

Automating Qakbot decode at scale

 docs.velociraptor.app/blog/2023/2023-04-05-qakbot/

This is a technical post covering practical methodology to extract configuration data from recent Qakbot samples. In this blog, I will provide some background on Qakbot, then walk through decode themes in an easy to visualize manner. I will then share a Velociraptor artifact to detect and automate the decode process at scale.

Matt Green - @mgreen27 2023-04-05



Qak!

Qakbot or QBot, is a modular malware first observed in 2007 that has been historically known as a banking Trojan. Qbot is used to steal credentials, financial, or other endpoint data, and in recent years, regularly a loader for other malware leading to hands on keyboard ransomware.

Typical delivery includes malicious emails as a zipped attachment, LNK, Javascript, Documents, or an embedded executable. The example shown in this post was delivered by an email with an attached pdf file:



An example Qakbot infection chain

Qakbot has some notable defense evasion capabilities including:

1. Checking for Windows Defender sandbox and terminating on discovery.
2. Checking for the presence of running anti-virus or analysis tools, then modifying its later stage behavior for evasion.
3. Dynamic corruption of payload on startup and rewrite on system shutdown.

Due to the commodity nature of delivery, capabilities and end game, it is worth extracting configuration from observed samples to scope impact from a given campaign. Hunting enterprise wide and finding a previously missed machine or discovering an ineffective control can be the difference in preventing a domain wide ransomware event, or a similar really bad day.

Configuration

Qakbot has an RC4 encoded configuration, located inside two resources of the unpacked payload binary. The decryption process has not changed significantly in recent times, but for some minor key changes. It uses a SHA1 of a hard coded key that can typically be extracted as an encoded string in the .data section of the payload binary. This key often remains static across campaigns, which can speed up analysis with the maintenance of a recent key list.

Current samples undergo two rounds of RC4 decryption with validation built in. The validation bytes dropped from the data for the second round.

After the first round:

- The first 20 bytes in hex is for validation and is compared with the SHA1 of the remaining decoded data
- Bytes [20:40] is the key used for the second round of decoding
- The Data to decode is byte [40:] onwards
- The same validation process occurs for the second round decoded data
 - Verification = data[:20]
 - DecodedData = data[20:]

type (1)	name	file-offset (2)	signature (1)	size (1103 bytes)
bitmap	COMPONENT_07	0x000200BC	bitmap	83
bitmap	COMPONENT_08	0x00020110	bitmap	1020



