Koutcheme, Charles; Tilanterä, Artturi; Peltonen, Aleksi; Hellas, Arto; Haaranen, Lassi

Exploring How Students Solve Open-ended Assignments

Published in:
ITiCSE 2022 - Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education

DOI:
10.1145/3502718.3524748

Published: 07/07/2022

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Koutcheme, C., Tilanterä, A., Peltonen, A., Hellas, A., & Haaranen, L. (2022). Exploring How Students Solve Open-ended Assignments: A Study of SQL Injection Attempts in a Cybersecurity Course. In *ITiCSE 2022 - Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education* (pp. 75-81). (Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE; Vol. 1). ACM. https://doi.org/10.1145/3502718.3524748

Exploring How Students Solve Open-ended Assignments: A Study of SQL Injection Attempts in a Cybersecurity Course

Charles Koutcheme
charles.koutcheme@aalto.fi
Aalto University
Espoo, Finland

Artturi Tilanterä
artturi.tilantera@aalto.fi
Aalto University
Espoo, Finland

Aleksi Peltonen
aleksi.peltonen@aalto.fi
Aalto University
Espoo, Finland

Arto Hellas
arto.hellas@aalto.fi
Aalto University
Espoo, Finland

Lassi Haaranen
lassi.haaranen@aalto.fi
Aalto University
Espoo, Finland

ABSTRACT

Research into computing and learning how to program has been ongoing for decades. Commonly, this research has been focused on novice learners and the difficulties they encounter, especially during CS1. Cybersecurity is a critical aspect in computing – as a topic in university education as well as a core skill in the industry. In this study, we investigate how students solve open-ended assignments on a cybersecurity course offered to university students after two years of CS studies. Specifically, we looked at how students perform SQL injection attacks on an web application system, and study to what extent we can characterize the process in which they come up with successful injections. Our results show that there are distinguishable strategies used by individual students who seek to hack the system, where these approaches revolve around exploration and exploitation tactics. We also find evidence of learning due to a more pronounced use of exploitation in a subsequent similar assignment.

CCS CONCEPTS

• **Security and privacy → Database and storage security**; **Web application security**; • **Applied computing → Education**.

KEYWORDS

SQL injection, database security, education, problem solving

ACM Reference Format:
Charles Koutcheme, Artturi Tilanterä, Aleksi Peltonen, Arto Hellas, and Lassi Haaranen. 2022. Exploring How Students Solve Open-ended Assignments: A Study of SQL Injection Attempts in a Cybersecurity Course. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3502718.3524748

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland

ACM ISBN 978-1-4503-9201-3/22/07…$15.00
https://doi.org/10.1145/3502718.3524748

1 INTRODUCTION

Everything is online, everyone is connected, and many if not most of our interactions are mediated over technological appliances. Although the first hack – piggybacking communications – happened nearly two centuries ago [40, p. 39], and despite the demand of workforce competent in cybersecurity (e.g. [13, 19]), cybersecurity has been only recently noted as an emerging focus area for education [17]: the first set of global cybersecurity curricular guidelines were released in 2017 [31], providing an outline of the core body of knowledge that such programs should provide.

Cybersecurity training prepares students to identify and fix weaknesses in a variety of systems, support the design of secure systems, and to help others become more aware of the problems that insecure systems may lead to [17, 31]. This, in combination with strong background in software engineering, has the potential to alleviate the prevalent issue in the industry, where the number of security flaws identified in software seems to not show a decreasing trend (e.g. [6]). Our present study takes place in a context, where students already possess some competence in software engineering as well as the basic understanding of database systems and the competence to write SQL queries.

In the present work, our overall goal is to explore how students approach open-ended problems in a cybersecurity course, which is a part of the computer science curriculum offered by Aalto University. In our case, we study how students seek to gain sensitive data from a web application using SQL injections, where the system under scrutiny does not always sanitize inputs that are used as a part of database queries. To answer the research question *how do students perform SQL injection attacks*, we analyze their sequences of injection attempts and we characterize the process in which they come up with successful solutions. In essence, we are looking for *tactics* that outline what students do. Based on these tactics we divide students into broader *strategy* groups which reflect how they used these different tactics.

