Scalable Undergraduate Cybersecurity Curriculum Through Auto-graded E-Learning Labs

Aspen Olmsted

1 Introduction

Demand for cybersecurity workers is estimated to increase 350% between 2013 and 2021 [1]. In response, large universities have created online cybersecurity graduate programs with reduced tuition to attract adult learners. New York University (NYU) established a scholarship program called Cyber Fellows that provides a 75% scholarship to all US eligible workers [2]. Georgia Institute of Technology (Georgia Tech) has created an online MS degree that can be earned at a cost of fewer than 10,000 dollars [3]. Both of these programs are designed to scale to thousands of students. Fisher College [4] is a small minority-serving private liberal arts college located in downtown Boston, MA. At Fisher College, we have designed an undergraduate program designed to serve our students online with not only scalability but also integrity.

Computer science, like many of the sciences, devotes a great deal of the students' time in the learning to hands-on lab activities. Cybersecurity is a sub-discipline within computer science that combines core application, programming, and database courses with upper-level information technology, computer science, and cybersecurity course. In programming courses, these labs take the forum of the students writing application code in the language of the course. In database courses, the students are often submitting SQL queries in response to question prompts. In information technology and cybersecurity courses, the labs are often steps taken on real systems to configure systems or to eliminate vulnerabilities.

In face-to-face classes, instructors often use reverse classrooms so the students can have hands-on time with the instructor or a teaching assistant (TA), so when

A. Olmsted (✉)
Fisher College, Department of Computer Science, Boston, MA, USA
e-mail: aolmsted@fisher.edu

© Springer Nature Switzerland AG 2021

H. R. Arabnia et al. (eds.), *Advances in Software Engineering, Education, and e-Learning*, Transactions on Computational Science and Computational Intelligence, https://doi.org/10.1007/978-3-030-70873-3_59

they get stuck, they can get started again quickly without a long duration between submissions. The students watch lectures, read and take quizzes at home, and work on the labs in person to facilitate the just-in-time assistance. We show that students who learn with uninterrupted time do better in the completion of the labs.

For students in online classes, there is often a long time between a question and submission and the response or feedback that allows the student to continue learning in the lab. Online auto-graded systems help in the sense that the student will get some feedback immediately, but the student may have to wait for online office hours or a response to a forum post to continue with work. There is also a problem of ensuring integrity that the student submitting the work is the student who did the activities in the lab.

In this paper, we describe a technique we use in developing auto-graders that allows the student to receive feedback quicker while improving the integrity that the submitter is the author of the lab. The feedback comes in the form of auto-grader unit test results, along with allowing for peer discussions around the assignments. The number and quality of peer discussions increased because, in some cases, each student has a unique derivate of the lab that the students are completing. So instead of trying to stop student peer communication about lab solutions, we can encourage student sharing. The peer-to-peer student sharing has increased the students' understanding of the lab.

The organization of the paper is as follows. Section 2 describes the related work and the limitations of current methods. In Sect. 3, we describe the elements in the secBIML programming language. Section 4 explains the auto-graders we developed for our database courses. Section 5 describes how we developed our auto-graders for programming courses. Section 6 drills into the way we auto-grade for information technology courses. Section 7 investigates the way we build auto-graders for upper-level computer science courses. In Sect. 8, we drill into the auto-graders in our cybersecurity upper-level courses. Section 9 looks are our research questions and preliminary empirical data. We conclude in Sect. 10 and discuss future work.

