FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

BUILDING AN INTELLIGENT ASSISTANT FOR DIGITAL FORENSICS

By

UMIT KARABIYIK

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2015

Umit Karabiyik defended this dissertation on July 13, 2015.
The members of the supervisory committee were:

Sudhir Aggarwal

Professor Directing Dissertation

Simon Foo

University Representative

Zhenhai Duan

Committee Member

Xiuwen Liu

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

This dissertation is dedicated to my beloved wife Tuğba and our lovely daughter Azra Hamide

ACKNOWLEDGMENTS

First of, I would like to express my deepest appreciation to my advisor and major professor Dr. Sudhir Aggarwal. I cannot admit how much I learned from him and how he inspired me during my studies. Specifically, I appreciate the patient guidance, advice and motivation that Dr. Aggarwal provided me during our meetings and would like to thank him for being very supportive and understanding over the past 5 years. Without his guidance and persistent help this dissertation would not have been possible. I feel very fortunate to have had the opportunity of working with him. I would like to extend my appreciation to my committee members Dr. Xiuwen Liu, Dr. Zhenhai Duan and the university representative Dr. Simon Foo, for their valuable time and recommendations during my studies.

I would also thank my friends Dr. Elvan Aktas, Dr. Fatih Oncul and Dr. Kemal Akkaya for their motivation and giving me excellent suggestions about the academic life in the USA. I also would like to thank my fellow friend Shiva Houshmand for her friendship and collaborations in several projects. I also extend my thanks to Clayton Butler for his assistance and the time that he spend with me to test my system at the ECIT lab.

I also express my gratitude to the FSU Department of Computer Science support staff. I particularly thank Eleanor McNealy, Daniel Clawson and Edwina Hall for their assistance handling many administrative issues at both department and university level. I would also like to thank all the FSU Global Engagement administrative staff, particularly to my exchange visitor adviser Tanya Schaad, for their support and help for dealing with all the immigration bureaucracy in the US.

I would like to thank my parents for believing in me and supporting me in every aspect for many years. I would also like to thank all my loyal friends in the US and in Turkey. Last and certainly not least, I would like to thank my beloved wife Tuğba Karabiyik and my lovely daughter Azra Hamide Karabiyik, for filling my life with energy, peace and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Software tools designed for disk analysis play a critical role today in digital forensics investigations. However, these digital forensics tools are often difficult to use, usually task specific, and generally require professionally trained users with IT backgrounds. The relevant tools are also often open source requiring additional technical knowledge and proper configuration. This makes it difficult for investigators without some computer science background to easily conduct the needed disk analysis.

In this dissertation, we present AUDIT, a novel automated disk investigation toolkit that supports investigations conducted by non-expert (in IT and disk technology) and expert investigators. Our system design and implementation of AUDIT intelligently integrates open source tools and guides non-IT professionals while requiring minimal technical knowledge about the disk structures and file systems of the target disk image. We also present a new hierarchical disk investigation model which leads AUDIT to systematically examine the disk in its totality based on its physical and logical structures. AUDIT's capabilities as an intelligent digital assistant are evaluated through a series of experiments comparing it with a human investigator as well as against standard benchmark disk images.

CHAPTER 1

INTRODUCTION

We live in a world in which most data is rapidly moving to a digital version, e.g. pictures, books, diaries, videos, calenders, and even our genetic information can be kept in a digital format. Tablets, smart phones, laptops, and wearable devices (smart watches, smart glasses, etc.) have became a part of our everyday life. As a result of this transformation we have all become possible targets for various types of cybercrimes wherein these devices are used for criminal purposes. In order to cope with this potential risk and solve digital crimes after they are committed, digital forensics, a branch of forensic science, is becoming increasingly needed in this early twenty-first century. It is becoming a scientific discipline which focuses on the recovery and investigation of digital information found in various digital devices, generally belonging to suspected criminals and often found at crime scenes.

Digital forensics is used in order to collect digital evidence for a variety of crimes including child pornography, financial fraud, identity theft, cyberstalking, homicide, abduction and rape. [74]. Digital forensics investigators use a variety of software tools during their examination of digital devices. These tools play a significant role in collecting and analyzing digital evidence.

Forensic investigation in general and especially of a hard disk is complex for an investigator. There is generally a fairly steep learning curve for such disk investigations because of the required technical background.

The steep learning curve arises partly because of the wide variety and availability of forensic investigation tools. There are many tools that must be considered, both commercial and open source. Newer tools are regularly becoming available, particularly open source. These tools, to varying degrees, provide levels of abstraction that allow investigators to identify and safely copy digital evidence, and perform routine investigations [24]. Investigators are however always expected to know how to use and configure/parameterize these tools, especially the open source tools, depending on the investigation type. Availability of a large number of these tools thus requires the capability to answer the following questions: “How do I properly use these tools?” and “where/when can I effectively use them?” In practice, forensic examiners have varying levels of IT background

and technical expertise ranging from being a computer security expert to an investigator having minimal computer skills. Thus investigators need usable tools that will help them get results easily [46] and with less usage complexity independent of their computer and IT expertise. When we take human involvement into account, especially for the analysis of today's typically large amount of data, it is increasingly required to have such tools that will be able to reduce the investigative burden on the human investigator regardless of their technical expertise.

Currently, learning even for investigators with computer expertise is necessary because investigators have to know details of the target disk image. For instance, investigators generally should know the details of each new disk type, file system, hidden places on the disk, etc., in order to perform correct disk forensics investigation. As Garfinkel [38] discusses, many people in the digital forensics area would like to be able to work with data on the target device without having a deep and specific knowledge about the target disk.

To deal with these issues, most digital forensics investigators typically take training sessions both on tool usage and also on digital targets [9]. According to the user study in [46], 68% of their "experts" indicate that they take intensive training sessions to learn the current tools while 31% say they do not. This latter set still finds the tools difficult to use but found different workarounds (such as online training). As for the open source tools, it is a common situation that one software tool alone cannot capture enough required data. Therefore, the examiner needs to use multiple tools to get relevant evidence from the target. This also requires more training and adds to the learning curve because of the technical knowledge required by the tools. These tools also do not tend to work with each other. Users of today's tools need to properly interpret what results they get from the tools and determine the further steps they need to take for conducting a deeper investigation.

Standardization is yet another challenge in the area of digital forensics. Digital forensics examiners conduct their investigations based on their expertise, previous examinations and their organizations' guidelines. This is mainly because there is no universal standard for digital evidence collection [9].

In this dissertation, we describe AUDIT, a novel automated disk investigation toolkit that is designed to support integration of open source digital forensics tools within an expert system to simplify and support disk forensics. Our goal is to provide an "intelligent assistant" to support forensic examiners. Our system design and implementation integrates some commonly used open

source tools via an expert system and knowledge base that we have developed to support investigations, while requiring only minimal technical knowledge about the tools, the hard disk structure and the file system on the target disk. Examiners can use our toolkit to analyze the disk for illegal image search, document search, email address and email file search, and also for more specialized searches such as for credit card and social security numbers. In this dissertation we also present a hierarchical disk investigation model. Using this model, we aim to help expert and non-expert investigators to systematically fully examine a hard disk based on its physical and logical structures wherever data can be stored.

Expert Systems (ES) are a class of computer programs that arose in work in artificial intelligence. One goal of AI technology is to build computer programs that demonstrate intelligent behavior [30]. Expert systems emulate human expertise in well defined problem domains by using a domain implemented knowledge base [77]. Concepts and methods of symbolic inference, or reasoning, are also a focus of such programs to represent knowledge that can be used to make appropriate inferences [30].

Automating the digital forensics process of course has its own challenges. It is cautioned in [52] and [62] that the automation of the digital forensics process should not let the forensics profession be "dumbed down" because of expert investigators relying on automation more than their own knowledge. Instead, they suggest that it is more important that the untrained investigators conduct their investigation at the level of expert investigators. This is our goal for AUDIT also. AUDIT generates two reports: we term these the examination and inference reports. We mainly desire to provide the details of the investigation, which tool is used when and where, and to explain the inference process on why these tools were used and in which order.

In Chapter 2 we present some related work. Chapter 3 explains the background and literature review of digital forensics and artificial intelligence techniques that we have used in our research. We specifically focused branch of digital forensics called computer forensics. For this branch, we give detailed information about the digital forensics tools used and the current problems in this area as well as including our proposed solutions to those problems. We also discuss knowledge representation and expert systems as AI techniques related to their use as part of our approach. In Chapter 4, we present a preliminary version of AUDIT, our novel automated disk investigation toolkit that supports investigations conducted by non-expert (in IT and disk technology) investi-

gators. We introduce its architecture and design with testing of AUDIT for two different types of investigations. In Chapter 5, we describe the new version of AUDIT that includes a dynamic database and knowledge base. We also introduce a hierarchical disk investigation model in this chapter along with a new reporting mechanism. With this new design, we believe we can get the attention of expert investigators to experiment with AUDIT since it can reduce their workload during an investigation. In Chapter 6, we discuss testing AUDIT's capabilities as an intelligent digital assistant through a series of experiments comparing it with a human investigator as well as against standard benchmark disk images. Finally, in Chapter 7, we conclude this dissertation with some future research directions.

CHAPTER 2

RELATED WORK

In this chapter we discuss related work on automating digital forensic processes during different phase of the investigation as well as work related to the application of AI techniques.

The work of Stallard et al. in [78] is one of the earliest applications of expert systems in the area of digital forensics and automated analysis for digital forensics science. The authors used an expert system with a decision tree in order to automatically detect network anomalies when attackers aim to clear all traces that could lead system administrators to them. In this work, an expert system is used in order to analyze log files. Another expert system approach applied to network forensics is described in [55]. In this work fuzzy logic and an expert system are used to again analyze log files related to attacks such as intrusion detection.

The Open Computer Forensics Architecture (OCFA) [82] is an example of automating the digital forensics process. OCFA consists of modules and each module works independently on a specific file type in order to extract the content of the file. In this work, automation is done at the analysis phase of the investigation process but OCFA is not designed to search and recover files from a given device. Instead, OCFA focuses on the collected data after the examination of the disk to generate indices for the text and metadata of the files. The examination is assumed to be done by an expert with IT knowledge.

The Digital Forensics Framework (DFF) is both an open source digital investigation tool as well as a development platform. This tool is designed for system administrators, law enforcement examiners, digital forensics researchers, and security professionals to quickly and easily collect, preserve and reveal digital evidence without compromising systems and data [8]. This work is a good example of tool integration and collaboration in order to reduce the burden on an investigator to use task specific tools. However, DFF still requires knowledge and expertise on the integrated tools and the disk structures. Although its interface is quite user friendly and does not require knowledge of what specific tool to use, it still requires users to have technical knowledge about the categorization of the tools and when they need to apply certain tools. The user is asked to select

any applicable module in order to analyze the disk image for certain tasks. For example, they do not have to know whether they need to use "scalpel" or "foremost" for data carving, but they must know how they need to use it and when to start performing data carving or file system analysis.

The closest work to ours related to automating the disk forensics processing was proposed by Simon Garfinkel in [38]. His tool "fiwalk" is used to automate the processing of forensic data for the purpose of assisting users who want to develop programs that can automatically process disk images [38]. fiwalk also integrates command line tools of Carrier's SleuthKit (TSK) [23]. The main difference between this work and ours is that fiwalk is specifically working on file system data only and without an integration of AI techniques. fiwalk makes file system analysis simpler especially for the expert examiners. Therefore, it also still requires knowledge of the file system and understanding of file and inode structures.

Hoelz et al. developed a program called MultiAgent Digital Investigation toolKit (MADIK) [48], a multiagent system to assist a computer forensics expert with an examination. The authors apply an AI approach to the problem of digital forensics by developing a multiagent system where each agent specializes in a different task such as hashing, keyword search, windows registry agent and so on. This work is related to our work through being an AI application in the digital forensics area. It is however not focused on building new knowledge about the tools used during the investigation. It learns from previous investigations in order to perform better in the future investigations, but does not use this knowledge for assisting non-expert users.

Fizaine and Clarke [32] proposed a crime dependent automated search engine for digital forensics. This tool focuses on the early stage of an investigation in order to collect information about a specific crime assuming most of the crimes have similar patterns. This information is later used for in depth analysis. This work however does not support automated tool integrating or configuration.

To our knowledge none of this related work is directed to assisting examiners during the examination and analysis phases of the investigation through the support of an expert system. With respect to tools integration, the existing systems do not support a general open source tools integration process but rather only integrate some task specific modules in order to automate certain tasks.

The research does often deals with the problem of reducing time during the data analysis phase (such as image clustering) of the target device(s) but generally does not address the problem of

reducing the technical knowledge required of the investigator. The data analysis phase is after the evidence collection phase when the large amount of returned data might need to be reduced and processed. After the evidence gathering phase, AUDIT does not currently deal with reducing the data analysis time. Nevertheless, tools for reducing the technical burden on the investigator are welcomed by practitioners [9]. Tools for reducing the data analysis could certainly be integrated into AUDIT. In our current implementation, we have created a file sorter tool to categorize recovered files from the disk into separate directories based on file extensions and metadata information. This tool also has an option to filter out the files collected during the examination by using hash value comparison against to known files hash set. After the output is sorted and/or filtered we then ask users to do a visual and manual analysis of the classified data. At this point, users are free to use any available data analysis or data mining tools and we do plan to integrate such tools into our toolkit in the future.

CHAPTER 3

BACKGROUND AND LITERATURE REVIEW

3.1 Digital Forensics

Digital forensics is a branch of forensic science and has been defined in various ways by many sources. However, it is best and may be the most clearly described in [70] as:

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

Digital forensics can be divided into sub-groups relating to the investigation of various types of devices, media or artifacts such as computer forensics, mobile device forensics, network forensics, and database forensics [65]. In our research we specifically focus on computer forensics. Therefore, most of the examples and explanations given in this dissertation will be given with this purpose in mind. The definition of digital forensics above covers digital forensics processes quite widely from seizure of the device to presenting the evidence to legal authorities. However, we would like to narrow down the process steps into more generalized steps as defined in "A guide to first responders" [65] published by the U.S. Department of Justice. The proposed model consists of four phases: collection, examination, analysis, and reporting (see Figure 3.1). We next give some brief information regarding each phase and we focus on the examination and analysis phases only in the rest of our research.

3.1.1 Digital Forensics Phases

The collection phase, evidence collection, is the step of gathering exact sector level copy of all seized digital media which may contain potential evidence. The types of media could be in a variety of forms such as hard drives, USB devices, physical RAM, CDs / DVDs and SD cards [65]. In this phase, the examiner needs to make sure that the copy of the media which will be later used in

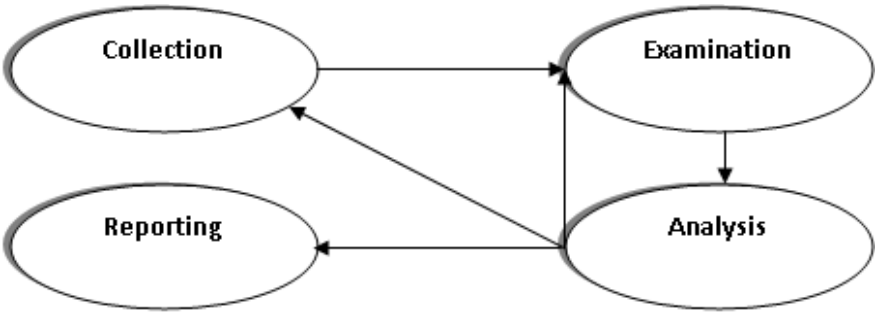


Figure 3.1: The digital forensics investigative process

the examination and analysis phases is accurate as well as ensure that the integrity of the original media is preserved. Otherwise, it is possible that the entire investigation will be considered invalid by the court.

The examination phase is the phase in which investigators comprehensively perform a systematic search of evidence relating to the suspected crime. This focuses on identifying and locating potential evidence, possibly within unconventional locations [26]. This process should accomplish several things. First of all, the state of the evidence must be documented completely. In this process examiners specifically search unconventional spaces for the presence of hidden or obscured data. Since the outcome of this process is potentially large, data reduction can be performed once all the information is made visible [65].

The analysis phase is quite different than the examination phase because it is the further analysis of data that was gathered in the examination phase. In this phase, the ultimate goal is to find the most significant and valuable data related to the case [65]. An investigator recovers evidence material using a number of different techniques and tools. Some of these tools will be discussed later in this chapter briefly. During this phase an investigator looks for data that would answer his/her questions related to the case. For instance, an investigator needs to know: "Given the examined files/data, could this data prove or disprove my hypothesis? And how?" Since they have minimal support from the investigation tools, much of the work is cognitive [56] and dependent on the investigator's experience and knowledge. There might be a need of collecting and examining other media that have never been covered before. Therefore, the investigation process might return back to the collection and examination phases from the analysis phase. It may also take several iterations of examination and analysis to support a crime theory (see Figure 3.1).

The final phase of the digital forensic investigation is *the reporting phase*. A written report usually contains and outlines the examination process, the relevant data recovered with their respective hash values and the conclusions that are drawn from the analysis phase [56]. In addition, the tools that are used during the previous processes and the reasons why they are used are also mentioned in the reporting phase. Examination notes must be preserved for discovery or testimony purposes. An investigator may need to testify about not only the conduct of the examination but also the validity of the procedure and his/her qualifications to conduct the examination [65]. Some of the automated (especially commercial) tools prepare reports for investigators. However, almost none of the single purpose tools creates such report therefore it is the utmost necessity for the investigator to keep notes for every action and tool used in all the phases mentioned above for reporting purpose.

3.1.2 Digital Forensics Branches

Digital forensics can be divided into sub branches based on the type of the investigated device, environment, media and digital artifacts. These branches mainly are computer forensics, network forensics, mobile device forensics and database forensics [27] (shown in Figure 3.2).

In the following subsections, we will discuss computer forensics, data hiding and data recovery techniques in details. As we mentioned earlier we only focus on computer forensics in our research; however, we believe that our model and system can be applied to the other digital forensics branches. Therefore, we will also explain other digital forensics branches briefly. Later, we will give introductory information about the digital forensics tools in general, and specifically we will briefly introduce each tool which is related to our research. Finally, we will present the current problems with respective solutions in the area of digital forensics in details.

3.1.3 Computer Forensics

Computer related crimes started to increase in 1980s as a result of personal computers being available for personal use. Some of the new "computer crimes" were recognized during that time such as hacking [85]. Because of the emergent need, a new discipline, computer forensics, arose as a method to collect, recover, investigate and reconstruct digital evidence for use in court of law [85]. Since then computer crimes and computer related crimes have dramatically increased. For example, the cost of cybercrimes increased in 2014 compared to 2013, with the average cost of a

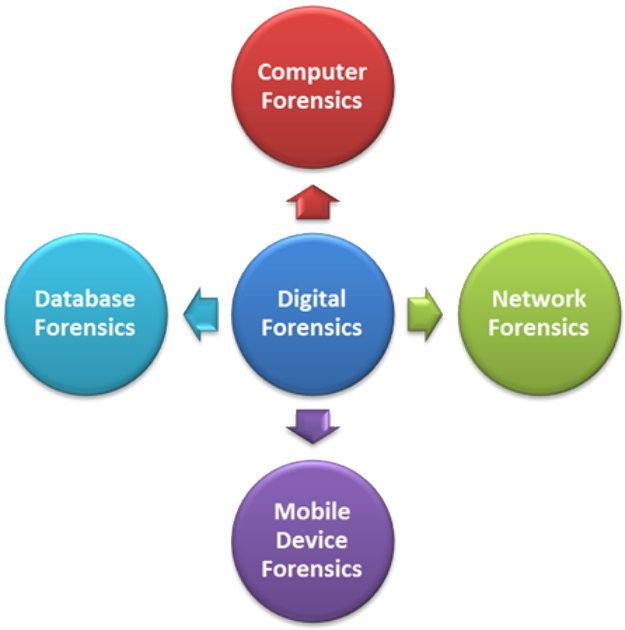


Figure 3.2: Digital forensics branches

crime in the U.S. rising from $11.56 million to $12.7 million. Also, the number of attacks that organizations experienced rose by 176% since 2010 according to a study released at the end of 2014 by HP and the Ponemon Institute [50], a research group that studies Internet security.

