Enhancing Cybersecurity Education using Scoring Engines: A Practical Approach to Hands-On Learning and Feedback

Christopher Morales-Gonzalez
christopher_moralesgonzalez@uml.edu
University of Massachusetts Lowell
Lowell, MA, USA

Matthew Harper
matthew_harper@uml.edu
University of Massachusetts Lowell
Lowell, MA, USA

Pranathi Rayavaram
nagapranathi_rayavaram@uml.edu
University of Massachusetts Lowell
Lowell, MA, USA

Sashank Narain
sashank_narain@uml.edu
University of Massachusetts Lowell
Lowell, MA, USA

Xinwen Fu
xinwen_fu@uml.edu
University of Massachusetts Lowell
Lowell, MA, USA

Abstract

In today's digital landscape, the demand for skilled cybersecurity professionals is higher than ever. However, many educational programs primarily focus on theoretical concepts, leaving students with insufficient practical skills. To address this gap, students need actionable feedback on their hands-on labs and assignments. We present an open-source scoring engine that provides iterative, step-by-step feedback, enabling students to solve complex cybersecurity problems progressively. Integrated into existing courses, this engine can enhance labs with detailed, structured feedback, bridging the gap between theoretical knowledge and practical application. A preliminary study with 11 students showed that all participants could complete complex tasks using the feedback provided by the engine, with limited instruction from the authors. Additionally, about 90% of the students reported high satisfaction with the structured feedback. This approach has the potential to transform cybersecurity education, making it more interactive, practical, and aligned with real-world requirements.

CCS Concepts

• **Security and privacy → Social aspects of security and privacy**; • **Applied computing → Interactive learning environments**; **Computer-assisted instruction**.

Keywords

Cybersecurity Education, Automated Assessment Tools, Scoring Engines, Iterative Feedback

ACM Reference Format:
Christopher Morales-Gonzalez, Matthew Harper, Pranathi Rayavaram, Sashank Narain, and Xinwen Fu. 2025. Enhancing Cybersecurity Education using Scoring Engines: A Practical Approach to Hands-On Learning and Feedback. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701905

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701905

1 Introduction

In today's increasingly digital world, cybersecurity has become a critical field, with threats growing in complexity and frequency [17]. Organizations face constant threats from sophisticated cyberattacks, heightening the demand for cybersecurity professionals with practical, hands-on skills [3]. Educational institutions must respond to this demand by equipping students with the skills required to tackle real-world cybersecurity challenges effectively.

Many current cybersecurity education programs remain heavily focused on theoretical concepts. While understanding the underlying principles of cybersecurity is essential, this focus often creates a gap between classroom learning and practical fieldwork [20]. Consequently, students frequently graduate with a solid understanding of cybersecurity theories but lack the practical skills needed to implement and manage security measures effectively [30].

A critical aspect of effective learning in cybersecurity is receiving structured, timely feedback [48]. Current educational models often do not provide sufficient feedback on whether student projects are functional and secure at each step of the process [19]. Without this feedback, students might continue with flawed implementations, leading to compounded errors and diminished confidence in their skills. The absence of step-by-step validation can result in students making critical errors that compromise the entire system, making incremental, comprehensive feedback crucial.

Scoring engines have been successfully utilized in cybersecurity events such as capture the flag (CTF) competitions and hackathons, where real-time scoring and feedback are integral to the learning process [53]. However, existing scoring engines primarily focus on network probes that check the final outcomes, limiting their ability to provide feedback for intermediate file checks and configuration validations [53]. Ensuring each step in a cybersecurity task is correctly implemented is crucial, as flaws in early steps can compromise the entire system. Accurate step-by-step validation helps prevent such issues, ensuring secure implementations.

In this paper, we introduce an open-source scoring engine designed to be integrated into existing cybersecurity courses, enhancing lab environments with detailed, structured feedback. Unlike existing engines that primarily focus on network probes, our proposed engine includes local file checks and configuration validations, providing comprehensive feedback throughout the entire process of solving complex cybersecurity challenges. The engine operates iteratively, offering step-by-step feedback for each task and aiding troubleshooting as students progressively understand each component. By breaking down complex cybersecurity tasks into manageable subprojects and providing automated, incremental feedback, students ensure each step is correct before moving on. This approach ensures students gain practical, structured, hands-on experience alongside theoretical knowledge in real-time. By bridging the gap between theory and practice, our method enhances learning through active engagement, error reduction, and confidence building. Our objective is to make cybersecurity education more interactive, practical, and aligned with real-world needs, preparing students to be more effective professionals.

To evaluate our scoring engine, we conducted a preliminary user study with 11 students, including both undergraduates and graduates with varying levels of cybersecurity knowledge. We designed two critical and complex real-world tasks: securing a remote SSH [52] server and setting up TLS [1] for a web server. These tasks were chosen to assess the impact of the engine's feedback on student learning with limited instruction. Preliminary results show that all the students, even those with no prior experience with SSH and TLS, successfully completed the tasks by reviewing and reflecting on the feedback. Additionally, 90% of the students expressed satisfaction with the step-by-step feedback, noting it significantly enhanced their understanding and ability to securely set up these services, demonstrating the engine's powerful impact on improving practical cybersecurity skills.

