Politecnico di Torino

Politecnico di Torino

Master's Degree in Cybersecurity

A.a. 2025/2026

Graduation Session March 2026

CyberForensics Arena

A Gamified Serious Game for Digital Forensics Education

Advisor:
Andrea Atzeni

Candidate:
Edoardo Maniero

Abstract

Digital forensics education faces significant challenges, characterized by steep learning curves, resource-intensive laboratory setups, and a lack of student engagement with traditional text-based exercises. This thesis presents the design, development, and evaluation of CyberForensics Arena, a web-based serious game designed to bridge the gap between theoretical knowledge and practical investigative skills. The platform immerses learners in a realistic 3D forensic laboratory, where they assume the role of an investigator solving complex cybercrimes.

Unlike traditional training environments that rely on static quizzes or abstract command-line interfaces, CyberForensics Arena integrates an interactive 3D environment with a fully simulated Linux terminal. This hybrid approach enables students to perform authentic forensic procedures, such as disk imaging, memory analysis, and network packet inspection, within a cohesive narrative context. The system features three progressively difficult scenarios based on real-world threat intelligence: a file system investigation involving data exfiltration, a network intrusion analysis of an APT attack, and a memory forensics case dealing with fileless malware.

From a technical perspective, the platform introduces a novel architecture for "validation without exposure," ensuring that client-side game mechanics do not compromise the integrity of the investigative challenges. Furthermore, a comprehensive telemetry system captures granular user interaction data, from command execution attempts to varying solution paths, enabling detailed learning analytics. The platform's effectiveness is evaluated through a pilot study involving participants with diverse technical backgrounds, assessing usability, engagement, and learning outcomes. The results demonstrate that gamified, situated learning environments can significantly enhance the acquisition of digital forensic competencies while maintaining high levels of user motivation.

Acknowledgements

Desidero innanzitutto ringraziare il mio relatore, per la guida, il supporto e i preziosi consigli fornitimi durante tutto lo sviluppo di questo lavoro di tesi.

Un Ringraziamento Speciale a Mamma e Papà, da sempre al mio fianco anche nei momenti più difficili, per avermi dato la possibilità di seguire i miei sogni e per aver creduto sempre in me.

Un grazie a Matteo, per avermi sostenuto e ascoltato come solo lui sa fare.

A special Thanks to Hannah, for always supporting me, and pushing me to be better. For always believing in me, even when I myself doubted. Thank you for your patience during the long days and late nights spent on this thesis, for celebrating every small milestone, and for reminding me who I am even in the toughest moments.

Infine, ringrazio i miei amici e i miei colleghi di corso, con i quali ho condiviso le fatiche e le gioie di questi anni universitari, per il loro supporto morale e per i tanti momenti passati insieme.

Table of Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

1.1 Cybersecurity and Digital Forensics: Context and Motivation

Nowadays, with the increase in digital devices, used in nearly every aspect of modern life, from the most basic tasks to the more complex ones, we can say that we have entered the Digital Era. Every human being, whether he likes it or not, depends on these digital services and infrastructures, and with the increase of these, new risks are being introduced: numerous cyberattacks, data breaches, identity theft and digital crimes keep happening every day.

In this perspective, Cybersecurity and Digital Forensic are every day more important in order to mitigate these threats, protect information and investigate incidents. In particular, Digital Forensic supports incident response by reconstructing timelines and correlating artifacts across systems, logs and other traces. Here the investigator is not limited in running tools and executing commands, but he has to create hypotheses, validate them against the available evidence and iteratively polishing them as new information and evidence emerges. Then the goal is to produce a coherent and supported explanation of what happened, when it happened and how it unfolded.

Teaching Digital Forensics nowadays could be hard:

- the learning curve is too steep: there's the need of a wide base knowledge (Linux, tooling, logs, evidence based reasoning);
- labs are too expensive to set up and to maintain, and sometimes it is unsafe to run malware in them;
- students tend to "follow steps" without understanding investigative reasoning;

- students experience low engagement when exercises feel abstract or purely text-based.

The application of serious games and gamification elements can prove to be game-changers in the field of Digital Forensics.

Through a combination of learning environments and game concepts, it is possible to achieve a learning platform where students and professionals can learn, test, and master their analytical capabilities in digital investigation. The application of such methodologies is not only an effective way to gain interest and motivation during learning, but it will allow them to work on problem-solving and experimentation activities. Additionally, this will enable them to work in a safe environment with a controlled space where they can work on tests without any risks of damage in a production environment.

This is why I have developed a new platform, called **Cyberforensics Arena**, where the user takes the role of a Forensic Investigator, and is placed inside an immersive 3D Forensic Office, where he has to interact with the environment and use a simulated Linux console in order to replicate the investigation workflow.

1.2 Problem Statement and Thesis Objectives

Based on what presented in Section 1.1, this thesis focuses on developing a solution for teaching digital forensics productively in classrooms with the help of Gamification and Serious Games. The students, instead of simply running tools, will have to formulate and test hypotheses, interpret traces, and piece together the available evidence, and find logical and supported conclusions.

But delivering this throughout the entire course means designing exercises that can be reused, followed step by step, and tracked for progress, without losing accuracy when grading.

The objective is to develop and build **CyberForensics Arena**, a Serious-Game platform for hands-on digital investigation exercises. With the use of progressive tasks and scenarios, the final user is guided through the field of Digital Forensics and Forensics Investigations.

The platform is intended to support:

- guided progression and feedback;
- persistent tracking of the user's progress and performance;
- validation techniques that avoid exposing solutions and hints to the client;
- a mechanism to let teachers create scenarios and tasks based on their goal.

The work is structured around three main objectives:

- **O1 (Learning Design):**Design learning scenarios that map digital forensics learning objectives to game tasks and mechanics that promote *investigative reasoning* rather than *step-following*.
- **O2 (Platform Development):** Develop a deployable serious-game platform with progression tracking, server-side validation, scenario management, and data collection capabilities suitable for university deployment.
- **O3 (Evaluation):**Evaluate the platform's usability, engagement, and educational effectiveness through a pilot study combining interaction logs and user questionnaires.

Each objective is addressed in several chapters. O1 is covered in Chapters 3 and 4, where learning objectives are related to game design elements and scenario structure. O2 is covered in Chapters 4 and 5, where the platform architecture and implementation are described. O3 is covered in Chapter 5, where a pilot evaluation is conducted and results are discussed.

1.3 Scope of the Work and Intended Audience

The final platform is developed within the context of a Master's degree in Cybersecurity and it is supposed to support teaching activities (also outside the academic field).

Intended audience: The target audience for this is MSc students in cybersecurity, as they require hands-on experience in the steps to be followed during investigation and arriving at conclusions based on the evidence obtained. It is targeted at people who are comfortable working within an operating system, able to run basic terminal commands, yet takes them through each step with handy hints along the way. The instructors and teaching assistants are another target audience, as it provides them with well-structured labs that fit within their teaching hours.

Self-learners and professionals might find it useful too, especially if they want organized drills; even then, classroom setup is what it's mostly used for.

Scope of the work: The study looks at these points:

- the definition of clear training objectives, and scenarios with tasks that can reflect a real Forensic Investigation;
- a learning space where people dive into the topic step by step, move ahead with help along the way, get responses as they go;
- gamification elements in order to enhance and make overall better the experience;

- ways to follow progress and results so teachers can check how students are doing plus understand their learning patterns.

Out of scope: The thesis is not meant to replace professional Forensics tools nor to provide a complete teaching plan for digital forensics. It does not claim legal admissibility of produced results or full chain-of-custody compliance for real world court proceedings. Instead the system focuses on safe learning: no live malware runs the performance of risky action in real machine is needed. Large-scale rollouts aren't part of this early version, and they are discussed as limitations and future work further in the thesis.

Assumptions and constraints: Since it's meant for guided learning settings, situations plus tools are picked on purpose to fit what students should learn and how hard it should feel. The final evaluation is performed minding practical constraints of an academic project, and conclusions will be interpreted based on those conditions.

1.4 Methodological Approach and Thesis Organization

The thesis follows a design and develop approach, which culminates in a designed, implemented and evaluated educational platform that supports digital forensic training. There are three primary stages the work takes.

The educational and technical requisites are first profiled according to the target audience, learning objectives and constraints of deploying hands-on work in an academic context. From these requirements, we can extract the platform's essential features and we can also use them as guides for the development of tasks and scenarios.

Second, the requirements are turned into a game & platform design. The learning objectives are mapped with tasks, mechanics, and feedback strategies that the notions of progressive difficulty and evidence-based reasoning. The design outcome has been developed into a fully-working prototype, combining the interactive learning environment with features related to user progression, validation and data collection. Gamifications elements were also planned in order to increase the engagement for the final user, and making the whole experience less boring and more compelling. Some attentions were also given for the teacher usability point of view: it should be easy for them to modify the content of the tasks and scenarios in order to use it for their purposes.

Third, after the implementation was done, the evaluation phase took tests. A mix of qualitative feedback (questionnaire) and quantitative data (extracted from logs) were collected and analyzed.

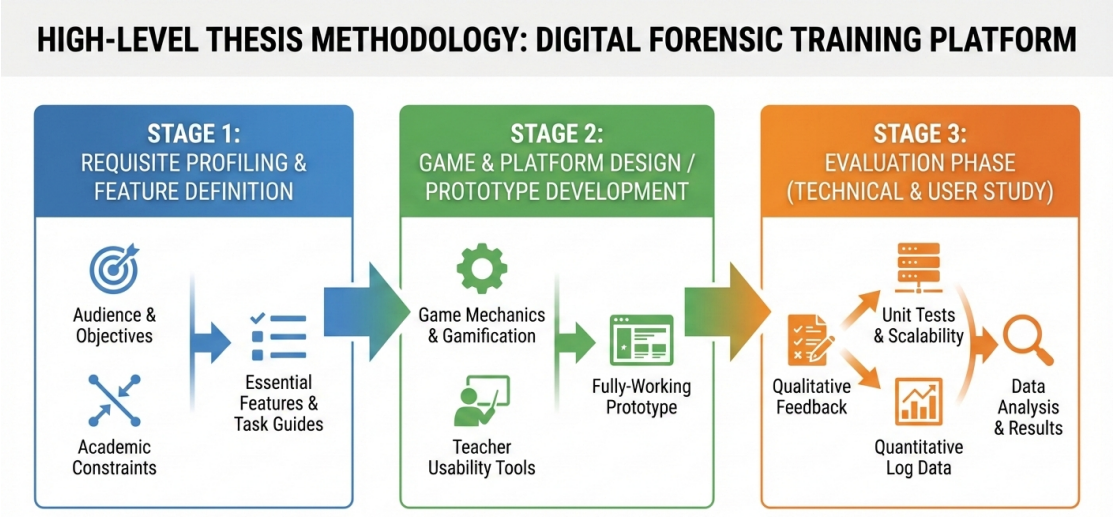

Figure 1.1: Design and Develop Approach: from Requirements to Evaluated Educational Platform

The thesis is structured as follows: Chapter 2 will discuss the theoretical foundation in cybersecurity, digital forensics, and gamification-based learning. Chapter 3 will describe the project and game design of CyberForensics Arena, including the mapping of learning goals to gamification elements and the scenario-based approach. Chapter 4 will describe the system architecture and implementation of the platform, including both client and server sides. Chapter 5 will outline the evaluation approach and results, including user feedback and learning analytics. Finally, Chapter 6 will recap the work completed, its implications, and future developments.

Chapter 2

Background: Cybersecurity, Digital Forensic, Serious Game and Gamification

2.1 Scope and Terminology

Chapter 2 describes the requirements and the background needed in order to give us a solid base for deriving the learning objectives and the requirements presented in the thesis, which will be mapped into tasks mechanics and feedback strategies, and then to understand the platform design and implementation decisions.

Additionally, this chapter is going to present the already existing platform to the problem space, in order to position CyberForensics Arena into the landscape of Cybersecurity Educational tools.

From the digital forensics point of view, the focus is centered on *post-incident investigation* and *evidence-based reasoning*, the process for which a forensic investigator make hypoteses and validates them against the available artifact found in the scene, and tries to reconstruct a coherent and reasonable narrative of events.

While the wide field of the Cybersecurity is introduced to give context to the types of incident Digital Forensics has to face, topics such as *intrusion detection*, *vulnerability assessment* and *penetration testing*, fall outside of the primary scope of this thesis.

For the educational perspective, the chapter focuses on game-level based learning approaches which have proven to be effective in technical skills development. The discussion is not going to extend to already e-learning existing paradigms, except where they intersect with gamified environments.

The target educational context remains university level instruction for students

who have a solid baseline of operative systems, and command line interfaces, as stated in section 1.3.

We are going to establish a shared vocabulary and a coherent baseline of the principles that will be used in the thesis.

Cybersecurity refers to the prevention of damage to, protection of, and restoration of computers, electronic communications systems, electronic communications services, wire communication, and electronic communication, including information contained therein, to ensure its availability, integrity, authentication, confidentiality, and nonrepudiation. [1]

Digital Forensic, in its strictest connotation, is the application of computer science and investigative procedures involving the examination of digital evidence, following proper search authority, chain of custody, validation with mathematics, use of validated tools, repeatability, reporting, and possibly expert testimony.[2]

Incident Response is the organized process of detecting, containing, eradicating and recovering from security breaches/event. NIST defines Incident Response as "the remediation or mitigation of violations of security policies and recommended practices." [3]

With **Digital Evidence**, we refer to any information of evidential value whether memorized or sent in a digital format, that may have probative value in an investigation. This includes file system artifacts, log entries, network captures, memory dumps and metadata.

Usually, Digital Evidence

- is **invisible** to the untrained eyes;
- needs to be **interpreted** by a specialist;
- can be **altered** or **destroyed** through normal use;
- can be **copied without limits**;

The legal requirements for digital evidence are:

- **Admissible**: compliant with law and best practices; an evidence can be inadmissible because the evidence is obtained by violating the rights of the owner for example.
- **Authentic**: any digital evidence tampering needs to be avoided, otherwise the evidence becomes unauthentic and not usable by a judge.
- **Reliable and Believable**: readily understandable for a judge.
- **Proportional**: respect fundamental rights of parties affected by the measure.

Chain of Custody is the process that tracks the movement of evidence through its collection, safeguarding, and analysis lifecycle by documenting each person who handled the evidence, the date/time it was collected or transferred, and the purpose for the transfer.[4].

Maintaining a safe and unbroken chain of custody gives us the guarantee that the digital evidence has not been altered or contaminated, and gives it the legal value, hence the possibility to be used with a probatory value during an investigation.

With **Serious Game**, we indicate a game designed with a primary goal that goes beyond the simple and pure entertainment, which is usually education, training, or behavioural change. The definition of Serious Games comes from Abt, who described Serious Games as "having an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement".[5]

Other researchers have sought to improve on Abt, proposing that Serious Games are defined as the integration of learning objectives with Game Design element.[6]

Cyberforensics Arena is positioned as a Serious Game where the entertainment component is used to enhance engagement and improve effectiveness in the training.

For what regards **Gamification**, we refer to the use of game design elements in a non-game context, in order to increase motivation, engagement, and participation. The definition by *Deterding et Al.* distinguishes serious game from gamification: "Gamification is the use of game elements in a non-game context to increase user engagement".[7]

Based on this definition, gamification differentiates itself from serious games because the former represents the application of game design elements, "in contrast to serious games, which are actual games used for a non-entertaining, usually serious, objective."[7]

An example of this concept, CyberForensics Arena, combines the definition of a serious game with the definition of gamification because it combines both a gaming (narrative, missions, challenges) and gamification elements (scoring, achievements, progression tracking).

As can be seen, digital forensics married with game-based learning has unique challenges of its own. Forensic examinations are, by definition, complex exercises in marrying understanding of technical issues (file systems, logging, networks, etc.) with understanding of procedures (how evidence is handled, documented, etc.) with an equally high-order understanding of causality, hypothesis construction, or correlating information in time-space history, etc. Somehow, these complexities need to be transferred into the world of game-based learning without it becoming so simplistic that important learnings are sacrificed in favor of playability or so complex that it overwhelms learners with an unforgiving or annoying level of intricacy.

2.2 Cybersecurity context needed for realistic forensic scenarios

Digital Forensics educational scenarios can be effective only when they reflect how cyber incidents are prosecuted in real life. For this reason a minimal cybersecurity background is needed, not to turn the learner into a penetration tester, but to provide a solid model that describes:

- how the systems and services that are typically present in organizations;
- the types of threat actors and incident objectives;
- the common sequences of attacker actions;
- the kinds of traces those actions leave behind.

This context is needed for being able to design realistic scenarios that require evidence based reasoning, and let students decide which hypoteses are plausible, which sources accept or decline, and how to connect the pecies together in order to get a defensible reconstructions of event.

In modern organization's infrastructures, incidents are not usually targeting one single "compromised machine", but since organizations have multiple endpoints, server, identities and services, the attacker often have a wide surface of attack.

Many workflows are "cloud-backend" (email, document sharing, authentication, collaborations platforms etc...), hence its difficult to define the boundary between "remote" and "local" evidence. As a results of this, investigators must be able to correlate artifacts between multiple and different origins. This means the distribution of artifacts between multiple surfaces, helps to increase the realism of the proposed scenario: a credible scenario must reflect that a single source of evidence is not enough to get to valid conclusions, and that cross validation is needed.

Another important distinction is the difference between security events, in order to being able to understand which signals we can follow.

- **Cybersecurity Incident**: An occurrence that results in actual or potential jeopardy to the confidentiality, integrity, or availability of an information system or the information the system processes, stores, or transmits or that constitutes a violation or imminent threat of violation of security policies, security procedures, or acceptable use policies. See cyber incident. See also event, security-relevant, and intrusion. [8]
- **Cybersecurity Event**: A cybersecurity change that may have an impact on organizational operations (including mission, capabilities, or reputation).

Events are usually observable occurrences (eg: failed login, new process execution, connection to externals IP).

- **Cybersecurity Campaign**: campaigns usually describe sequences of actions performed to get and an adversary objective over time. These actions can be different and executed on different endpoints of the objective, and can leave traces that help to corrrelate them.

This distinction matters since learners are presented with sequences of events, and they must be able to understand whether the they form an incident, what their scope is, and what additional evidence is needeed to support a coherent and logical explaination.

It is also important to understand how different motivations lead to different attacks and strategies. Financially motivated intrusions are often impact-oriented and typically "loud" (e.g., rapid privilenge changes, large-scale file operations, and obvious persitence. As opposite, stealth intrusion searche for persistence and low observability, which could lead the shift of the investigation towars small anomalies, long range correlation and carefull validation of the hypothesis.

When the attacker is an insider, things get more complicated since he's probably authorized in doing his moves, so the analysis is not focused on artifacts, but rather on access patterns, and differences from the standard behaviour. In the CyberForensics Arena, all the traces can be modified by selecting which of them are present, and deciding how noisy they are, in order to manage the difficulty and to shift the focus to the interested area.

Another pillar of realism is the employment of structured models that describes how the attacks are performed. Lifecycle models (such as phase models) and behavior categorizations (such as ATT& CK-style tactic and technique associations) are helpful in this regard as design aids, not as attack content.

In the CyberForensics Arena, these models enable a consistency check: if the scenario story suggests persistence, then the Virtual File System and logs must reveal artifacts credibly suggesting persistence; if lateral movement is a part of the scenario, then the learners must be presented with authentication-related artifacts or remote execution evidence consistent with that phase of the attack. This is necessary for ensuring that the story, the tasks, and the evidence all come together, which is critical for learning investigative reasoning and for supporting defensible server-side validation of task completion.

In CyberForensics Arena, the already existing scenarios are organized followed by three investigation archetypes, which can be expanded by directly modifying the sceanrios.json file (as we will see in the next chapters), and introducing new archetypes.

The following table summarizes the intial trigger for each archetype, the main evidence sources that are exposed in the simulated Linux console/ Virtual file

system, and the main reasoning skills they train.

Table 2.1: CyberForensics Arena scenario archetypes and the evidence families exposed to learners.

Archetype (CFA scenario)	Initial trigger	Primary objective	Evidence families in the platform
Disk / file-system investigation	Seized suspicious disk attached to workstation	Preserve evidence integrity and extract relevant artifacts	Device acquisition workflow (hashing, imaging, verification); read-only mounting; file system artifacts and recovered/deleted-file traces; incident hints embedded in disk content
Network + log investigation	Suspected compromised server; need to locate/confirm target	Identify attacker activity and reconstruct access pattern	Authentication logs (e.g., SSH failures/-success); firewall logs; packet capture file (pcap) as network evidence; integrity documentation and creation of working copies before analysis
Memory forensics	RAM dump suspected to contain malware traces	Identify malicious processes/artifacts present in volatile memory	Memory image handling (hashing + forensic copy); memory-derived artifacts (process list, persistence hints such as run keys); metadata summary; signature/indicator discovery activity

Another important concept that should be considered in digital forensic education is that many types of evidence are actually derived from security-related activities. This includes identity systems and multiple authentication levels providing login records, token issuance, and privilege changes. Application and system traces

may include records of process executions, file creation or modification dates, shell histories, and local logs.

Network-related evidence can include network flow records, DNS query logs, proxy server logs, and intrusion detection system alerts.

Application and system traces may include records of process executions, file creation or modification dates, shell histories, and local logs.

However, in reality, there are many constraints in digital forensic investigations that should be considered in education. This includes considerations of log retention times, lack of visibility, and time-related inconsistencies.

This cybersecurity context directly maps to the typical questions digital forensics attempts to answer:

- what happened?
- when it happened?
- how it happened?
- which systems or accounts were involved?
- what are the impact and the scope?

From these questions, some typical outputs that a forensic investigation might yield include a timeline of relevant events, a list of indicators and assets, and a narrative that relates everything to the underlying hypothesis.

The aim is not "absolute certainty," but a defensible explanation supported by multiple consistent observations, together with documentation of assumptions and uncertainties.

These considerations guided the design of CyberForensics Arena. The platform must provide learners with artifacts that are representative of real investigation (operating-system traces, log files, and scenario-specific evidence), while keeping the environment safe and controllable for classroom deployment.

The choice of what telemetry to track, how to design the incident archetypes, and how to unveil the information step by step become an integral part of the educational design. Realism is maintained by ensuring internal consistency between the incident model and the revealed evidence, whereas complexity is managed by ensuring appropriate scope and task guidance.

The cybersecurity context of the chapter provides the conceptual foundation for the discussion of the digital forensics process models, the handling of the evidence, and the data source families that appear in the following chapters. It is also the foundation for the requirements and mapping work introduced in the following chapters, 3 and 4.

2.3 Digital forensics: goals, principles, and typical outputs

Digital Forensic is a field of forensic science. It is used to investigate cybercrimes, but can also be useful in criminal and civil investigations. Cybersecurity teams can use computer forensics to identify the cybercriminals behind a malware attack, while law enforcement agencies might use it to analyze data from a murder suspect's devices.[9]

Digital forensics has broad applications because it treats digital evidence like any other form of evidence. Authorities follow specific procedures to collect physical evidence from a crime scene. Similarly, investigators specializing in digital forensics adhere to a strict forensic process, known as chain of custody, to ensure proper handling and protection against tampering.

Digital Forensic goal is not to "find the bad guy", but rather maintaing, preserve, identify, extract and document the digital evidence in a way that is legally admissible and defensible.

Unlike a conventional debugging process, where the state of the system can be modified to resolve an issue, forensic analysis is a process that prioritizes data integrity above all else, ensuring that the results can withstand scrutiny in court or at a disciplinary hearing.

To attain this legal defensibility, the discipline relies on two main pillars: integrity and repeatability. The integrity requirement states that the process of collecting and analyzing the evidence should not in any way change the evidence itself. Once the original evidence is changed during the process of investigation, it can no longer be considered a true record of the crime. On the other hand, repeatability is essential in attaining scientific validity in the process of investigation. This means that if a different investigator were to use the same tools and methodology on the same evidence, he or she should come up with the same conclusion. This is what constitutes the "investigative mindset" that students are required to learn how to treat the production environment as a crime scene rather than a broken server to be fixed.

The goal of this process is reconstruction. The job of the investigator is to answer the six questions of investigation:

- Who performed the action?
- What happened?
- When did it happen?
- Where is the evidence?
- Why did it happen?

- How did it happen?

This is a process of correlation of disparate data sets, such as a deleted file, a log entry, and a time stamp, to create a coherent story of events. In a learning environment such as the CyberForensics Arena, this is the learning objective. The student must go beyond the ability to simply run a tool and understand how individual data sets contribute to a coherent story of events.

The end result of this process is the Forensic Report. This is a report that acts as a bridge between the technical investigation and the legal or business decision-makers. This report is normally composed of an executive summary written for the lay person, a detailed timeline of events, and a technical appendix to verify the process. In a serious game environment, this is simulated by validation of the tasks and the findings. The student learns that it is not simply finding the artifact, but understanding its significance and placing it within the appropriate time sequence.

2.4 Digital evidence handling: integrity, preservation, chain of custody

Digital evidences are data that can be stored (at rest), used (present in main memory) or transmitted in digital form and that can be used in court of justice. Collecting digital evidences is not a simple task because data requires:

- **interpretation**: data needs to be connected to the reality; in other words, you have to find objective conclusions even if they are subjective-driven because in a court of justice, subjective is not accepted.
- **attention**: data needs attention because of its fragility. Data can be modified involuntarily or maybe voluntarily; so, techniques to avoid this must be adopted like bitstream copy (and always work on that copy, never on the original device/data), the use of hash function and so on. In other words, digital evidence requires technical understanding of different possible types (files, emails, logs, metadata) and the legal requirements for collecting and preserving it in order to avoid alteration of data both on the original device and on the device where they are collected.
- **knowledge**: sometimes, to find digital evidences, the most important thing is to know how to search them. For this reason, the knowledge of file systems, network protocols, and encryption methods are essential. If you've not got any knowledge, you will not be able to conduct a forensic investigation or to adopt an anti-forensic approach.

Another important concept is the chain of custody. It is a documented and unbroken process of handling evidence from the time it is collected until it is

presented in court. In other words, it is the proof of the correct custody of proofs starting from the collection to the presentation in a court. If not well documented or broken, proofs (aka digital evidences) will not be accepted anymore.

The chain of custody ensures that digital evidences have not been tampered, altered or accessed by unauthorized parties. If one of these things happens, evidence is not legally admissible anymore. In order to "perform" correctly the operations needed for the chain of custody, knowledge about how to document evidence collection, storage and access are needed otherwise the chain of custody will fail. We talked about the difficulty of collecting digital evidences from devices without altering or damaging the original data.

This task is called data acquisition and, in a forensic investigation, it is only the first. By the way, it is also the more difficult because requires knowledge and/or expertise of many things like the knowledge of disk imaging, live data capture, file systems, hashing, write-blockers or the expertise in forensic acquisition and analysis tools. Let us analyze a few of these things:

- **Hashing**: the process of converting data into a fixed-length string of bits to ensure the integrity of digital evidence. The hash value proves that a file has not been altered during the investigation (from the collection to the presentation).

  A deep understanding of hashing algorithms is needed because if you use a weak hashing algorithm (so, collisions are possible. e.g. MD5), you are subject to counteract in the court because collisions can be possible and not detectable. Lastly, the hashing process have to be used any time an evidence is managed that implies copied or moved to ensure the correctness of the process of moving or copying.

