EVALUATING STUDENTS' PERFORMANCE IN CYBERSECURITY SCENARIOS USING BINARY TREES

Alexandru-Andrei GHIȚĂ[1], Mihai-Daniel CHIROIU[2], Dinu ȚURCANU[3]

Cyber education has become a critical focus in universities worldwide, largely due to the growing shortage of experts that the cybersecurity domain is currently facing. To address this need, it is essential to provide efficient and practical training, with an emphasis on hands-on laboratories and real-world scenarios. This paper explores the process of designing and evaluating a cybersecurity scenario, in the context of Cybersecurity Basics course at UNST POLITEHNICA Bucharest, conducted over two student generations. The final aim of the evaluation is to gather and analyze live scenario statistics and integrate them in form of binary trees, as a term of novelty, consequently understanding the thought process and decisions of each student during the final assessment exam, ultimately maximizing the learning outcome for the next generations.

Keywords: cybersecurity, education, cyber range, cybersecurity exercise, evaluation

1. Introduction

There is a global shortage of approximately four million cybersecurity professionals across the industry. This gap has grown more significantly, especially considering the workforce only increased by 12.6% between 2022 and 2023 [1]. On the other hand, the global cost of cybercrime is expected to rise from $9.22 trillion in 2024 to $13.82 trillion by 2028 [2]. When compared to the gross domestic product of the world, projected to be around $109 trillion in 2024 [3], this number becomes particularly striking. To address this, more and more investments are made into the field of cybersecurity education. For example, the European Union has committed to investing €55 million in education programs in 2024 [4]. In the same vein, in 2024, the U.S. allocated investments totaling hundreds of millions of dollars under the National Cyber Workforce and Education Strategy (NCWES), in collaboration with the private sector. This initiative aimed to bridge the gap between

[1] Faculty of Automatic Control and Computer Science, The National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: alexandru.ghita2611@upb.ro
[2] Faculty of Automatic Control and Computer Science, The National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: mihai.chiroiu@upb.ro
[3] Technical University of Moldova, Faculty of Electronics and Telecommunications and National Institute of Innovations in Cybersecurity "CYBERCOR", Chisinau, Republica Moldova, e-mail: dinu.turcanu@adm.utm.md

the demand for cybersecurity professionals and the available workforce, with the goal of transitioning to skills-based hiring and fostering the development of robust cyber education ecosystems [5].

The process of teaching cybersecurity differs from other subjects because it cannot rely on a fixed curriculum. It must adapt to the ever-growing diversity of threats and evolve along with the emerging technologies in the market. Additionally, it demands extensive hands-on practice with real-world scenarios, and according to NIST [6] 59% of technologists consider this the most effective method for acquiring new skills. However, simulating cyber threats on physical or production systems poses risks such as irreversible damage and difficulty in recreating the environment. To overcome these issues, cyber range platforms are used—virtualized environments on dedicated hardware designed to safely deploy and replicate cybersecurity scenarios for educational use.

Cybersecurity scenarios have two main goals: to assess the relevance of the content for improving future versions and to evaluate participant performance by checking if the difficulty and objectives were appropriate. Achieving these goals requires collecting and analyzing live data during and after the exercise. However, this is often challenging due to the large volume of data from many participants. Despite its value, thorough analysis and application of findings in future scenarios are uncommon, and there is limited research addressing this process.

This paper analyzes the design and evaluation of a cybersecurity scenario conducted as a final exam for two student cohorts at University *POLITEHNICA* of Bucharest, using the *OpenStack* platform as a cyber range. Designed in alignment with the *COFELET* framework, the scenario features an innovative approach by using custom trackers to collect live statistics, structured as *binary trees* in *JSON* format. A post-exercise analysis of students' challenge-solving strategies helps assess difficulty and refine both the course content and future scenario iterations. The paper also presents a visual method for representing the collected data.

The rest of the paper is structured as follows: section 2 reviews the existing research done on the topics of the paper. Section 3 outlines the structure of the scenario in alignment with the COFELET framework. Section 4 explores each of the component challenges within the scenario and details the methods used for extracting exam statistics. In section 5 we describe the data pre-processing method and delve into the binary tree-creation algorithm, illustrating how the data can be visualised on a time axis. Section 6 is dedicated to the evaluation of the implementation and tracing the remaining features left to be accomplished. Section 7 is reserved for final conclusions.

2. Background and Related Work

2.1. Educational Frameworks

An educational framework represents a set of guidelines and rules intended to aid in the process of delivering information to a targeted group of individuals regarding a given subject.

Existing research found in this direction falls into two broad categories: papers that expand an already existing conceptual framework and papers that propose their own framework. The first category is comprised of papers [7], [8] that structure their framework on top of *COFELET* (Conceptual Framework for Developing Cyber Security Serious Games) and [9] that presents a framework based on *PBL* (Problem-based Learning). On the other hand [10], proposes a framework that extracts a list of evaluation criteria based on other papers that asses cyber exercises, and further expands the list with three additional key points, namely *Learning Impact*, *Learning Outcome* and *Learning Experience and Teaming*, finally composing the *TARGET* framework that is used to evaluate the *Iceberg* cyber scenario.