Computer forensics is defined by United States Computer Emergency Readiness Team as a multi-disciplinary area that encompasses computer science and law in order to collect and analyze data from computers, networks, and storage devices so that findings can be presented to the court of law as evidence [64]. Thus, the main goal of computer forensics is identifying, preserving, examining, analyzing and presenting all the steps and procedures to find interesting digital evidence in the computer systems in a forensically sound manner.

Today, computer forensics is used to investigate a wide variety of crimes, including child pornography, financial fraud, identity theft, cyberstalking, homicide, abduction and rape. In addition to the criminal cases, it is also used to provide relevant and valid information as evidence in civil, administrative, and other cases such as adultery and inheritance cases[54].

Disk Forensics. Disk forensics is a sub category of computer forensics, and it specifically targets hard disk drives as the source of the investigation to extract forensic information. During the digital forensics investigation, investigator will look at different parts of the computers that are

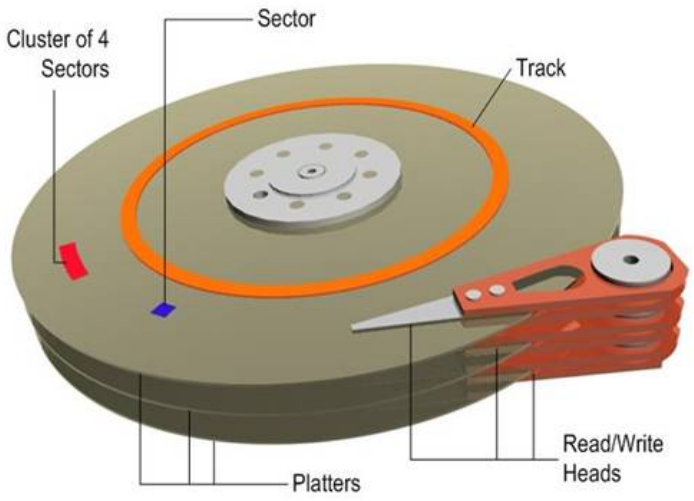


Figure 3.3: Internal structure of a hard disk drive [58]

involved in the case, where digital evidence resides or can be deleted or hidden [54]. For the purpose of our research we will use the term "computer forensics" to refer to performing digital forensics on hard disk images. We will also use the term "network forensics" to mean digital forensics on network traffic, or devices that are in a network. In order to understand the disk forensics, first let's briefly take a look at the hard disk structure. Then, we will explain some of the terms that are needed to understand which part of a hard disk is analyzed during the investigation.

Physical structure of a hard disk consists of several metal platters, an arm assembly, read-write heads on each arm, and a motor to rotate the disk at speeds of up to 10,000 rpm [28]. The way that binary data is written to the hard disk is magnetically. Data is recorded on "tracks" which are imperceptible and closed centered circles. Each track on the disk is further divided into smaller, more manageable units called "sectors" (see Figure 3.3).

Sector and Cluster. A sector is the smallest addressable unit on a disk, and was generally 512 bytes in size until January 2011. As of 2011, sector size for all applications using hard disk is standardized by International Disk Drive Equipment and Materials Association (IDEMA) by asking the hard drive industry to move to 4K as the new standard sector size [86]. Since a sector size of 4K is still small this yields too many sectors on a hard disk for the operating system to keep track of (A 500 GB hard disk has over 1 billion sectors!) [28].

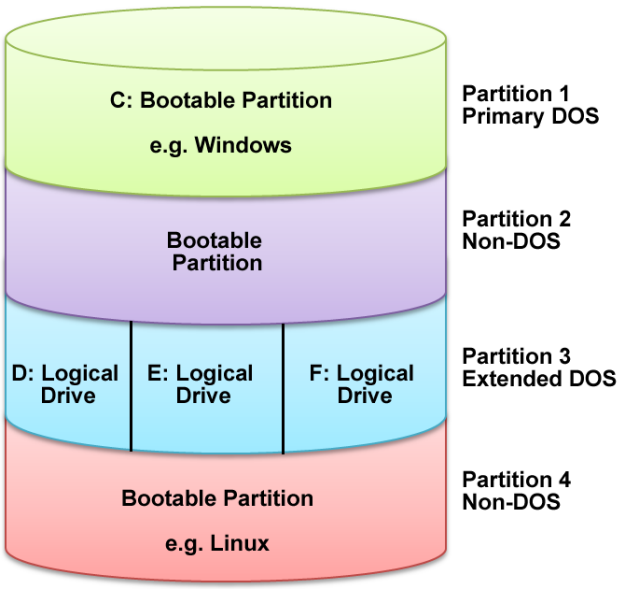


Figure 3.4: A hard disk divided up into partitions and volumes

In order to deal with this problem, operating systems create logical groups called "clusters" (blocks in Unix systems). These clusters group the sectors in multiples of 2 (eg, 1, 2, 4, 8, 16, 32, 64 or 128). The total number of the sectors in each cluster is called "cluster size" of the disk [28].

PARTITION. A hard disk may be "split" into several smaller logical disks, called "partitions". Each partition on a hard disk is treated like a separate disk. This allows a single hard disk to store different operating systems, which can be selected when the computer is booted [28]. For example, a hard disk could be divided up into four partitions as shown in Figure 3.4.

VOLUME. Another term in the disk structure is called "volume". "Volume" and "partition" are the terms that are frequently used together and sometimes cause confusion. Brian Carrier makes a clear distinction in his book [14]. Here is what he says: "A volume is a collection of addressable sectors that an Operating System (OS) or application can use for data storage. The sectors in a volume need not be consecutive on a physical storage device; instead, they need to only give the impression that they are. A hard disk is an example of a volume that is located in consecutive sectors. A volume may also be the result of assembling and merging smaller volumes. One of the concepts in a volume system is to create partitions. A partition is a collection of consecutive sectors in a volume. By definition, a partition is also a volume, which is why the terms

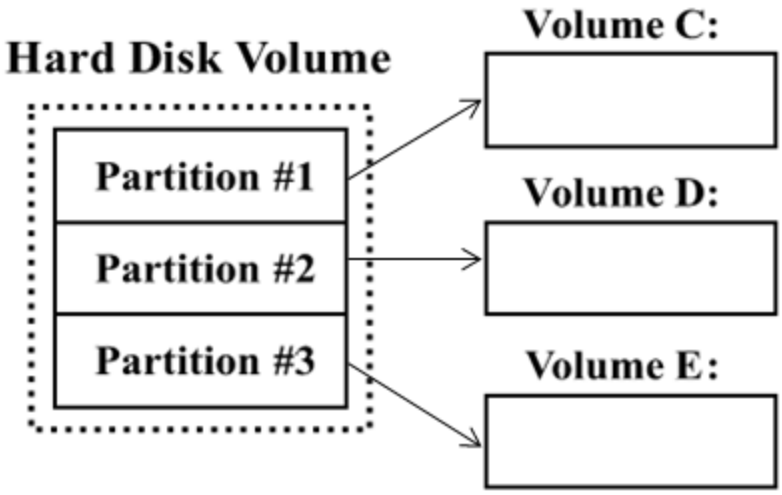


Figure 3.5: Example hard disk volume organized into three partitions

are frequently confused." In order to visualize the above relationship let's consider a Microsoft Windows system with one hard disk. Assume that the hard disk volume is partitioned into three smaller volumes, and each consists of a file system. Windows assigns the names C, D, and E to each volume. We can see this in figure 3.5. It is quite important to understand and not be confused with terms volume and partition in order to perform volume system analysis correctly and in a forensically sound manner.

File System. A file is the common storage structure in a computer and the information in a file is used by the operating system [71]. File system is a data structure that keeps track of storage, access, and retrieval of data. In a file system data is stored in a file and directory hierarchy [14]. File systems organize sectors on disks and keep record of sectors which are assigned to files or not being assigned. Therefore, file system helps computer to know where to find files and directories. As we will discuss next, assigned sectors represent allocated space while unassigned sectors represent unallocated space.

Allocated and Unallocated Space. When a user wants to create a file, the operating system will assign certain sectors to the file on disk. The space that is given to that file is called allocated space and managed by the file systems.

When a user intends to delete files, operating system marks these files as deleted even though they are not truly deleted. When a file is marked as deleted, operating system sets the file's

allocation status to unallocated and the space may be allocated to another file when needed. However, the content of the file remains in the same space that was allocated before unless it is overwritten by the operating system or wiped via anti-forensics tools. As long as that previously allocated space is not overwritten or wiped then it is possible to recover data from the same space using digital forensics tools [80].

Slack Space. File systems allocate fixed sized clusters (Windows) or blocks (Linux) to each file. Slack space results when the file does not use the entire fixed length space that was allocated. If the allocated space was used by another application or any previous data was deleted, some data from previous use is still available in the slack space of the file. Even though the data found in the slack area may not reveal any complete information it can prove to a forensic examiner that a certain type of file might have existed and possibly deleted [80].

Data Hiding Techniques. Data hiding is a technique that is used by criminals or even for ordinary users in order to conceal any data to be seen by others. Digital data hiding might be divided into two categories, physical and non-physical data hiding such as cryptography, steganography, and watermarking [10]. Non-physical aspect of digital data hiding is out of the scope of this dissertation. In this dissertation we aim to collect and report physically hidden data from the hard disk. Using today's digital forensics tools it is often easy to find hidden data on the disk for the expert practitioners. However, it is not the case for the non-expert users since those tools require detailed background knowledge about the disk structure and/or the file system.

Hard disk drives have been the most frequently used device as for secondary storage of data in general purpose computers since the early 1960s [88]. In today's computers, most of the personal data are stored in the hard drive. Therefore, it plays crucial role and needs to be analyzed extensively for digital forensics investigation in most of the digital crimes to find any evidence. Any data can be hidden at each of the previously mentioned structures such as volume, partition, slack space, unallocated space and so on. Therefore we will cover some of unconventional structures that hidden or obscured data might be present in order to give more idea about how much detail knowledge an investigator is required to possess. Current automated digital forensics tools specifically work on one or more of these structures. They promise to look at every possible places that are in the scope of those tools. However, they sometimes skip or do not recognize these places due to altered or manipulated structures. Therefore, investigators may need to use other tools to specifically analyze

these hidden places. Now, we will explain some of the spaces that could be used for data hiding purposes and our system systematically searches those areas for evidence.

Every hard drive using a DOS partition has space at the beginning of the drive that is allocated for a Master Boot Record (MBR) [10] (see figure 3.6). MBR is a 512-byte sector at the very beginning of a hard disk drive that often contains a sequence of commands needed for operating system to be booted [7] and always contains partition table [10]. It holds the information about the organization of the logical partitions as well as the file systems. The maximum number of partitions that can be defined in the partition table is 4. All the partitions location and size informations are kept. In order to keep all this information in the partition table only 1 sector is required. However, because of the hard disk's cylindric structure, each partition starts on a cylinder boundary, there will be 62 empty sectors between the partition table and the first partition. This place can be used as a hidden area for data hiding.

Figure 3.6: Disk layout and master boot record (MBR) space

Today's users may need more than four partitions. In order to meet that need, extended partitions were designed. Whenever extended partitions are used then there will be 62 more empty sectors available for perpetrators to hide data because extended partitions also have the same structure as MBR [10]. Extended partitions has a nested structure because they can only contain one file system and one extended partition. However, this recursion helps creating as many partitions as needed while yielding yet another available space for hidden data.

Previously, we described how slack space for files occur in the file system. There is also another possible slack space arises when all the available space on a hard drive is not used for partitions. This space is called "volume slack" (see figure 3.7) and considered to be another digital warren for data hiding because of operating systems unavailability of searching this space [10]. The reality about file deletion is also the same for deleting partitions if a previously created partition is used and it is later deleted. As explained before, delete operation does not really deletes data from the

disk. So, it is highly possible that some data will be residing in the volume slack when a new partition with lesser size than the deleted partition is created after deleting the previous one [10].

Figure 3.7: Volume slack on hard disk

There is another possible slack space which is called "partitions slack" that is created when the size of a partition is not multiple of the block size. Due to this incompatibility, there will be some unused sectors at the end of the disk that are not available for file system access (see figure 3.8) [10]. This slack space may also be used by perpetrators for data hiding.

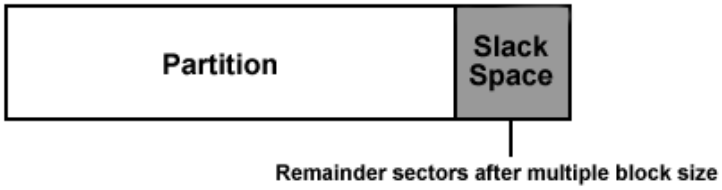


Figure 3.8: Partition slack on hard disk

Also, any unallocated space that is not assigned to files cannot be accessed by the operating system. Therefore, the unallocated space will be a potential space for data hiding until it is allocated to a file [10].

Data Carving Techniques. Data carving is one of the most crucial part of the digital forensics investigations. The term of "Data Carving" defined by Digital Forensics Research Workshop (DFRWS) as [29]:

> "Data Carving is the process of extracting a collection of data from a larger data set. Data carving techniques frequently occur during a digital investigation when the unallocated file system space is analyzed to extract files. The files are "carved" from the unallocated space using file type-specific header and footer values. File system structures are not used during the process."

We would like to emphasis on the last part of the definition above for a second that no filesystem structure is used during the data carving process. This actually tells us that data carving process itself is file system (e.g. FAT16, FAT32, NTFS, ext2, ext3, HFS, etc.) independent [60]. However, it will be helpful for the investigator to know the file system type in order to conclude or relate some of the findings after data carving is performed along with other methods of file recovery. Below, we will briefly define some of the terms that will be helpful to understand the data carving process.

The definition above is also mentioning file type-specific header and footer values. These values are called "magic numbers" and used as a standard of recognizing file types rather than only looking at their extensions which also represent specific file types. The reason that the industry has developed magic numbers is the ease in changing file extensions in order to obfuscate file types [53][60]. Thus a file may not actually be correct.

Simply, almost every file has a header/footer pair or at least an header in order to recognize the file's boundaries and file types correctly. For example, a *gif* file starts with magic number "GIF89a" in ASCII or "0x474946383961" in hexadecimal, and ends with "0x003B". As an another example, a *pdf* file begins with "%PDF" and ends with "%EOF" [91]. However, some of the files such as text files (e.g. txt, html, c, tex, etc.) do not have either header or footer, but it is still possible to recover these files based on the content of the file. For instance, txt files are carved when a tool detects several lines of text data based on frequent and consecutive newline characters ('\n') found on disk. As for other text files, the usually starts with certain keywords and they are carved based on these texts. For example, an *html* file starts with `<html` when *c* file starts with `#include` keyword. After these keywords are detected, carving tool starts carving the data from the disk using a predefined file specific maximum carving size for each file. In this case, carving tools do not care if any unrelated data is appended due to the incorrect size of the file to the original file.

Finally we describe the term of *file fragmentation*. File fragmentation occurs when a single file is divided into multiple pieces and saved on disk non-contiguously. Although file systems try to consecutively allocate all sectors in a file, it is sometimes not possible because of some performance issues [87]. So, this yields fragmented files on a hard disk drive. From digital forensics investigator's perspective, fragmented files make the investigation harder because many automated tools fail to find all the pieces when file system metadata information is lost or damaged.

Now, we will explain data carving concepts in order to show what would cause an investigator

to perform data carving methods.

Carving Concepts. Data carving might be classified as basic and advanced; with basic data carving it is assumed that the beginning of file is not overwritten, in other words header of the file is not missing. Also, the file is supposed to be unfragmented and uncompressed. For the basic carving method, the tools basically search for header and footer or only the header of a file via magic numbers [60]. When they are found, the tool carves all the information between the header and footer.

In advanced data carving, files are also searched when fragments are not sequential, out of order or missing. This type of carving is depending on file's internal structures rather than the footer and header information only [60]. Again, today's operating systems typically do not create fragmentation unless they have no other choice. However, a perpetrator may willingly save files in fragments and later delete the fragments in order to make the file recovery harder. In such case, fragmented files are carved using "smart carving" technique which will be discussed later.

In addition to the smart carving, there are also different carving techniques that are used by carving tools. As mentioned in the previous section, there are also files that are carved based on their contents. This type of carving is also considered as advanced carving method. In our research, we used aforementioned file carving techniques and other types of file carving methods can be found in the file carving taxonomy proposed by Simson Garfinkel and Joachim Metz in [33].

3.1.4 Other Digital Forensics Branches

Before we start explaining the other digital forensics branches in addition to computer forensics, it is important to mention that there are different technologies we use in our everyday life and these technologies are not considered to be under single branch. For example, cloud forensics may also be considered to be another digital forensics branch at the intersection of computer and network forensics since both environments are involved during its usage. Thus, we will be discussing only the main digital forensics branches in this section.

Network Forensics. Network forensics is a branch of digital forensics which focuses on analyzing computer networks and their traffic in order to detect anomalies and malicious operations performed on these networks. Main purpose is to gather information and possible evidence related to the investigation in order to present it to the court of law [93]. However, network forensics is

somehow different than the other digital forensics branches due to the type of information that is dealt with. Most of the data in the network is dynamic and volatile which makes the investigation quite harder because when the network traffic is transmitted the data could be lost [25].

Mobile Device Forensics. This branch of digital forensics deals with mobile devices such as cellular and smart phones, tablets, smart glasses, digital cameras, GPS devices and so on. Basically, all the hand held portable devices are investigated and analyzed using certain mobile forensics techniques and tools [92] under this branch. Investigators usually target these devices to collect information from calls, contacts, pictures, videos, geo-locations, chat logs, documents, applications, etc.

Especially in recent years mobile device forensics has played significant role in the digital forensics investigations. People in different age range, from kids to elders, go mobile every day. Our life in digital world is now in our hands. This yields to tremendous information being used via mobile devices and this data is analyzed comprehensively by the investigators. Despite the availability of this large data, mobile forensics investigations are not as easy as computer forensics investigations in terms of the standards and manufacturers that dealt with. In computer industry there are certain number of manufacturers and certain standards are being used. In the case of mobile devices, there thousands of manufacturers put new devices into the market every once a while. It is therefore quite challenging for the investigators to deal with suspects in the mobile world [57].

Database Forensics. Database forensics deals with forensic analysis of databases and raw data in them as well as the metadata that describes the data [66]. Database forensics needs an attention because of the amount of data required to be analyzed for such crimes that databases may be used. For the instance of a financial crimes case, an investigator may need to analyze tremendous data in companies' databases. As Craig Write testifies in his report in [94] that he dealt with a database with 68TB of data in a business's database in 2009. So, it is quite clear that this extra ordinarily large data cannot be ignored.

Same as other digital forensics branches, there are certain methods and tools to perform forensically sound investigation on databases. Since, it is out of the scope of our research, we will refer potential ways of discovering evidence in databases to [94] and [66].

3.1.5 Digital Forensics Tools

There are large number of digital forensics tools on the market, either open source (mostly free) or closed source (mostly commercial). The majority of these tools are developed for specific problems and search purposes - volume system analysis, file system analysis, memory analysis, network analysis, etc. - and they are mostly not designed to be integrated with one another [24]. Although many of these tools overlap in terms of their capabilities, one might be better then another on a specific task. As we have discussed before, this situation results in a situation where the examiners have to purchase/have each of these tools for any deep investigation since the evidence might reside in any part of the media.

There are hundreds of tools that are used by digital forensics investigators. For example, only for disk imaging purpose there are more than thirty tools, and for data recover and carving purpose there are more than twenty tools developed for both Unix-based and Windows-based systems [34]. In this section, we will give only brief information about some of the digital forensics tools. However, we will give more detailed information about these tools in the relevant chapters that we will explain how they are specifically used.

One of the purposes of this section is to provide more information about the open source tools that we have already integrated with each other. Another purpose is that we would like to give some information about the tools such as FTK [5] and EnCase [76] which are complex, commercial, and frequently used tools for relatively complete investigations. Today, almost all the investigators mainly use FTK, EnCase and The SleuthKit (TSK) [23] when they start their investigations. However, they oftentimes use other open source command line tools because those relatively complete toolkits do not help them to completely analyze the disk. Since FTK and EnCase are closed source toolkits, we do not integrate them into our system. However, we add some of the TSK's tools into our automated system.

