Stealc Malware Analysis

By Glyc3rius

Published: 2023-10-03 · Archived: 2026-04-05 22:01:21 UTC

Stealc is just a typical information stealer written in C. The malware encodes/encrypts its strings with base64 and RC4 methods and imports its functions with the help of PEB while several anti-analysis and evasion techniques are also applied. It drops 7 additional third-party DLLs (such as `sqlite3.dll`) from the C2 server. The stealing procedure targets browsers, browser extensions, desktop cryptocurrency wallets and applications such as Outlook, Steam, Discord, Telegram, Tox, and Pidgin. It gathers information about the victim's machine as well and after its all done, removes itself and the dropped DLLs from the computer.

The analysed sample's SHA-256 is `e978871a3a76c83f94e589fd22a91c7c1a58175ca5d2110b95d71b7805b25b8d`.

Inside the disassembler, we discover two functions which contain likely important strings that are encoded with base64. After decoding them, we can conclude that an encryption mechanism is used as well. In search of it, an RC4 algorithm is found with its hard-coded key: `5212972219813087498979555738126126481424934832398 6`. I decrypted these strings with a simple python script. The deobfuscated strings are:

```
 11
20
23
GetProcAddress
LoadLibraryA
lstrcatA
OpenEventA
CreateEventA
CloseHandle
Sleep
GetUserDefaultLangID
VirtualAllocExNuma
VirtualFree
GetSystemInfo
VirtualAlloc
HeapAlloc
GetComputerNameA
lstrcpyA
GetProcessHeap
GetCurrentProcess
lstrlenA
ExitProcess
GlobalMemoryStatusEx
GetSystemTime
SystemTimeToFileTime
advapi32.dll
gdi32.dll
user32.dll
crypt32.dll
ntdll.dll
GetUserNameA
CreateDCA
GetDeviceCaps
ReleaseDC
CryptStringToBinaryA
sscanf
VMwareVMware
HAL9TH
JohnDoe
DISPLAY
%hu/%hu/%hu
```

```
http://185.106.94.206
/4e815d9f1ec482dd.php
/49171d9bb28d893a/
GoogleMaps
GetEnvironmentVariableA
GetFileAttributesA
GlobalLock
HeapFree
GetFileSize
GlobalSize
CreateToolhelp32Snapshot
IsWow64Process
Process32Next
GetLocalTime
FreeLibrary
GetTimeZoneInformation
GetSystemPowerStatus
GetVolumeInformationA
GetWindowsDirectoryA
Process32First
GetLocaleInfoA
GetUserDefaultLocaleName
GetModuleFileNameA
DeleteFileA
FindNextFileA
LocalFree
FindClose
SetEnvironmentVariableA
LocalAlloc
GetFileSizeEx
ReadFile
SetFilePointer
WriteFile
CreateFileA
FindFirstFileA
CopyFileA
VirtualProtect
GetLogicalProcessorInformationEx
GetLastError
lstrcpynA
MultiByteToWideChar
GlobalFree
WideCharToMultiByte
GlobalAlloc
OpenProcess
TerminateProcess
GetCurrentProcessId
gdiplus.dll
ole32.dll
bcrypt.dll
wininet.dll
shlwapi.dll
shell32.dll
psapi.dll
rstrtmgr.dll
CreateCompatibleBitmap
SelectObject
BitBlt
DeleteObject
CreateCompatibleDC
GdipGetImageEncodersSize
GdipGetImageEncoders
GdipCreateBitmapFromHBITMAP
GdiplusStartup
```

```
GdiplusShutdown
GdipSaveImageToStream
GdipDisposeImage
GdipFree
GetHGlobalFromStream
CreateStreamOnHGlobal
CoUninitialize
CoInitialize
CoCreateInstance
BCryptGenerateSymmetricKey
BCryptCloseAlgorithmProvider
BCryptDecrypt
BCryptSetProperty
BCryptDestroyKey
BCryptOpenAlgorithmProvider
GetWindowRect
GetDesktopWindow
GetDC
CloseWindow
wsprintfA
EnumDisplayDevicesA
GetKeyboardLayoutList
CharToOemW
wsprintfW
RegQueryValueExA
RegEnumKeyExA
RegOpenKeyExA
RegCloseKey
RegEnumValueA
CryptBinaryToStringA
CryptUnprotectData
SHGetFolderPathA
ShellExecuteExA
InternetOpenUrlA
InternetConnectA
InternetCloseHandle
InternetOpenA
HttpSendRequestA
HttpOpenRequestA
InternetReadFile
InternetCrackUrlA
StrCmpCA
StrStrA
StrCmpCW
PathMatchSpecA
GetModuleFileNameExA
RmStartSession
RmRegisterResources
RmGetList
RmEndSession
sqlite3_open
sqlite3_prepare_v2
sqlite3_step
