The Tale of SettingContent-ms Files

By Matt Nelson

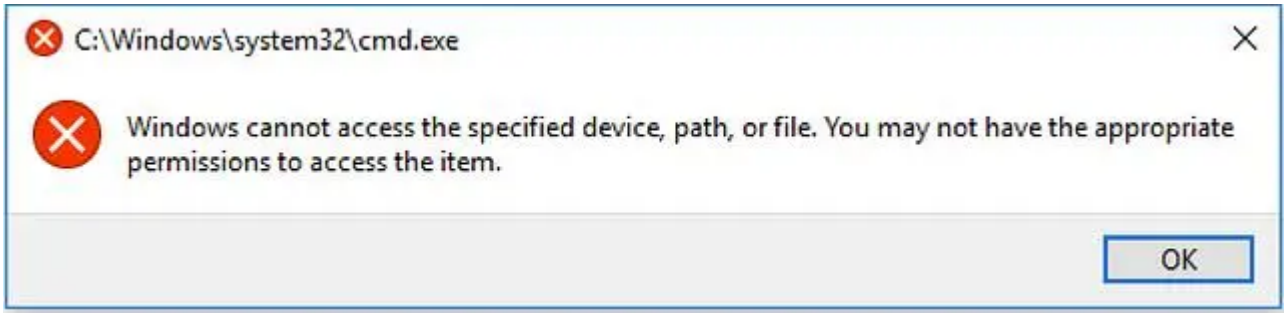
Published: 2018-06-11 · Archived: 2026-04-06 01:04:29 UTC

As an attacker, initial access can prove to be quite the challenge against a hardened target. When selecting a payload for initial access, an attacker has to select a file format that allows arbitrary code execution or shell command execution with minimal user interaction. These file formats can be sparse, which is why attackers have relied on file types like .HTAs, Office macros, .VBS, .JS, etc. There are obviously a finite number of built-in file extensions on Windows, and as defenses improve, the number of effective payloads continues to shrink.

Additionally, an attacker has to get that file to the end user in a way that will result in execution. Again, options for this can be limited as directly linking to payloads or attaching them to emails tend to get blocked by antivirus or browser protections. This is why attackers have resorted to Object Linking and Embedding (OLE), ZIP files, etc. To combat the file delivery vector, Office 2016 introduced blocking all of the “dangerous” file formats from being embedded via OLE by default. This reduces the effectiveness of one of the most relied upon payload delivery methods. When trying to activate a blocked file extension, Office will throw an error and prevent execution:

Office 2016 blocking one of many “dangerous” file extensions

In addition to OLE blocking, Microsoft also introduced Attack Surface Reduction (ASR) rules into Windows 10 (which requires Windows Defender AV as a dependency). The purpose of these rules are to reduce the functionality that an attacker can abuse or exploit to obtain code-execution on the system. One of the most touted and effective ASR rules is “Block Office applications from creating child processes”. This rule will block any attempt to spawn a process as a child of an Office application:



Attack Surface Reduction Rules blocking Office child processes

When you combine OLE blocking and ASR together, the options to execute code on an endpoint from the internet become a little more limited. Most useful file types can't be delivered via OLE per the new block in Office 2016, and ASR's child process creation rule prevents any instances of a child process spawning under an Office application.

How could we circumvent these controls? I first decided to tackle the file format problem. I spent a lot of hours going through the registry looking for new file formats that might allow execution. Most of these formats can be found in the root of the HKCR:\ registry hive. The process involved pulling all of the registered file formats out and then looking at them to see if the format itself allowed for anything interesting.

After countless hours reading file specifications, I stumbled across the ".SettingContent-ms" file type. This format was introduced in Windows 10 and allows a user to create "shortcuts" to various Windows 10 setting pages. These files are simply XML and contain paths to various Windows 10 settings binaries. An example .SettingContent-ms file will look like this:

Normal SettingContent-ms file for the Windows Control Panel

All this file does is open the Control Panel for the user. The interesting aspect of this file is the **<DeepLink>** element in the schema. This element takes any binary with parameters and executes it. What happens if we simply substitute "control.exe" to something like "cmd.exe /c calc.exe"?

