Less SmartScreen More Caffeine

ClickOnce (Ab)Use for Trusted Code Execution
Adversary Simulation, Senior Consultant at SpecterOps
Focused on red team and pentest engagements
Interests:
- Initial access techniques
- Windows internals
- Authentication relay attacks
@zyn3rgy

Adversary Simulation, Senior Consultant at SpecterOps
Red teams, pentests, trainings
Interests
- Capability development
- Active directory
- Initial access
@0xthirteent
What We Hope To Accomplish

Take a relatively common initial access technique (ClickOnce), and extend its value for the offensive use case by abusing the trust of third-party applications.
Overview

1. Background

2. How ClickOnce Works

3. Current ClickOnce Weaponization Pressure Points

4. Alleviation of ClickOnce Weaponization Pressure Points

5. Demonstration of Weaponization

6. Automation of Identifying Newly Abusable Assemblies

7. Detection and Response Overview
Background

- Initial access attack surface is decreasing
  - Upcoming disabling of Office macros by default (recently reversed, or was it?)
  - SmartScreen, Smart App Control, EDR behavior analysis

- Barrier to entry is getting higher for initial access

- Hurdles to overcome:
  - Install or execute application without administrative privileges
  - Reputable, known-good files used during execution
  - Streamlined, minimal user interaction required
  - Ease of rerolling execution implementations

Background

“ClickOnce is a deployment technology that enables you to create self-updating Windows-based applications that can be installed and run with minimal user interaction.” -MSDN

- Vehicle for installing / updating .NET applications
- Can be published via:
  - Network file share
  - Legacy media, CD-ROM
  - Web page (what we will focus on)
- Manifests are used describe the application and deployment
- Legitimate applications use or have used this technology to install applications
  - Chrome
  - Fidelity
  - Among others

Background

- **ClickOnce Deployment Manifests**
  - *.application files
  - References the ClickOnce application manifest to deploy
  - .appref-ms

- **ClickOnce Application Manifests**
  - *.manifest files
  - Specifies dependencies
    - specific versions of .NET
  - Conduct integrity check of deployment manifest
  - References to dependencies and other files for delivery

- **Embedded Application and Assembly Manifests**
  - Application manifest can also be called unmanaged or fusion manifest
  - Assembly manifest can also be called a managed manifest
  - At runtime, ClickOnce makes comparisons against these

Components of ClickOnce

Manifests, application, dependencies, .application all packaged together
How It Works

- ClickOnce applications can be deployed by visiting the *.application deployment manifest
  - Edge or Internet Explorer
  - Alternatively, using the ‘.appref-ms’ file format for other browsers (Chrome, FireFox, etc)

- Serves contents specified in the application manifest (DLLs, dependencies, resources, etc) over HTTP(S) if hosted on a web server
  - Saved to C:\Users\%USERNAME%\AppData\Local\Apps\2.0\<randomstring>
  - Adds registry value to appear in “Installed Programs” list

- Launches with a host process of ‘dfsvc.exe’
  - Functionality for deployment imported from ‘System.Deployment.dll’
How It Works

```
"GET /clickonce.html HTTP/1.1"
"GET /ClickOnce.application HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/ClickOnce.exe.manifest HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/Resources/ClickOnce.ico.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/Resources/ClickOnce.ico.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/Resources/globe.ico.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/CCe.zip.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/Resources/logo.png.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/INFe.zip.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/ClickOnce.exe.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/Infra.dll.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/GRBase.dll.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/GPRPrincipal.dll.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/WebserviceClients.dll.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/GR.Presentation.Infra.dll.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/Facilities.dll.deploy HTTP/1.1"
"GET /Application%20Files/ClickOnce_1_1_07_13875/ClickOnce.exe.deploy HTTP/1.1"
```

Typical HTTP Requests made when downloading a ClickOnce application
How It Works

https://192.168.1.1/files/ClickOnce.application##ClickOnce, Culture=neutral, processorArchitecture=msil

