Meet Oski Stealer: An In-depth Analysis of the Popular Credential Stealer

cyberark.com/resources/threat-research-blog/meet-oski-stealer-an-in-depth-analysis-of-the-popular-credential-stealer

January 7, 2021

Meet Oski Stealer: An In-depth Analysis of the Popular Credential Stealer

Credential theft malware continues to be one of the most prevalent types of malware used in cyber attacks. The main objective of nearly all credential theft malware is to gather as much confidential and sensitive information, like user credentials and financial information, as possible.

The Oski stealer is a malicious information stealer, which was first introduced in November 2019. As the name implies, the Oski stealer steals personal and sensitive information from its target. “Oski” is derived from an old Nordic word meaning Viking warrior, which is quite fitting considering this popular info-stealer is extremely effective at pillaging privileged information from its victims.

In this blog, we provide an in-depth analysis of an Oski stealer sample.

Background

As noted above, the Oski stealer is a classic information stealer that is being sold on Russian underground hacking forums at a low price of **$70-$100**.

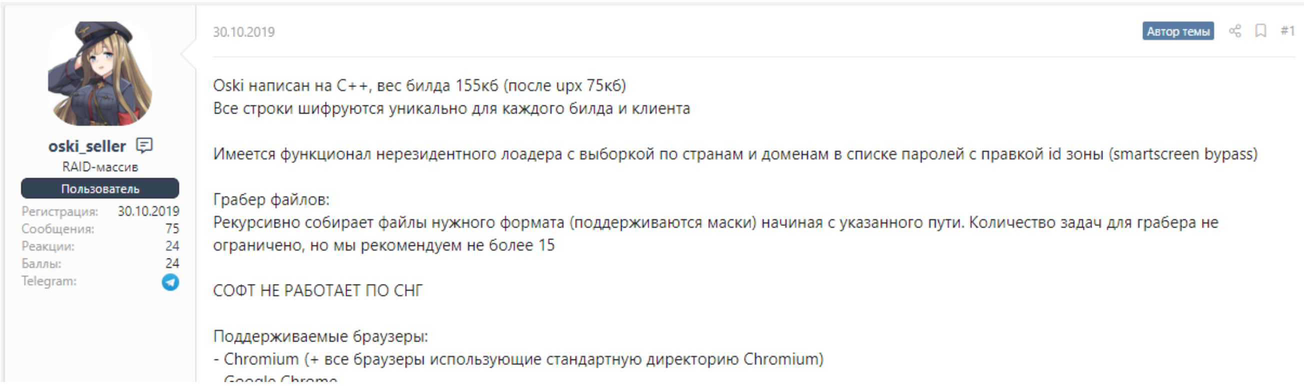

Figure 1: Forum thread for selling Oski Stealer

The stealer is written in C++ and has all the typical features of credential theft malware. Oski targets sensitive information including:

- Login credentials from different applications
- Browser information (cookies, autofill data and credit cards)
- Crypto wallets
- System information
- Screenshots
- Different user files

Beyond these, the stealer can function as a **Downloader** to download a **second-stage malware**.

Every infection involving three parties:

1. Malware authors
2. Malware customers
3. Malware victims

The “customers,” also known as the attackers, contact Oski authors on underground forums to purchase the malware and, once purchased, they configure it and distribute it to their victims.

Oski has a very strong reputation within the underground community, with many of its “customers” regularly providing positive feedback and reviews about the functionality of the malware.

And, even we have to admit that Oski’s functionality works pretty well. From setting up and checking the environment to stealing information by application type, Oski’s code is written with purpose and care. The code is neat and clean, without any presence of useless code lines, however it does lack sophisticated anti-analysis tricks like anti-debugging and dynamic anti-analysis tricks.

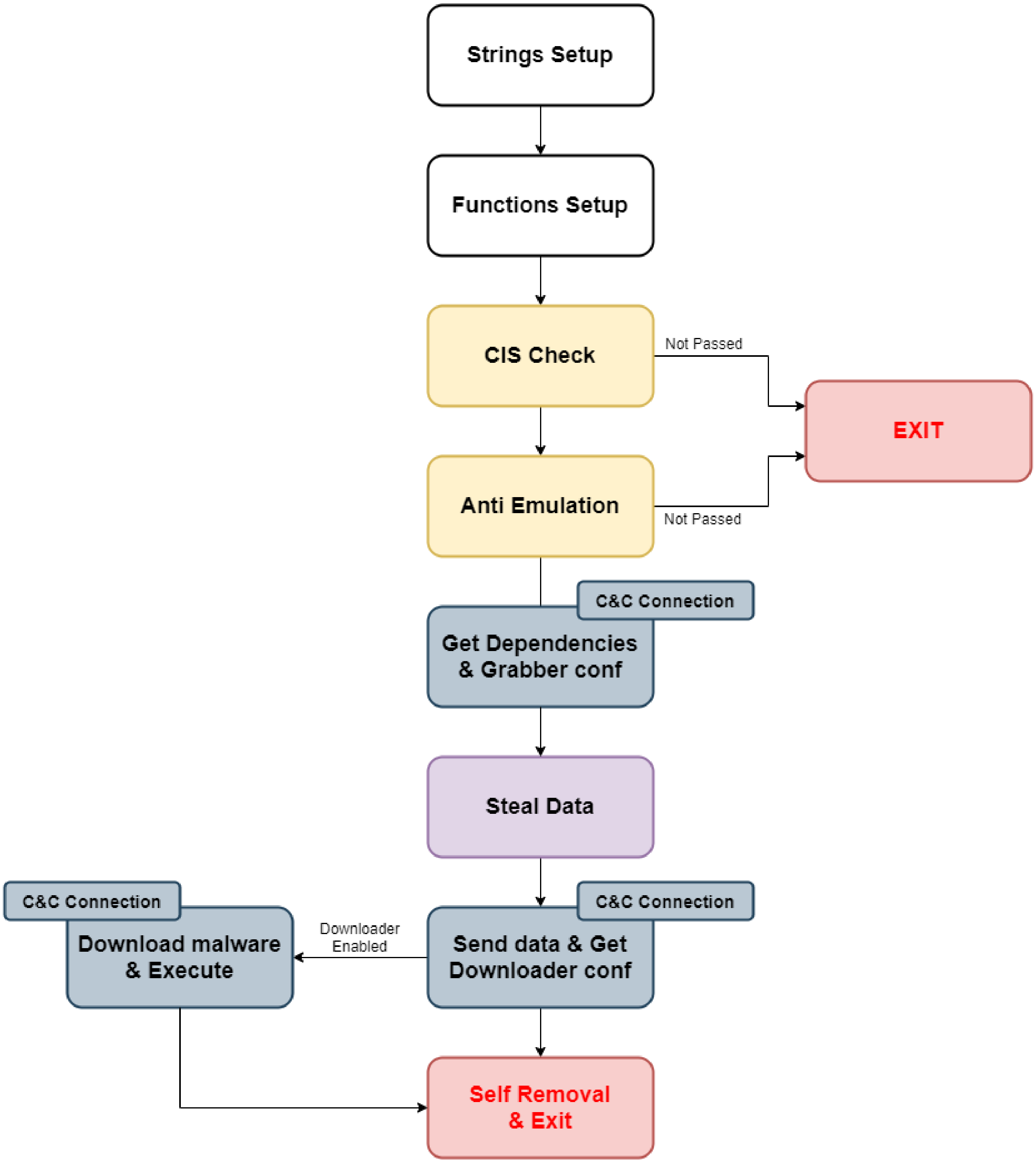

Figure 2: Malware Flow

In-depth Analysis

The sample of Oski stealer analyzed in this blog post is: *aa33731aa48e2ea6d1eaab7c425f9001182c0e73e0226eb01145d6b78d7cb9eb*.

As soon as we opened the Oski stealer sample in IDA, we noticed that it was packed. In our case, the packer used a **self-injection** technique to pack Oski's payload. It then unpacks the payload and writes it to a new memory region – making it easy to notice the new memory region and dump it from memory.

Looking at the TimeDataStamp from the file header of the unpacked PE reveals the compilation time – *0x5EDFAA70* (compiled on 9 Jun 2020). The latest version for Oski stealer v9.1 was released on 19 June 2020, and version v9 was released on 3 Jun 2020, which means that our sample of Oski is **Oski stealer v9**.

Before diving into the stealer's capabilities, it's important to note that the malware uses two obfuscation techniques:

- **Strings encryption**
- **Dynamic loading** of DLLs and functions

To be able to start reverse-engineering the sample statically, we have to decrypt the strings and resolve the loaded functions and DLLs.

