AMSI: How Windows 10 Plans to Stop Script-Based Attacks and How Well It Does It

Nikhil Mittal
whoami

- Twitter - @nikhil_mitt
- Blog – http://labofapenetrationtester.com
- Github - https://github.com/samratashok/
- Creator of Kautilya and Nishang
- Penetration Tester and Trainer
- Associate Consultant with NoSoSecure
- Spoken/Trained at: Defcon, Blackhat, CanSecWest, Shakacon and more.
Outline

- Script based attacks
- Introduction to AMSI
- AMSI – Detection and Blocking capabilities
- Failed attempts to avoid detection
- Bypassing AMSI
- Conclusion
Script Based Attacks


Why? – Low rate of detection, very effective.

- Already present on targets.
- Used by system administrators.
- Provides access to various OS and Network components.
- PowerShell is future of Windows Remote Administration.
- Anti Virus vendors have only recently, 2013 onwards, started to flag PowerShell scripts.
Script Based Attacks

How? –

• Execute from disk
• Execute from memory – encodedcommand, downloadstring, reflection.

Detection is easy for scripts saved to disk.
How to stop execution from memory?
AntiMalware Scan Interface (AMSI)

According to Microsoft AMSI:

- Provides File, memory and stream scanning, content source URL/IP reputation checks, and other techniques.
- Can be integrated in any application.
- Includes additional calls for scripts that use obfuscation or layer dynamic code evaluation.
- As of now (21st July 2016), Windows Defender and AVG uses it.

AMSI Architecture

Win32 API Layer
- AMSI + AMSI+ + AMSI.dll
  - AMSIScanBuffer()
  - AMSIScanString()

COM API Layer
- AMSI.dll
  - AMSI5.dll (Antimalware / Scan)

AV Provider Layer
- Windows Defender Provider Class
- External AV Provider Class
- 3rd Party AV Provider Class

What makes AMSI effective?

AMSI tries to catch the scripts at the Scripting host level. It means:

- Input method (disk, memory, interactive) doesn’t matter.
- Use of System.Automation.dll (PowerShell scripts without powershell.exe) doesn’t help as well.
- Less help from obfuscation.

https://github.com/Ben0xA/nps
All demonstrations on 64-bit Windows 10 build 10586
Putting AMSI to test – Unusual storage

What if PowerShell scripts are loaded from unusual places like:

- WMI namespaces
- Registry Keys
- Event logs

Traditional (disk based) detection is unable to catch such scripts as the storage is rather unusual.
PowerShell code and scripts can be executed without using PowerShell.exe. Please see:
https://github.com/leechristensen/UnmanagedPowerShell
https://github.com/Ben0xA/nps
https://github.com/PowerShellEmpire/PowerTools/tree/master/PowerPick

Interesting methods to bypass Application whitelisting
http://subt0x10.blogspot.in/2016/04/bypass-application-whitelisting-script.html
http://subt0x10.blogspot.in/2015/08/application-whitelisting-bypasses-101.html
https://raw.githubusercontent.com/subTee/ApplicationWhitelistBypassTechniques/master/TheList.txt
DEMO – Putting AMSI to test – Unusual Execution
Is it all gloom and doom for Red Teams?

Bypass and/or avoid AMSI

• Use PowerShell version 2 (needs .Net 3.0 which is not present in a default Windows 10)
• Significantly change the signature of your scripts – limited effectiveness
• Disable AMSI
Bypass or avoid AMSI

Signature bypass

• Obfuscation
  • Not really hard to bypass AMSI using this.
    1. Remove help section
    2. Obfuscate function and variable names
    3. Encode parts of script
    4. Profit
  • Obfuscation functionality in ISESteroids Module – Fast and very effective at the time of writing.
Bypass or avoid AMSI

