



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.

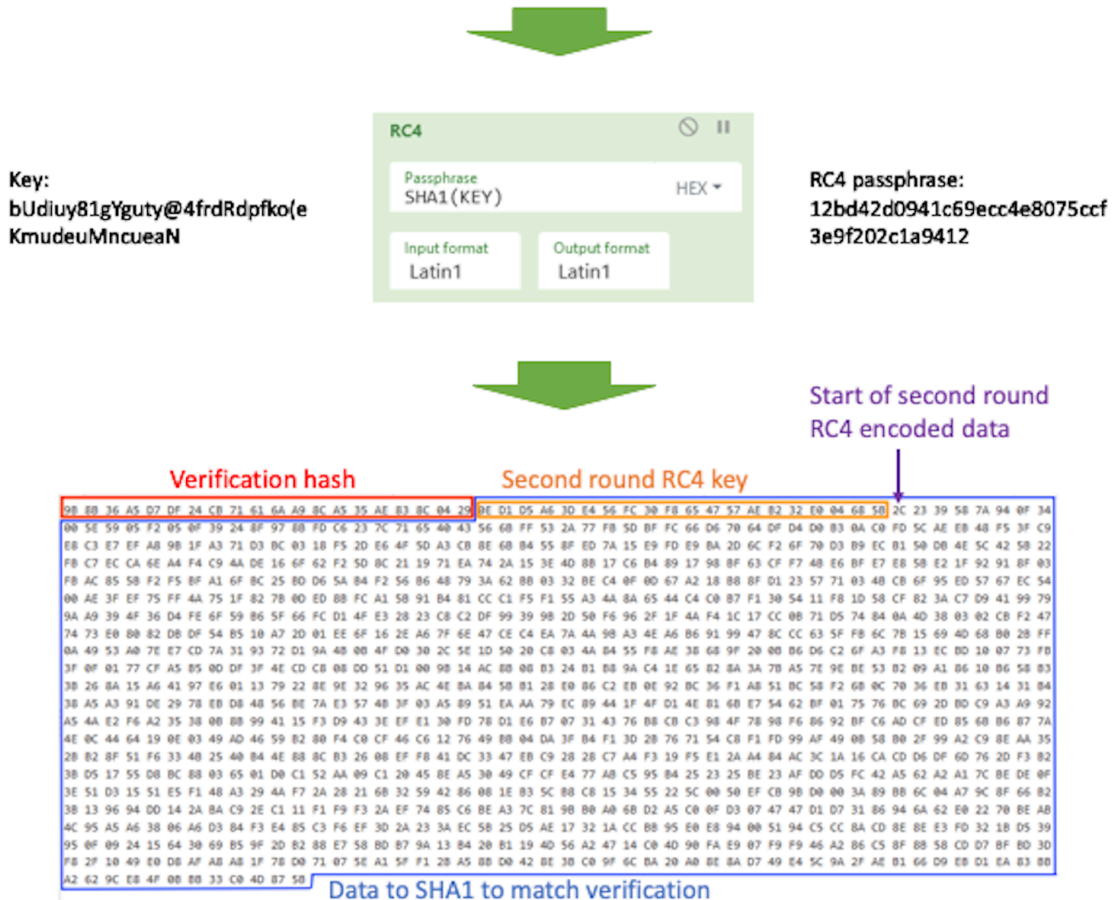
Current samples undergo two rounds of RC4 decryption with validation built in. The validation bytes dropped from the data for the second round.

