Designing a Platform to Train Secure Programming Skills With Attack-and-Defend Exercises

Leo St. Amour
Dept. of Computer Science
Virginia Tech
Blacksburg, USA
lstamour@vt.edu

Eli Tilevich
Dept. of Computer Science
Virginia Tech
Blacksburg, USA
tilevich@cs.vt.edu

***Abstract*—The increasingly poor state of software security poses significant threats to many of modern society's critical functions. Computing educators play a pivotal role in equipping future software engineers with the necessary skills to build secure systems. However, while traditional security courses often focus on conceptual knowledge, practical application is crucial for ensuring students can develop robust, secure software. Despite the importance of hands-on experience, students often lack suitable platforms for practicing secure programming. Inspired by drill-and-practice platforms that effectively train general programming skills, we have been working on a similar platform that focuses explicitly on teaching secure coding practices through active learning strategies.**

In this paper, we discuss the design of our prototype implementation: SECURECODER. Rooted in active learning principles, SECURECODER's design aims to promote student-centered education by encouraging students to actively apply theoretical secure programming concepts. Specifically, SECURECODER engages students with interactive attack-and-defend exercises, challenging them to exploit or patch software vulnerabilities in a sandboxed environment. Our ultimate objective is to bridge the gap between theoretical knowledge and practical application, fostering a deeper understanding and retention of secure coding principles.

Through immediate and actionable feedback on validated exercises, SECURECODER is designed to reinforce learning and empower students to iteratively refine their solutions and build confidence in their skills. To study the potential of our design, we conducted a pilot study. The study results indicate that participants found SECURECODER to be relevant and engaging. Further, participant perceptions toward the attack-and-defend exercises suggest that SECURECODER's design has the potential to enhance secure programming education.

Encouraged by SECURECODER's initial positive reception, we plan to open-source the project, inviting the broader education community to contribute to and benefit from shared security expertise. These collaborative efforts are essential for educating the next generation of security-aware software engineers. By integrating hands-on practice and active learning techniques, SECURECODER's design aims to address the urgent need for practical, skill-based security education, preparing students to meet the ever-evolving challenges of engineering secure solutions in the real world.

***Index Terms*—Software security, Active learning, Drill-and-practice exercises, Web-based training**

I. INTRODUCTION

Despite software's widespread role in critical aspects of modern society, dangerous security vulnerabilities remain alarmingly common [1]. In response to this state of affairs, the Cybersecurity and Infrastructure Security Agency, in collaboration with 17 U.S. agencies and international partners, recently issued a call-to-arms manifesto pleading for software developers to adopt a "secure by design" approach [2]. The publication emphasizes that achieving this goal requires "foster[ing] a software developer workforce that understands security." In other words, software engineers must be thoroughly trained in secure programming practices—a goal that depends on adequate classroom training.

Higher education has long recognized the importance of incorporating security into computer science curricula [3]. However, despite widespread agreement, security education remains an open problem. A 2017 study found that only 30% of survey respondents felt that their formal security education adequately prepared them for completing professional security tasks [4]. Although many universities have revamped their programs to emphasize security, students continue to produce vulnerable code [5] and graduate into the workforce with profound gaps in secure programming knowledge [6].

Addressing these challenges requires leveraging long-standing academic theory. Active learning posits that students learn more effectively when actively engaging in their learning via discussions, activities, or exploration [7]. To internalize important computing principles, students must reinforce their conceptual knowledge through deliberative practice [8]. Consider the example of teaching data structures: an instructor introduces a data structure in a lecture, and then students spend numerous hours working on programming assignments that apply the data structure across various contexts.

Drill-and-practice tools are educational platforms designed to help students effectively retain knowledge in subjects such as math, language, and sciences. The term *drill-and-practice* refers to reinforcing learning through repetitive exercises paired with immediate feedback [9]. By engaging students with targeted topics, these tools build foundational skills by adjusting difficulty—simplifying tasks when students struggle and increasing complexity as they improve. Notable examples include Duolingo [10] and Khan Academy [11], which use repetition and feedback to strengthen core language acquisition and math competencies, respectively.

Several drill-and-practice tools have been developed to help

build fundamental computing skills. These platforms have been proven effective in academic settings [12], [13] and seen commercial success [14], [15]. However, despite the growing role of drill-and-practice platforms in computing education, their application to teaching and refining secure programming skills remains unexplored. This gap presents a valuable opportunity to address the secure programming skills deficit by creating a specialized drill-and-practice platform tailored to this critical domain.

A key challenge in teaching security skills is addressing the diverse perspectives that underpin fundamental security tasks. These perspectives are commonly categorized as either "red" or "blue" indicating whether they are offensively or defensively oriented, respectively. Red teams adopt the perspective of attackers, identifying and exploiting vulnerabilities, whereas blue teams focus on defending against attacks by mitigating damage or eliminating vulnerabilities [16].

However, the distinction between red and blue skills is not always clear-cut; a common security principle asserts that the best offense is a good defense. Analysts trained in both perspectives—often referred to as "purple"—develop a holistic understanding of security threats, enabling them to anticipate and counteract malicious activities effectively [16]. This dual training strategy equips analysts to address the complex and evolving challenges in cyber security. Inspired by the success of this approach, we aim to adapt and apply the "purple" strategy to secure programming education. We posit that by learning to exploit vulnerabilities and understand their implications, students should be better able to mitigate and patch them—or, ideally, write secure code from the outset.

To realize this vision, we have designed and prototyped SECURECODER, a drill-and-practice platform that offers a web-based environment for students to practice secure programming skills. Students enhance their "purple" skills by engaging in two types of exercise: (1) red, or attack-focused, which involves exploiting vulnerabilities, and (2) blue, or defense-focused, which involves identifying and remediating vulnerable code. SECURECODER's initial set of exercises is rooted in the MITRE common weakness enumeration (CWE) dataset [17], which catalogs well-known vulnerability classes and provides a foundation for secure programming skills.