Strings Setup

The first function Oski calls from *Main* is stringsSetup – the function responsible for decrypting all the strings for the malware and saving them in memory. The function holds several Base64 strings and a decryption key.

```
stringsSetup proc near

var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+var_4], ecx
mov     key, offset a11015147250010 ; "110151472500104935"
push    offset aGiy5ppzzrvrkzk ; "GIy5pPzzrVrkzKRX4dhTFs7EZFq4QyGfbmVzbu2"...
call    decryptB64
add     esp, 4
mov     dword_431678, eax
push    offset aHa      ; "HA=="
call    decryptB64
add     esp, 4
mov     Mode, eax
push    offset aGjnzinipsemrzq ; "GJnziNipsEmrzQ=="
call    decryptB64
add     esp, 4
mov     dword_4315C0, eax
push    offset aOjnzinipvhdLlo ; "OJnziNipvhD+lLo/g8VJbLfdexmiCDjECiZrJZr"...
call    decryptB64
add     esp, 4
mov     dword_4315BC, eax
push    offset aPinumnkz7qfzno ; "PInumNKz7QfznOQ="
call    decryptB64
add     esp, 4
mov     dword_431350, eax
push    offset aKyn0xp3h7q ; "KYn0xp3h7Q=="
call    decryptB64
add     esp, 4
mov     dword_43146C, eax
push    offset aPppljofku04 ; "PpPljofku04="
call    decryptB64
add     esp, 4
mov     dword_43163C, eax
push    offset aKiTjmiw0Z9ZY9j ; "KI/tjMiw+0/z9/Z/y9JEZOk="
call    decryptB64
add     esp, 4
mov     dword_431520, eax
push    offset aOjnzinipvngy1B ; "OJnziNipvnGy1/Bnz48Be7rVJQ=="
```

Figure 3: stringSetup function

The function decryptB64 (figure 3) gets the decryption key (which in our case is ***110151472500104935***) and the base64 string.

decryptB64 decodes the base64 string and decrypts the decoded information by using **RC4**. Finally, the function returns the decrypted string to the string's setup function, which saved the decrypted string within memory (Figure 3).

TIP: RC4 is a pretty common cipher that's used by malware developers. When trying to figure out which decryption/encryption routine is used in malware, the standard process we tend to follow is to first start by finding any constant ("magic") values to help reveal the decryption/encryption routine. For RC4, there are no constant values – in fact, it’s the most popular algorithm that doesn’t use constant values.

Function Setup

The second function Oski calls for after setting up all the strings in memory is *procsSetup*, which is responsible for loading different DLLs, resolving function addresses and saving the addresses within memory.

The names of the functions and DLLs are encrypted, therefore we must first decrypt the strings and then we will be able to determine which functions and DLLs are loaded.

Oski gets the address for the functions LoadLibraryA and GetProcAddress from memory. This part of the code is written as a Position-Independent code (PIC).

There are two operations Oski performs in order to get the functions from memory:

- Find the base address of *dll* from the *PEB* structure of the process
- Resolve the address of the functions from the export table of *kernel32*.dll by parsing the PE within memory

The next part describes these methods and how Oski stealer implemented them.

If you are already familiar with these techniques, you can skip ahead to Back to Functions Setup>.

Find kernel32.dll

In x86 programs, the FS segment register holds the Thread Information Block (_TEB struct) for the current thread.

The _TEB structure holds a pointer within *offset **0x30*** to the Process Environment Block (_PEB), which contains information about the running process in the form of several data structures and many different fields.

One of those structures is a pointer to _PEB_LDR_DATA within *offset **0x0c*** from the start of the PEB.

The _PEB_LDR_DATA struct provides information about the DLLs that are loaded into the process.

```
typedef struct _PEB_LDR_DATA
{
    ULONG         Length;                             /* +0x00 */
    BOOLEAN       Initialized;                        /* +0x04 */
    PVOID         SsHandle;                           /* +0x08 */
    LIST_ENTRY    InLoadOrderModuleList;              /* +0x0c */
    LIST_ENTRY    InMemoryOrderModuleList;            /* +0x14 */
    LIST_ENTRY    InInitializationOrderModuleList;/* +0x1c */
} PEB_LDR_DATA,*PPEB_LDR_DATA; /* +0x24 */
```

Figure 4: _PEB_LDR_DATA structure

The _PEB_LDR_DATA holds 3 pointers to 3 doubly linked lists – InLoadOrderModuleList, InMemoryOrderModuleList and InInitializationOrderModuleList. All provide information about the loaded DLLs in the process, however the second and the third lists are good for finding the desired DLL.

The list InMemoryOrderModuleList holds the DLLs loaded by the process sorted by their **order in memory,** and the list InInitializationOrderModuleList holds the DLLs by their **order of initialization**.
The entry within all three lists is LDR module (_LDR_DATA_TABLE_ENTRY) for the current DLL in the list.

```
getKernel32BasePIC proc near            ; CODE XREF: 00089706↑p
                                        ; 006E9706↑p ...

pKernel32Base= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+pKernel32Base], 0
mov     eax, large fs:30h               ; pPEB
mov     eax, [eax+0Ch]                  ; pLdr
mov     eax, [eax+0Ch]                  ; pInLoadOrderModuleList
mov     eax, [eax]                      ; pInLoadOrderLinks->Flink|ntdll.dll
mov     eax, [eax]                      ; pInLoadOrderLinks->Flink|kernel32.dll
mov     eax, [eax+18h]                  ; pDllBase
mov     [ebp+pKernel32Base], eax
mov     eax, [ebp+pKernel32Base]
mov     esp, ebp
pop     ebp
retn
getKernel32BasePIC endp
```

Figure 5: _LDR_DATA_TABLE_ENTRY structure

The _LDR_DATA_TABLE_ENTRY contains information about the loaded DLL. From offset ***0x18*** from the address of _LDR_DATA_TABLE_ENTRY, we can obtain the DllBase*,* which is a pointer to the **DLL base address** in memory.

After explaining the theory for getting the modules base address independently, we will check how Oski implements this technique.

```
typedef struct _LDR_DATA_TABLE_ENTRY
{
    LIST_ENTRY InLoadOrderLinks;           /* +0x00 */
    LIST_ENTRY InMemoryOrderLinks;         /* +0x08 */
    LIST_ENTRY InInitializationOrderLinks; /* +0x10 */
    PVOID DllBase;                         /* +0x18 */
    PVOID EntryPoint;                      /* +0x1c */
    ULONG SizeOfImage;                     /* +0x20 */
    UNICODE_STRING FullDllName;            /* +0x24 */
    UNICODE_STRING BaseDllName;            /* +0x28 */
    ...
    ...
```

Figure 6: Oski function for getting kernel32.dll base address

Oski gets the base address of ***kernel32.dll*** from memory, which is the third entry within the LIST_ENTRY in InLoadOrderModuleList (The first entry is a pointer for the executable and the second is for *ntdll.dll*).

Oski's next steps are to get the address of LoadLibraryA and GetProcAddress; both functions are exported by *kernel32.dll*.

Find Exported Functions

Once Oski gets the base address of *kernel32.dll*, it parses the PE file and loops over the exported functions of the DLL to get the address of the desired functions.

To do so, it needs to traverse serval headers of the DLL.