In [11] the proposed framework is designed upon *adversial thinking*, in which the participants act from the position of defenders, but think as if they were attackers in the attempt to solve a series of vulnerabilities implemented by the planning team.

We chose to expand the COFELET framework implementation for our scenario approach, as showcased in [12] over the PBL implementation found in [13], as it is precisely designed for developing cyber scenarios in a straightforward and structured manner, with specific index terms such as *Learning Objectives*, *Scenario Complexity* and so forth.

2.2. Cybersecurity Scenarios

Cybersecurity exercises or scenarios are practical approaches that provide specialists with valuable skills and insights on various topics and vulnerabilities encountered in real-world environments. Their broad classification involves two categories, namely hands-on and tabletop [14]. The latter refers to purely theoretical exercises, which were not addressed in this research. While hands-on exercises come in various forms, the classification made by M. Yamin et al. [15] is noteworthy in the educational context, dividing them into: *Jeopardy Style CTF*, *Attack and Defense* and *Red & Blue*. Among this three categories, we found *CTF* (Capture the Flag) exercises to be the most suitable technique for evaluating students' skill sets, considering the introductory nature of the course.

CTF scenarios, as the name suggests, consist of small sub-scenarios across different categories such as cryptography, steganography and web exploitation.

Upon completion, a string know as *flag* is yielded, which the participants can later use for scoring.

2.3. Statistics and evaluation

While the design and the format of the cyber scenario are key elements of its definition, the collected statistics and the post-execution evaluation of the participants are equally important and hence, central aspects of this paper.

In [16], the term *Learner Analytics Stack* is defined as mechanism of processing events generated by students throughout their interaction with the learning environment. Moreover, they provide an example of an event definition captured by *Syslog* and integrated in the *ELK*[4] stack. ELK monitoring is also approached in [17].

The idea of representing data captured from student activity in the form of binary trees has its roots in [18]. Here, the actions are modeled using oriented graphs from the beginning up until the end of the assessed activity. The paper also showcases the idea of a reference graph, intended to serve as a potential solution to the scenario the students are working on. The start vertex of this graph represents the initial state and the end vertex represents goal of the exercise. This idea was also taken into account in our implementation.

3. Outlining the Scenario

This section is dedicated to describing the proposed scenario in accordance with the COFELET framework.

3.1. Course Background

As previously stated, the scenario was delivered in the form of a CTF contest, with the purpose of assessing the knowledge students gained from *Cybersecurity Basics* subject at *National University of Science and Technology POLITEHNICA Bucharest*. The subject was designed to be taught over a semester, covering introductory aspects of cybersecurity such as: forensics, application security, web security, access control and related topics. Two cohorts of students were involved in this research during the 2023-2024 academic year: one cohort from the third year and another from the fourth year, both having studied the same subject.

After each lecture, students completed hands-on lab sessions using pre-configured virtual machines on the university's OpenStack platform. They were graded on their progress and also completed two homework projects during the semester that expanded on the lab exercises.

[4] Elasticsearch, Logstash and Kibana

3.2. Scenario Environment

The overall design of the scenario was made regarding the COFELET framework as mentioned in **2.1**. We will proceed presenting the environment with an eye to the framework elements.

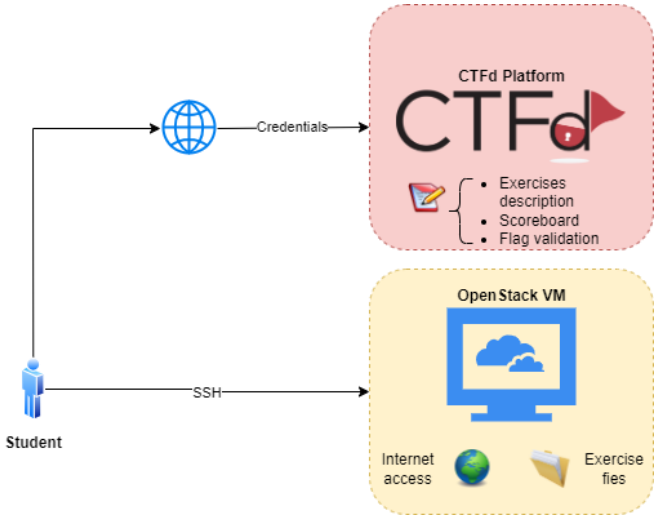

Fig. 1. Gaming context

3.2.1. *Goal*. The goal of the scenario is to simply extract and validate as much flags as one possible can before the allocated time expires.

The goal of an exercise is to solve it and extract the flag. The flag consists of a string with a format presented before the scenario starts. In our case, the flag had the following format: ISC {fl@g_ex@mpl3}.

3.2.2. *Task*. The tasks within the scenario are represented by exercises that must be completed to extract the flag. Each exercise includes its own set of tasks necessary to achieve the final goal.

3.2.3. *Gaming Context*. The gaming context consists of the conditions and properties of the scenario. The exam was conducted in the format of a CTF exercise as mentioned in **2.2.** Each student had access to a dedicated VM hosted on the university's OpenStack platform, which included all the exercise files and tools required for solving the tasks. Moreover, the VM had Internet access. We configured the VM and replicated it in advance.

