

FEASIBILITY OF A DIGITAL FORENSIC LOGGING TOOL

Researcher: Matthew Bayliss

Supervisor (1): Asher Rashid

Supervisor (2): Martin Brown

UNIVERSITY OF TEESSIDE
School of Science & Engineering

Document Statistics

Abstract Length	250 words
Total Length	4561 words
References	13

Correspondence

Email: matthew.bayliss1990@gmail.com

Phone: 07891145004

Feasibility of a Digital Forensic Logging Tool

Matthew Bayliss^{a,*}

^a*Teesside University, Borough Rd, Tees Valley, Middlesbrough TS1 3BA*

Abstract

The aim of this project is to discover the feasibility of using a computer program to collect contemporaneous notes for Digital Forensic Investigations. This will attempt to show that it is possible for an application to collect notes for the examiner which will save time and introduce more accuracy into the examination of computer systems, by reducing the amount of human effort in the investigation.

The first part of the research will focus on the requirements for forensic audit trails. Primarily to see what information would need to be collected by the tool and what must be done to make this information usable in a court.

The research will then go on to look at digital forensic tools and attempt to find criteria these must fulfil to be accepted into court as evidence, or have evidence collected by them accepted in a court.

A third part of this research will be into what Application data can be harvested to show what the examiner's actions were. This part of the research will be targeted towards gaining information from Autopsy 3, which is the forensic application the prototype will be tested against. The combination of these three parts will give the design requirements for the creation of a prototype application.

The prototype application may only contain a limited set of the features required by parts one and two. However, it should show whether it is feasible for a computer program to log an examiners actions and produce a report on them.

Keywords: Audit Trail, Logging Tool, Software, Forensic Tools, Investigation Tools

1. Introduction

The ACPO Good Practice Guide for Computer-Based Electronic Evidence ([ACPO, n.d](#)) states that all actions undertaken on digital evidence have to be logged so that an independent third party can later achieve the same result. There is no clearly defined way of doing this, though the common practise is for the examiner to type or hand-write their notes.

This article aims to *discuss the feasibility of using a computer program to log many of the actions an examiner performs*, such as starting a new case. This, if shown to be feasible, would remove the strain from the examiner to write down each ac-

tion in minute detail and instead focus on what this information means.

The first stage of this project can be broken into two sub-goals. Firstly it is important to know *what are the requirements for forensic audit trails* and secondly *what are the requirements for digital forensic case tools*. The answers to both questions can be used to formulate the requirements for the computer program mentioned above.

After the initial research a prototype application will be created. The application will log only Autopsy Version 3 (Autopsy 3) and will not contain the functionality that would be expected of a final tool. However, it should provide an base for testing the feasibility of the tool as a whole.

The final stage of this project will be to test the prototype tool itself. This will be done by running through a number of test cases selected from online resources. After each test there will then be a loop back to the development phase to fix bugs, no fixes will be made at this stage.

*This article was created as part of a BSc Project. As such the author will have graduated at the time of publication. All correspondence should be by email.

Email address: matthew.bayliss1990@gmail.com
(Matthew Bayliss)

In short the article aims to answer the following questions:

1. What are the requirements for forensic audit trails?
2. What are the requirements for digital forensic logging tools?
3. Is it feasible for a computer program to log these actions on behalf of the examiner?

2. Requirements

2.1. Audit Trails

The main requirement for a forensic audit trail is that it be reproducible (ACPO, n.d). This guideline is backed by the Crown Prosecution Guidelines (Crown Prosecution Service, 2010) which states "[The investigator] should keep records of all the work you have carried out and any findings you make in relation to the investigation".

Another requirement of audit trails can be found in Casey (2011) who gives the additional requirement that the audit trail also contains the investigators hypothesis. This adds significant depth to the notes taken which could become meaningless without annotations.

A third requirement for audit logs is that they can be made tamper evident as outlined by Richard III et al. (2007). This requirement should aim to ensure no log entry was changed after the fact. Or if it was (for example if a typing error was made) then the change is clearly highlighted. Editing of any part of the data should be disallowed as this would destroy the reproducibility (i.e. a step could be changed to ensure that a wrong result is obtained).

2.2. Digital Forensics Software

There is a distinction between "standard" software and forensic software. Digital forensic tools will often need to parse data which is corrupted. Garfinkel (2012) gives the example of data being corrupted by partial overwriting but also in dealing with encryption the only way to even begin an investigation is to work with data that can become quickly corrupt as shown by Müller et al. (2012).

