QakBot reducing its on disk artifacts

Security Lab December 15, 2020

Summary

QakBot has been updated with more evasion techniques. QakBot's configuration is now stored in a registry key instead of a file. The run key for persistence is not permanently present in the registry but only written right before shutdown or reboot, and deleted immediately after QakBot is executed again. QakBot's executable is also not stored permanently on the file system anymore, but similarly to the run key registry entry, dropped onto the file system before reboots and deleted afterwards. This way security software can only detect QakBot artifacts on disk, right before system shutdown, and shortly after system boot. However, at that time security software itself is shutting down and booting up, hence may not detect QakBot's new persistence method.

Other changes include dynamic just-in-time decoding and destruction of strings at runtime. So any string used in the malware is only decoded at runtime into memory only and destroyed right afterwards.

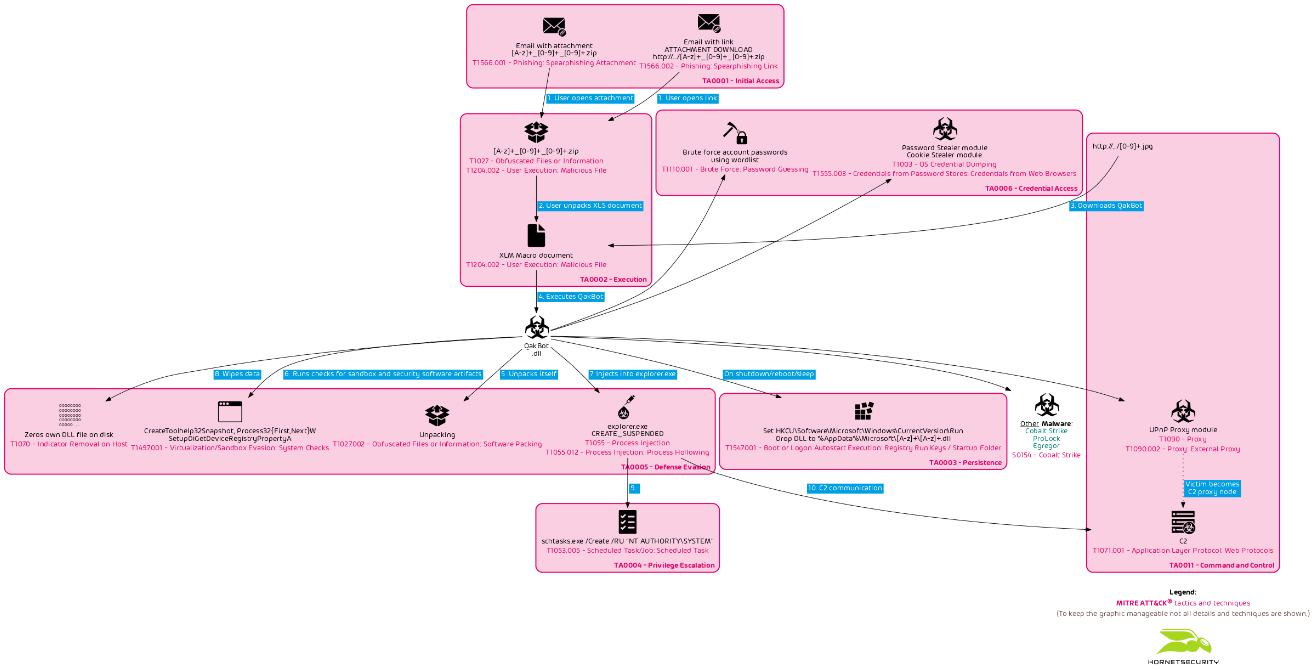
The delivery method for the observed QakBot campaigns identified via the regular expression pattern of `abc[0-9]+` is still XLM macro documents as reported previously.

Background

QakBot (also known as QBot, QuakBot, Pinkslipbot) has been around since 2008. It is distributed via Emotet, i.e., Emotet will download QakBot onto victims that are already infected with Emotet but it is also distributed directly via email. To this end, it uses email conversation thread hijacking in its campaigns[1], i.e., it will reply to emails that it finds in its victim's mailboxes. QakBot is known to escalate intrusions by downloading the ProLock ransomware[2] or lately the Egregor ransomware.

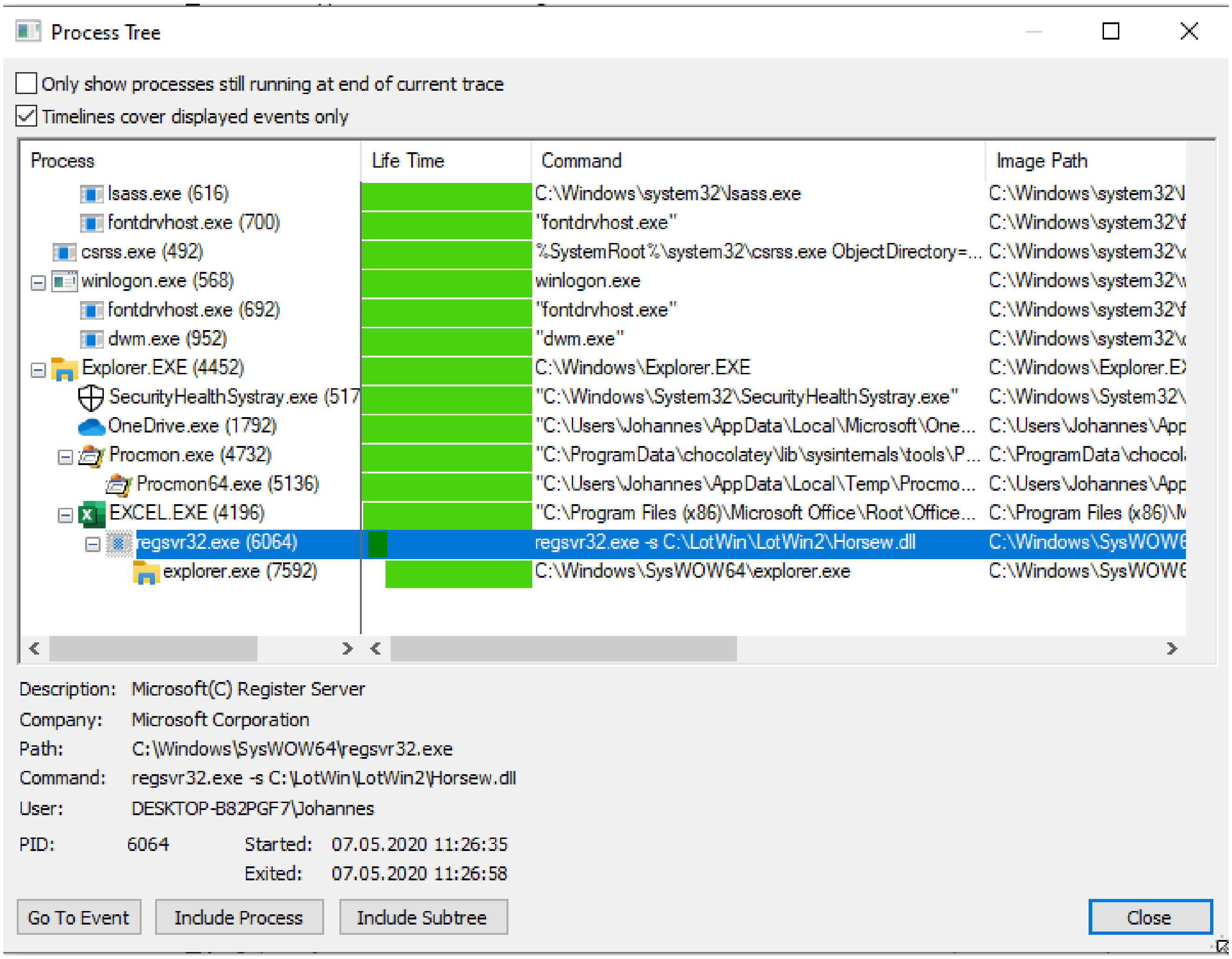
The observed QakBot campaigns identified by campaign ID `abc` use XLM macro documents for infection. We previously reported on their low detection.[3]

An overview of the current chain of infection used by the QakBot campaign with identifiers following the regular expression pattern of `abc[0-9]+` can be seen in the following flow graph.