AccessData Forensics Toolkit (FTK). *FTK* is a well known, commercial and court-accepted digital forensics toolkit that is built for speed, analytics and enterprise-class scalability. *FTK* is a product of AccessData [6]. Some the features of *FTK* are intuitive interface, ability to perform file system and email analysis, customizable data views, advanced volatile and memory analysis, and stability [5]. *FTK* is not integrated into our system.

EnCase Forensics. *EnCase* is a Guidence Software [45] product and it is another most widely used and court-accepted commercial investigation solution. *EnCase* has its own evidence file format to save evidence for the court. Some features of *EnCase* are advanced file system and disk analysis, forensically sound acquisition, acquiring data from all types of sources, providing scripting and automated reporting [76]. Same as *FTK*, we will not integrate this tool into our system either.

The SleuthKit & Autopsy Browser. *TSK* and *Autopsy* [22] are open source digital forensics tools that run on both Windows and Unix systems such as Linux, OS X, FreeBSD, OpenBSD, and Solaris. *TSK* and *Autopsy* are mainly developed by Brian Carrier and now they are supported by him and Basis Technology [81]. *TSK* is a library and set of command line tools which are designed to help examiners performing an investigation on disk images. *TSK* mainly allows investigators analyze volume and file system data [23]. *Autopsy* is a graphical interface of the *TSK* tools library and it helps investigators perform their investigation visually. We will use some of the standalone command line and automated tools under *TSK* to incorporate with other forensics tools for our research purpose. Now, we will give brief information about the *TSK* tools that we used in our research and integrated with each other. It is needed to mention that the order that the tools are explained has no significance.

blkls [15] takes a disk image as an input and copies the blocks in the file system. We use this tool in order to extract slack space of the disk image by providing specific parameters [15].

fsstat [16] is used when details of a file system is needed. It takes a disk image as an input and outputs either set of block/cluster information or a specific information about the file system. We use it in order to get the type of a given file system on disk image [16].

img_cat [17]basically outputs all the contents of a given disk image [17]. However, we use it for a specific purpose especially when we have a disk image in "Expert Witness Compression Format (EWF)" which is an EnCase image file format used in order to store different types of digital evidence [61]. *img_cat* helps us to convert an EWF type disk image to a raw disk image so that all of the open source tools that we use can read the contents of the disk image without possible complications.

mmls [18] takes a disk (or other media) image as an input and analyze its partition structures. *mmls* displays the layout of a disk, including both allocated and unallocated spaces [18].

mmstat [19]is volume system analysis tool. It outputs the detailed information for a volume system including partition tables [19]. We use this tool in order to get the disk image type for our further investigation on disk image where disk image is asked by other tools.

tsk_loaddb [20] collects metadata information from the disk image and loads all the information into a SQLite database [20] which is then used to parameterize other tools and update our systems knowledge base when they are integrated.

tsk_recover [21] is an automated tool that extracts allocated and/or unallocated files from a disk image and save the recovered files to a specified local directory. This tool is very useful if many files are deleted and we would like to recover them to a certain folder [21].

Scalpel. *Scalpel* [49] is developed by Golden G. Richard III and it is currently maintained by him and Lodovico Marziale. Scalpel is a rewrite of *foremost* in order to increase the speed of carving process. Same as *foremost* it also works on various type of disk image files including raw device files. Since data carving is file system independent so *Scalpel* is. Therefore, it is able to carve files from FATx, NTFS, ext2/3, HFS+, or raw partitions using the user defined configuration file [49]. We also have integrated *Scalpel* with other tools for the purpose of data carving.

Photorec. *Photorec* [43] is an open source file recovery (carving) tool which carves files (documents, emails, pictures, videos, compressed files, etc.) from variety of devices including disk images, CRD-ROMS and digital camera memory. *Photorec* is developed and maintained by Christophe Grenier. Same as other file carvers, *photorec* is also file system independent and recovers files based on header/footer information as well as some internal structures of the files [43]. *Photorec* is integrated into out system for data carving purposes.

TestDisk. *TestDisk* [42] is also developed and maintained by Christophe Grenier. It is mainly designed to recover lost partitions on disks. It also helps fixing the non-booting disks in order to make them bootable again. *TestDisk*'s command line interface is not fully parameterizable for the lost partition recovery and it required user's interaction. This make it unsuitable to be integrated into our system. Therefore we use *TestDisk* in command line mode to get the input disk's geometry information only. We then use that information in another tool to automatically recover the lost partitions.

gpart. *gpart* [68] is a command line tool that is used to recover lost partition from an input disk. It uses the disk's geometry with a specific parameter and guesses all possible partitions in

the disk. In our system, *gpart* gets cylinder, head and size information from TestDisk in order to recover lost partitions.

Find_SSNs. *Find_SSNs* [51] is an open source tool developed at Virginia Polytechnic Institute and State University. *Find_SSNs* searches for U.S. social security and credit card numbers. It searches a variety of types of credit cards such as Visa, Mastercard, Discovery Card, American Express, and many others. We use this tool in order to find these sensitive information in disk.

bulk_extractor. *Bulk_extractor* [37] is developed and maintained by Simson Garfinkel. It is a command line tool that allows investigators to extract data such email addresses, credit card numbers, URLs, and many others from a disk image. *Bulk_extractor* has been used in real cases such as an identity theft and financial fraud, and it is eventually found to be successful, robust and fast tool [12].

ReviveIT (revit). *Revit* [41] is another file recovery tool (carver). It is an implementation of the concept which is presented at the 2006 DFRWS forensic (carving) challenge and it is known as Smart Carving method [41]. Smart Carving is a file carving technique that recovers fragmented files from a given disk image [69].

Multimedia File Carver. *Multimedia File Carver (mmc)* [72] is also a file carving tool that is developed by Rainer Poisel in order to carve fragmented multimedia files from a disk.

strings. *Strings* [4] is an open source command line tool that is used in Unix-like systems. It accepts file(s) as an input and prints out all printable characters. It is mostly used to extract printable characters from binary files. In default, it prints out strings with at least 4 consecutive characters followed by a non printable character(s), but it can also be changed via parameterization. We use this tool when reading printable texts from the slack space for sensitive numbers search.

mount & umount. *mount* [3] and *umount* [2] are both Unix command line tools which are used for mounting a file system and removing mounted file system respectively to/from the host Unix system. We use these command line tools to make the file system available for sensitive number search.

file_sorter. We designed a Perl script tool, *file_sorter*, in order to give more useful output to our users. *file_sorter* sorts all the files from an input directory, then saves them in specific folders based on their extensions and metadata information. As our system supports documents, graphics

and email files search, we provide these files in separated folders using *file_sorter*. It also has an option to activate SHA-1 hash filtering when hash values are provided in a text file. If the hash set is provided then *file_sorter* does not output the files that their hash values are found in the given hash set. Using this tool we our system eliminates known good files and provide less output files to the users for their further analysis.

dd. *dd* [84] is one of the commonly used data acquisition tool. It is written by Paul Rubin, David MacKenzie, and Stuart Kemp. *dd*'s simplicity and flexibility makes it being many investigators' one of the top open source data acquisition tool. It is simple because it is file system independent and does not require any knowledge about the files that will be acquired. However, it is complex in a sense that it needs to be configured for sophisticated acquisition. It accepts a data from the source (e.g. directory, file, disk device, disk image, etc.) in block-sized (512 bytes in default) pieces [84] and outputs to the specified location as an image. We have already integrated *dd* tool into our system in order to divide up a whole disk image into smaller parts when needed.

3.1.6 Current Problems

In this section we describe some of the current computer forensics problems that investigators deal with. We will also briefly explain how we address to these problems in this dissertation.

One of the problems in disk forensics was described by Simson Garfinkel from the researchers point of view in [38]. According to Garfinkel, many people in the area of digital forensics would like to be able to work with data that resides on hard drives without having specific knowledge of disk structures. However, the current state of the art requires the researchers to be able to understand the disk image formats, volume system, partitions, file system, etc., and also requires them to have domain-specific knowledge. Again, according to Garfinkel, other researchers would like to develop one button applications that can process disk images without having user's involvement [38]. For this reason Garfinkel et al. developed a program called fiwalk [38]. fiwalk automates the initial forensic analysis of a disk image to reduce the required user expertise, we would like to extend this work to all of the investigation steps. By doing so, we aim to help investigators to perform complete disk analysis with very limited or no technical knowledge about the investigated media as a one button tool.

Another problem in computer forensics which also can be extended to disk forensics specifically is a standardization problem. According to Nicole Beebe's digital forensics research in [9], there

is no single, universal standard for digital evidence collection. Many organizations have their own standards and guidelines for data collection. In Beebe's paper, it's also discussed that these problems are arguable since the technological development is fast and often requires a different approach. Therefore, it may not be practical to build up a universally accepted standards and guidelines [9]. In our research we developed an open source framework with an embedded expert system which has a developer friendly interface. This interface will help tool developers easily integrate their open source forensic tools into our framework. By this way, we aim to define a standard way of integrating and developing open source digital forensics tools.

Computer forensics research deals with another problem which can be defined as toolkit development. Today's tools are not user friendly and requires domain specific knowledge that we mentioned before. They were designed to help examiners while they are searching specific pieces of evidence or even a single clue about the evidence, but not designed to assist them in investigations [39]. Therefore, it is extremely significant to take the investigators' background, computer science skills, workflow, and practices into account[46]. However, in practice it is needed that any user might be able to use these tools regardless of their computer usage skills or investigative expertise. Taking the diverse range of computer expertise into account, practitioners need usable and comprehensive tools that will help them to achieve better or even faster results for their investigations [46]. In our research, we address to this problem by creating a user friendly open source tool so that everybody (experts and non-experts) can use it regardless of their technical knowledge and background.

In addition to the standardization problem, Beebe also points out the challenge of scalability and volume as well as lack of intelligent analytical approaches as unaddressed issues in the area[9]. Since the size of the data storage is rapidly increasing and also the number of digital storages that are involved in the investigation, the digital forensics community needs to take action to solve this issue. As for the intelligent analytical approaches, the need for an relatively intelligent system that reduces the burden on experts for the time consuming actions is increased. For example, expert investigators spend most of their times to review hits that are out of the investigation's scope (i.e. false positives in the investigative sense) [9]. Our research addresses this issue as an automated digital forensics toolkit with an embedded expert system technology. We developed an expert system with embedded expert's knowledge into our integrated digital forensics tool system. This

integration will help investigators to do their analysis in a way that most of the time consuming decisions will be made by the expert system and related tools will be automatically used.

As of our observation and knowledge, another problem of computer forensics is the integration of the forensics tools. There are many forensic tools in the market and the market appears to be growing. Especially, new tools for specific purposes are being added to the market intermittently because of the technological improvements and related frequent needs in the area. Although a few commercial companies update their products with new methods, it is still not enough for users. This leads users to buy one of each tools in order overcome one's weakness with another one. This is obviously not a feasible solution for a small budgeted companies. Therefore, investigators tend to use open source tools and integrate the results in order to perform a complete investigation [39]. Thus, we address the lack of tool integration which is possible only for open source tools now, so that one tool might meet the requirements of another tool via this integration.

In Chapter 5 and Chapter 5, we will introduce the preliminary and complete versions of Automated Disk Investigation Toolkit (AUDIT) to show how we address all of these problems in details.

3.2 Artificial Intelligence

As we have discussed in the previous section, one of the current problems in the digital forensics research is the lack of intelligent analytical approaches and intelligent tools (systems). In order to meet this need we aimed to apply suitable artificial intelligence (AI) techniques to the problem of disk forensics. As defined by John McCarthy in 1955, artificial intelligence is the "science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence" [59].

Since there are many different AI branches and techniques, we found Knowledge Representation (KR) and Expert Systems (ESs) being the most suitable one for our research purposes. We used KR in order to represent and build a domain specific knowledge in the area of digital forensics. We then used ESs to automatically investigate the media based on the represented knowledge without asking users to provide technical and detailed information. Using expert system's inference we also help users to answer the questions "What to do next?", "What tool should be used?", "Where to look for evidence?".

AI has many branches including pattern recognition, knowledge representation, inference, common sense knowledge and reasoning, learning, ontology, heuristics, genetic programming, and machine learning. There are also many different applications of AI such as understanding natural language, expert systems, computer vision, and speech recognition [59][90].

3.2.1 Knowledge Representation

Knowledge Representation (KR) is the branch of Artificial Intelligence dealing with representing the human knowledge symbolically and automatically modify the represented knowledge via invoking reasoning programs when needed. In other words, it aims to explain what the computer program needs in order to perform certain tasks in an intelligent way and how computational methods could provide this required knowledge to the program [11][89].

The main purpose of KR research is analyzing how the reasoning can be done accurately and effectively as well as defining how the set of symbols should be used in order to represent set of facts and rules in the knowledge domain. In order to make inference possible, a symbol vocabulary and a set of logical operators are used and then a new KR sentence is created. Logic is also a crucial part of the KR and it is used to formally define how the reasoning functions should be employed systematically on the symbols. The logical operators and operations may be used are negation, conjunction, adverbs, adjectives, quantifiers and modal operators [89].

3.2.2 Expert Systems

Expert Systems (ESs) are computer programs that are derived from AI. As we discussed before, the main goal of using AI is to understand intelligence via developing software programs that their behaviors are intelligent enough. In ESs, the first matter is to perform inference and reasoning using computer program, and second is to explicitly design how the represented knowledge will be used in order to execute inference on it [31]. It is also important to gather knowledge for the knowledge base from human experts not from textbooks or non-experts [31].

Expert systems in general can be designed as static or dynamic. An expert system is called static when it uses the knowledge base as a fixed source of information, and it is called dynamic when it keeps tracking and recording any changes (addition, removal and modification) in its knowledge base [79]. Historically expert systems and databases are designed to be separated however for the increasing need of accessing both technologies simultaneously inspired system developers to

integrate both technologies. Therefore, in the last decade there has been systems that are developed with integrated experts system and database [73].

Programming in expert systems is actually different than conventional programming languages (e.g. C/C++, Java, etc.). Because, conventional languages are designed to be procedural and the work flow of the program is easy to follow. On the contrary, humans usually solve complex problems using abstract thinking or finding symbolic ways to represent the solution. The way that human solves problems is not quite appropriate for conventional languages [77]. Expert systems are designed to be able to model high level abstraction and define human expertise in especially well defined domains. The area of digital forensics and specifically computer forensics is in fact a well defined domain to be emulated in such systems. The availability of expert system tools, such as CLIPS [77] and Jess [35], has greatly reduced the effort and cost involved in developing an expert system. In our research, we created the knowledge base and designed our expert system on CLIPS as a preliminary system design and Jess as an extended and complete system. We will explain CLIPS and Jess expert system designs in the details in Chapter 4 and Chapter 5 respectively.

In the rest of this section, we will discuss some of the benefits of using expert systems in real life problems. Later, we will explain CLIPS and Jess experts system shells briefly.

Benefits of Expert Systems to End Users. There are certain benefits of using ESs for the users and we will briefly itemize them here [31]:

- Reduces processing time of human involved professional or semi-professional work. The increase in terms of finishing the work is usually tenfold and sometimes hundredfold depending on the design of the system and the size of the work.
- Reduces the internal costs within the companies. Especially in large companies for large systems using expert systems may save up to hundreds of millions of dollars.
- Exceptional quality of decision making. Especially in the systems that the problem is well defined, precision of decision can be ten times better.
- Availability of preserving rare expertise in the system. ESs are used to perpetuate rare knowledge about certain issues in companies. This type of knowledge is usually gathered from an retiring expert in the area and distributed to the other offices or branches of the company.

CLIPS: A Tool for Building Expert Systems. CLIPS is created in 1985 by NASA and made available as an open source expert system developing environment. It allows users to develop expert system tools in an environment that the production of rule and/or object based expert systems is made available [77]. CLIPS has been and still is being used by people in government, industry, and academia. Some of the key features of CLIPS are itemized below [77].

- **Knowledge Representation:** CLIPS provides an environment for creating knowledge base in rule-based, object-oriented and procedural programming paradigms. Rule-based programming is the one that we used in our research.
- **Portability:** CLIPS is portable because it is written in C and it supports different operating system platforms such as Windows, MacOS X, and Unix. CLIPS comes with all the source code so it can be extended and modified as user's interests.
- **Integration/Extensibility:** CLIPS can be integrated with any procedural codes as a subroutine of the program written in C, FORTRAN, Java and ADA.
- **Fully Documented:** CLIPS is well documented open source program and has a growing online community. The source code comes with detailed documentation including a Reference Manual, Basic and Advanced User's Guides.
- **Low Cost:** CLIPS is free to use as an open source software.

Jess: Java Expert System Shell. Jess is yet another rule engine and expert system development environment which allows constructing rule and/or object based expert systems [35]. It is written in Java language and designed at Sandia National Laboratories based on CLIPS expert system shell. Jess provides user the ability of creating Java programs in order to reason on given knowledge base. When Jess is compared to other rule engines, it is found to be smaller, lighter and faster than most of them. One of the most powerful aspects of Jess is the capability of accessing all of Java's APIs. Jess uses Rete algorithm to match the rules to the facts and it makes Jess super fast when compared to *if.. then* statements in the procedural programs [47]. Jess is free for academic use only and it must be licensed for commercial use.

In the early stages of our research we used CLIPS in order to create an expert system in order to develop preliminary version of AUDIT. Besides some other advantages, we moved AUDIT design to Jess mainly because of the flexibility in creating database connections and availability of creating GUI interface in Java. These reasons for migration to Jess will also be discussed in Chapter 5.

CHAPTER 4

AUDIT: AUTOMATED DISK INVESTIGATION TOOLKIT

In this chapter we explain the early design of AUDIT tool in details. The content of this chapter was presented at the *Sixth International Conference on Digital Forensics & Cyber Crime* and published in the *Journal of Digital Forensics, Security and Law*. Here, we will show all the components of AUDIT and explain them in details. We also show how the knowledge base is created and used in AUDIT. As discussed in previous chapters, the most important property of AUDIT is the ability of automatically configuring, parameterizing and integrating the tools. Finally, we will discuss how AUDIT is used by users along with some simple testing cases of the program.

4.1 Preliminary Design of AUDIT

We designed AUDIT with the goal that very little technical knowledge would be required of the users. Given some high-level direction as to what the examiner is searching for, AUDIT is able to integrate and configure the tools automatically for the purpose of both general and specific investigations, searching the disk for evidence in graphic files, emails, documents, and "hidden" locations. Detailed search for items such as credit card and social security numbers can also be done.

AUDIT consists of three components: a database of investigative tasks and tools; a knowledge base with constructs defining rules and facts; and a core engine (expert system). The high-level design of AUDIT is shown in Figure 4.1.

We designed and implemented the domain specific knowledge base and the expert system to assist non technical users under two circumstances. First, when configuration and/or parameterization of the tools is needed, and especially when technical knowledge is involved to do this properly. Second, when tools integration is needed. By this we mean the order and use of multiple open source software tools to properly achieve the investigative task. Again, we assume the user may have very little technical knowledge about this.

