Volatility Analysis Contest 2019

Memory Analysis

in Data Leakage Cases

TEAM. HSLFL

INDEX

1. Introduction

1.1. Team & Analyst

TEAM. HSLFL

(Hope Seyeong Lee Find Love)

- Shinwoo Ko
- Geonwoo Lee
- Seyeong Lee
- Chungho Lee
- Huitae Jeong

- Hyunji Moon
- Hyunwoo Shin
- Jongmin Kim
- Jonghyun Kim

We are TEAM. HSLFL from South Korea. Our team consists of nine members, five members(left) are analysts. Four members(right) are team mentors and advisors.

1.2. Background

We introduce several keywords helpful for understanding our scenario selection.

1.2.1. Data Leakage

Data leakage, often called data breach, is a security violation in which sensitive, protected or confidential data is copied, transmitted, viewed, stolen or used by an individual unauthorized to do so [1]. Leaked data include user account, customer information, or company secrets. In the 21st century, more than 5 billion user accounts and customer information have been leaked from various companies, including Yahoo, eBay, Adobe [2].

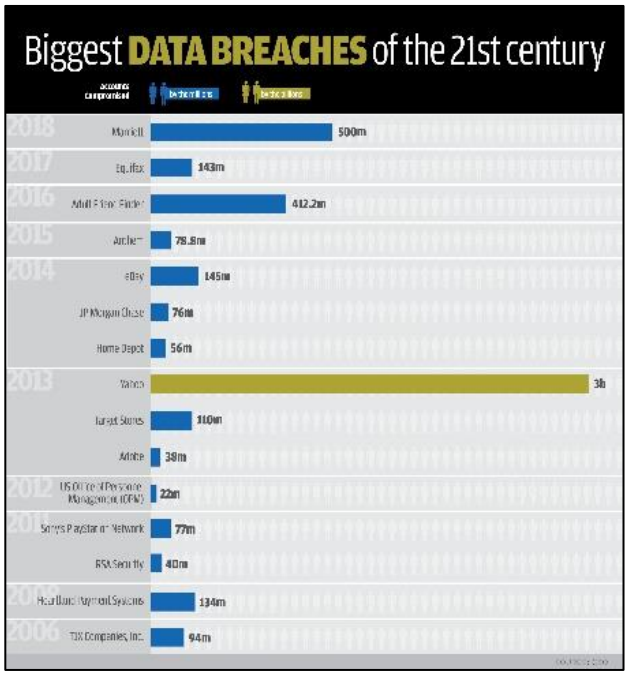


Figure 1. History of Data Breaches of the 21st century

Moreover, recently the almost entire population of Ecuador has data leaked [3]. Internet security firm vpnMentor found this data leakage during a routine project and leaked data include detailed information of almost every person in Ecuador. Data leakage is always a sensitive issue for plenty of companies and nations.

In this report, we focus on data leakage by some malicious hacker, who takes whole control of protected computer that stores specific data that should not be leaked.

1.2.2. Remote Access Trojan (RAT)

RAT is a malicious program that remotely accesses infected resources [4].

RAT enables malicious intruder to perform unauthorized action, including

- Monitoring, Logging User Activity
- Installing, Uninstalling Program
- Transferring, Receiving Data
- Sabotaging the infected resources

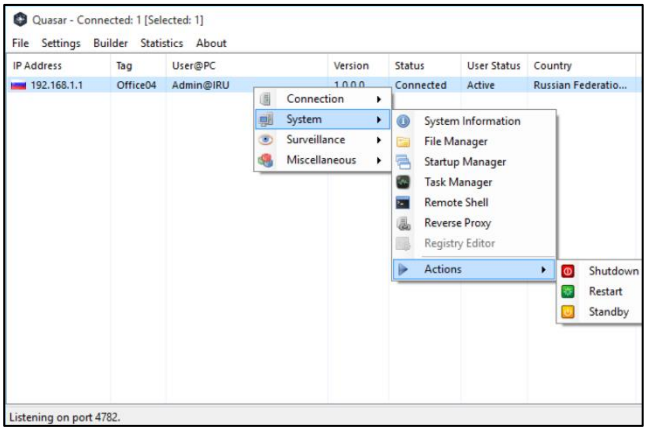


Figure 2. Example of RAT operation (QuasarRAT)

1.2.3. FTP

FTP is the abbreviation for File Transfer Protocol. It usually uses 21/TCP as a control channel and 20/TCP as a data channel. Some schools, companies, institutions, and many groups still rely on FTP for data hosting services. Filezilla, xftp, is one of the well-known FTP based software.

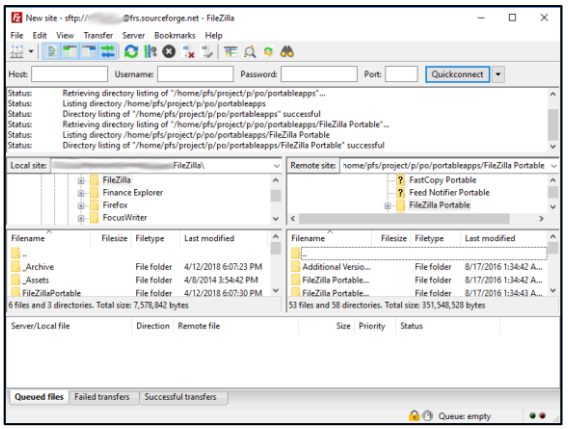


Figure 3. Filezilla

1.2.4. Cloud Storage

Cloud storage is one branch of cloud computing. Wherever the cloud storage is, users can upload their data to cloud storage and download data using devices connected to a network. Google Drive, DropBox, AWS S3 are widely used cloud storage. And some users can use this cloud storages anonymously to minimize their traces on the web.

1.3. Goal

We will show how the Volatility framework can be used to find relevant artifacts of data leakage cases within memory through this analysis report.

2. Overview

2.1. Methodology of Data Leakage

Data leakage can be done by the following:

Physically taking, copying out data resources. It is usually done by downloading protected data to some data storage (CD, USB, etc) and taking out the data from the protected area. If a company fails to build proper security policy (i.e. making the employees pass through some 'Security Checkpoint' system, or not permitting to bring non-authorized electronic devices), one can easily get the secured data out of the company.

Leaking Data via Network

When a malicious hacker takes control of data resources (usually a server or PC), the hacker can use various programs to leak sensitive data to the place where the hacker can access. Since this method does not need physical access to a data resource, hacker can leave less footprint than previous methods.

Still, many methods exist. However, this report focuses on data leakage via network.

2.2. Tools and Storages

We used the following tools and storage in the scenario.

2.2.1. QuasarRAT

QuasarRAT is a fast and light-weight remote administration tool coded in C#. The usage ranges from user support through day-to-day administrative work to employee monitoring [5].

When installed without the user's notice, this RAT tool can perform malicious actions not being detected, including remote shell, file upload, accessing web.

2.2.2. FileZilla

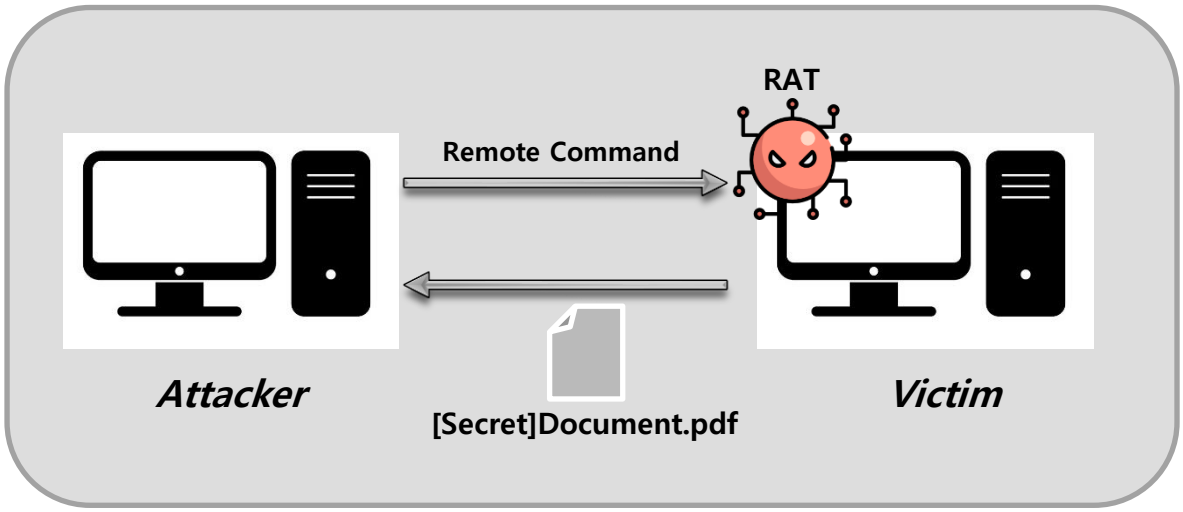
FileZilla is one of the most popular FTP clients available in the market today. This FTP client open source software is distributed free of charge under the terms of the GNU General Public License [6].

2.2.3. Google Drive

Google Drive is one of the most widely used cloud storage. It offers 15GB storage by default.

2.3. Scenario Diagram

Three different data leakage cases are analyzed. Each scenario uses different data leakage method. Only the memory dumps of victim PC are analyzed



Scenario 1. Data Leakage by RAT

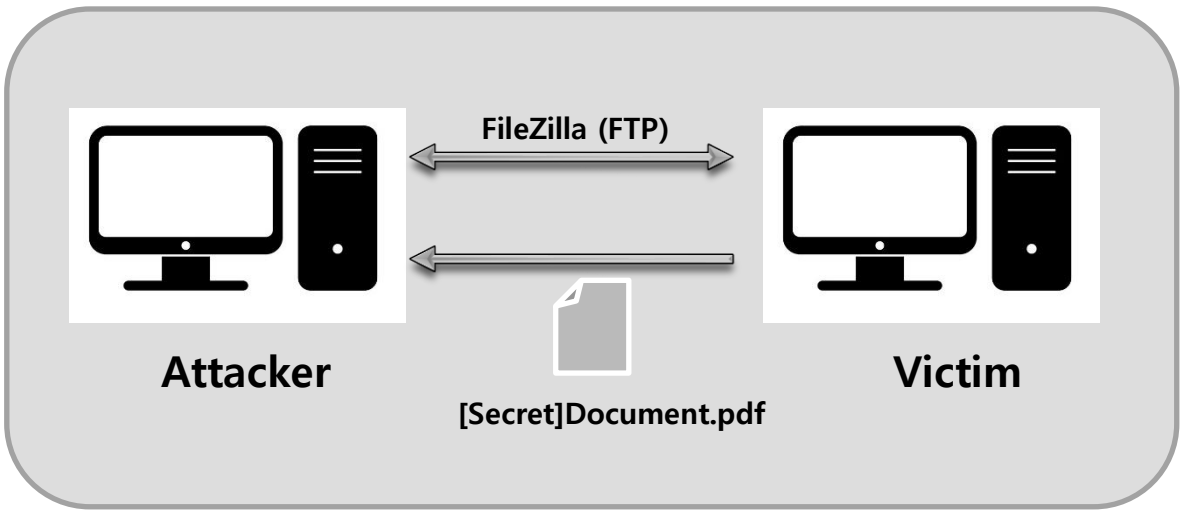
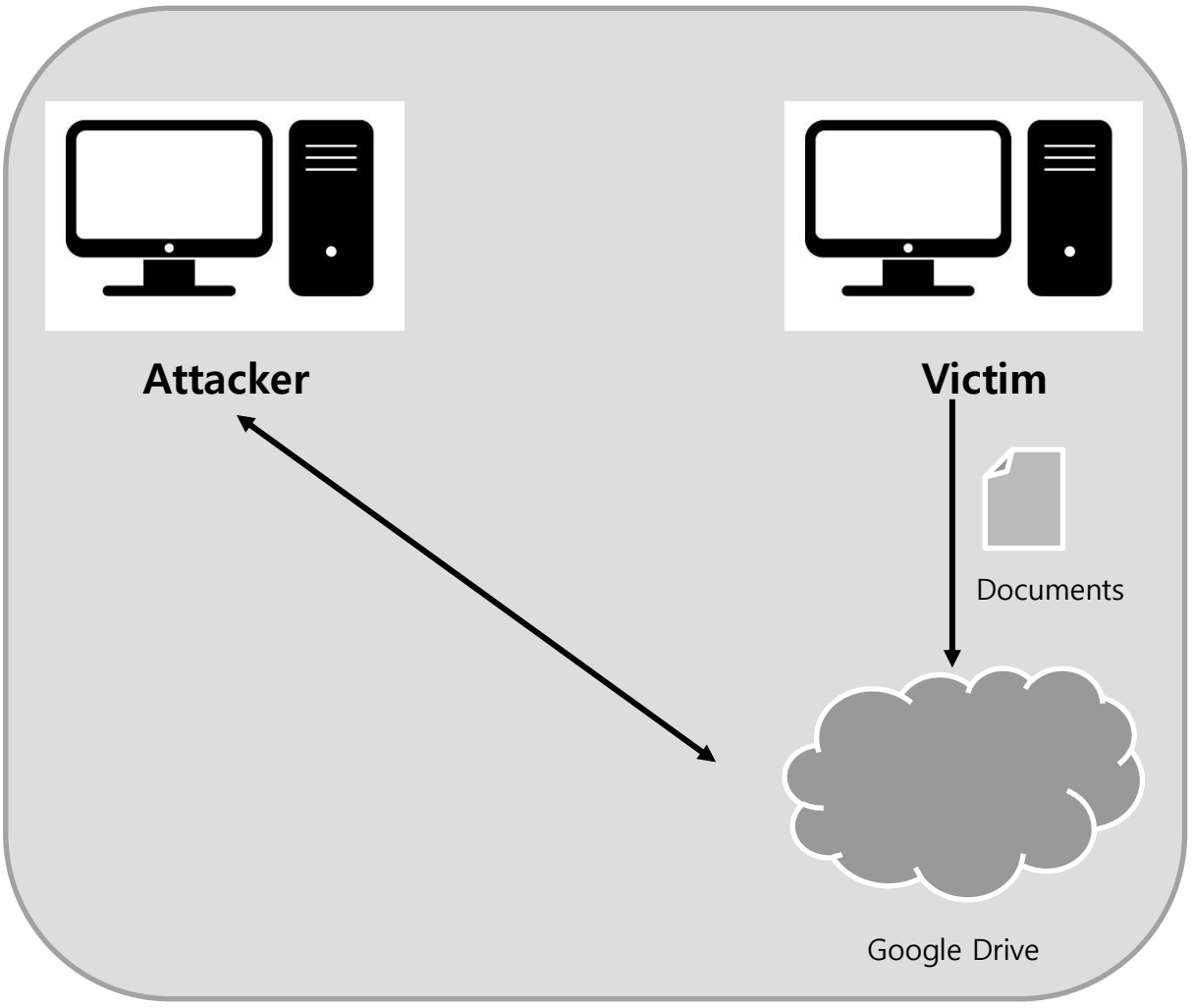


Figure 2. Data Leakage by FTP



Scenario 3. Data Leakage via Web

3. Environment

3.1. Analyst PC Info

We did a memory analysis on a normal-performance laptop.

OS	Windows 10 Pro 64bit
RAM	8GB
CPU	Intel i7-8565U
HDD	256GB(SSD)

3.2. Subject PC Info

The environment of the victim PC is as follows.

OS	Windows 7 32bit
RAM	2GB

3.3. Acquired Memory Dump

3 memory dumps are acquired. Each memory dump is acquired from each scenario. Detailed information is as follows.

Scenario #1 (RAT)

Type	Raw Memory Dump
File Name	RAT_mem.raw
File Size	2.00GB(2,147,483,648 bytes)
SHA1	3044F8FC61DC45421DCFF4F480A26E1C3E642344

Scenario #2 (FTP)

Type	Raw Memory Dump
File Name	ftp_mem.raw
File Size	2.00GB(2,147,483,648 bytes)
SHA1	1163EE968D090559BAEF974EA0AB5294817C63D0

Scenario #3 (Web)

Type	Raw Memory Dump
File Name	drive_mem.raw
File Size	2.00GB(2,147,483,648 bytes)
SHA1	E52DB81B5B93C709E7044FB1C350DC13F75D80CD

4. Analysis

4.1. Scenario 1. Data Leakage by RAT

4.1.1. Tools Used

The tools used for the analysis are as follows.

Tool name	Version	Source
Volatility	2.6 (standalone)	https://github.com/volatilityfoundation/volatility
Notepad++	7.7.1	https://notepad-plus-plus.org/downloads/
strings	2.53	https://docs.microsoft.com/en-us/sysinternals/downloads/strings

4.1.2. Detailed Analysis

Profiles should be specified for accurate analysis of memory images in the Volatility tool. The imageinfo plugin confirms the profile information for the corresponding note image.

The profile of this memory image is Win7SP1x86 as follows.

```
> python vol.py -f RAT_mem.raw imageinfo
```

```
C:\Users\user\Desktop\volatility-master>python vol.py -f RAT_mem.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000,
 Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (C:\Users\user\Desktop\volatil
ity-master\RAT_mem.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82942b78L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x80b96000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2019-09-30 13:05:30 UTC+0000
     Image local date and time : 2019-09-30 22:05:30 +0900
```

Fig 1.1. Imageinfo Result

Commands entered through the cmdscan plugin were identified. After checking user account information and finding a PDF document file containing the word "Secret", it was found that the file "[Secret]Document.pdf" was viewed.

