Linux.BackDoor.Fysbis.1 — Dr.Web Malware description library

Published: 2015-05-12 · Archived: 2026-04-05 23:35:21 UTC

A multicomponent Trojan presumably related to the Sednit hacker group. It uses a module structure where every module is implemented as a separate class. Modules can be of the following two types: plug-ins and controllers. The researched sample contained two plug-ins: one designed to work with the file system and another one consisting of a remote control shell and a network controller (executes POST and GET requests in a specified format).

During the installation, the Trojan attempts to gain root privileges. If it succeeds, the malware is installed in the folder /bin/ with the name rsyncd and with the description “synchronize and backup service”. If it does not succeed, Linux.BackDoor.Fysbis.1 is installed in ~/.config/dbus-notifier as an executable file with the name dbus-inotifier and with the description “system service d-bus notifier”.

Once it is launched, the Trojan verifies that its copy is not running and that the malware itself is not launched using the command interpreter nash.

1. Verifies that the “echo $0” command’s output is different from “nash”.
2. Verifies that there is no process with the name “rsyncd” on the active process list (“dbus-inotifier” if the Trojan does not have root privileges).

Next, the malware checks whether it is configured to start automatically at system startup.

1. It searches the active process list for the systemd process. If this process is found, the Trojan recursively traverses the “/usr/lib/systemd/” directory and checks every file for the “/bin/rsyncd” string. Otherwise, it runs a search for the “/bin/rsyncd” string within the files found in the /etc/ folder.
2. Verifies that there is no “rsyncd” file in the “/bin/” folder.

If the Trojan does not have root privileges, it checks the “~/.config/autostart/” directory for the “dbus-inotifier” file.

If Linux.BackDoor.Fysbis.1 is not installed, it registers itself in autorun using one of the following methods:

1. Adds the “/bin/rsyncd & exit 0” string to the end of all “rc.local” files found in the /etc/ folder.
2. Creates the service file /usr/lib/systemd/system/rsyncd.service

```
 [Unit]Description= synchronize and backup service.After=syslog.target
[Service].ExecStart=/bin/rsyncd.OOMScoreAdjust=-500
[Install].WantedBy=multi-user.target
```

   Then it installs the service by executing the following commands:

```
 ln -s '/lib/systemd/system/rsyncd.service' '/etc/systemd/system/multi-user.target.wants
/rsyncd.service'
systemctl daemon-reload
```

The running systemd process determines, which option will be used to register the Trojan in autorun. For example, if this process is active, the first option will be used. If it is not active, the second option is to be used.

If the Trojan does not have root privileges to enable its automatic launch, it creates the “~/.config/autostart/dbus-inotifier.desktop” file with the following contents:

```
 [Desktop Entry]
Type=Application
Exec=/home/user/.config/dbus-notifier/dbus-inotifier
Name[en_EN]=system service d-bus notifier
Name=system service d-bus notifier
Comment[en_EN]=
Comment=
```

“/home/user/” stands here for the environment variable HOME.

During the next step, the malware copies itself to the “/bin/rsyncd” folder (or to the “~/.config/dbus-notifier/dbus-inotifier” folder if the Trojan does not have root privileges) and launches the copy from this folder.

The address of the command and control server is stored in the Trojan’s body. All strings used by the Trojan are encrypted with the XOR algorithm. Depending on which task the string corresponds to, different keys are utilized.

Linux.BackDoor.Fysbis.1 creates the directory “/usr/lib/cva-ssys” to store its files in it (“~/.local/cva-ssys”—if the Trojan does not have root privileges). When operating, the Trojan uses the SQLite3 database with the name My_BD. The database is located in the “/usr/lib/cva-ssys/My_BD” folder (“~/.local/cva-ssys/My_BD”—if the Trojan does not have root privileges). The database contains the following two tables: Chnnl(id,binary) and prms(id,dword). The dwell time value with “id == 0x310031” for the standby mode is stored in the prms table. The value stands for the interval, during which the Trojan does not receive a reply with the payload from the command and control server. The value with “id == 0x320032” stands for the dwell time value for active mode. The Chnnl table contains configuration data of the backdoor. This data is encrypted with the RC4 algorithm.

