Predator The Thief: In-depth analysis (v2.3.5)

Published: 2018-10-15 · Archived: 2026-04-05 23:07:13 UTC

Well, it’s been a long time without some fresh new contents on my blog. I had some unexpected problems that kept me away from here and a lot of work (like my tracker) that explain this. But it’s time to come back (slowly) with some stuff.

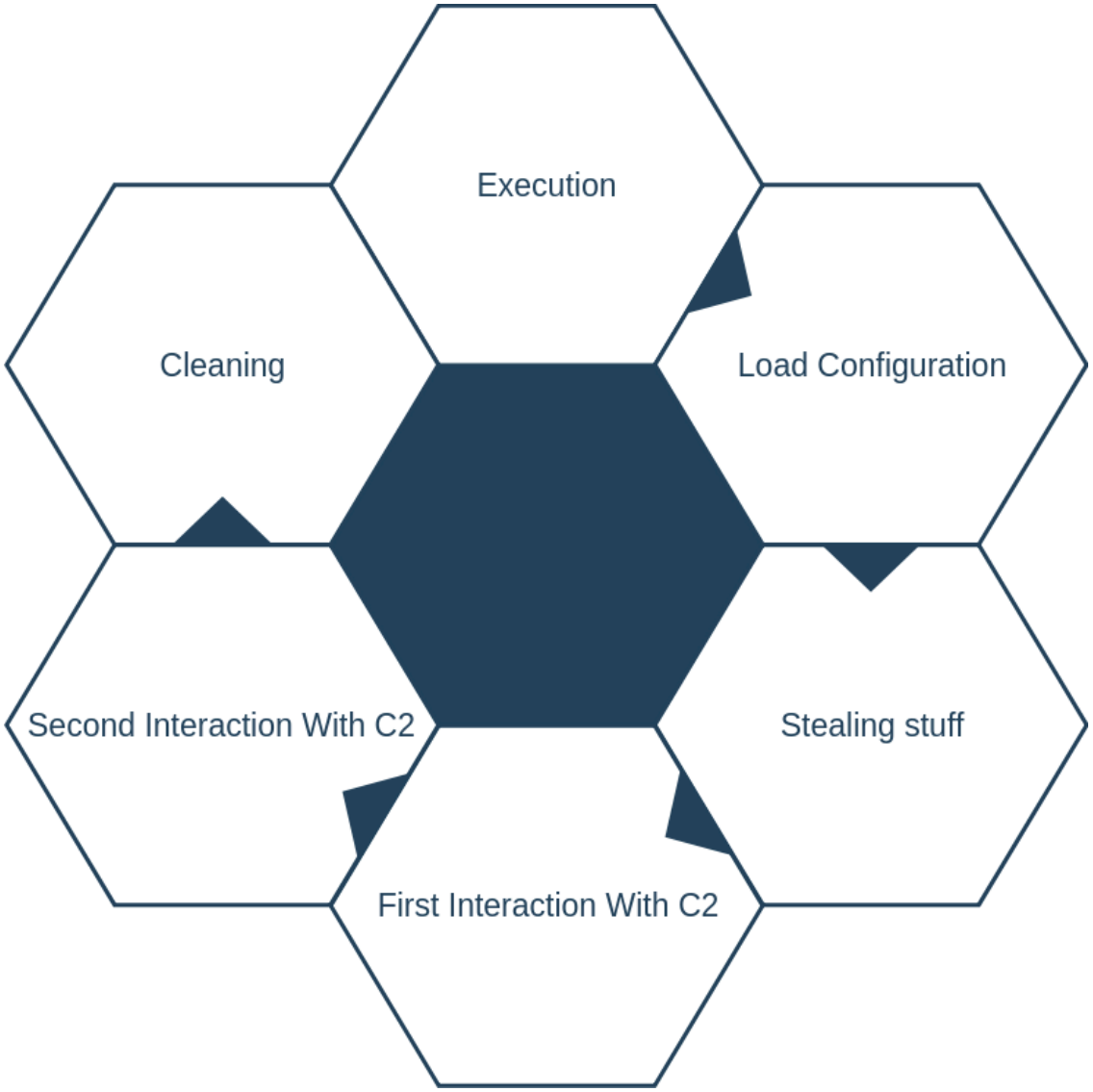
So today, this is an In-Depth analysis of one stealer: “Predator the thief”, written in C/C++. Like dozen others malware, it’s a ready to sell malware delivered as a builder & C2 panel package.

The goal is to explain step by step how this malware is working with a lot of extra explanations for some parts. This post is mainly addressed for junior reverse engineers or malware analysts who want for future purposes to understand and defeat some techniques/tricks easily.

So here we go!

Classical life cycle

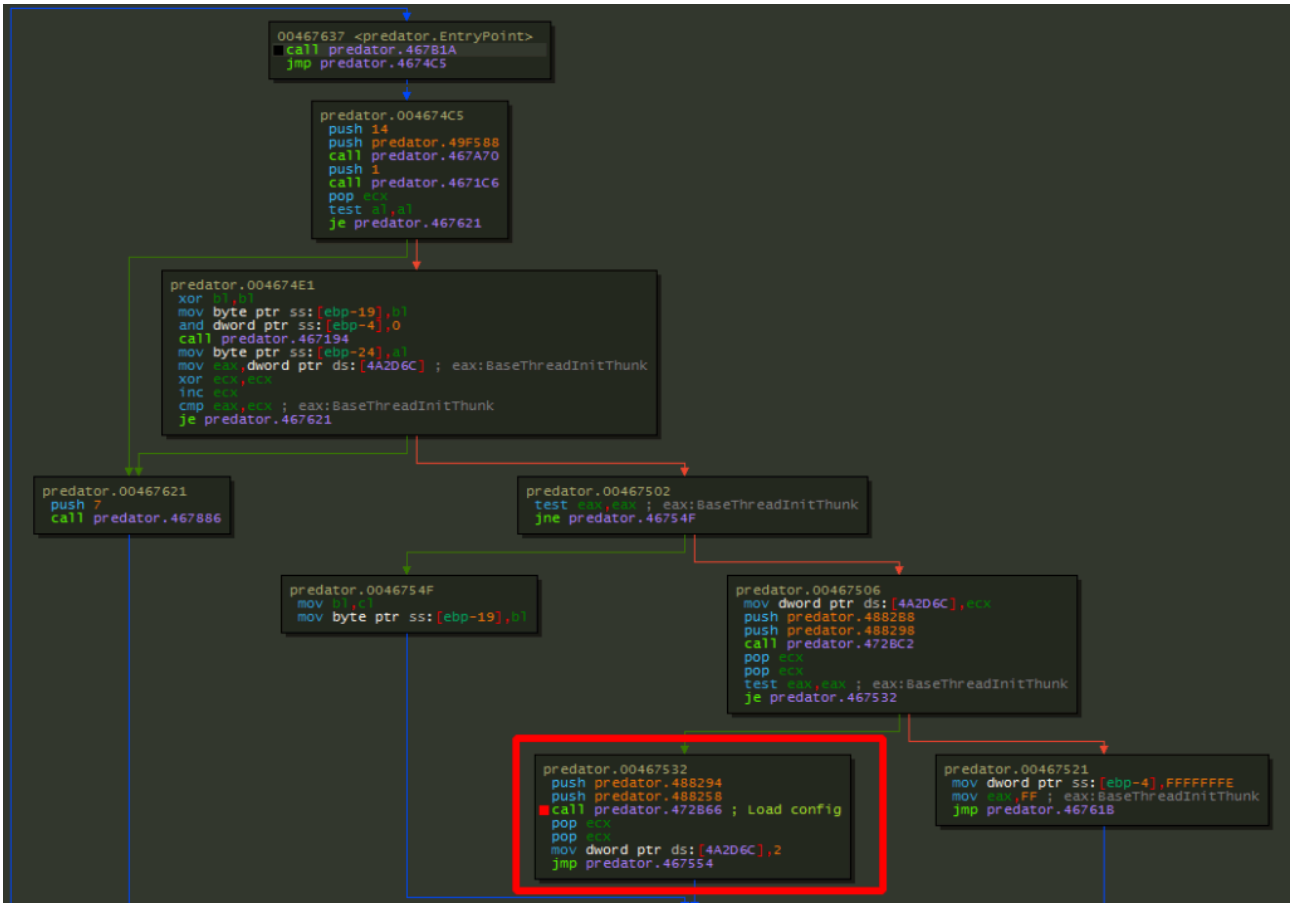
The execution order is almost the same, for most of the stealers nowadays. Changes are mostly varying between evading techniques and how they interact with the C2. For example, with Predator, the set up is quite simple but could vary if the attacker set up a loader on his C2.



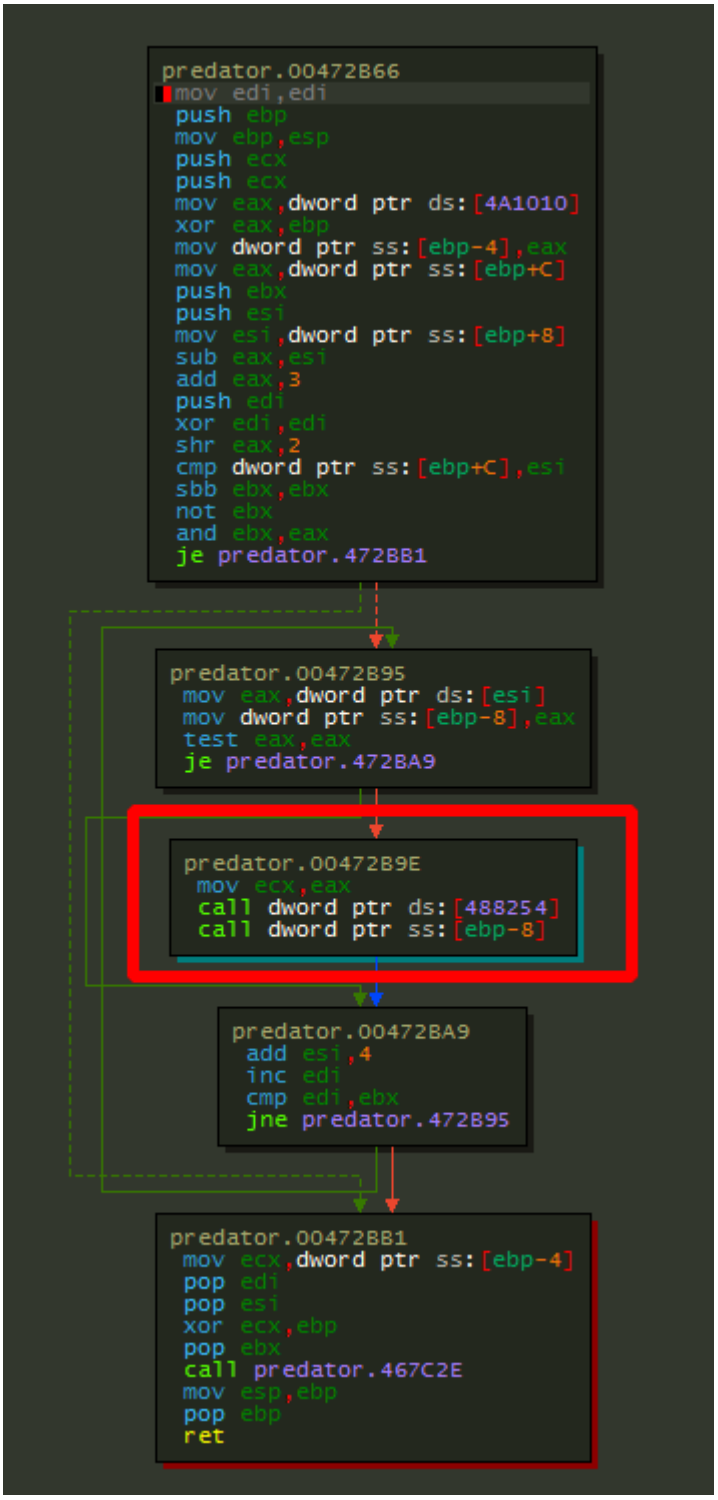
The life cycle of Predator the thief

Preparing the field

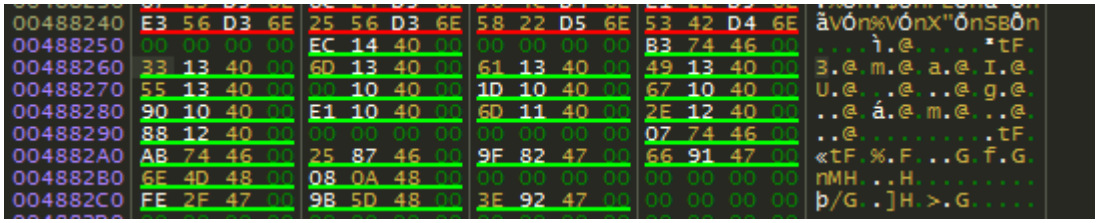
Before stealing sensitive data, Predator starts by setting up some basics stuff to be able to work correctly. Almost all the configuration is loaded into memory step by step.