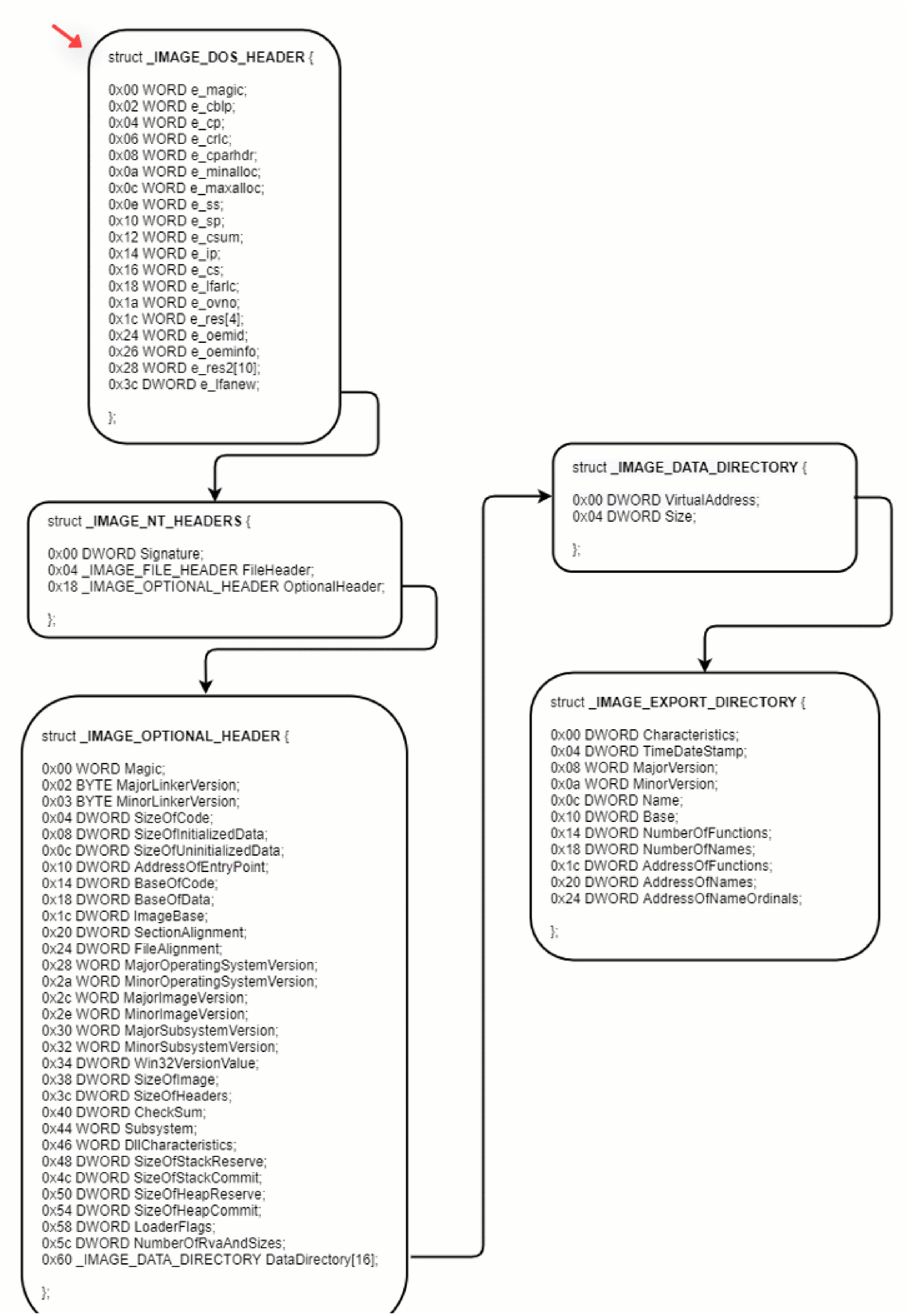
struct _IMAGE_DOS_HEADER {
0x00 WORD e_magic;
0x02 WORD e_cblp;
0x04 WORD e_cp;
0x06 WORD e_crlc;
0x08 WORD e_cparhdr;
0x0a WORD e_minalloc;
0x0c WORD e_maxalloc;
0x0e WORD e_ss;
0x10 WORD e_sp;
0x12 WORD e_csum;
0x14 WORD e_ip;
0x16 WORD e_cs;
0x18 WORD e_lfarlc;
0x1a WORD e_ovno;
0x1c WORD e_res[4];
0x24 WORD e_oemid;
0x26 WORD e_oeminfo;
0x28 WORD e_res2[10];
0x3c DWORD e_lfanew;
};
struct _IMAGE_NT_HEADERS {
0x00 DWORD Signature;
0x04 _IMAGE_FILE_HEADER FileHeader;
0x18 _IMAGE_OPTIONAL_HEADER OptionalHeader;
};
struct _IMAGE_OPTIONAL_HEADER {
0x00 WORD Magic;
0x02 BYTE MajorLinkerVersion;
0x03 BYTE MinorLinkerVersion;
0x04 DWORD SizeOfCode;
0x08 DWORD SizeOfInitializedData;
0x0c DWORD SizeOfUninitializedData;
0x10 DWORD AddressOfEntryPoint;
0x14 DWORD BaseOfCode;
0x18 DWORD BaseOfData;
0x1c DWORD ImageBase;
0x20 DWORD SectionAlignment;
0x24 DWORD FileAlignment;
0x28 WORD MajorOperatingSystemVersion;
0x2a WORD MinorOperatingSystemVersion;
0x2c WORD MajorImageVersion;
0x2e WORD MinorImageVersion;
0x30 WORD MajorSubsystemVersion;
0x32 WORD MinorSubsystemVersion;
0x34 DWORD Win32VersionValue;
0x38 DWORD SizeOfImage;
0x3c DWORD SizeOfHeaders;
0x40 DWORD CheckSum;
0x44 WORD Subsystem;
0x46 WORD DllCharacteristics;
0x48 DWORD SizeOfStackReserve;
0x4c DWORD SizeOfStackCommit;
0x50 DWORD SizeOfHeapReserve;
0x54 DWORD SizeOfHeapCommit;
0x58 DWORD LoaderFlags;
0x5c DWORD NumberOfRvaAndSizes;
0x60 _IMAGE_DATA_DIRECTORY DataDirectory[16];
};
struct _IMAGE_DATA_DIRECTORY {
0x00 DWORD VirtualAddress;
0x04 DWORD Size;
};
struct _IMAGE_EXPORT_DIRECTORY {
0x00 DWORD Characteristics;
0x04 DWORD TimeDateStamp;
0x08 WORD MajorVersion;
0x0a WORD MinorVersion;
0x0c DWORD Name;
0x10 DWORD Base;
0x14 DWORD NumberOfFunctions;
0x18 DWORD NumberOfNames;
0x1c DWORD AddressOfFunctions;
0x20 DWORD AddressOfNames;
0x24 DWORD AddressOfNameOrdinals;
};

Figure 7: how to get the EXPORT_DIRECTORY

After getting the *Export Table,* Oski must find the desired function by looking for the function name. The process is as follows:

- The AddressOfNames is a pointer to an array of the exported functions **names**, so Oski loops through the array and compares each function name to the desired function, while counting the position of the string in the array.
- Oski gets the ordinal number for the function from the *Ordinal Table.* Each entry in the table is 2 bytes, therefore, it must multiply the position of the function name by 2.
- Finally, Oski calculates the address for the function from the *Address Table.* Each entry in the table is 4 bytes, therefore, it must multiply the ordinal number by 4.

Back to Functions Setup

Oski uses a function that implements this technique for getting the function's address from memory. The function GetProcAddrPIC (figure 8) gets a pointer to the DLL base address and a name for an exported function.

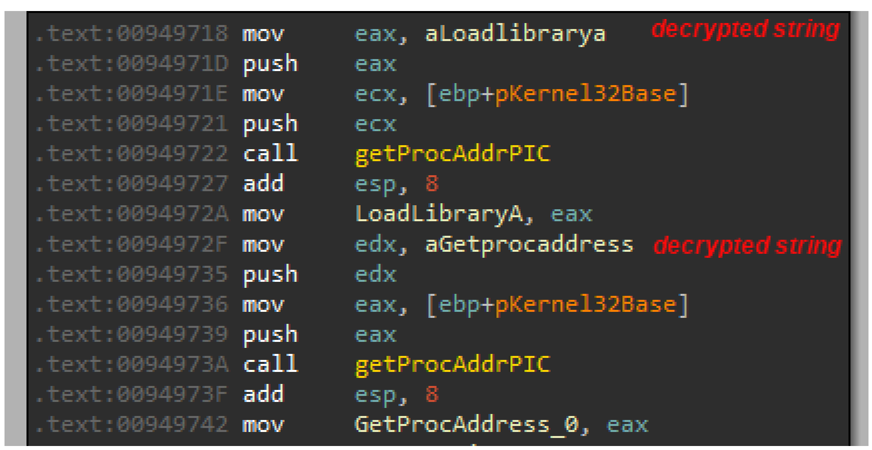

Figure 8: Oski get the address for LoadLibraryA and GetProcAddress

Finally, after getting the address of those APIs, Oski can start loading DLLs and resolving function addresses. As we mentioned earlier, all the strings are encrypted, so we have to decrypt them first to be able to understand statically which functions and DLLs Oski uses.

GetProcAddress and LoadLibraryA are being called many times in order to load different DLLs and resolve functions.

To make our analysis easier, we made an IDA Python script that automates Oski setup stages and deobfuscates the code.

Oski Deobfuscator: An IDA Python Script

The script automates all the analysis of the setup stages for Oski stealer (v9+) and defeats its obfuscation to make the static analysis easier and more convenient.

Strings Setup

- Find the decryption key
- Decrypt all the strings (B64, RC4)
- Give meaningful names (IDA)
- Add comments with the full decrypted string

Functions Setup

- Find LoadLibraryA and GetProcAddress
- Resolve the loaded DLLs and functions
- Give meaningful names to functions and DLLs (IDA)