Based on these circumstances, since the location of the secret document was determined through the command and the contents of the secret document were verified (Fig. 1.2), it was judged as an act of leakage of confidential document and the analysis was carried out.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 cmdscan
```

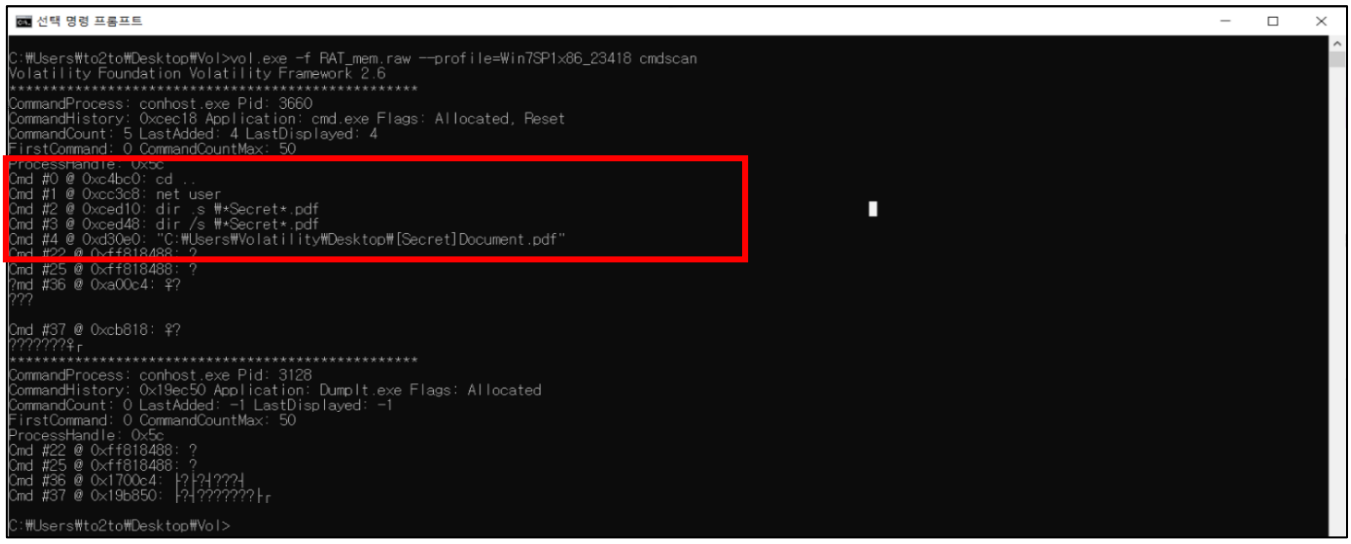


Fig 1.2. cmdscan Result

To check the list of processes, the pstree plugin was run to verify the analysis results.

```
> vol.exe -f RAT_mem.raw --profile=Win7SP1x86_23418 pstree
```

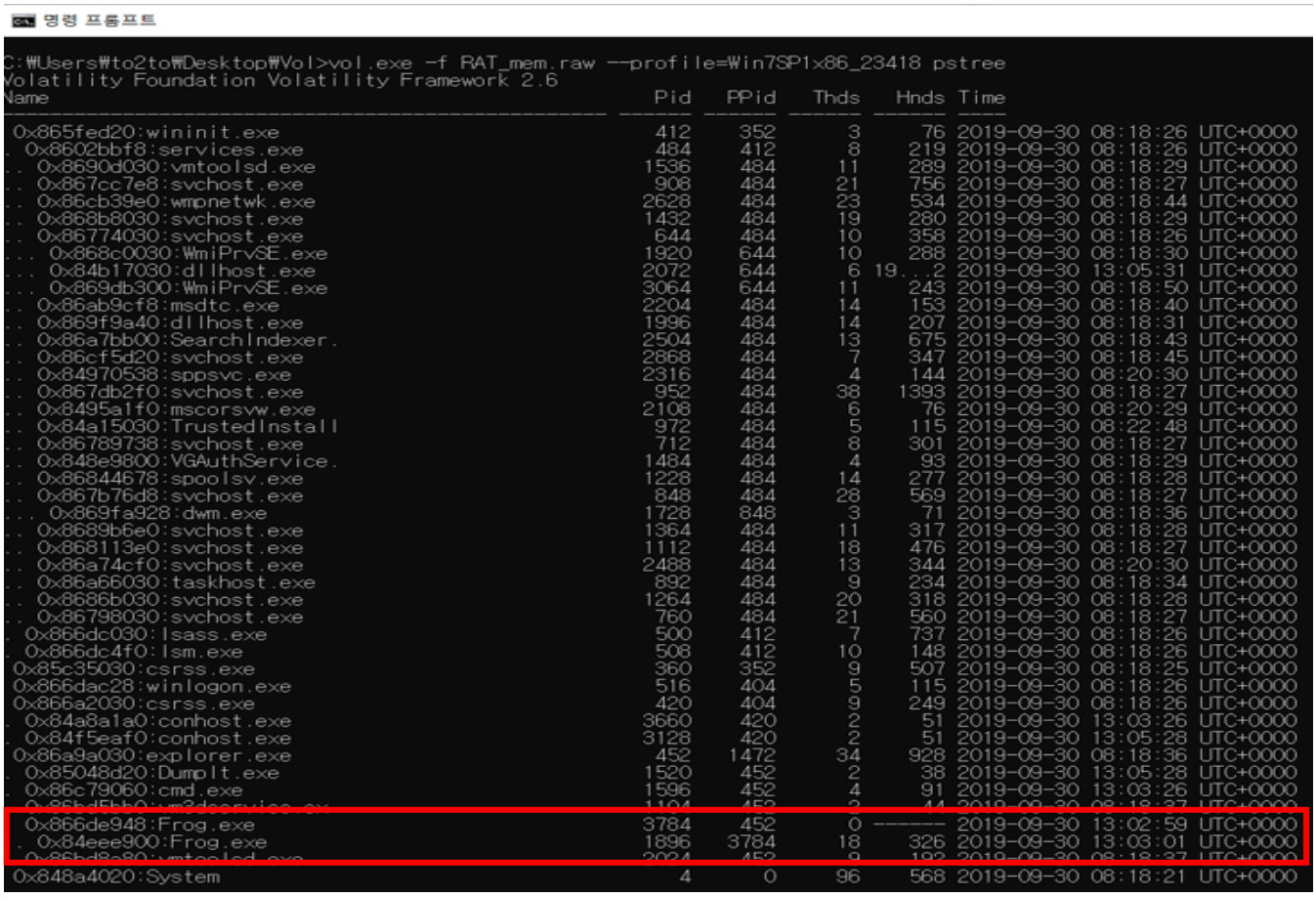


Fig 1.3. pstree Result

A parent, child process (Fig 1.3) with a suspicious name "Frog.exe" could be identified between normal service processes.

Parent process	Child process
Frog.exe (PID : 3784)	Frog.exe (PID : 1896)

The Handles plugin was used to check the handled files of the processes.

```
> vol.exe -f RAT_mem.raw --profile=Win7SP1x86_23418 handles -p 3784
```

For the Frog.exe (3784) process, no information was available.

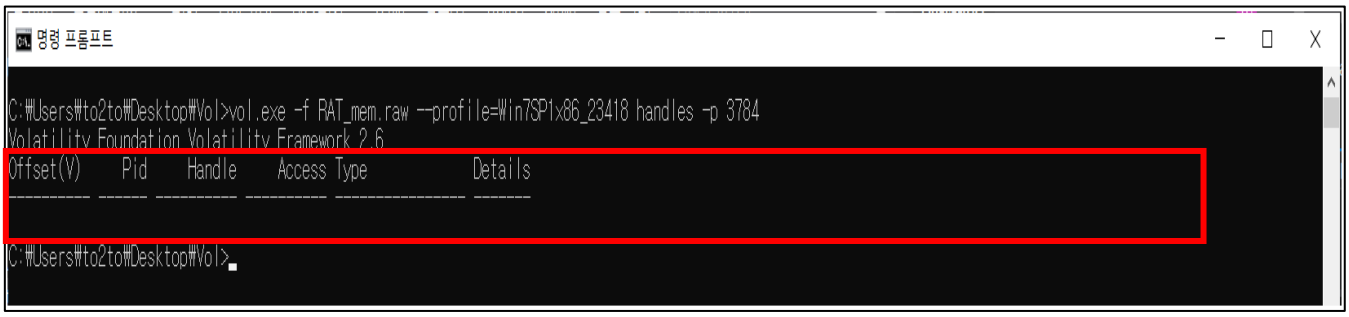


Fig 1.4. handles Result - 1

For the Frog.exe (1896) process, various information could be found.

```
> vol.exe -f RAT_mem.raw --profile=Win7SP1x86_23418 handles -p 1896
```

However, no information related to the secret documents was found.

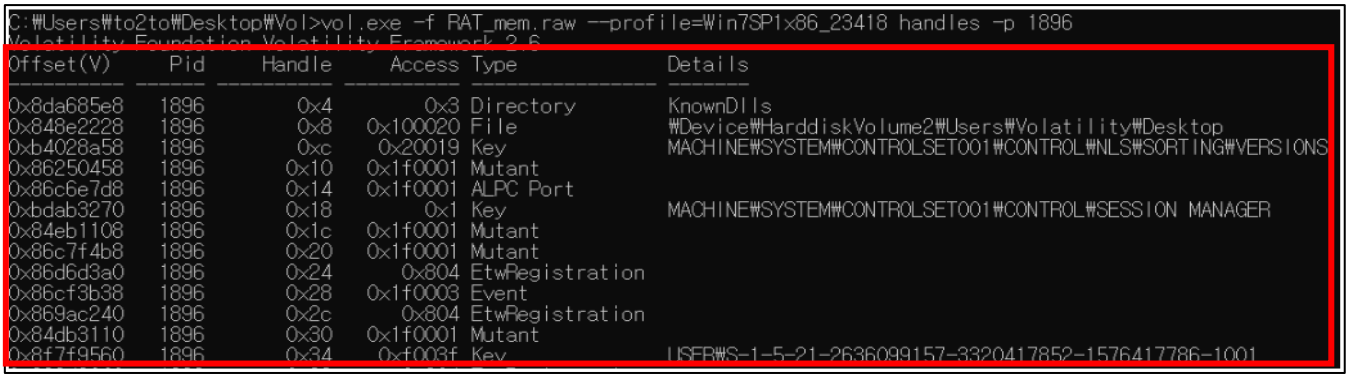


Fig 1.5. handles Result - 2

To investigate the behavior of the Frog.exe process in more detail, network communication records in the memory were analyzed using the netscan plugin. (Fig 1.6)

```
> vol.exe -f RAT_mem.raw --profile=Win7SP1x86_23418 netscan
```

A total of two IP records were confirmed. One IP was in the closed state and the other IP was in the established state of the connection.

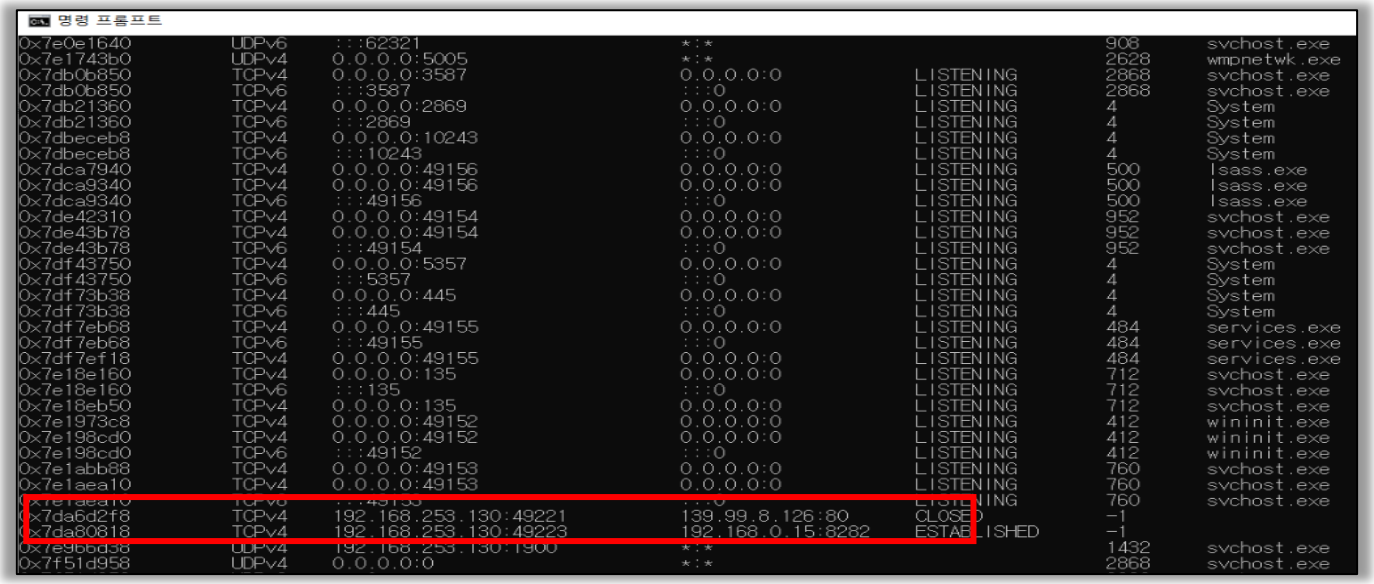


Fig 1.6. netscan Result

In general, the netscan can be used to obtain recipient, caller IP, and process ID information. However, in the above two records (Fig 1.6) no information was obtained about the Process Name, Process ID.

IP	Port	State
139.99.8.126	80	Close
192.168.0.15	8282	ESTABLISHED

The memdump plugin was used to obtain information about the process of communicating with that IP. The analysis was performed by dumping the process memory of Frog.exe (1896) and Frog.exe (3784).

```
> vol.exe -f RAT_mem.raw --profile=Win7SP1x86_23418 memdump -p 1896
```

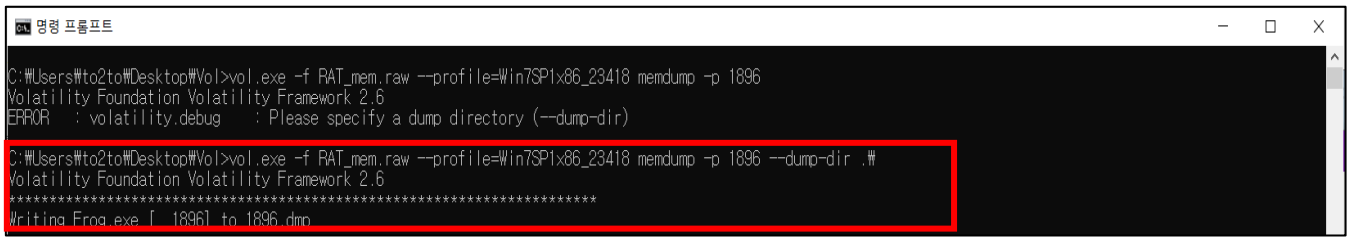


Fig 1.7. memdump Result - 1

```
> vol.exe -f RAT_mem.raw --profile=Win7SP1x86_23418 memdump -p 3784
```

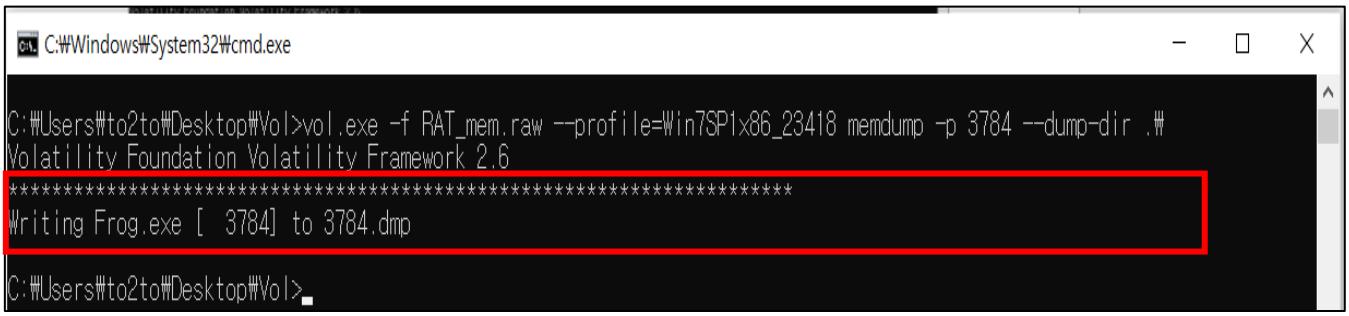


Fig 1.8. memdump Result - 2

"Secret" was filtered from the memory dump of the Frog.exe (3784) process and the information about the secret documents was verified.

```
> strings 3784.dmp | find “Secret”
```

```
C:₩Users₩user₩Desktop₩volatility-master>strings 3784.dmp | find "Secret"
₩Device₩HarddiskVolume2₩Users₩Volatility₩Desktop₩Secret
₩Device₩HarddiskVolume2₩Users₩Volatility₩Desktop₩Secret
*Secret<.pdf
BCryptDestroySecret
BCryptSecretAgreement
NCryptSecretAgreement
BCRYPT.BCryptDestroySecret
BCRYPT.BCryptSecretAgreement
GetSecretAgreementInterface
NCryptSecretAgreement
BCryptDestroySecret
BCryptSecretAgreement
GetSecretAgreementInterface
Secretaria de Economia Mexico
LocalStoreAccountSecret
[Secret]Document.pdf
[Secret]Document.pdf[SECRET]DOCUMENT.PDF
<<[Secret]Document.pdf
₩Device₩HarddiskVolume2₩Users₩Volatility₩Desktop₩[Secret]Document.pdf
```

Fig 1.9. strings Result - 1

However, the analysis results obtained through the NetScan plugin failed to verify information about the IPs that were in the process of establishing the communication connection.

```
> strings 3784.dump | find “192.168.0.15:8282”
```

```
C:\Users\user\Desktop\volatility-master>strings 3784.dmp | find "192.168.0.15:82
82"
```

Fig 1.10. strings Result - 2

In the Frog.exe (1896) process memory dumps, information related to the secret document were also found.

```
> strings 1896.dmp | find “Secret”
```

```
C:\Users\user\Desktop\volatility-master>strings 1896.dmp | find "Secret"
- "C:\Users\Volatility\Desktop\[Secret]Document.pdf"
BinarySecret
1714x884\Volatility\Desktop\[Secret]Document.pdf
C:\Users\Volatility\Desktop\[Secret]Document.pdf
[Secret]Document.pdf
C:\Users\Volatility\Desktop\[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Document.pdf
C:\Users\Volatility\Desktop\[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Document.pdf
C:\Users\Volatility\Desktop\[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Document.pdf
[Secret]Doc
[Secret]Doc
VC:\Users\Volatility\Desktop\[Secret]Document.pdf
```

Fig 1.11. strings Result - 3

Finally, the process of communicating 192.168.1.15:8282 (IP:PORT) Frog.exe (1896) could be checked.

```
> strings.exe 1896.dmp | find “192.168.0.15”
```

Fig 1.12. strings Results – 4

After checking the corresponding results (Fig 1.13), further memory dumps were analyzed.

