Evaluating the Forensic Image Generator Generator

Christian Moch and Felix C. Freiling

Department of Computer Science
University of Erlangen-Nuremberg
Am Wolfsmantel 46, 91058 Erlangen, Germany
{christian.moch,felix.freiling}@cs.fau.de

Abstract. The Forensic Image Generator Generator (Forensig^2) is a system that allows to produce file system images for training in forensic computing. We report experiences of using Forensig^2 within a course on forensic computing. Apart from revealing the pitfalls when using artificially generated images in class, we argue that they can be used to quantify the difficulty of an analysis problem and, in turn, help to understand misinterpretation issues in practice.

Keywords: forensic images, privacy protection, forensic computing education.

1 Introduction

The *forensic image generator generator* (Forensig^2) [3, 4] is a program to generate "interesting", albeit artificial, hard disk images for forensic analysis exercises. The input to Forensig^2 is a script (a program) that produces an image, i.e., the generation of complex images can be automated. This is an advantage over creating such images by hand, as it is done today in many courses on forensic computing. The script itself also forms the *ground truth*, i.e., the sequence of operations that caused the state of the evidence. This is important for the examiner to evaluate the performance of a student in the digital investigation. Furthermore, since the image is totally artificial, there are no privacy concerns in comparison to using real evidence or second hand hard disks in class.

The structure of using the tool is depicted in Figure 1. As input, a script written by the instructor, which determines the actions performed on the image, is used. The input language is Python and so the use of random numbers in scripts is possible. The use of randomness makes it possible to create multiple slightly different images using only one input script. To make the image generation repeatable, Forensig^2 does not directly produce an image. In the first step it eliminates randomness and produces a generator for the final image. In the second step, the actual image is produced. This is the reason of the square in the acronym Forensig^2 because, in effect, it *generates* an image *generator*.

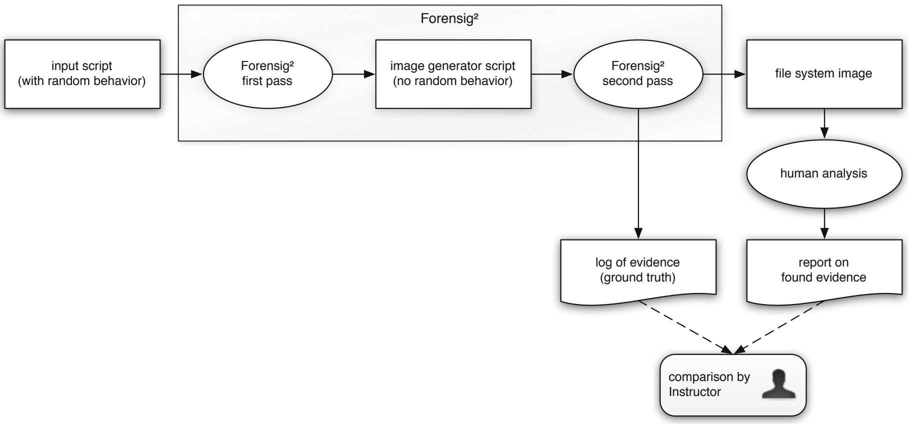

Fig. 1. Using Forensig^2 in digital investigation exercises [3]

1.1 Deficiencies of Forensig^2

Forensig^2 is used in different universities and with law enforcement to train students in forensic analysis. However, it has some clear deficiencies. First of all, since an instructor has to write an input script, it is not usable by instructors who have no programming experience. Another problem is that Forensig^2 has problems creating evidence caused by user interaction through the GUI. This is unfortunate, since such evidence is particularly interesting in practice. Some of the disadvantages can be compensated with manual intervention, by interrupting the input script at a certain time, after which the script can be continued. Finally, the usefulness of the tool has never been properly evaluated (or if so, we are not aware whether it has been reported).