The configuration data used by the backdoor has the following structure:

```
 #pragma pack(push, 1)
struct st_cncconfig
{
  _WORD id;
  _BYTE byte2;
  _BYTE byte3;
  _QWORD pCnCBeg;
  _QWORD pCnCEnd;
```

```
  _QWORD pLastElement;
};
#pragma pack(pop)
```

To be able to enter the data into the database, Linux.BackDoor.Fysbis.1 converts the configuration data into the following structure:

```
 #pragma pack(push, 1)
struct st_crypted_config_data
{
  _WORD id;
  _BYTE byte2;
  _BYTE byte3;
  char* pCnC; //list of CnC addresses separated by '&'
};
#pragma pack(pop)
```

Before the configuration data is encrypted with the RC4 algorithm, 11 signature bytes are added to the end of the data (11 bytes are stored in the backdoor's body). Next, the buffer is encrypted using the RC4 algorithm with the 50-byte key (also stored in the backdoor’s body). If there are keys for the string encryption with the XOR algorithm, the configuration data will be also encrypted with the XOR algorithm.

Then the buffer with the encrypted package is modified as follows:

1. Two DWORD values are added to the beginning of the buffer.
2. The first DWORD value is equal to zero.
3. The second DWORD value is a hashtag and is calculated using the following function (MakeHash):

```
 unsigned __int16 CCryptor::ComputeHash(_BYTE *rc4_key, _DWORD rnd, _BYTE *crypted_data,
 _QWORD size)
{
  _QWORD i;
  _WORD result;
  _BYTE CryptedByte;
  _BYTE j;
  i = 0LL;
  result = 0LL;
  while ( i < size )
  {
    CryptedByte = crypted_data[i];
    j = 0;
    while ( 1 )
    {
      result = ((unsigned __int8)result ^ CryptedByte) & 1 ? (rnd ^ (result >> 1)) :
 (result >> 1);
```

```
      ++j;
      if ( j == 8 )
        break;
      CryptedByte >>= 1;
    }
    ++i;
  }
  return result;
}

unsigned __int32 CCryptor::MakeHash(struct st_cryptor *cryptor, _BYTE *crypted_data,
 __int64 size)
{
  _DWORD rnd;
  rnd = GetRand(0, -1);
  return (unsigned __int16)(HIWORD(rnd) ^ rnd) ^ (CCryptor::ComputeHash
(&cryptor->rc4_key->buffer, (HIWORD(v4) ^ v4), crypted_data, size) << 16);
}
```

The process of the configuration data extraction proceeds opposite to the method described above. When the configuration data is extracted form the database, the backdoor verifies that the hash's calculated value corresponds to the one saved in the database. It also checks the accuracy of the 11-byte signature.

Then the Trojan activates streams for every plug-in that waits for the package containing a command. It also activates one stream to monitor database status, and another one to exchange data with the command and control server.

When the backdoor establishes a connection to the command and control server, it sets the request period time equal to the specified dwell time for the standby mode. Once the Trojan receives the payload, it changes the request period to the dwell time value for the active mode. If the dwell time value for the active mode has been set, but the package has not been received, the dwell time value is incremented by the dwell time value for the active period. This action is repeated until the dwell time value is bigger or equal to the dwell time value for the standby mode.

The Trojan sends the following GET request to the command and control server:

```
 azureon-****.com/watch/?from=W2KIa&oe=YDxQ&aq=KDRHmedegqk&btnG=G&utm=DQ&ai=Y9DmdXRnRMCsX9Mm2KPXQOTAC
azureon-****.com/search/?oe=BiQCNKF&aq=wl&oe=Zcl0al2GeHD&from=rfkpqRi-&ags=KZde&text=x
&ags=AS79lq&channel=YJa3f673&aq=GyZCExee0D&ai=CgX0bplH8YtBf2ZtNYNiCwngv
```

The from, oe, aq, btnG, utm parameters stand for random strings encoded with the BASE64 algorithm. The string length is from 1 to 14 characters. From the list of available parameters, the Trojan randomly chooses the ones it will use (from 2 to 11 parameters).

```
 text=
from=
ai=
ags=
oe=
aq=
btnG=
oprnd=
ai=
utm=
channel=
```

