QAKBOT BB Configuration and C2 IPs List

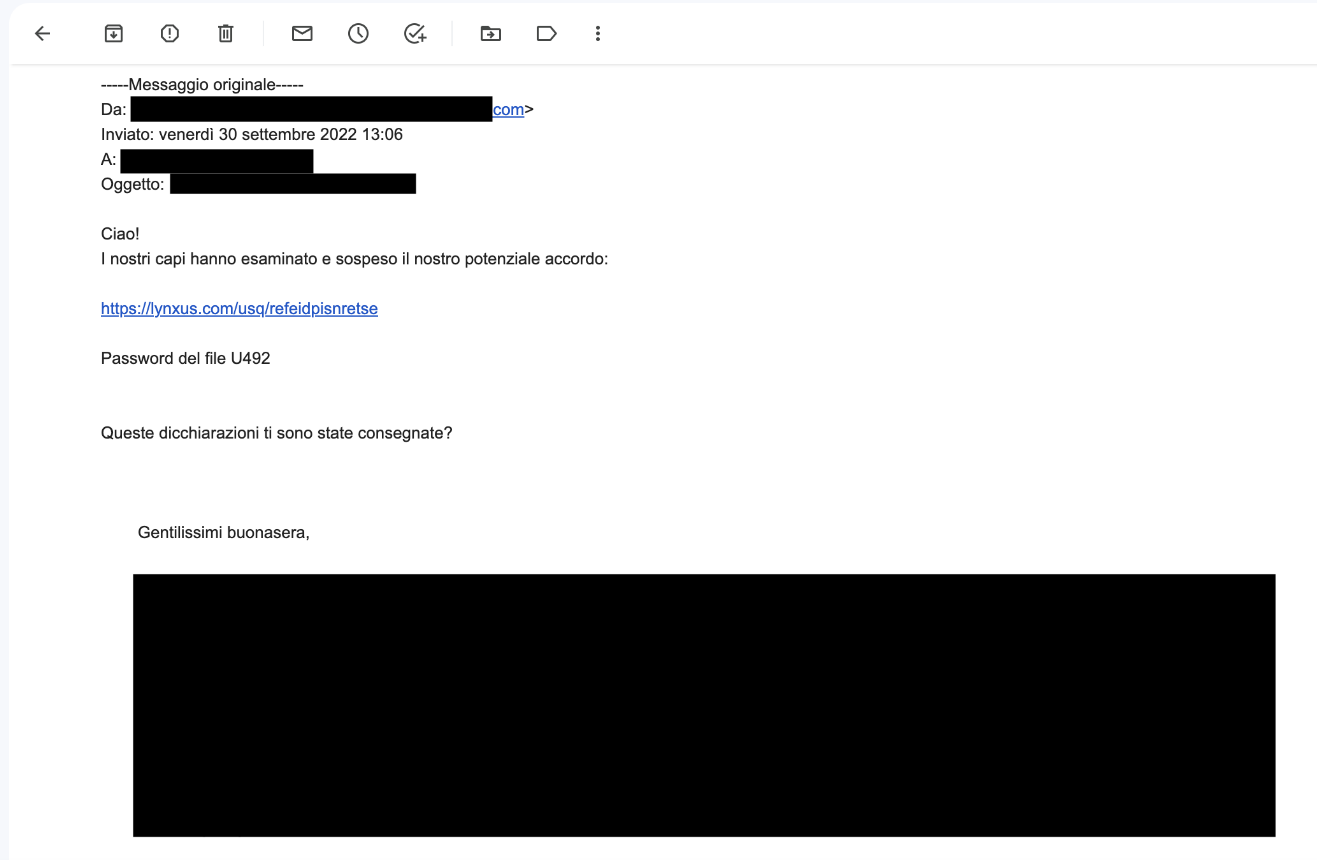
syrion.me/malware/qakbot-bb-extractor/

October 13, 2022

October 13, 2022 4 minute read

This is my first malware blog post, hope it will be useful to someone, I'll not go deeper in the malware details because there are plenty of detailed reports related to **QAKBOT**. I'll describe how the malware changed its resource decryption mechanism and report some IoCs.

On September 30, 2022 a friend of mine received a phishing email pretending to be sent by one of his customers, the email contained an URL, a password and a legit old message.



-----Messaggio originale-----
Da: [redacted]com>
Inviato: venerdì 30 settembre 2022 13:06
A: [redacted]
Oggetto: [redacted]

Ciao!
I nostri capi hanno esaminato e sospeso il nostro potenziale accordo:

https://lynxus.com/usq/refeidpisnretse

Password del file U492

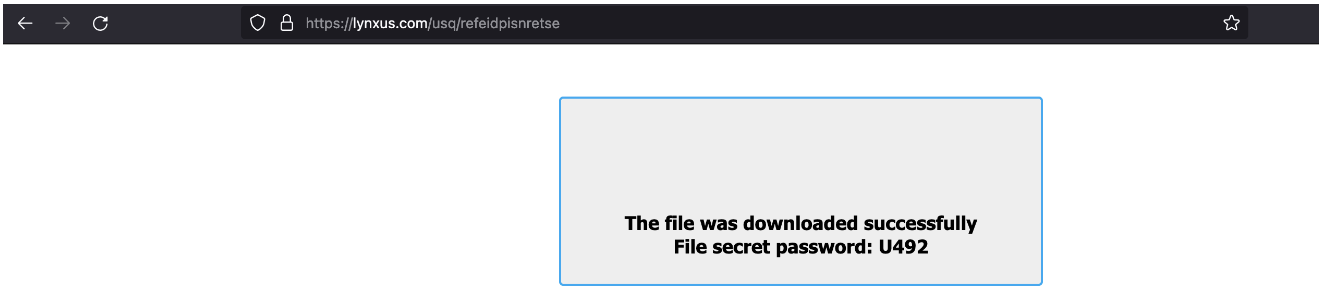
Queste dicchiarazioni ti sono state consegnate?

Gentilissimi buonasera,

[redacted]

Figure 1 - Phishing Email

By visiting the URL **https://lynxus[.]com/usq/refeidpisnretse** with a user agent related to Windows, a working zip named **Card654141047.zip** is provided, if the user agent is not "ok" the server responses with a fake zip file that doesn't work.



