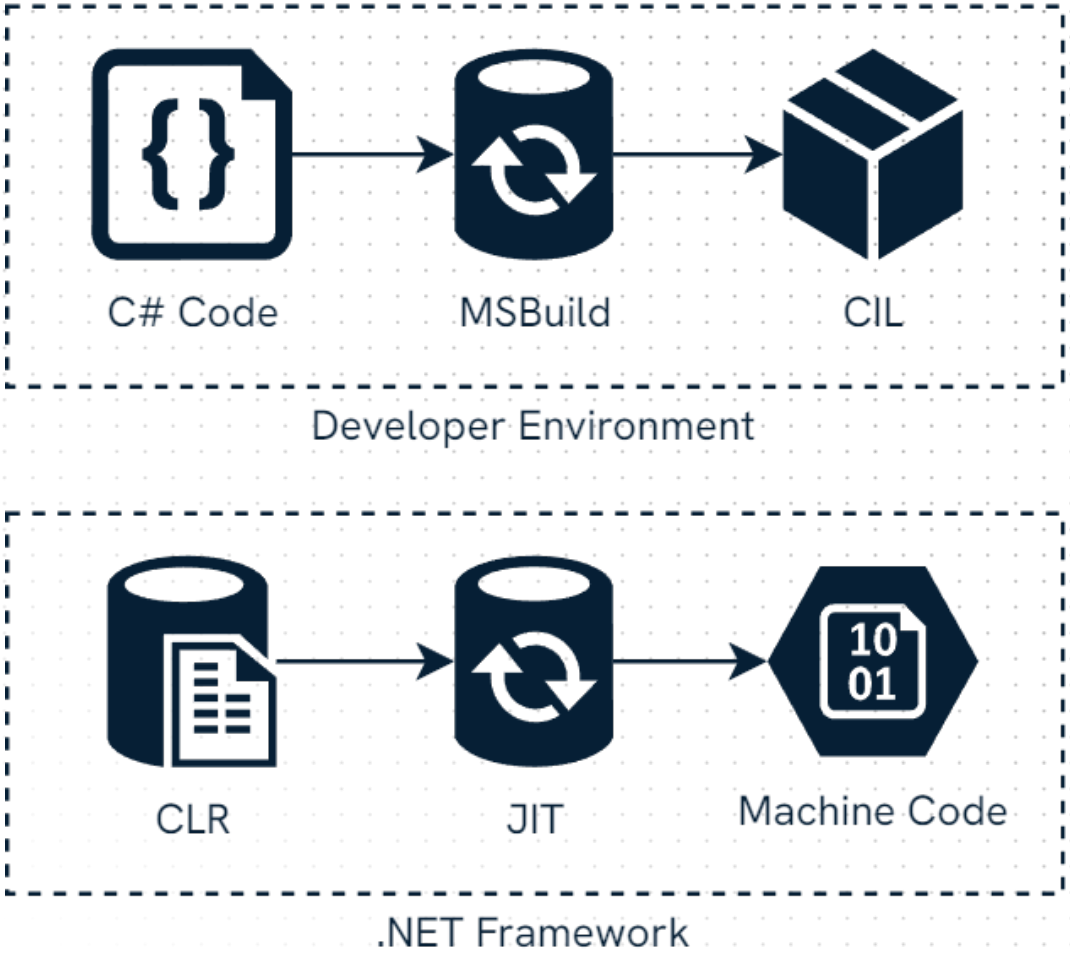
Implementations

- Typically interpreted scripts or bytecode
- Most popular: PowerShell, C#
- Others: VB.NET, F#, IronPython, Boo

INTRODUCTION – WHAT IS .NET?