In summary, this paper makes the following contributions:

- We design an open-source scoring engine that extends capabilities beyond network probes to include file and configuration validations, offering comprehensive feedback and iterative guidance aligned with classroom feedback.
- A preliminary study with 11 students showed that all participants, even those with no prior experience, successfully completed tasks using the feedback. Nearly all students reported satisfaction, highlighting the significant impact on enhancing practical cybersecurity skills.

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 details the design of the scoring engine. Section 4 covers the design of feedback scripts for evaluating our system. Section 5 describes the user study and evaluation. Finally, Section 6 presents the conclusion.

2 Related Work

In this section, we focus on assessment and feedback systems most relevant to our research, particularly Automated Assessment Tools (AATs) for cybersecurity tasks. AATs are typically categorized into static analysis, dynamic analysis, and machine learning-based tools [18, 22, 26, 27, 31].

Static Analysis: Industry-standard static analysis tools such as pylint [24], cpplint [15], CheckStyle [6], SpotBugs [41], and PMD [33] help identify code security issues like vulnerabilities, insecure coding practices, and potential exploits. However, they do not focus on providing feedback about custom files and configurations in cybersecurity tasks. Tools like PyTA [34], Gauntlet [14], and Expresso [47] are more geared toward general code quality analysis and are not well-suited for detailed cybersecurity feedback. These tools often use abstract syntax trees (ASTs) to compare student submissions to model solutions [28, 51]. Works such as Pedal [2] have developed more advanced analysis techniques, but are limited to Python and not scalable to other platforms.

Dynamic Analysis: Dynamic analysis systems evaluate running programs by identifying properties during execution. They use functional and unit test techniques, which can be extended to check students' work for security problems [4, 7–9]. These systems, like static analysis tools, are rule-based and limited in the types of checks they can perform. Additionally, these tools are typically integrated into IDEs, limiting their scalability and applicability in lab environments for cybersecurity education. Being IDE-based means they cannot provide detailed feedback to all students uniformly and lack a unified instructor dashboard. Moreover, specific test cases need to be built for each cybersecurity problem separately. Common frameworks for dynamic analysis include JUnit [21] for Java, PyTest [35] for Python, and NUnit [29] for .NET.

Machine Learning-Based Tools: Machine learning approaches [12, 45, 50] are increasingly integrated into AATs, leveraging large datasets and sophisticated algorithms to identify complex issues within student submissions. These tools can adapt to varied coding styles and provide nuanced feedback, enhancing the assessment process by addressing more complex, context-specific errors. However, ML-based techniques may struggle to provide step-by-step feedback and evaluate the incremental steps of a student's work unless specifically trained for that purpose, requiring thousands of student samples. Existing tools are also geared toward programming and not designed to assess cybersecurity problems.

Cybersecurity-Specific Tools: AATs, as discussed before, are often inadequate for cybersecurity tasks. Secure systems require not only an assessment of user code or commands but also a detailed evaluation of configuration files and security management, which these tools do not provide. Tools specifically designed for cybersecurity education are limited [25, 43]. Platforms like Code Hunt [13] and KYPO [49] offer advanced scoring capabilities, however, they are restricted to network probes and a binary scoring decision instead of structured feedback. Resources compiled in the "awesome-ctf" repository on GitHub [40], including platforms such as rootthebox, picoctf, and scorebot [32, 37, 39], are lacking in the detailed, step-by-step feedback crucial for effective learning. Works such as Truong et al. [46] focus on developing scalable cybersecurity hands-on labs and environments, however, they do not focus on student feedback.

While existing AATs and cybersecurity tools provide valuable assessments, they often focus on general programming tasks or basic security checks, lacking the detailed, iterative feedback essential for comprehensive cybersecurity education. Our proposed scoring engine addresses this critical gap by delivering granular, context-sensitive feedback and performing core system-level validations. It provides iterative, step-by-step guidance, enabling students to troubleshoot effectively and ensure the accuracy of each step. This comprehensive feedback mechanism deepens students' understanding of cybersecurity principles. By breaking tasks into manageable segments and offering detailed feedback at each stage, our engine bridges the gap between theoretical knowledge and practical application, significantly enhancing both learning outcomes and the development of robust, practical skills.

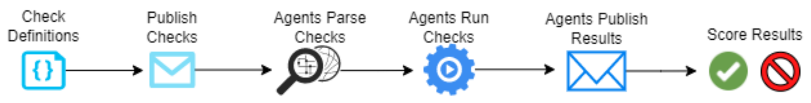


Figure 1: Automated Scoring Engine Workflow

3 System Design

This section outlines the workflow and architectural design of our scoring engine. We also detail the simple deployment process to demonstrate its feasibility. The source code is published here[1] and is made open-source to encourage broader adoption and collaboration.

3.1 Workflow

Implementing an automated scoring engine for cybersecurity education requires a structured workflow to assist instructors and students. This approach ensures a clear and consistent framework for learning and assessment. Figure 1 illustrates a simplified workflow of our scoring engine.