Key:
bUdiuy81gYguty@4frdRdpfko(e
KnudeuMncueaN

RC4

Passphrase
SHA1(KEY) HEX ▾

Input format: Latin1 Output format: Latin1

RC4 passphrase:
12bd42d0941c69ecc4e8075ccf
3e9f202c1a9412



Start of second round
RC4 encoded data

Verification hash	Second round RC4 key
9B EB 36 A5 D7 DF 24 CB 71 61 6A A9 BC A5 35 AE B3 8C 04 28	8E D1 05 A6 3D E4 56 FC 30 F8 65 47 57 AE B2 32 E0 04 68 50
00 5E 59 05 F2 05 0F 39 24 8F 97 88 FD C6 23 7C 71 65 40 43 56 68 FF 53 2A 77 F8 50 BF FC 66 D6 70 64 DF D4 D0 03 0A C0 FD 5C AE EB 48 F5 3F C9	2C 23 39 58 7A 94 0F 34
EB C3 E7 EF A8 98 1F A3 71 D3 BC 03 18 F5 2D E6 4F 50 A3 CB 8E 68 84 55 8F ED 7A 15 E9 FD E9 BA 2D 6C F2 6F 70 D3 09 EC B1 50 D8 4E 5C 42 58 22	EB 48 F5 3F C9
FB C7 EC CA 6E A4 F4 C9 4A DE 16 6F 62 F2 50 8C 21 19 71 EA 74 2A 15 3E 40 8B 17 C6 B4 89 17 98 BF 63 CF F7 48 E6 BF E7 E8 58 E2 1F 92 91 BF 03	EB 48 F5 3F C9
FB AC 85 58 F2 F5 BF A1 6F 8C 25 8D D6 5A 84 F2 56 86 48 79 3A 62 8B 03 32 BE C4 0F 00 67 A2 18 88 BF D1 23 57 71 03 48 CB 6F 95 ED 57 67 EC 54	EB 48 F5 3F C9
00 AE 3F EF 75 FF 4A 75 1F 82 78 00 ED 88 FC A1 58 91 B4 81 CC C1 F5 F1 55 A3 4A 8A 65 44 C4 C0 87 F1 30 54 11 F8 1D 58 FC 82 3A C7 09 41 99 79	EB 48 F5 3F C9
9A A9 39 4F 36 D4 FE 6F 59 86 5F 66 FC D1 4F E3 28 23 C8 C2 DF 99 39 98 2D 50 F6 96 2F 1F 4A F4 1C 17 CC 08 71 D5 74 84 0A 40 38 03 02 CB F2 47	EB 48 F5 3F C9
74 73 E0 80 82 D8 DF 54 B5 10 A7 2D 01 EE 6F 16 2E A6 7F 6E 47 CE C4 EA 7A 4A 98 A3 4E A6 B6 91 99 47 8C CC 63 5F FB 6C 7B 15 69 40 68 8B 28 FF	EB 48 F5 3F C9
0A 49 53 A0 7E E7 CD 7A 31 93 72 D1 9A 48 00 4F D0 30 2C 5E 1D 50 20 C8 03 4A 84 55 F8 AE 38 68 9F 20 00 B6 D6 C2 6F A3 F8 13 EC D0 10 07 73 F8	EB 48 F5 3F C9
3F 0F 01 77 CF A5 05 00 DF 3F 4E CD C8 08 D0 51 D1 00 98 14 AC 88 08 B3 24 81 B8 9A C4 1E 65 B2 8A 3A 7B A5 7E 9E BE 53 B2 09 A1 06 10 06 58 03	EB 48 F5 3F C9
3B 26 BA 15 A6 41 97 E6 01 13 79 22 8E 9E 32 96 35 AC 4E 8A 84 58 B1 28 E0 86 C2 EB 0E 92 BC 36 F1 A8 51 BC 58 F2 68 0C 70 36 EB 31 63 14 31 84	EB 48 F5 3F C9
3B A5 A3 91 DE 29 78 EB D8 48 56 BE 7A E3 57 48 3F 03 A5 89 51 EA AA 79 EC 89 44 1F 4F D1 4E B1 68 E7 54 62 BF 01 75 76 BC 69 2D D0 C9 A3 A9 92	EB 48 F5 3F C9
A5 4A E2 F6 A2 35 38 08 88 99 41 15 F3 D9 43 3E EF E1 30 FD 78 D1 E6 B7 07 31 43 76 B8 CB C3 98 4F 78 98 F6 B6 92 BF C6 AD CF ED 85 68 B6 87 7A	EB 48 F5 3F C9
4E 0C 44 64 19 0E 03 49 AD 46 59 82 80 F4 C0 CF 46 C6 12 76 49 88 0A D4 3F B4 F1 3D 20 76 71 54 C8 F1 FD 99 AF 49 08 58 B0 2F 99 A2 C9 EA A5 35	EB 48 F5 3F C9
2B 82 BF 51 F6 33 48 25 40 84 4E 88 8C B3 26 08 EF F8 41 DC 33 47 EB C9 28 28 C7 AA F3 19 F5 E1 2A A4 84 AC 3C 1A 16 CA CD D6 DF 60 76 2D F3 B2	EB 48 F5 3F C9
3B 05 17 55 D8 BC 88 03 65 01 D0 C1 52 AA 09 C1 20 45 8E A5 30 49 CF CF E4 77 AB C5 95 84 25 23 25 BE 23 AF D0 05 FC 42 A5 62 A2 A1 7C BE DE 0F	EB 48 F5 3F C9
3E 51 D3 15 51 E5 F1 48 A3 29 4A F7 2A 28 21 68 32 59 42 86 06 1E B3 5C 88 C8 15 34 55 22 5C 00 58 EF CB 98 D0 00 3A 89 B8 6C 04 A7 9C 8F 66 B2	EB 48 F5 3F C9
3B 13 96 94 D0 14 2A BA C9 2E C1 11 F1 F9 F3 2A EF 74 85 C6 BE A3 7C 81 98 B0 A0 68 D2 A5 C0 0F D3 07 47 D1 D7 31 86 94 6A 62 E0 22 70 BE AB	EB 48 F5 3F C9
4C 95 A5 A6 38 06 A6 D3 84 F3 E4 85 C3 F6 EF 3D 2A 23 3A EC 58 25 D5 AE 17 32 1A CC 8B 95 E0 E8 94 00 51 94 C5 CC 8A CD 8E BE E3 FD 32 18 D5 39	EB 48 F5 3F C9
95 0F 09 24 15 64 30 69 85 9F 2D 82 88 E7 58 B0 87 9A 13 84 20 81 19 4D 56 A2 47 14 C0 4D 90 FA E9 07 F9 F9 46 A2 86 C5 8F 88 58 C0 D7 BF 8D 3D	EB 48 F5 3F C9
F8 2F 10 49 E0 D8 AF AB AB 1F 78 D0 71 07 5E A1 5F F1 28 A5 88 D0 42 BE 30 C0 9F 6C BA 20 A0 BE 8A D7 49 E4 5C 9A 2F AE B1 66 D9 EB D1 EA B3 8B	EB 48 F5 3F C9
A2 62 9C EB 4F 08 B0 33 C0 4D B7 58	EB 48 F5 3F C9