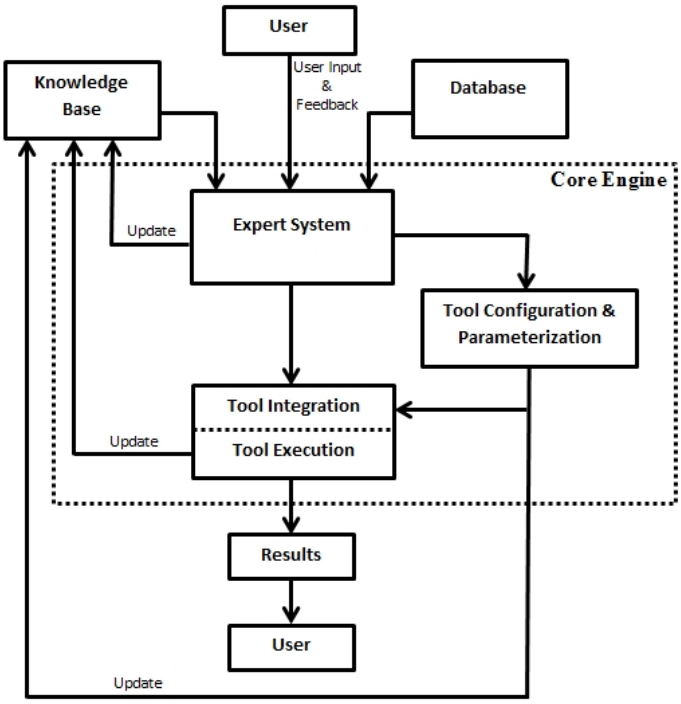
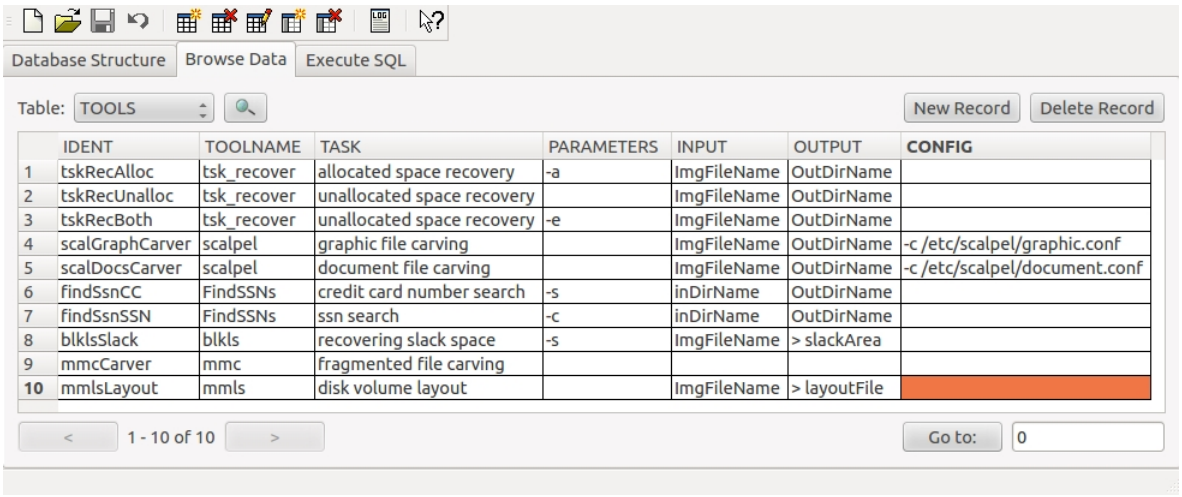


Figure 4.1: High-level design of AUDIT

The database component contains two tables that maintain information regarding the tools that will be used by AUDIT and the investigative tasks that an average investigator generally performs. In the first table, we have an entry for each tool that specifies a particular configuration and/or parameterization for different specific or general tasks. The entry also specifies other aspects such as the input requirements and the outputs of the tool with that configuration / parameterization, and the user expertise level. For example, in one entry, we have defined that the forensic tool *blkls* needs the disk image as an input from the user and needs parameter *'-s'* for searching the slack space. It is also specified that the output of *blkls* is redirected to another file in order to subsequently use other tools on the output data. Note that the user is not required to know what parameters to use in order to do slack space analysis or even what is slack space analysis search. See Figure 4.2. The second table currently simply contains a set of tasks (for example image search or credit card number search) that are linked to the knowledge base as well as the tools table.

The knowledge base contains facts and rules, some of which are predefined and embedded into the system and others that are created during the investigation. Facts and rules can be added,



Database Structure | Browse Data | Execute SQL

Table: TOOLS

New Record | Delete Record

	IDENT	TOOLNAME	TASK	PARAMETERS	INPUT	OUTPUT	CONFIG
1	tskRecAlloc	tsk_recover	allocated space recovery	-a	ImgFileName	OutDirName	
2	tskRecUnalloc	tsk_recover	unallocated space recovery		ImgFileName	OutDirName	
3	tskRecBoth	tsk_recover	unallocated space recovery	-e	ImgFileName	OutDirName	
4	scalGraphCarver	scalpel	graphic file carving		ImgFileName	OutDirName	-c /etc/scalpel/graphic.conf
5	scalDocsCarver	scalpel	document file carving		ImgFileName	OutDirName	-c /etc/scalpel/document.conf
6	findSsnCC	FindSSNs	credit card number search	-s	inDirName	OutDirName	
7	findSsnSSN	FindSSNs	ssn search	-c	inDirName	OutDirName	
8	blklsSlack	blkls	recovering slack space	-s	ImgFileName	> slackArea	
9	mmcCarver	mmc	fragmented file carving				
10	mmlsLayout	mmls	disk volume layout		ImgFileName	> layoutFile	

< 1 - 10 of 10 >

Go to: 0

Figure 4.2: The tools table in the AUDIT database

deleted and modified as needed. For example, at the beginning of an investigation by the user, the knowledge base is updated once the user enters information such as the input disk location, output directory, and investigative task. It is also updated after AUDIT finishes processing some tasks.

The core engine controls the running execution of the system using the database component, the knowledge base, and the user input. The core engine reads tool specifications and investigative tasks from the database and creates new rules and facts as needed. It also links the investigative tasks and the tools with respect to the knowledge base and user input and feedback.

For example, consider the actions of the core engine in Figure 1 after the expert system acquires all the inputs. Tool configuration and parameterization may be needed when the user wants to perform certain tasks. For example *scalpel* uses different configuration files for different categories of files that are to be carved (i.e. graphic files, document files, compressed files, etc.). The knowledge base would contain the information of which configuration file will be used in the desired search. These configuration files have been pre-designed for each target task. Parameterization does not require changing the configuration file but is important when running the tool. For example *tsk_recover* uses the parameter *'-a'* for allocated space analysis and this might be the setting that would be used initially when the core engine first invokes this tool.

After the configuration and/or parameterization, task specific tools are integrated in order to provide the requisite search capabilities. For example, we run *tsk_recover*, *blkls* and *scalpel* to provide complete search of the disk image to the credit card number search tool *Find_SSNs* which

is not designed to work on the raw disk image or the file system. Thus the core engine would have run the tools in this appropriate order.

In the next three subsections, we explain in more detail: (1) the design of the knowledge base; (2) the tools that are configured and integrated through the expert system; and (3) the user interface.

4.2 Building the Knowledge Base for AUDIT

The AI part of AUDIT is mainly the embedded expert system and knowledge base that is represented in it. In AUDIT, we used the open source expert system tool CLIPS which provides a complete platform to create rule and/or object based expert systems and is also used to represent an expert's technical knowledge [77]. Knowledge representation in CLIPS can be done by using different programming styles. We used rule-based programming which allows knowledge to be represented as heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation [77].

In AUDIT, knowledge is represented via rules and facts. A rule in CLIPS consists of two parts: IF and THEN "portions". In the IF portion of the rule, facts are listed that determine whether the rule is to be applied or not. A collection of facts is called a pattern and pattern matching is done by CLIPS to decide if the THEN portion is activated. In this case the rule is said to be active, else it is passive. If the facts hold (pattern matches), then actions in the THEN portion will be executed by the CLIPS inference engine. Multiple rules may be active at anytime and the ordering of execution can depend on the *"salience"* value in the IF portion. The IF portion of the rule has a different characteristic than an IF statement in conventional programs. It works as WHENEVER, because facts can be changed anytime during the program execution. The inference engine executes actions of all active rules [77]. In Figure 4.3, we show an example of a very simple rule used in order to illustrate how rules are used. Most of the actual rules used in AUDIT are more complex.

In this rule, the user is asked to provide his/her technical expertise and need of help for investigation. Based on the answer received from the user some certain facts will be added to the facts list by using the *assert* command of CLIPS. The IF portion of the rule consists of the two lines before the => symbol and the THEN portion of the rule is after that. This rule will be activated when we have no information about the user's expertise. The code line in Figure 4.4 is added to

```
(defrule determine-investigator-level ""
    (declare (salience 10))
    (not (investigator is ?))
    =>
    (if (yes-or-no-p "Are you an expert (yes/no)? ")
    then (assert (investigator is expert))
         (if (yes-or-no-p "Do you need help (yes/no)? ")
         then
             (assert (expert needs help))
             (assert (determine disk-layout needed))
             (assert (extracting partitions from disk needed))
         else
             (assert (expert needs no-help))
             (assert (self usage mode on))
             (assert (provide available tool list to user)))
    else (assert (investigator is non-expert))
         (assert (non_expert needs help))
         (assert (determine disk-layout needed))
         (assert (extracting partitions from disk needed))))
```

Figure 4.3: Simple Example of a CLIPS Rule in AUDIT

```
(declare (salience 10))
```

Figure 4.4: Salience declaration rule

the rule to make sure this rule will be processed before all other active rules by declaring *salience* value to 10 which is the highest value we used.

The higher the value of *salience*, the earlier the execution of the rule happens.

In AUDIT we have defined two different levels of knowledge: Investigator Level and Tools Level. These levels includes initially defined knowledge and new knowledge that is created based on previous knowledge and new knowledge created by use of tools and feedback from the user.

4.2.1 Investigator Level Knowledge

Investigator level knowledge relates to the technical skill level of the user. This is defined to be either non-expert or expert. When AUDIT starts, the user is asked about their level of technical expertise. In the rest of this section we will mostly focus on explaining how AUDIT works and technically assists non-expert practitioners. Depending on the user's technical skills, some certain facts are added to the fact list. For example, if we determine that the user is a non-expert, then we start adding new facts (inside parentheses in CLIPS, see Figure 4.5) to the initial knowledge base:

```
(investigator is non-expert)
(non_expert needs help)
(configuration needed)
```

Figure 4.5: Facts initially added to the knowledge base

```
(run tsk_recover for allocated-space)
(run tsk_recover for unallocated-space)
(run blkls for slack-space)
(run scalpel for data-carving)
(configure scalpel for graphic-files)
(configure scalpel for document-files)
(configure mmc for smart-carving)
```

Figure 4.6: New facts added to the knowledge base

Of course this new knowledge may trigger other rules in the rules list to be activated. The fact *(configuration needed)* triggers the other facts in Figure 4.6 being added.

Addition of new facts may not necessarily activate a rule since there might be other facts that are required to match the pattern. For instance, activation of the "data carving" rule is based on the user being non-expert, the type of investigation, completion of analysis of the file system (including allocated, unallocated and slack space) and negative feedback. Negative feedback means that during user interaction AUDIT determined that the user did not find evidence of interest from the previous analysis. It is very useful to keep almost all related knowledge in the knowledge base even though it might not activate rules right away. For example, we do not have to add allocated and unallocated space analysis in distinct facts, but doing so we can make sure that our system includes knowledge of different parameters for use of *tsk_recover* to perform analysis on both allocated and unallocated spaces.

4.2.2 Tools Level Knowledge

Tools level knowledge in AUDIT relates to usage and integration of the tools. One example of the use of this knowledge is to provide some information for one or more tools which are not

```
(defrule credit-card-search ""
    (image-file-path is ?imagePath)
    (output-path is ?outputPath)
    (investigative-task is ?task)
    (investigation-type is ccsearch)
    =>
    (printout t "Find_SSNs is running on the disk!" crlf)
    (mount-disk-image ?imagePath ?outputPath ?task)
    (run-blkls ?imagePath ?outputPath ?task)
    (run-strings ?imagePath ?outputPath ?task)
    (run-tsk_recover ?imagePath ?outputPath ?task)
    (run-Find_SSNs ?imagePath ?outputPath ?task)
    (assert (ccsearch performed)))
```

Figure 4.7: The rule runs for credit card search

originally designed to gather that information from the given disk. AUDIT provides this information through running other useful tools. For example, *TSK* is not designed to carve out files from a disk image when file system metadata information is lost or damaged. Therefore, we run *scalpel* and *mmc* (multimedia file carver) [72] tools to carve out files which could be both fragmented and unfragmented. The Figure 4.7 shows a high-level rule which in turn causes other rules to run. These rules integrate different tools in order to provide available search places on the disk image to the credit card number search tool. Each of the asserted lines between last *printout* and *assert* are the function calls for each tool to work with the specific parameters passed (such as *"?imagePath"*). Other information needed for the function is obtained from the knowledge base.

4.3 Configuration, Parameterization and Integration of Tools

The open source command line tools that we used in this initial version of AUDIT are *tsk_recover*, *blkls*, *mmls*, *scalpel*, and *Find_SSNs*. In our integration, we also used Linux commands such as *strings*. We briefly explain characteristics of those tools and show how we perform the integration of the tools within the expert system. As previously discussed, in order to use some of the above

tools for a specific task we need to either configure or parameterize these tools. This is also discussed in the relevant tools section.

tsk_recover is a command line tool in TSK (The SleuthKit). Depending on the parameters given, it extracts allocated and/or unallocated files from a disk image to a local directory [23]. We use *tsk_recover* for all of the search techniques that we used. We use parameter *'-a'* in order to extract files from allocated space since it runs on unallocated space in default.

mmls is a command line tool under TSK. It provides layout of the given disk image and prints out volume system contents. We use *mmls* to find out each partition location for use of other tools.

blkls is also a command line tool under TSK. It lists the details about data units (e.g. block, cluster, fragment etc.) and can extract the unallocated space of the file system [23]. The main purpose of integrating *blkls* in AUDIT is to extract the slack space of the disk image by using the parameter *'-s'*. After retrieving the slack space AUDIT uses the Linux command *strings* and sets the parameter *'-n'* to 1 to write all printable characters to a text file. We set *'-n'* to 1 because it is possible that targeted numbers may be obscured with whitespace(s) between each digit. The new text file can then be used by other tool for content analysis to gather credit card and social security numbers.

scalpel is a very successful file carver designed based on another carver tool *foremost* version 0.69. *scalpel* reads header and footer information of files in order to recognize files in the given media based on the pre-configured configuration file. If any specified type of file is found, it carves out the file and write it to the given output directory. Since *scalpel* checks header and footer for specific magic numbers, it is file system independent and it carves files from FATx, NTFS, ext2/3, HFS+, or raw partitions. We integrated *scalpel* into AUDIT in order to retrieve files from a disk image when file system metadata does not exist or is damaged. *scalpel* is successful for unfragmented files therefore we also used *mmc* when the files are fragmented. We use two different pre-defined configuration files for two categories: document files (i.e. doc, pdf, xls, rtf, etc.) and graphic files (i.e. jpg, png, gif, psd, etc.).

We give a detailed example of the configuration and use of *scalpel* for picture and document search to more clearly illustrate how the knowledge and rules are used in AUDIT. The facts in Figure 4.8 are assumed to have been added to the knowledge base:

The first fact is feedback from the user whether any evidence or interesting file is found or not.

```
(evidence found no)
(run scalpel for data-carving)
(tsk_recover is unsuccessful)
(image-file-path is ?imagePath)
(output-path is ?outputPath)
(investigation-type is psearch)
```

Figure 4.8: Facts assumed to be added to de knowledge base

The second fact is part of the initial knowledge that shows which tool to use for data carving. The next fact is true when *tsk_recover* was run on both allocated and unallocated spaces and it failed to find any useful file for investigator. The forth and the fifth facts stand for path of the target disk image and output directory for results to be saved. The last fact is used to hold the search type which is picture search for this example. If all of these facts are true in the fact list, this means the pattern for performing data carving matches and the actions in Figure 4.9 will be taken:

After this rule is activated and run by the inference engine, the user is again asked to provide feedback regarding the success of this particular process. Based on the feedback given, AUDIT creates new facts and updates the knowledge base.

The last tool that we use in AUDIT is *Find_SSNs* which is introduced in Chapter 3. *Find_SSNs* uses multiple and comprehensive validation steps to make sure the credit card number is a valid number. As for the social security numbers, it uses data from Social Security Administration to guarantee that valid association between area number and group number is found for the number [51]. The patterns of the numbers that AUDIT searches using *Find_SSNs* are as follows:

For SSN: #########, ###-##-####, ### ## ####

For CCN: #(13,16) with dashes or spaces anywhere

4.4 Working with AUDIT

AUDIT interacts with users via the CLIPS expert system shell. The user is asked to specify his/her technical expertise level, define the disk image file and specify an output directory for results to be saved by AUDIT. Then the user is asked to select what type of search he/she wants to perform. As discussed before, AUDIT works on picture search, financial document search, email

```
(deffunction run_scalpel (?imagePath ?outputPath ?configuration)
   (if (eq ?configuration graphic)
      then
        (system ?carverTool ?imagePath " -o " ?outputPath
                "/scalpel/ " ?graphicConfigFileName)
        (assert (data carving done))
        (system "nautilus " ?outputPath "/scalpel/")
      else
               (system ?carverTool ?imagePath " -o " ?outputPath
                       "/scalpel/ " ?documentConfigFileName)
               (assert (data carving done))
               (system "nautilus " ?outputPath "/scalpel/")))
```

Figure 4.9: Function used for invoking scalpel for graphic file carving

search and sensitive number search. The starting screen of the user interface of our prototype implementation of AUDIT is shown in Figure 4.10.

We are currently categorizing searches conducted by forensic examiners into general, intermediate and specific. Picture search is an example of a general search and we have implemented it in AUDIT because it is one of the most important searches that investigators are interested in. Our goal was to first have AUDIT do many of the general searches that investigators would do as discussed in [46]. Credit card and social security numbers search on the other hand is a specific search and is implemented in AUDIT in order to show how our tool integration model can be applied to a very specific search task. Credit card number search might not be direct evidence for an investigation but could lead the investigator to other evidence. Given that a sophisticated specific open source tool is available, we show how it can be integrated into our system. These specific search tools can be incorporated into AUDIT over time. We also wanted to address what we term an intermediate search problem and we labeled financial document search in this category. Our goal in part for this classification was to see if there were different requirements that were needed when adding the different classes of tools into AUDIT.

When the user selects one of the search options from the list of available tasks, the related expert

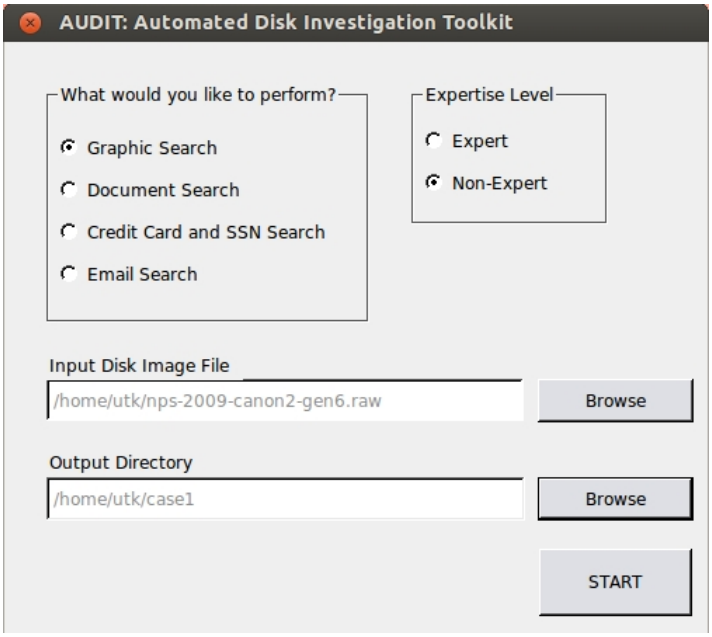


Figure 4.10: Starting screen of the user interface of AUDIT

system knowledge is processed by AUDIT. The represented knowledge regarding which tool will be used and how it will be used are embedded in AUDIT and pulled from the database. Predefined rules are added to the inference engine of CLIPS based on the user's search selection.

If the user chooses to search sensitive numbers on the disk image, AUDIT mounts the disk image to the system and recovers files from both allocated and unallocated spaces. Files that potentially have text will also be carved from the disk image. After the data retrieval, *Find_SSNs* starts running on both mounted disk and retrieved files. *Find_SSNs* creates both html and text files for the user's view and continues working with respect to user's feedback. The tool integration and an expert system knowledge use for this example is explored further in the next section.

Until this point of the investigation, the only questions that is asked from the user is providing the input image and the output directory in addition to feedback. Feedback is basically whether any forensically interesting data related to the investigation was found or not and whether the examiner wants to continue to do a deeper investigation.

4.5 Testing AUDIT

Our current version of AUDIT runs on Ubuntu 12.04 LTS. In this section we will present two simulated cases (picture search and sensitive number search) that we used to test AUDIT. We used the NPS test disk images from Digital Corpora [36] and also Brian Carrier's digital forensics tool testing images [13]. We also used some disk images that we created by adding files from Digital Corpora Govdocs1 [36].

4.5.1 Graphic Files Search

When AUDIT starts, it asks user to provide the input disk image, the output directory path for results, and the user level of expertise. When the user selects graphic files search, AUDIT first starts *mmls* tool in order to figure out the content of the volume system. It gets all the partitions and their starting and ending sectors. By doing so, AUDIT becomes able to work on each partition by separating them using the *dd* command line tool if there are multiple partitions.