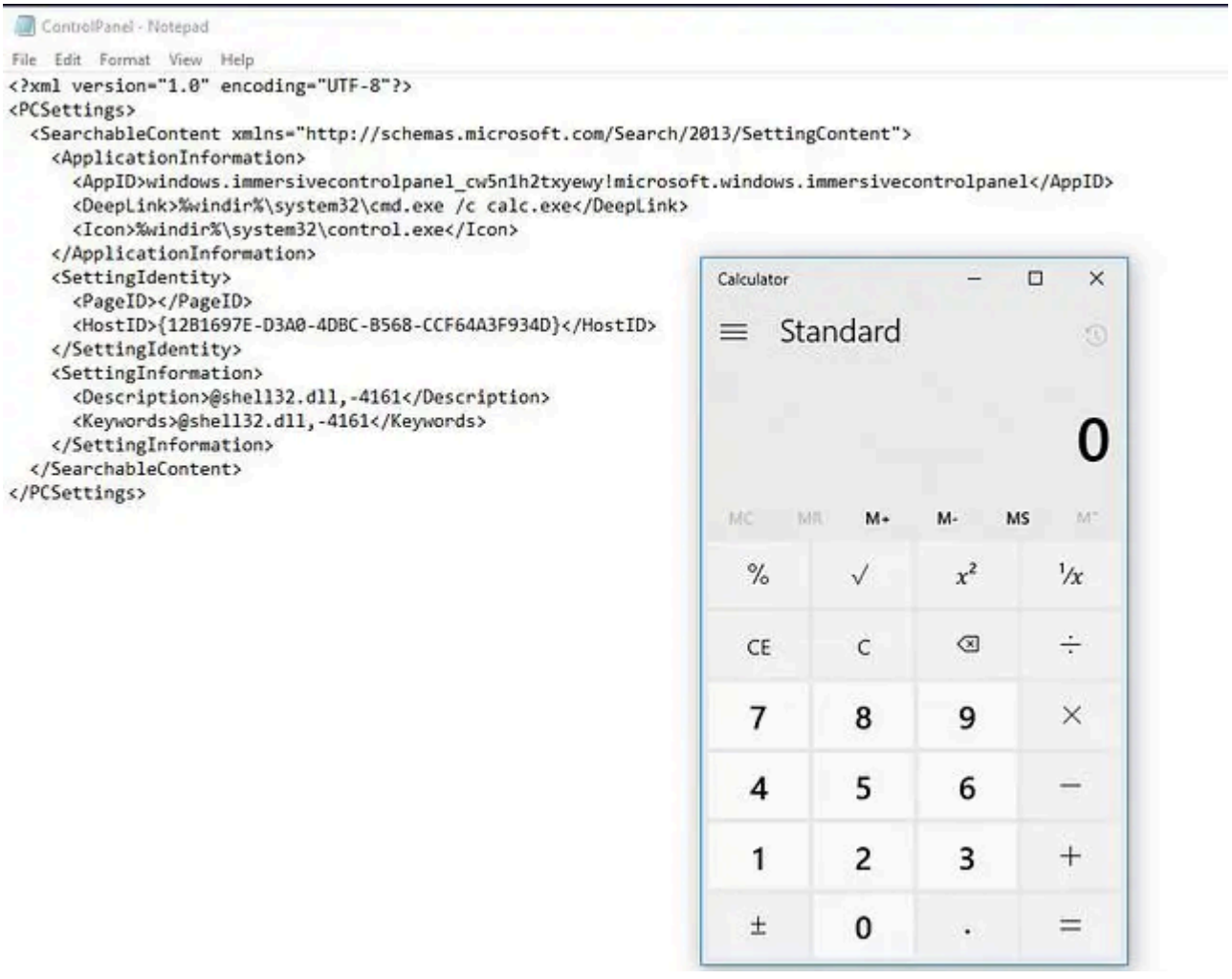
ControlPanel - Notepad

File Edit Format View Help

```
<?xml version="1.0" encoding="UTF-8"?>
<PCSettings>
  <SearchableContent xmlns="http://schemas.microsoft.com/Search/2013/SettingContent">
    <ApplicationInformation>
      <AppID>windows.immersivecontrolpanel_cw5n1h2txyewy!microsoft.windows.immersivecontrolpanel</AppID>
      <DeepLink>%windir%\system32\cmd.exe /c calc.exe</DeepLink>
      <Icon>%windir%\system32\control.exe</Icon>
    </ApplicationInformation>
    <SettingIdentity>
      <PageID></PageID>
      <HostID>{12B1697E-D3A0-4DBC-B568-CCF64A3F934D}</HostID>
    </SettingIdentity>
    <SettingInformation>
      <Description>@shell32.dll,-4161</Description>
      <Keywords>@shell32.dll,-4161</Keywords>
    </SettingInformation>
  </SearchableContent>
</PCSettings>
```

Replacing the <DeepLink> element with an attacker controlled value

Then if we double-click the file:



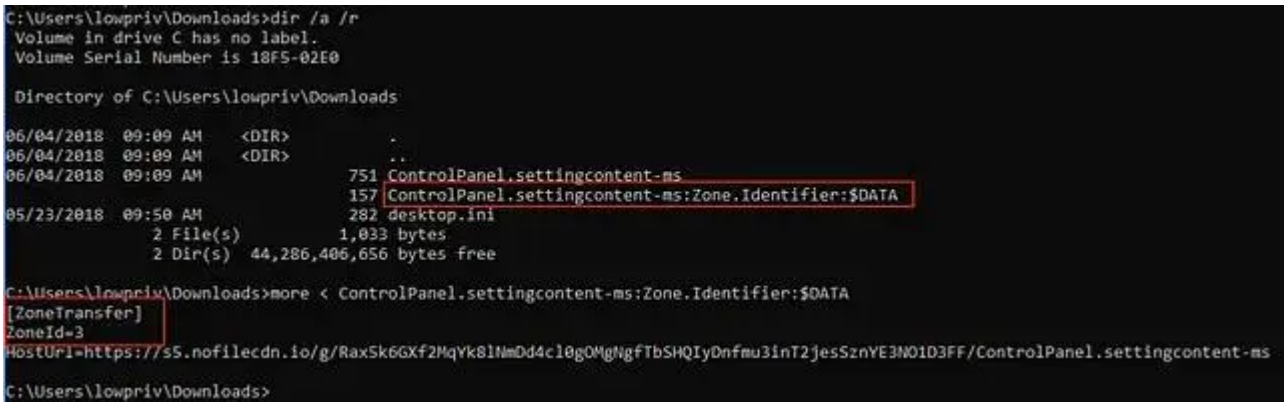
Execution of the modified SettingContent-ms file

What is interesting is that when double-clicking the file, there is no “open” prompt. Windows goes straight to executing the command.

Great! So we have a file format that allows shell command execution via a file open. This solves the “what file format to use” problem of initial access. Now, how can we deliver it? My next thought was to see what happens if this file comes straight from the internet via a link.

Execution of a SettingContent-ms file directly from the internet via a hyperlink

Yikes!! When this file comes straight from the internet, it executes as soon as the user clicks “open”. Looking at the streams of the file, you will notice that it does indeed grab Mark-Of-The-Web:

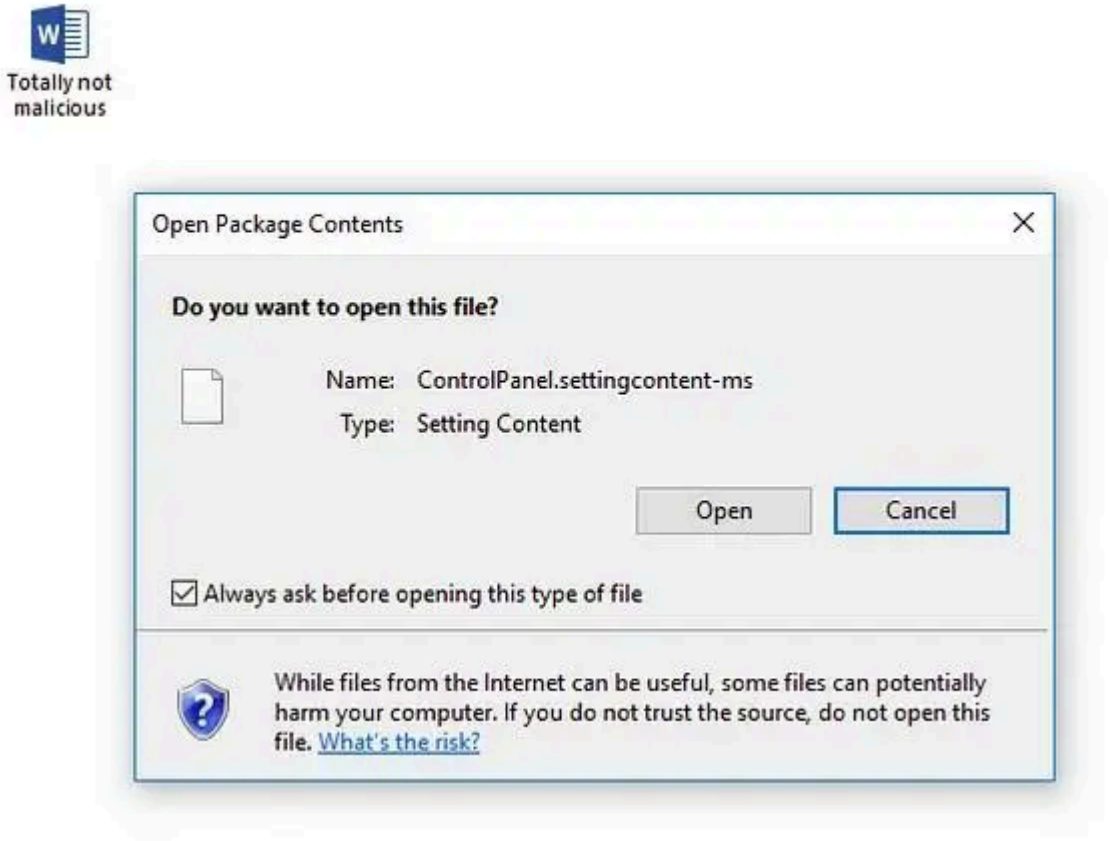


The MOTW ADS being applied to the downloaded

When looking up the ZoneIds online, “3” equals “URLZONE_INTERNET”. For one reason or another, the file still executes without any notification or warning to the user.

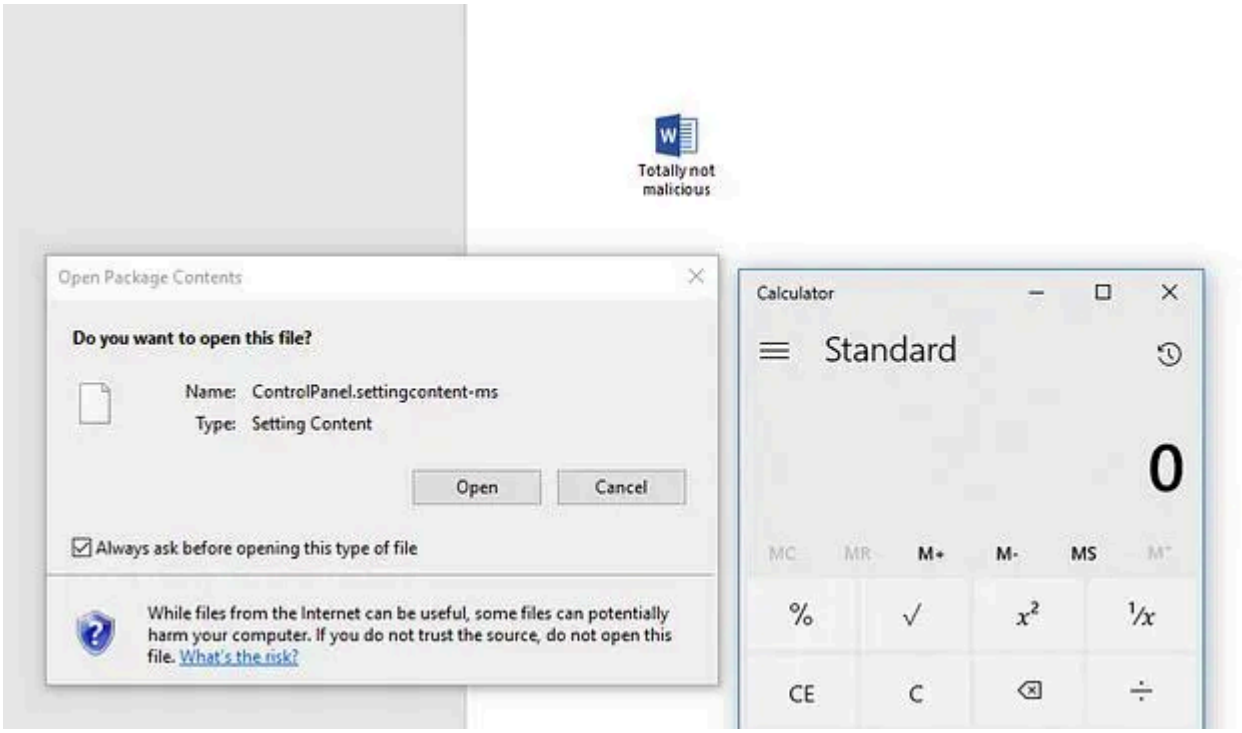
So, we now have a file type that allows arbitrary shell command execution and displays zero warnings or dialogues to the user. When trying to get initial access, going across a target’s perimeter with an unusual file type can be risky. Ideally, this file would be placed in a container of a more common file type, such as an Office document.