Data to SHA1 to match verification

First round of Qakbot decode and verification

Campaign information is located inside the smaller resource where, after this decoding and verification process, data is clear text.

```
10=BB16
3=1677046917
```

Decoded campaign information

The larger resource stores Command and Control configuration. This is typically stored in *netaddress format* with varying separators. A common technique for finding the correct method is searching for common ports and separator patterns in the decoded data.

01 bb 443 next previous all match case regex by word

Replace replace replace all

IP Port Seperator

01	2f	15	33	8a	01	bb	01	01	48	50	07	06	c3	53	01	01	52	7f	cc	52	08	ae	01	01	31	af	48	38	01	bb	01	01	c9	f4	6c	b7	03	e3	01	01	7a	b8	8f	52	01	bb	01
01	66	9c	fd	56	01	bb	00	01	4a	3a	47	ed	01	bb	00	01	2f	15	33	8a	03	e3	01	01	4d	56	62	ec	01	bb	01	01	47	1f	65	b7	01	bb	01	01	88	e8	b8	86	03	e3	01
01	56	e1	d6	8a	08	ae	01	01	5f	f2	65	fb	03	e3	00	01	6d	0b	af	2a	08	ae	01	01	5a	4e	8a	d9	08	ae	01	01	b8	b0	23	df	08	ae	01	01	23	8f	61	91	03	e3	01
01	ca	ba	b1	58	01	bb	01	01	72	4f	b4	0e	03	e3	01	01	56	96	2f	db	01	bb	00	01	b7	57	a3	a5	01	bb	01	01	32	44	ba	c3	01	bb	01	01	be	4b	5f	a4	08	ae	00
01	62	91	17	43	01	bb	01	01	43	0a	af	2f	08	ae	01	01	47	d4	93	e0	08	ae	00	01	58	7e	5e	04	c3	50	01	01	67	8c	ae	13	08	ae	00	01	67	e7	d8	ee	01	bb	01
01	4e	54	7b	ed	03	e3	00	01	b4	97	6c	0e	01	bb	01	01	50	2f	39	83	08	ae	00	01	c6	02	33	f2	03	e1	01	01	32	44	cc	47	03	e3	01	01	cd	a4	e3	de	01	bb	01
01	93	db	04	c2	01	bb	01	01	4d	7c	06	95	01	bb	01	01	31	f5	52	b2	08	ae	01	01	2e	0a	c6	6b	01	bb	00	01	4c	50	b4	9a	03	e3	01	01	0c	ac	ad	52	7d	65	01
01	44	96	12	a1	01	bb	01	01	44	ad	aa	6e	20	fb	00	01	18	09	dc	a7	01	bb	01	01	0c	ac	ad	52	08	27	01	01	32	44	cc	47	03	e1	01	01	6b	92	0c	1a	08	ae	01
01	51	e5	75	5f	08	ae	01	01	1b	00	30	e9	01	bb	01	01	45	85	a2	23	01	bb	01	01	3b	1c	54	41	01	bb	01	01	4c	aa	fc	99	03	e3	01	01	59	20	9f	c0	03	e3	01
01	ca	8e	62	3e	03	e3	01	01	49	4e	d7	68	01	bb	01	01	b5	a4	d9	d3	01	bb	01	01	5c	61	cb	33	08	ae	00	01	74	4a	a4	1a	01	bb	00	01	67	8d	32	66	03	e3	01
01	95	4a	9f	43	08	ae	01	01	74	48	fa	12	01	bb	01	01	7d	63	45	b2	01	bb	01	01	ca	8e	62	3e	01	bb	01	01	43	3d	47	c9	01	bb	01	01	67	7b	df	a8	01	bb	00
01	50	0d	cd	45	08	ae	00	01	50	00	4a	a5	01	bb	01	01	56	63	36	27	08	ae	00	01	d5	43	ff	39	08	ae	01	01	b0	8e	cf	3f	01	bb	01	01	32	43	11	5c	01	bb	01
01	d9	a5	01	35	08	ae	01	01	46	40	4d	73	01	bb	01	01	02	32	2f	4a	01	bb	01	01	42	bf	45	12	03	e3	01	01	4b	8f	ec	95	01	bb	01	01	c5	5c	88	7a	01	bb	01