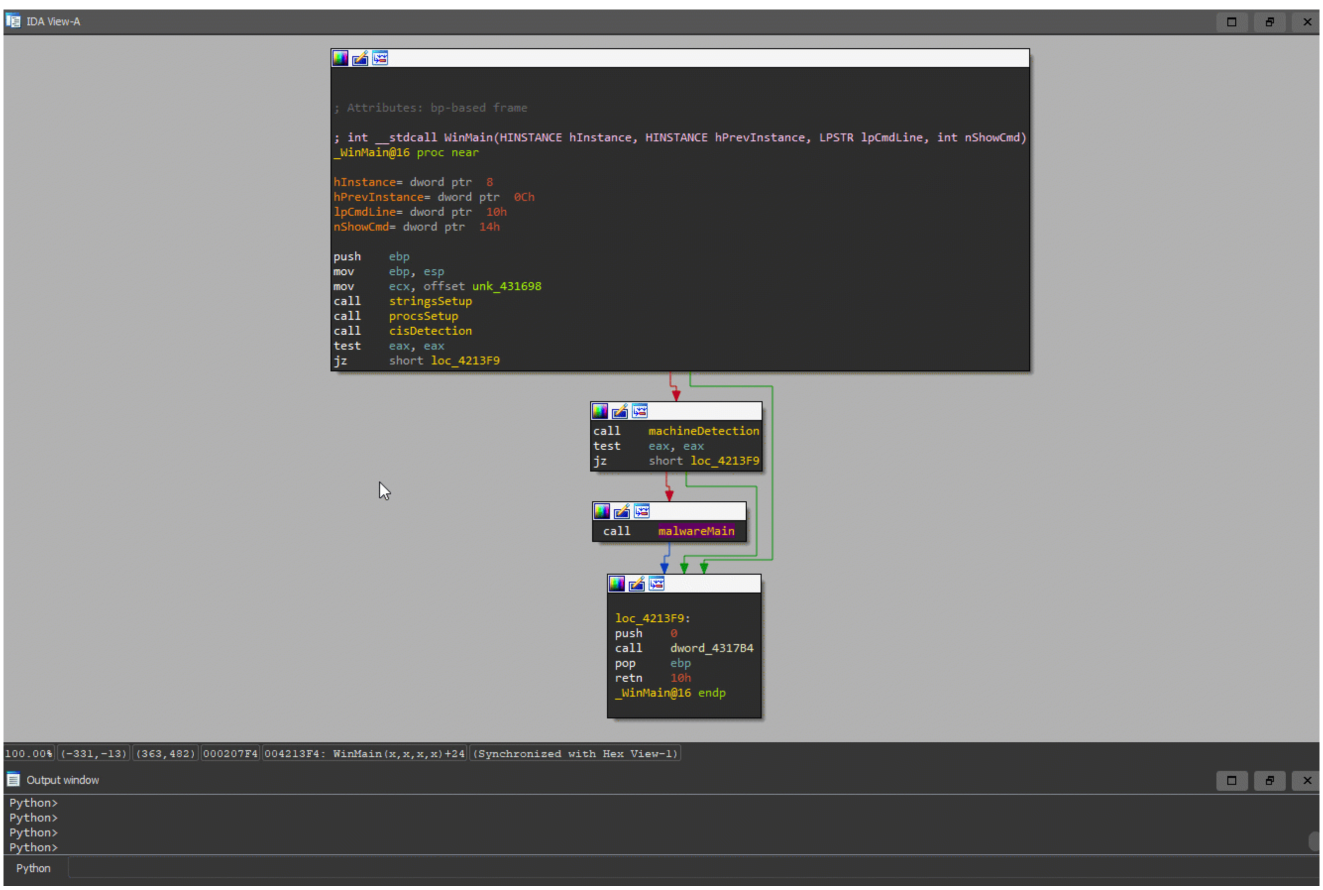

Figure 9: Before and after using oski_ida.py

The script decrypted **380 strings**, resolved **107 functions,** and **11 DLLs**.
In addition, the script dumps the addresses and the full decrypted strings to a JSON file.

You can find the script ***oski_ida.py*** on our repo

Finally, after setting up the names for the strings and functions, we can move to analyzing the sample statically.

Environment Checks

CIS Check

Oski checks the user language to determine if it's part of the **Commonwealth of Independent States** (CIS) countries. This behavior is popular, especially within crimeware tools that are sold on Russian underground forums.

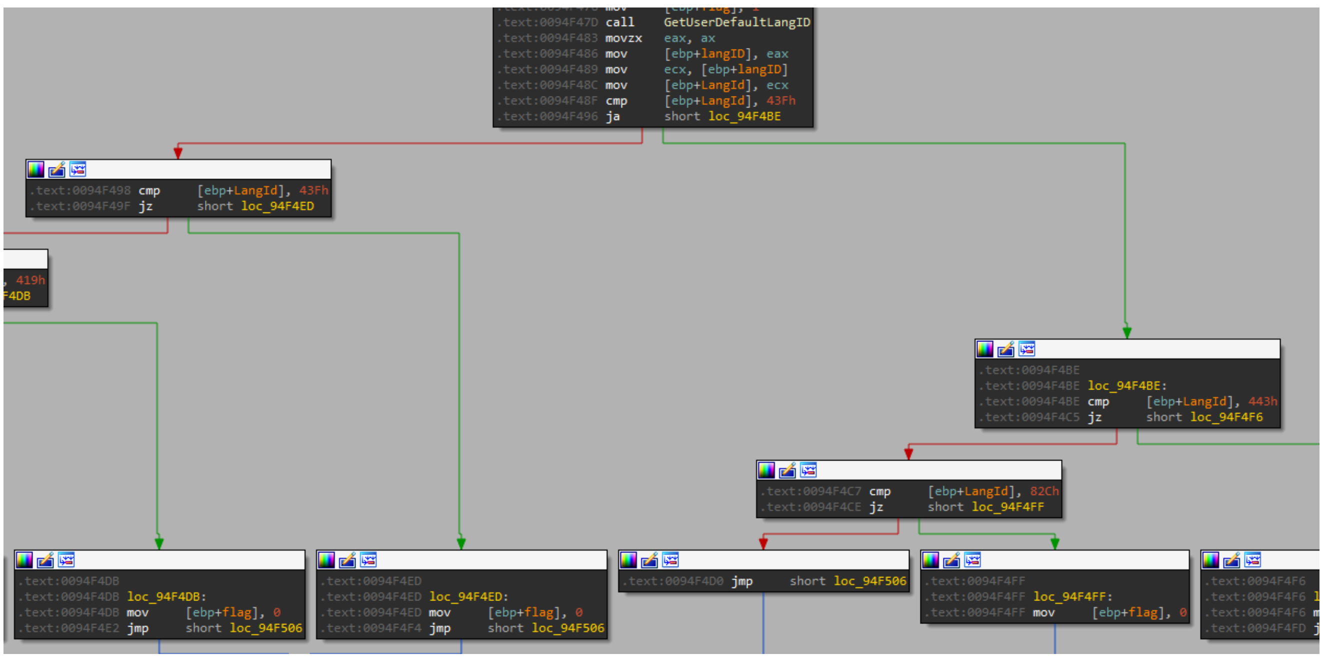

Figure 10: cisDetection function

Oski gets the user language ID by using GetUserDefaultLangID and it compares the user language ID to:

0x423

Language ID	Language-tag	Location
0x43F	kk-KZ	Kazakhstan
0x443	Us-Latb-US	Uzbekistan
0x82C	Az-Cyrl-AZ	Azerbaijan
0x419	Ru-RU	Russia

0x422	uk-UA	Ukraine
Be-BY	Belarus	

If the user language ID matches one of the IDs above, the stealer will exit.

Anti-Emulation Check
The second check is an anti-emulation check for Windows Defender Antivirus. The malware calls to GetComputerNameA and compares the computer name to ***HAL9TH***. In addition, it checks if the username is ***JohnDoe*** by calling to GetUserNameA. Those two parameters are being used by the Windows Defender emulator.

The Stealer's Main Functionality

Oski steals confidential and sensitive data from ~60 different applications, including browsers, email clients, and crypto wallets. Among its stealing features, it can also function as a **Grabber** and **Loader**.

Before stealing credentials from different applications, Oski sets up its "working environment." However, in order to steal data by different methods from different applications, Oski has to download serval DLLs.

Oski downloads 7 DLLs from the C&C server and saves them in the *ProgramData* folder.

- sqlite3.dll
- freebl3.dll
- mozglue.dll
- msvcp140.dll
- nss3.dll
- softokn3.dll
- vcruntime140.dll