.NET Framework

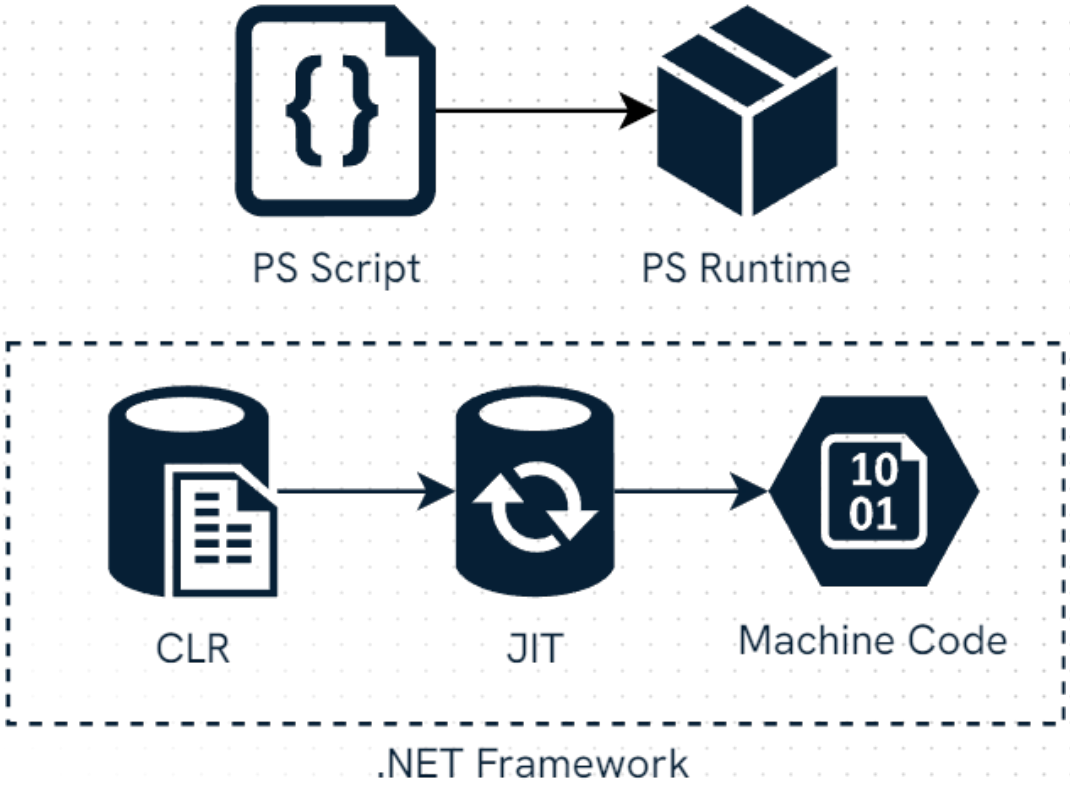
- Preinstalled on Windows
- CIL – compiled C#, F#, VB.NET
- CLR – Memory/task manager
- JIT compilation



INTRODUCTION – WHAT IS .NET?

PowerShell

- Preinstalled on Windows
- Interpreter written in C# - no SDK or compilation required
- Other interpreted .NET languages: JScript, IronPython, Boo

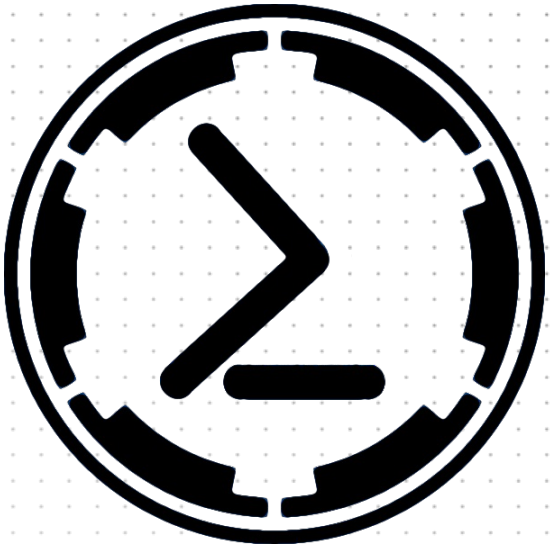


.NET POST-EXPLOITATION HISTORY

OUTFLANK

clear advice with a hacker mindset

POST-EX HISTORY – POWERSHELL



Offense: PowerShell (2012)

- First (public) use of .NET for offensive security
- Open-source tools: PowerShell Empire, PowerSploit, PowerView

Defense: PS Module Logging (2012)

- Logs portions of scripts, but information is obfuscated and incomplete
- Events generated at execution, AV unable to react in real-time