Email with attachment
[A-z]+_[0-9]+_[0-9]+.zip
T1566.001 - Phishing: Spearphishing Attachment
Email with link
ATTACHMENT DOWNLOAD
http://.../[A-z]+_[0-9]+_[0-9]+.zip
T1566.002 - Phishing: Spearphishing Link
TA0001 - Initial Access
1. User opens attachment
1. User opens link
[A-z]+_[0-9]+_[0-9]+.zip
T1027 - Obfuscated Files or Information
T1204.002 - User Execution: Malicious File
2. User unpacks XLS document
XLM Macro document
T1204.002 - User Execution: Malicious File
TA0002 - Execution
Brute force account passwords
using wordlist
T1110.001 - Brute Force: Password Guessing
Password Stealer module
Cookie Stealer module
T1003 - OS Credential Dumping
T1555.003 - Credentials from Password Stores: Credentials from Web Browsers
TA0006 - Credential Access
http://.../[0-9]+.jpg
3. Downloads QakBot
4. Executes QakBot
QakBot
.dll
8. Wipes data
6. Runs checks for sandbox and security software artifacts
5. Unpacks itself
7. Injects into explorer.exe
On shutdown/reboot/sleep
Zeros own DLL file on disk
T1070 - Indicator Removal on Host
CreateToolhelp32Snapshot, Process32{First,Next}W
SetupDiGetDeviceRegistryPropertyA
T1497.001 - Virtualization/Sandbox Evasion: System Checks
Unpacking
T1027.002 - Obfuscated Files or Information: Software Packing
explorer.exe
CREATE_SUSPENDED
T1055 - Process Injection
T1055.012 - Process Injection: Process Hollowing
TA0005 - Defense Evasion
Set HKCU\Software\Microsoft\Windows\CurrentVersion\Run
Drop DLL to %AppData%\Microsoft\[A-z]+\[A-z]+.dll
T1547.001 - Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
TA0003 - Persistence
Other Malware:
Cobalt Strike
ProLock
Egregor
S0154 - Cobalt Strike
UPnP Proxy module
T1090 - Proxy
T1090.002 - Proxy: External Proxy
Victim becomes
C2 proxy node
9.
10. C2 communication
schtasks.exe /Create /RU "NT AUTHORITY\SYSTEM"
T1053.005 - Scheduled Task/Job: Scheduled Task
TA0004 - Privilege Escalation
C2
T1071.001 - Application Layer Protocol: Web Protocols
TA0011 - Command and Control
Legend:
MITRE ATT&CK® tactics and techniques
(To keep the graphic manageable not all details and techniques are shown.)
HORNETSECURITY

Technical Analysis

In the following analysis we briefly analyze the infection chain of QakBot after being downloaded and launched by the malicious Excel document.



☐ Only show processes still running at end of current trace
☑ Timelines cover displayed events only

Process	Life Time	Command	Image Path
lsass.exe (616)	🟩	C:\Windows\system32\lsass.exe	C:\Windows\system32\l
fontdrvhost.exe (700)	🟩	"fontdrvhost.exe"	C:\Windows\system32\f
csrss.exe (492)	🟩	%SystemRoot%\system32\csrss.exe ObjectDirectory=...	C:\Windows\system32\c
winlogon.exe (568)	🟩	winlogon.exe	C:\Windows\system32\w
fontdrvhost.exe (692)	🟩	"fontdrvhost.exe"	C:\Windows\system32\f
dwm.exe (952)	🟩	"dwm.exe"	C:\Windows\system32\c
Explorer.EXE (4452)	🟩	C:\Windows\Explorer.EXE	C:\Windows\Explorer.E
SecurityHealthSystray.exe (517	🟩	"C:\Windows\System32\SecurityHealthSystray.exe"	C:\Windows\System32\
OneDrive.exe (1792)	🟩	"C:\Users\Johannes\AppData\Local\Microsoft\One...	C:\Users\Johannes\App
Procmon.exe (4732)	🟩	"C:\ProgramData\chocolatey\lib\sysinternals\tools\P...	C:\ProgramData\chocol
Procmon64.exe (5136)	🟩	"C:\Users\Johannes\AppData\Local\Temp\Procmo...	C:\Users\Johannes\App
EXCEL.EXE (4196)	🟩	"C:\Program Files (x86)\Microsoft Office\Root\Office...	C:\Program Files (x86)\M
regsvr32.exe (6064)	🟩	regsvr32.exe -s C:\LotWin\LotWin2\Horsew.dll	C:\Windows\SysWOW6
explorer.exe (7592)	🟩	C:\Windows\SysWOW64\explorer.exe	C:\Windows\SysWOW6

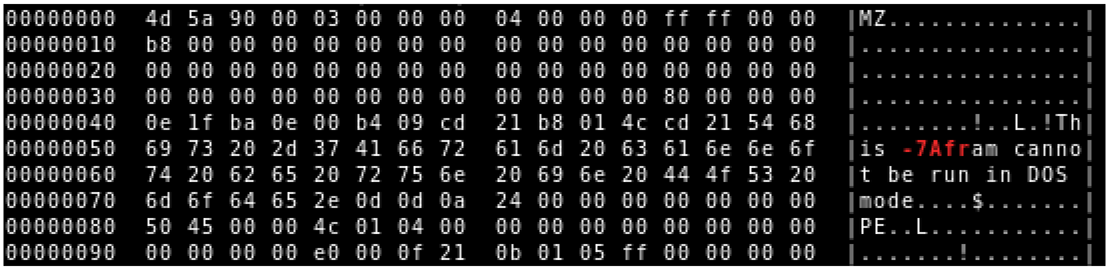
Description: Microsoft(C) Register Server
Company: Microsoft Corporation
Path: C:\Windows\SysWOW64\regsvr32.exe
Command: regsvr32.exe -s C:\LotWin\LotWin2\Horsew.dll
User: DESKTOP-B82PGF7\Johannes
PID: 6064 Started: 07.05.2020 11:26:35
Exited: 07.05.2020 11:26:58

Evasion

QakBot uses various evasion techniques to avoid detection by anti-virus software.

PE header manipulation

We observed some QakBot DLLs with a manipulated PE header. The message text `This program cannot be run in DOS mode.` has been altered.



```
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 2d 37 41 66 72  61 6d 20 63 61 6e 6e 6f  |is -7Afram canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  50 45 00 00 4c 01 04 00  00 00 00 00 00 00 00 00  |PE..L...........|
00000090  00 00 00 00 e0 00 0f 21  0b 01 05 ff 00 00 00 00  |.......!........|
```

This seems like an attempt to circumvent some static detection rules matching for this message in the legacy MS-DOS stub of PE binaries.

Code signing

First, the initial downloaded and executed DLL is signed with a (at the time the analyzed sample was distributed) valid code signing certificate.

```
$ chktrust 904400.jpg
Mono CheckTrust - version 6.8.0.123
Verify if an PE executable has a valid Authenticode(tm) signature
Copyright 2002, 2003 Motus Technologies. Copyright 2004-2008 Novell. BSD licensed.

WARNING! 904400.jpg is not timestamped!
SUCCESS: 904400.jpg signature is valid
and can be traced back to a trusted root!
```

The signing CA is Sectigo and the organization is given as Aqua Direct s.r.o., which is an existing company.

```
$ osslsigncode verify 904400.jpg
Current PE checksum   : 00091021
Calculated PE checksum: 00091021

Message digest algorithm  : SHA1
Current message digest    : 632DCB214EE9FB08441C640D240F672A7ABA6EB1
Calculated message digest : 632DCB214EE9FB08441C640D240F672A7ABA6EB1

Signature verification: ok

Number of signers: 1
    Signer #0:
        Subject: /C=CZ/postalCode=619 00/L=Brno/street=\xC5\xBDelezn\xC3\xA1
646/8/O=Aqua Direct s.r.o./CN=Aqua Direct s.r.o.
        Issuer : /C=GB/ST=Greater Manchester/L=Salford/O=Sectigo Limited/CN=Sectigo
RSA Code Signing CA

Number of certificates: 4
    Cert #0:
        Subject: /C=CZ/postalCode=619 00/L=Brno/street=\xC5\xBDelezn\xC3\xA1
646/8/O=Aqua Direct s.r.o./CN=Aqua Direct s.r.o.
        Issuer : /C=GB/ST=Greater Manchester/L=Salford/O=Sectigo Limited/CN=Sectigo
RSA Code Signing CA
    Cert #1:
        Subject: /C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA
Certificate Services
        Issuer : /C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA
Certificate Services
    Cert #2:
        Subject: /C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST
Network/CN=USERTrust RSA Certification Authority
        Issuer : /C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA
Certificate Services
    Cert #3:
        Subject: /C=GB/ST=Greater Manchester/L=Salford/O=Sectigo Limited/CN=Sectigo
RSA Code Signing CA
        Issuer : /C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST
Network/CN=USERTrust RSA Certification Authority

Succeeded
```

It is unknown whether the certificate was obtained from Sectigo by giving false information, the certificate was stolen from Aqua Direct s.r.o., or whether the certificate was obtained from Sectigo by giving stolen information from Aqua Direct s.r.o..