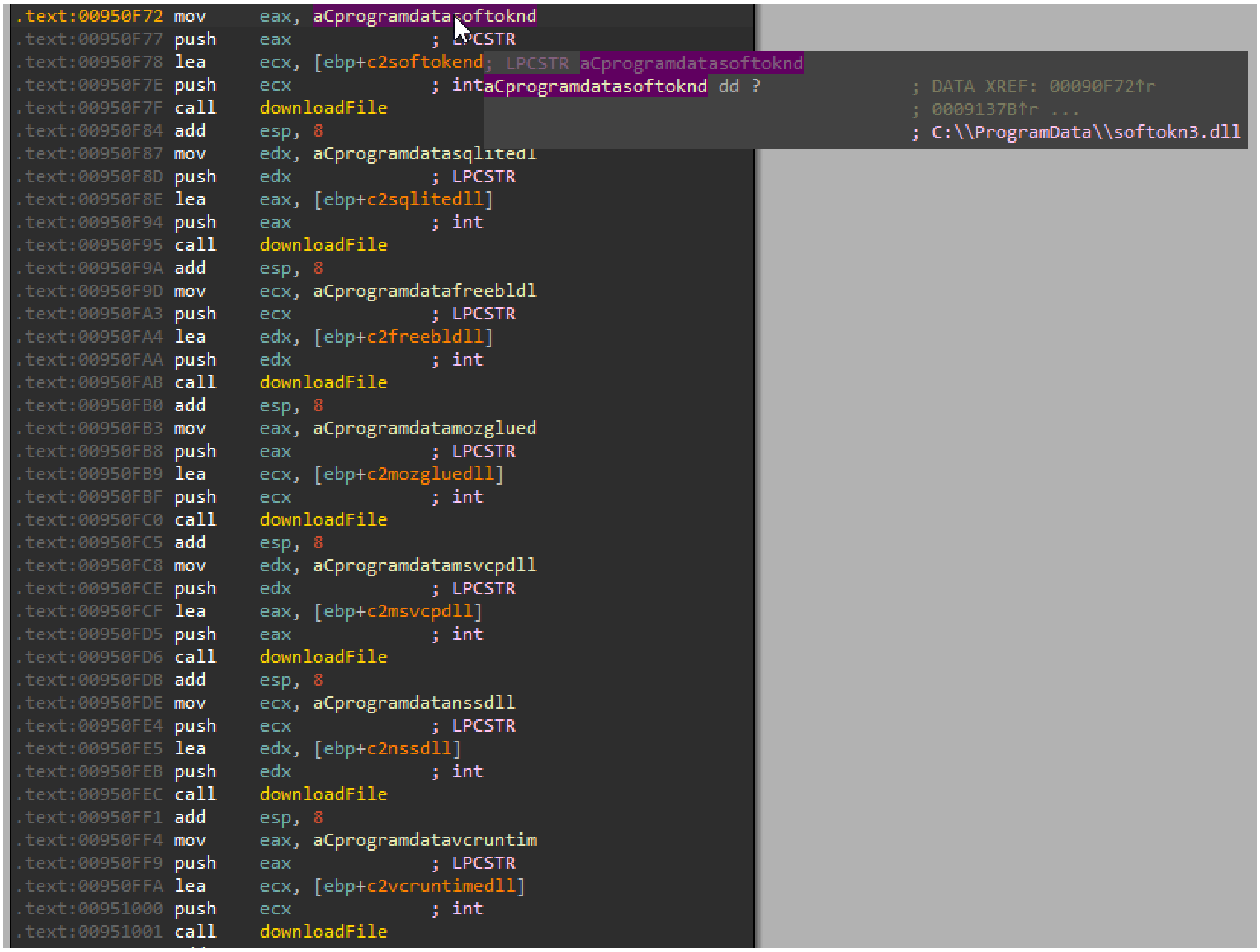

Figure 11: Oski downloads dependencies (7 DLLs)

Each DLL has its own URL address. In the Oski version we sampled, the URL for the DLL is the DLL's name – *evil.cc/sqlite3.dll*.

In some other versions, Oski makes the requests to *evil.cc/1.jpeg*, *evil.cc/2.jpeg* and so on, to download the DLLs.

(1.jpeg = sqlite3.dll, 2.jpeg = freebl3.dll, 3.jpeg = mozglue.dll, 4.jpeg = msvcp140.dll, 5.jpeg = nss3.dll, 6.jpeg = softokn3.dll, 7.jpeg = vcruntime140.dll)

Because Oski makes those **seven requests** to the C&C server to download its dependencies, it is not very stealthy.

Oski creates its working folder which is named with a 15 digits randomly generated string within *ProgramData* like *C:\ProgramData\234378117851778*, for example. This folder will contain all the stolen logs and data. In addition, it creates four folders inside the working folder:

- autofill – autofill data from browsers
- cc – credit card data
- cookies – browsers cookies
- crypto – cryptocurrency wallets

Browsers and Email Clients

Oski steals **login credentials**, **cookies**, **credit card** and **autofill** information from 30+ different browsers using well-known and familiar stealing methods.

It has four different methods to steal data from different types of browses, like **Mozilla** based applications, **Opera**, **Internet Explorer** and **Chromium**-based browsers.

It's worth mentioning that Oski updated its stealing technique regarding Chromium-based browsers and now supports the new method (v80+) by Chromium for encrypting credentials and cookies with a global AES key that is stored within *%localappdata%\Google\Chrome\User Data\Local State* and encrypted by using DPAPI. Prior to version 80 of Chromium, the credentials and cookies were simply encrypted by DPAPI instead that AES key.

Furthermore, Oski collects information about the connected **Outlook** accounts from the registry like passwords and confidential data about the IMAP and SMTP servers and it dumps all the data to file named *outlook.txt*.

```
.text:0094F330 mw_outlookSteal proc near
.text:0094F330 push    ebp
.text:0094F331 mov     ebp, esp
.text:0094F333 push    offset aSoftwareMicros ; "Software\\Microsoft\\Windows NT\\Curren"...
.text:0094F338 call    mw_mainOutlook
.text:0094F33D add     esp, 4
.text:0094F340 push    offset aSoftwareMicros_0 ; "Software\\Microsoft\\Windows NT\\Curren"...
.text:0094F345 call    mw_mainOutlook
.text:0094F34A add     esp, 4
.text:0094F34D push    offset aSoftwareMicros_1 ; "Software\\Microsoft\\Windows NT\\Curren"...
.text:0094F352 call    mw_mainOutlook
.text:0094F357 add     esp, 4
.text:0094F35A push    offset aSoftwareMicros_2 ; "Software\\Microsoft\\Windows NT\\Curren"...
.text:0094F35F call    mw_mainOutlook
.text:0094F364 add     esp, 4
.text:0094F367 push    offset aSoftwareMicros_3 ; "Software\\Microsoft\\Office\\13.0\\Outl"...
.text:0094F36C call    mw_mainOutlook
.text:0094F371 add     esp, 4
.text:0094F374 push    offset aSoftwareMicros_4 ; "Software\\Microsoft\\Office\\13.0\\Outl"...
.text:0094F379 call    mw_mainOutlook
.text:0094F37E add     esp, 4
.text:0094F381 push    offset aSoftwareMicros_5 ; "Software\\Microsoft\\Office\\13.0\\Outl"...
.text:0094F386 call    mw_mainOutlook
.text:0094F38B add     esp, 4
.text:0094F38E push    offset aSoftwareMicros_6 ; "Software\\Microsoft\\Office\\13.0\\Outl"...
.text:0094F393 call    mw_mainOutlook
.text:0094F398 add     esp, 4
.text:0094F39B push    offset aSoftwareMicros_7 ; "Software\\Microsoft\\Office\\14.0\\Outl"...
.text:0094F3A0 call    mw_mainOutlook
.text:0094F3A5 add     esp, 4
.text:0094F3A8 push    offset aSoftwareMicros_8 ; "Software\\Microsoft\\Office\\14.0\\Outl"...
.text:0094F3AD call    mw_mainOutlook
.text:0094F3B2 add     esp, 4
.text:0094F3B5 push    offset aSoftwareMicros_9 ; "Software\\Microsoft\\Office\\14.0\\Outl"...
.text:0094F3BA call    mw_mainOutlook
.text:0094F3BF add     esp, 4
.text:0094F3C2 push    offset aSoftwareMicros_10 ; "Software\\Microsoft\\Office\\14.0\\Outl"...
.text:0094F3C7 call    mw_mainOutlook
.text:0094F3CC add     esp, 4
.text:0094F3CF push    offset aSoftwareMicros_11 ; "Software\\Microsoft\\Office\\15.0\\Outl"...
.text:0094F3D4 call    mw_mainOutlook
.text:0094F3D9 add     esp, 4
.text:0094F3DC push    offset aSoftwareMicros_12 ; "Software\\Microsoft\\Office\\15.0\\Outl"...
.text:0094F3E1 call    mw_mainOutlook
.text:0094F3E6 add     esp, 4
.text:0094F3E9 push    offset aSoftwareMicros_13 ; "Software\\Microsoft\\Office\\15.0\\Outl"...
.text:0094F3EE call    mw_mainOutlook
.text:0094F3F3 add     esp, 4
```

Figure 12: stealing data from Outlook registry profiles

We won't cover Oski's stealing techniques as they aren't terribly innovative and have been reviewed many times, but you can find an explanation about most of these techniques in this whitepaper on the Raccoon stealer.

Cryptocurrency Wallets

Oski also steals wallets and confidential files that are related to crypto wallet applications. It targets 28 crypto wallet applications, which store sensitive data in files. An example is the most known file- ***wallet.dat*** *which* contains the confidential data about the wallet including **private keys**, public keys, etc.