So let’s put a breakpoint at “0x00472866” and inspect the code…



1. EBX is set to be the length of our loop (in our case here, it will be 0x0F)
2. ESI have all functions addresses stored



3. EAX, will grab one address from ESI and moves it into EBP-8

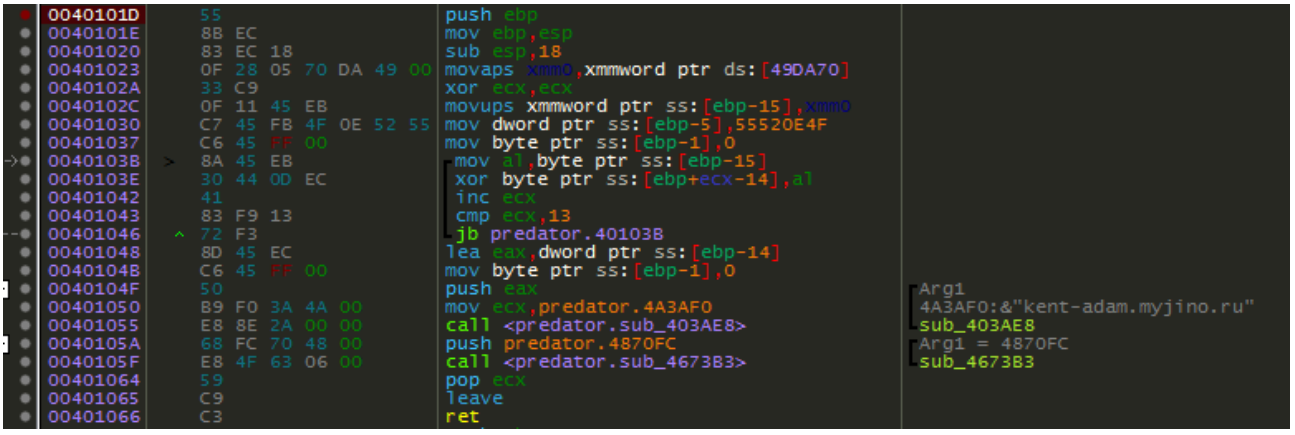
4. EBP is called, so at this point, a config function will unpack some data and saved it into the stack)
5. ESI position is now advanced by 4
6. EDI is incremented until reaching the same value as stored EBX
7. When the EDI == EBX, it means that all required configuration values are stored into the stack. The main part of the malware could start

So, for example, let's see what we have inside ***0040101D*** at 0x00488278

So with x32dbg, let's see what we have… with a simple command

```
Command: go 0x0040101D
```

As you can see, this is where the C2 is stored, uncovered and saved into the stack.



So what values are stored with this technique?

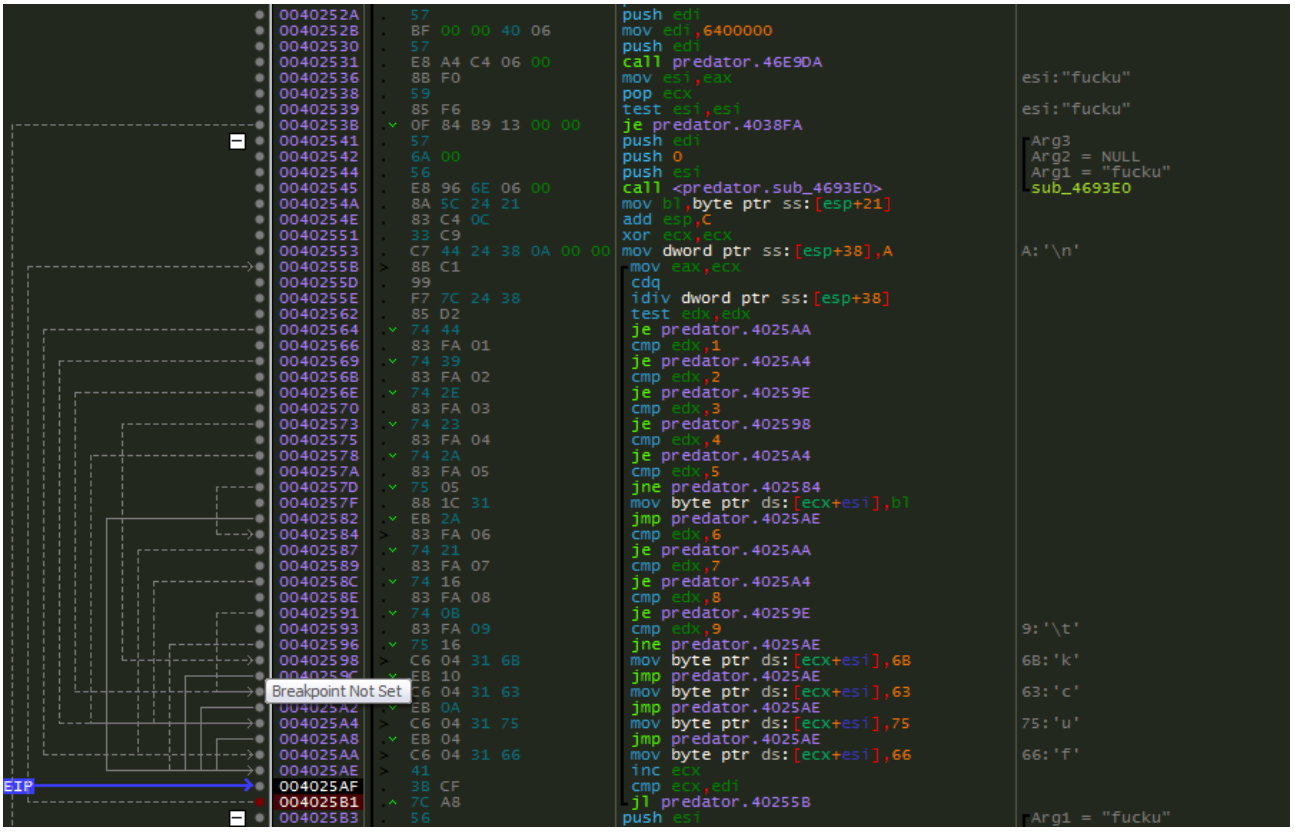
- C2 Domain
- %APPDATA% Folder
- Predator Folder
- temporary name of the archive predator file and position
- also, the name of the archive when it will send to the C2
- etc…

With the help of the %APPDATA%/Roaming path, the Predator folder is created (\ptst). Something notable with this is that it's hardcoded behind a Xor string and not generated randomly. By pure speculation, It could be a shortcut for "**P**redator **T**he **ST**ealer".

This is also the same constatation for the name of the temporary archive file during the stealing process: ***"zpar.zip"***.

The welcome message…

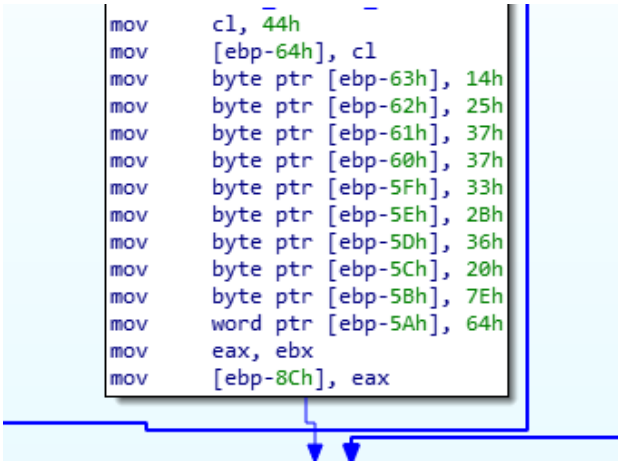
When you are positioned at the main module of the stealer, a lovely text looped over 0x06400000 times is addressed for people who want to reverse it.



Obfuscation Techniques

The thief who loves XOR (a little bit too much…)

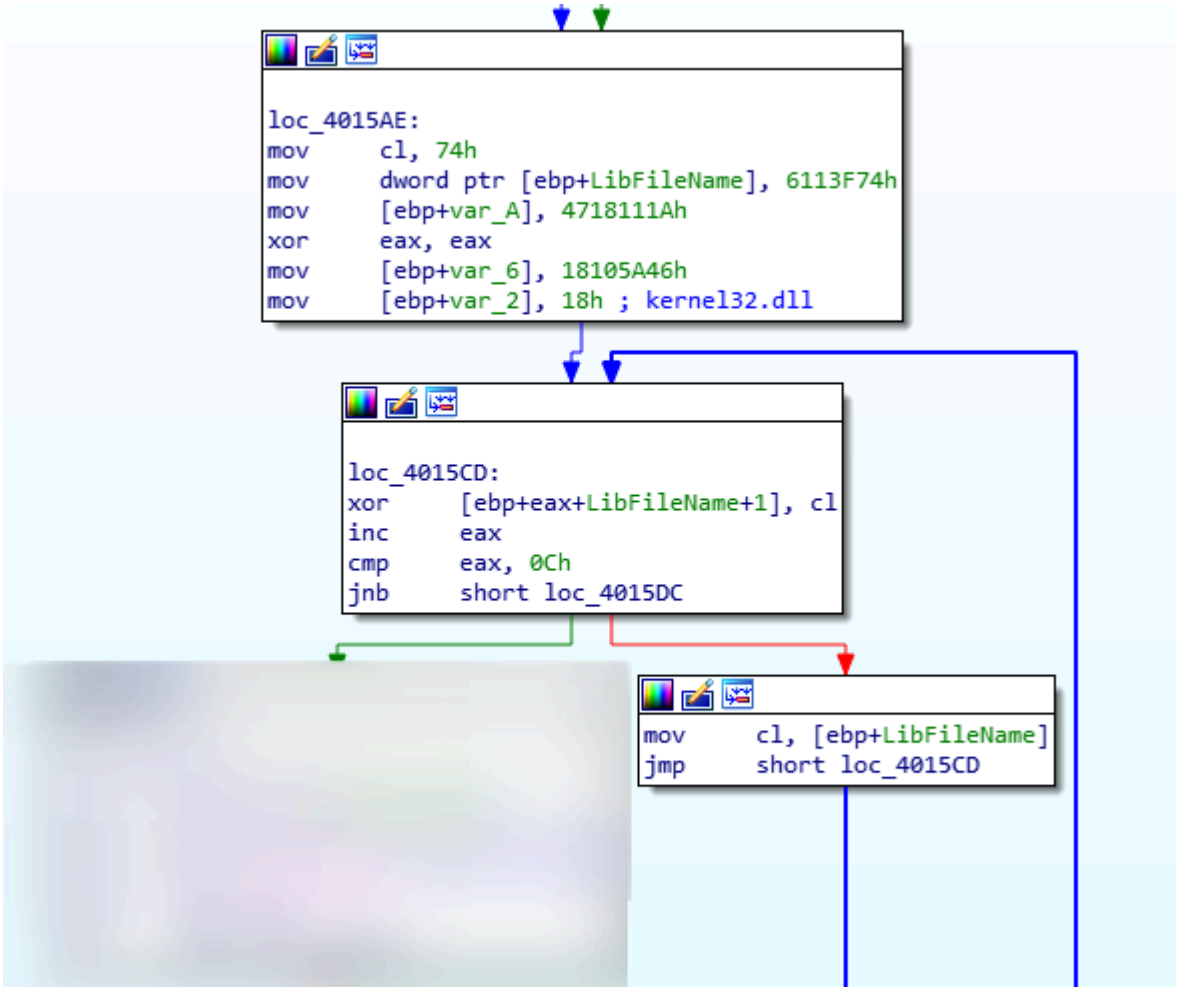
Almost all the strings from this stealer sample are XORed, even if this obfuscation technique is really easy to understand and one of the easier to decrypt. Here, its used at multiple forms just to slow down the analysis.



