Qakbot Series: Configuration Extraction

April 13, 2022

2022-04-13

Malware Analysis , Qakbot

In late March 2022, I was requested to analyze a software artifact. It was an instance of Qakbot, a modular information stealer known since 2007. Differently to other analyses I do as part of my daily job, in this particular case I can disclose wide parts of it with you readers. I'm addressing them in a post series. Here, I'll discuss about the configuration extraction based on this specific sample.

The configuration of recent Qakbot samples is often stored in two distinct resources of the unpacked payload. Each resource contains a different part of the configuration. A first resource, usually the shortest one in size, contains general settings such as the botnet identifier and the campaign timestamp. A second resource, usually bigger in size, contains the list of command and control IP addresses and ports. Both resources are encrypted and this post is all about discussing how Qakbot decrypts its configuration.

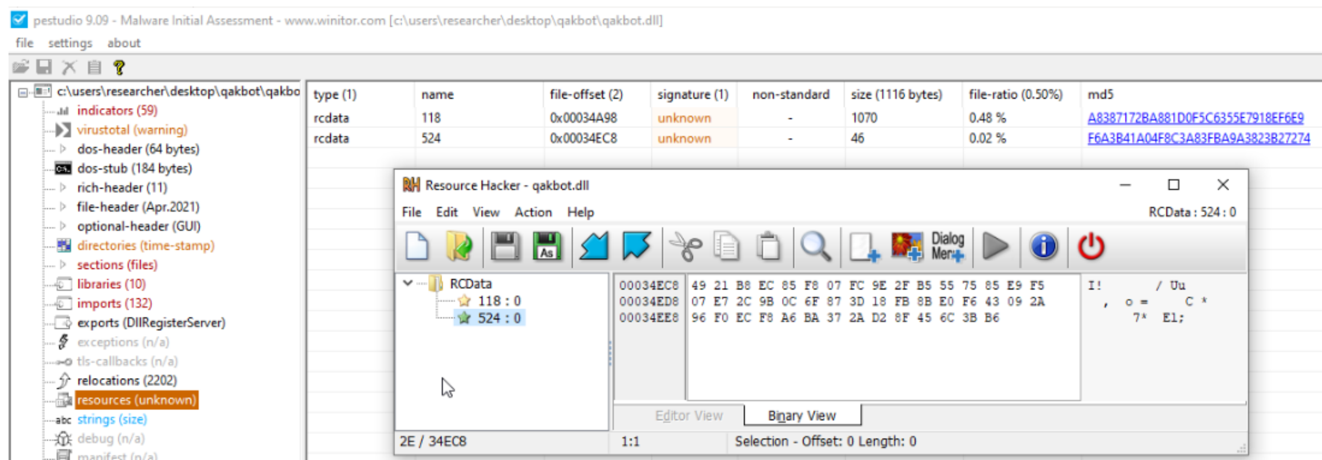

Figure 1

-

Qakbot resouces storing the malware configuration

As you may notice from **Figure 1**, our sample contains two encrypted sections. The smaller in size is labelled "524" and the bigger in size is labelled "118". Those two sections labels aren't new to the malware research community since the existence of other Qakbot samples sharing the same resource labels is documented (see, as an example, Tavares - 2021). Both resources are encrypted with a custom implementation of the RC4 algorithm. The RC4 key-

scheduling algorithm is implemented in the function starting at *0xb339f9*. The RC4 pseudo-random generation algorithm is implemented in the function starting at *0xb33924*. The key used to decrypt the resources is the SHA1 of the string:

```
\System32\WindowsPowerShell\v1.0\powershell.exe
```

Notice that you won't find that string into the sample because it is obfuscated. If you are interested in the Qakbot strings obfuscation technique, I discuss it in this post.

```
int * __cdecl extract_configuration(undefined4 param_1)

{
  int buffer;
  int *piVar1;
  char *local_10;
  short *local_c;
  SIZE_T local_8;

  piVar1 = (int *)0x0;
  local_8 = 0;
                    /* "524" */
  local_10 = (char *)deobfuscate_string_2(0xbc);
  local_c = (short *)load_resource(param_1,local_10,&local_8);
  BitBlt((HDC)0x0,0x27,4,8,0x46,(HDC)0x0,0x46,0x5f,0x2c);
  erase_string_wrapper(&local_10);
  if (local_c != (short *)0x0) {
                    /* "\System32\WindowsPowerShell\v1.0\powershell.exe" */
    local_10 = (char *)deobfuscate_string_2(0x19);
    buffer = decrypt_validate_resource(local_c,local_8,local_10);
    if (buffer != 0) {
      piVar1 = parse_configuration(buffer);
    }
    erase_string_wrapper(&local_10);
    erase_string(&local_c,local_8);
  }
  return piVar1;
}
```

