Masaryk University
Faculty of Informatics

Tool for forensic analyses of digital traces

Master's Thesis

Bc. Mária Hatalová

Brno, Spring 2018

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Mária Hatalová

Advisor: prof. RNDr. Václav Matyáš M.Sc. Ph.D.

Acknowledgements

I would like to express my gratitude to my advisor prof. Václav Matyáš for the continuous guidance throughout writing the thesis, for his insightful comments, and for enhancing my motivation.

My thanks also go to my technical consultant Mgr. Roman Pavlík from Trusted Network Solutions, a.s. for providing me an insight into the work of expert witnesses in the field of IT, for his encouragement, and for giving me the perception that the work is meaningful and useful in practice.

I would also like to thank Petr Svoboda from Trusted Network Solutions, a.s. for consultations regarding IT Forensic Tool v1.0 and for technical support.

Last but not least, my gratitude goes to my family and friends for their support and patience.

Abstract

The thesis examines selected issuesin the area of forensic analysis and their impact on expert witnesses that work for the Police of the Czech Republic. A description of a typical forensic task as it is assigned by the police is provided, together with an overview of available tools that could be used for solving the task. The most promising candidate – Autopsy – is tested on a real task, but it turns out to be not suitable for this purpose. The rest of the thesis is devoted to the IT Forensic Tool that is being developed by the consultant of the thesis for solving forensic tasks. A number of improvements on functionality and optimization of the tool are proposed and implemented in the thesis.

Keywords

Autopsy, database, expert testimony, forensic analysis, indexing, LaTeX, optical character recognition, pattern matching, PostgreSQL

Contents

List of Tables

List of Figures

1 Introduction

Forensic analysis is a dynamically growing area of IT. Its purpose is to reconstruct – from samples of supplied data – the timeline of events that occurred or to find specific digital traces of users' activities. Forensic analysis of digital traces, also referred to as digital forensics, is widely used by the police when investigating crimes. With the aim to find evidence that could be used at the court, the police often seizes computers and data carriers throughout house search. They delegate an expert witness with the specialization in IT to analyze the data carriers and to write an expert testimony with interpretation of outcomes of the analysis. The analysis consists of finding all files with occurrences of predefined keywords (e.g., names of suspected persons or companies).

The thesis studies the work of an expert witness starting with an acquiry of images of the data carriers from the police, through analysis of the data, ending with the completion of an expert testimony that is subsequently submitted to the judge. The aim of the thesis is to simplify and speed up the work of expert witnesses and to reduce the amount of manual work that needs to be done. The thesis focuses on the requirements and needs of expert witnesses that work for the Police of the Czech Republic (later referred to as "police").

Chapter 2 describes a typical forensic task as it is assigned by the police, the formats of the images of the data carriers, and how the images are created. Chapter 3 provides an overview of several digital forensics tools and environments. Based on their properties and functionality, we suggest the best candidate for the use by expert witnesses for completion of the typical forensic task. Chapter 4 is devoted to Autopsy – the most prominent tool out of the tools that were described in Chapter 3. The aim of this chapter is to test the use of Autopsy on a real case with real data and to evaluate how it copes with completing the typical forensic task. We describe the methodology of the test and the hardware on which the test was executed. It also provides evaluation of the performance and stability of the tool and it concludes whether it is suitable or not for fulfillment of the forensic task. Due to multiple issues – described in Section 4.5 – that were

encountered during the testing, Autopsy turned out not to be suitable for the task.

As Autopsy failed in the test on a real task, another way how to efficiently fulfill the forensic tasks needs to be found. The rest of the thesis is devoted to the IT Forensic Tool (ITFT) that is being developed by the technical consultant of the thesis for this purpose. Chapter 5 introduces the tool and its features and it provides a step-by-step description of the whole process of solving a typical forensic task by this tool.

Chapter 6 is devoted to a new version of IT Forensic Tool – IT Forensic Tool v2.0. It discusses the requirements for further functionality that should be added to ITFT and it proposes and implements solutions for each of the requirements. The first requirement is integration of a database design into the tool in order to simplify the way how it stores data and to enhance its extensibility over time. There is also a need for speeding up the keyword search. Several options of its acceleration are discussed and the best approach is selected. Besides this, in order to be able to search automatically for keywords in the text that is inside images, the keyword search should be equipped with Optical Character Recognition (OCR). The possibilities of integrating OCR into ITFT are discussed and a suitable OCR tool is chosen for this purpose. Last but not least, the manual work required for creating an expert testimony as a pdf document should be minimized. The thesis suggests and implements means for partially automating the process.

Chapter 7 suggests more improvements for further work and Chapter 8 concludes the thesis.

2 Typical forensic task

This chapter describes a typical forensic task as it is assigned by the police. We specify the most common formats in which the data for the analysis are provided and the procedure of creating images from the original data.

A typical task looks as follows:

1. Find electronic files (including the deleted ones) containing documents, e-mail communications and history of online communications of specified persons on the data carriers provided by the police.

2. Out of the found files choose those that contain the listed keywords or their parts (business companies and institutions, natural persons, words).

3. Create an expert testimony as a pdf document containing the chosen files together with their metadata. The report is to be accompanied by electronic copies of the files on a suitable data carrier.

The provided data carriers contain bit-by-bit images of the carriers seized throughout house search or the seized devices themselves. Typically, there are images of hard disks or flash disks and they can be stored in various formats. The most common formats are raw formats and the EnCase format, and for the purpose of the thesis we will consider only these two. The total capacity of the provided data carriers is typically thousands of gigabytes.

2.1 Raw formats

A raw format is a bit-by-bit copy of the original [1]. It does not contain any metadata and it is not compressed. Some tools provide the metadata in separate files. Typically, raw images have suffixes .dd, .raw, or .img, and can be created by a Linux tool *dd* or by one of its followers, for example, *dc3dd*, *dcfldd*, or *dd.exe* that is a Windows implementation of the original *dd* [2].

2.2 EnCase Evidence File format (version 1)

The EnCase Evidence File format is a proprietary format developed by the Guidance Software Inc. [3]. It uses the file extension .E01 and it is based on the Expert Witness Format (EWF) by ASR Data. These image files are commonly referred to as Expert Witness, E01 or EWF files [4].

EnCase files contain a header and a footer with metadata, such as the source disk operating system, timestamps, and cryptographic hashes. They also contain a Cyclic Redundancy Check (CRC) in order to preserve the integrity and they are compressed by default.

Version 2 of the EnCase Evidence File format with some additional features was introduced in Encase 7. It is not backward compatible with version 1.

2.3 Split images

Both raw and EnCase allow for separation of disk images into so-called split images that are more manageable. The naming conventions for split image file sets are as follows:

- EnCase: suffix Enn

  image_set.E01
  image_set.E02
  image_set.E03
  ...

- Raw: suffix nnn

  image_set.001
  image_set.002
  image_set.003
  ...

2.4 Creating raw and EnCase images

Before creating a bit-by-bit copy of an original data carrier gained from house search, e.g., a hard disk of a laptop, the disk is removed

from the powered down system and then the image can be created. This is called dead imaging. When creating forensic images, it is essential that the original data is not altered. In order to ensure the integrity of the data, write-blockers are used. As the name implies, they prevent any data from being written to the original disk and allow only for the read access [4]. Therefore, timestamps of the files on the disk remain the same. Write-blockers can be either hardware or software based.

Another requirement for forensic images is that we have to be able to verify that the created image is the same as the original disk [5]. This is done by computing hashes of both the original disk and its image and comparing them. Imaging tools support various hash functions, such as SHA-2 or MD5.

Multiple tools for creating forensic images are available. The most common are *FTK Imager*, *EnCase Forensic Imager*, *ewfacquire* from the library *libewf*, *dd*, and other *dd*-like tools that have already been mentioned in Section [4]. All these tools support the raw formats and *FTK Imager*, *EnCase Forensic Imager*, and *ewfacquire* support also the EnCase format.

3 Forensic tools

This chapter provides an overview of several commonly used digital forensics tools and environments that could be used for solving a typical forensic task assigned by the Police of the Czech Republic. The tools were chosen based on recommendations from the technical consultant of the thesis and based on experience and opinions of members of the digital forensics community at Forensic Focus online forum [6] [7].

Together with the tools overview we introduce the criteria that the best candidate for being used for solving the typical task should satisfy. Then, based on applying the criteria, the most suitable tool is chosen. The tool should support the basic functionality needed by expert witnesses, such as extraction of files from disk images and keyword search. It should be up-to-date, and preferably it should be open source.

3.1 Tools overview

This section introduces four candidates for a forensic tool for the typical forensic task: EnCase Forensic, CAINE, The Coroner's Toolkit, and The Sleuth Kit with Autopsy.

3.1.1 EnCase Forensic

Encase Forensic is a commercial product by Guidance Software Inc. It is a widely used forensic tool that provides a variety of forensic features, including disk imaging, keyword search, etc. Images are stored in EnCase Evidence File format (see 2.2) [8]. The tool supports a wide range of operating and file systems and since the version 8, released in 2016, also file system acquisition on mobile operating systems is provided [9].

Encase Forensic allows efficient management of the individual investigation workflows and reporting is done using customizable templates. Extensibility of the tool is ensured through EnScripts which are automated code commands that streamline tasks.

3.1.2 CAINE

CAINE (Computer Aided INvestigative Environment) is an Italian Linux live distribution that offers a complete forensic environment [10]. It integrates existing software tools, such as dc3dd, exif, libewf, and Autopsy. The complete list of the tools can be found at [11].

3.1.3 The Coroner's Toolkit

The Coroner's Toolkit (TCT) is a collection of forensic utilities written by Wietse Venema and Dan Farmer for a post-mortem analysis of a UNIX system [12]. The software dates to the year 1999, when it was first presented at the Thomas J. Watson Research Center. TCT is currently not being developed anymore, however, it has its official successor called the Sleuth Kit. It was developed by Brian Carrier and it uses some code and design of TCT.

TCT consists of multiple components including *grave-robber* tool that serves for the capture of forensic information, *mactime* for time analysis, *lazarus* for recovery of deleted files, and *findkey* for recovery of cryptographic keys form running processes or from files [13].

3.1.4 The Sleuth Kit and Autopsy

As already stated, The Sleuth Kit (TSK) is the successor of The Coroner's Toolkit. It consists of a collection of command line file and volume system forensic analysis tools and a C library and it serves for analysis of disk images and for recovering files from them [14]. It is written in C and Perl and it runs on both Windows and Unix platforms.

In order to avoid using the command line tools of TSK directly, a graphical interface called Autopsy can be used. Autopsy provides case management, image integrity, volume and file system analysis, keyword search, and other features [15]. It was designed with a modular architecture. Some of the modules come with it out of the box while others are available from third parties.

When it comes to reporting, Autopsy provides three default types of reports – HTML, XLS, and Body file. Users can also create their own customized reporting modules to match their specific needs or they can edit the existing ones.

3.2 Comparison criteria

1. **Supported features:** The tool should provide at least the following features that are essential for the typical forensic task:

   - **Extraction of files:** The tool should support extraction of files from the obtained images, in order to be able to further analyze them and to perform the keyword search. The tool should be able to detect also hidden and deleted files.
   - **Keyword search:** The key requirement for the tool is that it can find all files that contain the keywords specified by the police.
   - **Report generation:** It is desirable that the tool provides an option to automatically generate a report summarizing the results of the keyword search, so that it does not have to be created manually by an expert witness.

2. **Supported image formats:** The tool should support the formats of the images that are supplied by the police. These are typically in the raw/dd format or in the EnCase (E01).

3. **Supported platforms:** The tool should be supported on Linux. Linux operating systems are open source and the tools and applications for them are typically also open source.

4. **License:** Preferably, the tool should be open source. Buying a license for a commercial tool can be cost prohibitive for self-employed expert witnesses or for smaller companies that offer expert witness services. Also, the functionality that is needed by expert witnesses with respect to the typical forensic task is very narrow compared to what forensic tools typically offer. Therefore, it might be inadequate to pay for a whole toolkit when only a small part of it is needed.

5. **Extensibility:** Since a typical forensic task is very specific in its requirements, it would be beneficial if the tool was extensible and could be adjusted to the needs of expert witnesses.

6. **Up to date, support, documentation:** It is essential that the tool is up to date, has a thorough documentation, and a sufficient support is available in case of any problems. An advantage of commercial products is that the support is usually included in the license, but open source tools can also have an adequate support in form of mailing lists, wiki pages, or discussion fora.

