AUDIT: AUTOMATED DISK INVESTIGATION TOOLKIT

Umit Karabiyik, Sudhir Aggarwal
Department of Computer Science, Florida State University
Tallahassee, Florida, USA
{karabiyi, sudhir}@cs.fsu.edu

ABSTRACT

Software tools designed for disk analysis play a critical role today in 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 paper, 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 proof of concept 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.

Keywords: digital forensics, expert systems, disk forensics, forensic tools, CLIPS

1. INTRODUCTION

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.

Complexity 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 (Case et al., 2008). Investigators are however always expected to know how to use and configure and/or 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 to do I properly use these tools?" and "where/when can I effectively use them?" In practice, forensic examiners might have any level of IT background and technical expertise ranging from a computer security expert to a criminal investigator having minimal computer skills. Thus investigators need usable tools that will help them get results easily (Hibshi et al., 2011) and with less usage complexity independent of their computer and IT expertise.

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, etc. in order to perform correct disk forensics investigation. As Garfinkel (Garfinkel, 2009) 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 currently, most digital forensics tool users typically take training sessions both on tool usage and also on digital targets (Beebe, 2009). According to the user study in (Hibshi et al., 2011), even 68% of their expert responders indicate that they take intensive training sessions to learn the current tools while 31% do not take such training sessions. 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 what the further steps they need to take for conducting a deeper investigation.

In this work, 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 proof of concept 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 images, for document search and email search, and also for more specialized searches such as for credit card and social security numbers.

Expert Systems (ES) are a class of computer programs that arose in work in artificial intelligence. In general, one goal of AI technology is to build computer programs that demonstrate intelligent behavior (Engelmore et al., 1993). Expert systems emulate human expertise in well-defined problem domains by using a domain implemented knowledge base (Riley, 2013). 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 (Engelmore et al., 1993).

Automating the digital forensics process of course has its own challenges. James et al. (2013) and Meyers et al. (2004) caution 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.

In the rest of this paper we will use the term pictures to refer to images on disk, in order to avoid confusion with the term disk image which is the target of the investigation. We assume that users of *AUDIT* do not necessarily have expertise on technical aspects of an investigation, but do have expertise about the investigation process itself. *AUDIT* is not a replacement for a forensic expert in an investigation. It is an intelligent assistant for investigators who lack technical knowledge about either the tools or hard disk structures.

In this paper, *AUDIT* is designed currently with a static database that includes knowledge related to digital forensics tools and investigative tasks. This knowledge is derived from an expert who is knowledgeable in the tools and thus tools configuration is not currently learned by our system. In future work, such configurations could also possibly be learned. As far as we are aware, *AUDIT* is unique in its capabilities to use an AI based environment in order to properly configure and integrate open source forensic tools and guide the forensic examination of a hard disk.

Section II discusses related work and approaches that automate disk forensics processing and that apply AI techniques to the domain of digital forensics. Section III describes *AUDIT*, the toolkit that we have developed for supporting the examination of hard disks using open source tools. In section IV, we illustrate the use of *AUDIT* through two example investigations. In section V we conclude and discuss some future research directions.

2. RELATED WORK

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

The work of Stallard et al. (2003) 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 (Liao et al., 2009). 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) (Vermaas et al., 2010) 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 and *OCFA* is not designed to search and recover files from the given device. Instead, it focusses 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 and 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 evidences without compromising systems and data (ArxSys, 2014). This work is a good example of tool integration and collaboration in order to reduce the burden on 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 Garfinkel (2009).The proposed program, *fiwalk*, is used to automate the processing of forensic data for the purpose of assisting users who wanted to develop programs that can automatically process disk images (Garfinkel, 2009). *fiwalk* also integrates command line tools of Carrier's *SleuthKit* (TSK) (Carrier, 2014a). 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. (2009) developed a program called MultiAgent Digital Investigation toolKit (MADIK), a multiagent system to

assist the computer forensics expert on its examinations. They applied AI approach to the problem of digital forensics by developing multiagent system where each agent specializes on a different task such as hashing, keyword search, windows registry agent and so on. This work is related to our work as being an AI application of 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.

To our knowledge none of this work is directed to assisting examiners during the analysis phase 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 deal 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 (Beebe, 2009). Tools for reducing the data analysis could certainly be integrated into *AUDIT*. In our current implementation, we simply ask users to do a visual and manual analysis of the gathered evidence from the disk (to get feedback). Users would be free to use any such available data analysis or data mining tools and we do plan to integrate such tools into our toolkit in the future.

