Chapter 10
File Signature Searching Forensics

Learning Objectives
The objectives of this chapter are to:

- Understand concept of file signature
- Understand procedures and forensic technique of file signature searching
- Know how to use open-source tools for file signature searching forensics

As a forensics technique that searches for known files or documents, file signature searching technique is widely used to find evidence of the theft of confidential company files (documents) or detect the existence of malware by comparing unknown software (program) under investigation with a repository of known instances. In this chapter, we will study procedures and forensic technique of file signature searching. Specifically, you will learn how to generate hashes from a file, to create index on the hash database (Ignored databased and Alert databased), and to search for file with specific hash value. Also, you will become familiar with open-source tools for file signature searching forensics, including hfind, md5sum, and sha1sum.

10.1 Introduction

File signature search is a common technique used in forensic analysis to identify or verify existence of a known file in a disk used by a suspect. In doing so, we assume that we have a large number of files in custody, either good (trustworthy ones) or bad (inappropriate or harmful ones). Then, databases are created to include the hash values of all the known files, also known as hash databases. Afterwards, the forensic analyst would use the hash function (e.g. MD5 hash function) to generate a hash

X. Lin, *Introductory Computer Forensics*,
https://doi.org/10.1007/978-3-030-00581-8_10

value (or file signature) of the file and compare it to a hash database. A good hash database to reference is NIST National Software Reference Library (NSRL) [1], which contains hashes of commercial software packages and files found in operating systems and software distributions. These files are known to be good since they came from trusted sources and are typically found on authorized systems. The analyst would calculate the MD5 hash of the given file and search the md5 hash databases. If there is a hit, it means the given file is known to investigator.

Note that in computing, file signature also refers to data used to identify the file type of a file, particularly file headers (the beginning bytes of the file content) or file footer (the ending bytes of the file content) [2]. For example, a file beginning with "25 50 44 46" (in hex) means a pdf file, and the magic number "25 50 44 46" (or file header here) is known as file signature of pdf files. It can be distinguished from the concept of file signature used in the book, which refers to a magic number (or the hash value of the file), uniquely identifying the file itself instead of its file type.

In the industry, the process of comparing entries against approval registry is called "whitelisting." "Blacklisting" is the opposite of whitelisting, where the practices is to identify entries from untrusted sources. In this chapter, we dub the whitelist of hashes **Ignored database** and blacklist of hashes **Alert database**, respectfully. Instances that would flag the file suspicious are malicious software (Malware) or inappropriate content.

A hash value or simply the hash, also known as message digest or simply the digest, is a usually shorter fixed-length value that is the output of a cryptographic hash function. For example, the MD5 algorithm [3], a widely used hash function, generates hash values which are only 128-bit long. It can be used to detect data modifications and tampering or uniquely identify data. A cryptographically secure hash function $h(x)$ must satisfy the following properties:

1. It takes arbitrary length of data and produces a fixed-length output;
2. Given a message m of arbitrary length, there is a computationally efficient way to calculate $h(m)$ but it's infeasible to go the other way. It is also known as one-way or pre-image resistance property;
3. Given a hash value y, it is computationally infeasible to find a message m with $h(m) = y$. It is also known as second pre-image resistant or weak collision resistance property;
4. It is computationally infeasible to find any pair of messages m_1 and m_2 such that $h(m_1) = h(m_2)$. It is also known as collision resistant or strong collision resistance property;

10.2 File Signature Search Process

It is very common for investigators to look for some suspicious file in suspected computer, so a straightforward method to look for a file is to go through and analyze each data bit and bit, but it would be too time consuming and inefficient. Therefore, the best practice is to automate the process by prebuilding a database of file