01	6c	be	cb	2a	03	e3	01	01	32	44	cc	47	01	bb	01	01	0c	ac	ad	52	03	e3	01	01	46	4d	74	e9	01	bb	01	01	a2	f8	0e	6b	01	bb	01	01	4b	62	9a	13	01	bb	01
01	3a	f7	73	7e	03	e3	01	01	b8	44	74	92	ef	12	01	01	29	63	32	4c	01	bb	00	01	b8	44	74	92	0d	3d	01	01	48	cb	d8	62	08	ae	01	01	67	fc	07	e7	01	bb	01
01	0c	ac	ad	52	c3	51	01	01	46	a0	50	d2	01	bb	01	01	0c	ac	ad	52	01	d1	01	01	0c	ac	ad	52	00	15	01	01	2f	22	1e	85	01	bb	01	01	ca	bb	e8	a1	03	e3	01
01	62	93	9b	eb	01	bb	01	01	7c	7a	38	90	01	bb	01	01	4b	8d	e3	a9	01	bb	01	01	67	90	c9	35	08	1e	01	01	ac	f8	2a	7a	01	bb	01	01	0c	ac	ad	52	03	de	01
01	18	ef	45	f4	01	bb	01	01	ad	12	7e	03	01	bb	01	01	49	a5	77	14	01	bb	01	01	0c	ac	ad	52	03	e3	01	01	5a	68	16	1c	08	ae	01	01	0e	c0	f1	4c	03	e3	00
01	4a	21	c4	72	01	bb	01	01	4a	5d	94	61	03	e3	01	01	56	c4	30	8e	08	ae	01	01	ae	68	b8	95	01	bb	01	01	0c	ac	ad	52	00	14	01	01	6d	97	90	25	01	bb	00
01	68	23	18	9a	01	bb	01	01	72	8f	b0	ea	01	bb	01	01	54	23	1a	0e	03	e3	01	01	2d	32	e9	d6	01	bb	01	01	40	ed	b9	3c	01	bb	01	01	49	a1	b0	da	01	bb	01

Easy to spot C2 patterns: port 443

Encoded strings

Qakbot stores blobs of xor encoded strings inside the .data section of its payload binary. The current methodology is to extract blobs of key and data from the referenced key offset which similarly is reused across samples.

Current samples start at offset 0x50, with an xor key, followed by a separator of 0x0000 before encoded data. In recent samples I have observed more than one string blob and these have occurred in the same format after the separator.


```

0 : "netstat -nao"
1 : "Component_07"
2 : "cmd.exe /c set"
3 : "Component_08"
4 : "%s \"%s = \\\"%s\\\\; & %s\"""
5 : "net share"
6 : "c:\ProgramData"
7 : "SELF_TEST_1"
8 : "Microsoft"
9 : "schtasks.exe /Create /RU "NT AUTHORITY\SYSTEM" /SC ONSTART /TH %u /TR "%s" /NP /F"
10 : "Self test FAILED!!!"
11 : "net localgroup"
12 : "Self check"
13 : "whoami /all"
14 : "ipconfig /all"
15 : "ERROR: GetModuleFileNameW() failed with error: ERROR_INSUFFICIENT_BUFFER"
16 : "runas"
17 : "powershell.exe -encodedCommand %S"
18 : "ProfileImagePath"
19 : "microsoft.com,google.com,cisco.com,oracle.com,verisign.com,broadcom.com,yahoo.com,xfinity.com,irs.gov,linkedin.com"
20 : "ERROR: GetModuleFileNameW() failed with error: %u"
21 : "SoNuce]ugdIB3c[doMuce2s81*uXncvP"
22 : "\System32\WindowsPowerShell\v1.0\powershell.exe"
23 : "SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList"
24 : "anstream.dll"
25 : "bUdiuy81gYguty@4FrdRdpFko{eKnudeuMncueaN"
26 : "\"%s\system32\schtasks.exe" /Create /ST %02u:%02u /RU "NT AUTHORITY\SYSTEM" /SC ONCE /tr "%s" /Z /ET %02u:%02u /tn %s"