After getting the image disk and the partition location (assuming there is one partition on the disk), AUDIT starts file system analysis on the partition since the file system is the area where evidence is mostly searched for [14] by investigators. AUDIT automatically provides the required parameters (input file, output directory, *'-a'* for allocated space search, and *'-o'* for sector offset gathered from mmls) for *tsk_recover* in order to start analyzing the allocated space of the partition. For presenting results to the examiner, AUDIT provides directory structure of the partition similar to what Carrier's tool Autopsy [23] does. It classifies the recovered files by file type and lets the user check whether any forensically interesting graphic file exists. At this stage of the process, the user is provided high level information regarding where the files are found. The examiner is also given an option to do deeper investigation for more information. If the examiner does not want to go step by step but would rather do a search of all possible areas on disk (allocated space, unallocated space, data carving, and slack space) this can be done by AUDIT at once in any stage of the process.

Assuming the user would like to go to the next stage, AUDIT starts *tsk_recover* with the required parameters as mentioned above except parameter *'-a'*, since *tsk_recover* works on unallocated space by default. AUDIT returns directories and files to the user from unallocated space. See Figure 4.11. AUDIT then informs the user that deleted files were recovered from the disk image instead of using

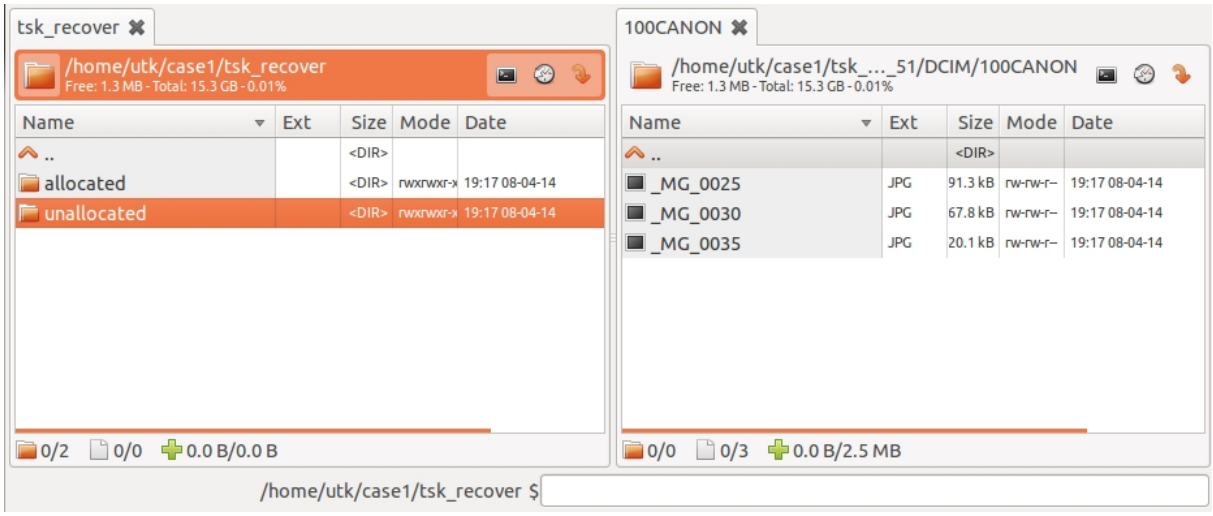


Figure 4.11: Popup showing files recovered from unallocated space

the term unallocated space since the user's knowledge level is non-expert. If the user informs AUDIT that there is still no interesting data, AUDIT continues to a deeper analysis and starts recovering files from the slack space.

AUDIT uses the *blkls* tool in order to get the total file slack area of the disk image and creates another disk image from it. Then, it runs *scalpel* on the new image file in order to carve any hidden graphic file. If found, the user is informed with the list of hidden images that are found in this unconventional area of the disk image.

During all of the above stages, AUDIT updates the knowledge base and the expert system uses that knowledge whenever it is applicable to any rule. In this test we showed how tools are configured and parameterized via the expert system and knowledge base. In the next example we will present how tools are integrated for a specific search purpose.

4.5.2 Sensitive Number Search

One of the search options that AUDIT provides to users is sensitive number search and specifically credit card and social security number search. This search type is activated and the related knowledge base updated in the expert system after the user selects the sensitive number search option.

As explained in Section 4.3 we primarily used *Find_SSNs* tool in order to find sensitive numbers on the disk image. This test case is a good example of how AUDIT integrates different tools for

Host 127.0.1.1 | Tue_Apr_08_2014_18_25_19_PM | Total Suspect Numbers 143 | Search For SSNs + CCNs|

Suspect Number Count	File Extension	File Path
2	.DOC	/home/utk/caseSSN/mnt/carvedFiles/DOC-5-0/00000003.DOC
5	.DOC|DOT|	/home/utk/caseSSN/mnt/carvedFiles/DOC|DOT|-11-0/00000268.DOC|DOT|
2	.DOC	/home/utk/caseSSN/mnt/carvedFiles/DOC-7-0/00000098.DOC
2	.XLS	/home/utk/caseSSN/mnt/carvedFiles/XLS-35-0/00000704.XLS
5	.XLS	/home/utk/caseSSN/mnt/carvedFiles/XLS-36-0/00000758.XLS
2	.DOC|DOT|	/home/utk/caseSSN/mnt/carvedFiles/DOC|DOT|-11-0/00000228.DOC|DOT|
2	.DOC|DOT|	/home/utk/caseSSN/mnt/carvedFiles/DOC|DOT|-11-0/00000229.DOC|DOT|
6	.DOC	/home/utk/caseSSN/mnt/carvedFiles/DOC-5-0/00000043.DOC
2	.DOC	/home/utk/caseSSN/mnt/carvedFiles/DOC-5-0/00000001.DOC

Figure 4.12: Find_SSNs output report for Credit Card and Social Security Numbers

a specific purpose because *Find_SSNs* is not originally designed to work on various places that AUDIT makes available for it.

Find_SSNs is not originally designed to work on disk images or raw data directly, therefore it needs the disk image being mounted to the file system in order to make files and directories available for sensitive number search. Since this requires technical knowledge of the user, AUDIT performs mounting via its knowledge base. Mounting the disk image however does not make available all space on the disk. AUDIT however makes sure that all reachable space of the disk image is made available for the search including data in the file system, unallocated space, and slack space. In order to provide all of this information to *Find_SSNs*, we use *tsk_recover* with parameter *'-e'* to extract files from both allocated and unallocated spaces. We also integrate *scalpel* and *mmc* tools to perform data carving on the given disk image for both fragmented and unfragmented files. As discussed above *blkls* is used to make data in the slack space available for *Find_SSNs*. All of this is done automatically by AUDIT without any further input from the non-expert user.

After AUDIT integrates and runs all the tools, *Find_SSNs* runs on all the available spaces and generate a report for the user. The report is created in both html and txt format for the user's analysis. Example of an html report can be seen in Figure 4.12.

CHAPTER 5

ADVANCED AUTOMATED DISK INVESTIGATION TOOLKIT

In Chapter 4 we have presented a design for a toolkit that can be used as an automated assistant for forensic investigations. In this chapter we expand on that work and describe a more advanced system. The new AUDIT is designed using the Java Expert Systems Shell (Jess). The most important internal change is the ability to dynamically update the database component with information from the expert system component which substantial extends the capability of our system. We have also developed a reporting mechanism that reports on activities of the system including inferences about decisions made which is useful when explaining how AUDIT is working. In addition we have developed a hierarchical disk investigation model that eventually could give guarantees on what parts of the disk have been successfully analyzed.

We now start explaining the major components of AUDIT in its new design. Then, we explain a novel hierarchical disk investigation design that will help examiners to conduct their investigations in a complete sense. As the last part of this chapter, we will discuss how our new reporting mechanism is developed and how it works in AUDIT.

5.1 New High-Level Design of AUDIT

The high level design of AUDIT is shown in Figure 4.1. It consists of three components: a database of tasks and tools, a knowledge base and a core engine (that includes an expert system, a tools integration component and a configuration component). The elements in bold indicate major changes from the previous version of AUDIT. Bold boxes show new or substantially changed components. Bold connections show new update capabilities.

Now, we explain the updates in details in the next sections. However, we leave the discussion on the reporting capability to Section 5.3.

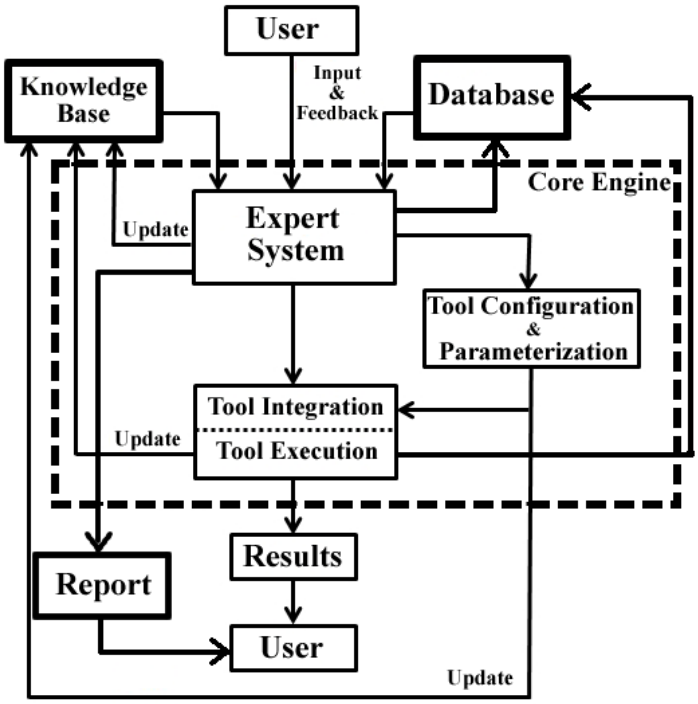


Figure 5.1: High-level design of AUDIT

5.1.1 Database Component

The database component maintains information regarding the tools that will be used by AUDIT and the investigative tasks that an average investigator generally performs. Table 5.1 shows the schema of the database and Figure 5.2 shows the sample entries in the new tools table in the database. We have an entry for each tool that specifies a potential configuration and/or parameterization for different specific or general tasks. The entry also specifies other aspects such as the input requirements and the outputs of the tool with that configuration / parameterization. Note that some of the configuration parameter values are fixed whereas others are variables that can be defined during execution.

For example, in the *emailsBulkExt* entry, we have defined that the forensic tool *bulk_extractor* needs parameter *'-x'*. Bulk_extractor can use different scanners for various types of information such as email addresses, credit card numbers, etc. Parameters *'-x'* in column *'p_conf'* and *'all'* in column *'config'* disable all scanners. Parameters *'-e'* in column *'p1'* and *'email'* in column *'p2'* enable the scanner for email address search.

Table 5.1: Schema of the tools table in the AUDIT database

Identifier	Type	Length	Description
ident	VARCHAR	50	A unique identifier for each entry
toolname	VARCHAR	50	An actual name of a tool
task	VARCHAR	50	The task of a certain tool
params	VARCHAR	50	Initial parameters needed for a tool
p_in	VARCHAR	50	Input parameter of a tool
input	VARCHAR	100	Input of a tool
p_out	VARCHAR	50	Output parameter of a tool
output	VARCHAR	100	Output of a tool
p_config	VARCHAR	50	Configuration parameter of a tool
config	VARCHAR	100	Configuration file of a tool
p1	VARCHAR	100	Other parameters may be used
p2	VARCHAR	100	Other parameters may be used
p3	VARCHAR	100	Other parameters may be used
S	INTEGER	10	Total number of successes of a tool for a certain task
F	INTEGER	10	Total number of failures of a tool for a certain task
R	INTEGER	5	Ranking value of a tool for a certain task

ident	toolname	task	params	p_in	input	p_out	output	p_conf	config	p1	p2	p3	S	F	R
blklsSlack	blkls	recovering_slack_space	-s	N/A	?imagePath	>	?outputPath	N/A	N/A	N/A	N/A	N/A	0	0	0
blklsUnalloc	blkls	recovering_unallocat...	-A	N/A	?imagePath	>	?outputPath	N/A	N/A	N/A	N/A	N/A	0	0	0
documentFile...	photorec	document_files_search	N/A	N/A	?imagePath	/d	?outputPath	/cmd	fileopt,	everythi...	doc,en...	freespac...	0	0	0
documentFile...	photorec	document_files_search	N/A	N/A	?imagePath	/d	?outputPath	/cmd	fileopt,	everythi...	doc,en...	wholesp...	0	0	0
emailFilesPR	photorec	email_files_search	N/A	N/A	?imagePath	/d	?outputPath	/cmd	fileopt,	everythi...	pst,ena...	wholesp...	0	0	0
emailsBulkExt	bulk_extractor	email_address_search	N/A	N/A	?imagePath	-o	?outputPath	-x	all	-e	email	N/A	0	0	0
emailsMultigrep	perl src/tools/...	email_address_search	N/A	N/A	?outputP...	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0
fileFilterSorter	./src/tools/filt...	filtering_sorting_files	N/A	N/A	?outputP...	N/A	/sortedOut...	N/A	src/tools/...	N/A	N/A	N/A	0	0	0
fileSystemType	fsstat	getting_filesytemm_t...	-t -o	?img...	?imagePath	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0
findCC	FindSSNs	cc_number_search	src/to...	-p	?outputP...	-o	?outputPath	-t ht...	N/A	N/A	N/A	N/A	0	0	0
findSSN	FindSSNs	ssn_search	src/to...	-p	?outputP...	-o	?outputPath	-t ht...	N/A	N/A	N/A	N/A	0	0	0
findSSNCC	FindSSNs	ssn_cc_search	src/to...	-p	?outputP...	-o	?outputPath	-t	html	-a	N/A	N/A	0	0	0
graphicFilesFr...	photorec	graphic_files_search	N/A	N/A	?imagePath	/d	?outputPath	/cmd	fileopt,	everythi...	jpg,ena...	freespac...	0	0	0
graphicFilesPR	photorec	graphic_files_search	N/A	N/A	?imagePath	/d	?outputPath	/cmd	fileopt,	everythi...	jpg,ena...	wholesp...	0	0	0
mmcCarver	mmc	fragmented_file_carvi...	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0	0	0
mmlsLayout	mmls	disk_volume_layout	N/A	N/A	?imagePath	>	layout.txt	N/A	N/A	N/A	N/A	N/A	0	0	0
scalpelDocum...	scalpel	document_file_carving	N/A	N/A	?imagePath	-o	?outputPath	-c	src/confi...	N/A	N/A	N/A	0	0	0
scalpelEmail	scalpel	document_file_carving	N/A	N/A	?imagePath	-o	?outputPath	-c	src/confi...	N/A	N/A	N/A	0	0	0
scalpelGraphics	scalpel	graphic_file_carving	N/A	N/A	?imagePath	-o	?outputPath	-c	src/confi...	N/A	N/A	N/A	0	0	0
scalpelIdentity	scalpel	document_file_carving	N/A	N/A	?imagePath	-o	?outputPath	-c	src/confi...	N/A	N/A	N/A	0	0	0
tskRecAlloc	tsk_recover	allocated_space_reco...	-a	N/A	?imagePath	N/A	?outputPath	N/A	N/A	N/A	N/A	N/A	0	0	0
tskRecBoth	tsk_recover	alloc/unalloc space re...	-e	N/A	?imagePath	N/A	?outputPath	N/A	N/A	N/A	N/A	N/A	0	0	0
tskRecUnalloc	tsk_recover	unallocated_space_re...	N/A	N/A	?imagePath	N/A	?outputPath	N/A	N/A	N/A	N/A	N/A	0	0	0

Figure 5.2: Sample entries in the new tools table in the AUDIT database

In the previous version of AUDIT this database table was static and could not be updated with the knowledge that is collected during the investigation.

In the current AUDIT, this database is initially read by the expert system. Then, fields filled with keywords such as *"?imagePath"* are recognized by the expert system and then changed as

```
(defrule update-knowledge-base-rule "updating knowledge base"
  (image-file-path is ?imagePath)
  (output-path is ?outputPath)
  ?toolInfo <- (tools (toolname ?tName)(input ?in)(output ?out))
  =>
  (if (eq ?in "?imagePath") then (modify ?toolInfo (input ?imagePath))
    else
      (if (eq ?in "?outputPath")
        then (modify ?toolInfo (input ?outputPath))))
  (if (eq ?out "?outputPath")
    then (modify ?toolInfo (output ?outputPath))))
```

Figure 5.3: A rule used for updating the knowledge base

```
(MAIN::tools (ident "emailsBulkExt") (toolname "bulk_extractor")
  (task "email_address_search") (input "?imagePath") (p_out "-o")
  (output "?outputPath") (p_conf "-x") (config "all") (p1 "-e") (p2 "email"))
```

Figure 5.4: Original fact before updating

needed with the related values collected during the investigation. Note that the fields filled with *"N/A"* are used in order to help the expert system to recognize empty fields correctly.

5.1.2 Knowledge Base Component

The knowledge base contains facts and rules, some of which are predefined and embedded into the system and others that are created during the investigation. Facts and rules can be added, deleted and modified as needed. In expert systems facts can be quite simple such as *"(John is male)"* or more complex such as *"(?person (gender ?gen)(age 25)(weight 180))"*.

In the current version of AUDIT we use complex facts not used in the previous version. These facts are typically modified frequently with update rules as shown in Figure 5.3. When a user enters the input and output of the case, all the tools' input and output values are changed. Figure 5.4 and Figure 5.5 show a fact in the knowledge base before and after the update respectively.

```
(MAIN::tools (ident "emailsBulkExt") (toolname "bulk_extractor")
  (task "email_address_search") (input "/home/utk/part5.img") (p_out "-o")
  (output "/home/utk/out1") (p_conf "-x") (config "all") (p1 "-e") (p2 "email"))
```

Figure 5.5: Modified fact after updating

5.1.3 Core Engine Component

The core engine controls the running execution of the system using the database component, the knowledge base, and the user input. The core engine reads tool specifications and investigative tasks from the database and creates new rules and facts as needed. It also links the investigative tasks and the tools with respect to the knowledge base and user input and feedback.

```
(defrule update-database-rule "updating the database"
  (image-file-path is ?imagePath)
  (output-path is ?outputPath)
  ?toolInfo <- (tools (toolname ?tName)(input ?in)(output ?out))
  =>
  (updateDatabase ?imagePath "?imagePath"
    "UPDATE TOOLS SET input = ? WHERE input = ?")
  (updateDatabase ?outputPath "?outputPath"
    "UPDATE TOOLS SET output = ? WHERE output = ?"))
```

Figure 5.6: A rule used for updating the database

In the current AUDIT the database is updated dynamically when related new information is gathered by the expert system and after the tools are executed. The update process is performed by the core engine via updating rules. For example, Figure 5.6 shows the update rule that is used to update the database when input and output paths are entered by a user. By doing so, previously loaded incomplete data (variables used with *?<variableName>* structure) from the database become completed and get ready for being used by other rules. Regarding the update after the tool execution, AUDIT asks the user whether a tool performs successfully or it fails. Based on the feedback received from the user, related fields (S, F and R) for that tool are updated in the

database. This information can actually be used to compare similar tools against each other. We do not discuss this further in this chapter and leave it to Chapter 7.

5.1.4 Expert System Design in Jess

One of the important changes we made from the previous version of AUDIT is that we migrated our expert system shell from CLIPS to JESS (see Section 3). Jess is written in Java and has the capability of accessing all of Java's APIs. The reasons for migration include:

- **GUI user interface:** CLIPS does not have a mechanism to allow users to create a GUI interface for the expert system applications. Although extensions to CLIPS (e.g. PyCLIPS [40] and wxCLIPS [75]) exist for this purpose they are not maintained well, have limited access to CLIPS functionality and are not used by a large global community.
- **External user functions:** CLIPS and Jess both support users in creating their own external functions. However, CLIPS requires recompilation to integrate these functions whereas Jess provides direct access to any external function from a file that contains Jess constructs without recompilation.
- **Database support:** CLIPS does not have direct support for relational database management systems. Jess however has full support for most common relational database management systems (e.g. SQLite [1], MySQL [67], and PostgreSQL [44]) because many are available in Java libraries.