The first step in the workflow is the *Check Definitions* phase, where instructors create checks to score students' labs or assignments. These checks, written in a JSON file, specify the target system(s), an evaluation script, and an expected result code. This approach allows instructors to tailor checks to learning objectives, making assessments customizable. Instructors can reuse, improve, or modify checks to match evolving educational and cybersecurity needs. Once defined, checks are automatically uploaded during the *Publishing* phase to a centralized location. Agents in students' lab environments access these checks for grading, ensuring uniformity.

In the *Parsing* phase, agents in the students' lab environment analyze the checks to identify the necessary scripts to execute. During the *Run Checks* phase, the agents download and run these scripts on their host systems with administrative privileges, inspecting files and configurations, and performing runtime checks on the students' work. After running the checks, the agents move to the *Publishing* phase, where they report the results and provide detailed feedback to the main scoring engine. In the *Scoring* phase, the engine evaluates the results using standardized criteria to ensure fairness and offers appropriate feedback to the students.

The entire workflow is automatic and instant, providing students with real-time feedback on their tasks. This promotes active learning and eliminates the wait for TA/RA or instructor reviews. Instructors can view all students' results on a unified dashboard, while each student privately tracks their own progress. This maintains privacy and fosters individualized learning. Detailed feedback gives instructors insights into students' strengths and weaknesses, enabling targeted support. Students receive comprehensive feedback to identify areas for improvement and develop growth strategies, empowering them to take charge of their education with fair and transparent assessments.

3.2 Architecture

Figure 2 depicts our scoring engine's architecture, aligned with the workflow phases discussed earlier. Each component is chosen to simplify its respective phase. The selected technologies minimize instructor and student interaction with the infrastructure, allowing students to focus on experiential learning.

[1]GitHub Link- https://github.com/ChrisM09/CyberSeer

3.2.1 *ScoreStack.*

Our scoring engine design builds upon the open-source system ScoreStack [5]. We chose ScoreStack for its capability to create visual dashboards that track progress and tabulate results over extended periods. However, ScoreStack has two significant limitations: 1) it is primarily network-based, limiting its ability to perform complex, host-based granular checks, and 2) the feedback it provides is limited and requires complex code changes and recompilation to modify. Despite these limitations, ScoreStack's built-in visualizations and long-term storage make it a valuable tool for us to extend, allowing us to focus more on the educational component rather than technological details.

3.2.2 *Scoring API.*

To address ScoreStack's limitations, we developed a Python-based API using Flask [16] to seamlessly integrate our new infrastructure. This API interacts directly with ScoreStack and simplifies the addition of new functionality. When an instructor needs to publish checks to the lab infrastructure agents, they will use our API, which distributes the checks to the agents. The API also receives results and feedback from the agents and forwards them to ScoreStack for visualization. The API is highly customizable, allowing it to adapt to various requirements while maintaining compatibility with future ScoreStack updates.

3.2.3 *Agent Design.*

The infrastructure in Figure 2, aside from ScoreStack, automates the scoring engine for ease of use. This engine performs checks on each system via installed agents, providing quality feedback to students and promoting their growth in cybersecurity. The agent runs scripts specified within the defined checks. Key concerns include: *Q1*) How does the agent know which checks to run? *Q2*) Where are the target scripts located? *Q3*) Where does the agent report results? We address these questions next while explaining the infrastructure in Figure 2.

Q1: Check Dissemination: The issue of how each agent knows which checks to carry out is solved using the MQTT protocol [42]. MQTT operates on a subscriber-publisher model, where messages are organized into *topics* on the MQTT server. Entities subscribe to a topic to receive information or publish data on a topic to share information. We chose the open-source MQTT server Mosquitto [23] for its lightweight design and ease of deployment. This server provides a centralized way for our system to store and read information sent by our Scoring API or lab-environment agents. For example, when an instructor needs to publish checks for agents, they create a JSON file which the Scoring API publishes to the *checks* topic. As all agents are subscribed to the *checks* topic on the MQTT server, they receive and parse the checks ②. Each check contains a *target agent* field indicating which agent it is for. We used domain names instead of hard-coding IP addresses. These domain names are resolved by a Domain Name System (DNS) server included in the

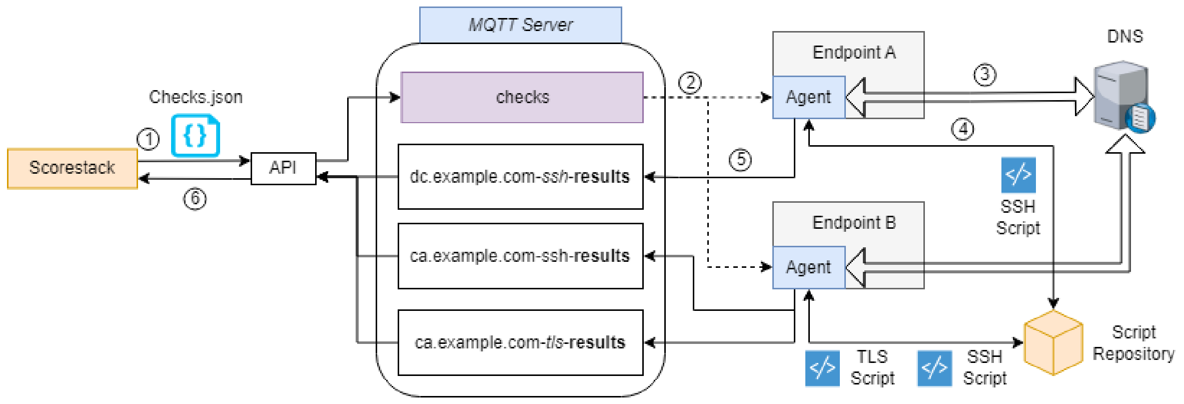


Figure 2: Architecture of the Agent-Based Scoring Engine Infrastructure for Cybersecurity Education.