```
> strings.exe 1896.dmp
```

It was confirmed that the rarely seen cmd commands "echo DONT CLOSE THIS WINDOW!", "ping -n 10 localhost > null" are hard-coded inside the binary.

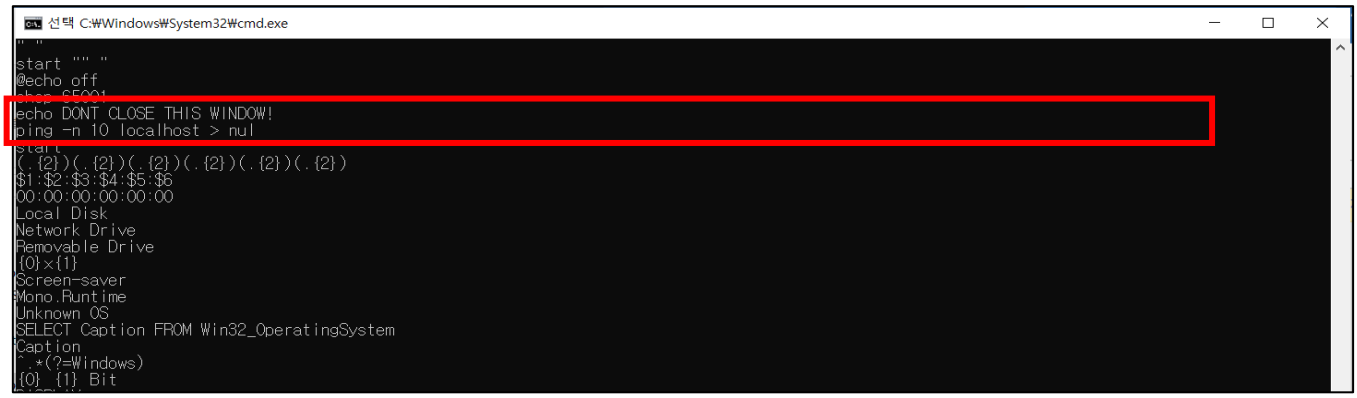


Fig 1.13. strings Result - 5

In addition, the word "Remote" was frequently found, and it can be inferred that the process involved remote control.

```
<RemotePort>k__BackingField
get_RemotePort
set_RemotePort
RemotePort
<RemotePath>k__BackingField
get_RemotePath
set_RemotePath
RemotePath
<RemoteAdresses>k__BackingField
<RemotePorts>k__BackingField
get_RemoteAdresses
set_RemoteAdresses
get_RemotePorts
set_RemotePorts
RemoteAdresses
RemotePorts
RemoteAddress
```

Fig 1.14. strings Result - 6

Based on the information obtained through further analysis, we looked over the Internet and found a YARA RULE that detects QuasarRat.

https://github.com/search?q=%E2%80%9Cecho+DONT+CLOSE+THIS+WINDOW%21%E2%80%9D+%22RAT%22&type=Code

To prove that the process was created using QuasarRat, we compared the strings used in the QuasarRat project v1.3.0.0 and the strings extracted from the memory dump.

```
> strings.exe 1896.dmp > strings.txt
```

The project uses .Net Framework 4.0 in its basic settings and the extracted string refers to .NetFramework 4.0.

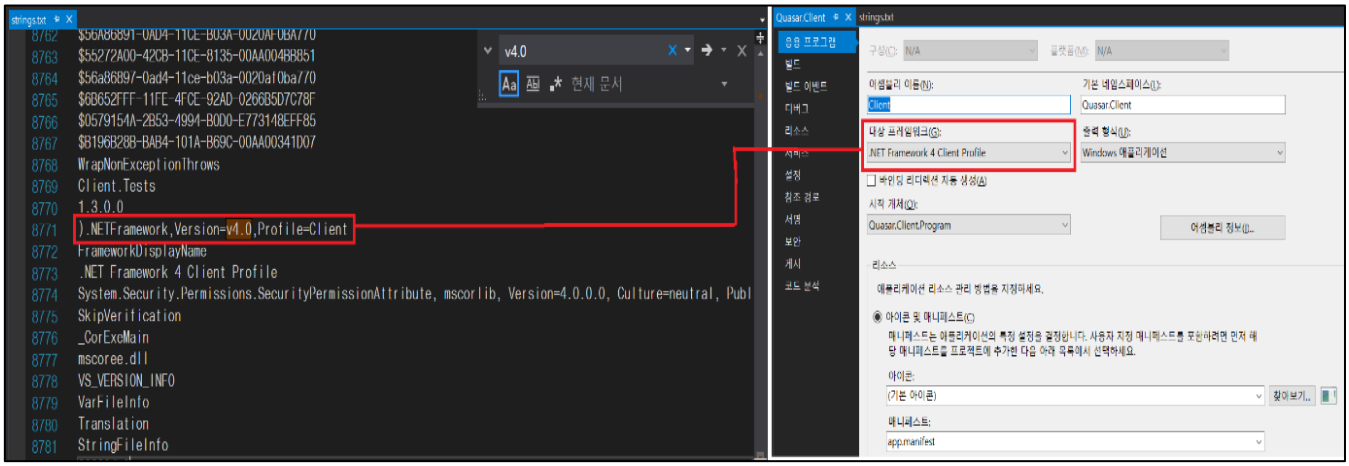


Fig 1.15. .Net FrameWork Version = 4.0

The strings such as "bat", "@echo off", "chcp 65001", "echo DONT CLOSE THIS WINDOW" used in the project and the extracted strings matched.

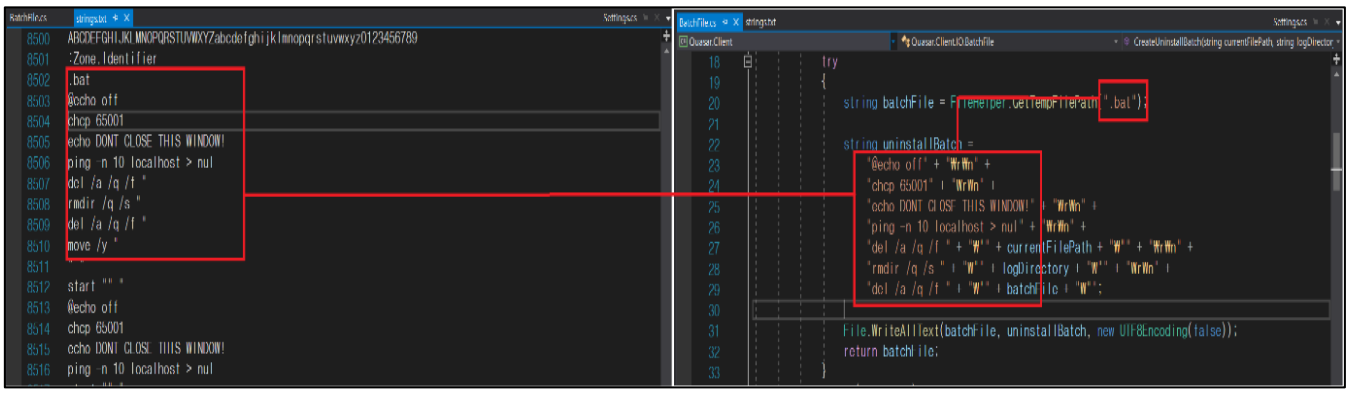


Fig 1.16. String Comparison

Long strings such as “Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/7046A194A” also match the extracted string.

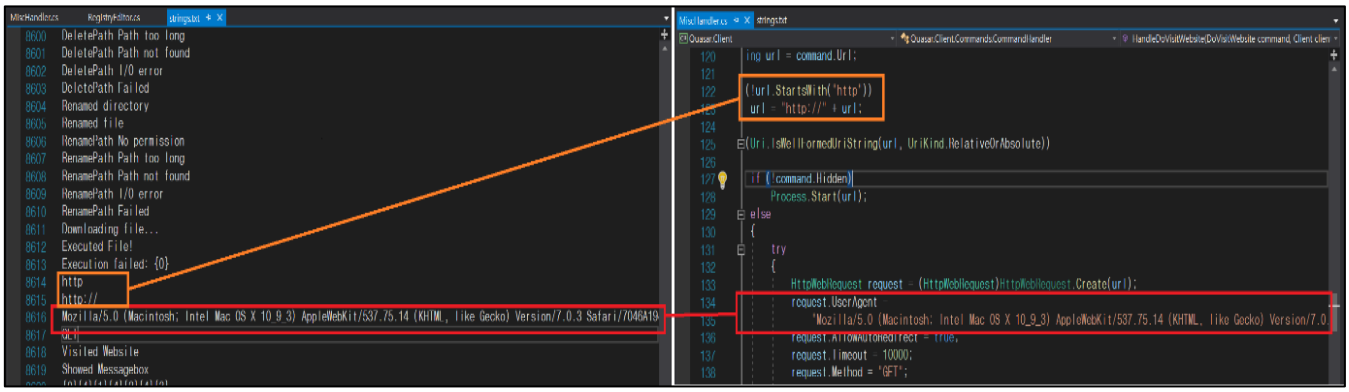


Fig 1.17. String Comparison in Advance

Therefore, the Frog.exe process was created using QuasarRat and can be identified as having leaked the [Secret]Document.pdf file to the remote IP of 192.168.0.15.

4.2. Scenario 2. Data Leakage through FTP

4.2.1. Tools Used

Tool name	Version	Source
Volatility	2.6 (standalone)	https://www.volatilityfoundation.org/releases
hxd	2.3.0	https://mh-nexus.de/en/hxd/
SublimeText	3.2.1	https://sublimetext.com/3
WinprefetchView	1.35	https://www.nirsoft.net/utils/win_prefetch_view.html

4.2.2. Detailed Analysis

The first thing to do is to acquire OS information. The Plugin of the Volatility tool requires Profile information, which can be obtained from the Imageinfo command.

```
> vol.exe -f ftp_mem.raw imageinfo
```

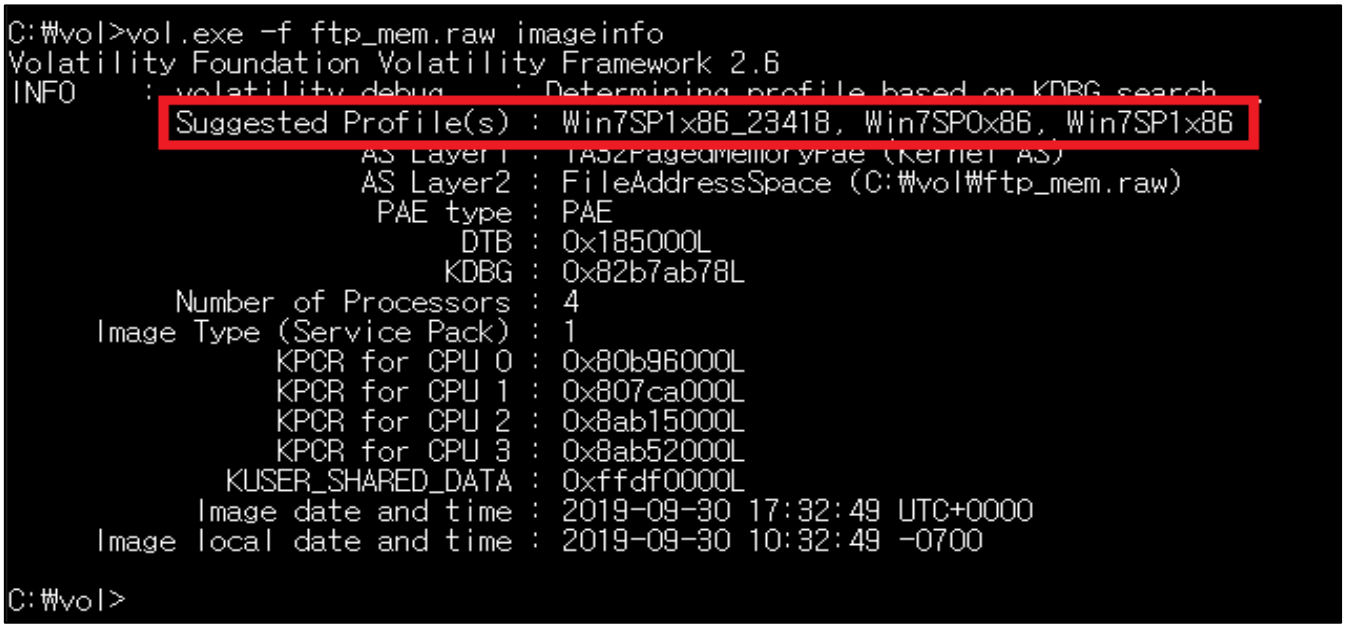


Fig 2.1. Imageinfo Results

Generalizing with the information obtained through Imageinfo allows us to specify that it is a Windows 7 x86 environment.

The second thing to do is to find a suspicious process. Process information can be obtained through Pslist plugin.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 pslist
```

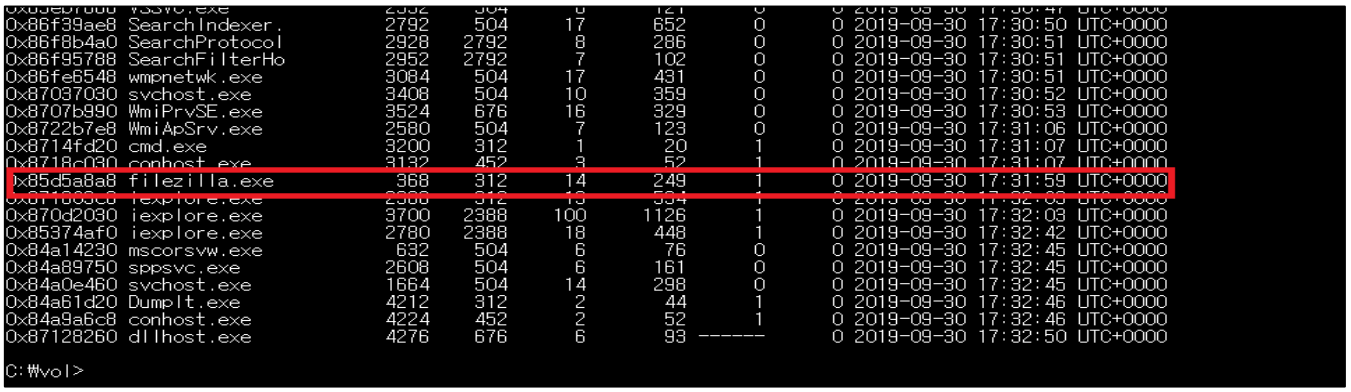


Fig 2.2. Pslist Result

To specify a process that is assumed to be related to data leakage, filezilla.exe can be suspected first, as shown in Figure 2.3. Extracting the file through the Procdump plugin and check the attribute, we can check the Filezilla FTP Client. (Fig 2.3)

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 procdump --dump-dir . -p 368
```

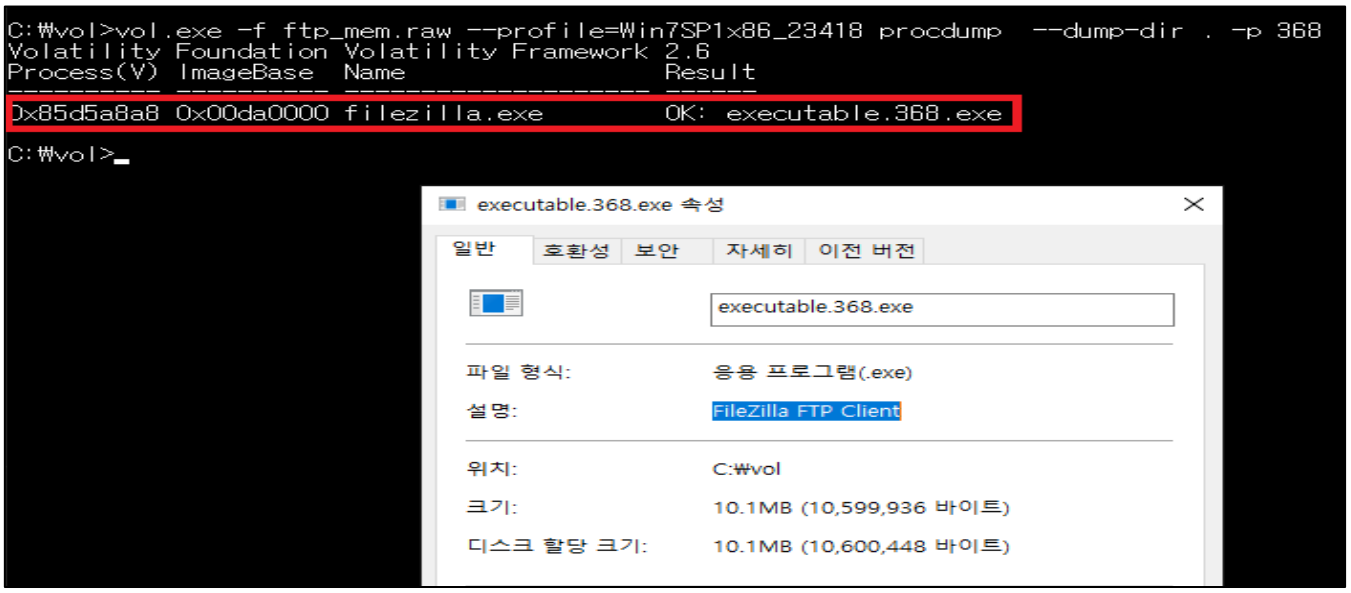


Fig 2.3. Export filezilla.exe

It could be a normal "Filezilla Client" software, but it could be an attacker's tampered file, so the file was looked upon VirusTotal. As Figure 2.4. shows, it can be assumed that it is a normal file or Malware that is not detected by the vaccine because it is not Detectable by many Vaccines

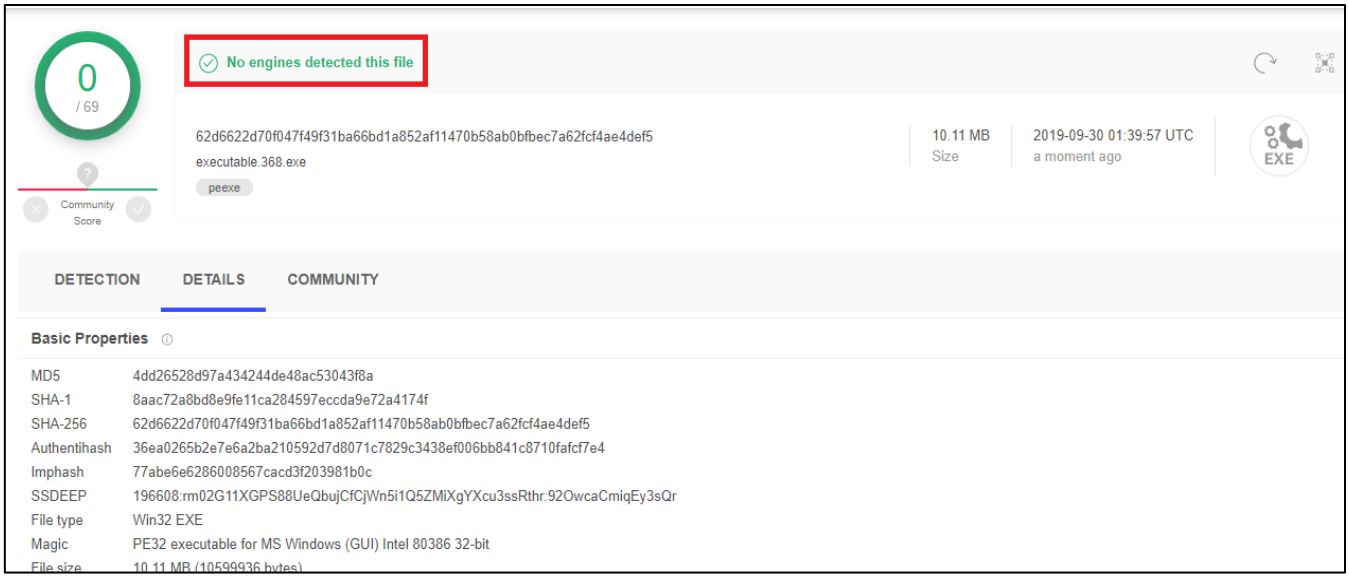


Fig 2.4. VirusTotal Result

To check if there is currently "filezilla.exe" on victim PC, we used a filescan plugin to search.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 filescan | findstr -i filezilla | findstr -v png
```