```

Decoded strings: RC4 key highlighted

Payload

Qakbot samples are typically packed and need execution or manual unpacking to retrieve the payload for analysis. Its very difficult to obtain this payload remotely at scale, in practice the easiest way is to execute the sample in a VM or sandbox that enables extracting the payload with correct PE offsets.

When executing locally Qakbot typically injects its payload into a Windows process, and can be detected with yara targeting the process for an unbacked section with `PAGE_EXECUTE_READWRITE` protections.

Below is an example of running PE-Sieve / Hollows Hunter tool from Hasherezade. This helpful tool enables detection of several types of process injection, and the dumping of injected sections with appropriately aligned headers. In this case, the injected process is `wermgr.exe` but it's worth to note, depending on variant and process footprint, your injected process may vary.

```

C:\Users\REM\Desktop>pe-sieve64.exe /pid 39092
PID: 39092
Output filter: no filter: dump everything (default)
Dump mode: autodetect (default)
[*] Using raw process!
[*] Scanning: C:\Windows\SysWOW64\wermgr.exe
Scanning workingset: 345 memory regions.
[*] Workingset scanned in 156 ms
[+] Report dumped to: process_39092
[*] Dumped module to: C:\Users\REM\Desktop\\process_39092\a60000.wermgr.exe as UNMAPPED
[*] Dumped module to: C:\Users\REM\Desktop\\process_39092\120000.dll as REALIGNED
[+] Dumped modified to: process_39092
[+] Report dumped to: process_39092
---
PID: 39092
---
SUMMARY:
Total scanned:      57
Skipped:            0
-
Hooked:             1
Replaced:           0
Hdrs Modified:     0
IAT Hooks:          0
Implanted:          1
Implanted PE:       1
Implanted shc:      0
Unreachable files: 0
Other:              0
-
Total suspicious:  2
---

```

Dumping Qakbot payload using pe-sieve

Doing it at scale

Now I have explained the decode process, time to enable both detection and decode automation in Velociraptor.

I have recently released [Windows.Carving.Qakbot](#) which leverages a PE dump capability in Velociraptor 0.6.8 to enable live memory analysis. The goal of the artifact was to automate my decoding workflow for a generic Qakbot parser and save time for a common analysis. I also wanted an easy to update parser to add additional keys or decode nuances when changes are discovered.

TargetGlob	Glob to target payloads on disk															
PidRegex	.															
ProcessRegex	? for suggestions															
StringOffset	0x50															
ResourceRegex	BITMAP RCDATA															
Keys	<table border="1"> <thead> <tr> <th></th> <th>Type</th> <th>Key</th> </tr> </thead> <tbody> <tr> <td>+ </td> <td>double</td> <td>Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd</td> </tr> <tr> <td>+ </td> <td>double</td> <td>bUdiuy81gYguty@4frdRdpfko(eKmudeuMncueaN</td> </tr> <tr> <td>+ </td> <td>single</td> <td>\System32\WindowsPowerShell\v1.0\powershell.exe</td> </tr> <tr> <td>+ </td> <td>single</td> <td>\System32\WindowsPowerShell\v1.0\powershell.exe</td> </tr> </tbody> </table>		Type	Key	+	double	Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd	+	double	bUdiuy81gYguty@4frdRdpfko(eKmudeuMncueaN	+	single	\System32\WindowsPowerShell\v1.0\powershell.exe	+	single	\System32\WindowsPowerShell\v1.0\powershell.exe
	Type	Key														
+	double	Muhcu#YgcdXubYBu2@2ub4fbUhuiNhyVtcd														
+	double	bUdiuy81gYguty@4frdRdpfko(eKmudeuMncueaN														
+	single	\System32\WindowsPowerShell\v1.0\powershell.exe														
+	single	\System32\WindowsPowerShell\v1.0\powershell.exe														