To support both red and blue exercises, we designed a software architecture that differs fundamentally from traditional drill-and-practice programming platforms. Specifically, SECURECODER requires learners to perpetrate an attack with demonstrable results, which demands that SECURECODER itself withstand all types of attacks. As a result, the architecture must be robustly hardened against both intentional exploits and malicious learners. To operate within this threat model, our design incorporates a code execution sandbox and logic for detecting and handling intended program faults or exploits.

We conducted a pilot study to assess the potential of our approach and gather initial feedback. Despite its limited size of 15 users, our study suggests that a platform like SECURECODER could be well-received due to its potential to benefit secure programming education. Moreover, working on our proof-of-concept highlighted that delivering meaningful educational impacts with SECURECODER will require a communal effort.

This paper makes the following contributions:

- We introduce SECURECODER, a prototype drill-and-practice platform designed to train secure programming skills through attack-and-defend exercises.
- We describe SECURECODER's design and discuss the challenges of providing a platform for exercising red and blue secure programming skills.
- We present the results of a pilot study, discussing user perceptions of SECURECODER and how our approach aligns with established education theories.

The remainder of this paper is organized as follows: Section II reviews related work on active learning, drill-and-practice platforms, and secure programming education. Section III, details SECURECODER's design and implementation. Section IV presents the results of our pilot study, followed by a discussion of their implications in Section V. Finally, we outline our vision for SECURECODER in Section VI and provide concluding remarks in Section VII.

II. RELATED WORK

Three fields of inquiry inspire this work: (1) active learning theory, (2) enhancing programming skills through drill-and-practice exercises, and (3) secure programming education. Each of these fields has garnered significant attention from education researchers, leading to successful outcomes and the development of widely used tools for computing education. Despite their individual successes, these fields have traditionally remained separate, and our work seeks to bridge this gap. Next, we provide an overview of each field and discuss their relevance to this work.

A. Active Learning

The educational theory of *active learning* describes a straightforward yet powerful principle: actively involving students, rather than limiting them to passive listening, enhances the learning process [7]. Active learning emphasizes incorporating activities that foster development, higher-order thinking, engagement, and exploration.

One approach to implementing active learning is the *gradual release of responsibility* method [18], which formalizes the process of progressively transferring responsibility for learning from teacher to student. Lesson plans based on this method typically consist of three phases: (1) "I do it," in which the teacher demonstrates a skill or concept; (2) "we do it," which involves guided or collaborative practice of the skill or concept; and (3) "you do it," in which students independently demonstrate or apply the skill or concept.

Another approach that facilitates active learning is the *flipped classroom*, which reverses traditional in-class and out-of-class activities [19]. In this model, students learn new concepts independently at home and use class time to ask specific questions or engage in exercises that apply the concepts in practice. The flipped classroom emphasizes the importance of

both guided practice and independent labs. Prior studies have assessed the positive and widespread impact of the flipped classroom model on computing education at the university and college levels [20], [21].

Based on these prior works, active learning holds promise for engaging students across disciplines in practical activities that reinforce theoretical concepts. Inspired by this observation, we aim to apply active learning to enhance secure software engineering teaching practices. By designing an educational intervention congruent with active learning principles, our ultimate goal is to provide a versatile tool that supports active learning frameworks, such as the gradual release of responsibility and flipped classroom models.

B. Drill-and-Practice Programming Platforms

Drill and practice is an educational technique that employs systematic repetition of a skill or concept through exercise, examples, or practice problems [9]. This technique has proven effective in disciplines like the sciences [22]. Several popular programming training websites, including HackerRank [14], LeetCode [23], and CodeWars [15], have adopted drill-and-practice as their exercise model, offering a large bank of challenges that focus on core programming concepts, such as data structures and algorithms, across various programming languages. While these platforms are valuable tools for job preparation, their objectives, though related, differ from those of formal education.

Beyond commercial platforms, extensive research has explored the efficacy of drill-and-practice in academic contexts [24], [25], [26]. In this review, highlight three representative examples: CodingBat, Infandango, and CodeWorkout. First, CodingBat [27] is a free site hosting an extensive collection of small Python and Java programming exercises. Each exercise evaluates a submission's correctness based on unit tests. CodingBat provides but does not require accounts to use the service. Next, Infandango focuses on Java programming skills [28]. Its modular architecture separates the front-end and unit-testing infrastructures, enabling plug-in replacement of components like the database engine. Unlike CodingBat, Infandango requires users to authenticate before submitting exercises. Finally, CodeWorkout offers short programming exercises and multiple choice questions [12]. Specifically designed for educational research, CodeWorkout features robust data collection and evaluation infrastructure.

Irrespective of whether they are commercial products or academic projects, these platforms verify the correctness of user submissions by comparing expected and actual outputs. The results of these unit tests serve as user feedback [12]. Unfortunately, this design is inherently ill-suited for secure programming exercises. The reason is subtle but profound: an insecure program's behavior may not always result in incorrect output. Instead, security issues can manifest as a program fault, hijacked control flow, unauthorized access, or other unexpected behaviors. Capturing and verifying these issues requires a fundamentally different approach. Because a unit-test-based architecture does not adequately support exercising secure programming skills, this work explores using a different strategy to verify correctness.

C. Secure Programming Education

The computing education community has long recognized the importance of teaching secure programming. In 2010, the Summit on Education in Secure Software explored the challenges of secure software education and how to improve educational secure programming practices [3]. Among its key recommendations was the need for students to actively practice secure programming through additional textbook content, labs, and exercises. Despite these efforts, more than a decade later, studies indicate that the knowledge and skill gap in secure programming persists [6].