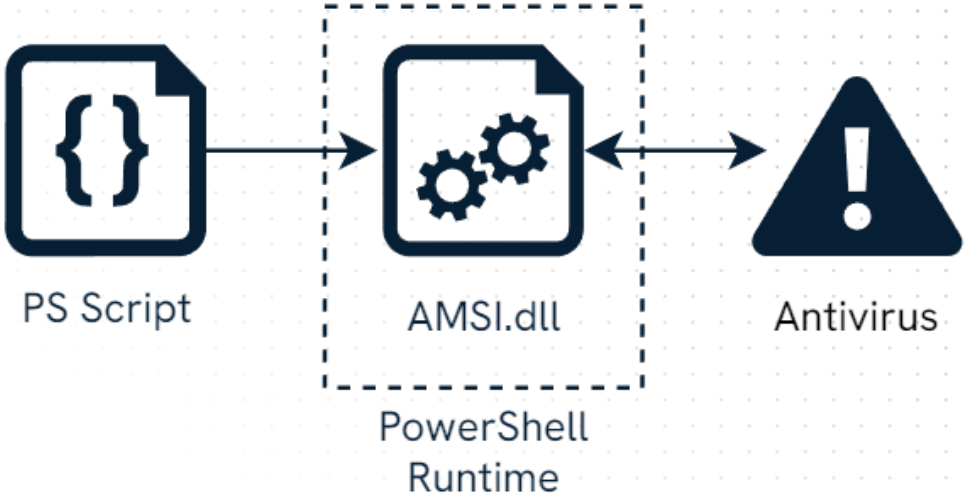
POST-EX HISTORY – POWERSHELL

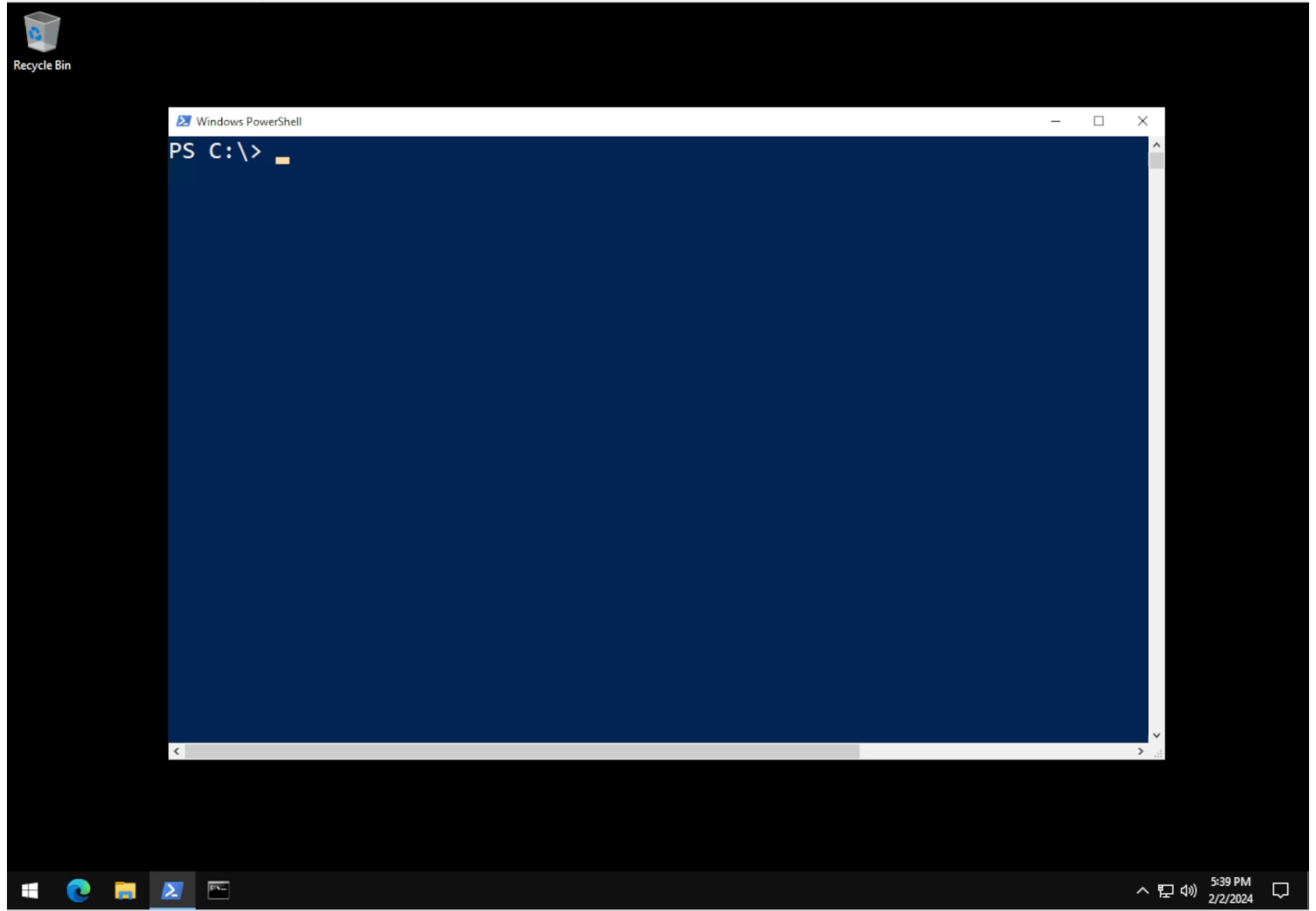
Defense: PS Logging Expanded (2015)

- Script block logging captures the any code as it is executed
- Transcription captures the full input and output of a session

Defense: AMSI (2016)