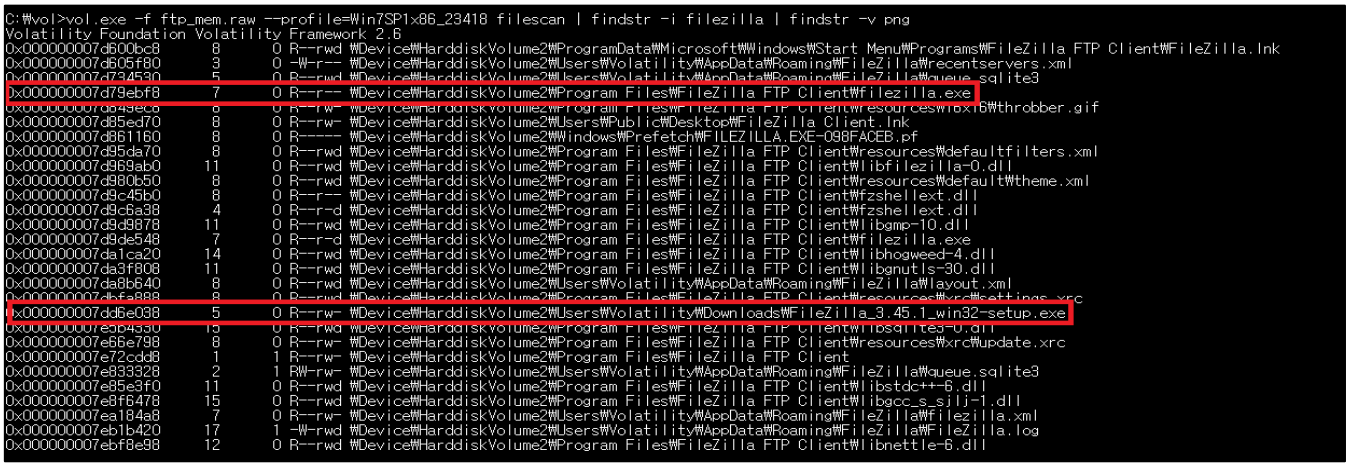


Fig 2.5. Filescan Results(for Execute File)

FileZilla.exe existed and the file "FileZilla_3.45.1_win32-setup.exe" exists separately. Filezilla.exe failed to run due to a Dependency problem. Additionally, for the "FileZilla_3.45.1_win32-setup.exe" file, the Image Section was corrupted and the file could not be exported.

vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007d9de548 -D .
vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007dd6e038 -D .

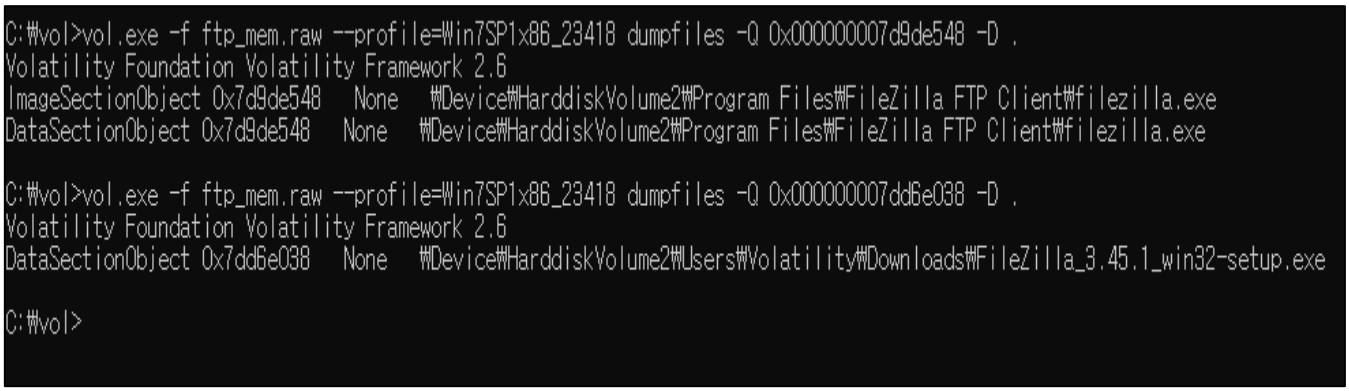


Fig 2.6. Dumpfiles Results

Comparing the version of "filezilla.exe" and “https://download.filezilla-project.org/client/FileZilla_3.45.1_win32-setup.exe", it can be inferred that the installation file was downloaded for updates from the "Filezilla.exe" file.

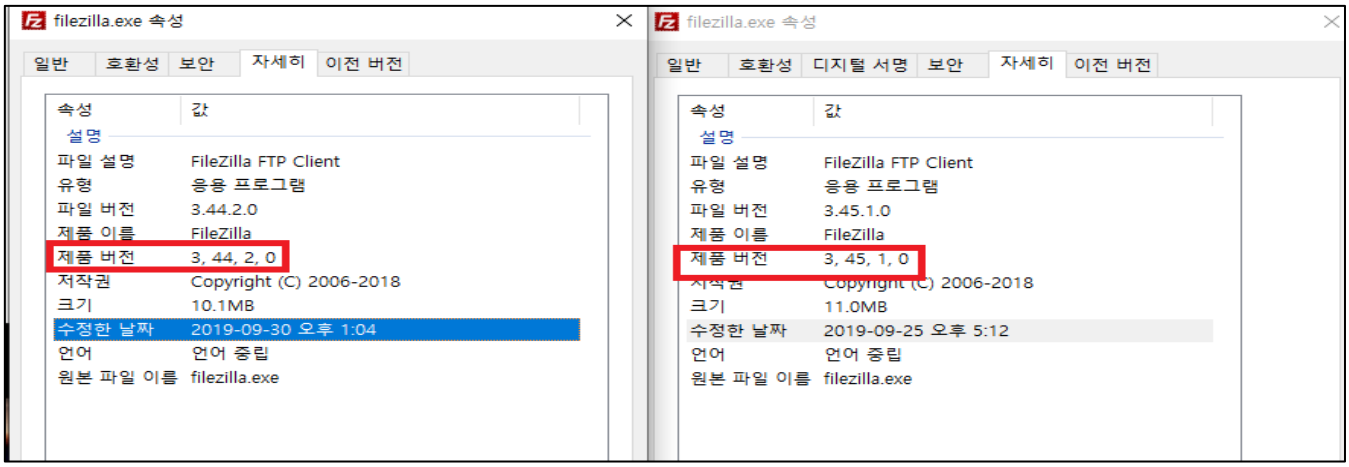


Fig 2.7. Compare filezilla.exe

To prove this, we extracted the "FILEZILLA.EXE-098FACEB.pf" file and checked it.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q
0x000000007d861160 -D .
```

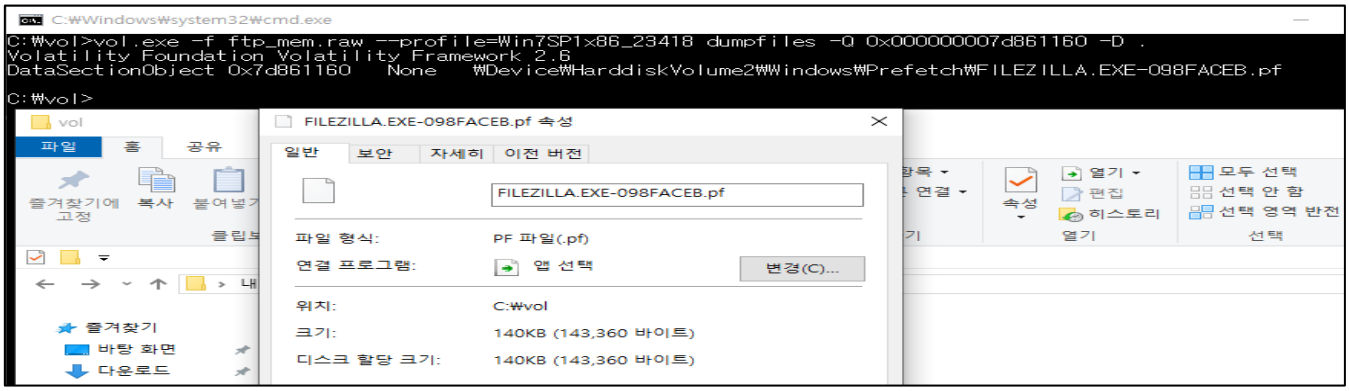


Fig 2.8. Dumpfiles Result (Prefetch)

From the following Figure 2.9, it can be inferred that the file was downloaded because access to the file "FileZilla_3.45.1_win32-setup.exe" from FileZilla.exe is checked by WinPrefetchView. In addition, the artifacts accessed from the filezilla.exe are listed.

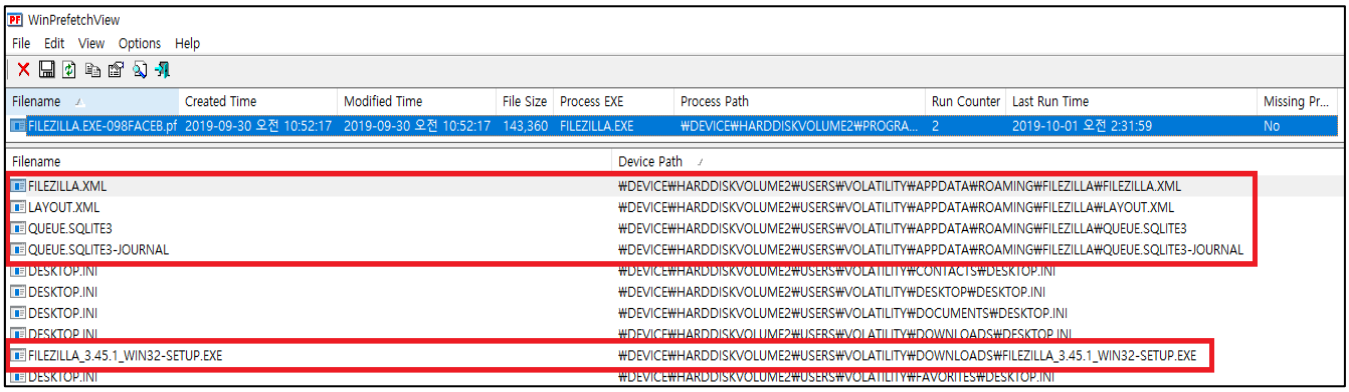


Fig 2.9. Filezilla.pf

Some of the normal Filezilla Software's artifacts are as follows:

File Name
%APPDATA%₩FileZilla₩filezilla.xml
%APPDATA%₩FileZilla₩layout.xml
%APPDATA%₩FileZilla₩queue.sqlite3
%APPDATA%₩FileZilla₩recentservers.xml

Since most of the artifacts remain in line with the tables written above, "filezilla.exe" is slightly more likely to be a normal file. Thus, the "%APPDATA%FileZilla₩" path was looked up to obtain the Artifact to be analyzed.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 filescan | findstr
₩Roaming₩FileZilla₩
```

```
C:₩vol>vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 filescan | findstr ₩Roaming₩FileZilla₩
Volatility Foundation Volatility Framework 2.6
0x000000007d605f80      3      0 -W-r-- ₩Device₩HarddiskVolume2₩Users₩Volatility₩AppData₩Roaming₩FileZilla₩recentservers.xml
0x000000007d734530      5      0 R--rwd ₩Device₩HarddiskVolume2₩Users₩Volatility₩AppData₩Roaming₩FileZilla₩queue.sqlite3
0x000000007da8b640      8      0 R--rwd ₩Device₩HarddiskVolume2₩Users₩Volatility₩AppData₩Roaming₩FileZilla₩layout.xml
0x000000007e833328      2      1 RW-rw- ₩Device₩HarddiskVolume2₩Users₩Volatility₩AppData₩Roaming₩FileZilla₩queue.sqlite3
0x000000007ea184a8      7      0 R--rw- ₩Device₩HarddiskVolume2₩Users₩Volatility₩AppData₩Roaming₩FileZilla₩filezilla.xml
0x000000007eb1b420     17      1 -W-rwd ₩Device₩HarddiskVolume2₩Users₩Volatility₩AppData₩Roaming₩FileZilla₩FileZilla.log

C:₩vol>
```

Fig 2.10. Filescan Result(Filezilla Artifact)

Analyzed the file "recentservers.xml" first. The file contains the most recently connected information, such as Figure 2.11.

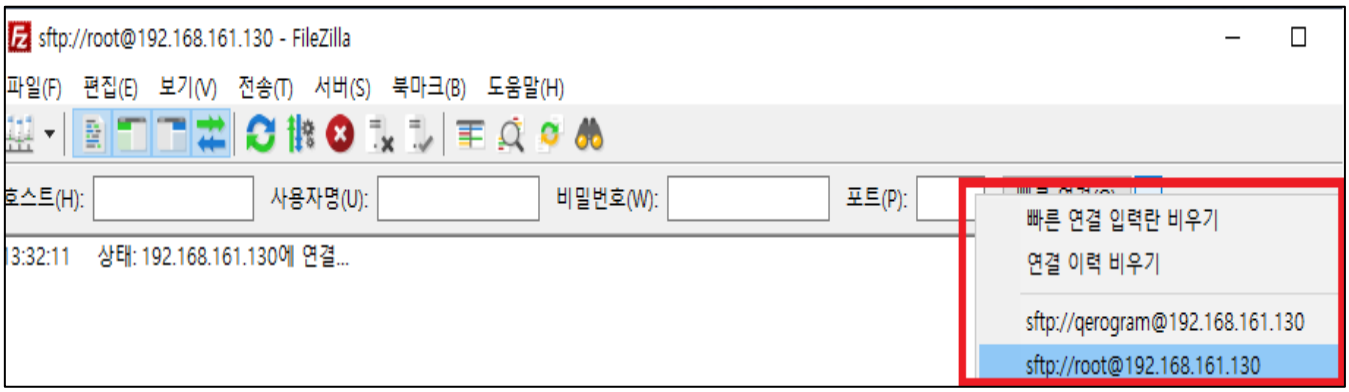


Fig 2.11. Filezilla Recent Connect Menu

Extracting the “recentservers.xml” gives a trace of accessing the domain “ftp.hackers.vol” domain with port 21.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q
0x000000007d605f80 -D .
```

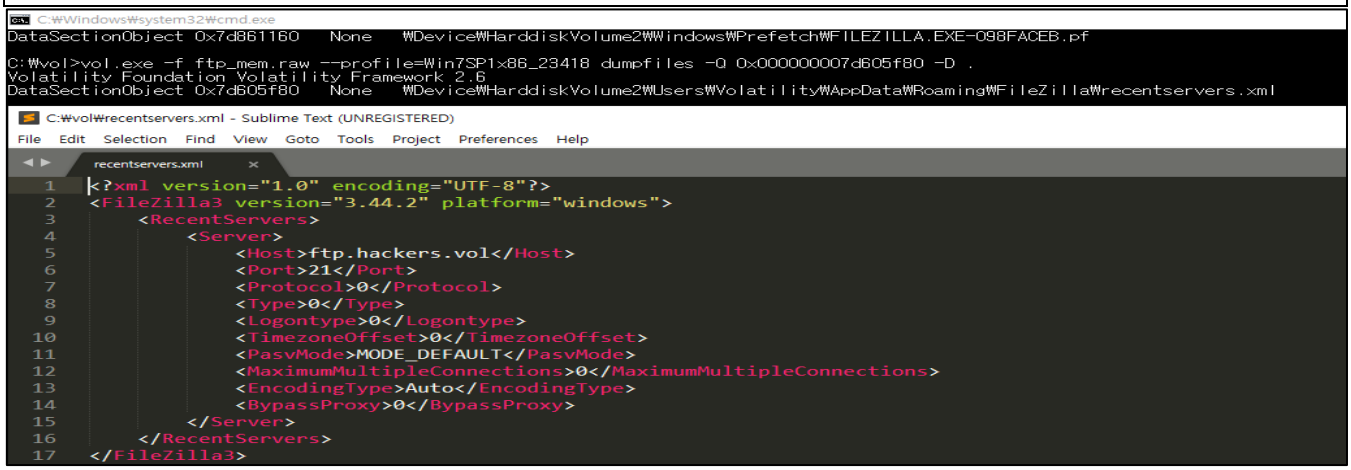


Fig 2.12. Content in recentsserver.xml

I inquired if the domain existed through Fig. 2.13 and found that it did not exist. Afterward, domain modulation through hosts file modulation was found.

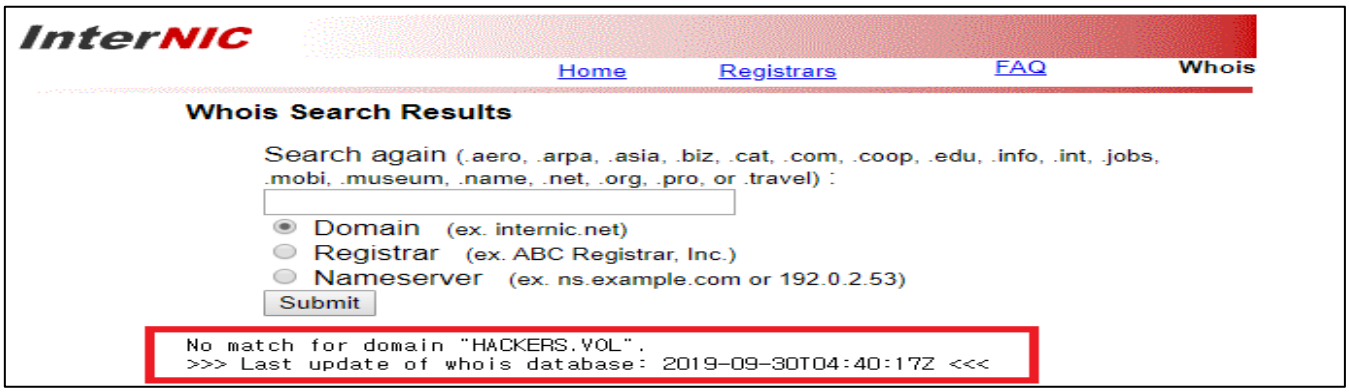


Fig 2.13. Not Found domain("hackers.vol")

Extracting the hosts file gives that real IP address of "ftp.hackers.vol" is 192.168.161.1

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q
0x000000007dd45308 -D .
```