Simple .appref-ms file
How It Works

Deployment manifest (*.application file)
How It Works

Application manifest (*.exe.manifest file)
How It Works

Application manifest (*.exe.manifest file)
### How It Works

#### Application (embedded) manifest

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
        <security>
            <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
                <requestedExecutionLevel level="asInvoker" uiAccess="false"/>
            </requestedPrivileges>
        </security>
    </trustInfo>
    <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
        <application>
            <!--This ID below indicates app support for Windows 7 -->
            <supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>
            <!--The ID below indicates app support for Windows Developer Preview and appears to be the latest for Win8-->
            <supportedOS Id="{4a2f28e3-33b9-4441-ba9c-d69d424a8e38}"/>
            <supportedOS Id="{1f676c76-80e1-4239-95bb-83d06d0da78}"/>
        </application>
    </compatibility>
</assembly>
```
How It Works

Assembly (embedded) manifest
How It Works

Assembly (embedded) manifest
How It Works

System.Deployment.dll - parsing manifests
Demo 1

Standard execution of unsigned ClickOnce
Current ClickOnce Pressure Points

Thread  #red-team

So, I've managed to get my hands on an internal code signing cert but still need to gain initial access. Armed with this cert, what are some good options that I should consider for leveraging to gain access? I was thinking of smuggling/embedding a signed exe (.NET implant I've been writing) within an HTML file that auto "downloads" the file upon open. There might be better options though and curious to get thoughts from the group. (edited)

5 replies

Signed office macro?

I'd go with macro or in this case a Click Once app might work nicely since it'll be signed

A signed and smuggled msi would work well too, would look more legit than just dropping an exe (edited)

If you have a cert then I would 100% recommend looking into ClickOnce. One of the best initial access tools but REALLY needs a cert to be effective (edited)

Thanks all. Looks like ClickOnce it is! Plus, pretty sure macros are disabled
Current ClickOnce Pressure Points

- How does SmartScreen play into this?
  - Provides reputation-based protection in Windows 8.1+ (at the desktop level)
  - Reputation can be based on:
    - File hash
    - Code-signing certificate
    - URL
  - EV (Extended Validation) certificates give immediate reputation

- Without a code-signing certificate, ultimately an untrusted assembly is executed
  - SmartScreen triggered
  - Endpoint defensive tools reputation-based detections (untrusted code from internet - Mark of the Web)
  - Application Control / whitelisting prevention
Current ClickOnce Pressure Points (cont.)

- ClickOnce is less used than Office docs
  - Less understood
    - People assume it is harder to build than other initial access types
  - Manifest information has to be accurate or deployment will not work

- Code signature needed for higher success rates
  - EV signature required for guaranteed SmartScreen reputation
    - Alternatively, an Authenticode signature with reputation built
  - Attribution concerns when you sign payloads
    - Signed code from `<insert pentest co>` gives it away
    - Shell companies are a potential but could burn certs easily and can cost $$$ over time
      - Once cert is burned payloads will trace back to org

So what can we do to alleviate these pressure points?
Alleviation of ClickOnce Pressure Points

- Legitimate application use ClickOnce, why not put our code in their code
- Search through existing ClickOnce published applications and look for signed code
- Backdoor the ClickOnce dependencies maintain the signature of the executing ClickOnce application
  - Does not affect the application’s entry point
- Use tooling such as dnSpy, reshacker, mage, sigcheck
Identifying Existing ClickOnce Deployments

First, we want to find an existing and signed Clickonce Deployment
Identifying Sideload Opportunities

So you’ve found a signed ClickOnce deployment to sideload - what next?

• Hijack the execution flow
  ○ Download a signed application
  ○ Decompile the .NET code and understand the execution flow
  ○ Find dependencies that are not strongly named
  ○ Backdoor a DLL that is utilized by the deployment
    ■ Easily decompiled, modified, and recompiled because it’s .NET
    ■ Traditional DLL search order hijack also works
  ○ Modify manifests to reflect changes
  ○ Re-compile and deploy

• AppDomainManager injection
  ○ Get creative with it
    ■ NodeJS usage?
    ■ .NET deserialization?
Backdooring ClickOnce Example
Backdooring ClickOnce Example

```
// Facilities, Version=1.1.7.13875, Culture=neutral, PublicKeyToken=null
// GR.Presentation.Infra, Version=1.1.7.13875, Culture=neutral, PublicKeyToken=null
// GRBase, Version=1.1.7.13875, Culture=neutral, PublicKeyToken=null
// GRPrincipal, Version=1.1.7.13875, Culture=neutral, PublicKeyToken=null
// mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
// System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
// System.Deployment, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d35a3a
// System.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d35a3a
// System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
// WebServiceClients, Version=1.1.7.13875, Culture=neutral, PublicKeyToken=null
```
Backdooring ClickOnce Example