3.3 Tools comparison

This section provides a comparison of the selected tools with respect to the criteria defined above.

3.3.1 Supported features

As already mentioned, the tool should support certain features in order to be usable for solving the typical forensic task. Table 3.1 summarizes the features that are provided by the tools[1].

Autopsy and EnCase Forensic support all the required features, while The Coroner's Toolkit supports only extraction of deleted files.

Tool	Extraction of files	Keyword search	Report generation
EnCase Forensic	✓	✓	✓
The Coroner's Toolkit	✓[2]		
Autopsy	✓	✓	✓

Table 3.1: Supported features.

1. CAINE is not included in the table due to the fact that it contains Autopsy. Therefore, it supports all the features that are supported by Autopsy. The same holds for the image formats.
2. Only extraction of deleted files is directly supported (*lazarus* and *unrm* tools). The authors expect the users to mount the image manually to extract the regular files.

3.3.2 Supported image formats

Table 3.2 provides an overview of the image formats supported by the tools. Autopsy supports both raw and E01 formats, while EnCase Forensic supports only the E01 format owned by Guidance Software – the author of Encase Forensic. The Coroner's Toolkit does not directly support any of the formats.

Tool	raw/dd	E01
EnCase Forensic		✓
The Coroner's Toolkit[3]		
Autopsy	✓	✓

Table 3.2: Supported image formats.

3.3.3 Supported platforms

The preferred platform is Linux. The Coroner's Toolkit together with Autopsy are supported on Linux systems, CAINE itself is a Linux distribution, however, EnCase Forensic is strictly designed for Windows.

Tool	Linux	Windows	Mac
EnCase Forensic		✓	✓
CAINE	✓		
The Coroner's Toolkit	✓		
Autopsy	✓	✓	✓

Table 3.3: Platforms.

3. The authors expect the users to use the `mount` Linux command line utility which supports both raw and E01 formats.

3.3.4 License

On one hand, CAINE, The Coroner's Toolkit, and Autopsy are issued under free software licenses, on the other hand, EnCase Forensic has a commercial license.

Tool	IBM Public License	Apache License 2.0	LGPL[4]	Commercial License
EnCase Forensic				✓
CAINE			✓	
The Coroner's Toolkit	✓			
Autopsy		✓		

Table 3.4: License.

3.3.5 Extensibility

This section provides an overview of the extensibility options of the individual tools. In short, all the tools can be extended in some way.

- **EnCase Forensic:** Extensibility in EnCase Forensic is provided by so-called EnScripts, which are automated code commands that streamline tasks. They can be written by developers via EnCase AppCentral[5].
- **CAINE:** As a free GNU/Linux distribution, CAINE can be arbitrarily modified under the terms of the LGPL license.
- **The Coroner's Toolkit:** As a free software, The Coroner's Toolkit can be further extended. Extended versions containing additional functionality implemented by other developers can be, based on a mail request, listed at the official TCT page.

4. LGPL – GNU Lesser General Public License.
5. AppCentral is an EnCase platform that features dozens of applications from users. The applications are tested and verified by the team at Guidance Software Inc. [16]

- **Autopsy:** Autopsy has been designed with a modular architecture and it provides an option to incorporate third-party modules. These can be written in Python or Java.

3.3.6 Up to date, support, documentation

Table 3.5 provides an overview of what kind of support is provided with the individual tools, the form of and the quality of the documentation and, last but not least, whether the tool is still being developed.

Tool	Up to date	Support	Documentation
EnCase Forensic	✓	technical support for customers	When customers purchase EnCase Forensic they receive a delivery email containing User Guides/Manuals in pdf.
CAINE	✓	discussion forum	At the official page a brief description of CAINE is provided together with the links to related tutorials. Some of the links are not operational anymore.
The Coroner's Toolkit		FAQ mailing list	Handouts from a class on UNIX computer forensic analysis given by the authors of the tool – Dan Farmer and Wietse Venema. They provide a brief description and guidelines for the individual tools of TCT.
Autopsy	✓	wiki page mailing list discussion forum	A very thorough and understandable documentation for both users (*User's Guide*) and developers (*Developer's Guide*).

Table 3.5: Up to date, support, documentation.

3.3.7 Conclusion

Examining the properties of each of the individual tools, Autopsy seems to be the most promising one. It supports all the required features and image formats, and it is available for Linux systems. Thanks to being issued under a free software license and thanks to having a modular architecture, it is possible to add new functionality and to adjust the tool to the needs of expert witnesses. The tool is up to date, it is provided with a thorough documentation and an adequate support.

Comparing Autopsy to The Coroner's Toolkit, it is a clear choice as TCT is not being developed anymore and Autopsy is its official successor. In regard to CAINE distribution that integrates Autopsy and, therefore, provides the required functionality, it is a better option to use Autopsy as a standalone. There is no need for using the other tools that are incorporated into CAINE and to maintain the whole distribution instead of a single application. Autopsy also has an advantage over EnCase Forensic. On one hand, EnCase supports the functionality needed for processing the typical forensic task, it is extensible, and it is supplied with a customer support. On the other hand, it is a commercial tool with a yearly license of several thousands of dollars, which might not be affordable by some expert witnesses.

To conclude, Autopsy has an advantage over the rest of the tools and will be further examined using real data in order to see how it is usable in practice.

4 Autopsy

Now that we have chosen Autopsy we need to look at its functionality in more detail and to test how it copes with solving the task assigned by the police in practice. We are interested in the usability of the tool in terms of performance and stability.

The first part of this chapter provides a closer look at the features Autopsy is equipped with, while the rest of the chapter is devoted to testing it on a real forensic task using real data. The test methodology is presented together with the used hardware, the results of the performance test and the problems encountered during the testing. Finally, a conclusion on usability of Autopsy for solving the typical forensic task is provided.

4.1 Features

This section outlines the process of analysis of a typical forensic task using Autopsy. Descriptions of the key features/modules that are used throughout the individual steps are provided.

1. **File system analysis:** Autopsy provides a File system analysis feature. It accepts disk images in raw (i.e., dd), EnCase, and AFF formats. Firstly, the images are analyzed using volume system tools that examine the layout of disks and other media and they identify and export partitions. Secondly, the identified partitions are analyzed with file system analysis tools. Autopsy supports a wide range of file systems, including NTFS, Ext2/Ext3/Ext4, FAT12/FAT16/FAT32/ExFAT, HFS+, and many others. The whole process of image analysis covering volume and file system analysis is referred to as "image ingestion".

2. **Keyword search:** The Keyword search module can be run either during the image ingestion process or after the ingest. Before running the module, a list of keywords and regular expressions to be searched for has to be defined. The keyword search consists of two steps. Firstly, Autopsy extracts texts from different file types (html, Microsoft Office, pdf, ...). For this purpose, Apache

Tika toolkit and other libraries are used. Secondly, a full text search using Apache Solr is performed. Results of the search are provided continuously in the "Keyword hits" section of the Autopsy navigation panel.

3. **Reporting:** After the keyword search is performed, a report needs to be created in order to document the findings. By default, Autopsy provides three types of reports: html, xls, and Body file. Each of them is configurable and the user can choose what information he wants to include. For example, the html and xls reports can be, for example, configured to display keyword hits, which is particularly useful for the purpose of the typical forensic task. The Body file report is intended for timeline analysis, which is not applicable in this case. A significant advantage of the reporting infrastructure of Autopsy is that it allows editing the existing reporting modules and also creating new ones to customize the behavior to suit users' specific needs.

On top of the above described features that are essential for the typical forensic task, Autopsy provides several other features that may be helpful in improving the efficiency of the analysis.

- **Hash set filtering:** This feature allows users to filter out known files automatically, e.g., various system files. By omitting these from the keyword search, the analysis can be speeded up.

- **Tags:** Users can tag files with arbitrary labels and they can add comments. The tags and comments can be also included in the report.

- **Android support:** Autopsy supports extraction of data from SMS messages, call logs, contacts, and more. This could be used in cases when there are also mobile phones or tablets with the Android operating system among the data carriers supplied by the police.

4.2 Test methodology

This section introduces the methodology of our tests. It describes the data set for the analysis and it outlines the actions to be taken in the Autopsy user interface.

As was stated in the description of the typical task in Chapter 2, the amount of data to be analyzed is typically in terabytes. For the purpose of the test we will use a set of 15 images (raw or E01) with their total size of 2 TB. The task is to inspect all the images and find all occurrences of predefined keywords. The keyword list contains 20 words. The case is ignored and diacritic in not taken into account.

The steps of the test are as follows:

1. Configure Autopsy to use 4 threads. The rest of the configuration stays in default settings.
2. Create a keyword list. For simplification, set each keyword to be looked for as an exact match.
3. For each image ("data source" in the Autopsy terminology) issue the "Add data source" command and select the keyword search module with the keyword list to be run simultaneously with the image ingestion.

4.3 Hardware

This section provides a description of the hardware used for the performance testing.

- Model: DELL PowerEdge(TM) 2950.
- Processor: 2x quad core Intel(R) Xeon(R) X5460, 3.16 GHz, L2 cache 12 MB.
- Chipset: Intel 5000X.
- RAM: 8 x 4096 MB, DDR2 FB-DIMM, 667 MHz.
- Network cards: 2 x onboard Broadcom(R) NetXtreme II(TM) 5708.
- Storage:
  - LSI Logic / Symbios Logic MegaRAID SAS 1078 (rev 04),

- Dell PERC 6/i Integrated RAID Controller,
- RAID-5: 4 x WD10EZRX-00L4HB0, SATA, 1 TB, 64MB Cache, 5400 RPM.

4.4 Performance test

This section provides an overview of the results of the performance test. The test was executed on a sample case for which 15 disk images of the total size of 1895 GB had been provided. The following chart displays the sizes of the images and the corresponding amount of time needed for processing them with Autopsy. The average processing speed is 0.3 GB per minute. This means that for processing 1 TB of data, about 55 hours are needed.

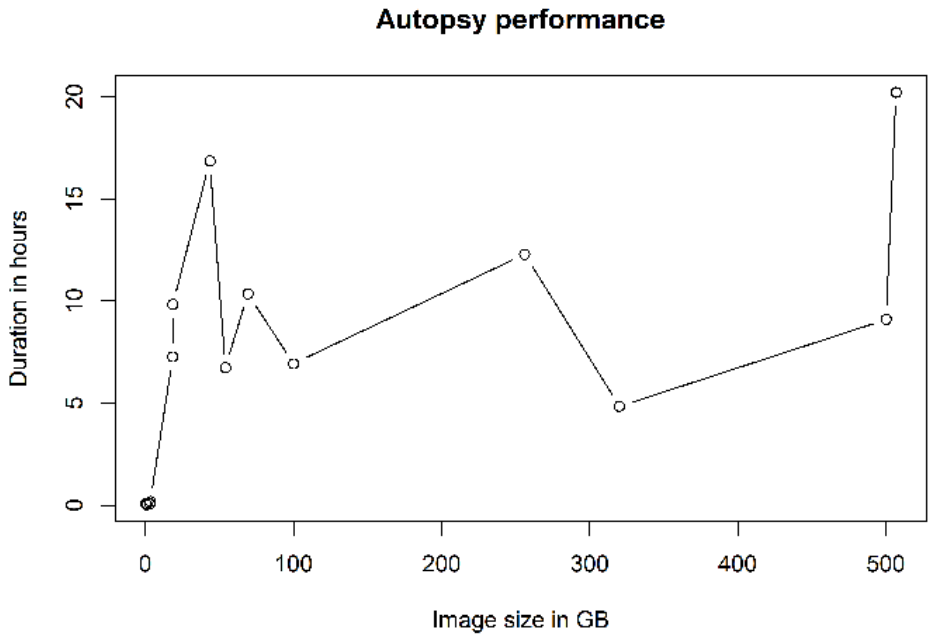


Figure 4.1: Autopsy performance.

4.5 Problems encountered

Several more or less severe problems were revealed throughout the run of the performance test.