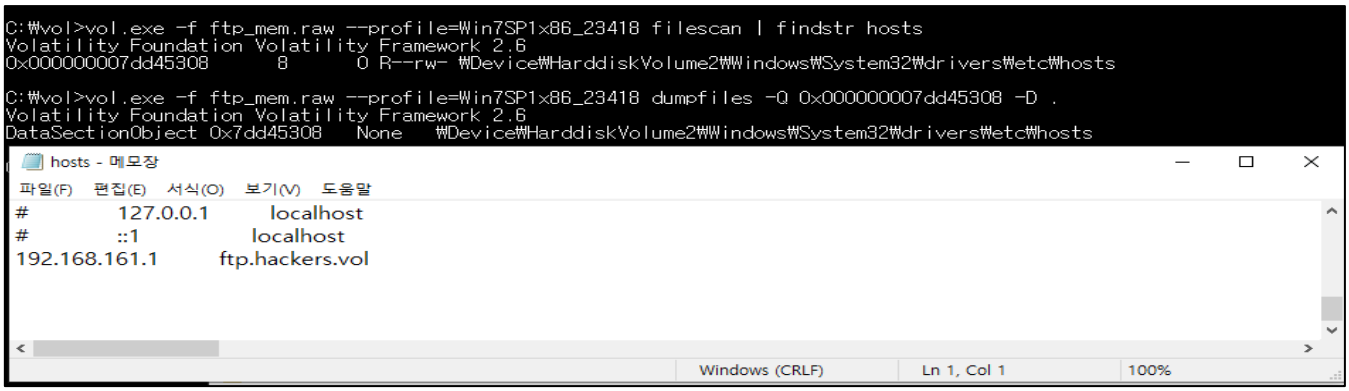


Fig 2.14. Content in hosts

The netscan plugin was used to verify that the acquired IP and user are currently communicating. The results showed that the 192.168.161.133 (User IP) is currently in communication with 192.168.161.1:21. (Fig 2.15)

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 netscan | findstr 192.168.161.1:2
```

```
C:₩vol>vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 netscan | findstr 192.168.161.1:2
Volatility Foundation Volatility Framework 2.6
0x7d7230c8         TCPv4    192.168.161.133:49374          192.168.161.1:21     ESTABLISHED
0x7d733ba8         TCPv4    192.168.161.133:49400          192.168.161.1:21     ESTABLISHED

C:₩vol>
```

Fig 2.15. Netscan Result

The next suspect is the "Filezilla.log" file. This is because, as mentioned earlier, this is not a Filezilla Artifact list left for “Roaming”. Again, extract "Filezilla.log" using dumpfiles plugin.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q
0x000000007eb1b420 -D .
```

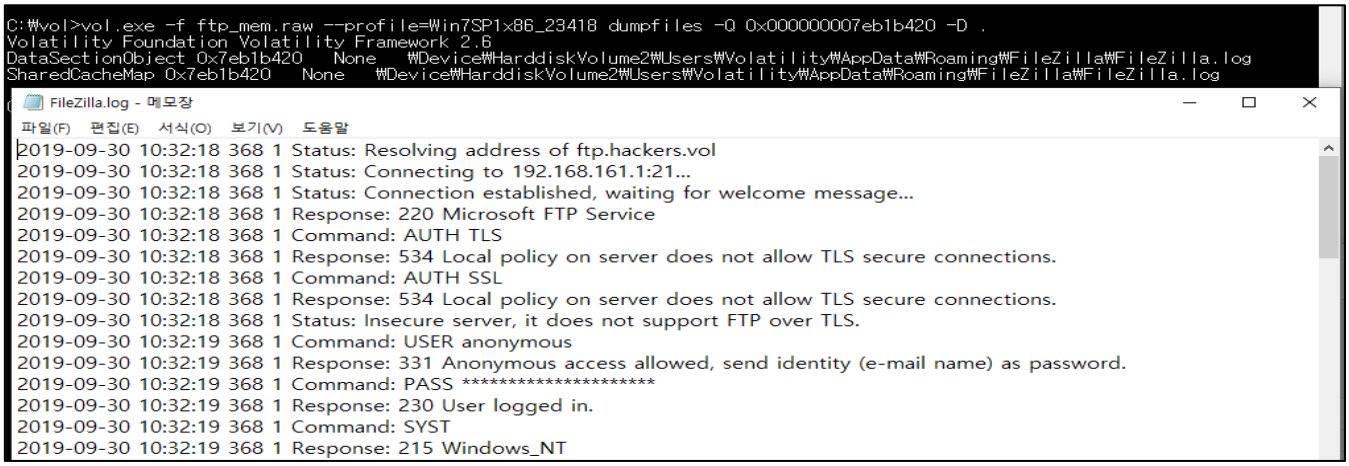


Fig 2.16. Content in Filezilla.log

We were able to capture the context of FTP communications with "ftp.hackers.vol" through Fig 2.16. In addition, we can check there is login to the target FTP server with an anonymous account, which means that you have deployed the server to enable Anonymous FTP. The response from the SYST Command shows that the server is a Windows environment through "Response: 215 Windows_NT."

```
2019-09-30 10:32:28 368 2 Response: 250 CWD command successful.
2019-09-30 10:32:28 368 2 Command: PWD
2019-09-30 10:32:28 368 2 Response: 257 "/" is current directory.
2019-09-30 10:32:28 368 2 Command: TYPE I
2019-09-30 10:32:28 368 2 Response: 200 Type set to I.
2019-09-30 10:32:28 368 2 Command: PASV
2019-09-30 10:32:28 368 2 Response: 227 Entering Passive Mode (192,168,161,1,254,36).
2019-09-30 10:32:28 368 2 Command: RETR petya.bin.gz
2019-09-30 10:32:28 368 2 Response: 125 Data connection already open; Transfer starting.
2019-09-30 10:32:28 368 2 Response: 226 Transfer complete.
2019-09-30 10:32:28 368 2 Status: File transfer successful, transferred 133,423 bytes in 1 second
2019-09-30 10:32:37 368 2 Status: Starting upload of C:₩Users₩Volatility₩Desktop₩[Secret]Document.pdf
2019-09-30 10:32:37 368 2 Command: PASV
2019-09-30 10:32:37 368 2 Response: 227 Entering Passive Mode (192,168,161,1,254,47).
2019-09-30 10:32:37 368 2 Command: STOR [Secret]Document.pdf
2019-09-30 10:32:37 368 2 Response: 125 Data connection already open; Transfer starting.
2019-09-30 10:32:37 368 2 Response: 226 Transfer complete.
2019-09-30 10:32:37 368 2 Status: File transfer successful, transferred 787,939 bytes in 1 second
```

Fig 2.17. Content in Filezilla.log

There exists critical evidence of data leakage. 2019-09-30T10: 32:27, "[Secret]Document.pdf" file is transferred through FTP, and we extracted the file to specify what it is like.

```
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 filescan | findstr pdf
> vol.exe -f ftp_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007d68c2a8 -D .
```

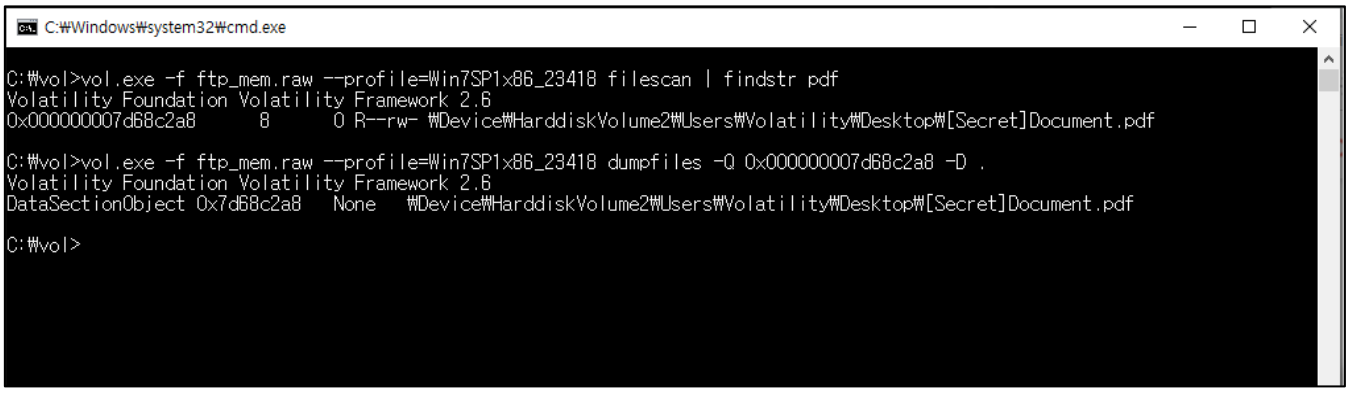


Fig 2.18. Dumpfiles Result([Secret]Document.pdf)

However, extracted files and FTP-transferred data do not have the same size. This means that after 787,939 bytes, all bytes have been padded to zero by the size of the cluster.

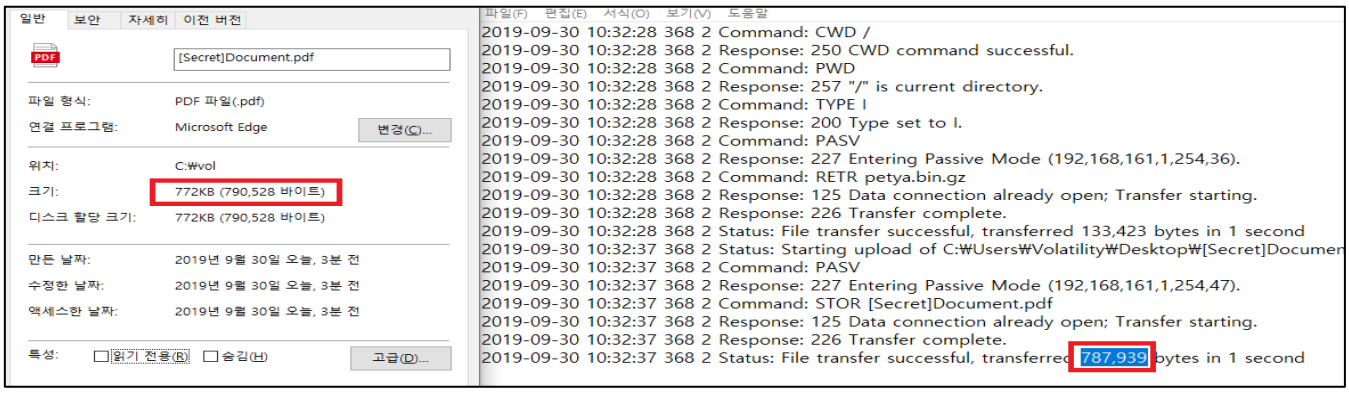


Fig 2.19. Compare Size

As shown below, all values have been padded to 00.

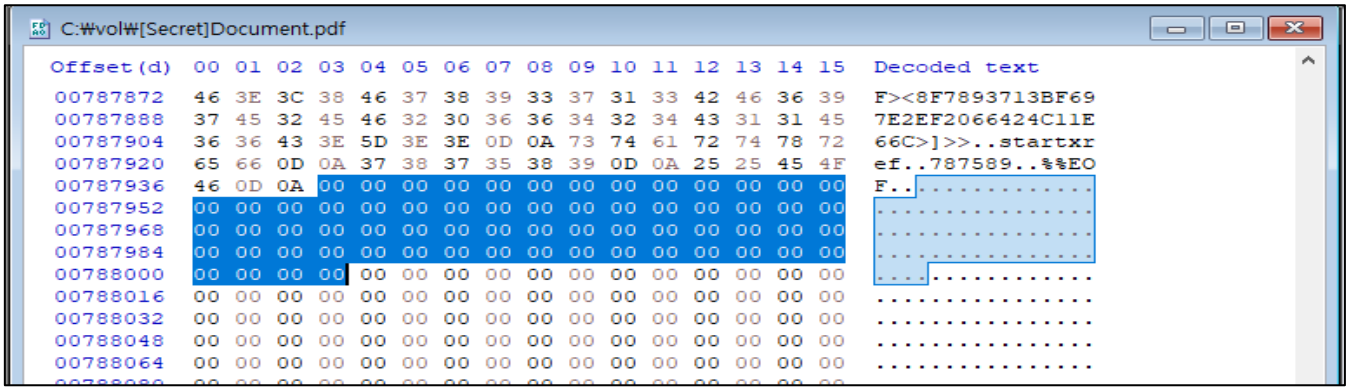


Fig 2.20. Append Padding in [Secret]Document.pdf

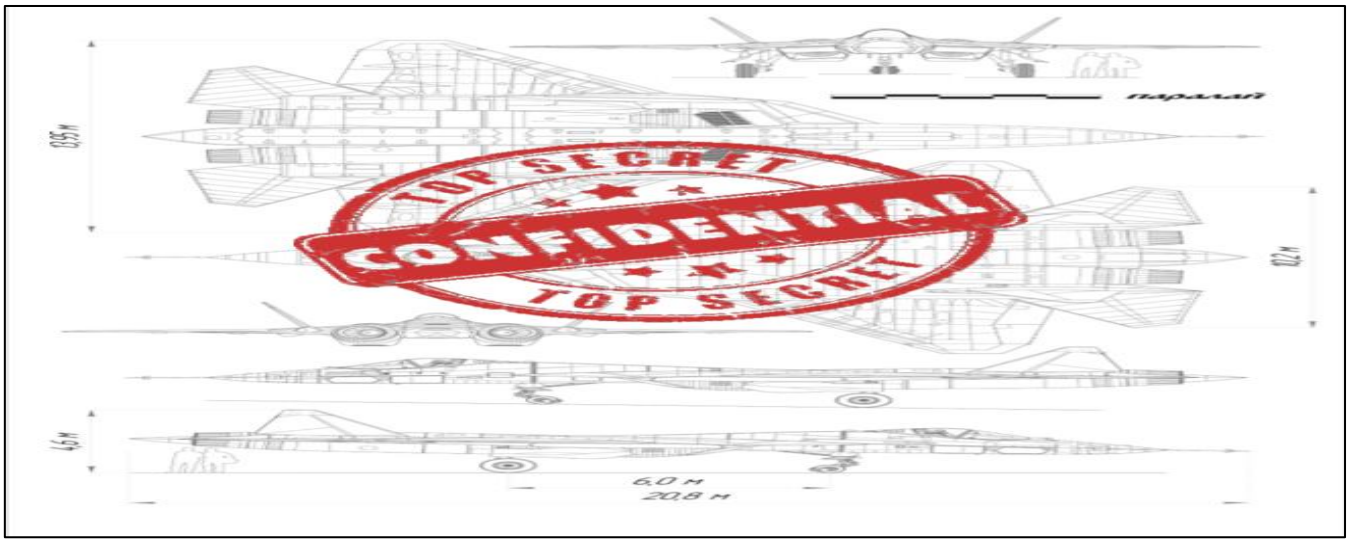


Fig 2.21. Content in [Secret]Document.pdf

Removing the padding from the transferred file, the file is the same as the original file. The hash can also be seen in Figure 2.23. Therefore, the actual leaked file is "[Secret]Document.pdf" with the above image.

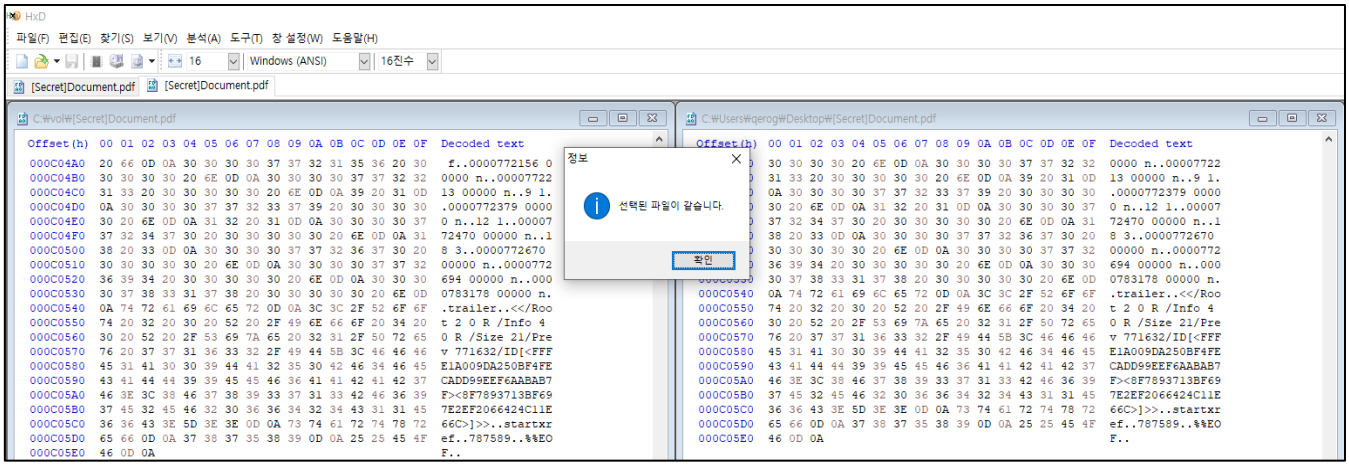


Fig 2.22. Compare Original File & Export File

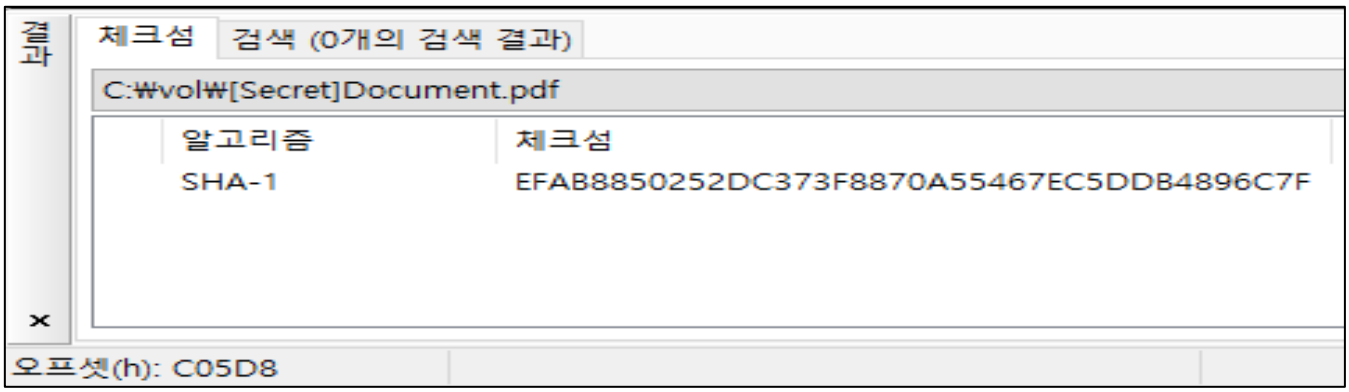


Fig 2.23. “[Secret]Document.pdf” Hash

The reason why the FTP login "Filezilla.log" file was present was that the Logging settings were set in the Filezilla Client Software, as shown in Figure 2.24, so the logs were acquired.

