FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Tommi Järvenpää

RECONSTRUCTING THE PROGRESS OF DIGITAL FORENSIC EVIDENCE EXAMINATION AND ANALYSIS

Master's Thesis
Degree Programme in Computer Science and Engineering
July 2022

Järvenpää T. (2022) Reconstructing the Progress of Digital Forensic Evidence Examination and Analysis. University of Oulu, Degree Programme in Computer Science and Engineering, 71 p.

ABSTRACT

Many commands and tools are used during the evidence examination and analysis stages of digital forensics. If the need to replicate the exact steps from these stages arises later, doing so without proper documentation can be an arduous task. Thus, this thesis focuses on determining how the story of digital forensics progression could be told. To tell the story, this thesis contributes a three-piece system consisting of an updated version of the data collection tool titled Hardtrace, an Application Programming Interface (API) for summarizing and storing collected data to the cloud, and lastly a visualizer application allowing forensic researchers to visually inspect the steps taken during examination and analysis.

To obtain data on digital forensics progression and to test the system, a case study was conducted. The study's participants had to complete a memory forensics Capture the Flag challenge while using Hardtrace. Collected data from each participant was sent to the cloud API. The system's ability to reconstruct and detail the progression of participants work was tested by performing visual and statistical analysis on the summarized data. System performance testing was also conducted.

The results demonstrated that the presented system was able to detail, through visualization, the steps case study participants took while solving the challenge. Statistical summary analysis provided a large quantity of information on how each participant worked, deepening the understanding gained from just visual analysis. Finally, performance analysis showed that the system is able to summarize and visualize data in seconds. Updates to Hardtrace reduced command execution times significantly, nonetheless, the more system calls a tool or command performs, the more execution time overhead is still added by Hardtrace.

Keywords: digital forensic data, information visualization, capture the flag, case study

Järvenpää T. (2022) Digitaaliforensiikan todisteiden tutkimisen ja analyysin rekonstruointi. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 71 s.

TIIVISTELMÄ

Digitaaliforensiikan todisteiden tutkimis- ja analyysivaiheissa käytetään useita komentoja ja työkaluja. Jos nämä vaiheet on myöhemmin toistettava samoin tai tutkijan on kerrottava, miten todisteita käsiteltiin, voi tehtyjen toimenpiteiden muistaminen olla haastavaa ilman kunnollista dokumentaatiota. Täten, tämä tutkielma keskittyy ratkaisemaan miten tutkimis- ja analyysivaiheiden eteneminen voitaisiin kertoa ohjelmallisesti. Tätä tarkoitusta varten tässä tutkielmassa toteutettiin kolmiosainen järjestelmä, jonka osat ovat Hardtrace, ohjelmointirajapinta ja visualisointiohjelma. Hardtrace on jo olemassa oleva datankeräystyökalu, jota tässä työssä päivitetään. Pilveen sijoitetun ohjelmointirajapinnan tehtävä on vastaanottaa ja säilyttää Hardtracen tuottamaa dataa, sekä luoda siitä tiivistelmiä. Visualisointiohjelman avulla forensiikkatutkija pystyy tarkastelemaan visuaalisesti tekemänsä forensiikkatutkimuksen etenemistä.

Toteutetun järjestelmän kykyä rekonstruoida digitaaliforensiikan vaiheiden eteneminen testattiin tapaustutkimuksella. Tutkimuksen osallistujat suorittivat muistiforensiikka Capture the Flag -haasteen ja heidän suorituksista kerättiin dataa Hardtracella. Ohjelmointirajapinnan kerätystä datasta tuottamia tiivistelmiä analysoitiin visuaalisesti ja tilastollisesti.

Tutkielman tulokset näyttivät, että järjestelmä kykeni kertomaan visualisoinnin keinoin, miten tapaustutkimuksen osallistujat selättivät heille annetun haasteen. Osallistujien suoritusten tilastollinen analyysi tuotti paljon lisätietoa osallistujien toiminnasta. Järjestelmän suorituskyvyn havaittiin olevan hyvä dataa tiivistäessä ja visualisoidessa. Hardtraceen tehdyt päivitykset laskivat komentojen ja työkalujen suoritusaikoja huomattavasti, mutta tästä huolimatta mitä enemmän järjestelmäkutsuja komento tai työkalu käyttää, sitä enemmän Hardtrace suoritusaikaa kasvattaa.

Avainsanat: digitaalinen forensiikka-data, informaation visualisointi, capture the flag, tapaustutkimus

TABLE OF CONTENTS

FOREWORD

I would like to thank Rauli Kaksonen (technical supervisor) and Prof. Juha Röning (principal supervisor).

Oulu, July 5th, 2022

Tommi Järvenpää

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
Avg	Average
CLI	Command-Line Interface
CPU	Central Processing Unit
CTF	Capture the Flag
D3	Data-Driven Documents
DB	Database
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
Inode	Index Node
JSON	JavaScript Object Notation
Max	Maximum
Min	Minimum
OS	Operating System
PID	Process Identifier
PNG	Portable Network Graphics
SD	Standard Deviation
SHA	Secure Hashing Algorithm
SVG	Scalable Vector Graphics
Syscall	System Call
URL	Uniform Resource Locator
VA	Visualizer Application
VM	Virtual Machine
XOR	Exclusive Or

1. INTRODUCTION

Digital forensics is a multi-stage process in which digital evidences are identified, collected, examined with digital forensic tools, and analyzed in order to uncover events leading to different incidents, e.g., computer crime [1]. The actions performed during the process need to be documented in the investigation's chain of custody [2]. In the final stage, called presentation, the investigation's findings, supported by the chain of custody, are presented in a court of law [1].

The ever-increasing complexity and volume of evidence data along with the vast variety of forensic tools make maintaining a reliable chain of custody increasingly difficult [3, 4]. As such, there is a need for tools that document how digital evidences were handled [2, 5]. Unfortunately, the availability of such tools is lacking [2].

Now, the evidence handling done during the examination and analysis stages can be automatically recorded with Hardtrace, easing documentation. Hardtrace is a command line tool specifically created to collect information detailing the action done by forensic researchers. Yet, one more problem exists. Hardtrace is only able to collect this information from individual users, whereas forensics researchers are seeking the benefits of using the cloud for data management and result sharing [5]. As such, more benefit could be provided to researchers if data from multiple Hardtrace users could be collected into central storage from where authorized users could access it. Further benefit could be gained if the researchers had access to visual representations of the collected data, supporting the presentation of evidence handling.

With the aforementioned issues in mind, this thesis aims to answer the following questions: How can data detailing the progression of forensic examination and analysis be collected from a group of people? How can the data be transformed into a concise summary? And lastly, how can the steps taken during these stages be visualized?

This thesis contributes a system with three components: Hardtrace, an Application Programming Interface (API) and a Visualizer Application (VA). The existing Hardtrace implementation is updated to submit collected data to the API that is containerized with Docker and hosted in the cloud. The API converts Hardtrace data into two types of machine-readable summaries, one supporting further analysis, and one for easy visualization. Using the popular Data-Driven Documents (D3) visualization framework [6], the VA illustrates the summaries as directed graphs displaying the used commands, tools and files, as well as, their connections.

Capture the Flags (CTF) pit participants against computer security related challenges that often require the use of digital forensic tools and the further analysis of extracted information. As such, completing a CTF challenge could be considered as performing the examination and analysis stages of digital forensics. Since existing reconstruction systems for digital forensics, e.g. [7, 8, 9, 10], focus on the incidents and related events instead of the actual digital forensics process, the implemented three-piece system is compared with systems previously proposed for the purpose of collecting data on and for the visualization of CTF progression. The system is validated through a case study in which Hardtrace data is first collected from participants completing a memory forensics CTF challenge. Visual and statistical analysis is then performed on the data, demonstrating the types of information that can be gathered from Hardtrace data. The performance of the system is also tested to determine how it affects the speed of the digital forensics process.

2. BACKGROUND

This chapter provides background information on topics relevant to this thesis, starting from the digital forensics process, continuing on to how digital forensic data can be collected, then followed by CTFs, information visualization and cloud computing.

2.1. Digital Forensics Process

Digital forensics employ scientific principles to investigate artifacts that pertain to incidents, such as computer crime, obtained from different sources in order to reconstruct and to gain knowledge on the event chains that resulted in the creation of these artifacts. The event sequences are uncovered utilizing a multi-stage process that includes the following individual stages: [1]

1. **Identification**: At this stage, sources of information, relevant to the current investigation, are identified. The sources can include different devices capable of storing digital information.

2. **Collection and preservation**: Once relevant data sources are identified, a copy of their contents is acquired and preserved for later use. The types of obtained data can vary widely.

3. **Examination**: Examination is the process of making digital evidence available for analysis from the previously collected data. Relevant evidences are extracted with the use of forensic tools. Since tools are typically highly specialized, the use of multiple tools is often required.

4. **Analysis**: At this stage, the extracted evidences are analyzed to determine the event sequence that leads to the incident under investigation. Analysis aims to answer the fundamental questions of an investigation, such as who, why and how.

5. **Documentation**: This stage entails the thorough documentation of the actions taken during the previous stages. Good documentation is important, especially if digital forensics were performed as a part of an actual criminal investigation.

6. **Presentation**: Finally, the gathered evidence and other findings are presented to relevant parties.

2.2. Collecting Digital Forensic Data

Many types of data are available for collection and use in digital forensics. Whether the data are synthetic or captured from the real world, datasets include, among others malware, collections of files, memory dumps, computer disk images, and network traffic [11].

As this thesis is centered around the recreation of the examination and analysis digital forensics stages, this section focuses on the collection of two specific types

of data. These being data detailing how files have been modified, and what commands the user has entered into their computer's Command-Line Interface (CLI). The forensic tool Hardtrace is able to collect data on both of the aforementioned actions.

2.2.1. File Modification

Multiple options exist for file modification monitoring and data collection. E.g., file signatures can be calculated and compared, a computer systems input and output operations or system calls can be tracked.

Signature calculation is a straight forward way to determine if a file has been manipulated. Hashing algorithms are commonly used for the calculations. When a signature has been created for a file, it can then be recalculated at a later time and the new and old signature can be compared. If they are different, the file has been modified. This approach was used, e.g., by Kim and Spafford in their integrity checker tool Tripwire [12].

Keeping track of a computer system's input and output operations for file usage monitoring was used, e.g., by Mehnaz and Bertino in their approach titled Ghostbuster [13]. They used the blktrace Linux Operating System (OS) kernel module to extract information on block level input and output. The collected data was then used to create profiles detailing users' normal file access behaviors for the purpose of anomaly detection.

By using a mountable file system that allows additions to be easily made to the underlying OS without modifying kernel level functionalities, a software sensor can be added to audit the file access taking place, as done in [14]. However, the functioning of the OS kernel can also be changed to allow file access monitoring through the modification or interposing of system calls [15, 16]. As an example, the system call for opening files can be altered to record relevant information to a log file.

Monitoring file operations through system calls can also be done using tools like ptrace [17], which is a Linux kernel system call that allows one process to observe and control the execution of another process.

2.2.2. Command Line Input and Output

When using a computer system's CLI, i.e. the shell, users specify commands they want to execute. These commands detail the program or tool to be used and any possible additional parameters. As an example, the command "*volatility -f MemoryDump.raw imageinfo*" calls for the use of the memory forensics tool Volatility[1]. The command's additional parameters present the target file and the Volatility plugin to use.

Collecting data on such actions can provide detailed information on what the users did with the CLI. Shell usage can be recorded, e.g. in text, Graphics Interchange Format, or even in Hypertext Markup Language (HTML) using tools like ttyStudio, Script or TermRecord [18].

[1] https://github.com/volatilityfoundation/volatility

The widely used Bash shell [19] keeps a history of commands executed by the user. These textual logs have been used, e.g., by Hance and Straub [20] to calculate anomaly scores for user actions with the isolation forest algorithm for the purpose of intrusion detection. Similarly, shell histories were also collected in [21], however, from a different shell, for anomaly detection with the instance-based learning model. Weiss et al. [22] used EDURange, a public cloud-based cybersecurity exercise framework [23], to collect Bash histories which were then used to create paths detailing the chains and cycles of used commands.

A different method for CLI command collection was used by Lee et al. [24]. Employing the forensics tool called Bro[2], they extracted users' commands from telnet sessions.

Noting the fact that most shell history logs only include the input of users, Mirkovic et al. [25] created a system titled ACSLE, which employs a modified version of ttylog to capture both the shell input and output. In addition, their system captured the username, shell identifier, current working directory, and time of input. ACSLE's monitoring component was used in the DeterLab testbed[3], from which the monitor forwarded collected data to centralized storage.

Opting again to only collect CLI inputs, Švábenskỳ et al. [26] utilized sandboxed environments, each with an attacker and target hosts. The network hosts recorded executed commands, along with the same metadata from [25] with the addition of the Internet Protocol (IP) address, using the Syslog protocol which then automatically forwarded them to a central storage from where the data could be accessed in real-time.

In [22, 25, 26] forensic data were collected from students completing different CTF challenges in the context of aiding tutors assess the students' work.

2.2.3. Hardtrace

Hardtrace is an open source CLI-based data collection tool written in the Go programming language and developed by Rauli Kaksonen and Petri Partanen of the Oulu University Secure Programming Group. Hardtrace is available on GitLab[4]. According to Kaksonen, the creation of Hardtrace began when he noticed a lack of tools that would allow forensic researchers to determine, at a later time, how different files were created and modified. Due to the reasoning for its creation, Hardtrace records information on processes, system calls (syscall) and accessed files. From syscalls, Hardtrace also extracts the commands its user enters into the CLI.

To be more precise, Hardtrace collects the following information on accessed files: Full path, file system Index Node (inode), whether the file is a pipe, and the Secure Hashing Algorithm (SHA) 256 hash, i.e. the signature. For processes, Hardtrace records the Process Identifier (PID) and parent identifier. Finally, for syscalls Hardtrace records their name, i.e. type (read, write, etc.), what files they access, calling process, command and parameters. These data are stored to a local SQLite Database (DB) using the GORM[5] Go library. Figure 1 elucidates Hardtrace's DB structure.

[2] https://zeek.org/
[3] https://deter-project.org/about_deterlab
[4] https://gitlab.com/CinCan/hardtrace
[5] https://github.com/go-gorm/gorm

To collect the data, Hardtrace utilizes ptrace. When Hardtrace is started by, e.g., entering the command "*./hardtrace -session test bash*", Hardtrace starts tracking a new Bash shell session and any child processes it spawns. Relevant information is then extracted mainly by reading the register values of the encountered processes with ptrace.

Using ptrace, calculating SHA-256 hashes for files and writing to the local DB add overhead to program execution. If not properly addressed, the execution times of tracked commands are increased. Significant increases in the execution time can possibly occur, e.g., when a tool performs a very large amount of syscalls that Hardtrace has to store into its DB. To tackle this issue, Hardtrace performs the overhead adding actions in a separate goroutine. The goroutines are lightweight threads which enable the aforementioned actions to be performed during command execution [27].

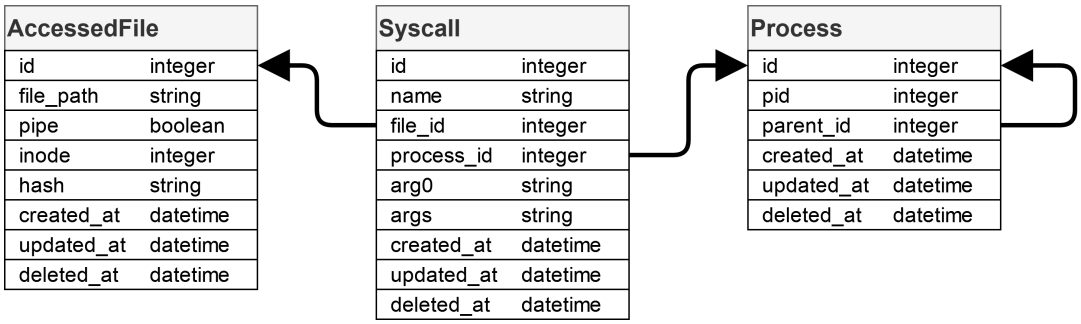


Figure 1. Structure of Hardtrace's DB.

2.2.4. Related Operating System Concepts

The aforementioned systems for data collection touch on many OS concepts, such as processes, files, and the CLI. This section provides relevant information on these subjects.

OSs provide users with different options for interaction. Examples include the CLI and the Graphical User Interfaces (GUI). The command interpreters are also called shells. These shells are used to obtain and execute textual user-specified commands. The code required to execute the commands can either be stored in the shell itself or invoked from a specific file. [28 pp. 56, 58-60]

The OS enables users to execute programs. Once an executable program is loaded into the computer's main memory, its execution begins and it becomes a process. Through processes, users are able to request actions from the OS. [28 pp. 24-25, 105-106]

Processes are also able to create new processes creating a parent-child relationship between the creator and the newly created process. The creator process is called the parent while the created one is the child. To differentiate between all the processes, the OS assigns a unique PID to processes. The PID is typically an integer. [28 pp. 116-117]

For some tasks, processes, e.g., the parent and the child, need to be able to communicate with each other. For these situations, the OS provides ways for

interprocess communication. One example of these mechanisms is the pipe. Pipes can be ordinary or named. Ordinary pipes enable the half-duplex communication between two process. Named pipes, on the other hand, allow for the bidirectional communication of multiple processes. [28 pp. 142-145]

OSs represent each process with a process control block. Its content include the process' state, program counter, Central Processing Unit (CPU) scheduling and memory-management information, as well as, CPU registers. The registers include accumulators, stack pointers, index registers, and general-purpose registers which can store, e.g., data. [28 pp. 107-108]

With processes, users are able to, e.g., manipulate files. However, processes require the aid of the OS's syscalls for these actions. Syscalls offer the means for processes to request the OS to perform certain actions that only the OS is able to complete. Uses for syscalls include process control, file and device management, information maintenance, communication, and protection. [28 pp. 23, 62-67, 506]

Through file management syscalls processes are able to perform actions related to files. A file is the logical storage unit abstracted by the OS from the physical properties of its secondary storage devices. Each file is a named collection of information recorded on non-volatile memory. Typically, files represent programs and data with the meaning of its contents defined by the files creator and users. All files also have attributes. While different OSs use different attributes, commonly used file attributes include, among others: [28 pp. 26-27, 503-504]

- Human-readable symbolic name.
- Type of the file, usually included in the name by adding a dot and the file extension.
- A unique identifier, i.e. the non-human-readable name of the file.
- Pointer to the location of the file on a storage device.

Additionally, files have a file-control block, also called inode in UNIX filesystems. The inode is a data structure that stores metadata about files, including ownership, permissions and the location of the file's contents. [28 pp. 493, 545]

2.3. Capture the Flag

Capture the Flag [29, 30] is a kind of competition, dating back to at least 1996, where participants test and hone their computer related skills. Typically, the goal of these competitions is to find flags by solving challenges related to, e.g., web technologies, cryptography or digital forensics.

Flags in CTFs are a proof of completion, showing that a competitor solved a challenge successfully. Commonly, these flags are small strings of text. Many CTFs allow flags to be submitted to a scoring engine or a website, rewarding the submitter with points. More difficult challenges reward more points, and the person or team with the most points wins the competition.

The two main types of CTFs are Attack and Defence, and Jeopardy. From these, Jeopardy is more common and consists of one or more individual challenges. In

Attack and Defence-style CTFs, teams attempt to exploit the software provided for other teams and to defend their own.