Figure 2

-

Decompiled listing of the Qakbot general configuration extraction

Qakbot contains a custom implementation of the SHA1 algorithm. The implementing function is located at *0xb3745c*. **Figure 2** shows the listing of the function responsible for the configuration extraction from the resource "524" located at *0xb234bb*. As you may notice, the resource label and the key string are obfuscated and we added their de-obfuscated

counterpart as comments . Moreover, you may notice an example of junk API call (BitBlt) placed into the code just to distract the analyst. I'll pospone a general discussion about junk code injection and other Qakbot anti-analysis techniques in a dedicated post.

```
int __cdecl
decrypt_validate(short *resource,int content_offset,short *content,int content_size,short *buffer)

{
  int iVar1;
  int iVar2;
  short s [134];
  short local_18 [10];

  set_buffer(buffer,content,content_size);
  crypt_rc4_ksa(resource,content_offset & 0xffff,s);
  crypto_rc4_prga(buffer,content_size,s);
  iVar2 = content_size + -0x14;
  crypto_sha1(buffer + 10,iVar2,local_18);
  iVar1 = check_sha1(local_18,buffer,0x14);
  if (iVar1 == 0) {
    set_buffer(buffer,buffer + 10,iVar2);
  }
  else {
    iVar2 = -1;
  }
  return iVar2;
}
```

Figure 3

-

Decompiled listing of the Qakbot resource decryption and validation

The actual decryption and validation is accomplished by the function located at *0xb2f7ac* and showed in **Figure 3**. From that listing you may notice that the sample decrypts the resource by calling the RC4 functions. The format of the decrypted resource is composed of two fields: a first field sized in 20 bytes containing the expected SHA1 of the content and a second field consisting of the actual content of the resource. The validation is accomplished by computing the SHA1 of the content field and then by comparing it with the expected SHA1 in the first field. If the validation succeeds, the just decrypted content is placed in a buffer.

```
remnux@remnux:~/Downloads/qakbot$ cat conf.txt
10 -> obama35
3 -> 28/04/2021 09:45:57
```

Figure 4

-

Decrypted content of resource "524"

As already mentioned, the two resources contain different aspects of the Qakbot configuration. The content of resource “524” is structured in key-value pairs where each pair represent a configuration field. The key component is an integer identifier of the field. The value is unstructured since it may contain strings, integers, or even timestamps. As you may notice from **Figure 4**, showing the decrypted content of resource “524”, the configuration contain just two fields: field 10, containing the botnet identifier, and field 3 containing what is often referred as the campaign identifier. Actually, field 3 is a Unix timestamp probably indicating the creation date of the campaign. Older Qakbot samples were used to contain a richer configuration (Kremez - 2018).

c2conf.bin

```
00000000 2F 12 11 23 7C D2 E7 8B 69 23 1D 7A 4F 62 53 18 19 B7 D5 E7 01
00000015 18 75 6B 78 01 BB 01 4B 89 2F AE 01 BB 01 69 C6 EC 65 01 BB 01
0000002a 51 61 9A 64 01 BB 01 B8 C9 A2 9E 01 BB 01 47 BB AA EB 01 BB 01
0000003f 2F C4 C0 B8 01          01 88 E8 22          BB 01 2F 16 94 06 01 BB 01
00000054 4B 43 C0 7D 01 BB 01 18 E5 96 36 03 E3 01 AC 4E 28 3D 01 BB 01
00000069 90 8B 2F CE 01 BB 01 47 A3 DE F3 01 BB 01 2D 3F 6B C0 03 E3 01
```

SHA1 IP Port

Figure 5

-

Structure of the network configuration resource (decrypted)

Resource “118” contains the network configuration of the sample. It is structured in records separated by the byte *0x01*. Each record is composed of four bytes each one representing an IP address octet and additional two bytes representing the port. The network configuration for the sample object of analysis contains 150 command and control IP addresses and ports. **Figure 5** shows the format of the unencrypted network configuration resource. You may find all the C2 addresses here.

As always, if you want to share comments or feedbacks (rigorously in broken Italian or broken English) do not esitate to drop me a message at **admin[@]malwarology.com**.