The closest matches to our work are (1) studies on students learning to write SQL queries (e.g. [4, 11, 45]) and (2) studies that explore how students solve (often programming) problems (e.g. [22, 29, 36]). At the same time, in the present study, the problem is ill-defined in that the students do not explicitly know how to reach a solution.

This article is organized as follows. Subsequently, in Section 2, we outline related work in solving problems, learning and using

SQL, and cybersecurity education. In Section 3, we outline our study methodology, including the study context and data, and the research approach. Section 4 outlines the results of our analyses, which are discussed in the context of computer science education and cybersecurity education research in Section 5. Finally, Section 6 summarizes our work.

2 RELATED WORK

2.1 Solving problems

The ability to solve problems depends on a number of characteristics, where skills, expertise, experience level, knowledge, and the problems themselves all contribute to how particular problems are solved. In the context of programming, novices lack detailed mental models and fail to apply knowledge, while experts understand the bigger picture [28, 48]. In understanding code, experts often use a top-down approach, where they follow the control flow of the program, while novices often utilize a bottom-up approach, where they read the code line by line [7, 30]. There are also considerable differences between novices – a classic example in computing education research comes from Perkins et al. [34], who categorized novice programmers as stoppers, movers and tinkerers based on how they sought to solve a problem. Stoppers tended to give up, movers tried to solve the problem through exploration of options, and tinkerers often made small and at times even seemingly random changes when hoping to solve the problem.

An integral part of problem solving are schemas that refer to the (human) memory structure that represent categories of information and the connections between these categories [20]. When acting, also when solving problems, schemas for the situation are identified and retrieved from memory. If no corresponding schemas exist, a person applies general problem-solving strategies, such as *seeking* to identify smaller steps needed to solve the problem, and then *solving* those steps [42]. Due to such actions and new information, over time, new information is incorporated into existing schemas and new schemas are created [35]. Overall, becoming better at problem-solving is linked with incorporating domain- and problem-specific information into new and existing schemas [43].

Schemas are also linked with learning to program [14, 15, 38]. When faced with a programming problem, if an existing schema to a problem exists, the solution is written in a linear manner [14, 38]. When no such schema exists, experimentation is used instead – this experimentation then in part guides the learning process where new information is incorporated to existing schemas and new schemas are formed. This exploration process is effectively also what happens when the problem is not familiar, although we note that the structure of the exploration process may differ between individuals.

2.2 Learning and using SQL

Formulating SQL queries can be challenging for novices. Students make different types of errors and mistakes while they are learning SQL [4, 5, 11, 44, 45]. The ability to write SQL has also been linked to students' overall academic performance [3]. In order to to successfully solve tasks with SQL, the student also requires knowledge regarding the domain. Two factors contribute to the challenge of solving SQL-related database task: how the problem is phrased, and what is the structure of the database schema [8, 10, 12, 27]. To aid in teaching SQL, multiple interactive tools have been developed, such as tutoring systems (providing scaffolding to learning) and automatically assessing student solutions to exercises[1, 16, 23, 25, 26, 39]. More recently, this work has been extended to provide learning platforms designed to practice SQL injections [9, 37].

2.3 Cybersecurity education

A recent literature survey [41] reviewed publications between 2010-2019 in ACM SIGCSE & ACM ITiCSE related to cybersecurity and computing education research. They found 71 relevant publications on cybersecurity discussing different knowledge areas defined by JTF Cybersecurity Curriculum[1] with data security being the most prevalent with 29 publications. Majority of the studies focused on university education with few papers investigating K-12 or professional education. Finally, based on the small number of citations, they noted that the area of cybersecurity in computing education research seems to be fragmented.