2 Related Work

Jeffrey Ulman [2] developed an e-learning system with derivate questions called Gradiance Online Accelerated Learning (GOAL). GOAL provided quizzes and labs for several core computer science topics, including operating systems, database design, compiler design, and computer science theory. Each course was linked to a textbook with several quizzes per chapter and sometimes a few labs. The examinations were composed of questions with separate pools of correct and incorrect answers. When a student takes an exam, they are presented with a multiple-choice quiz where one right answer and several wrong answers are displayed for the student to choose the correct answer. A standard configuration of the system was four correct answers and eight incorrect answers. This configuration yield 224

derivate questions per each original item in the quiz. We have used GOAL over the years in database courses and found the derivative quiz questions allowed the students to discuss the exam without giving away the answer. The derivatives also will enable an instructor to answer a single derivate of an item in an online lecture. Unfortunately, GOAL only proved derivatives in the quizzes and not in the labs. The labs provided by GOAL were auto-graded, giving students immediate feedback, but since all students were working on the same labs at home, it was hard to stop answer sharing. Our work here supplements the job done in GOAL by providing not only automated grading of labs but also derivate questions per student (Fig. 1).

McGraw Hill [3] produces a commercial e-learning product called SIMnet. SIMnet's ambition is to teach students the skills required in the utilization of the Microsoft Office suite. SIMnet provides auto-graded labs that grade the students' submissions of database, spreadsheet, presentation, and word processing software. Students can learn the skills through online lessons that present the tasks in both reading and video format. SIMnet does protect the integrity of each student's work by inserting a unique signature into the starter file that the students download. If a student tries to upload a file with a different student's signature, the system catches the integrity violation. Either the upload is rejected, or the instructor is notified, depending on the lab configuration. Unfortunately, the labs that the students perform are not differentiated between students, so nothing stops one student from copying the work in the other students' files. Our work here improves on the integrity of the students' submission by deriving a different problem per student so they cannot just copy the other student's work.

Gradescope [4] sells a commercial e-learning product that allows instructors to scan student paper-based assignments. The grading of the paper-based assignments can then be automated through the e-learning system. The scanning feature has driven many mathematics and science departments in universities to adopt the system. A relatively unknown function of the system is the auto-graded programming framework. Gradescope designed a system that allows a student to upload a file for an assignment. The system then spins up a Docker [5] Linux session that is configured for the task. Test cases are developed in the auto-grader configuration with specific grading weights assigned for each test. We utilize this auto-grading environment for our derivative-based SQL, Python, and C#-based labs.

3 Database Auto-Grader

The auto-grader we developed for the database courses creates a Docker environment with a MySQL database running on a Linux environment. The students upload their query with a specific name: query.sql. The auto-grader then reads the metadata about the assignment to determine the assignment name and performs between three and five tests. Each test is weighted at two points each.

