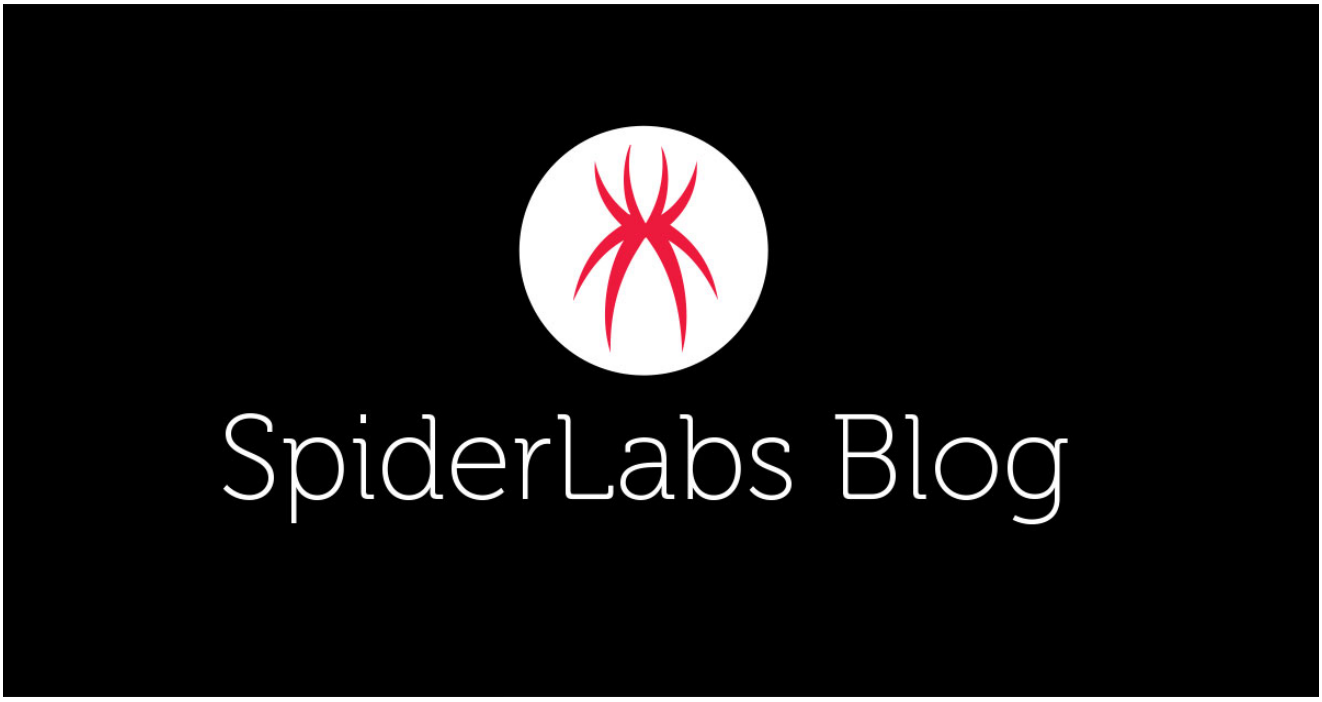
Decrypting Qakbot's Encrypted Registry Keys

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 SquirrelWaffle 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.