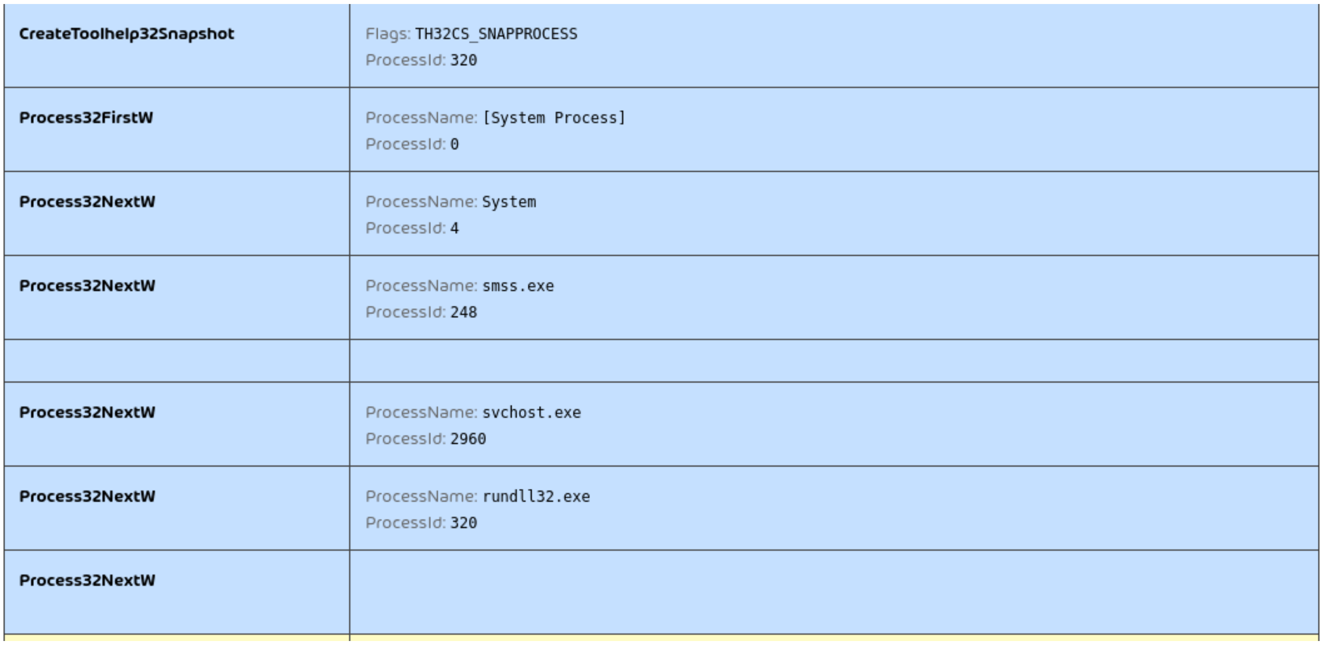
QakBot is known to steal victim emails and use them in future malspam campaigns. So it is likely that they also use stolen victim data to obtain code signing certificates. However, the actors behind QakBot can also buy the code signing certificate from a (malicious) third party.

Strings only decoded at runtime

QakBot will decode its strings only at runtime into memory. After usage the decoded strings are removed from memory again.

Processes

QakBot uses `CreateToolhelp32Snapshot` and `Process32{First,Next}W` to enumerate the running processes.



CreateToolhelp32Snapshot	Flags: TH32CS_SNAPPROCESS ProcessId: 320
Process32FirstW	ProcessName: [System Process] ProcessId: 0
Process32NextW	ProcessName: System ProcessId: 4
Process32NextW	ProcessName: smss.exe ProcessId: 248
Process32NextW	ProcessName: svchost.exe ProcessId: 2960
Process32NextW	ProcessName: rundll32.exe ProcessId: 320
Process32NextW	

It checks for the following processes:

- `CcSvcHst.exe`
- `avgcsrvx.exe`
- `avgsvcx.exe`
- `avgcsrva.exe`
- `MsMpEng.exe`
- `mcshield.exe`
- `avp.exe`
- `kavtray.exe`
- `egui.exe`
- `ekrn.exe`
- `bdagent.exe`
- `vsserv.exe`
- `vsservppl.exe`
- `AvastSvc.exe`
- `coreServiceShell.exe`
- `PccNTMon.exe`
- `NTRTScan.exe`
- `SAVAdminService.exe`
- `SavService.exe`
- `fshoster32.exe`
- `WRSA.exe`

- `vkise.exe`
- `iserv.exe`
- `cmdagent.exe`
- `ByteFence.exe`
- `MBAMService.exe`
- `mbamgui.exe`
- `fmon.exe`

QakBot will set specific bits in a bit mask for each running process it finds. Depending on the resulting bit mask the further infection path is altered, e.g., if `avp.exe` has been encountered. QakBot will later inject its code into `mobsync.exe` instead of `explorer.exe`. Because the searched process names are related to security solutions, we believe that this way QakBot tailors its execution path to evade detection by specific vendors.

Then in another loop, again using `CreateToolhelp32Snapshot` and `Process32{First,Next}W`, it checks for:

- `srvpost.exe`
- `frida-winjector-helper-32.exe`
- `frida-winjector-helper-64.exe`

If it detects any of those processes the execution flow will run into a loop continuously calling `WaitForSingleObject(handle, 0x1fa)` on a `handle` previously generated via `CreateEvent(NULL, FALSE, FALSE, ...)`, i.e., it runs in an infinite loop.

Device drivers

Next, QakBot uses `SetupDiGetDeviceRegistryPropertyA` (querying properties `SPDRP_DEVICEDESC` and `SPDRP_SERVICE`) to check for device drivers containing the following strings:

- `VBoxVideo`
- `Red Hat VirtIO`
- `QEMU`
- `A3E64E55_pr`

We believe the search for `A3E64E55_pr` is used to detect an artifact of the ANY.RUN sandbox.[4] Alternatively, but unlikely, it could be used to detect an artifact of the long ago defunct xCore Complex Protection AV solution using a similar driver with the name `A3E64E55_pr.sys`.

If it detects any of those device drivers the execution flow will run into the same infinite loop continuously calling `WaitForSingleObject(handle, 0x1fa)` on a `handle` previously generated via `CreateEvent(NULL, FALSE, FALSE, ...)`, as previously mentioned.