In addition to competitions and being a pastime activity, CTFs are also used in cybersecurity training [31]. Such training sessions have tutors, last around one to two hours, and are organized for multiple participants. During training, tutors observe the progress of each participant and offer help on request or if they notice someone lacking behind. Once training concludes, tutors present the correct solutions. [32]

When CTFs are used for training purposes, it is important that the trainees get the support they need. Being able to provide meaningful insight into training progression is needed to maximize learning [31]. However, the teaching staff have only limited insight into how participants are and were faring [32]. To help tutors discern when someone is in need of guidance and to help assess the work of trainees, data collection systems like the previously mentioned [22, 25, 26] have been proposed. Proposed systems have even utilized information visualization to further assist the tutors in their tasks and to provide additional, easy to understand, feedback to trainees [22, 25, 31, 32].

CTF progression monitoring tools document the actions performed by the trainees, and allow for the reconstruction of their work. Due to the nature of CTFs, this can be considered as documenting how the examination and analysis stages of digital forensics were performed. If further data collection capabilities are added, such system seem suitable for documenting actual digital forensics. Furthermore, the use of information visualization, like in CTF monitoring, could make understanding and presenting the progress of digital forensics evidence examination and analysis easier.

2.4. Information Visualization

Information visualization is a field of study that emerged from existing fields, such as computer science, graphics, and human computer interaction [33]. Information visualization itself refers to many things [34]. For one, it is the use of interactive graphical representations of information generated with computers. It is also the process of producing these visual representations. The goal of information visualization is to help users in analyzing and gaining a better understanding on data through visual exploration [35].

Research related to information visualization can be divided into four main categories: empirical methodologies, user interactions, frameworks, and applications. From these, empirical methodologies are focused on providing theoretical foundations for large numbers of different applications. User interaction studies interaction techniques and how users interact with visualization tools. The focus of visualization frameworks is on designing generic visualizations for widespread deployments or for sets of applications in a specific domain. The last category, visualization applications, is used when the design of visualizations depends heavily on the data to be visualized and requires more specialized tools. [35]

Actual visualizations in information visualization are created utilizing an information visualization pipeline. The resulting visualizations are often represented through a dashboard, i.e., the user interface of the visualization which provides users the means of interacting with the visualizations.

2.4.1. Information Visualization Pipeline

Information visualization commonly handles abstract, nonspatial data [34]. As such, to achieve the goal of information visualization, the transformation and processing of data into more meaningful and intuitive representations are of importance. These tasks can be addressed by utilizing an information visualization pipeline [36], illustrated in Figure 2. The pipeline includes three states for processing and four for data. Data states are: raw, processed, visual symbols, and visualization. Processing states and their goal are as follows:

- Preprocessing: Transformation of raw data into a suitable format by cleaning, organizing and/or aggregating the data.
- Visual Transformation: Converting processed data into visual symbols, such as dots, and line charts. The symbols depend on the used visualization technique.
- Visual Mapping: Mapping visual symbols to visual channels including among others, color and size, and finally combining the symbols into the final visualization format, such as an image.

Figure 2. Information visualization pipeline.

Many visual symbols and visualization techniques are available for use in information visualization. However, as the goal is to maximize the understandability of the final visual representation, the most appropriate techniques for the use case should be selected. Commonly used visualization techniques include, among others, scatter plots, histograms, radial charts, trajectories, heat maps, and node-link diagrams. [35, 37, 38].

2.4.2. Graph Visualization

From the point of view of this thesis, node-link diagrams, and more broadly, techniques from graph visualization are of special interest. Graph visualization produces illustrations in accordance with the rules from graph theory, a field of mathematical study [39]. A graph consists of nodes (vertices) and connecting edges (links). Edges represent the relations between nodes. Graphs provide powerful abstractions of data [35] and common use cases for graph visualization include, among others, the visualization of computer file hierarchies, website maps, and evolutionary trees. [40].

The suitability of graph visualization, based on data, can be determined with the question posed by Herman et al. *"Are there inherent relations among the data elements?"*. If the answer is no, the data is unstructured and thus graph visualization is most likely not the best option. Otherwise, representing the data with nodes and relations with edges is possible. [40]

Based on the relations portrayed by the edges of a graph, weight and/or direction can be added to the edges. Graphs with directional edges are called directed graphs. When edges have weights, the graphs are weighted. Furthermore, graphs can be divided into two categories: static and dynamic [41]. Static visualizations can be node-link or matrix based. Dynamic graphs can be animated diagrams or static charts that are based on a timeline, and unlike static graphs, dynamic can change over time.

The positioning of nodes in a graph is based on layouts. Tree and spring are examples of such layouts. The tree layout positions child nodes below their ancestors, using algorithms such as Reingold or Tilford. The spring layout, also known as force-directed, models nodes as physical bodies and the connecting edges as springs. Positioning of nodes is then produces by applying force between the bodies. [40]

Graph visualization is not without its problems. Key issues include the size of graphs and information overload [40]. When graphs grow larger, the performance of layout algorithms can degrade. Additionally, discerning between edges and nodes also becomes increasingly difficult as their number grows, and they begin to overlap.

2.4.3. Dashboards and Interaction

In visualization, dashboards are interfaces resembling a car's dashboard that organize and present information in an easy to read manner [42], consolidating important data into one place that is often accessed with a browser [43]. However, there is no single accepted definition for dashboards [44]. Dashboards are widely used, e.g., for monitoring and analysing business processes [45]. Similarly, to visualizations, the selection of right visualization techniques is important for dashboard effectiveness. Dashboards can be divided into two genres [44]: visual and functional. Visual dashboards represent data with simple visualizations laid out as individual tiles. Functional dashboards, on the other hand, are interactive displays enabling the real-time monitoring of dynamically updating data.

When considering interactions for visualizations and dashboards, it is good to keep in mind Shneiderman's visual information seeking mantra *"Overview first, zoom and filter, then details-on-demand"* [46]. Shneiderman summarizes the mantra's approach as seven tasks.

- Providing an overview of the entire data set.
- Zooming in on interesting items.
- Filtering out irrelevant data.
- Obtaining more detailed information on data once they are selected.
- Viewing relations among the data.
- Keeping a history of actions taken by the user and supporting undo and redo.
- Allowing the extraction of data subsets.

Interaction strategies employed by dashboards and visualizations include, among others, zoom and pan, focus+context, selection, filtering, and rearranging [33, 40].

Zooming can be geometric or semantic. Geometric zoom merely increases the size of the visualization. The semantic zoom increases the amount of displayed information on the area the user zooms in on. Focus+context interaction technique allows users to focus on the area of interest without losing context. This can be done, e.g., with fisheye distortion, which enlarges the area of interest but reduces the amount of detail in other areas. This, however, does not replace zoom and pan. Dashboards can enable data filtering with, e.g., range sliders and checkboxes [47]. For individual visualizations, rearranging can be, e.g., dragging the nodes of a graph into more suitable locations, and for dashboards, rearranging means moving around the displayed visualizations and other tiles. The use of such interaction strategies can improve the scalability of visualizations, reducing the negative effects of illustrating large quantities of data.

2.4.4. Data-Driven Documents

Data-Driven Documents is a visualization framework for web based visualization with a novel representation-transparent approach, created by Bostock et al. [6]. D3 promises good performance and compatibility with different browsers, as well as, easy debugging of the visualizations. With D3, visualizations are created by manipulating a web page's HTML contents. While the D3 visualizations can be created purely with use of HTML and Cascading Style Sheets, more complex visualizations can be created by utilizing Scalable Vector Graphics (SVG). As an example, a visualization can be created with D3 by first querying for a specific div HTML element, appending an SVG container to it, binding the input data to the SVG, and by lastly representing the data with SVG elements, such as circles, that correspond to wanted visual symbols. By allowing the mapping of arbitrary data to elements, D3 is not restricted to only simple visualizations.

D3 provides users with helper modules that simplify common visualization tasks. Examples of the modules include *d3.svg*, Scales and Layout. *d3.svg* provides different shapes, i.e. visual symbols. The Scales module simplifies visual encoding and it includes, e.g., a color scale, making the coloring of visual symbols an easier task. Layout, on the other hand, provides different layouts for visualizations, from which the most suitable can be selected. For the creation of node-link diagrams, Layout provides the force[6] layout that is used for the creation of force-directed graphs. In addition to helper modules, D3 includes event handlers that enable interaction by responding to user input. Animated visualizations are also supported by D3.

2.4.5. Capture the Flag Visualization

Information visualization has been used to visualize how participants progress in CTF challenges. Existing examples visualize the participants' steps after the challenge is completed, as well as during challenge completion.

Utilizing an information visualization pipeline where the raw data are logs of Bash histories, collected from CTF participants, Weiss et al. [22] created paths from the

[6] https://github.com/d3/d3-force

commands executed by the participants. The paths were visualized, once the CTF had concluded, as directed graphs in which the nodes represented the executed commands. Edges were used to represent command chains.

ACSLE, developed by Mirkovic et al. in [25], compared participants' shell inputs and outputs with a set of exercise-specific milestones. The system's Pattern-Analyzer visualized failed attempts at reaching the milestones using directed tree graphs. These visualizations detailed frequently used commands and additional parameters.

Other than CLI interaction data, visualizations for multi-level CTFs have also used player related events, such as staring, ending or skipping a level, flag submission and taking a hint as their source of data.

Using a CTF game's interface, Ošlejšek et al. [31] collected the aforementioned player events. Their system provided intuitive and easy to use post-game feedback for participants in the form of two interactive visualizations, created with *D3.js*. A clustering visualization employing bar chars combined with scatter plots illustrated players overall and level-specific scores. More time oriented visualization was provided by the timeline view which showed the evolution of players scores and provided information related to score affecting events.

The single-page Progress Visualization Tool by Burská et al. [32] developed with *D3.js*, illustrated participants' progress as they advanced through the CTF. Their visualization included four sections: Timeline, trainees, game level occupancy, and a detailed timeline. The timeline view displayed how much time was left for the challenge. The interactive trainees view showed the status of participants. Clicking on a participant's name or avatar made the detailed timeline display the progress and events related to that specific person. Finally, the game level occupancy view detailed the current number of players in each level.

2.5. Cloud Computing

Cloud computing is a popular model for computation that supports the processing of large quantities of data by utilizing clusters of commodity computers [48]. Cloud computing is defined as a model for enabling ubiquitous, on-demand network access to a shared pool of configurable computational resources (networks, servers, storage, etc.) [49]. Typically, these resources are located in data centers. The amount of available resources appears to consumers as virtually unlimited [50]. Access to the resources is commonly offered though a pay-as-you-go model, eliminating the upfront costs of buying computing infrastructure. However, what resources cloud computing provides to consumers depends on the used service model. The following explains the three most common service models [49] and provides an example service provider for each.

- Software-as-a-Service: Provides software that runs on the cloud infrastructure. Microsoft Office 365[7].
- Platform-as-a-Service: Provides the capability to deploy consumer created applications onto the cloud infrastructure. Google Cloud Run[8].

[7] https://www.microsoft.com/microsoft-365
[8] https://cloud.google.com/run

- Infrastructure-as-a-Service: Provides provisionable computational resources onto which the consumer is able to deploy arbitrary software. Amazon Elastic Compute Cloud[9].

Deployment models for the cloud include public, private, community and hybrid [49]. Public clouds provide their resources to the general public. Private and community clouds provide exclusive resource access to specific organizations and communities. Hybrid clouds combine aspects from the other deployment models.

Cloud computing is made possible by virtualization, as it allows computation resources to be provided for customers over the Internet [28 p. 716]. Serving as the backbone for resource provisioning, virtualization allows a single server to host multiple virtual environments simultaneously, making resource utilization more efficient. Compared with the traditional Virtual Machines (VM), container virtualization has become an increasingly desirable virtualization technique for cloud computing [51]. Containers include everything, from code to execution environment, an application requires to function [52]. A popular approach to container virtualization is the use of Docker[10] containers.

[9] https://aws.amazon.com/ec2/
[10] https://www.docker.com/

3. DESIGN AND IMPLEMENTATION

This chapter presents the digital forensics examination and analysis reconstruction system designed and implemented in this thesis. The system consists of three components: API, VA, and the updated version of Hardtrace. The system's basic idea is that Hardtrace uploads collected data, detailing the progress of the two digital forensics stages, to the API that is hosted in the cloud. No other data, other than what Hardtrace collects, is recorded from users. The API produces summaries from the data that reconstruct the order in which commands, tools and files were used, as well as, their connections. Graph data version of summaries are visualized by the VA, producing illustrations of the reconstructions. This idea is elucidated in Figure 3. Based on this concept, different requirements were set for each system component, serving as the basis for their design and implementation.

The system components are available on GitLab[11] [12] [13].

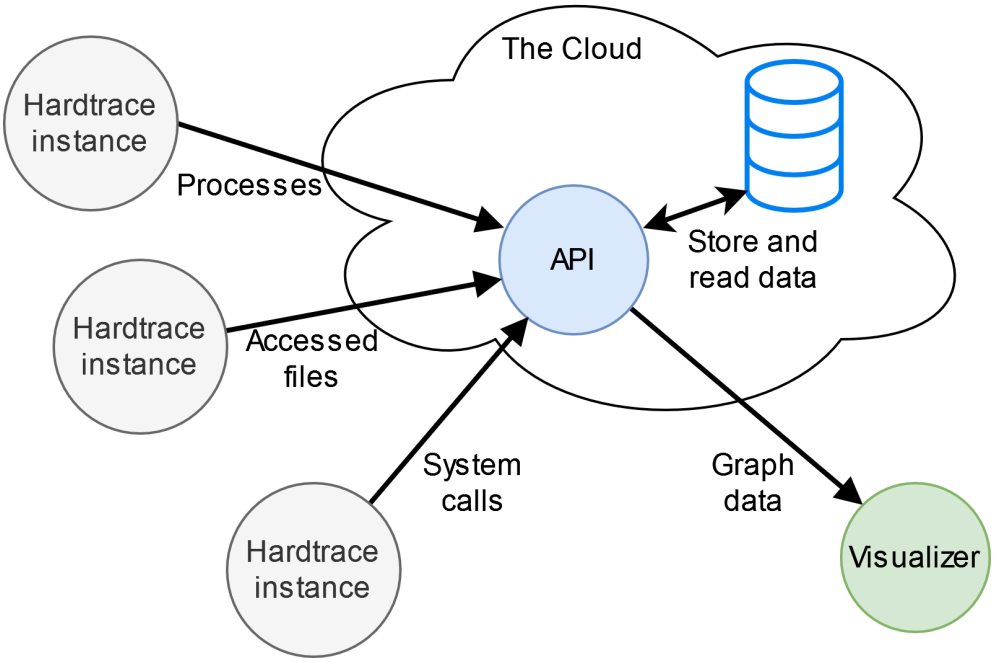


Figure 3. The system's concept.

3.1. Requirements

The following subsections present the requirements set for each of the three system components.

[11] https://gitlab.com/CinCan/hardtrace
[12] https://gitlab.com/Zabrakk/hardtrace-server
[13] https://gitlab.com/Zabrakk/hardtrace-visualizer

3.1.1. Application Programming Interface

Requirements set for the API:

R1 Receive and store session details and data from authorized Hardtrace users.

R2 Create authorization details for Hardtrace users and refer to users using a pseudonym.

R3 Provide summaries on Hardtrace user sessions and specific files for users with specific authorization details. The summaries should reconstruct the order of actions performed while using Hardtrace, and be easy to visualize.

R4 Containerized and deployable into the cloud.

R5 When deployed, data transmitted to and from the API should be secured.

3.1.2. Hardtrace

Requirements set for the updated version of Hardtrace:

R1 Record only the unique syscalls of each process. In some cases, the old Hardtrace version saves a large number of syscalls to its local DB, possible affecting performance.

R2 Register new and old sessions to the API.

R3 Upload accessed files, processes and syscalls from the local DB to the cloud in JavaScript Object Notation (JSON) format. Data from sessions conducted offline should also be uploaded once a connection to the cloud is established.

R4 Provide authorization details to the API when uploading data.

3.1.3. Visualizer Application

Requirements set for the visualizer application:

R1 Clear and understandable visualization of Hardtrace data.

R2 Allow users to interact with the actual visualization.

R3 Include a dashboard allowing authorized users to search for and select specific Hardtrace user sessions and files. Visualized data should be requested from API based on user input.

R4 Provide authorization details when requesting data from the API.

R5 Browser-based.

3.2. Application Programming Interface

Based on the API's requirements, the resource table elucidated in Table 1 was designed. Each one of the resources are accessed through Hypertext Transfer Protocol (HTTP) request methods, such as GET, POST and PUT. All resources, excluding Status and GetCreds, also require the inclusion of authorization details along the HTTP requests. This design helps fulfil the API's requirements **R1**, **R2** and **R3**.

Table 1. API resource table

Name	**Location**	**Description**
Status	/api/status/	Helps determine if the API is online
GetCreds	/api/create-user/	Generates new Hardtrace authorization details, returns as JSON
Sessions	/api/users/<user>/sessions/	Hardtrace session registration
AccessedFiles	/api/users/<user>/sessions/<session>/accessed-files/	Receives accessed files and stores them into the DB
Processes	/api/users/<user>/sessions/<session>/processes/	Receives processes and stores them into the DB
Syscalls	/api/users/<user>/sessions/<session>/syscalls/	Receives syscalls and stores them into the DB
UserInfo	/api/analysis/user-info/	Returns a JSON containing user pseudonyms and session names
FileInfo	/api/analysis/file-info/	Returns a JSON containing accessed files
Summary	/api/analysis/summary/	Creates and returns session and file summaries as JSON

In order to make the visualization of created summaries easier (**R3**), the API was designed to be able to provide two different versions of the same summary. The first version is geared towards being easily understandable by humans and simpler to perform statistical analysis on, and in the second version, the summary is converted to easily visualizable graph data.

To support the storage of data from multiple Hardtrace users, a DB mimicking Hardtrace's local DB was designed for the API. Additions include, among others, a table for users and sessions. The API's DB design is elucidated in Figure 4.

The API was implemented using the Python programming language. Its main dependencies include Flask[14], a lightweight web application framework, and two of its extensions. These being Flask-RESTful[15] and Flask-SQLAlchemy[16]. Additionally, jsonschema[17] was used to validate JSONs received by the API.

In the actual implementation, the API's requirement **R1** is fulfilled by the Sessions, AccessedFiles, Processes and Syscalls resources. Using the capabilities provided by

[14] https://github.com/pallets/flask
[15] https://github.com/flask-restful/flask-restful
[16] https://github.com/pallets/flask-sqlalchemy
[17] https://github.com/Julian/jsonschema

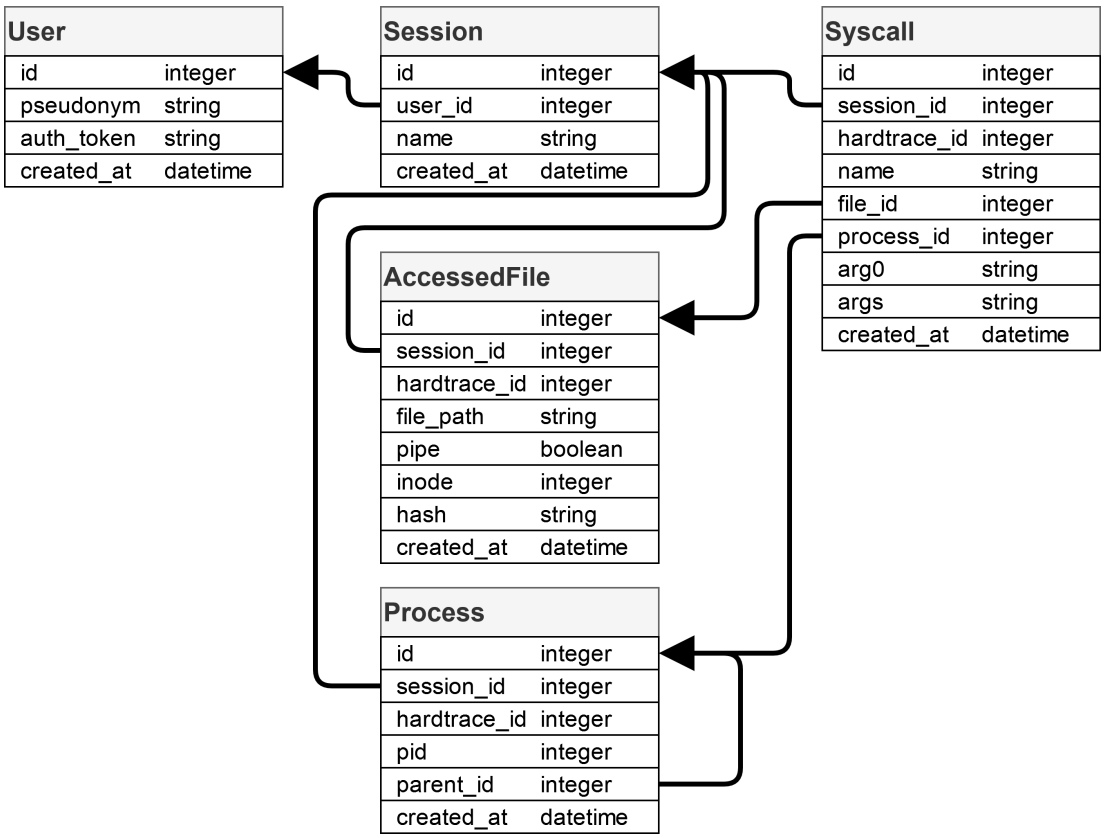


Figure 4. Cloud database structure.