- Script contents are sent to AV *before* execution for approval




Recycle Bin
Windows PowerShell
PS C:\>
5:39 PM
2/2/2024

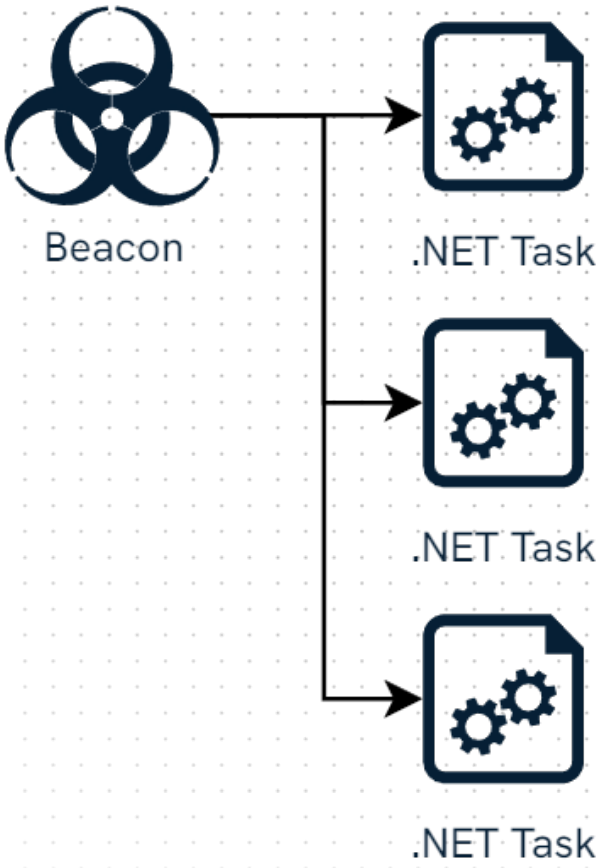
POST-EX HISTORY – POWERSHELL TO C#

Offense: PS Logging/AMSI Bypasses (2016-2018)

- In-process patching to block events and scans

Offense: .NET Assemblies (2018)

- Cobalt Strike creates “execute-assembly”
- Executes .NET CIL in memory, using “fork&run”
- Projects ported to C# – GhostPack, SharpSploit, SharpView



https://github.com/Flangvik/SharpCollection/tree/master/NetFramework_4.5_Any

Cobalt Strike

Cobalt Strike View Payloads Attacks Site Management Reporting Help

pid	process	internal	user	com...	arch	last	sleep	note
82384	beacon.exe	192.168.42.100	User	TEST	x64	4s	5 seconds	

Beacon 192.168.42.100@82384 X

[TEST] - x64 | User | 82384 - x64 last:4s

```
beacon> execute-assembly C:\Users\tester\Downloads\Seatbelt.exe -group=system
```

Size

1 KB
19 KB
75,557 KB
33,907 KB
151 KB
28 KB
154 KB
3 KB
265 KB

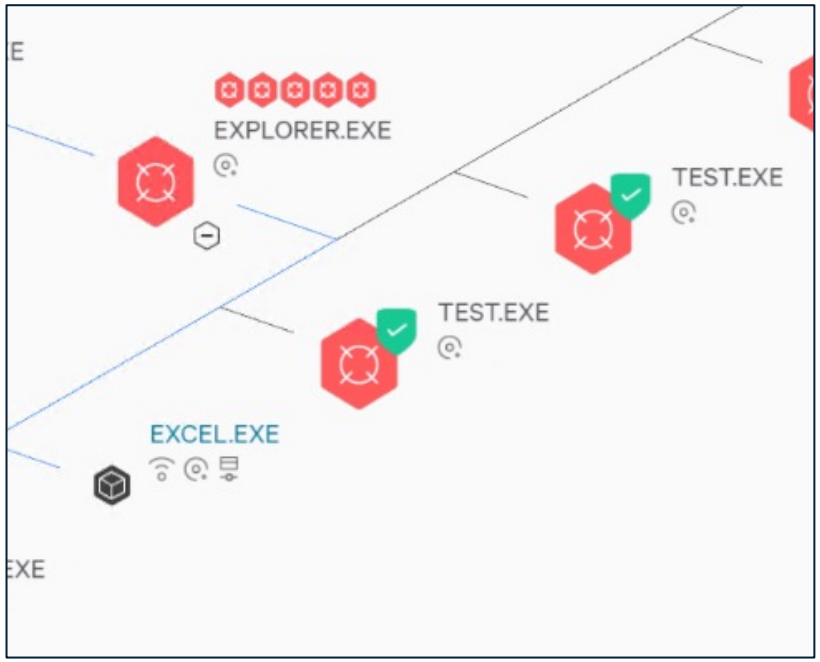
POST-EX HISTORY – C#

Defense: Rise of EDR (2016-2019)