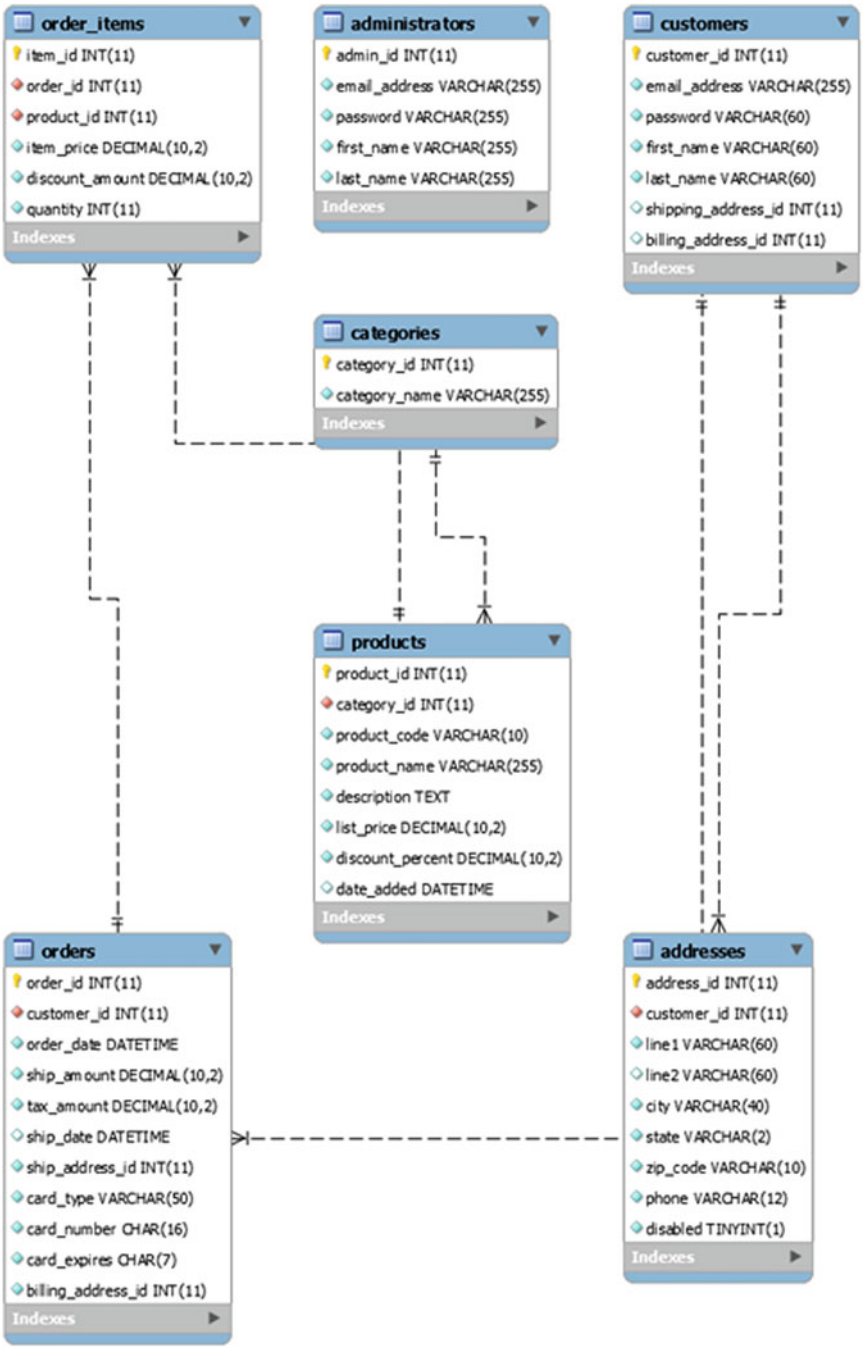

Fig. 1 Student lab ER diagram

Table 1 Sample value test

Field	Value
Name	Assignment 1
Value 1	Product_code
Value 2	Prodcut_name
Value 3	List_price
Order	List_price
Derivative	Random row

The test is actually stored in the MySQL database that is installed in the Docker session. The database connected has both the test information for the auto-graders and the same data provided to the students for their lab. There are two types of unit tests:

- Value tests
- Existence tests

For the value tests, each assignment has a row in the value_unit_test table. Table 1 shows a sample assignment row for a query that returns a specific product record. The primary key for the value_unit_test table is the name of the assignment. The value_unit_test also contains three value tests, along with an order test. The last value in the value_unit_test is the derivate method. Currently, supported derivate methods are:

- Random row – In this derivate method, the student's login is converted to a unique number between 1 and the maximum assignment number. The unique number comes from an order the student ID comes in the roster. The system will read the specific record in the order by value from the database to prompt the student to return that record.
- Random range – This is similar to random row but asks the students to return a tuple between a start and end value. The start value is the same as the random row value, and the end value is the fifth value after that record

Figure 3 shows the entity-relationship (ER) diagram for the student lab database. An assignment description website was built to display the question values that match the auto-grader tests for the student logged in. The auto-grader will score the student on ten possible points distribute across five tests:

1. Did the query execute?
2. Did the values match for the first value?
3. Did the values match for the second value?
4. Did the values match for the third value?
5. Did the order match?