In a 2018 ITiCSE working group Parrish et al. [33] proposed that cybersecurity should be viewed as a meta-discipline with its own competency model. This would more closely match the increasing need of security experts in the industry as well as provide a framework for cybersecurity education. To achieve this, they proposed two complementary approaches: (1) integrate traditional CS programs with cybersecurity content and (2) develop new programs focused on cybersecurity.

The amount of published research related to SQL injections in education seems to be very limited. This fact is also reflected in the textbooks used for undergraduate database education. Taylor & Sakharkar reviewed seven database textbooks and only two of them included discussion of SQL injection – five of the books did not discuss the topic at all [46].

Yuan et al. [49] compared two approaches of teaching SQL injections in a hands-on lab environment. As a control group, they had a system mimicking a real web service with multiple vulnerabilities that the students needed to exploit with SQL injections. No step-by-step instructions were provided and the problem was very open-ended, similarly to our present work. The experimental group used OWASP WebGoat SQL injection lesson to learn about the topic. Overall, they found no statistically significant differences between the two groups in learning outcomes, motivation, or students' experience in the labs.

3 METHODOLOGY

3.1 Context and data

Our data is collected from a cybersecurity course offered by Aalto University. It is a 5 ECTS[2] course offered to both bachelor and master students with at least two years of previous computer science studies. The prerequisites of the course include strong programming skills and a broad knowledge of computer science concept. Hence, the majority of the participants have traditionally been computer science majors. However, the increased popularity of cybersecurity has in recent years increased the number of participants from other

[1] http://cybered.acm.org/
[2] European Credit Transfer and Accumulation System, 1 ECTS corresponds to approximately 27 hours of work

programs as well. The course aims to teach key concepts and abstractions of cybersecurity, and to give students practical hands-on experience in threat analysis and vulnerability exploitation. The focus of the course is on how to identify and mitigate common vulnerabilities, rather than teaching hacking or Capture the Flag (CTF) skills.

In addition to the theoretical background presented during the lectures, the course consists of weekly hands-on exercises in topics such as software security, access control, and web security. The assignments are done on a custom web platform, which for each topic introduces new features with vulnerabilities related to it. The exercises are designed to resemble realistic scenarios: students do not know the specifics of the back-end system and the feedback provided by the platform is limited. The main task for the students is to identify the weakness for each feature and figure out how to exploit it. Each weekly assignment consists of three types of exercises: (1) basic exercises, (2) advanced exercises, and (3) bonus challenges. Exercises of type one can typically be solved using generic approaches and limited knowledge of the topic, whereas type two exercises often require applying skills from different topics and performing multi-step attacks. Type three exercises are optional bonus challenges, which require a more complicated attack design and often require writing supporting software.

In this study, we focus on analyzing how students approach solving exercises involving SQL. Students are asked to exploit a flaw in the search functionality of a website, which instead of prepared statements uses string concatenation to construct a database query from a given input. They are given the following Node.js code snippet to explain how search results are handled by the back end of the website:

```
var sql_query = 'SELECT uid, name FROM table WHERE
  uid = ${user_id} AND name LIKE "%${search_term}%";';
```

The variable `user_id` contains the identifier of the logged in user account, and the input of the search bar is stored in the variable `search_term`. The assignment consists of three exercises. The first task (Exercise A) is to inject an SQL query that returns data belonging to another user (i.e. where the `user_id` is different). The second task (Exercise B) is to retrieve user information and password hashes from another table. Finally, the third task (Exercise C) is to crack some of the stolen passwords.

For the first two parts, the students are given some basic information about the system, including the software used to implement the database and the previously explained query. Using this information, they are asked to craft malicious inputs to the website that return more information about the database and its content. Since the only user input to the query is stored in the `search_term` variable, the main challenge of Exercise A is to exploit the fact that the query is not properly sanitized by escaping the string and modifying the intended query. Exercise B, on the other hand, requires understanding the basic structure of a database with multiple tables, and once again exploiting the query in order to retrieve the required information. Since Exercise C is mainly done offline, we consider it to be out of scope for this study.