Flask-SQLAlchemy, these resources store the data sent by Hardtrace into the API's DB. To ensure that only the data sent by authorized users are saved, users are expected to provide their authorization token in the headers of the HTTP request.

As per requirement **R2**, the GetCreds resource generates pseudonyms and authorization tokens for Hardtrace users. A simple naming convention consisting of the word "user" followed by a number is used for pseudonyms. The tokens, on the other hand, are generated with version 4 Universally Unique Identifier [53].

The creation of summaries, stated in requirement **R3**, is implemented in the Summary API resource. The user requesting summary creation defines what data is used in the process by providing the information along with the request. As an example, the JSON shown in Figure 5(a) can be used to request the summarization of Hardtrace sessions. A graph data summary detailing the creation of two specific files can be requested, e.g., with the JSON in Figure 5(b). The details needed to construct the JSONs can be obtained from the UserInfo and FileInfo resources. The authorization token must be provided when accessing these resources. These resources use a set of authorization details stored in environment variables. This means that the credentials required to access the stored data are separate from the ones Hardtrace uses. Thus, only selected users can access them.

The summaries created by the API are actually collections of summarized processes. By iterating over syscalls and checking their PID, the command entered into the CLI by Hardtrace's user, as well as, accessed files, are connected to a specific process. Accessed files are marked as either new or old. New files have not appeared previously,

whereas old files have already been seen during the summary creation. In the case that a summary for specific files was requested, the processes that have no relation to these files are removed. This way of summarizing data results into the more human-readable version of the summary. An example session summary is provided in Appendix 1 Figure 21 and a file summary in Figure 22.

The aforementioned session and file summaries serve as the basis when the API creates graph data versions of summaries. In the graph data, files are turned into nodes and processes into edges. This creates a directed graph where the commands run by users connect affected files to each other. Nodes with no incoming edges are marked as source nodes. In graphs that are based on file summaries, the files of interest are also marked. Node marking is done to support highlighting in visualization. To reduce clutter and visual occlusion, the API also combines edges that represent very similar actions performed by different users. Figure 23 in Appendix 2 provides example session graph data. Figure 24 illustrates file summary graph data.

Further reduction of clutter in the graph data is also supported by the Summary resource, as it allows the user to ignore files based on, e.g., file extension or hash.

```
{
    "network": false,
    "includes": "users",
    "summarize": [
        {
            "user": "user1",
            "session": "test"
        },
        {
            "user": "user2",
            "session": "test_session"
        }
    ]
}
```

```
{
    "network": true,
    "includes": "files",
    "summarize": [
        {
            "filename": "file1.txt",
            "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
        },
        {
            "filename": "file1-modified.txt",
            "hash": "b22b009134622b6508d756f1062455d71a7026594eacb0badf81f4f677929ebe"
        }
    ]
}
```


(a) Session. (b) File.

Figure 5. Example session summary and file graph data request JSONs.

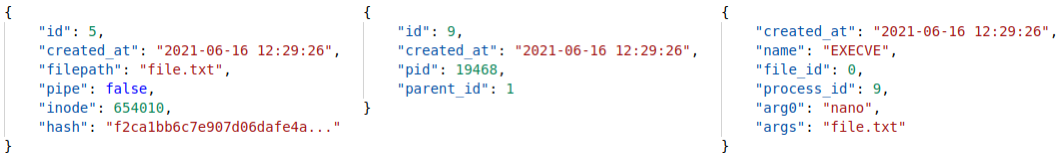
To comply with requirement **R4**, a Dockerfile was created for the API. This allows the API to be containerized using Docker. The containerization enables the API to be easily deployed to the cloud. When the API is deployed, request to and from the API are sent over Hypertext Transfer Protocol Secure (HTTPS). This way data are not transmitted as plain text over the Internet, fulfilling requirement **R5**.

3.3. Updated Hardtrace

With the requirements set for the updated version of Hardtrace in mind, the following design choices were made:

- To ensure that only unique syscalls are recorded, a hash is calculated for each syscall. The hashes are then kept in memory and used as reference when determining whether or not a syscall is unique.
- Hardtrace should work offline and online. Thus, if Hardtrace is not connected to the API, data is only stored to the local DB. Once a connection to the API is established, Hardtrace uploads the data from offline sessions and then the data from the current session.

- When sending data on accessed files, processes and syscalls to the API, the JSON formats shown in Figure 6 are used.
- Authorization details obtained from the API are stored locally and provided to the API along HTTP requests.



(a) Accessed file. (b) Process. (c) Syscall.

Figure 6. Example JSONs sent to the cloud by Hardtrace.

The hashes used to fulfill requirement **R1** are obtained by performing the following. The information of each syscall is concatenated into a single string of text. SHA-256 is then used to convert the strings into hashes. The list of unique hashes is emptied every time Hardtrace starts tracking a new process.

The optional command line parameter *-server* was added to the updated version of Hardtrace. With it, the user can provide the Uniform Resource Locator (URL) of the API. When this parameter is used, Hardtrace connects to the API, obtains the authorization details, registers old and the current sessions, and finally starts the tracking process. During a Hardtrace session, accessed files, processes and syscalls are sent to the API in a separate goroutine in order to reduce the created overhead. With these modifications, Hardtrace complies with requirements **R2** and **R3**. When sending data to the API, Hardtrace includes its authorization token in the request's header and the pseudonym in the URL (**R4**).

3.4. Visualizer Application

A concept image of the VA's design is illustrated in Figure 7. As shown in the image, directed graphs were selected as the visualization technique for requirement **R1**. Since relations between files and processes are clearly present in Hardtrace data, based on the question posed by Herman et al. in [40], graph visualization is a suitable choice. Requirements **R2**, **R3** and **R4** are also supported by this design.

Implementation of the VA was done using Python, HTML, Cascading Style Sheets and JavaScript. Its main dependencies include Flask, jQuery[18], saveSvgAsPng[19], *D3.js*[20] and its modules: d3-force, d3-scale[21], d3-scale-chromatic[22]. Flask makes the application accessible via a web browser, jQuery allows the application to fetch data from the API, and D3 performs the visualization. With such an implementation, the application is browser-based (**R5**).

[18] https://jquery.com/
[19] https://github.com/exupero/saveSvgAsPng
[20] https://d3js.org/
[21] https://github.com/d3/d3-scale
[22] https://github.com/d3/d3-scale-chromatic

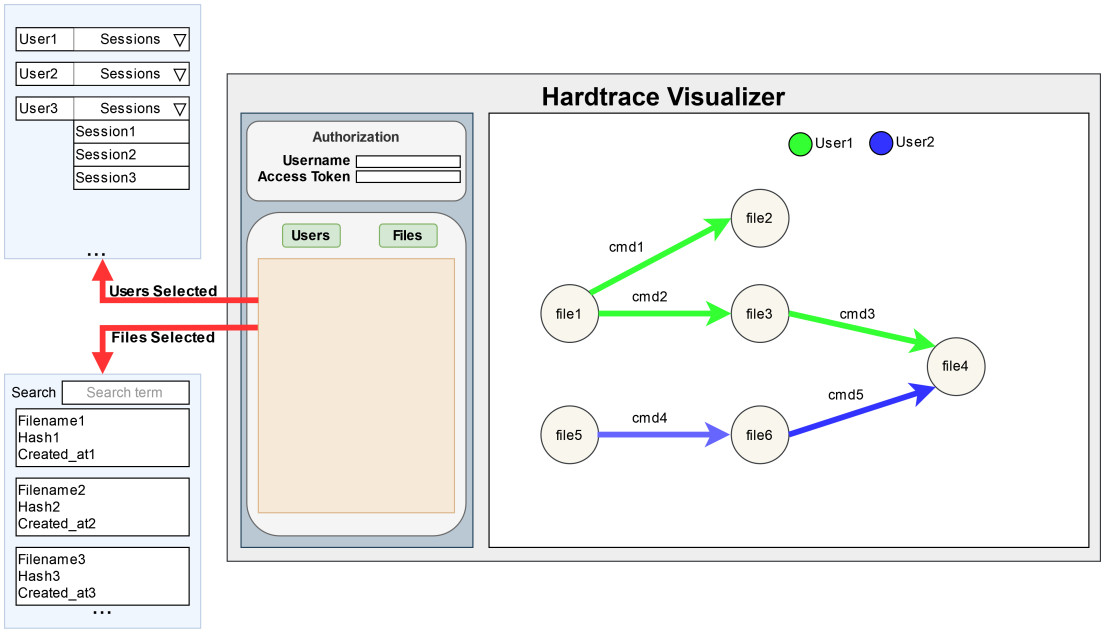


Figure 7. Concept image of the visualizer application.

All information displayed by the VA is obtained from the API resources UserInfo, FileInfo and Summary. This means that, when authorized, the VA is able to see the pseudonyms and session names of all Hardtrace users who have submitted data to the API. The accessed files of these users, as well as, the summarized contents of their Hardtrace sessions are also visible.

Figure 8 displays the implemented VA. The implementation consists of five different tiles, each with a different purpose.

- Tile 1 allows the application's user to input their authorization details. These details are then included in the header of requests sent to the API.
- Tile 2 provides the user with a way to select and search for Hardtrace sessions and accessed files. The user's selection in this tile decides the visualization's contents. When the "Update" button is pressed, the application updates the displayed list of sessions and files.
- Tile 3 includes the interactable visualization created based on the graph data obtained from the API.
- Tile 4 contains the visualization's legend.
- Tile 5 consists of three buttons. The "Visualize" button start visualizing the graph based on the sessions or files selected by the user. Visualized graphs can be saved in the Portable Network Graphics (PNG) by pressing "Download". The "Settings" button opens the view shown in Figure 9. This view allows the user to exclude files and users from visualizations. It can also be used to change the quality of saved PNGs.

Interaction techniques supported by the VA include zoom and pan, filtering, rearrangement of nodes, and details on demand. Geometric zoom and panning

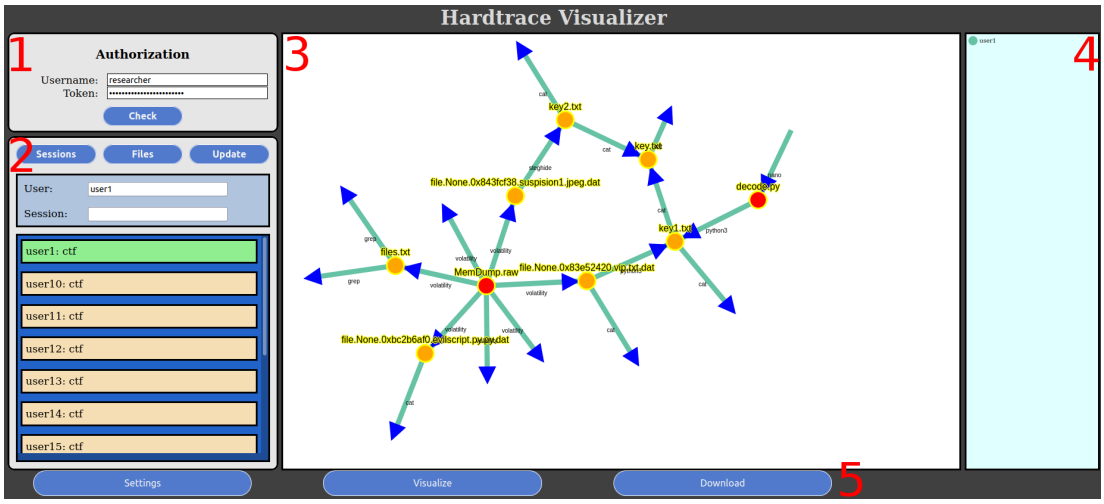


Figure 8. Dashboard and visualization implementation with numbered tiles.

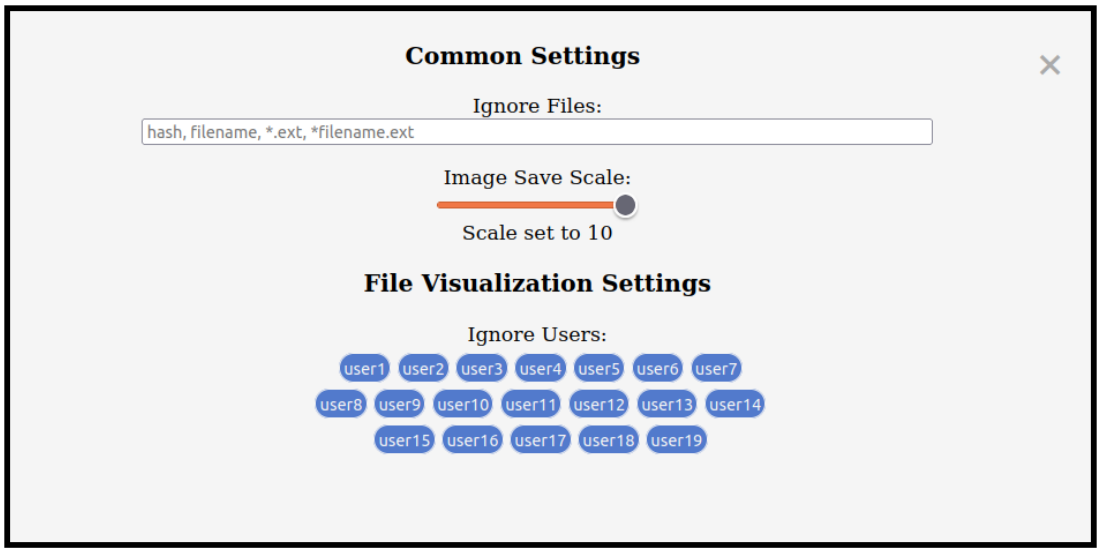


Figure 9. Visualization settings view.

are implemented utilizing D3. Data filtering is enabled by the settings view. Rearrangement of the graphs' nodes is done using the d3-force module. Details on demand are provided by both, the dashboard and the visualization. The dashboard provides an overview of users, sessions and files. Additional details are then seen from the visualized graph. In addition, the visualization only displays tool and command names on edges. Full executed commands can be seen by hovering the computer's mouse on top of edges. Displaying only the start of each command reduces clutter and visual occlusion in the visualizations.

In the information visualization pipeline of the VA, the raw data is produced by Hardtrace, preprocessing is performed by the API, conversion to visual symbols and the visual mapping process are performed by the application using D3.

4. EXPERIMENTS

This chapter present, in the form of a case study where the participants play the role of forensic researcher, how the updated version of Hardtrace, the API, and the VA can be used to collect, visualize and analyze data detailing the progress of digital forensic evidence examination and analysis from multiple users. Additionally, performance testing of the implemented system is presented.

The case study divides into the visual analysis of collected Hardtrace data, and the statistical analysis of Hardtrace session summaries. System performance, on the other hand, presents how using Hardtrace affects command execution time, as well as, how long the summarization and visualization of Hardtrace sessions takes.

In order to collect data from different users in the case study, participants were enlisted to complete a CTF challenge while using the new version of Hardtrace.

During all experiments, the containerized API was deployed to Google's Cloud Run service. The collected data was stored to Google's Cloud SQL[23] DB.

4.1. Setup

The basic setup for all experiment, in regard to the API and the visualizer application, is the same. The API is containerized using Docker and deployed to Cloud Run. Data sent to the API by Hardtrace is stored to and read from a PostgreSQL DB provided by Cloud SQL. The browser-based VA was used locally. All communication between the system components was done over HTTPS.

4.1.1. Case Study and Selected CTF

In total, 18 participants took part in the case study between the months of October and December 2021. The group consisted of students from the Computer Science and Engineering degree program of the University of Oulu, as well as, staff from the Oulu University Secure Programming Group. Each participant was given basic instructions on how to participate. However, all participants worked on the challenge individually and without further guidance.

The CTF case study participants had to complete was the freely available Jeopardy-style memory forensics challenge, titled *MemLabs Lab 3 - The Evil's Den*, created by P. Abhiram Kumar [54]. This specific CTF was selected since it can be completed in its entirety by just entering commands into the Bash shell. From the context of digital forensics, the evidence identification and collection stages have already been done, individuals completing the challenge only need to perform the examination and analysis stages.

The challenge's description states that two different digital forensic tools are required to find the flag (text string) which is divided into two differently hidden parts. These tools are Volatility[24] and steghide[25]. In order to complete the CTF, participants

[23] https://cloud.google.com/sql
[24] https://www.volatilityfoundation.org/
[25] http://steghide.sourceforge.net/

had to find and extract three specific files from the collected evidence, i.e. a memory dump, using Volatility. The first is a text file which contains the flag's first half. However, it is encrypted. The second file is a Python script that was used to encrypt the text file. Thus, to obtain the first part of the flag, participants had to decrypt the text file's contents. The last important file is an image, which includes an embedded text file, i.e., the flag's missing half. By using steghide, the rest of the flag can be extracted from the image. Instructions provided to the case study participants also requested both halves of the flag to be combined into a single file before finishing.

To make participating in the study easier, a VM which included the required forensic tools, the challenge's files, Hardtrace, and basic instructions, was created. Participants were provided a download link to the VM.

4.1.2. Performance

During performance testing, Hardtrace and the visualizer application were run on a Lenovo ThinkPad laptop with an Intel i7-8565U 1.80 GHz CPU and 16 GB of memory. Both, the old and new version of Hardtrace were used.

4.2. Experiment Descriptions

The following subsections describe the case study and performance analysis.

4.2.1. Case Study

The goal of the case study was to validate that the system, as a whole is able to collect, summarize and visualize data detailing how multiple Hardtrace users performed evidence examination and analysis, allowing for the reconstruction of their work. This was verified though the analysis of Hardtrace user session summaries, as well as, through the visual analysis of participants' graph data.

Session summaries and the graph data tell the same story. However, the summaries provide more details. Thus, visual analysis was utilized to determine how individual users completed the challenge. Furthermore, visual analysis was used to showcase the capabilities of the visualizer application and the API, as well as, to highlight that graphs for different participants varied widely. Summary analysis, on the other hand, was used to obtain statistical information on how the participants, as a whole, completed the CTF challenge, as well as, to demonstrate the different types of information that can be extracted from the summaries.