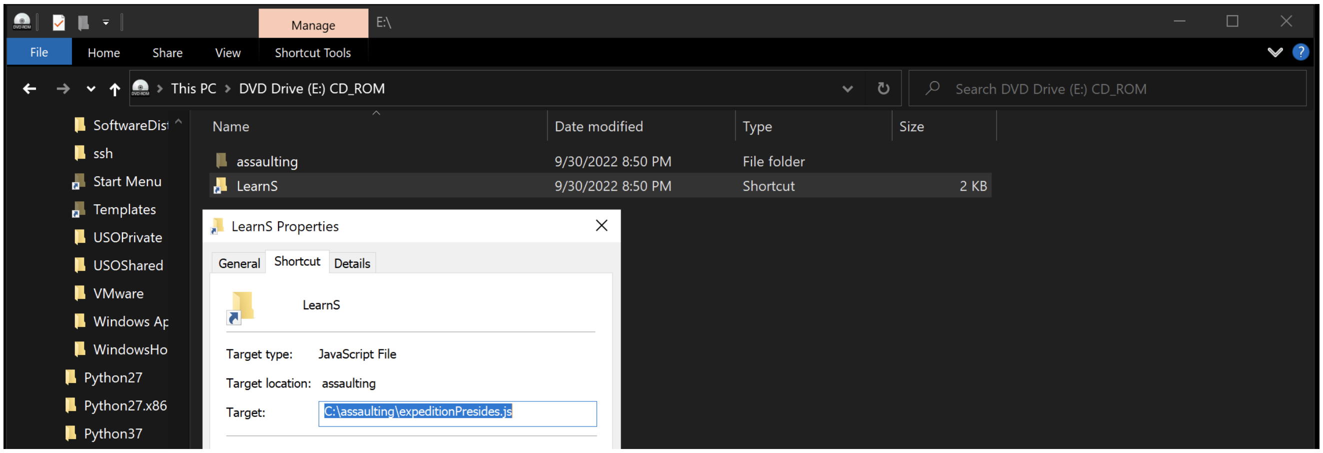
https://lynxus.com/usq/refeidpisnretse

The file was downloaded successfully
File secret password: U492

Figure 2 - Malcious URL message containing the zip password

Using the provided password “**U492**”, it is possible to extract an **ISO file** from the zip. The ISO file contains a **LNK** file and a **hidden folder** with the following files:

- expeditionPresides.js
- redressingLamentations.cmd
- regressing.txt
- rougher.gif
- tiddler.dat



Name	Date modified	Type	Size
assaulting	9/30/2022 8:50 PM	File folder	
LearnS	9/30/2022 8:50 PM	Shortcut	2 KB

LearnS Properties

General | Shortcut | Details

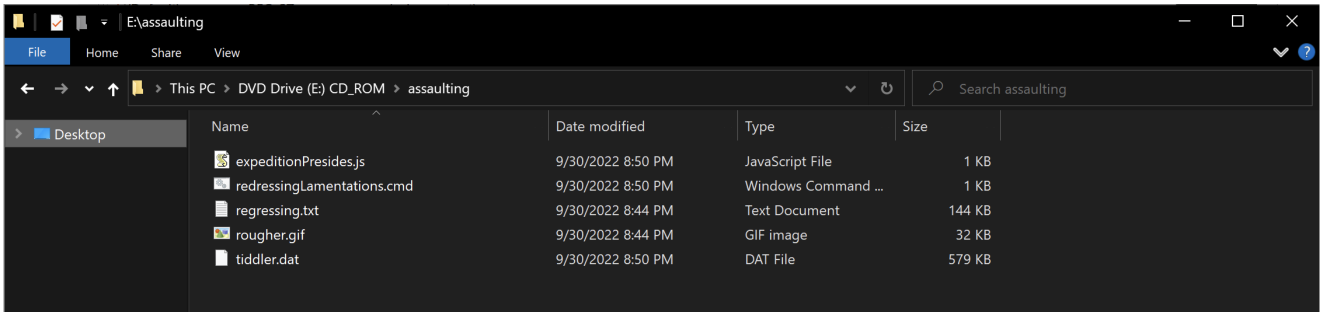
LearnS

Target type: JavaScript File

Target location: assaulting

Target: C:\assaulting\expeditionPresides.js

Figure 3 - Lnk File and hidden folder



Name	Date modified	Type	Size
expeditionPresides.js	9/30/2022 8:50 PM	JavaScript File	1 KB
redressingLamentations.cmd	9/30/2022 8:50 PM	Windows Command ...	1 KB
regressing.txt	9/30/2022 8:44 PM	Text Document	144 KB
rougher.gif	9/30/2022 8:44 PM	GIF image	32 KB
tiddler.dat	9/30/2022 8:50 PM	DAT File	579 KB

Figure 4 - Hidden folder content

The LNK file is a link to **expeditionPresides.js**, it contains the following JScript:

```
// observablyCleaned
var undisruptedPuzzles = "rund DllRegis";

// ShellExecute
var bridgeheadsLibels = new
ActiveXObject("shell.application").shellexecute("assaulting\\redressingLamentations.cm
 undisruptedPuzzles, "", "open", 0);
```

it runs **redressingLamentations.cmd** by proving two parameters "**rund DllRegis**".
Following the content of **redressingLamentations.cmd**.