The computing education research community has explored both pedagogical approaches and practical tools for teaching secure programming. Universities often teach software security through dedicated courses [29] or by integrating security concepts into the broader computer science curriculum [30], [31]. Both approaches commonly cover secure programming. To improve the effectiveness of teaching secure programming, researchers have proposed leveraging well-known educational process models [32] as well as creating and evaluating novel methodologies [33]. Beyond the classroom, educators have explored improving secure programming skills by organizing clinics or entrusting their teaching assistants with engaging students on secure programming topics [34].

Other research examines the efficacy of applying competition and gamification to instill the importance of software security and equip programmers with the requisite skills. Cyber competitions, including capture-the-flag (CTF) competitions, have become popular and effective methods for teaching and practicing security concepts [35]. In a CTF competition, participants solve security-related challenges—such as exploiting vulnerabilities or reverse engineering binaries—to capture a "flag," a hidden piece of data that serves as proof of completion. Other types of games focus on recognizing patterns of secure programming [36].

To facilitate the introduction of important secure programming skills, computing educators have experimented with various innovative tools. Examples include ASIDE and ES-IDE, Java Eclipse plug-ins that provide real-time warnings whenever a student's workspace contains potentially insecure code [37], [38].

Despite prior efforts to improve secure programming education, the gap between theoretical knowledge and practical skills persists. This ongoing gap presents a promising research opportunity, especially given the consensus that secure programming skills are critical and must be practiced [3]. Our vision is that a drill-and-practice platform can bridge this gap by offering the community a versatile tool for honing secure programming skills.

III. SecureCoder

Recognizing the need to revitalize secure programming education with active learning practices, we propose a novel

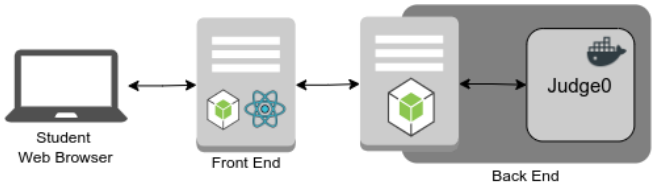
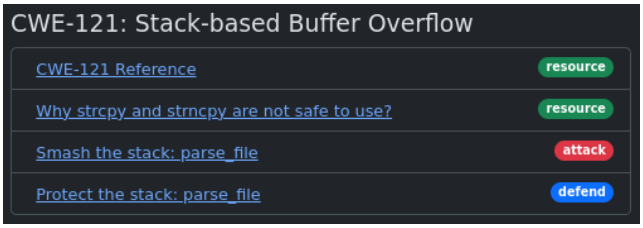
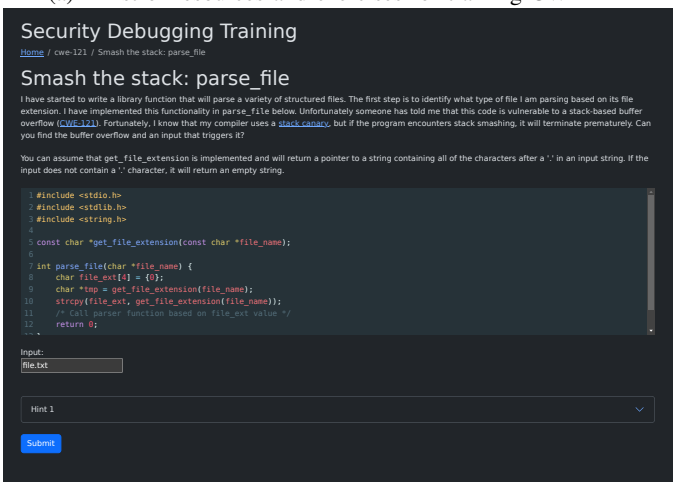


Fig. 1: SECURECODER architecture overview



(a) A list of resources and exercises for training CWE-121



(b) Interface for completing an attack exercise for CWE-121

Fig. 2: Examples of the SECURECODER front end interface

drill-and-practice platform specifically designed to focus on secure programming. We have prototyped this design as SECURECODER, a web-based framework. SECURECODER's design aims to provide a practical learning aid that aligns with established active learning frameworks [7], [18], [19]. This section details the platform's design, its prototype implementation, the collection of pilot exercises, and SECURECODER's unique threat model.

A. Design Goals

SECURECODER's design aims to address the following key concepts:

- **Learner engagement**: facilitates active learning through interactive exercises.
- **Security focus**: emphasizes the development of secure programming skills.
- **Comprehensiveness**: supports both attack and defense exercises, providing a "purple" learning experience.
- **Extensibility**: enables the addition of new exercises and categories to support evolving content.
- **Robustness**: protects itself from intended and unintended exploits.

As argued above, the design of existing drill-and-practice platforms lacks the necessary features to support exercises focused on secure programming skills. This gap highlights the need to design a new architecture that harmoniously integrates both attack and defense exercises. Our design's unique focus is on teaching secure programming skills, specifically the ability to both exploit and patch vulnerable code. In line with the "purple" team model in cyber security, our design adopts a comprehensive approach emphasizing the relationship between offensive and defensive skills. Due to a vast diversity of known vulnerabilities, the platform must be easily extensible to accommodate new exercises targeting a variety of security bugs. Additionally, since exercises involve exploiting vulnerabilities, the design must prevent successful exploits from compromising the functions of the underlying system.

B. Platform Architecture

SECURECODER's design follows the classic client-server architecture, specially adapted to meet the needs of secure programming education. The platform's functionality is divided between the front end (client) and the back end (server). The front end is implemented as a web application interface, providing a user-friendly environment for students to navigate and complete exercises. The back end serves as a web service, hosting the exercises, executing code submissions, and generating feedback that is communicated back to the user. Figure 1 presents an overview of SECURECODER's architecture.

