Decrypting Qakbot's Encrypted Registry Keys

trustwave.com/en-us/resources/blogs/spiderlabs-blog/decrypting-qakbots-encrypted-registry-keys/



Since the return of the Qakbot Trojan in early September 2021, especially through <u>SquirrelWaffle</u> malicious spam campaigns, we've received a few Qakbot samples to analyze from our Trustwave DFIR and Global Threats Operations teams.

Qakbot is a banking Trojan that has been around since 2007. It has been continually developed, with new capabilities introduced such as lateral movement, the ability to exfiltrate email and browser data, and to install additional malware. One new skill is to insert encrypted data into the registry. One of the requests we received from Trustwave's DFIR and Global Threats Operations teams is for us to decrypt the registry data that Qakbot created. We duly jumped into this task, and, as it was a bit of fun, decided to blog about it.

·	CalendarRT	^	Name				Туре				Data
·	Command Processor		ab (Defau	ult)			REG SZ				(value not set)
·	CommsAPHost		15e72	52h			REG RIN	ARV			13 55 d4 06 f8 76 8f 23 11 6
· 🗌	ContactsRT		20706	65			DEC DIN				21 hA 00 d1 d6 66 66 0a h6 26
·	Cryptography		00 6 4701	0.6			DEC DIN				31 04 09 01 01 11 11 0C 01 31
·	CTF			Uer			REG_BIN	AKY			58 91 10 44 C6 9D 01 05 40
	Efwramsn		100 513190	193			KEG_BIN	AKY			69 94 87 25 c6 d9 c0 11 09
·	EventSystem		56 6aae4	Ddd			REG_BIN/	ARY			a4 e4 6e fb dd 18 62 45 f8 5
·	Exchange			1000			DEC DIAL	01/		\sim	cf 4b 9f 05 e8 6b 0f e0 20 2
·	F12		Edit Bin	Edit Binary Value X						^	4b 0d 07 53 fd e5 7b aa 10
·	Fax		Value na	Value name: 8e 59 73 5							8e 59 73 5c 4f 49 f4 b8 3a c
٠Ē	Feeds		Conce 40								
	Fiddler2	-	63364000								
٠Ē	FTP		Value data:								
Ē	GameBarApi		0000	À4	E4 (6E F	B DD 1	8 62 4	¤änûΫ.bE	^	
	IdentityCRL		0008	F8 16	5F 1	F7 9 30 B	C CD E	5 AD 7	3 ø_÷.Iå−s 3 G//*b∵		
	Installer		0018	21	F8 :	3B 3	2 4E 0	D 9Å C) 1ø;2N.Å		
	Internet Connection Wizard		0020	46	OE :	36 B	A F7 6	9 4 4 6	E F_6°÷iJn		
Ē	Internet Explorer		0028	73	28	9E B 52 O	3 FB 6 F 2D 8	1 65 8 4 FE 0	s(R - b)		
Ē	Internet Mail and News		0038	8B	79	81 1	8 3B D	B A4 C	3 .y.;Û¤Ê		
Ē	Messaging		0040	2D	A1 1	B2 2	C E8 7	0 7C D	5 −i°,èp 0		
Ē	Microsoft Management Cons		0048	3C	37	91 B	0 a/ a D 43 5	3 19 F	4 (7.½CS.ô		
Ē	Microsoft SDKs		0058	28	1F (CE D	A 44 B	1 97 8	3 (.10D±		
Ē	MS Design Tools		0060	54	09	82 D	F 66 1	9 C9 D	T.Bf.EU	~	
	MSDAIPP							04	Canad		
	MSF							UK	Cance		
	Multimedia										

Figure 1. A sample of an encrypted registry key that Qakbot creates

There are only a few good detailed analyses of Qakbot out there (see <u>here</u>, <u>here</u>, and <u>here</u>) but in them we didn't really find any technical details on how to decrypt these registry keys. In this blog, we will do our best to explain that trick and we hope this will help fellow malware reversers.

The Flow

For those who don't have time to read the whole blog, we've prepared a graph below to show the decryption flow:



Figure 2. Qakbot's registry data decryption flow.

Key generation

Initially, system information is gathered by Qakbot from the infected host, including:

- 1. Computer Name (using <u>GetComputerNameW</u>)
- 2. Volume Serial Number (using <u>GetVolumeInformationW</u>)
- 3. User Account Name (using LookUpAccountSidW)

Let's take, for example, our infected machine's information:

Computer name: DESKTOP-4NQG47A (converted to UPPERCASE) Volume Serial: 2797280851 (converted from the hexadecimal serial number A6BB1E53) User Account Name: SECRET ACCOUNT (converted to UPPERCASE)

This information is then concatenated to form a password:

DESKTOP-4NQG47A2797280851SECRET ACCOUNT

The password is then hashed using a modified CRC32_shift4 algorithm.

```
1 int __usercall mit_crc32_shift4@<eax>(unsigned int len@<edx>, BYTE *data@<ecx>, int seed)
2 {
3
    int crc; // esi
    unsigned int i; // ecx
4
    unsigned int v7; // esi
5
6
7
    cnc = ~seed;
   if ( !len )
8
9
     return 0;
0
    for ( i = 0; i < len; ++i )
1
      v7 = g_precalc_table[(data[i] ^ (unsigned __int8)crc) & 0xF] ^ ((data[i] ^ (unsigned int)crc) >> 4);
2
      crc = g_precalc_table[v7 & 0xF] ^ (v7 >> 4);
13
4
    3
5
   return word;
6}
```

Figure 3. Modified CRC32 shift4 function.

The resulting hash in this example is AC E9 B5 8D - we will call this PASSWORDHASH.