Windows.Carving.Qakbot: parameters

This artifact uses Yara to detect an injected Qakbot payload, then attempts to parse the payload configuration and strings. Some of the features in the artifact cover changes observed in the past in the decryption process to allow a simplified extraction workflow:

- Automatic PE extraction and offset alignment for memory detections.
- **StringOffset** - the offset of the string xor key and encoded strings is reused regularly.
- PE resource type: the RC4 encoded configuration is typically inside 2 resources, I've observed **BITMAP** and **RCDATA**
- Unescaped key string: this field is typically reused over samples.
- Type of encoding: single or double, double being the more recent.
- Hidden **TargetBytes** parameter to enable piping payload in for analysis.
- Worker threads: for bulk analysis / research use cases.

ProcessInfo	DecodedStrings	Campaign	C2
<pre>{ "ProcessCreateTime": "2023-03-06T00:00:26.7481723Z" "Pid": 39892 "ProcessName": "wermgr.exe" "Exe": "C:\Windows\SysWOW64\wermgr.exe" "CommandLine": "C:\WINDOWS\SysWOW64\wermgr.exe" "Username": "DESKTOP-2C3IQH0\REM" "Offset": 1179648 "PayloadSize": 143368 }</pre>	<pre>0 : [...] 1 : [0 - 100] 100 : "%u.%u.%u.%u.%u.%04x" 101 : "Win32_Product" 102 : "%SystemRoot%\System32\wermgr.exe" 103 : "fmon.exe" 104 : "LocalLow" 105 : "%SystemRoot%\SysWOW64\msra.exe" 106 : "Packages" 107 : "WBJ_IGNORE" 108 : "%SystemRoot%\SysWOW64\SearchIndexer.exe" 109 : "image/jpeg" 110 : "t=%s time=[%02d:%02d:%02d-%02d/%d]" 111 : "ccSvcHst.exe;NortonSecurity.exe;nsWscSvc.exe" 112 : "SpyNetReporting" 113 : "CSFalconService.exe;CSFalconContainer.exe" 114 : "%SystemRoot%\SysWOW64\grpconv.exe" 115 : "S:(ML;NW;;;LW)" 116 : "%SystemRoot%\explorer.exe" 117 : "Name" 118 : "Caption,Description,DeviceID,Manufacturer,Name,PNPDeviceID,Service,Status" 119 : "APPDATA" 120 : "MsMpEng.exe" 121 : "%SystemRoot%\System32\OneDriveSetup.exe" 122 : "urlmon.dll" 123 : "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"</pre>	<pre>{ "Timestamp": "2023-02-22T06:21:57Z" "Name": "BB16" }</pre>	<pre>0 : "47.21.51.138:443" 1 : "72.80.7.6:50003" 2 : "82.127.204.82:222" 2 : "49.175.72.56:443" 4 : "201.244.108.183:995" 5 : "122.184.143.82:443" 3 : "102.156.253.86:443" 3 : "74.58.71.237:443" 7 : "47.21.51.138:995" 8 : "77.86.98.236:443" 9 : "71.31.101.183:443" 10 : "136.232.184.134:995" 11 : "86.225.214.138:222" 12 : "86.225.214.138:222"</pre>

Windows.Carving.Qakbot: live decode

Research

The Qakbot parser can also be leveraged for research and run bulk analysis. One caveat is the content requires payload files that have been dumped with offsets intact. This typically requires some post collection filtering or PE offset realignment but enables Velociraptor notebook to manipulate post processed data.

Some techniques I have used to bulk collect samples:

- Sandbox with PE dumping features: api based collection
- Virustotal search: `crowdsourced_yara_rule:0083a00b09|win_qakbot_auto AND tag:pedll AND NOT tag:corrupt` (note: this will collect some broken payloads)