GetProcAddress Alternatives

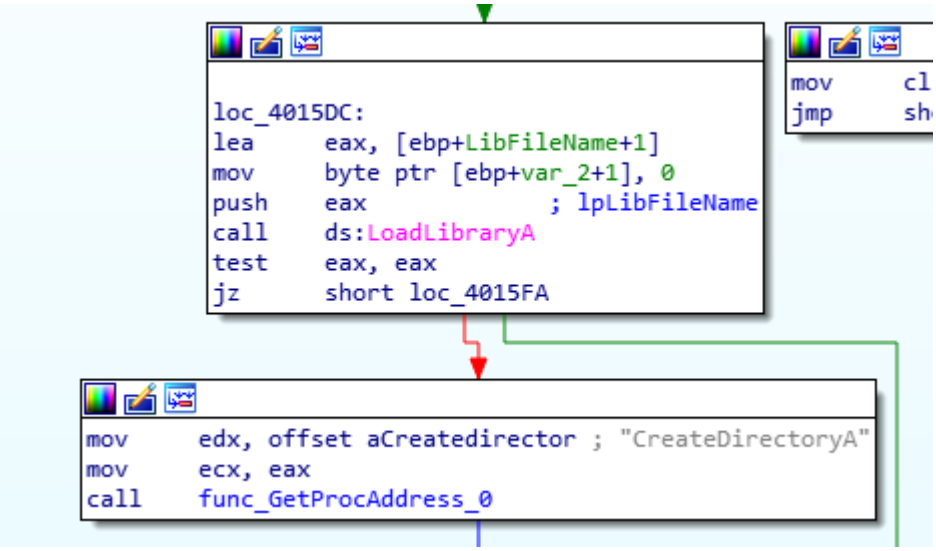
For avoiding to call directly modules from different libraries, it uses some classic stuff to search step by step a specific API request and stores it into a register. It permits to hide the direct call of the module into a simple register call.

So firstly, a XORed string (a DLL) is decrypted. So for this case, the kernel32.dll is required for the specific module that the malware wants to call.



When the decryption is done, this library is loaded with the help of "LoadLibraryA". Then, a clear text is pushed into EDX: "CreateDirectoryA"… This will be the module that the stealer wants to use.

The only thing that it needs now, its to retrieve the address of an exported function "CreateDirectoryA" from kernel32.dll. Usually, this is done with the help of GetProcAddress but this function is in fact not called and another trick is used to get the right value.



So this string and the **IMAGE_DOS_HEADER** of kernel32.dll are sent into “func_GetProcesAddress_0”. The idea is to get manually the pointer of the function address that we want with the help of the Export Table. So let’s see what we have in the in it…

```
struct IMAGE_EXPORT_DIRECTORY {
        long Characteristics;
        long TimeDateStamp;
        short MajorVersion;
        short MinorVersion;
        long Name;
        long Base;
        long NumberOfFunctions;
        long NumberOfNames;
        long *AddressOfFunctions;    <= This good boy
        long *AddressOfNames;        <= This good boy
        long *AddressOfNameOrdinals; <= This good boy
}
```

After inspecting the structure de IMAGE_EXPORT_DIRECTORY, three fields are mandatory :

- AddressOfFunctions – An Array who contains the relative value address (RVA) of the functions of the module.
- AddressOfNames – An array who stores with the ascending order of all functions from this module.
- AddressOfNamesOrdinals – An 16 bits array who contains all the associated ordinals of functions names based on the AddressOfNames.

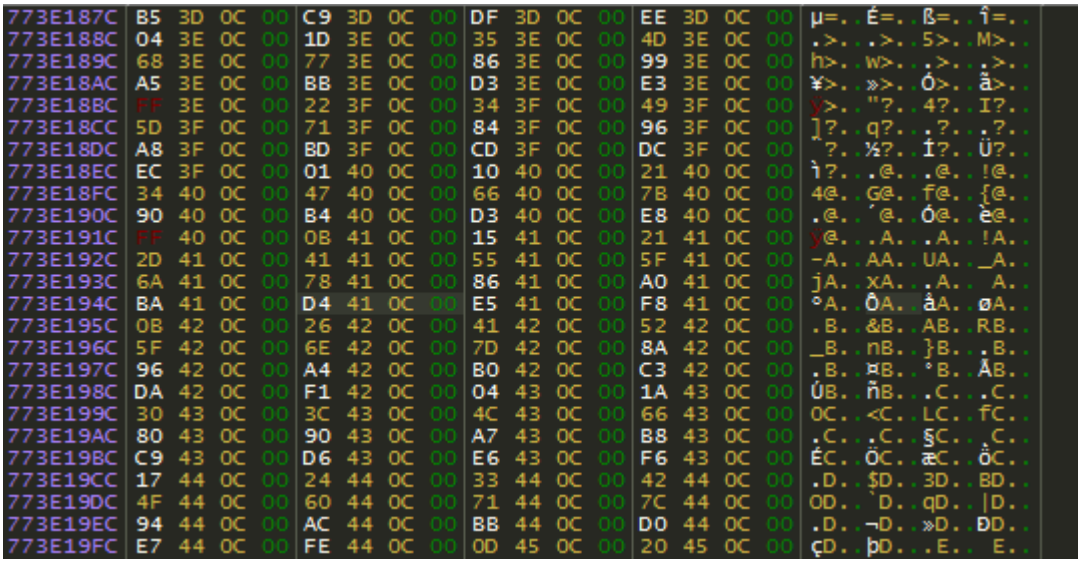
source

So after saving the absolute position of these 3 arrays, the loop is simple

1. Grab the RVA of one function
2. Get the name of this function
3. Compare the string with the desired one.

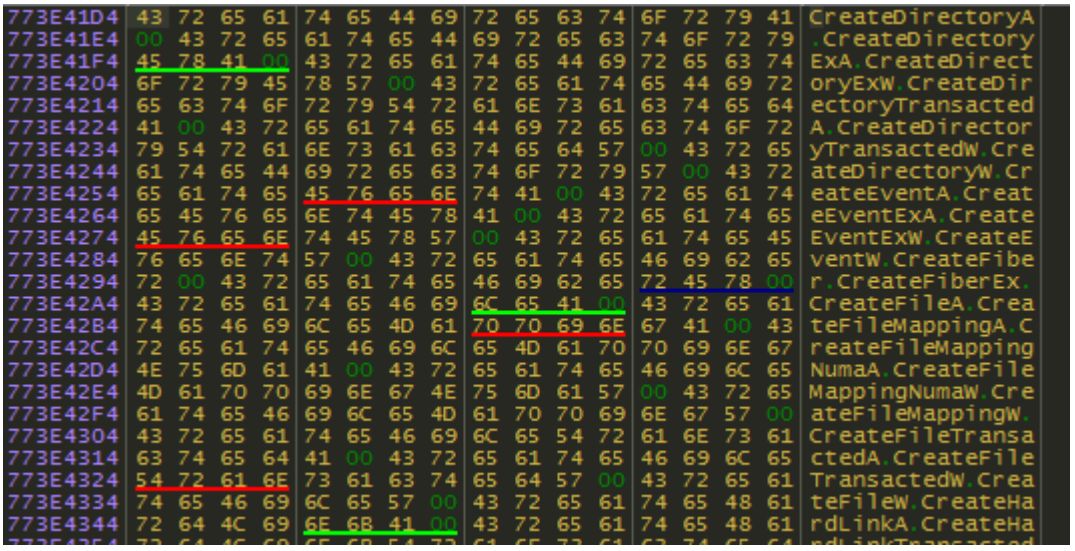
So let's see in details to understand everything :

If we dig into **ds:[eax+edx*4],** this where is stored all relative value address of the kernel32.dll export table functions.



With the next instruction **add eax,ecx**. This remains to go at the exact position of the string value in the "AddressOfNames" array.

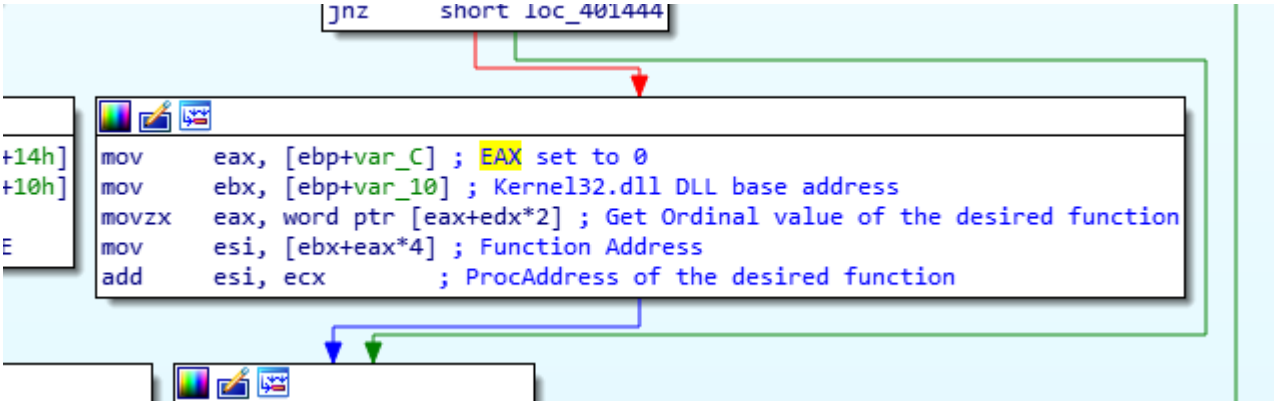
```
DLLBaseAddress + AddressOfNameRVA[i] = Function Name
   751F0000    +       0C41D4        = CreateDirectoryA
```



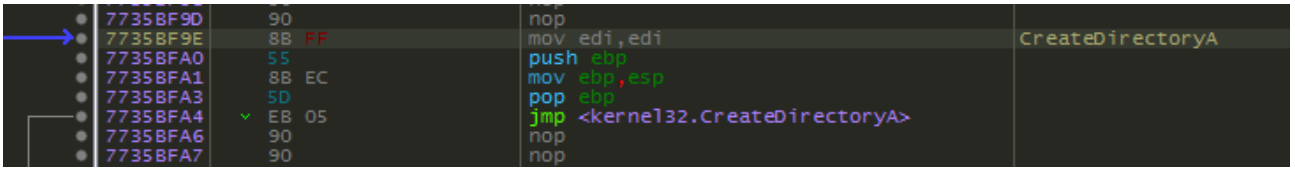
The comparison is matching, now it needs to store the “procAddress. So First the Ordinal Number of the function is saved. Then with the help of this value, the Function Address position is grabbed and saved into ESI.

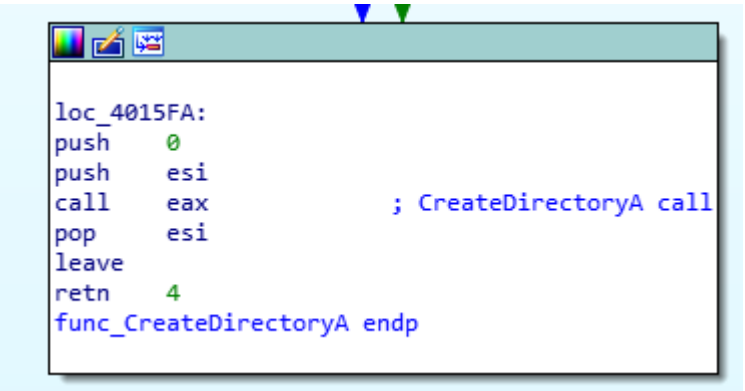
```
ADD           ESI, ECX
ProcAddress = Function Address + DLLBaseAddress
```

In disassembly, it looks like this :