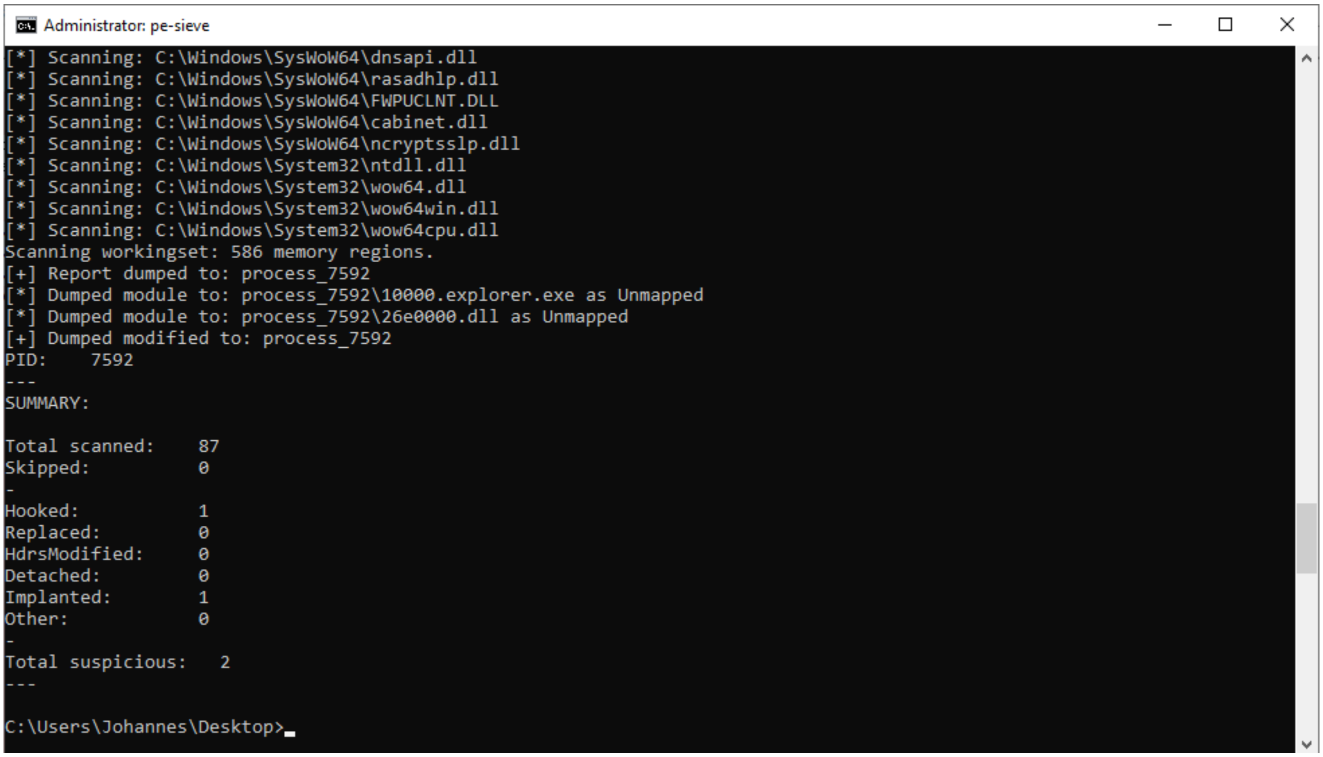
Process injection

QakBot starts `C:\Windows\SysWOW64\explorer.exe` in suspended state and injects a DLL into it using `CreateProcessInternalW` , `NtMapViewOfSection` , `NtAllocateVirtualMemory` , `WriteProcessMemory` , `memcpy` , `NtProtectVirtualMemory` and `NtResumeThread` .

CreateProcessInternalW	ApplicationName: CommandLine: C:\Windows\explorer.exe CreationFlags: CREATE_SUSPENDED ProcessId: 1516 ThreadId: 2508 ProcessHandle: 0x0000019c ThreadHandle: 0x00000198 StackPivoted: no
NtCreateSection	SectionHandle: 0x000001a4 DesiredAccess: SECTION_MAP_READ|SECTION_MAP_WRITE|SECTION_MAP_EXECUTE ObjectAttributes: FileHandle: 0x00000000
NtMapViewOfSection	SectionHandle: 0x000001a4 ProcessHandle: 0xffffffff BaseAddress: 0x004a0000 SectionOffset: 0x00000000 ViewSize: 0x00020000 Win32Protect: PAGE_EXECUTE_READWRITE StackPivoted: no
NtMapViewOfSection	SectionHandle: 0x000001a4 ProcessHandle: 0x0000019c BaseAddress: 0x00060000 SectionOffset: 0x00000000 ViewSize: 0x00020000 Win32Protect: PAGE_EXECUTE_READWRITE StackPivoted: no
NtAllocateVirtualMemory	ProcessHandle: 0x0000019c BaseAddress: 0x00080000 RegionSize: 0x00002000 Protection: PAGE_READWRITE StackPivoted: no
WriteProcessMemory	ProcessHandle: 0x0000019c BaseAddress: 0x00080000 Buffer: \x9c\x00\x00\x00\x06\x00\x00\x00\x01\x00\x00\x00\xb1\x1d\x00\x00\x02\x00\x00\x00Service Pack 1\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x01\x01\x1e\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x8e\x19\x89\xa7bdvmxiiz\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00b\x00d\x00v\x00m\x00x\x00i\x00i\x00z\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 BufferLength: 0x00001ac4 StackPivoted: no
memcpy	DestinationBuffer: MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00\xb8\x00\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x08\x01\x00\x00\x0e\x1f\xba\x0e\x00\xb4 \xcd!\xb8\x01L\xcd!This program cannot be run in DOS mode. $\x00\x00\x00\x00\x00\x00\x00\x98\x0b\x00v\xdcjn%\xdcjn%\xdcjn%\xc7\xf7\xf2%\xdejn%\xd5\x12\xed%\xd8jn%\xc7\xf7\xc4%\xd0jn%\xdcjo%Wjn%\xd5\x12\xfd%\xcfjn%\xc7\xf7\xf0%\xdejn%\xc7\xf7\xc5%\xdfjn%\xd9fa%\xddjn%\xc7\xf7\xc1%\x99jn%\xc7\xf7\xf5%\xddjn%\xc7\xf7\xf3%\xddjn%Rich\xdcjn%\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 source: 0x10000000 destination: 0x004a0000 count: 131072
NtUnmapViewOfSection	ProcessHandle: 0xffffffff BaseAddress: 0x004a0000 RegionSize: 0x00020000
NtResumeThread	ThreadHandle: 0x00000198

```
SuspendCount: 1
ProcessId: 1516
```

The injected DLL can be extracted via PE-sieve[5] or other tools for simplyfied further analysis.

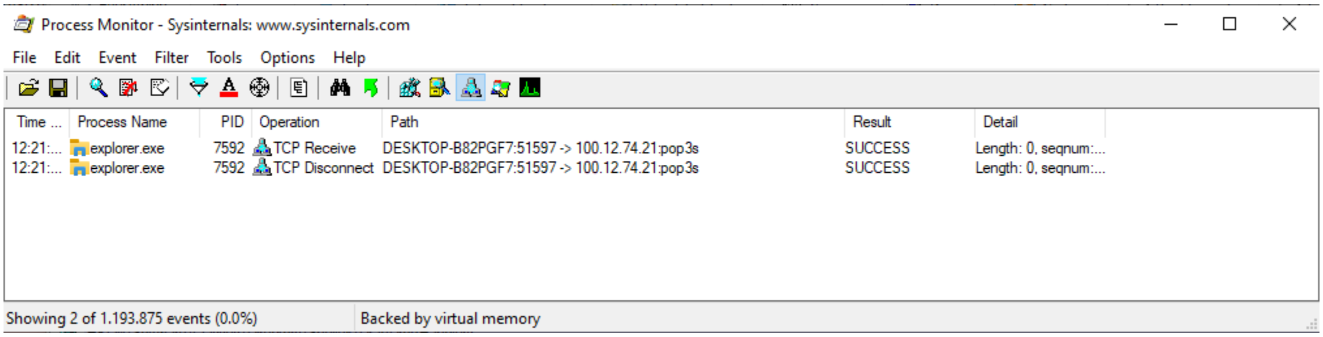


```
[*] Scanning: C:\Windows\SysWoW64\dnsapi.dll
[*] Scanning: C:\Windows\SysWoW64\rasadhlp.dll
[*] Scanning: C:\Windows\SysWoW64\FWPUCLNT.DLL
[*] Scanning: C:\Windows\SysWoW64\cabinet.dll
[*] Scanning: C:\Windows\SysWoW64\ncryptsslp.dll
[*] Scanning: C:\Windows\System32\ntdll.dll
[*] Scanning: C:\Windows\System32\wow64.dll
[*] Scanning: C:\Windows\System32\wow64win.dll
[*] Scanning: C:\Windows\System32\wow64cpu.dll
Scanning workingset: 586 memory regions.
[+] Report dumped to: process_7592
[*] Dumped module to: process_7592\10000.explorer.exe as Unmapped
[*] Dumped module to: process_7592\26e0000.dll as Unmapped
[+] Dumped modified to: process_7592
PID:    7592
---
SUMMARY:

Total scanned:    87
Skipped:          0
-
Hooked:           1
Replaced:         0
HdrsModified:     0
Detached:         0
Implanted:        1
Other:            0
-
Total suspicious:   2
---

C:\Users\Johannes\Desktop>_
```

Depending on whether the previous process enumeration yielded results on the list, QakBot will inject into `mobsync.exe` (e.g., in case a `avp.exe` process is found running) instead of `explorer.exe`. But for simplicity we will only follow the `explorer.exe` process injection path we observed in our analysis environment.

