Noname manuscript No.
(will be inserted by the editor)

A framework for the evaluation of trainee performance in cyber exercises

Mauro Andreolini · Vincenzo Giuseppe Colacino · Michele Colajanni · Mirco Marchetti

the date of receipt and acceptance should be inserted later

Abstract This paper proposes a novel approach for the evaluation of the performance achieved by trainees involved in cyber security exercises implemented through modern cyber ranges. Our main contributions include: the definition of a distributed monitoring architecture for gathering relevant information about trainees activities; an algorithm for modeling the trainee activities using directed graphs; novel scoring algorithms, based on graph operations, that evaluate different aspects (speed, precision) of a trainee during an exercise. With respect to previous work, our proposal allows to measure exactly how fast a user is progressing towards an objective and where he does wrong. We highlight that this is currently not possible in the most popular cyber ranges.

Keywords Cyber range · Cyber exercise · graph algorithms · Monitoring framework

1 Introduction

Current products and services are almost exclusively made available through Internet-based systems that allow for a wide reach through high performance and scalability [12], and ubiquitous fruition through a wide range of devices (smartphones, desktops, laptops) [9]. Security of the underlying subsystems and protocols has become of uttermost importance to managers, designers, programmers, administrators and end users, since a breach might have unforeseen and critical consequences, leading often to business disruption [7]. This growing attention to cyber security clearly arises from external threats, which mainly exploit outdated, vulnerable systems and weak security controls. Moreover, inexperienced and insufficiently trained staff often constitutes the greatest hindrance to security in the area of computer crimes. Previous studies [11,5,1] show that most cyber security incidents result from human errors, misinterpretation of system policies and posture. Experience shows that security is best learned by practice on existing systems. However, on the one hand practicing on production systems is risky at best, and on the other hand, building a testing replica of a large enterprise network might prove too much of a burden for an already overworked staff.

To increase the resilience of their infrastructures, both military and civilian organizations have started to train security personnel on *cyber ranges* that is, prearranged virtual environments connectable via a VPN through which it is possible to effectively simulate realistic attacking scenarios (nation-state red teams, cyber criminals, hacktivists) on a system architecture closely resembling the original one. Training goals are many and diverse in nature: to discover vulnerabilities in existing systems, to harden existing systems, to evaluate the security of a soon-to-be deployed component, to teach secure programming practices, to perform incident response on a compromised system. The most popular cyber ranges [4,8,14,6] offer a prearranged, realistic environment on which to perform blue team vs. red team exercises. Here, red teams are external entities brought in to test the effectiveness of a security program. They are hired to emulate the behavior and techniques of likely attackers to make it as realistic as possible. On the other side lies the blue team, the internal security team that is charged with stopping these simulated attacks. A growing number of companies, however, are not using formal blue teams in their exercises. The idea is to get a more realistic evaluation

University of Modena and Reggio Emilia, Italy
{mauro.andreolini, vincenzogiuseppe.colacino, michele.colajanni, mirco.marchetti}@unimore.it

of their true defensive capabilities by seeing how their security teams react to the simulation without formal preparation. The ultimate goal of red vs. blue team engagements is to test organization security maturity as well as its ability to detect and respond to an attack. Red team goals, which may be diverse in nature, all testify the ability of an attacker to violate confidentiality, integrity or availability of a vulnerable system. Typical objectives are service disruption, exfiltration of sensitive data, stealthy movement across systems. The blue team tries its best to thwart these attacks and attribute them to the red team.

Besides the virtual environment, cyber ranges also typically provide support for "cyber awareness" in the form of a dashboard that constantly displays how many of the stated goals have been fulfilled by the red team. The dashboard provides a rough measure of how well the teams are behaving. While useful, this approach has a severe limitation: being focused exclusively on goal achievement, it cannot measure the performance of a single player and, more generally, the reasons behind success or failure. For example, let's consider a single red team player who has successfully escalated privileges to those of an administrator. Was this the result of a perfect attack, achieved in the quickiest and stealthiest fashion? Or was this the result of repeated trial and error (increasing the risk of being detected)? Or, even worse, was the result due to pure chance (random guessing, exploiting traces left by a previous player)? The dashboard, lacking any information about user activity, cannot give any useful feedback in this respect, and thus cannot make a real distinction between more or less competent players. However, we think that, besides reaching the desired training goals in a single engagement, an instructor should monitor the performance of every student on multiple engagements, pointing out improvement, stall, regression and tuning the learning material accordingly. It is worth noting that is no universal model capturing all learning objectives. Lab exams related to introductory courses should evaluate the ability of a student to reproduce basic techniques precisely. Penetration testers should be evaluated on the ability to find all known vulnerabilities in a system and exploit them systematically. Red teamers should display quickness and stealthiness in penetrating systems. Vulnerability researchers, on the other hand, should prove to be able to find not previously known paths of exploitation. In the end, different learning objectives call for different models and scores.

A *trainee* is anyone who wants/needs to improve his/her skills in one or more cyber security fields (attack, defense, source code auditing, cryptography) and is currently subject to evaluation. An *exercise* is a challenge (deployed on an hw/sw infrastructure called *lab network*) through which an instructor evaluates a trainee. A *learning objective* is a specific ability that a trainee will nurture through an exercise. In this paper we introduce the following contributions.

1. We propose a novel, modular, extensible framework that supports collection of all pieces of information needed to model the activity of a *trainee* involved in an exercise.
2. We define a model of trainee activity through directed graphs, built from collected data. These models are at the basis of performance evaluation. Models can be built online (for live exercises) or offline (for post-mortem analysis).
3. Given specific learning objectives, we propose novel scores that measure the proficiency of a user in reaching them. A score is defined as a scalar function over a set of graphs. We provide experimental evidence that the ability of a trainee is often correlated to a well-chosen score.

We consider three distinct, realistic exercises of varying difficulty. Each exercise is carried out by three trainees with different levels of experience. The trainees are evaluated with four different scores aimed at measuring different learning objectives. We show how the graphs related to user activities pinpoint inefficiencies (repeated trial and errors, falling into rabbit holes, halting without pursuing the objective) and merits (follow strictly a single intended path, find all intended paths, discovering unintended paths). We also show how the score computed from the graphs can be used to effectively build a ranking amongst users.

The paper is organized as follows. Section 2 discusses related work in the fields of attack modeling and performance scores. Section 3 introduces oriented graph models to represent both an ideal conduct and actual behavior exhibited by a trainee during an exercise. Section 4 introduces several scores that capture different learning objectives. Section 5 briefly describes the architecture of the monitoring framework and some implementation details. Section 6 presents an experimental evaluation of scores. Finally, Section 7 concludes the paper with some final remarks.

2 Related work

The activities connected to a cyber attack are often achieved through a complex sequence of intermediate stages, in which a user gradually acquires privileges up to being able to carry out the required operations. In such circumstances, it can be very complicated to reconstruct the complete attack path and identify the con-

catenation of techniques and tools. An attack model is a formal representation aimed at describing an attacker's activities in terms of techniques used and vulnerabilities exploited in systems and configurations. The purpose of an attack model is to identify the most probable routes within this sequence and make them impractical, if not impossible.