Let’s inspect the code at the specific procAddress…

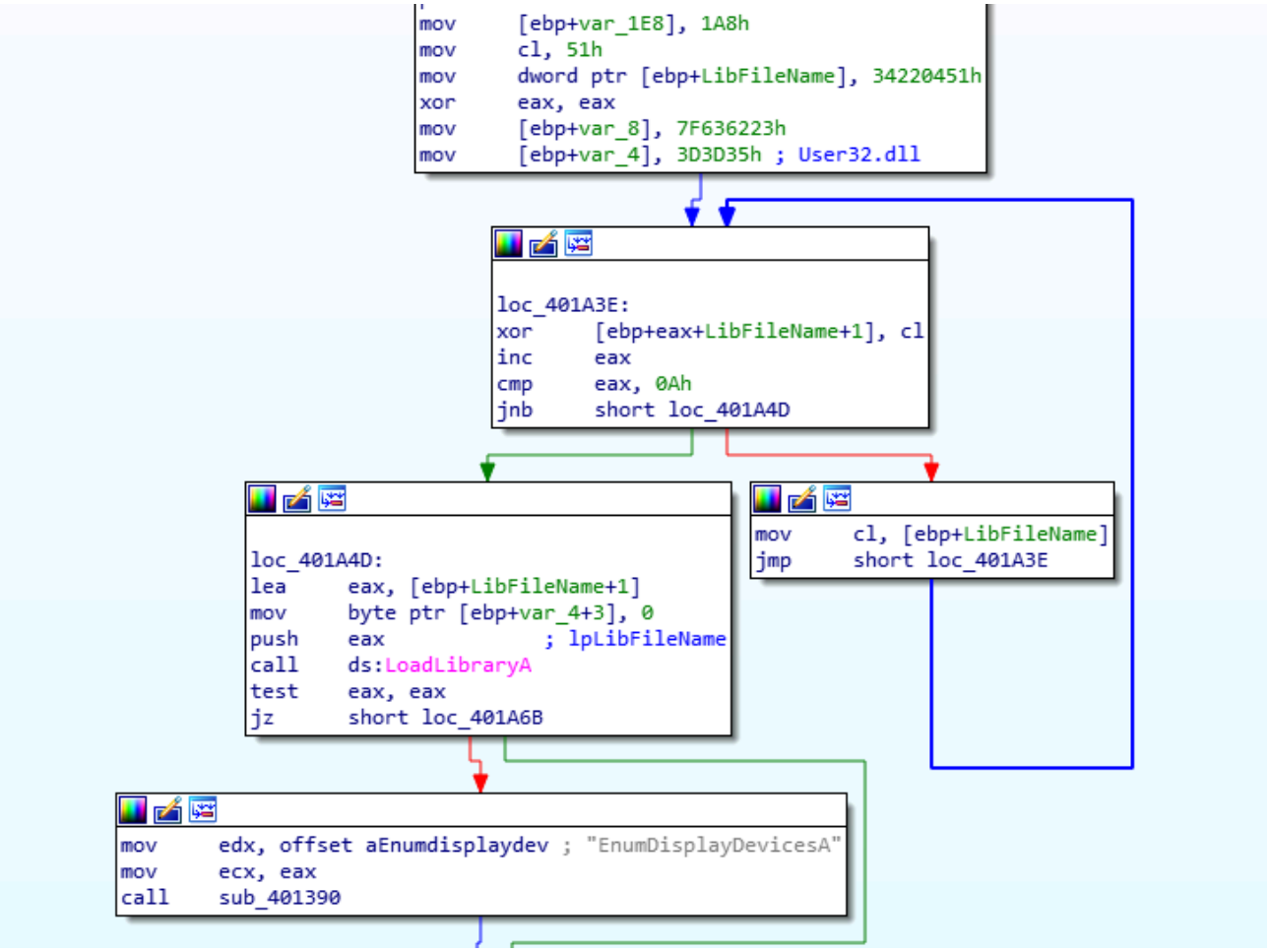




So everything is done, the address of the function is now stored into EAX and it only needs now to be called.

Anti-VM Technique

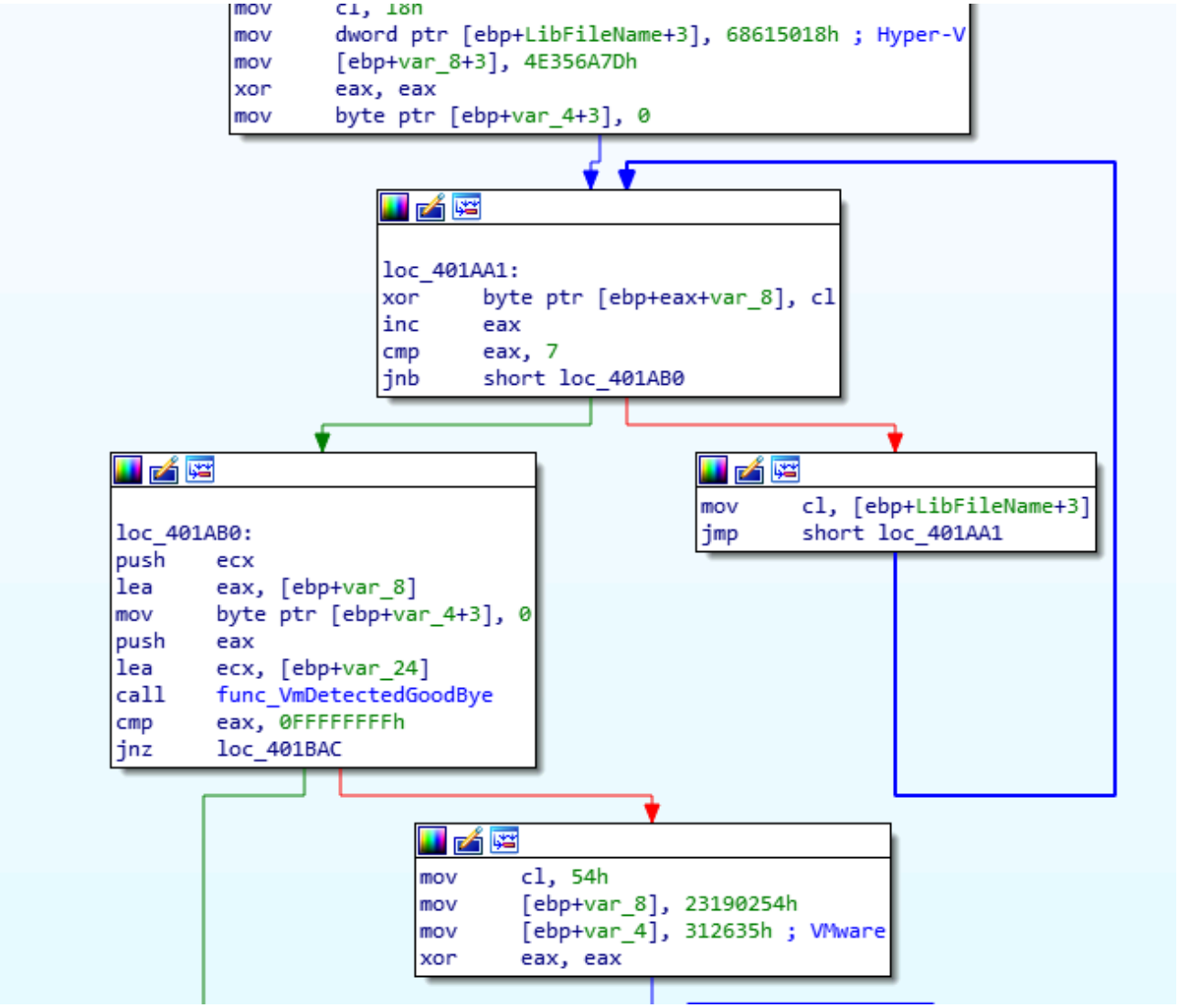
Here is used a simple Anti-VM Technique to check if this stealer is launched on a virtual machine. This is also the only Anti-Detection trick used on Predator.



First, User32.dll (Xored) is dynamically loaded with the help of "LoadLibraryA", Then "EnumDisplayDevicesA" module is requested with the help of User32.dll. The idea here is to get the value of the "Device Description" of the current display used.

When it's done, the result is checked with some values (obviously xored too) :

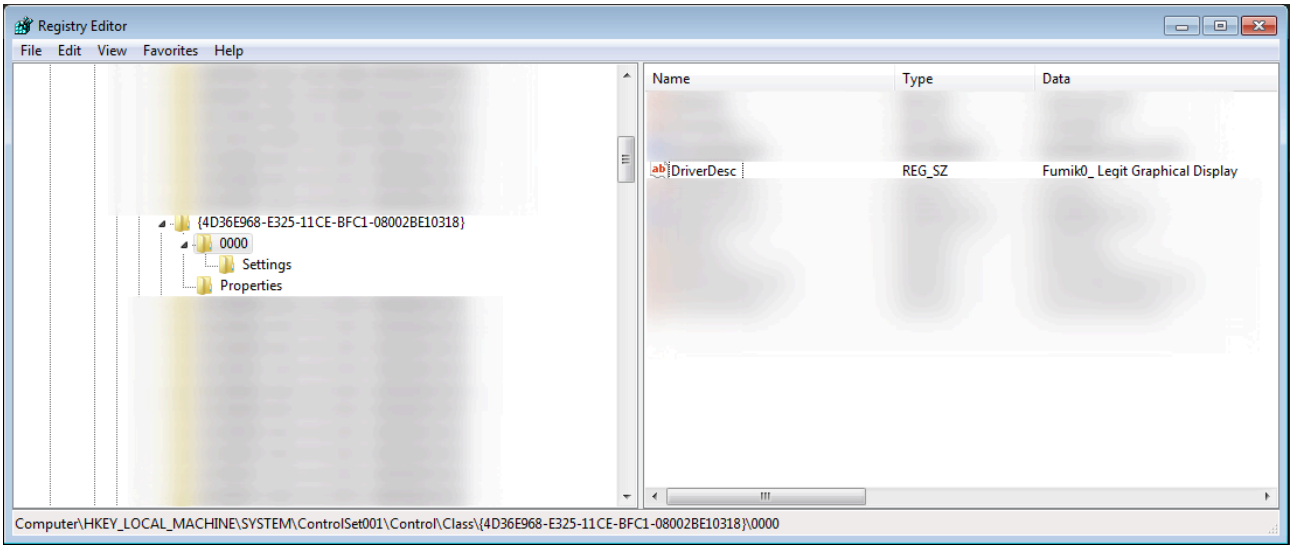
- Hyper-V
- VMware
- VirtualBox



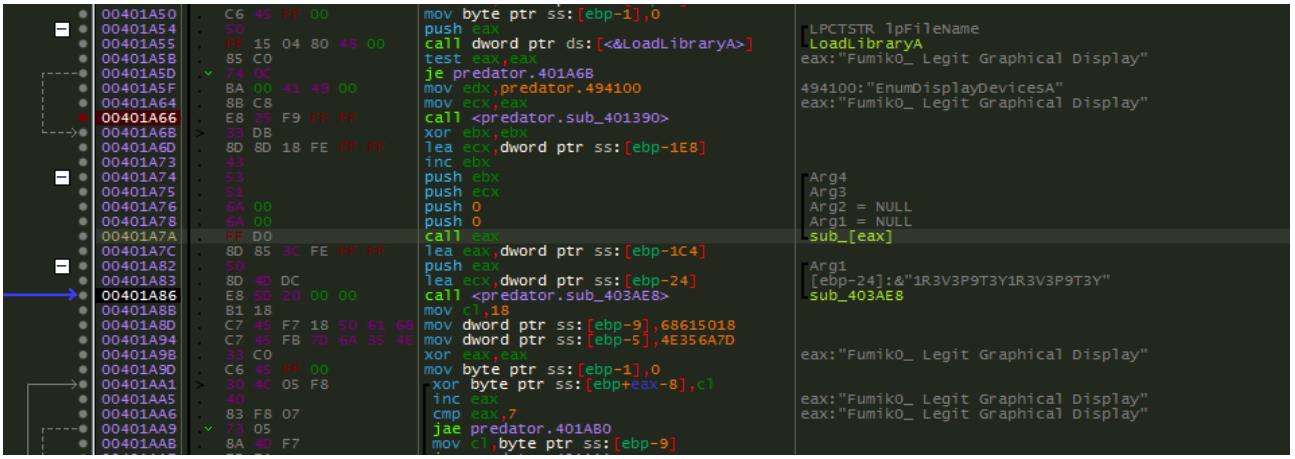
If the string matches, you are redirected to a function renamed here "func_VmDetectedGoodBye.

How to By-Pass this Anti-VM technique?

For avoiding this simple trick, the goal is to modify the REG_SZ value of "DriverDesc" into {4d36e968-e325-11ce-bfc1-08002be10318} to something else.



And voilà!