3.2 Approach

We analyze the results and solving attempts of over 300 students. Our logs contain information about the submitted answers, as well as all queries executed in the website's search bar. Our goal is to characterize how students come up with successful SQL injections. We adopt a bottom-up approach, starting from the sequences of all queries submitted by each student for solving the exercises. To simplify the analysis of these sequences, we cluster the queries in the sequence of each student separately. From the resulting clustering, we group students based on two characteristics of the simplified processes: high-level statistics and transition behaviors across clustered queries. In the following sections, we detail and justify the components of our approach.

3.2.1 Preprocessing. We perform preprocessing to ease the subsequent analysis. We only select students who submitted a correct solution to Exercises A and B. We also remove students with an exceptionally high number of submitted queries (more than 275). We believe these students might have done automatic testing of solutions. We further remove from the original sequences the queries written for solving Exercise C.

3.2.2 Query clustering. We identify groups of syntactically similar queries in the sequence of each student by clustering each student's queries. This per-student clustering (compared to a general class level clustering) has the advantage that it captures individual patterns in how each student writes their queries. We use the affinity propagation clustering algorithm [18]. This algorithm works similarly to the more well-known k-means. It assumes that a representative element can represent each identified cluster. The algorithm automatically selects an ideal number of clusters suited for the data. It works particularly well for a relatively small number of elements. We ran the algorithm by computing the normalized Levenshtein distance [32] between all pairs of normalized queries in the student's query sequence. We normalize each query in the sequence into lower case, and we remove extra spaces. We hypothesized that queries belonging to the same query group would often be close to each other in the ordered sequence. We then assign each query in the ordered sequence to the query group/cluster it belongs to, resulting in a sequence of cluster assignment. In the rest of our analysis, we take a look at these cluster assignment sequences.

3.2.3 Statistical analysis. First, we took a high-level look at the cluster assignment sequences. We computed the following frequency based statistics for each student:

n_queries Total number of queries $\in \mathbb{N}$ for both exercises
n_queries_A Total number of queries $\in \mathbb{N}$ for Exercise A
n_queries_B Total number of queries $\in \mathbb{N}$ for Exercise B
n_groups Number of query groups $\in \mathbb{N}$
average_elements Average number of queries within each query group $\in \mathbb{R}$

We also looked at the type of transitions students perform. We walked the sequence of cluster assignments, in order, while keeping track of the ids of the encountered query groups, and we computed the following statistics:

same_type Number of times the student transitioned to a query belonging to the same group $\in \mathbb{N}$

known_type Number of times the student transitioned to a query belonging to a group tried previously $\in \mathbb{N}$

new_type Number of times the student transitioned to a query belonging to a group never tried before $\in \mathbb{N}$

Grouping students. In the end, using the computed statistic, we identify groups of students by clustering them using k-means algorithm in the $\mathbb{R}^8$ feature space. We select the number of clusters that maximizes the clustering assignment's silhouette score.

3.2.4 Transition analysis. Second, we are also interested in discovering which tactics students use to transition between query group (identified by the earlier query clustering). We model students' transition tactics using a Hidden Markov Model [2] (HMM). Under this model, the student's tactic (which is unobserved) at each transition is defined by the (hidden) state of the model at the current time step. At the same time, the emitted variable corresponds to the observed choice of transitioning to a query that belongs or does not belong to the same group. For each student, we create a sequence Y of transitions between query groups. We set $Y_t = 0$ when, at the t transition, the student transitions to a query belonging to the same group (same_type transitions). We set $Y_t = 1$ when the student transitions to a query that belongs to another group (new_type or known_type transitions). Since we are interested in discovering common tactics across students, we create one HMM for the whole class by fitting the model with all sequences Y of students. Although we fit a single model on all sequences, the HMM results in an assignment for each student, for each transition, to a state that defines the student's strategy. We select the number of states of the model (i.e., the number of strategies) by fitting ten models with a number of states between 1 and 10. We select the model which minimizes the BIC score $BIC = -2L + p\log(T)$, where L is the logarithmic likelihood of the model, T is the number of sequences fit, and p is the number of states (i.e., strategies) of the model. The literature currently uses the BIC score to select the best explanatory model.