infrastructure ③. Using a DNS server helps instructors set up their courses by allowing them to refer to lab systems by names rather than IP addresses, enhancing maintainability.

Q2: Check Repository: To carry out a check defined by the Scoring API, an agent must download the script specified within the check. Agents retrieve these scripts from a private repository, ensuring they use up-to-date versions and a consistent evaluation method, which promotes fairness. Our scoring engine employs the popular reverse proxy Nginx [44] to automatically direct agents to the server containing these checks ④. This modular design eliminates the need for preloaded scripts; agents download the latest versions automatically based on what the Scoring API publishes to the *checks* topic, reducing manual interactions. This capability to run complex code on a machine with a deployed agent addresses the first limitation in ScoreStack described in Section 3.2.1.

Q3: Reporting The agents report the output of an executed script to the MQTT server under a unique topic for each check/agent pair ⑤. The Scoring API reads this output, parses it to ensure fair and objective evaluation without interference from other students' work, and then sends the results to ScoreStack for display ⑥. The check scripts can be dynamically modified to adjust the amount of feedback given to each student, addressing the second limitation in ScoreStack mentioned in Section 3.2.1.

The defined architecture supports the dynamic addition, removal, or modification of checks, enabling adjustable granularity and flexibility. By embedding feedback within the scripts, the system provides detailed insights pertinent to the performed checks, which is crucial for practical cybersecurity education. This design allows students to receive immediate, specific feedback on their tasks, enhancing their understanding of each component of the cybersecurity workflow. Students can progressively build their practical cybersecurity skills, rectify errors in real-time, and gain confidence in their abilities, ultimately leading to a more comprehensive and hands-on learning experience.

3.3 Deployment

Our system is designed for ease of use, allowing instructors to deploy the entire scoring engine and agents with minimal effort. The deployment is fully automated and can be executed by running an Ansible [36] playbook. We chose Ansible because it is freely available, widely supported across different platforms, simplifies provisioning and configuration, and ensures consistent deployment. Instructors only need to install Ansible on their system and trigger the deployment without additional configurations in the student lab environment. The preparation steps are straightforward: instructors create scripts for testing student work, place them in a specified directory, and specify in a JSON file which systems will run these scripts—whether for all students or specific systems for team projects. The Ansible playbook requires the IP or domain of a Linux-based system that will serve as the scoring engine, the IPs or domains of the lab environment systems to deploy the agents, and the necessary credentials to access these systems.

The framework is deployed using Docker Swarm [11] and Docker Stack [10], which are platform-agnostic; however, we focus on Linux for consistent deployments. Docker Swarm clusters multiple Docker-installed machines into a single virtual system, simplifying the deployment, scaling, and management of containerized applications. Docker Stack, a feature of Docker Swarm, deploys multi-container applications, making it ideal for our architecture where each component (Scoring API, Script Repository, DNS Server) is containerized. Ansible playbooks automate the creation of the Docker Swarm cluster, interacting with Docker Stack to set up the scoring engine infrastructure, and initializing agents. The playbook concludes by providing a URL for instructors to access the scoring engine and access-restricted credentials to share with students. Additionally, we offer playbooks that can add or remove agents to accommodate different infrastructure sizes, ensuring scalability.

The system requires minimal resources, needing just two vCPUs, 10 GB of RAM, and less than 50 GB of disk space for the scoring engine. The lightweight agents require only a partial vCPU, less than 10 MB of RAM, and a few KBs of disk storage, minimizing resource issues in the student lab environments. Leveraging Ansible, Docker Swarm, and Docker Stack, our deployment process is efficient and streamlined, allowing instructors to focus on teaching rather than managing infrastructure. Students interact solely with the visual ScoreStack dashboard to receive feedback, while the rest of the system operates autonomously. This maximizes students' time on valuable cybersecurity experiences and minimizes setup and maintenance for instructors. Instructors can dynamically update checks for different labs or assignments during the semester, enhancing flexibility and usability. This approach ensures easy deployment in most educational settings, providing a robust platform for practical cybersecurity education.

4 Evaluation Scripts

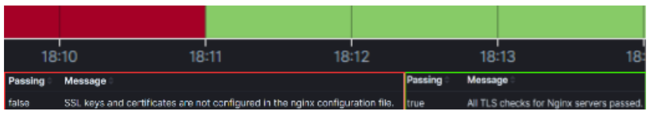
We created two scripts to evaluate the effectiveness of the scoring engine in practical cybersecurity education: one for SSH configurations and another for TLS setups. These tasks are crucial as misconfigurations in these services can lead to significant vulnerabilities. For example, an improperly configured SSH server might allow unauthorized access, resulting in data breaches or system compromises. At the same time, a misconfigured TLS setup could expose sensitive web data to interception during transmission.