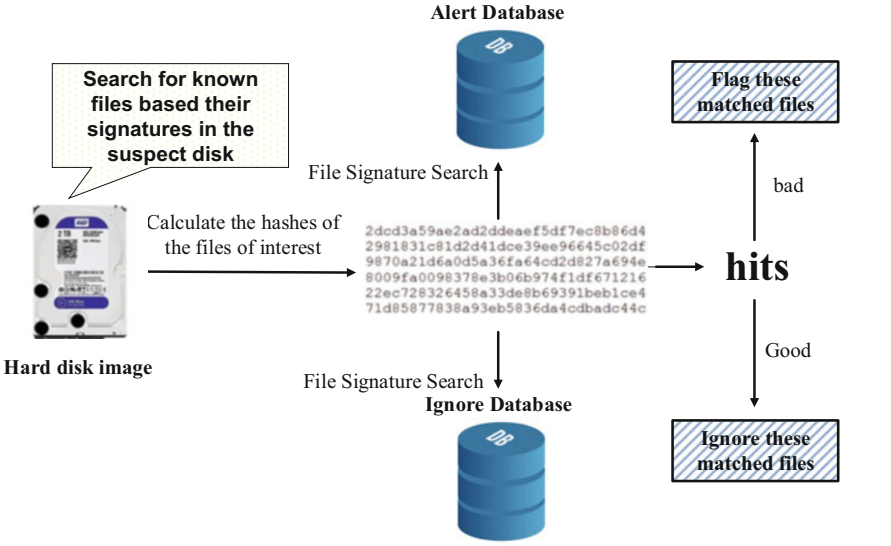

Fig. 10.1 File signature search process overview

signatures (hashes of files) and comparing them with the hash of the target file. This is also called File Signature Search. The Sleuth Kit (TSK) provides a tool called "hfind", which can be used to perform file signature search and will be detailed in the next section.

The file signature search process typically follows Fig. 10.1 workflow, where an analyst acquires a questionable amount of files. This could be hundreds of thousands of files, and going through manually in order to deduce whether it's good or bad would be impossible. Instead, we can simply calculate the hash values of these files being investigated. Further, we assume that we have already created two hash databases, one containing the hash values of all the known good files, known as **Ignore Database**, and another including the hash values of all the known bad files, known as **Alert Database**. Then, we search both **Ignore Database** and **Alert Database** for the hash values we calculated for the files being investigated. If the search finds a hit in the **Alert Database**, there is a known bad file among the files under investigation. In other words, more attentions must be paid and further investigation is required. Nevertheless, if there is a hit in the **Ignore Database**, it means a good file which has been found so we simply ignore it. As a result, we can quickly filter many known files out of a large amount number of files being investigated. Benefit of using hash is it is deliberately difficult to reconstruct two different messages having the same hash (as shown in Fig. 10.2 when a character is changed in the input) but is very easy to compute a block of data at any size into a fixed length output or hash; thus, enable fast database lookup by detecting duplicated record. Another big advantage is that we don't have to sift through millions of files found on the hard disk used by a suspect. Instead, we can quickly narrow down to a

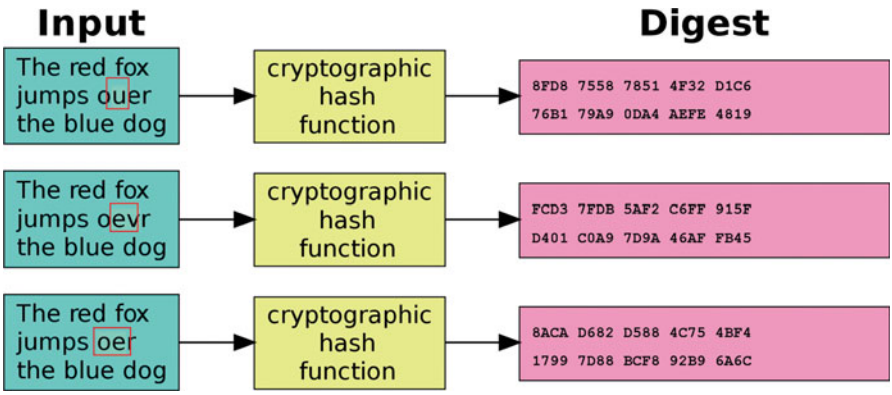

Fig. 10.2 Hash generated is never the same

small number of files which are still unknown to us. This is because there are many system files and the ones resulted from software installations, and these files can be quickly identified and then excluded or ignored. By doing so, our precious time and effort can be put to investigate these unknown files.

There are two common hash functions used to generate hashes (or signatures) of files in forensic investigation, which are md5 and sha-1. The investigator would create two hash databases (common source is from NIST National Software Reference Library), one with repository of known software, file profile, and file signature dubbed "**Ignore Database**"; and a second one with repository of known bad software, file profile, and file signature dubbed "**Alert Database**". The analyst would then create a hash value for the suspicious file being investigated, and traverse through the hash databases. If there is a hit from **Ignore database**, then analyst known file is good. If there is a hit from **Alert database**, then analyst know file is bad and further investigation and analysis is required on the suspect's hard drive.

10.3 File Signature Search Using hfind

Hfind is a TSK tool that is used to look up hash values in a hash database using a binary search algorithm. This allows one to easily create a hash database and identify if a file is known or not. It can work with the NIST National Software Reference Library (NSRL) and the output of "md5sum", which is the utility come with most Linux distributions and used to compute and check MD5 hash values of a file.

Next, we explain how file signature search works, using open source tools, hfind and md5sum.