After successfully extracting a flag from an exercise, the student would then submit it in a dedicated platform, hosted using the *CTFd* solution. On this platform,

students could view all the challenges and their associated difficulty, grouped by category. When a flag was submitted, the platform would validate or invalidate it. The gaming context schema is depicted in Fig. **1**.

The scenario had a time frame of four hours. After this period, the platform would reject all flag submissions. To prevent any form of fraud, the flag found in each exercise was randomized using the ID of each student and the files required regenerated before the start of the scenario.

3.2.4. *Scenario Execution Flows*. SEFs represent the order in which tasks have to be completed in order to reach the end goal. For the overall scenario, there is no SEF, as the student can tackle any exercise. On the exercise level, SEF is represented by a set of sub-tasks that need to be executed in a certain order in order to reveal the flag. For example, if the exercise flag is hidden in *base64* format inside the metadata of an image, one possible SEF would be:

Identify image type→ *Use exiftool to extract metadata*→*Observe the base64 flag in a metadata field*→*Decode base64 flag*.

3.2.5. *Scenario complexity*. The scenario was designed to be mid-level. As mentioned in section **3.2.3**, each exercise had an associated difficulty set in the CTFd platform, as determined by the assistant who designed it.

4. Monitoring student activity

This section discusses the exercises provided to each student cohort and the methods used to extract statistics from their real-time performance.

4.1. The exercises

Each generation of students were presented with a set of exercises, grouped by different categories and complexities. The approach used to design the challenges was not based on a specific point of reference but was instead subjective to the tutor's experience and creativity. The first generation of students, for which the subject was taught in the third year of university, was presented with the following group of challenges:

- *3pass*: binary exploitation challenge in which the students had to inspect the code of a provided executable using a debugger such as *gdb* in order to extract the required input. After the input was provided to the executable in the required order, the challenge would yield the flag;
- *chrono_crypto*: cryptography challenge that required the students to reverse an encryption algorithm that had an *UNIX* timestamp as key. When the required timestamp was used for decryption, the flag was revealed;

- *hiddenports*: there were a number of 3 hidden open ports located on the machine. The flag was a string comprised of the port numbers concatenated. The challenge category was miscellaneous;
- *packetbeat*: traffic analysis/forensics challenge based on DNS exfiltration;
- *privesc*: web exploitation challenge that required the students to gain administrative privileges on a *Docker* container running on the VM;
- *cuphead*: steganography/forensics challenge that had the flag hidden in the image metadata;
- *binbufff*: binary exploitation challenge that required the students to overflow a buffer in order to obtain the flag, using a provided executable;
- *dirb*: web exploitation challenge that required the students to use *gobuster* or *dirb* on a web application hosted in a Docker container running on the VM in order to locate the flag;
- *let_me_in*: web exploitation challenge consisting of a web application vulnerable to SQLInjection;
- *osint*: *Open-source intelligence* (OSINT) challenge that involved searching on a given domain for relevant information;
- *buried_treasures*: miscellaneous challenge that required students to extract a big number of nested archives using scripting. The initial archive was embedded in an image.

The second cohort of students, who studied the subject in the fourth year of university, was presented with a similar set of challenges:

- *cuphead* was renamed to *stegano_exif*;
- *buried_treasures* was renamed to *s3rp3nt_p1r4t3s*;
- *peanut_butter_jelly*: additional cryptography challenge involving *GPG*[5] decryption.

4.2. The trackers

Before exploring possible methods of monitoring student activity during the exam, we first needed to determine which indicators we were targeting. This becomes fairly straightforward when considering how the scenario was designed in section **3**. Solving each challenge, or reaching the end *goal*, becomes a matter of two factors: the *gaming context*, in this case the exercise files, and the *SEF*s, or the paths a student has attempted in the solving phase. To reference this indicators, we concluded to extract the following pieces of information, labeled as P_i, where i is the number of the item:

P_1: When the student began working on a challenge;

[5] GNU Privacy Guard

P_2: Which challenge is the student addressing at a given time;
P_3: When the student stopped working on a challenge and started focusing on another;
P_4: When the student finished a challenge;
P_5: The live list of commands executed by a student;
P_6: What files were created that assisted in solving the challenge;

We concluded to use scripts, or *trackers* to facilitate the monitoring process. One such tracker consists of a *Python* script, that is, at its core, a customized keylogger, with the following workflow:

(1) Create the tracker log file and write the starting timestamp;
(2) Loop over the *bash history file* to check for recently added commands;
(3) Whenever a command is added perform the following:
    (a) Check if the command is a *cd* (change directory). If so, check if the destination directory is corresponding to the currently tracked challenge (e.g. if the challenge name is *peanut_butter_jelly*, look for *cd peanut_butter_jelly*). If that is the case, begin recording all the commands from now on (tracker is set to record). If the cd is made to another directory, and the tracker already records commands, assume that the student began working on another challenge and stop tracking until he *cd*'s back into the exercise folder (set record flag to false);
    (b) Else, if the tracker is set to record and the command is not a cd, compare it to a predefined set of commands relative to the challenge objective. Should the command be in the given set, log the timestamp, the command and its arguments.
(4) Retain the last bash history line number for the next iteration;
(5) Check if there is a **flag.txt** file inside the challenge directory and whether or not it contains a valid flag. Stop the tracking, mark challenge as finished and copy all the files that might have assisted in the solving of the challenge, if true.