1) Front End: The front end provides a user interface that allows learners to navigate and complete the available exercises. It is implemented using Node.js and React, popular JavaScript libraries for web app development. The main page (index) presents users with a complete list of available exercises. Each exercise is categorized according to the secure programming skill it targets, using the CWEs classification for our purposes. Furthermore, each exercise is labeled as either "attack" or "defend," depending on whether the exercise aims to exploit or patch a vulnerable function. Alongside the exercise list, each category may include additional resources that offer examples or guidance to assist learners. Figure 2a illustrates an exercise category listing, specifically for CWE-121: stack-based buffer overflow.

After selecting an exercise, the user is directed to the exercise view, where they are presented with a detailed prompt and a vulnerable function or code snippet. In attack exercises, the code snippet is read-only, and the user is tasked with entering input in a separate text box to trigger the vulnerability. In contrast, defense exercises ask the user to modify the code snippet to remedy the bug or vulnerability. Figure 2b presents a screenshot of SECURECODER's exercise interface, specifically for the "attack" exercise targeting a CWE-121 vulnerability.

To enhance the educational experience, SECURECODER provides students with feedback on each submitted solution. If

```
long convert_server_port(char *port_str) {
  long port = atol(port_str);
  if (port <= 0 || port > 65535) {
      printf("port %s is invalid\n", port_str);
      return ERROR;
  } else if (port <= 1024) {
      printf("port %ld is privilleged\n", port);
      return ERROR;
  }
  return port;
}
```

Listing 1: Vulnerable code for CWE-476 exercises

```
const char *get_file_extension(const char *file_name);
int parse_file(char *file_name) {
    char file_ext[4] = {0};
    char *tmp = get_file_extension(file_name);
    strcpy(file_ext, get_file_extension(file_name));
    /* Call parser function based on file_ext value */
    return 0;
}
```

Listing 2: Vulnerable code for CWE-121 exercises

```
char *say_hello(char *name) {
    if (name == NULL) {
      printf("you must provide a name!\n");
      return NULL;
    }

    char cmd[64] = "cat greeting.txt && echo ";
    name = strtok(name, " ");
    strncat(cmd, name, 46);

    FILE *proc = popen(cmd, "r");
    char *output = calloc(sizeof(char), 32);
    fread(output, sizeof(char), 31, proc);
    fclose(proc);
    printf("%s\n", output);
    return output;
}
```

Listing 3: Vulnerable code for CWE-78 exercises

the solution successfully exploits or patches the vulnerability, SECURECODER responds with a message offering a detailed explanation of the particular bug and how it can be mitigated. If the solution is incorrect, the exercise informs the student about what went wrong. Additionally, each exercise may include a set of hints designed to guide the student in the right direction or suggest additional resources. Students can view one or all of the hints at any time.

2) Back End: The back end consists of two components: (1) an exercise server and (2) a code execution sandbox. The exercise server, implemented using Node.js, hosts and serves the exercises to the front end. Its RESTful application programming interface (API) for requesting exercises and submitting solutions offers for maximal flexibility: any front end can interact with the API. The API provides the following endpoints: `GET /exercises`, which returns a summary list of exercises by category; `GET /exercise/<id>`, which returns the content of exercise `id`; and `POST /exercise/<id>`, which submits a solution for exercise `id`.

This design allows SECURECODER to meet our extensibility requirements. The back end populates the exercise collection at runtime by ingesting a set of YAML files. Each YAML file specifies key information about an exercise, including its title, prompt, type (attack or defend), category, vulnerable code, hints, and the driver function, discussed in greater detail in the following subsection. Additional exercise YAML files can be added to introduce new exercises to the system. Relying on YAML files trades simplicity for scalability, a trade-off acceptable in early development stages. Specifically, YAML is a text-based protocol that incurs processing and transmission overheads. A more transmission-efficient format can be utilized for a high throughput system.

The back end's second component is the code execution sandbox, which compiles and executes student submissions. For this purpose, we selected an existing solution, Judge0, a sandbox environment that supports common programming languages, including C, C++, and Java. Judge0 exposes a robust API for submitting code and retrieving detailed execution results [39]. Conveniently packaged as a Docker container, Judge0 runs locally on any back-end environment. The exercise server acts as an intermediary, relaying submissions and resulting feedback between the front end and the execution sandbox.

C. Exercises

As a parameterizable and extensible platform, SECURECODER is designed to accommodate various collections of secure programming exercises. However, we chose to leverage widely recognized vulnerabilities for our proof-of-concept implementation and pilot study. The secure programming skills we chose to include in our pilot are courtesy of the MITRE common weakness enumeration (CWE) list, which defines common security weaknesses and vulnerabilities. We selected three CWEs that represent a diverse range of vulnerabilities and their associated security implications. This particular collection trains on the following CWEs:

- **CWE-476: NULL Pointer Dereference**. A NULL pointer dereference occurs when a program dereferences a pointer that is expected to be valid. This weakness typically results in the program crashing or terminating prematurely. The CWE-476 exercises feature the vulnerable function in Listing 1.
- **CWE-121: Stack-based Buffer Overflow**. A stack-based buffer overflow occurs when data written to a stack-allocated buffer exceeds its bounds, potentially corrupting stack-based variables or call frame data. The CWE-121 exercises feature the vulnerable function in Listing 2.
- **CWE-78: OS Command Injection** - OS command injection occurs when usanitized user input is passed to a function that executes system commands (e.g., `system` or `popen`). The CWE-78 exercises feature the vulnerable function in Listing 3.

```c
void segfault_handler(int signum) {
  switch (signum) {
  case SIGABRT:
    fprintf(stderr, "Well done!\n");
    exit(0);
  default:
    fprintf(stderr, "Error; report to site admin.\n");
  }
  exit(1);
}

int main(int argc, char *argv[]) {
  struct sigaction new_action = {0};
  new_action.sa_handler = segfault_handler;
  sigemptyset(&new_action.sa_mask);
  new_action.sa_flags = 0;
  sigaction(SIGABRT, &new_action, NULL);
  sigaction(SIGSEGV, &new_action, NULL);

  parse_file(argv[1]);
  fprintf(stderr, "You did not trigger an overflow.\n");
  return 1;
}
```