Existence tests are similar to value tests, except they are used to grade queries that mutate the database such as insert, update, and delete statements along with queries that create views, functions, stored procedures, and triggers. In the case of existence

tests, the auto-grader will score the student on six possible points distributed across five tests:

1. Did the query execute?
2. Did test 1 pass?
3. Did test 2 pass?

Table 2 shows an example of an entry for the existence_unit_test table. The case is from an assignment where the student needs to write a query to create a new index. The test types either exist or do not exist. The test will either pass or fail if there is a value returned from the query. For an existing unit test type, data should be returned for a success. For not existing unit test type, no data should be returned for a successful test. Instead of a derivative based on a specific record as we used in the value unit tests, replacement variables are used to change the queries. Table 3 shows the available replacement variables. The variables allow names to be based on the user logged in, tables, and columns to be different for each student and literal string and numbers to be randomized.

Table 2 Sample existence test

Field	Value
Name	Assignment 2
Test 1	Show index from @RandomTable where key_name = @login_orders_ix
Test 1 type	Exists
Test 2	Show index from @RandomTable where key_name = @login_orders_ix and column_name = '@RandomColumn
Test 2 type	Exists

Table 3 Replacement variables

Variable	Meaning
@login	The user code for the logged in user
@RandomTable	A random table from the students sample database. This variable can be suffixed with a number between 1 and 9
@RandomColumn	A random column from the random table selected in the variable above. This variable can be suffixed with a number between 1 and 9
@RandomWord	A random word from the dictionary
@RandomInt	A random possitive integer

4 Programming Auto-Grader

We had previously developed a set of auto-graded foundational programming assignments in Python, Java, and CSharp for students in a first programming class. Unfortunately, many of these assignments did not lend themselves to derivates that required different solutions per student. In our first attempt, we randomized the variables in the test cases to ensure students were not hard coding the output to match the input tests. To illustrate the challenge, we will itemize the labs below:

- Labs to practice programming expressions:
- Hello World – In this assignment, the student just outputs the words "Hello World."
- Coin Counter – In this assignment, the student would be sent input variables for the number of quarters, dimes, nickels, and pennies and would output the total in dollars and cents.
- Coin Converter – In this assignment, the students would be given dollars and cents, and they would output the minimum number of coins by denomination.
- BMI Metric – In this assignment, the student would sent input of weight in kilograms and height in meters, and they would output the BMI.
- BMI Imperial - In this assignment, the student would be sent input of weight in pounds and height in inches, and they would output the BMI. The students would need to convert the imperial measurements to metric before calculating the BMI.
- BMI Metric with Status – This assignment is a modification of the earlier assignment and adds a decision branch to display a status of underweight, normal, overweight, or obese. The students have not learned decision branching yet, so the expectation is they will use modular division for this problem.
- Labs to practice programming iteration and decision branching:
- Cash Register – This assignment allows multiple inputs of item prices along with a club discount card and tax rate. The student outputs the base price, price after discount, and total price.
- Call Cost – This assignment provides the students with a rate table based on the day of the week and time of day. Input is sent with the day, time, and duration of the call and the students' outputs the total cost for the request.
- Even Numbers – This assignment has the student output a certain number of event numbers based on the number input.
- Fibonacci – This assignment has the student produce the first n Fibonacci numbers. The number n is sent as input to the program.
- Labs to practice programming string operations:
- String Splitter – This assignment tests the student's ability to divide up an odd length input string into middle character, string up to the middle character, starting after the middle character
- Character Type – This assignment has the student read a character of input and classify it into a lowercase letter, uppercase letter, digit, or non-alphanumeric character.
- Labs to practice programming functions:

- Leap Year Function – This assignment has the student write a function that takes a parameter and return true if the year is a leap year
- First Word Function – This assignment sends a sentence as a parameter to a function the student writes, and the student returns the first word of the sentence.
- Remaining Word Function – This assignment sends a sentence as a parameter to a function the student writes, and the student returns the remaining words after the first word of the sentence.
- Labs to practice programming lists:
- Max in List Function – This assignment sends a list of integers as a parameter to a function the student writes, and the function should return the largest integer in the list.
- Max Absolute in List Function – This assignment sends a list of integers as a parameter to a function the student writes, and the function should return the largest absolute value of each integer in the list.
- Average in List Function – This assignment sends a list of integers as a parameter to a function the student writes, and the function should return the average of all the integers in the list.