The statistical information obtained during the analysis process includes the following:

1. Participant progress in the CTF.
2. Time between consecutive commands.
3. Time in total spent solving the challenge.

4. Time spent actively working on the challenge.
5. Number of executed commands.
6. Used commands and tools.
7. Session starting commands.
8. Number of accessed files.
9. Volatility plugin utilization.
10. Number of failed attempts at obtaining the flag's second part with steghide.

From the above mentioned statistics, **1.**, **2.**, **3.**, **5.**, **6.** and **7.** have been used in [25, 26]. Other targets of analysis can also be found in similar works. However, they were deemed unsuitable for different reasons:

- *Different shell usage*: The number of commands entered into the Bash and Metasploit shells was observed in [26]. However, the case study's CTF only requires the use of Bash shell. In addition, Hardtrace does not record which shell is being used.
- *Flag submissions*: In some CTFs, especially multi-level challenges, participants are required to submit the correct flag to, e.g., a website in order to progress to the next level. Taking advantage of this fact, different statistics related to submitted flags were used in [31, 55]. Examples include the ratio of correct and incorrect flags, time between taking a hint and submitting the correct flag, etc. The selected CTF does not include multiple levels or hints, nor is the implemented system able to detect flag submission and hint taking, making observing such actions infeasible.
- *Detecting plagiarism*: Patterns indicative of plagiarism were investigated by Vykopal et al. [55]. Patters included, among others, the number of CTF participants who completed the same level within 10 minutes of each other, as well as, the time participants took to complete two consecutive levels. Observing such patterns from the data collected in the case study is not possible, as participants did not work on the challenge at the same time.
- *Effect of a classroom setting*: Owens et. al [56] examined how a classroom setting affected the number of levels CTF participants were able to complete compared with participants not working in a classroom. As the case study's participants worked individually without guidance, this was deemed unsuitable.

4.2.2. Performance

Performance testing was carried out to determine how using Hardtrace affects the execution times of tools and commands used during forensic analysis. The number of syscalls Hardtrace stores into the database was also observed. To further demonstrate the effect of Hardtrace, the commands were also executed without Hardtrace.

For this experiment, the aforementioned information was collected for a set of 27 commands, shown in Figure 10. This set consists of commands found in the Hardtrace data, collected from participants during the case study. Input files used in the experiment are the same as the ones the case study's participants encountered while completing the CTF. Some of the commands write their output to files, others merely display it in the shell.

```
volatility -f MemDump.raw imageinfo
volatility -f MemDump.raw verinfo
volatility -f MemDump.raw –profile=Win7SP0x86 filescan > files.txt
volatility -f MemDump.raw –profile=Win7SP0x86 filescan
volatility -f MemDump.raw –profile=Win7SP0x86 iehistory
volatility -f MemDump.raw –profile=Win7SP0x86 pstree
volatility -f MemDump.raw --profile=Win7SP0x86 pslist
volatility -f MemDump.raw –profile=Win7SP0x86 psscan
volatility -f MemDump.raw --profile=Win7SP0x86 envars
volatility -f MemDump.raw –profile=Win7SP0x86 windows
volatility -f MemDump.raw –profile=Win7SP0x86 cmdscan
volatility -f MemDump.raw --profile=Win7SP0x86 cmdline
volatility -f MemDump.raw --profile=Win7SP0x86 cmdline -p 3736 ---dump-dir=files
volatility -f MemDump.raw --profile=Win7SP0x86 cmdline -p 3432 –output-file=files/notepad.txt
volatility -f MemDump.raw --profile=Win7SP0x86 memdump -p 3736 –dump-dir=files
volatility -f MemDump.raw --profile=Win7SP0x86 memdump -p 3432 –dump-dir=files
volatility -f MmeDump.raw --profile=Win7SP0x86 screenshot --dump-dir=files
volatility -f MemDump.raw --profile=Win7SP0x86 dumpfiles -Q 0x000000003de1b5f0 --dump-dir files/ -n
volatility -f MemDump.raw --profile=Win7SP0x86 dumpfiles -Q 0x000000003e727e50 --dump-dir files/ -n
volatility -f MemDump.raw --profile=Win7SP0x86 dumpfiles -Q 0x0000000004f34148 --dump-dir files/ -n
steghide info suspision1.jpeg -p inctf{0n3_h4lf
steghide extract --force -sf suspision1.jpeg -p inctf{0n3_h4lf -xf files/key2.txt
grep -E 'evilscript.py' test-files/files.txt
grep -E '*.png' test-files/files.txt
file MemDump.raw
base64 --decode vip.txt
strings suspision1.jpeg
```

Figure 10. Commands used in performance testing.

Before the visualizer application can illustrate Hardtrace sessions as graphs, it must first obtain the graph data from the API. However, the API has to first summarize the sessions and then transform them into graph data. Thus, to determine how long the summarization and visualization process takes, times to visualize the graphs of case study participants' Hardtrace session were observed.

4.3. Results

This section presents the experiments' results. Starting with the visual analysis on how a select group of case study participants completed the CTF challenge. Following that, the results of the statistical analysis of all participant Hardtrace sessions are presented. Finally, the performance of the developed system is elucidated.

In the results, the case study's participants are referred to with the pseudonym user2 - user19. The following abbreviations are used when presenting statistical information: Minimum (Min), Maximum (Max), Average (Avg) and Standard Deviation (SD).

4.3.1. Case Study: Visual Analysis

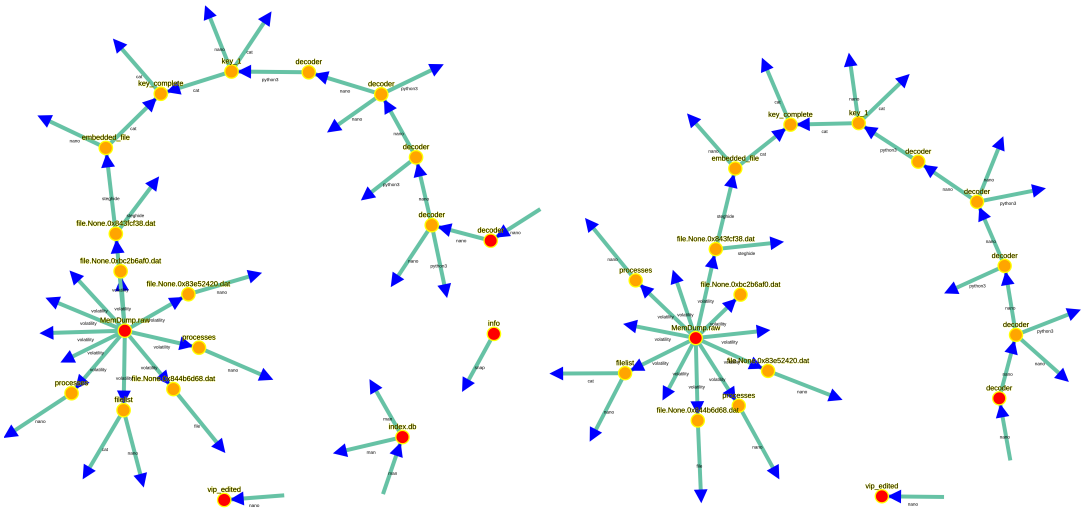
In this experiment, the graph data generated based on the Hardtrace sessions of case study participants user3, user8 and user14 were explored and analyzed visually utilizing the VA. The graph data was obtained from the API.

User3

When visualized, the graph for case study participant user3, is as shown in Figure 11(a). The graph contains five files highlighted with red color as source nodes. Paths leaving from two of the source nodes also combine into one. A slight overlap of two edges is also present near one source node.

From the five source nodes, index.db and info are not of major interest. The file index.db is accessed and modified when the man command is used to display the manuals of different tools and commands. This index DB contains information on the state of the manual page system, and is used to enhance the speed of manual-related functionalities [57]. The file named info, on the other hand, is recorded by Hardtrace when the user attempts to run a command that does not exist.

By utilizing the application's settings window, the two aforementioned files can be excluded from further visualizations. The user can also fix node overlap by moving nodes around. The result of these actions is illustrated in Figure 11(b).



(a) Original. (b) Modified.

Figure 11. Visualized graph for participant user3.

From the graphs, it can be seen that user3 used the memory forensics tool, Volatility, to perform multiple actions on the challenge's memory dump. Volatility plugins such as pstree, filescan, cmdline and dumpfiles were employed. The outputs of some Volatility commands were saved to files from where they were easy to view with the commands cat and nano.

Of importance to the challenge, are the three following files extracted by user3 from the memory dump:

- file.None.0xbc2b6af0.dat: A Python script used to encrypt a text file containing the flag's first part. Encryption used on the file includes an Exclusive OR (XOR) logical operation and base64 encoding.
- file.None.0x83e52420.dat: Text file containing the encrypted first half of the flag.
- file.None.0x843fcf38.dat: An image from which the flag's second half can be extracted using steghide.

After extracting the text file and the Python encryption script with dumpfiles, the graph shows that to decrypt the flag's first part, user3 created a file named decoder which contains a Python script. The decoder script was modified multiple times before the script finally worked and saved the decrypted flag section into the file called key_1.

By hovering the mouse over the edge between the image file and the file titled embedded_file, we can see that the command "steghide extract -sf img -xf embedded_file" was used by user3 to extract the flag's second half. This is illustrated in Figure 12.

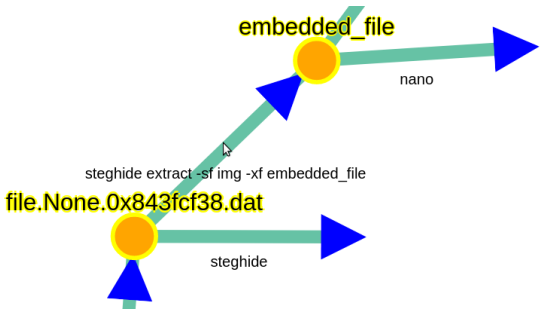


Figure 12. Command user3 used to extract the flag's second half from the image file.

Finally, the graph's edges connecting the files key_1 and embedded_file to key_complete show that user3 finished the challenge by combining the flag's both halves into a single file using the command cat.

User14

The graph created based on user14's Hardtrace session is illustrated in Figure 13. This graph includes three source node files. Unlike in user3's graph, here, the two paths containing the flags first and second part do not connect. Instead, the combined flag, found in the file final_key.txt, is completely disconnected from the graph's main part.

Rather than using a Python script to decode the flag's first part, user14 utilized the "base64 –decode" command to remove the flag's base64 encoding. However, the graph does not tell how the XOR encryption was removed.

The flag's second half was found in a very similarly way compared with user3. However, to locate the correct image file in the memory dump, user14 utilized grep to search their Volatility filescan result for four different image formats. The correct image format to search for was Joint Photographic Experts Group.

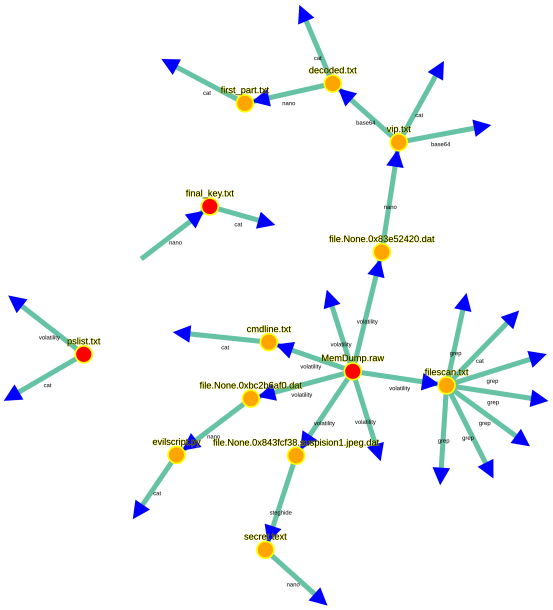


Figure 13. Visualized graph for participant user14.

To combine the flag parts, user14 used the nano text editor without accessing the files containing each part. It is possible that they, e.g., copied the parts from their clipboard to nano, causing the resulting file to be disconnected from the rest of the graph.

User8

Compared with the two previous participants, the graph for user8 is much more complex. This is elucidated in Figure 14(a). It contains many source nodes and much more variance in utilized commands. Additionally, a large cluster of nodes encompasses an important source node, the memory dump.

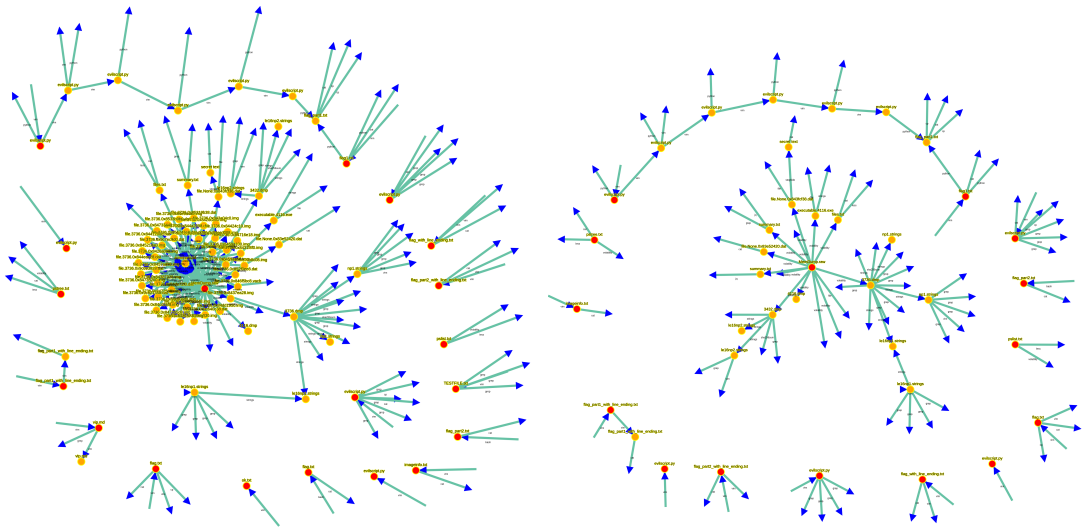
The cluster of nodes was created by user8 extracting a large number of files from the memory dump. A summary (summary.txt) of the extracted files and the memory dump was then created. The command jq was used to view the summary's contents.

To make the graph clearer, the cluster of nodes, among other less important seeming files, can be excluded. This produces the graph shown in Figure 14(b).

The graph shows that user8 performed many of the same actions as the two previously examined participants. These actions include, e.g., saving Volatility's filescan and pstree outputs to files, using a Python script to decrypt the flag's first part, and extracting the flag's second half with steghide. The chain of Python script modifying is very reminiscent of user3's work.

Unlike user3 and user8, this participant used the Volatility plugin memdump to extract the memory of two notepad processes which were used to access the flag's first half and the encrypting script. User8 then employed the commands grep and strings in an attempt to extract valuable information from these processes.

Whereas the others preferred to edit text with nano, user8 used vim exclusively. With vim, user8 created files for both halves of the flag and combined the flag parts into a single file.



(a) Original. (b) Modified.

Figure 14. Visualized graph for participant user8.

Specific file visualization

The specific file visualization functionality of the visualizer application can be used to summarize how each of the three participants obtained the flag parts. Figure 15 illustrates the paths for the participants. It should be noted that the illustration of all three participants' work is the same as in the previous graphs, just condensed to only show paths leading to the flag files. In the figure, file key_1 and first_part.txt are the flag's first half, embedded_file is the second, and key_complete is the combined key. The visualization includes two different nodes for flag part one because first_part.txt included a new line after the flag text. Thus, its SHA-256 hash was different.

This graph shows that all three participants obtained the flag's second half in the same way, using steghide. The flag's first part was obtained by user3 and user8 through the trial and error of modifying a Python script which would handle the decryption. User14 got the first half by using base64 and nano. Finally, the graph illustrates the three different ways of using cat, vim and nano the participants used to combine the flag halves into one single file.

4.3.2. Case Study: Session Summary Analysis

The data analyzed in this experiment consists of the summaries generated by the API based on Hardtrace data collected from all case study participants. The results were obtained using a specifically made Python script.

Participant progress in the CTF

To complete the CTF, participants had to find the flag's both halves and to combine them into a single file. Utilizing this fact, the progress of each participant was

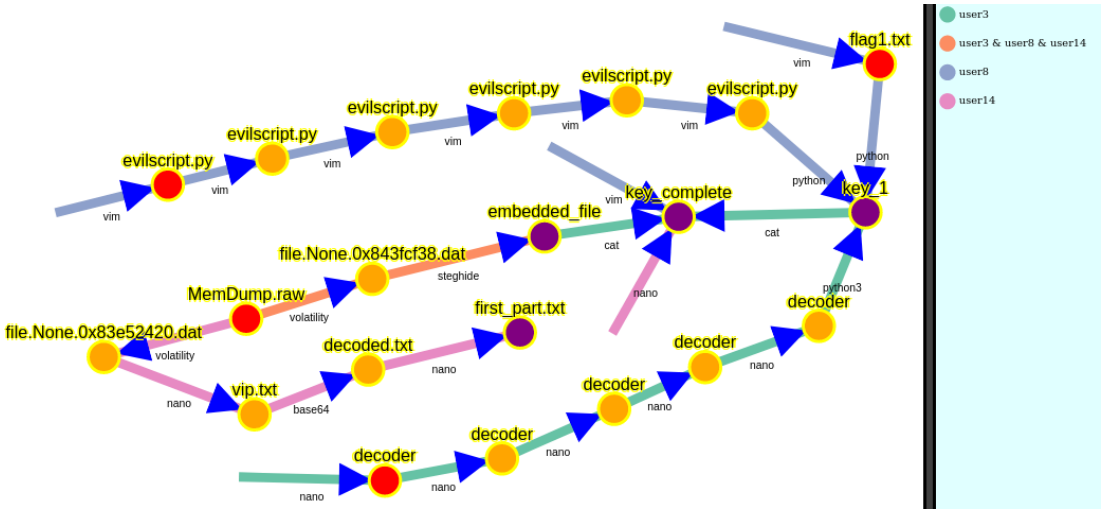


Figure 15. How user3, user8 and user14 obtained each flag part. The graph's legend is included in the top right corner.

determined by searching for the SHA-256 hashes of the flag files in Hardtrace session summaries.

The results show that out of the 18 participants, 15 successfully completed the CTF, giving a success rate of approximately 83%. Only three participants (user4, user16 and user18) were not able to find any part of the flag. While the results indicate that user15 was able to finish the CTF, the data collected by Hardtrace for that specific participant does not include the flag's first part. All who found the second part of the flag also followed the instruction of combining the flag and saving the result into a single file.

Command intervals

Table 2 presents the statistics of time differences between consecutive commands participants entered into their Bash shells while using Hardtrace. The average, standard deviation and median were first calculated separately for each participant. The numbers reported in the table are the average of these three values. Command interval distribution is illustrated in Figure 16.

On average, participants entered a new command into their shell approximately every minute. The longest gab between consecutive commands was nearly two hours. A large portion of intervals between two consecutive commands are 0-50 seconds long.

Participant time usage

Table 3 presents the statistics of the time spent working on the CTF by all 18 participants. Active and overall times for individual participants are elucidated in Appendix 3 Figure 25.

The overall time was calculated starting from the first executed command until the final, combined flag was found. If the final flag was not found, overall time was calculated until the final command execution. Active work times were obtained in a similar fashion, although, breaks longer than 270 seconds were not included. The exclusion of longer breaks was made based on Figure 16, as after 270 seconds, breaks

between commands start to become increasingly infrequent, and were thus considered as idle time.

The overall time participants worked on the CTF was approximately one hour and 8 minutes, on average. From that overall time, approximately 44 minutes was spent actively working on solving the challenge. The fastest completion of the challenge, in slightly less than 30 minutes, was done by user9. Most overall time was spent by user16. However, the active time for user16 indicates a three-hour-long break. Similarly, the participant with the second longest overall time, user11, took a long break from the CTF. User3, user9 and user14 spent the largest portion of their overall time actively solving the challenge. User8 spent the most time actively working.