sqlite3_column_text
sqlite3_finalize
sqlite3_close
sqlite3_column_bytes
sqlite3_column_blob
encrypted_key
PATH
C:\\ProgramData\\nss3.dll
NSS_Init
NSS_Shutdown
PK11_GetInternalKeySlot
```

```
PK11_FreeSlot
PK11_Authenticate
PK11SDR_Decrypt
C:\\ProgramData\\
url:
SELECT origin_url, username_value, password_value FROM logins
browser:
profile:
login:
password:
Opera
OperaGX
Network
cookies
.txt
TRUE
SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-11644480800, name, encrypted_value from cookies
FALSE
autofill
SELECT name, value FROM autofill
history
SELECT url FROM urls LIMIT 1000
cc
name:
SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards
month:
year:
card:
Cookies
Login Data
Web Data
History
logins.json
formSubmitURL
usernameField
encryptedUsername
encryptedPassword
guid
SELECT host, isHttpOnly, path, isSecure, expiry, name, value FROM moz_cookies
SELECT fieldname, value FROM moz_formhistory
SELECT url FROM moz_places LIMIT 1000
cookies.sqlite
formhistory.sqlite
places.sqlite
plugins
Local Extension Settings
Sync Extension Settings
IndexedDB
Opera Stable
Opera GX Stable
CURRENT
chrome-extension_
_0.indexeddb.leveldb
Local State
profiles.ini
chrome
opera
firefox
wallets
%08lX%04lX%lu
SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion
ProductName
x32
x64
```

```
%d/%d/%d %d:%d:%d
HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0
ProcessorNameString
DisplayName
SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall
DisplayVersion
Network Info:
\t- IP: IP?
\t- Country: ISO?
System Summary:
\t- HWID:
\t- OS:
\t- Architecture:
\t- UserName:
\t- Computer Name:
\t- Local Time:
\t- UTC:
\t- Language:
\t- Keyboards:
\t- Laptop:
\t- Running Path:
\t- CPU:
\t- Threads:
\t- Cores:
\t- RAM:
\t- Display Resolution:
\t- GPU:
User Agents:
Installed Apps:
All Users:
Current User:
Process List:
system_info.txt
freebl3.dll
mozglue.dll
msvcp140.dll
nss3.dll
softokn3.dll
vcruntime140.dll
\\Temp\\
.exe
runas
open
/c start
%DESKTOP%
%APPDATA%
%LOCALAPPDATA%
%USERPROFILE%
%DOCUMENTS%
%PROGRAMFILES%
%PROGRAMFILES_86%
%RECENT%
*.lnk
files
\\discord\\
\\Local Storage\\leveldb\\CURRENT
\\Local Storage\\leveldb
\\Telegram Desktop\\
key_datas
D877F783D5D3EF8C*
map*
A7FDF864FBC10B77*
A92DAA6EA6F891F2*
F8806DD0C461824F*
```

```
Telegram
Tox
*.tox
*.ini
Password
Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A66
Software\\Microsoft\\Office\\13.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\
Pidgin
Software\\Microsoft\\Office\\14.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\
accounts.xml
Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\
dQw4w9WgXcQ
Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676\\
ssfn*
Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\\9375CFF0413111d3B88A00104B2A6676\\
00000001
00000002
00000003
00000004
\\Outlook\\accounts.txt
\\.purple\\
token:
Software\\Valve\\Steam
SteamPath
\\config\\
config.vdf
DialogConfig.vdf
DialogConfigOverlay*.vdf
libraryfolders.vdf
loginusers.vdf
\\Steam\\
sqlite3.dll
browsers
done
soft
\\Discord\\tokens.txt
/c timeout /t 5 & del /f /q "
" & del "C:\\ProgramData\\*.dll"" & exit
C:\\Windows\\system32\\cmd.exe
https
Content-Type: multipart/form-data; boundary=----
POST
HTTP/1.1
Content-Disposition: form-data; name="
hwid
build
token
file_name
file
message
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
screenshot.jpg
```

Stealc uses the **Process Environment Block (PEB)** to dynamically load libraries and to avoid antivirus detection. In this case, the malware uses the **PEB** as an obfuscation technique to hide Windows API libraries and their imported functions.