When AUDIT starts execution, it connects to the database and reads all the data from it. The data is entered into a Jess template called *tools*. When a specific input, output and task is selected by a user, AUDIT starts running to collect certain information (e.g. partitioning on disk, disk volume type, disk's physical sector size, etc.) about the input disk. All of this information is also entered into two different templates called *disk_layout* and *diskInfo* located in the knowledge base. Figure 5.7 shows an example of a Java code explaining how a Jess template is created in the Java environment. All the Jess templates mentioned above keep knowledge about the tool usage and input disk information in the knowledge base.

AUDIT uses the data from the *'task'* column in Figure 5.2 in order to activate the rules when a specific task needs to be performed using a particular tool. For instance, Figure 5.8 shows a rule that gets relevant information from the database in order to use bulk_extractor for *email address search*. The line starting with *"(tools (ident ...)"* represents the pattern located in the tools

```
Deftemplate diskInfo = new Deftemplate("diskInfo", "disk information", engine);
diskInfo.addSlot("volumeType", Funcall.NIL, "STRING");
diskInfo.addSlot("sectorSize", Funcall.NIL, "INTEGER");
diskInfo.addSlot("blockSize", Funcall.NIL, "INTEGER");
diskInfo.addSlot("numPartitions", Funcall.NIL, "INTEGER");
```

Figure 5.7: Creating a Jess template in Java

template which is populated with the information from both the database and the knowledge base. *(toolname ?tName)* is one of the *slots* declared in the *tools* template, and *?tName* is a variable name declared for this slot in order to keep the tool's name.

```
(defrule find-email-addresses-BE "Search for e-mail addresses only"
    (search type is 4) ; type 4 is for email address search
    (output-path is ?outputPath)
    (image-file-path is ?imagePath)
    (tools (ident "emailsBulkExt")(toolname ?tName)(task "emails_address_search")
      (params ?params)(p_in ?par_in)(input ?input)(p_out ?par_out)
      (output ?output)(p_conf ?par_conf)(config ?config)(p1 ?p1)(p2 ?p2)(p3 ?p3))
    =>
    (my-system-bulk-ex ?tName ?params ?par_in ?input ?par_out
      (str-cat ?outputPath "/bulk_ext") ?par_conf ?config ?p1 ?p2 ?p3)
    (assert (bulk-extractor is called)) ; fact added to the KB
    (gui-response
      "Did you find interesting data for your EMAIL investigation (yes/no)?"))
```

Figure 5.8: A rule used in AUDIT for email address search

In Figure 5.8, it is specifically mentioned that the currently selected task is email address search. This is done by making the *(task "emails_address_search")* slot as part of the pattern. In case of multiple available tools for the same type of task we also add another slot, *(ident "emailsBulkExt")*, to make the rule run for a specific tool only. The same rule can thus be used for other tools which

```
bulk_extractor /home/utk/part5.img -o /home/utk/t2/bulk_ext -x all -e email
```

Figure 5.9: Executing bulk_extractor for email address search

perform the same task by specifying the *ident* slot, for example *(ident "emailsMultigrep")*. This rule can in fact be extended by changing the object of ident to a variable, thus allowing multiple tools to run or choosing a specific tool based on other knowledge. The Linux command line shown in Figure 5.9 shows the actual command that AUDIT runs for the selected tool after the rule in Figure 5.8 is activated.

The Linux command line shown in Figure 5.9 shows the actual command that AUDIT runs for the selected tool after the rule in Figure 5.8 is activated.

The new capabilities of AUDIT are illustrated by our ability to implement examination of multi partition disks. The knowledge base has to be changed dynamically in the analysis of these disks because each of the partitions in the disk has to be treated as an individual disk image. The new dynamic database and knowledge base capabilities allow us to do this efficiently.

5.2 Model of Hierarchical Disk Investigation

In this section, we present a new hierarchical disk investigation model which leads AUDIT to systematically examine the disk in its totality based on its physical and logical structures (e.g. sectors, partitions, file systems, blocks, etc.). In this model, AUDIT runs tools in a *Top-Down* analysis hierarchy based on how digital data is designed on disk. *Top* in this context refers to the level where the highest level of information is available while *Down* refers to the lowest level where information is at the byte level of the disk. A specific version of a similar hierarchy is also discussed in [14] for file system analysis only. However, our model is designed to automate the tools to analyze all possible (hidden or unhidden) areas in the disk if there is an open source tool available for use.

The data in a hard disk drive are stored in different physical and logical levels. Depending on the techniques and the spaces that data are located, investigators may access forensically useful information easily and faster. For example, it is very likely that an investigator finds detailed information at the file system level. If a file system is not corrupted then a file's metadata information such as authorship, creation/modification date, or even where the file was created (for image

files) can be easily accessed. This may reduce the time that is spent for analysis and may also avoid needlessly deepening the investigation into lower levels such as data carving which provides no metadata information. On the other hand, when the file system is corrupted or information is hidden in an unconventional area (file slack, unallocated space, etc.) investigators are required to search all possible places for evidence. In such a case, there is no standard way of searching the whole disk in its totality. Our proposed model aims to search the whole disk for a chosen task by using the tools systematically so that each structure is analyzed for evidence. What we mean with analyzing the whole disk in its totality is that the mathematical union of the all areas that are searched for evidence by AUDIT must give the total size of the disk.

5.2.1 The Model

Our architecture consists of 7 different levels called *granules* (see Figure 5.10). Granule is a term that we use to define the abstract level of the meta information available and ranges from the coarsest level (*granule 1*, disk image) to the finest level (*granule 7*, bytes). Each deeper level in the Figure 5.10 is of finer granularity. During analysis we first *recognize* granular structures creating the model of Figure 5.10 while also doing appropriate *examination* of the disk at that level. Consider the analysis for example at the file system level. At this granule level, AUDIT runs certain digital forensics tools that attempt to find a file system. When AUDIT runs the tools on the disk image, it checks if it can recognize all file system structures. The tools could either recover information at that level, say a normal file system or it might recover a corrupted file system on another part of the disk. For file systems found, the recognition process proceeds to the next deeper level. As for the remainder of the disk at the current level, the unrecognized (as a file system) part is simply marked and examined at the finer level (granule 6, carved files). If evidence is still not found, it is then examined at the lowest level (granule 7, bytes). Note that our goal is to fully carve the disk into various recognizable parts and examine every part of the disk with an appropriate tool.

In *granule 1*, we have the whole disk image as raw data. AUDIT could start file carving at this lowest level however it instead deepens the investigation to find other structures. Therefore, it runs volume system analysis tools *tsk_loaddb* and *mmstat* to gather detailed information about the disk (whole volume). For example, AUDIT gets the disk layout and retrieves some of the disk level information such as sector size of the disk. This may be useful in the later examination because other tools could require the user to enter the sector size. AUDIT also figures out the exact location

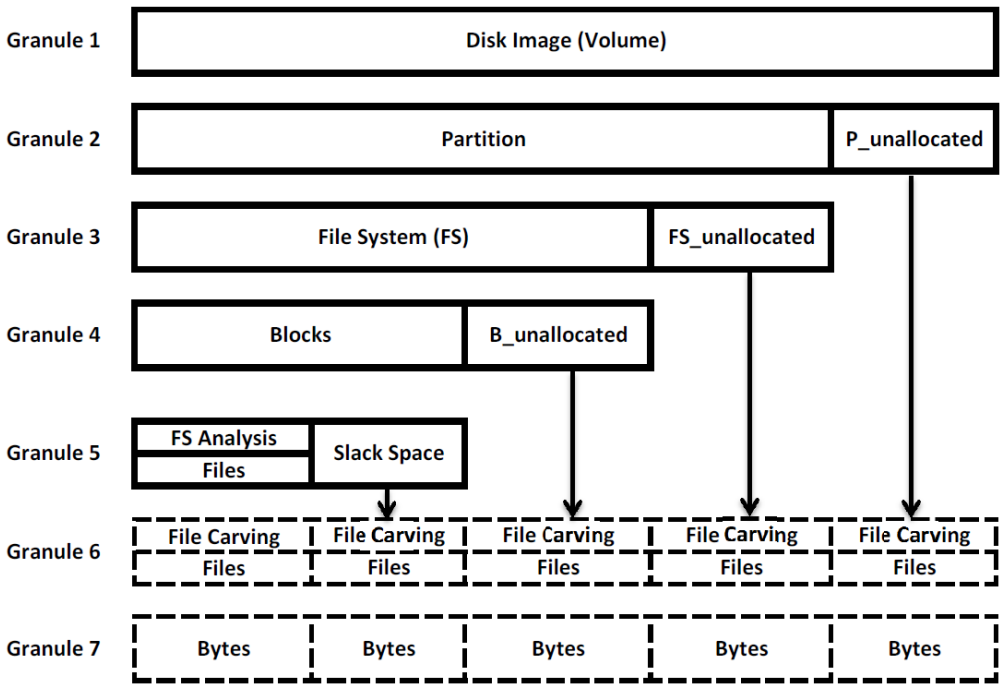


Figure 5.10: Model of hierarchical disk analyses

of the disk structures (allocated and unallocated) on the disk. This is known to AUDIT by learning the starting sectors (offset) and length of the structures from the disk layout. Once AUDIT knows the complete disk layout, it uses *dd* in order to divide the whole volume into sub-parts which will be further examined in later granules.

The volume system tools in AUDIT classify all the structures as either allocated or unallocated. This classification is also useful for us because we designed AUDIT to treat all the unallocated partitions as a new disk drive so that AUDIT could find structures that might be lost or corrupted. In *granule 2*, AUDIT can perform data carving in both *Partition* and *P_unallocated* spaces (see Figure 5.10). However, this method is very unlikely that an expert investigator would follow due to the following reasons. First, the investigator will have so many redundant files when carving is used and this will increase the analysis burden on the investigator. Second, the investigator will not be able to have any additional information (metadata) other than the file and its sector offset. This lack of information will make the investigation longer and even harder.

Instead of performing data carving, AUDIT follows another direction being a suspicious toolkit. It tries to recover lost or damaged structures from these unknown spaces. In this granule, unpar-

```
testdisk /cmd part_test.dd list
TestDisk 6.14, Data Recovery Utility, July 2013
Christophe GRENIER <grenier@cgsecurity.org>
http://www.cgsecurity.org
Disk part_test.dd - 419 MB / 400 MiB - CHS 6400 4 32
     Partition               Start        End    Size in sectors
 1 P Linux                  80   0  1  1599   3 32     194560
     ext2 blocksize=1024 Sparse superblock
 2 P FAT32 LBA            1680   0  1  3199   3 32     194560 [NO NAME]
     FAT32, blocksize=1024
```

Figure 5.11: Command line for TestDisk and sample output

```
gpart -C 6400,4,32 part_test.dd
Begin scan...
Possible partition(Linux ext2), size(95mb), offset(5mb)
Possible partition(DOS FAT), size(95mb), offset(105mb)
Possible partition(DOS FAT), size(95mb), offset(210mb)
Guessed primary partition table:
Primary partition(1)
   type: 131(0x83)(Linux ext2 filesystem)
   size: 95mb #s(194560) s(10240-204799)
   chs:  (80/0/1)-(1023/3/32)d (80/0/1)-(1599/3/32)r

Primary partition(2)
   type: 011(0x0B)(DOS or Windows 95 with 32 bit FAT)
   size: 95mb #s(194560) s(215040-409599)
   chs:  (1023/3/32)-(1023/3/32)d (1680/0/1)-(3199/3/32)r

Primary partition(3)
   type: 006(0x06)(Primary 'big' DOS (> 32MB))
   size: 95mb #s(194560) s(430080-624639)
   chs:  (1023/3/32)-(1023/3/32)d (3360/0/1)-(4879/3/32)r
```

Figure 5.12: Command line for gpart and sample output

titioned space is named as *P_unallocated* which means that this space will be searched for possible lost partition(s). In this case AUDIT suspects that the partition table of the disk might be altered or corrupted. Therefore AUDIT runs *TestDisk* and *gpart* in order to recover potential partitions.

In order to recover lost partitions with *gpart*, AUDIT first needs to know the disk geometry which is represented by the number of cylinders, heads, and sectors. AUDIT uses *TestDisk* to determine the disk geometry information. Figure 5.11 shows the command line that is used to invoke *TestDisk* along with its output. The highlighted text in this figure is the geometry of the disk. This geometry information is next used by AUDIT as an input to *gpart* as shown in Figure 5.12. The relevant information in this output is the type and size of each guessed partition. The size of each partition is given in MB (megabytes) along with the exact number of sectors, and

```
fsstat -o 430080 part_test.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: MSWIN4.1
Volume ID: 0x7930bffc
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory):
File System Type Label: FAT16

Sectors before file system: 430080

File System Layout (in sectors)
Total Range: 0 - 194559
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 190
* FAT 1: 191 - 380
* Data Area: 381 - 194559
** Root Directory: 381 - 412
** Cluster Area: 413 - 194556
** Non-clustered: 194557 - 194559
```

Figure 5.13: Partial output of fsstat

```
mmls -a part_test.dd
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors

     Slot    Start        End          Length       Description
02:  00:00   0000010240   0000204799   0000194560   Linux (0x83)
04:  00:01   0000215040   0000409599   0000194560   Win95 FAT32 (0x0C)
```

Figure 5.14: Output of mmls

beginning and ending sectors of the partition (highlighted in Figure 5.12). The size information is used in other tools to get more information about the lost partition. For example, AUDIT runs *fsstat* (see Figure 5.13) with the offset we have learned from *gpart* and gets more information about the lost partition. However, AUDIT did not know the existence of this partition when it only used *mmls* (see Figure 5.14).

If AUDIT would not be suspicious about the unknown part of the disk, it would not find the lost partition, and thus the lost file system. The same strategy is used at all the granules when any unrecognized parts are detected. These parts will be examined further for searching potential structures that give more information about the evidence.

Overall, at *granule 2*, AUDIT finds partitions and examines the unpartitioned areas for possible

legal partition patterns. Once the partitions are found, AUDIT moves to the *granule 3*. The unrecognized parts of the disk will be moved to *granule 5* for file carving process.

In granule 3, the same strategy is followed. The main purpose is to recognize healthy and legitimate file system(s) for further analysis. It is possible that a partition can have multiple file systems unless it is an extended partition. Furthermore, extended partitions can contain up to one file system and one extended partition [10]. If any legitimate file system is found then AUDIT moves to *granule 4* for that specific file system for deeper analysis. If any space is not allocated to a file system then AUDIT names this space as *FS_unallocated*. AUDIT tries to find/recover possible deleted/corrupted file system(s) structures using *TestDisk* from *FS_unallocated* space. If no additional file system is found then this space will be analyzed by a file carver in *granule 5*.

In granule 4, AUDIT examines block structures in each file system that is found in *granule 3*. The allocated blocks found in the *Blocks* space will be moved to *granule 5* for file analysis. See Figure 5.10. As discussed in Chapter 3, file systems may not allocate all the blocks to files and even some files may be deleted. This results in unused space called unallocated space in the file system. We call this space *B_unallocated* in this model and AUDIT analyzes this space with file system tools (*tsk_recover* and *blkls*). If no interesting file is found, AUDIT then moves that space to *granule 6* and uses file carvers (*scalpel* and *photorec*) in order to find possible hidden files. Finally, this space may be analyzed in *granule 7* if nothing is found in previous granules.

AUDIT examines file structures at *granule 5* which is the deepest level that AUDIT can use metadata information for file recovery. It is also known that files may not use all the allocated space to keep their contents which results in files slack. AUDIT recovers the all slack area in each file system via *blkls* then the slack space to *granule 6* for further analysis via file carving methods. In the case of unsuccessful file system analysis in *granule 5*, the space will be moved to *granule 6* for file carving.

Our new hierarchical model allows investigators to search for any data in any part of the disk using AUDIT. If AUDIT does not recognize any space on disk due to lack of available tools for analysis, it will automatically use file carvers on that space. As mentioned above, AUDIT performs file carving methods in *granule 6* via file carver tools *scalpel* and *photorec* for various file types. If this method does not result any useful information, it then allows investigators to use hex editors either on each part of the disk which were divided in the previous levels or on the whole disk.

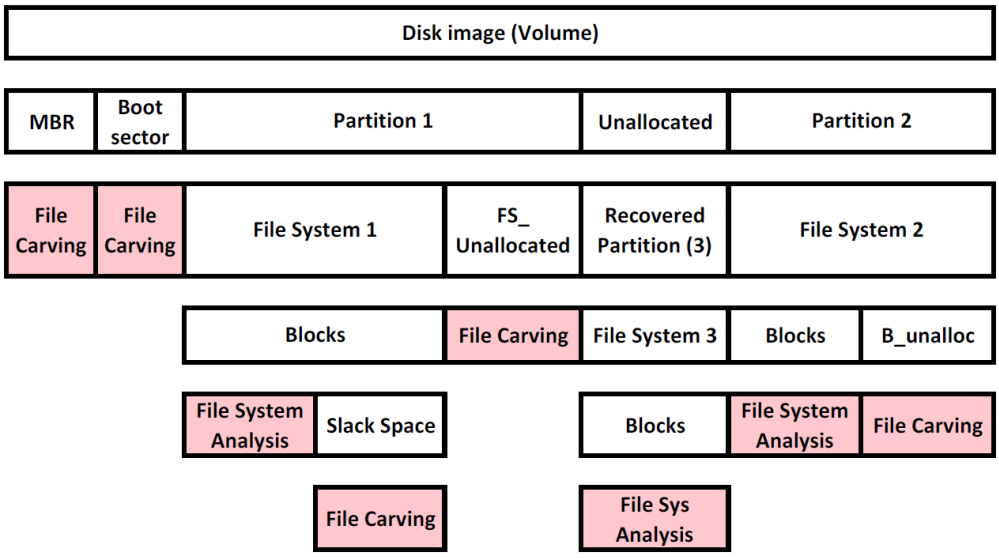


Figure 5.15: Example of a hierarchical disk analysis

Therefor, all the parts are moved to the *granule 7* for bytes level analysis. This also guarantees that every space in a given disk will be analyzed at least at the bytes level.

Regardless of what type of information is searched, files are the most important resources to accomplish the search task. Hence, AUDIT aims to find files in every single space including hidden places for the investigator. All the recovered files are categorized and filtered based on the investigative task specified by the user. This is done by using our new tool *file_sorter* which is discussed in Chapter 3.

Figure 5.15 shows an example of a hierarchical disk analysis using the new model. It also illustrates that any part of the disk is analyzed for files using at least one method. The highlighted boxes show the type of the first examination attempt for files. In this example we only show some of the examination steps of AUDIT therefore it is assumed that some of the parts are not present on disk such as *B_unallocated* space in *File System 1*.

5.3 Reporting in AUDIT

Court accepted digital forensics tools (FTK and EnCase) generate an investigation report that helps explain the findings with technical details. The reporting mechanism in AUDIT currently does not generate a full technical report; instead, it identifies all procedures and tools that are used

```
Single partition disk!
                              .
                              .
Volume Type = Unknown
Sector Size = 512
Disk (Physical) Block  Size = unknown
Path to the input file - >> /home/utk/ecitTest-3.raw
Path to the output directory - >> /home/utk/t1
Document file search will be performed!
*********************************************************
tsk_recover is running on the input disk image to extract
user created/deleted files! Command line below is used:
tsk_recover -a /home/utk/ecitTest-3.raw /home/utk/t1/tsk_recover/allocated -o 0
*********************************************************
tsk_recover is running on the input disk image to extract
user created/deleted files! Command line below is used:
tsk_recover /home/utk/ecitTest-3.raw /home/utk/t1/tsk_recover/unallocated -o 0
*********************************************************
blkls is running on the input disk image to extract
unconventional spaces! Command line below is used:
blkls -s /home/utk/ecitTest-3.raw -o 0 > /home/utk/t1/slackSpace/slack.dd
*********************************************************
photorec is carving files from the slack space for your search!
Command line below is used:
photorec /d /home/utk/t1/slackSpace/files /cmd /home/utk/t1/slackSpace/slack.d
    fileopt,everything,disable,doc,enable,zip,enable,txt,enable,qbb,enable,
    pdf,enable,wholespace,search
*********************************************************
                              .
                              .
Feedback: Interesting data is not found so far.
```

Figure 5.16: Partial examination report of a single partition disk image

during the investigation. However, it also provides the logic of how and why the tools were used which is not done in the other tools.