As mentioned before, Office 2016 blocks a preset list of “known bad” file types when embedded with Object Linking and Embedding. The SettingContent-ms file format, however, is not included in that list:



Office 2016 OLE block list not blocking the SettingContent-ms file

At this point, we can evade the Office 2016 OLE file extension block by embedding a malicious .SettingContent-ms file via OLE:



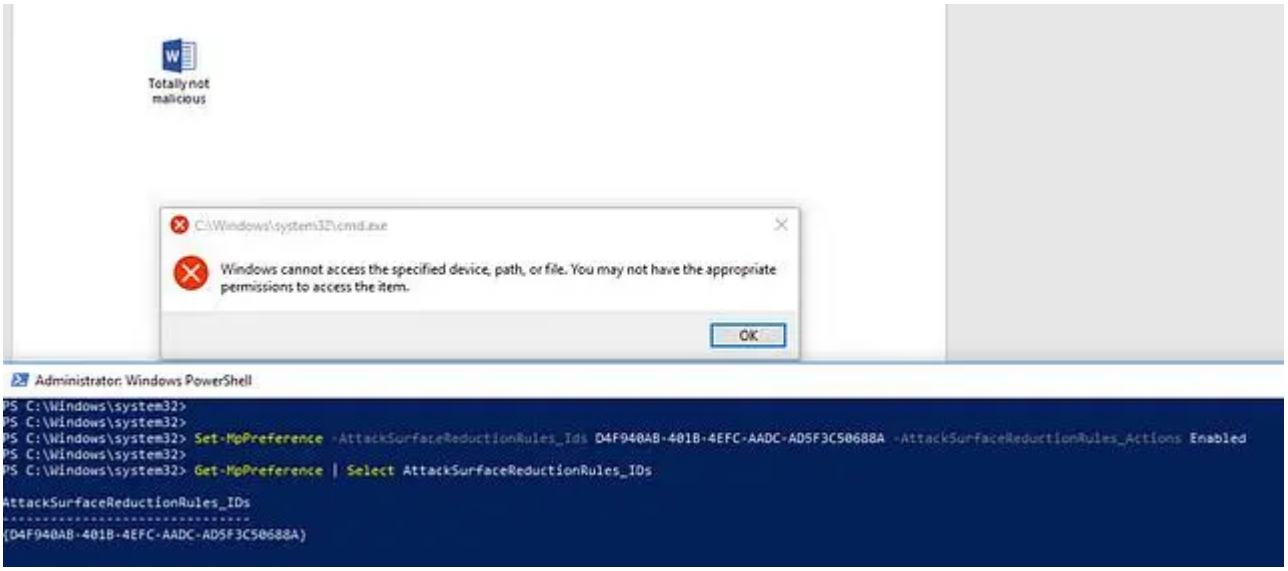
Office 2016 OLE block list not blocking the SettingContent-ms file

When a document comes from the internet with a .SettingContent-ms file embedded in it, the only thing the user sees is the “Open Package Contents” prompt. Clicking “Open” will result in execution. If an environment doesn’t have any Attack Surface Reduction rules enabled, this is all an attacker needs to execute code on the endpoint. I was curious, so I poked at how this holds up with ASR’s child process creation rules enabled. **It is also worth noting that at of the time of this post, ASR rules do not appear to work on Office if Office was installed from the Windows Store.**

Enabling these rules is pretty simple, and can be done with one PowerShell command: “Set-MpPreference -AttackSurfaceReductionRules_Ids <rule ID> -AttackSurfaceReductionRules_Actions Enabled”

The <rule ID> parameter is the GUID of the rule you want to enabled. You can find the GUID for each ASR rule documented here. For this test, I want to enable the Child Process Creation rule, which is GUID D4F940AB-401B-4EFC-AADC-AD5F3C50688A.

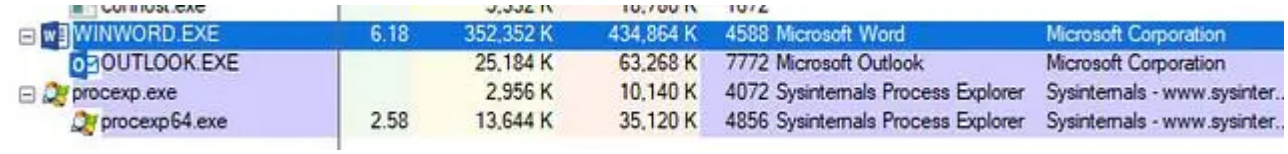
After enabling the rule, the attack no longer works:



Attack Surface Reduction rule blocking SettingContent-ms execution

Since the rule is designed to block child processes from being spawned from an Office application, our payload executed, but the rule blocked the command. This got me thinking into how ASR achieves that without breaking some functionality. I first started to just test random binaries in random paths to see if ASR was blocking based on the image path. This was quite time consuming and didn't get me very far.

I ended up taking a step back and thinking about what parts of Office have to work. After running ProcMon and looking at Process Explorer for a little bit while clicking around in Word, I noticed that there were still child processes being spawned by Word.