Table 2. Statistics of the time interval between consecutive CLI inputs

	Min	**Max**	**Median**	**Avg**	**SD**
Difference [m:s]	0:00	110:26	0:24	1:10	2:15

Table 3. Overall and active working time statistics for the participants

Type	**Total**	**Min**	**Max**	**Median**	**Avg**	**SD**
Overall time [m:s]	1232:59	29:41	218:34	60:16	68:30	45:46
Active work [m:s]	789:12	17:00	88:53	42:54	43:51	15:40

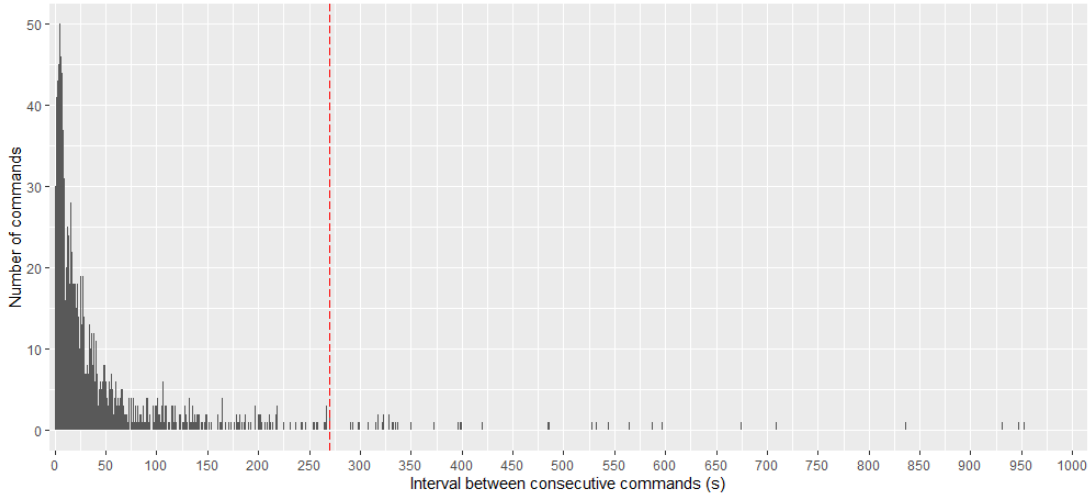


Figure 16. Histogram of consecutive command intervals between 0-1000 seconds long. Divided at the 270-second mark with a red line.

Number of executed commands

Table 4 elucidates the combined statistics of the number of commands entered into the Bash shell by all case study participants. The number of commands for individual participants is shown in Appendix 3 Figure 26.

On average, participants ran 75 commands during their Hardtrace session. Least commands were run by user4 and the most by user8. The number of commands for the group of participants who did not finish the challenge are all among the least. The participant with the second fastest completion time also used the second lowest number of commands during their session.

Table 4. Statistics of command execution for all 18 participants

Total	Min	Max	Median	Avg	SD
1343	16	210	57.5	74.61	44.87

Used commands and tools

Figure 17 details how many times different commands and tools were used by the participants.

The tool which saw the most usage was Volatility. Following Volatility was the commands *cat*, *ls*, *grep*, as well as, the text editor nano. The other forensic tool required to solve the CTF, steghide, was the sixth most used. Four of the observed tools and commands were used only once. These were gedit, eog, *clear* and hardtrace.

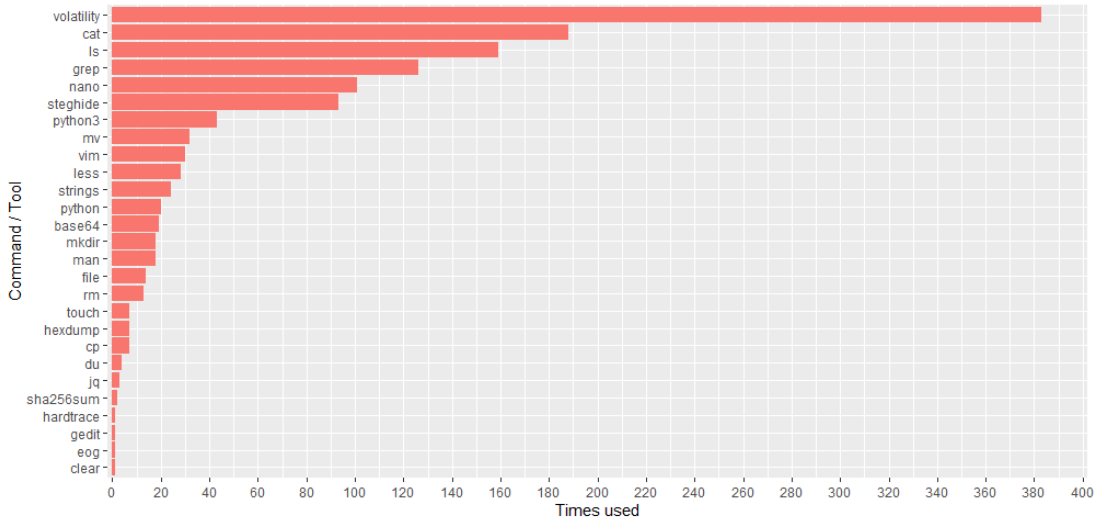


Figure 17. Number of times different commands and tools were used by the 18 participants.

Session starting commands

Table 4 elucidates the first command executed by each individual participant at the start of their Hardtrace session.

Out of the 18 participants 11 started off their session by using Volatility along with the *-h* and *–help* arguments. The command *ls* was used by five participants. Finally, *file MemDump.raw* and *man volatility* were both used only by one participant.

Table 5. First command executed by each participant

Command	Participants
volatility -h, *volatility –help*	user2, user4, user5, user6, user7, user10, user12, user13, user17, user18, user19
ls	user3, user9, user11, user15, user16
file MemDump.raw	user8
man volatility	user14

Number of accessed files

Table 6 presents the statistics related to the number of unique accessed files from every participant's session. The individual numbers of accessed files are shown in Appendix 3 Figure 27. File uniqueness was determined based on SHA-256 hashes.

In total, 945 files were accessed, created and/or modified across all sessions. The average participant accessed approximately 53 files during their session. The smallest number of unique files were accessed by user4. On the other side of the spectrum, user10's session included a grand total of 535 files. The largest number of unique files were found in user13's session. However, user13 accessed 444 files less than user10.

Table 6. Statistics of the number of accessed files for all 18 participants

Total	Min	Max	Median	Avg	SD
945	4	535	13	52.5	122.99

Volatility plugin utilization

Case study participants had to use Volatility and its host of plugins to analyze and extract files from a memory dump. Figure 18 illustrates all plugins that were used during Hardtrace sessions, as well as, how many times they were used. The figure also includes plugins that do not exist, but participants still tried to use them.

In total 269 commands involving the use of a Volatility plugin were found across all participant sessions. From the plugins, dumpfiles, filescan, imageinfo, pslist and cmdline were utilized the most. A total of 19 plugins were used four times or less.

Non-existent plugins, in Figure 18, that participants tried, include: console, extract, filescane, imageInfo, imagescan and dump. These were used a total of seven times.

Failed steghide extract operations

In the selected CTF, steghide had to be used to extract the second part of the flag from an image. When extracting, the flag's first part had to be provided as the password. Figure 19 present the number of times participants used steghide's extract command incorrectly. Participants user4, user16 and user18 were excluded, as they did not attempt to obtain the flag's second half. The results were obtained by calculating the times *steghide extract* was used but did not output a new file.

From the participants who found the flag's second part, only three were able to extract it correctly on the first try. The largest number of failed attempts was five. On average, individual participants used the command incorrectly approximately 2 times while attempting to obtain the flag's second part.

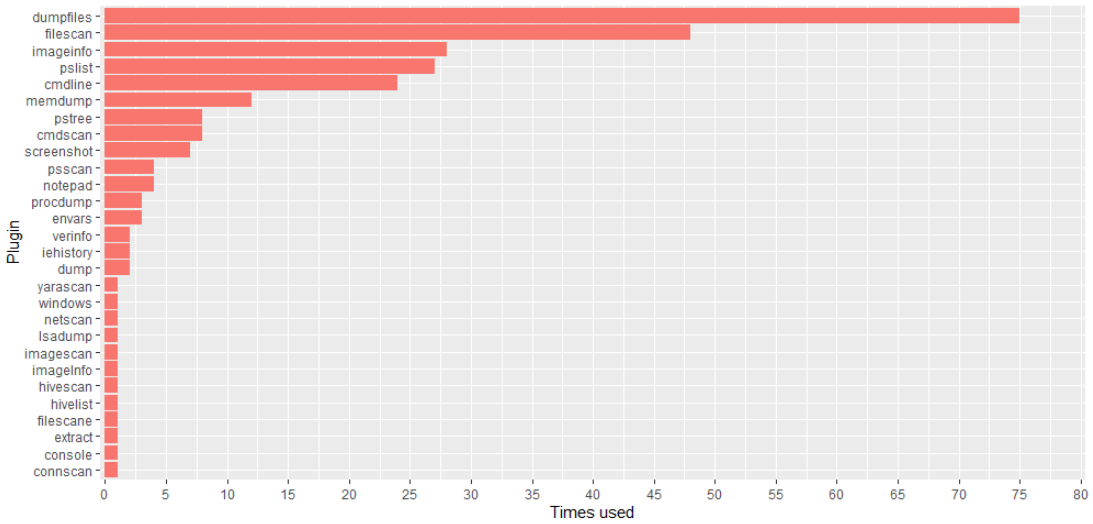


Figure 18. Number of times different Volatility plugins were used. Includes non-existent plugins.

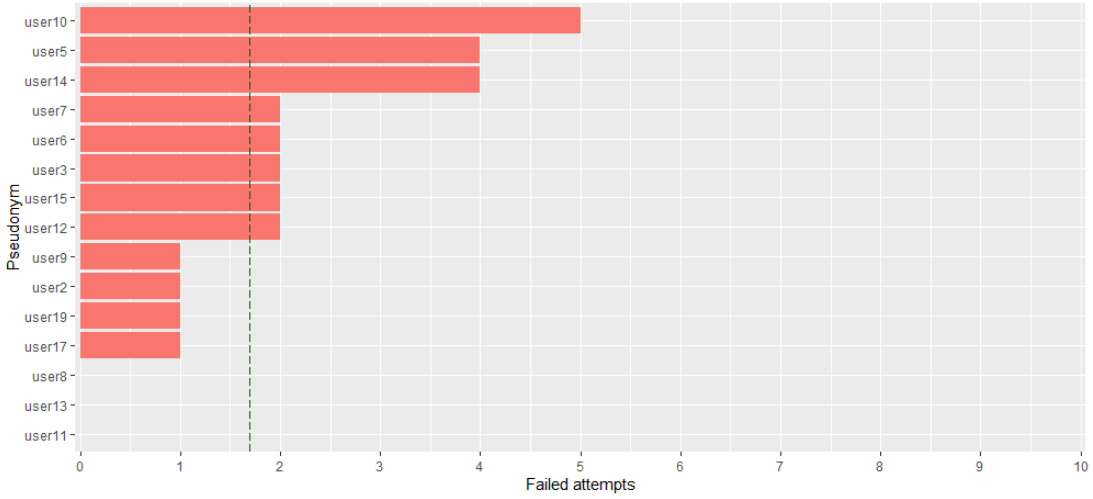


Figure 19. Number of times steghide's extract command was used unsuccessfully by each participant.

4.3.3. Hardtrace Performance

In this experiment, a set of 27 commands were executed multiple times and an average execution time was calculated for each. During this experiment, both the new and

old version of Hardtrace were used. The new version was used online and offline to determine how sending data to the cloud API affected execution times. To further elucidate the effect Hardtrace has, commands were also run without it. Additionally, as the old version of Hardtrace saves all syscalls to its local DB and the new version only saves the unique calls of each process, this experiment also observed the number of syscalls saved by both Hardtrace versions for the executed commands.

The results are presented in Appendix 4 Figure 29. The execution times column shows how many seconds each command took to execute in four cases: without Hardtrace, with old and new Hardtrace offline, and new Hardtrace while uploading collected data. Number of syscalls saved to the local DB for each command by both Hardtrace versions are elucidated in the saved system calls column.

Based on the results, Hardtrace, new or old, has an almost unnoticeable effect on the execution time of commands, such as grep and strings, and on the tool steghide. Significant increases can, however, be seen with Volatility. E.g., when using old Hardtrace, execution times of Volatility plugins like filescan and memdump jump up from seconds to over an hour. The increase is larger with iehistory and envars.

In many cases, the updated Hardtrace version adds only a couple of seconds to the run time of many plugins. The worst observed case for the new version was with the envars plugin, which took approximately 123 seconds to complete. Compared with the old version performance with the same plugin, however, execution time was reduced by approximately 99%. However, while execution times were lower with the new version, in cases where the execution times without Hardtrace are multiplied, the usability is still affected negatively.

At worst, uploading data to the cloud prolonged a command's run time by eight seconds. With many of the commands, the increase was less than a second.

The number of syscalls saved to the local DB was very similar for both Hardtrace versions when using steghide, grep, file, etc. When using Volatility, at minimum, old Hardtrace saved 263 syscalls with the psscan pluging, and at maximum nearly two million when envars was used. New Hardtrace saved only four syscalls for them.

4.3.4. API and Visualizer Application Performance

This experiment was conducted by requesting the visualization of each case study participants' graph, and taking note of how long it took for the visualized graph to be shown to the user. Statistical results for this experiment are presented in Table 7 and times taken to visualize each participant's graph are illustrated in Appendix 5 Figure 30.

On average, it took approximately 10 seconds to visualize the graph representing any participants Hardtrace session. The longest time of 31 seconds was required to visualize user8's graph, and the shortest time for user4.

Table 7. Statistics of visualization times for all 18 participants' graphs

	Min	**Max**	**Median**	**Avg**	**SD**
Time to visualize [s]	1.1	31.1	6.65	9.87	8.03

5. DISCUSSION

In this chapter, the experiment results are analyzed. In addition, this chapter presents the limitations of the performed experiments, as well as, possible risks to the results' reliability. How the experiments could be further improved and the future work in regard to the system developed in this thesis are also discussed.

5.1. Analysis of Results

The experiments, performed in this thesis, divided into two parts, i.e., the case study and system performance testing. The case study elucidated how data detailing the progression of digital forensic examination and analysis could be collected from multiple individuals utilizing the implemented system, as well as, how the reconstructions of the individuals' work could then be further analyzed visually and statistically. Performance testing examined the effects of using Hardtrace on the time commands and tools took to execute. The time taken to summarize and visualize the collected data was also observed.

Analysis of the case study results is presented in two parts, starting with the performed visual analysis of the select three participants' work, followed by the statistical examination of the summarized Hardtrace data from all participants.

Based on the results of the visual and statistical analysis, it is clear that the implemented system performed well in the tasks of collecting data on, summarizing and visualizing, thusly in documenting the progression of digital forensics. The design of Hardtrace session summaries, produced by the API, was well-suited to support the analysis and visualization of collected Hardtrace data. Using D3 to visualize the data as directed graphs was also a good choice.

The visual analysis demonstrated that the implemented system was indeed able to produce graphs from which the steps taken during the examination and analysis stages could be determined in a straightforward fashion. As such, by utilizing the VA, a forensic researcher is able to infer the meaningful parts of Hardtrace sessions while also having the ability to leave out the less important bits cluttering up the graphs.

Statistical analysis showed that a large variety of information on how participants work can be obtained by utilizing the Hardtrace session summaries created by the API. Statistical analysis results can thus be used to strengthen the understanding of participants work previously gained through the visual analysis, helping in assessing the overall quality of work and identifying aspects from the digital forensics process that should be improved on.

Finally, the performance analysis presented that updated done to Hardtrace were successful in reducing the overhead added to command and tool execution. Taking into account, both, the performance and the amount of data Hardtrace can collect, Hardtrace proofed to be a very useful tool. The API and the VA were also shown to be able to summarize and visualize data in a very reasonable time.

5.1.1. Case Study: Visual Analysis

In this experiment, graph data representing the Hardtrace session of three case study participants was obtained from the API. The VA was utilized to perform visual analysis on these graphs. The experiment's main goal was to demonstrate that the implemented system enables its user to visually determine what was done during the digital forensics process.

In all three demonstrated cases, determining how each participant solved the CTF was a straightforward process. A large majority of all important commands and files, from the context of solving the CTF, were found by just traversing the graphs' paths leaving from the node representing the memory dump file, *MemDump.raw*. Looking further was only necessary when determining how user8 obtained the flag's first half, as that information was contained in a disconnected section of the graph.

The ability to leave out files based on extensions, names and hashes through the VA's settings page also proofed useful. Excluding files from the graphs reduced clutter and made the result more readable. This can also help drill down on specific sections of graphs, by leaving out unnecessary sections.

Similarly to session visualization, the specific file visualization was able to produce an easily understandable visual summary on how each of the three participants obtained the flag's parts. The resulting illustration only included the command chains and files directly responsible for producing correct flag sections. This means that the result only contains important commands. However, not all of them. As an example of this, Volatility commands used to locate the flag files from the memory dump were not included.

When inspecting the graph for participant user14, the viewer is left with the question: How did user14 remove the XOR encryption from the flag's first half? Only the removal of the base64 encoding was included in the graph. This is due to the way Hardtrace records data. If the XOR decryption is not done in the Bash shell, in which Hardtrace is running, it will not be included in the collected data. It can, however, be speculated that user14 performed the decryption in another instance of the Bash shell or, alternatively, with the help of an online tool, such as CyberChef[26]. Graphs for participants, other than user3, user8 and user14 also suggest that actions were performed outside Hardtrace's reach. This is unfortunate as these actions are not recorded and thus not accessible to the researcher.

In addition to just detailing what was done, graphs generated based on Hardtrace data can convey knowledge on differing participant behaviors. As an example of this, from the analyzed graphs, user8's is vastly different from the two others. It includes more varied tool and command usage, and an overall more thorough CTF solving process.

As mentioned earlier, the graphs can help determine the most important commands from a forensic analysis. During the visual analysis, some of the commands performed by case study participants could also have been deemed redundant from the point of solving the CTF. Commands could have been deemed redundant, e.g., if the paths they create lead to a dead end with no important files. However, the redundancy of an executed command is very subjective. It is possible, e.g., that a command with

[26] https://gchq.github.io/CyberChef/

seemingly no importance to the observer, was actually very meaningful to the user based on its output or some other factor leading its user to the right path in regard to solving the challenge.

All in all, creating directed graph visualizations of participant sessions utilizing D3 proofed to be a good approach to visualizing how forensic examination and analysis progressed.

5.1.2. Case Study: Session Summary Analysis

In this experiment, the summaries of each case study participant's Hardtrace session were produced with the API. Statistical information on how participants completed the CTF was then extracted from the summaries with a separate Python script. Some results are also compared with the findings of two scientific works focused on CTF monitoring. Thanks to the created design of Hardtrace session summaries, performing the analysis was a simple process.

Participant progress in the CTF

In this test, the SHA-256 hashes of flag files were used as milestones. In total, there were three milestones to reach: the flags first and second half, as well as, their combination. Determining which milestones were reached by participants was done by searching for the hashes in session summaries.

Out of the 18 case study participants, 15 reached all three milestones and completed the CTF, whereas only three participants were not able to find even the flag's first half. Taking into account the somewhat vague instructions of the CTF, the number of successful completions was somewhat surprising. As the study was unsupervised, it is possible that some participants utilized online resources, such as writeups detailing what should be done to complete the CTF in order to reach the end.