C2 communication

After avoiding detection, the injected QakBot code within `explorer.exe` will start communicating with the C2 servers.

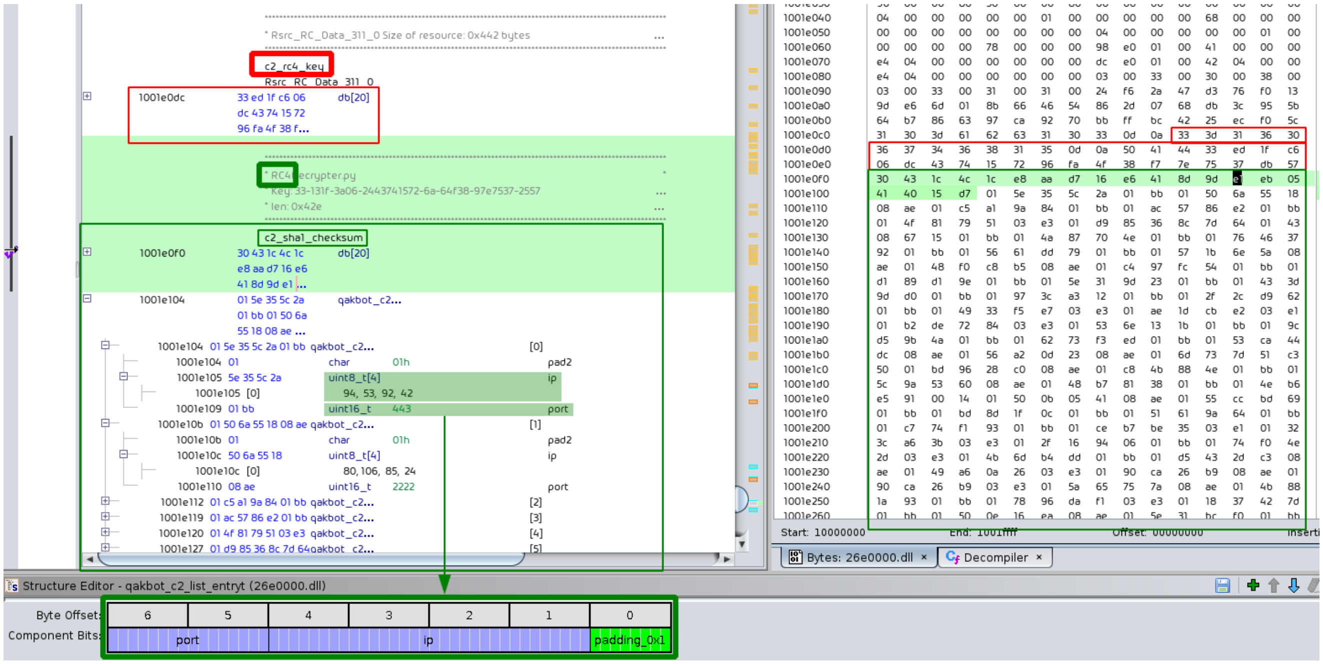


Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
12:21:...	explorer.exe	7592	TCP Receive	DESKTOP-B82PGF7:51597 -> 100.12.74.21:pop3s	SUCCESS	Length: 0, seqnum:...
12:21:...	explorer.exe	7592	TCP Disconnect	DESKTOP-B82PGF7:51597 -> 100.12.74.21:pop3s	SUCCESS	Length: 0, seqnum:...

Showing 2 of 1.193.875 events (0.0%) Backed by virtual memory

Like in previous versions of QakBot the C2 IP list is stored RC4 encrypted in resource section `311`. The first 20 bytes of the section contains the RC4 key with which the rest of the section is decrypted. The first 20 bytes of the decrypted data will contain the SHA1 sum calculated over the rest of the decrypted data. It is used as a verification for correct decryption. Unlike in previous version, the C2 list is now stored in binary form and not as ASCII text anymore.