Listing 4: Driver code for CWE-121 attack exercise

A serious challenge in designing SECURECODER was managing the wide range of system behaviors that successful exploits can cause, each of which must be appropriately identified and validated. For example, a NULL pointer dereference causes a segmentation fault, a buffer overflow sends an abort signal, and a command injection produces an unexpected output. To address this behavioral variability, we introduce exercise-specific drivers—functions that verify an exploit or patch's success. To illustrate this concept, consider Listing 2. The attack variant for this exercise includes the driver code shown in Listing 4.

The process's exit code determines whether the submission has succeeded. Specifically, an exit code of zero indicates success, while a non-zero exit code suggests failure. Additionally, the `stderr` output stream is used to communicate all feedback. For the exercise in Listing 2, if the student provides input that triggers the buffer overflow, the driver code will handle the `SIGABRT` signal and exit with code zero. In contrast, if execution continues beyond the call to `parse_file`, the driver will provide feedback and return a non-zero exit code.

The driver function needs to be modified to address the defense variant of this exercise. Instead of calling the vulnerable function with the student's input, the driver calls the patched function multiple times—first with valid arguments to ensure the functionality remains unchanged and again with arguments that trigger the vulnerability. The driver then inverts the logic of the attack variant: if the driver handles a `SIGABRT` signal, the bug remains unpatched.

Notice that the system capabilities required to capture and validate our exercises differ from the unit-test approaches used by drill-and-practice platforms to train programming skills. Rather than simply comparing actual and expected outputs, each SECURECODER exercise may require unique validation functionality. However, many drivers can be reused across exercises, either unchanged or slightly modified. Given the important role of SECURECODER's driver functions in meeting our design goals, we expect a certain degree of domain expertise from our exercise providers. Specifically, they should understand the impact of target vulnerabilities on system behavior and be comfortable writing system-level code snippets that accurately capture that behavior.

D. Threat Model

Robustness is a critical aspect of our design. As previously mentioned, SECURECODER intentionally solicits malicious inputs meant to attack or exploit an executing program. This threat is exacerbated by defense exercises, which allow students to submit arbitrary code for execution on the back end. Consequently, we must ensure that SECURECODER is designed to protect itself from both the intended exploits and potential attacks from malicious learners.

Fortunately, our decision to use Judge0 as an execution sandbox helps mitigate these threats. Judge0 offers additional configuration options to limit a sandboxed process's CPU time, memory, threads, and network access. Furthermore, Judge0's sandbox environment is built on isolate, a tool designed to ensure secure execution in programming contests and grading systems, even when operating under a malicious threat model [40], [41]. With Judeg0's resource constraints in place, SECURECODER can implement reasonable limits on system execution while benefiting from the protections inherent to the sandbox.

IV. EVALUATION

To obtain an initial assessment of SECURECODER's promises as an educational intervention, we conducted a pilot study, whose design and results we discuss next. The goal of our pilot study was to answer the following research questions:

- **RQ1:** What is the potential of a drill-and-practice approach for promoting active learning practices for teaching secure programming skills?
- **RQ2:** For a drill-and-practice platform for teaching secure programming skills, which types of exercise (i.e., defensive, offensive, combined) should be provided and/or prioritized?
- **RQ3:** How do the research and education communities perceive the adequacy of secure programming education?

To answer these questions, our pilot included (1) hands-on experience with SECURECODER and (2) a perception survey. The survey was distributed to willing participants among computer science graduate students and industry practitioners. Participants were asked to spend no more than 15 minutes attempting at least two exercises on SECURECODER, specifically both the attack and defense exercises for a CWE of their choice. After completing the exercises, each participant answered survey questions regarding their experiences with SECURECODER, their perceptions of its effectiveness, and their opinions on debugging within undergraduate computer science curricula. We deliberately framed the activities as debugging exercises, as they were performed outside of the context of a security course. Our aim was to present the

TABLE I: Pilot study survey questions and participant demographics

Question	Mean	Variance
Q1. The availability of a debugging exercise website would positively impact my debugging skills.	4.20	0.60
Q2. The debugging exercises were interesting and engaging.	4.20	0.60
Q3. Dividing exercises into offensive/defensive sub-tasks helped me better understand the underlying bug.	4.07	0.64
Q4. My experiences as an undergraduate adequately prepared me to perform the debugging tasks.	3.93	1.07
Q5. Bug finding should be included in an undergraduate computer science education.	4.80	0.17
Q6. We need to better incorporate bug finding into undergraduate computer science curricula.	4.80	0.17
Participant programming experience: 2-5 years: 7 participants, 5-10 years: 5 participants, 10+ years: 3 participants		
Participant primary language: Python: 10 participants, Java: 3 participants, C++: 1 participant, JavaScript: 1 participant		

exercises in a way that would be immediately accessible to any programmer.

The pilot study included 15 participants recruited through advertisements targeting computer science graduate students at our university and software engineers in professional development organizations. The survey consisted of nine questions: six five-point Likert scale questions, ranging from Strongly Disagree (1 point) to Strongly Agree (5 points), and three additional questions asking for participants' years of programming experience, primary programming language, and an open-ended question to provide additional feedback. Table I presents the Likert scale questions, their respective means and variances, and participant backgrounds. Questions Q1 and Q2 address RQ1, soliciting participant opinions on SECURECODER's engagement and the utility of including attack and defense exercises. Question Q3 addresses RQ2, aiming to gain insight into the value of SECURECODER's holistic exercises. Questions Q4-6 address RQ3, seeking participants' perceptions of secure programming and software bugs in undergraduate CS curricula.