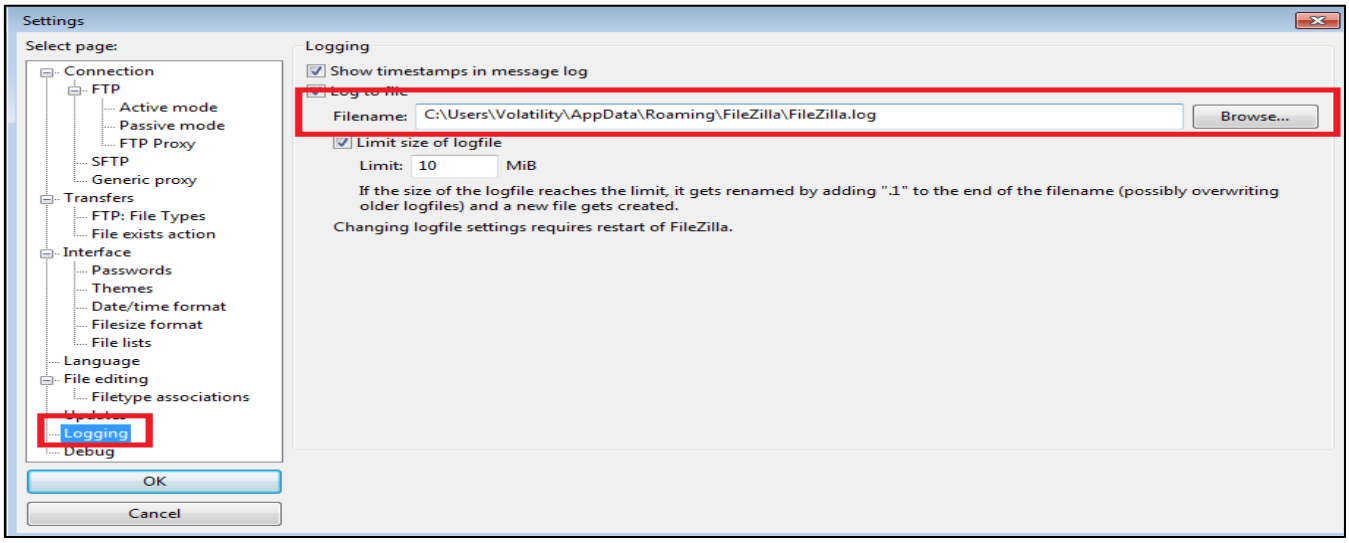


Fig 2.24. Set Logging in Filezilla Client

4.3. Scenario 3. Data Leakage via Web

4.3.1. Tools Used

Tool name	Version	Source
Volatility	2.6.1	https://www.volatilityfoundation.org/releases
hxd	2.2.1	https://mh-nexus.de/en/hxd/
SQLite Browser	3.11.2	https://sqlitebrowser.org/dl/
DCode	4.2.0.9306	https://www.digital-detective.net/

4.3.2. Detailed Analysis

First, we tried to analyze the operating system through the hardware information and generation time of the memory dump file. We used imageinfo command to gather Profile information.

```
> vol.py -f drive_mem.raw imageinfo
```

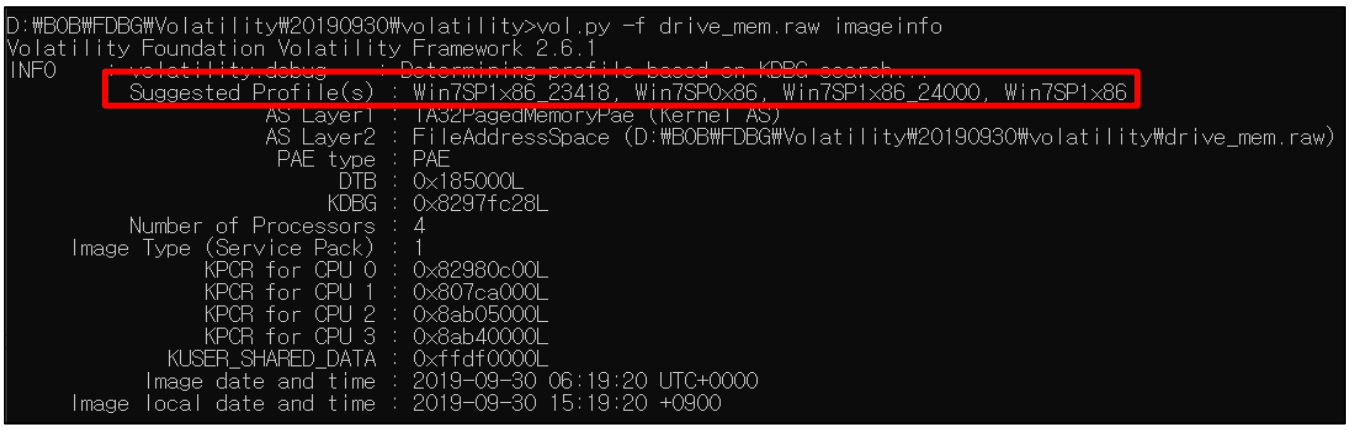


Fig 3.1. Imageinfo results

If you check the results through Netscan, we can assume that the browser the attacker used to leak data is chrome.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 netscan | findstr -i chrome
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 netscan | findstr -i chrome
Volatility Foundation Volatility Framework 2.6.1
0x7d3253c0         UDPv4    0.0.0.0:59231                  *:*                                    3676     chrome.exe     2019-09-30 06:18:18 UTC+0000
0x7d38a378         UDPv4    0.0.0.0:59228                  *:*                                    3676     chrome.exe     2019-09-30 06:17:23 UTC+0000
0x7d3ba9c0         UDPv4    192.168.243.132:58546          *:*                                    3512     chrome.exe     2019-09-30 06:19:19 UTC+0000
0x7d5eb848         UDPv4    0.0.0.0:53443                  *:*                                    3676     chrome.exe     2019-09-30 06:17:22 UTC+0000
0x7d86f7d0         UDPv4    0.0.0.0:54183                  *:*                                    3676     chrome.exe     2019-09-30 06:17:22 UTC+0000
0x7d9d9450         UDPv4    0.0.0.0:63416                  *:*                                    3676     chrome.exe     2019-09-30 06:17:28 UTC+0000
0x7d097df8         TCPv4    192.168.243.132:49206          216.239.32.116:443     ESTABLISHED     3676     chrome.exe
0x7d241008         TCPv4    192.168.243.132:49160          172.217.25.195:443     CLOSED          3676     chrome.exe
0x7d3ae910         TCPv4    192.168.243.132:49204          172.217.26.35:443      ESTABLISHED     3676     chrome.exe
0x7d3ba1f0         TCPv4    192.168.243.132:49172          172.217.161.46:443     CLOSED          3676     chrome.exe
0x7d3de4f0         TCPv4    192.168.243.132:49195          172.217.31.170:443     ESTABLISHED     3676     chrome.exe
0x7d3f0520         TCPv4    192.168.243.132:49170          172.217.26.35:443      CLOSED          3676     chrome.exe
0x7d6448e0         TCPv4    192.168.243.132:49162          216.58.197.131:443     CLOSED          3676     chrome.exe
0x7d76f468         TCPv4    192.168.243.132:49207          172.217.25.195:443     ESTABLISHED     3676     chrome.exe
0x7d854240         TCPv4    192.168.243.132:49196          172.217.26.14:443      ESTABLISHED     3676     chrome.exe
```

Fig 3.2. Netscan results

Based on the above information, it was confirmed that the attacker used the Chrome browser to leak internal confidential data, and the filescan was processed by filtering the history string to extract the Chrome browser history file.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 filescan | findstr -i History
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 filescan | findstr -i History
Volatility Foundation Volatility Framework 2.6.1
0x000000007d803390      8      0 R--rwd ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Microsoft₩Windows₩History₩desktop.ini
0x000000007d82fbe0     10      1 RW-rw- ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Google₩Chrome₩User Data₩Default₩History
0x000000007da165f8      9      1 RW-rw- ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Microsoft₩Windows₩History₩History.IE5₩index.dat
0x000000007efe9208     17      1 RW-rw- ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Google₩Chrome₩User Data₩Default₩History-journal
0x000000007fe2f6a0      9      1 RW-rw- ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Microsoft₩Windows₩History₩History.IE5₩MSHist012019093020191001₩index.dat
```

Fig 3.3. Filescan | findstr -i History results

Extract the history file of the chrome browser found through the filescan to the actual data file, using the dumpfiles command.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q
0x000000007d82fbe0 -D ./
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007d82
fbe0 -D ./
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject 0x7d82fbe0   None   ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Google₩Chrome₩User Data₩Defa
ult₩History
SharedCacheMap 0x7d82fbe0   None   ₩Device₩HarddiskVolume1₩Users₩volatility₩AppData₩Local₩Google₩Chrome₩User Data₩Default
₩History
```

Fig 3.4. Dumpfiles History results

The data file extracted through Dumpfiles is file.None.0x86d3a850.dat as shown below.

Fig 3.5. Extracted History files

Original File	History
Recovered File	file.None.0x86d3a850.dat
File Size	116KB (118,784 bytes)
SHA1	936B5039A43E6F65EF14FF36788D3B659703ADC4

When analyzing the Chrome History data file in detail using HxD, it is possible to find [Confidential]_2018_Contest.pdf and [Secret]Document.pdf stored on your PC. In addition, doing a closer look at the log, a drive link and link to the drive storage folder name “hackers_vol” were found.

At this time, we carefully looked at the [Confidential]2018_Contest.pdf] and [Secret]Document.pdf files stored on your PC remaining in the Chrome History. Considering the file, the Google Drive folder hackers_vol and the above Google Drive link comprehensively, we can assume that an attacker used a Google drive to leak internal confidential data.

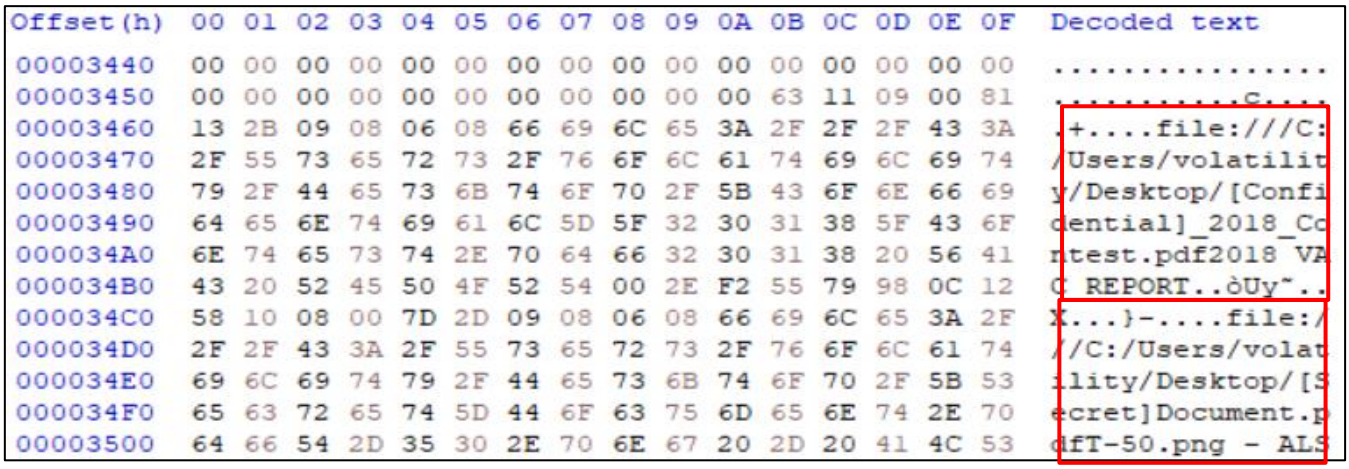
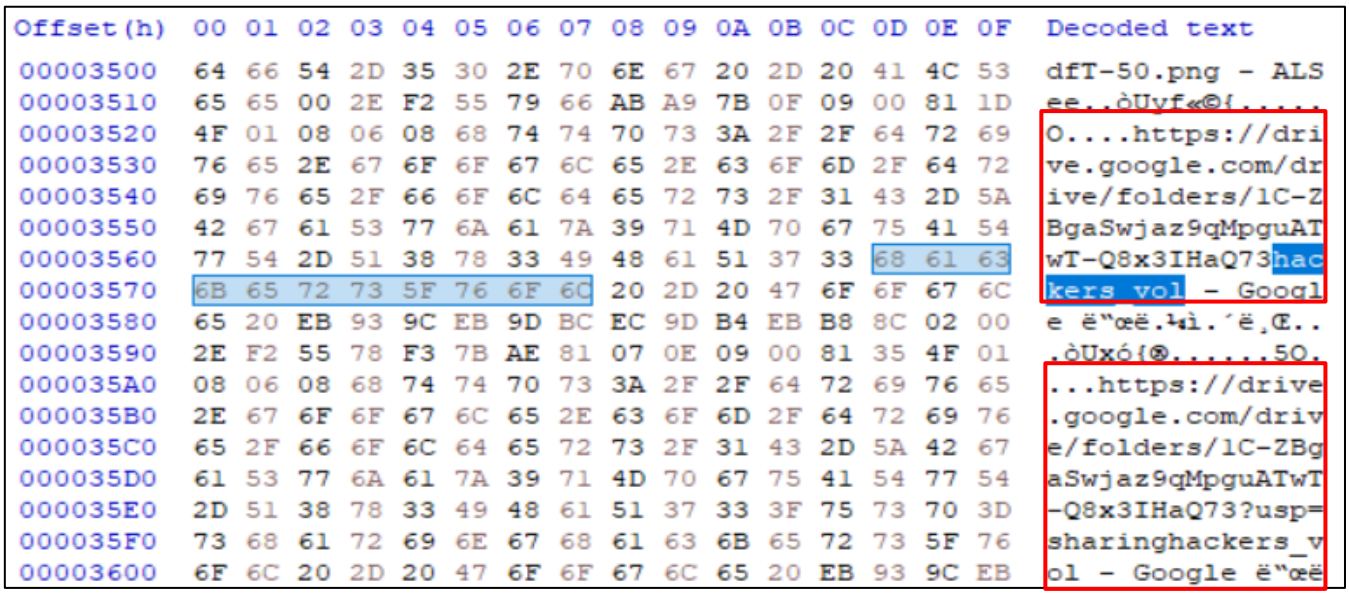


Fig 3.6. Uploaded Files to google drive



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00003500   64 66 54 2D 35 30 2E 70 6E 67 20 2D 20 41 4C 53  dfT-50.png - ALS
00003510   65 65 00 2E F2 55 79 66 AB A9 7B 0F 09 00 81 1D  ee..òUyf«©{.....
00003520   4F 01 08 06 08 68 74 74 70 73 3A 2F 2F 64 72 69  O....https://dri
00003530   76 65 2E 67 6F 6F 67 6C 65 2E 63 6F 6D 2F 64 72  ve.google.com/dr
00003540   69 76 65 2F 66 6F 6C 64 65 72 73 2F 31 43 2D 5A  ive/folders/1C-Z
00003550   42 67 61 53 77 6A 61 7A 39 71 4D 70 67 75 41 54  BgaSwjaz9qMpguAT
00003560   77 54 2D 51 38 78 33 49 48 61 51 37 33 68 61 63  wT-Q8x3IHaQ73hac
00003570   6B 65 72 73 5F 76 6F 6C 20 2D 20 47 6F 6F 67 6C  kers_vol - Googl
00003580   65 20 EB 93 9C EB 9D BC EC 9D B4 EB B8 8C 02 00  e ë"œë.¼ì.´ë¸Œ..
00003590   2E F2 55 78 F3 7B AE 81 07 0E 09 00 81 35 4F 01  .òUxó{®.....5O.
000035A0   08 06 08 68 74 74 70 73 3A 2F 2F 64 72 69 76 65  ...https://drive
000035B0   2E 67 6F 6F 67 6C 65 2E 63 6F 6D 2F 64 72 69 76  .google.com/driv
000035C0   65 2F 66 6F 6C 64 65 72 73 2F 31 43 2D 5A 42 67  e/folders/1C-ZBg
000035D0   61 53 77 6A 61 7A 39 71 4D 70 67 75 41 54 77 54  aSwjaz9qMpguATwT
000035E0   2D 51 38 78 33 49 48 61 51 37 33 3F 75 73 70 3D  -Q8x3IHaQ73?usp=
000035F0   73 68 61 72 69 6E 67 68 61 63 6B 65 72 73 5F 76  sharinghackers_v
00003600   6F 6C 20 2D 20 47 6F 6F 67 6C 65 20 EB 93 9C EB  ol - Google ë"œë
```

Fig 3.7. Link of google drive and hackers_vol folder

We need to know the identity of the chrome process to obtain the history of the chrome browser used for Google upload on that PC. We extracted the chrome PID to find out the process regarding the browser used for leakage.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 pslist | grep chrome
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 pslist | grep chrome
Volatility Foundation Volatility Framework 2.6.1
0x87405c98 chrome.exe            3512   1872     28      853      1      0 2019-09-30 06:13:19 UTC+0000
0x86eb0528 chrome.exe            3528   3512      8       82      1      0 2019-09-30 06:13:19 UTC+0000
0x86fe8c90 chrome.exe            3560   3512      2       54      1      0 2019-09-30 06:13:19 UTC+0000
0x865ecab0 chrome.exe            3668   3512      9      225      1      0 2019-09-30 06:13:19 UTC+0000
0x861227c0 chrome.exe            3676   3512     18      376      1      0 2019-09-30 06:13:19 UTC+0000
0x875f2c80 chrome.exe            1848   3512     15      371      1      0 2019-09-30 06:15:29 UTC+0000
0x849c2980 chrome.exe            1884   3512     12      182      1      0 2019-09-30 06:15:32 UTC+0000
0x84a3eb30 chrome.exe            2664   3512     12      168      1      0 2019-09-30 06:17:34 UTC+0000
```

Fig 3.8. Pslist results

The PID of the chrome through the above procedure are 3512, 3528, 3560, 3668, 3676, 1848, 1884, 2664. For the chrome process, search for the byte sequence and Unicode string contained in the user and kernel mode memory areas based on the yara rule specified using the yarascan plugin. To find traces uploaded by an attacker using a Google drive, specify the search rule "drive.google.com" and analyze it.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 yarascan -p
3512,3528,3560,3668,3676,1848,1884,2664 -Y "drive.google.com" > “drive_yara.txt"
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 yarascan -p 3512,3528,3560,
3668,3676,1848,1884,2664 -Y "drive.google.com" > "drive_yara.txt"
Volatility Foundation Volatility Framework 2.6.1
```

Fig 3.9. Yarascan results

Because there is a large amount of output, the search results were stored separately in the same directory in "drive_yara.txt".