Stealing Part

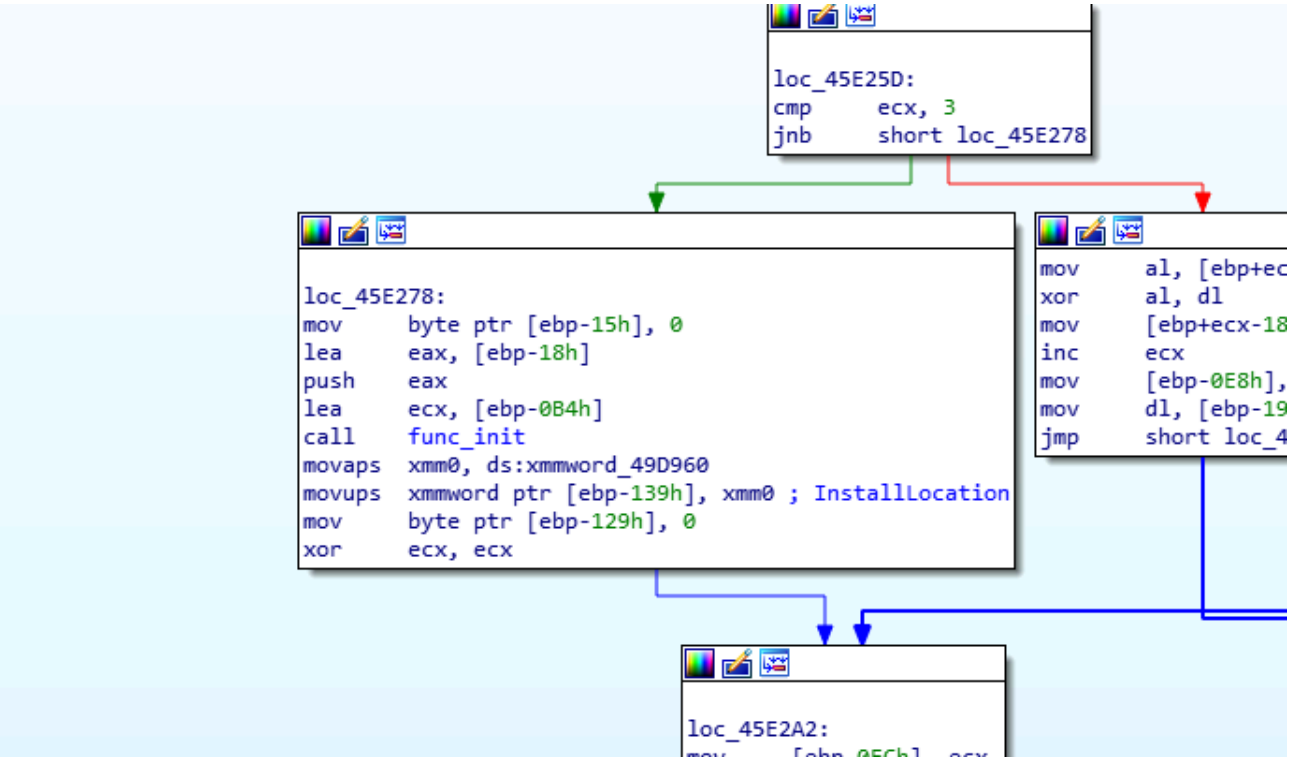
Let's talk about the main subject… How this stealer is organized… As far I disassemble the code, this is all the folders that the malware is setting on the "ptst" repository before sending it as an archive on the C2.

- Folder
  - **Files**: Contains all classical text/documents files at specifics paths
  - **FileZilla**: Grab one or two files from this FTP
  - **WinFTP**: Grab one file from this FTP
  - **Cookies**: Saved stolen cookies from different browsers
  - **General**: Generic Data
  - **Steam**: Steal login account data
  - **Discord**: Steal login account data
- Files
  - Information.log
  - Screenshot.jpeg <= Screenshot of the current screen

Telegram

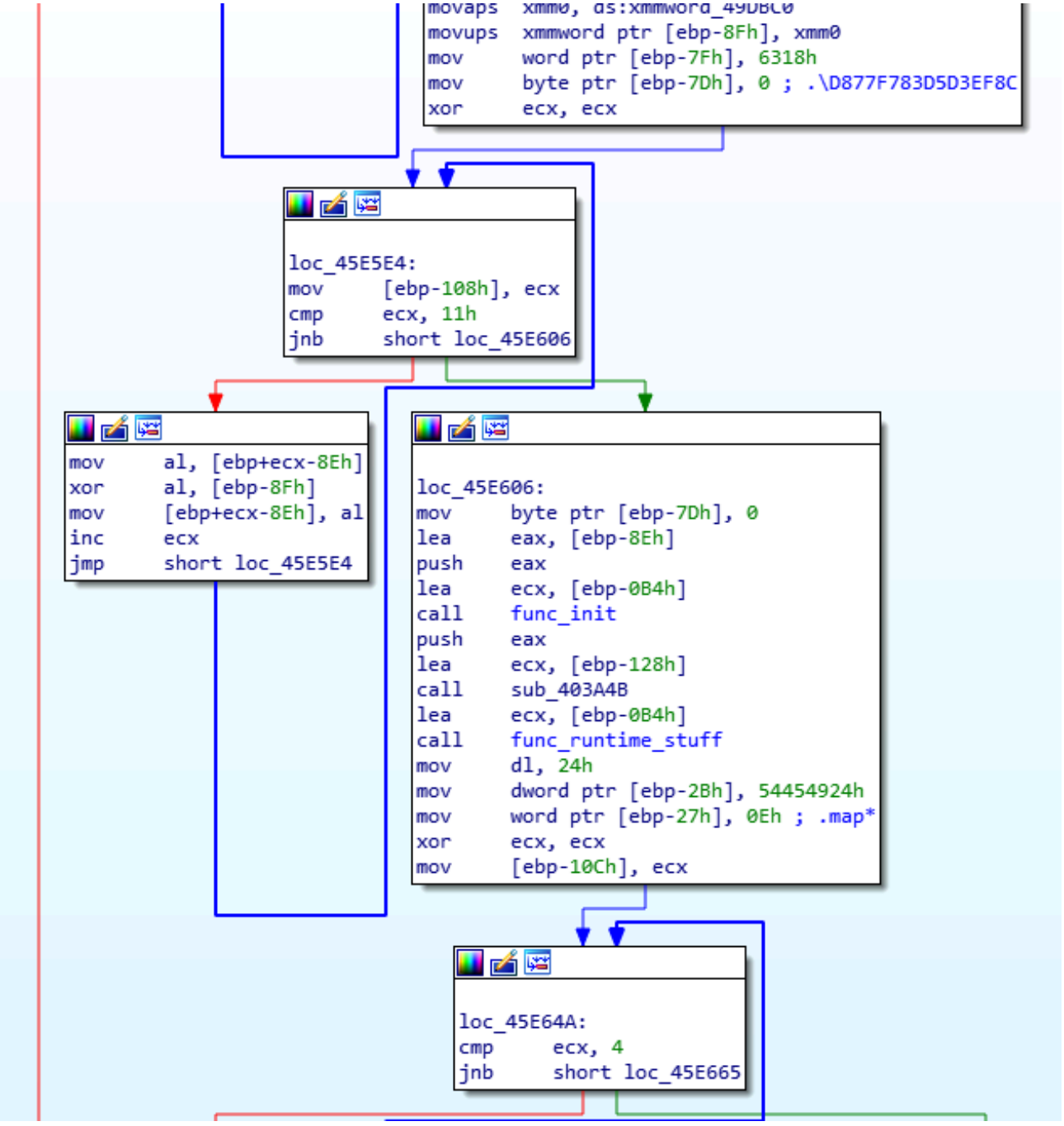
For checking if Telegram is installed on the machine, the malware is checking if the KeyPath "Software\Microsoft\Windows\CurrentVersion\Uninstall\{53F49750-6209-4FBF-9CA8-7A333C87D1ED}_is1" exists on the machine.

So let's inspect what we have inside this "KeyPath"? After digging into the code, the stealer will request the value of "InstallLocation" because of this where Telegram is installed currently on the machine.



Step by step, the path is recreated (also always, all strings are xored) :

- %TELEGRAM_PATH%
- \Telegram Desktop
- \tdata
- \D877F783D5D3EF8C



The folder “D877F783D5D3EF8C” is where all Telegram cache is stored. This is the sensitive data that the stealer wants to grab. Also during the process, the file map* (i.e: map1) is also checked and this file is, in fact, the encryption key. So if someone grabs everything for this folder, this leads the attacker to have an access (login without prompt) into the victim account.

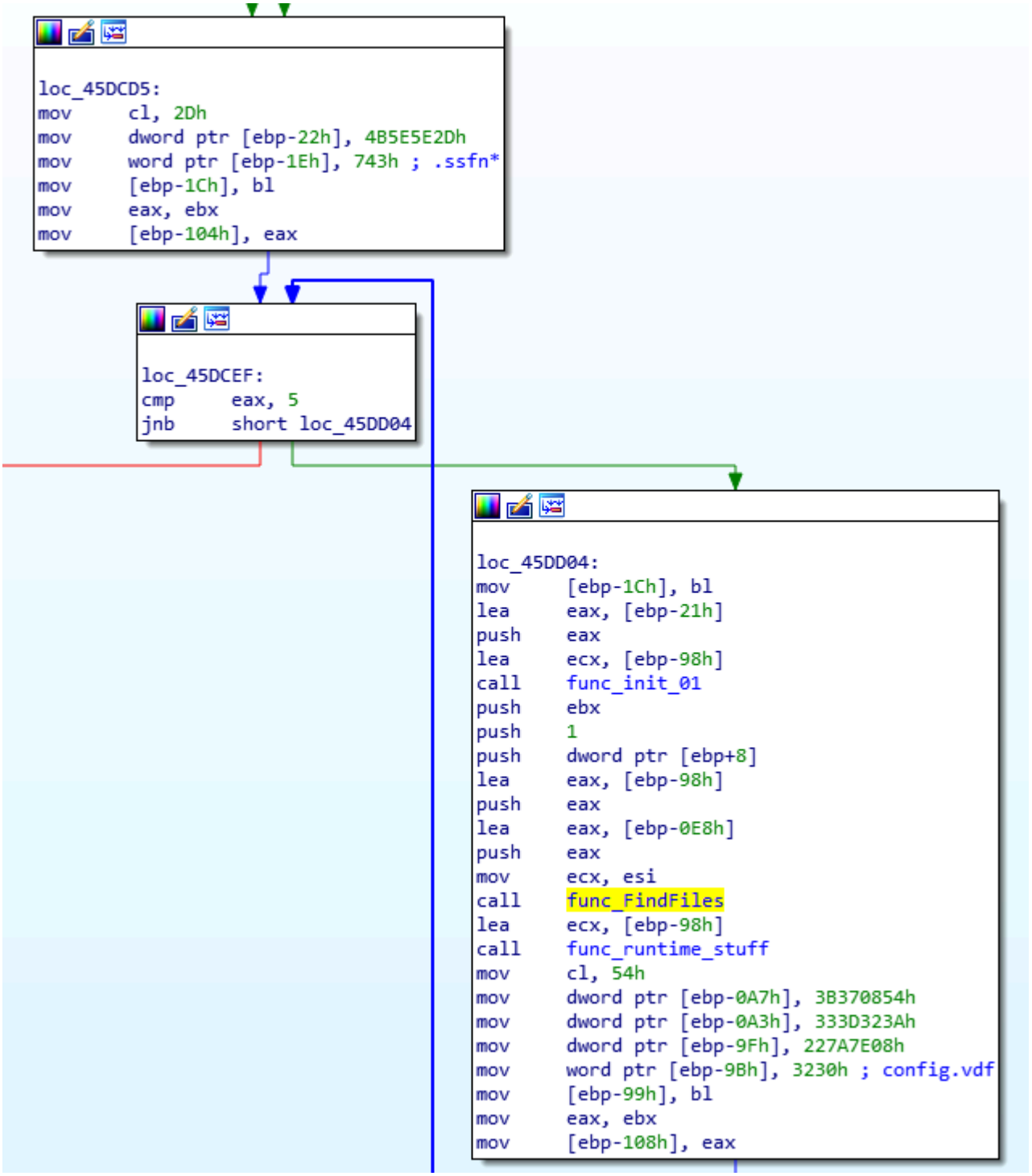
Steam

The technique used by the stealer to get information for one software will remain the same for the next events (for most of the cases). This greatly facilitates the understanding of this malware.

So first, it’s checking the “SteamPath” key value at “HKCU\Software\Valve\Steam” to grab the correct Steam repository. This value is after concatenating with a bunch of files that are necessary to compromise a Steam

Account.

So it will check first if ssfn files are present on the machine with the help of “func_FindFiles”, if it matches, they are duplicated into the temporary malware folder stored on %APPDATA%/XXXX. Then do the same things with config.vdf



So what the point with these files? First, after some research, a post on Reddit was quite interesting. it explained that ssfn files permit to by-pass SteamGuard during the user log-on.

randomstranger454 1 point · 3 years ago

If you still have the whole drive, copy the steam client folder to your new pc and launch steam from there, it should launch without asking you for a steam guard code. Same for your firefox or chrome profile, if you want to avoid steam guard there.