The flow diagram of the tracker is illustrated in Fig. **2**.
The tracker log file contains a list of commands formatted as follows:

[Timestamp] [Command description] [Command arguments] [Command tag],

where:

(1) *timestamp* represents the time of the issued command;

(2) *command description* consists of a short description of the issued command. For example, if the command was *nano file.txt*, the description would be "Created/modified flag.txt";
(3) *command arguments* are represented by the arguments passed to the issued command. For example. if the command was *binwalk -e image.png*, the arguments would be "-e" and "image.png";
(4) *command tag* is represented by a short indicative of the issued command. Examples include: *FILE_CREATION*, *SCRIPT_EXECUTION* and the like.

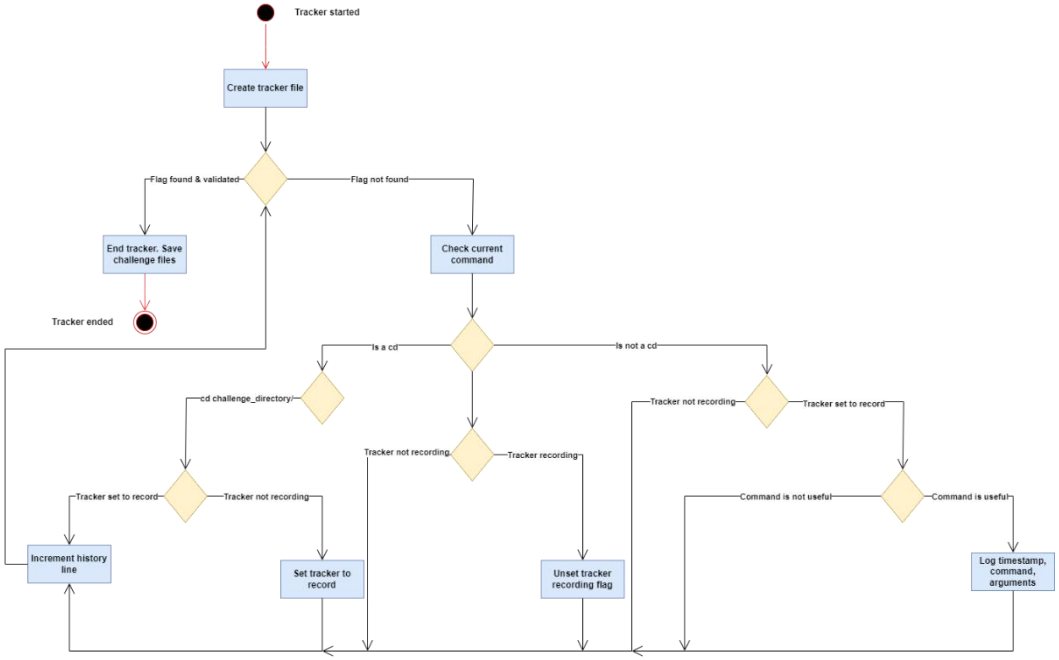

Fig. 2 Tracker flow diagram

First generation of students had the following challenges tracked:

- chrono_crypto;
- buried_treasures;
- let_me_in.

The second generation of students had the following challenges tracked:

- chrono_crypto;
- s3rp3nt_p1r4t3s;
- let_me_in;
- 3pass;
- binbufff;

- stegano_exif;
- peanut_butter_jelly.

4.3. Requirements

For the trackers to function as intended, the following requirements had to be met:

(1) The Bash history file needed to be updated in real time, immediately after each command was issued by the user, since the default behavior is to append to the history file only after the session ends. For this to occur, the following lines were added to *bashrc* file, according to [19]:

shopt -s histappend
PROMPT_COMMAND="history -a; $PROMPT_COMMAND"

(2) All progress for a challenge, including the scripts used, had to be made within the challenge directory;

(3) To stop the tracker, students were required to place the flag into a "flag.txt" file in the challenge directory after completing the challenge. This action would mark the challenge as completed and trigger the extraction of the associated files.

(5)

5. Decision Binary Trees

The main purpose of this section is to address the process of constructing decision binary trees, starting with an examination of how the raw student data was preprocessed in subsection **5.1**. Subsection **5.2** provides a detailed description of the tree creation algorithm. Finally, subsection **5.3** explains how the final product can be visualised on a time axis.

5.1. Scrapping the data

After the examination session was concluded, all the directories corresponding to each of the challenges were extracted from all of the deployed VMs. This resulted in 175 sets of directories for the first generation of students and 194 for the second generation. This numbers also included the VMs that were created but not used (primarly because some students chose not to participate in the practical examination), resulting in empty challenge directories. The data was automatically extracted by establishing *SSH* sessions to the virtual machines.

5.2. Building the trees

In this paper, we model student actions using binary trees, where each leaf has two children representing the sets of good or bad decisions at a possible step in the challenge-solving progress. The criteria for classifying a decision as either good or bad are based upon a set of commands predefined by the tutors, which represents one possible solution to the challenge objective. This idea has its roots in [18] as mentioned in **2.3**.