3. AUDIT: AUTOMATED DISK INVESTIGATION TOOLKIT

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 graphics 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 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.

The database component contains two tables (Tools and Knowledge) that maintain information regarding the tools that will be used by *AUDIT* and the investigative tasks that an average investigator generally performs.

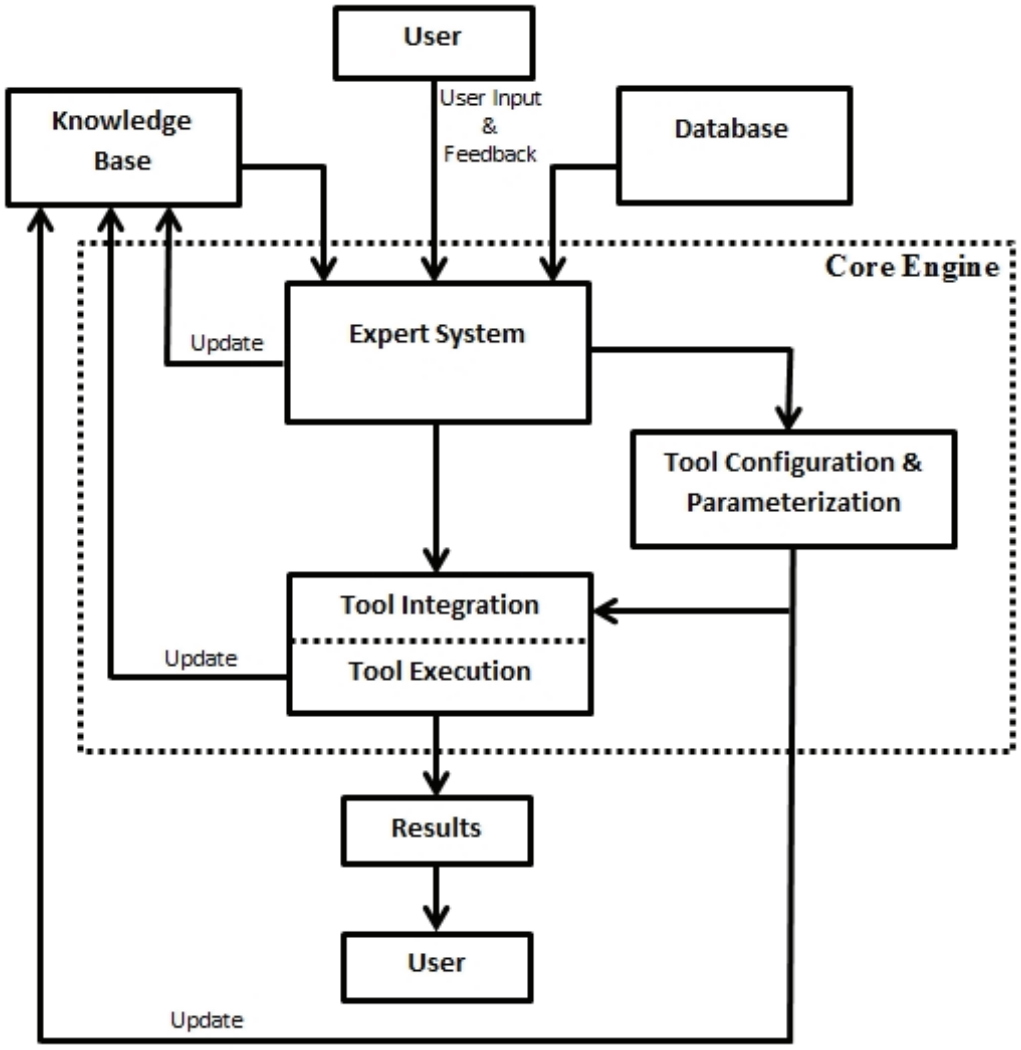

Figure 1 High-level Design of AUDIT

In the Tools table, the field IDENT is a unique identifier for the row entry and is used in the expert system. Each entry specifies the specific tool used (TOOLNAME), the TASK, and the corresponding configuration and/or parameterization. The entry also specifies other aspects such as the INPUT requirements and the OUTPUT of the tool with that configuration/parameterization. See Figure 2. For example, in one entry, we have defined that the forensic tool blkls (Carrier, 2014a) 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.

The Knowledge table currently simply contains a set of investigative tasks (graphics search, document search, credit card number search, SSN search, and email search). Each investigative task is linked to the knowledge base as well as the Tools table through the expert system.