SSH Configuration: The SSH configuration script provides crucial feedback to ensure the security of an SSH server setup, guiding students to correct potential issues. It starts by checking if the SSH binary is installed, prompting installation if necessary. Next, it verifies that the SSH service is running, alerting users if inactive. The script sequentially validates the SSH configuration file (*/etc/ssh/sshd_config*), focusing on critical security aspects. It ensures logging is set to 'INFO' or 'DEBUG', checks that root login is restricted, and verifies a low maximum number of authentication attempts to resist brute-force attacks. Additionally, it checks for enforced public key authentication, ensures password authentication is turned off, disallows empty passwords, and verifies that X11 forwarding is disabled to prevent exploits. Proper management of authorized keys is also checked to control and monitor access. These feedback checks provide step-by-step guidance, helping students understand each aspect and secure the SSH server effectively, building practical skills in a real-world context.

TLS Configuration: The TLS configuration script ensures the security of a web server's TLS setup, guiding students to build a configuration from scratch using step-by-step feedback. We chose Nginx for this evaluation due to its widespread use and simplicity. The script starts by checking if Nginx is installed and prompts installation if necessary. It then verifies that the Nginx service is running, alerting users to start it if inactive. The script sequentially reviews the Nginx configuration file to ensure strong TLS security. It checks that ports 80 (HTTP) and 443 (HTTPS) are correctly set up, verifies the presence and correct configuration of SSL keys and certificates, and ensures TLS protocols are set to 1.2 or higher, as earlier versions have vulnerabilities [38]. The script also performs syntax checks on the Nginx configuration file to prevent errors that may stop the service. Additionally, it performs runtime checks by making requests to the server, verifying that HTTPS returns a valid response, and ensuring redirection from HTTP to HTTPS.

Figure 3 illustrates a subset of the TLS checking Python code, showcasing the simplicity of setting up checks with our system. The code verifies whether the Nginx server config file includes key parameters (ssl_certificate and ssl_certificate_key) for TLS setup. If these parameters are missing, the script exits with a non-zero error code 1, showing a red score and a feedback message to the student, as shown in the figure. Once all TLS checks pass, the script exits with status code 0, indicating success, and the display changes to green with feedback confirming all checks succeeded.

```
ssl_keys = {"ssl_certificate_key", "ssl_certificate"}
if not ssl_keys.issubset(server_config):
    print(
        f"SSL keys and certificates are not configured "
        f"in the nginx configuration file."
    )
    sys.exit(1)
```



Passing	Message
false	SSL keys and certificates are not configured in the nginx configuration file.

Passing	Message
true	All TLS checks for Nginx servers passed.

Figure 3: Snippet of the TLS-checker Python code used in our system, and the presented student feedback.

5 Evaluation

This section overviews our user study's design and key insights into students' experiences and learning outcomes. It highlights how iterative feedback from our scoring engine improved their understanding of complex cybersecurity tasks, ability to correct errors, and overall confidence in applying practical skills.

5.1 User Study

Participants: The study involved 11 participants, including 7 undergraduate and 4 graduate computer science students from our university. Recruitment targeted computer science students interested in cybersecurity via Discord and email. We focused on students with some cybersecurity experience to ensure they had a foundational understanding of the principles, enabling them to better engage with and benefit from the complex tasks and feedback provided. Prior to the study, all participants were interviewed to assess their cybersecurity experience. With the exception of two students, participants had limited or no experience with the specific study tasks. The two experienced students had practical experience in setting up services, but not from scratch or using the structured approach of our scoring engine.

Study Procedure: Participants attended a 30-minute session at our university's cybersecurity lab, where they received an orientation on the scoring engine, the study objectives, and their tasks: setting up secure SSH and TLS configurations. This session aimed to ensure participants understood the importance of the tasks and the role of the scoring engine's feedback. For the SSH task, participants were given a misconfigured service to fix and secure. They needed to identify and correct configuration errors to ensure the SSH server was set up securely. For the TLS task, participants were provided with a webpage and required to deploy the Nginx webserver from scratch and then secure it using TLS. Participants had two weeks to complete these tasks, allowing them to work independently and accommodate their individual schedules and learning styles. The scoring engine ran continuously, tracking each student's progress and providing detailed feedback, as described in Section 4. At the end of the two weeks, interviews were conducted to understand their experiences and how the feedback influenced their learning process and task completion.

Data Collection: Data collection involved several components to ensure a comprehensive understanding of participants' experiences and outcomes. Initial interviews assessed participants' knowledge and skills in cybersecurity and Linux, along with details of prior experience, to establish a baseline for measuring the study's impact. During the study, the scoring engine tracked all interactions, including steps completed, errors made, and feedback provided. After completing the tasks, participants were interviewed to evaluate changes in their knowledge, skills, and confidence, as well as to gather feedback on their experience with the scoring engine and the usefulness of the step-by-step feedback.