Each possible solution used as reference for evaluating a student's progress in solving a challenge follows a standard layout, written in *JSON* format as a series of ordered steps.

```
{
    "step1_name": {
        "description": "Step description",
        "optional": true/false,
        "possible_commands": {
            "command1_identifier": {
                "description": "Command description",
                "command_arguments": ["name", "arg1","arg2", "etc"],
                "requires_repetition": true/false
            },
            "command2_identifier": {
                "description": "Command description",
                "command_arguments": ["name", "arg1","arg2", "etc"],
                "requires_repetition": true/false
            },
            "so_on_and_so_forth": {
            }
        }
    },
    "step2_name": {
        "description": "So on and so forth"
    }
}
```

A step is the current logical approach in the challenge solving procedure and is defined by a series of commands that can lead the path to the next step. A step can be optional, meaning that its occurrence in the solving process at any point would not be considered a bad decision. Therefore, the order in which the non-optional steps are defined matters. The commands comprised in the definition of a step have the same effect and are often variations of the

same command. Thus, progressing to the next step requires executing only one command from the list. A command might also require repetition (for example when extracting a group of archives).

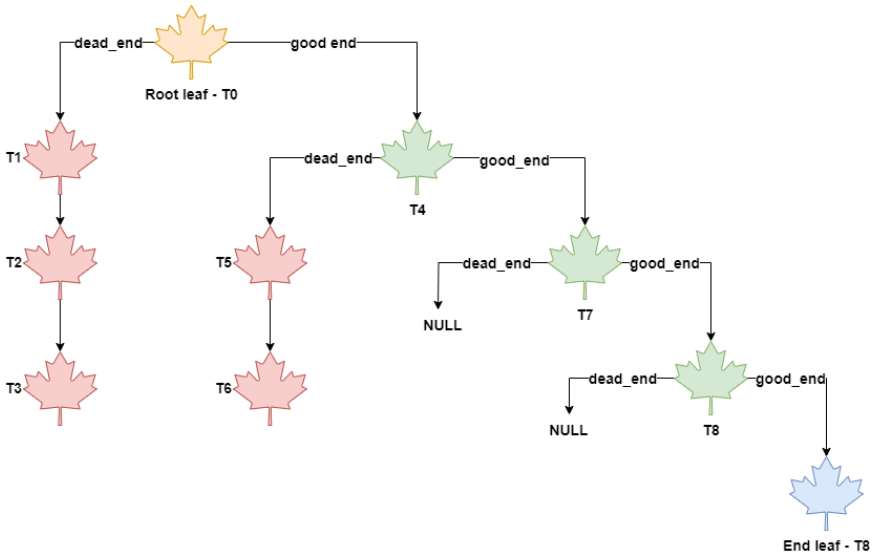

Fig. 3 Decision tree concept

The decision tree is also a JSON dictionary, build by iterating the *tracker log* of a given challenge, with a format constructed using the following rules:

(1) Each leaf is represented by an issued command. The attributes of a leaf are the following: *description, arguments*, *timestamp*, *good_end*, *dead_end*;
(2) Root leaf is represented by the "Start of task" tag, with no arguments set, the timestamp corresponding to the time a student began working on the challenge;
(3) dead_end (JSON array) consists of the set of commands that were not found in the solution used as reference; A leaf found in the dead_end has its good_end and bad_end lists set to null;
(4) Only leaves found in the good_end (JSON array) list can have both decision lists set;
(5) When a command that corresponds to the next step in the possible solution is found, backtracking is performed to the last leaf classified as "good end". The command is then added as a new leaf in the good_end array of that node;
(6) If the task was completed, the last leaf is represented by the "Student completed the task" tag, with an associated timestamp.

The concept of a decision tree can be seen in Fig. **3**.

5.3. Visual representation

Constructing the binary trees as JSON arrays facilitates their visual representation. Each leaf has an associated timestamp, as mentioned in **5.2**. This approach allows the decision trees to be plotted as timelines of commands issued during the challenge-solving process, thereby enhancing the post-analysis of student activity.

The plotting was done by extending the idea found in [20] }, where each command is represented as a point on a horizontal time axis, along with its associated timestamp. An example of one such tree plot made for an successful attempt on *chrono_crypto* challenge can be found in Fig. **4**.

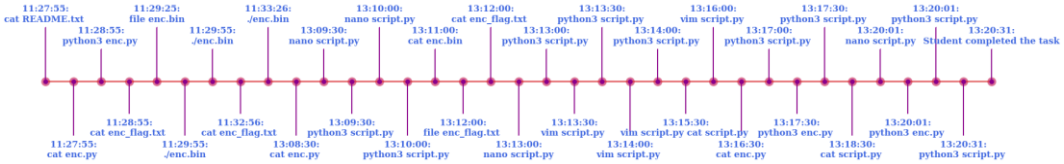

Fig. 4 Decision tree plot example

6. Comments

This section of the paper is dedicated to discussing the evaluation of the showcased implementation, including its shortcomings, potential improvements, and the desired features that remain to be accomplished.