User15 was an outlier in this test, as the collected data shows them reaching only the milestones two and three, suggesting that the first flag section was never found. However, it had to be found so the other milestones could be reached. By looking further into the participant's summary, it can be noted that they used a Python script to decrypt the flag's first part. The script did not save the result in a file, and thus, the first half is not present in the collected data.

In a similar fashion to this test, the progress of participants in a multi-level CTF was tracked with exercise-specific milestones in [25]. Milestones in that study, however, were commands and their outputs. If a participant entered the expected command into the shell and obtained the correct output, a milestone would be met. Results were presented per level as unique milestones met versus the number of attempts, as well as, the ratio of milestone completions over the total number of attempts. Unlike in this thesis, the milestones met by each individual participant were not presented.

In regard to reaching milestones, [25] found that in all exercises, tasks requiring participants to discover the right arguments for a command were the most difficult. The data collected during the case study from the three participants who did not complete the CTF points to decrypting the flag's first half being the most difficult part in this

case. All three participants found the flag's first part but failed to remove the base64 encoding and the XOR encryption.

Command intervals

In this test, the statistics of time intervals between consecutive commands entered into the Bash shell by case study participants were observed.

It was found that during the average participant Hardtrace session, a command was entered into the CLI approximately once every minute. This is sensible as it leaves time for commands to execute and for the participants to think about their next move. The longest interval between two commands was observed to be nearly two hours. Such a gap strongly indicates a break from solving the CTF. The shortest time between commands was found to be zero seconds. This was likely caused by participants piping two command together.

The command intervals in a CTF were also studied in [26] by Švábenskỳ et al. The study's results showed an average gap of one minute and 13 seconds between consecutive commands which is only three seconds longer than the average found in this thesis. Just as in this thesis, the shortest gap between commands was zero seconds in [26]. The longest break observed by Švábenskỳ et al. was approximately ten and a half hours, which is significantly more than the maximum command interval of this thesis. The standard deviation of command intervals was also lower in this thesis than in [26].

Participant time usage

Time spent by the participants actively working on solving the CTF, as well as, the time spent in total on the CTF were examined in this test. Times were calculated from the first executed command until the complete flag was found or, alternatively, until the last command. Gaps longer than 270 seconds between consecutive commands were excluded when calculating active working times.

The results showed that on average, the selected CTF took approximately an hour and eight minutes to complete for the case study's participants. From this average, around 43 minutes was active work. Only two participants spent all their time actively working. Overall times being longer than active times makes sense, as participants were free to, e.g., take breaks as they pleased. None of the participants worked on the challenge during more than one day, even though it would have been possible to split the work over multiple days.

The fastest completion of the CTF took merely half an hour. It is possible that the participants with the fastest times, i.e. user9 and user19 had completed the selected CTF previously or utilized an online writeup on the challenge.

The smallest amount of active work was done by participant user4. As this participant did not complete the challenge, it could be argued that they had very little motivation towards solving the CTF. Similarly, the participant with the longest total time usage, user16, did not finish the challenge. However, based on the collected data, user16 was more invested in solving the challenge, but unfortunately they ultimately failed.

Time spent solving a CTF which required its participants to break into a network was investigated in [26]. The average overall time spent on the challenge by the study's participants was found to be slightly less than two hours. The shortest observed time was approximately 16 minutes, while the longest time was 19 hours. The study did not investigate what portion of the time was active work.

Comparing the results from [26] suggests that the memory forensics related CTF in this thesis was faster to complete for the average participant. Deviations in the overall times, observed in this thesis, are also significantly lower. However, the skill levels of participants from both studies can be vastly different, making comparisons less convincing.

Number of executed commands

The number of commands executed by the case study participants was investigated in this test.

The results seem to support the previously made observation on individual participants. As an example of this, the smallest number of commands was executed by user4. This corroborates the notion of the participant lacking the motivation to solve the challenge. On the other end of the spectrum is user8 with the most commands run while completing the CTF, along with the most time spent actively working. These factors, in combination with the findings from the visual analysis, support the idea that user8 performed a much more thorough analysis of the data given in the CTF compared with other participants.

Like user4, the two others who did not manage to complete the challenge are all among the participants with the lowest number of executed commands. This is a sensible result, as they did not get very far in the challenge.

User19 executed the second least commands. This participant also had the second fastest overall completion time, indicating that they knew exactly what should be done, further strengthening the idea that they had help in some form or had completed the challenge previously. User9, on the other hand, executed nearly twice as many commands compared with user19 but still had a slightly faster completion time. It can still be argued that user9 knew what had to be done to solve the CTF, possibly even in advance, however, the required steps were not as clear to user9.

Mirkovic et al. [25] utilized scatter plots to illustrate the time spent on individual levels of a multi-level CTF versus the number of commands executed by participants. Their findings showed that participants engaged sporadically with the levels over a long period of time. Excluding a few, most of the participants put in the effort to solve challenges. They were also able to determine that some levels were harder than others, as they required more time and commands to complete.

The data collected during the case study of this thesis can be used to create a similar plot. Figure 20 elucidates the case study participant's active time usage versus executed commands. Compared with the findings in [25], most participants in both cases made the effort to complete the challenge, however, the participants did not work on the challenge over a long period of time and instead opted to mainly complete it in one sitting. The data collected in this thesis does not allow comparisons to be made in the difficulty of levels, as the selected CTF had only one level.

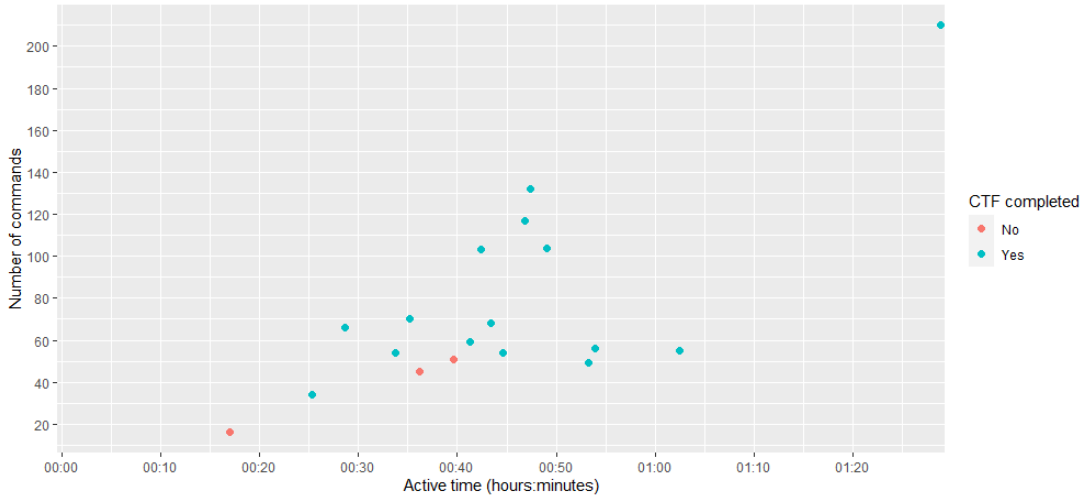


Figure 20. Time spent actively working versus number of executed commands.

In their work, Švábenskỳ et al. [26] observed a strong positive correlation between the time spent on a challenge and the number of executed commands (Spearman's $\rho = 0{,}79$). They used Python's *scipy.stats*[27] module to calculate the result.

Using the same means as in [26] to calculate the correlation between active time spent and the number of executed commands from the data collected in the case study of this thesis produces the result of Spearman's ρ being 0.5 with the p-value of 0.017. The null hypothesis is that the two datasets are uncorrelated. However, since the p-value is low, the alternative hypothesis of a positive correlation existing appears to be true.

Session starting commands

This test was used to determine what was the first command executed by each individual participant at the start of their Hardtrace session.

It was found that when starting to solve the CTF the first course of action for a large majority of the case study's participants was to consult Volatility's short usage description using the *-h* and *–help* command line arguments. This is a good command to start with, as it provides information on how to use Volatility, the tool which plays a very important role in solving the CTF.

The command *man volatility* was used by one participant. *man* fills a similar role as *-h* and *–help*, as it displays Volatility's entire manual. For this reason, it also seems like a good choice to start with. However, the output of *man* contains a lot of information, from which a large portion is unnecessary for solving the CTF, so it can be more overwhelming than *–help*.

Run first by five participants, the command *ls* saw the second most usage. *ls* displays the contents of the currently selected directory. The collected data does not show in which directory participants used the command. In the case that *ls* was used in the

[27] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html

provided VM's directory that contained the files related to the case study, there was not much to see, just the memory dump file and the binary for Hardtrace.

Finally, the command *file MemDump.raw* was used by one participant. The command is used to determine the type of data stored in a file. Like participant user8, who used this command, with the more complex process of solving the CTF, *file* is an outlier in the collected data, as it was used only once and has a totally different use than the more often used session starting commands.

All five session starting commands led to the completion of the CTF. However, by taking into account participant performance, i.e., time to completion and the number of executed commands, displaying tool usage descriptions with the commands *volatility -h*, *volatility –help* and *man volatility* appear to be the best choice to use when starting to solve the selected CTF. Although, it should be noted that two of the three participants who did not manage to complete the CTF started by using the *-h* argument. Even so, the data collected from them shows that they managed to use Volatility correctly.

Using the *–help* command line argument to obtain information on how to use a forensics tool that is important for solving a CTF challenge was also observed to be a common action to start with in [26].

Used commands and tools

In this test, the different commands and tools, as well as, how many times they were used by the case study's participants were determined.

It is no surprise that Volatility was used the most, as many different actions had to be performed with it in order to complete the CTF. Its usage was also focused to the start of the CTF, so even the three participants who did not get very far used Volatility multiple times.

Steghide, the other forensic tool required to solve the challenge, saw only the sixth most usage. This is, again, understandable since whereas Volatility had to be used with multiple of its plugins, it was possible to use steghide only once or twice while working on the CTF. As an example of this, steghide could have been first used with the *–help* argument to determine how it works. Now knowing how to use steghide, the second step was to extract the flag's second half from the image found in the memory dump. This would amount to steghide being used merely twice by a single participant. Another reason for steghide not being used more is that the three participants who did not complete the CTF never used it.

Other than the two important forensic tools, commands that are common in normal use of Linux OSs were among the most utilized. The second most used command was *cat*. It was commonly used by the participants to print out the contents of files, as well as, to concatenate the flag's two halves into a single file. Following *cat*, were the commands *ls* for directory content displaying and *grep* for finding specific file contents.

The fifth most used tool was nano. Compared with the two other text editors participants used, i.e., vim and gedit, nano was used significantly more. This shows that participants clearly preferred to use nano as their text editor of choice.

Located among the commands and tool that were only used once by the participants is Hardtrace. This shows that one participant attempted to start another Hardtrace session inside their current one. Although, Hardtrace does not allow its user to do that.

Unlike in this thesis, in [26] it was found that instead of the challenge's main forensic tool, i.e. nmap being the most used, the command *ls* was executed the most (568 times). The command *cd* was also used more (320 times), whereas nmap was only used 240 times and the password cracking tool, named john, 201 times.

The number of times different commands and tools were used very not reported in [25]. However, they used their ACSLE system to produce illustrations of the frequently used commands in failed attempts to reach milestones of the CTF's levels. Commands and tools observed in this thesis and in [25] included *cd*, nano and vim.

Volatility plugin utilization

Determining the different Volatility plugins used by the case study participants, along with how many times each plugin was used, were the focus of this test.

It should be noted that while Volatility has many plugins [58], only a few of them were actually useful for completing the selected CTF as Volatility had to only be used to extract a text file, a Python script and one image from a memory dump that were related to the challenge's flag.

The results show that the top five most used plugins were all essential for finishing the challenge. A good example order of action for completing the parts where Volatility is needed was to use the plugins in the following order:

1. **imageinfo**: Obtains profile information on the target memory dump. Profile info had to be provided as an argument when using the other plugins.

2. **pslist**: Lists all processes that were running when the memory dump was created. The output contained two text editor processes with which two of the flag related files were opened.

3. **cmdline**: Displays the command-line arguments of each running process. Output included the name of the two aforementioned flag files.

4. **filescan**: Displays all files included in the memory dump. Searching for the names of flag related files from this plugin's output produced the files' memory location.

5. **dumpfiles**: Extracts files based on a given memory address. Used to extract the three important files from the memory dump.

From the less utilized commands, pstree and pscan could be employed for the exact same purpose as pslist. Although, based on the collected data, participants clearly preferred to use pslist. The use of cmdline could have been replaced with the plugin iehistory, as it was also able to display the names of the encrypted text file and the Python script.

The collected data also included seven uses of misspelled and non-existent plugins. The misspelled plugins were imageInfo, filescane and console. It seems clear in these cases that the participants attempted to use the plugins imageinfo, filescan and consoles. Plugins with the names dump, imagescan and extract do not exist.

Other than the aforementioned plugins, out of 28 observed ones, 13 can be considered not useful for this challenge, as they produced irrelevant information when

used. Additionally, the plugins notepad and connscan did not work when used on the challenge's memory dump.

Failed steghide extract operations

This test observed how many times steghide's extract argument was used incorrectly as participants tried to obtain the flag's second half from an image file.

In order to be able to extract the flag half from the image, the first flag half had to be provided as either an argument or when extract prompted the user to provide a password. While the correct password might not have been immediately obvious to the case study's participants, the CTF's instructions did state that the first half was required to obtain the second one.

The collected data indicated that out of all participants who reach this point in the challenge, only three struggled at least somewhat with using extract correctly, requiring four or five attempts to succeed whereas others only needed up to two tries.

Number of accessed files

This test examined the number of unique files the case study participants accessed while solving the CTF.

It was found that the total number of unique files from each session was 945. From this total, 535, i.e. approximately 57 percent, were from user10's Hardtrace session. The least files were accessed by user4, though this can be expected as they used the least time and commands out of all participants. On average participants created and/or modified approximately 53 files. Although, this number is largely affected by the data collected from user10. The deviation in the number of files was also very large due to user10.

Further examination of user10's Hardtrace session reveals that they left Volatility's *dumpfiles* plugin running for long time. Due to the same reason, the number of accessed files is inflated, although not as much, also for participants user6 and user8. The last participant for whom the number of accessed files is over the average is user13. In their case, the use of the image viewing tool, eog, added 65 cache files to the total.

In all occasions, apart from user8, the large numbers of files created by Volatility and eog were used for nothing by the participants. Thus, to get a more accurate view on how many files participants actually used, such files can be left out. The effect of this on the statistical values is shown in Table 8, and the numbers of files for each participant in Appendix 3 Figure 28.

Table 8. Accessed file statistics with unused files removed

Total	**Min**	**Max**	**Median**	**Avg**	**SD**
315	4	83	13	17.5	17.03

By excluding unused files, the total number of files dropped down to only 315, a reduction of approximately 67% from the original total. Now, on average participants accessed around 18 files during their session. The standard deviation is also now much lower.

With these changes, user8 still retains the high number of files, as they were used by the participant. However, user10 whose session originally included the most files, now includes the second least. User6 dropped down to a group of five other participants, all with 13 accessed files. Even with unused files removed, user13 still used the second most files.

5.1.3. System Performance

This experiment aimed to determine two things. Firstly, how does Hardtrace affect the time spent performing forensic analysis? Secondly, how long does the implemented system take to produce visualizations detailing the progress of forensic analysis based on the data collected by Hardtrace. To these ends, execution times in both cases were observed.

Hardtrace performance

In this test, a group of commands, selected from the ones used by case study participants, were executed multiple times in four different situations: While using the old version of Hardtrace, updated Hardtrace both offline and online, and lastly without Hardtrace. The average execution times of commands were calculated for each. In addition, the number of syscalls saved by both Hardtrace versions was documented.

In the collected results, the execution times of commands run while not using Hardtrace serve as the baseline to which the usage of Hardtrace can be compared with, enabling the evaluation of execution time overhead added by Hardtrace.

Starting with the old Hardtrace version, the overhead added to running basic commands, such as file or grep was at most one second, so from the point of view of the user, no noticeable degradation in execution times occurred. The same goes for the forensic tool steghide. However, when using Volatility with its various plugins, it is very clear that Hardtrace added significant amounts of overhead to all but three. Only for the plugins verinfo, psscan and dumpfiles did the execution time increase by a maximum of approximately four second. Although, using dumpfiles was only fast in one of three cases, when extracting the Python script used for encryption.

When using, e.g., the iehistory, envars or filescan Volatility plugins, recording the steps taken during forensic analysis with old Hardtrace becomes quite detrimental, as the time taken just to execute the commands jumps from mere seconds to hours. As such, the forensic researcher would either have to skip using these plugins altogether or accept the fact that a large amount of time will be spent waiting instead of working.

Like the old version, updated Hardtrace also adds very minimal overhead to the basic commands and to steghide, but a clear difference lies with Volatility. However, the modifications done to Hardtrace largely alleviated the problem of increased command execution times, as shown in the results of this test. Even with the fastest Volatility plugins, the new version is approximately 26% faster for verinfo, 23% for psscan, and 62% for dumpfiles when extracting the Python script. From the more time-consuming plugins, the new version shaved off hundreds and even thousands of seconds of execution time. The largest improvement was seen with the envars plugin, as the execution time went down from nearly three and a half hours to just two minutes.

For just over half the plugins, new Hardtrace is a few seconds off from the baseline. For the plugins filescan, windows, memdump and screenshot the added overhead, compared with the baseline, was between approximately 26 and 40 seconds. The largest increase from the baseline, 114 seconds, was with envars.

While working online with updated Hardtrace, the collected data is uploaded to the cloud API. In most cases, the increased execution time overhead caused by this is shown to be less than a second. The largest amount of additional execution time was added to the Volatility plugins iehistory, envars and dumpfiles.

Looking into the results from the point of view of how many syscalls the old version of Hardtrace saved, shows a strong positive correlation between execution times and the number of syscalls (Spearman's $\rho = 0{,}98$, one-sided p-value $< 10^{-21}$). This correlation, however, is non-existent for the updated version. Clearly, saving all syscalls to the local DB had a large impact on command execution times.

The results demonstrate that the updates done to Hardtrace were a success. The only main change designed to enhance performance compared with the original version, i.e. saving only unique syscalls, worked very well, reducing execution times significantly. Uploading data in a separate goroutine was also a good choice to make, as demonstrated by the mostly small increases in command execution times. The performed changes made Hardtrace a more viable option for collecting data, as its negative impact on time usage was lessened.

API and Visualizer Application Performance

To produce visualizations of collected Hardtrace data, the implemented API has to first create the graph data and send the result to the VA, which then creates the actual visualization. This test investigated how long this process took for the data collected during the case study.

Overall, producing visualization detailing the work of each case study participant was fast, with the longest time to visualize being just over half a minute. Over half of the graphs took less than 10 seconds to visualize. The most time was spent on the graphs for participants user8 and user10, the Hardtrace sessions of both individuals included very high numbers of accessed files and unique commands.

Although, while it would seem logical to assume that the more commands and files a session includes, the longer it would take to visualize, this, however, does not appear to hold true in all cases. Graphs for participants user9, user12 and user15 serve as great examples of this. The number of accessed files for all of them were among the lowest. User12 and user15 also executed only a few commands. User9, on the other hand, was close to the average number of executed commands. Yet despite these factors, the graphs for these participants were among the top five most time-consuming ones.

Based on the performance results, the implemented system is a valid choice for the summarization and visualization of Hardtrace data, requiring only little time to illustrate graphs of differing sizes. As such, during the visual analysis of collected data, very little time has to be spent waiting, allowing the researcher to focus on the important tasks.

5.2. Comparison to Related Work

In this section, the implemented system is compared with similar works from scientific literature. The comparisons are drawn in two separate parts, first for data collection, secondly for the visualization.