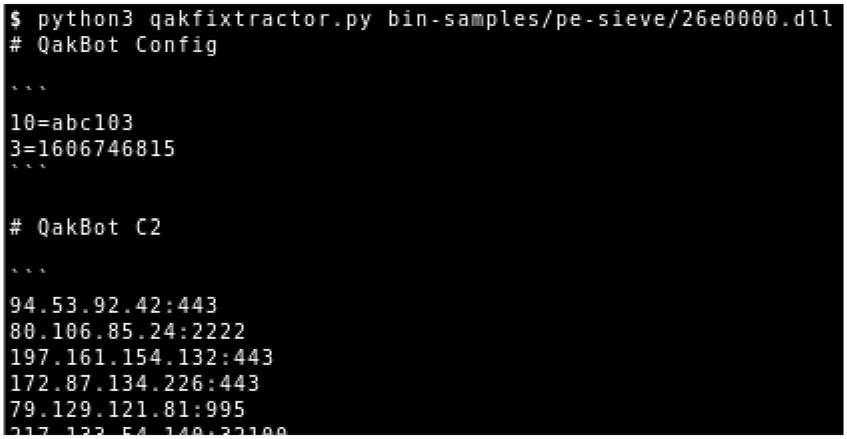


```
* Rsrc_RC_Data_311_0 Size of resource: 0x442 bytes          ...

                 c2_rc4_key
                 Rsrc  RC  Data  311  0
1001e0dc         33 ed 1f c6 06      db[20]
                 dc 43 74 15 72
                 96 fa 4f 38 f...

* RC4 decrypter.py                                           *
* Key: 33-131f-3a06-2443741572-6a-64f38-97e7537-2557        ...
* len: 0x42e                                                ...

                 c2_sha1_checksum
1001e0f0         30 43 1c 4c 1c      db[20]
                 e8 aa d7 16 e6
                 41 8d 9d e1 ...
1001e104         01 5e 35 5c 2a      qakbot_c2...
                 01 bb 01 50 6a
                 55 18 08 ae ...
   1001e104  01 5e 35 5c 2a 01 bb qakbot_c2...                  [0]
      1001e104  01                 char       01h                 pad2
      1001e105  5e 35 5c 2a        uint8_t[4]                     ip
         1001e105  [0]               94, 53, 92, 42
      1001e109  01 bb              uint16_t   443                 port
   1001e10b  01 50 6a 55 18 08 ae qakbot_c2...                  [1]
      1001e10b  01                 char       01h                 pad2
      1001e10c  50 6a 55 18        uint8_t[4]                     ip
         1001e10c  [0]               80, 106, 85, 24
      1001e110  08 ae              uint16_t   2222                port
   1001e112  01 c5 a1 9a 84 01 bb qakbot_c2...                  [2]
   1001e119  01 ac 57 86 e2 01 bb qakbot_c2...                  [3]
   1001e120  01 4f 81 79 51 03 e3 qakbot_c2...                  [4]
   1001e127  01 d9 85 36 8c 7d 64qakbot_c2...                   [5]
```

```
1001e040   04 00 00 00 00 00 01 00 00 00 00 00 68 00 00 00
1001e050   00 00 00 00 00 00 00 00 04 00 00 00 00 00 01 00
1001e060   00 00 00 00 78 00 00 00 98 e0 01 00 41 00 00 00
1001e070   e4 04 00 00 00 00 00 00 dc e0 01 00 42 04 00 00
1001e080   e4 04 00 00 00 00 00 00 03 00 33 00 30 00 38 00
1001e090   03 00 33 00 31 00 31 00 24 f6 2a 47 d3 76 f0 13
1001e0a0   9d e6 6d 01 8b 66 46 54 86 2d 07 68 db 3c 95 5b
1001e0b0   64 b7 86 63 97 ca 92 70 bb ff bc 42 25 ec f0 5c
1001e0c0   31 30 3d 61 62 63 31 30 33 0d 0a 33 3d 31 36 30
1001e0d0   36 37 34 36 38 31 35 0d 0a 50 41 44 33 ed 1f c6
1001e0e0   06 dc 43 74 15 72 96 fa 4f 38 f7 7e 75 37 db 57
1001e0f0   30 43 1c 4c 1c e8 aa d7 16 e6 41 8d 9d e1 eb 05
1001e100   41 40 15 d7 01 5e 35 5c 2a 01 bb 01 50 6a 55 18
1001e110   08 ae 01 c5 a1 9a 84 01 bb 01 ac 57 86 e2 01 bb
1001e120   01 4f 81 79 51 03 e3 01 d9 85 36 8c 7d 64 01 43
1001e130   08 67 15 01 bb 01 4a 87 70 4e 01 bb 01 76 46 37
1001e140   92 01 bb 01 56 61 dd 79 01 bb 01 57 1b 6e 5a 08
1001e150   ae 01 48 f0 c8 b5 08 ae 01 c4 97 fc 54 01 bb 01
1001e160   d1 89 d1 9e 01 bb 01 5e 31 9d 23 01 bb 01 43 3d
1001e170   9d d0 01 bb 01 97 3c a3 12 01 bb 01 2f 2c d9 62
1001e180   01 bb 01 49 33 f5 e7 03 e3 01 ae 1d cb e2 03 e1
1001e190   01 b2 de 72 84 03 e3 01 53 6e 13 1b 01 bb 01 9c
1001e1a0   d5 9b 4a 01 bb 01 62 73 f3 ed 01 bb 01 53 ca 44
1001e1b0   dc 08 ae 01 56 a2 0d 23 08 ae 01 6d 73 7d 51 c3
1001e1c0   50 01 bd 96 28 c0 08 ae 01 c8 4b 88 4e 01 bb 01
1001e1d0   5c 9a 53 60 08 ae 01 48 b7 81 38 01 bb 01 4e b6
1001e1e0   e5 91 00 14 01 50 0b 05 41 08 ae 01 55 cc bd 69
1001e1f0   01 bb 01 bd 8d 1f 0c 01 bb 01 51 61 9a 64 01 bb
1001e200   01 c7 74 f1 93 01 bb 01 ce b7 be 35 03 e1 01 32
1001e210   3c a6 3b 03 e3 01 2f 16 94 06 01 bb 01 74 f0 4e
1001e220   2d 03 e3 01 4b 6d b4 dd 01 bb 01 d5 43 2d c3 08
1001e230   ae 01 49 a6 0a 26 03 e3 01 90 ca 26 b9 08 ae 01
1001e240   90 ca 26 b9 03 e3 01 5a 65 75 7a 08 ae 01 4b 88
1001e250   1a 93 01 bb 01 78 96 da f1 03 e3 01 18 37 42 7d
1001e260   01 bb 01 50 0e 16 ea 08 ae 01 5e 31 bc f0 01 bb
Start: 10000000        End: 1001ffff        Offset: 00000000        Inserti
```

Byte Offset:	6	5	4	3	2	1	0
Component Bits:	port		ip				padding_0x1

For details on how to extract the C2 list and QakBot's configuration see the Python3 script in the appendix. The input to the script is the path to the DLL that QakBot injected into `explorer.exe`, which we previously extracted via PE-sieve[5].

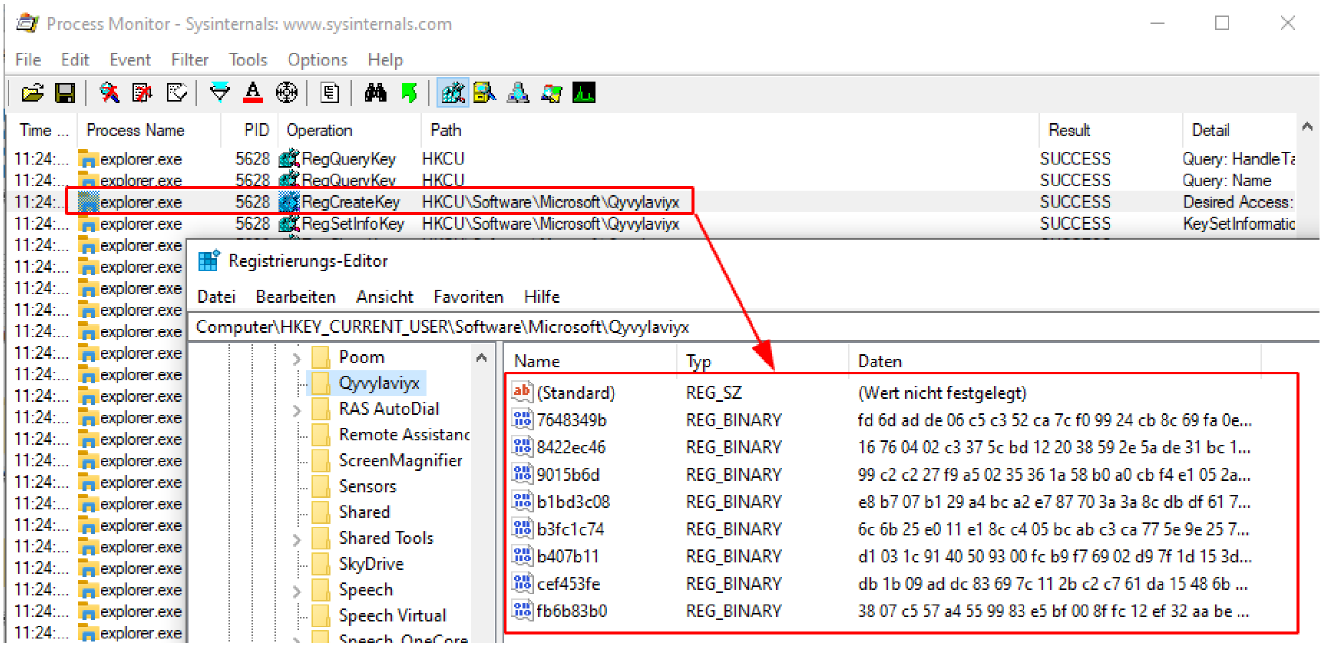


~~~
$ python3 qakfixtractor.py bin-samples/pe-sieve/26e0000.dll
# QakBot Config