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



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
    
```

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  regexp  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	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	ca	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.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text	
00000000	04	04	00	00	09	00	00	00	05	00	00	00	FF	FF	FF	FF	.....ÿÿÿÿ	
00000010	06	00	00	00	FF	FF	FF	FF	07	00	00	00	FF	FF	FF	FF	....ÿÿÿÿ...ÿÿÿÿ	
00000020	9F	45	19	00	99	45	19	00	76	98	00	00	00	00	00	00	ÿE..™E..v".....	
00000030	DF	B0	08	99	00	00	00	00	00	00	00	00	00	00	00	00	B°.™.....	
00000040	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
00000050	CD	BC	EC	F3	C1	83	F4	4D	C2	46	4C	3F	9D	F9	74	F5	ÍáíóÁfóMÁFL?.ùtõ	
Start offset	CD	C4	94	20	AF	08	D8	41	40	E1	69	í9	D6	2C	14	53	íÄ" -.ØA@áíUÖ,.S	
00000070	09	A8	B1	5B	CF	<b>XOR key 1</b>						1F	5A	44	52	FB	03	.`±[İc0...&ZDRû.
00000080	06	5A	3F	62	5F	43	25	D5	50	65	31	54	C7	54	1A	F3	.Z?b C%ÓPeItÇT.ó	
00000090	34	4B	6B	8D	C2	32	06	DE	11	63	5F	58	C3	9D	A1	16	4Kk.Ā2.Ē.c XĀ.i.	
000000A0	F9	BA	A4	40	9B	9C	F8	E8	B3	42	45	BE	14	06	20	25	ù°µ@)ææè³BE%.. %	
000000B0	C2	AF	43	FF	C0	8F	F7	EC	58	33	CE	7D	6F	FC	50	D0	Ā`CÿĀ.÷iX3İ)ouPD	
000000C0	C2	33	D1	FA	C0	8E	C4	37	A8	28	5E	A1	09	EA	18	1C	Ā3ÑúĀŽĀ7" (^; .è..	
Separator	00	00	00	00	00	00	00	00	A3	D9	98	80	B5	E2	80	6D	.....èÛ`εµáεm	
000000E0	EF	28	2D	50	9D	BA	1B	98	BD	AB	FA	45	C1	7C	87	71	ī (-P.°.™«úEĀ †q	
000000F0	77	E1	0A	B4	B2	02	71	2B	6C	88	9E	38	EF	10	55	7F	wá.'².q+l`ž8ī.U.	
00000100	90	65	70	37	<b>Encoded strings 1</b>						67	43	00	A6	.ep74=°fh.`RgC.;			
00000110	70	39	13	70	E2	27	3A	CE	14	17	37	D1	E0	17	75	82	p9.pā':İ.7Ñā.u,	
00000120	4D	3F	03	63	E3	BB	81	32	DC	C9	F8	62	9B	F2	9D	9C	M?.cā».2ÜÉøb`ò.æ	
00000130	93	31	2D	DF	66	63	20	46	F8	F3	13	8D	AF	E8	85	8D	"l-βfc Føó..`è....	
00000140	35	77	AF	09	0E	FC	03	95	8E	75	8E	AE	85	DD	90	68	5w`..ü.°ŽuŽø...ÿ.h	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
00000690	DA	CE	7F	79	2B	61	4B	7D	F1	96	9E	1C	3E	ED	04	BA	Úİ.y+aK)ñ-ž.>i.°	
000006A0	9F	70	25	11	A9	87	90	C2	2B	08	5B	1A	67	2F	FE	00	ÿp%.@†.Ā+. [g/p.	
000006B0	E3	25	7B	9F	33	F6	FE	33	33	29	07	81	27	39	27	B0	ā%(ÿ3øp33)..`9'°	
000006C0	1B	2D	4F	BF	<b>Encoded strings 1</b>						69	83	4E	46	.-Oç<-Sý.+İ»ifNF			
000006D0	1E	D7	DD	52	E6	1B	DB	71	C3	21	E1	DB	6A	A9	B6	66	.×ÿRæ.ÛqĀ!áÛj@qf	
000006E0	32	1C	60	97	E0	F5	5B	44	00	00	00	00	00	00	00	00	Separator).....	
000006F0	08	33	FF	44	C8	4C	78	DA	6D	C1	F7	B3	F8	EA	90	29	.3ÿDÈLxÚmĀ=³øè.)	
00000700	DD	41	D6	CD	FD	61	01	AB	9D	FF	DF	A7	73	AB	32	4A	ÝĀÖİýā.«.ÿßSs«2J	
00000710	CE	0F	BF	8B	61	<b>XOR key 2</b>						60	89	B2	8E	25	32	İ.ç<as`-Ó.`%²ž%2
00000720	8E	D1	6A	C2	58	B4	D1	5D	E4	B8	90	02	A6	6C	F1	E7	ŽÑjĀX`Ñ]ā...!lñç	
00000730	49	F5	F8	C3	18	6D	DC	8B	8C	82	C3	51	5C	3A	6D	7A	IðøĀ.mÛ<Ē,ĀQ\mz	
00000740	E1	8A	FE	17	A9	77	82	BA	5E	59	30	86	A4	2B	4A	67	áŠp.°w, °^Y0†µ+Jg	
00000750	49	2E	E5	27	D6	4D	DC	35	67	B4	93	A0	7B	E1	31	77	I.ā'ÖMÛ5g`" {álw	
00000760	48	FC	93	72	C9	7A	F9	7C	D8	7F	D2	27	17	20	78	A1	Hü`rÉzù ø.Ò'. x;	
Separator	00	00	00	00	00	00	00	00	5B	56	91	30	A1	22	1D	B6	.....[V`0;"`¶	
00000780	3E	A4	85	C5	91	89	F5	61	B2	32	A2	E3	98	19	64	90	>µ.Ā`%øa²2cā`.d.	
00000790	CE	9A	B1	D3	1A	C5	57	26	9D	7B	DE	FF	08	10	25	D9	İš±Ó.ĀW&.{ÿÿ..%Û	
000007A0	B4	77	0E	EC	<b>Encoded strings 2</b>						2C	DD	BF	38	`w.ìæè]Wµ,.-,Ýç8			
000007B0	88	F9	F7	67	C8	18	DF	81	31	90	C3	90	7D	03	A8	E2	^ù÷gÈ.ß,1.Ā.}.`ā	
000007C0	E2	E7	AF	02	28	5B	19	1	82	CF	90	70	C0	19	E7	E9	âç`. ([.,İ.pĀ.çé	

Encoded strings .data

Next steps are splitting on separators, decode expected blob pairs and drop any non printable. Results are fairly obvious when decoding is successful as Qakbot produces clean strings. I typically have seen two well defined groups with strings aligning to Qakbot capabilities.