An attack tree [16] represents the attacks to a system and related countermeasures as a tree structure. The root node is the ultimate goal of the attack. Intermediate nodes represent intermediate stages of an attack. Intermediate siblings can be combined in AND or OR mode; AND nodes represent the different steps in achieving a goal, while OR nodes represent different ways to achieve the same goal. Leaf nodes are attacks; they can be labelled to enrich the context of the attack. Typical labels are "possible/impossible", "legal/illegal", "cost", "probability of success". Label values of siblings combine and propagate to the parent node; for example, if reaching an intermediate stage requires two tasks with costs c_1 and c_2, the total cost is $c = c_1 + c_2$. Figure 1 shows an example attack tree, labeled by cost.

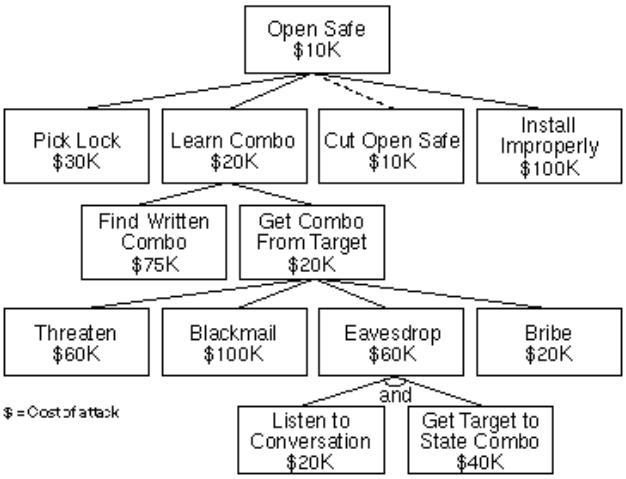

Fig. 1 An example attack tree labeled by cost

The notion of attack tree has been extended in literature. Kordy et al. [10] introduce attack-defense trees that also include possible counteractions of a defender. Since interactions between an attacker and a defender are modeled explicitly, this extended formalism allows for a more thorough and accurate security analysis compared to regular attack trees. Zonouz et al. [18] introduce the attack-response tree, basically an attack-defense tree that also includes intrusion detection uncertainties due to false positives and negatives in detecting successful intrusions.

Attack trees can quickly become complex as the number of vertexes and edges increases; in particular, it becomes increasingly expensive to identify all paths from a leaf node to the root node. Keep in mind that in realistic scenarios the number of nodes and interconnections can easily exceed thousands. In these conditions the addition of a single node is sufficient to cause a significant increase in the number of arcs, with a consequent increase in the new possible attack paths. Furthermore, since the root node represents the ultimate goal of the attack, it may be necessary to resort to multiple attack trees to model a complex multi-stage attack.

The attack graph [13] is another popular model of cyber attacks that is able to represent the infrastructure to be protected. An attack graph combines information related to network topology, eligible vulnerabilities and exploits available on the assets of an IT infrastructure, by providing a visual representation of the attack paths that an attacker must undertake to achieve specific objectives. An attack graph allows the analyst to highlight the structure of a network and to quickly identify the critical paths most subject to attacks. These activities are essential and preparatory to the subsequent phases of hardening and remediation.

An extension of the traditional attack graph is the Bayesian attack graph [15], which introduces the probabilities on the edges for modeling uncertainty in state transitions between nodes. In particular, edges include the probability of exploitation by an attacker. Therefore, the overall probability of reaching the last state is computed based on the combinations of these probabilities.

Although very useful, attack trees and graphs offer a static view of attacks and mitigations to a system; they do not model the actions an attacker actually carries out on a live system. Furthermore, research on scoring systems for cyber ranges seems to be currently in its infancy; current approachs are limited do signaling goal completion [3,17,2] rather than measuring trainee performance. To the best of our knowledge, this paper is a first step in these two directions.

3 Modeling trainee activities

In this paper, we model trainee activities through oriented graphs. Vertices represent intermediate states that are reached by a trainee during an exercise, while edges represent the actions performed by a trainee to move from a particular state to the next one. Both vertices and edges may be labelled with additional information (timestamps on vertices and edges to track progress

over time, or command options on edges to better discern useful from useless commands).

A *reference graph* is meant to model ideal behavior of a trainee during an exercise. It is prepared by an instructor and input into the framework. Every reference graph has a *start* vertex that models the beginning of trainee actions, and an *end* vertex that models the final goal of the exercise. These two nodes are not taken into account during actual calculations; they only serve to frame a sequence of actions. Figure 2 shows a very simple reference graph. Here, the trainee is supposed to reach states S_j (j=1, 2, 3) through simple actions (for example, shell commands) a_k (k=1, 2, 3) in that specific order. Model granularity may vary from

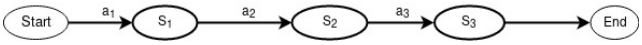

Fig. 2 Reference graph - fine granularity

single command outcomes to entire sub-goals per vertex, according to the learning objectives of the exercise. For example, introductory course exams evaluate strict adherence of trainees to standard attack procedures. The underlying reference graph must therefore be as detailed as possibly (ideally, at the level of single trainee commands and interactions). On the other hand, in red team/blue team exercises the attackers are usually free to choose their attacking strategy and are driven by goals rather than adherence to standard techniques. In this scenario, a reference graph should model intermediate mission goals rather than single command outcomes. Figure 3 shows such a reference graph, where the attacker has clear mission subgoals SG_i (i=1, 2, 3, 4, 5) carried out through operation campaigns o_k (k=1, 2, 3, 4, 5, 6, 7). Finally, vulnerability research activities might not even be represented by a clear reference graph, since the path is unknown a priori. In this case, the instructor might just build a reference graph out of a single vertex and a labelled edge indicating a possible event in case of successful exploitation (typically, command execution, information leak or service disruption). The framework provides an initial set of

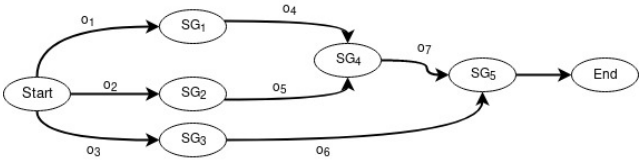

Fig. 3 Reference graph - coarse granularity

uniform and standardized events at different granularities that represent the most common actions (or even entire campaigns) performed by a trainee during an engagement. Edges in a graph are represented through these events. A reference graph might grow quickly in terms of vertices and edges as its granularity increases or as the lab network grows in terms of machines and reachable goals. To reduce graph complexity, several reference graphs can be defined (one for every intermediate goal).