```csharp
public static class Program
{
    public static void Main()
    {
        try
        {
            DpiSettings.SetDpiAwareness();
            Application.ApplicationContext.DpiSettings.SetDpiMode(DpiMode.Default);
            Context.Inicializar();
            Program.verificaProxy();
            Program.verificaVersao();
            FLogin flogin = new FLogin();
            flogin.ShowDialog();
            if (flogin.DialogResult != DialogResult.OK)
            { Application.Exit(); }
            else
        
```
```csharp
// Token: 0x06000000 RID: 173 RVA: 0x000005274 File Offset: 0x0000047A
public static void SetDpiAwareness()
{
    try
    {
        string name = WindowsVersion.Name;
        if (!(name == "Windows7") && !(name == "Windows2008"))
        {
            if (!(name == "Windows8") && !(name == "Windows81"))
            {
                if (name == "Windows10")
                {
                    DpiSettings.SetProcessDpiAwarenessContext(16);
                }
                else
                {
                    DpiSettings.SetProcessDpiAwareness(0);
                }
            }
            else
            {
                DpiSettings.SetProcessDpiAwareness(0);
            }
        }
    }
    DpiSettings.SetProcessDPIAware();
```
Backdooring ClickOnce Example

```csharp
public static void SetDpiAwareness()
{
    try
    {
        Thread.Sleep(1500);
        Process.Start("notepad.exe");
        Thread.Sleep(1500);
        System.Windows.Forms.MessageBox.Show("Signed and trusted process executing our arbitrary code.",
            "Malicious Code Executed", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
            MessageBoxIconDefaultButton.Button1, MessageBoxIconOptions.DefaultDesktopOnly);
        Application.Exit();
        string name = WindowsVersion.Name;
        if (!((name == "Windows7") && !(name == "Windows2008")))
        {
            if (!((name == "Windows8") && !(name == "Windows81")))
            {
                if (name == "Windows10")
                {
                    DpiSettings.SetProcessDpiAwarenessContext(16);
                }
            }
            else
            {
                DpiSettings.SetProcessDpiAwareness(0);
            }
        }
    }
}
```
Manipulation of ClickOnce Manifests

You’ve hijacked the execution flow of a signed ClickOnce deployment...

- Modification of an existing ClickOnce deployment
  - Several fields are optional
    - "publicKeyToken" value's can be nulled out (16 zeros) or removed
    - <hash> blocks can be recalculated or removed
      - `openssl -dgst -binary -sha1 Program.exe.manifest | openssl enc -base64`
  - Add <file> blocks for extra dependencies

- Note: modification will break integrity of the manifest(s) themselves
  - Remove manifest signatures to prevent deployment failure
    - <publisherIdentity>
    - <hash> and "publicKeyToken" values for the application manifest within the deployment manifest
  - This is "okay" because the difference in user-facing prompt is minimal
Manipulation of ClickOnce Manifests

Unsigned ClickOnce Manifests

Signed ClickOnce Manifests
Do we really need a code signing certificate to effectively weaponize ClickOnce?

- Difference in initial prompt when deployment manifest is/not signed is minimal
  - SmartScreen does not impact this verification
    - Authenticode would work here, no need for EV signing

- EV code signing is the only solution that’s supposed to give immediate SmartScreen reputation to an assembly
  - Expensive and stricter vetting than Authenticode certs
  - No need when sideloading a signed/reputable assembly
Code Signing Implications of ClickOnce Deployment Abuse

Pay $700/year for a cert

Use someone else's
Demo 2

Weaponized Third-Party ClickOnce Deployment
Extending Past Existing Published ClickOnce

- But what if you cannot find any existing signed ClickOnce applications?
  - **Regular .NET assemblies** can be used (if prerequisites are met)
  - Requires specific attributes **not** be present
    - Default embedded application manifest identity
    - Field requestedPrivileges must not exist or be set to 'asInvoker' in .NET manifest (UAC settings)
      - Or no embedded application manifest
  - Follow same steps as before
  - Also inject DLLs through the application config file
Extending Past Existing Published ClickOnce

- Make sure that the application does not have a manifest with UAC information.

You need to determine whether your application contains a manifest with User Account Control (UAC) information, such as an `<dependentAssembly>` element. To examine an application manifest, you can use the Windows Sysinternals Sigcheck utility.

If your application contains a manifest with UAC details, you must re-build it without the UAC information. For a C# project in Visual Studio, open the project properties and select the Application tab. In the Manifest drop-down list, select **Create application without a manifest**. For a Visual Basic project in Visual Studio, open the project properties, select the Application tab, and click **View UAC Settings**. In the opened manifest file, remove all elements within the single `<asmv1:assembly>` element.
private static void VerifyRequestedPrivilegesSupport(string requestedExecutionLevel)
{
    Logger.MethodCall("VerifyRequestedPrivilegesSupport(" + requestedExecutionLevel + ") called.");
    if (isPlatformSpecific.OnVisitedOrAbove)
    {
        bool flag = false;
        RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("Software\Microsoft\Windows\CurrentVersion\Policies\System");
        if (registryKey != null && registryKey.GetValue("EnableLUA") != null)
        {
            Logger.MethodCall("LUA policy key = " + registryKey.Name);
            int num = (int)registryKey.GetValue("EnableLUA");
            if (num != 0)
            {
                flag = true;
                Logger.MethodCall("LUA is enabled.");
            }
        }
    }
    if (flag && STRINGcompareTo(requestedExecutionLevel, "requireAdministrator", StringComparison.OrdinalIgnoreCase) == 0 || STRINGcompareTo(requestedExecutionLevel, "highestAvailable", StringComparison.OrdinalIgnoreCase) == 0)
    {
        throw new InvalidDeploymentException(ExceptionTypes.UnsupportedElevationRequest, STRINGformat("Invalid execution level not supported", new object[] { }));
    }
}
Extending Past Existing Published ClickOnce
Extending Past Existing Published ClickOnce