Grouping students. Finally, we aim to identify problem solving strategies based on students usage of the identified transition tactics. We manually define a strategy depending on how much time (i.e. how many transitions) each student spent in each tactic mode.

3.2.5 Example. We illustrate our transformation process. Let us suppose that a student submitted the following toy queries:

```
1  "1"="1";
2  name OR 1=1;
3  SELECT * FROM table --;
4  "1=1 --";
5  SELECT * FROM users --;
6  SELECT password FROM users WHERE uid=1; --;
```

Figure 1 depicts the whole transformation process from the original sequence (above) to the clustering assignment and the transition sequence. The clustering algorithm regrouped the following queries together: [q1-q2-q4], [q3-q5], and [q6]. Queries in the same cluster have a similar syntactic form, although they differ slightly in content. We then map each query to the id of the cluster it was assigned to. We use this cluster assignment sequence for our statistical and transition analyses.

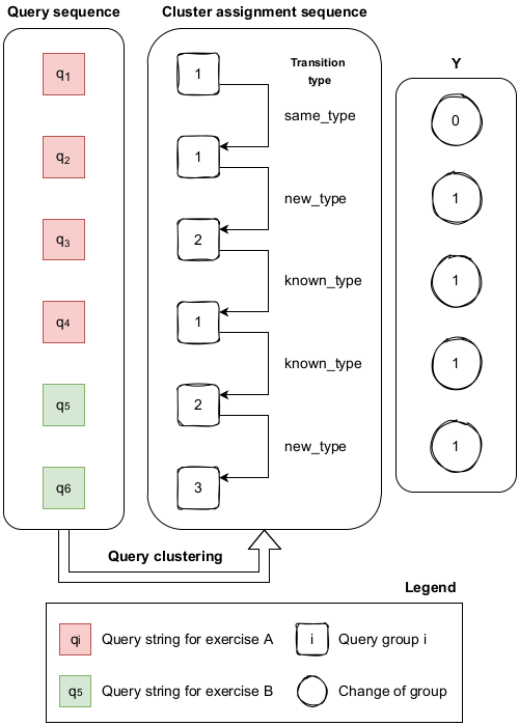

Figure 1: Illustration of the transformation process on the given toy queries. We transformed the original sequence of queries into a sequence of cluster assignments. We also create a chain Y depicting the transition events from one query cluster to another.

4 RESULTS

In this section, we outline the main results of our analysis. Our data consists of logs of 312 students. In the preprocessing step, we excluded 33 students who did not complete all exercises, 8 students with a number of queries exceeding 275, and 1 student for which the clustering algorithm of the individual queries did not converge. In the rest of the analysis, there were thus 270 students.

4.1 Statistical analysis

We identified two groups of students in the clustering based on the eight statistics. Table 1 shows the average value for the computed statistics for each group. We tried to characterize each group. As commonly done in clustering analysis, we performed a Principal Component Analysis (PCA) on the same dataset used for clustering to visualize the two groups in a lower dimensional space. We then noticed that the two groups vary mainly along the principal component (i.e. the principal axis), and we observed that the number of queries (n_queries) explained the majority of the variance of this principal component. Since the two groups vary mainly in the number of queries submitted, we considered them as "performance groups". Essentially, students who submit more queries can be considered less successful as they require more attempts to reach a solution. Based on this interpretation, there were 67 low performing students, and 203 high performing students.

statistic	high_performance	low_performance
n_queries	45.12	140.91
n_queries_A	20.40	63.81
n_queries_B	23.89	76.10
n_groups	7.30	16.61
average_elements	6.33	8.87
new_type	9.33	23.90
same_type	25.18	77.61
known_type	9.61	38.40

Table 1: Mean values of the computed statistics for the two clusters of students found. We interpreted these two clusters as performance groups.