PASSWORD = "DESKTOP-4NQG47A2797280851SECRET ACCOUNT"

mit_crc32_shift4(PASSWORD) // returned value "\xac\xe9\xb5\x8d"

PASSWORDHASH = "\xac\xe9\xb5\x8d"

Configuration ID

Each registry key value name that the Qakbot malware created is a configuration field defined by a one-byte ID. This ID is also used to salt the PASSWORDHASH.

Joining both the ID and PASSWORDHASH, then hashing them with the SHA1 algorithm, will get a derived key, that we will call DERIVED_KEY.

```
SHA1(<1 bytes ID> + <3 byte \x00 padding> + < 4 bytesCRC32 Hash KEY_B>) = DERIVED_KEY
```

Let's take for example: ID = 0Eh and PASSWORDHASH = \xac\xe9\xb5\x8d

SHA1("\x0e" + "\x00\x00\x00" + "\xac\xe9\xb5\x8d") = \x7a\x2b\x30\xb1\xaf\x46\xeb\xc0\xe3\xc7\xf6\x9b\xf1\x97\x2b\x05\xd5\xca\x06\x8f

The SHA1 hash result will be used as a derived key to decrypt the registry key value data respective to the ID using the RC4 algorithm.

Decrypting the Registry:

To determine which specific registry key value name it will decrypt the ID and DERIVED_KEY are joined together and hashed using the CRC32_shift4 algorithm to obtain the registry value name.

```
mit_crc32_shift4("\x0e\x00\x00\x00" + " \xac\xe9\xb5\x8d") -> "\x6a\xae\x40\xdd"
```

The screenshot below shows the specific registry key value name (6aae40dd) that can be decrypted with RC4 Algorithm using the DERIVED_KEY:

Applying the RC4 algorithm to decrypt the registry key-value data from the value name "6aae40dd" reveals the configuration containing the malware installation timestamp.

6aae40dd id=14 (0x0e)

```
03 01 16 00 00 00 35 3b 31 3b 30 7c 33 3b 32 31 | .....5;1;0|3;2
3b 31 36 33 38 37 35 32 30 35 35 00 8e 53 03 0b | 1;1638752055..S.
df e5 f0 2d bf 42 cb 70 bf 1d 62 d1 d8 ec 1a c5 | ....-B.p..b....
a8 f4 cf d8 e1 c4 bd 52 18 d6 68 a6 e2 95 03 f8 | .....R..h....
c8 c9 a3 41 7a ff 6b 69 11 2b 1b 9b 60 d4 19 49 | ....Az.ki.+..`..
00 eb f5 7f 08 24 86 c0 10 6d 55 d7 bd ce 2c 23 | I.....$...mU...,
e9 d7 91 b1 | #....
```

Decryption Tool:

We wrote a decryption tool to aid this process and it is available in our <u>Github account</u> repository. This tool may help malware reversers and security researchers decrypt Qakbot's registry keys.

Usage: qakbot-registry-decrypt.py [options]

Options:

-h, --help show this help message and exit

-r REGISTRY_PATH, --regpath=REGISTRY_PATH

registry path where Qakbot's encrypted data is stored.

(e.g. 'HKEY_CURRENT_USER\SOFTWARE\Microsoft\Efwramsn')

-p PASSWORD, --password=PASSWORD

password (optional)

Example Usage:

C:\Malware>python qakbot-registry-decrypt.py -r HKEY_CURRENT_USER\Software\Microsoft\Tvojlulgjuu\ Using password {in UTF-16>: "WIN-1391FE15DAF186117 Password CRC32_shift4 Hash: 0x20abcfb8	Â
Registry key path: HKEY_CURRENT_USER\Software\Microsoft\Tvojlulgjuu\\f5335acd RC4 key: 2f f? d3 76 9b 62 52 04 00 6e 21 f0 8b 3f e6 20 57 f8 a8 03 Decrypted value:	=
00000000: 03 01 1F 00 00 00 35 3B 31 3B 31 36 34 30 30 375;1;164007 00000010: 35 30 38 32 7C 33 3B 32 31 3B 31 36 34 30 30 37 5002:3;21;164007 00000020: 35 30 38 32 00 28 1E BF CE 5082.<	
Registry key path: HKEY_CURRENT_USER\Software\Microsoft\Tvojlulgjuu\\c0ac8a83 RC4 key: 6d b7 d4 36 c9 20 5a 80 5d fa ac cd d6 12 3b 55 00 3f 40 f9 Decrypted value:	
00000000: 04 01 82 00 00 00 43 00 3A 00 5C 00 55 00 73 00C.:.\U.s. 00000010: 00000020: .\.A.p.p.D.a.	
00000040: 6E 00 67 00 5C 00 4D 00 69 00 63 00 72 00 6F 00 n.g.\.M.i.c.r.o. 00000040: 6E 00 6F 00 66 00 74 00 5C 00 55 00 6D 00 79 00 s.o.f.t.\U.m.y. 000000060: 61 00 65 00 63 00 79 00 67 00 61 00 79 00 5C 00 a.e.c.y.g.a.y.\.	
00000070: 74 00 76 00 6F 00 6A 00 6C 00 75 00 6C 00 2E 00 t.u.o.j.l.u.l 00000080: 64 00 6C 00 6C 00 00 00 2D 3A EF E3 50 9F 9D D6 d.l.l:P 00000090: 93 0F 26 FA 40 5E 80 37 29 3C 5F 71 8B A5 78 A9&.@^.7><_qx.	

Figure 4. Qakbot registry decryptor tool

IOCs:

Qakbot DLL

MD5: 90aac91ba4336bdb252dee699d32d78d MD5: a53c130fe120348b9bfa188ab93b6ad4