The stealer checks for the default wallet file location in *AppData* and copies it to the working folder.

```
.text:00954D00 mw_cryptoSteal proc near
.text:00954D00
.text:00954D00 workingFolder= dword ptr  8
.text:00954D00
.text:00954D00 push    ebp
.text:00954D01 mov     ebp, esp
.text:00954D03 push    104h             ; Size
.text:00954D08 push    0                ; Val
.text:00954D0A push    offset workingFolder ; void *
.text:00954D0F call    _memset
.text:00954D14 add     esp, 0Ch
.text:00954D17 mov     eax, [ebp+workingFolder]
.text:00954D1A push    eax              ; lpString2
.text:00954D1B push    offset workingFolder ; lpString1
.text:00954D20 call    lstrcatA
.text:00954D26 mov     ecx, aWaldat
.text:00954D2C push    ecx              ; senstaiveFile
.text:00954D2D mov     edx, aBitcoin
.text:00954D33 push    edx              ; cryptoAppFolder
.text:00954D34 mov     eax, aBitcoin
.text:00954D39 push    eax              ; pAppFolder
.text:00954D3A call    mw_find_copyFileAppData
.text:00954D3F add     esp, 0Ch
.text:00954D42 mov     ecx, aKeystore
.text:00954D48 push    ecx              ; senstaiveFile
.text:00954D49 mov     edx, aEthereum
.text:00954D4F push    edx              ; cryptoAppFolder
.text:00954D50 mov     eax, aEthereum
.text:00954D55 push    eax              ; pAppFolder
.text:00954D56 call    mw_find_copyFileAppData
.text:00954D5B add     esp, 0Ch
.text:00954D5E mov     ecx, aDefaultwallet
.text:00954D64 push    ecx              ; senstaiveFile
.text:00954D65 mov     edx, aElectrumwallets
.text:00954D6B push    edx              ; cryptoAppFolder
.text:00954D6C mov     eax, aElectrum
.text:00954D71 push    eax              ; pAppFolder
.text:00954D72 call    mw_find_copyFileAppData
.text:00954D77 add     esp, 0Ch
.text:00954D7A mov     ecx, aDefaultwallet
.text:00954D80 push    ecx              ; senstaiveFile
.text:00954D81 mov     edx, aElectrumltcwallets
.text:00954D87 push    edx              ; cryptoAppFolder
.text:00954D88 mov     eax, aElectrumltc
.text:00954D8D push    eax              ; pAppFolder
.text:00954D8E call    mw_find_copyFileAppData
.text:00954D93 add     esp, 0Ch
```

Figure 13: Oski stealing from crypto wallets apps

The configuration for this module:

App Name	App Folder	Regex (sensitive file)
Anoncoin	\Anoncoin\	*wal*.dat
BBQCoin	\BBQCoin\	*wal*.dat
Bitcoin	\Bitcoin\	*wal*.dat
DashCore	\DashCore\	*wal*.dat
devcoin	\devcoin\	*wal*.dat
digitalcoin	\digitalcoin\	*wal*.dat
ElectronCash	\ElectronCash\wallets\	default_wallet
Electrum	\Electrum\wallets\	default_wallet
Electrum-LTC	\Electrum- LTC\wallets\	default_wallet
Ethereum	\Ethereum\	keystore
Exodus	\Exodus\	exodus.conf.json window- state.json
Exodus	\Exodus\exodus.wallet\	passphrase.json seed.seco info.seco
Florincoin	\Florincoin\	*wal*.dat
Franko	\Franko\	*wal*.dat
Freicoin	\Freicoin\	*wal*.dat
GoldCoinGLD	\GoldCoin (GLD)\	*wal*.dat
Infinitecoin	\Infinitecoin\	*wal*.dat
IOCoin	\IOCoin\	*wal*.dat
Ixcoin	\Ixcoin\	*wal*.dat
jaxx	\com.liberty.jaxx\IndexedDB\file__0.indexeddb.leveldb\	*
Litecoin	\Litecoin\	*wal*.dat

Megacoin	\Megacoin\	*wal*.dat
Mincoin	\Mincoin\	*wal*.dat
MultiDoge	\MultiDoge\	*wal*.dat
Namecoin	\Namecoin\	*wal*.dat
Primecoin	\Primecoin\	*wal*.dat
Terracoin	\Terracoin\	*wal*.dat
YACoin	\YACoin\	*wal*.dat
Zcash	\Zcash\	*wal*.dat

Collect System Information

Similar to other classic stealers, Oski gathers information about the system and takes a screenshot of the user's desktop. It then writes the information to *system.txt* and saves the screenshot to *screenshot.jpg.*

- **System**
  Windows version, computer architecture, username, computer name, system language, Machine ID, GUID, domain name and Workgroup name.
- **Hardware**
  Processor type, number of processors, video card type, display resolution, RAM size, and checks if the computer is a laptop or desktop. Oski checks if the computer is a laptop by calling to GetSystemPowerStatus – the function retrieves information about the power status of the system. The returned struct contains a one-byte flag named batteryFlag, which can indicate if the system has a battery or not.

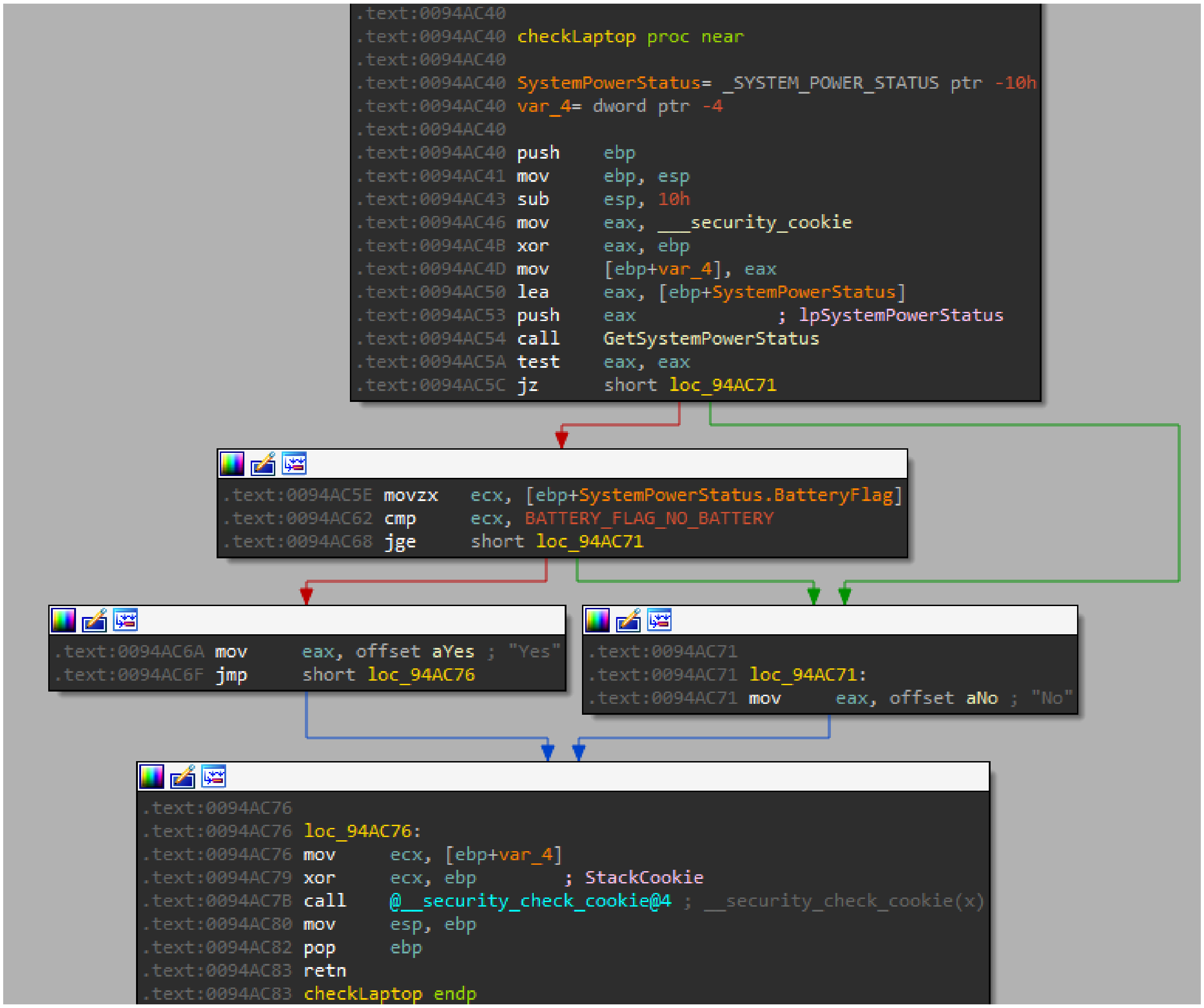

Figure 14: checkLaptop function

- **Local time**
- **Network**
  Oski has hardcoded values for this section, so the log will always contain unknown values – *IP: IP?* and *Country: Country?*