4.2 Transition analysis

We discovered clear transition tactics based on the Hidden Markov Model. The best explanatory model is a simple Hidden Markov Model with two states. Figure 2 depicts this model and its main components.

4.2.1 Interpreting the two states. As commonly done in the literature, we interpreted the meaning of the two states based on the values of the emission probabilities. We noticed that when in the first state, students transition relatively often from one query type to another ($P(Y) = 1$ is high). However, when in the second state, students are more likely to submit continuously queries belonging to the same query group (i.e. $P(Y) = 1$ is low) . For these reasons, we named the first tactic "exploration" and the second tactic "exploitation".

4.2.2 Grouping students. We grouped students into three categories (or strategy groups) depending on how they used each tactic: whether they only used the exploration tactic (explore strategy), only used the exploitation tactic (exploit strategy), or used a mix of both tactics (hybrid strategy). We found that 29 students only exploited (we call them exploiters), 58 students only explored (we call them explorers), and 183 students used both tactics.

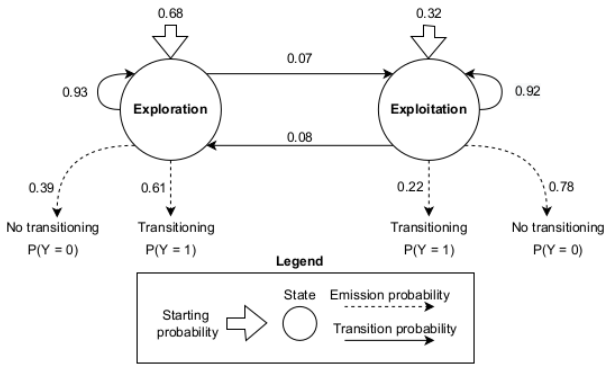

Figure 2: Hidden Markov Model of students transition tactics.

4.3 Links between performance and strategy

We investigated whether there were links between students' performance in the exercises and the strategies they employ for transitioning between query groups. Figure 3 shows the average number of students adopting each strategy per performance group as well as the average number of queries for each combinations. As a reminder, we hypothesized that the higher the number of queries, the lower the performance.

Exploiters and explorers. Interestingly, one of our first observations is that students who stick to only one tactic (exploiters and explorers) are more likely to be high performers. Indeed, there were no low-performance exploiters (low-performance students never used the exploitation strategy entirely) and only three low-performance explorers. When comparing the exploitation against the exploration tactic, we notice that high performers exploiters were slightly more performing (29.58 queries on average) than high performers explorers (32.34 queries on average).

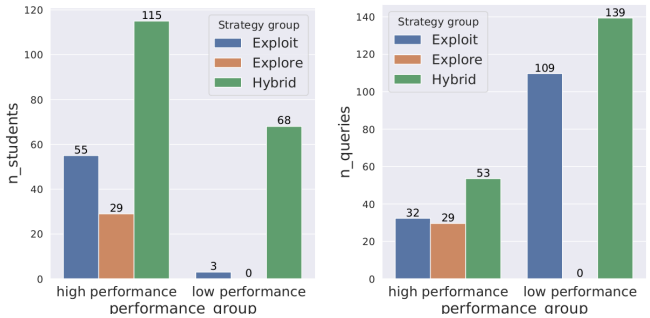

Figure 3: Interplay between performance group and strategy group. On the left, the average number of students who use each strategy for both performance group. On the right, the average number of queries submitted by students belonging to the different groups.