- **Keyword search module fails to load:** The keyword search module uses Apache Solr[1], which is statically configured to use the TCP port 23,232. During the test the keyword search failed to load and the following error was reported: "Indexing server port 23,232 is not available." Other Autopsy users have also experienced this error [18][19][20]. According to the Autopsy documentation [21] and a related conversation in The Sleuth Kit mailing list [20], such an error message can have several causes. Firstly, another application in the system can be using the same port. Secondly, a security software can be blocking the port. Thirdly, the system can be running very slowly and when Autopsy tries to use Solr, it is not up and listening on the port yet.
- **Freeze:** It happened several times that Autopsy froze and stopped running while performing the keyword search. In such cases Autopsy had to be forced to shut down and the keyword search had to be rerun from scratch. Other users reported the same issue in the Autopsy forum [22][23], however, the cause of the freezing was not identified.
- **Unexpected crash:** There was another issue that we experienced during the run of the keyword search ingest module. Autopsy occasionally crashed and the search had to be started again. The causes of the crashes were not investigated, however, other users also reported unexpected crashes when running ingest modules [24][25].
- **Keyword search and other errors:** In some cases, the keyword search or other actions ended up with an error (e.g., Corrupted bitsPerDocBase, IOException, No space left on device, Keyword Indexing Warning, Failed to open or create core [26]) and had to be repeated. Due to the nontrivial amount of different errors they

1. Solr is an open-source search platform which can be used to build search applications [17]. It was built on top of Lucene (full text search engine).

could not all be investigated more deeply. As a result, the keyword search could not be finished for some of the images and they remained unprocessed.

- **Progress Bar Inaccuracy:** The progress bar that continuously displays what percentage of files has already been processed with the keyword search module did not always reflect the real progress. It happened that at the moment of reaching 100 percent the keyword search had not finished yet and it continued running for a considerable amount of time. This was caused by the fact that the total number of files from which the current percentage is calculated does not include the contents of ZIP and other archives. Therefore, the keyword search will get stuck at 100 percent in case a particular image contains a large number of archives. The overall time needed for the search will significantly exceed the amount of time that could be presumed based on the progress bar. This is not a bug as such, however, it would be appreciated if the real progress of the keyword search could be known.

Other issues

The issues that are described below did not occur during our performance test, but they were reported by other users of Autopsy and they are closely related to the process of solving the typical forensic task. The issues were reported either to the GitHub project of Autopsy [27] or to the forum on the official website of The Sleuth Kit and Autopsy [28]. They have not been fixed yet at the time of writing the thesis.

- **Incorrect hash verification [29]:** For some specific images the E01 verifier returns a result "not verified", although other tools (e.g., FTK Imager and EnCase Forensic Imager) can successfully verify the hash. This is caused by the E01 verifier calculating wrong hashes in some cases.

- **Keyword search duplicate results [30]:** When running a new keyword search with a string that has already been sought for, the occurrences are not unique but are added to the previous

request results. Therefore, the keyword hits in the results are duplicate.

- **Error opening case [31]:** Some users experienced issues when trying to reopen an existing case, no matter if it had been closed properly or not. Autopsy reported "Error opening the case." According to one of the users, it could have had been caused by an improper encoding handling.
- **Failed to add data source [32]:** After creating a new case or opening an existing one, it was not possible to add a data source for some of the users.

4.6 Conclusion

Based on the overview of several currently available tools for digital forensic analysis, Autopsy seemed to be a very good candidate for solving the forensic task assigned by the police. However, multiple issues appeared when it was tested on a real case. Due to the issues, the whole process of the analysis was prolonged significantly. It was not possible to finish the keyword search for some of the images, therefore, the analysis is incomplete. Autopsy is considered unsatisfactory for this kind of task due to the amount of issues that occurred during the testing. A different approach needs to be considered.

5 IT Forensic Tool

Based on the fact that no tool that would be suitable for solving the typical forensic task had been found, programmers from Trusted Network Solutions, a.s. developed a prototype with the aim to match the requirements and to reduce the manual work of an expert witness when creating an expert testimony. The name of the prototype is IT Forensic Tool (ITFT).

This chapter describes the functionality of ITFT together with the whole process of case analysis when this tool is used.

5.1 Description

ITFT is a command line utility written in Perl. It provides analysis of images in different formats (raw, E01), extraction of files that are present in the images (including deleted files and files in free space), keyword search, and file export. It recognizes multiple file formats (plaintext, doc, docx, mdb, pdf, xls, xlsx, ...), it allows for extraction of information from applications, such as Outlook, Skype, and Pohoda (economic software), and it is able to analyze zip and rar archives. The tool uses other opensource utilities, such as ExifTool, Foremost, and PhotoRec, that provide functionality which is useful for completing the typical forensic task.

ITFT provides a user with a set of commands that allows for case management, evidence management, processing of evidence data, keyword search, and file export.

Table 5.1 presents a list of available user commands together with their description. Besides the user commands, ITFT also allows to run internal commands and tools from the command line. However, these are present only for test and debugging purposes. The manual for ITFT including user commands, internal commands, and tools, can be found in Appendix A.1.

User command	Description
addcase	Create a new case.
addevidence	Add a new evidence to the case.
deletecase	Delete a case.
digest	Process the data of a specified piece of evidence or, if no evidence is specified, process all evidence images of a specified case.
export	Export specified files and their metadata summaries.
exportkeywords	Export all files that contain predefined keywords.
listcases	List all cases.
listevidence	List all evidence of a specified case.
modifycase	Change the name of a case.
mount	Mount all evidence of all cases.
mark	Mark a file with a star (create a symlink into a directory with starred files).

Table 5.1: IT Forensic Tool – User commands.

5.2 Case analysis

Here is a step-by-step description of the whole process of solving a typical forensic task with ITFT:

1. Add a new case (`addcase`).
2. Add all images that are to be analyzed (`addevidence`).
3. Process the images (`digest`). All files are extracted from the images, archives are uncompressed, and information from applications, such as Skype, Outlook, and Pohoda is extracted.
4. Define the keywords to be searched for (`keywords.conf`).
5. For each keyword create a list of all files where it occurs (`exportkeywords`) and give the lists to an investigator.
6. The investigator chooses the files that are important and should be included in the expert testimony, that will be used as an evidence at the court.
7. Export the files chosen by the investigator (`export`). A separate text file with a metadata summary is created for each file. It contains information about the file, such as its path, source (which evidence it comes from), access and modification times, checksums, and metadata of the content of the file.
8. Create an expert testimony as a pdf document containing the chosen files together with their metadata summaries. This functionality is not supported by ITFT and it needs to be done manually.

Visualized process of the case analysis in the form of a sequence diagram is provided in Figure 5.1. Besides the ITFT commands, it covers also the communication with the Police.

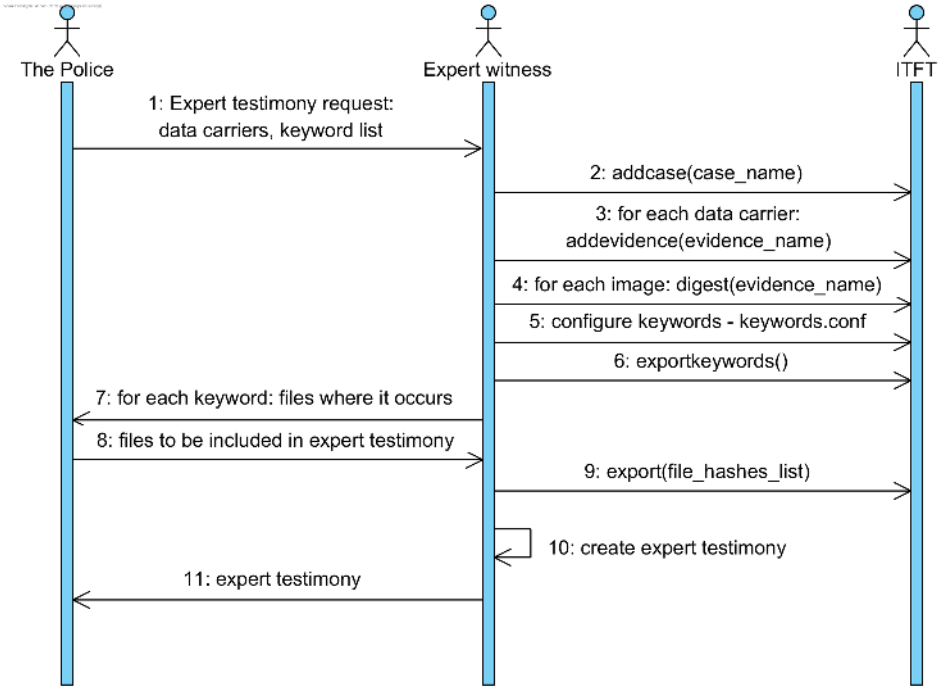


Figure 5.1: Sequence diagram.

6 IT Forensic Tool v2.0

ITFT has been successfully used in Trusted Network Solutions, a.s. for processing multiple cases and creating multiple expert testimonies. Although the tool provides all the essential features for completing typical forensic tasks, there is still room for improvements. The following requirements emerged when working with ITFT:

1. **Database design:** As the functionality and complexity of ITFT grows,it will be more and more difficult to keep the code simple and readable. The way how it stores different kinds of information about cases should be unified in order to enhance extensibility of ITFT over time. This could be done by integrating a database into the current design of the tool.

2. **Keyword search:** The keyword search performance should be optimized.

3. **Optical Character Recognition:** It would be useful to have a feature that would automize the keyword search on the text that is contained in pictures, such as photographed documents or scanned pages. This could be done using Optical Character Recognition (OCR).

4. **Automated reporting:** The reporting process should become more automated. Currently, an expert witness has to create an expert testimony by manually merging the chosen files and their metadata summaries into a single pdf document. The fact that this is not done automatically makes the time needed for case analysis significantly longer.

The aim of the thesis is to incorporate the above mentioned improvements to the current version of ITFT and to create a new version of the tool – IT Forensic Tool v2.0. Each section of this chapter is devoted to one of the suggested improvements: it defines the requirements, presents the solution, and describes the implementation.

6.1 Database design

Currently, most of the data related to cases, images, or files is stored on the disk. Each case has its own directory with multiple subdirectories that contain mounted images, files detected by *photorec*, exported files, uncompressed archives, and files with other information about the particular case. The structure of the case directory had over time become rather complex and there is a need to make it more compact and easier to take in. With the integration of a database into the current design of the tool a significant amount of data can be moved from the case directory into the database. The use of a database brings simplification of operations over the data, it creates space for optimization on the database level, and it increases extensibility of the tool.

This section introduces PostgreSQL database system that is used for the database design of ITFT v2.0 and it presents an entity-relationship model of the database.

6.1.1 PostgreSQL

PostgreSQL is an open source object-relational database system [33]. It is available for all major operating systems, including Linux, UNIX, and Windows. It is ACID compliant [34], it supports foreign keys, joins, views, triggers, and stored procedures and its SQL implementation strongly conforms to the ANSI-SQL:2008 standard [35]. It has a thorough documentation and it provides native programming interfaces for multiple programming languages, including C/C++, Java, .NET, Python, and Perl. The PostgreSQL can be integrated with Perl using Perl DBI module [36], which is a database access module for the Perl programming language. The DBI module needs to be installed along with its DBD::Pg driver [37].

6.1.2 Entity-relationship model

During the process of solving a typical forensic task the tool ITFT works with five basic entities:

1. **case** that the investigation is related to,
2. **images** of the data carriers seized throughout house search,

3. **files** from the images,

4. **keywords** to be searched for,

5. and occurrences – **hits** – of the keywords in the files.

This section introduces an entity-relationship model of the database for ITFT v2.0. Figure 6.1 displays an entity-relationship diagram containing the entities, their attributes, and relationships between them.

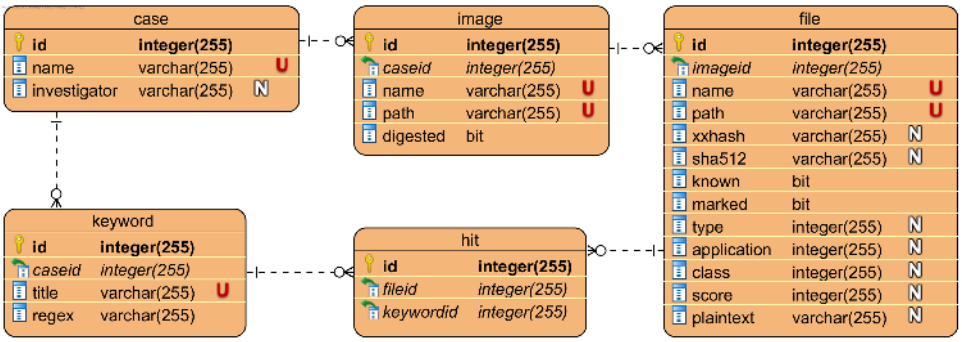


Figure 6.1: Entity-relationship diagram.