Accessing the PEB structure can be described with the table below:

	Offset	**Description**
1.	`FS:[offset ProcessEnvironmentBlock]`	The equivalent of `FS:[0x30]`, that means we access the **PEB** directly

	Offset	Description
2.	`0xc`	Access to the `LoaderData`
3.	`0xc`	Access to the `InLoadOrderModuleList`
4.	`0x18`	Access to the `DllBase`

Inside the `DllBase`, the malware looks for the base address of `kernel32.dll`. After that it loads the `GetProcAddress` function that is used to dynamically resolve the address of the `LoadLibraryA` function. Then, `LoadLibraryA` loads additional DLLs that the malware can utilize and their functions are also called with `GetProcAddress` dynamically.

Anti-analysis and Evasion Methods

First, Stealc checks the number of pixels the screen can show vertically and compares the `GetDeviceCaps` function's return value to 666 which represents the screen height. In case the screen height is below 666 pixels, the malware stops execution (for example, if the screen resolution is `800x600` then the `ExitProcess` is called, since 600 is lower than 666). This is a technique to avoid virtual machines with lower resolutions which are not expected in a regular environment.

It checks whether the victim's machine has a processor with at least 2 cores. If it doesn't, the malware stops execution, since it assumes that the machine is in a virtualized environment.

With `GlobalMemoryStatusEx`, the malware retrieves information about the system's physical and virtual memory. If the total physical memory is under 1111 MB of memory, the malware calls the `ExitProcess` function. This ensures that the malware is not running under a virtual machine.

The stealer also tries to allocate memory with `VirtualAllocExNuma`. It is an anti-emulator technique, since an AV emulator can't perform this kind of allocation, making the API call fail. If the allocation doesn't happen, the malware exits immediately.

Stealc checks the language ID of the current user with the `GetUserDefaultLangID` function and if they return one of the hexadecimal values from the table below, then the malware exits and won't run on the victim's computer.

Language ID (Hex)	Country
0x419	Russia
0x422	Ukraine
0x423	Belarus
0x43F	Kazakhstan
0x443	Uzbekistan

It calls `GetComputerNameA` with **HAL9TH** parameter and `GetUserNameA` with **JohnDoe**. Both of these parameters are used by Microsoft Defender emulator and they are compared to the victim's computer name and username. It checks whether the malware is in a virtual or sandbox environment.

Before the malware starts its information stealing procedure, it downloads 7 DLLs from the C2 server: `hxxp://185[.]106[.]94[.]206/49171d9bb28d893a/`. These DLLS are: `sqlite3.dll`, `nss3.dll`, `freebl3.dll`, `mozglue.dll`, `msvcp140.dll`, `softokn3.dll`, `vcruntime140.dll`. All of them are valid third-party DLLs and they are placed within the `C:\ProgramData\` folder and all the stolen data is also stored under this path. The `sqlite3.dll` and `nss3.dll` are essential for the information stealing that Stealc performs. The imported functions of these two and their usage are represented in the tables below.

`sqlite3.dll` which is used to interact with SQLite databases in C/C++ applications:

Function	Usage
`sqlite3_open`	Opens or creates a new SQLite database file
`sqlite3_prepare_v2`	Compiles an SQL statement into a prepared statement
`sqlite3_step`	Executes a prepared statement one step at a time

Function	Usage
`sqlite3_column_text`	Retrieves the text value of a column in the current row of the result set
`sqlite3_finalize`	Finalizes a prepared statement and releases associated resources
`sqlite3_close`	Closes the SQLite database connection
`sqlite3_column_bytes`	Retrieves the number of bytes in a column value in the current row of the result set
`sqlite3_column_blob`	Retrieves a blob (binary) value from a column in the current row of the result set