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. 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* (Scalpel, 2014) uses different configuration files for different categories of files that are to be carved (i.e. graphics 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.

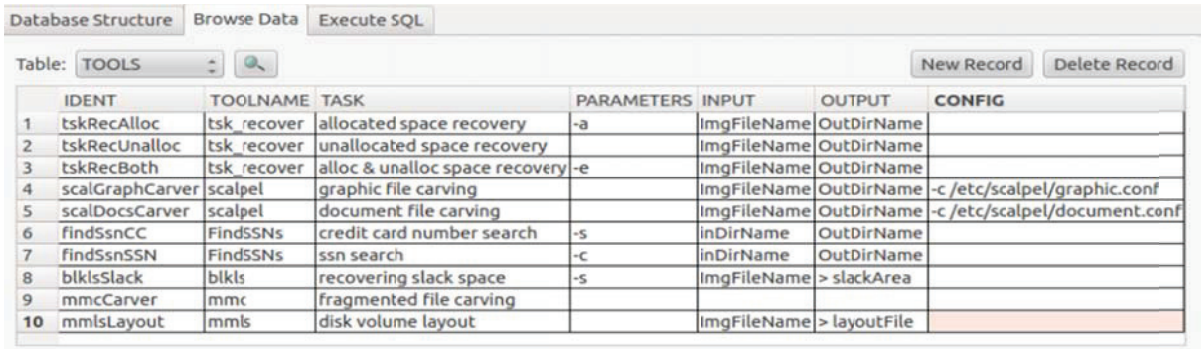
Database Structure | Browse Data | Execute SQL

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	alloc & unalloc 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	

Figure 2 The Tools Table in the AUDIT Database

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* (Carrier, 2014a), *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.

3.1 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 (Riley, 2014). 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 (Riley, 2014).

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 (Riley, 2014). In the following program segment, 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 and even require multiple pages to define.

Simple Example of a CLIPS Rule in AUDIT

```
1.  (defrule determine-investigator-level ""
2.     (declare (salience 10))
3.     (not (investigator is ?))
4.     =>
5.     (if (yes-or-no-p "Are you an expert (yes/no)? ")
6.     then (assert (investigator is expert))
7.        (if (yes-or-no-p "Do you need help (yes/no)? ")
8.        then
9.           (assert (expert needs help))
10.          (assert (determine disk-layout needed))
11.          (assert (extracting partitions from disk needed))
12.       else
13.          (assert (expert needs no-help))
14.          (assert (self usage mode on))
15.          (assert (provide available tool list to user)))
16.    else (assert (investigator is non-expert))
17.       (assert (non_expert needs help))
18.       (assert (determine disk-layout needed))
19.       (assert (extracting partitions from disk needed))))
```

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 following line is added to 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.

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 following line is added to 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.

(declare (salience 10))

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 include 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.

3.1.1 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) to the initial knowledge base:

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

Of course this new knowledge may trigger other rules in the rules list to be activated. The "(configuration needed)" triggers the following:

(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)

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.

3.1.2 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 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) (Poisel, et al. 2011) tools to carve out files which could be both fragmented and unfragmented. The following program code 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.


```
1.  (defrule credit-card-search ""
2.     (image-file-path is ?imagePath)
3.     (output-path is ?outputPath)
4.     (investigative-task is ?task)
5.     (investigation-type is ccsearch)
6.     =>
7.     (printout t "Find_SSNs is running on the disk!" crlf)
8.     (mount-disk-image ?imagePath ?outputPath ?task)
9.     (run-blkls ?imagePath ?outputPath ?task)
10.    (run-strings ?imagePath ?outputPath ?task)
11.    (run-tsk_recover ?imagePath ?outputPath ?task)
12.    (run-Find_SSNs ?imagePath ?outputPath ?task)
13.    (assert (ccsearch performed)))
```

3.2 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* (Carrier, 2014a), *scalpel*, and *Find_SSNs* (Find_SSNs, 2014). 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 (Carrier, 2014a). 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 (Carrier, 2014a). 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 by Golden G. Richard III (Scalpel, 2014) 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 graphics 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 following facts are assumed to have been added to the knowledge base:

(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)