1.2 Evaluating Forensig^2

We obtained a copy of Forensig^2 from the project website [4] and used it to perform four studies during an undergraduate course on forensic computing ("Forensische Informatik") at the University of Mannheim in the summer term 2010 in which more than 20 students participated. In each study, students had to analyze or create a hard disk image produced by Forensig^2. Each study evaluated a different aspect of Forensig^2 or digital forensics in general:

1. In the first study, we investigated the usefulness of Forensig^2 as a tool to create images for analysis by students. They were given an artificial image and had to write a report of their findings within six weeks. A comparison between the findings of the students and the script of Forensig^2 allowed us to precisely assess the coverage of digital evidence that students achieved.
2. In the second study, we wanted to test the usability of Forensig^2 from the view of an instructor. In this study, students had to create their own images

using Forensig^2. We analyzed the scripts from the students and measured the lines of code, the unique commands and the externally prepared files that were needed to create an interesting image.
3. The third study focused on logical and technical flaws within images created by Forensig^2. Logical flaws are mistakes made by the author of the script. For example, a logical mistake would be the use of Ubuntu Linux at a time before Ubuntu Linux was founded. Technical flaws are any functional mistakes that allow to detect the use of Forensig^2. For example, a known technical flaw is an irregularity in the unused space of the partition tables created by Forensig^2. Within this study we wanted to find other technical flaws too.
4. The last study had a more general scope: It aimed on determining the degree of difficulty of a specific forensic problem. Our students had to analyze three images within a fixed time in the lab and after each image they had to fill out a questionnaire. With a statistical analysis we were able to determine the degree of difficulty of the three images with respect to time-consumption and knowledge needed to solve the image. With the ability to seamlessly generate and vary images for such studies, Forensig^2 can enable multiple such studies in the future. In the long run, this may help in the quantifying the trust in an investigator's analysis depending on the type of evidence investigated.

1.3 Contributions

In this paper, we report on the results of the studies sketched above:

1. In the first study, most findings of students corresponded with the Forensig^2 report, although our students also found some irritating evidence unintentionally introduced by Forensig^2. The large overlap of found (intentional) evidence shows that Forensig^2 can be used for educational purposes. Most of the evidence that our students could not recover was due to the fact that this evidence was technically unrecoverable, e.g., a deleted file which was overwritten physically. The Forensig^2 report still represents this evidence although it is impossible to recover it. The unintended findings of our students pointed to some technical shortcomings in Forensig^2 and to inconsistencies in the "story" of the case. (see also study 3)
2. The second study showed the possible uses of Forensig^2 by instructors. We had more than 20 scripts written by students who had never used Forensig^2 before. The analysis of the scripts showed that it is possible to create interesting images with the use of only 20 unique commands (of Forensig^2) or even less.
3. In the third study the students were able to find further technical flaws of the tool. However, evidence that was not recoverable on the final image could almost always be traced back to input scripts that made the evidence technically unrecoverable, e.g., if a command overwrites the evidence completely or leaves no evidence at all. Students were also able to find some unintended traces within the image created by Forensig^2 which would make it possible to create a signature for a Forensig^2 image.

4. The last study resulted in statistical data that allowed us to prove which forensic problem is hard to solve and which is not. The data shows us that it is possible to use Forensig^2 to determine the complexity of a forensic problem. We were able to show that there was no difference between finding a regular file or a deleted file, while a deleted partition table is much harder to analyze with respect to time and difficulty of the analysis.

Overall, we conclude that the use of Forensig^2 in an educational environment is possible, although this task has to be approached with caution.

1.4 Paper Structure

This paper is structured as follows: We first give some technical background on Forensig^2 in Section 2. This part can be skipped if the reader is already familiar with the tool. We then present the results of the four studies in Sections 3, 4, 5 and 6. Section 7 concludes the paper.

2 Background

2.1 Forensig^2 in a Nutshell

Forensig^2 [3] offers the possibility to describe an entire computer together with its operation system. It is possible to perform a complete installation of the operating system in particular, to create realistic timestamps on the file system. This is done by using the system emulator Qemu. Python was chosen to be the scripting language for Forensig^2. It allows the creation of many different (but similar) disk images from one input file. Forensig^2 can access a source of randomness in the generator system but also can reproduce a particular image as well.

Forensig^2 input scripts can use a couple of auxiliary libraries that are part of the extension of plain Python. These libraries can be divided into five areas: (1) building the system, (2) time control, (3) execute operations, (4) documentation, and (5) auxiliary modules.

Building the System. The building modules of the system are `System`, `Disk`, `CPU`, `Memory` and `GenericCard`. With a combination of these modules it is possible to define an entire computer system including the installed operating system.