`nss3.dll` which is associated with Firefox:

Function	Usage
`NSS_Init`	Initializes the NSS library
`NSS_Shutdown`	Cleans up and releases resources acquired during NSS initialization
`PK11_GetInternalKeySlot`	Obtains a cryptographic slot
`PK11_FreeSlot`	Releases the slot
`PK11_Authenticate`	Authenticates a cryptographic module or unlocks a cryptographic token
`PK11SDR_GetInternalKeySlot`	Used for decrypting data using the NSS library

Stealc attempts to get the information from browsers that are based on **Chromium, Opera, or Mozilla**. The data related to Opera-based browsers (Opera, Opera GX) are exfiltrated the same way as the Chromium ones. The stolen information could include: login credentials, cookies, autofills, history and credit card details. The information stealer utilizes SQLite with `sqlite3.dll` library and uses SQL queries with `SELECT` statements to get the victim's stolen data. In case of Mozilla-based applications, the `nss3.dll` library is also used to decrypt credentials.

First, the malware searches the `Local State` JSON file to locate Chromium users, then it attempts to gain information from 5 databases:

1. `Login Data`
2. `Web Data`
3. `Cookies`
4. `Network`
5. `History`

Before getting the information from these databases, the stealer needs to **take care of the encrypted values that Chromium browsers have**. First, it searches the `Local State` file and locates the key that is encoded with base64. This is decoded with the `CryptStringToBinaryA` function and with its `CRYPT_STRING_BASE64` parameter. Since this string contains `'DPAPI'` (the name of one of the encryption method) as a prefix of the key, the stealer has to drop this prefix, so that it only decrypts the actual key string. Then it calls the `CryptUnprotectData` and decrypts the actual key. Finally, the AES-GCM decryption is done with the `BCryptDecrypt` function. After these steps, the encrypted information from the SQLite databases are easily retrievable.

The `SELECT origin_url, username_value, password_value FROM logins` query gets the username and password values. The `password_value` is in an encrypted form and decrypted with the method disclosed above.

After the decryption, the information are saved in the following form:

```
 1
2
3
4
5
```

```
 browser: %value%\n
profile: %value%\n
url: %url%\n
login: %username_value%\n
password: %password_value%\n\n
```

`SELECT HOST_KEY, is_httponly, path, is_secure, (expires_utc/1000000)-11644480800, name, encrypted_value from cookies` is the query to collect the cookies. The encrypted cookie value is decrypted with the described method and saves the stolen data into `cookies.txt`. Inside the text file, each column name is separated with a `'\t'` (tab) and each cookie is separated with a `'\n'` (newline) character inside the text file.

In order to get the autofill information, the malware uses the `SELECT name, value FROM autofill` query. After that, it saves the stolen data into `autofill.txt` where each autofill data is split with `'\n'`.

`SELECT url FROM urls LIMIT 1000` gets the history data of the web browser. The `LIMIT 1000` restricts the number of rows returned, so only the first 1000 results are collected. It saves the stolen data into `history.txt` and splits each URL with a `'\n'`.

`SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards` query collects the credit card details and saves the information to a text file called `cc.txt`. As the query suggests, the card number comes in an encrypted form which is decrypted with the method above.

After the decryption process, the card details are written in the following form to the text file:

```
 1
2
3
4
```

```
 name: %name_on_card%\n
month: %expiration_month%\n
year: %expiration_year%\n
card: %card_number_decrypted%\n\n
```

The malware first checks the used profiles in the `profiles.ini` configuration file and targets them. It uses the `nss3.dll` and the `sqlite3.dll` libraries to steal data. The `sqlite3.dll` is used to extract the victim's information with its functions disclosed above in the table. The information that it wants to obtain are: login credentials, cookies, autofills and history data.

The credentials are stored in `logins.json`. The stealer looks for the following 5 values inside the JSON file and extracts them: `formSubmitURL`, `usernameField`, `encryptedUsername`, `encryptedPassword`, `guid`. Since both the username and password values are encrypted, the stealer attempts to decrypt it with the help of the Network Security Services (NSS) library.