10.3.1 Create a Hash Database Using md5sum

On your Windows machine, start PuTTY and connect to your forensic workstation (virtual machine). Suppose that all the files under /usr/local/bin (not including files in the subfolders) are good, create a hash database using the following command.

```
md5sum /usr/local/bin/* > Ignore.db
```

where "Ignore.db" is the created hash database file to be considered "**Ignore Database**" in our example.

Note that md5sum may complain by saying an entity in specified folder is a directory and you can simply ignore these warnings.

Figure 10.3 shows part of the file from line to line, each line containing a 32-hex-character MD5 hash and its corresponding file of which the hash is calculated.

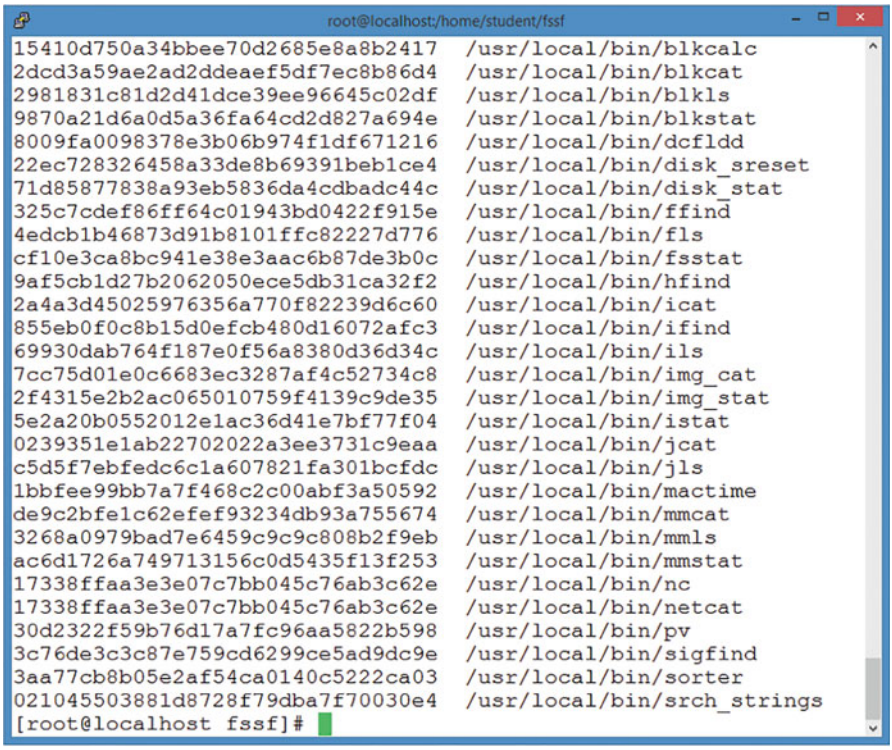

Fig. 10.3 Hash database

10.3.2 Create an MD5 Index File for Hash Database

Next we use a TSK tool, hfind, which uses a binary search algorithm to lookup hashes in a hash database, for example NIST NSRL, "Ignore.db" created above. Thus, an index must be created to do faster searching through the database, which is important if using large databases. The following command will create a MD5 index using the newly created MD5 hash database.

```
hfind -i md5sum Ignore.db
Index Created
```

An index file called "Ignore.db-md5.idx" will be created and can be found in the current folder.

Now, we can perform file signature search through the indexed database "Ignore.db".

10.3.3 Search Hash Database for a Given Hash Value

Suppose we have a file /home/student/abc.dat in custody and want to know whether it is good.

First, we need to calculate the hash value of the file/home/student/abc.dat using the following command

```
md5sum /home/student/abc.dat
2fb5de8146328ac2b49e3ffa9c0ce5a3 /home/student/abc.dat
```

where "2fb5de8146328ac2b49e3ffa9c0ce5a3" is the resulted MD5 hash value of the file/home/student/abc.dat. Recall that a MD5 hash value is 128 bits (or 32 hexadecimal digits) in length.

Then, we do file signature search through the database "Ignore.db" using the following command.

```
hfind Ignore.db 2fb5de8146328ac2b49e3ffa9c0ce5a3
2fb5de8146328ac2b49e3ffa9c0ce5a3 Hash Not Found
```

In the above example, we can conclude that the file/home/student/abc.dat is not a known good file. In other words, a further investigation is needed for the file. Or, you may see the following output from the command above.

```
hfind Ignore.db 9af5cb1d27b2062050ece5db31ca32f2
9af5cb1d27b2062050ece5db31ca32f2 /usr/local/bin/hfind
```

Now it means the file under investigation is a known good file, which is the TSK utility *hfind*.