The primary source chosen for the requirements for this tool is Carrier (2003). As this is the most cohesive source for forensic software tools that could be found. The requirements outlined are

Usable The usability of software is important in digital forensics as it is not only the investigator, who can be thought of as highly computer literate that needs to be catered to. In addition to this the software could also potentially be used by the police and courts. This means that all outputs need to be in a format that assumes no knowledge or computer background.

In addition to this the usability of the output should be maintained as cross-platform so any output will be given in a format readable on all systems with minimal effort (HTML, plain text, etc).

Comprehensive A tool must be capable of showing all of the gathered data and provide details on the source of The information. This can be at odds with the usable criteria, if we take the example of a computer hard disk drive. There is information on the file system that is required for the system to function that the investigator will very rarely need.

This information should be available for the software to pull up when it is requested. However it would be in the interests of usability to hide this information except for where it is relevant. This will not be initially required as there will be little data logged. However this could become relevant later in the applications life cycle.

Accurate The accuracy of tests should also be displayed to the user. For 100% accurate tests, such as reading from trusted log files, which will be used by this system, this is not necessary. However, for any test which is not accurate the chances of false-positive, false-negative, etc should be given alongside the data.

For the prototype this is largely irrelevant as the autopsy 3 log is trusted. However there is no way of ensuring that only autopsy 3 logs to the file. A malicious application could modify this file and the application would not be able to detect the difference.

Verifiable Software should make use of verification to ensure that data has not been changed this is important as any change to the information could obscure the meaning. For this purpose it is possible to use cryptographic

hashes to verify the integrity of the information. Another way of defining verifiable in the digital forensic context is that of what the software does. The best way of doing this is to make the source code visible to the investigator. This opinion is backed by Carrier (2002) who states that under the Daubert test "open source tools may more clearly and comprehensively meet the guideline requirements than would closed source tools". In the article Carrier (2003) Deterministic is also a criteria, however this can be kept under the title of verifiable. The meaning of deterministic in this context is that an input to the system should always result in the same output, as opposed to a random number generator which should give a different output each time it is run.

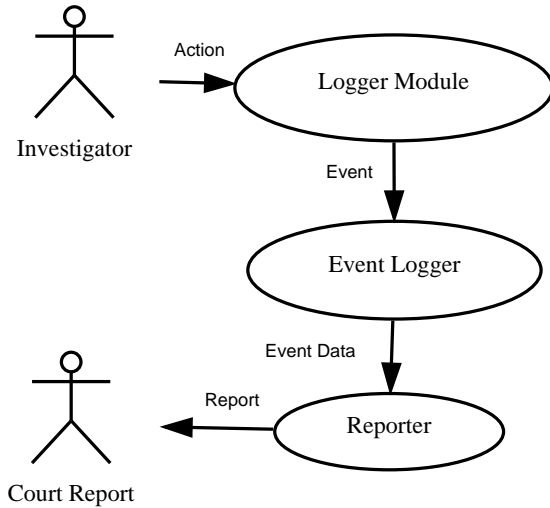


Figure 1: This figure describes the layout and public / foreign key relations of the database used by the logger prototype.

3. Design

3.1. Software Development

The program will be designed and constructed according to the Waterfall model for software development. This structure fits best with this project as research needs to be done to complete the remaining steps.

The requirement stage of the Waterfall model is already fulfilled by the previous section of this article. The current section describes the design of the software, which will then be implemented and

tested in the testing section. After each stage the work done is then "locked" to its state with the exception of implement and verify which tend loop while bugs are fixed.

For future work it may be better to use another development cycle such as Agile, which allows for new criteria to be plugged into the development process as the investigative process is rarely static. However for prototyping based on fixed criteria this model works best.

3.2. Program Architecture

The application must accept input either directly from the user or by parsing log files from multiple places. Process this information for metadata and then store the information in a database. Java will be used for the prototype application as it has a large library of code that can be used to quickly create a prototype application.

The application will require a modular architecture as it would be impractical to create one application to log every forensic application. This method will allow for a single purpose module to log each application individually, this should reduce the complexity of the code.

In practical terms this means that there is a central event logger module, an observer, with an interface that allows new modules to be added at any time. This will allow the application to be extremely simple overall as the eventlogger will only require a single method for event logging which will be usable by all modules because of the nature of the observer / observable design pattern. This is shown in Figure 2

The logging application produced for prototyping is written in Java. This was primarily due to the speed at which applications can be developed in the java language. Java does have some advantage over other languages such as its portability and it can be much easier to program in than languages such as C which will allow for external entities to create modules with ease. As such Java should be a strong contender as the language for any tools built on this research.