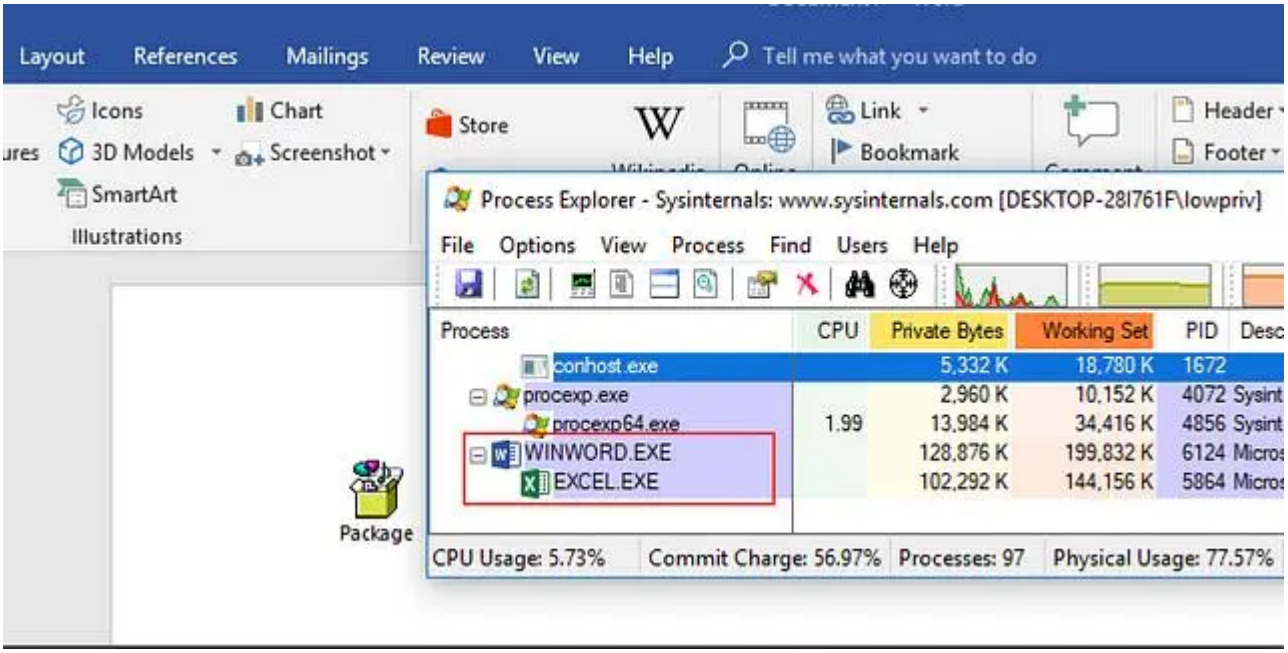
Valid child process creation of Word

This makes sense, as Office needs to use features that rely on other programs. I thought that maybe the ASR rule blocks child processes based on image path, but images in the Office path are allowed to be spawned when features are activated.

To test this theory, I changed my .SettingContent-ms file to the path of "Excel.exe":

Changing the <DeepLink> element to Excel.exe

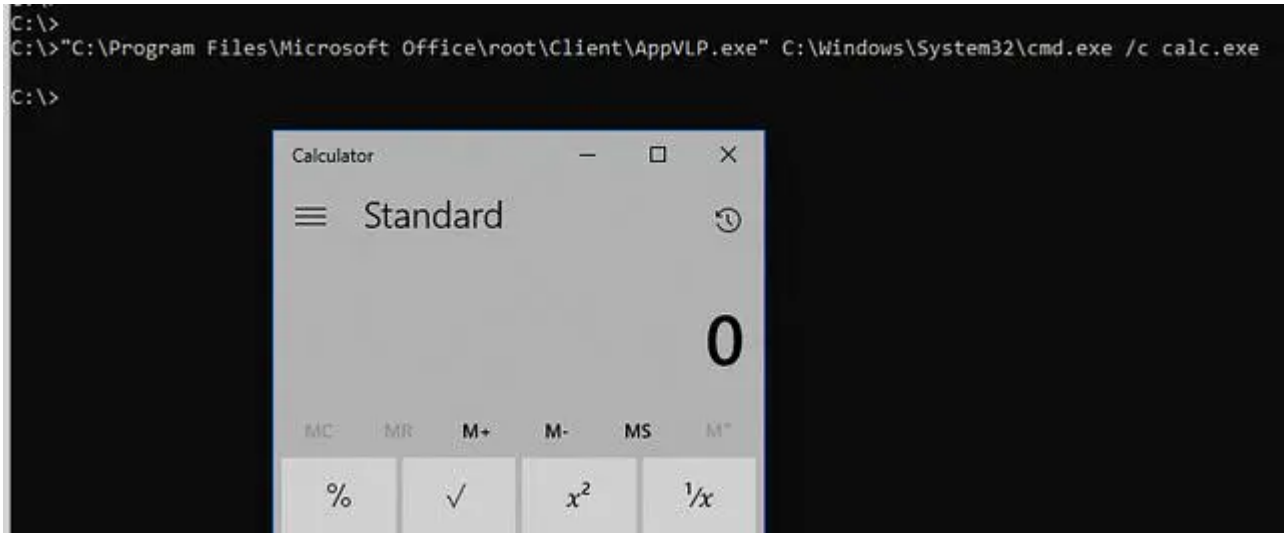
The next step is to embed this new file into a Word document and see if ASR blocks “Excel.exe” from being spawned.