The first fact is feedback from the user whether any evidence or interesting file is found or not. 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 following actions will be taken:

```
1.  (deffunction run_scalpel (?imagePath ?outputPath ?configuration)
2.    (if (eq ?configuration graphic)
3.      then
4.      (system ?carverTool ?imagePath " -o " ?outputPath "/scalpel/ " ?graphicConfigFileName)
5.      (assert (data carving done))
6.      (system "nautilus " ?outputPath "/scalpel/")
7.      else
8.      (system ?carverTool ?imagePath " -o " ?outputPath "/scalpel/ " documentConfigFileName)
9.      (assert (data carving done))
10.     (system "nautilus " ?outputPath "/scalpel/")))
```

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 an open source tool that 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 (Find_SSNs, 2014). *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 (Find_SSNs, 2014). 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

3.3 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 search and sensitive number search. The

starting screen of the user interface of our prototype implementation of *AUDIT* is shown in Figure 3.

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 (Hibshi et al., 2011). 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 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.

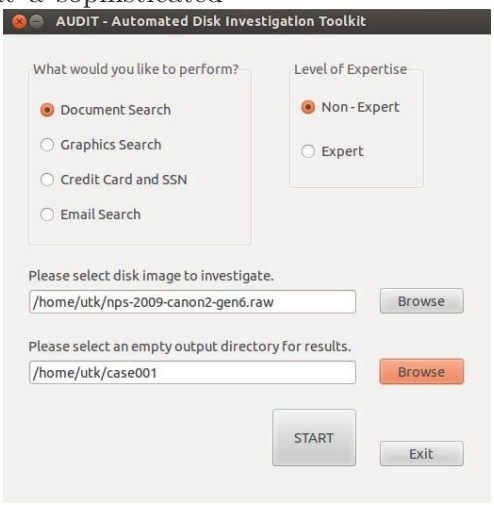

Figure 3 Starting Screen of the User Interface of AUDIT

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 asked from the user are 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. 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 (Garfinkel et al., 2009) and also Brien Carrier's digital forensics tool testing images (Carrier, 2014b). We also used some disk images that we created by adding files from Digital Corpora Govdocs1 (Garfinkel et al., 2009).

4.1 Graphics 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 graphics 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 (Carrier, 2005) 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 (Carrier, 2014a) does. It classifies the recovered files by file type and lets the user check whether any forensically interesting graphics 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.

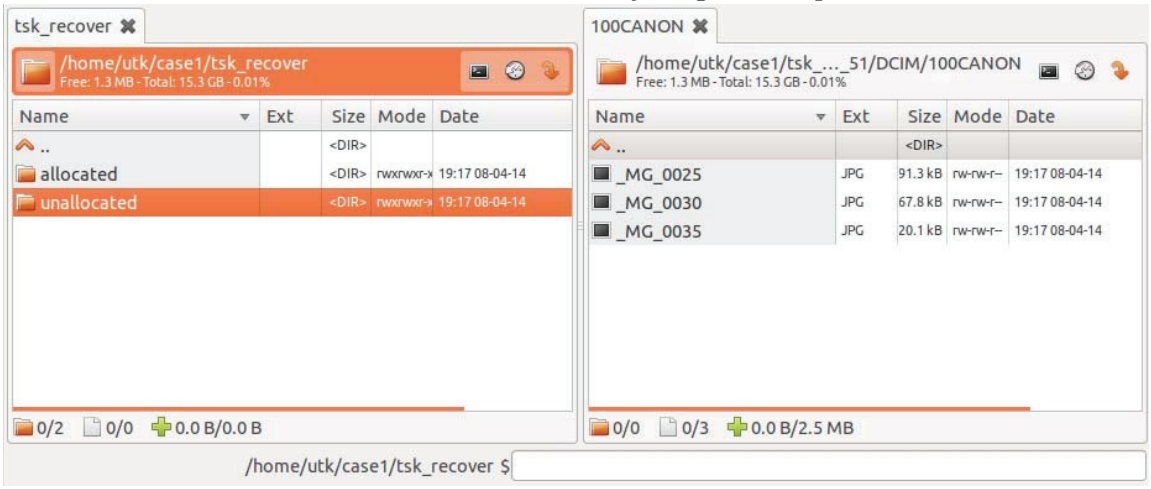

Figure 4 Popup showing files recovered from unallocated space

Assuming the user would like to go to the next stage, *AUDIT* starts *tsk_recover* tool 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. *AUDIT* then informs the user with information about the type of files (e.g., deleted files) that were recovered from the

disk image instead of only using 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 graphics 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.

4.2 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 3.2., 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 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.

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 5 Find_SSNs output report for Credit Card and Social Security Numbers

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 5.