5.2 Evaluation Results

This section presents the evaluation results, focusing on students' prior experience with automated grading systems, the scoring engine's impact on task difficulty and learning outcomes, and the effectiveness of the feedback provided.

Previous Student Experience: Our group consisted of undergraduate and graduate Computer Science majors, with the majority (9 out of 11) reporting some prior exposure to automated grading or scoring systems. This allowed them to compare our system against their previous experiences. The average rating of familiarity with the evaluation tasks was 3.7/5 for SSH and 2.2/5 for TLS configurations. The average difficulty rating for these tasks without the scoring engine's feedback was reported as 2.8/5 for SSH and 3.6/5 for TLS. These ratings indicate that students were relatively comfortable with SSH, likely because our courses emphasize SSH use, but found TLS configurations more challenging.

Difficulty and Learning Outcome: Once the scoring engine was activated and students received real-time feedback, they reported significantly lower difficulty ratings for both SSH and TLS: from 2.8/5 to 1.2/5 for SSH and from 3.6/5 to 2.1/5 for TLS configurations. Our analysis of the logs found that students made steady progress once they started the tasks, suggesting that the feedback helped them overcome obstacles quickly. While the scoring engine provided extra assistance, 6 out of 11 students stated they were confident about completing the tasks again without it, and 5 out of 11 said they might be able to. This indicates that the engine effectively built their understanding and skills, making them more capable and independent in handling similar tasks in the future. Additionally, 9 out of 11 students reported increased confidence in applying these security practices in real-world scenarios, and 2 students said their confidence might have increased. All students reported that the scoring engine improved their learning experience compared to traditional methods, citing real-time feedback and guided problem-solving as primary benefits. Specifically, students appreciated the real-time feedback for guiding them toward solutions; one student noted, "I never felt lost in my objective." Furthermore, 10 out of 11 students found the visual progression of checks turning from red to green motivating and satisfying, making the task completion process "more rewarding and satisfying."

Scoring Check Benefits: All students reported that the granular, step-by-step feedback from the checks was helpful, as it allowed them to receive specific real-time feedback at each stage of the configuration and deployment process without explicit instruction from us. This feedback effectively guided students in debugging their configurations and identifying possible issues. These results underscore the positive impact our scoring engine had on their learning experience. By providing real-time iterative checks on host-based and network-based services, we significantly enhanced the students' learning experience by giving them clear direction when they might otherwise be lost.

Scoring Check Feedback: Although students found the checks beneficial, there were some concerns about the level of feedback. Specifically, 2 out of 11 students suggested decreasing the feedback slightly, while another 2 out of 11 wanted to increase it. This indicates a need to carefully design the checks and feedback levels to avoid overwhelming students or giving away too much information. The feedback must guide students toward the correct solution without simply providing it, with the level of detail tailored to their prior experience. In summary, as a majority of students (7 out of 11) found the feedback satisfactory, our scoring engine provided valuable, balanced feedback that enhanced students' learning experiences. While it is essential to customize feedback specific to prior experiences, the overall positive responses demonstrate that the idea behind using scoring engines for feedback supports learning and skill development in cybersecurity.

Application and User Interface: We also asked the students whether they would like to see this engine deployed in other courses. Given that the engine is automated and uses limited resources, it is feasible to deploy it effectively even in large courses. The students found the visual feedback of boxes changing color, combined with the real-time textual feedback, very helpful in understanding what they needed to do and whether they were progressing correctly. The usability received an average rating of 4.3/5, with students primarily requesting small changes in the user interface or feedback provided by the score checks. Suggested improvements included modifying the dashboard for easier navigation, having score checks turn green based on a baseline rather than upon 100% completion, and adding more tasks and checks to complete. Overall, students strongly endorsed the engine, highlighting its potential to enhance learning across various courses. The positive feedback underscores the efficacy and adaptability of the scoring engine, demonstrating its ability to provide clear, actionable guidance and significantly improve the educational experience.

6 Conclusion

This paper introduces an open-source scoring engine to enhance cybersecurity education by providing comprehensive feedback on student tasks. The engine's iterative feedback aids students in troubleshooting and building confidence as they progress. Integrating this engine into courses can offer immediate, actionable feedback, bridging the gap between theoretical knowledge and practical skills. Our preliminary user study showed improved task completion and higher satisfaction, with all students completing complex tasks with minimal instructor help and 90% expressing high satisfaction with the feedback. This approach promotes active learning, reduces errors, and builds confidence, fostering competent cybersecurity professionals. Future work will expand the engine's capabilities and integrate it with more advanced tasks, further aligning cybersecurity education with industry needs.