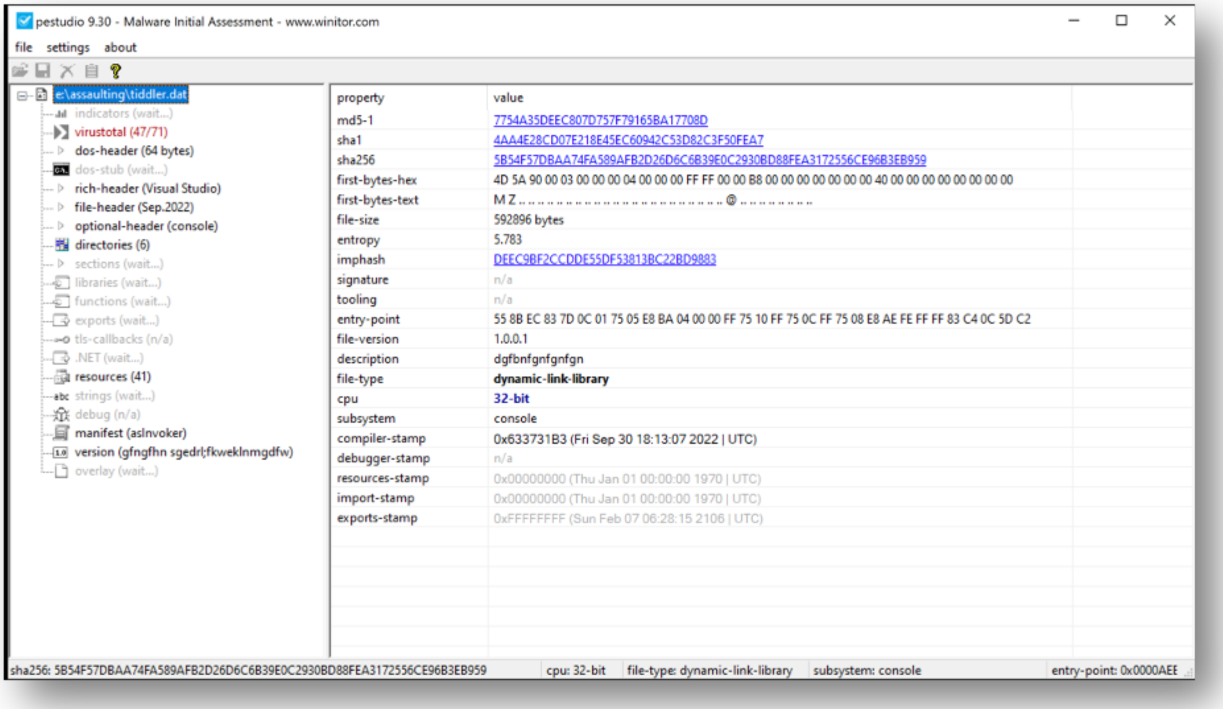
```
@echo off

set a=ll
set e=32

:: tankageLicentiously
%1%a%%e% assaulting\tiddler.dat,%2terServer

exit
```

It uses **rundll32** in order to execute the **DllRegisterServer** export function from **tiddler.dat**, following some details of the DLL.


pestudio 9.30 - Malware Initial Assessment - www.winitor.com
file
settings
about

e:\assaulting\tiddler.dat
indicators (wait...)
virustotal (47/71)
dos-header (64 bytes)
dos-stub (wait...)
rich-header (Visual Studio)
file-header (Sep.2022)
optional-header (console)
directories (6)
sections (wait...)
libraries (wait...)
functions (wait...)
exports (wait...)
tls-callbacks (n/a)
.NET (wait...)
resources (41)
strings (wait...)
debug (n/a)
manifest (asInvoker)
version (gfngfhn sgedrl;fkweklnmgdfw)
overlay (wait...)

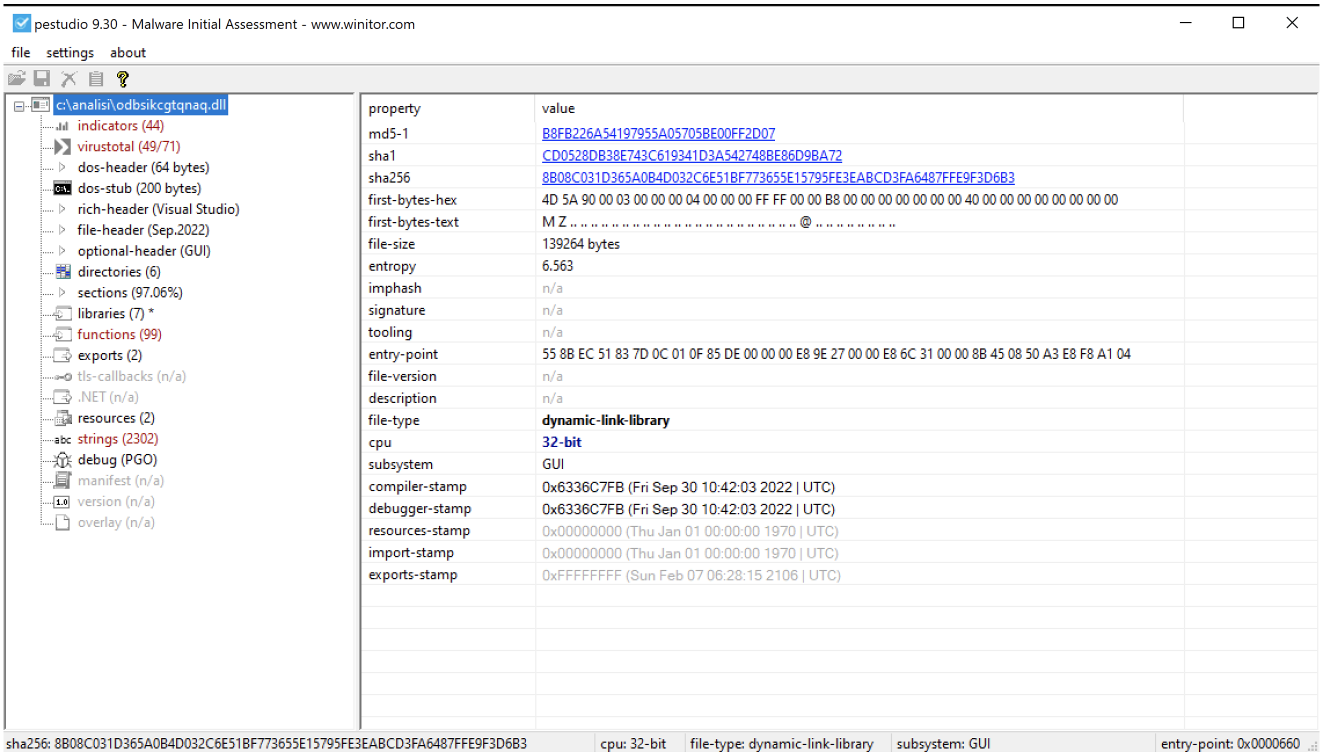
property	value
md5-1	7754A35DEEC807D757F79165BA17708D
sha1	4AA4E28CD07E218E45EC60942C53D82C3F50FEA7
sha256	5B54F57DBAA74FA589AFB2D26D6C6B39E0C2930BD88FEA3172556CE96B3EB959
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	592896 bytes
entropy	5.783
imphash	DEEC9BF2CCDDE55DF53813BC22BD9883
signature	n/a
tooling	n/a
entry-point	55 8B EC 83 7D 0C 01 75 05 E8 BA 04 00 00 FF 75 10 FF 75 0C FF 75 08 E8 AE FE FF FF 83 C4 0C 5D C2
file-version	1.0.0.1
description	dgfbnfgnfgnfgn
file-type	**dynamic-link-library**
cpu	**32-bit**
subsystem	console
compiler-stamp	0x633731B3 (Fri Sep 30 18:13:07 2022 | UTC)
debugger-stamp	n/a
resources-stamp	0x00000000 (Thu Jan 01 00:00:00 1970 | UTC)
import-stamp	0x00000000 (Thu Jan 01 00:00:00 1970 | UTC)
exports-stamp	0xFFFFFFFF (Sun Feb 07 06:28:15 2106 | UTC)

sha256: 5B54F57DBAA74FA589AFB2D26D6C6B39E0C2930BD88FEA3172556CE96B3EB959 cpu: 32-bit file-type: dynamic-link-library subsystem: console entry-point: 0x0000AEE