The `System` module is the central module which takes care of all other system components. Building an entire system uses the metaphor of building systems in the real world. To install a CPU, the system module provides a method `System.addCpu(CPUObject)`. In the same way a hard disk can be installed or any other component. The module is also responsible to power the defined computer system on or off.

The `Disk` module divides itself into `SystemDisk` and `DataDisk` and provides the basic functionality for a hard disk. The `SystemDisk` represents a hard disk with an installed operating system, while the `DataDisk` only has a filesystem. A data disk, however, offers more possibilities in partitioning than a system disk.

Time Control. Since a lot of important information is conveyed through time stamps, taking control over time is very important for building disk images. The `Time` module provides two different time structures, the `truetime` and the `systemtime`, both starting at January 1, 1970. The concept of *true time* reflects the time elapsed since the script started, while the concept of *system time* represents the actual time the system uses (the system clock). True time only flows in one direction, i.e., it can be increased by arbitrary values but can not decrease. So the true time acts like the time in the real world, with the difference that time travels to the future are allowed. In contrast to this behavior the system time has no restrictions.

Like the real time in the real world, both timelines are flowing constantly. Every second the script is running, both timelines increment by a second, which is important when using the true time. When the user sets the truetime to a determined time, lets say May 19, 2009, 12:00 and then installs a system, the truetime constantly flows while installing. If the install takes 20 minutes, then true time will be May 19, 2009, 12:20.

Executing Operations. With the modules discussed before, a system can be installed and the time of these actions can be determined. Having only a fresh installed system on a given time would not be interesting for forensic analysis. Further commands should be executed on the system, so a kind of user interaction has to be integrated. The `commander` module models user interaction. Every command that runs on the installed system must be passed through the `commander` module.

Documentation. Knowing the ground truth is important for the instructor so that results obtained from forensic analysis by students can be well assessed. Most actions are documented by the input script itself. However, if randomness is used, it is necessary to be able to output specific values into the log file of the image. Thus, in every module a logging function is implemented that writes the actions taking place to a single report file. Sometimes, the instructor also wants the status of a module at a specific time within the generation process to be documented. The `Reporter` module provides this functionality, since it writes a report on demand while the generation process takes place.

Auxiliary Modules. The auxiliary modules collect helpful support tools for forensic image generation. The `Converter` module provides many kinds of conversions: From decimal to hexadecimal, from ASCII to decimal, and even some specific conversions like CHS (cylinder, head, sector) to LBA (logical block address). The `Hexer` module can be used to manipulate files byte by byte.

3 Analysis of Artificial Images in Class

In this exercise, the task for our students was to analyze an image produced by Forensig[2]. We did not tell the students about the source of the image; they did not know if the image is artificial or not.

In total, 24 students participated in this exercise. They were randomly assigned to one of two different image stories which we now explain.

The first story is taken directly from the code base of the Forensig^2 distribution [4]. It describes the criminal acts of a famous German blackmailer in the 1990's. The alias of this blackmailer was Dagobert (the German name for Uncle Scrooge McDuck). He blackmailed a department store chain several times and to enforce his demands, he planted bombs in the stores. The hard disk image contains many hints to this story. There are several text-files where the blackmailer writes down his thoughts before and after his criminal acts. There are also some newspaper articles and some maps of the department stores on the image. The time line of the files corresponds to the time line of the real world story. To confuse the analysts, a second partition exists on the image containing files which are not related to the case. In the story, this partition is used by Dagoberts wife. At the end of the time line, some files are deleted and finally the partition table was destroyed.

We created the second story ourselves, and it tells a modern version of the Robin Hood saga. The basic idea of "robbing from the rich and giving to the poor" was mimicked by text files containing an amount of money. These text files were digitally signed by the Sheriff of Nottingham and distributed (copied) to the rich people. In the story we build a system with an installed Ubuntu Linux. On this system, we created four users. "Robert.de.Rainault" which represents the Sheriff of Nottingham, "Robert.Hut" as Robin Hood, and two other users which are called "rich" and "poor".