Tables 6.1, 6.2, 6.3, 6.4, and 6.5 list the attributes of the particular entities of the database together with their description.

Attribute name	Domain	Constraints	Description
case_id	integer	primary key	A numeric identifier of the case.
name	character varying(255)	not null, unique	Name of the case.
investigator	character varying(255)	-	Name of the investigator/expert witness who is working on the case.

Table 6.1: Case.

The database makes use of enumerated types. Table 6.6 provides an overview of declared types together with their description and their values.

Attribute name	Domain	Constraints	Description
image_id	integer	primary key	A numeric identifier of the image.
case_id	integer	foreign key	A numeric identifier of the case which the image belongs to.
name	character varying(255)	not null, unique	Name of the image.
path	character varying(255)	not null, unique	Path to the image.
digested	boolean	-	Information whether the image was already digested – whether the files were extracted from the image.

Table 6.2: Image.

6.2 Keyword Search

The way how ITFT v1.0 performs keyword search is as follows. On the input, there is a list containing the keywords that are to be searched for and a directory with the files to be browsed. At the output, for each keyword there is a text file created that contains paths to all the files that contain the particular keyword. Also, a directory with all the files with keyword hits sorted by images they belong to in the first place and file types (e.g., pdf, doc) in the second place.

Now let us get back to the keyword list and its format. The keywords are listed in `keywords.conf` file. For each keyword, there is a unique label, the text and a regular expression to be searched for. The regular expressions are not case sensitive and take into account both diacritic and diacritic free versions of the words. Due to inflection in the Czech language, the regular expressions contain only the roots of the words in order to cover all grammatical cases.

6.2.1 Keyword search in PostgreSQL

As described in Section 6.1, ITFT in the version v2.0 uses a database for storing data related to cases. Therefore, the keyword search can be performed over the database. This chapter provides an overview

Attribute name	Domain	Constraints	Description
file_id	integer	primary key	A numeric identifier of the file.
image_id	integer	foreign key	A numeric identifier of the image which the file was detected in.
name	character varying(255)	not null, unique	Name of the file.
path	character varying(255)	not null, unique	Path to the file.
xxhash	character varying(255)	-	A 64-bit xxHash value of the file. Used as an identifier for fast manual recognition of identical files.
sha512	character varying(255)	-	A SHA-512 value of the file.
known	boolean	-	Information whether the file is a well-known file (e.g. a system file). Keyword search is not performed for such files. The function *horizon* is used for determining whether a file is well-known or not.
marked	boolean	-	Information whether the file was selected by an investigator to be included in the expert testimony.
type	enum	-	Type (format) of the file. See file_type enumerated type in Table 6.6.
application	enum	-	Application the file was created by. See file_application enumerated type in Table 6.6.
class	enum	-	Class which the file belongs to. See file_class enumerated type in Table 6.6.
plaintext	text	-	Plaintext (printable characters) extracted from the file.

Table 6.3: File.

Attribute name	Domain	Constraints	Description
file_id	integer	primary key	A numeric identifier of the file.
case_id	integer	foreign key	A numeric identifier of the case which the keyword belongs to.
name	character varying(255)	not null, unique	Name of the file.
regex	text	not null	Regular expression to be matched with.

Table 6.4: Keyword.

Attribute name	Domain	Constraints	Description
hit_id	integer	primary key	A numeric identifier of the file.
keyword_id	integer	foreign key	A numeric identifier of the case which the hit relates to.
file_id	integer	foreign key	A numeric identifier of the file in which the keyword was found.

Table 6.5: Hit.

of PostgreSQL features for pattern matching and full text search, it describes how they work, and it introduces the ways how they can be optimized using indexes[1]. The different approaches are compared in terms of speed and then, based on the needs of expert witnesses, the most suitable approach is chosen and implemented into ITFT v2.0.

6.2.2 Pattern matching

PostgreSQL provides several approaches to pattern matching: LIKE operator, SIMILAR TO operator, and POSIX-style regular expressions.

LIKE. The LIKE expression (~~) returns true if the string matches the supplied pattern. In order to make the match case insensitive,

1. The description of pattern matching, full text search, indexes, and pg_trgm module is based on the official PostgreSQL documentation at [38].

Enumerated type	Set of values	Description
file_type	'sqlite', 'pst', 'dbx', 'mdb', 'vcard', 'vfat', 'doc', 'docx', 'xls', 'xlsx', 'zip', 'rar', 'pdf', 'cdf', 'txt', 'unknown', 'gif', 'application/octet-stream'	File type is the format of the file based on the magic number.
file_class	'image', 'archive', 'plaintext', 'mail-archive', 'document', 'contacts', 'database', 'unknown'	File class represents a group of file types that have some common properties. Files that belong to the same class are processed in a similar way.
file_application	'skype', 'outlook', 'outlook-express', 'pohoda', 'word', 'excel', 'access', 'unknown', 'unspecified', 'image'	This type labels application specific files, e.g. an SQLite database file containing skype conversations.

Table 6.6: Enumerated Types.

ILIKE operator (~~*) can be used. PostgreSQL provides also NOT LIKE (!~~) and NOT ILIKE (!~~*) expressions

```
string LIKE pattern
```

The pattern can contain special characters – an underscore (_) and a percent sign (%). An underscore represents an arbitrary character and a percent sign matches any sequence of zero or more characters. The percent sign is of a great importance to the keyword search, as the keyword can be preceded and followed by any sequence of characters. Therefore, the search pattern will typically have the form "%keyword%". The complete query to get a list of all file IDs that contain a specific keyword looks as follows:

```
SELECT file_id FROM file
    WHERE plaintext LIKE "%keyword%";
```

SIMILAR TO. As the LIKE expression, the SIMILAR TO operator returns true or false depending on whether the string matches the supplied pattern or not. It uses an underscore (_) and a percent sign (%) to represent an arbitrary character or any string, respectively.

In addition to these, SIMILAR TO utilizes a group metacharacters used by regular expressions. These metacharacters can for example be used for alternation (use either of two alternatives) or for repetition of the previous item a specified number of times. Thanks to the metacharacters SIMILAR TO it is more powerful than LIKE, however, the extended functionality degrades the performance.

The syntax of SIMILAR TO is as follows:

```
string SIMILAR TO pattern
```

POSIX regular expressions. Regular expressions, as defined in POSIX 1003.2, represent a concise and flexible means for pattern matching. They are more powerful than LIKE and SIMILAR TO operators.

A string is said to match a regular expression if it is a member of the regular set (a set of strings) that is represented by the expression. As opposed to LIKE and SIMILAR TO, a regular expression returns true not only if the string matches the expression as a whole, but also when its substring matches it.

```
'abc' ~ 'abc'    true
'abcd' ~ 'abc'   true
```

PostgreSQL provides the following operators for regular expressions:

- ~ for a case sensitive match,
- ~* for a case insensitive match,
- !~ for a negation of a case sensitive match,
- !~* for a negation of a case insensitive match.

Regular expressions consist of *branches* that are separated by a vertical bar (|). A string matches a regular expression if it matches any of its branches. Branches are constructed of *atoms* and *constraints*. Atoms can contain alphanumeric characters to be matched, non-alphanumeric characters to be matched (need to be preceded by a backslash), a dot (.) that matches any single character, special escape characters, and bracket expressions. Atoms can be followed by *quantifiers* that specify the length of the sequence of the matches of the atom.

More detailed information on regular expressions use in PostgreSQL can be found in the PostgreSQL documentation [39].

6.2.3 Full text search

Full text search provides identification of *documents* that match a given *query*. A *document* is a unit of searching. In the context of the keyword search in ITFT v2.0 it corresponds to the plaintexts extracted from files detected in the images. A *query* represents the search pattern.

The main difference between full text search and the pattern matching operators described above is, that the full text search has linguistic support and recognizes also derived words with respect to the configured language. Moreover, it provides an option to order the results based on their similarity to the query.

Before the full text search is performed, the documents are preprocessed in the following way: firstly, the documents are parsed into tokens, secondly, the tokens are converted into lexemes, and thirdly, the preprocessed documents are stored in a form that is optimized for searching.

Full text search in PostgreSQL is conducted using the operator @@. A sample full text search query looks as follows:

```
SELECT 'to be or not to be'::tsvector
@@ 'to & be'::tsquery;
```

As can be seen from the example above, the document is of the type `tsvector` and the query is of the type `tsquery`. The `tsvector` type is used for storing preprocessed documents and `tsquery` is used for preprocessed queries, respectively. PostgreSQL provides functions `to_tsvector` and `to_tsquery` to convert text documents and queries to `tsvector` and `tsquery` types.

Configuration. On top of a simple text search, as demonstrated in the example, the full text search provides a number of enhancements to document processing, such as skipping indexing of certain words, processing synonyms and whole phrases, and more. These options can be set in the `postgresql.conf` configuration file, using an `ALTER DATABASE` query, or set `default_text_search_config` for each session.

The configuration options also allow for use of different languages for the search instead of English, which is the default language. This is essential for the keyword search that is performed when processing

a forensic task assigned by the Police of the Czech Republic. The processed documents and the predefined keywords to be searched for are typically in the Czech language, or in some cases also in English. Czech is not supported by PostgreSQL, however, a Docker image with Czech full text search dictionaries is available [40].

Indexing in full text search. In order to speed up full text searches, indexing can be used. The full text search in PostgreSQL supports two types of indexes: Generalized Inverted Index (GIN) and Generalized Search Tree (GiST). GIN contains an index entry for each word together with a compressed list of matching locations. GiST indexes are lossy, as they may produce false matches. Therefore, additional checks need to be done and it causes degradation of the performance.

The following examples show how GIN and GiST indexes can be created. `file_gin_index` and `file_gist_index` are names of the indexes, `file` is the table, and `plaintext_tsvector` is the column on which the indexes are created. The column must be of `tsvector` type, alternatively, in the case of GiST indexes it can be also of `tsquery` type. In these examples `plaintext_tsvector` is a column that contains plaintexts extracted from files related to a case, that have been converted to `tsvector` type using `to_tsvector` function.

```
CREATE INDEX file_gin_index ON file
        USING GIN (plaintext_tsvector);

CREATE INDEX file_gist_index ON file
        USING GIST (plaintext_tsvector);
```

After creating an index, the full text search can be performed in the following way. Note that the search can be performed even without explicitly creating an index.

```
SELECT * FROM file
        WHERE plaintext_tsvector
                @@ to_tsquery(keyword);
```

Limitations. PostgreSQL's full text search has limitations to the length of lexemes, the length of a `tsvector`, the number of lexemes, and more. The length of a `tsvector` must be less than 1 megabyte and this may

be an issue in the searches performed within a typical forensic task. This issue will be discussed in more detail in Section 6.2.6.

6.2.4 Indexes

Indexes are a commonly used method for enhancing database performance. See the following example of a query.

```
SELECT name FROM file
    WHERE plaintext LIKE '%keyword%';
```

With no preprocessing, the system would need to scan the entire `file` table row by row, to find matches of the pattern. However, after creating an index on the `plaintext` column a more efficient method for finding the matching rows can be used, for example, walk several levels into a search tree.

Indexes are created in the following way:

```
CREATE INDEX index_name
    ON table_name
    (column_name [operator_class_name]);
```

When creating an index (`index_name`) on a particular column (`column_name`), an operator class (`operator_class_name`) can be specified. The operator class determines what operators can be used by the index for that column.

Index types. PostgreSQL supports several types of indexes. Here is an overview of the types together with a discussion whether they would be suitable for speeding up the keyword search that is performed when solving a typical forensic task assigned by the police.

- **B-tree index:** The B-tree index can be used for equality and range queries on data that can be ordered. It can also be used for pattern matching queries with `LIKE` and `~` operator. However, it works only for constant search patterns that are anchored to the beginning of the string, e.g. `LIKE 'keyword%'`. For the purpose of solving typical forensic tasks, a wildcard search is needed (`LIKE '%keyword%'`). Therefore, B-trees are not suitable for the task as they do not support wildcards.