It's the reason why I have installed steam and moved my firefox profile in another drive.

Share Report Save

jordguitar 31 1 point · 3 years ago

Steamguard would detect new hardware and ask for a code.

Share Report Save

randomstranger454 2 points · 3 years ago

Steamguard is hardware agnostic, why do you think there are so many scams where they ask to upload your SSFN file or trojans that grab it, do you think they emulate the associated pc hardware?

Anyway to be 100% sure I copied the steam client folder from my 2nd pc and logged in a steam account that had never logged in before in my 1st pc and no steam guard prompt popped up.

Share Report Save

Now what the point of the second file? this is where you could know some information about the user account and all the applications that are installed on the machine. Also, if the ConnectCache field is found on this one, it is possible to log into the stolen account without steam authentication prompt. if you are curious, this pattern is represented just like this :

```
"ConnectCache"
{
	"STEAM_USERNAME_IN_CRC32_FORMAT" "SOME_HEX_STUFF"
}
```

The last file, that the stealer wants to grab is “loginusers.vdf”. This one could be used for multiple purposes but mainly for setting the account in offline mode manually.

```
push    eax
call    func_GetCopyFileA
lea     ecx, [ebp-98h]
call    func_runtime_stuff
lea     ecx, [ebp-130h]
call    func_runtime_stuff
lea     ecx, [ebp-0B0h]
call    func_runtime_stuff
lea     ecx, [ebp-0C8h]
call    func_runtime_stuff
movaps  xmm0, ds:xmmword_49D670 ; .loginusers.vdf
movups  xmmword ptr [ebp-128h], xmm0
mov     ecx, ebx
```

For more details on the subject there a nice report made by Kapersky for this:

- Steam Stealers

Wallets

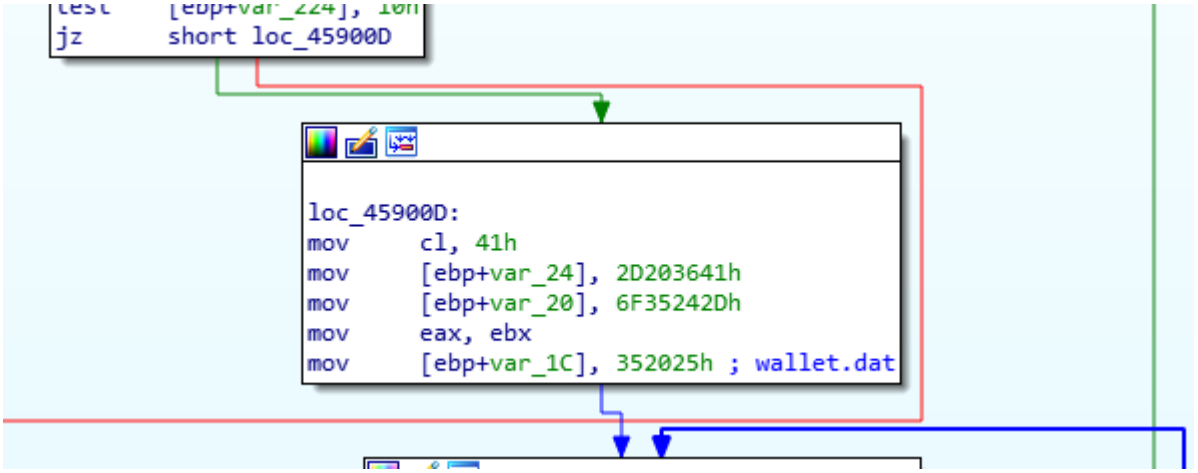
The stealer is supporting multiple digital wallets such as :

- Ethereum
- Multibit
- Electrum
- Armory
- Bytecoin
- Bitcoin
- Etc…

The functionality is rudimentary but it's enough to grab specific files such as :

- *.wallet
- *.dat

And as usual, all the strings are XORed.



FTP software

The stealer supports two FTP software :

- Filezilla
- WInFTP

It's really rudimentary because he only search for three files, and they are available a simple copy to the predator is done :

- %APPDATA%\Filezilla\sitemanager.xml
- %APPDATA%\Filezilla\recentservers.xml
- %PROGRAMFILES%\WinFtp Client\Favorites.dat

Browsers

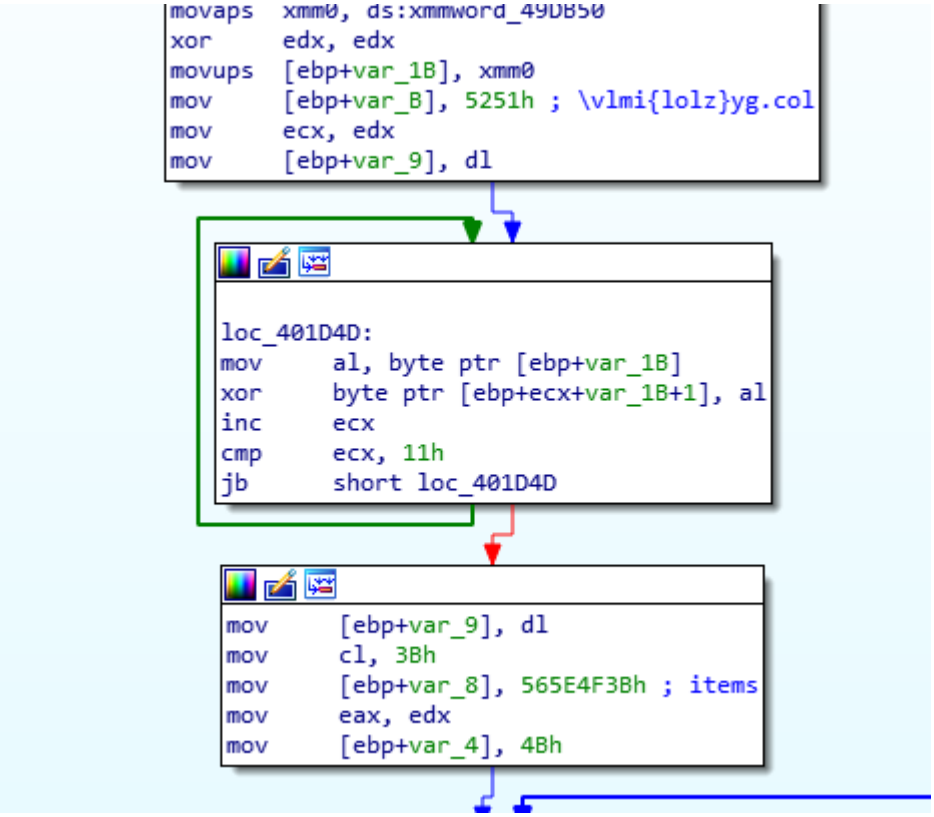
It's not necessary to have some deeper explanation about what kind of file the stealer will focus on browsers. There is currently a dozen articles that explain how this kind of malware manages to steal web data. I recommend you to read this article made by @coldshell about an example of overview and well detailed.

As usual, popular Chrome-based & Firefox-based browsers and also Opera are mainly targeted by Predator.

This is the current official list supported by this stealer :

- Amigo
- BlackHawk
- Chromium
- Comodo Dragon
- Cyberfox
- Epic Privacy Browser
- Google Chrome
- IceCat
- K-Meleon
- Kometa
- Maxthon5
- Mozilla Firefox
- Nichrome
- Opera
- Orbitum
- Pale Moon
- Sputnik
- Torch
- Vivaldi
- Waterfox
- Etc…

This one is also using SQLite for extracting data from browsers and using and saved them into a temporary file name “vlmi{lulz}yg.col”.

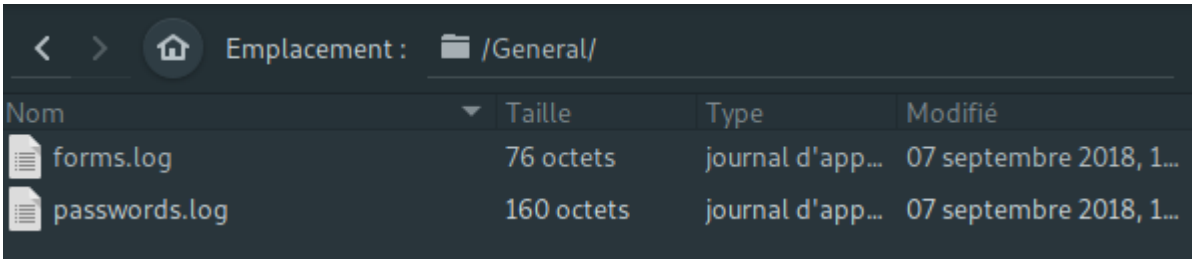


So the task is simple :

- Stole SQL Browser file
- Extract data with the help of SQLite and put into a temporary file
- Then read and save it into a text file with a specific name (for each browser).

When forms data or credentials are found they’re saved into two files on the General repository :

- forms.log
- password.log
- cards.log



Discord

If discord is detected on the machine, the stealer will search and copy the “https_discordapp_*localstorage” file into the “ptst” folder. This file contains all sensitive information about the account and could permit some authentication without a prompt login if this one is pushed into the correct directory of the attacker machine.

```
mov     [ebp-4], ebx
movaps  xmm0, ds:xmmword_49D800
movups  xmmword ptr [ebp-50h], xmm0
mov     dword ptr [ebp-40h], 465B4067h
mov     dword ptr [ebp-3Ch], 515355h ; .\discord\Local storage
mov     ecx, ebx
```

```
lea     ecx, [ebp-84h]
call    func_runtime_stuff
movaps  xmm0, ds:xmmword_49DCD0
movups  xmmword ptr [ebp-0A8h], xmm0
movaps  xmm0, ds:xmmword_49DCC0
movups  xmmword ptr [ebp-98h], xmm0
mov     dword ptr [ebp-88h], 6A686Eh ; https_discordpage | https_discordapp
mov     ecx, ebx
```

```
mov     ecx, esi
call    func_FindFiles
lea     ecx, [ebp-50h]
call    func_runtime_stuff
movaps  xmm0, ds:xmmword_49D7A0
movups  xmmword ptr [ebp-7Dh], xmm0 ; *.localstorage
mov     [ebp-6Dh], bl
mov     ecx, ebx
```

Predator is inspecting multiple places…

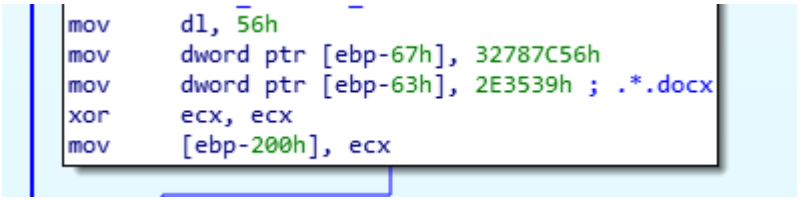
This stealer is stealing data from 3 strategical folders :

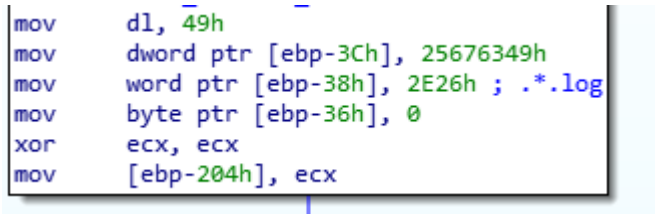
- Desktop
- Downloads
- Documents

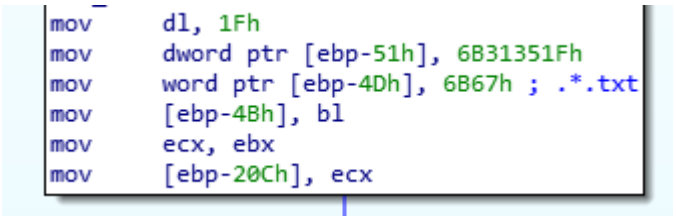
Each time, the task will be the same, it will search 4 type of files with the help of GetFileAttributesA :

- *.doc
- *.docx
- *.txt
- *.log

```
mov     dl, 4
mov     dword ptr [ebp-43h], 602A2E04h
mov     word ptr [ebp-3Fh], 676Bh ; .*.doc
mov     byte ptr [ebp-3Dh], 0
xor     ecx, ecx
mov     [ebp-208h], ecx
```







When it matches, they have copied into a folder named “Files”.