4.1 Derivates of Expression Labs

We modified the above labs that allow students to practice programming expressions so that each student received a derivative. Each of these labs initially provided the student with a formula or included an inherent method. So, for example, the Metric BMI lab provided students with the formula to calculate BMI by taking the weight in kilograms and dividing by the height in meters squared. The currency-based labs used an inherent method for converting the value of each coin. For example, a nickel is worth five pennies. All of these labs were modified by adding fake names for calculations and currencies and applying random constants and exponents. For example, one student would calculate the BMIA by using the formula of three times weight divided by two times height raised to the fourth power.

4.2 Derivates of Advanced Labs

After tackling the expression labs, we looked at the advanced programming labs listed above. None of these labs lend themselves to derivates in the problem statement. So we focused on ways to randomize the values in the unit tests to ensure integrity that the student was writing code that did not hard code output based on the inputs they saw. Each time a student submits their work, the test uses different input values that are randomly generated.

5 IT Course Auto-Graders

5.1 Helpdesk Course Auto-graders

In a helpdesk course, students learn technical problem-solving skills so they can solve end-user IT problems. We developed deployed through Docker sessions with problems and recipe-type instructions for the students to perform to solve the technical issues. We utilize the Linux Bash history file to auto-grade the student's work to ensure they executed all the commands in the recipe.

5.2 Networking Admin Course Auto-graders

Similar to the helpdesk course, the networking admin course teaches the student the core competency around network tools. We developed labs deployed through Docker sessions and provided recipe-type labs for the students to build familiarity with the networking tools. We utilize the Linux Bash history file to auto-grade the students' work to ensure they executed all the commands in the recipe.

6 Computer Science Course Auto-graders

6.1 Operating System Auto-graders

In an operating system course, students learn how operating systems manage limited hardware resources so that many application programs can run simultaneously. We developed auto-graders that allowed the students to explore the data structures and algorithms used to manage physical memory, virtual memory, hard disks, and the central processing unit (CPU).

6.2 Networking Programming Course Auto-graders

In a networking course, students learn about the Open Systems Interconnection (OSI) model and Transmission Control Protocol/Internet Protocol (TCP/IP) layers. The students write programs in Python that utilize TCP/IP services that talk to a cloud application.

7 Cybersecurity Science Course Auto-Graders

7.1 Information Security Auto-graders

In an information security course, students learn about threat modeling, security policy models, access control policies, and reference monitors. We developed a set of auto-graded reference monitor labs. In each lab, the student implements a reference monitor that implements different access control policies and security policies.

7.2 Secure Programming Auto-graders

In a secure programming course, students learn how to develop code free of vulnerabilities. The perspective in a secure programming course comes from the concept that the code is a white box. The students have full visibility of the source code as they perform labs to secure the code. We developed labs where students are provided code with vulnerabilities. Gradescope auto-graders are provided that exploit the vulnerabilities. Students need to improve the code and submit a version without the original vulnerability in their code to receive credit.

7.3 Penetration Testing Auto-graders

In a penetration course, students think about security from a different perspective. The perspective in a penetration testing course comes from the concept that the code is a black box. The students do not have visibility into the source code they are trying to penetrate in the labs. We developed labs where students are provided a signature for a code library with vulnerabilities. Gradescope auto-graders are equipped to execute the students' code and determine if they found a weakness.

8 Empirical Data

In this section, we examine the data we gathered from three sections of a database course. There were three questions we wanted to answer about our use of auto-graders in the cybersecurity curriculum:

- Do the auto-graders help students progress quicker through a lab?
- Do the auto-graders help increase participation at the undergraduate level in labs?
- Do the derivate auto-graders help students by facilitating peer discussion?