References

[1] Ghada Arfaoui, Xavier Bultel, Pierre-Alain Fouque, Adina Nedelcu, and Cristina Onete. 2019. The Privacy of the TLS 1.3 Protocol. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019). https://doi.org/10.2478/popets-2019-0065

[2] Austin Cory Bart and Luke Gusukuma. 2024. Autograding Python Code with the Pedal Framework: Feedback Beyond Unit Tests. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. https://doi.org/10.1145/3626253.3633416

[3] Scott Bell and Michael Oudshoorn. 2018. Meeting the Demand: Building a Cybersecurity Degree Program With Limited Resources. In *2018 IEEE Frontiers in Education Conference (FIE)*. https://doi.org/10.1109/FIE.2018.8659341

[4] Alessandro Bertagnon and Marco Gavanelli. 2020. MAESTRO: A Semi-AutoMated Evaluation SysTem for Programming Assignments. In *Proceedings of the 2020 International Conference on Computational Science and Computational Intelligence*. https://doi.org/10.1109/CSCI51800.2020.00177

[5] Simon Buchheit, Sean Newman, Kyle Carretto, and Hulto. 2021. Scorestack. https://github.com/scorestack/scorestack. Accessed: 2024-07-13.

[6] Checkstyle Community. 2024. Checkstyle. https://checkstyle.sourceforge.io/. Accessed: 2024-07-13.

[7] Benjamin Clegg, Maria-Cruz Villa-Uriol, Phil McMinn, and Gordon Fraser. 2021. Gradeer: An Open-Source Modular Hybrid Grader. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training*. IEEE/ACM. https://doi.org/10.1109/ICSE-SEET52601.2021.00015

[8] Ricardo Conejo, Beatriz Barros, and Manuel F. Bertoa. 2019. Automated Assessment of Complex Programming Tasks Using SIETTE. *IEEE Transactions on Learning Technologies* 12, 4 (2019). https://doi.org/10.1109/TLT.2018.2876249

[9] Daniel Coore and Daniel Fokum. 2019. Facilitating Course Assessment with a Competitive Programming Platform. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.3287511

[10] Docker, Inc. 2023. *Docker Stack CLI Reference*. https://docs.docker.com/reference/cli/docker/stack/ Accessed: 2024-07-13.

[11] Docker, Inc. 2024. *Docker Swarm*. https://docs.docker.com/engine/swarm/ Accessed: 2024-07-13.

[12] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated Repair of Programs from Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press. https://doi.org/10.1109/ICSE48619.2023.00128

[13] Sandro Fouché and Andrew H. Mangle. 2015. Code Hunt as Platform for Gamification of Cybersecurity Training. In *Proceedings of the 1st International Workshop on Code Hunt Workshop on Educational Software Engineering* (Baltimore, MD, USA) *(CHESE 2015)*. https://doi.org/10.1145/2792404.2792406

[14] Gauntlet.io. 2024. Gauntlet.io: The Continuous Application Security Platform. https://gauntlet.io/. Accessed: 2024-07-13.

[15] Google. 2024. cpplint - A Style Checker for C++ Code. https://github.com/cpplint/cpplint. Accessed: 2024-07-13.

[16] Miguel Grinberg. 2018. Flask: Python Web Applications. Accessed: 2024-07-07.

[17] Abdulla Hussain, Azlinah Mohamed, and Suriyati Razali. 2020. A Review on Cybersecurity: Challenges & Emerging Threats. In *Proceedings of the 3rd International Conference on Networking, Information Systems & Security* (Marrakech, Morocco) *(NISS '20)*. Article 28. https://doi.org/10.1145/3386723.3387847

[18] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. https://doi.org/10.1145/1930464.1930480

[19] Sabrina Jahn and Jürgen Mottok. 2020. Work in Progress: Towards an Academic Secure Software Engineering Curriculum for Engineers. In *2020 IEEE Global Engineering Education Conference (EDUCON)*. https://doi.org/10.1109/EDUCON45650.2020.9125210

[20] Samuel Ndueso John, Etinosa Noma-Osaghae, Funminiyi Oajide, and Kennedy Okokpujie. 2020. *Cybersecurity Education: The Skills Gap, Hurdle!* Springer International Publishing. https://doi.org/10.1007/978-3-030-50244-7_18

[21] JUnit Team. 1997. *JUnit: A Simple Framework to Write Repeatable Tests*. http://junit.org/ Last Accessed: 2024-07-13.

[22] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* (sep 2018). https://doi.org/10.1145/3231711

[23] Roger Light. 2009. *Mosquitto: An Open Source MQTT Broker*. https://mosquitto.org/ Last Accessed: 2024-07-13.

[24] Logilab and Pylint Contributors. 2024. Pylint. https://pypi.org/project/pylint/. Accessed: 2024-07-13.

[25] Mac Malone, Yicheng Wang, and Fabian Monrose. 2023. Securely Autograding Cybersecurity Exercises Using Web Accessible Jupyter Notebooks. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. https://doi.org/10.1145/3545945.3569862

[26] Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi. 2024. Automated Grading and Feedback Tools for Programming Education: A Systematic Review. *ACM Transactions on Computing Education* 24, 1 (2024). https://doi.org/10.1145/3636515

[27] Sidhidatri Nayak, Reshu Agarwal, and Sunil Kumar Khatri. 2022. Automated Assessment Tools for Grading of Programming Assignments: A Review. In *2022 International Conference on Computer Communication and Informatics (ICCCI)*. https://doi.org/10.1109/ICCCI54379.2022.9740769

[28] Anh-Tu Nguyen and Van-Dung Hoang. 2024. Development of Code Evaluation System Based on Abstract Syntax Tree. *Journal of Technical Education Science* 19 (2024). https://doi.org/10.54644/jte.2024.1514

[29] NUnit Community. 2024. NUnit: A Unit-Testing Framework for All .Net Languages. https://nunit.org/. Accessed: 2024-07-13.