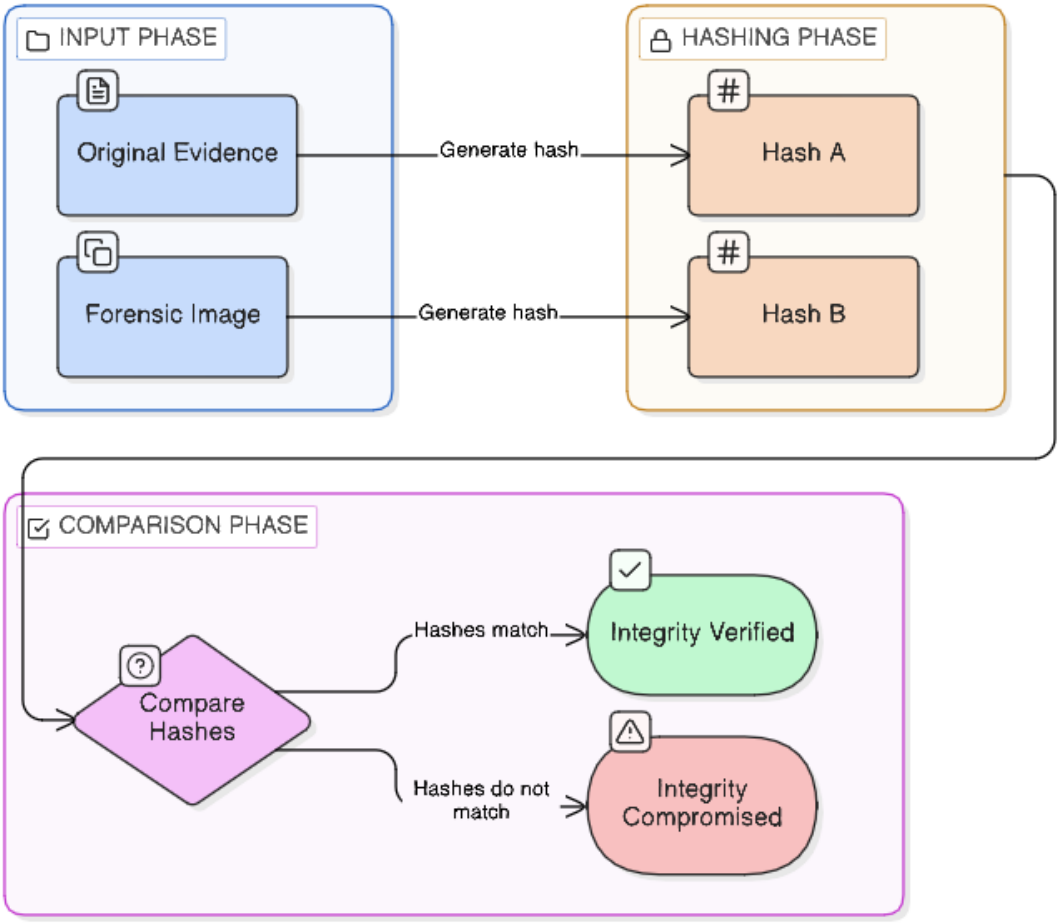

Figure 2.1: The Hashing Process

- **Write Blocker**: it is a tool (hardware or software one) used to prevent any data from being written to a storage device during analysis, preserving the original data content. If you modify data during the acquisition, nothing will be admissible later because you cannot prove the correctness of nothing.
- **Forensic Image**: it is a bit-by-bit copy of digital media, including deleted files and data in slack space.

  It will be an exact replica of the original device. This is used because you have to preserve the same image in the same organization of bits (not bytes), for this reason also slack spaces and partitions are included. There are a lot of tools useful for acquiring a bit-by-bit copy. You need to deeply understand how to use these tools because if you wrongly use them, you simply overwrite/damage the original data. Some of these tools can also be used to forensically erase

data to avoid a successful bit-by-bit copy of your data. The final goal of having a forensic image is to preserve the original evidence.

Remember: the (file) slack space is the leftover storage space on a computer's hard disk drive when a file does not need all the space it has been allocated by the operating system (OS). More specifically, it refers to all the unused storage space in a hard drive's file allocation block or memory page, and often holds residual data. The examination of slack space is an important aspect of e-discovery and computer forensics.

2.5 The Investigation Process

There are some typical computer forensics scenarios such as internet abuse from employee, data unauthorized manipulation (data theft or data disclosure) and in general, any time digital evidences may be involved in an incident.

Whatever the scenario, after an incident in which digital evidences are involved, the investigation process starts. Nowadays, many computer forensic standards exist to guide the investigation process based on the different scenario. For example, think about military scenarios, in which the maximum grade of security is required in respect of other scenarios, so a different standard to guide the investigation process is needed. Moreover, we have also international, national and sometimes local standards, so each time, for each scenario, the best approach must be chosen. Maybe each country has its own standard.

Some examples of standards are:

- **NIST family (SP 800-88, SP 800-101, SP 800-86)** to guide digital forensics and/or mobile forensic into Incident response, media sanitization and so on;
- **SWGDE** (Scientific Working Group on Digital Evidence) that describes the best practices for chain of custody documentation;
- **ACPO Guidelines (UK)** that describes the best practices for digital evidence handling.

The investigation process is divided into five main phases:

- Identification
- Collection
- Acquisition
- Evaluation
- Presentation

2.5.1 Identification

This phase corresponds to the identification of potential sources of relevant data (digital evidences) before any acquisition begins. Some potential sources can be: memory (RAM), hard drives (HDD/SSD), cloud storage, network traffic, removable media (USB drives, DVDs) but also mobile devices (smartphones, tablets and so on) and especially IoT devices and embedded systems (such as smart dishwashers, smart fridges and so on).

The process of identifying the potential sources is quite complex. Typically, you have to perform an initial survey of the scene and of the subject (maybe using OSINT) in order to easily and more accurately understand the scenario. Then, you have to identify key devices and data locations (local storage, remote servers, cloud services). After that, check for connected devices, removable media or network-attached devices and map all potential data sources using network topology diagrams or asset inventories.

Beware: pay attention to ephemeral storage of data. The concept of order of volatility becomes very important. It categorizes data based on its volatility starting from the most ephemeral (stored in RAM for example) to relatively stable forms. In other words, you need to pay attention about data which can be lost/modified, for example at the shutdown of a device if in RAM.

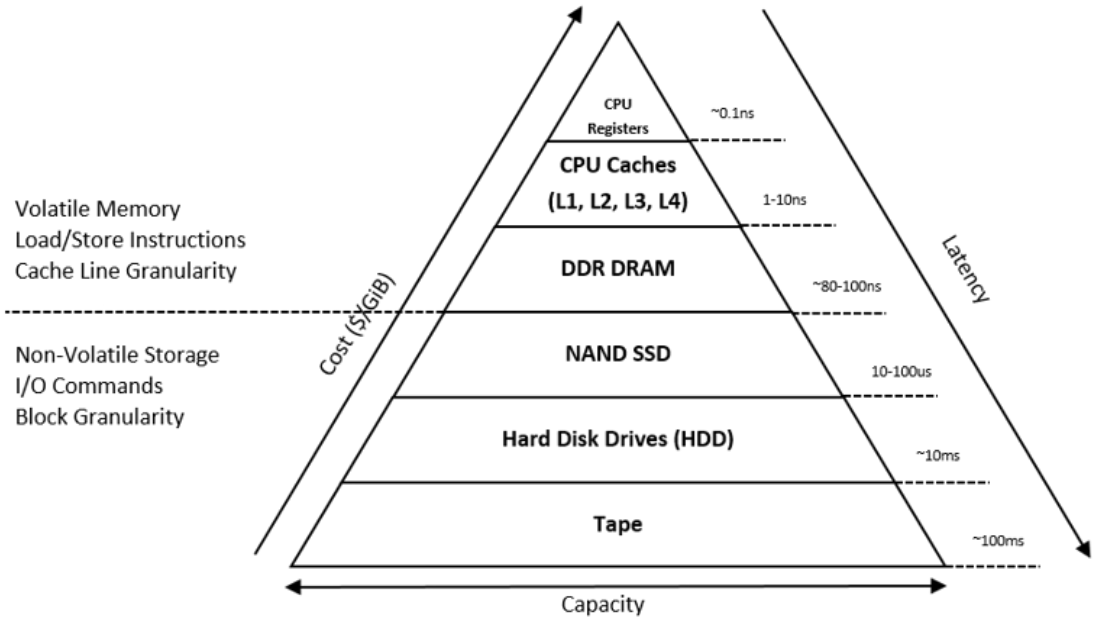

Figure 2.2: Memory Storage Hierarchy

Another example can be the cloud syncing: some of the data found on the

local device may also be stored in the cloud, but the synchronization process can be ephemeral. For example: changes made on the device (like a deletion or modification) can be quickly reflected in the cloud, resulting in the loss of earlier versions; live sync operations might overwrite or erase certain logs, application data, or file versions if not captured promptly.

At the end, investigators must identify cloud sync mechanisms to retrieve the full set of data before it is altered or deleted. To conclude, in digital forensics, especially during the identification phase, timeliness is critical when dealing with ephemeral storage. Once identified, investigators must act swiftly to capture this volatile information before it is lost or overwritten.

2.5.2 Collection

The collection phase is the process of gathering evidence from identified data sources while ensuring the preservation of its integrity. The collection phase can take place, for example, by seizing potential sources and then making copies of the data as needed (here we have raw data). In addition, hashing and other tools should be used to prove that the copied data is the same as the original data and that the original data has not been altered. We can think on collection as the phase in which data are collected (maybe taking a bitstream copy) but with no possibility to analyze data yet. More in detail, you need to:

- Isolate devices to prevent them from being tampered with remotely (e.g network disconnection). You should use devices to block external communication, for mobile (faraday bags, used to preserve evidences) or wireless devices, and you should use network isolation tools for virtual and cloud environments to prevent remote access (e.g. firewall rules). You must pay attention in live systems to ensure evidence integrity while maintaining system uptime because shutting down can cause the loss of volatile data (e.g. RAM) unless you are able to maintain RAM a lot of grades below of 0.
- Create a detailed record of the condition and state of evidence. So, take photographs of the devices on the scene, record serial numbers, device models and any other identifiable information. In addition, document the scene, noting which devices were running, whether screens were active or locked and any other visible indicators. These papers are crucial to prevent legal challenges about the integrity/correctness of the evidence; without it, the integrity/correctness will be discussed.
- Preparing for acquisition ensuring no alteration will take place. For example, you can obtain this using write blockers or disabling connection and syncing notification. Note: maintain integrity on live systems is complex, so, you may use remote collection methods that minimize data alteration risks.

2.5.3 Acquisition

it is the process of creating a forensic copy (bit-by-bit image) of the original data (here data is not raw anymore like in collection phase but analysis is possible). Integrity of data must be maintained and the acquired data must be an exact replica of the original one. First of all, a complete and well documented chain of custody for the evidence throughout the acquisition process must be maintained. In this document, each step in the acquisition phase, including the personnel, the tools, the time and the date must be detailed. Moreover, evidence have to be stored securely to avoid unauthorized access or tampering. If something is the chain of custody is wrong, all the evidence will be inadmissible in legal or forensic context. In addition, the integrity of data must be ensured in order to prove that the copy is the exact replica of the original data and the copied data has not been altered. For this reason, you must choose a method for acquiring data that minimize system interference while generating a hash of the acquired image or data dump. The system interference is an alteration of the normal behaviour of the device that can cause the alteration of the result of the hash; so, the acquiring method must minimize the system interferences. Also the hashing process, including algorithms used and the results, must be included into the chain of custody documentation

because results can be used later to check if the data copied are the same as the original one (simply comparing the hash result). We said that the acquisition methods must be chosen carefully, but what are the acquisition methods? There are two methods for the acquisition:

- **Static**: the system is powered down, but the shutdown process must be done carefully to avoid losing data (e.g. take the decryption key before it is wiped from RAM in order to decrypt data later). Maybe a write blocker should be used and forensic imaging tools to create a complete image of the storage device must be used. Moreover, it is needed to generate a hash value of the original media before and after acquisition to verify integrity and to store the image on a secure forensic storage device. The static is the most common method for acquiring data from hard drives and external media.

- **Live**: the system is running , so it is needed to deal with volatile data like RAM, network connections, running processes and so on. To minimize system interference while capturing volatile data, it is essential to select a method that has a low impact on the system's operation (in other words, to minimize system interference while capturing volatile data). The process should include capturing memory (RAM) data, extracting information from active processes and network connections, and conducting a capture of network traffic. This should be done with minimal disruption to the system's normal functioning, ensuring the integrity and availability of the data during the acquisition process.

At the end, it is important to document all acquisition actions and steps to ensure chain of custody and admissibility. For this reason, if possible, hash the volatile data to maintain and prove data integrity.

2.5.4 Evaluation

In this phase, it is needed to ensure the integrity, authenticity and admissibility of the digital evidence collected during an investigation. After that, data can be analyzed, verified and validated to extract digital evidence for presentation later. To ensure that everything is correct you need: to compare fresh and stored hash values; to validate the chain of custody; to verify forensics image; finally to verify live data. For performing these actions, you may use timestamp and metadata to analyze them or to reconstruct a timeline to understand the evolution of things (e.g. who access data? when?). Moreover, during this phase, a cross-reference analysis or consistency verification can be done maybe through comparison of data from different sources. At the end, after analysis, a double check to prove the compliance with current legal/internal standards. Pay attention: data must remain unaltered during the whole phase.

2.5.5 Presentation

This phase consist in preparing and presenting the findings of the investigation in a clear, accurate, and legally admissible manner. The goal is to ”translate” the technical details of the forensic analysis into a format that can be understood by non-technical stakeholders, such as lawyers, judges, or company executives. This is the most important step because quality and clarity of the presentation can impact the result of legal proceedings or internal investigations. To generate a presentation, first of all, a whole review to ensure correctness of the entire process (from the first to the last phase) must be done to avoid problems later. After that, key evidences must be selected (the evidences that can be comprehended by the audience to prove what you want; maybe in the appendix some reinforcements can be inserted), then correlated, if possible, to show what you notice. At the end, the whole document needs to include the forensic process (free from technical jargon) and the key evidences presentation. Moreover, the report must be managed securely as well. Beware: some appendices with timestamps, metadata, hash values and technical evidence can be used as reinforcement of what stated into the report; try to avoid personal (or objective) interpretation unless asked for expert opinion because objective cannot exist in a legal environment.

2.6 Forensic Data Sources, Analysis Families, and Tools

In digital forensics, one has to manage a variety of data sources, each of which requires its own set of techniques and tools. Understanding these data families is a necessity for every digital forensic practitioner and, therefore, every learning system that hopes to teach investigative skills.

In this section, we shall discuss the three major families of digital forensic analysis, which are at the core of digital forensic practice: file system forensics, network forensics, and memory forensics.

These three families of digital forensic analysis map to the three types of scenarios presented in the CyberForensics Arena, ensuring that students learn the entire gamut of digital forensic investigation techniques.

2.6.1 File System Forensic

The file system forensics domain examines the organization and structure of file systems to retrieve hidden or deleted data. The most important things in this domain are:

- **Slack Space Analysis**: slack space refers to the unused portions of disk sectors that are left over when a file does not completely fill its allocated space.
- **File Carving**: this technique involves recovering deleted files by analyzing clusters of residual data in slack space and other unallocated areas. Data clusters refer to the basic units of data storage on a filesystem where files are stored. A cluster (also called an allocation unit) is a group of contiguous disk sectors that the filesystem allocates together as a single unit to store file data. When a file is saved, it's divided into one or more clusters, and when files are deleted, these clusters may retain residual data until they're overwritten by new files. By searching for recognizable file headers, footers, and known patterns, file carving can piece together fragments of deleted files, even if they're no longer accessible through the normal filesystem.
- **Registry Analysis / OS config analysis**: forensic investigators also examine operating system settings and user-specific configurations to gather insights into past activities. In Windows, this involves analyzing the Windows Registry, which logs system and user actions and stores configuration data. In Unix-like systems, configuration files in locations such as /etc, ↑/.config, or ↑/.bashrc may contain similar information, recording user preferences, recent commands, and more to perform a sort of timeline reconstruction of actions.

A file is the smallest logical unit from a user perspective that can be stored and later mapped to physical storage entities (such as computer RAM, hard drives, cloud storage, etc.) by the operating system (OS). The OS, through a file system, establishes rules for reading, writing, and organizing data according to the structure defined by the file system itself. Each file has some attributes such as, for example:

- a **name**, which is a mnemonic identifier used for reference;
- a **type** used to categorize the file and indicate how it should be handled or manipulated. Different file types are often identified by a magic number, a specific sequence of bytes typically located at the start of the file. Some attacks exploit this by altering the magic number to disguise a file as a different type. For example, an attacker might change the magic number of a malicious executable file to mimic a harmless image file, allowing it to bypass security filters and trick users into opening it;
- **protection information** used to enforce access control, such as the owner and group, as well as the read, write, and execute permissions;
- **location**, indicating where the file is stored on the device;
- **size**, specifying the file's storage footprint

The first operation used to prepare a mass storage device for data storage is file system formatting, which configures the device with specific file system structures. The steps involved in this process are:

- Erases existing data on the mass storage device. This can be done in two ways: full formatting, which involves replacing all sectors with zeros but is slower; and quick formatting, which is faster but only removes the file system structures, leaving some residual data behind.
- Creates one or more partitions. Each partition can be formatted independently and assigned its own file system.
- Selects a specific file system for each partition, such as NTFS for Windows, APFS for macOS, and ext4 for Linux. The choice of file system depends on the operating system and requirements for the storage device.
- Creates foundational structures necessary for managing files and directories. These structures typically include:
  - **Root directory**: The top-level directory of the file system, from which all other directories and files branch. It is the starting point for the directory hierarchy.

- **FAT (File Allocation Table)**: A structure used in some file systems (like FAT32) to track the location of files on disk. It maps the logical file system structure to the physical clusters on the storage medium. The FAT is a table that stores information about which clusters are used by files and which are free. Note: FAT table is different from FAT file system.
- **Inode table**: In file systems like ext4, an inode (index node) stores metadata about files, including the file's size, owner, permissions, timestamps, and pointers to the actual data blocks on the disk. The inode table is a collection of all the inodes in the file system.
- **Superblock and Boot Sector**: The superblock (in file systems like ext4) contains critical information about the file system's structure, such as the size of the file system, block size, and other metadata necessary for accessing and managing the file system. The boot sector, typically used in systems like FAT, contains the initial loading information to start the operating system from the disk.

One of the most important operations on files is the **copy**. The OS provides simple commands to perform this operation, ensuring that the file content is preserved, but the metadata (e.g. creation date) are altered. To avoid altering metadata or any other modifications, a bit-by-bit copy is needed. Some tools that can be used for this task are:

- **dd (data dump)**: allows for a bit-by-bit copy/convert using a simple command pattern like: dd if=<inputfile> of=<outputfile>.
- Forensic tools such as **dcfldd** and **dc3dd** also perform bit-by-bit copies, but they include additional forensic features such as on-the-fly hashing (with any chosen hash algorithm), pattern wiping (specifying how data should be erased, e.g., replaced with zeros or a specific pattern), and progress reporting.

File identification is another important operation, that requires more than just checking the file extension, as extensions can be easily changed by anyone and thus are unreliable for determining true file type. Instead, more accurate identification methods involve examining a file's metadata and its initial bytes, which act as a unique signature and is a sort of "internal" metadata. File signatures, or magic numbers, are distinctive byte patterns at the beginning of a file that can be compared against known file types to reliably identify its format. **xdd** is a command line tool that allow us to to create a hexadecimal dump of a file's content, and consequently, to check these initial bytes and verify the signature against known magic numbers.

In terms of metadata, the file system itself stores a range of information about each file. This includes basic attributes like the file name, ownership, and access

permissions, as well as the specific data blocks allocated to the file on the storage medium. Additional metadata typically covers the file's size, time stamps (creation, modification, and access times), and any recovery data, such as journaling information, which helps in restoring data if there's a system crash or if we are in a forensic investigation. The type and detail of metadata available vary significantly with different file systems.

Consequently, the **recovery process** of a file is both a crucial and challenging task in digital forensics. It involves carefully analyzing file system structures, metadata, and system files to recover data, even if the file has been deleted or corrupted. This process requires a deep understanding of how data is stored, how it can be overwritten, and how forensic tools can retrieve deleted or orphaned files. To recover files, the steps to follow are:

- **Analysis of File System Structures**: in digital forensics, one of the first steps in recovering data is to analyze the structure of the file system. File systems, such as NTFS or FAT32, organize and store information about files on a storage medium (e.g., hard drives, SSDs). This organization includes details about file names, types, sizes, and locations. Key components of a file system, like the Master File Table (MFT) in NTFS, play a crucial role in managing this metadata. The MFT stores detailed information about each file on the system, including its name, creation date, modification date, and location on the disk.

  To help us in this analysis, **fsstat** is a great tool that can help us: it displays general details of a file system, such as volume name, last mount time, and the layout of the metadata and data units.

- **Role of Metadata**: metadata are essential for forensics, as it provides information about a file's attributes, even when the file itself is deleted. For example, timestamps such as creation, modification, and access times can help investigators build a timeline of events. This timeline is particularly useful in criminal investigations to determine when certain files were accessed or modified, and to verify the integrity of evidence (i.e., confirming that files have not been tampered with).

  However, it's important to note that metadata is not trusted by definition. While metadata can be a valuable tool for investigators, it is not immune to manipulation. In some cases, metadata can be intentionally altered or fabricated to mislead or obscure the true nature of the file. Therefore, while metadata analysis is a powerful method for recovering and analyzing files, investigators must be cautious and verify the integrity of the metadata to ensure its reliability. This is why careful metadata analysis and cross-referencing with other data sources are crucial steps in the forensic process, helping to ensure that the findings are accurate and trustworthy.

- System Files and Security: Operating Systems (OSs) store metadata in system files, which help manage permissions, integrity, and tracking of files. For example, system files track who has access to a file, when it was last accessed, and whether it has been modified. This information is vital for maintaining the security and reliability of the system, especially in legal contexts where data must be preserved in its original state. Ensuring that data is not altered or destroyed inadvertently is crucial for maintaining the integrity of digital evidence.

Moreover, even after files are deleted, forensic tools can often recover them by analyzing "orphaned" data or file signatures. When a file is deleted, its data is usually not immediately erased from the disk. Instead, the operating system marks the file's space as available for reuse, but the data itself may remain on the disk until it is overwritten by new information. Forensic tools can search for these "deleted" files and use file signatures, the magic numbers, to recover and reconstruct the file. A famous tool used for this specific task is **foremost**, a command line tool that recovers data based on their headers, footers, and internal structure.

Now, simply deleting a file is clearly not enough. When a file is deleted, the operating system doesn't actually remove the data but instead just marks the space as available for reuse. This means that the file's fragments can still be recovered until they are overwritten by new data. For example, the rm command, when used on Linux or similar OSs, doesn't physically erase the data from the storage device because it simply removes the references to the file in the file system, leaving the data intact until it's overwritten. This is why recently deleted files can often be recovered partially or even completely. Additionally, some parts of deleted files may remain on the device for years, hidden in slack space. To properly and securely remove files (filesystems, OSs), even from slack space, data sanitization is essential. Data sanitization ensures that files cannot be recovered after deletion, providing a higher level of security for sensitive information. There are several methods and tools designed for this purpose:ù

- **File Shredder Programs**: these programs permanently delete selected files by overwriting them using a specified data sanitization method. This ensures that the data is completely erased and cannot be recovered by forensic tools.
- **Data Destruction Software**: this software completely erases all data from a hard drive (HDD) by overwriting it with one or more sanitization methods. It's particularly useful when preparing a disk for disposal, recycling or for ensuring that no traces of data remain after virus removal.

  They also analyze logs from devices such as routers and firewalls to reconstruct the steps of a potential attacker, helping to identify the source, method, and

scope of an intrusion. Tools like Nmap assist by scanning for open ports and vulnerable devices on a network, providing clues about possible entry points.

The primary challenge in network forensics lies in dealing with encryption and obfuscation. These methods can hide the contents of network traffic, making it harder to inspect packets directly. However, by combining packet analysis, log data, and advanced forensic tools, investigators can often piece together the sequence of an attack and determine how best to respond, securing the network and mitigating future risks.

2.6.2 Network Forensics

Network forensics focuses on monitoring and analyzing network traffic to detect unauthorized access, data breaches, or other suspicious activities. Investigators capture packets of data (PCAP - Packet Capture) using tools like Wireshark to inspect packet details for signs of unusual activity.

They also analyze logs from devices such as routers and firewalls to reconstruct the steps of a potential attacker, helping to identify the source, method, and scope of an intrusion. Tools like Nmap assist by scanning for open ports and vulnerable devices on a network, providing clues about possible entry points.

Below, an example of nmap output of "*nmap -v scanme.nmap.org*"

```
Starting Nmap ( https://nmap.org )
NSE: Loaded 153 scripts for scanning.
Initiating Ping Scan at 10:00
Scanning scanme.nmap.org (45.33.32.156) [2 ports]
Completed Ping Scan at 10:00, 0.15s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 10:00
Scanning scanme.nmap.org (45.33.32.156) [1000 ports]
Discovered open port 22/tcp on 45.33.32.156
Discovered open port 80/tcp on 45.33.32.156
Completed SYN Stealth Scan at 10:00, 1.2s elapsed (1000 total ports)
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.050s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
135/tcp  filtered msrpc
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
```

```
Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 2.50 seconds
           Raw packets sent: 1045 (46.012KB) | Rcvd: 1005 (40.200KB)
```

The primary challenge in network forensics lies in dealing with encryption and obfuscation. These methods can hide the contents of network traffic, making it harder to inspect packets directly. However, by combining packet analysis, log data, and advanced forensic tools, investigators can often piece together the sequence of an attack and determine how best to respond, securing the network and mitigating future risks.

In digital investigations, evidence identification and collection involve tracing the online activity and digital footprint of the individuals or organizations involve.

We can summarize the network forensic process in 5 steps:

1. **Evidence identification**: Often involves reviewing publicly available information to locate relevant sources. This may include examining social media profiles (OSINT is very important here), email addresses, domain names, IP addresses and public repositories. These digital traces offer insights into the affiliations, activity patterns, and connections of the individuals involved, sometimes revealing related accounts, aliases, or hidden online identities that may not have been immediately apparent.

2. Once investigators identify these sources, they move into **evidence collection**, carefully gathering relevant data without altering it to maintain its integrity. Publicly accessible data (OSINT) often supplements formal digital forensics, offering additional forensic data. For example, archived website snapshots, information on registered domains, and public posts made by the suspect on forums or social media can provide valuable insights. Collecting metadata, IP history, and timestamps from these sources also supports the construction of timelines and helps verify specific activities, enabling investigators to piece together events more accurately.

3. Once data is identified and collected, it must then be preserved. **Data preservation** ensures that all evidence remains in its original state and can be verified later to confirm its integrity. Tools are available to create evidence snapshots, including metadata, which enable future verification and support the defensibility of the evidence in court.

4. At this point analysis can begin. **Data analysis** focuses on identifying relationships, patterns and connections within the evidence, and reconstructing timelines to understand the sequence of events. This may include social network analysis to identify key assets, such as individuals, groups, or online platforms

involved in the case, and tracing digital artifacts, like emails or messages, to determine their origin. The goal is to uncover connections between individuals, domains, IP addresses or entities based on shared activities, posts or affiliations. Discover these correlations could help also to find the origin of digital incidents.

5. At the end, a **report** will be generated including the findings. In the report, information helpful to reinforce forensic findings must be inserted

Network forensics provides the investigator with the unique 'in-motion' view of the external and internal communications of the system, which records information that might otherwise elapse and not leave any recoverable evidence on the physical storage media of the system. Indeed, by monitoring the 'traffic artifacts,' an investigator can trace the progression of the intrusion or the actions of the attacker.