The page address in the domain of the command and control server is chosen randomly from the list.

```
 watch/?
search/?
find/?
results/?
open/?
search/?
close/?
```

The “ai” value stands for the payload title. This value is generated using the following method:

1. The Trojan takes a random DWORD value and 7 bytes of the UID value for GET/POST requests stored in the backdoor’s body. The UID value is followed by the DWORD value equal to -1 if the first DWORD value is zero. Otherwise, the second DWORD value is taken as the first value.
2. 11 bytes of this buffer are encrypted with the XOR algorithm as follows:

```
 i = 0
while ( 1 )
{
  crypted_buffer = (_BYTE *)this_->crypted_buffer;
  if ( i gt;= this-gt;crypted_buffer_size - 4 ) // this-gt;crypted_buffer_size == 15
    break;
  ++i;
  crypted_buffer[i + 4] ^= crypted_buffer[i & 3];
```

3. The generated buffer in encoded using the BASE64 alphabet, where the last two characters are replaced with “-” and “_”.
4. A string with the 5-character length and encoded with BASE64 is added to the beginning of the buffer encoded using the BASE64 algorithm.

In return, the server can send an encoded package or the “400” value. The Trojan checks whether the server's reply is positive by searching for the “OK” substring in it. Then the backdoor checks the reply’s size. If the size is 7 bytes or more, the backdoor verifies that the command and control server sent an encoded package. To decode the package, the BASE64 alphabet is used. The last two characters are replaced with “-” and “_”. If after the package has been decoded its size is bigger than 3 bytes, the Trojan decrypts its first 11 bytes with XOR using the method similar to the one described above.

The first 4 bytes in the received package are ignored; the next 7 bytes are the key that will be used for the next POST requests. The rest of the package is the payload.

The main module of the Trojan can execute the following commands:

Command	**Description**
0x1F	Set the dwell time value for the standby mode
0x29	Activate the controllers
0x2A	Set new configuration data and update the list of command and control servers
0x32	Set the dwell time value for the active mode
0x33	Set up the plug-in
0x33	Save the dwell time values into the database
0x34	Activate the plug-ins
0x35	Add configuration data
0x36	Delete the specified configuration data

The Remote Shell Module can execute the following commands:

Command	**Description**
0x66	Exit
0x65	Open a remote Shell
0x68	Verify that the Shell in running
0x67	Execute a command

The module, which interacts with the file system, can execute the following commands:

cmd	**Description**
0x65	Find the file(s)

0x66	Read the file(s)
0x67	Save the file
0x68	Remove the file(s)
0x69	Run the file(s)

A report on the operations' execution by this module is displayed as the HTML code. The string with this code is generated in the infected computer's memory and is used without being saved into the file.

The module monitoring the database checks the connection to the command and control server every millisecond. If the connection is established, checks the values in the prms table of the database. If these values are other than zero, the module sends them to the command and control server using the POST request.

To send the POST request, the Trojan uses a random DWORD value and 7 bytes of the key from the encrypted package received in reply to the GET request. 11 bytes that the Trojan receives in the reply are encrypted with the XOR algorithm (similar to the one used to decrypt the reply to the GET request). Then the data is added to the encrypted 11 bytes of the key. The generated buffer is encoded using the BASE64 alphabet to be sent in the POST request. Next, to the beginning of the BASE64 string a random BASE64 string with the length of 5 characters is added. The POST request's title is generated similarly to the GET request's title. The payload is generated using the following method:

1. Using zero correction, a random DWORD value is written.
2. Then 11 bytes of the key for POST requests are written. The key is received as a reply to the GET request.
3. Next, other data is added.
4. The first 11 bytes of the received buffer are encrypted with the XOR algorithm.
5. Once it is encrypted using the XOR algorithm, the buffer is encoded with BASE64. A random string with the length of 5 characters is added to the beginning of the buffer.

After the Trojan has sent 4 POST requests via this stream, it pauses for 1 second and then sends another POST request to the command and control server. This request contains the sqlite3 library's functions, whose addresses were successfully acquired (maximum 13).