V. DISCUSSION

Our answers to the research questions are based on our experiences designing and implementing SECURECODER, our proof-of-concept drill-and-practice platform, and the preliminary results from a pilot study. This section discusses each research question, identifies potential threats to validity, and outlines our ultimate vision for SECURECODER as an educational aid.

A. RQ1: Teaching Secure Programming with Active Learning

To the best of our knowledge, our approach is the first to propose a drill-and-practice platform specifically designed to teach secure programming skills. Therefore, we aim to understand SECURECODER's applicability and its potential to support active learning within this domain.

Based on the results of our pilot study, all participants expressed a positive opinion regarding SECURECODER's value and utility. Specifically, we analyzed the responses to Q1 and Q2, which assess the participants' perceptions of SECURECODER's training value and engagement, respectively. The mean scores for Q1 and Q2 were 4.2, indicating that the average participant more than agreed with the statements. Additionally, several participants mentioned in their free-form feedback that both the exercises and SECURECODER's format were "fun," suggesting that our design objective of providing learners with a positive and engaging experience has been met.

We would not want to overgeneralize our conclusions—after all, SECURECODER is only a proof of concept, and our user study is only a pilot. However, the results suggest that a drill-and-practice platform targeting secure programming skills holds promise as an effective means for reifying active learning principles, providing an opportunity to train relevant skills in practical and engaging settings.

B. RQ2: Types of Secure Programming Exercises

One could argue that security training presents unique challenges, as students must develop a multifaceted perspective that encompasses both the mindsets of attackers and defenders. Secure programming is, in essence, a bit of a chicken-and-egg problem: patching a vulnerability requires understanding how it could be exploited, and exploiting a vulnerability requires students to understand how it manifests and how it can be patched. Our insight is that there is significant value in training these concepts holistically rather than treating them in isolation.

It would be impossible to definitively demonstrate the value of a holistic approach through a pilot study. Therefore, our study aimed to assess initial perceptions of a "purple" (i.e., combining both offensive and defensive characteristics) approach to training secure programming. As expected, respondents overwhelmingly viewed our attack-and-defend approach positively. Specifically, Q3 asked whether the holistic approach improves vulnerability comprehension. The mean score for Q3 was 4.07, indicating that the average participant agreed with the statement.

Based on this promising feedback, we hope to encourage the security education community to contribute exercises that combine offensive and defensive traits to our platform. With a sufficiently large collection of exercises, we plan to address this research question more definitively through a quantitative evaluation.

C. RQ3: Perceptions

Industry needs and requirements should drive secure programming education [42]. There is a shared understanding that modern software engineers often struggle to produce software that meets the required security standards [4]. Moreover, formal education plays a crucial role in the software engineering training pipeline [3], [4]. Consequently, educators must find innovative and effective methods to teach security.

Successful computing professionals often gain confidence in their technical knowledge and skills through professional experiences rather than formal education. Rather than relying on their undergraduate security courses, engineers learn necessary practical skills through on-the-job experience. Tools, like SECURECODER, aim to better prepare future software professionals by equipping them with these practical skills during their education rather than waiting until after they enter the workforce.

In addition to assessing perceptions of SECURECODER, our pilot study also aimed to gain insight into participants' views on secure programming within an undergraduate computer science curriculum. To gain this insight, Q4 asked participants how adequately their undergraduate education had prepared them to complete the exercises. Q5 and Q6 focused on participants' opinions regarding the current and future state of secure programming in the undergraduate curriculum. The mean score for Q4 was 3.97, indicating that the average participant felt somewhat unprepared to complete the exercises. In particular, four participants disagreed with the statement. The mean scores for Q5 and Q6 were 4.80, reflecting a strong consensus among participants that secure programming should be an important component of an undergraduate computer science curriculum; however, current practices fall short of their expectations.

These results suggest that this work addresses an important and timely issue in computing education, especially considering that all participants have already completed their undergraduate studies. We are encouraged to continue developing SECURECODER and studying its potential impact on software engineering education. Our ultimate goal is to create a communal resource that applies active learning principles to revitalize the acquisition of secure programming skills for aspiring software engineers.

D. Classroom Applications

Intended as a practical educational aid, SECURECODER has great potential for directly supporting active learning in classroom settings. To illustrate this potential, we describe two scenarios in which SECURECODER can be applied to support an active learning framework: (1) general release of responsibility and (2) flipped classroom.

Suppose an instructor plans to teach students about buffer overflow vulnerabilities. In the first scenario, the instructor would use SECURECODER to introduce and demonstrate the vulnerability as part of the lecture material. Following the lecture, in lab settings, students can complete assigned group exercises on the SECURECODER platform. After collectively developing the skills to patch and exploit buffer overflows, students are assigned SECURECODER exercises to complete individually, either as in-class assignments or homework.

In the second scenario, students would be assigned to learn about buffer overflows before class through read-ahead slide decks or instructional videos. During class, the instructor would use SECURECODER exercises to address any lingering questions interactively. Once these questions have been adequately answered, students would use SECURECODER to drill and practice exploiting and patching buffer overflows, reinforcing the learned material.

While SECURECODER's implications in an actual classroom setting remain to be fully explored, its potential to support active learning practices is evident. The scenarios discussed above can serve as templates for teaching other security topics through the gradual release of responsibility or flipped classroom methods. However, SECURECODER's versatile design extends beyond these frameworks and can be adapted to other active learning approaches.