Chrome PID	3512,3528,3560,3668,3676,1848,1884,2664
Yarascan rule	drive.google.com
Output File	drive_yara.txt
Size	3.11MB (3,271,242 byte)
SHA1	EBD6F52ADC9F9A9290E0B1427E17F10A35DA09D5

First, within the drive_yara.txt file saved separately, the contents of the file are retrieved by specifying "Secret" as the keyword, based on the analysis results above.

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>grep "Secret" "drive_yara.txt"
0x098497a7  88 5b 53 65 63 72 65 74 5d 44 6f 63 75 6d 65 6e   .[Secret]Documen
```

Fig 3.10. Grep “Secret” results

In addition, search the contents of the file by specifying the Google drive folder name "hackers_vol" as the keyword that was found through the analysis of the Chrome History.

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>grep "hackers_vol" drive_yara.txt
0x02b6356d  68 61 63 6b 65 72 73 5f 76 6f 6c 20 2d 20 47 6f   hackers_vol.-.Go
0x09cec515  68 61 63 6b 65 72 73 5f 76 6f 6c 20 2d 20 47 6f   hackers_vol.-.Go
```

Fig 3.11. Grep “hackers_vol” results

As shown above, the contents of the memory search based on drive.google.com include a Secret file and the name of the attacker's Google drive folder “hacker_vol” are stored directly. Based on this, the analysis is started more specifically using HxD.

We can find the leakage file [Secret]Document.pdf as shown below, in addition, the contents of the file uploaded to the google drive are stored with the byte order contained in the memory area.

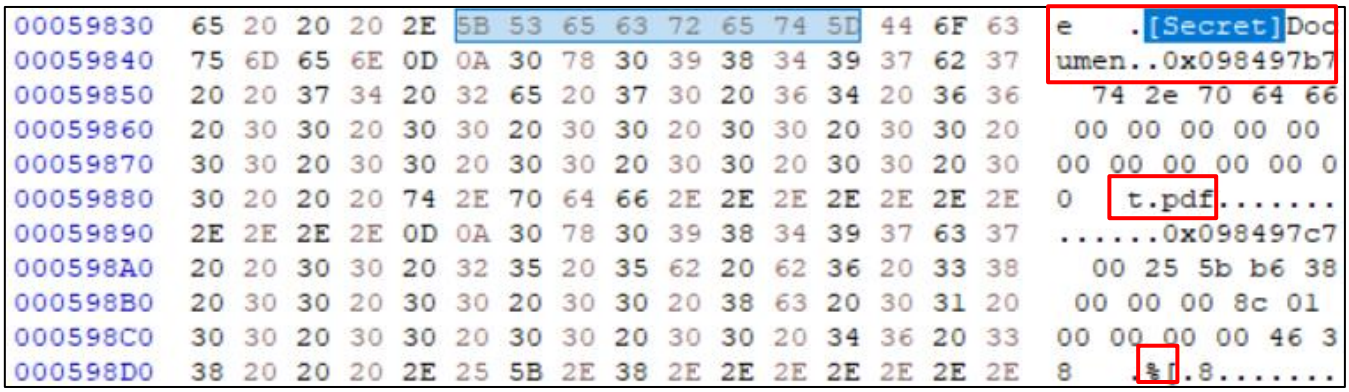
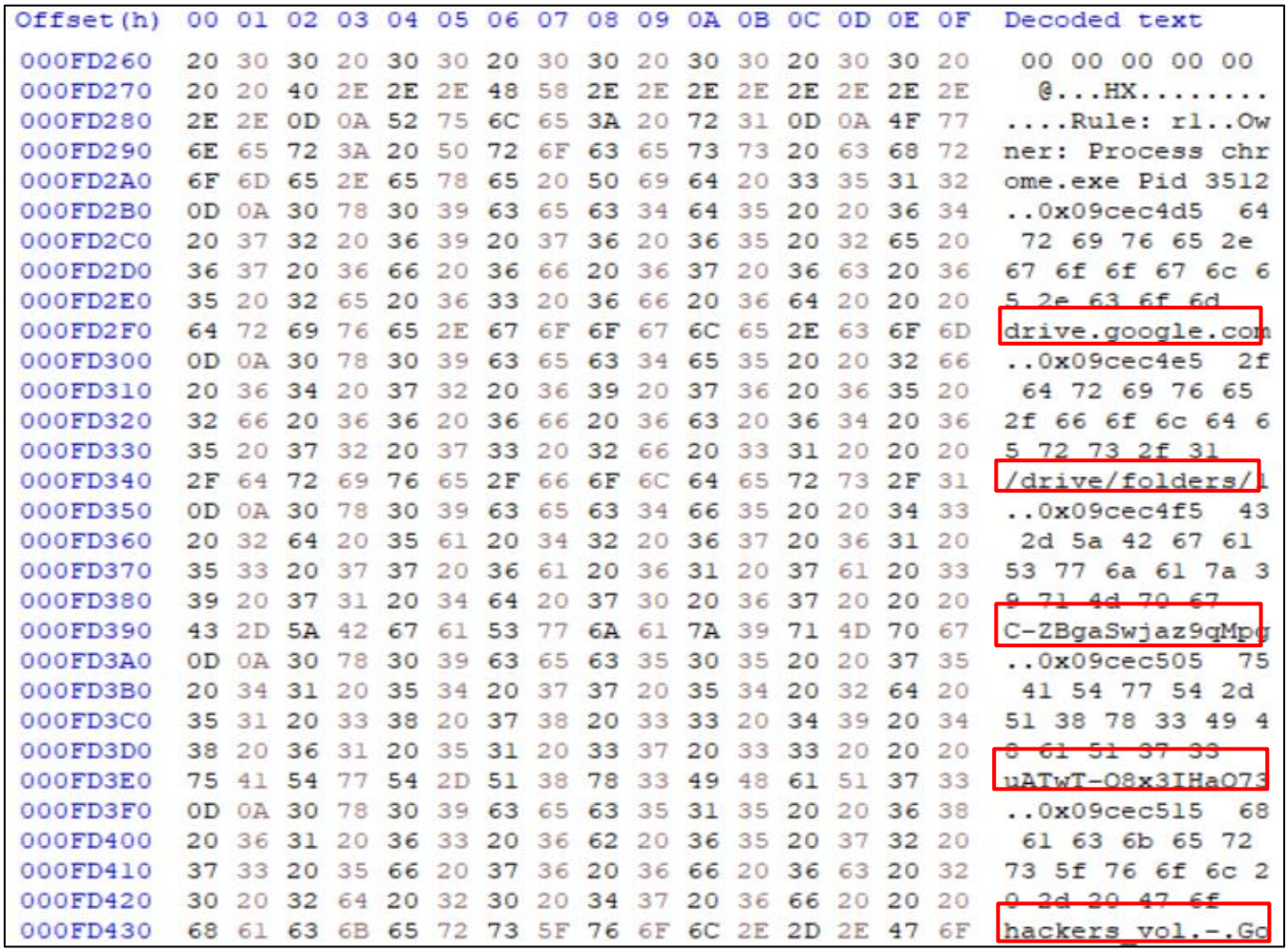


Fig 3.12. YaraScan Analysis

Moreover, we can check again https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3lHaQ73?usp=sharing is same as the attack’s google drive folder (i.e. “hackers_vol”)



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000FD260	20	30	30	20	30	30	20	30	30	20	30	30	20	30	30	20	00 00 00 00 00
000FD270	20	20	40	2E	2E	2E	48	58	2E	2E	2E	2E	2E	2E	2E	2E	@...HX.....
000FD280	2E	2E	0D	0A	52	75	6C	65	3A	20	72	31	0D	0A	4F	77	....Rule: rl..Ow
000FD290	6E	65	72	3A	20	50	72	6F	63	65	73	73	20	63	68	72	ner: Process chr
000FD2A0	6F	6D	65	2E	65	78	65	20	50	69	64	20	33	35	31	32	ome.exe Pid 3512
000FD2B0	0D	0A	30	78	30	39	63	65	63	34	64	35	20	20	36	34	..0x09cec4d5 64
000FD2C0	20	37	32	20	36	39	20	37	36	20	36	35	20	32	65	20	72 69 76 65 2e
000FD2D0	36	37	20	36	66	20	36	66	20	36	37	20	36	63	20	36	67 6f 6f 67 6c 6
000FD2E0	35	20	32	65	20	36	33	20	36	66	20	36	64	20	20	20	5 2e 63 6f 6d
000FD2F0	64	72	69	76	65	2E	67	6F	6F	67	6C	65	2E	63	6F	6D	drive.google.com
000FD300	0D	0A	30	78	30	39	63	65	63	34	65	35	20	20	32	66	..0x09cec4e5 2f
000FD310	20	36	34	20	37	32	20	36	39	20	37	36	20	36	35	20	64 72 69 76 65
000FD320	32	66	20	36	36	20	36	66	20	36	63	20	36	34	20	36	2f 66 6f 6c 64 6
000FD330	35	20	37	32	20	37	33	20	32	66	20	33	31	20	20	20	5 72 73 2f 31
000FD340	2F	64	72	69	76	65	2F	66	6F	6C	64	65	72	73	2F	31	/drive/folders/1
000FD350	0D	0A	30	78	30	39	63	65	63	34	66	35	20	20	34	33	..0x09cec4f5 43
000FD360	20	32	64	20	35	61	20	34	32	20	36	37	20	36	31	20	2d 5a 42 67 61
000FD370	35	33	20	37	37	20	36	61	20	36	31	20	37	61	20	33	53 77 6a 61 7a 3
000FD380	39	20	37	31	20	34	64	20	37	30	20	36	37	20	20	20	9 71 4d 70 67
000FD390	43	2D	5A	42	67	61	53	77	6A	61	7A	39	71	4D	70	67	C-ZBgaSwjaz9qMpg
000FD3A0	0D	0A	30	78	30	39	63	65	63	35	30	35	20	20	37	35	..0x09cec505 75
000FD3B0	20	34	31	20	35	34	20	37	37	20	35	34	20	32	64	20	41 54 77 54 2d
000FD3C0	35	31	20	33	38	20	37	38	20	33	33	20	34	39	20	34	51 38 78 33 49 4
000FD3D0	38	20	36	31	20	35	31	20	33	37	20	33	33	20	20	20	8 61 51 37 33
000FD3E0	75	41	54	77	54	2D	51	38	78	33	49	48	61	51	37	33	uATwT-Q8x3IHaQ73
000FD3F0	0D	0A	30	78	30	39	63	65	63	35	31	35	20	20	36	38	..0x09cec515 68
000FD400	20	36	31	20	36	33	20	36	62	20	36	35	20	37	32	20	61 63 6b 65 72
000FD410	37	33	20	35	66	20	37	36	20	36	66	20	36	63	20	32	73 5f 76 6f 6c 2
000FD420	30	20	32	64	20	32	30	20	34	37	20	36	66	20	20	20	0 2d 20 47 6f
000FD430	68	61	63	6B	65	72	73	5F	76	6F	6C	2E	2D	2E	47	6F	hackers_vol.-.Go

Fig 3.13. YaraScan Analysis2

To sum up, attack's google drive folder URL is https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3IHaQ73 and folder name is "hackers_vol"

Accessing the given URL, we can find confidential data (i.e. pdf).

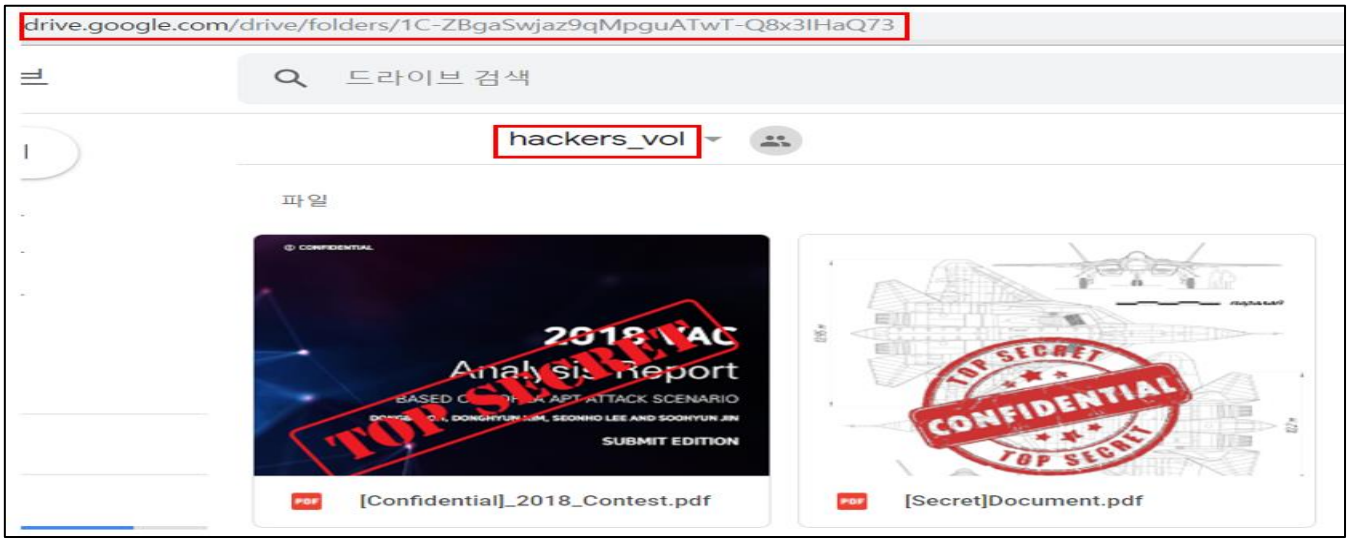


Fig 3.14. Google Drive link

Based on this, filescan is performed on the pdf file to confirm that the file exists in memory.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 filescan | findstr -i pdf
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 filescan | findstr -i pdf
Volatility Foundation Volatility Framework 2.6.1
0x000000007e0dbf80      4      0 R--rw- ₩Device₩HarddiskVolume1₩Users₩volatility₩Desktop₩[Confidential]_2018_Contest.pdf
0x000000007e19fd70      8      0 R--rw- ₩Device₩HarddiskVolume1₩Users₩volatility₩Desktop₩[Secret]Document.pdf
```

Fig 3.15. Filescan Results

Extract the pdf file found through the Filescan to the actual data file using the dumpfiles command.