2.6.3 Memory Forensics

Mmemory forensics **examines** the contents of volatile memory (RAM) to extract running processes, decrypted data, user activities, and even decryption keys that are actively used in RAM. Real time extraction is complicated by the risk of tampering and corruption. The main challenges include the volatility of data, as it disappears upon shutdown, and the correct acquisition of a memory dump, which can be achieved using tools like LIME, as previously mentioned. As a recap, a memory dump is a snapshot of the contents of a computer's RAM at a specific point in time and it is acquired in a forensic way, so bit-by-bit. After the acquisition of the memory dump, it needs to be analyzed, using tools like Volatily or Rekall. The analysis of memory dump allows to:

- **Recover volatile data** , like transient data not stored on disk (e.g. encryption keys and in memory malware).
- **Understand the state of the system** through active processes, open file and network connections (network traffic and socket analysis).
- **Highlight malicious activity**, such as the presence of malwares mainly operating in memory.
- **Perform process analysis** to identify suspicious processes like running or hidden processes or even anomalies like injected code. For instance, a malicious process masquerading as a legitimate system service might be identified through memory dump analysis.
- **Detect code injection** by locating and analyzing injected code or malicious payloads, often using plugins like malfind (part of the Volatility tool), to identify unusual memory regions that are associated to legitimate processes.

- **Examine registry and file system** by analyzing registry keys (configuration settings stored in the Windows registry) and open files in memory, which can help identify malware(s) because it often modifies registry keys to ensure persistence on the system.
- **Perform network forensics** by analyzing active network connection to identify suspicious one, such as exfiltration attempts/connections to malicious servers.
- **Recover credentials** by extracting encryption keys and passwords from RAM, which are useful for decrypting files or communications and for gaining access to critical evidence.
- **Perform crash and failure analysis** to diagnose vulnerabilities, exploits or system crashes that could indicate the presence of malicious activity or software flaws.

The challenges of memory dump analysis include:

- **the size/volume** of the data, which may be very large and result in resource intensive analysis;
- **obfuscation techniques**, such as encryption or anti-debugging, employed by advanced malware to make detection more difficult;
- **Forensic soundness**, which ensures that the dump is collected and handled in a way that preserves its integrity and maintains its admissibility in court (bit-by-bit copy is needed to ensure this).

Moreover, to access and copy each zone of central memory, the tool used for performing the bit-by-bit copy must operate directly at kernel level, or use a module that operates at kernel level and then communicate directly with the tool itself. There are many memory access tools for capturing memory dumps:

- **Debugging tools**, such as **gdb** (GNU Debugger), can capture memory regions limited to specific processes. However, in a forensic context, these tools act as a middleware between the investigator's access point and the central memory itself. This intermediary role can introduce potential issues, such as the risk of retrieving altered or incomplete data if the operating system has been compromised, as well as the possibility of introducing artifacts during the debugging process, which may affect the integrity and authenticity of the evidence collected.

- **Crash dump utilities**, designed primarily for system administrators to capture the memory state during system crashes, can also be used in forensic investigations. Similar to debugging tools, these utilities are commonly employed for debugging purposes but are suitable for forensic use as well. An example is kdump, which creates a crash dump when the kernel crashes, enabling administrators or investigators to analyze the state of the system and uncover what happened

- **LiME** (Linux Memory Extractor) is an open-source forensic tool designed for live memory acquisition on Linux-based systems. It enables the acquisition of the entire contents of a system's RAM in real time, facilitating the recovery of volatile data such as passwords, encryption keys (useful for decrypting communications, memory regions, etc.), and evidence of running processes. LiME functions as a kernel module, allowing it to interact directly with the operating system and access all memory regions without interference. The tool maps the entire physical memory of the system, including user space, kernel space, and other volatile data. Once mapped, it reads memory contents sequentially and writes them to the specified output destination, depending on whether the acquisition is local or remote. LiME's output format preserves the integrity of the memory content, as it performs no compression, modification, or addition of metadata. It also accounts for the system's endianness to ensure compatibility with the architecture being analyzed. It supports remote memory acquisition, enabling memory data to be extracted over a network. In a remote acquisition scenario, the process is initiated remotely, and the memory data is transmitted via sockets to a LiME server for storage. For local acquisitions, the memory contents are written directly to a file on the machine and then transmitted to forensic workstations for further analysis if needed.

- **Fmem**,a kernel module that provides a device file for raw physical memory access (e.g., /dev/fmem), can be accessed for reading using tools such as dd for instance. Since the dump created by fmem is accessible at the user space level, analysis can be performed using tools that do not operate at the kernel level, making it a versatile option for forensic investigations.

Once the memory dump is acquired, it needs to be analyzed while ensuring forensic correctness (e.g., preserving integrity). One of the most widely used frameworks for this purpose is Volatility. It is an open-source, memory forensics framework organized as a set of plugins, designed to analyze memory dumps from various operating systems. At a high level, its primary goals are to uncover evidence, identify malicious activity, or debug systems by examining the contents of a specific system's RAM. Volatility is highly powerful due to its multiplatform capability,

allowing it to analyze memory dumps from Windows, Linux, macOS, and Android systems. Moreover, it supports a variety of input formats, including raw dumps, crash dumps, LiME files, and hibernation files. Hibernation files, for instance, are files used by systems to restore their state upon waking from hibernation. Since these files contain memory data, otherwise the state cannot be restored, they can be useful for forensic analysis when needed. Volatility achieves its goals through a rich set of plugins, which perform low-level tasks to extract specific information. These include, for example, identifying running processes, open network connections, loaded kernel modules, and artifacts such as command histories or encryption keys.

In conclusion, it may be said that the field of memory forensics offers a crucial 'live' picture of a particular system which remains impossible to achieve with the help of traditional disk-based forensics. By collecting volatile information such as encryption keys, network connections, etc., it is possible to recreate the reality of a particular security incident in its live state.

2.7 Serious games: definition and evidence for learning impact

Although the term "Serious Games" (SG) has come into fashion with the arrival of the digital age, its history can be traced back to Clark Abt (1970), "who suggested that serious games are those with an educational goal that is not primarily intended as an amusement."

As described in the related literature, Serious Game is an application of three fundamental components: Experience, Entertainment, and Multimedia.

The serious element, as Michael and Chen (2005) note, is highlighted in terms of "conveying a specific message or skill, and the player is placed in an environment in which he or she experiences this information derived from professional know-how."[6] The generally accepted definition in terms of the technical elements of a game is found in the work by Zyda (2005): "A mental contest, played with a computer according to specific rules, that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives."[10] The "birth" of the modern SG movement is commonly traced back to 2002, when America's Army was released, a proof of concept work that showed a sophisticated simulation environment could effectively link together recruitment, advertising, and technical training, thus demonstrating that games can, indeed, achieve "total public awareness" as a legitimate form of learning. The effectiveness of Serious Games is not merely a result of their "fun" factor, but rather their alignment with several key pedagogical theories:

- **Situated Learning**: Learning with symbols and theories is often disconnected by traditional education methods, which create a division between the symbols

or theory and the reliability of their application. SGs provide the opportunity of "situated understanding". That is, the learner will experience the reality of the concepts often discussed in abstract terms. In the case of Digital Forensics, the learner will go from reading about the identification of file headers, to the hands-on identification of the headers in a simulated compromised enviroment.

- The **"Flow" and Immersion**: In a classroom, its easy for focus and attention to drop after a few minutes. In game enviroments, it's possible to sustain high levels of focus and concentration for long periods of time, and this can often be explained in terms of "flow": the completely immersive experience in which the level of of challenge of the tasks matches the abilities of the learner.

- **Active Knowledge Construction**: Information is not knowledge. In SGs, learners are required to process information by cognitive loops: taking actions and obtaining immediate visual feedback (cause and effect). In other words, "learning by doing" is ensured, and new information is successfully integrated with pre-existing knowledge.

The move towards Game Based Learning (GBL) has been supported by the solid market growth and, more importantly, empirical research, with meta analyses offering the "evidence of impact" needed to incorporate these tools in higher education settings.

The groundbreaking study by Wouters et al. (2013) examined 39 independent cohorts and found Serious Games to be "significantly more effective than conventional instruction." Their results revealed:

- 11% higher knowledge acquisition compared to conventional instruction.

- 20% higher retention, indicating that the "active" component of game based learning generates more potent memory paths than the "passive" listening associated with conventional instruction.

Additionally, Garris et al. (2002) identify the Game Based Learning Cycle, which serves to demonstrate the repetitive process of user judgment and action, initiated by game feedback, and how this process sustains motivation and facilitates particular learning goals.

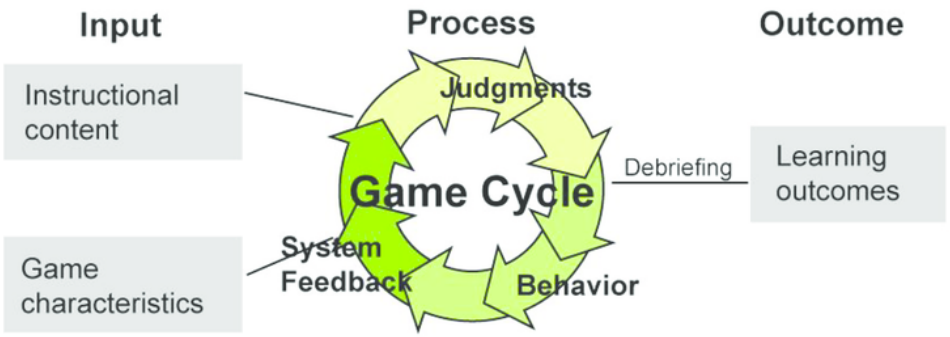

Figure 2.3: Model Game Based learning Garris et al

In order to highlight the relevance of the model to technical training, the evolution of the learning model is summarized in the table below.

Table 2.2: Pedagogical shift from Traditional Instruction to Serious Games.

Feature	Traditional Pedagogy	Serious Games (SG)
Context	Abstract; theory-first.	Situated; practice-first.
Failure	Penalized; seen as a final result.	Encouraged; seen as a feedback mechanism.
Feedback	Delayed (exams/reports).	Immediate (real-time consequences).
Model	Information transmission.	Model-based simulation of reality.

2.8 Gamification: definition, mechanics, and why it can help

After presenting the theoretical and empirical basis of Serious Games in the previous section, in this section, we will introduce another concept closely related to but distinct from Serious Games: gamification. While both concepts are frequently confused with each other in everyday conversations, they represent different design philosophies. Understanding the distinction between them is critical in order to properly understand the design choices of CyberForensics Arena.

2.8.1 Defining Gamification

However, gamification, as most people refer to it, was first defined by Deterding, Dixon, Khaled, and Nacke in 2011: "the use of game design elements in non-game contexts."[7] The idea is not to create a game but to take elements of games—points, badges, a leaderboard, a progress bar—and integrate them into a system that is not a game, such as a productivity app, a fitness tracker, or a learning platform.

This is more than just semantics. A Serious Game is, first and foremost, a game. It has rules, mechanics, goals, and a storyline all rolled together. Gamification is simply a set of incentives that happen to be game-based. The game remains the same, but the incentives get a boost from the game-based feedback and rewards.

Over the past ten years, gamification has become widespread across many industries. Marketers use gamification to encourage loyalty among customers. Businesses use gamification to encourage more people to engage with corporate training. Health and wellness apps use gamification to encourage users to be more healthy. In the field of education, gamification has been highlighted as a potential solution to the motivational challenges that technical and theoretical subjects pose.

2.8.2 Core Gamification Mechanics

Gamification is based on a set of familiar patterns, each with a particular motivational outcome. The most common gamification patterns you're likely to encounter are:

Table 2.3: Common gamification mechanics and their motivational functions.

Mechanic	Description and Function
Points	Numerical feedback awarded for completing actions. Provides immediate, quantifiable reinforcement.
Badges / Achievements	Visual icons or titles awarded for reaching milestones. Serve as status markers and sources of recognition.
Leaderboards	Rankings that compare user performance. Introduce social comparison and competitive dynamics.
Progress Bars	Visual indicators of advancement toward a goal. Reduce uncertainty and encourage persistence.
Levels / Unlocking	Gated content that becomes accessible as the user demonstrates mastery. Creates a sense of progression and discovery.
Narrative / Quests	Story driven framing of tasks. Adds context and emotional engagement to otherwise abstract activities.

It must be said that not all the mechanics are always appropriate for all contexts.

The choice of which gamification elements we want to use must be guided by the definition of learning objectives and the characteristics of the target audience.

Simply adding points and leaderboards, also known as "pointsification", may not actually help learners learn. In addition, if rewards seem trivial or if the competition seems unfair, it can actually discourage learners.

2.8.3 Psychological Foundations: Why Gamification Works

Gamification is grounded on firm foundations because it leverages what drives human behavior. The most influential framework is Self-Determination Theory (SDT) by Deci and Ryan[11]. SDT proposes that human motivation is driven by the satisfaction of three fundamental psychological needs: autonomy, competence, and relatedness.

- **Autonomy**: Feeling like you're calling the shots on your own learning process. When students feel like they're in control of their learning path instead of being controlled by it, their intrinsic motivation increases.
- **Competence**: Feeling like you're making progress and getting better at

things. When students feel like they're capable of succeeding and overcoming obstacles, their motivation increases.

- **Relatedness**: Feeling like you're meant to be around other humans and be social. When students work together on things or even in a little friendly competition, this need is fulfilled.

Gamification mechanics simply help fulfill these needs: a progress bar is a clear measure of competence, leaderboards satisfy relatedness, and branching paths satisfy autonomy.

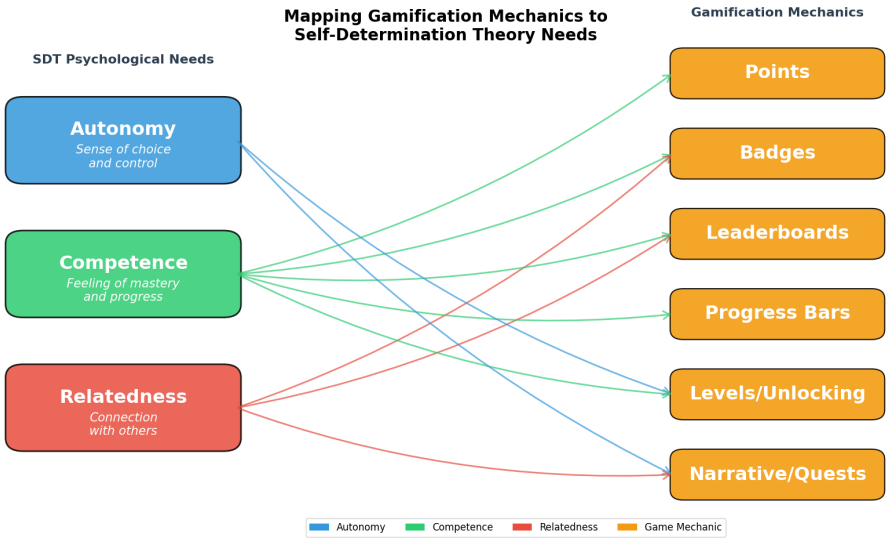

Figure 2.4: Mapping gamification mechanics to Self-Determination Theory needs.

Another aspect is the difference between intrinsic and extrinsic motivation. While intrinsic motivation is about being naturally interested or enjoying something for its own sake, extrinsic motivation is about being rewarded for it or feeling obligated to do it. Gamification is said to work on both types of motivation, but research says intrinsic motivation is more powerful and tied to deeper learning. However, it is worth noting that extrinsic motivation can be a problem when it is overused: the overjustification effect is when external rewards actually undermine intrinsic motivation that was already present.

2.8.4 Empirical Evidence and Limitations

Another line of research is focused on how gamification can influence learning and behavior. In 2014, Hamari, Koivisto, and Sarsa conducted a meta-analysis

of gamification research and found that most gamification research supported its effectiveness, particularly in increasing engagement and motivation.[12] However, it is also important to note that gamification is not universally effective and that its success depends on various factors, including the game elements used, as well as the learners and tasks involved.

Gamification appears to have great value in educational settings, particularly in tasks that students find boring and abstract. Gamification elements, such as feedback and progress, have been shown to maintain engagement and motivation in tasks where other approaches might fail. However, it is also important to note that gamification is not a panacea for poor instructional design. If the core activity is not well designed, gamification will not make it better. Gamification is an amplifier, and it can make good experiences better but cannot make poor experiences good.

2.8.5 Gamification in CyberForensics Arena

Following on from the above ideas, CyberForensics Arena combines elements of Serious Game design with some thoughtful touches of gamification. First and foremost, it is a Serious Game in which the learner takes on the role of forensic investigator and enters scenarios in an immersive 3D world. However, on top of this, there are some elements of gamification.

Table 2.4: Gamification elements implemented in CyberForensics Arena.

Mechanic	Implementation in CFA
Points	Task completion awards points based on task difficulty. Points are validated server-side and contribute to the user's cumulative score.
Badges	Achievements awarded for completing scenarios (20 points), completing scenarios within a time threshold (speed runner, 30 points), and completing scenarios without using hints (hint-free expert, 30 points).
Leaderboard	A real-time leaderboard ranks users by total score, combining task points and badge points. Caching strategies ensure scalability without sacrificing responsiveness.
Progress Bars	A visual progress indicator displays the number of completed tasks relative to the total within each scenario, providing immediate feedback on advancement.
Hint Economy	Hints are available for each task but require spending accumulated points to unlock. This creates a strategic tradeoff between immediate assistance and long-term score maximization.
Task Structure	Each scenario is decomposed into discrete, sequential tasks with clear objectives, enabling checkpoint style progression and reducing cognitive overload.

The mechanics were chosen with the learning goals in mind, as well as the realities of the classroom. The points and badges system offers a clear demonstration of competence, rewarding specific skills such as speed and working without hints. The leaderboard element offers a social component, allowing students to compare their contribution with others, catering to the need for relatedness. At the same time, focusing on overall points rather than who is quickest makes it easier for new learners. The hint economy offers a further layer of strategy, allowing learners to choose when they want to spend points in return for a hint, allowing them to maintain autonomy.

In addition, narrative framing is used throughout the system, with role play at work as well. The idea of the learner as a forensic investigator, with a clear context, increases engagement as well as understanding, particularly with tasks that could be seen as abstract. The narrative element is used in conjunction with the gamification mechanics, which increases purpose as well as a sense of ongoing progression, helping to maintain motivation during longer periods of engagement.

In short, gamification in CyberForensics Arena is not an afterthought, but a design element with clear purpose. The mechanics used complement the immersive nature of the Serious Game foundation, offering a means of sustaining engagement, demonstrating progression, as well as reinforcing the skills used in digital forensics education.

Chapter 3

Project and Game Design: Cyber Forensics Arena

3.1 Main Idea and Role of Human Factors

The underlying principle of CyberForensics Arena is that digital forensics learning cannot be achieved through passive instruction. Real investigators do not just run tools, they develop theories, tie different pieces of evidence together, and adjust theories as new evidence arrives. The learner is thus presented as an active forensic investigator working in a simulated crime scene, and not as a student passively receiving instructions.

The user enters in the enviroment as a forensic analyst whose task it is to examine digital devices that are under seizure. This role play framing has a pedagogic purpose that can be situated in the theory of situated learning: acquisition is more powerful of learners being involved in concrete exercises within an authentic context, rather than decontextualized exercises[13]. The 3D office space, interactable evidence items and terminal based analysis tools accomplish this; in doing so they render nonspecific technical content into concrete investigative actions.

3.1.1 The Human Factor Role

The design was grounded on principles of human factors

- **Cognitive Load Management**. Digital forensic analysis is a mentally demanding task: an investigator needs to keep track of abstractions of system states, evidence chains and think about related time-ordered sequences. To avoid cognitive load, CyberForensics Arena decomposes an investigation task into several focused tasks. Each lesson offers targeted instruction and gradually moves toward sophisticated analysis.[14]

- **Scaffolded Autonomy**. New learners need support to prevent frustration, but too much guidance robs them of opportunities to become independent problem solvers. The architecture of the platform addresses this tension via progressive difficulty: whereas initial tasks are explicitly provided with instructions, further tasks require learners to apply transferred knowledge. There is a hint system to give you some help if you need it, but at the cost of points, so there's an incentive to work independently before getting unnstick.
- **Sustained Motivation**. Forensic drills can appear abstract or boring unless linked to some outside meaning. The investigative story provides outside meaning for procedures that would otherwise seem meaningless. Mounting a disk image becomes like gathering evidence from a suspected breach. The gamification components, which are discussed in Section 2.8, help sustain interest through feedback loops.
- **Onboarding and Accessibility**. Beginning users are guided through an extremely interactive tutorial that shows movement, object interaction and console use before letting you try out a 'real' scenario. That means there are no technical barriers to the attainment of learning goals and users with all backgrounds can use and benefit from our platform.

Taken together, these considerations seek to balance between a learning environment that is rigorous but supportive, one that teaches investigative thinking as opposed to recitation of procedures.

3.2 From Learning Objective to Task Mapping

One of the challenges when designing educational games is making sure that game elements do actually serve learning goals, and not just act as window dressing for obviously non-gamey exercises. This section illustrates how CyberForensics Arena links game tasks to digital forensics capabilities, and pedagogically aligns them at multiple levels.

3.2.1 Core Learning Objectives

The focus of the platform is on the competencies that are taught in existing digital forensic curricula and professional certification standards. These goals include both technical abilities as well as investigative techniques:

1. **Evidence Integrity**: understanding the importance of the golden rule: hash and forensic copy. (Hash first analyse second) "Data never goes to waste" (MD5/SHA256).

2. **Forensic Imaging**: Creating forensically sound disk images using forensics grade tools (e.g., `dd`) to allow for analysis without affecting source media.
3. **File System Analysis**: Browsing and recovering data from mounted disk images, understanding directory structure, storing metadata.
4. **Log Interpretation**: Analyzing and correlating system logs (authentication, application, network) to recreate event timelines.
5. **Network Traffic Analysis**: Packet captures analysis for discovery of Abnormal connections, data exfiltration and attacks signatures.
6. **Memory Analysis**: Pulling artifacts from the memory dumps, e.g., active processes and artefacts of potential malware.
7. **Evidence Correlation**: Aggregating findings from different artifacts to establish a consistent investigation result.

3.2.2 From Objectives to Tasks

One or more tasks in the scenarios of the platform itself operationalise each learning objective. This mapping for some exemplary examples is shown in Table 3.1.

Table 3.1: Mapping of Learning Objectives to Game Tasks

Learning Objective	**Game Task**	**Required Action**
Evidence Integrity	"Hash the original evidence"	Execute `sha256sum` on seized device before copying
Forensic Imaging	"Create a forensic copy"	Use `dd` to image device to working directory
File System Analysis	"Mount and explore the image"	Mount image read-only, navigate with `ls`, `cat`
Log Interpretation	"Identify the attacker IP"	Analyze `auth.log` entries to find unauthorized access
Network Analysis	"Trace the suspicious connection"	Use `nmap` results and PCAP analysis to identify C2 server
Memory Analysis	"Find the malicious process"	Examine memory dump for process with known malware signature

3.2.3 Progressive Complexity

Within each scenario, tasks are organized according to a progression from concrete, tool-oriented activities on one end of the spectrum toward abstract reasoning tasks at the other.

1. **Physical Interaction**: Connect devices, power on equipment (establishes context.
2. **Procedural Compliance**: : Hash evidence, create forensic copies (reinforces Methodology)
3. **Technical Execution**: Run analysis commands, interpret output (Develops Tool Proficiency)
4. **Analytical Synthesis**: Draw Conclusion from multiple Artifacts (Cultivates Reasoning).

This progression ensures that learners are able to master basic procedures before being given any that require independent judgment. Initial tasks require learners to "follow specific instructions like 'Run `sha256sum` on the device,'" while later tasks encourage the reader to make inferences, such as "Determine the timestamp of initial compromise from the available evidence."

3.2.4 Validation Without Exposure

The validation of the tasks is a problem that must be addressed during the design process without allowing the correct answers to be known to the learners. CyberForensics Arena achieves this by verifying the answers on the server side; that is, the client sends the answers to the server, where they are compared with the correct ones that were previously defined in the scenario. This way, the learners cannot try to get the answers from the browser storage or the network traffic, which is important from the point of view of the integrity of the learning process and the correct handling of the submitted work.

The validation system offers various types of answers: such ,

- exact string matches of command sent in trough the console, necessary when providing exact values like a hash digest;
- command string pattern matching, needed when a flexible answer is required (rg: ip addresses in various formats);
- flag based challenges (Capture the Flag style), where the learner must discover/identify hidden strings within a simulated enviroment.
-

3.3 Scenario and Narrative Design

A forensic investigation is a story made from evidence. The investigator doesn't just run programs and write down what comes out, they make and test hypotheses, follow leads and piece together a narrative of what happened. If learning methods aren't wrapped in stories, they simply become disconnected exercises, a type of reasoning that merely requires training. To motivate and to set the context for learning, CyberForensics Arena writes 241 every task in a story-level narrative that provides learners with motivation, context, and direction.

3.3.1 The Role of Narrative in Forensic Training

Studies in narrative education support that when instruction is organized within stories, there are improved levels of understanding and retention.[**schank1995tell**] In digital forensics, narrative framing serves multiple purposes:

- **Contextualisation of procedures.** Technical actions, such as hashing a disk image with cryptography, can seem quite random to an amateur. When the same action is explained in terms of ensuring the authenticity of evidence that was collected from a suspected insider, now, the learner knows *why* this step matters in an investigation.
- **Sustained engagement.** A narrative approach motivates students to keep on walking through tasks, as they receive more knowledge of the event with each step. This is in line with the inspirational design from Section 3.1.
- **Investigative reasoning practice.** By tracing a story, students are rehearsing the same cognitive moves as archaeologists: making conjectures based on incomplete evidence, choosing which trails to investigate further and revising explanations when new finds undermine initial interpretations.

All challenges in CyberForensics Arena are constructed based on a fictitious but plausible security incident. The story is delivered by a briefing that offers an orienting sense of the surroundings, an explanation about what sparked it in the first place, and what's going to be required from the analyst. As the learner completes tasks, a story will unfold, facts come to light, connections become obvious, and the big picture of the attack chain takes shape. This follows the natural progression of a real digital forensics investigation, in which the investigator typically begins without all the facts and gradually builds up a full understanding of the events.

3.3.2 Scenario Design Principles

Each scenario is designed with four principles in mind to keep the learning experience cohesive and the investigation as realistic as possible:

- **Authenticity**: The scenarios are based on real attacker behaviors described in threat intelligence reports and real-world forensic experience. For example, the network scenario follows the attacker from brute-force login to stealing credentials, escalating privileges, and exfiltrating data via DNS tunneling. In memory forensics, the attacker's actions focus on fileless malware using process injection and beaconing to command and control, indicating a trend towards "living off the land" attacks with minimal disk presence.
- **Progressive disclosure**: The student is never given all the pieces to solve the problem. Rather, they uncover one layer at a time, such as a log entry indicating a suspicious IP address, a network scan showing which services are exposed, or a capture showing the exfiltration channel. This mimics real-world investigation, in which we seek, interpret, and connect the pieces of evidence rather than merely consume them.
- **Multi-source correlation**: It's not as if criminal trials typically get hinged on a single piece of evidence. CyberForensics Arena scenarios are engineered such that students have to merge the information from these disparate sources of evidence — event logs, network traffic captures, memory dumps, file system artifacts – to make inferences. In the network-based scenario, for instance, the learner needs to match SSH authentication logs with firewall logs and also analyze packet captures in order to piece together what happened during full lineage of events. This teaches learners to think beyond evidence "jurisdictions", a distinction that often separates those who are effective investigators from those who just use tools.
- **Forensic procedure compliance**: Each one of these scenarios enforces of the basic principles of digital forensics: hash before examining, make forensic images and only work on a duplicate and not on the original evidence. In any case, these process steps are not optional and are specified as necessary step to perform before one is allowed to access the analysis phase.

3.3.3 The Three Investigation Scenarios

CyberForensics Arena includes three scenarios, each representing one of the investigation archetypes from Chapter2. They also address the three most important aspects of digital forensic analysis: storage (disks and file systems), network, and data from volatile storage. Each one of these scenarios uses different storytelling, a

different set of simulated forensics tools and a specific evidence terrain. Table3.2 provides a high-level comparison.

Table 3.2: Comparison of the three investigation scenarios in CyberForensics Arena.

