Chapter 3

Concepts of digital forensics

Digital forensics is a branch of forensic science concerned with the use of digital information (produced, stored and transmitted by computers) as source of evidence in investigations and legal proceedings. Digital Forensic Research Workshop has defined digital forensics as

> "The use of scientifically derived and proven methods toward the preservation, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations." [4]

This chapter introduces concepts of digital forensics and digital forensic analysis techniques described in the literature.

3.1 Investigative process

Investigative process of digital forensics can be divided into several stages. According to [4] and [71], there are four major stages: preservation, collection, examination, and analysis.

- *Preservation.* Preservation stage corresponds to "freezing the crime scene". It consists in stopping or preventing any activities that can damage digital information being collected. Preservation involves operations such as preventing people from using computers during collection, stopping ongoing deletion processes, and choosing the safest way to collect information.

- *Collection.* Collection stage consists in finding and collecting digital information that may be relevant to the investigation. Since digital information is stored in computers, collection of digital information means either collection of the equipment containing the information, or recording the information on some medium. Collection may involve removal of personal computers from the crime scene, copying or printing out contents of files from a server, recording of network traffic, and so on.

- *Examination.* Examination stage consists in an "in-depth systematic search of evidence" relating to the incident being investigated. The output of examination are data objects found in the collected information. They may include log files, data files containing specific phrases, timestamps, and so on.

- *Analysis.* The aim of analysis is to "draw conclusions based on evidence found".

Other works including [25], [19], and [72] proposed similar stages summarised in Table 3.1.

The aims of preservation and collection are twofold. First they aim to provide examination and analysis with as much relevant information as possible. Second they aim to ensure integrity of the collected information.

Preservation and collection are not discussed in this dissertation, because this research is primarily concerned with the analysis stage of the investigative process. In the rest of this dissertation it is simply assumed that all neces-

Stage	Analogue in [25]	Analogue in [19]	Analogue in [72]
preservation	preservation	preservation	preservation
collection	recognition, collection, documentation	collection	identification, retrieval
examination	classification, comparison, individualisation	formulating leads, focused searches	extraction, processing
analysis	reconstruction	temporal analysis	interpretation

Figure 3.1: Stages of investigative process

sary information has been collected, and that the integrity of the collected information has been preserved.

Interested readers can obtain more information about collection and preservation stages of investigative process from the following sources. The best practice guides, such as [3], [8], and [17], specify standard procedures and check lists of things to be done by investigators during preservation and collection stages. An approach to testing correctness of tools used during collection of information is described in [19]. A more formal approach is being developed by the U.S. National Institute for Standards and Technology (NIST) in [60]. Methods for detecting tampering after collection are described in [12] and [68]. They are based on checksumming and one-way hashing of collected information.

3.2 Examination and analysis techniques

This section describes techniques and tools used at examination and analysis stages. Techniques are grouped into subsections according to the type of question they answer.

3.2.1 Search techniques

This group of techniques searches collected information to answer the question whether objects of given type, such as hacking tools, or pictures of certain kind, are present in the collected information. According to the level of search automation, this dissertation classifies techniques into *manual browsing* and automated searches. Automated searches include *keyword search*, *regular expression search*, *approximate matching search*, *custom searches*, and *search of modifications*.

Manual browsing

Manual browsing means that the forensic analyst browses collected information and singles out objects of desired type. The only tool used in manual browsing is a viewer of some sort. It takes a data object, such as file or network packet, decodes the object and presents the result in a human-comprehensible form.

Manual browsing is slow. Most investigations collect large quantities of digital information, which makes manual browsing of the entire collected information unacceptably time consuming.

Keyword search

Keyword search is automatic search of digital information for data objects containing specified key words. It is the earliest and the most widespread technique for speeding up manual browsing. The output of keyword search is the list of found data objects (or locations thereof).

Keywords are rarely sufficient to specify the desired type of data objects precisely. As a result, the output of keyword search can contain *false positives*, objects that do not belong to the desired type even though they contain specified keywords. To remove false positives, the forensic scientist has to manually browse the data objects found by the keyword search.