Data collection

From the point of view of acquiring data, the implemented system compares best with the works [22, 25, 26], previously discussed in the Background chapter.

Compared with the three examples, starting to collect data using Hardtrace and the API is simpler. Their user does not need to start using cloud-based CTF frameworks or setup sandboxed networks. Hardtrace can be run on Linux machines as is, and based on the experience of conducting the case study, the API can easily be deployed locally, or if necessary, to the cloud in a Docker container. Nothing more is required.

The data collected by the related works include executed commands, textual CLI outputs, timestamps, directory locations, Internet Protocol (IP) addresses and host names. From these, Hardtrace does not record CLI outputs, host names, IP addresses, or the precise times when a command was executed. However, whereas the others are only able to obtain basic information on CLI interactions, Hardtrace is able to also provide data on files, processes and syscalls, providing a greater variety of information.

Like the systems in [25, 26], Hardtrace is also able to forward data to a centralized location, i.e. the API, for storage and analysis. Similarly, from there, the data can be accessed in real-time, enabling data analysis to be performed even while one or more Hardtrace instances are running.

An issue with the related works is that they can only track events occurring based on commands entered into the CLI. Thus, the usage of GUI tools or other actions occurring outside the CLI can not be tracked. Due to the nature of how Hardtrace works, it likely shares the same issue, although to what extent this issue is present was not examined in this thesis. It is possible that as long as the tools are launched from the CLI, Hardtrace is able to obtain partial information from them, e.g., what files the tools access. In comparison to the others, Hardtrace also has the additional disadvantage of only being able to collect data from the CLIs in which Hardtrace sessions have been started. Although, this can also be considered beneficial, as Hardtrace's users are able to perform actions elsewhere without being tracked, improving privacy. Lastly, the system implemented in this thesis has the disadvantage of causing more overhead to command execution due to Hardtrace.

Visualization

In this section, the visualizations produced by the implemented system's VA are compared with the visualizations done in [22, 25, 31, 32].

The main goal of the related visualizations was to help in assessing the work of the students and participants of CTF training sessions. On the other hand, the goal for visualization in this thesis was to allow forensic researchers to determine how exactly they or, e.g., their co-workers had previously performed digital forensic evidence examination and analysis. However, based on the case study's results, the system

implemented in this thesis could likely also work in the role of the CTF monitoring systems.

The related works [22, 25] created visualizations based on data similar to what the VA used. Like the VA, both works also used directed graphs as the visualization technique of choice for this type of data. Although, as [22, 25] only focused on commands, differences exist in how the graphs were used. The VA portrayed commands as the graph's edges and files as the nodes. In the related works, nodes represented commands and arguments. Edges indicated command chains and connections between commands and arguments.

In [22] Weiss et al. collapsed similar commands into one node. Similarly, edges representing similar commands were collapsed in this thesis to reduce clutter. Weiss et al. also noted that visualizations of this kind run the risk of losing some temporal information. In the case of commands that did not create any new files, this certainly holds true for the implemented VA, as it is difficult to say in which order commands represented by non-connected edges were executed in.

A different approach to visualizing how individuals fare while completing CTFs was used in [31, 32]. In these works, the data consisted of game events, such as flag submissions or taking hints. To present these data, bar charts, scatter plots, glyphs, and timelines were used as the visualization techniques. These techniques do not seem suitable for the way the current implementation of the VA presents participant actions. Furthermore, using these techniques to present accurate information on what was done during the digital forensics process does not seem feasible.

Compared with the VA, the visualizations in [31, 32] presented macro level details on participant progress, whereas the VA was able to presents micro level details thanks to Hardtrace data. Combining both levels of details into one could provide even more valuable knowledge. As the VA and the related works were all created with D3, this could be possible without too much trouble.

5.3. Risks to Experiment Result Reliability

The reliability of the experiment results and their analysis, presented in this thesis, are affected by different factors which should be taken into account when assessing their significance. This section discusses these factors.

The affecting factors are as follows:

- **Number of participants**: The conducted case study had 18 participants which is a fairly low number. This directly affected the amount of collected data, reducing the accuracy of all experiments except the visual analysis. Especially affected was the correlation calculated between active time usage and the number of executed commands, which would require more data points to have more significance.

- **Determining participant time usage**: Hardtrace does not record the exact time when a command was executed or when a file was accessed. Thus, when calculating time usages, timestamps automatically added by GORM in the *created_at* rows of Hardtrace's local DB were used. These timestamps represent

the time when Hardtrace saved an entry to the DB, possibly causing a slight skew in the related results.

- **Selected CTF**: The CTF, that case study participants had to complete, included only one level in which only two digital forensic tools had to be used. As such, it was only possible to observe the usage of these two tools, as well as, some basic commands available in the Bash shell. This limited the options for examining participant behaviors, and the number of possible commands with which to test Hardtrace's performance.

- **Use of CTF writeups**: It was possible that some case study participants used a writeup on the selected CTF in order to help complete the challenge. The possible utilization of writeups makes results, such as time usage, the number of executed commands and accessed files less realistic.

- **Usefulness and usability of the system**: While the experiments and analysis showed that the implemented system is able to provide lots of information on the progress of digital evidence examination and analysis in a timely manner, they did not tell how outside users would rate the system's usability and usefulness.

5.4. Improving the Experiments

This section discusses how the reliability and applicability of the analysis and experiment results could be further improved. Additionally, discussed is how the experiments could be further improved to gain more insight into what individuals do while solving a CTF.

Reliability and applicability can be improved with the following changes:

- **More participants**: The benefit of having more participants is clear. This would increase the amount of collected data, and thusly the accuracy and reliability of obtained results.

- **More variety in the CTF**: Increasing the variety of tasks participants need to complete, by switching to a multi-level CTF, e.g., would allow for data to be collected on the use of more digital forensic tools, commands, and on overall user behaviours when completing different challenges. Having a larger set of executed commands to test would provide a greater view into Hardtrace's performance.

- **Improved time logging**: Having a way to log the exact time of file access and command execution would make participant time usage information more precise.

- **Surveying participant skill-levels**: Before participating in a case study, participants could fill out a form surveying how much experience they have with the topics related to the selected CTF. The survey results could then be compared with the observations made on the progress of participants, possibly affirming conclusions.

- **Determine participant opinions**: In order to figure out what others think of the usability and usefulness of the system, implemented in this thesis, case study participants could be surveyed on, e.g., how Hardtrace, in their opinion, affected completing the CTF. The generated graphs could also be shown to participants to determine whether or not they find the visualization clear and understandable, etc.

5.5. Future Work

The use case for the system presented in this thesis was to help forensic researchers determine, at a later time, what was done during the examination and analysis stages of digital forensics. However, the system shows promise for other purposes as well. As an example of this, based on the case study, the system could be used in the future to assist the tutors of cybersecurity training events in their work. By using the combination of Hardtrace, the API and the VA, tutors could get a clear view into how each participant is advancing, helping detect struggling individuals so that they can be provided with the assistance they need. In this case, the system could also be used to illustrate example challenge solutions to participants.

To be better suited for various use cases, the system could be further improved in different ways. At the moment, the only information given on files by the visualization is their name from the time they were first accessed. To make the visualization more informative, popups displaying additional information on files could be made to appear when the user hovers their pointer on top of a node. The popup could include, e.g., the file's SHA-256 hash and any different names the file has had during the Hardtrace session.

Another view could be added to the VA's dashboard. From there the user could see the statistics presented in the statistical data analysis. Additionally, some of the macro level details visualized in [31, 32] could be present in this new view. Combining the macro and micro level details could proof very useful. This would, however, require a way to detect game events to be added to the system. To present statistics from the data, the API's capabilities would need to be extended.

In the current implementation, only individuals with the specific visualization credentials are able to see the generated graphs. It would likely be a beneficial addition to allow each Hardtrace user to access the graphs generated from their own data. The visualization could also be made to update in real time.

As mentioned earlier, this thesis did not examine to what extent Hardtrace can be used to track GUI tools. Thus, future research is needed to determine how well Hardtrace pairs with such tools. Furthermore, in future research Hardtrace data could be augmented with usage logs generated by different forensic tools to see if it would bring further benefits to the reconstruction of digital forensics progression. However, this would likely require the addition of logging functionalities to the forensic tools used in the studies.

6. CONCLUSION

This thesis examined how data detailing the progression of the examination and analysis digital forensics stages could be collected from multiple users, how the data could be transformed into summaries that detail what was done during these stages, and finally, how the data could be visualized. The overall goal was to enable the programmatic reconstruction of the work done during the two digital forensics stages.

To enable data collection, the existing implementation of the experimental Hardtrace was updated to submit collected data to a central location in the cloud. An API handling the storage of data from multiple Hardtrace users was created. From the API, two different types of summarized Hardtrace data could be requested. One type supported further analysis, the other supported easy visualization. To make deploying the API to the cloud simple, the API was containerized with Docker. A VA capable of visualizing the summaries as directed graphs, utilizing the D3 visualization framework, was implemented. Updates to Hardtrace, and the design and implementation of the API and VA were done according to specific requirements set for each of them.

The presented system's ability to serve as the answer to the research questions was tested through a case study in which 18 participants completed a memory forensics CTF challenge while using Hardtrace. Collected Hardtrace data was then used in two tests. In the first test, data from three different participants were visually analyzed to determine if the steps they took while solving the challenge could be deduced from the visualization. In the second test, data from all participants were used for statistical analysis examining their time and command usage, etc. Additionally, the system's execution times were observed during performance testing.

By performing the visual analysis utilizing the VA, determining what each participant did during their Hardtrace session was simple. While the produced visualizations were already easy to understand, any irksome understandability reducing clutter was straightforward to remove using the application's dashboard. The visual analysis showed that the implemented system was able to convey to its user, through visualization, what was done during digital evidence examination and analysis.

The summary design made performing statistical analysis of the Hardtrace data effortless. Analysis results demonstrated that a large quantity of information on the work of a group of users can be extracted from the summaries created by the API. These results are able to deepen the understanding gained from visual analysis on how the digital forensics process was carried out.

Even though Hardtrace previously suffered from large execution time overheads, the updates done in this thesis alleviated this issue significantly, as shown by the performance results. However, it was shown that Hardtrace can still multiply execution times. Although, the increases are much more tolerable. Thus, the performance testing results in tandem with the usefulness of data produced by Hardtrace validate the taken approach as a viable choice for collecting data. Summarizing data with the API and creating visualizations with the VA were also shown to take very little time, making them suitable for examining the collected data.

In summary, the implemented system was able to perform the tasks of data collection, digital forensics progress reconstruction and visualization well.

7. REFERENCES

[1] Raghavan S. (2013) Digital forensic research: current state of the art. CSI Transactions on ICT 1, pp. 91–114.

[2] Prayudi Y. & Sn A. (2015) Digital chain of custody: State of the art. International Journal of Computer Applications 114.

[3] Giova G. (2011) Improving chain of custody in forensic investigation of electronic digital systems. International Journal of Computer Science and Network Security 11, pp. 1–9.

[4] Kaksonen R., Järvenpää T., Pajukangas J., Mahalean M. & Röning J. (2021) 100 popular open-source infosec tools. In: IFIP International Conference on ICT Systems Security and Privacy Protection, Springer, pp. 181–195.

[5] Vaughan J., Baker N. & Underhill M. (2020), Digital forensic science strategy. URL: https://www.npcc.police.uk/DigitalForensicScienceStrategy2020.pdf. Accessed 26.5.2022.

[6] Bostock M., Ogievetsky V. & Heer J. (2011) D^3 data-driven documents. IEEE transactions on visualization and computer graphics 17, pp. 2301–2309.

[7] Hargreaves C. & Patterson J. (2012) An automated timeline reconstruction approach for digital forensic investigations. Digital Investigation 9, pp. S69–S79.

[8] Debinski M., Breitinger F. & Mohan P. (2019) Timeline2gui: A log2timeline csv parser and training scenarios. Digital Investigation 28, pp. 34–43.

[9] Shosha A.F., Tobin L. & Gladyshev P. (2013) Digital forensic reconstruction of a program action. In: 2013 IEEE Security and Privacy Workshops, IEEE, pp. 119–122.

[10] Schelkoph D.J., Peterson G.L. & Okolica J.S. (2018) Digital forensics event graph reconstruction. In: International Conference on Digital Forensics and Cyber Crime, Springer, pp. 185–203.

[11] Grajeda C., Breitinger F. & Baggili I. (2017) Availability of datasets for digital forensics–and what is missing. Digital Investigation 22, pp. S94–S105.

[12] Kim G.H. & Spafford E.H. (1994) The design and implementation of tripwire: A file system integrity checker. In: Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 18–29.

[13] Mehnaz S. & Bertino E. (2017) Ghostbuster: A fine-grained approach for anomaly detection in file system accesses. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 3–14.

[14] Stolfo S.J., Hershkop S., Bui L.H., Ferster R. & Wang K. (2005) Anomaly detection in computer security and an application to file system accesses. In: International Symposium on Methodologies for Intelligent Systems, Springer, pp. 14–28.

[15] Ansari M.S.A., Chattopadhayay A. & Das S. (2010) A kernel level vfs logger for building efficient file system intrusion detection system. In: 2010 Second International Conference on Computer and Network Technology, IEEE, pp. 273–279.

[16] Xia Y., Fairbanks K. & Owen H. (2008) A program behavior matching architecture for probabilistic file system forensics. ACM SIGOPS Operating systems review 42, pp. 4–13.

[17] ptrace(2) — linux manual page. URL: https://man7.org/linux/man-pages/man2/ptrace.2.html. Accessed 27.10.2021.

[18] Shameer S. (2021), 8 popular and lesser know linux terminal recording tools. URL: https://linoxide.com/linux-terminal-recording-tools/. Accessed 27.10.2021.

[19] Baldescu D. (2020), Top 5 linux shells and how to install them. URL: https://bigstep.com/blog/top-5-linux-shells-and-how-to-install-them. Accessed 28.10.2021.

[20] Hance J. & Straub J. (2020) Use of bash history novelty detection for identification of similar source attack generation. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, pp. 759–766.

[21] Lane T. & Brodley C.E. (1999) Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security (TISSEC) 2, pp. 295–331.

[22] Weiss R., Locasto M.E. & Mache J. (2016) A reflective approach to assessing student performance in cybersecurity exercises. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp. 597–602.

[23] Weiss R., Turbak F., Mache J. & Locasto M.E. (2017) Cybersecurity education and assessment in edurange. IEEE Security & Privacy 15, pp. 90–95.

[24] Lee W., Stolfo S.J. & Mok K.W. (1999) A data mining framework for building intrusion detection models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344), IEEE, pp. 120–132.

[25] Mirkovic J., Aggarwal A., Weinman D., Lepe P., Mache J. & Weiss R. (2020) Using terminal histories to monitor student progress on hands-on exercises. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 866–872.

[26] Švábenskỳ V., Vykopal J., Tovarňák D. & Čeleda P. (2021) Toolset for collecting shell commands and its application in hands-on cybersecurity training. IEEE.

[27] McGranaghan M. & Bendersky E., Go by example: Goroutines. URL: https://gobyexample.com/goroutines. Accessed 27.10.2021.

[28] Silberschatz A., Galvin P.B. & Gagne G. (2012) Operating System Concepts, 9th Edition. Wiley.

[29] What is capture the flag? URL: https://ctfd.io/whats-a-ctf/. Accessed 27.10.2021.

[30] What is capture the flag? URL: https://ctftime.org/ctf-wtf/. Accessed 27.10.2021.

[31] Ošlejšek R., Rusňák V., Burská K., Švábenskỳ V. & Vykopal J. (2019) Visual feedback for players of multi-level capture the flag games: Field usability study. In: 2019 IEEE Symposium on Visualization for Cyber Security (VizSec), IEEE, pp. 1–11.

[32] Burská K.D., Rusňák V. & Ošlejšek R. (2021) Enhancing situational awareness for tutors of cybersecurity capture the flag games. In: 2021 25th International Conference Information Visualisation (IV), IEEE, pp. 235–242.

[33] Khan M. & Khan S.S. (2011) Data and information visualization methods, and interactive mechanisms: A survey. International Journal of Computer Applications 34, pp. 1–14.

[34] Chen C. (2010) Information visualization. Wiley Interdisciplinary Reviews: Computational Statistics 2, pp. 387–403.

[35] Liu S., Cui W., Wu Y. & Liu M. (2014) A survey on information visualization: recent advances and challenges. The Visual Computer 30, pp. 1373–1393.

[36] Chen W., Guo F. & Wang F.Y. (2015) A survey of traffic data visualization. IEEE Transactions on Intelligent Transportation Systems 16, pp. 2970–2984.

[37] Guimaraes V.T., Freitas C.M.D.S., Sadre R., Tarouco L.M.R. & Granville L.Z. (2015) A survey on information visualization for network and service management. IEEE Communications Surveys & Tutorials 18, pp. 285–323.

[38] Shiravi H., Shiravi A. & Ghorbani A.A. (2011) A survey of visualization systems for network security. IEEE Transactions on visualization and computer graphics 18, pp. 1313–1329.

[39] Komarek A., Pavlik J. & Sobeslav V. (2015) Network visualization survey. In: Computational Collective Intelligence, Springer, pp. 275–284.

[40] Herman I., Melançon G. & Marshall M.S. (2000) Graph visualization and navigation in information visualization: A survey. IEEE Transactions on visualization and computer graphics 6, pp. 24–43.

[41] Beck F., Burch M., Diehl S. & Weiskopf D. (2017) A taxonomy and survey of dynamic graph visualization. In: Computer Graphics Forum, vol. 36, Wiley Online Library, vol. 36, pp. 133–159.

[42] Elias M. & Bezerianos A. (2011) Exploration views: understanding dashboard creation and customization for visualization novices. In: IFIP conference on human-computer interaction, Springer, pp. 274–291.

[43] Pappas L. & Whitman L. (2011) Riding the technology wave: Effective dashboard data visualization. In: Symposium on Human Interface, Springer, pp. 249–258.

[44] Sarikaya A., Correll M., Bartram L., Tory M. & Fisher D. (2018) What do we talk about when we talk about dashboards? IEEE transactions on visualization and computer graphics 25, pp. 682–692.

[45] Toasa R., Maximiano M., Reis C. & Guevara D. (2018) Data visualization techniques for real-time information—a custom and dynamic dashboard for analyzing surveys' results. In: 2018 13th Iberian Conference on Information Systems and Technologies (CISTI), IEEE, pp. 1–7.

[46] Shneiderman B. (2003) The eyes have it: A task by data type taxonomy for information visualizations. In: The craft of information visualization, Elsevier, pp. 364–371.

[47] Sedrakyan G., Mannens E. & Verbert K. (2019) Guiding the choice of learning dashboard visualizations: Linking dashboard design and data visualization concepts. Journal of Computer Languages 50, pp. 19–38.

[48] Rimal B.P., Choi E. & Lumb I. (2009) A taxonomy and survey of cloud computing systems. In: 2009 Fifth International Joint Conference on INC, IMS and IDC, Ieee, pp. 44–51.

[49] Mell P. & Grance T. (2011) The NIST definition of cloud computing.

[50] Armbrust M., Fox A., Griffith R., Joseph A.D., Katz R., Konwinski A., Lee G., Patterson D., Rabkin A., Stoica I. et al. (2010) A view of cloud computing. Communications of the ACM 53, pp. 50–58.

[51] Bachiega N.G., Souza P.S., Bruschi S.M. & De Souza S.D.R. (2018) Container-based performance evaluation: A survey and challenges. In: 2018 IEEE International Conference on Cloud Engineering (IC2E), IEEE, pp. 398–403.