- **Hash index:** Hash indexes can be used only for simple equality comparisons, therefore, they are not sufficient for typical forensic tasks.
- **Generalized Inverted Index (GIN):** GIN indexes are inverted indexes and they are suitable for cases where items that are to be indexed are composite values. For example, they can be used for finding strings that contain a specific substring, which is exactly what is needed for the keyword search in typical forensic tasks. GIN can use many indexing strategies and besides the operator classes that are a part of the standard PostgreSQL distribution, it allows for the development of custom operator classes. Many GIN operator classes are included in the `contrib` directory of the PostgreSQL distribution and many are available as separate projects.
- **Generalized Search Tree Index (GiST):** GiST is a tree-structured access method. It works as a base template for which different indexing strategies can be implemented. As with GIN, custom operator classes can be developed for GiST as well. In the `contrib` directory, there is a module called `pg_trgm` which provides both GiST and GIN operator classes for working with strings. The description of the module is provided in the following section.

6.2.5 pg_trgm module

The `pg_trgm` module is used to determine the similarity of alphanumeric text. It provides functions and operators for searching for similar strings together with operator classes for speeding up string search.

The module is based on trigram matching. A trigram is a sequence of three successive characters from a string. Similarity of two strings can be measured in such a way that the number of trigrams that they share is computed. This approach is a very powerful and efficient for measuring similarity of words in natural languages.

The `pg_trgm` module provides a number of functions and operators for determining similarity of strings and to set a similarity threshold (the minimum similarity between two strings for them to be considered similar, e.g. to be misspellings of each other). For solving a typical forensic task, it would be beneficial to be able to detect also

misspellings of the keywords that are searched for. However, in this task we would compare a keyword to the whole plaintext extracted from a file instead of a potentially misspelled keyword which would typically be only a substring of the plaintext. Therefore, the `pg_trgm` module is not usable for the task in this way.

Indexing. Besides similarity functions and operators, the `pg_trgm` module also provides a support for indexes. There are GiST and GIN operator classes available. They can be used for creating indexes that can subsequently be used for very fast similar searches and also for trigram-based index searches for LIKE, ILIKE, ~ and ~* operators. The trigram-based indexes could be used to speed up keyword search using these operators. The indexes for keyword search are created as follows:

```
CREATE INDEX trgm_gist_index ON file
        USING GIST (plaintext gist_trgm_ops);

CREATE INDEX trgm_gin_index ON file
        USING GIN (plaintext gin_trgm_ops);
```

6.2.6 Keyword search method choice

This section discusses usability of the above described approaches to pattern matching, full text search, and indexing, for handling keyword search in typical forensic tasks. It justifies the choice of pattern matching rather than full text search and it chooses a suitable indexing method. At the end, it provides a comparison of the speed of different pattern matching operators with and without using indexing.

Pattern matching vs full text search. The full text search has a significant advantage over the pattern matching, as it provides linguistic support and it can detect derived words with respect to a configured language. However, there is an issue with the size of `tsvector`, as has already been outlined in Section 6.2.3. The `tsvector` size is limited to 1 megabyte. Within a set of files that are related to a particular forensic task it is likely to happen that some of the `tsvector` values generated from their plaintexts would exceed the limit.

```
UPDATE file SET plaintext_tsvector =
        to_tsvector('english', plaintext);

ERROR:  string is too long for tsvector
        (1215888 bytes, max 1048575 bytes)
```

An option would be to split the files that would exceed the limit into parts and analyze the parts one by one. Another option would be to omit the full text search on the large files and to process them, for example, by using the simple pattern matching. However, there is no reasonable way how to predict the size of a `tsvector` based on the original size of the file so that it would be possible to set the maximum file size. There is an approximation of the `tsvector` size that is based on the number of unique words in the file, the average word size, and the word count. Nevertheless, finding out the values of these parameters that are needed for computation of the approximation would consume too much time.

Another workaround would be to remove the 1 MB limit. In 2017, a suggestion for a patch that would remove it appeared in PostgreSQL mailing list [41], but at the time of writing the thesis the patch has not been issued yet.

Since there is currently no reasonable workaround for the `tsvector` size limit, the full text search is not a good candidate for processing a typical forensic task. The pattern matching will be used instead. However, if the patch is issued later on, switching to the full text search should be considered.

Indexing. PostgreSQL provides several types of indexes that can be used for different operator classes. Out of the supported index types the B-tree, GIN, and GiST indexes are considerable for the use in keyword search. The table 6.7 summarizes, which operator classes for textual data types, are these three index types usable with.

The default operator class for the text data type is available only for the B-tree index, not for the GIN and GiST indexes.
The `text_pattern_ops` class is also supported only for the B-tree indexes. Unluckily, for both these operator classes some of the files related to forensic tasks exceed the maximum index size. Based on the size of a file it is difficult to estimate the size of the index. A solu-

Operator class	B-tree index	GIN index	GiST index
default operator class (data type text)	exceeded maximum index size	no default operator class	no default operator class
text_pattern_ops	exceeded maximum index size	not supported	not supported
pg_trgm	not supported	OK	exceeded maximum index size

Table 6.7: Indexing in the keyword search.

tion suggested in the PostgreSQL documentation and by PostgreSQL users in different discussion fora [42][43] is to hash the value on which the index is to be created and then to create the index on the fixed hash instead. However, by using the hash, the resulting index can be used only for simple equality checks and not for searching for substring matches. Therefore, the B-tree index is not a good candidate to be used in the keyword search.

```
CREATE INDEX text_bt_idx
       ON file
       USING btree
       (plaintext text_pattern_ops);

ERROR: index row requires 8848 bytes,
       maximum size is 8191
```

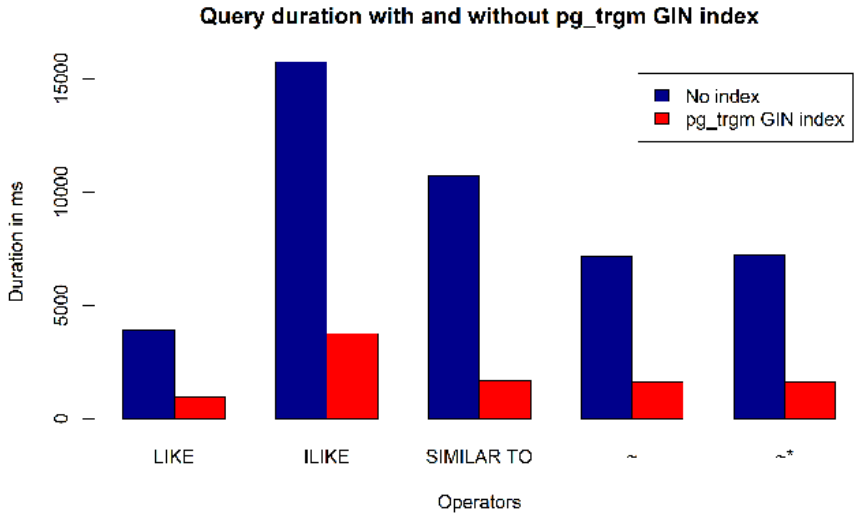
The `pg_trgm` module provides operator classes for GIN and GiST indexes. For the GiST index the same problem occurs as with the B-tree index. The `CREATE INDEX` query ends up with an error saying that an index row exceeded the maximum size. Only the GIN index could be created successfully and so it will be used for the keyword search.

Performance test. Figure 6.2 displays the time needed for processing different pattern matching queries using either no index or the `pg_trgm` GIN index. The test was performed on a table containing 49,460 entries, which correspond to files extracted from images. The queries in the following form were executed during the test. `OPERATOR` stands

for the LIKE, ILIKE, SIMILAR TO, ~, and ~* operators, respectively. The values used for the graph are in a table in A.2.

```
SELECT count(*) FROM file
        WHERE plaintext OPERATOR '%onion%';
```

Figure 6.2: Pattern matching – Performance.



As can be seen from the graph, the search using the `pg_trgm` GIN index is significantly faster for all the tested operators. Nevertheless, it should be noted that the index creation takes a non-negligible amount of time. In this particular test the duration of the `CREATE INDEX` query was 146176.687 ms, which is almost 10 times more than the duration of the most time consuming `SIMILAR TO` query. No matter what search operator is used, it is clear that if a single keyword is searched for, it is not efficient to use indexing, as the sum of the duration of the index creation and the search query itself exceeds the duration of the search query without indexing. The dilemma at this point is, whether it is efficient to use indexing in the case of a typical forensic task where the search query is executed multiple times for different keywords. Typically, tens of keywords are searched for.

Figure 6.3: Pattern matching – Performance (40 keywords).

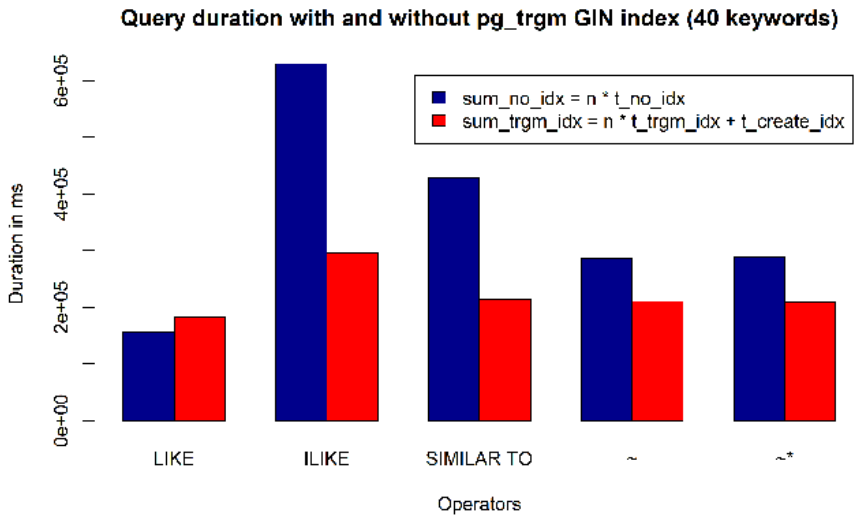


Figure 6.3 compares the total durations sum_{no_idx} and sum_{trgm_idx} of search queries for 40 keywords with and without using the `pg_trgm` index. sum_{no_idx} and sum_{trgm_idx} are defined as

$$sum_{no\_idx} = n \times t_{no\_idx}$$
$$sum_{trgm\_idx} = n \times t_{trgm\_idx} + t_{create\_idx},$$

where n is the number of keywords, t_{no_idx} is the duration of a search query without using indexing, t_{trgm_idx} is the duration of a search query with the use of `pg_trgm` index, and t_{create_idx} is the duration of the `CREATE INDEX` query.

For most of the tested operators, it can be seen that for 40 keywords it is more efficient to perform the search using indexing. In order to optimize the keyword search in ITFT v2.0, indexing should be used only when the number of keywords exceeds a threshold max_n, which can be defined as the maximum number of keywords for which $sum_{no\_idx} < sum_{trgm\_idx}$.

Choice of pattern matching operator. This section is devoted to a choice of a suitable pattern matching operator. The criteria to be

considered are the supported functionality and the performance of the search with and without indexing. Based on the performance, the `LIKE` operator seems to be the best candidate (see Figure 6.3). However, it is case sensitive and for the typical forensic task a case insensitive search is preferred. The two operators that support the case insensitive search are `ILIKE` and `~*` (regular expression). Out of these two, the `~*` operator has a better performance and it is also more powerful than `ILIKE`. Therefore, it will be used for the keyword search.

As already discussed in the section above, pattern matching is to be performed using indexing only when the number of keywords exceeds the threshold max_n. The threshold needs to satisfy $sum_{no\_idx} < sum_{trgm\_idx}$. Taking the sum_{no_idx} and sum_{trgm_idx} values from the conducted performance test, the threshold value for the `~*` operator is $max_n = 26$.

6.2.7 Implementation

The keyword search is performed as a part of the `exportkeywords()` function, which is executed after issuing the `exportkeywords` command. The function operates as follows:

1. The keywords that are defined in the configuration file `keywords.conf` are added to the database to the table `keyword`.