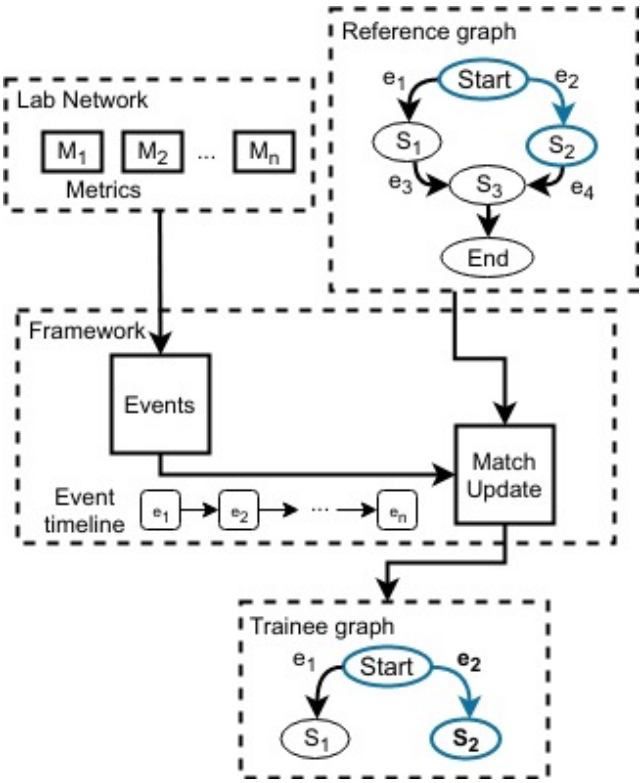

Fig. 4 Incremental update of the trainee graph

A *trainee graph* tracks the actions performed by a trainee during an exercise. It is built automatically by the framework from two elements: a set of reference graphs and a set of metrics collected on the game network. The latter ones capture trainee activity (command history, Web browsing history, GUI interaction), network events, OS process activity, service and application availability. More specifically, the framework initializes the trainee graph with a start vertex. Then, it collects all metrics relevant to a specific trainee on a specific engagement, maps them into pre-defined events and builds an event timeline. Finally, it tries to match the intermediate states of a reference graph with the events in the timeline. Whenever an event in the timeline matches an edge (V_i, V_j) in the reference graph, the trainee graph is updated as follows:

- vertex V_j is added to the trainee graph;

– vertex V_i is located in the trainee graph;
– an edge (V_i, V_j) is added to the trainee graph.

Figure 4 shows the incremental update of a trainee graph with vertex S_2 and edge e_2. Matching of timeline events with trainee actions is done in ordered fashion on all nodes of the reference graph.

If, on the other hand, a timeline event e cannot be matched against any edge in the reference graph, the trainee graph is updated as follows:

– the current vertex V_i in the trainee graph is located;
– the next expected vertex V_j in the reference graph is identified (such a vertex always exists, be it "start" if the trainee hasn't yet followed the recommended solution, or an intermediate one if the trainee has followed some steps of the exercise);
– the label N_j of the vertex V_j is identified;
– a new *dummy* vertex V_j is added to the trainee graph with label N_j_err (if it already exists, omit vertex insertion);
– an edge (V_i, V_j) is added with label set to e.

Figure 5 shows the insertion of a dummy node in a trainee graph with event e. Here, the current node is *Start* and not S_1, since the specific visit order of vertexes in the reference graph is *Start*, S_1, S_2, S_3, *End* and the framework is expecting completion of S_2. To

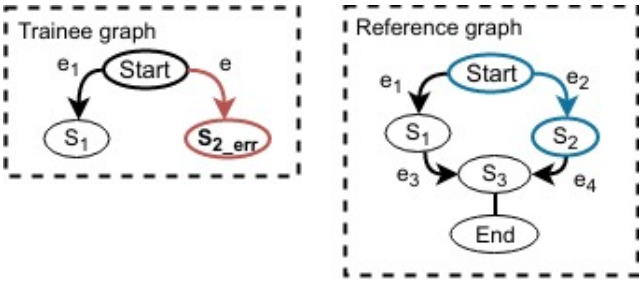

Fig. 5 Insertion of a dummy node in the trainee graph

prevent an excessive growth of edges in the trainee graph due to repeated, irrelevant trainee actions, the framework holds a per-engagement event black list that allows to filter them (for example, the UNIX `clear` command that clears the terminal display).

Figure 6 shows an excerpt trainee graph showing a correct evolution (using the `hydra` network login cracker to crack SSH credentials) and failed attempts (using the `telnet` and `zcat` commands). Here, events are detailed with metadata such as "event type" (UNIX command), "action" (the actual command given) and "status" (indicating whether the result allows to make progress or not). At the end of the matching process, a trainee graph reflecting the activities of a trainee during the

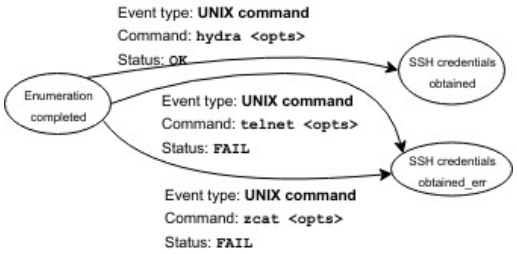

Fig. 6 Example of dummy node and correct node with event labels

whole exercise has been produced. Figure 7 shows (top) a reference graph related to a SQL database dump activity performed through the `sqlmap` command, and the correspondent trainee graph (bottom) with mistakes represented as dummy nodes. A trainee graph

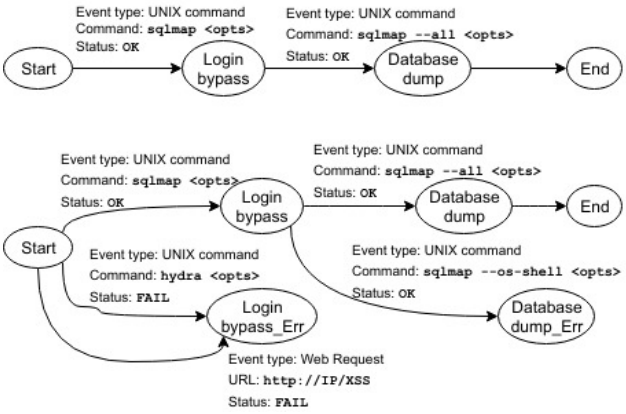

Fig. 7 Trainee and reference graph compared

may be computed offline (at the end of an exercise) for a post-mortem performance analysis. It may also be computed online (during an engagement) to track live trainee progress on a dashboard. In the latter case, the matching algorithm operates on a subset of events collected during a specific time window, and the starting vertex will be the first one recognizable in the event timeline.