AUDIT reporting includes detailed information about the input disk, tools invoked and usage, inference information about what caused a tool to run, and layout of the disk in the case of multiple partitions. AUDIT's report is created automatically and user feedback is added to the report after any interaction. Figure 5.16 shows part of the generated examination report after analysis of a single partition disk.

Figure 5.17 shows the report related to one of the rules (slack-space-extraction-rule) that fired during the analysis. The firing facts (facts start with a *fact-id*) explain what actions were previously

```
Fired Rule Name : MAIN::slack-space-extraction-rule

Firing Facts    : [Token: size=7;sortcode=14478322;negcnt=0
f-49 (MAIN::start slack space carving);
f-56 (MAIN::unallocated space analyzed);
f-54 (MAIN::allocated space analyzed);
f-36 (MAIN::disk-image-path is "/home/utk/part5.img");
f-37 (MAIN::output-path is "/home/utk/t1");
f-48 (MAIN::search type is 1);
f-10 (MAIN::tools (ident "blklsSlack") (toolname "blkls") (task "recovering_slack_space")
 (params "-s") (p_in "N/A") (input "/home/utk/part5.img") (p_out ">")
 (output "/home/utk/t2") (p_conf "N/A") (config "N/A") (p1 "N/A") (p2 "N/A") (p3 "N/A")
 (S "0") (F "0") (R "0.0"));]
```

Figure 5.17: Partial inference report for slack space extraction

taken and why. Note that the fact id number is actually the time (and thus the order) that the fact was added to the knowledge base.

The inference report tells the user that AUDIT has learned input, output and task information from the facts *f-36*, *f-37* and *f=48* respectively. The user can see that slack space carving starts only if the expert system knows that both allocated and unallocated spaces were previously analyzed as evidenced by fact-id *f-54* and *f-56*. The rule is fired when all the facts are true including fact-id *f-10*. Note that this complex fact indicates that the tool which is run is *"blkls."* The execution order can clearly be seen in Figure 5.16. Furthermore, *f-48* is added when the user selects "document search" and *f-49* is added because a document search analysis rule when fired adds the fact *f-49* to the knowledge base.

We believe that the examination and inference reports can be quite useful for both expert and non-expert users. Expert users for example can use the inference order to know in which order the tools were invoked and what parts of the disk were analyzed. They could then redo certain analyses (using information in the command lines reported by AUDIT) with other tools or for verification purposes. Through AUDIT's initial disk analysis they would already have gotten substantial information about the disk with a great deal of time saved. As for the non-expert user, it is clear that they could for example get useful information related to learning about the usage of a standard carving tool.

CHAPTER 6

EVALUATION & EXPERIMENTAL RESULTS

It is difficult to do a real user study since we do not have access to a set of law enforcement examiners. Furthermore, even testing the data hiding process is non-trivial with few support tools available. In this section we simply report on some testing that we did do to explore AUDIT's capabilities. Our experiments are divided into two different classes of tests. We first evaluated AUDIT against a human investigator (an intern working at our laboratory). The investigator had moderate technical knowledge and experience relating to forensic investigations, open source tools and hard disk structures. In the second class of tests we run AUDIT on widely used tool testing disk images gathered from NIST [63] and Digital Corpora [36].

6.1 Experimental Setup

For our testing we used a high performance desktop workstation with the following specifications.

- 16GB DDR3 memory
- Intel Xeon(R) CPU E3-1230 V2 @ 3.30GHz with 8 cores
- 2TB SATA HDD

We created disk images using ForGe [83], a forensics disk image generator. The first step requires setting up a "case". Our case was set up to generate 1GB disk images with sector size 512 bytes and cluster size 8 sectors or 4KB. Our case is set up to create an NTFS file system on the disk image as this is currently the only file system type fully supported by ForGe. Note that ForGe does not allow creation of multi-partition disk images.

The next step is to create the "trivial strategy." This represents the directory tree (see Figure 6.1) and the files normally found in the file system. Our directory tree consisted of 31 directories named and structured to mimic a Windows OS folder hierarchy. All directories contain 10 files,

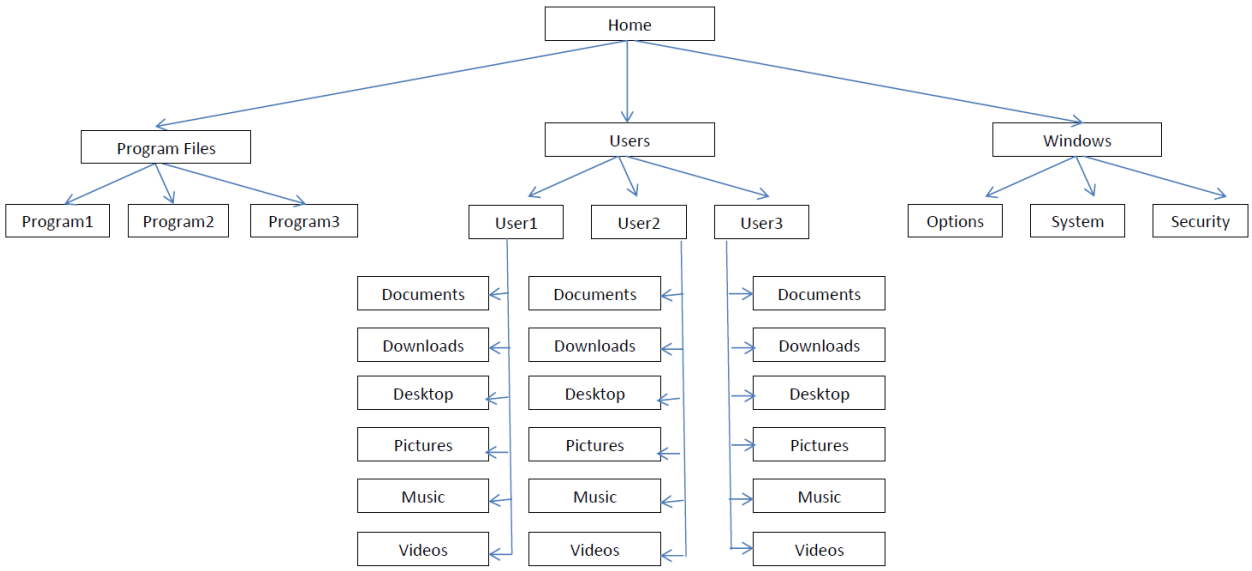


Figure 6.1: Directory tree structure in test disk images

except for the root directory which contains 0. So for each generated disk image we have 300 "trivial" files which are not hidden.

We created an initial dataset of 1000 random unique files (see Table 6.1) obtained from the Govdocs1 digital corpus [36].

Table 6.1: Extension type and quantity of files in dataset

Ext	**Qty**	**Ext**	**Qty**	**Ext**	**Qty**	**Ext**	**Qty**	**Ext**	**Qty**
pdf	257	xls	60	csv	17	log	5	sys	1
html	227	ppt	54	pst	9	png	3	tmp	1
jpg	104	xml	31	unk	7	text	3	dbase3	1
txt	84	gif	27	gz	7	kmz	2	rtf	1
doc	67	ps	22	swf	7	pps	2	kml	1

We also added 50 horse pictures (representing illegal pictures) to the initial dataset. See Table 6.2. The 300 trivial files were chosen randomly from the resulting set of 1050 files.

The final step in creating a disk image using ForGe is to create a "secret strategy" which represents files that are hidden in forensically interesting ways. For the purposes of testing AUDIT we used three different hiding methods: putting a file into file slack space, putting a file into disk unallocated space, and deleting a file from the file system. The files to hide are first chosen randomly

Table 6.2: Size, type and quantity of horse pictures

Size	**Ext**	**Qty**	**Ext**	**Qty**	**Ext**	**Qty**
> 4KB	.jpeg	14	.jpg	16	.png	0
< 4KB	.jpeg	11	.jpg	5	.png	4

from our original set of 1050 files. We divide the original set of 1050 files into three subsets: graphic files of horse pictures smaller than 4KB (type 1), Outlook email archive (pst) files (type 2), and all remaining files (type 3). We need files less than 4KB to hide them in file slack as anything larger will not fit due to our set cluster size. We also wanted to include the possibility of hiding a *pst* file in each disk image to test AUDIT's new email file search capability.

ForGe has limitations in how files can be hidden. It only allows a maximum of one file for each hiding method to be hidden on disk. So we choose a file at random from each of the three subsets and then assign it to a hiding method as follows: A type 1 file is assigned to the *File Slack* hiding method only. Files chosen from the other two subsets are randomly assigned to the *Unallocated* and *Deleted* hiding methods, one in each. Next we randomly determine whether a hiding method will actually be used or not for a test. A minimum of zero and maximum of three hiding methods will thus be present in each test. This hiding process is also illustrated in Figure 6.2.

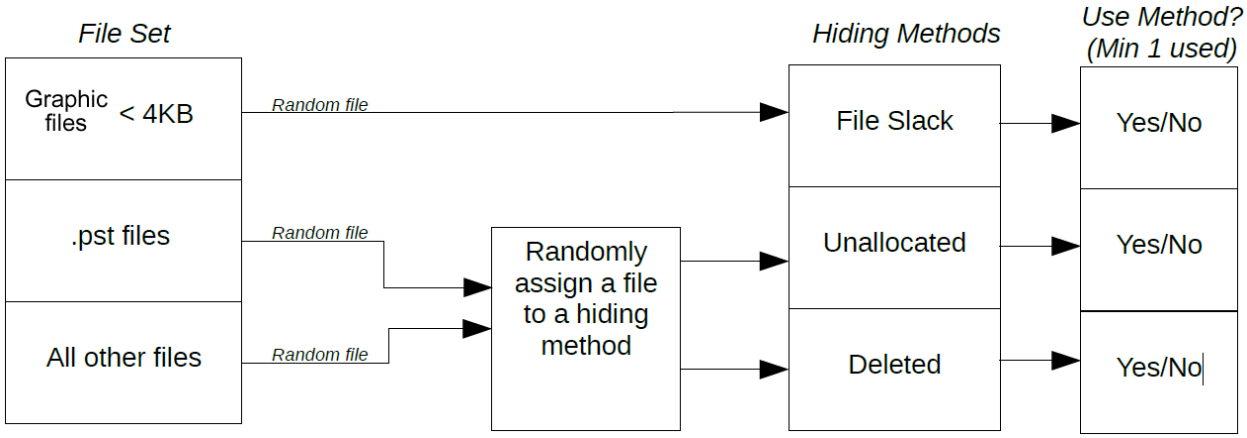


Figure 6.2: File hiding process

Very few hidden files are thus in a disk image. Furthermore, it is unlikely that any files contained credit card numbers (CCN) or social security numbers (SSN). Thus for our tests, we manually hid some document files that included such numbers, email addresses and user names in unallocated

and deleted spaces. Table 6.3 shows the resulting number of forensically interesting files located in our test cases. Note that what was hidden was not known to our test investigator.

Table 6.3: Number of files in each space on the disk

	Allocated	**Deleted**	**Slack**	**Unallocated**
Disk 1	16	2	1	0
Disk 2	15	3	0	1
Disk 3	16	3	1	2
Disk 4	22	4	0	1
Disk 5	26	1	1	0

6.2 Testing Part 1

In this testing part we conducted experiments in two phases to evaluate AUDIT's performance on five test disks. In phase one the investigator was asked to use his own skills and tools without AUDIT. In the second phase we asked the investigator to completely depend on AUDIT's examination. We only present a subset of our results in this dissertation. The investigator analyzed the disks in order and was given a set of instructions for each disk that are explained in the narrative.

For all disks, the investigator was asked to find graphics (horse pictures) and email files on the disks and report on the locations of the files. For Disk 1, the exact number of graphics (17) and email (2) files was also given. Table 6.4 shows the results for this scenario.

Table 6.4: Finding graphics and email files

Disk		**Graphic**	**Email**	**Location**
1	**Expert**	17/17	2/2	**X**
	AUDIT	17/17	2/2	✓
2	**Expert**	**15/17**	2/2	✓
	AUDIT	17/17	2/2	✓
3	**Expert**	13/13	5/5	✓
	AUDIT	13/13	5/5	✓
4	**Expert**	21/21	3/3	✓
	AUDIT	21/21	3/3	✓
5	**Expert**	24/24	2/2	✓
	AUDIT	24/24	2/2	✓

Table 6.4 shows the performance of AUDIT was as good as the investigator's. AUDIT even

outperformed the investigator on the first two disks. For Disk 1, the investigator was unable to report the location of one horse picture (located in slack space) because he found the picture using a file carver which knows only sector level information. When analyzing the rest of the disks, he was able to correctly report the locations because he had learned how slack space is analyzed using AUDIT. On the second disk, the investigator missed two graphics *png* files because he did not extend his search to all graphic file types. AUDIT however found all the files and locations.

For disks 3 and 4, the investigator was also told that there were hidden document files on these disks. The task was to recover them from the hidden places and report their types, quantities and locations. Table 6.5 shows that both AUDIT and the investigator correctly found all the hidden files.

Table 6.5: Hidden document files and their locations

Disk		**Qty**	**Type**	**Location**	**Qty**	**Type**	**Location**
3	**Expert**	2	pdf	unallocated	2	doc & xls	deleted
	AUDIT	2	pdf	unallocated	2	doc & xls	deleted
4	**Expert**	1	xls	unallocated	1	pdf	deleted
	AUDIT	1	xls	unallocated	1	pdf	deleted

For disk 3, the investigator was also asked to find files containing CCN, SSN and email address to test AUDIT's search capabilities. For disk 4 he was only asked to find files containing email addresses. For test 5, he was asked to find SSN and email addresses. As Figure 6.6 shows, both AUDIT and the investigator correctly found all the files containing related sensitive numbers.

Table 6.6: Results of CCN, SSN and email address search

Disk		**CCN**	**SSN**	**Email**
3	**Expert**	✓	✓	✓
	AUDIT	✓	✓	✓
4	**Expert**			✓
	AUDIT			✓
5	**Expert**		✓	✓
	AUDIT		✓	✓

In Table 6.7 we compare the times to analyze each disk. AUDIT outperformed the human investigator in all cases. For disk 1 the investigator was not yet familiar with AUDIT's output, hence times were similar. For later disks AUDIT generally took about half the time. The time

for AUDIT on Disk 5 was surprising to us until we determined that the investigator did not scan AUDIT's output of allocated files until very late. Therefore, larger portion of the time spent for disk 5 was for searching the unconventional spaces.

Table 6.7: Analysis times with and without AUDIT

	Disk 1	**Disk 2**	**Disk 3**	**Disk 4**	**Disk 5**
Expert	9m26s	13m45s	25m10s	18m45s	26m13s
AUDIT	7m55s	5m33s	12m8s	10m16s	20m51s

6.3 Testing Part 2

In this part, we test AUDIT on some of the benchmark disk images. Note that this also tests how AUDIT performs on multi partition disk images. Table 6.8 shows the overview of the test disk images.

Table 6.8: Sample benchmark disk images

Disk Image	**Category**	**Target**	**Source**
dfr-04-fat.dd	deleted file recovery	36 files	NIST CFReDS
dfr-05-ntfs.dd	deleted & fragmented file recovery	7 files	NIST CFReDS
L0_Documents.dd	non-fragmented file carving	7 files	NIST CFReDS
L0_Graphics.dd	non-fragmented file carving	6 files	NIST CFReDS
L1_Documents.dd	fragmented file carving	7 files	NIST CFReDS
L1_Graphics.dd	fragmented file carving	6 files	NIST CFReDS
nps-2010-emails.E01	email address recovery	30 emails	Digital Corpora

Test Disk 1. The purpose of this test is to run AUDIT for recovering several non-fragmented files with non ASCII file names (see Figure 6.3 for sample file names) from a multi partition disk image (*dfr-05-ntfs.dd*). Figure 6.4 shows the layout of the disk image. This disk image contains 36 files located across three partitions containing FAT12, FAT16 and FAT32 file systems respectively. Note that, this test is a good example of examining multi partition disk image using AUDIT with its hierarchical examination model.

AUDIT recovered all 36 non-fragmented files with non ASCII file names from three different file systems when we compared our results with the disk creation report given in [63].

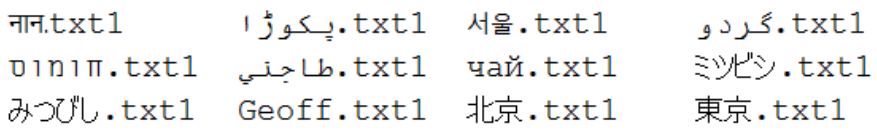


Figure 6.3: Sample of non ASCII file names

	Slot	Start	End	Length	Description
00:	Meta	0000000000	0000000000	0000000001	Primary Table (#0)
01:	-----	0000000000	0000000127	0000000128	Unallocated
02:	00:00	0000000128	0000016511	0000016384	DOS FAT12 (0x01)
03:	00:01	0000016512	0000082047	0000065536	DOS FAT16 (0x06)
04:	00:02	0000082048	0000213119	0000131072	Win95 FAT32 (0x0b)
05:	-----	0000213120	0002097152	0001884033	Unallocated

Figure 6.4: Layout of test disk 1

Test Disk 2. For this test disk, we ran AUDIT for recovering several fragmented and deleted files from another multi partition disk image (*dfr-05-ntfs.dd*). Figure 6.5 shows the layout of the disk image. In this disk there was only one allocated partition contained an NTFS file system. There were 2 fragmented and deleted files and 5 non-fragmented files in the file system. When we ran AUDIT, it successfully found all 7 files in the allocated partition. Although fragmented file recovery is typically a hard task it was not the case here because AUDIT gathered all the fragment locations from the healthy file system.

	Slot	Start	End	Length	Description
00:	Meta	0000000000	0000000000	0000000001	Primary Table (#0)
01:	-----	0000000000	0000000127	0000000128	Unallocated
02:	00:00	0000000128	0000614527	0000614400	NTFS (0x07)
03:	-----	0000614528	0002097152	0001482625	Unallocated

Figure 6.5: Layout of test disk 2

Test Disk 3. In this test, we ran AUDIT for carving non-fragmented documents files from *L0_Documents.dd* disk image. In this disk image, file system was corrupted therefore no metadata information was available. When file system corrupted, AUDIT runs file carving tools to find the target files. In this test disk image, there were 7 files in different types (2 pdf, 2 xlsx, 2 docx, and 1 pptx).For this disk image, AUDIT successfully carved all the files at the correct locations as in the disk creation report [63]. Since the files were non-fragmented, the contents of the files were not

corrupted after the carving process.

Test Disk 4. Here we tested AUDIT for sequentially fragmented documents files carving from *L1_Documents.dd* disk image. Sequentially fragmented file means that the fragments of a file are located in consecutive blocks on disk. Table 6.9 shows the test disk layout and file fragmentation information.

Table 6.9: Type and fragmentation information for documents files

Fragment	File size	Start Sector	End Sector
D1.pdf (1)	1,587,712	10,000	13,100
D2.pdf (2)	1,588,563	13,101	16,203
D2.pdf (1)	841,728	26,204	27,847
D2.pdf (2)	841,728	27,848	29,491
D2.pdf (3)	841,958	29,492	31,136
D3.xlsx (1)	7,680	41,137	41,151
D3.xlsx (2,3)	15,833	41,152	41,182
D4.xlsx (1,2)	9,216	51,183	51,200
D4.xlsx (3)	4,608	51,201	51,209
D5.docx (1)	1,024	61,210	61,211
D5.docx (2)	1,024	61,212	61,213
D5.docx (3)	2,376	61,214	61,218
D6.docx (1)	1,536	71,219	71,221
D6.docx (2)	2,468	71,222	71,226
D7.pptx (1)	300,544	81,227	81,813
D7.pptx (2,3)	602,101	81,814	82,989

For this test disk, AUDIT was also able carve all 7 sequentially fragmented documents files. AUDIT did not know the location of the fragments however files were carved successfully because no other data fragment was found between the file fragments. Thus, all the contents of the carved files were complete when compared with the disk image report [63].

Test Disks 5 & 6. Here we ran AUDIT in order to carve non-fragmented and sequentially fragmented graphics files from disk images *L0_Graphics.dd* and *L1_Graphics.dd* respectively. Similar to the disks 3 and 4, file systems in these disk images were also possibly corrupted or lost. In both test disks there were 6 files in total in different types representing majority of graphics files that are typically created by users. Table 6.10 shows the layout of *L1_Graphics.dd* disk image and fragmentation information of the located files.