E. Threats to Validity

This research is subject to several threats to validity, which we discuss next. To study user perception, we conducted a pilot study with 15 participants. Typically, a larger sample size is more likely to yield generalizable results [43]. However, the initial assessment suggests a positive reception of SECURECODER and its potential. Additionally, our participants were recruited from our university and through personal industry contacts. As a result, a sample from the broader software engineering student and professional population might have yielded different findings. Nevertheless, the small variances in our survey responses indicate that increasing the sample size is unlikely to drastically affect our main takeaways.

Some study design decisions may threaten the generalizability of our study results. First, the selected exercises may not fully represent the range of security concepts that software engineering students struggle with or the issues they will face in the industry. However, we carefully selected our exercise topics from popular vulnerabilities in the CWE, which are prevalent in real-world applications.

Second, our survey questions used the terms "bug finding" and "debugging" to refer to offensive and defensive concepts. This phrasing was a deliberate choice to re-frame the problem in a way that would be more approachable for participants completing the exercises outside of a dedicated security context. Our findings might have differed if we had used more specialized secure programming terminology. We suspect that software developers may exhibit less confidence in their abilities when explicitly asked about their security competencies than when tasked with familiar debugging challenges. Nonetheless, we do not believe re-framing the questions would have yielded drastically different results.

Finally, our questions' affirmative phrasing may have influenced participant responses. We asked participants to reflect on their experiences with our proof of concept. However, this facet of our evaluation does not unfairly benefit our approach. Had participants interacted with a fully mature drill-and-practice tool, their experiences would likely have been even more positive.

VI. VISION

Encouraged by the preliminary evaluation results, we envision maturing SECURECODER into a versatile active learning

platform. To this end, we outline efforts to perfect and scale our proof of concept alongside the following directions.

Two key concepts that drive effective drill and practice are self-paced learning and real-time difficulty adjustment [9]. While designing SECURECODER, we considered these concepts but lacked the time and resources to implement them in our prototype. One promising avenue for inspiration comes from CTF competitions, which successfully gamify security concepts. In a CTF competition, each challenge is assigned a point value indicative of its difficulty. Participants are free to attempt challenges that align with their perceived skill level. Additionally, some competitions require participants to complete certain challenges before others become available. A mature implementation of SECURECODER would build upon this design by assigning difficulty levels to exercises and creating exercise progressions that reinforce and synthesize individual skills.

Ultimately, for SECURECODER to become a practical drill-and-practice platform, it must provide adequate opportunities for repetition, which requires a large and diverse set of exercises. To this end, we envision a communal development model where experts within the broader software engineering community contribute and vet exercises. Inspired by the source code pull request model, proposed exercises would be collaboratively reviewed, refined, and approved before being added to the exercise repository. This approach combines an individual contributor's expertise with the crowd's wisdom. By adopting this communal development model, SECURECODER can maintain an extensive and up-to-date collection of exercises. To further this goal, we intend to open-source our implementation, inviting experts to contribute to future exercises and advance the platform's functionality.

Regarding our future evaluation directions, we plan to design a user study to assess the educational impact of drill-and-practice platforms, such as SECURECODER, on enhancing software engineering security and quality. A question that warrants further exploration is the value of adopting a "purple" approach to secure programming training. Additionally, understanding how SECURECODER influences student performance in a classroom setting will provide valuable insights into its potential as a practical educational aid.

VII. CONCLUSIONS

This paper has introduced the design of a drill-and-practice platform for training secure programming with attack-and-defend exercises. Our prototype, SECURECODER, is a web-based application supported by a back end capable of hosting and executing various secure programming exercises. Findings from a pilot study suggest that SECURECODER holds great potential as an effective teaching aid for fostering secure programming skills through active learning. By presenting SECURECODER's novel concepts and design, this work serves as a foundation for developing other educational technologies aimed at helping students master security concepts and skills, preparing them to address the security challenges modern software engineers face.

REFERENCES

[1] MITRE. (2024) CVE Website. [Online]. Available: https://cve.org

[2] CISA, "Shifting the balance of cybersecurity risk: Principles and approaches for secure by design software," 2023. [Online]. Available: https://cisa.gov/sites/default/files/2023-10/SecureByDesign_1025_508c.pdf

[3] K. Nance, B. Hay, and M. Bishop, "Secure coding education: Are we making progress?" in *Proceedings of the 16th Colloquium for Information Systems Security Education*. Tampa, FL, USA: CISSE, 2012.

[4] J. Zorabedian, "Veracode survey research identifies cybersecurity skills gap causes and cures," 2017. [Online]. Available: https://veracode.com/blog/security-news/veracode-survey-research-identifies-cybersecurity-skills-gap-causes-and-cures

[5] A. Sanders, G. S. Walia, and A. Allen, "Assessing common software vulnerabilities in undergraduate computer science assignments," in *Journal of The Colloquium for Information Systems Security Education*, vol. 11, no. 1, 2024, pp. 8–8.

[6] J. Lam, E. Fang, M. Almansoori, R. Chatterjee, and A. G. Soosai Raj, "Identifying gaps in the secure programming knowledge and skills of students," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2022, pp. 703–709.

[7] C. C. Bonwell and J. A. Eison, *Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports*. Washington, DC, USA: George Washington University, Graduate School of Education & Human Development, 1991.

[8] P. M. Ortega-Chasi, "Computer science undergraduate students' perspectives on deliberate practice in introductory programming courses," Ph.D. dissertation, State University of New York at Buffalo, 2019.

[9] C. S. Lim, K. N. Tang, and L. K. Kor, *Drill and practice in learning (and beyond)*. Boston, MA, USA: Springer, 2012, pp. 1040–1043.

[10] Duoulingo. (2024). [Online]. Available: https://duolingo.com

[11] Khan Academy. (2024). [Online]. Available: https://khanacademy.org

[12] S. H. Edwards and K. P. Murali, "CodeWorkout: Short programming exercises with built-in data collection," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2017, p. 188–193.