The decryption routine with the `nss3.dll` functions:

1. `PK11_GetInternalKeySlot` –> Obtains a cryptographic slot
2. `PK11_Authenticate` –> Authenticates a cryptographic module or unlocks a cryptographic token
3. `PK11SDR_GetInternalKeySlot` –> Used for decrypting data
4. `PK11_FreeSlot` –> Releases the slot

After the decryption, the stolen values are collected in the following form:

```
 1
2
3
4
5
```

```
 browser: %guid%\n
profile: %usernameField%\n
url: %formSubmitURL%\n
login: %username%\n
password: %password%\n\n
```

The applications' cookies are stored in the `cookies.sqlite` database. It is extracted with the `SELECT host, isHttpOnly, path, isSecure, expiry, name, value FROM moz_cookies` query. The stealer collects information about the domain and path for which the cookie is valid, whether the cookie is "http only" and secure, the cookie's expiration time, name and

value. Then it is written in the `cookies.txt` file. Each column name is separated with a `'\t'` and each cookie is separated with a `'\n'` character inside the text file.

The autofill information are kept in the `formhistory.sqlite` database which remembers what the victim searched for in the Firefox search bar and what they've entered into forms on websites. It is extracted with the `SELECT fieldname, value FROM moz_formhistory` query and the stolen data is saved in the `autofill.txt` file. The `fieldname` and `value` columns are separated with a `'\t'` and each autofill data is on a new line (`'\n'`).

The history data can be found in the `places.sqlite` database which contains the user's Firefox bookmarks, downloaded files and visited websites. The `SELECT url FROM moz_places LIMIT 1000` query is used to extract the URL and returns the first 1000 rows. Then the retrieved information is written in the `history.txt` file where each URL is separated with a `'\n'` character.

The stealer also targets Chrome-based browser extensions. It searches for the following folders related to extensions:

- `Local Extension Settings` —> refers to the configuration and settings specific to a particular Chrome extension
- `Sync Extension Settings` —> the ability to synchronize the settings of an extension across multiple devices
- `IndexedDB\chrome-extension__0.indexeddb.leveldb` —> where an extension might store data using IndexedDB such as user preferences, cached content

The `CURRENT` value is also used to get the most recent information related to the extensions. Cryptocurrency wallets are in danger if they are browser-based ones.

Stealc targets desktop cryptocurrency wallets as well. First, it searches for the wallets with the following file path: `C:\Users\%user%\AppData\Roaming\%WalletAppName%\*.*` . This is the default location of the applications that are associated with cryptocurrency wallets. The end value of `*.*` is a wildcard that returns all files with any filename and any file extension within the directory. The `%WalletAppName%` folder is a randomly generated string of 20 letters, such as `AFHIEBKKFHIEGCAKECGH` .

Microsoft's email client Outlook is also targeted by the infostealer. It looks for registry paths to identify default Outlook profiles:

```
 HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\9375CFF0413111d3B88A

HKEY_CURRENT_USER\Software\Microsoft\Office\13.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676\

HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676\

HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676\

HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676\

HKEY_CURRENT_USER\Software\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676\
```

Under the `9375CFF0413111d3B88A00104B2A6676` key in the registry path, it searches for the subkeys `00000001` , `00000002` , `00000003` , `00000004` . The stealer enumerates through these registry paths and if it confirms that Outlook is installed, it targets the password of the victim's account and attempts to decrypt it with the `CryptUnprotectData` function. After the extraction and decryption of the information, it saves the data into a text file: `soft\Outlook\accounts.txt` .

First of all, the stealer checks whether it can open the registry key at `HKEY_CURRENT_USER\Software\Valve\Steam` . If it can be opened that means the Steam application is available on the victim's computer and Stealc retrieves the Steam installation path from the `SteamPath` registry value. It also goes into the `\config\` folder and extracts VDF (Valve Data Format) files such as `config.vdf` , `DialogConfig.vdf` , `DialogConfigOverlay*.vdf` , `libraryfolders.vdf` , `loginusers.vdf` and `ssfn*` as well. Finally, exfiltrated data is written in the `\soft\Steam` folder.