[30] Ogugua Chimezie Obi, Onyinyechi Vivian Akagha, Samuel Onimisi Dawodu, Anthony Chigozie Anyanwu, Shedrack Onwusinkwue, and Islam Ahmad Ibrahim Ahmad. 2024. Comprehensive Review on Cybersecurity: Modern Threats and Advanced Defense Strategies. *Computer Science & IT Research Journal* 5, 2 (2024). https://doi.org/10.51594/csitrj.v5i2.758

[31] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Transactions on Computing Education* (jun 2022). https://doi.org/10.1145/3513140

[32] picoCTF Team. 2024. picoCTF. https://github.com/picoCTF/picoCTF. Accessed: 2024-07-07.

[33] PMD Team. 2024. PMD. https://pmd.github.io/. Accessed: 2024-07-13.

[34] PyTA Project. 2024. PyTA: PythonTA, a Static Analyzer for Python. https://github.com/pyta-uoft/pyta. Accessed: 2024-07-13.

[35] pytest-dev team. 2023. pytest: Simple Powerful Testing with Python. https://pytest.org Last Accessed: 2024-07-13.

[36] Red Hat. 2024. Ansible Automation Platform. https://www.ansible.com/. Accessed: 2024-07-13.

[37] RootTheBox Team. 2024. RootTheBox. https://github.com/moloch--/RootTheBox. Accessed: 2024-07-07.

[38] Ashutosh Satapathy and Jenila Livingston. 2016. A Comprehensive Survey on SSL/TLS and Their Vulnerabilities. *International Journal of Computer Applications* 153 (2016). https://api.semanticscholar.org/CorpusID:13638447

[39] Scorebot Team. 2024. Scorebot. https://github.com/LegitBS/scorebot. Accessed: 2024-07-07.

[40] Amanpreet Singh. 2024. Awesome CTF. https://github.com/apsdehal/awesome-ctf. Accessed: 2024-07-07.

[41] SpotBugs. 2024. SpotBugs: A Tool for Finding Bugs in Java Programs. https://spotbugs.github.io/. Last Accessed: 2024-07-13.

[42] Andy Stanford-Clark. 2010. MQTT V3.1 Protocol Specification. https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html. *IBM Corporation* (2010). Last Accessed: 2024-07-13.

[43] Valdemar Švábenský, Jan Vykopal, Pavel Čeleda, and Ján Dovjak. 2023. Automated Feedback for Participants of Hands-On Cybersecurity Training. *Education and Information Technologies* (2023). https://doi.org/10.1007/s10639-023-12265-8

[44] Igor Sysoev. 2024. NGINX. https://nginx.org/en/ Accessed: 2024-07-13.

[45] Botond Tarcsay, Fernando Perez-Tellez, and Jelena Vasic. 2023. Using Machine Learning to Identify Patterns in Learner-Submitted Code for the Purpose of Assessment. *Pattern Recognition* 13902 (2023). https://doi.org/10.1007/978-3-031-33783-3_5

[46] Nghi Truong, Paul Roe, and Peter Bancroft. 2004. Static Analysis of Students' Java Programs. In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30* (Dunedin, New Zealand) *(ACE '04)*.

[47] Ultrapico. 2024. Expresso Regular Expression Tool. https://ultrapico.com/Expresso.htm. Accessed: 2024-07-13.

[48] Jan Vykopal, Radek Ošlejšek, Karolína Burská, and Kristína Zákopčanová. 2018. Timely Feedback in Unstructured Cybersecurity Exercises. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. https://doi.org/10.1145/3159450.3159561

[49] Jan Vykopal, Pavel Čeleda, Pavel Seda, Valdemar Švábenský, and Daniel Tovarňák. 2021. Scalable Learning Environments for Teaching Cybersecurity Hands-On. In *2021 IEEE Frontiers in Education Conference (FIE)*. https://doi.org/10.1109/FIE49875.2021.9637180

[50] J. Walker Orr and Nathaniel Russell. 2021. Automatic Assessment of the Design Quality of Python Programs with Personalized Feedback. *Educational Data Mining (EDM)* (2021). https://api.semanticscholar.org/CorpusID:235313490

[51] Chengguan Xiang, Ying Wang, Qiyun Zhou, and Zhen Yu. 2024. Graph Semantic Similarity-Based Automatic Assessment for Programming Exercises. *Scientific Reports* 14 (2024). https://doi.org/10.1038/s41598-024-61219-8

[52] Tatu Ylonen and Chris Lonvick. 2006. The Secure Shell (SSH) Protocol Architecture. RFC 4251. https://www.rfc-editor.org/rfc/rfc4251.

[53] Abdullah Zafar, Muhammad Mudassar Yamin, Basel Katt, and Espen Torseth. 2024. All Flags Are Not Created Equal: A Deep Look into CTF Scoring Algorithms. *Expert Systems with Applications* 254 (2024). https://doi.org/10.1016/j.eswa.2024.124436