6.1. Evaluation

The overall purpose of modelling student activity as decision binary trees was to facilitate the understating of the various approaches taken by the participants in the Cybersecurity Basics exam at University POLITEHNICA of Bucharest. Moreover, it also assisted in assessing the level of difficulty each challenge imposed from students' perspective. After all the data had been scrapped and the decision trees modelled, we began sorting the statistics and retained only the relevant material. The statistics were considered relevant based on the following criteria:

- The challenge tracker was initialized (the VM was used in the exam by one of the students);
- The challenge was attempted by the student (the tracker log file was not empty);

- The challenge was not completed in a different environment other than the provided VM (in that case, the log would contain only the "Student completed the task" tag, with no other commands).

There was also a second sort made, which targeted the data for the challenges that were completed.

6.1.1. *First generation - 3rd year students*. This generation studied the subject in the 3rd year of university, and thus had limited knowledge in using scripting languages such as *Python* or *Bash*, as well as limited familiarity with networking and databases concepts. Although these shortcomings were addressed over the course of the semester, through laboratories and associated homework, students still required assistance with more difficult concepts, such as exploiting a SQLInjection vulnerability without guidance. The amount of solves per challenge can be observed in Fig. **5**.

After analyzing all the decision trees, we have reached the following conclusions regarding the challenges:

- *buried_treasures*: This challenge had the most solves. Although it required the largest amount of scripting among all the challenges, as it comprised a considerable number of archive extraction operations, it could also be approached in a straightforward manner by extracting the archives manually. Most students opted for the latter approach, solving the challenge in a significantly larger time frame. According to the analysis conducted using binary trees, most of the time was spent gathering information about the archive and performing the extraction process, which are common time-consuming steps. This suggests that students were not stuck and had a clear approach to solving such challenges. The challenge lived up to its "medium" difficulty tag;
- *chrono_crypto*: The small number of solves to this challenge can be attributed to the fact that it could not be solved without scripting, as it required the reversing of an encryption algorithm. Binary tree analysis of the solving process showed that students who failed to complete the task either remained stuck inspecting challenge scripts and files or did not attempt the challenge at all, indicating a lack of familiarity with a scripting-based approach. This led us to consider introducing a scripting-based introductory laboratory at the start of each semester. Labeled as "hard," the challenge retained its difficulty;
- *let_me_in*: Solving this challenge was an easy task either by approaching it manually or by using *sqlmap*. The small number of solves was surprising, given that sqlmap was included in one of the homework exercises assigned during the semester. The decision trees for both solves had a very large

number of leaves, indicating that sqlmap was used in each of the cases, generating a large number of commands. Despite being classified as medium by the organizers, this challenge turned out to be in the hard category.

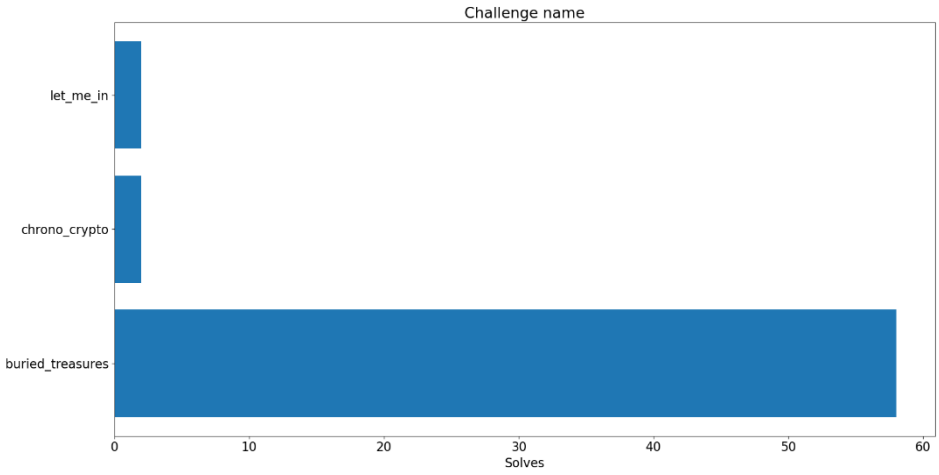

Fig. 5 Solves per challenge - 3rd year students

This edition of the exam has led us to the following improvements:

- Students should be presented with a Python introductory course before undertaking Cybersecurity basics;
- The lab contents should be more detailed with more hints and guidance along the way;
- The homework solves should be more strictly checked.

6.1.2. *Second generation - 4th year students*. This generation studied the subject in their 4th year of university and had significantly more experience regarding cybersecurity and scripting concepts, with a vast majority of the participants being part-time employees in the IT sector. There was also a larger number of tracked challenges, allowing for a more comprehensive view of the solving process. The amount of solves per challenge can be observed in Fig. **6**.