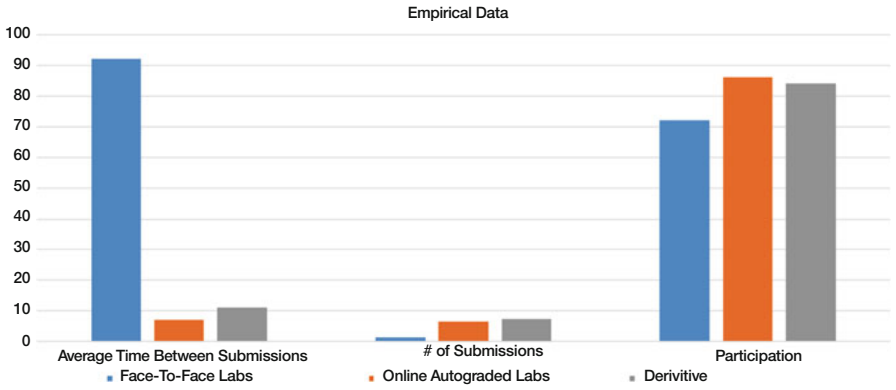

Fig. 2 Empirical data

We choose the database course because every lab had a derivate version so that each student was working on a unique problem in the lab. The original face-to-face section used manual graded lab submissions without derivatives, one online section used auto-graded nonderivative labs, and one section used the derivative labs. The students in the face-to-face section had a reverse classroom where they worked individually on labs during class time, and the instructor would answer questions as they ran into problems. In the section with the derivative labs, a discussion forum was provided for students to help each other with the lab.

Figure 2 shows a summary of the data we used to answer the questions. The average time between submissions was reduced significantly for the two sections that utilized auto-graders. The number of submissions was increased for the two sections that used auto-graders. Lastly, the participation rate was raised for the two sections that used auto-graders.

The answer to the three research statements was a strong yes to the first two and a weaker yes to the third question. The auto-graders helped online student progress quicker through the lab by shortening the time between submissions. In our small study, the auto-graders helped increase participation at the undergraduate level in both versions of the labs. Lastly, we believe the derivate auto-graders helped students by facilitating peer discussion. The participation rate was a little lower for the derivative version of the labs. Still, we felt it was close enough to the non-derivate lab to show progress in learning since students were performing unique work, and the increased student communication help to facilitate that progress.

9 Conclusions and Future Work

Based on our research, we demonstrate that the use of our e-learning auto-graded assignments improves participation in the cybersecurity course labs. We also show

that the use of our technique of creating derivatives of the lab for each student can lead to increased communication between students and therefore increased learning. Our future work will continue to develop labs in the advanced courses that not only randomize the unit test data but also provide for derivate problems per student. We will also gather more empirical evidence in the future to show how the auto-graders improve the learning experience for online e-learners.

References

1. S. Morgan, Cybersecurity talent crunch to create 3.5 million unfilled jobs globally by 2021, Cybercrime Magazine, 24 Oct 2019. [Online]. Available: https://cybersecurityventures.com/jobs/. Accessed 8 Apr 2020
2. NYU Tandon, NYU cyber fellows, 2020. [Online]. Available: https://engineering.nyu.edu/academics/programs/cybersecurity-ms-online/nyu-cyber-fellows. Accessed 8 Apr 2020
3. Georgia Institute of Technology, Online master of science in cybersecurity, 2019. [Online]. Available: https://info.pe.gatech.edu/oms-cybersecurity/?utm_source=cpc-google&utm_medium=paid&utm_campaign=omsc-search-converge-top5&gclid=CjwKCAjw7LX0BRBiEiwA__gNw2xT3-grxlfiyfYGdGDGIpZZSkf6_bIttgoipp830ue3le5MByUu0hoCGtoQAvD_BwE. Accessed 8 Apr 2020
4. Fisher College, Find the world at Fisher, 2020. [Online]. Available: www.fisher.edu. Accessed 8 Apr 2020
5. J.D. Ullman, Gradiance online accelerated learning, in *Proceedings of the 28th Australasian Conference on Computer Science*, (Newcastle, Australia, 2005)