3.3. Database Design

Once the logger modules have collected their assigned data and sent this to the eventlogger the data will need to be stored. This will be done through the Derby database format which is natively usable in Java. For the final tool a stronger database engine may be required such as either MySQL which

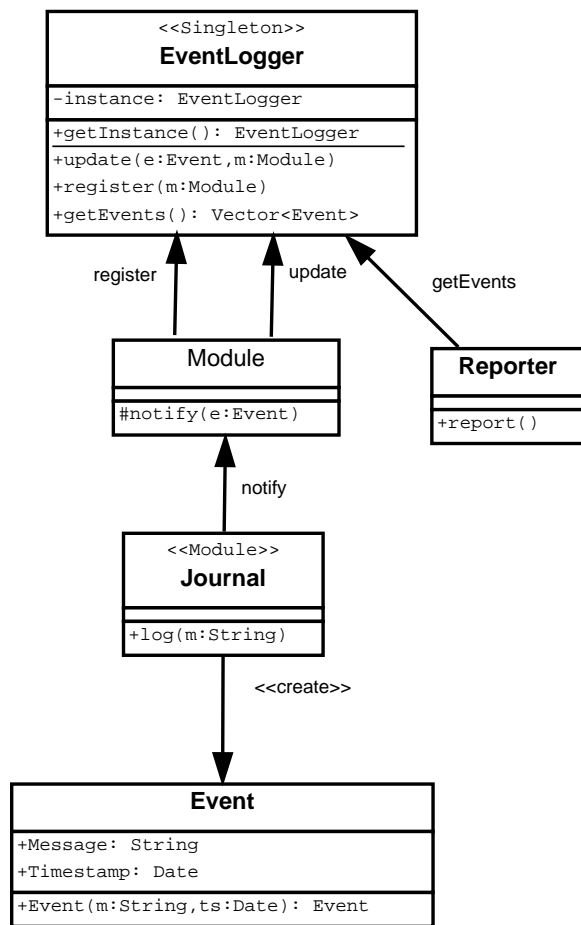


Figure 2: This figure is a basic class diagram containing the most important aspects of the core classes.

would allow for the logging to be done on a different machine than the storage, which would make the database more secure.

The primary table in this database should contain a list of events with the a description and the event module. In keeping with good practise for databasing, the sender module will be stored in a separate table and linked through a primary key. The design of the database is shown in Figure 3

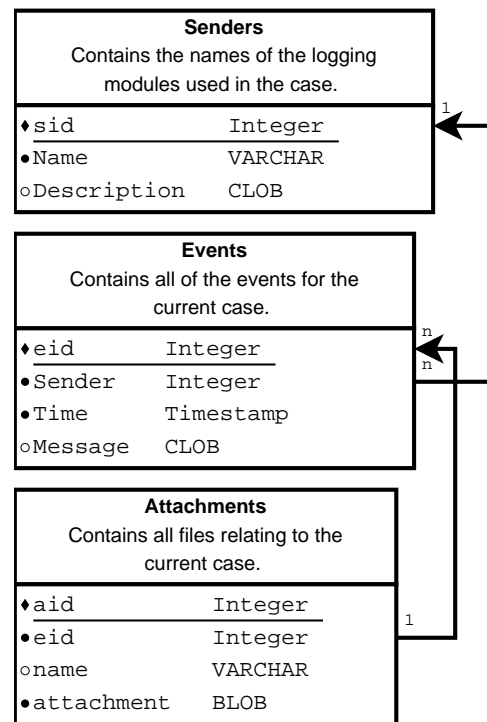


Figure 3: This figure describes the layout and public / foreign key relations of the database used by the logger prototype.

The first module created will allow for the examiner to make notes on the investigation. This will allow a basic test of the above functionality and create a base for the autopsy module to be created.

3.4. Autopsy Module

The autopsy module will work by reading the log files generated by autopsy ([Carrier](#)). This will allow the information on the active case to be pulled into the software's database. For the prototype only a subset of the events will be captured.

The module will work using Java's operating system interface to listen for changes to the autopsy log folder. Once a change is detected the file will be opened and any new lines will be captured and sent

to the event logger. Using the Java Native Interface means that if the file system in use supports it the update events can be discovered without polling the disk [Oracle](#).

An alternate way of doing this which may be required to log more advanced data would be to add a new module to autopsy itself. This could be done by monitoring the Autopsy Blackboard `CitePaskPlatform`. This method of accessing data could provide easier access than reading files.