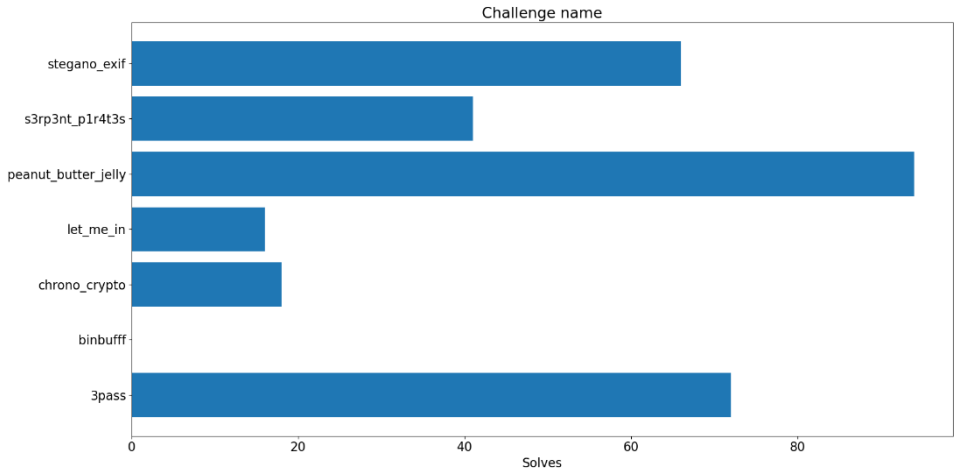

Fig. 6 Solves per challenge - 4th year students

After analyzing all the decision trees, we have reached the following conclusions regarding the challenges:

- *stegano_exif*: This challenge was by far the easiest. The main reason for which it did not have the most solves is that the students were unaware of any tool that could extract EXIF data out of an image. Decision tree analysis showed that students understood how to approach a steganography challenge, as most commands focused on gathering file information or searching for a specific flag pattern. The challenge maintained its "easy" difficulty rating;
- *s3rp3nt_p1r4t3s*: Also known as *buried_treasures*, the challenge had less solves in this edition of the exam. One reason for this is that the challenge directory contained many hidden hints that indicated towards a scripting approach. Moreover, the number of nested archives to be extracted was exponentially larger, meaning a manual approach was not feasible. The thought process observed in the binary tree plots was specific to solving a task through scripting: a concise set of commands focused on file information gathering, followed by a sequence of *file access* and *file run* commands, reflecting the trial-and-error approach applied to script samples. Medium remained the proper difficulty;
- *peanut_butter_jelly*: This challenge had the most solves and was relatively straightforward. "Easy" was the correct difficulty rating for this challenge.
- *let_me_in*: Far more solves in this generation. Very few manual approaches, almost all solves being done with sqlmap, observation confirmed by the decision tree plotting. The difficulty for this exercise remained medium;

- *chrono_crypto*: Once again, it was clear that students were more confident in their use of scripting during the challenge-solving process. Comparing this with the results from the previous generation, it is evident that students typically become familiar with scripting in the second semester of their third year. This highlights the need for a scripting laboratory at the beginning of the semester to bridge the gap in scripting knowledge. The difficulty could be adjusted from "hard" to "medium";
- *binbufff*: This challenge had no solves (at least on the deployed VMs). This is primarily because most students chose to solve it externally, using tools with graphical interface (for example *Ghidra*). Initially labeled as "medium," it was later categorized as "hard."
- *3pass*: Second most solved challenge. It was approachable using any tool used for machine-code inspection. Labeled as "medium," it turned out to have an easy difficulty.

This edition of the exam has led us to the following improvements:
- Binary exploitation laboratories should have more explanations and hands-on examples;
- Hints provided along with challenge descriptions should be as clear as possible;

6.2. Shortcomings and improvements

There were a number of shortcomings in the showcased implementation:
(1) If a student manages to solve a task using a different approach other than the one provided by the organizers, the task would still be considered completed, but all (or some) the commands would be dead end leaves. To overcome this, the algorithm should create another solution file for each solve that is considered a dead end, but for which the task is labeled as finished. The subsequent solves would be compared to both the teacher's solution and the student's newly discovered solution;
(2) If a student is working on a task but changes the working directory to a sub-folder within the challenge directory, the tracker would interpret this as the student having stopped working on that specific task. One solution to this would be to use *ptrace* to track the current Bash process and the forks it makes when executes a command, then attach to the process issued by that exact command and observe its arguments, including the working directory. Several attempts were made to integrate this into the tracker behaviour, but it proved to be resource intensive due to the large number of commands and concurrently running trackers;

(3) If a tracker log contains a large number of commands (especially when the challenge was approached using an automated tool), the visual representation of the tasks timeline becomes heavily clustered and hence, unreadable. To overcome this, image resolution could be enlarged. The decision tree analysis can be performed regardless of the number of issued commands, as it is represented in JSON format;
(4) To mark a challenge as finished and stop the tracker, a *flag.txt* file containing the flag must be created by the student in the challenge directory. Oftentimes, due to the exam context, this detail gets overlooked. To address this issue, a correlation between the scoring platform and the student VM can be established, allowing the tracker to be stopped when a flag submission is validated;

7. Conclusions

Our approach for integrating live student activity data using binary trees has proven to be an efficient approach, improving both data visualization and the accuracy of post-exercise analysis. The binary trees, constructed by comparing each student's set of commands during the exam with a pre-defined solution set created by the tutors, aided in understanding the thought process of solving a challenge. By analyzing the flow of the solution, we were able to pinpoint which steps took the most time and determine whether a participant had fully assimilated the necessary knowledge related to the challenge topic. Moreover, this analysis allowed us to adjust the difficulty level of each challenge for future iterations of the scenario (such as the exam held in the following semester) and refine the laboratory materials accordingly.