Figure 5 - tiddler.dat details

Tiddler.dat is the first stage DLL used to extract the unpacked version of the malware, by setting a breakpoint on **NtAllocateVirtualMemory** it's easy to find the unpacked version, I'll not describe how to get it.

After unpacking the DLL, we can analyse it, the details are in the image below.


pestudio 9.30 - Malware Initial Assessment - www.winitor.com
file
settings
about

c:\analisi\odbsikcgtqnaq.dll
- indicators (44)
- virustotal (49/71)
- dos-header (64 bytes)
- dos-stub (200 bytes)
- rich-header (Visual Studio)
- file-header (Sep.2022)
- optional-header (GUI)
- directories (6)
- sections (97.06%)
- libraries (7) *
- functions (99)
- exports (2)
- tls-callbacks (n/a)
- .NET (n/a)
- resources (2)
- strings (2302)
- debug (PGO)
- manifest (n/a)
- version (n/a)
- overlay (n/a)

property	value
md5-1	B8FB226A54197955A05705BE00FF2D07
sha1	CD0528DB38E743C619341D3A542748BE86D9BA72
sha256	8B08C031D365A0B4D032C6E51BF773655E15795FE3EABCD3FA6487FFE9F3D6B3
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	139264 bytes
entropy	6.563
imphash	n/a
signature	n/a
tooling	n/a
entry-point	55 8B EC 51 83 7D 0C 01 0F 85 DE 00 00 00 E8 9E 27 00 00 E8 6C 31 00 00 8B 45 08 50 A3 E8 F8 A1 04
file-version	n/a
description	n/a
file-type	dynamic-link-library
cpu	32-bit
subsystem	GUI
compiler-stamp	0x6336C7FB (Fri Sep 30 10:42:03 2022 | UTC)
debugger-stamp	0x6336C7FB (Fri Sep 30 10:42:03 2022 | UTC)
resources-stamp	0x00000000 (Thu Jan 01 00:00:00 1970 | UTC)
import-stamp	0x00000000 (Thu Jan 01 00:00:00 1970 | UTC)
exports-stamp	0xFFFFFFFF (Sun Feb 07 06:28:15 2106 | UTC)

sha256: 8B08C031D365A0B4D032C6E51BF773655E15795FE3EABCD3FA6487FFE9F3D6B3 cpu: 32-bit file-type: dynamic-link-library subsystem: GUI entry-point: 0x0000660

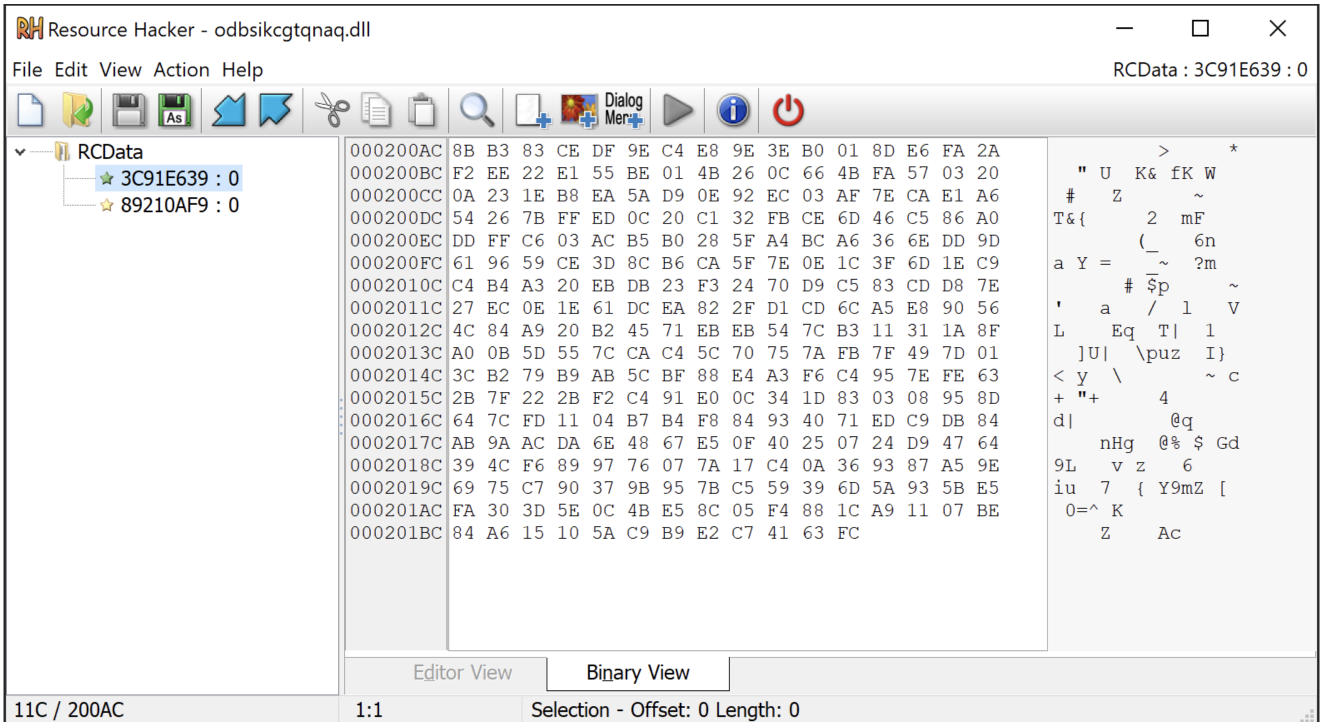
Figure 6 - Unpacked DLL details

After some analysis we can confirm that the malware is **QAKBOT**, the malware seems to be similar to the one reported by several blog post, anyway the **BOT Configuration** and the **C2 IPs** list are encrypted in a different way, so I'll only describe how to decrypt it instead of write something already reported in a very clear way by several blog posts:

- Elastic
- Hornetsecurity

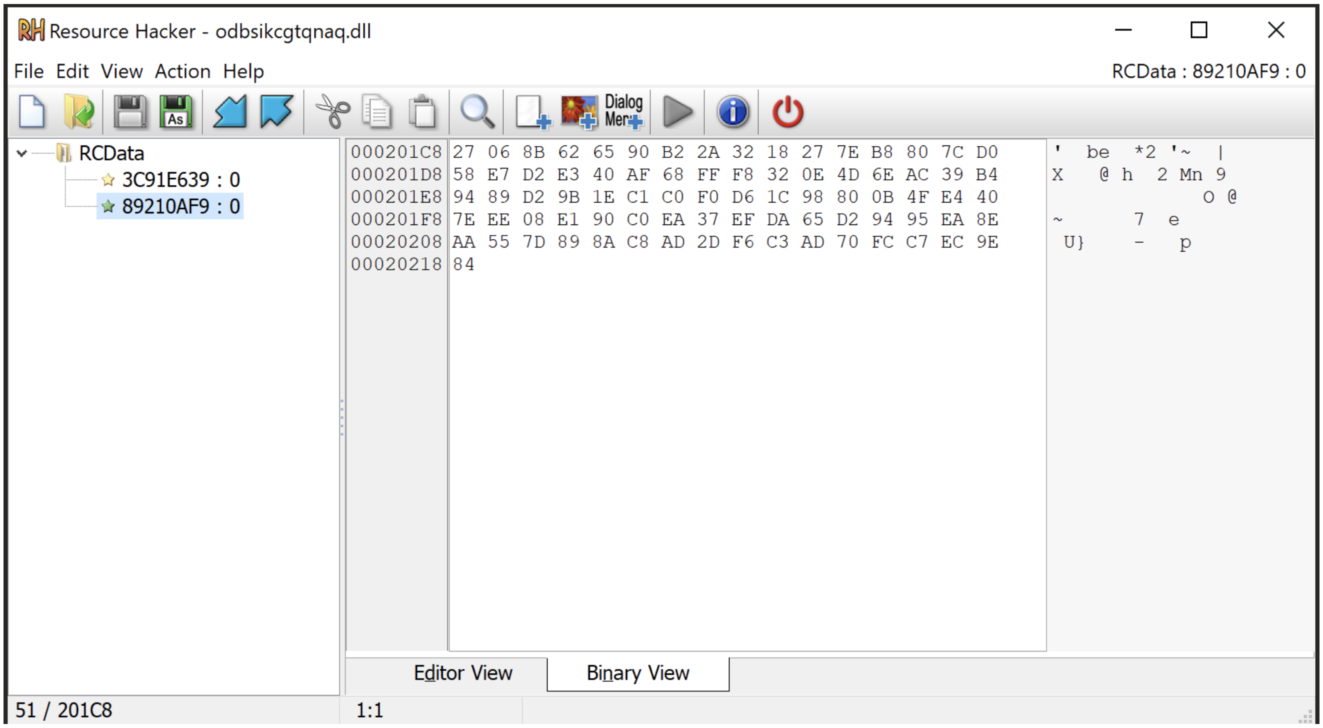
You can find all the decrypted strings and the scripts in my GitHub.

The file has two resources, one containing the encrypted **Configuration** and one containing the encrypted **C2 IPs list**.



```
000200AC 8B B3 83 CE DF 9E C4 E8 9E 3E B0 01 8D E6 FA 2A          >     *
000200BC F2 EE 22 E1 55 BE 01 4B 26 0C 66 4B FA 57 03 20    " U  K& fK W
000200CC 0A 23 1E B8 EA 5A D9 0E 92 EC 03 AF 7E CA E1 A6   #   Z      ~
000200DC 54 26 7B FF ED 0C 20 C1 32 FB CE 6D 46 C5 86 A0  T&{     2  mF
000200EC DD FF C6 03 AC B5 B0 28 5F A4 BC A6 36 6E DD 9D         (_   6n
000200FC 61 96 59 CE 3D 8C B6 CA 5F 7E 0E 1C 3F 6D 1E C9  a Y =   _~  ?m
0002010C C4 B4 A3 20 EB DB 23 F3 24 70 D9 C5 83 CD D8 7E        # $p     ~
0002011C 27 EC 0E 1E 61 DC EA 82 2F D1 CD 6C A5 E8 90 56  '   a   /  l   V
0002012C 4C 84 A9 20 B2 45 71 EB EB 54 7C B3 11 31 1A 8F  L    Eq  T|  1
0002013C A0 0B 5D 55 7C CA C4 5C 70 75 7A FB 7F 49 7D 01    ]U|  \puz  I}
0002014C 3C B2 79 B9 AB 5C BF 88 E4 A3 F6 C4 95 7E FE 63  < y  \       ~ c
0002015C 2B 7F 22 2B F2 C4 91 E0 0C 34 1D 83 03 08 95 8D  + "+     4
0002016C 64 7C FD 11 04 B7 B4 F8 84 93 40 71 ED C9 DB 84  d|        @q
0002017C AB 9A AC DA 6E 48 67 E5 0F 40 25 07 24 D9 47 64      nHg  @% $ Gd
0002018C 39 4C F6 89 97 76 07 7A 17 C4 0A 36 93 87 A5 9E  9L   v z   6
0002019C 69 75 C7 90 37 9B 95 7B C5 59 39 6D 5A 93 5B E5  iu  7  { Y9mZ [
000201AC FA 30 3D 5E 0C 4B E5 8C 05 F4 88 1C A9 11 07 BE   0=^ K
000201BC 84 A6 15 10 5A C9 B9 E2 C7 41 63 FC                  Z    Ac
```