Excel spawning under winword via modified SettingContent-ms file

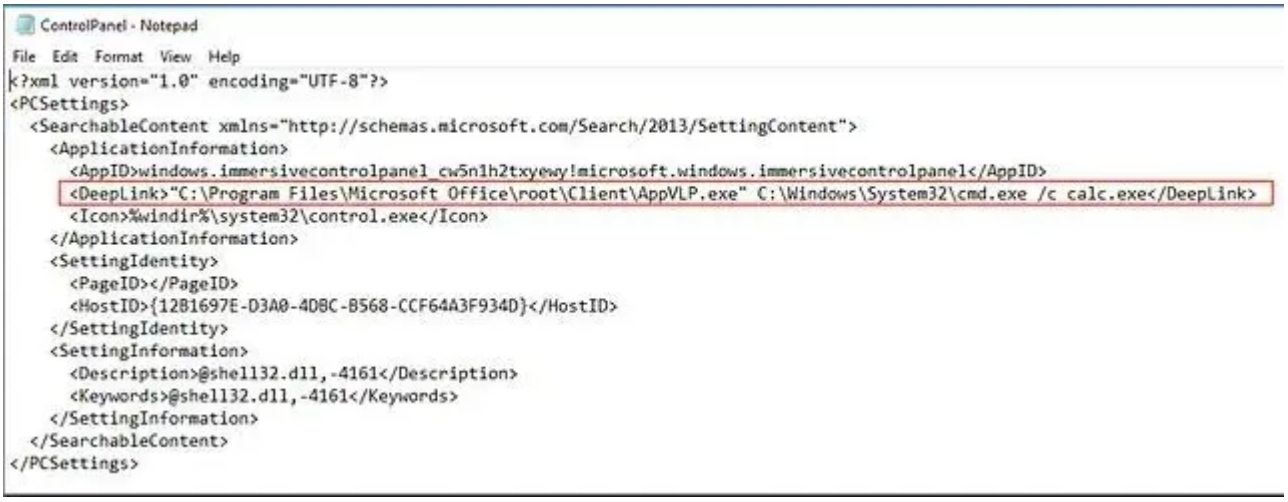
Interestingly enough, ASR allows Excel to start. So the child process creation ASR rule appears to be making decisions based on whitelisted paths.

This sent me down a long, long path of trying to find a binary I could use that existed in the path “C:\Program Files\Microsoft Office”. After going through a few of them and passing “C:\Windows\System32\cmd.exe” to them as a parameter on the command line, one of them kicked:



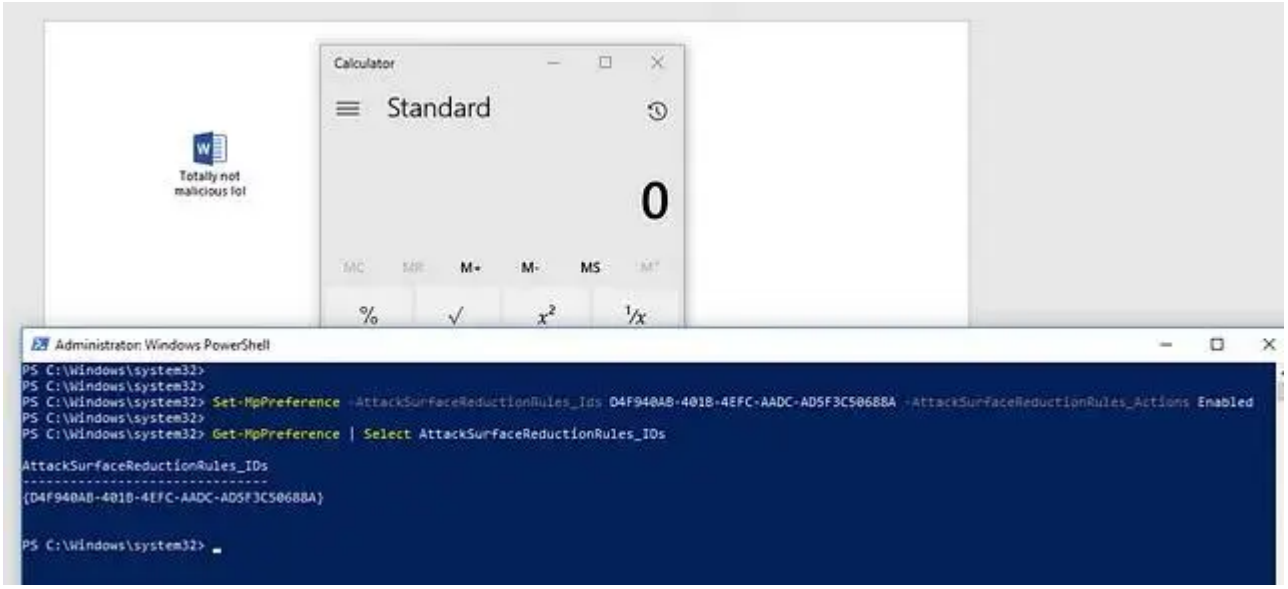
Execution of commands via ASR whitelisted “AppVLP.exe”