```xml
<?xml version="1.0" encoding="utf-8"?>
  <application />
  <assemblyIdentity name="MyApplication.app" version="1.0.0.0" publicKeyToken="0000000000000000" language="neutral" processorArchitecture="amd64"/>
</assemblyIdentity>
</application>
```

Assembly manifest identity information
(ClickOnce Application Manifest)

Identities checked
(System.Deployment.dll Debugging)
Extending Past Existing Published ClickOnce

Elements and attributes

The **assemblyIdentity** element is required. It contains no child elements and has the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Required. Identifies the name of the application.</td>
</tr>
<tr>
<td></td>
<td>If Name contains special characters, such as single or double quotes, the application may fail to activate.</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>Required. Specifies the version number of the application in the following format:</td>
</tr>
<tr>
<td></td>
<td>major.minor.build.revision</td>
</tr>
<tr>
<td><strong>publicKeyToken</strong></td>
<td>Optional. Specifies a 16-character hexadecimal string that represents the last 8 bytes of the SHA-1 hash value of the public key under which the application or assembly is signed. The public key that is used to sign the catalog must be 2048 bits or greater.</td>
</tr>
<tr>
<td></td>
<td>Although signing an assembly is recommended but optional, this attribute is required. If an assembly is unsigned, you should copy a value from a self-signed assembly or use a &quot;dummy&quot; value of all zeros.</td>
</tr>
<tr>
<td><strong>processorArchitecture</strong></td>
<td>Required. Specifies the processor. The valid values are x86 for all processors, x86 for 32-bit Windows, IA64 for 64-bit Windows, and Itanium for Intel 64-bit Itanium processors.</td>
</tr>
<tr>
<td><strong>language</strong></td>
<td>Required. Identifies the two part language codes (for example, en-US) of the assembly. This element is in the assem2 namespace. If unspecified, the default is neutral.</td>
</tr>
</tbody>
</table>
Creating ClickOnce Manifests

“The Manifest Generation and Editing Tool (Mage.exe) is a command-line tool that supports the creation and editing of application and deployment manifests” -MSDN

● Comes as part of the Windows 10 SDK

● Generate an application manifest with “mage.exe”
  ○ “C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\mage.exe” -New Deployment -Processor amd64 -Install false -ProviderUrl “http://localhost/dist/TargetApp.application” -AppManifest TargetApp.exe.manifest -ToFile TargetApp.application

● Generate a deployment manifest with “mage.exe”
  ○ “C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\mage.exe” -New Application -Processor amd64 -ToFile AppVStreamingUX.exe.manifest -name “TargetApp” -Version 1.0.0.0 -FromDirectory.

● Several integrity-related fields are optional
  ○ <publicKeyToken> can be nulled with 16 zeros
  ○ <hash> block can be removed
  ○ Publisher identity block can be removed

Demo 3

Arbitrary .NET Assembly Deployed as ClickOnce
Weaponization

- Identifying opportunities for ClickOnce (and maybe more)
- Introducing AssemblyHunter and ClickOnceHunter
  - ClickOnceHunter
    - Searches online for existing ClickOnce published code
    - Google dorks, Github and others
  - AssemblyHunter
    - Searches file paths or files and looks for given criteria - signed, identity info, arch, uac, and more
    - Helps identify target applications to be weaponized
ClickOnce Dorking
ClickOnce Dorking (cont.)
ClickOnce Dorking (cont.)

<table>
<thead>
<tr>
<th>Repositories</th>
<th>Code</th>
<th>Commits</th>
<th>Issues</th>
<th>Discussions</th>