In the story line the user "Robert.de.Rainault" created some digitally signed text files with a certain amount of money and moved the files to the home folder of the user "rich". "Robert.Hut" moved these files to the home folder of the user "poor", furthermore he was able to steal (copy) the private keys of "Robert.de.Rainault". From now on he was able to create his own digitally signed text files. "Robert.Hut" created some of these files and copied them to the home folder of the user "poor". In the end, the home directory of "Robert.Hut" was deleted. To have at least one piece of evidence that is not related to the case, we made a small brute force attack on the system. The user "root" tried to login via `ssh` 10 times, traces could be found in the `auth.log` file.

All in all, we had a great coverage between the evidence we had hidden on both images and the evidence found by our students. Some of the students found all traces that were recoverable. They also analyzed the side scenes we placed in the image. Nearly all students were able to recover the end state of the image and could determine which files were located in which directory at the end. Some students also managed to recover the time line, which is particularly essential to understand the "Robin Hood" case.

An interesting approach to recover the time line was made with a field experiment by a student in the "Robin Hood" case. He placed the files in the directories where he assumed they were stored at the beginning. Then, he executed every single command he found in the *bash* history of each user, to prove which order of the commands ends in the end state he found on the image. Finally he could determine the exact order of the commands and the origin storage place of the files with a 100% coverage to the Forensig^2 script.

To quantify the coverage we analyzed the reports of our students. We defined five main parts they had to find during the investigation. This parts were related to the criminal act we introduced them to analyze.

Overall we had a coverage of about 77% in both cases ("Dagobert" and "Robert.Hut") which deviates from 40% to 100%. In both cases we had some students who found all the traces we placed on the image. For the "Dagobert" case, we defined the following pieces of evidences: *maps of three cities*, *a picture of the blackmailer*, *newspaper articles of the criminal acts*, *diary of the blackmailer* and *a picture of bomb*. While only 38% of our students found the newspaper articles, 92% found the *picture* and the *diary entires*. The *maps* were found by 84% and the *bomb* was found by 76%.

On the "Robert.Hut" case, we defined the following pieces of evidences: *signed files from Robert.Hut*, *signed files from Robert de Rainault*, *invalid signature*, *stolen private key* and *reconstruction of the time flow*. On this study, 64% of our students figured out that the user *Rober.Hut* had stolen the private key from the user *Robert de Rainault*. Therefore, 72% noticed the invalid signature on one file and also 72% managed to reconstruct the time line in the correct order. While 81% found digitally signed files created by Robert.Hut, all of our students find the digitally signed files created by Robert de Rainault.

We did not analyze the coverage of the side scenes, although we had reports that described these scenes in detail. The investigation of the side scenes were not explicitly described as task of the investigation process and therefore many of our students did not write their findings about these scenes in their final report.

The reports produced by our students varied from 6 to 19 pages, the appendix of the reports from 0 to 1144 pages. The mean values of an average report is around 9 pages. These results do not differ from other courses we held earlier with real (second hand) hard disk images instead of artificially generated images. An interesting result is, that the reports of a `SystemDisk` is longer than the report of a `DataDisk` (see Figure 2). The mean value of a Dagobert report, which uses only a `DataDisk`, is 9.61 pages, while the mean value for the "Robin Hood" image is above 10.

4 Evaluating the Usability of Forensig^2

In the next study, we aimed to evaluate the usability of Forensig^2. To measure the usability, the students should create their own images using Forensig^2. The study resulted in many different images created by our students. Some students borrowed their stories from real life, movies or books while other students wrote the story from scratch. The result were stories of all kinds of criminal acts, for example bank robbery, kidnapping and murder. A creative script mimicked the time line of one episode of the series "Knight Rider". The system build by the Forensig^2 script should be the system of "KITT" an artificially intelligent electronic computer built into a car.