```
> vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q
0x000000007e19fd70 -D ./
```

```
D:₩BOB₩FDBG₩Volatility₩20190930₩volatility>vol.py -f drive_mem.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007e19
fd70 -D ./
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject 0x7e19fd70   None   ₩Device₩HarddiskVolume1₩Users₩volatility₩Desktop₩[Secret]Document.pdf
```

Fig 3.16. Dumpfiles pdf file

Fig 3.17. Extracted [Secret]Document

Original File	[Secret]Document.pdf
Recovered File	File.None.0x85260bc8.dat
SHA1	13DFB3B5CD56BCEE10D183DA0725F9A5D81DE094
Size	772B

Compared to the original file [Secret]Document.pdf, unlike the original file, it has been padded to 00 since C05E2 address and its padding size is 2KB.

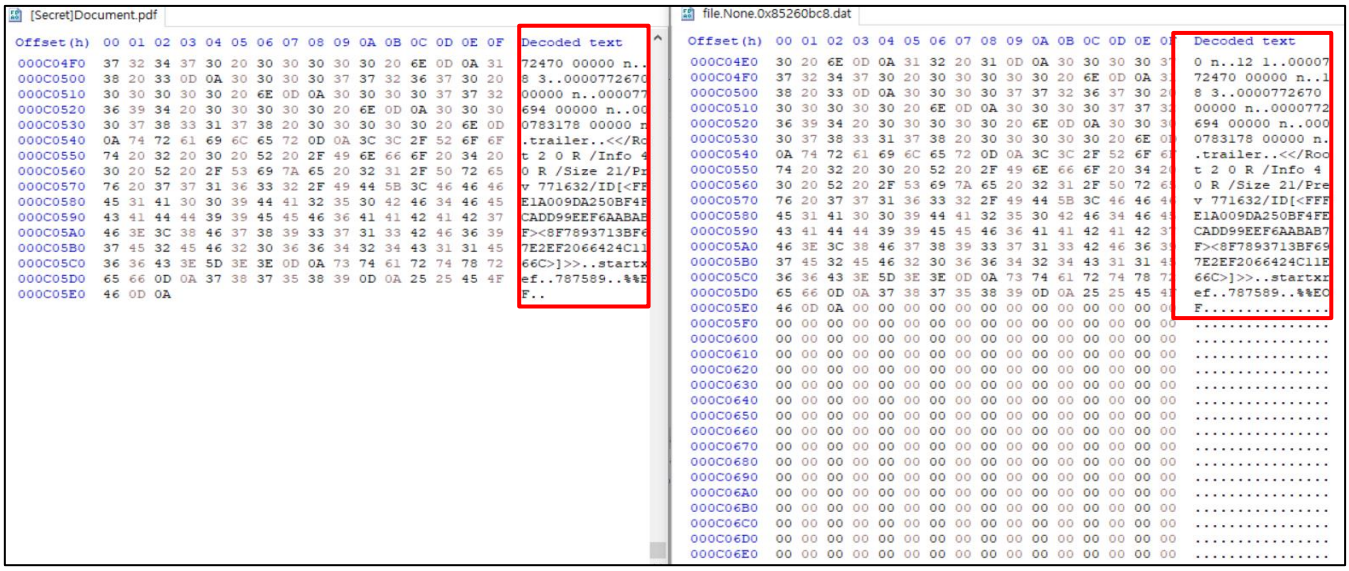


Fig 3.18. Compare Original/Recovered Files

Removing the parts that are padded to the 00 shows that it has the same hash value as the original file [Secret]Document.pdf.

Original File	[Secret]Document.pdf
Recovered File	File.None.0x85260bc8.dat
SHA1 (After removing padding)	EFAB8850252DC373F8870A55467EC5DDB4896C7F
Size	770KB

After changing the file extension to .pdf based on file signature, you can verify that the file is the same content as the file uploaded to the google drive.

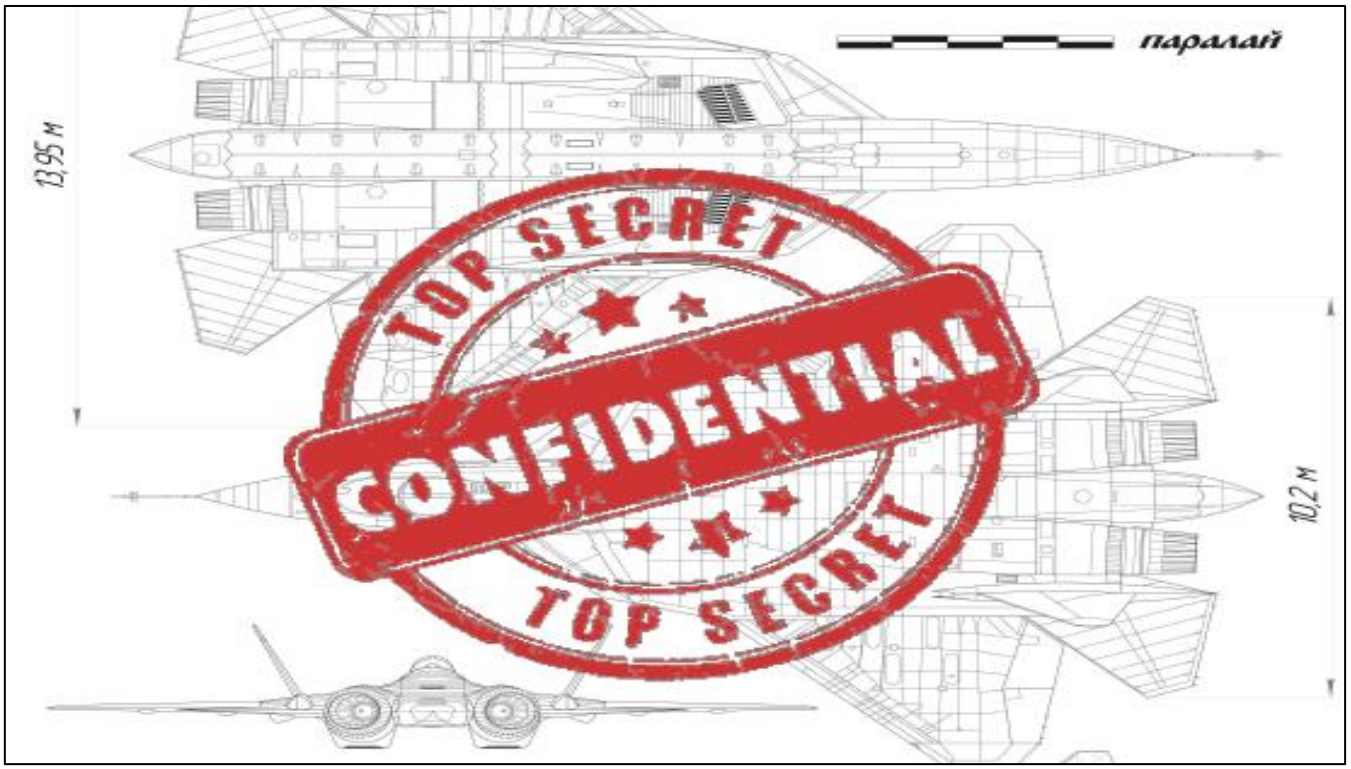


Fig 3.19. [Secret]Document.pdf

5. Conclusion

5.1. Result

5.1.1. Data Leakage by RAT

Information obtained through detailed analysis shows that the leakage process on User PC is as follows.

① Through the Command, the attacker identified the path and contents of important documents ("[Secret]Document.pdf ") existing on the User PC.

② An attacker uploaded and leaked important documents ("[Secret]Document.pdf ") to a remote server of "192.168.0.15" communicating through a malicious file he had built.

5.1.2. Data Leakage through FTP

Information obtained through detailed analysis shows that the leakage process on User PC is as follows.

① Filezilla Software Install (Version. 3.44.2.0)

② Filezilla Software – Logging Setting. (PATH = %APPDATA%₩Filezilla₩Filezilla.log)

③ Modify Hosts File (192.168.161.1 ftp.hackers.vol)

④ Hacker Storage(domain = ftp.hackers.vol) is running on Windows, Anonymous FTP is also supported. (port 21)

The file [Secret]Document.pdf was leaked from the user PC (192.168.161.133) to the FTP server (192.168.161.1) presumed to be the attacker's storage space via the "Filezilla" software on the 2019-09-30T10:32:27 (UTC+09:00).

The actual file information of the leakage is shown in the table below.

File Name	[Secret]Document.pdf
File Size	769KB (787,939 Bytes)
SHA1	EFAB8850252DC373F8870A55467EC5DDB4896C7F

5.1.3. Data Leakage via Web

Information obtained through detailed analysis shows that the leakage process on User PC is as follows.

① The user logged in with a Google account in the Chrome browser.
② An attacker leaked confidential user data to his or her Google drive link using a chrome browser while logged in.
③ The following table provides information on Google drive links, accounts and leakage files used in the analysis.
④ Google drive upload timeline is as follows (we used SQLite Browser).

Google Drive	Hackers_vol (https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3lHaQ73?usp=sharing)
Victim Account	victim.vol (ID) / vol.victim123 (PW)
Leaked File #1	[Secret]Document.pdf SHA1: EFAB8850252DC373F8870A55467EC5DDB4896C7F
Leaked File #2	[Confidential]_2018_Contest.pdf SHA1: 3FBA3BACF95A9AB341033D644E780D03E4BE1889

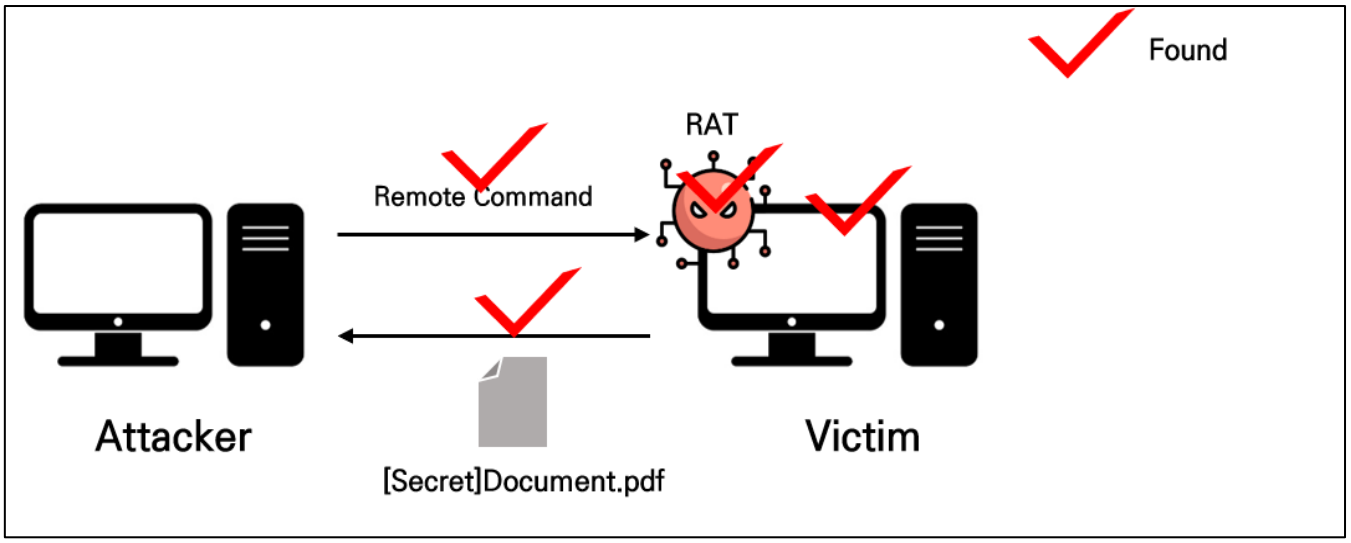
https://drive.google.com/drive/my-drive	내 드라이브 - Google 드라이브	13214297733094811
file:///C:/Users/volatility/Desktop/[Secret]Document.pdf	T-50.png - ALSee	13214297851603881
file:///C:/Users/volatility/Desktop/[Confidential]_2018_Contest.pdf	2018 VAC REPORT	13214297854839826
https://drive.google.com/drive/?utm_source=ko	내 드라이브 - Google 드라이브	13214297732429565
https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3lHaQ73?usp=sharing	hackers_vol - Google 드라이브	13214297843294569
https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3lHaQ73	hackers_vol - Google 드라이브	13214297844054958

URL	Title	time in Google Chrome Value	Date & Time (UTC+09:00)
https://drive.google.com/drive/my-drive	Google drive	13214297733094811	2019-09-30T15:15:32
file:///C:/Users/volatility/Desktop/[Secret]Document.pdf	T-50.png - ALSee	13214297851603881	2019-09-30T15:17:31
file:///C:/Users/volatility/Desktop/[Confidential]_2018_Contest.pdf	2018 VAC REPORT	13214297854839826	2019-09-30T15:17:34
https://drive.google.com/drive/?utm_source=ko	Google drive	13214297732429565	2019-09-30T15:15:32
https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3lHaQ73?usp=sharing	hackers_vol	13214297843294569	2019-09-30T15:17:23
https://drive.google.com/drive/folders/1C-ZBgaSwjaz9qMpguATwT-Q8x3lHaQ73	hackers_vol	13214297844054958	2019-09-30T15:17:24

5.2. Overall Diagram

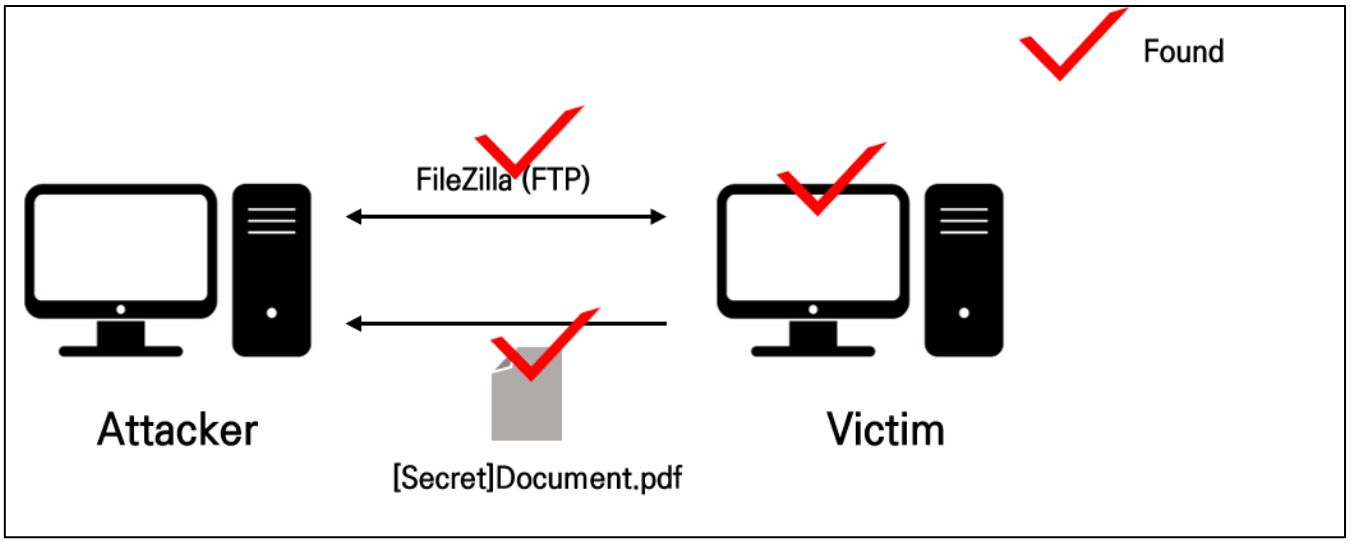
5.2.1. Scenario #1 (RAT)

Victim's activity of transferring protected data, memory of the RAT program, remote command are detected.



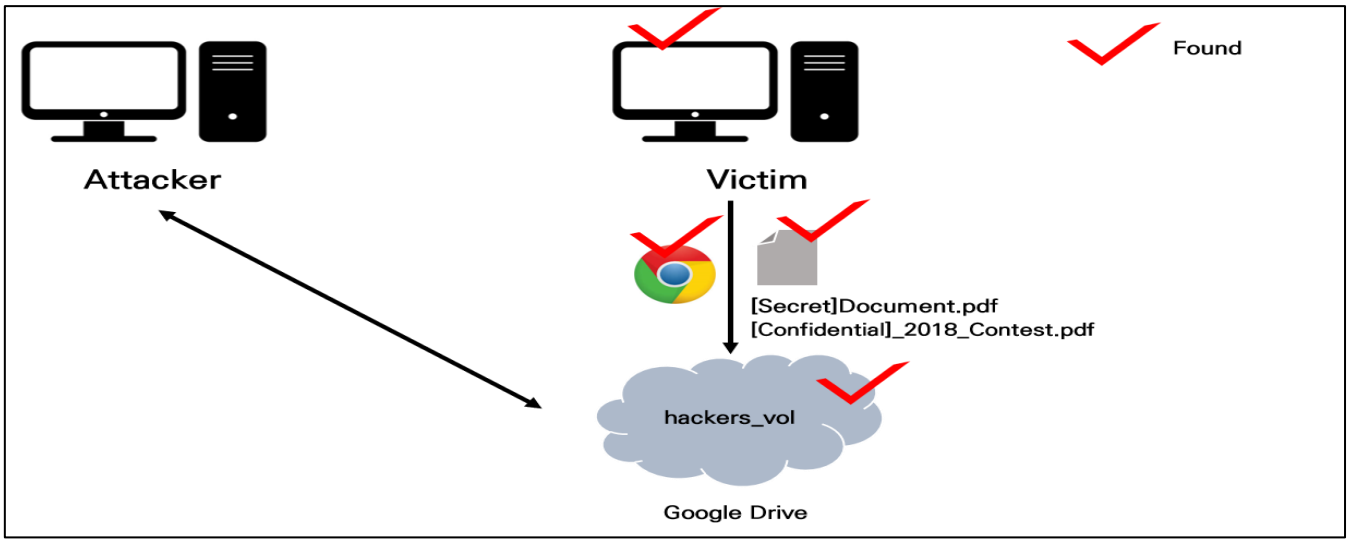
5.2.2. Scenario #2 (FTP)

Victim's activity of transferring protected data, using FileZilla(FTP) are detected and data is recovered.



5.2.3. Scenario #3 (Web)

Victim's activity of transferring protected data, using google chrome, are detected and data is recovered.



6. Why this report should win the contest

6.1. Data leakage is always a hot topic.

As explained in 1.2, Data leakage is always a hot topic throughout the world. Even a nation fails to manage data protection. Various groups are spending their budgets to buy data leakage protection (DLP) solutions, and plenty of IT companies (security) develop DLPs. However, no one can absolutely (100%) protect data leakage and various data leakage happens. Thus, there are also needs to properly respond to data leakage incident. We believe this report can help memory analyst to analyze various data leakage case in the real world.

6.2. We satisfied the standard to be a winner

Volatility Foundation suggested 7 criteria for deciding a winner. And we claim we satisfied most of the criteria to be the winner of volatility analysis contest 2019.

The Volatility Project core developers will decide the winners based on the following criteria:

1. Accuracy of the analysis
   -> It can be proven by reviewing our analysis report, we did our best to build a logical correlation between each analysis step and conclusion.

2. Completeness of the analysis
   -> We succeeded to find artifacts generated within data leakage and recovered the data that is being leaked.

3. Clarity of the documentation
   -> Documentation is clear enough for everyone to reproduce the same result by following our report

4. Novelty of the analysis and malware/framework selected
   -> Rather than selecting specific malware and re-construct scenario, we tested general data leakage cases using Volatility, and the result can be applied analyzing data leakage cases in the real world. The idea is novel.

5. Sophistication of the malware/framework selected

-> We chose various data leakage and sophisticated tools used in the real world.

6. Submission date
   -> We kept submission date!

7. NOTE: Submissions based on malware/frameworks that have been publicly documented using non-memory analysis techniques are not discouraged, but contestants are encouraged to focus on the analysis aspects that are unique to memory analysis.
   -> Most of our analysis focused on memory analysis, which is done by Volatility Framework.

2019.10.01 / Team HSLFL

7. Reference

[1] https://www.acf.hhs.gov/sites/default/files/cb/im1504.pdf

[2] https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

[3] https://edition.cnn.com/2019/09/17/americas/ecuador-data-leak-intl-hnk-scli/index.html

[4] https://searchsecurity.techtarget.com/definition/RAT-remote-access-Trojan

[5] https://github.com/quasar/QuasarRAT

[6] https://windowsreport.com/ftp-client-windows-10/