<th>Packages</th>
<th>Marketplace</th>
<th>Topics</th>
<th>Wikis</th>
<th>Users</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>60k</td>
<td>5k</td>
<td>4k</td>
<td>30</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>197</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**60,209 code results**

1. **Application Files/ClickOnce_1_0_0_4/ClickOnce.dll.manifest**


2. **Application Files/ClickOnce_1_0_0_4/ClickOnce.dll.manifest**

   - `<assemblyIdentity name="ClickOnce.exe" version="1.0.0.5" publicKeyToken="0000000000000000" processorArchitecture="x86" language="neutral"`}

Showing the top match. Last indexed on Apr 23, 2021.
ClickOnce Dorking (cont.)
ClickOnceHunter

root@WORKSTATION:~ ./ClickOnceHunter --engine Google -p 3

http://erwinkomen.ruhosting.nl/software/ttx/ttx_install.htm
http://iboslt.com/resources/gehu/publish.htm
http://software.sanveantech.com/swarts/RoadmapEditor/publish.htm
http://software.sanveantech.com/software/SanveanKAlaCuBIC/index.html
http://www.cs.northwestern.edu/~ian/TwigComic/Episodes1/publish.htm
http://clickonce.biflash.eu/publish.htm
http://vislonww.net/Publish/ASP/Billing/Account/publish.htm
http://amfcorrea.ddns.net/pmAgi/
http://www.smartcon.rs/Products/SmartTrack/ClickOnce/publish.htm
http://deploy.leds.stage.ediscovery.state.co.us/
https://university.connectwise.com/install/psaserverChecker/
https://greytrix.com/freshbooks/freshbooks_peachtreeConnector.html
https://www.felixinstruments.com/Software/Beta/CI-690/
https://www.siscoconnect.com/admin/
https://www.fieldmaestroapp.com/apps/publish.htm
https://skyline.gs.washington.edu/software/Skyline-release_1.2/index.html
http://www.flimed.nl/ftp/it/publish.htm
http://www.redware.com/software/mscrmmconnector/clickonce/publish.htm
https://www.sportscrm.net/installv4/
https://www.ncaag.gov/nutrientmanagement/
https://www.cat-ing.de/jetcat-hexfiles/JetCatUpdaterV12.htm
http://annotationpro.org/download/
https://www.axxispetro.com/PrivateMarkets/
https://www.lars4d.com/translation/
http://www.proxlimiti.com/ce/connector/publish.htm
Identifying Abusable .NET Assemblies

PS C:\> .\AssemblyHunter.exe help

AssemblyHunter.exe

Run Type Options:
- path (ex: path=C:\Users) full path to search
- file (ex: file=C:\file.exe) check if a specific file is an assembly
- collection (ex: file=C:\files.txt) run checks against assemblies listed in a file
- services (ex: services=true) enumerates all services for assemblies
- tasks (ex: tasks=true) enumerates all scheduled tasks for assemblies
- autoruns (ex: autoruns=true) enumerates common autorun locations for assemblies

Optional args:
- recurse (ex: recurse=true) recurse the path given
- allpaths (ex: allpaths=true) recurses all directories, by default some directories with common Microsoft assemblies are skipped
- exeonly (ex: exeonly=true) look for exes only
- getarch (ex: getarch=true) get assembly architecture
- servicename (ex: servicename=true) check a specific service (needs services run)
- isservice (ex: isservice=true) check if an exe is a service executable
- getuac (ex: getuac=true) gets UAC settings of assembly
- getrefs (ex: getrefs=true) gets references for target assembly
- getasmid (ex: getasmid=true) gets internal assembly manifest identity
- getasmid (ex: getasmid=true) gets internal application manifest identity
- getappmanifest (ex: getappmanifest=true) gets internal application manifest
- getasmmanifest (ex: getasmmanifest=true) gets internal assembly manifest

* path, file, collection, services, tasks, or autoruns should indicate the type of search performed, all other options narrow down the search
Identifying Abusable .NET Assemblies