	File System	**Network**	**Memory**
Narrative	Seized disk from insider threat	APT intrusion at financial firm	Fileless malware at trading firm
Initial trigger	Employee suspected of data exfiltration	IDS alert: anomalous outbound traffic	Unauthorized trades detected
Evidence sources	Disk image, file system artifacts, metadata	Authentication logs, firewall logs, PCAP	RAM dump, process memory, registry hives
Key tools	`sha256sum`, `dd`, `mount`, `stat`	`nmap`, `tshark`, `whois`	Volatility plugins (`pslist`, `malfind`, `netscan`, `dlllist`)
Tasks	18	14	15
Points	400	325	340

Scenario 1: File System Forensic Investigation

The first scenario is designed to give the learner a basic insight into digital forensic methodology. At the heart of that story: a confiscated hard disk from one employee who allegedly exfiltrated data. The disk has been wiped but perhaps a little forensic work will see it unravelled and shamed.

This hypothetical case highlights the fundamental forensic process which applies to all subsequent cases as well:

1. **Physical acquisition:** the learner connects the seized disk to the forensic workstation by interacting with the 3D environment.

2. **Integrity verification:** the original device is hashed using `sha256sum` to create a chain-of-custody baseline.

3. **Forensic imaging:** a bit-for-bit copy is made using `dd`, which is then hashed to verify it matches the original.

4. **Read-only analysis:** the forensic image (never the original) is mounted and examined.

Once this basic process is complete, the situation progresses to file system analysis techniques such as file metadata analysis using `stat`, analyzing slack space for hidden fragments of files, reading deleted files using file carving techniques to find them and identifying camouflaged files using magic number identification. The investigation itself ends with the learner piecing together a timeline of the suspect's actions and determining how data was removed.

This is the "first pass" that a learner gets, so the descriptions of tasks are more explicit and the guidance that we give them is mor e direct. Scenarios that comes later reduce the level of instrunction given, requiring the learners to use the procedural knowledege they have just developed.

Scenario 2: Network Forensic Investigation

The second one changes the evidence type domain: it goes from disk artifacts to network level evidence. The story, *Operation Shadow Gate*, begins when the learner is exposed to a reported incident at a bank where an IDS has alerted on traffic that appears to be encrypted and exhibits anomalous sending outbound to what seems like be an APT style intrusion.

The investigation is structured around a realistic attack lifecycle:

1. **Reconnaissance and target identification:** the student starts network discovery to find the compromised server and uses `nmap` to scan it for open services.

2. **Evidence preservation:**prior to analysis the learner hashes the log evidence in order to maintain chain of custody.

3. **Log correlation:** the brute force pattern and the moment they break in is revealed by authentication logs, and firewall logs reveal attacker lateral movement.

4. **Deep packet analysis:**via `tshark)`, the learner analyzes a packet capture identifying exflitrating data, DNS tunneling payloads, and Command and Control (C2) traffic.

5. **Attribution and threat intelligence:** through `whois` lookups and traffic pattern analysis, the student is able to tie this attack to known threat actor infrastructure.

This scenario highlights the importance of multi-source correlation as a fundamental student skill. Typically, one piece of evidence will not suffice to describe

an entire attack; the student needs to stitch together log files, network captures, and scan output. The scenario includes false indicators of compromise, such as real traffic mixed with attack indicators, to make it difficult for the student to differentiate signal from noise.

Scenario 3: Memory Forensic Investigation

The third scenario, Operation Phantom Thread, represents the most difficult analysis problem. A quantitative trading firm discovers unauthorized trades on its platform. The initial disk forensic analysis did not reveal any signs of malware; this is a fileless attack using only volatile memory. The student needs to analyze 8GB of RAM dump to reveal the entire attack chain. Two pedagogical goals make memory forensics analysis different from the other domains:

- **Volatile evidence awareness.** Memory evidence is not persistent, and it will be lost when the system is shut down, unlike disk evidence, which remains static and unchanging. This scenario highlights the importance of RAM dump evidence, which is obtained through live acquisition with LiME, a live acquisition tool.
- **Advanced adversary techniques.** The techniques used by the adversary, such as spear phishing, PowerShell download cradles, process hollowing, and beaconing, are not visible through disk forensics, and learners must be aware of these techniques and how to identify them through memory analysis techniques.

The investigation process will be carried out through the use of simulated plugins using the Volatility framework, which is considered the industry standard for memory forensics. Learners will use `vol-imageinfo` to identify the OS profile, `vol-pslist` to identify running processes and detect unusual process behavior, `vol-malfind` to identify injected code, `vol-netscan` to identify connections to command and control servers, and `vol-dlllist` to identify DLLs loaded from untrusted locations, with the results compared to the documentation to assess their progress.

A known good list of running processes is given to the learner, and they must identify the differences. This is a real world skill because the learner must identify what is normal and what is not.

This is similar to the previous scenarios in terms of the workflow. The learner must hash and image the evidence prior to analysis and perform all analysis on the forensic copy. This reinforces the main rule that evidence handling is the same regardless of whether the evidence is volatile or non-volatile.

3.3.4 Narrative as a Design Tool

But beyond the goal of motivating the learner, the scenario stories serve another purpose: they constrain the design of the clues provided, the tool outputs presented to the learner, and the tasks given to the learner. If the scenario is about a brute-force SSH attack, the authentication log will contain a realistic pattern of several failures followed by a success, with timestamping that is internally consistent, and the IP address of the attacker appearing consistently in multiple log entries.

This is important on two levels. First, it helps the learner develop a realistic mental model of the investigation process. If the clues provided were unrealistic or contradictory, the learner's mental model of the investigation process would be flawed. Second, it helps the server-side validation of the learner's answers. Because all the clues provided are realistic and consistent, there is only one correct solution to any given problem, and the system can validate that solution unambiguously.

The scenario stories also influence the difficulty progression of the tasks provided to the learner. The early tasks are procedural and well guided (for example, "Run sha256sum on the device"); the latter tasks require the learner to integrate several evidence types (for example, "Determine the initial compromise timestamp by cross-referencing process start times with log entries"). Again, this is not an arbitrary progression; it is a realistic progression from evidence collection to attribution.

3.4 Game Structure: Missions, Levels and Tasks

The content of CyberForensics Arena is organized in three levels: a set of *missions/scenarios* on the platform, where each mission is divided into *phases*, and each phase consists of specific *tasks*. This establishes the connection between the narrative level, as discussed in 3.3, and the gamification mechanics, as discussed in Subsection 2.8.2.

3.4.1 Overall Hierarchy

At the top level, there are three missions, each linked to a different archetype of digital forensic investigation. You can jump into any mission by yourself, as each one is independent, although there is a suggestion to do them in an order of increasing difficulty: file system, network, memory.

However, within each mission, tasks are presented in a strict linear fashion. That is, there is no option to do tasks out of order. However, this is not just a design choice; it is an important one. In real-world digital forensics, certain steps have to be done first. For example, it is not possible to examine a disk image before it has been created, or to examine any files before mounting an image. This linearity

ensures that the logical order is maintained, guaranteeing that no steps in a mission can be skipped.

Lastly, as is true in any mission, tasks are divided into four phases, as illustrated in Table 3.3. The phases reflect a real-world digital forensic investigation, as well as the progressive complexity model that was introduced earlier in Section 3.2.

Table 3.3: Task phases within each mission.

Phase	**Typical tasks**	**Purpose**	**Example**
Evidence acquisition	1–5	Establish physical access and preserve evidence integrity	Connect device, hash original, create forensic copy
Tool-guided analysis	5–10	Execute forensic tools and interpret direct outputs	Run `vol-pslist`, examine file metadata with `stat`
Cross-source correlation	8–12	Combine findings from multiple evidence sources	Compare process list against baseline, match log entries with packet captures
Synthesis and attribution	12–15	Draw conclusions from accumulated evidence	Identify the initial compromise timestamp, determine the persistence mechanism

3.4.2 Task Anatomy

All CyberForensics Arena tasks share the following common design elements:

- **Title and description**: The title is short and indicates the investigation goal, such as "Hash the original evidence." The description is longer and provides context, guidance, and sometimes background information, as needed.
- **Point Value**: Every task has a point reward, which is proportional to the difficulty of the task. Less demanding procedural tasks are worth 10-15 points, while analysis and synthesis-oriented tasks are worth 25-35 points.
- **Completion mechanism**: This is what the system does to determine if the learner has successfully achieved a task, which was illustrated in Section3.4.3.

- **Optional hint**: A hint is offered as an optional guide when learners are stuck, at the expense of points, as further elaborated in Section 3.5.1.

This uniform structure makes possible for any task to be represented using the same interface, regardless of the content or the way it is validated.

3.4.3 Task Types

The CyberForensics Arena has four different types of tasks, each of which focuses on one particular skill. The types of tasks used at each stage of the investigation are determined by the type of investigative action they represent.

- **Interaction tasks**. For interaction tasks, you have to physically interact with objects in a 3D scene, such as clicking on a hard disk to connect it to the forensic workstation or powering up a server to start the network analysis. The investigation is grounded in the physical world, creating a spatial setting before delving into the virtual world of computers. Every mission starts off with an interaction task, which initiates the process of acquiring the evidence.
- **Command tasks**. For command tasks, you have to enter a specific command in the simulated command line of the forensic workstation. The system verifies the name of the command as well as the arguments used, such as `sha256sum /dev/sdb` or `mount /forensic/memdump.img /mnt/memdump`. Command tasks are used for actions in the process of acquiring the evidence where the exact tool command is necessary, such as hashing, imaging, mounting, or scanning.
- **Flag tasks**. The flag task is based on the Capture the Flag (CTF) concept, where the player is given an open-ended question that requires an exact answer, such as an IP address, process ID, file name, or time stamp. The answer cannot be found in the output of any command, requiring the player to analyze the results of the commands given in the scenario, draw conclusions, and obtain the answer. Flag-type tasks appear later in the mission.
- **Mini-game tasks**. There are some activities in the process of acquiring the evidence that can better be represented as an interactive visual puzzle rather than a command-line task. CyberForensics Arena has two types of mini-game tasks:
  - **Signal tracing**, which is a visual puzzle involving tracing connections to identify a compromised system, is used as part of network forensics cases.
  - **Decryption challenge**, which is a pattern matching puzzle simulating the process of analyzing encrypted information, is used as part of malware analysis cases.

Mini-games are used to introduce variety and prevent the monotony of console based interactions. Mini-games allow the platform to depict processes that would be difficult to implement using a plain text terminal. Examples include pattern recognition or thinking about the topologies of networks in space. Figure 1 (Figure 3.1) shows the distribution of the four task types over the three missions.

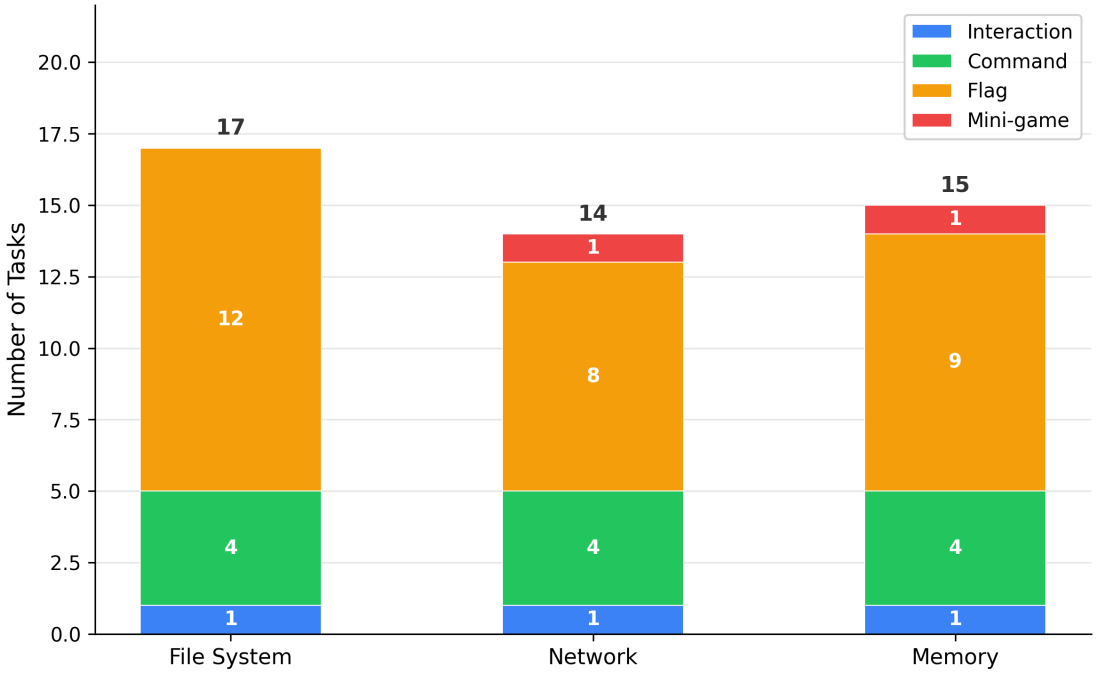

Figure 3.1: Distribution of task types across the three investigation missions.

3.4.4 The Tutorial: Guided Onboarding

Before diving into any given forensic scenario, first-timers are presented with a tutorial that explains the fundamental interaction paradigms of the platform. This is not a reduced mystery experience; it's more like a step-by-step explanation to familiarize yourself with:

- Moving within 3D space with keyboard and mouse;
- Interacting with objects within the scene (approaching objects, clicking on objects, reading the feedback);
- Using the forensic console to type commands and reading the output;

- Familiarizing yourself with the task panel to understand what you are currently trying to accomplish and how it's going.

The tutorial is linked to each user and will appear when they first log in. However, if a user is already familiar with the interface, he can bypass it. This is a good balance between making sure new users get a good introduction and not forcing experienced users to go through steps he doesn't need to know. After the tutorial is complete or skipped, it will be recorded and next time log in directly to the mission selection screen.

3.4.5 Mission Lifecycle and Persistence

A mission within CyberForensics Arena follows a predictable lifecycle:

- **Briefing**: Before the first task appears, the learner is given a scenario introduction, a narrative-style briefing that provides the context for the investigation. This introduction is only displayed to the learner when first entering the scenario and is not repeated later.
- **Progressive task completion**: The user completes tasks sequentially. When a user completes a task, they are provided with instant feedback (task level alerts), and the next task becomes visible. This ensures that when a user completes a task, their progress is saved on the server. This means that a user can start a mission and leave after completing some tasks. When they return to the system, they can pick up where they left off with their session.
- **Completion of a mission**:he user has completed all tasks in a mission, given a badge representing one modern problem and the relevant domain of knowledge within cyber forensics.

In this model, learning is organized with the possibility of interruptions of any duration. This matters because the sessions could be as long (or short) as the student wants them to be, which can be problematic when trying to figure out how it would fit into a traditional educational schedule..

3.5 Assessment, Progression and Data Collection for Learning Analytics

A learning platform which does not have any means to measure what its users are doing cannot scale. With CyberForensics Arena, we provide a full event logging system which caters to two different user groups. Learners can get instant feedback and review their own progress, while instructors can review aggregated data to

assess class performance, identify common problems, and refine the scenarios over time.

3.5.1 Scoring Model

On CyberForensics Arena, players can score points by accomplishing missions and they can use the points to buy hints. There are certain points determined and assigned to every task according to the level of difficulty it comprises.

For instance, hashing evidence and imaging could be weighted at 10 to 20 points (procedural activities) and identifying attack vectors and constructing the timeline of compromise could be weighted at 25 to 35 points (analytic activities).

To obtain a hint about a task, the learner's general score is decreased by some predetermined amount. This makes sure that students have a true choice to make. Learners receive hints for the task, but at a cost.

Upon completion of all tasks within a scenario, the learner is rewarded with a scenario badge, which is a permanent recognition of expertise within the relevant domain of forensic science (e.g., "*File System Forensic Expert*," "*Network Forensic Expert*," "*Memory Forensic Expert*"). Two further badges are awarded for other aspects of the learner's performance: Speed Runner (completing a scenario within five minutes) and Hint-Free Expert (completing a scenario without hints). Badges contribute their own point value to the learner's overall score.

3.5.2 Progression and persistence

The learner's progress is saved on the server. The completion of tasks, score, hints, badges, and Virtual File System state are stored in the database and are maintained across interrupted sessions. If the learner closes the browser during a scenario, they are guaranteed to resume exactly where they left off, with the same state of the Virtual File System, the same task index, and the same score. This is important for use within a classroom setting, where sessions may be interrupted by various factors.

The learner progresses through a scenario linearly, i.e., task n must be completed before task $n + 1$ is available, as discussed under Game Structure. Scenarios are independent, i.e., completing the network scenario does not affect the learner's progress within the memory scenario. The learner's overall profile aggregates information from all the scenarios, including the number of task completed, the accumulated store and earned badges.

3.5.3 Event Logging and Data Collection

All the important interactions in CyberForensics Arena are logged as an event. The platform has nine event types, as summarised in Table 4.3.

Table 3.4: Event types recorded by the tracking system.

Category	Event type	Data captured
Scenario	`scenario_start`	Scenario code, timestamp
	`scenario_end`	Completed tasks, total tasks, total score
Task	`task_submit`	Task ID, correctness, submitted answer
	`flag_submit`	Task ID, correctness, submitted flag
Hint	`hint_request`	Task ID, hint cost
Command	`command_execute`	Full command string, error status
Mini-game	`mini_game_start`	Game type, scenario
	`mini_game_complete`	Game type, success status
	`mini_game_fail`	Game type

Each event has the following fields:

- **participant ID**;
- **optional user ID (if the user is logged in)**;
- **event type**;
- **optional scenario code**;
- **optional task ID**;
- a **JSON object** containing event-specific data;
- **the timestamp**

This allows for various ways to view the data: per-user, per-scenario difficulty, etc.

The use of the command execution log is especially significant for forensic education. Logging occurs at each and every command entered in the console, including incorrect or malformed commands, in a manner which captures the thought process of the student rather than merely the final correct answer. Because the data in this dataset show many misconceptions, including analysis of original and not simulation copy, or the choice of an inappropriate tool for the task.

3.5.4 Anonymous Participant Tracking

To strike a balance between analytics and privacy, CyberForensics Arena takes an approach of dual identity. Users who are logged in will be identified by their user ID, which is associated with their account and visible to the instructor. However, all client instances, irrespective of whether they are logged in or not, will be assigned an *anonymous participant ID* in the format of `CFA-XXXXXX`. This ID will be generated after the first visit and will be stored locally in the browser.

This allows for data collection for unauthenticated users, for example, when giving public demos or evaluating anonymously. It will also help differentiate between users when they share the same device. The participant ID will be included along with the optional user ID in each event record.

3.5.5 Instructor Dashboard and Analytics

In addition, instructors who have admin access can view platform usage through a special dashboard. The dashboard has four main views:

- **Aggregated Statistics**:A top-down perspective of the data going around the platform. These are: total users registered, last 24h active users, total logged events, completed tasks, commands executed, hints requested and a usage trend of the last seven days.
- **Event Log Browser**: Avent Log Browser - A view that shows all logged events and allows searching and paging through the log. Logs can be filtered by event type, scenario, ID of participant, user and time it belongs to. Each event log entry provides full information collected for an instructor can be able to examine individual learner sessions.
- **User Directory**: A list of all registered users, including some statistical information such as tasks completed, total score, events completed, and last active time. Selecting a user will show detailed information for this learner, including per-scenario completion records, badges earned, command success rate, hints taken, and last active time.
- **Data Exports**: Export logs as CSV to review data outside the platform. Exports may be limited based on event type, scenario, participant id, user and time range (as in the event log browser). This feature enables an instructor to export some slices of the data and analyze them further (e.g., command executions on a particular scenario, task submissions from a given group over an interval of time).

The analytics environment supports the day-to-day teaching, and a wider pedagogical investigation into how scenarios can be better decoded, how levels of difficulty can be set and what are normal patterns in digital forensics learning.

Chapter 4

System Architecture and Implementation

4.1 Overview of the Cyber Forensics Arena Platform

CyberForensics Arena is a web-based serious game platform implemented as a client-server application. The client-side application is built as a Single Page Application running entirely in the browser, combining a 3D world in real-time with an interactive terminal emulator. The server-side application has a REST API that handles authentication, task validation, virtual filesystem management, and event logging. The client-server application interacts over HTTP with session cookies, with no page reloads in regular usage. Table 4.1 summarises the main technologies used in each layer of the system.

Table 4.1: Technology stack of CyberForensics Arena.

Layer	Technology	Role
3D Rendering	Babylon.js 5.x (WebGL)	Real-time 3D scene, camera, collision, highlights
Terminal emulator	xterm.js + xterm-addon-fit	In-browser forensic shell
Frontend build	Vite	Module bundling, hot reload, static asset serving
Backend runtime	Node.js + Express.js	REST API, middleware, routing
Database	SQLite (`sqlite` package)	Persistent storage for users, progress, events
Session store	express-session + connect-sqlite3	Server-side session management
Authentication	Passport.js (local strategy)	Username/password login with bcrypt
Security headers	Helmet.js	HTTP security headers (CSP, HSTS, etc.)

SQLite was chosen over a client-server solution such as PostgreSQL because of the lack of need for a separate process, configuration, and the end result of having a single portable file. This is particularly beneficial in an academic environment where support is often lacking. The drawback is that SQLite is not well suited for high concurrency write operations, but in the environment of the platform where there are dozens of concurrent users in a classroom setting, this is not a concern in practice.

The remainder of this chapter is structured as follows. Section 4.2 provides an overview of the client-server architecture and the communication flow between the client and server. Section 4.3 provides an overview of the three main backend systems of the platform: the database schema, the virtual file system, and the scenario engine. Section 4.4 provides an overview of the frontend implementation of the platform, the 3D scene, the forensic console, and the mini-game system.

Finally, Section 4.5 provides an overview of the security and privacy features of the platform.

4.2 System Architecture: Client–Server Design and Main Components

The CyberForensics Arena follows a traditional client-server model in which the separation between the browser-based client and the Node.js-based server is clear. The client is responsible for rendering, user interaction, and interface handling, whereas the server is the only place for validation, scoring, and storage of data. The only way in which the two interact is through a REST API, with no business logic or solution data being sent to the client.

4.2.1 Communication Model

The client and server communicate over HTTP and utilize *session cookies* to store state about the authentication. Once the client has logged in, the server will store the session in memory and send back an `httpOnly` cookie on the client. All future API calls will have this cookie included, allowing the server to identify the user without requiring any token management on the client.

CORS configuration on the server allows only requests from the client's origin and sets `credentials: true` to enable cookies.

In production, `secure: true` and `sameSite: 'none'` are used on the cookie to enable cross-origin usage of the client and server, such as when the client is hosted on a CDN and the server is on a different host.

4.2.2 Client-Side Architecture

The client is a single-page application (SPA) implemented with vanilla JavaScript and packaged with Vite. The application does not reload any pages, and the transitions between states (login, tutorial, scenario selection, active investigation) are handled entirely by JavaScript, changing the DOM and the Babylon.js scene accordingly.

In order to keep the codebase clean and easy to maintain, the client is divided into three loosely coupled layers, as depicted in Figure 4.1.

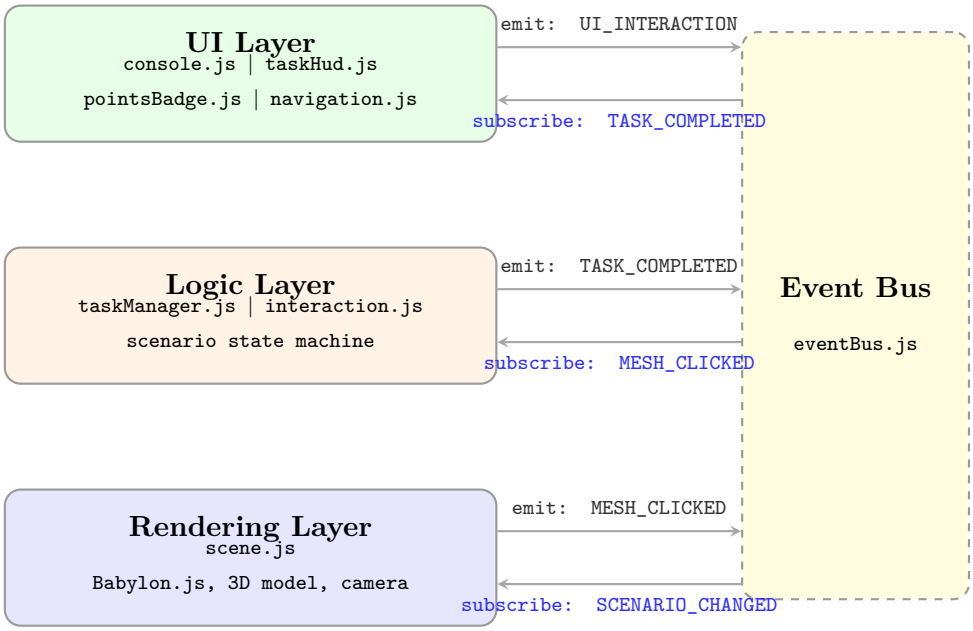

Figure 4.1: Three-layer client architecture decoupled via Publish-Subscribe Event Bus.

- **UI Layer** (`console.js`, `taskHud.js`, `pointsBadge.js`, `navigation.js`). This layer is the 2D face of the app. It controls DOM-based UI elements such as the interactive forensic terminal, the task progress HUD, and the scoring display. Essentially, it's just there to watch for changes in state, update the display, and handle user input that isn't on the 3D canvas.
- **Logic Layer** (`taskManager.js`, `interaction.js`). This layer is the heart of the client. It runs the game's state machine. It controls the flow of the scenarios, checks tasks by retrieving context from the server's API, and manages the results of user interactions in the virtual world.
- **Rendering Layer** (`scene.js`). : This layer is entirely about the WebGL engine. It's just about the game engine itself, which in this case is the Babylon.js library. It's sole concern is managing the 3D space, loading assets, controlling the cameras and lighting, calculating collisions, and applying highlights to objects that are interactable. It knows nothing about the game logic.

One of the design decisions made here is to use a lightweight `publish-subscribe` pattern to ensure that the different layers remain decoupled. This pattern is achieved through a central Event Bus (`eventBus.js`). Instead of using direct imports or function calls to bind the different modules, the different components interact by

broadcasting globally defined events such as `SCENARIO_CHANGED`, `MESH_CLICKED`, `TASK_COMPLETED`, and so on, and listening to the events that relate to them.

This asynchronous message-passing mechanism effectively blocks the creation of circular dependencies. For instance, when the Rendering Layer recognizes that a 3D mesh has been clicked, it simply broadcasts the `MESH_CLICKED` event. The Logic Layer can then independently choose what to do next and broadcast the `TASK_COMPLETED` event. The UI Layer will then respond to the `TASK_COMPLETED` event to refresh the objective tracker, but there has been no direct communication between the different layers. The specifics of the publish-subscribe pattern are illustrated in Listing 4.1, which presents the specifics of the `EventBus` class, and Listing 4.2, which presents the specifics of the interlayer communication.

```
1  class EventBus {
2    constructor() {
3      this.listeners = new Map();
4      this.eventHistory = []; // rolling debug log (max 100 entries)
5    }
6
7    // Subscribe: returns an unsubscribe function
8    on(eventName, handler) {
9      if (!this.listeners.has(eventName))
10       this.listeners.set(eventName, []);
11     const handlers = this.listeners.get(eventName);
12     handlers.push(handler);
13     return () => { handlers.splice(handlers.indexOf(handler), 1);
       };
14   }
15
16   // One-time subscription: auto-unsubscribes after first call
17   once(eventName, handler) {
18     const unsub = this.on(eventName, (...args) => {
19       handler(...args); unsub();
20     });
21   }
22
23   // Publish: stores event in history, then calls all handlers
24   emit(eventName, data = null) {
25     this.eventHistory.push({ name: eventName, data,
26                              timestamp: Date.now() });
27     const handlers = this.listeners.get(eventName) ?? [];
28     for (const h of handlers) h(data);
29   }
30 }
31
32 export const eventBus = new EventBus();
```

Listing 4.1: Core of the `EventBus` class (`eventBus.js`).

```
// --- Rendering Layer (scene.js) ---
// Publishes when the user clicks a 3D object
canvas.addEventListener('click', () => {
  eventBus.emit(Events.MESH_CLICKED, { meshName: pickedMesh.name
   });
});

// --- Logic Layer (taskManager.js) ---
// Subscribes to mesh clicks; publishes task result to UI
eventBus.on(Events.MESH_CLICKED, ({ meshName }) => {
  const result = validateInteraction(meshName);
  if (result.correct)
    eventBus.emit(Events.TASK_COMPLETED, { score: result.score });
});

// --- UI Layer (taskHud.js) ---
// Subscribes to task completion; updates the HUD display
eventBus.on(Events.TASK_COMPLETED, ({ score }) => {
  hudScoreElement.textContent = `+${score} pts`;
  advanceHudToNextTask();
});

// --- Auth flow (main.js) ---
// One-time subscription: unblocks init after first login
eventBus.once(Events.AUTH_SUCCESS, () => resolve());
```

Listing 4.2: Event bus usage across all three layers.