Information.log

When tasks are done, the malware starts generating a summarize file, who contains some specific and sensitive data from the machine victim beside the file “Information.log”. For DFIR, this file is the artifact to identify the name of the malware because it contains the name and the specific version.

So first, it writes the Username of the user that has executed the payload, the computer name, and the OS Version.

```
User name: lolilol
Machine name: Computer
OS version: Windoge 10
```

Then copy the content of the clipboard with the help of GetClipBoardData

```
Current clipboard:
--------------
Omelette du fromage
```

Let’s continue the process…

```
Startup folder: C:\Users\lolilol\AppData\Local\Temp\predator.exe
```

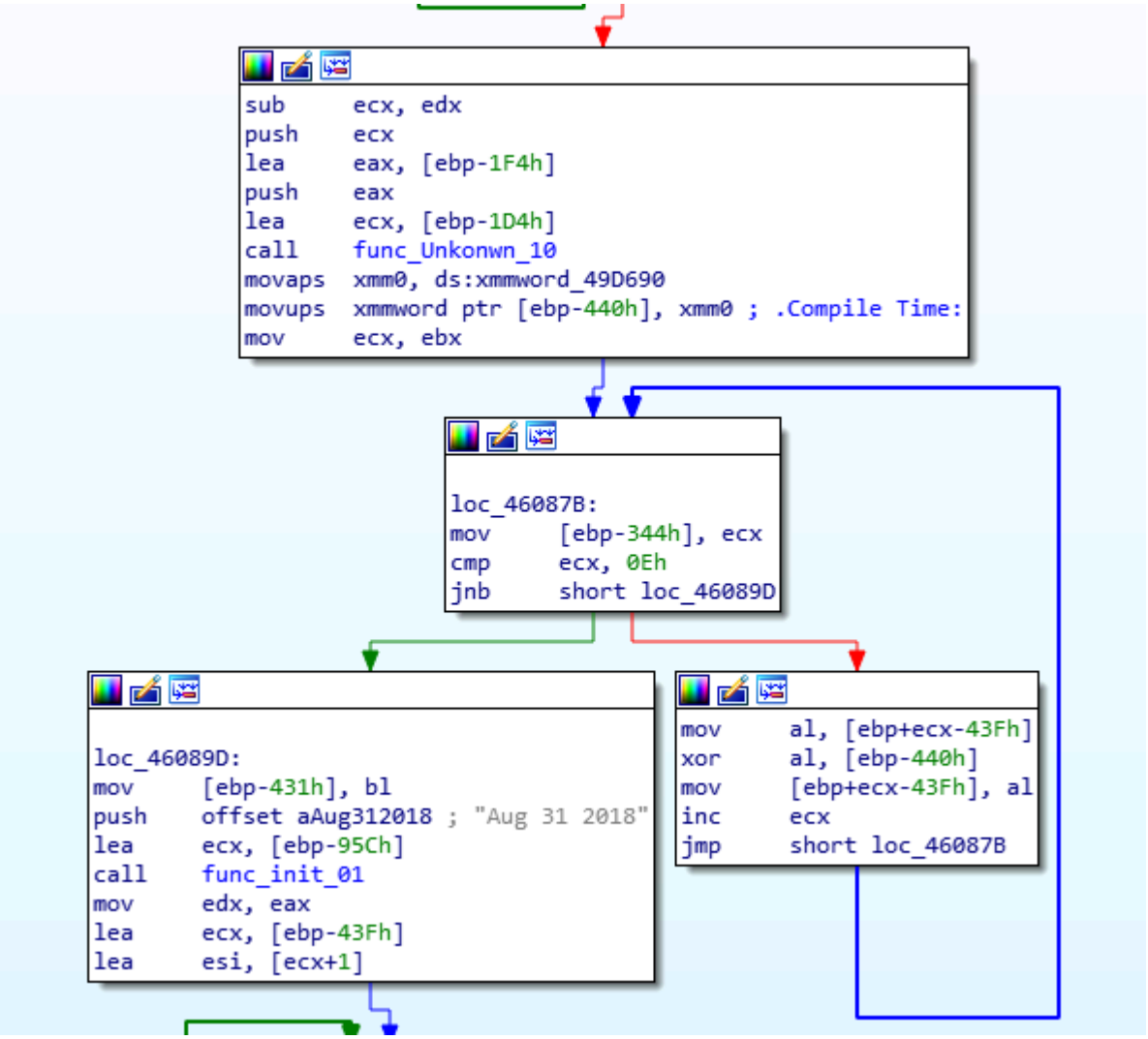
Some classic specification about the machine is requested and saved into the file.

```
CPU info: Some bad CPU | Amount of kernels: 128 (Current CPU usage: 46.112917%)
GPU info: Fumik0_ graphical display
Amount of RAM: 12 GB (Current RAM usage: 240 MB)
Screen resolution: 1900x1005
```

Then, all the user accounts are indicated

```
Computer users:
lolilol
Administrator
All Users
Default
Default User
Public
```

The last part is about some exotics information that is quite awkward in fact… Firstly, for some reasons that I don't want to understand, there is the compile time hardcoded on the payload.



Then the second exotic data saved into ***Information.log*** is the grabbing execution time for stealing contents from the machine… This information could be useful for debugging some tweaks with the features.

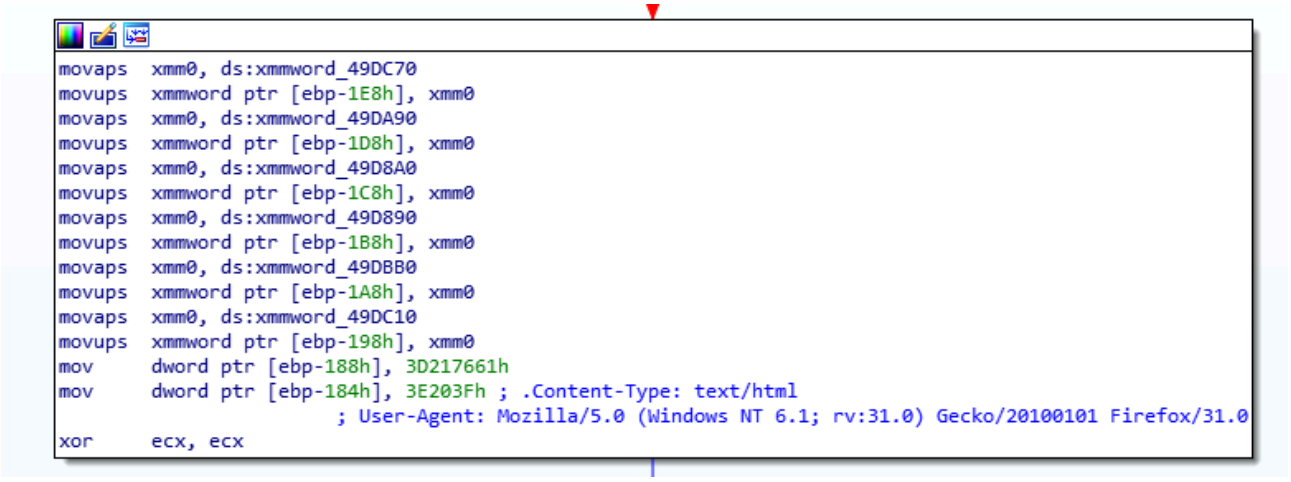
```
Additional information:
Compile time: Aug 31 2018
Grabbing time: 0.359375 second(s)
```

C2 Communications

For finishing the information.log, a GET request is made for getting some network data about the victim…

First, it set up the request by uncovered some Data like :

- A user-agent
- The content-type

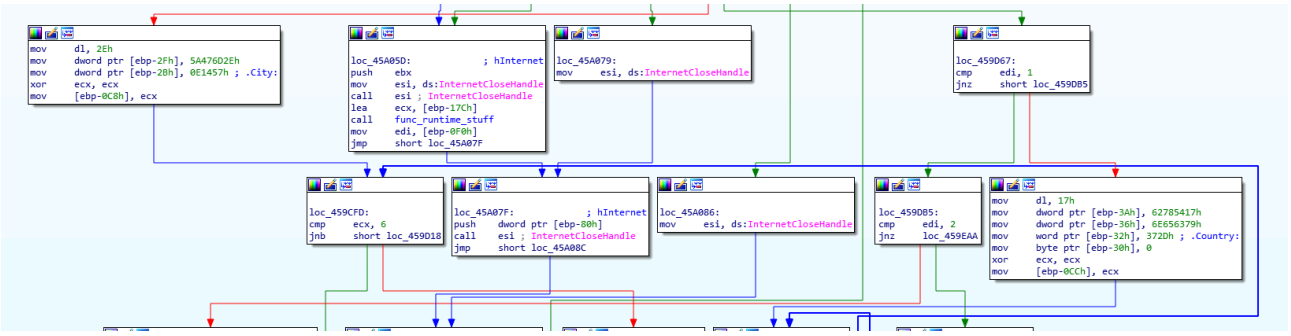


- The API URL (/api/info.get)

We can have for example this result :

```
Amsterdam;Netherlands;52.3702;4.89517;51.15.43.205;Europe/Amsterdam;1012;
```

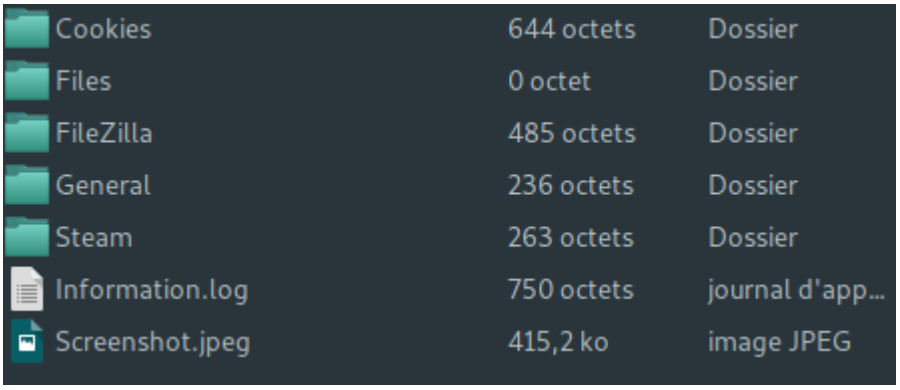
When the request is done, the data is consolidated step by step with the help of different loops and conditions.



When the task is done, there are saved into Information.log

```
City: Nopeland
Country:  NopeCountry
Coordinates: XX.XXXX N, X.XXXX W
IP: XXX.XXX.XXX.XXX
Timezone: Nowhere
Zip code: XXXXX
```

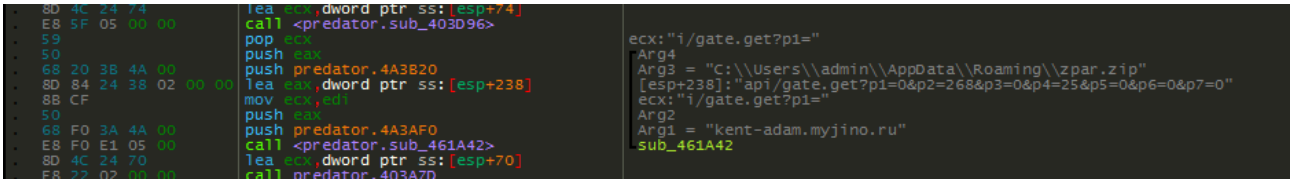
The Archive is not complete, it only needs for the stealer to send it to the C2.