Figure 7 - Resouce 3C91E639 containing the C2 list



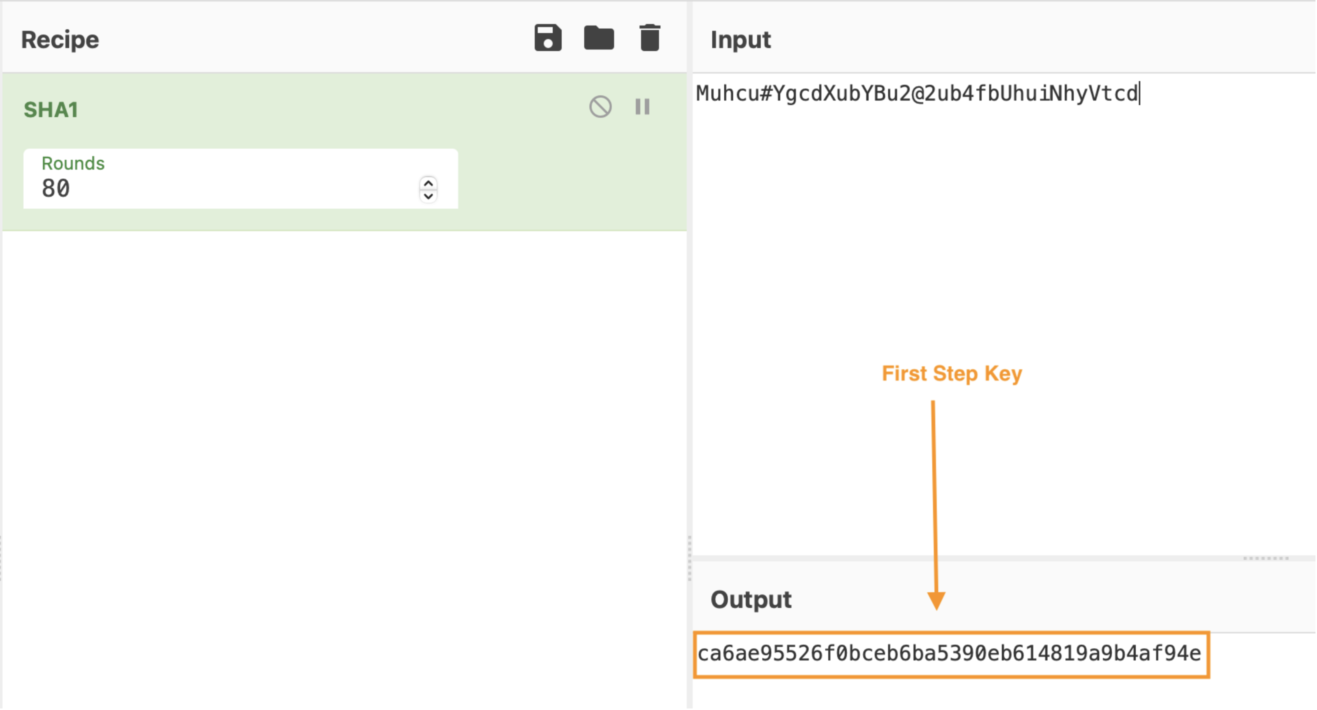
```
000201C8 27 06 8B 62 65 90 B2 2A 32 18 27 7E B8 80 7C D0  '  be  *2 '~  |
000201D8 58 E7 D2 E3 40 AF 68 FF F8 32 0E 4D 6E AC 39 B4  X   @ h  2 Mn 9
000201E8 94 89 D2 9B 1E C1 C0 F0 D6 1C 98 80 0B 4F E4 40              O @
000201F8 7E EE 08 E1 90 C0 EA 37 EF DA 65 D2 94 95 EA 8E  ~      7   e
00020208 AA 55 7D 89 8A C8 AD 2D F6 C3 AD 70 FC C7 EC 9E   U}    -   p
00020218 84
```

Figure 8 - Resource 89210AF9 containing the bot configuration

The resources are encrypted in the same way, so let’s use the configuration resource as example.

Two “steps” of **RC4 encryption** are used, let’ see it on CyberChef in order to be clearer.

As shown in the image below, in the first step, the **SHA1 Hash** is calculated on the string, “**Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd**”, the SHA1 Hash result is “**CA 6A E9 55 26 F0 BC EB 6B A5 39 0E B6 14 81 9A 9B 4A F9 4E**”, this will be the **RC4 key** (the string used is different in each qakbot sample, for example in another sample I analyzed it was “**bUdiuy81gYguty@4frdRdpfko(eKmudeuMncueaN**”, you have to figure out which string it uses).



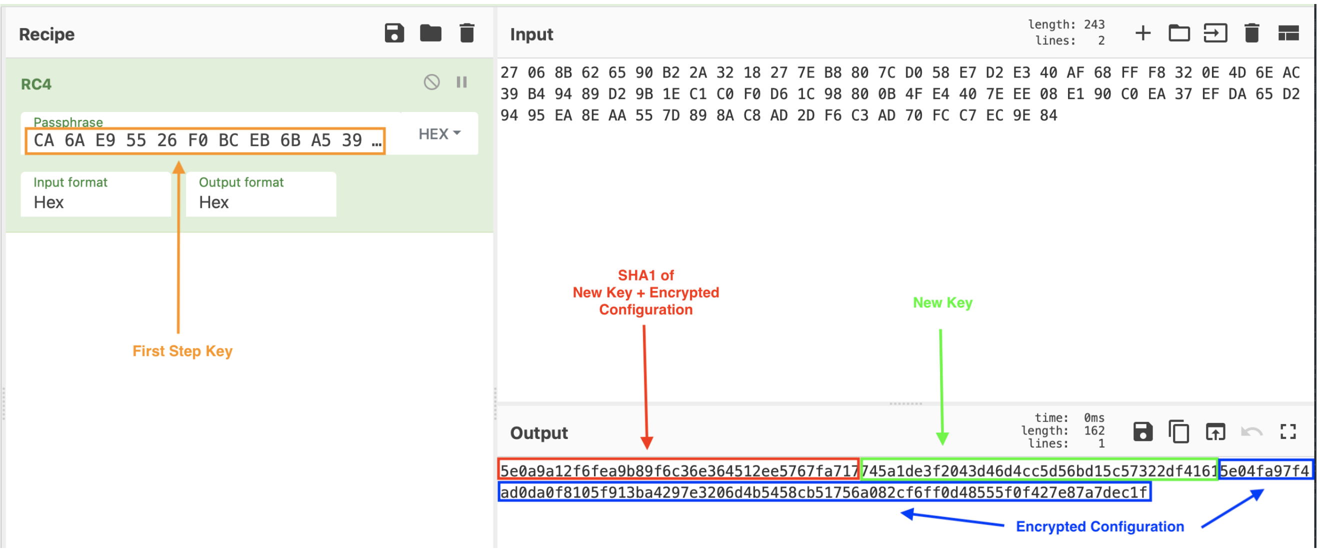
```
Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd
```

```
ca6ae95526f0bceb6ba5390eb614819a9b4af94e
```