4.2.3 Server-Side Architecture

The server is an Express.js application whose entry point (`server/src/index.js`) assembles the middleware stack and mounts the route modules; 4.2 summarises the main route modules and their responsibilities.

Middleware stack (applied in order to every request):

1. `helmet`: sets HTTP security headers (Content Security Policy, HSTS, X-Frame-Options, etc.);
2. `compression`: gzip response compression;
3. `cors`: restricts cross-origin requests to the configured client origin;
4. `express.json`: parses JSON request bodies;
5. `express-session` + `connect-sqlite3`: server-side session management;
6. `passport`: deserialises the authenticated user from the session;

7. `extractParticipantId`: reads or generates the anonymous participant header for event tracking.

The **rate limiting** for each route group can be defined using the express-rate-limit package, depending on the rate at which the specific route is being used:

- Authentication routes: 10 requests within 15 minutes to prevent brute-force attacks on the login functionality.
- Execution of console commands: 5 requests per second, as each command can potentially result in 2 to 3 database queries.
- Submissions of tasks: 3 requests per second to limit spam clicks.
- All other API routes: 100 requests within a minute.

Route modules are mounted under `/api/` and each covers one functional area of the platform:

Table 4.2: Server route modules and their responsibilities.

Route module	Responsibility
`auth`	Registration, login, logout, session check
`tasks`	Task submission, answer validation, hint delivery, completion history
`console`	Server-side execution of forensic shell commands against the user's VFS
`scenarios`	Listing available scenarios and their metadata
`devices`	Device interaction (connecting and mounting evidence devices)
`leaderboard`	Aggregated score ranking across all users
`tracking`	Ingest client-side interaction events into the event log
`admin`	Aggregated statistics, event log browser, user directory, CSV export

Such separation ensures that each module has only one concern to deal with and can be easily updated or tested independently. In particular, the `tasks` module has the highest security risk since it's the only one where the provided answers are compared with the correct ones, which are taken from `scenarios.json` and reside only on the server.

4.3 Backend: Database, Virtual File System and Scenario Engine

4.3.1 Database Schema and Data Management

In order to ensure simplicity, portability, and independence from specific infrastructure, especially important in academic settings where database administrators or sophisticated containerization may not be readily available, the CyberForensics Arena uses **SQLite** as its main persistent data store.

Although client-server relational databases like PostgreSQL are more appropriate for heavy concurrent write operations, our expected load of dozens of users actively playing the game during a classroom period is well within the capabilities of SQLite. Additionally, since the entire database resides in a single file with a `.db` extension, it is trivially simple to back up, reset for a new class of students, or distribute alongside our application. To prevent blocking the single-threaded Node.js event loop during disk operations, we leverage the `sqlite` package to wrap the driver with Promises, allowing asynchronous database queries.

The database schema has been constructed to monitor user progress, control access, and monitor detailed statistics to inform learning analytics. The schema also initializes automatically on server start if it has not been created before, making it trivially simple to deploy our platform. The main entities of our relational schema are as follows:

- **Users and Authentication (`users`)**: This table stores the login credentials for each and every participant. Passwords are not stored as plaintext. Instead, they're hashed using `bcrypt` with a work factor of 10. This table also stores the role for each user (whether they're an `admin` or a simple `user`) and whether or not they have completed the mandatory tutorial.

- **Scenarios and Tasks (`scenarios`, `tasks`)**: Although the actual scenarios and validation criteria for each scenario are stored in a JSON file (detailed in Section 4.3.3, the metadata and scoring for each scenario are stored in the database. This allows relational queries to easily compute the maximum possible score and completion percentage without needing to parse the JSON file.

- **Progression and Scoring (`task_completions`, `task_attempts`)**: This is the core scoring functionality. `task_completions` logs a successful completion of a task, the score awarded for the task, and the time taken. `task_attempts` logs unsuccessful attempts. This allows for `partial scoring` because each and every unsuccessful attempt is recorded so the backend can subtract points from the max score when the task is finally completed.

- **Gamification (`badges`, `user_badges`)**: This monitors the player's accomplishments as they are earned in the course of the investigation and any bonus points associated with specific badges.
- **State Persistence (`user_vfs_state`, `user_devices`)**:The backend maintains the state of the virtual environment the player has constructed. This includes the current working directory, any files created or modified in the Virtual File System, and any mounted devices containing evidence. This state is isolated on a per-user and per-scenario basis.
- **Evaluation Telemetry (`event_log`)**: As discussed in Chapter 3, learning analytics is a primary objective. This table aggregates all the events from the client, which may include typed terminal commands, hints requested, and other interactions. For the sake of privacy in research studies, events may be recorded using a `participant_id` cookie instead of the authenticated `user_id`.

By maintaining all state and progress in the backend database, the client application is stateless with regard to the game rules, which eliminates the possibility of cheating.

Listing 4.3 shows the definitions of the four tables that are most directly tied to the platform's core mechanics: progression tracking, partial scoring, and evaluation telemetry.

```
1  -- Successful task completions (one row per user per task)
2  CREATE TABLE IF NOT EXISTS task_completions (
3    id           INTEGER PRIMARY KEY AUTOINCREMENT,
4    user_id      INTEGER NOT NULL REFERENCES users(id),
5    task_id      INTEGER NOT NULL REFERENCES tasks(id),
6    score_awarded INTEGER NOT NULL,
7    time_ms      INTEGER,                          -- time-on-task
       in ms
8    completed_at TEXT NOT NULL DEFAULT (datetime('now')),
9    UNIQUE(user_id, task_id)                       -- no duplicate
       completions
10 );
11 -- Wrong submissions (one row per incorrect attempt)
12 -- scoringService reads the count to apply partial-score penalties
13 CREATE TABLE IF NOT EXISTS task_attempts (
14   id           INTEGER PRIMARY KEY AUTOINCREMENT,
15   user_id      INTEGER NOT NULL REFERENCES users(id),
16   task_id      INTEGER NOT NULL REFERENCES tasks(id),
17   attempted_at TEXT    NOT NULL DEFAULT (datetime('now'))
18 );
19 CREATE INDEX IF NOT EXISTS idx_task_attempts_user_task
20   ON task_attempts(user_id, task_id);
```

```
21 -- Per-user, per-scenario VFS snapshot and working directory
22 CREATE TABLE IF NOT EXISTS user_vfs_state (
23   user_id       INTEGER NOT NULL REFERENCES users(id),
24   scenario_code TEXT NOT NULL,
25   cwd           TEXT NOT NULL DEFAULT '/home/user',
26   vfs_data      TEXT NOT NULL DEFAULT '{}',  -- JSON tree
27   updated_at    TEXT NOT NULL DEFAULT (datetime('now')),
28   UNIQUE(user_id, scenario_code)
29 );
30 -- Event log for anonymous evaluation tracking
31 CREATE TABLE IF NOT EXISTS event_log (
32   id             INTEGER PRIMARY KEY AUTOINCREMENT,
33   participant_id TEXT    NOT NULL,   -- CFA-XXXXXX, never linked
       to email
34   user_id        INTEGER,            -- nullable: pre-login events
35   event_type     TEXT    NOT NULL,
36   scenario_code  TEXT,
37   task_id        TEXT,
38   event_data     TEXT DEFAULT '{}',  -- JSON blob
39   created_at     TEXT NOT NULL DEFAULT (datetime('now'))
40 );
41 CREATE INDEX IF NOT EXISTS idx_event_log_participant ON event_log(
       participant_id);
42 CREATE INDEX IF NOT EXISTS idx_event_log_type        ON event_log(
       event_type);
```

Listing 4.3: Key table definitions from `db/schema.js`: scoring, state persistence and event telemetry.

Three design decisions are represented directly in the schema. The first is represented by the fact that `task_completions` has a `UNIQUE (user_id, task_id)` constraint, ensuring at a database level that a student cannot earn points twice for the same task, with a duplicate-check query in `tasks.js` serving as a secondary check. The second is represented by `task_attempts`, which only logs failed attempts, with `task_completions` and `task_attempts` providing all necessary information for calculating partial-score penalties to `scoringService` without redundant data. The third is represented by the fact that `event_log.user_id` is optional, meaning it is null until a session is established, but is then populated with information if both a tracking id and a user id are recorded in the study questionnaire.

4.3.2 Virtual File System and Command Execution

Another constraint facing the students in the CyberForensics Arena is that any shell commands they write, e.g., `ls`, `cat`, `grep`, `dd`, etc., can never execute on the real file system. This would create security holes, potentially leaking confidential files

or even allowing arbitrary code execution on the server. To avoid this, a `Virtual File System (VFS)` is used. This is a tree-like data structure stored entirely in memory, implemented as a tree-like hierarchy of JavaScript objects that mimic a Linux file hierarchy. All command execution occurs within this virtual tree and is entirely isolated from the underlying operating system.

Structure and Isolation

The VFS is a recursive JavaScript object where each node can either be a `dir` (holding a `children` map) or a `file` holding a `content` string. The VFS starts with the same empty tree-like structure: `/home/user`, `/evidence`, `/mnt`, `/forensic`, etc. The VFS is stored in the `user_vfs_state` table within the SQLite database for each user and each scenario they are running, with a unique `user_id` and `scenario_code` combination. This means each student gets their own, completely independent VFS for each scenario. There is no shared state between scenarios, and one student's state does not interfere with another's.

The path resolution is managed by two pure functions in `vfs.js`: `resolvePath`, which resolves a potentially relative path against the current working directory, and `normalizePath`, which collapses `..` and `.` segments. These two functions combined prevent *directory traversal attacks*: since `normalizePath` collapses the segment stack and stops at the VFS root, a path such as `../../../../etc/passwd` can only ever resolve to `/`, and `getNode` will simply return the virtual root node, never touching the host OS.

```
1  // Normalize path (collapse . and ..)
2  export function normalizePath(path) {
3    const parts = path.split('/').filter(Boolean);
4    const stack = [];
5    for (const seg of parts) {
6      if (seg === '..') stack.pop();   // can never go above root
7      else if (seg !== '.') stack.push(seg);
8    }
9    return '/' + stack.join('/');
10 }
11 // Get node at an absolute path in the VFS tree
12 export function getNode(vfs, path) {
13   const p = normalizePath(path);
14   if (p === '/') return vfs;
15   const parts = p.split('/').filter(Boolean);
16   let node = vfs;
17   for (const part of parts) {
18     if (!node.children || !node.children[part]) return null;
19     node = node.children[part];
20   }
21   return node;
22 }
```

Listing 4.4: Path normalisation that prevents directory traversal (`vfs.js`).

Server-Side Command Emulation

The `console` route module (`POST /api/console/execute`) is the sole entry point for command execution. The client will send a JSON payload with the raw command entered by the student and the current `scenarioCode`. The server will then parse the command and execute it against the VFS, returning the text results. The client only ever sees text and never directly accesses the VFS data structure.

At the heart of this is the core dispatcher, which is the `executeCommand` function in `console.js`. It maps the most common UNIX utilities to a `switch` statement, and each one operates only on the virtual filesystem.

- **`ls`**: reads `node.children` of the resolved target path and returns the list of entry names, filtering hidden files unless `-a` is passed.
- **`cd`**: It resolves the target path, checks if the resolved node is a directory, and returns the new `cwd` to be saved.
- **`cat`**: It resolves the path, checks if it is a file, and returns the file's content.
- **`grep`**: It reads the content of the resolved node, splits it into individual lines, and returns only those that match the search pattern.
- **`cp, mkdir, touch, rm`**:These commands modify the in-memory VFS object and set a `vfsModified` flag; if set, the updated structure is persisted back to the `user_vfs_state` table.

After each command, if the current working directory has changed and if the VFS tree has been updated, these changes are saved to the database asynchronously. This ensures that the student session remains intact after page reloads or reconnections without any loss of state.

Mounting Evidence: Simulating Forensic Disk Operations

Another significant aspect of the forensic process is *attaching* a seized device, followed by *mounting* its contents, which is familiar to all digital forensic analyst. CyberForensics Arena simulates all of this without touching a real disk. In `scenarios.json`, each scenario contains interactive 3D objects. When a student clicks on one of these objects, for example, a hard disk, within the Babylon.js scene, a `POST request` is sent to `/api/devices/attach`. This adds the device, along with its simulated contents: a JSON mapping of filenames to content strings, to the

user_devices table. Note, however, that the device's contents are not yet accessible via the shell.

Later, when the student executes the emulated mount command, for example, `mount -o ro /forensic/evidence.img /mnt/evidence`, the console route catches this as a custom command, executing internally a `POST request` to `/api/devices/mount`. This discovers the device, mounts the device's content to VFS using `mountDeviceContentToVFS`, writing all of the filename/content pairs to the chosen VFS subtree, for example, `/mnt/evidence`, and marks it as mounted by setting `user_devices.mounted = 1`. From this point on, `cat /mnt/evidence/suspicious.log` behaves exactly as it would on a real Linux system. Figure 4.2 shows the workflow of attaching and mounting a device.

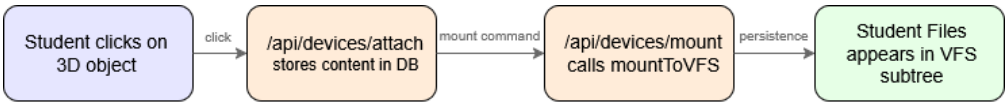

Figure 4.2: Lifecycle of evidence attachment and mounting in the VFS.

This design provides a realistic simulation of the forensic procedures: attach the device, compute the hash, create the image, and mount it read-only, yet it remains fully self-contained, with no disk images actually moved to or stored on the server.

4.3.3 Scenario Engine and Task Validation

Data-Driven Design

The scenario engine is based on a *data-driven* paradigm, meaning there is no game logic coded into the application. Everything about a scenario, including the name, story, objects you can interact with, actions you can take, and all steps of a scenario, is driven by data loaded from a JSON file during startup: `server/data/scenarios.json`. This has a significant implication for education, as a professor or administrator can modify or add scenarios, new tasks, or adjust point values by changing a text file, without changing code or requiring a restart.

Task Structure

The structure of each task within scenarios.json is a small JSON object with a small, consistent set of attributes. The three native check types are represented by the representative examples in Listing 4.5.

```
1 // --- Type 1: Interaction task (click a 3D object) ---
2 {
3   "id": "fs_task_1",
4   "title": "Connect the seized hard disk",
```

```
5     "details": "Click on the hard disk to connect it to the
       workstation.",
6     "points": 10,
7     "checkType": "interaction",
8     "interactionTarget": "HD1-01_HD-1_0"
9   }
10  // --- Type 2: Flag task (free-text answer) ---
11  {
12    "id": "fs_task_2",
13    "title": "Identify the device partition",
14    "details": "Use 'lsblk' to list block devices. Enter the
       partition name.",
15    "points": 15,
16    "checkType": "flag",
17    "solutionValue": "sdb1",
18    "hint": "The partition will have a number suffix.",
19    "hintCost": 3
20  }
21  // --- Type 3: Command task (exact shell command) ---
22  {
23    "id": "fs_task_4",
24    "title": "Create forensic image",
25    "details": "Use dd to create a bit-by-bit forensic image.",
26    "points": 25,
27    "checkCommand": "dd",
28    "checkArgs": ["if=/dev/sdb", "of=/forensic/evidence.img"]
29  }
```

Listing 4.5: Examples of all three native task check types from `scenarios.json`.

The validation rules are specified by three separate, non-overlapping fields. If you specify `checkType: "interaction"` with an `interactionTarget`, you are validating real physical interaction with the 3D scene. If you specify `checkType: "flag"` with a `solutionValue`, you are validating a free-text answer. And if you specify `checkCommand` together with `checkArgs`, you are validating that the student typed an exact shell command with correct arguments. The task can have an optional `hint` and a `hintCost` (in points), which enables the progressive hint system.

The Validation Flow

The full validation lifecycle for a task submission is illustrated in Figure 4.3 and described step by step below.

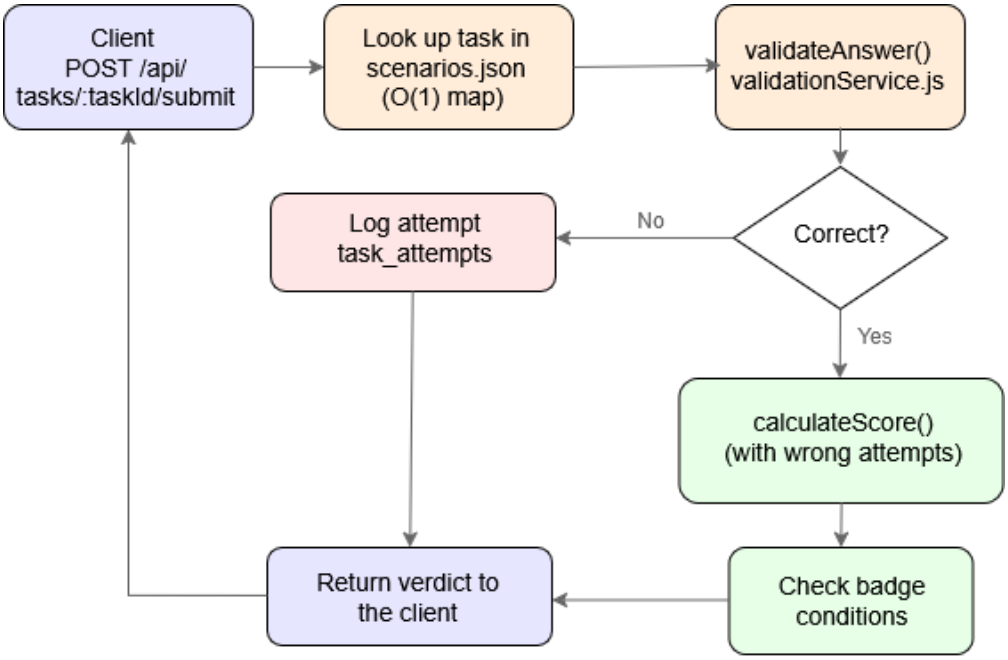

Figure 4.3: Task validation and scoring flow handled server-side by `tasks.js`.

1. **Request.** A `POST request` is made by the client to `/api/tasks/:taskId/submit` with the student's answer and the time elapsed in milliseconds.

2. **Task lookup:** The code, as seen in the `tasks.js`, retrieves a task by traversing the in-memory data structure `scenariosData`, which was created by parsing the `scenarios.json` data structure once when the server starts and then sits cached at the module level. The hint route uses a separate data structure, the `taskLookupMap`, which is a `Map<taskId, task>` created at startup, to perform O(1) lookup, and the submission route uses the cached object directly.

3. **Duplicate check.** A query on task_completions is made to avoid scoring an already completed task.

4. **Validation:** The answer is passed on to `validateAnswer(task, answer)` in `validationService.js`, which is a stateless pure function. Depending on the number of fields in the task object, the appropriate validation is performed (see Listing 4.6

5. **Scoring:** In scoring the answer, `calculateScore(task, isCorrect, wrongAttempts)` is called. This function decides how many points are awarded for the answer. `wrongAttempts` are retrieved from `task_attempts` and are passed in so that the scoring logic can decide on partial scoring if desired. For now, full points are awarded.

6. **Badge evaluation:** If the answer is correct and the student completes all tasks in the scenario, badge evaluation is performed in `tasks.js`. A badge is awarded for the scenario completed, a *Speed Runner* badge if total time is under the threshold, and a *Hint-Free Expert badge* if no hints are used. Any badges earned along with the bonus points are included in the response object.

7. **Response:** A JSON object is returned back to the client with the final values of `correct`, `scoreAwarded`, `newTotalScore`, and `badgesUnlocked`.

```
export function validateAnswer(task, answer) {
  // --- Interaction tasks ---
  if (task.checkType === 'interaction' && task.interactionTarget)
   {
    if (answer.startsWith('interaction:')) {
      const target = answer.substring('interaction:'.length);
      return { correct: target === task.interactionTarget };
    }
    return { correct: false };
  }
  // --- Flag / CTF tasks ---
  if (task.checkType === 'flag' && task.solutionValue) {
    const norm = s => s.trim().toLowerCase();
    return { correct: norm(answer) === norm(task.solutionValue) };
  }
  // --- Console command tasks ---
  if (task.checkCommand) {
    const parsedArgs = parseCommandArgs(answer);
    if (parsedArgs[0] !== task.checkCommand) return { correct:
   false };
    if (task.checkArgs?.length > 0) {
      const strip = s => s.replace(/\/$/, '');
      const actual   = parsedArgs.slice(1).map(strip);
      const expected = task.checkArgs.map(strip);
      return { correct: JSON.stringify(actual) === JSON.stringify(
   expected) };
    }
    return { correct: true };
  }
  return { correct: false }; // unknown type
}
```

Listing 4.6: The `validateAnswer` pure function (`validationService.js`), simplified for presentation.

It is important to note that the solution values (`solutionValue`, `checkCommand`, `checkArgs`) are only read on the server from `scenarios.json`. They are not sent to the client; therefore, the student cannot access them using the browser developer tools.

4.3.4 Authentication and Session Management

The authentication and session process is configured using Passport.js with its *Local Strategy*, as configured in `config/passport.js`. The strategy maps the email field of the request body to the `usernameField`, then looks up the `users` table for a row with a matching email address, finally using `bcrypt.compare()` as a password verification function. As can be seen in Listing 4.7, both the "user not found" and "wrong password" conditions return the same error message, a deliberate security measure against user-enumeration attacks.

```
passport.use(new LocalStrategy(
  { usernameField: 'email', passwordField: 'password', session: true },
  async (email, password, done) => {
    const user = await db.get(
      'SELECT id, email, password_hash, display_name, role, tutorial_completed
       FROM users WHERE email = ?', email
    );
    // Same message for both "not found" and "wrong password"
    // to prevent user enumeration
    if (!user) return done(null, false, { message: 'Invalid email or password' });
    const valid = await bcrypt.compare(password, user.password_hash);
    if (!valid) return done(null, false, { message: 'Invalid email or password' });
    // Return user WITHOUT password_hash
    return done(null, { id: user.id, email: user.email,
                        displayName: user.display_name, role: user.role,
                        tutorialCompleted: user.tutorial_completed === 1 });
  }
));
// Only the user ID is stored in the session cookie
passport.serializeUser((user, done) => done(null, user.id));
// Full user object is rehydrated from the database on every request
passport.deserializeUser(async (id, done) => {
  const user = await db.get('SELECT ... FROM users WHERE id = ?', id);
  done(null, user ?? false);
});
```

Listing 4.7: Passport.js Local Strategy with user-enumeration-safe error handling (`config/passport.js`).

Registration Flow

The `POST /api/auth/register` route first checks the request before it interacts with the database in any way. It checks for the existence of an email, password, and display name, and it also checks the email for validity with a regex and makes sure the password is at least six characters long. If all checks pass, it hashes the password with `bcrypt.hash(password, 10)` and stores it instead of the plain password in the database. By default, all users are assigned the role of user, and admin users are created in `db/schema.js` during initialization and cannot be created through the registration route.

Session Serialisation and the `/me` Route

The `deserializeUser` callback for all requests ensures that the entire user object is retrieved from the database instead of just the session snapshot, so any changes made by admins will be reflected instantly. The `GET /api/auth/me` route makes another query to the database in addition to the session object, so the client will always be given the current values for `tutorialCompleted` and `role` whenever the page loads.

4.3.5 Scenario API and Response Filtering

The main route that the client will use to retrieve all scenarios and their tasks is `/api/scenarios`, and this will be executed each time someone logs in or re-authenticates. If we were simply reading `scenarios.json` from disk each time, we would block the event loop in Node.js, which is single-threaded. The second problem is that we would send solution values and hint text in the JSON data, which we shouldn't send to the client. The problems are fixed by the `processScenarios()` function and a small cache that is kept at the module level.

Solution Stripping

The `processScenarios()` function goes through each and every one of the scenarios and tasks, creating a new version that is safe to send to clients and leaving out sensitive fields. The version that is safe to send is what is actually sent to clients, as is shown in Listing 4.8.

```
1 function processScenarios(scenariosData) {
2   return Object.keys(scenariosData)
3     .filter(key => !key.startsWith('_'))   // skip internal
    metadata keys
4     .reduce((acc, key) => {
5       const scenario = scenariosData[key];
6       acc[key] = {
```

```
        id: key,
        title: scenario.title,
        description: scenario.description,
        introduction: scenario.introduction,
        badge: scenario.badge,
        interactableObjects: scenario.interactableObjects || [],
        customCommands: scenario.customCommands || [],
        tasks: scenario.tasks.map(task => ({
          id: task.id,
          title: task.title,
          details: task.details,
          points: task.points,
          checkType: task.checkType || null,
          interactionTarget: task.interactionTarget || null,
          checkCommand: task.checkCommand || null,
          checkArgs: task.checkArgs || null,
          hintCost: task.hintCost || 0,
          hasHint: !!(task.hint && task.hint.trim()),
          // solutionValue, hint text: intentionally excluded
        }))
      };
      return acc;
    }, {});
}
```

Listing 4.8: Solution fields stripped before the scenario catalogue is sent to the client (`routes/scenarios.js`).

The flag `hasHint` informs the client if it should display the Hint button, but it does not give the actual hint text. The hint text is only accessible with the `GET /api/tasks/:taskId/hint` call, which requires authentication and writes a `hint_request` event to the event log.

In-Memory Cache and Hot Reload

The cache is invalidated whenever an admin saves their updated scenario by sending a `POST` request to `/api/scenarios`, which is what the visual editor does. This causes the route to clear the `publicScenariosCache` by setting it to null and immediately call `loadScenariosData()`, which is in `tasks.js` and rebuilds the `taskLookupMap` used for validation. There is also an exported function for invalidating the cache in case someone modifies the `scenarios.json` file directly on disk, which requires a manual trigger. This hot reloading feature allows the professor to add a new task or correct a typo in the scenario and have it take effect in the current session without needing to restart the server.

4.3.6 Event Logging and Participant Tracking

The platform collects extensive interaction data that is used by learning analytics. The system has two main components: the `extractParticipantId` middleware and the `services/eventLog.js` service.

Anonymous Participant ID Middleware

Each incoming request, with or without a logged-in user, is passed through `extractParticipantId` (`middleware/participantId.js`), which reads the `X-Participant-Id` HTTP header and validates it against the regex `/^CFA-[A-Z0-9]{6}$/` (e.g. `CFA-AB12CD`). If the value is valid, it is set as `req.participantId` and is thus available to all route handlers that follow. The logic of this code is that tracking and authentication identities are separate; `/api/tracking` routes do not require a session cookie and thus can log events from the very first page load and throughout the tutorial, even before submitting the first task. The participant ID is client-side created and stored and is never linked with the user's email or display name and thus maintains anonymity.