```
10=abc103
3=1606746815
```

# QakBot C2

```
94.53.92.42:443
80.106.85.24:2222
197.161.154.132:443
172.87.134.226:443
79.129.121.81:995
217.133.54.140:32100
~~~

The configuration is stored using the same RC4 encryption scheme in resource section `308`. In it we can see the bot and/or campaign ID `abc103` that is associated with the analyzed sample. It is still stored in plain ACSII text. For each campaign the number is increased by one. This allows the operators behind QakBot to keep track to which campaign each victim connecting to their C2 server belongs to. Another currently observed identifier is `tr02`. This identifier, however, stayed the same over multiple malspam campaigns.

Via the C2 connection the operators behind QakBot can remote control the malware and deploy additional malicious modules.

QakBot will not store its configuration and C2 list on disk anymore. It will use the registry for storage.



Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
11:24:...	explorer.exe	5628	RegQueryKey	HKCU	SUCCESS	Query: HandleTa
11:24:...	explorer.exe	5628	RegQueryKey	HKCU	SUCCESS	Query: Name
11:24:...	explorer.exe	5628	RegCreateKey	HKCU\Software\Microsoft\Qyvylaviyx	SUCCESS	Desired Access:
11:24:...	explorer.exe	5628	RegSetInfoKey	HKCU\Software\Microsoft\Qyvylaviyx	SUCCESS	KeySetInformatio
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					
11:24:...	explorer.exe					

Computer\HKEY_CURRENT_USER\Software\Microsoft\Qyvylaviyx

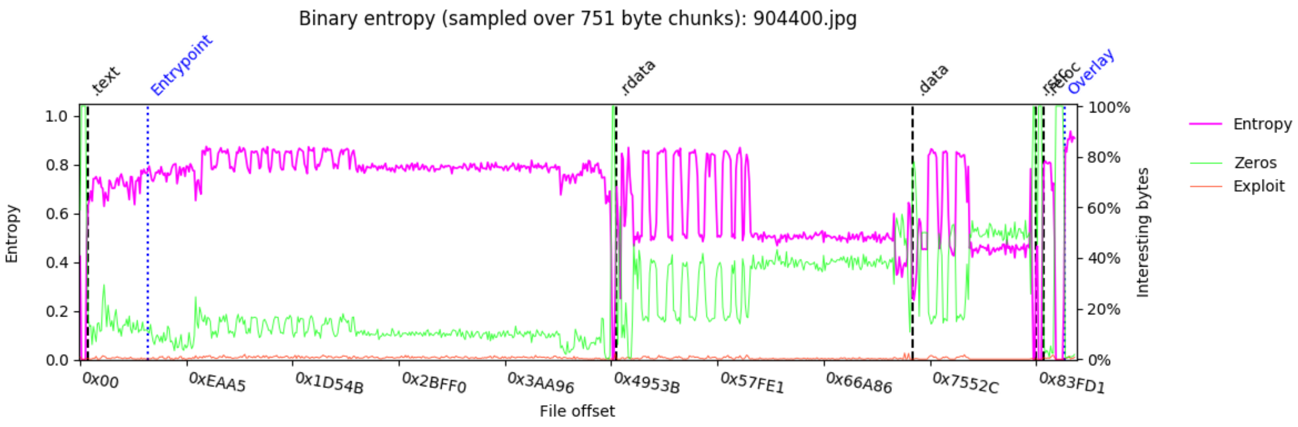
- Poom
- Qyvylaviyx
- RAS AutoDial
- Remote Assistanc
- ScreenMagnifier
- Sensors
- Shared
- Shared Tools
- SkyDrive
- Speech
- Speech Virtual
- Speech_OneCore

Name	Typ	Daten
(Standard)	REG_SZ	(Wert nicht festgelegt)
7648349b	REG_BINARY	fd 6d ad de 06 c5 c3 52 ca 7c f0 99 24 cb 8c 69 fa 0e...
8422ec46	REG_BINARY	16 76 04 02 c3 37 5c bd 12 20 38 59 2e 5a de 31 bc 1...
9015b6d	REG_BINARY	99 c2 c2 27 f9 a5 02 35 36 1a 58 b0 a0 cb f4 e1 05 2a...
b1bd3c08	REG_BINARY	e8 b7 07 b1 29 a4 bc a2 e7 87 70 3a 3a 8c db df 61 7...
b3fc1c74	REG_BINARY	6c 6b 25 e0 11 e1 8c c4 05 bc ab c3 ca 77 5e 9e 25 7...
b407b11	REG_BINARY	d1 03 1c 91 40 50 93 00 fc b9 f7 69 02 d9 7f 1d 15 3d...
cef453fe	REG_BINARY	db 1b 09 ad dc 83 69 7c 11 2b c2 c7 61 da 15 48 6b ...
fb6b83b0	REG_BINARY	38 07 c5 57 a4 55 99 83 e5 bf 00 8f fc 12 ef 32 aa be ...

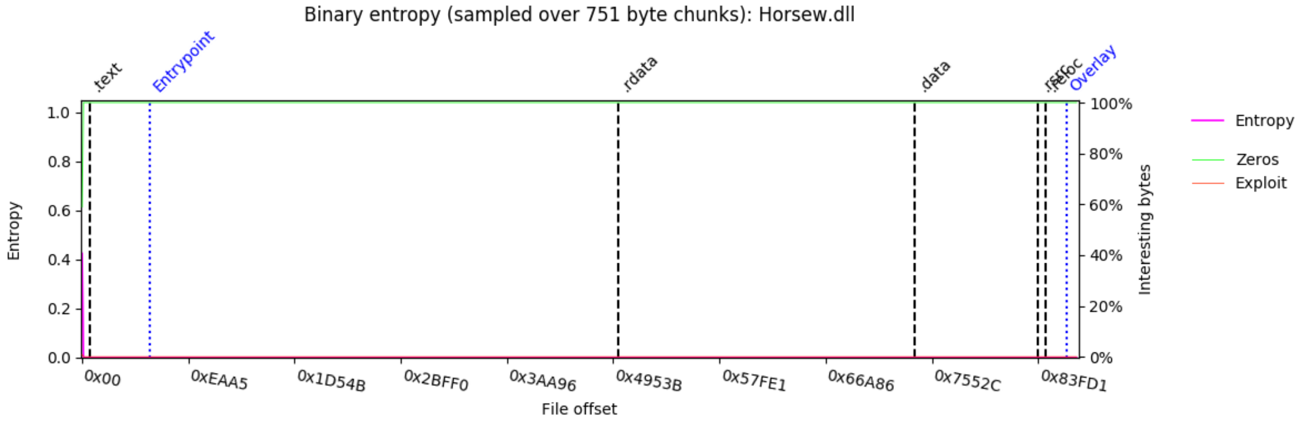
Wiping

The previous QakBot version used to overwrite its initial executable with a copy of `cmd.exe`. This version will overwrite the portion of the initially downloaded DLL after the PE header with zeros.

Here is the entropy of the QakBot DLL as downloaded.



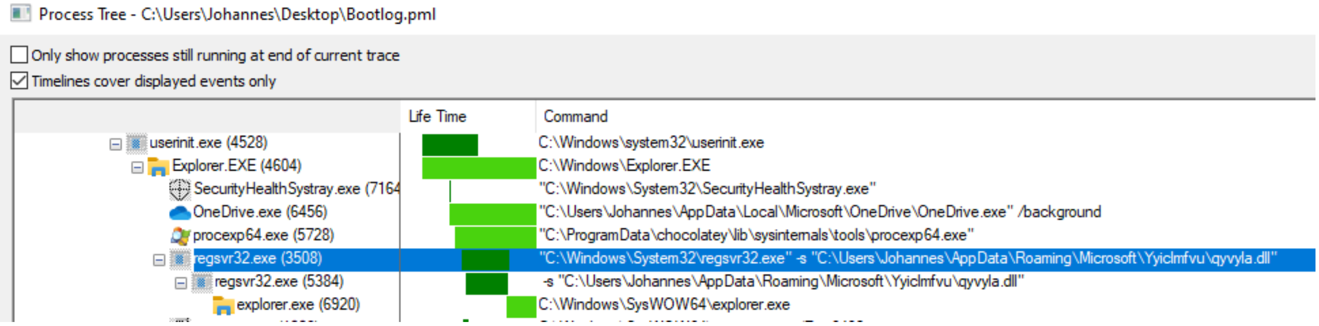
The zeroing of data after the header can be clearly seen when comparing the previous plot against a plot of the DLL file after wiping.



Persistence

The persistence mechanism of QakBot has also changed. While it still uses a run key registry entry under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`, this key is only set right before the system is shutdown, rebooted or put to sleep. The corresponding DLL is also only dropped to disk right before shutdown, rebooted or sleep.

After the system boots up again, QakBot is started via the run key. The execution tree also starts via `regsvr32.exe -s ...` like the initial execution from Excel. QakBot follows the same steps as previously outlined resulting in process injection into `explorer.exe`.

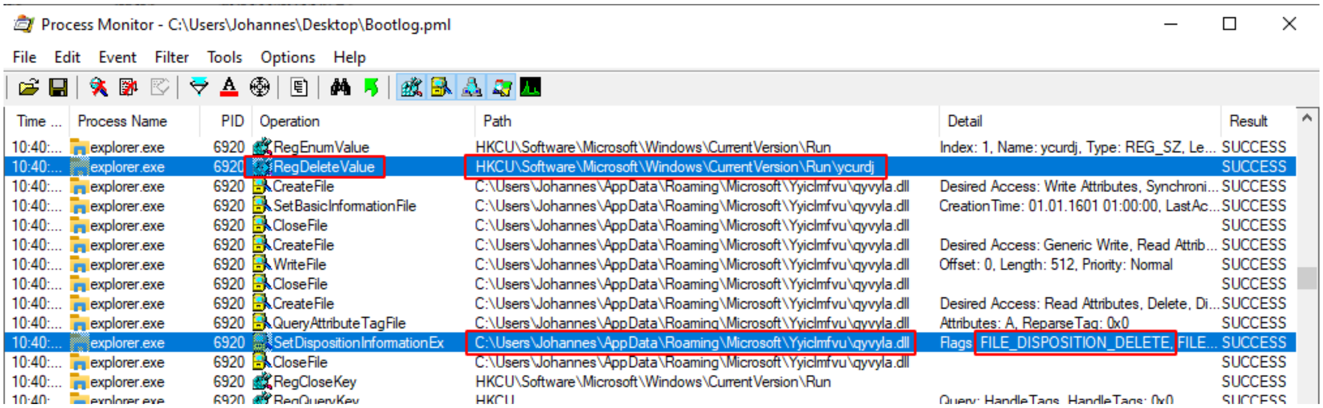


Process Tree - C:\Users\Johannes\Desktop\Bootlog.pml

□ Only show processes still running at end of current trace
☑ Timelines cover displayed events only

	Life Time	Command
userinit.exe (4528)		C:\Windows\system32\userinit.exe
Explorer.EXE (4604)		C:\Windows\Explorer.EXE
SecurityHealthSystray.exe (7164		"C:\Windows\System32\SecurityHealthSystray.exe"
OneDrive.exe (6456)		"C:\Users\Johannes\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
procexp64.exe (5728)		"C:\ProgramData\chocolatey\lib\sysinternals\tools\procexp64.exe"
regsvr32.exe (3508)		"C:\Windows\System32\regsvr32.exe" -s "C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll"
regsvr32.exe (5384)		-s "C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll"
explorer.exe (6920)		C:\Windows\SysWOW64\explorer.exe

QakBot will then delete the run key registry entry and delete the DLL it dropped to disk prior to the reboot.



Time ...	Process Name	PID	Operation	Path	Detail	Result
10:40:...	explorer.exe	6920	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	Index: 1, Name: ycurdj, Type: REG_SZ, Le...	SUCCESS
10:40:...	explorer.exe	6920	RegDeleteValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ycurdj		SUCCESS
10:40:...	explorer.exe	6920	CreateFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	Desired Access: Write Attributes, Synchroni...	SUCCESS
10:40:...	explorer.exe	6920	SetBasicInfomationFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	CreationTime: 01.01.1601 01:00:00, LastAc...	SUCCESS
10:40:...	explorer.exe	6920	CloseFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll		SUCCESS