Another problem of keyword search is *false negatives.* They are objects of desired type that are missed by the search. False negatives occur if the search utility cannot properly interpret the data objects being searched. It may be caused by encryption, compression, or inability of the search utility to interpret novel data format[1].

A strategy for choosing key words and phrases is described in Chapter 6 of [82]. In summary, it prescribes (1) to choose words and phrases highly specific to the objects of the desired type, such as specific names, addresses, bank account numbers, etc.; and (2) to specify all possible variations of these words.

Regular expression search

Regular expression search is an extension of keyword search. Regular expressions described in [43] provide a more flexible language for describing objects of interest than keywords. Apart from formulating keyword searches, regular expressions can be used to specify searches for Internet e-mail addresses, and files of specific type. Forensic utility EnCase [68] performs regular expression searches.

Regular expression searches suffer from false positives and false negatives just like keyword searches, because not all types of data can be adequately defined using regular expressions.

Approximate matching search

Approximate matching search is a development of regular expression search. It uses matching algorithm that permits character mismatches when searching for keyword or pattern. The user must specify the degree of mismatches allowed. Approximate matching can detect misspelled words, but mismatches also in-

[1] There is a constant lag between development of new data formats and the ability of forensic search tools to interpret them. An attempt to reduce time gap between appearance of new data formats and their incorporation into search tools was made in [36].

crease the number of false positives. One of the utilities used for approximate search is `agrep` described in [83].

Custom searches

The expressiveness of regular expressions is limited. Searches for objects satisfying more complex criteria are programmed using a general purpose programming language. For example, the *FILTER_I* tool from new Technologies Inc. uses heuristic procedure described in [7] to find full names of persons in the collected information. Most custom searches, including *FILTER_I* tool suffer from false positives and false negatives.

Search of modifications

Search of modification is automated search for data objects that have been modified since specified moment in the past.

Modification of data objects that are not usually modified, such as operating system utilities, can be detected by comparing their current hash with their expected hash. A library of expected hashes must be build prior to the search. Several tools for building libraries of expected hashes are described in the “file hashes” section of [59].

Modification of a file can also be inferred from modification of its timestamp. Although plausible in many cases, this inference is circumstantial. Investigator assumes that a file is always modified simultaneously with its timestamp, and since the timestamp is modified, he infers that the file was modified too. This is a form of event reconstruction and is further discussed in the following subsection.

3.2.2 Reconstruction of events

Search techniques are commonly used for finding incriminating information, because “currently, mere possession of a digital computer links a suspect to all the data it contains" (see page 10 of [4]).

However, the mere fact of presence of objects does not prove that the owner of the computer is responsible for putting the objects in it. Apart from the owner, the objects can be generated automatically by the system. Or they can be planted by an intruder or virus program. Or they can be left by the previous owner of the computer. To determine who is responsible, the investigator must reconstruct events in the past that caused presence of the objects.

Reconstruction of events inside a computer requires understanding of computer functionality. Many techniques emerged for reconstructing events in specific operating systems. This dissertation classifies these techniques according to the primary object of analysis. Two major classes are identified: *log file analysis* and *file system analysis*.

Log file analysis

A log file is a purposefully generated record of past events in a computer system. It is organised as a sequence of entries. A log file entry usually consists of a timestamp, an identifier of the process that generated the entry, and some description of the reason for generating an entry.

It is common to have multiple log files on a single computer system. Different log files are usually created by the operating system for different types of events. In addition, many applications maintain their own log files.

Log file entries are generated by the system processes when something important (from the process's point of view) happens. For example, a TCP wrapper process described in [72] generates one log file entry when a TCP connection is established and another log file entry when the TCP connection is released.