It may happen that a trainee finds an *unintended path* that is, a sequence of actions that has not been considered by the instructor and allows to reach the end goal. In this case, the matching algorithm detects reaching the end vertex and operates backwards to reconstruct a sequence of events that includes a valid (yet unforeseen) solution. The following operations are carried out:

– the dummy vertex S_j_err in the trainee graph containing the latest trainee events is identified (this must exist, otherwise the trainee would be following an intended path);

– all events e_j pointing to S_j_err are removed from the trainee graph and ordered temporally;
– for each event e_j, a new *unintended vertex* U_{e_j} is created;
– the unintended vertexes are connected through the corresponding events in an ordered fashion.

This approach does not identify the exact sequence of actions leading to an unintended path; the sequence will also include failed attempts, which the framework cannot distinguish from correct ones. However, it serves as a basis to pinpoint alternative solutions which could be analyzed by an instructor and aid in building more precise reference graphs, rather than computing actual scores.

Figure 8 illustrates a trainee graph after the reconstruction of an unintended path involving events e_x and e_y. The dashed edges have been removed and added back as links to the unintended vertexes U_{e_x} and U_{e_y}.

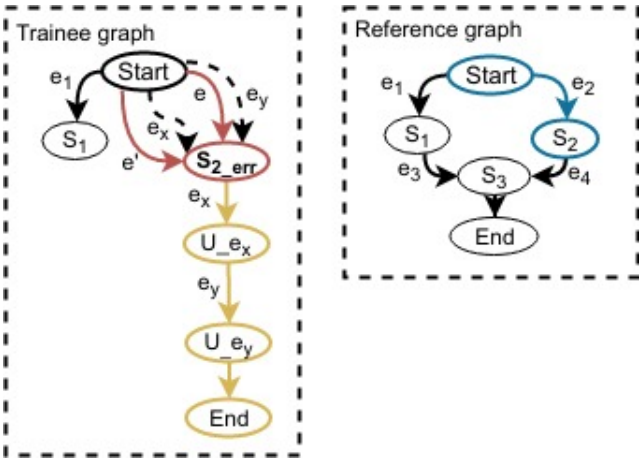

Fig. 8 Reconstruction of an unintended path in a trainee graph

4 Scoring algorithms

A *score* is a function $f : G^n \to \mathbb{R}$ that takes as input a set of n oriented graphs (including at least one trainee and a reference graph), and outputs a real number s in a specified interval $[a, b]$. In this paper, we consider the $[0, 1]$ interval, but any other one can be chosen. No single score is capable of capturing every single ability learned in cyber exercises; different learning objectives call for different scores. In this section, we introduce some scores to measure the progress of specific abilities such as:

– reproduce basic attack/defense techniques precisely to exploit/remediate all known vulnerabilities in a system according to a defined plan;
– penetrate a system quickly and stealthily (both in operating steps and time);
– discover unknown vulnerabilities in a system in the least amount of operative steps.

4.1 Basic scores

Score s_1. Let G_u be a trainee graph and G_r a reference graph. Let l be the length of the shortest path from the start vertex to the end vertex. We have:

$$s_1(G_u, G_r) = \begin{cases} \frac{1}{l} & \text{if the end node is reached} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

This score rewards trainees who are able to reach the final goal of an exercise in as few operative steps as possible. If no paths are present, a null score is given. This is a good candidate for vulnerability research activities, where trainees are asked to find unknown vulnerabilities and reproduce them in the conceptually fastest possible way. The score is very lightweight, being computable in $(O(|E| + |V|log|V|)$ steps $(G = (V, E))$ through Dijkstra's shortest path algorithm. Unfortunately, s_1 suffers two drawbacks. First, it lacks a notion of "wall time" that might make it useless in exercises where wall time is paramount (for example, red teaming). Second, it doesn't track the actual actions performed by a trainee during an exercise, so it can't judge whether he is following the right path or not. For this reason, s_1 is not well suited for teaching labs and penetration tests (that require following a precise plan).

Score s_2. Let G_u be a trainee graph and G_r a reference graph. Let l_u and l_r be the length of the shortest path from the start vertex to the end vertex in the trainee and reference graph, respectively. We define the *efficacy* ν as following:

$$\nu = \frac{l_u}{l_r} \quad (2)$$

If the trainee completes the exercise, we have $l_u = l_r$, thus $\nu = 1$ and the highest possible score. On the other hand, if the trainee can't complete any step of the exercise, we have $l_u = 0$, thus $\nu = 0$ and the lowest possible score. Intermediate performance produces scores in the $[0, 1]$ interval. Efficacy measures the progress of a trainee towards the exercise goal.

Similarly, let t_u be the wall time taken by a trainee to complete the exercise (this can be easily computed if the trainee graph has timestamp labels on its vertexes). Let t_{max} be the maximum wall time interval allotted to an exercise (this information may be inserted as a label of the end node in the reference graph). If the trainee

withdraws from the challenge, $t_u = t_{max}$. We define the *efficiency* η as following:

$$\eta = 1 - \frac{t_u}{t_{max}} \tag{3}$$

The same observations hold for η. A trainee that completes the exercise in a shorter time is assigned a higher score, while a trainee that doesn't complete the exercise in time is penalized with the lowest score. Intermediate performance produces scores in the $[0, 1]$ interval.

Score s_2 is defined as a linear combination of ν and η:

$$s_2(G_u, G_r) = \alpha\nu + (1 - \alpha)\eta \tag{4}$$

In this paper, we choose $\alpha = 0.5$, but the score may be tuned to weigh more efficacy or efficiency, according to the specific learning objective. This score rewards trainees who are able to reach the final goal of an exercise in the shortest time possible within the allotted time frame. Trainees who could not reach the goal at t_{max} are penalized with a null efficiency. This is a good candidate for red teaming exercises, since it takes into account the most important factor in this kind of engagement (wall time). Score s_2 shares the same problem as score s_1: it doesn't track trainee actions during the exercise. For this reason, s_2 might not be well suited for pointing out technical inefficiences in teaching labs and penetration tests.

Score s_3. Let $G_u = (V_u, E_u)$ be a trainee graph and $G_r = (V_r, E_r)$ a reference graph. We consider the *symmetric difference* of two sets A, B:

$$A \triangle B = (A \setminus B) \cup (B \setminus A) \tag{5}$$

In other words, $A \triangle B$ contains all members of A not in B and all members of B not in A. We use two interesting propertìes of symmetric difference:

- if two sets A and B coincide, $A \triangle B = \emptyset$;
- if two sets A and B are disjoint, $A \triangle B = A \cup B$.

We define the *efficacy* ν as following:

$$\nu = 1 - \frac{|V_u \triangle V_r|}{|V_u \cup V_r|} \tag{6}$$

If the trainee and reference graph coincide, we have $V_u \triangle V_r = \emptyset$, thus $\nu = 1$; the trainee has followed exactly the sequence of intermediate states modeled by the reference graph, and obtains the highest possible score. On the other hand, if the trainee and reference graph are completely disjoint, we have $V_u \triangle V_r = V_u \cup V_r$, thus $\nu = 0$; the trainee hasn't reached one single intermediate state of those in the reference graph, and obtains the lowest possible score. Intermediate performance produces scores in the $[0, 1]$ interval. Efficacy measures the ability of a trainee to follow the paths of a solution described in the reference graph.