4. Features

4.1. Case Management

The initial view of the application is somewhat empty, this is because it does not automatically connect to a database. The intention behind this is an examiner will have many cases on the go at once and mixing the notes for different cases would lead to doubt being thrown on the examiners work. When the new (or open) button is clicked on the interface a file manager will appear for the examiner to select where they want the notes for this case to be. The file created by the software will then start keeping logs for this case.

There is no set format for where these files need to be as the log modules will find what they need regardless. However, a suggestion would be for them to be placed with other files from the same case. As the database is self contained it will not interfere with any other software.

4.2. Journaling and Autopsy Logging

The journal module can be used by an examiner to make notes on the case, this is useful for contextual information. For example the addition of 5 images to a case even if they are well named will be hard to understand. But if the investigator comments on this process or give extra detail then this will be much more understandable. The journal module can be accessed by typing in the applications text area and typing return when finished.

The autopsy logging module, when activated, searches for the autopsy log files and then monitors for certain patterns that are related to events, see Fig ???. This module can be extended to look for anything stored in these logs. Currently it ignores most of the file as a lot of the entries are for debugging and other uses. This module will need additional work to be made portable. as when moved to linux the directory structure it searches for will

not translate. This cannot be done now as Autopsy does not support other operating systems.

4.3. Reporting

The report function pulls the entire database and streams this into a HTML File. Currently this is the only format for output however it would be possible to create a modular output format to support many different formats. This would be advantageous as different courts / examiners / police forces etc will prefer a different format and this can be made to give them something they will feel is familiar. HTML was chosen for this prototype as it is accessible to most systems and is easy to create.

All reports generated for the case will appear in the "reports" subdirectory of the folder created at the start of the process. Each time the report button is activated a new folder will be created for the report data. This could result in a large file if used too often.

5. Testing

5.1. Plan

The testing process was done using example works, both case studied given by Teesside University or found online such as "Joe Jacobs". This gives a more realistic look at how the performance of this application would improve case work.

Some simple functionality testing was done as development was ongoing (Modules were initially tested in this way). This was primarily to ensure that each class behaves as it should with the others and not a serious test of the application's abilities. However as of version 0.1.0 ¹. Testing was based on a case file.

5.1.1. Test 1

Test 1 was completed against a not fully working system (v0.1.0) to see what items should be added to the final version. This test used the Joe Jacobs case files but mostly focused on what items were logged and not the investigative process. The findings of this test were that the interface of the application needs some work to make it useful to the investigator (showing the time and date, and fixing the table will improve the user experience vastly).

¹see <https://code.google.com/p/LazyLogger/source/list?name=master> for the code at any stage of development.

Another problem found during this test was the change to the autopsy log format which broke several regexes. This problem was eventually tracked down however it bears mention that this had to be fixed by editing the module in such a way that it would no longer work on previous versions. In future this should be done by creating a generic ASK3 logger module which will then either use internal logic to find the version and fix this or call a sub module for a specific version.

5.1.2. Test 2

Test 2 was completed against v0.2.0, this version incorporates all but the final suggestion of test 1. The test uses Joe Jacobs as a case file. The system works well. A small amount of bugs still remain, however they have no affect on the applications results.

Some additional changes will be made based on the results of this test. Graphical User Interface glitch, this is caused by the database table being reloaded in its entirety each time. A better way to do this would be to give the interface access to the SQL result set. This would have to be done as a read-only access as giving the interface the ability to edit the underlying data would be against the nature of contemporaneous logging.

Another change was to fully implement the time stamp feature of the system. This was previously implemented but optional. Each event now requires a date and time to be entered into the constructor. This will ensure that the notes can be ordered by their timestamp.

5.2. Evaluation

As mentioned in the section describing test 1 the application struck a problem when the logging format for autopsy changed. The fix for this was a temporary one, to change the system which pulls information from autopsy. However, this is not a "good" solution. A better way to do this would be to create sub-modules for each version and detect which to load.

The first test showed that the greatest weakness of the application is the interface. The underlying system appears to work well while the interface has numerous glitches. Many of the glitches were resolved for test 2. In this state the application does improve the performance of the examiner as they only need to write their "thoughts" and not their actions.

Additionally, having a screen shot of each stage of the investigation adds contextual information to the report which will aide the reader in understanding the results, as they can see them exactly as the examiner did.