The knowledge of circumstances, in which processes generate log file entries, permits forensic scientist to infer from presence or absence of log file entries that certain events happened. For example, from presence of two log file entries generated by TCP wrapper for some TCP connection X, forensic scientist can conclude that

- TCP connection X happened
- X was established at the time of the first entry
- X was released at the time of the second entry

This reasoning suffers from implicit assumptions. It is assumed that the log file entries were generated by the TCP wrapper, which functioned according to the expectations of the forensic scientist; that the entries have not been tampered with; and that the timestamps on the entries reflect real time of the moments when the entries were generated. It is not always possible to ascertain these assumptions, which results in several possible explanations for appearance of the log file entries. For example, if possibility of tampering cannot be excluded, then forgery of the log file entries could be a possible explanation for their existence. The problem of uncertainty in digital evidence is discussed at length in [24]. To combat uncertainty caused by multiple explanations, forensic analyst seeks *corroborating evidence*, which can reduce number of possible explanations or give stronger support to one explanation than another.

Determining temporal order with timestamps. Timestamps on log file entries are commonly used to determine temporal order of entries from different log files. The process is complicated by two time related problems, even if the possibility of tampering is excluded.

First problem may arise if the log file entries are recorded on different computers with different system clocks. Apart from individual clock imprecision, there may be an *unknown* skew between clocks used to produce each of the timestamps. If the skew is unknown, it is possible that the entry with the smaller timestamp could have been generated after the entry with the bigger timestamp.

Second problem may arise if resolution of the clocks is too *coarse*. As a result, the entries may have identical timestamps, in which case it is also not possible to determine whether one entry was generated before the other.

File system analysis

Log files is not the only source of evidence that can be used for event reconstruction. Other data objects can also be used. This subsection describes how structural information stored by the file system can be used for event reconstruction.

In most operating systems, a data storage device is represented at the lowest logical level by a sequence of equally sized storage blocks that can be read and written independently. Most file systems divide all blocks into two groups. One group is used for storing user data, and the other group is used for storing structural information.

Structural information includes structure of directory tree, file names, locations of data blocks allocated for individual files, locations of unallocated blocks, etc. Operating system manipulates structural information in a certain well-defined way that can be exploited for event reconstruction.

Detection of deleted files. Information about individual files is stored in standardised *file entries* whose organisation differs from file system to file system. In Unix file systems, the information about a file is stored in a combination of i-node and directory entries pointing to that i-node. In Windows NT file system (NTFS), information about a file is stored in an entry of the Master File Table.

When a disk or a disk partition is first formatted, all such file entries are set to initial “unallocated” value. When a file entry is allocated for a file, it becomes *active.* Its fields are filled with proper information about the file. In most file systems, however, the file entry is not restored to the “unallocated” value when the file is deleted. As a result, presence of a file entry whose value is different from the initial “unallocated” value, indicates that that file entry once represented a file, which was subsequently deleted.

File attribute analysis. Every file in a file system — either active or deleted — has a set of attributes such as name, access permissions, timestamps and location of disc blocks allocated to the file. File attributes change when applications manipulate files via operating system calls.

File attributes can be analysed in much the same way as log file entries. Timestamps are a particularly important source of information for event reconstruction. In most file systems a file has at least one timestamp. In NTFS, for example, every *active* (i.e. non-deleted) file has three timestamps, which are collectively known as MAC-times.

- Time of last Modification (M)
- Time of last Access (A)
- Time of Creation (C)

Imagine that there is a log file that records every file operation in the computer. In this imaginary log file, each of the MAC-times would correspond to the last entry for the corresponding operation (modification, access, or creation) on the file entry in which the the timestamp is located. To visualise this similarity between MAC-times and the log file, the mactimes tool from the coroner's toolkit [34] sorts individual MAC-times of files — both active and deleted — and presents them in a list, which resembles a log file.

Signatures of different activities can be identified in MAC-times like in ordinary log files. Given below are several such signatures, which have been published.

Restoration of a directory from a backup. According to [10], the fact that a directory was restored from a backup can be detected by inequality of timestamps on the directory itself and on its sub-directory '.' or '..'. When the directory is first created, both the directory timestamp and the timestamp on its sub-directories '.' and '..' are equal. When the directory is restored from a backup, the directory itself is assigned the old timestamp, but its sub-directories '.' and '..' are timestamped with the time of backup restoration.

Exploit compilation, running, and deletion. In [79], the signature of compiling, running, and deleting an exploit program is explored. It is concluded that "when someone compiles, runs, and deletes an exploit program, we expect to find traces of the deleted program source file, of the deleted executable file, as well as traces of compiler temporary files."