Similarly, we define the *efficiency* η as following:

$$\eta = 1 - \frac{|E_u \triangle E_r|}{|E_u \cup E_r|} \tag{7}$$

The same observations hold for η. A trainee that performs the exact actions modeled in the reference graph is assigned the highest score, while a trainee that misses every action is assigned the lowest score. Intermediate performance produces scores in the $[0, 1]$ interval.

Score s_3 is defined as a linear combination of ν and η:

$$s_3(G_u, G_r) = \alpha\nu + (1 - \alpha)\eta \tag{8}$$

In this paper, we choose $\alpha = 0.5$, but the score may be tuned to weigh more efficacy or efficiency, according to the specific learning objective. This score rewards trainees who are able to reach the final goal of an exercise following the path defined by the reference graph. Trainees who could not reach the goal or make many mistakes are penalized with a low score. This is a good candidate for teaching exercises, since it tracks trainee actions during the exercise, taking into account the ability to follow the taught path to reach a goal. Score s_3 might not be well suited for red teaming exercises, since strict adherence to a particular technique is not as important as execution speed.

4.2 Aggregating scores

The aforementioned scores are intended as basic building blocks that aid in the construction of more elaborate trainee evaluation models. On different occasions it makes sense to compute n sub-scores $ss_1, ss_2, \ldots, ss_n$ and aggregate them into a new one s. For example, an instructor might want to evaluate both accuracy and execution speed, so he might want to consider aggregating scores s_2 and s_3. Alternatively, an exercise might be so complex to consider splitting it in several sub-exercises, each one with its own sub-goal. An instructor might then want to aggregate n instances of the same score (one for every sub-exercise considered).

Given n sub-scores $ss_1, ss_2, \ldots, ss_n$, we use the following aggregating function that restricts the final score in the [0,1] interval:

$$s_4 = \frac{\sqrt{ss_1^2 + ss_2^2 + \cdots + ss_n^2}}{\sqrt{n}} \tag{9}$$

5 Implementation details

Figure 9 shows the high level design of the proposed framework. The system architecture includes the following main components: a *collection cluster*, a *computation cluster* and a *dashboard*. The early decisions that

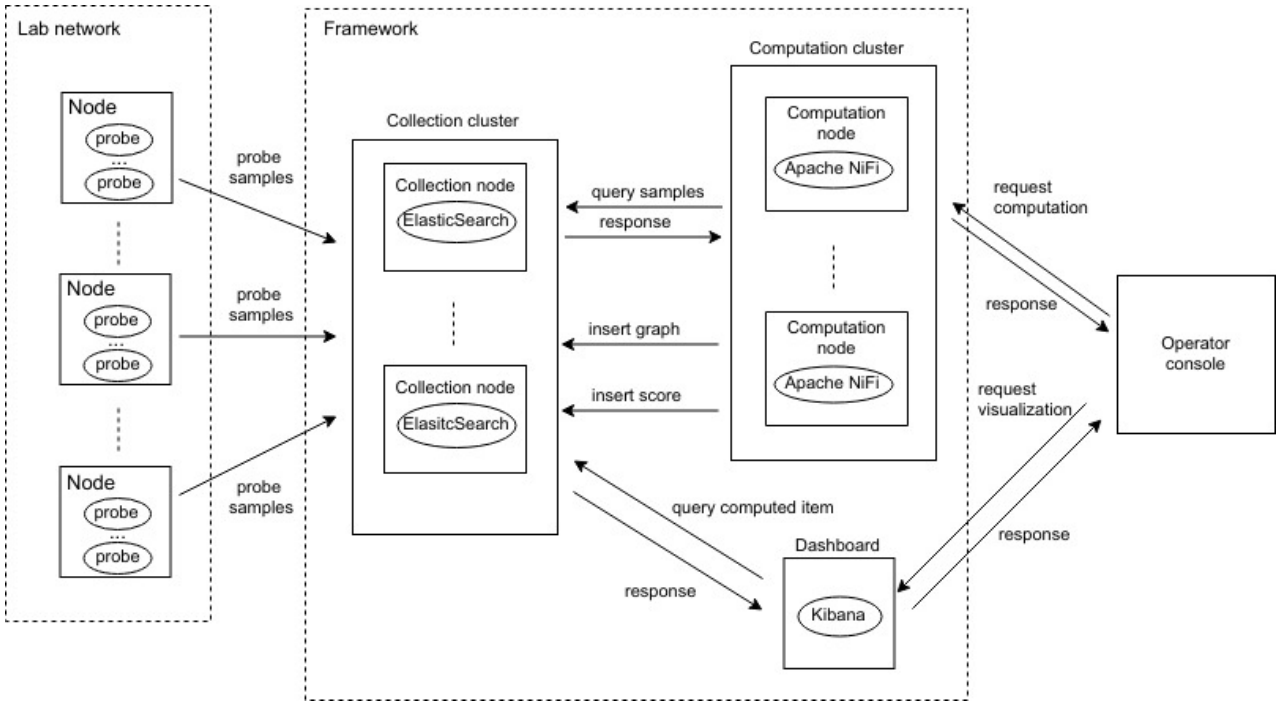

Fig. 9 High level architecture of the framework

inspired the design of the proposed architecture share the following important goals.

Scalability. The system must monitor a copious amount of data concerning multiple actions by different trainees in potentially large systems. These pieces of information must be collected, analyzed, stored and transmitted to system operators (or even displayed at real-time during an exercise), without loss of performance. Thus, the system must be designed from the ground up to be scalable to increasing workloads in terms of trainees, nodes in a lab, host, process and network metrics.

Fault tolerance. Hardware and software failures might compromise the computation of scores in an irreversible fashion. This might have bad consequences especially in a live exercise, where it disrupts cyber awareness. Framework components should thus be designed with redundancy and replication in mind.

Ease of use. The system should be easily accessible through standard Web interfaces and provide support for graphical visualizations of trainee behavior (mainly graphs and scores).

Each trainee node in the lab network runs a number of *probes* that monitor host, process, network, trainee activities during an exercise. The collection cluster is responsible for:

- storing these probes and making them available to compute graphs and scores;
- storing already computed graphs and scores;
- storing reference graphs.

Depending on the type of exercise performed, collected data will be in different formats, so the choice of a document-based storage backend is the most suitable. We rely on ElasticSearch as a scalable, fault-tolerant, document-based storage backend. We have also implemented a Firefox extension to intercept all HTTP requests by a trainee and log them to a file in the browser sandbox. Logs are collected with Filebeat, a light weight node agent (part of the ElasticSearch suite) for forwarding and centralizing log data to the collection cluster. Network probes are collected and filtered with `tshark`.