[52] da Silva V.G., Kirikova M. & Alksnis G. (2018) Containers for virtualization: An overview. Appl. Comput. Syst. 23, pp. 21–27.

[53] Leach P., Mealling M. & Salz R. (2005) A universally unique identifier (uuid) urn namespace. Accessed 23.05.2022.

[54] Kumar P.A. (2020), Memlabs lab 3 - the evil's den. URL: https://github.com/stuxnet999/MemLabs/tree/master/Lab3. Accessed 18.10.2021.

[55] Vykopal J., Švábenskỳ V. & Chang E.C. (2020) Benefits and pitfalls of using capture the flag games in university courses. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 752–758.

[56] Owens K., Fulton A., Jones L. & Carlisle M. (2019) pico-boo!: How to avoid scaring students away in a ctf competition.

[57] Wilford G., Polacco F. & Watson C., Ubuntu manpage: mandb - create or update the manual page index caches. URL: https://manpages.ubuntu.com/manpages/trusty/man8/mandb.8.html. Accessed 22.1.2021.

[58] Volatility framework - volatile memory extraction utility framework. URL: https://github.com/volatilityfoundation/volatility. Accessed 7.2.2022.

8. APPENDICES

```
[
    {
        "pseudonym": "user1",
        "session": "test",
        "process_summary": [
            {
                "cmd": "nano file1.txt",
                "cmd_type": "WRITE",
                "run_at": "2021-06-16 17:04:07",
                "old_files": [],
                "new_files": [
                    {
                        "created_at": "2021-06-16 17:04:13",
                        "filepath": "file1.txt",
                        "pipe": false,
                        "inode": 14946628,
                        "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
                    }
                ]
            },
            {
                "cmd": "nano file1.txt",
                "cmd_type": "WRITE",
                "run_at": "2021-06-16 17:04:17",
                "old_files": [
                    {
                        "created_at": "2021-06-16 17:04:17",
                        "filepath": "file1.txt",
                        "pipe": false,
                        "inode": 14946628,
                        "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
                    },
                    {
                        "created_at": "2021-06-16 17:04:17",
                        "filepath": "file1.txt",
                        "pipe": false,
                        "inode": 14946628,
                        "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
                    }
                ],
                "new_files": [
                    {
                        "created_at": "2021-06-16 17:04:47",
                        "filepath": "file1-modified.txt",
                        "pipe": false,
                        "inode": 14946646,
                        "hash": "b22b009134622b6508d756f1062455d71a7026594eacb0badf81f4f677929ebe"
                    }
                ]
            },
            {
                "cmd": "cat file1-modified.txt",
                "cmd_type": "READ",
                "run_at": "2021-06-16 17:04:55",
                "old_files": [
                    {
                        "created_at": "2021-06-16 17:04:47",
                        "filepath": "file1-modified.txt",
                        "pipe": false,
                        "inode": 14946646,
                        "hash": "b22b009134622b6508d756f1062455d71a7026594eacb0badf81f4f677929ebe"
                    }
                ],
                "new_files": []
            }
        ]
    }
]
```


Figure 21. Example summary of a Hardtrace user's session.

```
[
    {
        "pseudonym": "user1",
        "session": "test",
        "process_summary": [
            {
                "cmd": "nano file1.txt",
                "cmd_type": "WRITE",
                "run_at": "2021-06-16 17:04:07",
                "old_files": [],
                "new_files": [
                    {
                        "created_at": "2021-06-16 17:04:13",
                        "filepath": "file1.txt",
                        "pipe": false,
                        "inode": 14946628,
                        "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
                    }
                ]
            },
            {
                "cmd": "nano file1.txt",
                "cmd_type": "WRITE",
                "run_at": "2021-06-16 17:04:17",
                "old_files": [
                    {
                        "created_at": "2021-06-16 17:04:17",
                        "filepath": "file1.txt",
                        "pipe": false,
                        "inode": 14946628,
                        "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
                    },
                    {
                        "created_at": "2021-06-16 17:04:17",
                        "filepath": "file1.txt",
                        "pipe": false,
                        "inode": 14946628,
                        "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326"
                    }
                ],
                "new_files": [
                    {
                        "created_at": "2021-06-16 17:04:47",
                        "filepath": "file1-modified.txt",
                        "pipe": false,
                        "inode": 14946646,
                        "hash": "b22b009134622b6508d756f1062455d71a7026594eacb0badf81f4f677929ebe"
                    }
                ]
            }
        ]
    }
]
```


Figure 22. Example summary on how the file titled file1-modified.txt was created.

```
{
    "nodes": [
        {
            "id": 0,
            "name": "file1.txt",
            "created_at": "2021-06-16 17:04:13",
            "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326",
            "inode": 14946628,
            "source_node": true,
            "selected_file": false
        },
        {
            "id": 1,
            "name": "file1-modified.txt",
            "created_at": "2021-06-16 17:04:47",
            "hash": "b22b009134622b6508d756f1062455d71a7026594eacb0badf81f4f677929ebe",
            "inode": 14946646,
            "source_node": false,
            "selected_file": false
        },
        {
            "id": 2,
            "name": null,
            "created_at": null,
            "hash": null,
            "inode": null,
            "source_node": false,
            "selected_file": false
        },
        {
            "id": 3,
            "name": null,
            "created_at": null,
            "hash": null,
            "inode": null,
            "source_node": false,
            "selected_file": false
        }
    ],
    "edges": [
        {
            "cmd": "nano file1.txt",
            "short_cmd": "nano",
            "source": 2,
            "target": 0,
            "run_by": "user1"
        },
        {
            "cmd": "nano file1.txt",
            "short_cmd": "nano",
            "source": 0,
            "target": 1,
            "run_by": "user1"
        },
        {
            "cmd": "cat file1-modified.txt",
            "short_cmd": "cat",
            "source": 1,
            "target": 3,
            "run_by": "user1"
        }
    ]
}
```


Figure 23. Example graph data of a Hardtrace user's session.

```
{
    "nodes": [
        {
            "id": 0,
            "name": "file1.txt",
            "created_at": "2021-06-16 17:04:13",
            "hash": "1c5b885943f571434f0396da0cb5c2aa16cc546d16e679db2a6eea65bfe2b326",
            "inode": 14946628,
            "source_node": true,
            "selected_file": true
        },
        {
            "id": 1,
            "name": "file1-modified.txt",
            "created_at": "2021-06-16 17:04:47",
            "hash": "b22b009134622b6508d756f1062455d71a7026594eacb0badf81f4f677929ebe",
            "inode": 14946646,
            "source_node": false,
            "selected_file": true
        },
        {
            "id": 2,
            "name": null,
            "created_at": null,
            "hash": null,
            "inode": null,
            "source_node": false,
            "selected_file": false
        }
    ],
    "edges": [
        {
            "cmd": "nano file1.txt",
            "short_cmd": "nano",
            "source": 2,
            "target": 0,
            "run_by": "user1"
        },
        {
            "cmd": "nano file1.txt",
            "short_cmd": "nano",
            "source": 0,
            "target": 1,
            "run_by": "user1"
        }
    ]
}
```


Figure 24. Example graph data on how the file titled file1-modified.txt was created.

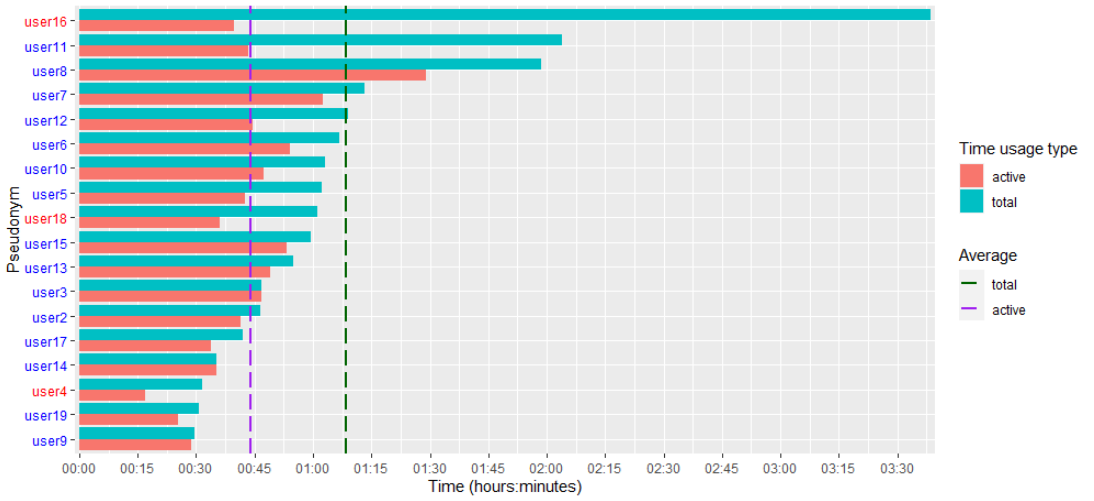


Figure 25. Overall and active time usage of each individual participant. Red pseudonyms indicate participants who did not complete the CTF.

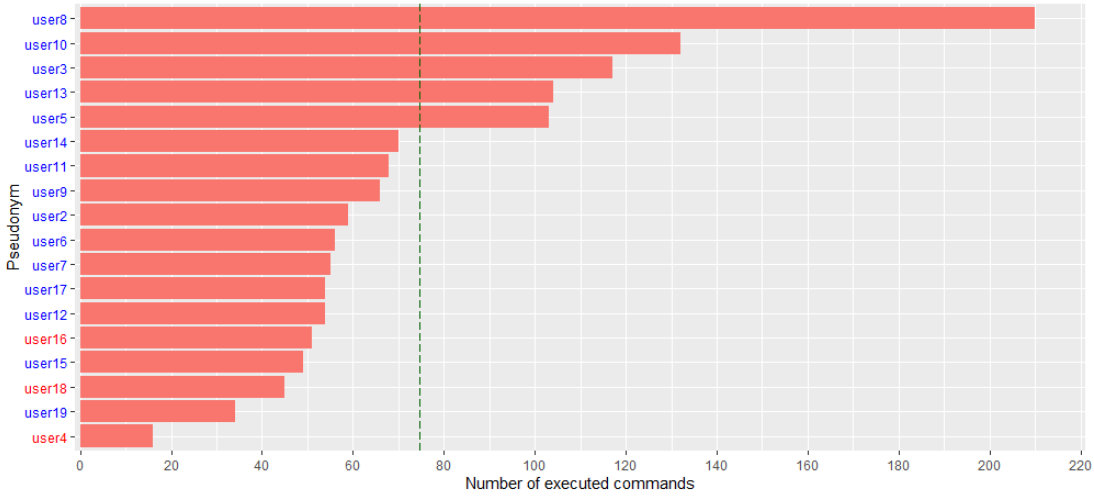


Figure 26. Number of commands run by each individual participant. Red pseudonyms indicate participants who did not complete the CTF.

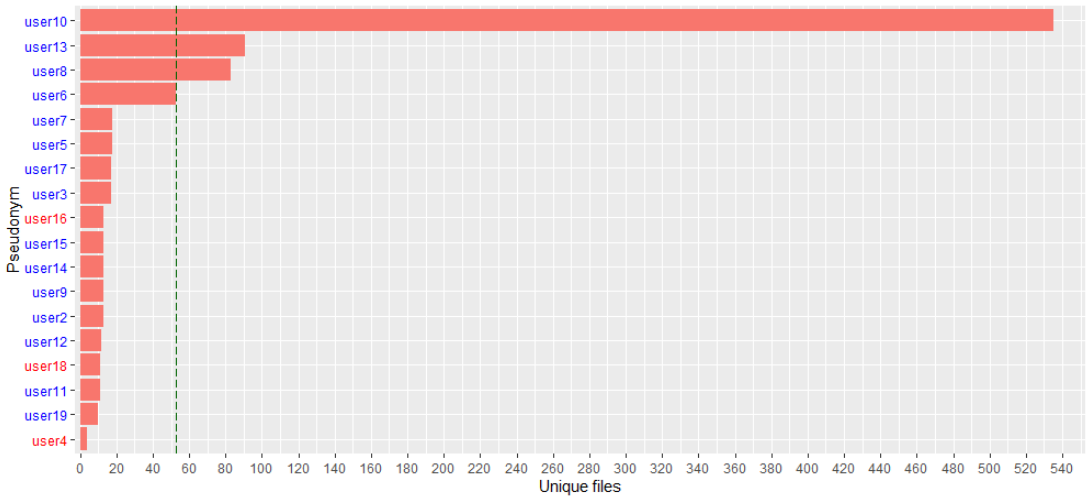


Figure 27. Number of accessed files for each individual participant. Red pseudonyms indicate participants who did not complete the CTF.

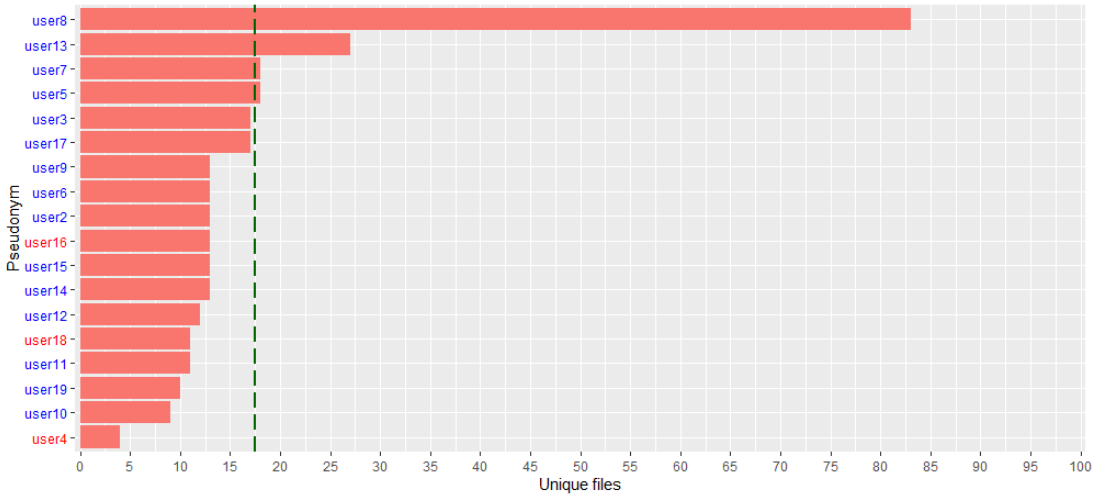


Figure 28. Number of accessed files for each participant with unused files removed. Red pseudonyms indicate participants who did not complete the CTF.

	Execution times (s)				Saved system calls	
		Hardtrace version			Hardtrace Version	
Command	Without	Old	New (offline)	New (online)	Old	New
volatility -f MemDump.raw imageinfo	18.9	68	25.6	25.9	10948	9
volatility -f MemDump.raw verinfo	5.6	8	5.9	5.4	972	4
volatility -f MemDump.raw –profile=Win7SP0x86 filescan > files.txt	18.7	5293	53	55.5	806903	7
volatility -f MemDump.raw –profile=Win7SP0x86 filescan	17.1	5551	56.7	56.8	806900	4
volatility -f MemDump.raw –profile=Win7SP0x86 iehistory	12.6	10721	105.9	111.2	1601684	4
volatility -f MemDump.raw –profile=Win7SP0x86 pstree	2.6	160	8.1	8.4	23527	4
volatility -f MemDump.raw --profile=Win7SP0x86 pslist	2.6	105	7.6	7.9	15611	4
volatility -f MemDump.raw –profile=Win7SP0x86 psscan	1.7	3	2.3	2.4	263	4
volatility -f MemDump.raw --profile=Win7SP0x86 envars	9.3	12497	123.3	129.4	1856155	4
volatility -f MemDump.raw –profile=Win7SP0x86 windows	9.9	420	35.5	37.2	62939	6
volatility -f MemDump.raw –profile=Win7SP0x86 cmdscan	3	678	13	13.7	101323	4
volatility -f MemDump.raw --profile=Win7SP0x86 cmdline	2.6	119	7.6	8.2	17479	4
volatility -f MemDump.raw --profile=Win7SP0x86 cmdline -p 3736 ---dump-dir=files	2.6	76	7.3	9.3	11209	7
volatility -f MemDump.raw --profile=Win7SP0x86 cmdline -p 3432 –output-file=files/notepad.txt	2.6	76	7.4	9.3	11209	7
volatility -f MemDump.raw --profile=Win7SP0x86 memdump -p 3736 –dump-dir=files	5.7	3855	44.5	52.8	574893	7
volatility -f MemDump.raw --profile=Win7SP0x86 memdump -p 3432 –dump-dir=files	5.7	3855	44.6	46.5	574659	7
volatility -f MmeDump.raw --profile=Win7SP0x86 screenshot --dump-dir=files	10.1	548	36.6	38.3	81812	34
volatility -f MemDump.raw --profile=Win7SP0x86 dumpfiles -Q 0x000000003de1b5f0 --dump-dir files/ -n	1.3	5	1.9	2	574	7
volatility -f MemDump.raw --profile=Win7SP0x86 dumpfiles -Q 0x000000003e727e50 --dump-dir files/ -n	1.4	95	2.8	2.8	14001	7
volatility -f MemDump.raw --profile=Win7SP0x86 dumpfiles -Q 0x000000004f34148 --dump-dir files/ -n	1.4	95	2.7	2.8	14055	7
steghide info suspision1.jpeg -p inctf{0n3_h4lf	0	0	0	0	8	5
steghide extract --force -sf suspision1.jpeg -p inctf{0n3_h4lf -xf files/key2.txt	0	0	0	0	9	6
grep -E 'evilscript.py' test-files/files.txt	0	1	0.1	0.1	4	4
grep -E '*.png' test-files/files.txt	0	0	0	0	28	4
file MemDump.raw	0	0	0	0.1	3	3
base64 --decode vip.txt	0	0	0	0	3	3
strings suspision1.jpeg	0	0	0	0	6	3

Figure 29. Command execution times without Hardtrace and with the new and old Hardtrace version. Also includes the number of system calls saved to the local DB by new and old Hardtrace.

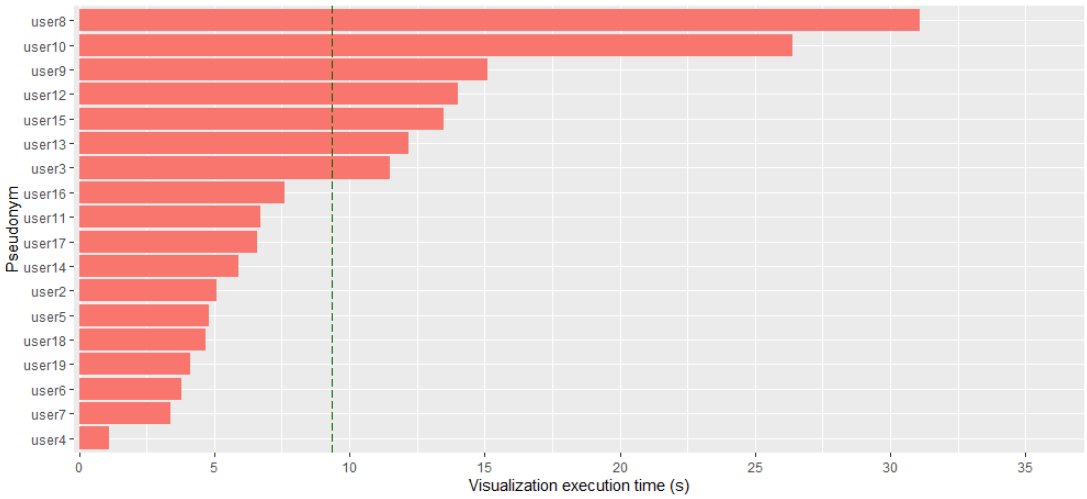


Figure 30. Time taken to visualize each participant's graph.