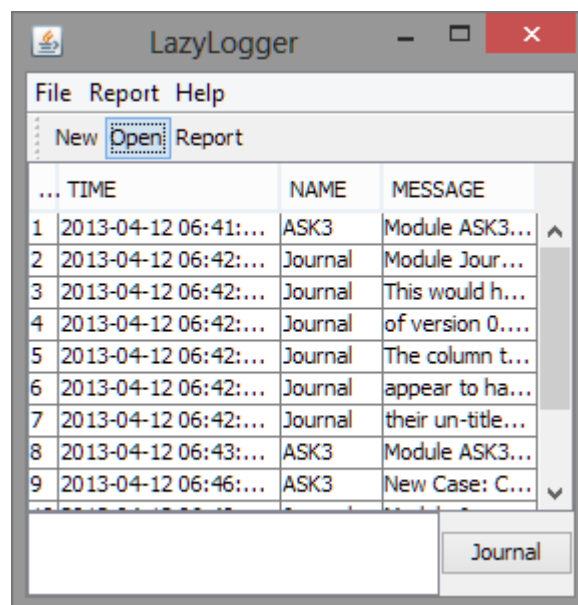


Figure 4: This figure shows the 0.2.0 version of the prototype with some sample data.

6. Future Work

6.1. Database Security

The current database format leaves a lot to be desired in terms of security. It is possible to open the database with a third party tool and edit any entry without being detected. There are multiple ways to prevent this from happening which should be included in any finalised version of this prototype. One way would be the encryption with a master key which would then be required by all of the investigators or people needing access.

6.1.1. Encryption

The Derby database format which is used for the prototype application allows for encryption of the database ([Apache](#)). This would be a powerful tool as the information in the encrypted database is not recoverable or editable by a third party without the secret key.

This adds confidentiality and integrity to the database without reducing the availability to its intended audience. In addition to the database encryption this could also apply to the reports as, if the database is encrypted, editing a report in the hope it will be submitted would also be a valid tactic.

6.1.2. Access Controls

In the current state the database can be edited by any user, although the database specification allows for users ([Apache](#)). This should be implemented so that the user is given the least amount of privileges possible with the database itself doing all of the heavy lifting.

In addition to this the code should be rewritten to prevent the use of SQL Injection based attacks which are currently feasible against the application due to the way SQL is processed. The Prepared-Statement Method appears to be the best method for the prevention of this kind of attack as discussed in [Thomas et al. \(2009\)](#), this is partly implemented in some parts of code. However, other parts still load SQL as a string which can cause these problems.

6.1.3. Hashing

Hashing is a valuable tool when it comes to verifying the integrity of data ([Tsudik, 1992](#)). This has many potential uses in the application. Firstly it could be used to verify each incoming message by creating a checksum for the input data and storing this so that it can be verified later.

Another potential use is to create hashes of the tables when the application is shutting down, then storing these in a new table. When the application is restarted these values will be read and then checked against the current database state, this will show that the database has not been tampered with. Or that the tamperer had enough access to modify both the tables and hashes.

6.2. Core Features

6.2.1. Module Signatures

On start up each module could be checked against a known good signature for that module. This will prevent the loading of malicious modules by only allowing ones which have been prevetted by either being part of the default module set or one added by the examiner.

An alternate method of implementing this system would be to utilise key signing, similar to the

android operating system. Requiring each module to be signed by its author would prove the ownership of that module. This could then be checked against a list of trusted authors.

6.2.2. Report System

The current report system was done quickly with a basic output format. This system should be overhauled into a more modular system which will allow for multiple output formats. This is in keeping with the usability criteria. The current HTML output is good enough for the prototype however it is not for a full tool.

The output formats should at least include a document format such as odf or doc, plain text (with images stored separately), raw SQL and the original HTML. In addition to creating the reports it would be a good idea to add a hash value of the compiled report to the database, this could later be used to ensure the report is unmodified.

6.3. Modules

6.3.1. Forensic Tool Kit (FTK) / Other forensic tools

FTK is one of the most highly regarded digital forensic analysis tools. The reason it was not chosen to be the initial tool logged is because its closed-source nature adds some complexity to the development process. Some research should be done into what kind of files FTK leaves and how they can be interrogated.

There are also other tools which would be worth adding as modules such as Internet Evidence Finder, etc. This could even include the results of tools like regripper or FTK Imager which pull data from the evidence and give reports. The implementation of this could be done through creating a new event and adding the report as an attachment.