Figure 9 - SHA1 Hash of the string "Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd"

Using the data we obtain from **SHA1** as key, we can use the **RC4 algorithm** to decrypt the data. The output from the first **RC4 decryption** will contains the following data:

- From bytes 0 to 20: **SHA1 Hash of New Key + Encrypted Configuration**
- From bytes 20 to 40: **New Key**
- From bytes 40 to end: **Encrypted Configuration**



Recipe

RC4

Passphrase: CA 6A E9 55 26 F0 BC EB 6B A5 39 … HEX

Input format: Hex

Output format: Hex

Input (length: 243, lines: 2)

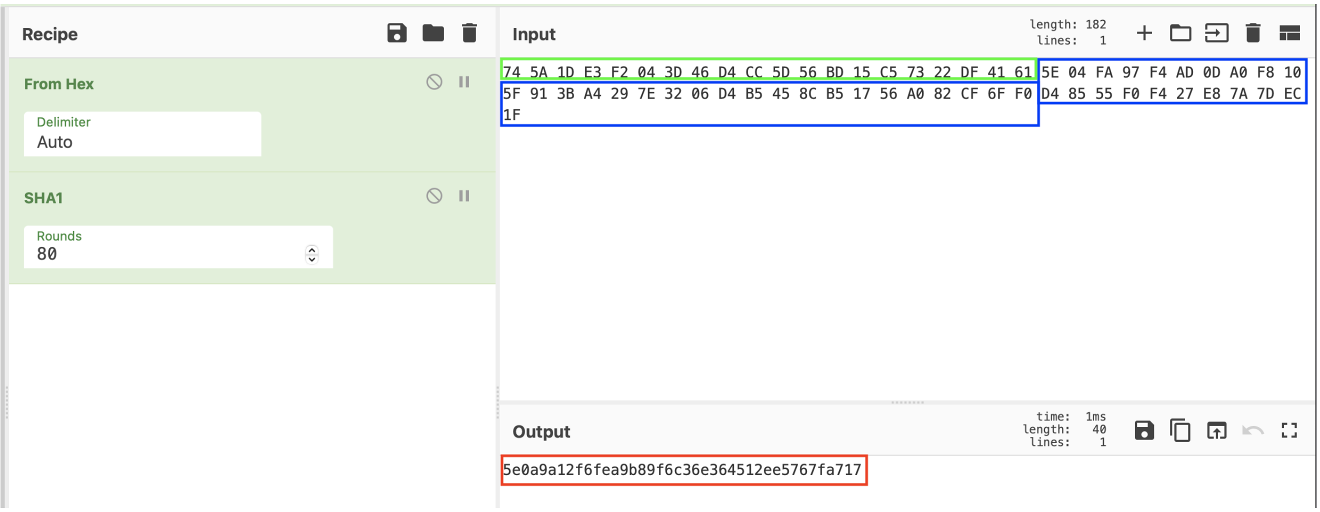
```
27 06 8B 62 65 90 B2 2A 32 18 27 7E B8 80 7C D0 58 E7 D2 E3 40 AF 68 FF F8 32 0E 4D 6E AC
39 B4 94 89 D2 9B 1E C1 C0 F0 D6 1C 98 80 0B 4F E4 40 7E EE 08 E1 90 C0 EA 37 EF DA 65 D2
94 95 EA 8E AA 55 7D 89 8A C8 AD 2D F6 C3 AD 70 FC C7 EC 9E 84
```

Output (time: 0ms, length: 162, lines: 1)

```
5e0a9a12f6fea9b89f6c36e364512ee5767fa717745a1de3f2043d46d4cc5d56bd15c57322df41615e04fa97f4
ad0da0f8105f913ba4297e3206d4b5458cb51756a082cf6ff0d48555f0f427e87a7dec1f
```

Figure 10 - Resource RC4 Decryption Step 1

In the image below we can see that the **SHA1 Hash** of **New Key** + **Encrypted Configuration** matches the first 20 bytes we got from the decrypted data.



Recipe

From Hex

Delimiter: Auto

SHA1

Rounds: 80

Input (length: 182, lines: 1)

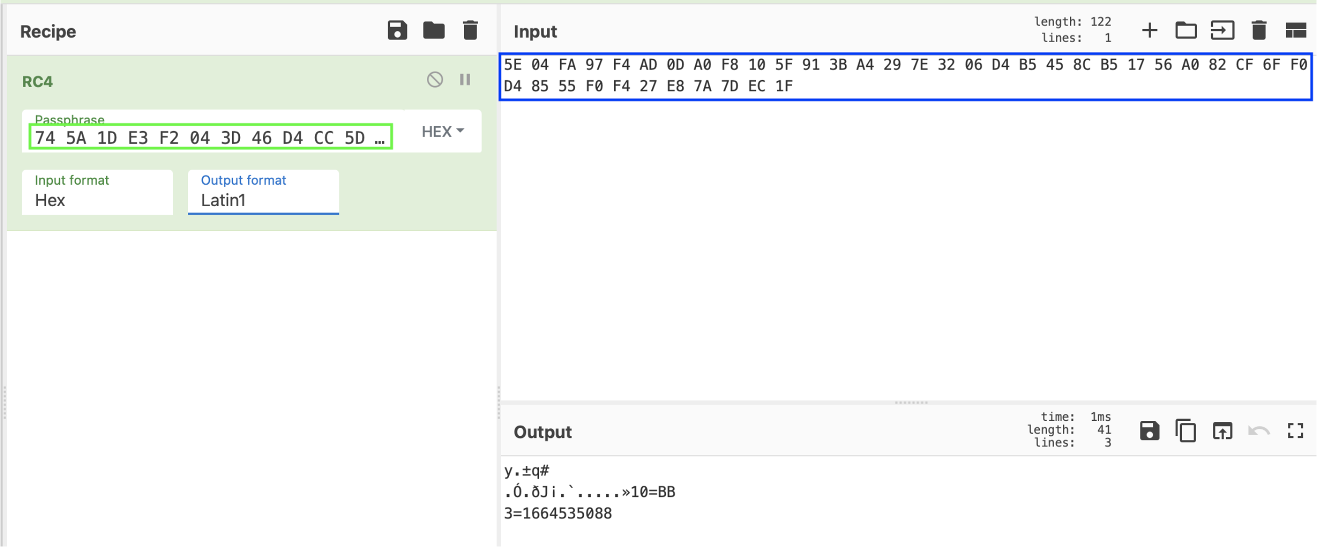
```
74 5A 1D E3 F2 04 3D 46 D4 CC 5D 56 BD 15 C5 73 22 DF 41 61 5E 04 FA 97 F4 AD 0D A0 F8 10
5F 91 3B A4 29 7E 32 06 D4 B5 45 8C B5 17 56 A0 82 CF 6F F0 D4 85 55 F0 F4 27 E8 7A 7D EC
1F
```