2. If the number of the keywords exceeds the defined threshold, the `pg_trgm` GIN index is created.

   ```
   CREATE INDEX trgm_gin_idx ON file
           USING gin (plaintext gin_trgm_ops);
   ```

3. Select IDs and regular expressions (`keyword_id`, `regex`) for all keywords in the `keyword` table.

   ```
   SELECT keyword_id, regex FROM keyword;
   ```

4. For each file ID do the following:

   - Select IDs of all files (`file_id`) that match the regular expression.

```
SELECT file_id FROM file
        WHERE plaintext ~* '$regex';
```

- For each file ID add a new keyword hit into the `hit` table.

```
INSERT INTO hit (hit_id, keyword_id,
        file_id)
        VALUES ('$hit_id', '$keyword_id',
        '$file_id');
```

6.2.8 Performance test

This section describes a performance test which compares the performance of the keyword search in ITFT v1.0 and ITFT v2.0. We describe the methodology, the sample data, and the results of the test.

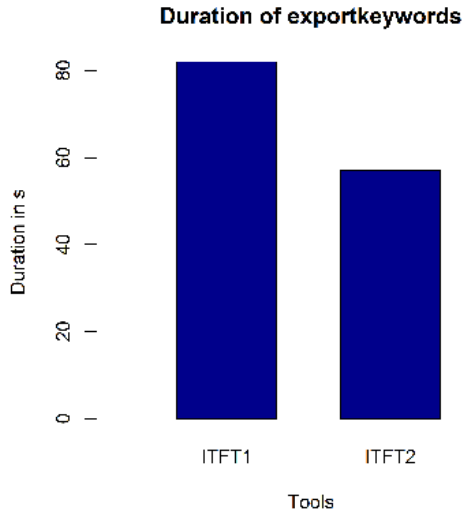
Methodology. In the test we measured the duration of the command `exportkeywords` for both tools. The test was performed in the following steps:

1. Add a new case (`addcase`).
2. Add a new piece of evidence (`addevidence`).
3. Extract all files from the piece of evidence (`digest`).
4. Create a keyword list (`keywords.conf`).
5. Perform the keyword search (`exportkeywords`[2]).

Hardware and sample data. The test was run on the hardware described in Section 4.3. We analyzed a case containing a disk image the size of 7.9 GB in the test. We were looking for occurrences of 11 keywords. As the number of keywords did not exceed the threshold $max_n = 26$, the `pg_trgm` GIN index was not created.

2. For ITFT v2.0 run the `exportkeywords` command with an option `-noocr`, which disables Optical Character Recognition (OCR). OCR is computationally demanding and it would influence the duration of the command.

Figure 6.4: Keyword search performance.



Results. Figure 6.4 displays the duration of the `exportkeywords` command for both ITFT v1.0 and ITFT v2.0. It can be seen that the command requires less time when it is run with ITFT v2.0. The difference between the durations in ITFT v1.0 and ITFT v2.0 would be even more significant for a number of keywords larger than $max_n = 26$. In such a case, the `pg_trgm` GIN index would be created and the time required for the search of a single keyword would decrease.

6.3 Optical Character Recognition

This chapter discusses the possibilities of integrating Optical Character Recognition (OCR) into ITFT v2.0. There are several open source tools that provide OCR functionality and the aim of this chapter is to compare them in terms of speed, file formats and languages they support, and how successful they are in recognizing a text in different types of images.

The process of finding a suitable tool is done in the following steps:

1. Find several OCR tools for the analysis.

2. Define criteria based on which the tools will be compared.
3. Create a set of sample images that will be processed using the tools.
4. Automatically process all the sample images using the chosen tools.
5. Manually analyze outputs of the tools.
6. Based on the results pick the most suitable tool.

6.3.1 Tools overview

The first step is to find several OCR tools that will be subsequently compared. It is desired that the tools for the analysis work on Linux platforms, they are runnable from the command line, and, preferably, are open source. There are multiple articles, reviews, and fora on the web that discuss OCR tools [44] [45] [46] [47], describe their properties, platforms, and languages they support, and, last but not least, their accuracy. Based on the information provided by the reviews the tools that seem to be the most promising ones for the use by expert witnesses are the following: Tesseract, GOCR, CuneiForm, and Ocrad.

Tesseract. Tesseract is an open source OCR engine available under the Apache 2.0 license [48]. It works on Linux, Windows, MacOS, and can be compiled for a variety of other targets, including Android and the iPhone. Tesseract is a command line program, however, there are third-party frontends that use it as an OCR engine. It supports tens of languages, including the Czech. In order to be able to use a non-standard language, one needs to simply install training data package for the particular language. The tool can read various file formats, including jpeg, png, and pdf, that are most likely to appear in the data carriers that are to be analyzed. A great plus of Tesseract is that it is still being maintained and updated and it has a thorough documentation.

GOCR. GOCR is an OCR program developed under the GNU Public License [49]. As with Tesseract, GOCR can be used with different

front-ends, which makes it very easy to port to different operating systems and architectures. It supports multiple file formats [50], such as png, jpg, jpeg, tiff, and pdf[3]. It can read text in different encoding schemes, including UTF-8 that covers also characters with diacritic which is crucial for being able to extract text in the Czech language. GOCR is documented in Linux manual pages [50] and its latest release GOCR 0.51 is dated May 2017.

CuneiForm. CuneiForm is an OCR system originally developed and open sourced by Cognitive technologies [51]. It is available under the Simplified BSD License. It does not have its own GUI but it can be successfully run from the OCRFeeder graphical interface [44]. It can process jpeg, png, pdf, and all other formats that GraphicsMagick knows how to open [52]. When it comes to languages, CuneiForm recognizes English by default, however, it supports many other languages, including the Czech. The tool is still being updated and the documentation is available on Ubuntu manual pages [53].

Ocrad. GNU Ocrad is an OCR program developed under GNU General Public License. It can be run on Linux operating systems and can be used as a stand-alone console application, or as a backend to other programs. It reads images in pbm (bitmap), pgm (greyscale) or ppm (color) formats and produces a text in byte (8-bit) or UTF-8 formats. In order to be able to process jpeg, png and pdf files using Ocrad, conversion to one of the supported formats needs to be applied first. The latest version of the tool is 0.26 released in March 2017 and is thoroughly documented.

6.3.2 Comparison criteria

This chapter defines the criteria that will be applied to compare the tools with respect to utilization by expert witnesses. Subsequently, based on this comparison the most suitable tool will be chosen.

1. **Output usability:** The output of OCR tools is to be subsequently processed by an automated keyword search. In order to be able

3. The pdf format is not listed in the manual pages, however, the tool accepts it and can extract text from it.

to detect the specified keywords, the output needs to correspond to the text in the original image. We want to compare the tools in the sense how successful the tools are in extracting the text correctly and whether their outputs are usable for further analysis.

2. **Speed:** An expert witness is given a specific amount of time for analyzing the data carriers, therefore, it is essential that the tool is fast enough to perform the analysis within the given time frame.

3. **Supported languages and character encoding:** For the use by the Police of the Czech Republic it is necessary that the tool is able to read Czech words and characters with diacritics. The tool should therefore support the Czech or at least a character encoding that contains characters with diacritics (e.g., UTF-8).

4. **Supported formats:** Images that are extracted from the evidence data carriers can be of various formats. In order to be able to process the image using the particular tool without the need of conversion to another format, it is preferable that the tool supports various formats, at least the most common ones, such as jpeg, png, or pdf.

5. **License:** If a suitable tool is found it is likely to be integrated into an existing proprietary software that is currently used by expert witnesses working for the Police of the Czech Republic. For this reason, open source tools issued under a free software license are preferred.

6. **Supported platforms:** It is preferred that the tool can be run on Linux due to the same reason described above when discussing the preferred licensing.

6.3.3 Sample data

This section describes the set of sample images that are to be processed by the tools. The set contains 100 images and in order to simulate the diversity of real evidence data the images are of various types and formats.

Types of pictures

- Scanned documents.
- Photographed typed text (e.g. photographed documents, titles).
- Images with text blocks (e.g. a logo with a title).
- Photographed handwritten text.

Formats. The evidence data can contain images in different formats. The formats that are the most common ones and are likely to contain some text that could be extracted using OCR are jpeg, png, and pdf. All these three are included in the sample set.

6.3.4 Tools comparison

This chapter is devoted to comparison of the chosen tools based on the predefined criteria. The outputs of the tools processing the prepared set of images are evaluated and the most suitable tool is chosen.

Output usability. In order to be able to enumerate usability of output of the tools, a score from 0 to 3 is assigned to each output text based on whether it is readable and whether it corresponds to the text that can be seen in the original image.

- **Score 3:** The output text is readable without any issues (few spelling mistakes tolerated) and corresponds to the original image.
- **Score 2:** The output text contains multiple misspellings and incorrectly interpreted characters, however, is still readable by a human and mostly corresponds to the original image.
- **Score 1:** The output text seems to be derived from a meaningful text, however, is hardly readable by a human or a significant amount of text from the original image is omitted.
- **Score 0:** The output text is a meaningless sequence of characters that does not correspond to the original image at all.

Figure 6.5: Usability of output – Overall score.

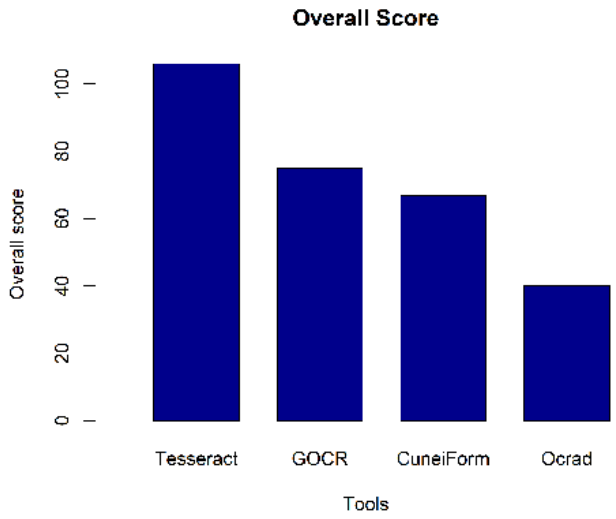


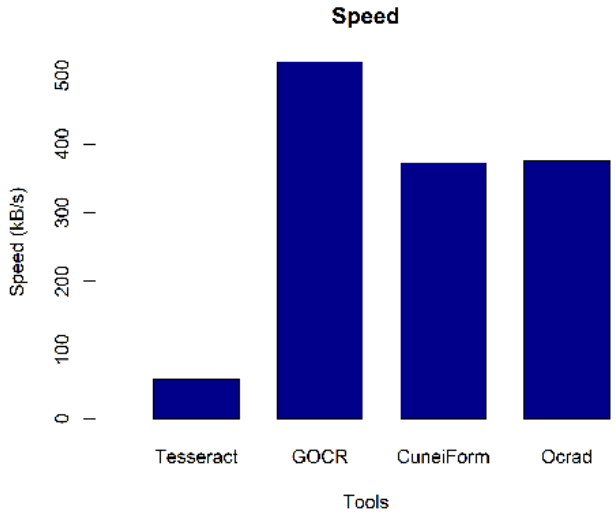
Figure 6.5 plots the sums of scores for 100 images of the sample set for each of the tools. Results with the highest overall score are produced by Tesseract meaning that the output texts it produced were more accurate than the texts produced by other tools.

Speed. Figure 6.6 displays the speed of each of the tools in kilobytes per second. The speed is calculated as a proportion of the total size of the sample set and the time spent with processing the sample set using the particular tool. The speed test was performed on the following hardware:

- Model: DELL PowerEdge 2950.
- Processor: 2x quad core Intel(R) Xeon(R) X5460, 3.16 GHz, L2 cache 12 MB.
- Chipset: Intel 5000X.
- RAM: 8 x 4096 MB, DDR2 FB-DIMM, 667 MHz.

Out of the tested tools, the fastest one is GOCR with the speed 519.9 kB/s, followed by Ocrad with 375.7 kB/s, CuneiForm with 372.5 kB/s, and the last one is Tesseract with the speed of only 57.6 kB/s.

Figure 6.6: Speed.



Supported languages and character encoding. As was already mentioned, we aim to be able to recognize the text in the Czech language. All of the tools support UTF-8 character encoding and therefore can read characters with diacritics, which is essential for the Czech language. However, only Tesseract and CuneiForm provide direct support for multiple languages (including the Czech) and, based on training data in the particular languages, they can perform linguistic analysis that increases the accuracy of the extraction.

Tool	UTF-8 encoding	Czech language
Tesseract	✓	✓
GOCR	✓	
CuneiForm	✓	✓
Ocrad	✓	

Table 6.8: Supported languages and character encoding.