```xml
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0" xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
  <asmv3:application>
    <asmv3:windowsSettings>
      <dpiAwareness xmlns="http://schemas.microsoft.com/SMI/2016/windowsSettings">PerMonitorV2, PerMonitor</dpiAwareness>
      <longPathAware xmlns="http://schemas.microsoft.com/SMI/2016/windowsSettings">true</longPathAware>
    </asmv3:windowsSettings>
  </asmv3:application>
  <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
    <application>
      <supportedOS Id="{e2011457-1546-43c5-a5fe-088dee3d3f0}" />
      <supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a244a225f93a}" />
      <supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4e38}" />
      <supportedOS Id="{1f676c76-80e1-4239-95bb-83d0f6d6da78}" />
      <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd56a15a9a}" />
    </application>
  </compatibility>
</assembly>
```
PS C:\> .\AssemblyHunter.exe path=C:\ recurse=true noreqpriv=true exeonly=true signed=true quiet=true

[+] Found assembly: C:\Program Files\Microsoft SDKs\Azure\Emulator\devfabric\dfMonitor.exe
  [+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
  [+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
  [-] No Manifest

[+] Found assembly: C:\ProgramData\chocolatey\choco.exe
  [+] Cert Issuer Name: CN=Digicert SHA2 Assured ID Code Signing CA, OU=www.digicert.com, O=Digicert Inc, C=US
  [+] Cert Subject Name: CN="Chocolatey Software, Inc.", O="Chocolatey Software, Inc.", L=Topeka, S=Kansas, C=US
  [-] No Manifest
Not usable

Usable

PS C:\> ..\AssemblyHunter.exe path=C:\Windows\Microsoft.NET\Framework64\v4.0.30319\exonly=true getasmsid=true getappid=true getuac=true signed=true

[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AddInProcess.exe
[+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] UAC setting: asInvoker
[+] Assembly Manifest Identity: AddInProcess, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e889, processorArchitecture=MSIL
[-] No Application Manifest Identity

[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AddInProcess32.exe
[+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] UAC setting: asInvoker
[+] Assembly Manifest Identity: AddInProcess32, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e889, processorArchitecture=x86
[-] No Application Manifest Identity

[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AddInUtil.exe
[+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] UAC setting: asInvoker
[+] Assembly Manifest Identity: AddInUtil, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e889, processorArchitecture=MSIL
[-] No Application Manifest Identity

[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_compiler.exe
[+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] UAC setting: asInvoker
[+] Assembly Manifest Identity: aspnet_compiler, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=AMD64
[-] No Application Manifest Identity

[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\CasPol.exe
[+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
[+] UAC setting: asInvoker
[+] Assembly Manifest Identity: caspol, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=AMD64
[+] Application Manifest Identity: 1.0.0.0, MyApplication.app,
Detection Opportunities

- ClickOnce applications are not super common (validate with baseline)
  - Dfsvc.exe process spawning
    - Check child process
    - Potentially whitelist allowed ClickOnce applications
  - Dfsvc.exe child process with an unsigned module load
  - Dfshim.dll can also be used for launching an application

- Look into ETW monitoring for ClickOnce execution
  - Keep in mind ETW bypasses or <etwEnable>.NET config value

- Baseline installation directories
  - (%LOCALAPPDATA%\Apps\2.0\<string>)

- Uncommon (application) signature making internet connection

Prevention Opportunities

- Create trust prompts, can disable all ClickOnce from the internet while allowing other zones
  - Options: Enabled, AuthenticodeRequired, and Disabled
  - Zones: MyComputer, LocalIntranet, TrustedSites, Internet, UntrustedSites
  - To disable all from the internet:
    - `\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NETFramework\Security\TrustManager\PromptingLevel - Internet:Disabled`
- If an application control solution is deployed
  - Prevent unreputable DLLs from being loaded

https://superuser.com/questions/1252575/unable-to-install-clickonce-application-due-to-security-settings-windows-10

Prevention Opportunities

Application Install - Security Warning

Your administrator has blocked this application because it potentially poses a security risk to your computer.

**Name:**
Defcon30 Demo, Inc.

**From:** (Hover over the string below to see the full domain):
defcon30.azureedge.net

**Publisher:**
Unknown Publisher

Your security settings do not allow this application to be installed on your computer.

Close
Closing

- Huge shoutout to Lee Christensen (@tifkin_), exploration of this technique wouldn’t have been done with him
- Shoutout to Casey Smith (@subTee) for previous .NET research
- Shoutout to William Burke (@0xF4B0) for previous ClickOnce research
- Still a huge attack surface with .NET/ClickOnce overall
  - Plenty of areas of research and places to look into for further capability

Thank you

Questions?