Output (time: 1ms, length: 40, lines: 1)

```
5e0a9a12f6fea9b89f6c36e364512ee5767fa717
```

Figure 11 - SHA1(Encrypted Configuration)

In the second step, the **RC4** algorithm is used with the **New Key** to decrypt the **Encrypted Configuration**. The following images shows the result of the second step decryption.



Passphrase: 74 5A 1D E3 F2 04 3D 46 D4 CC 5D …

Input format: Hex

Output format: Latin1

```
5E 04 FA 97 F4 AD 0D A0 F8 10 5F 91 3B A4 29 7E 32 06 D4 B5 45 8C B5 17 56 A0 82 CF 6F F0
D4 85 55 F0 F4 27 E8 7A 7D EC 1F
```

```
y.±q#
.Ó.ðJi.`.....»10=BB
3=1664535088
```

Figure 12 - Resource RC4 Decryption Step 2

The QAKBOT campaign ID is “**BB**” the timestamp **1664535088** corresponds to **Fri Sep 30 2022 10:51:28 GMT+0000**.

While writing this, a blog post by Trendmicro was published talking about this specific QAKBOT campaign.

To automatically extract the configuration and the C2 IPs, I wrote the following python script.

```
import hashlib
from arc4 import ARC4

file = open("89210AF9.bin","rb") #Resource with Qakbot configuration
resource = file.read()

key = hashlib.sha1(b"Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd").digest() #change with your
password
rc4 = ARC4(key)
data = rc4.decrypt(resource)

key = data[20:40]
rc4 = ARC4(key)

decrypted_data = rc4.decrypt(data[40:])
print("Qakbot Configuration:")
print((decrypted_data[20:]).decode("utf-8"))

file = open("3C91E639.bin","rb") #Resource with Qakbot C2
resource = file.read()

key = hashlib.sha1(b"Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd").digest() #change with your
password
rc4 = ARC4(key)
data = rc4.decrypt(resource)


key = data[20:40]
rc4 = ARC4(key)
#print(key)
decrypted_data = rc4.decrypt(data[40:])


print("Qakbot C2:")
for i in range(21,len(decrypted_data),7):
    c2 = bytearray(decrypted_data[i:i+7])
    print("%d.%d.%d.%d:%d" % (c2[0],c2[1],c2[2],c2[3],(c2[4]<<8)+c2[5]))
```

Hope this first malware blog post can help someone during his analysis of QAKBOT, you can find the samples at the following urls:

Configuration:

- 10=BB
- 3=1664535088

File Hashes:

- 5B54F57DBAA74FA589AFB2D26D6C6B39E0C2930BD88FEA3172556CE96B3EB959
- 796FF26DB045085EC8162D414CC2DEAFB2836D3F0BFFD8C58AF4595EBB4261E9

- D5F09EBC9B1F3FB9781ACA09E3B9FA63F90B909CC7418FF7D2AFA462F400DCE3
- 8B08C031D365A0B4D032C6E51BF773655E15795FE3EABCD3FA6487FFE9F3D6B3
- 93104C4834A27E39C13AC9D4663C6FA622AE6ECC5491A67DDF9125E6633CF07B
- 55AD915DCD65192548046ECBECDA5AD8AD6A92A11F07EC9A92744FCAC1599501
- 757D3C81555FBF635B2B9FD1D5222E6FE046710753395545A29E3E1F0A78FBF1
- BD3A47E0E27523044FEB2C30879EB684CFD174EC329350BAF5E0824FFFF1A22F

C2 IPs:

- 41.107.71[.]201:443
- 105.101.230[.]16:443
- 105.108.239[.]60:443
- 196.64.227[.]5:8443
- 41.249.158[.]221:995
- 134.35.14[.]5:443
- 113.170.117[.]251:443
- 187.193.219[.]248:443
- 122.166.244[.]116:443
- 154.237.129[.]123:995
- 41.98.229[.]81:443
- 186.48.199[.]243:995
- 102.156.3[.]13:443
- 41.97.190[.]189:443
- 197.207.191[.]164:443
- 105.184.14[.]132:995
- 196.207.146[.]151:443
- 105.158.113[.]15:443
- 196.89.42[.]89:995
- 86.98.156[.]229:993
- 177.174.119[.]195:32101
- 81.156.194[.]147:2078
- 80.253.189[.]55:443
- 197.49.175[.]67:995
- 177.45.78[.]52:993
- 89.187.169[.]77:443
- 196.92.59[.]242:995
- 41.13.200[.]19:443
- 41.97.195[.]237:443
- 92.191.56[.]11:2222
- 154.70.53[.]202:443
- 210.186.37[.]98:50002

You may also enjoy

Emotet Malicious Excel Analysis

August 26, 2022 1 minute read

Sometime ago a friend of mine sent me a suspicious email containg a zip file with an xls, at the time I didn’t focus too much on what the file does and simpl...

DVIA v2 iOS URL Runtime Manipulation with Frida

October 31, 2020 2 minute read

After my previous blog posts about DVIA v2 Anti-Debug and Frida with Swift some guys asked me about the URL Runtime Manipulation challenge in DVIA v2. I wil...

iOS Strings Obfuscation in Swift

October 13, 2020 4 minute read

Usually when reversing an iOS Application, it’s common to see methods and strings that can help an attacker to figure out how the application works. When I’...

ELF x64 Bypass NX with mprotect()

August 25, 2020 4 minute read

In this blogpost, I’ll explain how to bypass NX using mprotect() in order to make the stack executable.