The computation cluster is responsible for the computation of graphs and scores. Following an operator request, this component initially checks whether a graph or a score has already been computed and, if that is the case, it returns it. Otherwise, it runs a three-staged pipeline. In the first stage, the collection cluster receives a query for the necessary metrics obtained from lab nodes and for reference graphs and responds with the requested data. In the second stage, based on the data extracted and on the type of exercise performed, a set of relevant trainee graphs is built and the desired scores are computed. In the third stage, all computed trainee graphs and scores are indexed back in ElasticSearch and made available for future requests. We rely on Apache NiFi as a scalable, loss-tolerant solution for dataflow computations.

The dashboard component provides visualizations of graphs and scores stored in the collection cluster. We rely on Kibana, a component of the ElasticSearch suite used to build visualizations from indexes in an ElasticSearch cluster. Several visualizations are possible; we use the graph API (designed to display relations among documents) to plot trainee and reference graphs, and we define metrics to display scores read from indexes. Kibana also makes it possible to filter collection data based on exercise, trainee and node name.

6 Experimental evaluation

In Section 6.1 we define the experimental testbed in terms of exercises, trainees and reference graphs describing final goals. We also introduce several scores aimed at evaluating learning outcomes. In Section 6.2 we compare different scores used in every exercise, discussing strenghts and weaknesses. The main goal is to identify the adequateness of scores to exercise categories.

6.1 Testbed

In this paper we consider the following three exercises.

Web attacks lab. This exercise models a lab session in an introductory course on Web attacks. Figure 10 shows its reference graph. It provides a sequence of Web-based challenges that a trainee should solve by following exactly the steps provided in the course:

- bypass a login form with a SQL Injection;
- exploit a local file inclusion to read a textual flag;
- submit the flag on a verification page.

The lab is implemented through a Docker container hosting an Apache Web server with a vulnerable, custom made PHP application.

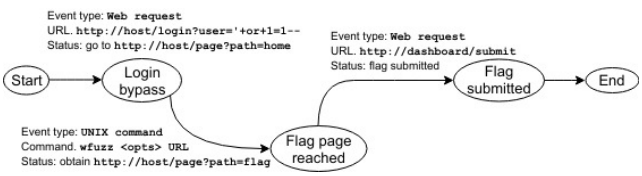

Fig. 10 Reference graph for the Educational Lab

Penetration testing lab. This exercise models a lab session in an introductory course on penetration testing. Figure 11 shows its reference graph. For reasons of space we have condensed the two privilege escalation paths into a single graph with two end vertexes and we have reduced labels to a bare minimum. The exercise provides a sequence of common activities (enumeration, remote-to-local and local-to-root privilege escalation) that a trainee should solve by following exactly the steps provided in the course:

- enumerate remote services;
- find a username via Apache userdir enumeration;
- brute force SSH passwords and log into the host;
- find two different ways to escalate to root (weak `sudo` permissions, vulnerable Python2 script).

The lab is implemented through a single VirtualBox guest hosting SSH, Apache, Python2, a vulnerable `sudo` configuration and a custom-made Python2 vulnerable script.

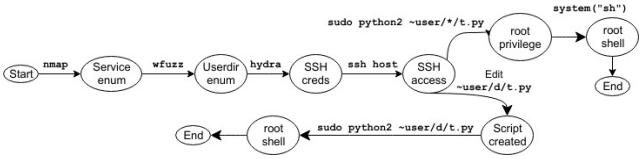

Fig. 11 Reference graph for the Penetration Test Lab

Red team engagement. This exercise models a red team engagement on a Windows infrastructure. Figure 12 shows its reference graph. Here the goal is to reach as many sub-goals as possible in the allotted time frame, regardless of the techniques used. For this reason, the granularity of the reference graph is lower than in the previous two examples (at the level of sub-goals, rather than individual commands). The exercise provides a sequence of common red teaming goals (enumeration, remote code execution through services, lateral movement, privilege escalation and data exfiltration) that a trainee should reach with the tools of choice, in no particular order (other than that indicated in the reference graph). The lab is implemented through two Vir-

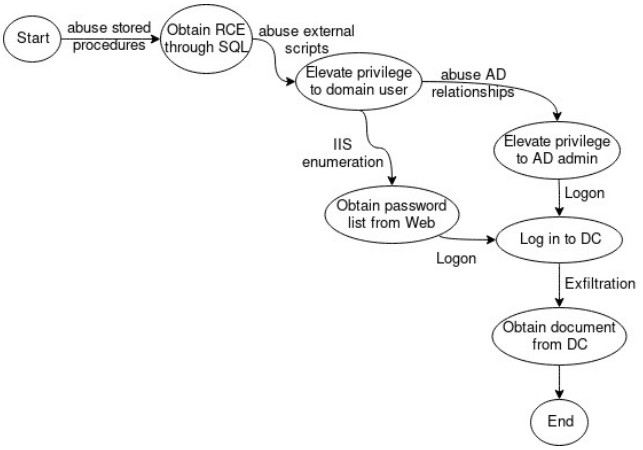

Fig. 12 Reference graph for the Red Teaming exercise

tualBox guests, one hosting Microsoft SQL Server and Internet Information Services, the other hosting Active Directory Services. Both machines have been misconfigured and populated with intermediate documents and flags required to prove reaching of a particular intermediate state.

We model three trainees with different background.

Trainee 1 is a Computer Science major with an interest in Cyber Security, basic knowledge on attacks on UNIX and Web-based systems, and no formal training on penetration testing or red teaming.

Trainee 2 is a junior pentester with formal training on attacks (UNIX, Windows and web-based systems) and penetration testing. He lacks training on red teaming.

Trainee 3 is a senior penetration tester with formal training on attacks (UNIX, Windows and web-based systems), defense, penetration testing and red teaming.

We consider the following scores for evaluation of trainee performance: s_1, s_2 ($\alpha = 0.5$, $t_{max} = 60m$ for Web attacks and Penetration Testing Labs, $t_{max} = 12h$ for Red Teaming), s_3 ($\alpha = 0.5$), s_4 (an aggregation over s_3). In the Education Lab and Red Teaming exercise, only one trainee graph is computed, and s_4 coincides with s_3. In the Penetration Testing exercise, two trainee graphs are computed, and s_4 aggregates the corresponding s_3 sub-scores.

6.2 Evaluation

In this section we briefly present the results of our experimental evaluation of scores for all trainees and all exercises. We outline merits and deficiencies of different scores and try to point out lessons learned.