Review Questions

1. How long in bits is a MD5 hash?
2. Does the MD5 hash function always generate a fixed length hash?
3. What is the "weak collision resistance" property of a hash function?
4. What does NSRL stand for?
5. Why not simply use grep to search for file signature from a list of hashes of all known files? Instead, before any file signature search can be performed, we have to do some preparation by creating md5sum hash databases and as well creating the index on the databases.
6. True or False (no need to explain)
   (a) Hash functions use secret keys.
   (b) Message integrity means that the data must arrive at the receiver exactly as sent.

10.4 Practice Exercise

The objective of this exercise is to practise file signature searching technique using hfind and md5sum.

10.4.1 Setting Up the Exercise Environment

- Login to your Forensics workstation
- Create a shell script provided in the appendix and run the shell script to generate test files for file hash databases, one set of files for **Ignore Database** and another set for **Alert Database**.

10.4.2 Exercises

Part A: Create Ignore Database and Alert Database
Create two md5sum hash databases, one for good files (or Ignore Database) and another for bad files (or Alert Database) by using the **md5sum** tool and as well create the index on the databases by using the **hfind** tool.

Q1. Write down the command used to create the md5sum hash database (or **Ignore Database**) named "*Ignore.db*".

_____

Q2. Writing down your command(s) issued to create the index on **Ignore Database**.

_____

Q3. Write down the command used to create the md5sum hash database (or **Alert Database**) named *"Alert.db"*.

_____

Q4. Writing down your command(s) issued to create the index on **Alert Database**.

_____

Part B: Searching for File Signatures

Search for the following file signatures using **hfind** and answer the following questions. Note that all the file signatures (or the hash values of the files) are in hexadecimal.

Q5. Is a file with a MD5 hash of "c0de7a481fddfa03b764fa4663dc6826" a good one (or contained in the Ignore Database)?

_____

Q6. Is a file with a MD5 hash of "574e321c1dfcb742849d6e2347003f41" a bad one (or contained in the Alert Database)?

_____

Q7. Finally, randomly choose one file you have created in the exercise, and use md5sum to calculate its MD5 hash value. Then, use hfind to search the resulted Md5 hash value against two hash databases (**Ignore Database** and **Alert Database**) you have created. Are you able to find a match to the md5 hash of your chosen file? _____ (Yes/No). Please provide details on your Yes or No answer.

Appendix A: Shell Script for Generating Files for File Hash Database

Usage

./fssf.sh numberOfFiles typeOfFile

where numberOfFiles means the total number of files to generate and typeOfFile is the type of files to generate, either "0" for good or "1" for bad.

It generates a list of either good or bad files for the experimental practice in this chapter. You must specify how many files you want to generate and what type of files you want, including good files for Ignored database and bad files for Alert database. The shell script can be found in the end.

For example,

./fssf.sh 100 0

It means the total 100 good files will be generated and saved into a subfolder named good (bad if specifying 1 as the type of file) of the current working folder.

```
#!/bin/bash
mynooffiles=$1
mytypeoffiles=$2
# Validate input arguments
# including number of arguments and the type of files to generate
if [ "$#" -lt 2 ]; then
        echo "usage: ./fssf.sh numberOfFiles typeOfFile"
        echo "where numberOfFiles means the total number of files to generate and typeOfFile is the type of files to generate, either 0 for
good or 1 for bad."
        echo "For example,"
        echo "./fssf.sh 100 0"
        echo "It means the total 100 good files will be generated and saved into a subfolder named good (bad if specifying 1 as the type
of file) of the current working folder."
        exit 1
elif [ "$2" -ne "0" ] && [ "$2" -ne "1" ]; then
        echo "usage: ./fssf.sh numberOfFiles typeOfFile"
        echo "where typeOfFile is the type of files to generate, either 0 for good or 1 for bad."
        exit 1
fi
# Create subfolder for the newly created fiels to be saved into
if [ "$2" -eq "0" ];then
        if [ -d "./good" ]; then
                # Will remove older folder first
                rm -rf good
        fi
        mkdir good
elif [ "$2" -eq "1" ];then
        if [ -d "./bad" ]; then
                # Will remove older folder first
                rm -rf bad
        fi
        mkdir bad
fi
x=1
while [ $x -le $mynooffiles ]
do
   echo "Welcome $x times"
        if [ "$2" -eq "0" ];then
                foo="./good/good_$x"
        else
                foo="./bad/bad_$x"
        fi
        cmd="dcfldd if=/dev/urandom of=$foo bs=512 count=2"
        echo $cmd
        eval $cmd
        x=$(( $x + 1 ))
done
```

References

1. NIST National Software Reference Library (NSRL). http://www.nsrl.nist.gov
2. File signature. https://en.wikipedia.org/wiki/File_
3. Bruce Schneier. Applied Cryptography. John Wiley & Sons, 1996