```

0 : "netstat -nao"
1 : "Component_07"
2 : "cmd.exe /c set"
3 : "Component_08"
4 : "%s \"%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 /TN %u /TR "%s" /HP /F"
10 : "Self test FAILED!!!"
11 : "net localgroup"
12 : "Self check"
13 : "whoami /all"
14 : "ipconfig /all"
15 : "ERROR: GetModuleFileNameV() 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: GetModuleFileNameV() failed with error: %u"
21 : "SoNuceJugdiB3c[doHuce2s81*uXncvP"
22 : "\\System32\WindowsPowerShell\v1.0\powershell.exe"
23 : "SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList"
24 : "anstream.dll"
25 : "bUdiuy81gYguty@4frdRdpfko(eKndeufncueaN"
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.

The screenshot shows a configuration interface for Windows.Carving.Qakbot. It features several input fields for search parameters:

- TargetGlob:** Glob to target payloads on disk
- PidRegex:** .
- ProcessRegex:** ? for suggestions
- StringOffset:** 0x50
- ResourceRegex:** BITMAP|RCDATA

Below these fields is a table titled "Keys" with columns for "Type" and "Key". Each row includes a plus sign (+) and a trash icon (🗑️) for management.

	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.



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	▼ [ <ul style="list-style-type: none"> <li>0 : "BB12"</li> <li>1 : "BB14"</li> <li>2 : "obama235"</li> <li>3 : "obama236"</li> <li>4 : "obama239"</li> </ul> ]	▶ [...]	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	▶ [...]	▼ [ <ul style="list-style-type: none"> <li>0 : "443"</li> <li>1 : "995"</li> </ul> ]	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;rtssniff.exe;packetcapture.exe;capturenet.exe;qak_proxy;dumpcap.exe;CFF Explorer.exe;not_rundll32.exe;ProcessHacker.exe;tcpview.exe;filemon.exe;procmom.exe;idaq64.exe;loadll32.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;rtssniff.exe;packetcapture.exe;capturenet.exe;qak_proxy;dumpcap.exe;CFF Explorer.exe;not_rundll32.exe;ProcessHacker.exe;tcpview.exe;filemon.exe;procmom.exe;idaq64.exe;loadll32.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;procdumperx64.exe;anti-virus.EXE;sysinfoX64.exe;sctoolswrapper.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

1. [Malpedia, Qakbot](#)
2. [Elastic, QBOT Malware Analysis](#)
3. [Hasherezade, Hollows Hunter](#)
4. [Windows.Carving.Qakbot](#)

---

Source: <https://docs.velociraptor.app/blog/2023/2023-04-05-qakbot/>