Table 6.10: Type and fragmentation information for graphics files

Fragment	File size	Start Sector	End Sector
09260002.jpg (1)	30,208	10,000	10,058
09260002.jpg (2)	30,208	10,059	10,118
100_0018.tif (1)	4,989,440	20,119	29,863
100_0018.tif (2)	4,989,440	29,864	39,608
100_0018.tif (3)	4,990,448	39,609	49,355
100_0304crop.bmp (1)	2,555,904	59,356	64,347
100_0304crop.bmp (2)	2,556,002	64,348	69,340
02010025.pcx (1)	268,800	79,341	79,865
02010025.pcx (2)	268,800	79,866	80,390
02010025.pcx (3)	269,064	80,391	80,916
100_0183.gif (1)	66,048	90,917	91,045
100_0183.gif (2)	66,900	91,046	91,176
000_021.png (1)	4,321,792	101,177	109,617
000_021.png (2)	8,644,847	109,618	126,502

For both test disks, AUDIT successfully carved all the graphics files at the correct locations as stated in the disk creation report [63].

Test Disk 7. In our last test case, we tested AUDIT for email address search from disk image *nps-2010-emails.E01*. This disk image contains 30 email addresses located in many different file types including documents and compressed files. Figure 6.6 the layout of the disk image.

```
     Slot    Start        End          Length       Description
00:  Meta    0000000000   0000000000   0000000001   Primary Table (#0)
01:  -----   0000000000   0000000000   0000000001   Unallocated
02:  00:00   0000000001   0000020479   0000020479   Win95 FAT32 (0x0b)
```

Figure 6.6: Layout of nps-2010-emails.E01 disk image

In order to find email addresses, AUDIT invokes the bulk_extractor tool. It retrieved all the email addresses in the narrative file for *nps-2010-emails* except one email address (plain_utf16@textedit.com) with non-ASCII content. However AUDIT also automatically recovers document files and a visual check showed this address was in a txt file. This shows some of the power of AUDIT's tool integration.

CHAPTER 7

CONCLUSION

7.1 Summary of the Problem

Digital forensics investigation is a critical step towards solving many of the crimes committed in this age of technology, and it will continue to be even more important in the future. Digital forensics investigations are performed by investigators who might have technical skills to various degrees by using specialized or general purpose software tools. In its very nature, digital forensics investigation in general and particularly the examination of a hard disk is complex for the investigator. Investigators are usually expected to deal with certain problems such as an intensive learning process, complexity, and lack of standardization regardless of their expertise in the area.

Learning arises because of the required technical knowledge and skills about the digital forensics tools and the target devices. Learning is a challenge for even expert investigators because they have to know every details about the target disk in order to justify their findings. For the purpose of dealing with this problem and building the required skills, investigators usually take long and intensive training sessions.

Complexity becomes a challenge because of the large collection of digital forensics tools which are available to the investigators for their use in order to analyze target devices and to collect evidence from them. The open source tools in the market are required to be configured or parameterized in certain way for specific search tasks. They even must be integrated in order to collect desired evidence since one may use the output of another tool.

Investigators mostly start their investigations with some of the questions in mind regarding where to start searching potential evidence. Based on their results, they drive the investigation in different directions. Each investigator may follow different approaches based on their past experiences and the guidelines provided by their institutions. This is mainly because there is no universal standard with regards to systematic disk investigation.

From the above problems, it is obvious that digital forensics investigators need usable tools that will help them get results easily (automatically), faster, and with less usage complexity independent

of their computer and IT expertise.

7.2 Solution: Automated Disk Investigation Toolkit

In this dissertation we described AUDIT, a novel automated disk investigation toolkit. This "intelligent assistant" helps expert and non-IT-expert investigators during their examination of a hard disk. We assume that the investigators understand forensic investigation but may not have technical skills or detailed knowledge about the current open source tools and may not have knowledge about the disk or file structures. AUDIT contains an expert system and domain specific knowledge base that is used to automatically configure, parameterize and integrate some of the commonly used open source command line digital forensics tools. AUDIT supports the investigator in conducting both general and specific investigative tasks. AUDIT also uses a hierarchical disk investigation model which performs systematic examination of the disk in its total based on its physical and logical structures.

We believe that using expert systems technology is a good way to develop tools that can support forensic examinations. Our goal was to develop a complete system for a non-trivial domain (forensic examination of a hard disk) that is both technically challenging and would be of utility to real investigators. The knowledge base and the expert system help us to represent, use, add, and modify domain specific technical knowledge regarding the investigations. This simplifies both the development and maintenance effort when adding or modifying the tools. In our experience, adding a new tool requires creating a few new rules and facts into the knowledge base and adding some entries into the database. Given technical knowledge about a tool, we can do this in hours.

7.3 Contributions

This dissertation adds several contributions to the literature. In this section we present these contributions.

Our first contribution is the novel design of Automated Disk Investigation Toolkit (see Figure 5.1). This design contains different components in order to automate disk analysis for the use of both expert and non-IT-expert investigators.

Our second contribution is the design of an expert system in AUDIT with a domain specific knowledge base by using Java Expert System Shell.

Our third contribution is the implementation of the novel design as the tool AUDIT which is the first tool that uses an expert system in order to automate the technical aspects of the investigation at an expert investigator level. Our implementation automates the following tasks that a typical investigator almost always has to do during the investigation when using open source tools for general and specific search tasks.

- Determine the level of knowledge of the investigator and use this knowledge to determine further choices.
- Using the domain specific knowledge we have embedded in the knowledge base and database, configure and parameterize the appropriate digital forensics tools for execution.
- Execute the appropriate tools at the correct times and also run tools based on the dynamic knowledge obtained from running earlier tools. This thus supports integrative use of the digital forensics tools.

AUDIT also creates two reports which are examination and inference reports for the investigator. The inference report contains the logic of how and why certain tools are automatically used. This is also a contribution since none of the existing tools provide this information to the investigators.

The final contribution that is presented in this dissertation is the design of hierarchical disk investigation model which is used by AUDIT to analyze the disk in its totality. By the help of this approach, AUDIT guarantees that all the possible spaces (hidden and unhidden) are systematically analyzed with an available open source tool. We also tested AUDIT for effectiveness.

7.4 Future Directions

AUDIT is a unique extensible tool which is designed to configure and integrate open source tools for disk investigation using expert system capabilities. Our experience with AUDIT is that it is quite useful as a forensic assistant. We believe that as we extend its scope and capabilities it could become an indispensable tool for digital forensics investigations. We expect to make it available soon in the public domain.

We believe our design and approach could also be used for other types of examinations such as network and mobile forensics. Currently AUDIT does not have integrated data analysis or data mining tools in order to reduce the findings. We currently only provide file sorting and filtering

based on hash values of the known good files. We then simply ask users to do visual and manual analysis of the collected evidence from the disk. However, we plan to integrate data analysis tools into our toolkit in the future. In addition, the reporting part of AUDIT is yet not a complete report which can be presented to a court. We also plan to develop a more complete reporting mechanism in the future to help investigators in adding findings and notes as part of the report during their investigations.

We also believe that our system could be able to learn how well the tools do while testing the tools, and run them preferentially when they work better than other tools for specific purposes. We call this feature "ranking" digital forensics tools based on their historical successes and failures on certain tasks. In order to rank the tools we plan in future work to study the techniques used in multi-armed bandit problems and adapt them as possible solutions to our ranking problem.

APPENDIX A

COPYRIGHT PERMISSION

Copyright permission for Figure 3.3.

All Microsoft Content is the copyrighted work of Microsoft or its suppliers, and is governed by the terms of the license agreement that accompanies or is included with the Microsoft Content. If the Microsoft Content does not include a license agreement, then you may make a reasonable number of copies of the Microsoft Content for your internal use in designing, developing, and testing your software, products and services that is made available to you on the Documentation Portals without a license agreement. You must preserve the copyright notice in all copies of the Microsoft Content and ensure that both the copyright notice and this permission notice appear in those copies. Accredited educational institutions, such as K-12 schools, universities, and private or public colleges may download and reproduce Microsoft Content for distribution in the classroom for educational purposes.

BIBLIOGRAPHY

[1] Sqlite. [online]. https://www.sqlite.org.

[2] About.com. Linux / unix command: mount. [online]. http://linux.about.com/library/cmd/blcmdl2_umount.htm.

[3] About.com. Linux / unix command: mount. [online]. http://linux.about.com/od/commands/l/blcmdl8_mount.htm.

[4] About.com. Linux / unix command: strings. [online]. http://linux.about.com/library/cmd/blcmdl1_strings.htm.

[5] AccessData. Ftk. [online]. http://www.accessdata.com/products/digital-forensics/ftk.

[6] AccessData. [online]. http://accessdata.com/.

[7] Akadia.com. How to setup windowslinux dual boot. [online]. http://www.akadia.com/services/dual_boot.html.

[8] ArxSys. Digital forensics framework. [online]. http://www.digital-forensic.org.

[9] Nicole Beebe. Digital forensic research: The good, the bad and the unaddressed. In Gilbert Peterson and Sujeet Shenoi, editors, *Advances in Digital Forensics V*, volume 306 of *IFIP Advances in Information and Communication Technology*, pages 17–36. Springer Boston, 2009. 10.1007/978-3-642-04155-6_2.

[10] Hal Berghel, David Hoelzer, and Michael Sthultz. Chapter 1 data hiding tactics for windows and unix file systems. In Marvin V. Zelkowitz, editor, *Software Development*, volume 74 of *Advances in Computers*, pages 1 – 17. Elsevier, 2008.

[11] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[12] Jessica R. Bradley and Simson L. Garfinkel. Bulk_extractor user manual. Technical report.

[13] Brian Carrier. Digital forensics tool testing images. [online] http://dftt.sourceforge.net.

[14] Brian Carrier. *File System Forensic Analysis*. AddisonWesley Professional, 2005.

[15] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/blkls.html.

[16] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/fsstat.html.

[17] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/img_cat.html.

[18] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/mmls.html.

[19] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/mmstat.html.

[20] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/tsk_loaddb.html.

[21] Brian D. Carrier. [online]. http://www.sleuthkit.org/sleuthkit/man/tsk_recover.html.

[22] Brian D. Carrier. The sleuthkit. [online]. http://www.sleuthkit.org/autopsy.

[23] Brian D. Carrier. The sleuthkit. [online]. http://www.sleuthkit.org/sleuthkit.

[24] Andrew Case, Andrew Cristina, Lodovico Marziale, Golden G. Richard, and Vassil Roussev. Face: Automated digital evidence discovery and correlation. *Digital Investigation*, 5, Supplement(0):S65 – S75, 2008. The Proceedings of the Eighth Annual {DFRWS} Conference.

[25] E. Casey. *Digital Evidence and Computer Crime.* Elsevier Science, 2004.

[26] Mark R. Clint, Mark Reith, Clint Carr, and Gregg Gunsch. An Examination of Digital Forensic Models, 2002.

[27] Insa Crop. Digital investigations and cybercrime analysis. [online]. http://insacorp.com/en/digital-forensics.

[28] Collingwood Collegiate Institute Technology Department. Hard disk structure. [online]. http://www.cci-compeng.com/unit_5_pc_architecture/5707_hdd_structure.htm.

[29] DFRWS. Dfrws 2006 forensics challenge overview. [online]. http://www.dfrws.org.

[30] Robert S. Engelmore and Edward Feigenbaum. Knowledge-based systems in japan. Technical report, 1993.

[31] Robert S. Engelmore and Edward Feigenbaum. *Knowledge-Based Systems in Japan.* WTEC Hyper-Librarian, 1993.

[32] JP Fizaine and NL Clarke. A crime depended automated search and engine for digital forensics. *Advances in Communications, Computing, Networks and Security Volume 10*, page 73, 2013.

[33] ForensicsWiki. File carving. [online]. http://forensicswiki.org/wiki/file_carving.

[34] ForensicsWiki. Tools. [online]. http://forensicswiki.org/wiki/tools.

[35] Ernest Friedman-Hill. Jess, the rule engine for the java platform. [online].

http://www.jessrules.com/.

[36] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. Bringing science to digital forensics with standardized forensic corpora. *Digit. Investig.*, 6:S2–S11, September 2009.

[37] Simson L. Garfinkel. bulk_extractor. [online]. http://digitalcorpora.org/downloads/bulk_extractor/.

[38] Simson L. Garfinkel. Automating disk forensic processing with sleuthkit, xml and python. In *Proceedings of the 2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*, SADFE '09, pages 73–84, Washington, DC, USA, 2009. IEEE Computer Society.

[39] Simson L. Garfinkel. Digital forensics research: The next 10 years. *Digital Investigation*, 7, Supplement(0):S64 – S73, 2010. The Proceedings of the Tenth Annual DFRWS Conference.

[40] Francesco Garosi. Pyclips python module. [online]. http://pyclips.sourceforge.net/web.

[41] Github. Revit. [online]. https://github.com/libyal/reviveit.

[42] Christophe Grenier. gpart. [online]. man.freebsd.org/gpart(8).

[43] Christophe Grenier. Photorec. [online]. http://www.cgsecurity.org/wiki/photorec.

[44] The PostgreSQL Global Development Group. Postgresql. [online]. http://www.postgresql.org.

[45] GuidenceSoftware. [online]. https://www.guidancesoftware.com/.

[46] H. Hibshi, T. Vidas, and L. Cranor. Usability of forensics tools: A user study. In *IT Security Incident Management and IT Forensics (IMF), 2011 Sixth International Conference on*, pages 81 –91, may 2011.

[47] Ernest Friedman Hill. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA, 2003.

[48] Bruno W. P. Hoelz, Célia Ghedini Ralha, and Rajiv Geeverghese. Artificial intelligence applied to computer forensics. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 883–888, New York, NY, USA, 2009. ACM.

[49] G. G. Richard III and L. Marziale. Scalpel. [online]. https://github.com/sleuthkit/scalpel.

[50] Ponemon Institute. [online]. http://www.ponemon.org/.

[51] Virginia Polytechnic Institute and State University. Find_ssns. [online] https://security.vt.edu/resources_and_information/find_ssns.html.

[52] Joshua James and Pavel Gladyshev. Challenges with automation in digital forensic investigations. *CoRR*, abs/1303.4498, 2013.

[53] Gary C. Kessler. File signatures table. [online]. http://www.garykessler.net/library/file_sigs.html, October 2012.

[54] Joseph Migga Kizza. Computer crime investigations - computer forensics. In *Ethical and Social Issues in the Information Age*, Texts in Computer Science, pages 263–276. Springer London, 2010. 10.1007/978-1-84996-038-0_13.

[55] Niandong Liao, Shengfeng Tian, and Tinghua Wang. Network forensics based on fuzzy logic and expert system. *Comput. Commun.*, 32(17):1881–1892, November 2009.

[56] Marziale Lodovico. *Advanced Techniques for Improving the Efficacy of Digital Forensics Investigations*. PhD thesis, 2009.

[57] Andrew Martin. Mobile device forensics. Technical report, 2008.

[58] Jimmy May and Denny Lee. Disk partition alignment best practices for sql server. Technical report, 2009.

[59] John McCarthy. *What is Artificial Intelligence?*, 2007.

[60] Antonio Merola. Data carving concepts. Technical report, 2008.

[61] Joachim Metz. libewf. [online]. https://github.com/libyal/libewf/.

[62] Matthew Meyers and Marc Rogers. Computer Forensics: The Need for Standardization and Certification. *International Journal of Digital Evidence*, 3, 2004.

[63] NIST. The cfreds project. [online]. http://www.cfreds.nist.gov/.

[64] U.S. Department of Homeland Security. *Computer Forensics*. U.S. Department of Homeland Security, 2008.

[65] National Institute of Justice. *Electronic Crime Scene Investigation: A Guide for First Responders*. National Institute of Justice, 2001.

[66] Martin S. Olivier. On metadata context in database forensics. *Digital Investigation*, 5(34):115 – 123, 2009.

[67] Oracle. Mysql. [online]. https://www.mysql.com.

[68] FreeBSD Man Pages. Testdisk. [online]. http://www.cgsecurity.org/wiki/testdisk.

[69] Anandabrata Pal, Husrev T. Sencar, and Nasir Memon. Detecting file fragmentation point using sequential hypothesis testing. *Digit. Investig.*, 5:S2–S13, September 2008.

[70] Gary Palmer. A road map for digital forensic research. Technical report, 2001.

[71] pcmag.com. Definition of:files vs. folders. [online]. http://www.pcmag.com/encyclopedia/term/60967/files-vs-folders.

[72] Rainer Poisel and Simon Tjoa. Roadmap to approaches for carving of fragmented multimedia files. In *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security*, ARES '11, pages 752–757, Washington, DC, USA, 2011. IEEE Computer Society.

[73] Galina Rybina and Victor Rybin. Static and dynamic integrated expert systems: State of the art, problems and trends. 2005.

[74] MIT Computer Science and Artificial Intelligent Laboratory. [online]. http://www.csail.mit.edu/events/eventcalendar/calendar.php?show=event&id=2334.

[75] Julian Smart. wxclips. [online]. http://www.anthemion.co.uk/wxclips/.

[76] Guidence Software. Encase forensics. [online]. http://www.guidancesoftware.com/encase-forensic.htm.

[77] Sourceforge. Clips: A tool for building expert systems. [online]. http://clipsrules.sourceforge.net/.

[78] Tye Stallard and Karl Levitt. Automated analysis for digital forensic science: Semantic integrity checking. In *Proceedings of the 19th Annual Computer Security Applications Conference*, ACSAC '03, pages 160–, Washington, DC, USA, 2003. IEEE Computer Society.

[79] V. L. Stefanuk. Dynamic expert systems. *Kybernetes*, 29(5/6):702–709, 2000.

[80] Computer Technologies. First responders guide to computer forensics. [online]. http://www.mhsv.org/, 2008.

[81] Basis Technology. Digital forensics. [online]. http://www.basistech.com/digital-forensics/.

[82] Oscar Vermaas, Joep Simons, and Rob Meijer. Open computer forensic architecture a way to process terabytes of forensic disk images. In Ewa Huebner and Stefano Zanero, editors, *Open Source Software for Digital Forensics*, pages 45–67. Springer US, 2010.

[83] Hannu Visti. Forge forensic test image generator. [online]. https://github.com/hannuvisti/forge.

[84] Wikipedia. dd (unix). [online]. http://en.wikipedia.org/wiki/dd_%28unix%29.

[85] Wikipedia. Digital forensics. [online] http://en.wikipedia.org/wiki/digital_forensics.

[86] Wikipedia. Disk sector. [online]. http://en.wikipedia.org/wiki/disk_sector.

[87] Wikipedia. File system fragmentation. [online] http://en.wikipedia.org/wiki/file_system_fragmentation.

[88] Wikipedia. Hard disk drive. [online]. http://en.wikipedia.org/wiki/hard_disk_drive.

[89] Wikipedia. Information science. [online]. http://en.wikipedia.org/wiki/information_science.

[90] Wikipedia. Machine learning. [online]. http://en.wikipedia.org/wiki/machine_learning.

[91] Wikipedia. Magic number (programming). [online]. http://en.wikipedia.org/wiki/magic_number_(programming).

[92] Wikipedia. Mobile device forensics. [online]. http://en.wikipedia.org/wiki/mobile_device_forensics.

[93] Wikipedia. Network forensics. [online] http://en.wikipedia.org/wiki/network_forensics.

[94] Craig Wright. Sql, databases and forensics. [online]. http://digital-forensics.sans.org/blog/2009/03/11/sql-databases-and-forensics, 2009.

BIOGRAPHICAL SKETCH

Umit Karabiyik was born in Istanbul, Turkey. He studied Computer Systems Teaching at Sakarya Univesity in Turkey. After his graduation, he worked as a computer science teacher for one and a half year before he moved to the USA. He earned full scholarship from Republic of Turkey Ministry of National Education for his graduate studies. He received her M.S. degree in Computer Science at FSU. His research interests are Digital Forensics, Social Network Forensics, Cyber Security, Password Cracking, Artificial Intelligence, Machine Learning, Expert Systems, Computer and Network Security, Cryptography, Email Accountability. He has worked under the supervision of Dr. Sudhir Aggarwal during his PhD studies.