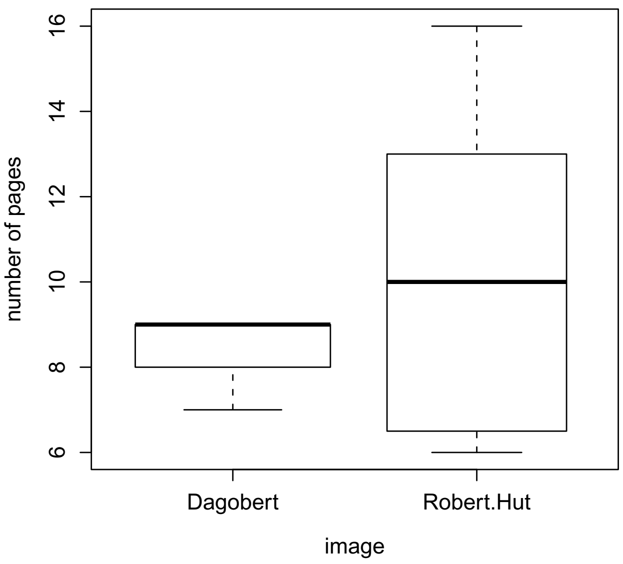

Fig. 2. Pages of the reports for different images

A closer look at the Forensig^2 scripts of our students showed us how much effort is needed to create a forensic image. The statistics of the scripts showed that the mean value of lines in the script is 93.13 with a maximum of 208 and a minimum of 47. More interesting is the use of commands in the script. We counted the unique commands of each script and calculated the mean value with the result of 18.06 commands per script (see Figure 3). This leads us to the conclusion that the use of Forensig^2 makes it possible to create an interesting image with about 100 lines of code and the use of about 20 commands.

However, not every file or every case can be solved using the pure Forensig^2 scripting language. For a good script there is also the need for some preparations. With Forensig^2, it is not possible to create complex files, like a picture of a criminal offender, emails or browser histories. Therefore the author of the script has to prepare these files. Afterwards, with the help of the Forensig^2 tool, he is able to copy these files to the image. Our students prepared the files in different ways. Some students created each single file on their own. The results were text files or PDF-Documents containing some hints related to the criminal act. Other students created a huge amount of slightly different files, for example hundreds of pictures created by a shell script using *imagemagick*. Out of these files, they randomly pick some files in the Forensig^2 script, with the result that with every run Forensig^2 will create a slightly different image. Finally, the students used a fresh installed computer to create some typical files, for example a browser history. They copied these files and used them later in the Forensig^2 script.

In our study, the mean value of prepared files was about 100. While one student created about 1500 files with a shell script which distorted the mean value.

Without this student, the mean value decreases to 13.56, with the minimum of 0 files and the maximum of 22 files prepared (see Figure 3). The preparation time to create about 14 files is manageable, although it should be considered in the calculation of the amount of time needed to create a whole Forensig^2 script.

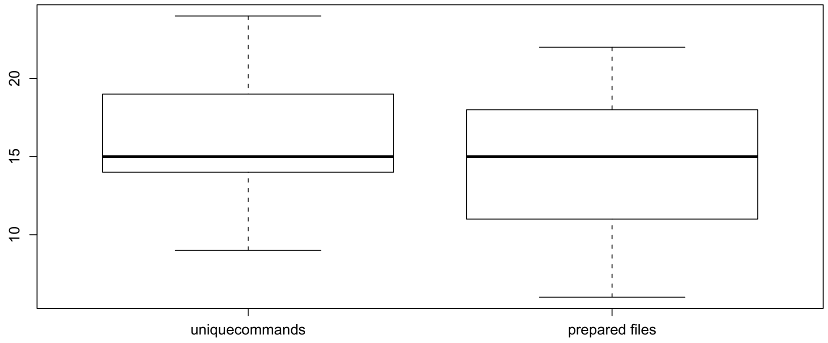

Fig. 3. Unique commands and prepared files per script

5 Testing and Fingerprinting Forensig^2

In the next study, we sought to evaluate Forensig^2 in two aspects: Firstly, we wanted to perform some tough functional testing. Students were given an input script together with the image generated by the script and had to analyze the image with the focus on the evidences described in the script. They had to prove that every single command of the script had been executed and had to figure out which traces of this command were left on the image. Secondly, students had to look for traces left by Forensig^2 itself in the image, i.e., they were asked to develop tools to *fingerprint* images produced by Forensig^2.

5.1 Testing Forensig^2

The coverage of the commands used in the script and the evidence found on the image was nearly 100%. Some evidence was not recoverable because it was completely overwritten by other data, i.e. copying a file over an existing file. However, a step-by-step analysis where the script was executed line-by-line and after each command the resulting image was analyzed showed that the commands had been executed. So far the the evaluation of Forensig^2 did not show any differences between the input script and the produced image, although we did not cover all functions Forensig^2 provides, the most commonly used functions are reliable.