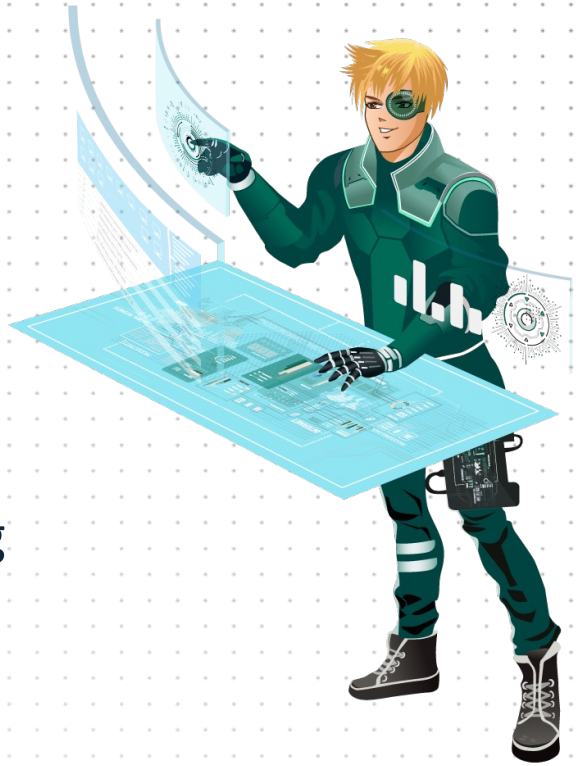
- Term coined in 2013, popularized later
- Collect more data than traditional AV

Defense: AMSI Expanded (2019)

- Key .NET Framework methods submit data



POST-EX HISTORY – C# TO C?



Offense: Beacon Object Files (2020)

- Cobalt Strike creates “inline-execute”
- Some tools ported to C, but not all

Offense: New AMSI Bypass (2020)

- PTP details hardware breakpoints for function hooking

Offense: In-Process .NET (2021)

- IBM X-Force Red publishes BOF to execute .NET assemblies
- Nighthawk released, supports in-process .NET

STATE OF THE ART – MODERN EDR

Defense: EDR Memory Scanning (2020-2022)

- Vendors extend telemetry to include scheduled or triggered scans of memory
- Utilize signatures to quickly detect known malicious bytes or strings

Results - beacon.exe (82384)

946 results.

Addr...	Base...	Len...	Result
0x1a...	0x1...	12	Seatbelt.exe
0x1a...	0x1...	13	Seatbelt.Util
0x1a...	0x1...	16	Seatbelt.Interop
0x1a...	0x1...	22	SeatbeltArgumentParser
0x1a...	0x1...	25	Seatbelt.Commands.Browser
0x1a...	0x1...	96	System.Collections.Generic.IEnumerable<Seatbelt.Commands.
0x1a...	0x1...	88	System.Collections.Generic.IEnumerable<Seatbelt.Commands.
0x1a...	0x1...	93	System.Collections.Generic.IEnumerable<Seatbelt.Commands.
0x1a...	0x1...	86	System.Collections.Generic.IEnumerable<Seatbelt.Commands.
0x1a...	0x1...	17	Seatbelt.Commands
0x1a...	0x1...	35	Seatbelt.Commands.Windows.EventLogs

SentinelOne™

VMware Carbon Black™

elastic

CROWDSTRIKE

STATE OF THE ART – .NET OBFUSCATION

Offense: .NET Obfuscation (2022)

- Outflank adds SharpFuscator to obfuscate and encrypt .NET assemblies before execution

Offense: .NET Memory Obfuscation (2023)

- Nighthawk adds CLR garbage encryption to mask .NET assemblies after execution

CURRENT RESEARCH

Considerations for in-process execution

1. Stability – Unhandled exceptions threaten the implant
2. Environment – We may not have control over the host process
3. Memory artifacts – CLR garbage collection means memory is out of our control

CURRENT RESEARCH – STABILITY

Patching .NET Frameworks APIs

- Some .NET functions cause issues that don't impact fork&run implementations
- We can resolve any .NET function reflectively and use traditional patching

```
using System;

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();
```

https://www.outflank.nl/blog/2024/02/01/unmanaged-dotnet-patching/

Stage1 C2 implant builder - Out | PE Payload generator - Outflank | Implant - Stage1 C2 Server

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search CSADemos

Debug | Any CPU | EnvExitPatch | Start

Program.cs

EnvExitPatch | EnvExitPatch.Program | Main(string[] args)

```
1  using System;
2
3  namespace EnvExitPatch
4  {
5      internal class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("About to exit...");
10             Environment.Exit(0);
11             Console.WriteLine("Survived exit!");
12         }
13     }
14 }
15
```