Figures 13, 14 and 15 show the trainee graphs computed by the framework during all engagements. For reasons of space, each figure condenses all graphs for Trainee 1, Trainee 2, Trainee 3, and labels are reduced to a minimum. Dashed labels named `TraineeX` (X=1, 2, 3) indicate where Trainee X has got stuck in an exercise. Dummy nodes are labelled with `TraineeX_S_Err` to collect inaccuracies made by a specific trainee X towards intermediate node S (inaccuracies shared by several trainees could be placed in an appropriate dummy node). The start and end node have been labelled with timestamps indicating the start and end time of activities for the trainee that has completed the challenge (Trainee 3). If more trainees complete the challenge, more timestamps are added accordingly. Correct paths are highlighted in green, mistakes in red. Each graph confirms the abilities of Trainee 1, Trainee 2, Trainee 3. Trainee 1 (enthusiastic but not trained and inexperienced) is able to bring only elementary tasks to completion (service enumeration, login bypass through a basic SQL injection) and cannot make any progress in more advanced exercises. On the other hand, Trainee 2 (formally trained on penetration testing, but still inexperienced) manages to complete the Education Lab (the simplest exercise), makes halfway progress in the Penetration Testing Lab (not being able to figure out sneakier privilege escalation paths) and struggles in the Red Teaming engagement (where he only reachs the first sub-goal). Finally, Trainee 3 (formally trained on penetration testing and red teaming, with a lot of experience on the field) completes all exercices almost effortlessly.

Table 1 shows the performance of Trainee 1, Trainee 2 and Trainee 3 through scores s_1, s_2, s_3 (including ν and η) and s_4 in all the considered exercises. It can be immediately seen that different scores provide different views of performance. We outline merits and deficiencies of the considered scores, starting with s_1. Score s_1 offers a simplistic evaluation and is not very indicative of trainee performance in the considered exercises. The reasons are manyfold; first, s_1 returns a non zero value only if a trainee completes a challenge, thus making it impossible to evaluate progress towards a goal, which is often paramount in teaching and training activities. For example, in the Penetration Testing exercise, Trainee 1 and Trainee 2 are evaluated in the same way, although Trainee 2 progresses a bit more towards the final goal and makes less mistakes. Second, by definition of s_1, a deeper reference graph implies a longer shortest path and a smaller top score, which makes it very difficult to compare scores across different graphs. Indeed, in Table 1 the lowest top s_1 score is as low as 0.17. The only scenario where s_1 might be applicable is one where speed in terms of operation steps has to be evaluated (such as, for example, in vulnerability research, where one is interested in finding the shortest path to exploitation). In other scenarios, shortest path to completion alone cannot be the only evaluation criterium.

Exerc.	**Trn.**	s_1	$s_2/\nu/\eta$	$s_3/\nu/\eta$	s_4
Web	Tr. 1	0	0.17/0.33/0	0.16/0.2/0.13	0.16
Web	Tr. 2	0.33	0.81/1/0.62	0.51/0.6/0.42	0.51
Web	Tr. 3	0.33	0.94/1/0.87	0.75/0.75/0.75	0.75
Pentest	Tr. 1	0	0.21/0.42/0	0.27/0.27/0.27	0.34
Pentest	Tr. 2	0	0.36/0.71/0	0.47/0.45/0.5	0.58
Pentest	Tr. 3	0.17	0.65/1/0.3	0.87/0.9/0.83	0.84
Red	Tr. 1	0	0/0/0	0/0/0	0
Red	Tr. 2	0	0.08/0.17/0	0.14/0.14/0.13	0.14
Red	Tr. 3	0.17	0.92/1/0.83	0.87/0.86/0.87	0.87