REFERENCES

[1] "The cybersecurity industry has an urgent talent shortage. Here's how to plug the gap," 28 April 2024. [Online]. Available: https://www.weforum.org/stories/2024/04/cybersecurity-industry-talent-shortage-new-report/. [Accessed 5 May 2024].

[2] "Cybercrime Expected To Skyrocket in Coming Years," 22 February 2024. [Online]. Available: https://www.statista.com/chart/28878/expected-cost-of-cybercrime-until-2027/. [Accessed 9 May 2024].

[3] "Global gross domestic product (GDP) at current prices from 1985 to 2029," October 2024. [Online]. Available: https://www.statista.com/statistics/268750/global-gross-domestic-product-gdp/. [Accessed 9 May 2024].

[4] "Commission to invest over €210 million in cybersecurity, digital capacities and technology under the Digital Europe Programme," 4 July 2024. [Online]. Available: https://digital-strategy.ec.europa.eu/en/news/commission-invest-over-eu210-million-cybersecurity-digital-capacities-and-technology-under-digital. [Accessed 9 May 2024].

[5] "Initial Stages of Implementation of the National Cyber Workforce and Education Strategy," Office of the National Cyber Director, 2024.

[6] "Cybersecurity Workforce Demand," National Institute of Science and Technology, 2023.

[7] M. N. Katsantonis, A. Manikas, I. Mavridis and D. Gritzalis, "Cyber range design framework for cyber security education and training," International Journal of Information Security, vol. 22, pp. 1005-1027, 2023.

[8] L. Amro, Gamifying the MITRE ATT&CK for Cyber Security Training using the COFELET Framework, Norwegian University of Science and Technology, 2022.

[9] Y. Deng, Z. Zeng, K. Jha and D. Huang, "Problem-Based Cybersecurity Lab with Knowledge Graph as Guidance," Journal of Artificial Intelligence and Technology, vol. 2, pp. 55-61, 2022.

[10] M. Glas, M. Vielberth and G. Pernul, "Train as you Fight: Evaluating Authentic Cybersecurity Training in Cyber Ranges," in Conference on Human Factors in Computing Systems, Hamburg, 2023.

[11] J. Whyte, G. Dagher and S. Hagenah, "BEACON Labs: Designing Hands-on Lab Modules with Adversarial Thinking for Cybersecurity Education, Journal of The Colloquium for Information Systems Security Education," Journal of The Colloquium for Information Systems Security Education, vol. 10, 2023.

[12] N. M. Katsantonis, I. Kotini, P. Fouliras and I. Mavridis, "Conceptual Framework for Developing Cyber Security Serious Games," IEEE Global Engineering Education Conference (EDUCON), pp. 872-881, 2019.

[13] M. Shivapurkar, S. Bhatia and I. Ahmed, "Problem-based Learning for Cybersecurity Education," Journal of The Colloquium for Information Systems Security Education, vol. 7, 2020.

[14] M. Yamin, E. Hashmi, M. Ullah and B. Katt, "Applications of LLMs for Generating Cyber Security Exercise Scenarios," Internation Journal of Information Security, 2024.

[15] M. Yamin and B. Katt, "Modeling and executing cyber security exercise scenarios in cyber ranges," Computers & Security, vol. 116, 2022.

[16] J. Vykopal, P. Čeleda, P. Seda, V. Švábenský and D. Tovarňák, "Scalable Learning Environments for Teaching Cybersecurity Hands-on," IEEE Frontiers in Education Conference, vol. 9, pp. 1-9, 2021.

[17] W. Lazrov, T. Stodulka, T. Schafeitel-Tähtinen, M. Helenius and Z. Martinasek, "Interactive Environment for Effective Cybersecurity Teaching and Learning," in International Conference on Availability, Reliability and Security, 2023.

[18] M. Andreolini, V. G. Colacino and M. C. e. al, "A Framework for the Evaluation of Trainee Performance in Cyber Range Exercises," Mobile Networks and Applications, vol. 25, p. 236–247, 2020.

[19] "Update bash history in real time," [Online]. Available: https://askubuntu.com/a/67306. [Accessed 1 May 2023].

[20] "Making timelines with Python," 21 August 2021. [Online]. Available: https://dadoverflow.com/2021/08/17/making-timelines-with-python/. [Accessed 15 February 2023].

[21] A. Alexandrescu, "Optimization and security in information retrieval, extraction, processing, and presentation on a cloud platform," Information 10.6, vol. 6, 2019.

[22] A. Alexandrescu and G. Butnaru, "An architecture of identity management and thirdparty integration for online teaching in a university," ICSTCC, pp. 850-855, 2020.

[23] R. Rughiniș, "Badge architectures in engineering education," CSEDU, no. 978-989-8565-53-2, pp. 548-554, 2013.

[24] A. R. Căciulescu, R. V. Rughiniș, D. Tsurcanu and A. Radovici, "Mapping Cyber-Financial Risk Profiles: Implications for European Cybersecurity and Financial Literacy," Risks, 2024.

[25] D. Rosner, D. Iorga, F. Oprea, C. Pătru and R. Rughiniș, "A conceptual framework for profiling engagement strategies used in high school engineering outreach activities," European Journal of Engineering Education, pp. 1269-1290, 2023.