10:40:...	explorer.exe	6920	CreateFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	Desired Access: Generic Write, Read Attrib...	SUCCESS
10:40:...	explorer.exe	6920	WriteFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	Offset: 0, Length: 512, Priority: Normal	SUCCESS
10:40:...	explorer.exe	6920	CloseFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll		SUCCESS
10:40:...	explorer.exe	6920	CreateFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	Desired Access: Read Attributes, Delete, Di...	SUCCESS
10:40:...	explorer.exe	6920	QueryAttributeTagFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	Attributes: A, ReparseTag: 0x0	SUCCESS
10:40:...	explorer.exe	6920	SetDispositionInformationEx	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll	Flags: FILE_DISPOSITION_DELETE, FILE...	SUCCESS
10:40:...	explorer.exe	6920	CloseFile	C:\Users\Johannes\AppData\Roaming\Microsoft\Yyiclmfvu\qyvyla.dll		SUCCESS
10:40:...	explorer.exe	6920	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run		SUCCESS
10:40:	explorer.exe	6920	RegQueryKey	HKCU	Query: HandleTags, HandleTags: 0x0	SUCCESS

This way QakBot's persistence can not be detected at runtime.

Egregor

While we have previously reported on QakBot deliverying the ProLock ransomware,[2] latests reports indicated that QakBot is now used to deliver the Egregor ransomware. We previously reported on the Egregor ransomware as part of an article on ransomware leaksites[6] in which we explain the practice of ransomware operators stealing their victims data before encrypting it to extort them not only with decryption but also public release of the stolen data.

Conclusion and Countermeasures

From our analysis we can conclude that QakBot is trying to avoid persistent file artifacts. In previous version the configuration and QakBot executable were permanently stored on disk. This made it easy for security tools to detect them. The new version tries to avoid permanently leaving its artifacts on disk. While QakBot is not going fully fileless, it new tactics will sure lower its detection.

But even though QakBot has changed, the delivery mechanism behind the QakBot " `abc[A-Z]+` " campaign did not. Hence, an infection by this threat actor can be successfully prevented by blocking the initial emails.

Hornetsecurity's Spam Filter and Malware Protection, with the highest detection rates on the market, already detects and blocks the outlined threat. Hornetsecurity's Advanced Threat Protection extends this protection by also detecting yet unknown threats.

References

Indicators of Compromise (IOCs)

Hashes

The hashes of the analyzed QakBot samples are:

MD5	Filename	Description
`6bc0584f6cbb74714add1718b0322655`	`904400.jpg`	QakBot DLL as downloaded by XLM macro
`e23bc27212f61520cfb130185d74cfb1`	`26e0000.dll`	Extracted QakBot DLL

MITRE ATT&CK

The tactics and techniques used by QakBot as defined by the MITRE ATT&CK framework are as follows:

Tactic	Technique
TA0001 – Initial Access	T1566.001 – Phishing: Spearphishing Attachment
TA0001 – Initial Access	T1566.002 – Phishing: Spearphishing Link
TA0002 – Execution	T1027 – Obfuscated Files or Information
TA0002 – Execution	T1204.002 – User Execution: Malicious File
TA0003 – Persistence	T1547.001 – Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
TA0004 – Privilege Escalation	T1053.005 – Scheduled Task/Job: Scheduled Task
TA0005 – Defense Evasion	T1027.002 – Obfuscated Files or Information: Software Packing
TA0005 – Defense Evasion	T1055 – Process Injection
TA0005 – Defense Evasion	T1055.012 – Process Injection: Process Hollowing
TA0005 – Defense Evasion	T1070 – Indicator Removal on Host
TA0005 – Defense Evasion	T1497.001 – Virtualization/Sandbox Evasion: System Checks
TA0006 – Credential Access	T1003 – OS Credential Dumping
TA0006 – Credential Access	T1110.001 – Brute Force: Password Guessing

Tactic	Technique
TA0006 – Credential Access	T1555.003 – Credentials from Password Stores: Credentials from Web Browsers
TA0011 – Command and Control	T1071.001 – Application Layer Protocol: Web Protocols
TA0011 – Command and Control	T1090 – Proxy
TA0011 – Command and Control	T1090.002 – Proxy: External Proxy

Appendix

Qakbot configuration extraction Python3 script

````
import sys
import pefile
from arc4 import ARC4

pe = pefile.PE(sys.argv[1])
c2list = []
for entry in pe.DIRECTORY_ENTRY_RESOURCE.entries:
    for e in entry.directory.entries:
        n = e.name.string.decode()
        data = pe.get_data(e.directory.entries[0].data.struct.OffsetToData,
e.directory.entries[0].data.struct.Size)
        data = ARC4(data[:20]).decrypt(data[20:])[20:]
        if n == '311':
            for i in range(1,len(data),7):
                c2 = list(data[i:i+6])
                c2list.append("%d.%d.%d.%d:%d" % (c2[0],c2[1],c2[2],c2[3],(c2[4]
<<8)+c2[5]))
        elif n == '308':
            config = data.decode().split()
print("# QakBot Config\n\n```\n" + "\n".join(config) + "\n```\n")
print("# QakBot C2\n\n```\n" + "\n".join(c2list) + "\n```\n")
````