5.2 Fingerprinting Forensig^2

As a second task of this study students had to find traces of the use of Forensig^2. In particular, they had to find evidence in the image that it was an artificially generated by Forensig^2. There are two ways to distinguish an artificial image from a real image: (1) Identification of technical issues of Forensig^2 in the image that allow it to be identified as artificially created, (2) logical mistakes in the story or the image, for example, the use of Ubuntu Linux before Ubuntu Linux was founded.

The results from this study were mostly logical mistakes where the author of the script used technologies which were not available at the time the script should take place. For example, in the Dagobert script described above, Dagobert used Google Maps screenshots of the blackmailed stores. However, the time where the story is set in was in the year 1994 and therefore Google Maps was not available. Another popular logical mistake was a very short time offset between actions usually executed manually. For example the execution of several commands on the command line within a few seconds is unrealistic and points to an artificially generated image.

There were not many findings regarding technical flaws of the tool. Since the students were undergraduate, they were probably too inexperienced to find technical flaws on the image, although some students found at least some traces of the use of Forensig^2. For example, some students proved the use of Qemu which is used as a virtual machine in Forensig^2. The absence of Qemu on a `SystemDisk` is strong evidence against the use of Forensig^2 and therefore a good way to falsify the hypothesis that Forensig^2 is used.

A way to affirm the usage was found by some students in the partition table of the `DataDisk`. Here Forensig^2 has a minor issue in the algorithm resulting in 62 bytes of unallocated space between the first and second partition and 63 bytes between any further partition. Though we knew about some other possibilities to test if an image is built by Forensig^2, no student of this exercise was able to find it without further hints. Other possibilities could be the the use of the linux-virtual kernel or a strange behavior of the default user. The default user connects regularly to the system and executes only one command. This is the heartbeat of Forensig^2 to prove the system's availability. On a `DataDisk`, it should be harder to find traces, the only subsystem built from scratch was the `Partitioner` module. Every other module uses regular Linux commands to interact with the `DataDisk` and therefore it is hard to distinguish between normal and artificial interaction on a technical level.

All in all, we learned from this exercise that it is not easy to create an artificial image which could not be identified as artificial. Although our students did not find many of the technical flaws we identified so far, it is easy to fingerprint Forensig^2 today.

6 Comparing Hardness of Investigation Tasks

In the final study, we did not evaluate Forensig^2 itself but rather tried to used the tool to compare and thereby evaluate different investigation tasks.

6.1 Rhino Hunt

The study was inspired by a previous DFRWS challenge [2]: Students were given three images. On every image our students had to solve the same task: they had to find a picture of a rhinoceros and figure out the file type and the filename of the picture file. The image was only declared solved when both items were found: filename and file type.

In this study, 23 students worked in real time in the lab at our university and performed the test at the same time under the same conditions (a linux system booted from a popular forensic live DVD). Due the time of an exercise slot of 90 minutes, we decided to pass 3 images to the students and grant 60 minutes of time for solving the images, 20 minutes per image. Within that time the students had to analyze the image and answer a short questionnaire. If a student finished the analysis earlier, he was allowed to continue with the next image immediately. The time limitation of 20 minutes should bring the participants in a time-pressure situation. This should be more comparable to real forensic analyses where time is a resource which is not endless. The images were solvable in less then 20 minutes when knowing the "ground truth", without this knowledge, it should be challenging to solve.

With the questionnaire we could measure some subjective data from our students like perceived time-pressure or the difficulty of the image. Beside this data, we also had some objective data like analysis time or the fact whether or not they had solved the image.

6.2 The Three Images

We now describe the three images that had to be solved by the students.

Image 1. The first image contained a file system formated with ext2. The position of the file system was correctly mapped in the partition table. Only a single file was stored in the file system which contained the picture of the rhinoceros. The tricky part of the image was that the file extension in the file system was not identical with the file type of the picture.

Image 2. Like the first image, the second image had only one file. However, the file was first moved on the file system from one directory to another and afterwards it was deleted. The students had to recover the file, which contained the rhinoceros and figure out the filename and file type. This time the file-extension of the file was not changed.

Image 3. The third image had a deleted partition table signature. The files in the file system were untouched, no single file was deleted. The task for our students was to restore the partition table signature *AA55* [1]. Afterwards they were able to find the file system and finally they would find the picture of a rhinoceros inside the file system.