Signature bypass

```c
function
{
    [CmdletBinding()] Param($)
    if (null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } elseif ($null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } elseif ($null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } elseif ($null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } elseif ($null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } elseif ($null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } elseif ($null) {
        write-verbose $(("$\{\[\{\}\}\}\})
        $\{\{\{\}\}\}\} = $\{\{\{\}\}\}\} -join "."
    } 
}```
Unload AMSI

- `Set-MpPreference`
- Unload from current process – Matt’s method
- `P0wnedshell`
Bypass or avoid AMSI

Set-MpPreference

- Handy PowerShell cmdlet to play with Windows Defender.

Set-MpPreference – DisableRealtimeMonitoring $True
Bypass or avoid AMSI

Set-MpPreference

- Shows a notification to the user
- Needs elevated privileges (not much headache in a post-exploitation scenario)
- Event ID 5001 (Microsoft-Windows-Windows Defender/Operational) - Windows Defender Real-Time Protection was disabled.

<table>
<thead>
<tr>
<th>Event 5001, Windows Defender</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
</tr>
<tr>
<td>Windows Defender Real-time Protection scanning for malware and other potentially unwanted software was disabled.</td>
</tr>
<tr>
<td><strong>Details</strong></td>
</tr>
<tr>
<td>Log Name:</td>
</tr>
</tbody>
</table>
Bypass or avoid AMSI

Set-MpPreference

• To target AMSI:
  Set-MpPreference -DisableIOAVProtection $True
Bypass or avoid AMSI

Set-MpPreference

- Doesn’t show any notification to the user
- Needs elevated privileges
- Event ID 5004 (Microsoft-Windows-Windows Defender/Operational) - Windows Defender Real-Time Protection feature (IE Downloads and Outlook Express attachments) configuration has changed.

<table>
<thead>
<tr>
<th>Event 5004, Windows Defender</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
</tr>
<tr>
<td>Windows Defender Real-time Protection feature configuration has changed.</td>
</tr>
<tr>
<td>Feature: IE Downloads and Outlook Express Attachments</td>
</tr>
<tr>
<td>Configuration: 0</td>
</tr>
</tbody>
</table>

Log Name: Microsoft-Windows-Windows Defender/Operational
Bypass or avoid AMSI

Unloading AMSI

- A one line AMSI bypass from Matt Graeber (screenshot) (@mattifestation)

  `[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,Static').SetValue(null,$true)

  - Unload AMSI from current process.
  - No need of elevated privileges
  - Event ID 4104 (Microsoft-Windows-PowerShell/Operational) – Suspicious script block logging
  - Bypass the automatic logging?

Source: https://twitter.com/mattifestation/status/735261176745988096
Bypass or avoid AMSI

Unloading AMSI

• A method discovered by Cornelis de Plaa (@Cneelis)
  • Implemented in p0wnedshell
    (https://github.com/Cn33liz/p0wnedShell)
  • Drop amsi.dll in the current working directory while loading
    the p0wnedshell runspace. The dll is loaded by the
    runspace and exits immediately to unload AMSI.
  • Event ID 4104 (Microsoft-Windows-
    PowerShell/Operational) – Suspicious script block logging
    (due to successful loading of scripts in memory)
  • Bypass the automatic logging?

Source: http://cn33liz.blogspot.com/2016/05/bypassing-amsi-using-powershell-5-
dll.html
Demo – Bypassing AMSI using a Client Side Attack

Image source: http://goo.gl/CmZbml
WMF5 Auto Logging

- Hard to execute a PowerShell attack without generating logs.
- Apparently, Obfuscation boils down to bypass the logging.
- Who is monitoring the logs?
Black Hat Sound Bytes

• AMSI is a big step forward towards blocking script based attacks in Windows.
• It is possible to avoid AMSI using already known methods and techniques.
• AMSI is useful only when used with other security methods. Monitor the logs!
Thank You

• Questions?
• Please provide feedback.
• Follow me @nikhil_mitt
• nikhil.uitrgpv@gmail.com
• http://www.labofapenetrationtester.com/2016/08/amsi.html
• https://github.com/samratashok/AMSI