Moving a file. When a file is being moved in Microsoft FAT file systems, the old file entry is deleted, and a new file entry is used in the new location. According to [74], the new file entry maintains same block allocation information as the old entry. Thus, the discovery of a deleted file entry, whose allocation information is identical to some active file, supports possibility that the file was moved.

Reconstruction of deleted files. In most file systems file deletion does not erase the information stored in the file. Instead, the file entry and the data blocks used by the file are marked as unallocated, so that they can be reused later for another file. Thus, unless the data blocks and the deleted file entry have been re-allocated to another file, the deleted file can usually be recovered by restoring its file entry and data blocks to active status.

Even if the file entry and some of the data blocks have been re-allocated, it may still be possible to reconstruct parts of the file. The lazarus tool described in [35], for example, uses several heuristics to find and piece together blocks that (could have) once belonged to a file. Lazarus uses the following heuristics about file systems and common file formats.

- In most file systems, a file begins at the beginning of a disk block;
- Most file systems write file into contiguous blocks, if possible;

- Most file formats have a distinguishing pattern of bytes near the beginning of the file;
- For most file formats, same type of data is stored in all blocks of a file.

Lazarus analyses disc blocks sequentially. For each block, lazarus tries to determine (1) the type of data stored in the block – by calculating heuristic characteristics of the data in the block; and (2) whether the block is a first block in a file – using well known file signatures. Once the block is determined as a “first block”, all subsequent blocks with the same type of information are appended to it until new “first block” is found.

This process can be viewed as a very crude and approximate reconstruction based on some knowledge of the file system and application programs. Each reconstructed file can be seen as a statement that that file was once created by an application program, which was able to write such a file.

Since lazarus makes very bold assumptions about the file system, its reconstruction is highly unreliable, which is acknowledged by [35]. Despite that fact, [35] states that lazarus works well for small files that fit entirely in one disk block. The effectiveness of tools such as lazarus can probably be improved by using more sophisticated techniques for determining the type of information contained in a disk block. One such technique that employs support vector machines has been recently described in [31].

3.2.3 Time analysis

Timestamps are readily available source of time, but they are easy to forge. Several attempts have been made to determine time of event using sources other than timestamps. Currently, two such methods have been published. They are *time bounding* and *dynamic time analysis.*

Time bounding

It was shown in Section 3.2.2 that timestamps can be used for determining temporal order of events. The inverse of this process is also possible – if the temporal order of events is known *a priori*, then it can be used to estimate time of events. Suppose that three events A, B, and C happened. Suppose also that it is known that event A happened before event B, and that event B happened before event C. The time of event B must, therefore, be bounded by the times of events A and C.

An example of time bounding is given in [72]. It considers a telnet session, which is carried over a TCP connection. The times of establishing and releasing the TCP connection are recorded by the TCP wrapper. Since (1) the TCP connection has to be established before the telnet session can be started, and (2) the telnet session must terminate before or together with the termination of the TCP connection, the time of the telnet connection is bounded by the times recorded by the TCP wrapper. This reasoning is illustrated in Figure 3.2.

Dynamic time analysis

A different approach to using external sources of time is described in [80]. It exploits the ability of web servers to insert timestamps into web pages, which they transmit to the client computers. As a result of this insertion, a web page stored in a web browser's disk cache has two timestamps. The first timestamp is the creation time of the file, which contains the web page. The second timestamp is the timestamp inserted by the web server.

According to [80], the offset between the two timestamps of the web page reflects the deviation of the local clock from the real time. It is proposed to use that offset to calculate the real time of other timestamps on the local machine. To improve precision, it is proposed to use the average offset calculated for a number of web pages downloaded from different web servers.