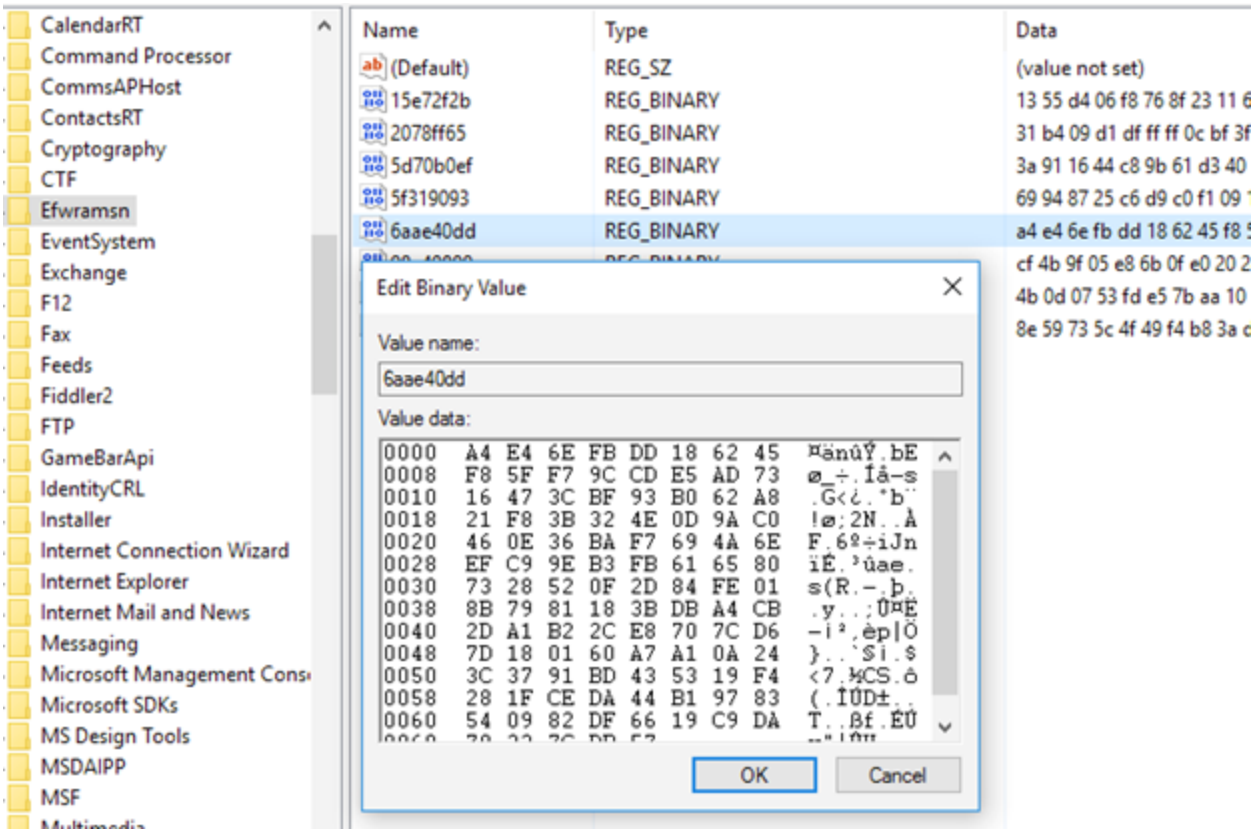


Figure 1. A sample of an encrypted registry key that Qakbot creates

There are only a few good detailed analyses of Qakbot out there (see here, here, and here) 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:

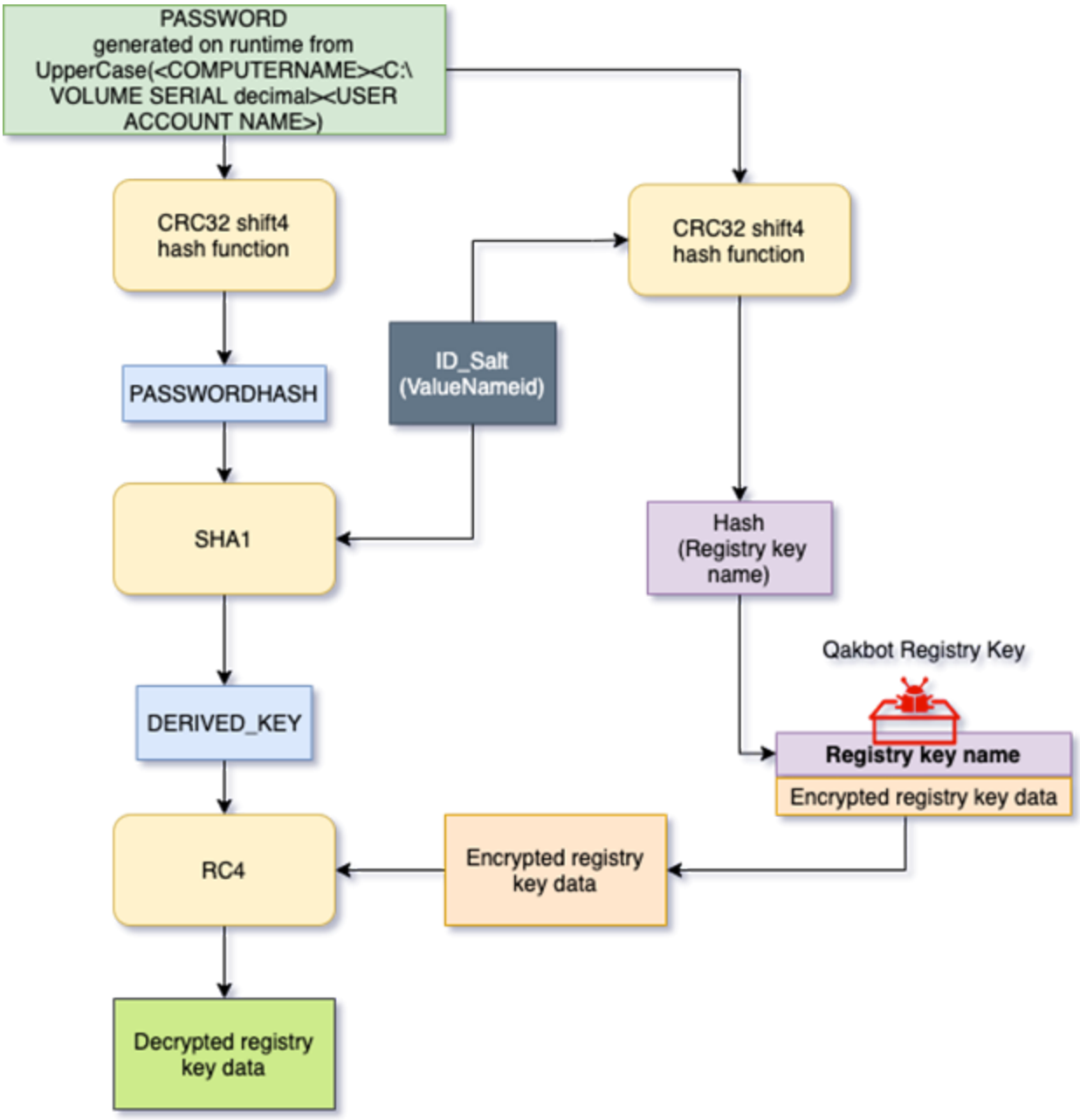


Figure 2. Qakbot's registry data decryption flow.

Key generation

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

1. Computer Name (using GetComputerNameW)
2. Volume Serial Number (using GetVolumeInformationW)
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.

```
int __usercall mit_crc32_shift4@<eax>(unsigned int len@<edx>, BYTE *data@<ecx>, int seed)
{
  int crc; // esi
  unsigned int i; // ecx
  unsigned int v7; // esi

  crc = ~seed;
  if ( !len )
    return 0;
  for ( i = 0; i < len; ++i )
  {
    v7 = g_precalc_table[(data[i] ^ (unsigned __int8)crc) & 0xF] ^ ((data[i] ^ (unsigned int)crc) >> 4);
    crc = g_precalc_table[v7 & 0xF] ^ (v7 >> 4);
  }
  return ~crc;
}
```

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:

\x7a\x2b\x30\xb1\xaf\x46\xeb\xc0\xe3\xc7\xf6\x9b\xf1\x97\x2b\x05\xd5\xca\x06\x8f

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 Github account 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:

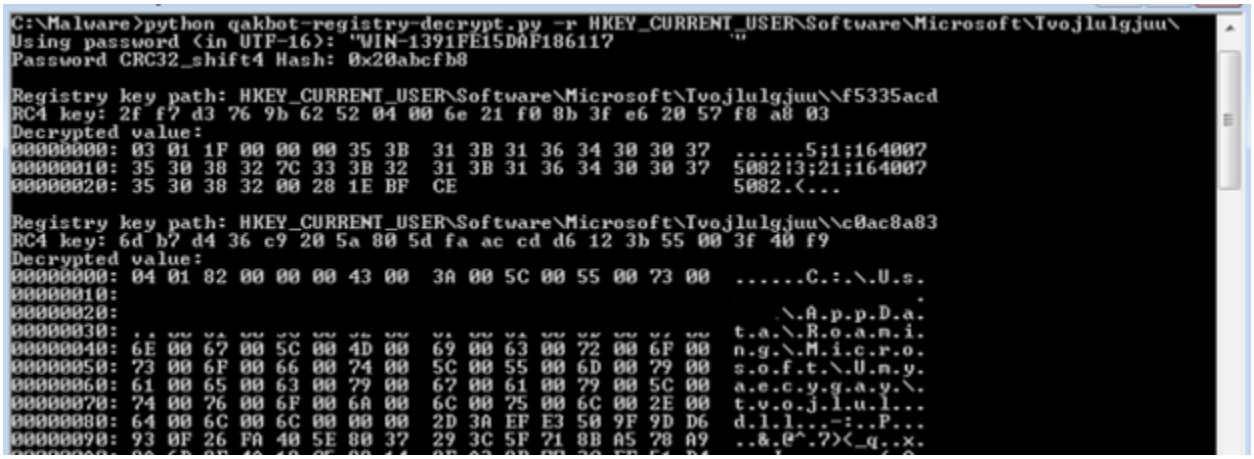


Figure 4. Qakbot registry decryptor tool

IOCs:

Qakbot DLL

MD5: 90aac91ba4336bdb252dee699d32d78d
MD5: a53c130fe120348b9bfa188ab93b6ad4