The `logEvent()` Service

All updates to the `event_log` table pass through one central function, `logEvent()`, in `services/eventLog.js` (as shown in Listing 4.9). If something goes wrong, the error is caught and printed to the console, but it isn't re-thrown. So a temporary database hiccup won't break the student's session.

```
export async function logEvent({
  participantId, userId = null,
  eventType, scenarioCode = null,
  taskId = null, eventData = {}
}) {
  if (!participantId) return;   // skip if no tracking ID
  try {
    await db.run(`
      INSERT INTO event_log
        (participant_id, user_id, event_type, scenario_code,
   task_id, event_data)
      VALUES (?, ?, ?, ?, ?, ?)
    `, participantId, userId, eventType,
       scenarioCode, taskId, JSON.stringify(eventData));
  } catch (err) {
    // Log but never throw, tracking must not break gameplay
    console.error('[EventLog] Failed to log event:', err.message);
  }
}
```

Listing 4.9: The `logEvent()` function: single write point for all telemetry (`services/eventLog.js`).

Nine event types are defined in the `EventTypes` constant and used consistently across the application:

Event type	Logged when
`scenario_start`	Student activates a scenario
`scenario_end`	Student completes or leaves a scenario
`task_submit`	A task answer is submitted (correct or wrong)
`flag_submit`	A flag answer is entered via the HUD
`hint_request`	Student requests a hint
`command_execute`	A local (client-side) command is run
`mini_game_start`	A mini-game is launched
`mini_game_complete`	A mini-game is completed successfully
`mini_game_fail`	A mini-game is failed or aborted

Table 4.3: Event types recorded in the `event_log` table(`services/eventLog.js`).

In the research analysis pipeline, `getParticipantMetrics()` processes raw logs received for each participant and generates a nice, clean report. The report contains information about the number of commands executed, how many of these ended with an error, how often tasks were submitted, how many first-attempt successes were recorded, the total number of hints used, and start and end times per scenario, enabling time-on-task calculations.

Admin Dashboard API

The `routes/admin.js` module, available only to users with the `admin` role, exposes all collected data. Four routes are available, designed specifically for researchers:

- `GET /api/admin/logs`: paginated browser of event logs, with filtering by event types, scenarios, participant IDs, user IDs, and dates.
- `GET /api/admin/stats`: aggregated platform stats, including total users, active users (those active in last 24 hours), total commands executed, total hints used, and a chart of activity over the last 7 days.
- `GET /api/admin/users`: list of all users, with task completion count and score, computed by joining task_completions.

- `GET /api/admin/users/:userId/stats` : breakdown of data for a specific student, with event types, commands executed, task submission history, and timestamps used for time-on-task calculations.

4.4 Frontend: User Interface, 3D Environment and Interaction Model

The client-side application is a standard JavaScript Single Page Application (SPA) that is packaged using Vite. It does not utilize any abstractions provided by client-side JavaScript frameworks such as React or Vue; instead, all components are simple ES modules that communicate only through the shared Event Bus described in Section 4.4.2. The `main.js` file is responsible for controlling the entire startup sequence.

4.4.1 Application Entry Point and Lifecycle

On `DOMContentLoaded`, `main.js` calls `initializeApp()`, a six-step asynchronous sequence that bootstraps every subsystem in strict order:

1. **Navigation bar.** `initNavigation()` renders the top navbar immediately, so the user sees a consistent chrome before any async work begins.

2. **Canvas discovery.** The WebGL `<canvas>` element (`renderCanvas`) is retrieved from the DOM and made visible.

3. **Session check.** `initSession()` issues a lightweight `GET /api/auth/me` request to determine whether a valid session cookie is already present.

4. **Authentication gate.** If the session is not authenticated, the login page is shown and `initializeApp` suspends on a `Promise` that resolves only when the `AUTH_SUCCESS` event fires on the Event Bus, no polling, no callbacks passed between modules.

5. **Core initialisation.** Once authenticated, `initCore()` is called (see Listing 4.10).

6. **Error handling.** Any uncaught exception hides the loading overlay and renders a fatal error panel via `showFatalError()`.

```
1 async function initCore() {
2   // 1. Create Babylon.js engine + scene
3   appState.engine = new BABYLON.Engine(canvas, true, CONFIG.
     ENGINE_OPTIONS);
```

```
4     appState.scene  = await createScene(appState.engine, canvas);
5     // 2. Wire subsystems
6     setupTaskManager(appState.scene);
7     setupInteractions(appState.scene, appState.scene.activeCamera);
8     initConsole();
9     // 3. Sync persisted state from server
10    await PointsBadge.init();
11    await syncTotalScore();
12    // 4. Load scenario catalogue (cached)
13    const scenarios = await loadScenarios();
14    if (scenarios) await loadCompletedScenarios();
15    // 5. Tutorial gate
16    const user = getCurrentUser();
17    if (user?.tutorialCompleted) {
18      await onTutorialComplete(true);   // skip tutorial, go
        straight to scenario
19    } else {
20      initializeTutorial();             // first-time user
21    }
22    // 6. Start render loop
23    appState.engine.runRenderLoop(() => {
24      if (appState.scene && !appState.engine.isDisposed) {
25        appState.scene.render();
26      }
27    });
28    appState.isInitialized = true;
29  }
```

Listing 4.10: Core initialisation sequence in `main.js`: systems are started in dependency order.

Logout and Re-initialisation

Another important aspect to be considered in any session-based application is the ability to carry out an exhaustive teardown when the user decides to log out. When the `USER_LOGGED_OUT` event is received by the application, which is triggered on the Event Bus, `main.js` carries out an exhaustive teardown in a particular order: it stops the render loop, disposes of the Babylon.js scene and engine, hides all the UI panels, and resets the task manager and points/badges widget, and finally renders the login page.

When the user logs in again, the `AUTH_SUCCESS` event is triggered, and `initCore()` is invoked once again. In order to prevent the creation of two instances of the Babylon.js engine, which is not permissible, the application utilizes two flags: `isInitialized` and `isInitializing`.

Pointer Lock and Tab Visibility

There are two additional event handlers that complement the above-mentioned event handlers. One is the function `installPointerLockSafety`, which is responsible for requesting the pointer lock every time the user switches back to the application's 3D scene. This is an imperative feature, as the browser does not allow the application to maintain the pointer lock when any modal dialog is invoked, which is often the case when the user is introduced to the scenario or when the user is engaged in mini-games. The application sets the `disablePointerLock` flag when the modal dialog is invoked by the `ScenarioIntroManager`.

Another feature is the `visibilitychange` event handler, which is responsible for addressing the problem where the browser does not allow keyboard input when the user switches tabs. In such a scenario, the camera controls are detached and attached again, and `ngine.resize()` is invoked if the dimensions of the window change during this time.

4.4.2 Event Bus: The Publish-Subscribe Backbone

Communication between modules within the client application follows a strict publish-subscribe model implemented with `eventBus.js`. The EventBus class represents a lightweight, dependency-free construct with a `Map<eventName, handlers[]>` data model. It offers three methods: `on(event, handler)` for subscribing and returning an unsubscribe function, `once(event, handler)` for one-time subscriptions, and `emit(event, data)` for publication. The design uses a shared singleton instance exported as eventBus and imported by each module requiring inter-module communication.

This design achieves the layer isolation described in Section : no layer holds a reference to the internal functions of another layer. For example, `scene.js` does not import `taskManager.js`. Instead, `scene.js` subscribes to `SCENARIO_CHANGED` and redraws the scene's highlights when this event is published by the task layer. Table 4.4 lists the entire range of named events implemented within the client application and indicates the direction for each event.

Event	Direction	Meaning
MESH_CLICKED	Rendering → Logic	User clicked a 3D mesh
MESH_HOVERED	Rendering → Logic	Crosshair is over a mesh
CONSOLE_TOGGLE	Rendering → Logic	E/C key pressed to open/close terminal
CONSOLE_COMMAND_EXECUTED	Logic → UI	A shell command was sent to the server
SCENARIO_CHANGED	Logic → Rendering	Active scenario switched
TASK_COMPLETED	Logic → UI	A task was validated correct
TASK_ADVANCED	Logic → UI	Task index moved forward
SCENARIO_COMPLETED	Logic → UI	All tasks in scenario done
TERMINAL_COMMAND_VALIDATED	Logic → UI	Server confirmed command task correct
PROGRESS_UPDATED	Logic → UI	Score/progress changed
UI_CONSOLE_VISIBLE_CHANGED	UI → Rendering	Console panel shown/hidden
AUTH_SUCCESS	Auth → All	Login succeeded
USER_LOGGED_OUT	Auth → All	Logout completed
TUTORIAL_MOVED ... TUTORIAL_COMPLETED	Rendering/Logic → Logic	Predicate events for tutorial steps

Table 4.4: Named events defined in `eventBus.js` and the layer boundary they cross.

The bus maintains a rolling history of the most recently occurring 100 events that have been emitted and includes timestamps for these events; these are accessible

through `eventBus.getHistory()`. Used in combination with the `enableEventBusDebug` utility function, which monkey-patches `emit` to route events through `console.log`, the entire publish/subscribe mechanism is observable without additional tooling.

4.4.3 3D Scene and Rendering Layer

The 3D Environment

The 3D lab environment, a key component of CyberForensics Arena, was not created from scratch. Instead of combining individual open-source content from a library to create a blank chamber, development began with a semi-complete, themed lab environment, *Flynn's Secret Lab* (TRON Legacy), published under a Creative Commons Attribution License (CC BY 4.0) via Sketchfab[15]. This provided a visually cohesive starting point, shifting development focus from environment construction to gameplay development.

The scene has been modified to match the needs of the Babylon.js game engine and align with pedagogical objectives:

- Digital forensics scene objects, including hard drives, USB drives, network switches, a forensics workstation, and evidence bags, were added to the scene.
- Existing scene objects were modified, rotated, and resized to maintain visual consistency with the original scene.
- All forensics scene objects were configured to be interactive by assigning a named mesh identifier to each object, mapping to `interactionTarget` fields in `scenarios.json`.
- The modified scene has been saved as a single GLB file, "secret_lab.glb," loaded by Babylon.js.

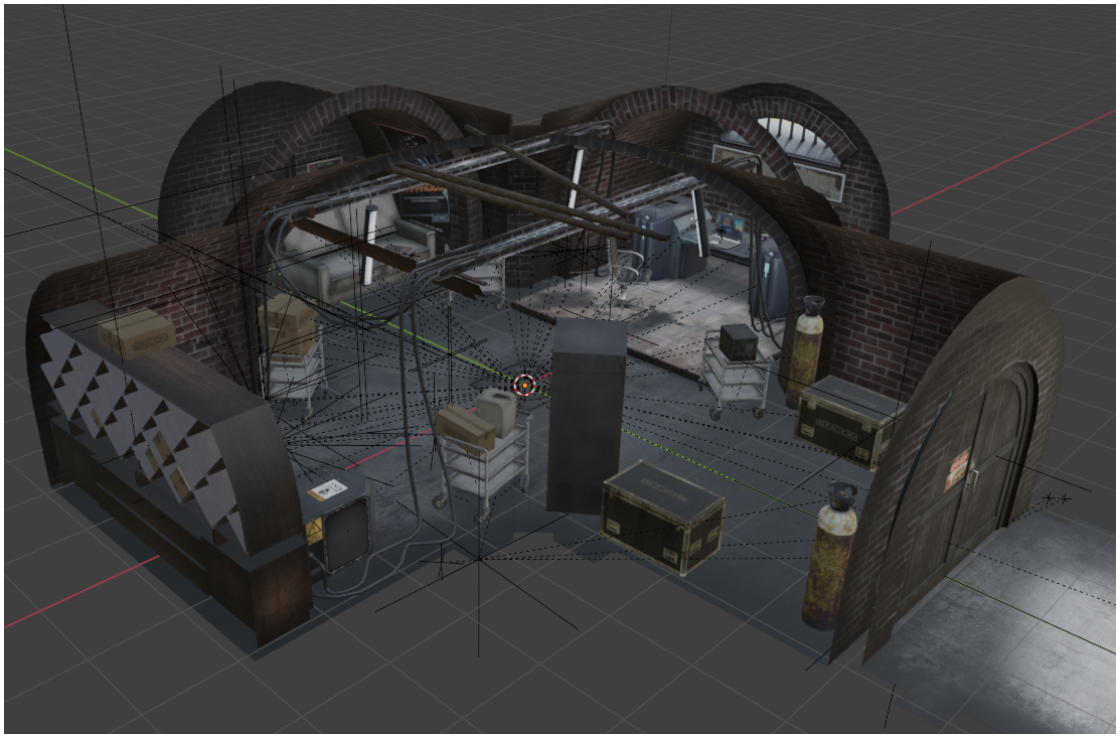

Figure 4.4: The 3D model opened in Blender

Babylon.js Scene Initialisation

The rendering layer is encapsulated exclusively in the `scene.js` file, where imports are made from the `eventBus.js` file and `Babylon.js`, making no reference to the logic/UI layer. When the `createScene(engine, canvas)` function is called from the `main.js` file, the following pipeline is executed:

1. **Scene Setup.** A `BABYLON.Scene` is created, where gravity is set to `SCENE_CONFIG.GRAVITY`, and collisions are enabled.

2. **Camera.** A `UniversalCamera` is created, positioned at eye level (y = 1.6), where WASD movement is enabled, ellipsoid collisions are set to `Vector3(0.3, 0.9, 0.3)`, and the camera is locked upon canvas interaction. The camera's sensitivity is set to 6000 to produce a natural feeling during mouse look interaction.

3. **Lighting.** A `HemisphericLight` is created, where the intensity is set to 0.9 to match the laboratory lighting.

4. **Model Loading.** `SceneLoader.ImportMeshAsync` is used to load the `secret_lab.glb` file asynchronously, after which `scene.whenReadyAsync` is awaited

before proceeding to the next step to ensure that all GPU resources are loaded.

5. **Mesh Setup.** The `setupMeshes()` function is called, where each loaded mesh is enabled to check for collisions (depending on the `gltf.extras.hasCollision` property, which is exported from the Blender file), each geometry is marked as pickable, and the `allInteractableMeshes` array is filled, where the interaction layer is dependent on this data structure.

6. **Highlight layer.** A `BABYLON.HighlightLayer` is created, where soft glow is enabled on the outer and inner areas, to draw the pulsating highlights on the objects that are interactable.

7. **Spawn Point.** The `setupCameraSpawn()` function is called, where a `TransformNode` named `Spawn` is exported from the Blender file, and the camera is placed at this position. If not found, the camera is positioned at the first surface that is solid upon ray-casting from directly above the center of the scene.

Pulsing Highlights on Interactable Objects

In order to guide the student towards the objects that are appropriate for investigation, a subtle pulsating blue-green glow surrounds each mesh relevant to the current scenario. This is achieved in the function `setupAnimationLoop()`, where a callback function `registerBeforeRender` is used to dynamically change the color of the highlight on a shared counter:

```
1  scene.registerBeforeRender(() => {
2    pulseTime += HIGHLIGHT_COLOR.PULSE_SPEED;   // 0.015 rad/frame
       ~= 0.9 Hz
3    const i = HIGHLIGHT_COLOR.PULSE_BASE        // 0.6
4              + Math.sin(pulseTime)
5              * HIGHLIGHT_COLOR.PULSE_RANGE;    // +/- 0.3
6    permanentHighlightedMeshes.forEach(mesh => {
7      if (mesh === currentHoveredMesh) return;  // hover handled
       separately
8      highlightLayer.removeMesh(mesh);
9      highlightLayer.addMesh(mesh, new BABYLON.Color3(0.4*i, 0.6*i,
       i));
10   });
11 });
```

Listing 4.11: Per-frame sine-wave highlight pulsing in `scene.js`.

Scenario-Aware Highlights

The rendering layer operates independently of task logic. When a transition between scenarios occurs due to the task manager, a `SCENARIO_CHANGED` event is dispatched on the Event Bus. This event is caught by the `scene.js` module, which calls the `updateScenarioHighlights()` function, clearing the list of highlights and repopulating `permanentHighlightedMeshes` based on matching names between `scenario.interactableObjects` and the meshes, based on their name, id, parent name, and a custom tag, utilizing substring matching in a case-insensitive fashion due to the Blender export conventions.

```
const isMatched = targetNames.some(targetName => {
  const t = targetName.toLowerCase();
  // exact match on name, id, parent name, parent id, or custom tag
  if (meshNameLower === t || meshIdLower === t) return true;
  if (parentNameLower === t || parentIdLower === t) return true;
  // prefix match (Blender appends _0, _1 to split meshes)
  if (meshNameLower.startsWith(`${t}_`)) return true;
  // substring fallback
  if (meshNameLower.includes(t) || meshIdLower.includes(t)) return true;
  return false;
});
```

Listing 4.12: Mesh-matching logic in `updateScenarioHighlights()` (`scene.js`).

This multi-strategy matching is necessary because the Blender GLB exporter adds numerical suffixes (`_0`, `_1`, etc.) to the names of a single object split into multiple mesh primitives, so that a single logical property (e.g., `HD1-01`) can be mapped to several runtime mesh nodes.

4.4.4 Interaction Model: 3D World and Game Logic

The `interaction.js` module acts as the only entry point that allows physical user interactions, like mouse movements, keyboard inputs, and mesh selections, to be mapped into the game environment. This module's only exported function, `setupInteractions(scene, camera)`, initializes all the interactions at startup, communicating only through the Event Bus, without any direct calls to the task manager, the console, or any UI component.

Hover Highlight

A raycast from the center of the screen into the scene occurs every frame, identifying a mesh that intersects the raycast, provided that the mesh is present in `allInteractableMeshes`, exported from the `scene.js` file. This identified mesh

is added to the `HighlightLayer` with a bright, static green color (`Color3(0, 0.8, 0)`) and becomes the `currentHoveredMesh`. The pulsating animation loop defined in the `scene.js` file ignores the hovered meshes, preventing any interference between the two highlight features. When the crosshair is moved away from the hovered mesh, it is removed from the hover highlight, and the pulsating animation resumes.

Keyboard Input

Three key bindings drive the game flow:

- **E**: triggered when the crosshair is over the forensic PC mesh. Emits `CONSOLE_TOGGLE` on the Event Bus, which the console module picks up to open the forensic terminal. The handler checks `isConsoleOpen()` and `isTutorialGateOpen()` before firing to avoid spurious events during the tutorial.
- **C**: toggles the terminal open or closed from anywhere in the scene, emitting the same `CONSOLE_TOGGLE` event.
- **Escape**: closes the terminal if open; emits `TUTORIAL_ESC_PRESSED` so the tutorial can advance its predicate.
- **Alt + R**: calls `safeRespawn()` to teleport the camera back to the spawn point, useful if the player walks out of bounds.

4.4.5 Mesh Click and Interaction Events

When a left click is performed in the three-dimensional scene, the file `interaction.js` makes use of `scene.pick()` for selection and checks if the selected mesh belongs to `allInteractableMeshes`. If the condition is satisfied, an event named `MESH_CLICKED` is dispatched with the mesh name as payload. The task manager receives and checks if the dispatched mesh name corresponds to the `interactionTarget` of the ongoing task. Once correspondence is established, the string `"interaction:<meshName>"` is sent to the endpoint `/api/tasks/:taskId/submit` for server validation. This series of events is illustrated in Figure 4.5.

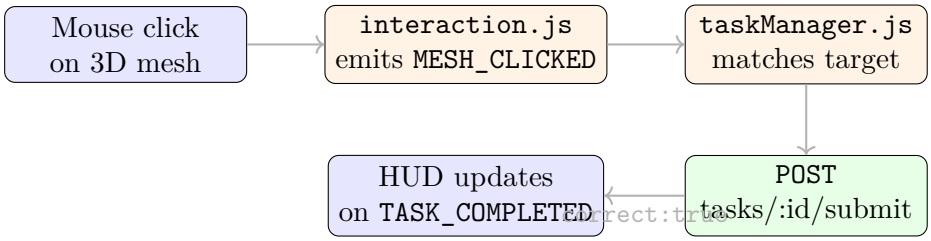

Figure 4.5: Event flow from a mesh click to task completion in the interaction model.

Input Isolation

One important design constraint is that the terminal and 3D canvas must not interact with each other by consuming input. This constraint is implemented by the interaction module. It tests three conditions before any keypress event can be handled:

- `isTypingInXterm()`: This condition returns true when the focus is within the ‘xterm-helper-textarea‘, effectively disabling WASD-based 3D camera movement while a student enters a command into the terminal.
- `isConsoleOpen()`: This condition prevents 3D interaction shortcuts (‘E‘, mouse click) when the terminal panel is visible.
- `isTutorialGateOpen()`: This condition disables all game input when the tutorial overlay is present and waiting for a student action.

4.4.6 Forensic Terminal

The forensic terminal is the primary interface for interaction between the student and the virtual evidence. It acts as a medium through which all shell commands are input, their output is viewed, and operations are performed. The entire terminal experience is provided by `console.js`, a script with 1639 lines of code that encapsulates the `xterm.js` library and provides a wide range of internal systems.

Figure 4.6: The Linux Simulated Console

xterm.js Integration

The terminal widget is based on the `xterm.js` Terminal class, wrapped in a fixed-position `<div>` that overlays the 3D canvas. The `xterm-addon-fit` extension is used to dynamically adjust the terminal's column count and row count based on changes in the viewport due to resizing of the browser window. The color scheme is set to a nearly black background `#0a0a0a`), and the prompt displays ANSI color codes: `forensic` in green, the working directory in blue, followed by the `$` , mimicking a real UNIX terminal.

The visibility of the terminal widget is controlled by adding/removing a CSS class from the overlay panel. The console module listens to the `CONSOLE_TOGGLE` event on the EventBus, calling `toggleConsoleVisibility()` that attaches/detaches the `xterm.js` widget's focus and emits `UI_CONSOLE_VISIBLE_CHANGED`. This notification tells `interaction.js` to disable 3D input when the terminal is visible.

Internal Architecture

`console.js` is structured as a set of single-responsibility objects, each encapsulating one aspect of terminal behaviour:

- **`VFSManager:`** a client-side shadow representation of the server's virtual file system (VFS), which is used exclusively for tab autocomplete path resolution and for conserving files created locally through output redirection (`>`, `»`). In all

cases, command execution is delegated exclusively to the server; the client-side tree has no authority.

- **`CommandRegistry:`** a `Map<name, handler>` containing all available commands. Built-in commands such as `help`, `echo`, and `clear` are registered at initialization. Scenario-specific custom commands are registered by `registerCustomCommands()` upon loading a scenario and deregistered by `unregisterCustomCommands()` upon a scenario transition.
- **`Parser`**: Tokenises the raw input line, handles double-quote grouping, and detects output redirection operators (`>` and `»`), returning a `{cmd, args, redir}` object.
- **`HistoryManager:`** maintains a cache of up to 200 commands within`sessionStorage`, with navigation through the command history by pressing the `up` and `down` arrow keys. A duplicate check prevents consecutive identical commands from being added to the cache.
- **`Autocomplete`**: performs tab autocomplete on the first token by querying `CommandRegistry.keys` synchronously or by querying the server's VFS through `VFSManager` with `ls` for path autocomplete. A directory listing cache prevents a second API call when the type of the completed entry must be immediately verified.
- **`InputHandler`**: the raw `xterm.js` key event handler. Reacts to printable characters (insert at cursor), `Backspace`, `Delete`, `Home`, `End`, arrow-key cursor movement, `Ctrl+C/X/V` for clipboard, `Ctrl+L` for clear, `ESC` to close the console, and `Tab` to trigger autocomplete.
- **`CommandExecutor`**: the dispatch core described below.

Command Execution Flow

When the student presses Enter, `InputHandler` passes the buffered line to `CommandExecutor.execute()`, which follows a two-path dispatch strategy (Listing 4.13):

```
1 async execute(line) {
2   const { cmd, args, redir } = Parser.parse(line);
3   HistoryManager.save(line);
4   const scenarioCode = getCurrentScenario()?.id;
5
6   // PATH 1: hardcoded VFS commands -> backend API
7   const vfsCommands = ['ls','cd','pwd','cat','grep','sha256sum',
8                        'dd','fsstat','foremost','xxd','stat','fls'
     , ...];
9   if (vfsCommands.includes(cmd) && scenarioCode) {
```

```
10      const result = await consoleAPI.execute(scenarioCode, line);
11      if (result.promptPath) TerminalState.cwd = result.promptPath;
12      TerminalUI.writeLine(result.error ?? result.output);
13      if (!result.error) await TaskManager.checkCompletion(cmd, args
        );
14      return;
15    }
16
17    // PATH 2: all other commands -> local CommandRegistry
18    const handler = getCommand(cmd); // built-ins + custom from
        scenarios.json
19    let out = await handler(args);   // returns predefined output or
        launches mini-game
20    if (out) this.handleRedirection(out, redir);
21    await TaskManager.checkCompletion(cmd, args);
22
23    // Log for analytics only (does not affect output)
24    await trackingAPI.logCommand(scenarioCode, cmd, hasError);
25  }
```

Listing 4.13: Two-path command dispatch in `CommandExecutor` (`console.js`).

Path 1: Server-routed commands: The client's hardcoded `vfsCommands` array contains the names of the commands that are sent to the server using the `POST /api/console/execute` API when the scenario is active. These are the standard filesystem commands that are expected to be executed on the server's VFS. The server's `executeCommand()` function inherently supports these standard filesystem commands, returning the output along with the updated promptPath to keep the client's prompt synchronized.

Commands such as `sha256sum`, `foremost`, `xxd`, `fsstat`, `dd`, `blkcat`, `fls`, etc., that are names of the forensic tools, are also part of the client's vfsCommands list, thus sent to the server. However, these are not natively supported by the server's `executeCommand()` function. Instead, these fall under the default case, where the function looks up the active scenario's `customCommands` list under the `scenarios.json` file, returning the output authored by the scenario developer to simulate the actual output that these commands would produce on the VFS contents.

Path 2: Client-side CommandRegistry. The remaining commands, such as the builtins (`help`, `echo`, `clear`), along with the custom commands that are not part of the vfsCommands list, are resolved entirely on the client side using the `CommandRegistry`. Once the execution is complete, the interaction is logged using the trackingAPI's `logCommand()` function, which is for learning analytics purposes.

Custom Commands and Mini-Game Intercept

Scenario authors can add custom commands in the `scenarios.json` file with a `type` field. If the type of the command is `"text"`, it registers the command as a synchronous handler returning either a static string or a string built from the provided arguments locally. If the type of the command is `"minigame"`, it intercepts the regular execution path. In this case, the handler creates an instance of the correct mini game class and invokes `miniGameManager.startGame(gameInstance, callback)`, thus suspending the terminal until the student either completes the game or aborts it. In the case where the student successfully completes the game, the Promise resolves with the provided output string. If the student fails to complete the game or aborts it, the Promise rejects with an Access Denied error. This allows any scenario task to require the student to complete an interactive mini game via a single JSON field, while still decoupling the game logic from the terminal execution path.

4.4.7 Task HUD and Gamification UI

The student-centric interface is segmented into four independent modules: task panel (`taskHud.js`), points and badges widget (`pointsBadge.js`), navigation bar (`navigation.js`), and leaderboard modal (`leaderboard.js`).

Task HUD (`taskHud.js`)

The TaskHud singleton manages the collapsible side panel that displays to the student all available tasks, completed tasks, and progress within the active scenario. TaskHud subscribes to `TASK_COMPLETED`, `TASK_ADVANCED`, and `SCENARIO_COMPLETED` events from the Event Bus and updates the interface reactively without relying on polling.

The `buildTaskItem()` function is used to construct each displayed task element and returns:

- A status icon and description text for pending, active, and completed tasks.
- A Hint button that, when clicked, uses the hints API and deducts points using `PointsBadge.subtractPoints()` before appending text content.
- For flag tasks, text input and submit buttons enable students to input their captured flag string directly in the HUD.