Tool	jpeg	png	pdf
Tesseract	✓	✓	✓
GOCR	✓	✓	✓
CuneiForm	✓	✓	✓
Ocrad			

Table 6.9: Supported formats.

Tool	Apache License	GPL	BSD License
Tesseract	✓		
GOCR		✓	
CuneiForm			✓
Ocrad		✓	

Table 6.10: License.

Supported Formats. As already stated, we focus on jpeg, png, and pdf formats. Therefore, we prefer to use a tool that supports all of them in order to avoid additional format conversion. Table 6.9 summarizes which of the three formats are supported by each of the tools.

License. Table 6.10 summarizes the licenses under which the chosen tools are developed. All of the listed licenses are free. Apache License 2.0 and BSD License under which Tesseract and Cuneiform are available are permissive software licenses meaning they have minimal requirements about how the software can be redistributed. GOCR and CuneiForm are available under GNU General Public License (GPL), which is a copyleft license. The copyleft license is more restrictive than a permissive license in a sense that when the software is being redistributed, the copyleft license enforces the publication of the source code to be issued under copyleft as well [54]. Due to the fact that the tool that will turn out to be the most suitable one will be likely to be integrated into an existing proprietary software, a permissive software license is more preferable.

Tool	Linux	Windows	Mac OS
Tesseract	✓	✓	✓
GOCR	✓	✓	
CuneiForm	✓	✓	
Ocrad	✓		

Table 6.11: Supported platforms.

Supported platforms. For each of the tools, Table 6.11 shows which platforms there are installation packages or binaries available for. Additionally, there are source codes available for all of the tools. Therefore, they can be compiled for other platforms in addition to those that are listed in the table.

Tool choice. The most important criterion for the choice of the tool is the output usability. Based on the analysis, Tesseract appears to be the best candidate thanks to accuracy of its output being significantly higher than the rest of the tools. Although it is significantly slower than the other tools, it is still feasible in the amount of time an expert witness is generally given for analyzing the data carriers. It provides support for the Czech language, it is issued under a permissive software license, and it can be run on Linux operating systems. It matches all the requirements and in spite of the deficiency in speed of processing, it is clearly the most suitable tool for solving a typical forensic task assigned by the Police of the Czech Republic. Therefore, it is to be used by ITFT v2.0.

6.3.5 Implementation

Tesseract is integrated into ITFT v2.0 in such a way, that before adding a new file of jpeg, png, or pdf (not text) into the database, Tesseract is executed on the file and the output is filled into the *plaintext* parameter.

6.4 Automated Reporting

As it was already mentioned in Section 5.2, the creation of the pdf report – the expert testimony – needs to be done manually. For each file that is to be included in the report, ITFT v1.0 generates a text file containing a summary of the file metadata. The expert witness has to manually merge all the selected files together with their metadata summaries to create the pdf report.

The goal of this chapter is to propose and implement a method for partial automation of the pdf report creation process. For a number of most frequent file formats, suitable tools for conversion to pdf are chosen. Also, the means for automated merging of the converted pdf files together with the metadata summaries into a complete report are introduced.

6.4.1 Limitations

It is not possible to automate the report creation process completely and a manual intervention of an expert witness could be necessary in some cases.

For many file formats, it is possible to convert them automatically to pdf using various pdf converters. However, for some of the formats there are either no converters available, or sometimes the layout of the created pdf files is not satisfactory (e.g., tables from an xls file may not be displayed nicely).

A reasonable workaround to this problem is to give the users an option to manipulate the pdf files that are to be included in the report, if necessary. The proposed solution will be discussed in more detail in the following section.

6.4.2 Report creation process

This section describes the individual steps of the enhanced report creation process in ITFT v2.0.

1. Run the function `preexport(dir)`, where `dir` is the directory containing copies of the files that are to be included in the report. The directory is provided by the police. The function converts all

the files that are in the supported file formats to pdf. It creates a new directory called `pdf`, where it puts the created pdf files.

2. In this step, the expert witness reviews the files in the `pdf` directory. If there are some files missing, i.e., some of the original files were in formats that are not supported by ITFT v2.0 (see 6.4.3), he creates the corresponding pdf files manually and saves them to `pdf`. Also, if he finds any of the pdf files created by ITFT v2.0 not satisfactory, he can manually create his own pdf files with a more suitable layout and replace them.

3. After the expert witness reviews the files in `pdf` and makes any changes, if necessary, the function `export()` can be run. The function takes the created pdf files from `pdf`, it appends a metadata summary to each of them, and finally, it merges all the pdf files together with the metadata summaries into a single pdf report (`report.pdf`), which is saved to a `report` directory.

6.4.3 Supported file formats

The automated reporting utility focuses on a set of file formats that tend to appear in the reports most often. Here is a list of the selected file formats:

- **Documents:** doc, docx, odt, pdf.
- **Presentations:** ppt, pptx.
- **Spreadsheets:** xls, xlsx.
- **Raster graphics:** jpeg, png, tiff.
- **Text files:** txt.

6.4.4 Conversion to pdf

After deciding which file formats are to be supported, suitable tools for converting them to pdf need to be selected. Based on opinions and experience of users from various IT fora [55][56], a Linux command line utility `unoconv` seems to be a good candidate. According to its manual

page [57], it can convert any file format that LibreOffice can import, to any file format that LibreOffice is capable of exporting. Therefore, it can convert all the file formats that are required by the automated reporting utility in ITFT v2.0.

The `unoconv` utility was tested on a set of files containing samples of files in the formats listed in 6.4.3 and all the resulting pdf files corresponded to the original files which they were generated from. Therefore, it will be used in ITFT v2.0. Although the users in [55] claim that there are some limitations in the accuracy of the outputs[4], for the purpose of the automated reporting in ITFT v2.0 it can be considered acceptable. In case of occasional inaccuracies in the resulting pdf files, the user has an option to replace the problematic ones manually.

6.4.5 Implementation

After creating the pdf versions of all the selected files, the report itself can be generated. It will be done using a document preparation system LaTeX [58].

This section provides a brief description of LaTeX and outlines the steps how the report is created.

TeX and LaTeX. TeX is a typesetting computer program that takes a "plain" text file and converts it into a high-quality document for printing or on-screen viewing [59]. LaTeX is a macro system built on top of TeX that aims to simplify its use.

A LaTeX plain text file describes the document's structure and presentation [60]. It contains the source text combined with markup. It works in a similar way as HTML which uses markup to describe the structure of Web pages.

LaTeX can be utilized in the report creation process in such a way, that the `export()` function will automatically generate the plain text file which will subsequently be converted into the pdf report. The following steps present the whole process of the generation of the report including the description of its contents.

4. More specifically, `unoconv` turned out to have problems with rendering smart art from Microsoft Office correctly.

Report creation with LaTeX

1. Create a plain text file `report.tex` where the LaTeX code and the contents of the report will be gradually inserted as described in the following steps.
2. Insert a preamble of the document specifying the document class and the used packages. The document class defines the layout style of the document. In this case, the `report` document class will be used. After the preamble, other commands defining the properties of the document, such as the style of the headers and the footers, are inserted.
3. Set the title of the document ("An expert testimony") and its author (the name of the expert witness).
4. Start the body of the document using `\begin{document}`.
5. For each file that is to be included in the report do the following:
   (a) Specify the number of its copies present in the analyzed images.
   (b) For each copy insert a table with its metadata including its source, path, access and modification times, and other information.
   (c) Insert a table with MD5, SHA1, and SHA256 checksums.
   (d) Insert a table with the metadata of the file contents.
   (e) Insert the pdf file that has been created by a conversion from the original file.
6. End the body of the document using `\end{document}`.
7. Generate the document – the report – by running the following command:

```
pdflatex report.tex.
```

8. Find the report (`report.pdf`) in the current directory.

7 Future work

This chapter discusses ideas and suggestions on what improvements could be made to ITFT and what other features could be added to it in order to ease the work of expert witnesses.

First of all, ITFT is still a prototype. As it is growing fast and new features are added on regular bases, it is starting to be difficult to understand the structure of the program and to follow the dependencies between the functions. It also makes the process of integrating and testing new components harder. ITFT has reached the point where it should be transformed from a prototype into a proper tool. The current imperative Perl script should be rewritten into an object-oriented program with clearly defined entities and relationships between them.

ITFT does not address situations when execution of an action is interrupted for some reason, for example, when the system is rebooted. Currently, if a command is interrupted it has to be rerun again from the start. This is particularly adverse in the cases where files are being carved from images or when the keyword search is being performed. These operations that can take hours for images that have the size of several hundreds of gigabytes. To mitigate the delays caused by interruptions ITFT should regularly save its current state. After an interruption, it should be able to continue from where it stopped.

The keyword search is performed regardless of the language of the analyzed text. Besides detecting exact string matches, it would be useful if it could detect derived or misspelled words with respect to the language of the text. This could be achieved by using the full text search, but due to the limitation to the size of `tsvector` it can not currently be used for solving the typical forensic task. However, if the proposed patch that would remove the limit is issued (see 6.2.6), the full text search should be integrated to ITFT. For this reason, the current state of the patch development and approval process should be checked regularly.

8 Conclusion

The thesis studied forensic analysis and its use by expert witnesses that work for the Police of the Czech Republic. We described a typical forensic task as it is assigned by the police and we introduced several digital forensic tools that could be used for solving the task. Autopsy seemed to be a good candidate. However, when it was tested on a real case it turned out not to be usable due to multiple issues that occured (e.g., unexpected crashes and keyword search errors).

The rest of the thesis was devoted to the IT Forensic Tool that is being developed by the technical consultant of the thesis for the purpose of processing forensic tasks. We described the functionality of the tool together with the process of solving a typical task with its use. We also proposed a number of improvements that were subsequently integrated into a new version of the tool – IT Forensic Tool v2.0.

The first improvement was integration of the PostgreSQL database. The goal was to simplify the way how the tool stores the information about forensic tasks. Instead of using multiple configuration files, most of the information is now stored into the database. It consists of five tables that represent the basic entities the forensic tasks work with – case, evidence, image, file, and keyword.

The second improvement was optimization of keyword search that is performed within forensic tasks. The purpose of keyword search is to find all files with occurrences of predefined keywords. We analyzed PostgreSQL features for pattern matching and full text search and the options for their optimization using indexes. The pattern matching operators were compared in terms of speed and functionality and, as a result, regular expressions were chosen to be used in ITFT v2.0. Full text search turned out not to be usable for the task due to limits to the size of `tsvector`, which is a data type used for storing preprocessed strings. Analysis of indexing features that are available in PostgreSQL showed that the only suitable option is to use `pg_trgm` GIN indexes. Due to non-negligible duration of index creation, indexing is set to be used only when the number of keywords to be searched for exceeds a threshold value 26. To conclude the optimization of keyword search, our performance test showed

that keyword search over the database in ITFT v2.0 is faster than in the previous version of ITFT.

The third improvement was integration of Optical Character Recognition (OCR) in order to be able to analyze texts in images automatically. Based on a comparison of several OCR tools, Tesseract was selected as the most suitable one and it was integrated into ITFT v2.0.

The fourth improvement was reduction of the amount of manual work needed for creating an expert testimony – a pdf report. For a number of most common file formats, we managed to generate the pdf report automatically. Still, manual intervention of an expert witness is needed for other file formats and for validating the results of automatic conversions to pdf.

Besides the implemented improvements, we submit a number of suggestions for the future work. Namely, to rewrite the tool in an object-oriented way, to create means to handle unexpected interruptions, and to consider implementing full text search if a patch to remove the `tsvector` size limit is issued.

To sum up, the thesis assessed the nature of a typical forensic task assigned by the police and it meets the needs of expert witnesses by optimizing and extending the existing forensic tool ITFT.

Bibliography

[1] Forensicswiki.org. *Category: Forensics File Formats*. 2015. URL: http://www.forensicswiki.org/wiki/Category:Forensics_File_Formats (visited on 05/09/2018).

[2] Forensicswiki.org. *Raw Image Format*. 2014. URL: http://www.forensicswiki.org/wiki/Raw_Image_Format (visited on 05/09/2018).

[3] Guidance Software. *Guidance Software*. URL: https://www.guidancesoftware.com/ (visited on 05/09/2018).

[4] Sans.org. *Forensic Images: For Your Viewing Pleasure*. 2016. URL: https://www.sans.org/reading-room/whitepapers/forensics/forensic-images-viewing-pleasure-35447 (visited on 05/09/2018).