Table 1 Trainee performance for the considered scenarios

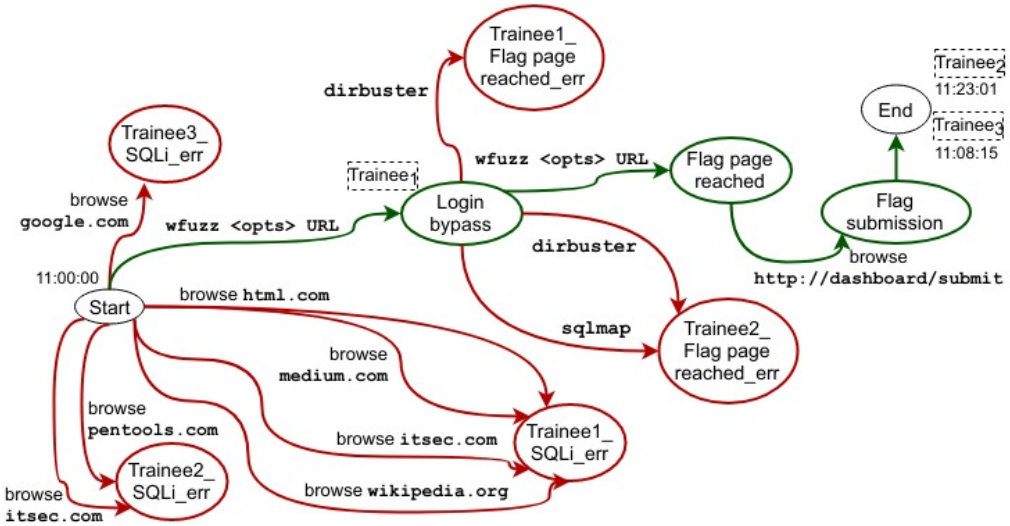

Fig. 13 Trainee graphs condensed - Web attacks lab

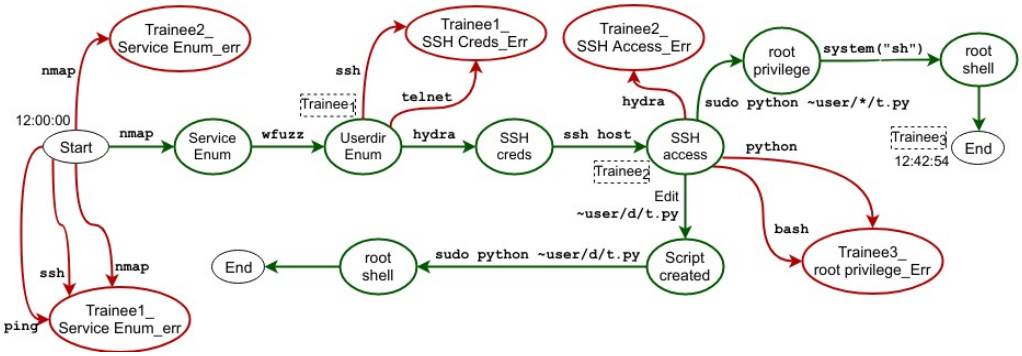

Fig. 14 Trainee graphs condensed - Penetration testing lab

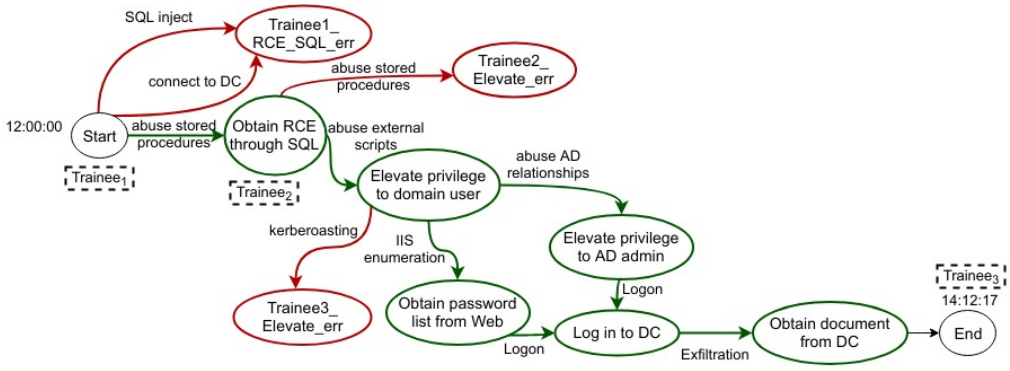

Fig. 15 Trainee graphs condensed - Red Teaming engagement

On the other hand, score s_2 allows to track goal progress through efficacy (ability to reach the final goal) and efficiency (speed). It does overall a pretty good job in differentiating trainee performance, especially in the Education Lab where both Trainee 2 and Trainee 3 complete the challenge in different times. Score s_2 also allows to precisely estimate sub-goal completion in the Red Team engagement (Trainee 2 completes one assignment over six, or ≈ 0.17). However, in other exercises that evaluate strict adherence to procedures s_2 tends to overestimate efficacy. Compare for example s_2 and s_3 for Trainee 2 and Trainee 3 in the Education Lab and Penetration Testing exercises; s_2 consistently outweighs s_3 in terms of ν.

Score s_3 shares the same structure as s_2. However, here efficacy measures the ability to follow a specific

sequence of intermediate steps (not only the ability to complete the challenge), and efficiency measures the ability to give a specific sequence of commands (rather than being fast). s_3 is a better score than s_2 with respect to adherence to specific paths. In the Education Lab, both Trainee 2 and Trainee 3 are rewarded by s_2 with $\nu = 1$, since they both complete the exercise; however, Trainee 2 makes many more mistakes than Trainee 1. Score s_3 captures this aspect, showing different efficacy values ($\nu = 0.6$ vs. $\nu = 0.75$, respectively) and different efficiency values ($\eta = 0.42$ vs. $\eta = 0.75$, respectively). On the other hand, s_3 fails to assign the highest possible efficacy value to Trainee 3 in the Red Teaming engagement (it only rewards $\nu = 0.86$), despite completion of all assignments. This might not be ideal if the primary evaluation criterium is sub-goal completion percentage.

Score s_3 is the most expensive to compute, and it rapidly degenerates with increasing graph complexity in terms of vertexes and edges. In very large scenarios it makes more sense to split a single, big reference graph in several, smaller ones (each one sharing a distinct subgoal), compute sub-scores and aggregate them. This is precisely the purpose of s_4, which is reported in Table 1 for sake of completeness. Score s_4 is identical to s_3 in Education Lab and Red Teaming (only one trainee graph is used). On the other hand, in Penetration Testing it is a normalized average of s_3 sub-scores for two different trainee graphs. As can be seen, s_4 isn't far away from s_3 scores in Penetration Testing exercise, thus making it a good candidate for score aggregation.

7 Conclusions

Cyber ranges are becoming a viable alternative for large-scale training of personnel involved in security at different levels (attack, defense, incident response, vulnerability research). However, current cyber range infrastructures lack the ability to monitor the performance of a player, and only report on exercise completion (or lack thereof). This paper addresses the problem by proposing a novel scoring framework and comparing candidate scores. Preliminary experimental results show how different scores may capture different abilities (execution speed and precision above all).

One interesting research direction is the classification of errors (which is not performed in the current prototype), that would allow an instructor to understand what a user is doing instead of following the intended path. Introducing new user probes and events to handle different types of user activities is another direction which is currently in the works. Finally, we are investigating new scores to evaluate team activities.

References

1. Bowen, B.M., Devarajan, R., Stolfo, S.: Measuring the human factor of cyber security. In: 2011 IEEE International Conference on Technologies for Homeland Security (HST). pp. 230–235. IEEE (2011)
2. Carlisle, M., Chiaramonte, M., Caswell, D.: Using ctfs for an undergraduate cyber education. In: 2015 {USENIX} Summit on Gaming, Games, and Gamification in Security Education (3GSE 15) (2015)
3. Čeleda, P., Čegan, J., Vykopal, J., Tovarňák, D.: Kypo–a platform for cyber defence exercises. M&S Support to Operational Tasks Including War Gaming, Logistics, Cyber Defence. NATO Science and Technology Organization (2015)
4. CISCO Cyber Range. `https://www.cisco.com/c/dam/global/en_au/\solutions/security/pdfs/cyber_range_aag_v2.pdf` (2016)
5. Evans, M., He, Y., Maglaras, L., Janicke, H.: Heart-is: A novel technique for evaluating human error-related information security incidents. Computers & Security **80**, 74–89 (2019)
6. Ferguson, B., Tall, A., Olsen, D.: National cyber range overview. In: 2014 IEEE Military Communications Conference. pp. 123–128. IEEE (2014)
7. Huang, K., Siegel, M., Stuart, M.: Systematically understanding the cyber attack business: A survey. ACM Computing Surveys (CSUR) **51**(4), 70 (2018)
8. IXIA Cyber Range. `https://www.ixiacom.com/solutions/cyber-range` (2014)
9. Jameel, A., Shahzad, K., Zafar, A., Ahmed, U., Hussain, S.J., Sajid, A.: The users experience quality of responsive web design on multiple devices. In: Proceedings of the 2nd International Conference on Future Networks and Distributed Systems. p. 69. ACM (2018)
10. Kordy, B., Kordy, P., Mauw, S., Schweitzer, P.: Adtool: security analysis with attack–defense trees. In: International conference on quantitative evaluation of systems. pp. 173–176. Springer (2013)
11. Kraemer, S., Carayon, P., Clem, J.: Human and organizational factors in computer and information security: Pathways to vulnerabilities. Computers & security **28**(7), 509–520 (2009)
12. Lampesberger, H.: Technologies for web and cloud service interaction: a survey. Service Oriented Computing and Applications **10**(2), 71–110 (2016)
13. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 336–345. ACM (2006)
14. Pernik, P.: Improving cyber security: Nato and the eu. International Centre for Defense Studies (2014)
15. Poolsappasit, N., Dewri, R., Ray, I.: Dynamic security risk management using bayesian attack graphs. IEEE Transactions on Dependable and Secure Computing **9**(1), 61–74 (2011)
16. Schneier, B.: Attack trees. Dr. Dobb's journal **24**(12), 21–29 (1999)
17. Vykopal, J., Vizváry, M., Oslejsek, R., Celeda, P., Tovarnak, D.: Lessons learned from complex hands-on defence exercises in a cyber range. In: 2017 IEEE Frontiers in Education Conference (FIE). pp. 1–8. IEEE (2017)
18. Zonouz, S.A., Khurana, H., Sanders, W.H., Yardley, T.M.: Rre: A game-theoretic intrusion response and recovery engine. IEEE Transactions on Parallel and Distributed Systems **25**(2), 395–406 (2013)