When tasks and scenarios are completed, toast notifications are displayed using `showToast()`: A slide-in notification displayed at the screen's bottom edge for a few seconds before automatically dismissing with points earned and, in case of a badge earned, displaying badge name and icon.

A scenario selector dropdown is integrated in the HUD for students to select from available scenarios. The Task HUD interface is responsive and collapses to a floating button on small screens.

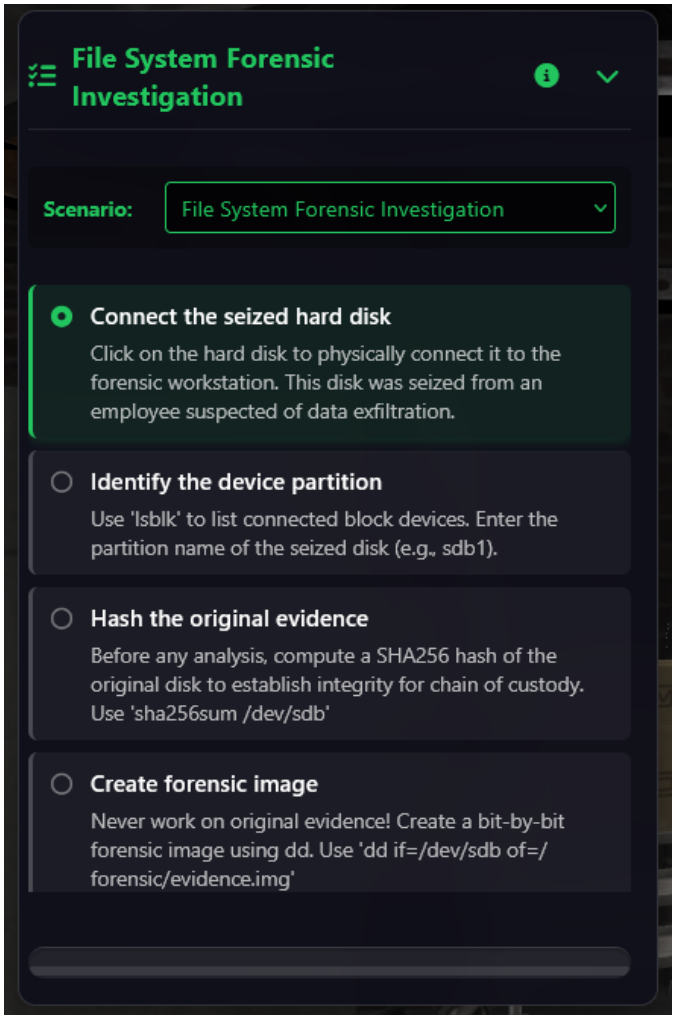

Figure 4.7: The Task HUD

Points and Badges (`pointsBadge.js`)

The `PointsBadge` singleton includes a floating widget in the top-right corner of the page, which displays the student's total points and the badges the student has acquired as icons. The widget's data is fetched from the server upon the student's login via the `init()` function and cached in `sessionStorage` for the entire session, thus reducing the number of redundant API calls during page re-renders. Three functions update the widget in real time:

- `addPoints(n)`: increments the counter and adds a scale-up animation via CSS.
- `subtractPoints(n)`: decrements the counter (used when the student uses a hint).
- `addBadge(badge)`: adds the student's acquired badges as icons in the same row, triggering a pop-in animation for the newly acquired badges. Duplicated badges are ignored.

When the student logs out, the `reset()` function resets the widget and the sessionStorage cache, thus preventing the display of the previous student's data in the same browser session.

Navigation Bar (`navigation.js`)

The navigation bar at the top of the page is initialized via the function `initNavigation()`, which dynamically updates the navigation bar throughout the student's session. It contains the following elements:

- Username and a logout button for the logged-in student.
- The student's participant ID, a compact anonymized ID used in the leaderboard and event log. By clicking the copy icon, the student can copy the ID to the clipboard for inclusion in the study form.
- Running total score: the function `updateNavScore()` fetches the score from the server upon the student's login and updates the score locally in response to every `PROGRESS_UPDATED` event.
- Link to the scenario editor (/editor) for the admin user, appearing if the current session user has the admin role.

Figure 4.8: The Navigation Bar (Admin Point of View)

Leaderboard (`leaderboard.js`)

The leaderboard is a modal panel and contains a table showing all the participants, including the total scores and the number of completed tasks, using a `GET request` on `/api/leaderboard` when the modal opens. The top three positions are decorated with medal emojis, and all the strings from the user (usernames) are sanitized using

the `escapeHtml()` utility before being inserted into the DOM to prevent cross-site scripting (XSS) attacks.

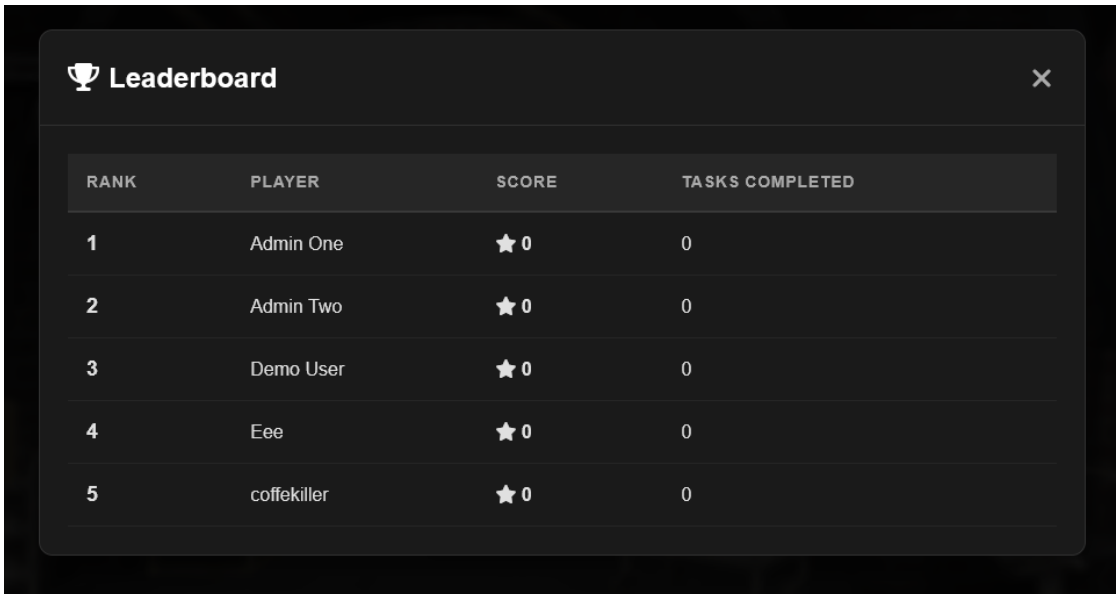

Figure 4.9: The Leaderboard

4.4.8 Tutorial System

For new users, a tutorial introducing the main game mechanics is implemented through a structured tutorial system controlled by `TutorialManager.js`. Once the user has logged in, `main.js` checks the `tutorialCompleted` flag within the user object retrieved from the session API. If the flag is `false`, a `TutorialManager` object is created and the entire tutorial process is executed. If the flag is `true`, the tutorial is bypassed and the task system is immediately activated.

Step Sequencing and Predicate Gates

The tutorial sequence is implemented as a list of step objects. Each step has the following properties:

- A **title** and **description** displayed within the overlay;
- An **event predicate**, a specific Event Bus event that must be generated by the user before the step can be completed and the next step executed.

Thus, the tutorial does not progress based on a time limit or a "Next" button click. Progression through the tutorial depends exclusively on user actions within the game world. Table 4.5 shows all steps and their associated event predicates.

Step	Trigger event	Required action
1	TUTORIAL_CANVAS_CLICKED	Click on the 3D canvas to engage pointer lock
2	TUTORIAL_MOVED	Move using WASD keys
3	TUTORIAL_INTERACTABLE_CLICKED	Click on a highlighted object
4	TUTORIAL_CONSOLE_OPENED	Open the forensic terminal (E or C)
5	TUTORIAL_COMMAND_TYPED	Type any command in the terminal
6	TUTORIAL_ESC_PRESSED	Close the terminal with Escape

Table 4.5: Tutorial steps and their predicate events in `TutorialManager.js`.

Input Blocking

During the active tutorial steps, a Tutorial Manager input blocker is used. This is done by calling the method `_installInputBlocker()`, which catches mouse and keyboard input on the canvas and prevents input that is not relevant to the current step. For example, during step 2, moving the meshes does not have any effect. The only input allowed is using the WASD keys. Moreover, the flag `window.tutorialHighlightPause` is set to prevent `scene.js` from changing object highlights during tutorial steps where highlighting is controlled by the user.

Completion and Persistence

Upon completion of the final step, the `TutorialManager` calls the `onDone()` callback method that has been passed by `main.js`. This calls `onTutorialComplete()`, which carries out the following operations: removes the input blocker, reattaches the camera controls, initializes the task system for the initial scenario, mounts and shows the Task HUD, and calls `switchScenarioWithIntro()` to display the intro modal for the first scenario. A PATCH request to `/api/auth/tutorial-complete` is made to log the completion on the server side. This prevents the tutorial from being shown again after a new login.

4.4.9 Mini-Game System

We have integrated two mini-games into the platform to enrich the command-line investigation experience with short, skill-testing interactive experiences that support the concepts introduced by the forensic scenarios. Both games utilize the same life cycle as the `MiniGameManager` class (Section 4.4.6).

Memory Dump Analyser (`DecryptionGame.js`)

This mini-game is, in fact, a **hex signature search** problem, not a decryption problem, as the module's name suggests. It is a simulation of the analysis of a memory dump to find a malware indicator of compromise. When the mini-game is initialized, the `generateMemory()` function generates a 1024-byte memory dump filled with random hex codes, where the Windows PE header signature `4D 5A 90 00` is embedded at a random position. Additionally, the function inserts eight **decoy** patterns, each matching the signature at three out of the four bytes. The task is to find the signature by clicking on the first byte (`4D`) of the signature sequence within a 90-second countdown. Incorrect clicks result in a time penalty. 5 seconds for a random mismatch, 10 seconds for clicking on a decoy that is labeled as `4D`. The hint, available after 15 seconds, points to the correct row at the cost of a penalty of 15 seconds.

Network Traffic Analyser (`SignalTracingGame.js`)

This mini-game is a simulation of the identification of Command and Control (C2) traffic from a captured network trace. The student is presented with a layered network diagram (Localhost → Gateway → ISP Node → Cloud Server → Target) and a *Traffic Inspector* window. The student selects one outgoing link at a time from the Localhost, based on the traffic characteristics, and then clicks on the **Trace** button on the *Traffic Inspector* window. The rule to identify the Command and Control traffic, as given in the mission briefing, is that the heartbeat traffic is sent with *fixed* size packets and *low jitter* (<5ms), while the legitimate traffic is bursty with variable size packets. The mini-game generates three different types of decoy links, namely, fixed-size high jitter, low jitter variable size, and random, to stop the students from memorising the patterns from the previous games. If the student incorrectly traces the path, they are given a penalty of 10 seconds. The student wins the game by tracing the path to the target `10.0.0.5` node, before the time limit of 60 seconds expires.

Game	Scenario	Mechanic	Time limit
Memory Dump Analyser	Memory forensics	Hex viewer, click correct PE header byte	90 s
Network Traffic Analyser	Network forensics	Graph traversal, analyse traffic metadata	60 s

Table 4.6: Summary of the two interactive mini-games integrated in the platform.

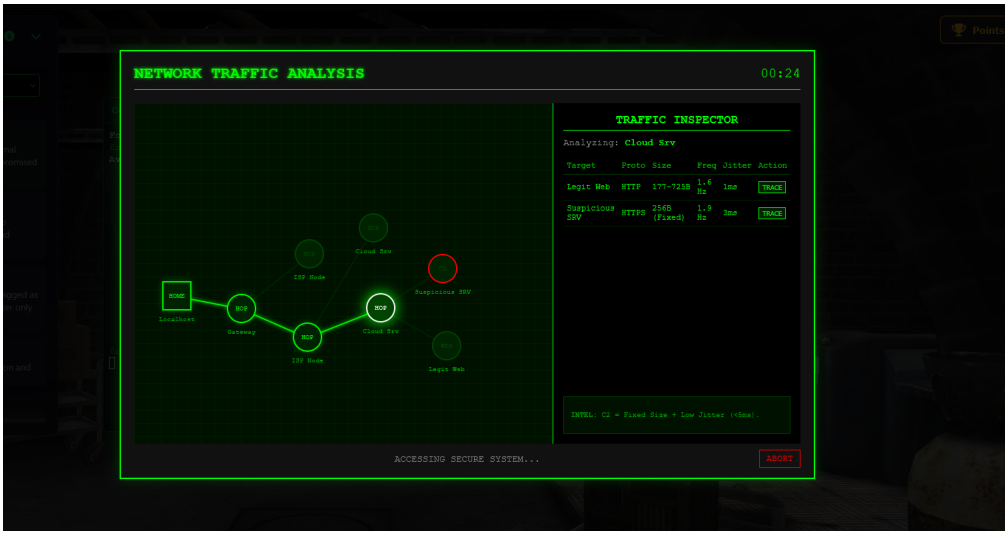

Figure 4.10: A minigame Example: trace-connection

4.4.10 Admin Interface

The platform also comes with two other admin-exclusive web pages, both of which are developed as self-contained JavaScript Single Page Applications, each served from a distinct route. Both applications make a call to `GET /api/auth/me` upon startup and automatically redirect to the main application if the session is not present or if the user does not have the role `admin`.

Admin Dashboard (`AdminApp.js`)

The Admin Dashboard, available at `/admin`, is the main analytical tool for the researcher conducting the study. It is developed as a class instance named `AdminApp` and consists of three tabs, each of which is fed data from one of the four admin API endpoints listed in Section 4.3.6:

- **Stats tab.** Upon loading, `loadStats()` makes a call to `GET /api/admin/stats` to populate five summary cards in a grid: total users, users active today, total events logged, total task completions, and total commands executed. Below the cards, `renderActivityCharts()` draws a proportional bar chart of events by type and a chronological list of activity for the last seven days, including event count and active user count.
- **Logs tab.** The event log browser makes a call to `GET /api/admin/logs` to obtain the event log table, paged on the server side (25 rows per page). The filter bar has drop-down menus for event type and scenario code, and free-text

input fields for participant ID and user ID, and filters are applied on the server side on the next page-1 request. An *Export CSV* button makes a signed URL request to `getLogsExportUrl()` and opens it in a new tab to download the filtered log without refetching in JavaScript.

- **Users tab.** A table of all registered users is populated from `GET /api/admin/users`, paged on the server side (25 users per page). Each table row contains two action buttons: *Stats* opens a modal dialog populated from `GET /api/admin/users /:userId/stats` to display a detailed breakdown of the student's commands, hints used, badges earned, scenarios played, and recent activity. *Logs* opens the Logs tab and pre-fills the filter bar with the user ID.

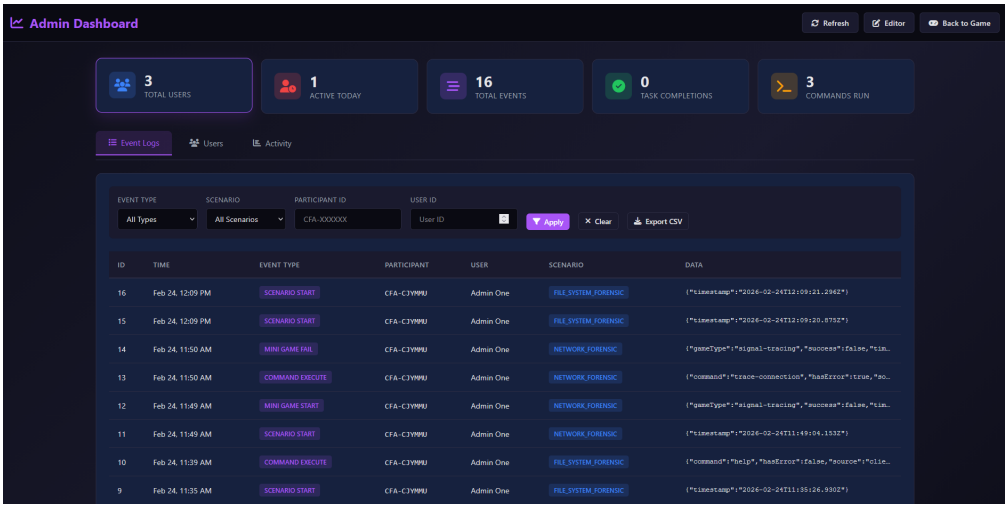

Figure 4.11: The admin Dashboard

Scenario Editor (`EditorApp.js`)

The Scenario Editor can be found at `/editor`. It's a visual tool for building and tweaking scenarios without editing `scenarios.json`. It's implemented as the `EditorApp` class and has three stages.

In stage one, `loadScenarios()` makes a `GET` request to `/api/scenarios/full`. This endpoint is only accessible by admins and returns all scenarios with `solution Value`, hint text, and `onInteract` logic blocks included. The scenarios are displayed in a sidebar, and clicking on a scenario fills the editor form on the right.

In stage two, the editor form displays all top-level fields for the current scenario: title, description, introduction, badge name, object ids to be interacted with, and custom commands JSON block. All tasks are displayed as collapsible cards. Depending on the current `checkType`, the fields within each card change: for

command tasks, `checkCommand` and `checkArgs` are shown; for interaction tasks, `interactionTarget` and `onInteract` are shown; for flag tasks, `solutionValue` is shown. Tasks can be added using the `addTask()` function and deleted using the `deleteTask()` function.

In stage three, `saveAll()` converts the current scenario to JSON using the current state of the application and sends it to the server using the `POST` request to `/api/scenarios`. As mentioned in Section 4.3.5, this will cause cache invalidation on the server and hot reload `taskLookupMap` to update the current session without requiring a server restart.

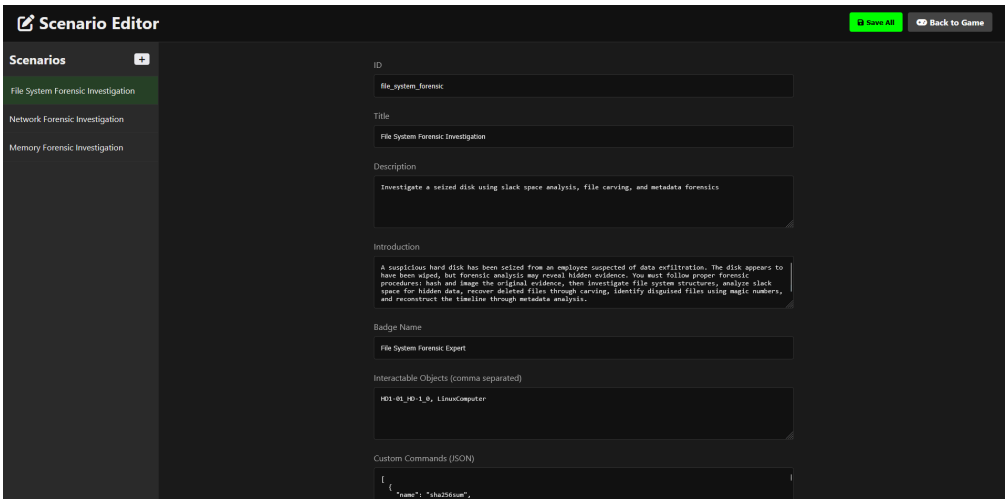

Figure 4.12: Scenario Editor

4.5 Security and Privacy Considerations in the Platform Design

Although CyberForensics Arena is a research and teaching prototype and not meant for production, we have implemented some security features during development to ensure student information is kept safe and our evaluation is reliable.

- **Authentication and Session Management**: Every single API endpoint is protected by the authentication middleware, which checks a signed cookie on every incoming request. This means, if you are not authenticated, you will be denied with an HTTP 401 response before any other code is executed. Passwords are also not stored in plain format; we use bcrypt to hash the password before storing it in the database.
- **Rate Limiting**: As discussed before, rate limiters are also employed for the

authentication and submission endpoints to protect against brute-force and flooding attacks.

- **Scenario Confidentiality**: The fields for solving tasks, such as `solutionValue`, `checkCommand`, and `checkArgs`, are read-only on the server side from `scenari os.json` and are not ever sent back to the client in our API responses. This means, even if a student is able to intercept our network traffic or use browser developer tools, they will not be able to access the actual answers.
- **Output Sanitisation**: Every string entered by the user and injected into the DOM, especially the username displayed on the leaderboard, is passed through an `escapeHtml()` function before injection to prevent reflected XSS attacks.
- **Privacy**: Students will be able to use our service under an anonymous *participant ID*, rather than their actual name. This ID is displayed on the leaderboard and event logs, allowing us to correlate the collected activity data with our questionnaires without exposing personal information to each other.

Chapter 5

Evaluation and Results

5.1 Evaluation Goals

The third objective of this thesis (O3) is to conduct an assessment of usability, engagement, and educational effectiveness, which is carried out through a pilot study. This chapter describes how this objective was achieved, including the process and results of this pilot study. To operationalize thesis objective O3, the pilot study examined three evaluation goals: usability, engagement, and perceived learning

- **EG1 - Usability**: This objective is to assess how easy it is for participants to use the platform, regardless of their technical background, and regardless of whether they have prior knowledge in digital forensics.
- **EG2 - Engagement**: This objective is to measure how immersive, engaging, and narratively cohesive the experience was for the participants, according to the gamification and human factors principles explained in Chapter 3.
- **EG3 - Perceived Learning**: This objective is to assess whether, from a pedagogical perspective, the platform was able to help participants understand how digital forensics works, and how it can be used, instead of just providing instructions to perform a series of steps.

It is important to note that these objectives were tested across three different groups of participants, each from a different technical background: MSc Cybersecurity, BSc Computer Science, and those without a technical background. This is a between-groups approach, which is useful to understand how prior knowledge affects the experience and whether it is accessible to those without a technical background, which is not necessarily part of the main target audience. To conduct this pilot study, two types of data are gathered. First, a series of questions was asked through a survey, which was given to the participants before and after using the platform.

This survey included a series of questions, including a System Usability Scale, as described in Brooke (1996) [16]. Second, a series of objective data was gathered automatically through a series of logs generated by the platform, as explained in Section 3.5. This is a unique characteristic of this platform, which allows it to automatically track all significant actions taken by each participant at a millisecond resolution, without needing to manually set up any additional tools to track this information. It is important to note that, due to the nature of a pilot study, where sample size is limited and selection is not random, this analysis is purely descriptive, and no other tests are carried out.

5.2 Study Design: Participants, Procedure and Experimental Tasks

5.2.1 Participants

A convenience sample was used, and participants were divided into three groups, each representing a different set of technical backgrounds, as shown in Table 5.1.

Table 5.1: Participant groups and their profiles.

Group	N	Profile
G1 – MSc Cybersecurity	5	Target audience; solid background in operating systems, networking, and cybersecurity
G2 – BSc Computer Science	3	General CS background; limited or no prior exposure to digital forensics
G3 – No technical background	4	Non-CS participants; baseline group to assess onboarding quality and general usability

This is a between-group design, and it is a conscious choice. Rather than seeing it as a weakness, it is a positive, as it allows us to examine how pre-existing knowledge affects the experience. G1 is focused on the main audience, but G2 and G3 provide a broader perspective on how accessible and usable it is for a broader audience.

In total, N=12 participants completed the study. Of these, 10 produced usable platform logs; two participants (one from G1 and one from G3) submitted the questionnaire but had no corresponding event-log entries due to session expiry or connectivity issues.

5.2.2 Procedure

This is a remote, asynchronous study, and each participant was sent a link to an online form and a link to the platform URL, and they were told to access it at their convenience.

The procedure is divided into three phases:

1. **Pre-session survey** (~2 minutes): Before they access the platform, they are asked to complete a form, which is the first part of the online form. This asks questions about background information, which includes what level of study they are at, how familiar they are with Linux and command line, how familiar they are with digital forensics, and how familiar they are with gamified learning environments.

2. **Platform session** (~45–60 minutes): participants were instructed to complete the onboarding tutorial followed by the first investigation scenario (File System Forensic Investigation), which constituted the main evaluated task sequence of the pilot study. Although the formal protocol focused on this path, platform logs showed that a small number of participants also explored content outside the requested scenario.. Participants were instructed to work at their own pace, use hints if needed, and note their `CFA-XXXXXX` participant ID displayed on the platform.

3. **Post-session survey** (~8–10 minutes): Once they have finished, or chosen not to continue, they access the online form again and complete a second part. This asks questions about usability, how engaging they find it, and how they rate it in terms of learning, and they are asked to enter their unique ID, `CFA-XXXXXX`.

5.2.3 Instruments

Pre-session Questionnaire

Before the session, we asked participants to provide five pieces of information to help create a profile of the participants. This included the level of study participants were currently engaged in (categorical), participants' self-rated experience with Linux and the command line (1-5 scale), participants' self-rated knowledge of digital forensics (1-5 scale), whether or not participants had experience with CTF (yes/no), and whether or not participants had experience with gamified learning platforms (yes/no). This provided us with the ability to verify the anticipated differences in participant profiles between G1, G2, and G3.

Post-session Questionnaire

The post-session questionnaire was comprised of three parts:

- **System Usability Scale (SUS)** [16]: A 10-question survey on a 5-point Likert scale that has been shown to provide a reliable measure of participants' perceptions of system usability. The answers can then be converted into a 0-100 scale using the standard SUS formula. Scores above 68 or higher are considered above average usability.
- **Custom Likert items**: Five questions about participants' engagement (such as immersion and usefulness of the narrative) and five questions about participants' learning experience (such as the participants' understanding of the forensic process and whether or not the experience was of appropriate difficulty).
- **Open-ended questions**: Three optional questions asking participants what they found most helpful, what was most confusing or frustrating for them, and what they would change for next time.

5.3 Metrics and Data Collection

The evaluation will make use of two types of information: automatically collected information from the interactions in the platform and answers from the questionnaire.

5.3.1 Platform Interaction Data

Any meaningful event in CyberForensics Arena is recorded in the `event_log`table. From the raw data in the event log table for each participant, a CSV file is exported in the administrator dashboard.

Table 5.2: Log-derived metrics used in the evaluation.

Metric	Source event	Description
Completion rate	`scenario_end`	Ratio of completed tasks to total tasks (`completedTasks / totalTasks`)
Session duration	`scenario_start`, `scenario_end`	Active session time, computed as the sum of inter-event gaps shorter than 30 minutes. This excludes idle time due to multi-day or multi-session usage.
Per-task completion time	`task_submit`	Elapsed time between consecutive correct submissions. The original `timeMs` field was `null` in all records due to a client-side instrumentation issue; therefore, per-task times were derived from event-log timestamps.
Wrong attempts per task	`task_submit`, `flag_submit`	Count of submissions where `correct = false`, per participant per task
Hint usage	`hint_request`	Number of hints requested per participant
Command error rate	`command_execute`	Proportion of commands where `hasError = true`
Most common wrong commands	`command_execute`	Frequency distribution of commands with `hasError = true`

5.3.2 Questionnaire Data

The following metrics are derived from the results of the post-session questionnaire:

- **SUS score**: The SUS score is derived from the 10 SUS questions using the standard formula to obtain a score from 0 to 100 for each participant.
- **Engagement score**: The engagement score is the average of the five custom engagement Likert questions with a scale of 1-5.
- **Perceived learning score**: The perceived learning score is the average of the five custom learning Likert questions with a scale of 1-5.

- **Qualitative themes**: The qualitative themes are derived from the three open-ended questions using thematic analysis.

These metrics are derived for each participant individually before aggregating them to obtain the average per group (G1, G2, G3). The participant ID `CFA-XXXXXX` in the post-session survey is used to match the questionnaire results with the corresponding event logs.