6.3 Hypotheses and Findings from Subjective Data

Our hypothesis was that the images were getting harder to solve with each image. We suspected Image 1 to be the easiest to solve, Image 2 should be an intermediate level and Image 3 should be the hardest image out of the three examples. This hypothesis should reflect on the data by an increasing analysis time, decreasing percentage of solved images, increasing subjective felt time-pressure and increasing subjective felt difficulty of the images. To prove the hypothesis we analyzed the data from the questionnaire combined with the objective data we could measure.

We found clear data that Image 3 is significantly harder to solve compared to the first two. The surprising result in our data was that there was only a marginal bias that Image 1 differs from Image 2 regarding the level of difficulty. With statistical methods, it was not possible to distinguish Image 1 from Image 2.

After our students solved Image 2, we asked them in the questionnaire if the image was more difficult to solve and if the image was more time-demanding compared to Image 1. Later we asked the same questions to compare Image 3 and 2. The answers were collected with a 9-point scale, were 1 means "much easier"/"much less time-consuming" and 9 means "much more difficult"/"much more time-consuming". A response of 5 means, there were no differences between the two tasks.

The mean value between Image 1 and 2 was 5.25 (difficulty) and 4.3 (time-consumption) (see Figure 4). The confidence interval calculated with a one sample *t*-test was between 4.59 and 5.9 (difficulty) and 3.57 and 5.029 (time-consumption). On each factor 5 is within the confidence interval and therefore the two images do not differ from each other. A minor tendency could be interpreted in the time-consumption factor, the value 5 is on the edge of the confidence interval, maybe with more participants it could fall significantly below 5. The surprising result is, that there is no difference between a file system where no data is deleted and a file system were data is deleted. Both problems seem to be equivalent. Much more surprising is the result that the tendency is towards less time-consumption for Image 2. This is in contrast to our initial hypothesis.

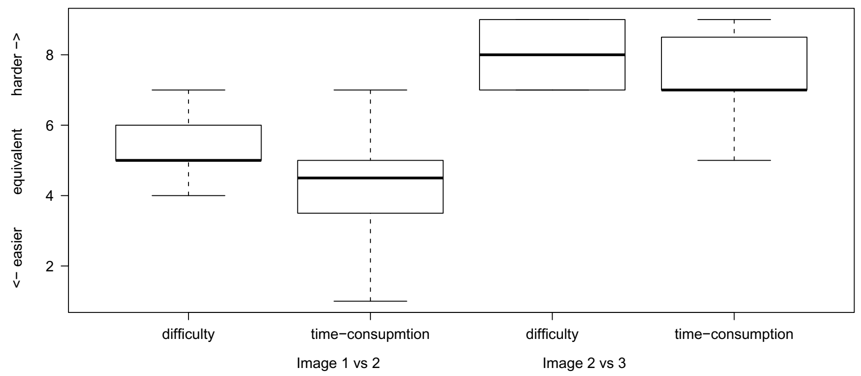

Fig. 4. Comparison of Images 1, 2 and 3

The analysis between Image 2 and 3 shows that the data conforms with our initial hypothesis. The mean of difficulty was at 7.8 with a confidence interval between 6.9 and 8.7. The time-consumption mean value was at 7.05 with a confidence interval between 6.12 and 7.98. The error probability of the *t*-tests tends against 0 (4.27e-13 and 4.64e-13). As in both factors the confidence interval is above 5 we can be sure that the problems differs from each other and Image 3 is significantly harder (in both factors, time and difficulty) to solve.

6.4 Findings from Objective Data

Of course this data is subjective, the participants themselves evaluated the forensic problem. There are a lot of factors which could influence this subjective impression. In the following part, we describe our objective measured data and compare it with the subjective data.

The objective data of the analysis time shows, like the subjective data, no significant difference between Image 1 and 2, however between Image 2 and 3 it becomes clearly significant. The mean values of the analysis time in Image 1 were 9.81 minutes against 9.15 minutes for Image 2 (see Figure 5). A two sample *t*-test leads to a *p*-value of 0.68 which is not significant. Therefore we cannot prove if the analysis time differs between Image 1 and 2. The mean values of Image 2 and Image 3 are 9.15 minutes versus 13.68 minutes. The *t*-test shows here a clearly significant data with an *p*-value of 0.026. The difference is proved statistically when the *p*-value is below 0.05. The confidence interval is between 0.56 and 8.489 which means that there is at least a minor difference of half a minute in the analysis time. With more participants, we could determine the difference more exactly, however, we have shown that there is a clear difference between the two images.