[13] M. Brito and C. Gonçalves, "Codeflex: A web-based platform for competitive programming," in *Proceedings of the 14th Iberian Conference on Information Systems and Technologies*. IEEE, 2019, pp. 1–6.

[14] HackerRank - Online Coding Tests and Technical Interviews. (2024). [Online]. Available: https://hackerrank.com

[15] Codewars - Achieve Mastery through Coding Practice and Developer Mentorship. (2024). [Online]. Available: https://codewars.com

[16] C. Dale, "Red, blue and purple teams: Combining your security capabilities for the best outcome," *SANS Institute Information Security Reading Room*, 2019.

[17] MITRE. (2024) CWE - Common Weakness Enumeration. [Online]. Available: https://cwe.mitre.org

[18] D. Fisher and N. Frey, *Better learning through structured teaching: A framework for the gradual release of responsibility*. ASCD, 2021.

[19] J. Bergmann, *Flip your classroom: Reach every student in every class every day*. International Society for Teaching in Education, 2012.

[20] J. H. Berssanette and A. C. de Francisco, "Active learning in the context of the teaching/learning of computer programming: A systematic review," *Journal of Information Technology Education. Research*, vol. 20, pp. 201–220, 2021.

[21] M. I. C. Ribeiro and O. M. Passos, "A study on the active methodologies applied to teaching and learning process in the computing area," *IEEE Access*, vol. 8, pp. 219 083–219 097, 2020.

[22] M. Rathakrishnan, A. Raman, M. A. B. Haniffa, S. D. Mariamdaran, and A. B. Haron, "The drill and practice application in teaching science for lower secondary students," *International Journal of Education, Psychology and Counseling*, vol. 3, no. 7, pp. 100–108, Mar. 2018.

[23] LeetCode - The World's Leading Online Programming Learning Platform. (2024). [Online]. Available: https://leetcode.com

[24] J. C. Burguillo, "Using game theory and competition-based learning to stimulate student motivation and performance," *Computers & Education*, vol. 55, no. 2, pp. 566–575, 2010.

[25] J. Spacco, P. Denny, B. Richards, D. Babcock, D. Hovemeyer, J. Moscola, and R. Duvall, "Analyzing student work patterns using programming exercise data," in *Proceedings of the 46th ACM Technical*

Symposium on Computer Science Education. New York, NY, USA: ACM, 2015, pp. 18–23.

[26] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online judge systems and their applications," *ACM Computing Surveys*, vol. 51, no. 1, pp. 1–34, 2018.

[27] C. Java. (2024). [Online]. Available: https://codingbat.com

[28] M. J. Hull, D. Powell, and E. Klein, "Infandango: automated grading for student programming," in *Proceedings of the 16th annual Joint Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2011, p. 330.

[29] J. Walden and C. E. Frank, "Secure software engineering teaching modules," in *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development*. New York, NY, USA: Association for Computing Machinery, 2006, p. 19–23.

[30] B. Taylor and S. Azadegan, "Threading secure coding principles and risk analysis into the undergraduate computer science and information systems curriculum," in *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development*. New York, NY, USA: Association for Computing Machinery, 2006, p. 24–29.

[31] B. Taylor and S. Azadegan, "Moving beyond security tracks: integrating security in cs0 and cs1," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2008, p. 320–324.

[32] V. Mdunyelwa, L. Futcher, and J. Van Niekerk, "An investigation into educational process models for teaching secure programming," in *Proceedings of the International Symposium on Human Aspects of Information Security and Assurance*. Springer, 2022, pp. 77–90.

[33] M. Zeng and F. Zhu, "Secure coding in five steps," *Journal of Cybersecurity Education, Research and Practice*, vol. 2021, no. 1, p. 5, 2021.

[34] M. Tabassum, S. Watson, B. Chu, and H. R. Lipford, "Evaluating two methods for integrating secure programming education," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2018, pp. 390–395.

[35] T. Balon and I. Baggili, "Cybercompetitions: A survey of competitions, tools, and systems to support cybersecurity education," *Education and Information Technologies*, vol. 28, no. 9, pp. 11 759–11 791, 2023.

[36] T. Xie, J. Bishop, N. Tillmann, and J. De Halleux, "Gamifying software security education and training via secure coding duels in code hunt," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*. New York, NY, USA: ACM, 2015, pp. 1–2.

[37] J. Zhu, H. R. Lipford, and B. Chu, "Interactive support for secure programming education," in *Proceeding of the 44th ACM technical symposium on Computer science education*. New York, NY, USA: ACM, 2013, pp. 687–692.

[38] M. Whitney, H. Lipford-Richter, B. Chu, and J. Zhu, "Embedding secure coding instruction into the IDE: A field study in an advanced CS course," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2015, pp. 60–65.

[39] H. Z. Došilović and I. Mekterović, "Robust and scalable online code execution system," in *Proceedings of the 43rd International Convention on Information, Communication and Electronic Technology*. IEEE, 2020, pp. 1627–1632.

[40] M. Mareš and B. Blackham, "A new contest sandbox," *Olympiads in Informatics*, vol. 6, pp. 100–109, 2012.

[41] M. Mareš, "Security of grading systems," *Olympiads in Informatics*, vol. 15, pp. 37–52, 2021.

[42] D. L. Burley and M. Bishop, "Summit on education in secure software final report," UC Davis College of Engineering, Davis, CA, USA, Tech. Rep., 2011.

[43] D. F. Polit and C. T. Beck, "Generalization in quantitative and qualitative research: Myths and strategies," *International Journal of Nursing Studies*, vol. 47, no. 11, pp. 1451–1458, 2010.