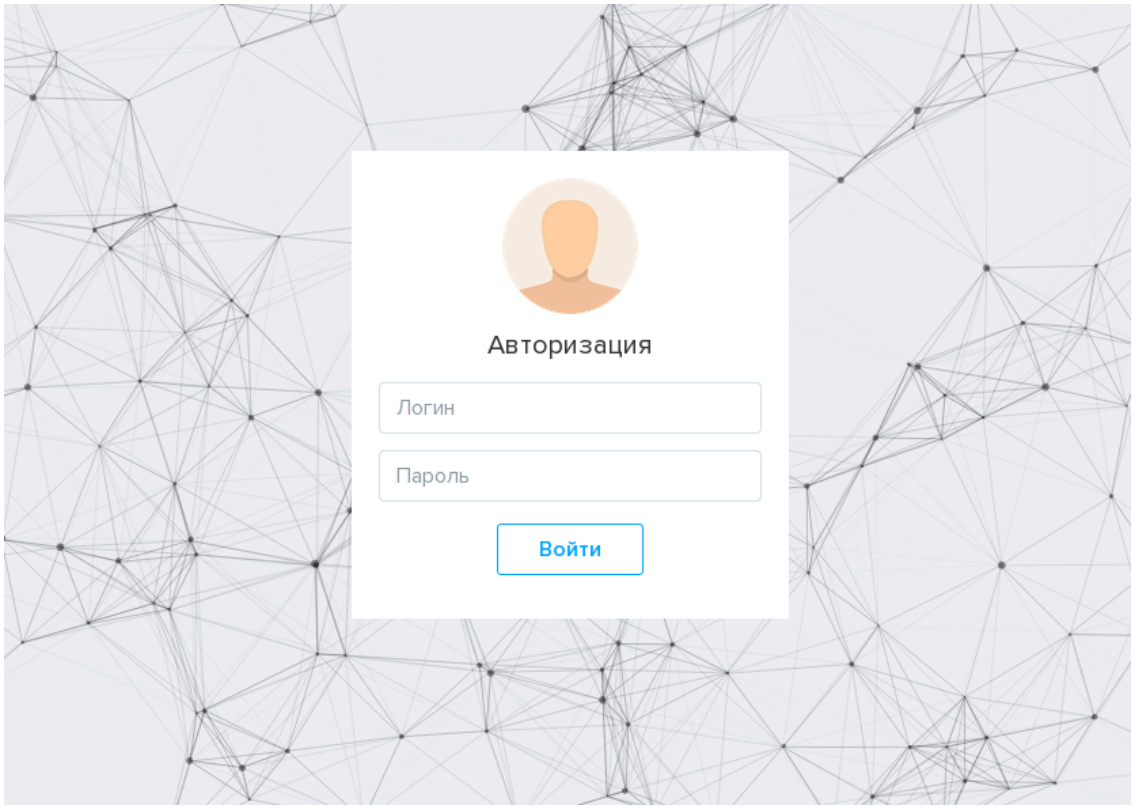
So now it set up some pieces of information into the gate.get request with specifics arguments, from p1 to p7, for example :

- p1: Number of accounts stolen
- p2: Number of cookies stolen
- p4: Number of forms stolen
- etc…

results :



The POST request is now complete, the stealer will clean everything and quit.

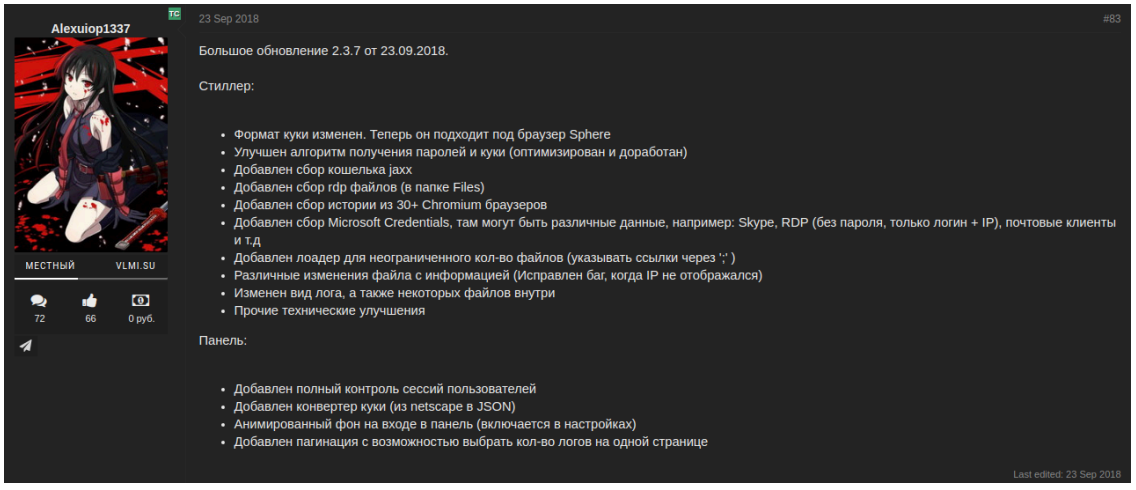


Example of Predator C2 Panel with fancy background…

Update – v2.3.7

So during the analysis, new versions were pushed… Currently (at the time where this post was redacted), the v3 has been released, but without possession of this specific version, I won't talk anything about it and will me be focus only on the 2.3.7.

It's useless to review from scratch, the mechanic of this stealer is still the same, just some tweak or other arrangements was done for multiple purposes… Without digging too much into it, let's see some changes (not all) that I found interesting.



Changelog of v2.3.7 explained by the author

As usual, this is the same patterns :

- Code optimizations (Faster / Lightweight)
- More features…

As you can see v2.3.7 on the right is much longer than v2.3.5 (left), but the backbone is still the same.

Mutex

On 2.3.7, A mutex is integrated with a specific string called “SyystemServs”

Xor / Obfuscated Strings

During the C2 requests, URL arguments are generated byte per byte and unXOR.

For example :

```
push 04
...
push 61
...
push 70
...
```

```
leads to this
HEX    : 046170692F676174652E6765743F70313D
STRING : .api/gate.get?p1=
```

This is basic and simple but enough to just slow down the review of the strings. but at least, it's really easy to uncover it, so it doesn't matter.

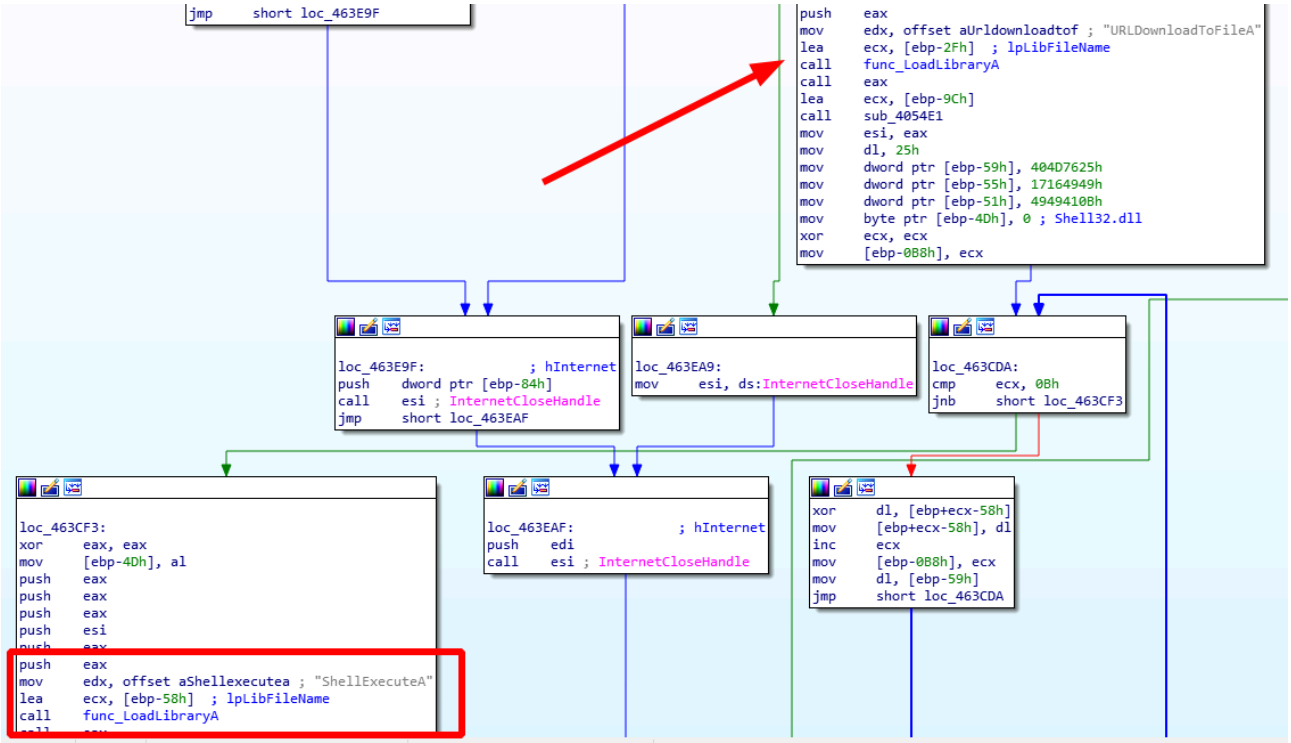
This tweak by far is why the code is much longer than v2.3.5.

Loader

Not seen before (as far I saw), it seems on 2.3.7, it integrates a loader feature to push another payload on the victim machine, easily recognizable with the adequate GET Request

```
/api/download.get
```

The API request permits to the malware to get an URL into text format. Then Download and saved it into disk and execute it with the help of [ShellExecuteA]



There also some other tweaks, but it's unnecessary to detail on this review, I let you this task by yourself if you are curious 🙂

IoC

v2.3.5

- 299f83d5a35f17aa97d40db667a52dcc | Sample Packed
- 3cb386716d7b90b4dca1610afbd5b146 | Sample Unpacked
- kent-adam.myjino.ru | C2 Domain

v2.3.7

- cbcc48fe0fa0fd30cb4c088fae582118 | Sample Unpacked
- denbaliberdin.myjino.ru | C2 Domain

HTTP Patterns

- GET – /api/info.get
- POST – /api//gate.get?p1=X&p2=X&p3=X&p4=X&p5=X&p6=X&p7=X
- GET – /api/download.get

MITRE ATT&CK

v2.3.5

- Discovery – Peripheral Device Discovery
- Discovery – System Information Discovery
- Discovery – System Time Discovery
- Discovery – Query Registry
- Credential Access – Credentials in Files
- Exfiltration – Data Compressed

v2.3.7

- Discovery – Peripheral Device Discovery
- Discovery – System Information Discovery
- Discovery – System Time Discovery
- Discovery – Query Registry
- Credential Access – Credentials in Files
- Exfiltration – Data Compressed
- **Execution – Execution through API**

Author / Threat Actor

- **Alexuiop1337**

Yara Rule

```
rule Predator_The_Thief : Predator_The_Thief {
    meta:
        description = "Yara rule for Predator The Thief v2.3.5 & +"
        author = "Fumik0_"
        date = "2018/10/12"
        update = "2018/12/19"

    strings:
        $mz = { 4D 5A }

        // V2
        $hex1 = { BF 00 00 40 06 }
        $hex2 = { C6 04 31 6B }
        $hex3 = { C6 04 31 63 }
        $hex4 = { C6 04 31 75 }
        $hex5 = { C6 04 31 66 }

        $s1 = "sqlite_" ascii wide

        // V3
        $x1 = { C6 84 24 ?? ?? 00 00 8C }
        $x2 = { C6 84 24 ?? ?? 00 00 1A }
        $x3 = { C6 84 24 ?? ?? 00 00 D4 }
        $x4 = { C6 84 24 ?? ?? 00 00 03 }
        $x5 = { C6 84 24 ?? ?? 00 00 B4 }
        $x6 = { C6 84 24 ?? ?? 00 00 80 }

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

Recommendations

- Always running stuff inside a VM, be sure to install a lot of stuff linked to the hypervisor (like Guest Addons tools) to trigger as much as possible all kind of possible Anti-VM detection and closing malware. When you have done with your activities stop the VM and restore it a Specific clean snapshot when it’s done.
- Avoid storing files at a pre-destined path (Desktop, Documents, Downloads), put at a place that is not common.

- Avoiding Cracks and other stupid fake hacks, stealers are usually behind the current game trendings (especially in those times with Fortnite…).
- Use containers for software that you are using, this will reduce the risk of stealing data.
- Flush your browser after each visit, never saved your passwords directly on your browser or using auto-fill features.
- Don't use the same password for all your websites (use 2FA and it's possible), we are in 2018, and this still sadly everywhere like this.
- Make some noise with your data, that will permit to lose some attacker minds to find some accurate values into the junk information.
- Use a Vault Password software.
- **Troll/Not Troll:** Learn Russian and put your keyboard in Cyrillic 🙂

Conclusion

Stealers are not sophisticated malware, but they are enough effective to make some irreversible damage for victims. Email accounts and other credentials are more and more impactful and this will be worse with the years. Behaviors must changes for the account management to limit this kind of scenario. Awareness and good practices are the keys and this will not be a simple security software solution that will solve everything.

Well for me I've enough work, it's time to sleep a little…

#HappyHunting

Update 2018-10-23 : Yara Rules now working also for v3