5.4 Data Analysis and Results

This section shows the results obtained for the pilot study. To calculate the metrics, a custom Python script has been designed, which reads the exported event log CSV file and the Google Forms responses CSV file, calculates the metrics defined in Section 5.3, and produces the figures shown throughout this section.

5.4.1 Participant Profiles

The background information for all participants (N=12) can be summarized as follows:

- Group G1 (5 MSc Cybersecurity students) reported the highest mean CLI rating ($\bar{x}$=4.6/5.0) and the most CTF experience.
- Group G2 (3 BSc Computer Science students) reported moderate CLI skills ($\bar{x}$=4.0) and Digital Forensics knowledge ($\bar{x}$=3.0).
- Group G3 (4 participants with non-technical backgrounds) reported the lowest CLI ($\bar{x}$=1.3) and DF ($\bar{x}$=1.3) ratings, with no prior CTF experience.

These differences are consistent with the intended evaluation groupings.

5.4.2 Task Completion and Session Duration

Seven out of ten participants with logs completed all 17 FS tasks (100% completion). P8 (G3) attempted 7 out of 17 tasks and then stopped, perhaps judging the tasks too difficult since their CLI was rated at about 1. P9 and P11, both from G1, only attempted the first task. This may be attributed to time constraints rather than difficulty since their CLI and DF self-rating were high.

The range of active sessions was between 1 and 37 minutes. On average, the sessions of those who successfully completed the entire FS scenario took about 21 minutes to complete. Among these, the expert participants P3, P7, and P10 took less than 12 minutes.

5.4.3 Wrong Attempts and Difficulty Hotspots

fs_task_2 stands out as the mayor difficoult task, with 32 wrong attempts and accounting for 59% of all incorrect attempts. As depicted in Figure 5.1, the majority of the incorrect attempts, i.e., 25 out of 32, were recorded for G3 participants who struggled with the syntax of the foremost command. This task represents the first challenge for participants to work with a specialized forensic tool, and it represents a sharp increase in task difficulty from Task 1.

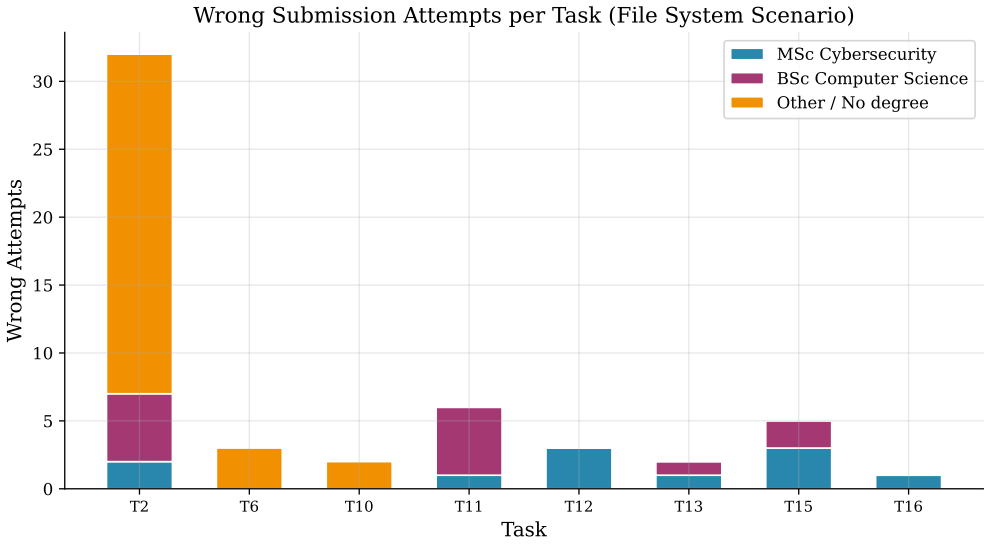

Figure 5.1: Wrong submission attempts per task (File System scenario), stacked by group.

5.4.4 Hint Usage and Command Errors

There were only three hint requests in total among participants. Two of these were observed within the main evaluated File System scenario: P12 (G3) requested two hints in `fs_task_2`. One additional hint request was logged for P7 (G1) during a Memory task, reflecting exploratory interaction outside the formal study path. Since the pilot evaluation focused on the tutorial and File System scenario, this latter event is reported only for completeness and is not interpreted as part of the main scenario-level analysis. It is possible that this indicates participants found the task descriptions and instructions clear enough, or they were simply not willing to use hints because of the penalty system's impact on their scores.

The average CER of participants is 13.9% in total. It is interesting to notice that this figure does not correlate directly with technical background: P5 (G1, MSc Cybersecurity) has the highest CER of 27.3%, while P12 (G3, no university degree)

has a relatively low CER of 10.0%. It seems that CER actually depends more on people's problem-solving strategies: some are more likely to use trial-and-error, while others think more carefully about their commands.

5.4.5 Self-Reported Measures: Usability, Engagement, and Learning

This section summarizes the results of the post-session questionnaire (N=12). As visualized in Figures 5.2 and 5.3, the data highlights distinct trends based on technical background.

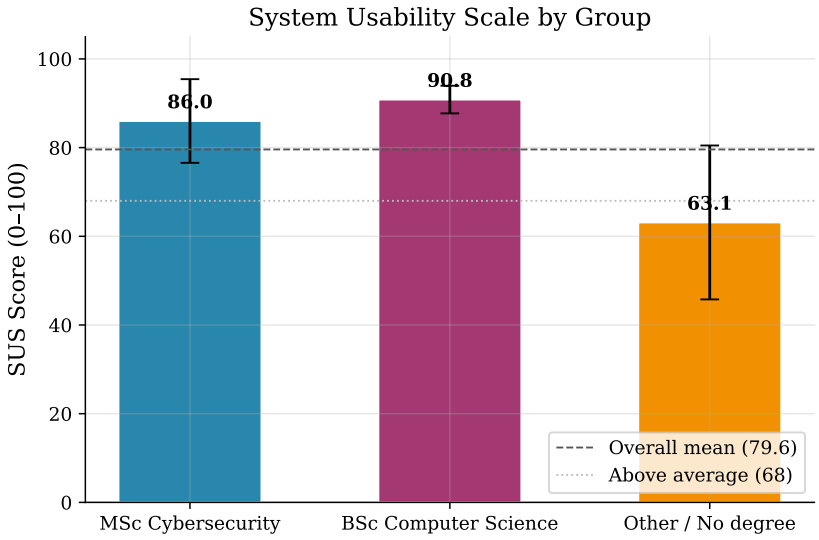

Figure 5.2: Mean SUS score by group, with standard deviation. The dashed line indicates the overall mean (79.6) and the dotted line represents the above-average threshold (68).

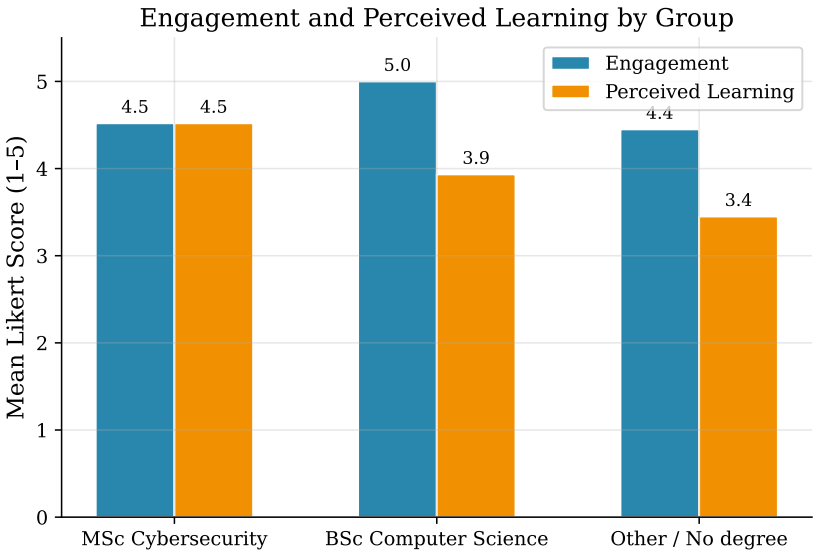

Figure 5.3: Mean engagement and perceived learning scores by group.

The SUS score for the entire system is 79.6, which falls under the "Good" category according to the adjective rating scale developed by Bangor et al[17]. Considering the individual groups, G1 had an average of 86.0 and G2 had an average of 90.8, both of which fall under the "Excellent" category. G3 had an average of 63.1, which is below the 68 benchmark, indicating that participants without a technical background found the system less intuitive.

Engagement scores are uniformly high across all groups, with an overall mean of 4.62 out of 5.0. Notably, G2 had a perfect score of 5.0, indicating that the participants from the BSc Computer Science course found the system very engaging. The overall perceived learning score was 4.04, indicating that participants believed the system helped them learn more about digital forensics.

5.4.6 Per-Task Completion Time

Per-Task Completion Time. Tasks T1 and T2 always took the most time for everyone to complete, which is understandable since they represent the early learning phase. After task T3, everyone's finish times stabilized to less than 100 seconds, which shows that the skills developed during the early tasks can be transferred to the later tasks.

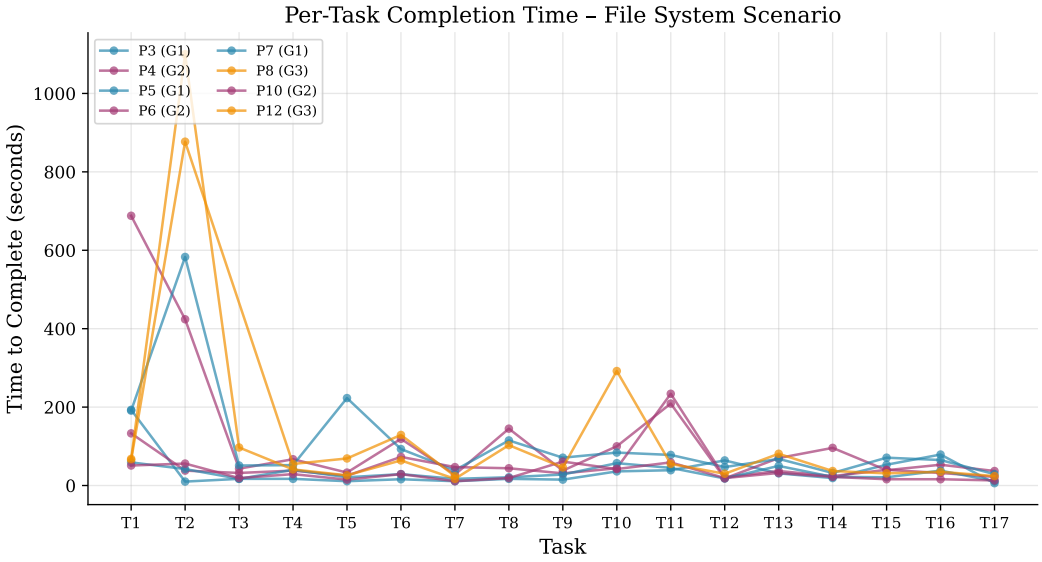

Figure 5.4: Per-task completion time for the File System scenario. Each line represents one participant; data points where the inter-submission gap exceeded 30 minutes are excluded.

5.4.7 Qualitative Feedback

Ten of the twelve participants provided at least one open-ended comment. These were positive towards the **narrative and immersion**, with P7 highlighting "the story and the investigation path that follows a forensic logic." The **progressive scaffolding** was also praised, as P12 stated that "the suggestions on each task displayed was really helpful." The major problem was with the **command syntax acting as a barrier**, which was as expected given the high error rate with the early tasks.

5.5 Discussion, Limitations and Threats to Validity

5.5.1 Discussion

The pilot study addresses the three evaluation objectives set in Section 5.1:

- **Usability (EG1):** The target audience found the platform highly usable (G1 and G2 scoring > 86.0 on SUS). The lower score for G3 highlights the inherent challenge of a command-line interface for novices.

- **Engagement (EG2):** All participants were highly engaged throughout the experiment, with an average rating of 4.62 out of 5.0, indicating that gamification elements were successful in keeping participants highly motivated.
- **Perceived Learning (EG3):** With an average rating of 4.04 out of 5.0, it's clear participants felt they were learning useful forensic skills, especially because of the progressive scaffolding design.

5.5.2 Limitations and Threats to Validity

However, the major limitation of this pilot study is that the sample size is limited to 12 participants only, collected based on convenience sampling, making it impossible to perform inferential statistics analysis. Moreover, since the participants did not have sufficient log data due to a tracking mistake, the behavioral data is based on 10 participants only. It is also important to note that since the study was conducted in an asynchronous manner, there is environmental variability involved. Finally, due to the aforementioned tracking bug invalidating the automatic `timeMs` duration metric, session lengths had to be manually derived from event timestamps by discarding inactive periods longer than 30 minutes, which captures estimated rather than exact active time.

Chapter 6

Conclusions and Future Work

6.1 Summary of the Work and Main Contributions

This thesis presents the design, implementation, and evaluation of CyberForensics Arena, a novel gamification tool to educate users in the field of digital forensics through engaging hands-on exercises in a browser-based environment. The overall objective of this thesis is to address the high learning barrier and lack of engaging hands-on experiences in cybersecurity education by using a realistic Virtual File System (VFS) and a gamification approach in a 3D interactive storyline.

In order to create CyberForensics Arena, a robust client-server architecture is required to securely simulate different Linux command-line tools (e.g., `dd`, `foremost`, `tshark`) without requiring any virtual machines or infrastructure. To achieve this, a realistic VFS is proposed and implemented. Subsequently, a scaffolding approach is proposed to create a progressive learning experience using this realistic VFS.

The main contributions of this thesis are threefold:

1. **Technical Platform:** A functional lightweight web-based digital forensics simulator using a client-server architecture, eliminating the need for virtual machines.

2. **Pedagogical Design:**An instructional design approach using a scenario-based learning methodology with gamification elements (scoring, badges, hints) to maintain engagement while progressively removing scaffolding as the learner's competence grows.

3. **Empirical Evaluation:** A pilot study (N=12, with 10 usable platform logs)

providing initial descriptive evidence on usability, engagement, and perceived learning, while also highlighting command-line-related barriers and differences associated with participants' prior technical background.

6.2 Lessons Learned for Cybersecurity and Digital Forensics Education

The design process and subsequent pilot study provided many interesting takeaways that apply to cybersecurity education as a whole.

Firstly, the **Virtual File System** approach proved to be an excellent middle ground between realism and usability. It does not offer the same degree of freedom that comes with virtual machines but does offer enough realism to cover the basic concepts of file carving, slack space analysis, and metadata recovery. This limits the environment in such a way that it makes it easier on the cognitive load of the learner, allowing them to focus on the pertinent concepts rather than getting sidetracked by system details.

Secondly, the research pointed out that there is a syntax barrier that exists in CLI applications. This is evidenced by the feedback that participants provided, especially in the high error rate that participants exhibited in the `fs_task_2` challenge, in which participants were tasked to identify the devices using the lsblk command. This shows that even non-technical participants would find it difficult not just to grasp the concepts of digital forensics but even basic concepts of Linux CLI syntax, such as distinguishing between physical disk partitions like sdb1 and physical disks like sdb.

Lastly, the power of storytelling proved to be an intrinsic motivator. Regardless of the participant's level of experience, the fact that participants were able to follow the investigative story and that it helped them in their task greatly motivated them. It provided them with context, such as computing the hash value of files.

6.3 Practical Implications and Possible Adoption in Courses/Labs

CyberForensics Arena is an immediately applicable extension to any learning and/or professional development curriculum. The browser-based installation is quick and easy and removes typical hurdles associated with digital forensics labs since there is no need to deal with VM image sizes and complicated local lab configurations.

Academic Integration. CyberForensics Arena is an appropriate tool for BSc and MSc courses in Computer Science and Cybersecurity. The tool is an excellent

starting point before proceeding to other digital forensics labs with real virtual machines. The automated objective grading system relieves instructors from the burden of grading, allowing them to effectively use this tool with large student numbers.

Remote and Asynchronous Learning. The tool is accessible from any current web browser and is therefore well-suited to remote and asynchronous learning. This is particularly relevant to continuing professional development and/or corporate learning scenarios, where flexibility is beneficial and IT support is not always available.

6.4 Future Work and Extensions of the Cyber Forensics Arena

Future directions for CyberForensics Arena are many and varied, and they build upon the success of the present system while introducing new possibilities and directions for the future.

Expanded Scenario Content. The immediate next step is the full implementation of the Memory Forensics and Incident Response scenarios. Expanding the simulated command set to include tools like `Volatility` will allow the platform to cover fileless malware analysis and live response techniques. Furthermore, allowing instructors to author their own scenarios via a graphical user interface or JSON templates would increase the platform's versatility.

Multiplayer and Collaborative Modes. In the real world of digital forensics, collaboration is the key for the Incident Response team. A multiplayer version of the game where students can work together, share results in a shared digital notebook, and communicate with each other through the in-game terminal will be similar to the real-world environment and will be useful in developing collaboration skills.

Enhanced Scoring and Formative Feedback. Currently, the system only provides binary scoring. However, the addition of partial scoring for partially correct command sets and AI-driven hints based on the student's specific terminal input will provide more useful and effective feedback for the student and help them avoid dead ends due to syntax errors in the command line. A proposed improvement for the terminal interface is the addition of tab auto-completion, identified in the user feedback survey, which will be implemented in the next minor version of the system.

Large-Scale Effectiveness Evaluation. Finally, future research should conduct a larger-scale, multi-institution evaluation featuring a controlled experimental design. . By doing pre- and post-knowledge tests, the actual knowledge gained by the student can be measured, not just the perceived knowledge gained, and the pedagogical impact of the gamified VFS approach can be validated.

Appendix A

User Manual

This appendix provides instructions on how to set up, deploy, and run the CyberForensics Arena platform. The system is designed to be lightweight and portable, requiring minimal dependencies.

A.1 Prerequisites

To run CyberForensics Arena, the following software must be installed on the host machine:

- **Node.js**: Version 18.x or higher is recommended.
- **npm**: Node Package Manager (usually bundled with Node.js).

No external database server is required, as the application uses an embedded SQLite database.

A.2 Installation and Setup

Follow these steps to initialize the environment:

1. **Clone or Extract the Repository**: Obtain the source code and navigate into the root directory of the project.
2. **Install Server Dependencies**:

```
cd server
npm install

```

3. **Install Client Dependencies**:

```
cd ../client
npm install

```

A.3 Running the Application Locally

For local development or testing, you can start the frontend and backend servers separately.

1. **Start the Backend Server**:

```
cd server
npm run dev

```

This will start the Node.js API server on `http://localhost:3000`. The SQLite database will be automatically created in the `server/db` folder if it does not already exist.

2. **Start the Frontend Development Server**:

```
cd client
npm run dev

```

This will launch the Vite development server. By default, it runs on `http://localhost:5173` Open this URL in your web browser to access the platform.

A.4 Production Deployment

To deploy the application in a production environment (e.g., for classroom use), it is recommended to build the frontend and serve it statically, or use a process manager like PM2 for the backend.

1. **Build the Frontend**:

```
cd client
npm run build

```

This generates an optimized static build in the `client/dist` directory.

2. **Configure the Backend for Production**: Ensure that the `NODE_ENV` environment variable is set to `production`. The backend can be configured to serve the static frontend files directly.

Appendix B

Programmer Manual

This appendix provides an overview of the platform's architecture and detailed instructions for extending its functionality. It acts as a reference for future developers, educators, or researchers who wish to evolve CyberForensics Arena by adding new investigation scenarios, custom console tools, 3D evidence, or additional telemetry tracking.

B.1 Codebase Architecture Overview

The CyberForensics Arena is structured as a decoupled client-server application.

B.1.1 Client-Side Architecture

The client is a vanilla JavaScript Single Page Application (SPA) built with Vite and Babylon.js. It adheres to a strict three-layer architecture to ensure maintainability:

- **UI Layer** (`client/src/ui/`): Manages the 2D DOM elements, including the forensic terminal (`console.js`) and the head-up display (`taskHud.js`).
- **Logic Layer** (`client/src/logic/`): Contains the scenario state machine (`taskManager.js`), which tracks progress, validates interactions, and communicates with the backend REST API.
- **Rendering Layer** (`client/src/3d/`): Handles the Babylon.js WebGL scene, 3D assets, camera controls, raycasting, and mesh highlighting (`scene.js`).

These layers are completely decoupled and communicate exclusively via a global Event Bus (`client/src/eventBus.js`), broadcasting events such as `MESH_CLICKED` or `TASK_COMPLETED`.

B.1.2 Server-Side Architecture

The backend is a Node.js Express application holding the single source of truth for validation, state, and scoring.

- **Virtual File System** (`server/src/vfs/`): An in-memory JavaScript tree data structure where user commands are executed. User-specific state is persisted to the `user_vfs_state` table in the SQLite database.
- **Validation Service** (`server/src/services/validationService.js`): A stateless pure function that verifies submitted answers against the requirements defined in the JSON configuration.
- **API Routes** (`server/src/routes/`): Modular endpoints for authentication, task submission, console execution, scenario fetching, and telemetry tracking.

B.2 Extending the Platform

B.2.1 Adding Custom Console Commands

The terminal emulator connects to the `/api/console/execute` route. To simulate a new forensic tool (e.g., `volatility` or `awk`):

1. Open `server/src/routes/console.js`.
2. Locate the `executeCommand()` switch statement.
3. Add a new case for the command name.
4. Implement the logic to interact with the `Virtual File System`. If the command creates or modifies files, you must invoke the `VFS` update methods and ensure the `vfsModified` flag is set so the changes persist to SQLite.
5. Return the simulated text output string to be displayed on the client.

B.2.2 Defining New Scenarios

CyberForensics Arena is data-driven. The structure, narrative, and validation logic for scenarios are defined entirely in `server/data/scenarios.json`. Modifying this file (or using the integrated Scenario Editor interface) updates the curriculum without requiring application code changes.

A standard scenario object includes:

```
{
  "title": "Operation Name",
  "description": "Short summary",
  "introduction": "The narrative briefing shown to the user",
  "badge": "Badge Identifier",
  "interactableObjects": [ ... ],
  "customCommands": [ ... ],
  "tasks": [ ... ]
}
```

A typical task definition looks like this:

```
{
  "id": "fs_task_4",
  "title": "Create forensic image",
  "details": "Use dd to create a bit-by-bit forensic image.",
  "points": 25,
  "checkCommand": "dd",
  "checkArgs": ["if=/dev/sdb", "of=/forensic/evidence.img"]
}
```

Supported `checkType` values for validation include:

- `interaction`: Validates that a user clicked a specific 3D mesh (defined in `interactionTarget`).
- `flag`: Validates a free-text input against `solutionValue`.
- *Command validation*: Omits `checkType` and instead uses `checkCommand` and `checkArgs` to ensure the user typed the exact shell command.

B.2.3 Adding New 3D Evidence Objects

If you wish to introduce new physical interactable evidence (such as a seized mobile phone or a USB drive) into a scenario:

1. Using a 3D modeling tool (like Blender), add the object to the GLTF/GLB scene file and assign it a unique, semantic mesh name (e.g., `USB_Drive_01`).
2. In `server/data/scenarios.json`, locate the relevant scenario and add this exact mesh name to the `interactableObjects` array.
3. When the learner clicks the object, the **Rendering Layer** will broadcast a `MESH_CLICKED` event. The **Logic Layer** will listen for this event and, if the object is listed in the scenario's `interactableObjects`, it will trigger the corresponding task completion or attach the device's contents to the VFS.

B.2.4 Integrating New Mini-Games

Mini-games act as visual interactive puzzles supplementing terminal analysis. To develop and add a new mini-game:

1. Create the mini-game UI logic and styling within the **UI Layer** (e.g., `client/src/miniGames`
2. Hook the logic into the **Event Bus**. When the puzzle is successfully solved, the component must emit a `TASK_COMPLETED` event.
3. Define a corresponding task in `scenarios.json` where the `checkType` relies on a custom flag or a manual client-side submission (via a `POST` request to `/api/tasks/:taskId/submit`), rather than standard console command validation.

B.2.5 Extending the Telemetry and Event Tracking

For researchers wishing to capture additional learning analytics or pilot new UI studies:

1. The database schema logging is localized in `server/src/db/schema.js`. Ensure the `event_log` table schema meets your analytical requirements.
2. New event variants must be registered in the `EventTypes` export (e.g., `services/eventLog.j`
3. Dispatch the new events using the `logEvent()` pure function, passing the required fields (`participantId`, `eventType`, and the `JSON` payload). This guarantees the event falls under the standard tracking pipeline and will be visible within the Admin Dashboard CSV exports.

Bibliography

[1] NIST Computer Security Resource Center. *cybersecurity*. National Institute of Standards and Technology. URL: https://csrc.nist.gov/glossary/term/cybersecurity (cit. on p. 7).

[2] NIST Computer Security Resource Center. *Digital Forensic*. National Institute of Standards and Technology. URL: https://csrc.nist.gov/glossary/term/digital_forensics (cit. on p. 7).

[3] NIST Computer Security Resource Center. *Incident Response*. National Institute of Standards and Technology. URL: https://csrc.nist.gov/glossary/term/incident_response (cit. on p. 7).

[4] NIST Computer Security Resource Center. *Chain of Custody*. National Institute of Standards and Technology. URL: https://csrc.nist.gov/glossary/term/chain_of_custody (cit. on p. 8).

[5] Clark C. Abt. *Serious Games*. New York: The Viking Press, 1970 (cit. on p. 8).

[6] David R. Michael and Sandra L. Chen. «Serious Games: Games That Educate, Train, and Inform». In: *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. Also published as book by Thomson Course Technology, 2006. ACM, 2005 (cit. on pp. 8, 32).

[7] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. «From Game Design Elements to Gamefulness: Defining "Gamification"». In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. ACM, 2011, pp. 9–15. DOI: 10.1145/2181037.2181040. URL: https://dl.acm.org/doi/10.1145/2181037.2181040 (cit. on pp. 8, 35).

[8] NIST Computer Security Resource Center. *Chain of Custody*. National Institute of Standards and Technology. URL: https://csrc.nist.gov/glossary/term/cybersecurity_incident (cit. on p. 9).

[9] IBM. *What is digital forensics?* IBM. URL: https://www.ibm.com/think/topics/digital-forensics (cit. on p. 13).

[10] Michael Zyda. «From Visual Simulation to Virtual Reality to Games». In: *Computer* 38.9 (2005), pp. 25–32. DOI: 10.1109/MC.2005.297 (cit. on p. 32).

[11] Edward L. Deci and Richard M. Ryan. «The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior». In: *Psychological Inquiry* 11.4 (2000), pp. 227–268 (cit. on p. 36).

[12] Juho Hamari, Jonna Koivisto, and Harri Sarsa. «Does gamification work? – a literature review of empirical studies on gamification». In: *47th Hawaii International Conference on System Sciences*. 2014, pp. 3025–3034 (cit. on p. 38).

[13] John Seely Brown, Allan Collins, and Paul Duguid. «Situated Cognition and the Culture of Learning». In: *Educational Researcher* 18.1 (1989), pp. 32–42 (cit. on p. 41).

[14] John Sweller. «Cognitive Load During Problem Solving: Effects on Learning». In: *Cognitive Science* 12.2 (1988), pp. 257–285 (cit. on p. 41).

[15] Arnaud Guedj. *Flynn's Secret Lab (TRON Legacy)*. Licensed under CC BY 4.0. 2021. URL: https://sketchfab.com/3d-models/flynns-secret-lab-tron-legacy-0429196769e040bd84e84040daccc40a (cit. on p. 84).

[16] John Brooke. «SUS: A Quick and Dirty Usability Scale». In: *Usability Evaluation in Industry*. Ed. by Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland. London: Taylor & Francis, 1996, pp. 189–194 (cit. on pp. 104, 106).

[17] Aaron Bangor, Philip T. Kortum, and James T. Miller. «Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale». In: *Journal of Usability Studies* 4.3 (2009), pp. 114–123 (cit. on p. 111).