Perfect! We are able to abuse “AppVLP” to execute shell commands. Normally, this binary is used for Application Virtualization, but we can use it as an abuse binary to circumvent the ASR file path rule. To test this full chain, I updated my .SettingContent-ms file to look like this:



Final OLE/ASR evasion payload

Now, the file just needs embedded into an Office document and executed:



Evading the OLE block with .SettingContent-ms and ASR with AppVLP.exe

As you can see, with Office 2016's OLE block rule and ASR's Child Process Creation rule enabled, .SettingContent-ms files combined with "AppVLP.exe" in the Office folder allow us to circumvent these controls and execute arbitrary commands.

While Office documents are often marked with MOTW and are opened in the Protected View Sandbox, there are file formats that allow OLE and aren't triggered by the Protected View sandbox. You can find more on that here.

Wrap Up:

After looking into ASR and the new file formats in Windows 10, I realized that it is important to try and audit new binaries and file types that get added in each release of Windows. In this case, the .SettingContent-ms extension can allow an attacker to run arbitrary commands on the latest version of Windows while evading ASR and Office 2016 OLE blocks. Additionally, the file type appears to give execution (even from the internet) immediately after it's opened, despite having the MOTW applied.

Defenses:

Great, so what can you do about it? Ultimately, a .SettingContent-ms file should not be executing anywhere outside of the "C:\Windows\ImmersiveControlPanel" path. Additionally, since the file format only allows for executing shell commands, anything being run through that file is subject to command line logging.

It is also a good idea to always monitor child process creations from Office applications. There are a few applications that should be spawning under the Office applications, so monitoring for outliers can be useful. One tool that can accomplish this is Sysmon.

Another option is to neuter the file format by killing its handler. I have **NOT** tested this extensively and can offer no guarantee that something within Windows won't break by doing this. For those who want to play around with the implications of killing the handler to the .SettingContent-ms file format, you can set the "DelegateExecute"

key in “HKCR:\SettingContent\Shell\Open\Command” to be empty. Again, this might break functionality on the OS, so proceed with caution.

You can find a PoC SettingContent-ms file here: https://gist.github.com/enigma0x3/b948b81717fd6b72e0a4baca033e07f8

Reporting timeline:

As committed as SpecterOps is to transparency (https://posts.specterops.io/a-push-toward-transparency-c385a0dd1e34), we acknowledge the rate at which attackers adopt new offensive techniques once they are made public. This is why prior to publicization of a new offensive technique, we regularly inform the respective vendor of the issue, supply ample time to mitigate the issue, and notify select, trusted vendors in order to ensure detections can be delivered to their customers as quickly as possible.

2/16/2018: Report sent MSRC

2/16/2018: MSRC acknowledged the report, case number assigned

3/2/2018: MSRC confirmed that they could reproduce the issue

4/24/2018: Requested status update

4/25/2018: MSRC informed me of a case handler change. An update was requested from the engineering team and would be relayed to me ASAP

6/1/2018: Requested another update from MSRC

6/4/2018: MSRC responded with a note that the severity of the issue is below the bar for servicing and that the case will be closed.

6/11/2018: Report published

[UPDATE] 8/14/2018: MSRC fixed the issue CVE-2018–8414 (https://portal.msrc.microsoft.com/en-us/security-guidance/advisory/CVE-2018-8414)

Cheers,

Matt N.