To investigate the difficulty of the three images in an objective way we analyzed how many participants actually solved the image in each level. The data of Image 1 shows that many participants were tricked by the wrong file extension. Only 13% of the participants were not able to solve the image, 39% were able to solve the image at least in parts (either the filename or the file type was correct) and 48% could solve the image completely. In Image 2, 55% of the participants could not solve the problem and 45% were able to solve it. The last image was the hardest image to solve for our students, 79% could not determine either the filename or the file type and only 21% were able to solve it.

The data shows clearly the descending proportion of solved images. Starting with 48% completely solved plus 39% partially solved, over 45% solved images in Image 2 and ending with only 21% solved images with the last image.

Both data, subjective and objective data shows clearly that Image 3 is the hardest problem in the factors of time and difficulty. The difference between Image 1 and Image 2 does not exist and therefore the problems are equivalent. The main difference in Image 1 and 2 was, that in Image 2 the file was moved

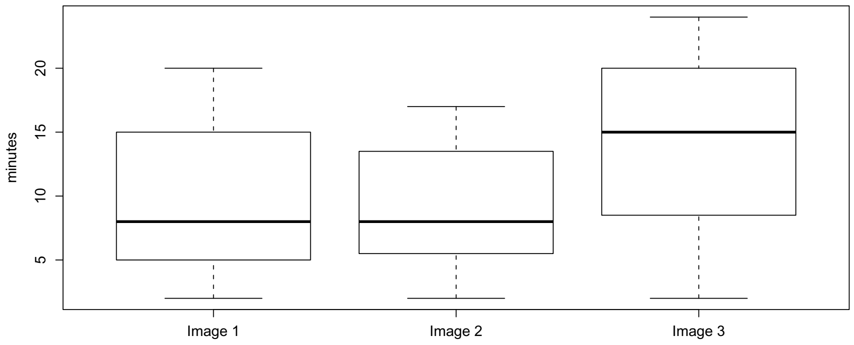

Fig. 5. Analysis time of Images 1, 2 and 3

on the file system and deleted afterwards. The data shows, that it makes no difference for a forensic analyst to recover a file or to find the file directly in the file system. In our interpretation of the data, it makes sense that the problems are equivalent. With current forensic tools for hard disk analysis, like Sleuthkit (in combination with autopsy), the forensic analyst gets access to either deleted and normal files on the fly.

7 Conclusions

Forensig^2 is a robust and useful tool to create artificial hard disk images automatically. The first three studies described in this paper, showed that (1) the findings on images produced by Forensig^2 had a great coverage with the input script for Forensig^2, (2) it is possible to create images with about 20 Forensig^2 commands and (3) the use of Forensig^2 is detectable. The last study used images as a central part of the study. Therefore the images had to be created in a controlled way to eliminate any interference which could influence the study itself.

However, the usability and the capabilities of the tool have to be improved. At the moment, it is only a console application, which makes it hard to use for inexperienced users. Overall, it is a good tool which opens new possibilities for digital forensics education, although there is a lot of work to do in the future to make the tool handy and comfortable to use, even to inexperienced users.

Acknowledgment. We wish to thank the anonymous reviewers for their constructive feedback.

Thank David Hendricks, Johannes Stüttgen, Chad Sentman and the students of the course in "Forensische Informatik" at the University of Mannheim in 2009 and 2010.

References

[1] Carrier, B.: File System Forensic Analysis. Addison-Wesley (2005)
[2] Richard III, G.G.: Rhino Hunt: DFRWS 2005 Rodeo Challenge (2005), `http://www.cfreds.nist.gov/dfrws/Rhino_Hunt.html`
[3] Moch, C., Freiling, F.C.: The forensic image generator generator (forensig2). In: Goebel, O., Ehlert, R., Frings, S., Günther, D., Morgenstern, H., Schadt, D. (eds.) IMF, pp. 78–93. IEEE Computer Society (2009)
[4] Moch, C., Freiling, F.C.: Forensig^2 homepage (2009), `http://pi1.informatik.uni-mannheim.de/forensig2/`