124 % | No issues found | Ln: 14 | Ch: 2 | SPC | CRLF

Solution Explorer

Search Solution Explorer (Ctrl+;)

- EnvExitPatch
  - Properties
  - References
  - App.config
  - Program.cs
- HandleException
  - Properties
  - References

Solution E... | Git Chang... | Class View

Properties

Output

Show output from: Build

```
Rebuild started...
1>------ Rebuild All started: Project: EnvExitPatch, Configuration: Debug Any CPU ------
1>  EnvExitPatch -> C:\Users\User\Documents\CSADemos\EnvExitPatch\bin\Debug\EnvExitPatch.exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
========== Rebuild started at 2:51 PM and took 00.170 seconds ==========
```

Error List | Output | Find Symbol Results

Rebuild All succeeded | Add to Source Control | Select Repository

Implant

Hostnam

Username

UID / reci

Console

Press ente

2:53 PM 2/2/2024

Cobalt Strike

Cobalt Strike View Payloads Attacks Site Management Reporting Help

pid	process	internal	user	com...	arch	last	sleep	note
95008	explorer.exe	192.168.42.100	User	TEST	x64	366ms	1 second (3...	

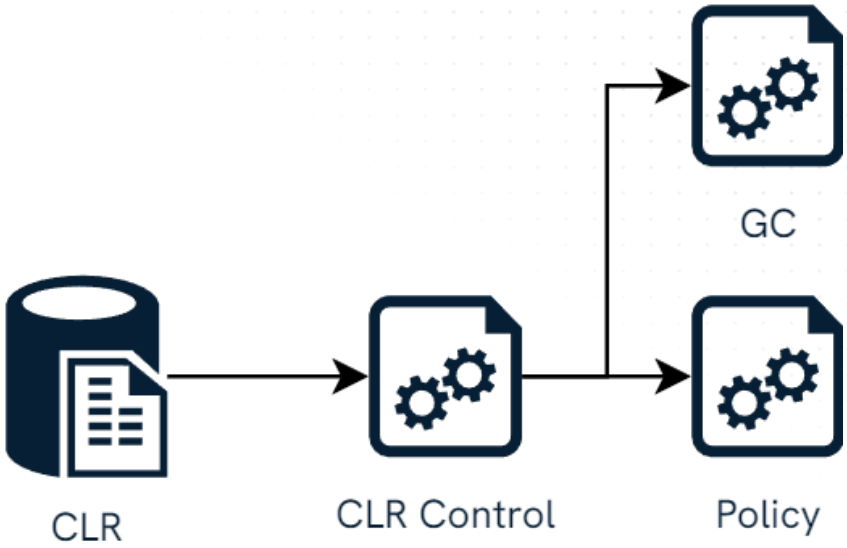
Scripts X | Script Console X | Beacon 192.168.42.100@95008 X

```
[TEST] - x64 |  User | 95008 - x64                                  last:366ms
beacon>
```

CURRENT RESEARCH – STABILITY

Other threats to stability

- Unhandled exceptions still threaten our implant
- CLR host can set a “Failure Escalation Policy”




```
using System;

namespace HandleException
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("About to throw...");
            throw new Exception("Uh oh!");

            // This line will never be reached
            Console.WriteLine("Survived throw!");
        }
    }
}
```

Stage1 C2 implant builder - Out | PE Payload generator - Outflank | Implant - Stage1 C2 Server

Implant

Hostnam

Usernam

UID / reci

Console

```
[2/2/2024 2
[*] Running

About to ex
Survived ex
```

Press ente

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search CSADemos

Debug | Any CPU | EnvExitPatch | Start

Program.cs

HandleException | HandleException.Program | Main(string[] args)

```
using System;

namespace HandleException
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("About to throw...");
            throw new Exception("Uh oh!");

            // This line will never be reached
            Console.WriteLine("Survived throw!");
        }
    }
}
```

124 % | 0 | 1 | Ln: 1 | Ch: 1 | SPC | CRLF

Solution Explorer

Search Solution Explorer (Ctrl+;)

- References
- App.config
- Program.cs
- HandleException
  - Properties
  - References
  - App.config
  - Program.cs

Solution E... | Git Chang... | Class View

Properties

Output

Show output from: Build

Error List | Output | Find Symbol Results

Ready | Add to Source Control | Select Repository

2:54 PM 2/2/2024

CURRENT RESEARCH – ENVIRONMENT

Capturing output

- Inline-ExecuteAssembly creates a "conhost" child process
- Public solutions:
  - Create a new process with a console
  - Hook Write/WriteLine
- There is another way!

explorer.exe	40420
beacon.exe	82384
conhost.exe	39580

Cobalt Strike

Cobalt Strike View Payloads Attacks Site Management Reporting Help

...	pid	process	internal	...	user	com...	arch	last	sleep	note
...	42568	EXCEL.EXE	192.168.42.100	...	User	TEST	x64	1s	1 second (3...	