Students who use both tactics. We took a closer look at students who use both tactics. For this group of students (for this mixed strategy), Figure 4 shows their usage of the exploration tactic while highlighting differences across performance groups and exercises. We can observe that there are no interesting variations across performance groups. In general, our result confirms that students generally explored more than they exploited. However, we can observe interesting variations in tactic usage across exercises. Our results reveal that students who used the hybrid strategy switched from an exploration-focused tactic in Exercise A to a more balanced mix of tactics in Exercise B.

5 DISCUSSION

5.1 A model of novices hacking a system

Based on our results, we formulate the following model of how students attempt to hack a system using SQL injection. The process starts with the student constructing a candidate SQL injection string, which they then test against the system. If the SQL injection does not work (i.e. the solution is incorrect), the student will build a new

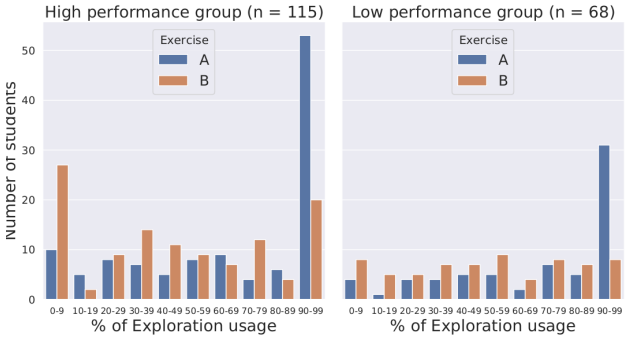

Figure 4: Distribution of exploration tactic in the mixed strategy. If a student spent y% of their time in a certain exercise using the exploration tactic, then the same student spent $(100 - y)$% of their time using the exploitation tactic for the same exercise.

candidate SQL injection string and then attempt to use that. This is in line with earlier research on how one acts when faced with an unfamiliar problem; in such a case, general problem-solving strategies such as trying out a range of options are applied [42, 43]. This behavior related to constructing a candidate SQL injection string is somewhat separate from the behavior that finally leads to a working solution. Finding an SQL injection string that works is more often related to local optimization of a query. Here, the student starts with a specific query type that they likely believe would work – finding this can be a product of trying out a a range of options – and then refine the query by trying out variations such as exploring with different column names and column counts. Interestingly, even this behavior is exploratory but the trajectory towards a solution is more explicit – moving to another type of query is more rare when on a path to solution.

Our model assumes that queries that are syntactically similar are close to each other in time. The model postulates that there are, contrary to more often studied programming processes with clear linear progress [14, 38], students who constantly explore what works and what does not work. We associate the "explore" mentality with the idea that students try out one type of query and then, instead of trying variations of the query, they move to another type of query. On the other hand, the "exploit" mentality refers to students trying out multiple variations of one type of a query, exploring with different column names and column counts. Both of these behaviors could be viewed as "tinkering" – small and at times even random changes when seeking to solve a problem – in terms of the research by Perkins et al. [34]. At the same time, upon manual analysis – confirmed in discussions with course assistants – we also observe that trying out multiple variations of a query often starts with a shorter query which is then incrementally built on. This has been previously also observed in programming by Hosseini et al. [22], who labeled such behavior as "builders" through the consideration of increments in programming size. Hosseini et al. [22] saw this behavior similar to "movers" by Perkins et al. [34], although the latter did not explicitly consider program lengths.

Interestingly, our model and study also captures learning. When considering the subsequent SQL injection exercise (Exercise B) that is similar to the first one (Exercise A), students are more likely to exploit through writing a query that they then refine to hack the system. This is again in line with prior research; becoming better at problem-solving involves building an understanding of the domain, which in turn helps working in that domain [42, 43].