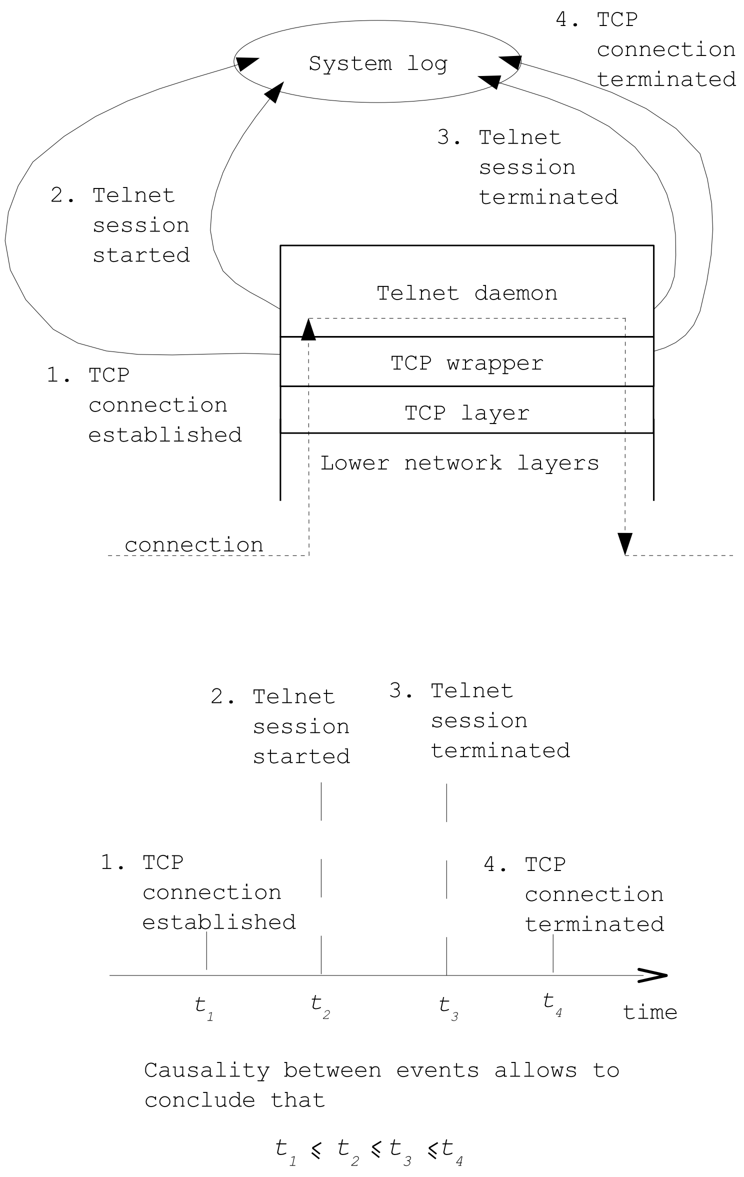
System log
4. TCP connection terminated
3. Telnet session terminated
2. Telnet session started
Telnet daemon
TCP wrapper
TCP layer
Lower network layers
1. TCP connection established
connection
2. Telnet session started
3. Telnet session terminated
1. TCP connection established
4. TCP connection terminated
t_1
t_2
t_3
t_4
time
Causality between events allows to conclude that
$t_1 \leqslant t_2 \leqslant t_3 \leqslant t_4$

Figure 3.2: An example of time bounding

This analysis assumes that (1) timestamps are not tampered with, and that (2) the offset between system clock and real time is constant at all times (or at least that it does not deviate dramatically).

3.3 Summary

This chapter presented a review of digital forensic concepts. It identified the stages of digital investigative process and described the major types of digital forensic techniques used for examination and analysis of digital evidence.

As a final point, note that the need for effective and efficient digital forensic analysis has been a major driving force in the development of digital forensics. Manual browsing was initially the only way to do digital forensics. It was later augmented with various search utilities and, more recently, with tools such as mactimes and lazarus that support more in-depth analysis of digital evidence. Due to the limited time and manpower available to a forensic investigation, there is a constant demand for tools and techniques that increase the accuracy of digital forensic analysis and minimise the time required for it.

The next chapter looks in more detail at the problem of event reconstruction in digital investigations. It explains why a theory of event reconstruction is required in digital forensics, analyses the state of the relevant art, and, ultimately, formulates the research problem addressed in this dissertation.