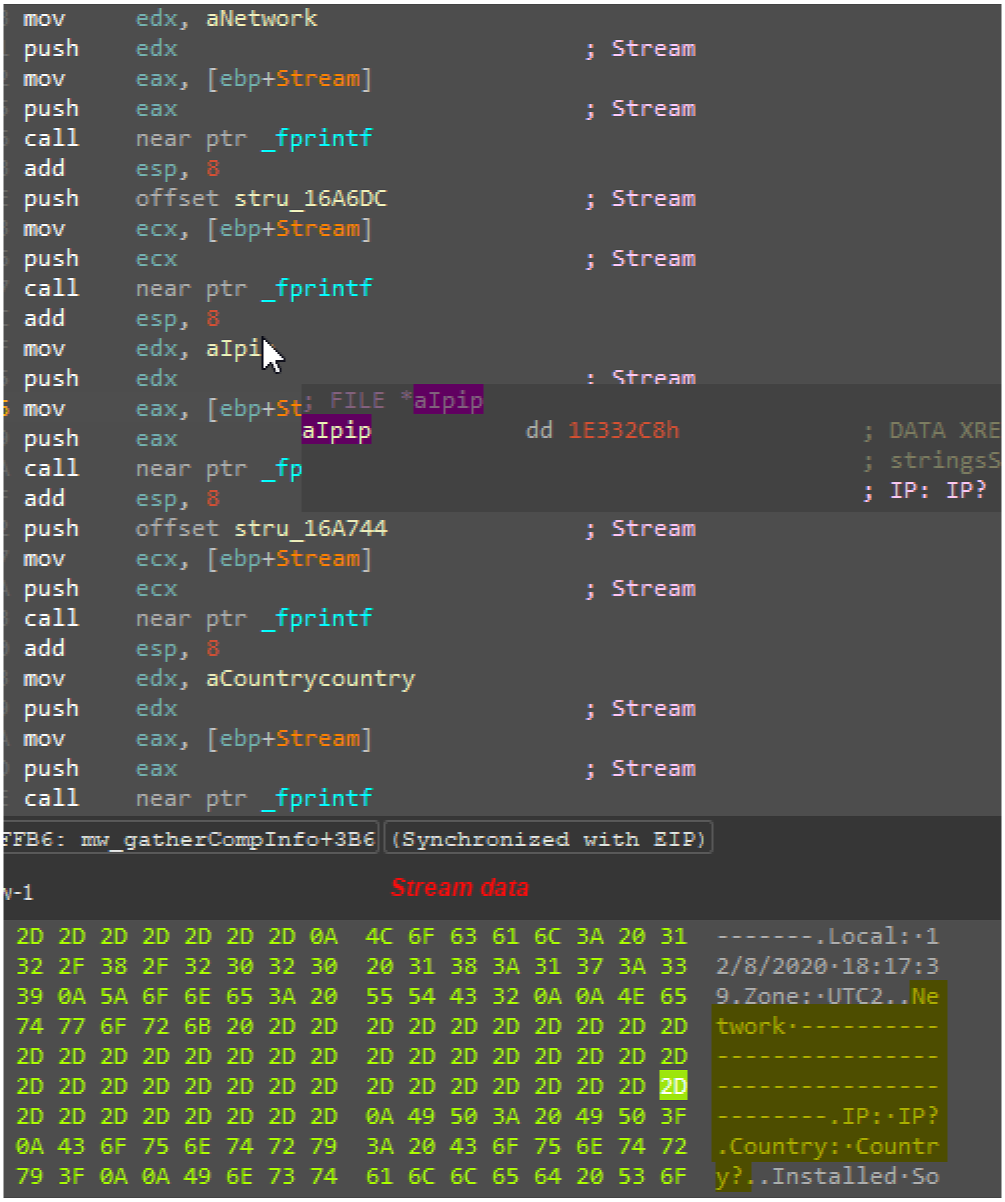

Figure 15: Oski writes the useless values

Installed Software

Get the installed applications on the machine and its version. Oski has a typo in this section, the title is *Installed* ***Softwrare***, instead of "Software," so this typo is unique for Oski logs.

```
Network -------------------------------------------------
IP: IP?
Country: Country?

Installed Software --------------------------------------
Foxmail 7.2.14.410
Google Chrome 83.0.4103.116
Mozilla Thunderbird 68.3.1 (x86 en-US) 68.3.1
Visual C++ Compiler/Tools X86 ARM Cross Package 14.0.24210
Visual C++ MSBuild ARM Package 14.0.25420
```

Figure 16: Oski system log

Screenshot

Grabber Module

Oski also has a recursive grabber that collects particular files from the victim’s computer. The module is configurable, allowing the attacker to decide whether to enable this module and if so, which files to collect from the user.

Oski creates a POST request to *main.php* in the C&C. In our case, the URL is http://sl9XA73g7u3EO07WT42n7f4vIn5fZH[.]biz/main.php. The response from the C&C contains the configuration for the grabber.

The first part of the Grabber function is parsing the response data. The parsing function uses strtok function while passing the delimiter “*;*” and the response data from the C&C.

It extracts the first three tokens from the configuration and passes them to the “main function” of the grabber. After the first three tokens, the parsing function takes the next three tokens, and so on.

In this way, we can figure out that the structure of the configuration has three parts (parameters) and that the configuration can hold several tasks.

Let’s focus on *mainGrabber* function. This function gets three arguments, which are the three tokens from the configuration, each call to *mainGrabber* is called “task.”

The task structure has three fields (parameters):

1. A name for the zip file – will contain all the stolen files that related to the current task. Oski concatenates to this name an underscore at the beginning, so the name for the zip will be *_%name%.zip*.

2. An environment variable name and folder name – a starting point for the *recursiveGrabber*.
3. A regex list – contains multiply parameters that are separated by "," each one of them is a regex that represents a file type.

The *recursiveGrabber* gets those three "task" parameters.

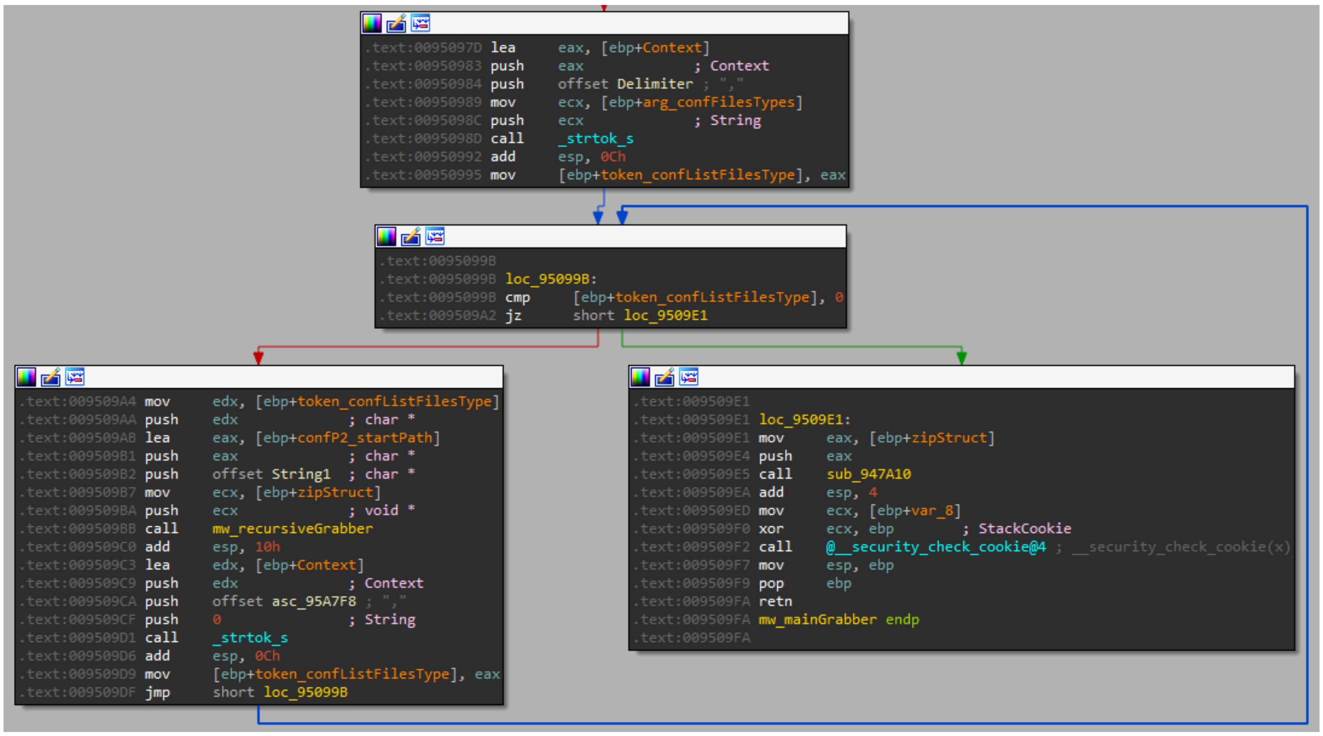

Figure 17: calling to recursiveGrabber and loop of the regex list

While doing this research, we extracted several configurations from other C&Cs, so the grabber configuration looks like:
Documents;USERPROFILE\Documents;*.jpg,*.img,*.json,*.txt;
desktop;USERPROFILE\Desktop;*.jpg,*.img,*.json,*.txt;
For this C&C, the attacker created two tasks to collect jpg, img, json and txt files from the user Desktop and Documents. Oski will save those files in 2 separate zip files named *_Documents.zip* and *_deksop.zip*.

From reviewing the extracted configurations from other C&C servers, we understand that other attackers have intents to collect different files, like 2fa files, wallet files from different locations or even personal documents.

The extracted configuration for other C&C servers can be found in our IoCs page: [IoCs.pdf]

Downloader

After stealing the sensitive data from the user and grabbing the files, Oski adds the stolen files to a new zip file whose name of the contains the 10 characters from the working folder name and an underscore at the beginning.