[5] Sans.org. *An Overview of Disk Imaging Tool in Computer Forensics*. 2001. URL: https://www.sans.org/reading-room/whitepapers/incident/overview-disk-imaging-tool-computer-forensics-643 (visited on 05/09/2018).

[6] *Forensic Focus*. URL: https://forensicfocus.com/ (visited on 05/09/2018).

[7] twebster01. *Autopsy 3: The Limitations*. 2014. URL: https://www.forensicfocus.com/Forums/viewtopic/t=11981/ (visited on 05/09/2018).

[8] Guidance Software. *EnCase Forensic*. URL: https://www.guidancesoftware.com/docs/default-source/document-library/product-brief/encase-forensic-product-overview.pdf?sfvrsn=6 (visited on 05/09/2018).

[9] Guidance Software. *What's new in EnCase Forensic 8*. URL: https://www.guidancesoftware.com/docs/default-source/document-library/product-brief/whats-new-in-encase-forensic_061417.pdf?sfvrsn=40 (visited on 04/01/2018).

[10] caine-live.net. *CAINE: Computer Forensics Linux Live Distro*. URL: http://www.caine-live.net/ (visited on 05/09/2018).

[11] caine-live.net. *CAINE Tools List*. URL: http://www.caine-live.net/page11/page11.html (visited on 05/09/2018).

[12] Dan Farmer and Wietse Venema. *Forensic Discovery*. 2004. URL: http://www.fish2.com/security/wf-book.pdf (visited on 05/09/2018).

[13] Brian Carrier. *The Coroner's Toolkit (TCT)*. URL: http://www.porcupine.org/forensics/tct.html (visited on 05/09/2018).
[14] Brian Carrier. *Open Source Digital Forensics*. URL: https://www.sleuthkit.org/ (visited on 05/09/2018).
[15] Brian Carrier. *Autopsy*. URL: https://www.sleuthkit.org/autopsy/ (visited on 05/09/2018).
[16] Guidance Software. *EnCase App Central*. 2013. URL: https://www.forensicmag.com/product-release/2013/03/encase-app-central (visited on 05/09/2018).
[17] Tutorialspoint.com. *Apache Solr - Overview*. URL: https://www.tutorialspoint.com/apache_solr/apache_solr_overview.htm (visited on 05/09/2018).
[18] Digi Forensics. *Autopsy Error*. 2015. URL: https://sourceforge.net/p/sleuthkit/mailman/message/33242031/ (visited on 05/09/2018).
[19] peachy189. *Error initializing keyword search module*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=2432 (visited on 05/09/2018).
[20] lucacalcaterra. *Indexing server port 23232 is not available*. 2013. URL: https://github.com/sleuthkit/autopsy/issues/317 (visited on 05/09/2018).
[21] wiki.sleuthkit.org. *Autopsy 3 Troubleshooting*. 2014. URL: https://wiki.sleuthkit.org/index.php?title=Autopsy_3_Troubleshooting (visited on 05/09/2018).
[22] steorra. *hanging at 77%*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=2486 (visited on 05/09/2018).
[23] Bunnysniper. *Complete Freeze*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=1581 (visited on 05/09/2018).
[24] DarkShadow316. *Program Crashes*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=2496 (visited on 05/09/2018).
[25] bbrezi57. *Autopsy ingest module crash*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=2609 (visited on 05/09/2018).
[26] edp05373. *Failure to open or create core*. 2018. URL: https://github.com/sleuthkit/autopsy/issues/3422 (visited on 05/09/2018).
[27] The Sleuthkit. *Autopsy Issues*. URL: https://github.com/sleuthkit/autopsy/issues (visited on 05/09/2018).

[28] Sleuthkit.org. *Autopsy Troubleshooting*. URL: https://forum.sleuthkit.org/viewforum.php?f=6&sid=275932b05c086a8bcc90361d761c265d (visited on 05/09/2018).

[29] volavka. *E01 Verifier fails to verify*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=5&t=2473 (visited on 05/09/2018).

[30] colino. *Keyword Search duplicate results*. 2015. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=2438 (visited on 05/09/2018).

[31] johnmccash. *Error opening the case*. 2014. URL: https://forum.sleuthkit.org/viewtopic.php?f=6&t=586 (visited on 05/09/2018).

[32] likangqi666. *Couldn't add data source*. 2016. URL: https://github.com/sleuthkit/autopsy/issues/2001 (visited on 05/09/2018).

[33] The PostgreSQL Global Development Group. *PostgreSQL - About*. URL: https://www.postgresql.org/about/ (visited on 05/09/2018).

[34] Wikipedia. *ACID*. 2018. URL: https://en.wikipedia.org/wiki/ACID (visited on 05/09/2018).

[35] International Organization for Standardization. *ISO/IEC 9075-1:2008*. 2008. URL: https://www.iso.org/standard/45498.html (visited on 05/09/2018).

[36] Perl.org. *About DBI*. URL: https://dbi.perl.org/about/ (visited on 05/09/2018).

[37] Cpan.org. *DBD::Pg*. URL: http://search.cpan.org/dist/DBD-Pg/Pg.pm (visited on 05/09/2018).

[38] Postgresql.org. *Part II. The SQL Language*. URL: https://www.postgresql.org/docs/10/static/sql.html (visited on 05/09/2018).

[39] Postgresql.org. *Pattern Matching*. URL: https://www.postgresql.org/docs/9.3/static/functions-matching.html (visited on 05/09/2018).

[40] Datlowe. *postgres-tsearch-czech*. URL: https://github.com/datlowe/postgres-pgq-tsearch-czech (visited on 05/09/2018).

[41] Ildus Kurbangaliev. *Remove 1MB size limit in tsvector*. 2017. URL: https://www.postgresql.org/message-id/flat/20170801170846.66e3ab06%40wp.localdomain#20170801170846.66e3ab06@wp.localdomain (visited on 05/09/2018).

[42] Tute Costa. *How to fix PostgreSQL error on index row size*. 2015. URL: https://github.com/doorkeeper-gem/doorkeeper/wiki/How-to-fix-PostgreSQL-error-on-index-row-size (visited on 05/09/2018).

[43] Huang Suya. *Thread: werid erroríndex row size 3040 exceeds btree maximum, 2712öccur randomly*. 2013. URL: https://postgrespro.com/list/thread-id/1538018 (visited on 05/09/2018).

[44] Holger Gehrke. *OCR*. 2015. URL: https://help.ubuntu.com/community/OCR (visited on 05/09/2018).

[45] Linux.com. *How to scan and OCR like a pro with open source tools*. 2008. URL: https://www.linux.com/learn/how-scan-and-ocr-pro-open-source-tools (visited on 05/09/2018).

[46] Karthick Murugadhas. *What's the best, simplest OCR solution?* 2010. URL: https://askubuntu.com/questions/16268/whats-the-best-simplest-ocr-solution (visited on 05/09/2018).

[47] Slant.co. *What are the best Linux OCR programs?* 2017. URL: https://www.slant.co/topics/4148/~linux-ocr-programs (visited on 05/09/2018).

[48] Tesseract. *Tesseract Wiki*. 2017. URL: https://github.com/tesseract-ocr/tesseract/wiki (visited on 05/09/2018).

[49] Joerg Schulenburg. *GOCR*. 2017. URL: http://www-e.uni-magdeburg.de/jschulen/ocr/ (visited on 05/09/2018).

[50] Tim Waugh. *gocr(1) - Linux man page*. 2006. URL: https://linux.die.net/man/1/gocr (visited on 05/09/2018).

[51] Jussi Pakkanen. *Cuneiform for Linux*. URL: https://launchpad.net/cuneiform-linux/ (visited on 05/09/2018).

[52] Ubuntu Manuals. *GraphicsMagic*. URL: http://manpages.ubuntu.com/manpages/trusty/man1/gm.1.html (visited on 05/09/2018).

[53] Ubuntu Manuals. *Cuneiform - multi-language OCR system*. URL: http://manpages.ubuntu.com/manpages/trusty/man1/cuneiform.1.html (visited on 05/09/2018).

[54] Gnu.org. *What is Copyleft?* 2017. URL: https://www.gnu.org/licenses/copyleft.html (visited on 05/09/2018).

[55] Askubuntu.com. *Convert docx to PDF*. 2017. URL: https://askubuntu.com/questions/396825/convert-docx-to-pdf?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa (visited on 05/09/2018).

[56] Superuser.com. *Converting doc to pdf*. 2011. URL: https://superuser.com/questions/337667/converting-doc-to-pdf/337668?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa (visited on 05/09/2018).

[57] Linux.die.net. *unoconv(1) - Linux man page*. URL: https://linux.die.net/man/1/unoconv (visited on 05/09/2018).

[58] The LaTeX Project. *An introduction to LaTeX*. URL: https://www.latex-project.org/about/ (visited on 05/09/2018).

[59] en.wikibooks.org. *LaTeX*. URL: https://en.wikibooks.org/wiki/LaTeX (visited on 05/09/2018).

[60] en.wikibooks.org. *LaTeX/Basics*. URL: https://linux.die.net/man/1/unoconv (visited on 05/09/2018).

A Appendix

A.1 Usage of IT Forensic Tool

```
Usage:

        ft [<general option>] <command>

General Options:

        [--debug]

User commands:

        addcase|ac <name>

        addevidence|ae <case> [--name <evidence_name>] <file|directory>

                A <file> can be in raw ("dd") or EWF format.

        deletecase|dc <name>

        digest|dg <case> [--force] [<evidence>] [<path>]

                Process the data of <evidence> or all evidence images of <case>,
                if no <evidence> is specified. If <path> is specified, process only
                this file/directory (/cases/<case>/data/<path>).

                --force   force processing even if already done

        export|ex <case> [--sha512 <file>]

                Export files and it's metadata.

                --sha512 <file>   export files with SHA512 checksums from <file>.
                        The file should contain one hash per line (everything from
                        column 129 to the end of line is ignored).

        exportkeywords|ek <case>

                Export all files matching predefined keywords.

        listcases|lc [--verbose]

        listevidence|le <case> [--verbose]

        modifycase|mc <name> [--name <new_name>]

        mount

                Mounts all evidence under all cases.

        star <file>
```

```
            Mark file with a star (create symlink to <file> in /cases/case/star).

Internal commands:

        carve <case> <evidence>

            Carves deleted files and files on freespace from given <case> <evidence>.

Tools:

        blackhole|bh [--delete] <path> [<path>, ...]

            Store file's (or directory name) checksum in database of uninteresting
            data, optionaly deleting it (--delete). See also "suck" command.

            The chechsum is SHA512 hash. File's and direcorie's names are normalized
            (lowercased, '/' characters removed).

        deduplicate [--list sha512list] [--keep-empty-dir] <dir>

            Delete all duplicate files from <dir> (by comparing their
            SHA512 checksums). Unless --keep-empty-dir option is given, delete also
            all empty subdirectories.

            Options:

            --list <sha512list>   delete also all files from <directory> that match
            SHA512 hash listed in <sha512list> file. The file should contain
            one hash per line (everything from column 129 to the end of line
            is ignored). This option can be specified multiple times.

            --keep-empty-dir      do not delete empty subdirectories

        describe|ds <file>

            Print <file>'s metadata.

        dkim <file> [<file2> ...]

            Verify DomainKeys signature from saved email.

        hash <file> [<file> ...]

            Display <file>'s hash.

        horizon|hz <file>

            Display whether is <file> uninteresting. See "blackhole" command.

        ibackup2timeline <backupdir> <outputdir>

            Create timeline from iTunes backup. The backup must be in the plain form, as
            produced by ibackupextract command.

    ibackupextract <backupdir> <outputdir>

            Renames files from iTunes backup to it's original name.
```

```
suck|sk <path> [<path>, ...]
        Delete all uninteresting files under <path>. See "blackhole" command.
sum <file> [<file2> ...]
        Print <file>'s checksum (xxHash).
tidyup <dir>
        Sort files in <dir> into subdirectories according to their type.
type <file>
        Display <file>'s type.
ufed2timeline <Report.xml> <outputdir>
        Create timeline from UFED's XML report.
```

A.2 Pattern matching: Performance test

Table A.1: Performance test.

Operator	Duration (no index)	Duration (pg_trgm - GIN)
LIKE	3905.356	929.795
ILIKE	15748.788	3758.209
SIMILAR TO	10728.921	1697.714
~	7164.954	1605.714
~*	7212.436	1600.380