Scripts X | Beacon 192.168.42.100@42568 X | Script Console X

[TEST] - x64 | User | 42568 - x64 last:1s

beacon>

Stage1 C2 implant builder - Out
PE Payload generator - Outflank
Implant - Stage1 C2 Server
Implant - Stage1 C2 Server

System Informer [TEST\User] (Administrator)

System View Tools Users Help

Refresh Options

excel

Processes Services Network Disk Firewall Devices

Name	PID
EXCEL.EXE	77496

CPU usage: 4.83% Physical memory: 10.86 GB (89.21%

Stage

Dashboa

Implant

Downloa

Settings

Docume

Kyle Avery

Comments Share

Cells Editing Sensitivity Add-ins Analyze Data

Sensitivity Add-ins

Ready Accessibility: Good to go

100%

mmand

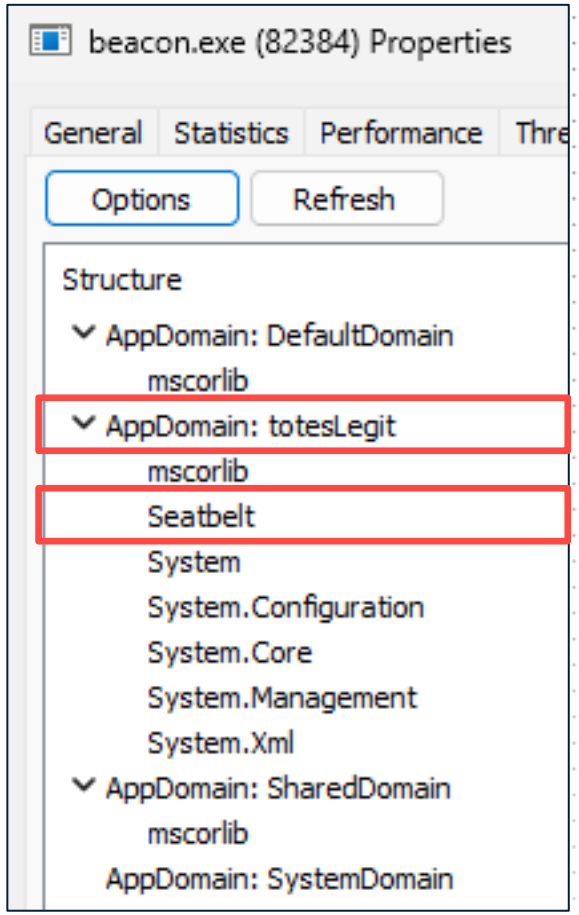
Go!

2:59 PM
2/2/2024

CURRENT RESEARCH – MEMORY ARTIFACTS

PE headers and App Domains

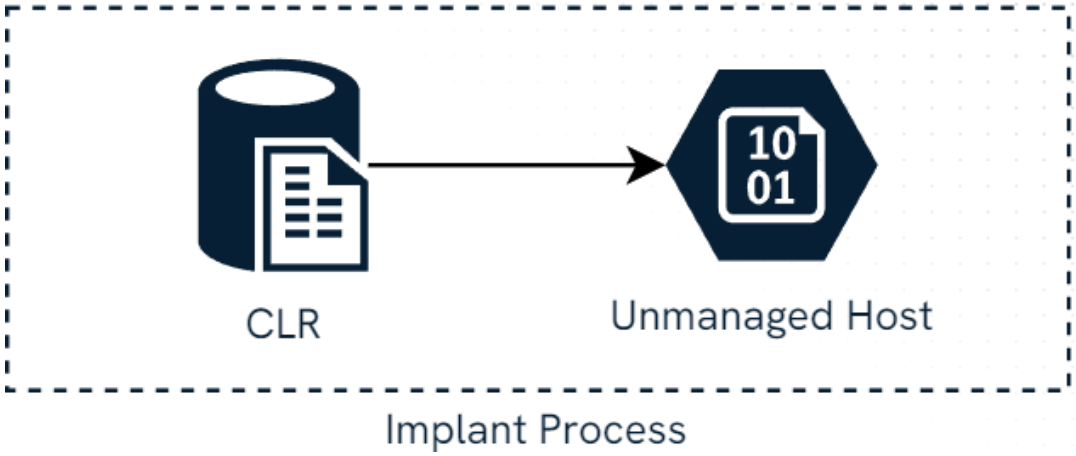
- .NET assemblies are still PE files, they have headers that can be found in memory
- PE headers can be safely wiped after an assembly is loaded by the CLR
- .NET keeps a record of it, even after its unloaded