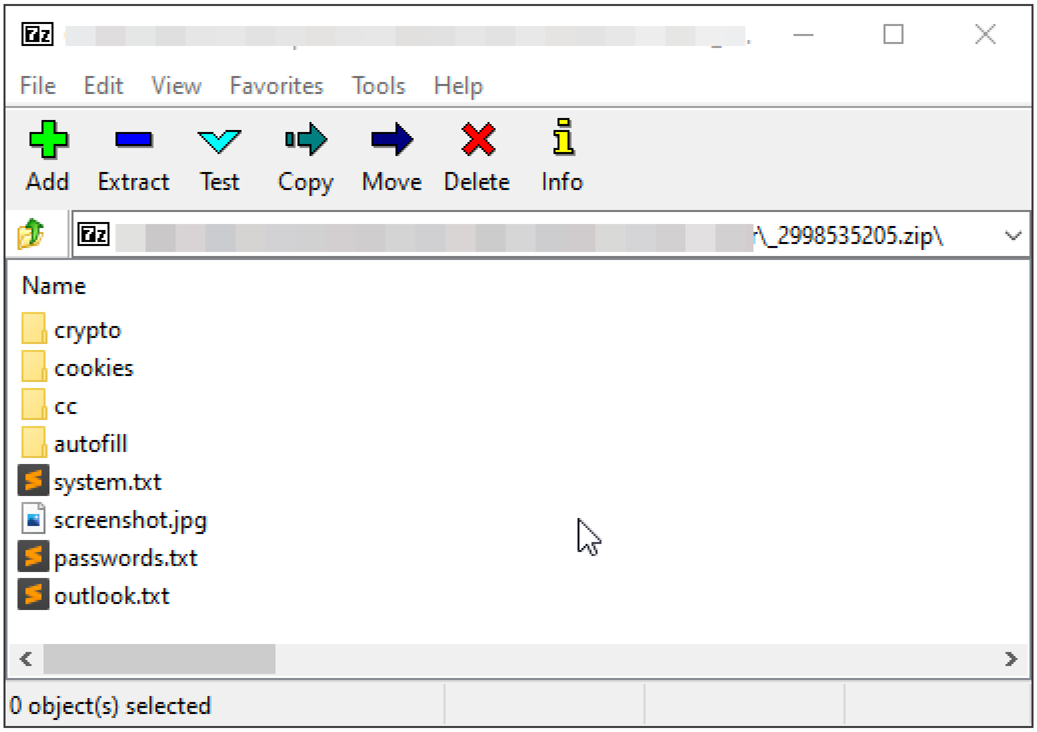

Figure 18: zip file content

After sending the zip file, the C&C server should send within the response the **domain** for the downloader. The response might be empty if the feature isn't enabled.

Oski downloads the next malware from the given domain and executes it.

The stealer creates a random file name with a *.exe* extension and sets the stream Zone.Identifier of the file to [ZoneTransfer] ZoneId=2*,* which indicates that the file has been downloaded from a **trusted site**.

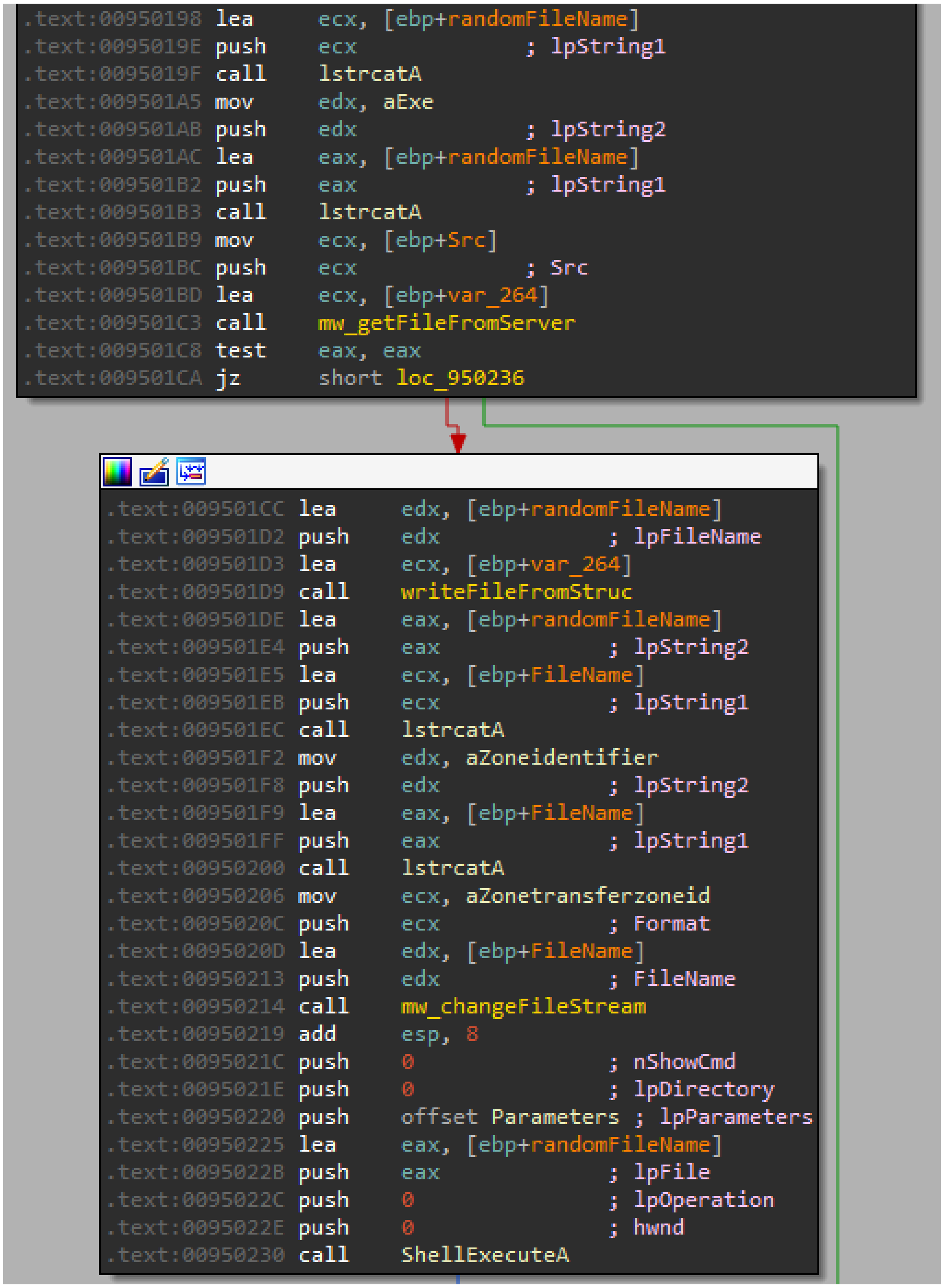

Figure 19: Loader function

Self-Removal

Oski removes its traces from the machine and deletes all the files, logs, DLLs, etc. from the disk.
In addition, it creates a new process of cmd.exe while the parameters for cmd.exe are /c /taskkill /pid <pid> & erase <path> & RD /S /Q <working_folder>* & exit to kill the malware process and delete other files.

Conclusion

Although Oski stealer doesn't target as many types of software as other stealers, it is still effective, continues to be updated and improved and maintains a strong reputation in the underground community.

The unique characteristic of credential theft malware is that they don't require any special permissions. Because of this, they are a popular resource for attacks and ultimately can cause significant damage – especially as attackers continue to seek out privileged credentials and look for opportunities to escalate their privileges for massive data theft or business disruption.

To combat against credential theft malware like Oski, we recommend the following:

- **Be aware** – avoiding clicking suspicious URLs, opening unknown attachments, or downloading and running unfamiliar applications.
- **Deploy MFA** – using multi-factor authentication where applicable.
- **Use strong and unique passwords** – don't use the same passwords for all the services and replace them on a regular cadence.
- **Leverage credential protection solutions** – A credential protection solution can defend against the fundamental nature of credential stealers and protect credentials from getting harvested by attackers.

Appendix

YARA Rule

Oski_Stealer.yara

Targeted Applications

Browsers
Internet Explorer
Google Chrome, Chromium, Kometa, Amigo, Torch, Orbitum, Comodo Dragon, Nichrome, Maxthon, Sputnik, Epic Privacy Browser, Vivaldi, CocCoc Browser, Uran Browser, QIP Surf, Cent, Elements Browser, TorBro, Microsoft Edge, CryptoTab, Brave
Opera
Mozilla Firefox, Pale Moon, Waterfox, Cyberfox, BlackHawk, IceCat, KMeleon

Email Clients
Thunderbird
Outlook

Crypto Wallets
Anoncoin, BBQCoin, Bitcoin, DashCore, ElectronCash, Electrum, Electrum-LTC, Ethereum, Exodus, Florincoin, Franko, Freicoin, GoldCoinGLD, IOCoin, Infinitecoin, Ixcoin, Litecoin, Megacoin, Mincoin, MultiDoge, Namecoin, Primecoin, Terracoin, YACoin, Zcash, devcoin, digitalcoin, jaxx

IoCs

IoCs.pdf

[1] Basics of Windows shellcode writing