FirstCampaignTime	LastCampaignTime	IP	CampaignNames	Ports	Total
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	12.172.173.82	▶ [...]	▶ [...]	482
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	50.68.204.71	▶ [...]	▶ [...]	161
2023-02-01T08:41:31Z	2023-02-09T09:10:35Z	24.64.112.40	▼ [0 : "BB12" 1 : "BB14" 2 : "obama235" 3 : "obama236" 4 : "obama239"]	▶ [...]	80
2023-01-31T10:31:56Z	2023-03-23T07:33:58Z	202.142.98.62	▶ [...]	▶ [...]	77
2023-01-31T10:31:56Z	2023-03-09T07:14:51Z	47.21.51.138	▶ [...]	▼ [0 : "443" 1 : "995"]	68
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	174.104.184.149	▶ [...]	▶ [...]	59
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	81.229.117.95	▶ [...]	▶ [...]	58
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	75.143.236.149	▶ [...]	▶ [...]	58
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	90.104.22.28	▶ [...]	▶ [...]	55
2022-12-13T07:59:10Z	2023-03-15T14:19:18Z	72.80.7.6	▶ [...]	▶ [...]	55
2022-11-30T07:40:48Z	2023-03-23T07:33:58Z	47.34.30.133	▶ [...]	▶ [...]	54
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	98.145.23.67	▶ [...]	▶ [...]	54
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	86.225.214.138	▶ [...]	▶ [...]	53
2022-11-28T09:42:44Z	2023-03-23T07:33:58Z	76.80.180.154	▶ [...]	▶ [...]	53

Bulk collection: IPs seen across multiple campaign names and ports

Some findings from a small data set ~60 samples:

- Named campaigns are typically short and not longer than a few samples over a few days.
- IP addresses are regularly reused and shared across campaigns
- Most prevalent campaigns are **BB** and **obama** prefixed
- Minor campaigns observed: **azd**, **tok** and **rds** with only one or two observed payload samples each.

Strings analysis can also provide insights to sample behavior over time to assist analysis. A great example is the adding to process name list for anti-analysis checks.

EarliestCampaignTime	LatestCampaignTime	String
2022-11-28T09:42:44Z	2023-02-22T06:21:57Z	frida-wininjector-helper-32.exe;frida-wininjector-helper-64.exe;tcpdump.exe;windump.exe;ethereal.exe;wireshark.exe;ettercap.exe;rtsniff.exe;packetcapture.exe;capturenet.exe;qak_proxy;dumpcap.exe;CFF Explorer.exe;not_rundll32.exe;ProcessHacker.exe;tcpview.exe;filemon.exe;procmmon.exe;idaq64.exe;loaddll32.exe;PETools.exe;ImportREC.exe;LordPE.exe;SysInspector.exe;proc_analyzer.exe;sysAnalyzer.exe;sniff_hit.exe;joeboxcontrol.exe;joeboxserver.exe;ResourceHacker.exe;x64dbg.exe;Fiddler.exe;sniff_hit.exe;sysAnalyzer.exe
2023-02-27T09:37:23Z	2023-03-23T07:33:58Z	frida-wininjector-helper-32.exe;frida-wininjector-helper-64.exe;tcpdump.exe;windump.exe;ethereal.exe;wireshark.exe;ettercap.exe;rtsniff.exe;packetcapture.exe;capturenet.exe;qak_proxy;dumpcap.exe;CFF Explorer.exe;not_rundll32.exe;ProcessHacker.exe;tcpview.exe;filemon.exe;procmmon.exe;idaq64.exe;loaddll32.exe;PETools.exe;ImportREC.exe;LordPE.exe;SysInspector.exe;proc_analyzer.exe;sysAnalyzer.exe;sniff_hit.exe;joeboxcontrol.exe;joeboxserver.exe;ResourceHacker.exe;x64dbg.exe;Fiddler.exe;sniff_hit.exe;sysAnalyzer.exe;BehaviorDumper.exe;processdumperx64.exe;anti-virus.EXE;sysinfoX64.exe;scotoolswrapper.exe;sysinfoX64.exe;FakeExplorer.exe;apimonitor-x86.exe;idaq.exe

Bulk collection: Strings highlighting anti-analysis check additions over time

Conclusion

During this post I have explained the Qakbot decoding process and introduced an exciting new feature in Velociraptor. PE dumping is a useful capability and enables advanced capability at enterprise scale, not even available in expensive paid tools. For widespread threats like Qakbot, this kind of content can significantly improve response for the blue team, or even provide insights into threats when analyzed in bulk. In the coming months the Velociraptor team will be publishing a series of similar blog posts, offering a sneak peek at some of the types of memory analysis enabled by Velociraptor and incorporated into our training courses.

I also would like to thank some of Rapid7's great analysts - [Jakob Denlinger](#) and [James Dunne](#) for bouncing some ideas when writing this post.

References