6.3.2. Operating System Logs

Some research was initially done into implementing windows logging into the prototype, however, the requirements for this were beyond the scope for a simple test. This will require JNI calls to allow access to the underlying operating system in Windows. Some simple events could be parsed from the windows registry and other files, such as the currently logged on user and possibly the currently running programs.

In a linux environment this could be simplified as the linux OS logs are all text files and are typically

easy to find. Additional information could be found from linux process accounting.

7. Conclusion

The requirements for a forensic audit trail are reproducibility, tamper evidence and the logging of thoughts. Reproducibility is an important part of the forensic process as it allows for third party verification of results by the defence. The logging of thought adds contextual information to the investigation explaining actions and theories as the examiner comes to them.

The third requirement, tamper evidence, has not been included in this prototype. However, a number of ideas on how it could be included are mentioned in future work. This includes verification check-sums (hashes) and a much more verbose logging system that stores both the event and captures the cause of the event.

This article has shown that it is possible to log examiners actions using software. Although the prototype would require a lot of work to bring it to a usable standard. If given further development it would be possible for an application based on this model to log the computer systems activities so that an examiner will only need to focus on logging his or her own thoughts.

This will speed up the investigation time as less time will be spent logging minute details of routine actions and settings of applications as this will be handled by the application. This will mean the examiner spends less time on the mundane task of logging each option of a test.

The accuracy of the reporting will also be enhanced as the computer will never miss anything it is programmed to find. This is important as human error in contemporaneous notes can be a reason for the evidence being dismissed from a court. The application however is incapable of such errors meaning that the chances of this are smaller.

It is also conceivable that this information could be used in a court both as the examiners contemporaneous notes and as part of an expert witness report to state the actions taken to reach a result which the experts opinion is based on as the log will include the evidence recovered by the forensic tool, the experts thoughts and print screened notes that the expert has taken.

The application designed for this article should not be taken as a serious contender for a forensic

tool, for numerous reasons, including the bugs that still exist in it. Instead this should be thought of as a starting point from which another application could be designed, this will ensure that it is designed for "forensic purposes" and not as a proof of concept.

8. References

- ACPO . Good Practice Guide for Computer-Based Electronic Evidence Official release version. n.d. URL: www.7safe.com/.../ACPO_guidelines_computer_evidence.pdf.
- Apache. Derby Reference Manual. The Apache Software Foundation; 2012. URL: <http://db.apache.org/derby/docs/10.9/ref/index.html>.
- Carrier B. Autopsy 3 logging and error checking. <http://wiki.sleuthkit.org/index.php?title=Autopsy.3.Logging.and.Error.Checking>. Accessed: 2013-04-29.
- Carrier B. Open source digital forensics tools: The legal argument. Technical Report; stake; 2002.
- Carrier B. Defining digital forensic examination and analysis tools using abstraction layers. International Journal of digital evidence 2003;1(4):1–12.
- Casey E. Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet. 3rd ed. Academic Press. Academic Press, 2011. URL: <http://books.google.co.uk/books?id=6gCbJ404f-IC>.
- Crown Prosecution Service . Guidance Booklet for Experts, Disclosure: Experts' Evidence, Case Management and Unused Material. 2010. URL: http://www.cps.gov.uk/legal/assets/uploads/files/Guidance_for_Experts_-_2010_edition.pdf.
- Garfinkel S. Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus. Digital Investigation 2012;9, Supplement(0):S80 –9. URL: <http://www.sciencedirect.com/science/article/pii/S1742287612000278>. doi:10.1016/j.diin.2012.05.002; jce:title; The Proceedings of the Twelfth Annual DFRWS Conference; /ce:title; jxocs:full-name; 12th Annual Digital Forensics Research Conference; /xocs:full-name; .
- Müller T, Spreitzenbarth M, Freiling FC. Forensic recovery of scrambled telephones; 2012. URL: <http://loccs.sjtu.edu.cn/typecho/usr/uploads/2012/11/3651077734.pdf>.
- Oracle. Java Platform SE7 API Specification - WatchService. Oracle; 2013. URL: <http://docs.oracle.com/javase/7/docs/api/java/nio/file/WatchService.html>.
- Richard III GG, Roussev V, Marziale L. Forensic discovery auditing of digital evidence containers. Digital Investigation 2007;4(2):88–97.
- Thomas S, Williams L, Xie T. On automated prepared statement generation to remove sql injection vulnerabilities. Information and Software Technology 2009;51(3):589–98.
- Tsudik G. Message authentication with one-way hash functions. ACM SIGCOMM Computer Communication Review 1992;4:29–38. URL: <http://dl.acm.org/citation.cfm?id=141812>.