In case of Discord, it looks for the `\Local Storage\leveldb` and the `\Local Storage\leveldb\CURRENT` directories and targets Discord tokens. The tokens are encrypted and every one of them starts with this hard-coded string: `dQw4w9WgXcQ` . The actual token is after that and the stealer uses the following functions to decrypt: `CryptStringToBinaryA` (from base64), `CryptUnprotectData` (DPAPI) and `BCryptDecrypt` (AES with GCM mode). After the decryption is done, it saves the stolen information in the `\soft\Discord\tokens.txt` file.

Under the `Telegram Desktop` folder, the stealer searches for different subdirectory values like `key_datas`, `D877F783D5D3EF8C*`, `map*`, `A7FDF864FBC10B77*`, `A92DAA6EA6F891F2*`, `F8806DD0C461824F*`. The data stolen from the app is then placed in the `\soft\Telegram` folder.

Tox application's configuration files are also aimed at by the stealer. First, it looks for the `\Tox` directory and inside that the `*.tox` and `*.ini` files.

Pidgin messaging application is also aimed at by the stealer. It looks for the configuration directory of the app which is `%APPDATA%\.purple`. If it is located, inside the directory the stealer specifically seeks for the `accounts.xml` file in order to gain Information about the victim's account and credentials.

Stealc also collects network information and a summary of the system, as well as user agents, installed apps, users, current user and process list. The network information consists of the victim's IP address and their country's ISO code. The stolen information is saved in the `system_info.txt` in the following manner:

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

```
 Network Info:
\t- IP: IP?
\t- Country: ISO?
\n\n
System Summary:
\t- HWID:
\t- OS:
\t- Architecture:
\t- UserName:
\t- Computer Name:
\t- Local Time:
\t- UTC:
\t- Language:
\t- Keyboards:
\t- Laptop:
\t- Running Path:
\t- CPU:
\t- Threads:
\t- Cores:
\t- RAM:
\t- Display Resolution:
\t- GPU:
User Agents:
Installed Apps:
All Users:
Current User:
Process List:
```

The malware opens 3 registry paths with the functions `RegOpenKeyExA` and `RegQueryValueExA` in order to gain information about the victim's machine:

1. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion` gives us the information about the version and edition of the Windows NT operating system. The given `ProductName` value within the `RegQueryValueExA` carries the name or edition of the Windows NT operating system.
2. `HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\0` registry path contains information about the CPU. The `ProcessorNameString` value within the `RegQueryValueExA` returns the name or description of the CPU installed on the computer.
3. `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall` path stores information about all of the installed software and lists them on the infected machine. The `RegQueryValueExA` function has the value `DisplayName` which retrieves the name of the programs.

It also takes a screenshot of the desktop with the help of `gdiplus.dll` and saves it as `screenshot.jpg`.

Stealc sends multiple HTTP POST requests back to the `hxxp://185.106.94[.]206/4e815d9f1ec482dd.php` C2 server that gives a response. Here are the 4 most important requests:

1. `browsers` —> web browser data
2. `plugins` —> browser extensions
3. `wallets` —> desktop cryptocurrency wallets
4. `files` —> file grabber

The 7 DLLs are also dropped from the C2 to the host as already mentioned. The malware then gets the information related to the 4 requests and sends them back to the C2. The data from applications like Outlook, Steam, Discord, Tox, Pidgin as well as the network and system information and the screenshot of the desktop are also exfiltrated. Every configuration and all the gathered information are encoded in base64 during the HTTP communication.

After the infostealer finished its job by stealing the targeted information, it attempts to remove itself and the 7 imported third-party DLLs from the victim's machine with the command below:

```
"C:\Windows\system32\cmd.exe" /c timeout /t 5 & del /f /q "Malware_Path" & del "C:\ProgramData\*.dll" & exit
```

IOCs	Description
e978871a3a76c83f94e589fd22a91c7c1a58175ca5d2110b95d71b7805b25b8d	Stealc Sample
hxxp://185.106.94[.]206/4e815d9f1ec482dd.php	C2 Server

- Stealc: a copycat of Vidar and Raccoon infostealers gaining in popularity – Part 1
- Stealc: a copycat of Vidar and Raccoon infostealers gaining in popularity – Part 2
- Stealc: A new stealer emerges in 2023
- StealC Delivered via Deceptive Google Sheets
- Stealc Malware Technical Analysis Report

Source: https://glyc3rius.github.io/2023/10/stealc/