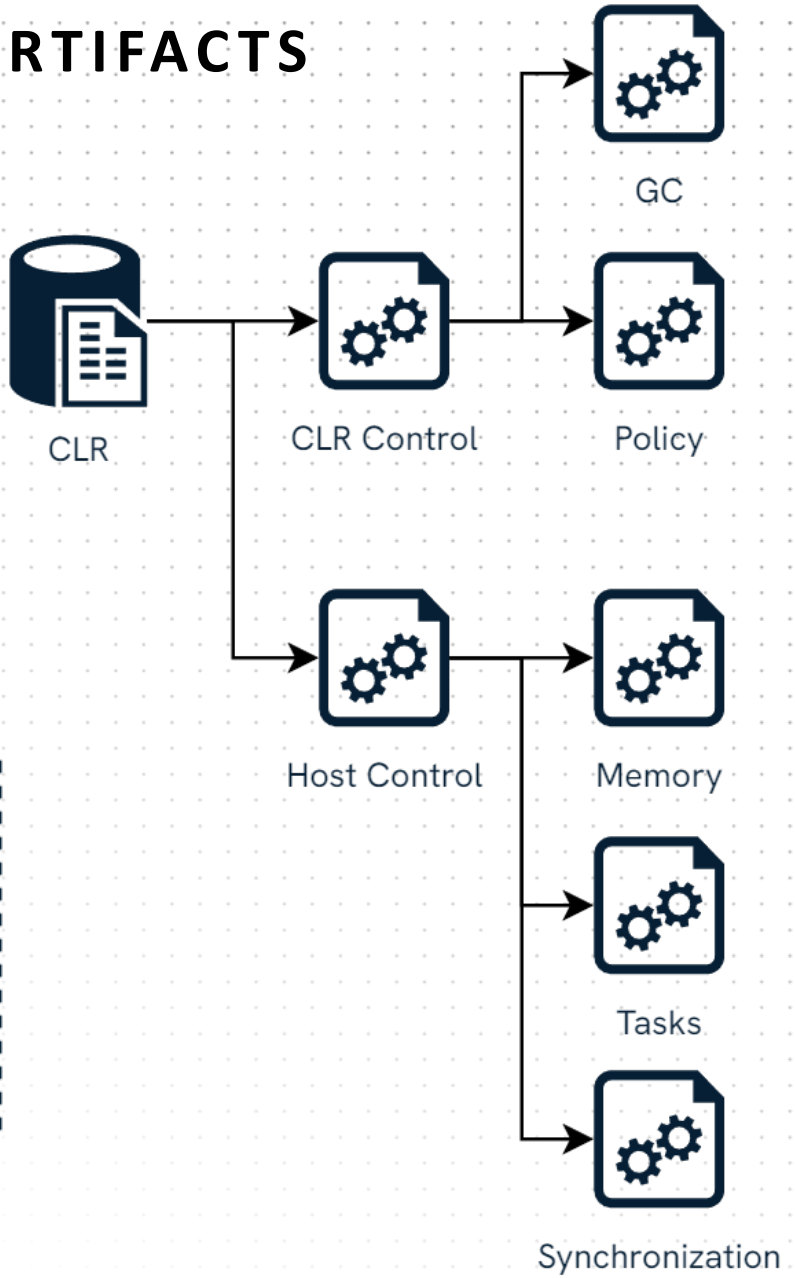


CURRENT RESEARCH – MEMORY ARTIFACTS

CLR unmanaged host APIs

- Key CLR interfaces can be overridden to control their functionality:
  - Memory allocations
  - Task management
  - Synchronization





Stage1 C2 implant builder - Out | PE Payload generator - Outflank | Implant - Stage1 C2 Server | Implant - Stage1 C2 Server

192.168.42.102:11000/implant.html?uid=LVB14RV8 120%

kyle | test | 09-01-2024

Implant details

Hostname	TEST	**Process**	38384 (dllhost.exe) x64	**First seen**	2/2/2024 3:05:25 PM
Username	TEST\User	**OS**	Windows 11.0 (OS Build 22621)	**Last seen**	3:05:42 PM (2s)
UID / recipe / version	/ S9lPy / 2.8.0 LVB14RV8	**Note**		**Kill date**	2/3/2024 4:26:33 AM

Console

```
 ________ __                    ____
/   ____// |_____     ____   ____   /_   |
\____  \\  __\__  \   / ___\_/ __ \   |   |
/       \| |  / __ \_/ /_/  >  ___/   |   |
/_______  /|__| (____  /\___  / \___  >  |___|
        \/           \//_____/      \/
```

Press enter to send command | Go!

3:05 PM 2/2/2024

OUTFLANK

clear advice with a hacker mindset