5.2 Programming paths and SQL injections

When contrasting our study to previous studies in computing education research that have explored how students solve problems, many of the previous studies have been focused on programming (as a review on the topic, see e.g. [24]). In previous studies, researchers have linked students' programming behavior with e.g. the Schema theory [14, 15, 38], pointing out the use of existing schemas in the case of known problems. What separates our study from many of the previous studies is that the problem itself is ill-defined, the students do not know the system, and the process data that is collected (i.e. the SQL injection attempts) is only a small part of big picture. In addition, contrary to courses focusing on learning exploitation techniques and strategies, the course aims to teach how secure systems are designed and implemented, while using the practical assignments as a way of demonstrating common mistakes. Also, while students in introductory programming courses seek to build complete programs, in the case of SQL injections, students seek to identify a type of a string should be injected to an unknown program to achieve a desired outcome. In practice, this means that due to the vagueness the initial exploration space can be broader (and even significantly broader when compared to block-level analyses or analyses of very simple programs – see e.g. [21, 47]). At the same time, interestingly, due to the students starting with an unknown system, we are also able to capture learning in terms of seeing more profound use of exploitation in the subsequent exercise.

6 CONCLUSIONS

We investigated log data related to SQL injection exercises of over 300 students on a cybersecurity course. The exercises involved a custom web platform where students must discover vulnerabilities to gain access to data. We studied students' problem-solving process by simplifying it with a model of sequential transitions between types of queries. We found a natural grouping of students based on the number of submissions they perform. We identified high-performing (lower amount of submissions) and low-performing (higher number of submissions) students. We also found two tactics students adopted for transitioning: exploration and exploitation. The exploration tactic refers to students trying out successively *different types of SQL* queries when attempting to retrieve data. With the exploitation tactic, students used *variations of the same kind of query* when trying to access the information. From these tactics, we highlighted three strategies, defined by the use of either one or both identified tactics. Our results suggested that students who stick to only one tactic were more likely to be high-performing. Furthermore, we also discovered that students who used both tactics changed from a heavy usage of the exploration tactic in the first exercise to an equilibrated mix of both tactics in the subsequent one.

REFERENCES

[1] Alberto Abelló, M Elena Rodríguez, Toni Urpí, Xavier Burgués, M José Casany, Carme Martín, and Carme Quer. 2008. LEARN-SQL: Automatic assessment of SQL based on IMS QTI specification. In *2008 Eighth IEEE International Conference on Advanced Learning Technologies*. IEEE, 592–593.

[2] Charu C. Aggarwal. 2015. *Data Mining: The Textbook*. Springer, Cham. https://doi.org/10.1007/978-3-319-14142-8

[3] Alireza Ahadi, Vahid Behbood, Arto Vihavainen, Julia Prior, and Raymond Lister. 2016. Students' syntactic mistakes in writing seven different types of SQL queries and its application to predicting students' success. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 401–406.

[4] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2015. A quantitative study of the relative difficulty for novices of writing seven different types of SQL queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 201–206.

[5] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. 2016. Students' semantic mistakes in writing seven different types of SQL queries. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 272–277.

[6] Nikolaos Alexopoulos, Sheikh Mahbub Habib, Steffen Schulz, and Max Mühlhäuser. 2020. The tip of the iceberg: On the merits of finding security bugs. *ACM Transactions on Privacy and Security (TOPS)* 24, 1 (2020), 1–33.

[7] Christoph Aschwanden and Martha Crosby. 2006. Code scanning patterns in program comprehension. In *Proc. of the 39th Hawaii int. conf. on system sciences*.

[8] Micheal Axelsen, A Faye Borthick, and Paul Bowen. 2001. A model for and the effects of information request ambiguity on end-user query performance. *ICIS 2001 Proceedings* (2001), 68.

[9] Nada Basit, Abdeltawab Hendawi, Joseph Chen, and Alexander Sun. 2019. A learning platform for SQL injection. In *Proceedings of the 50th ACM technical symposium on computer science education*. 184–190.

[10] A Faye Borthick, Paul L Bowen, Donald R Jones, and Michael Hung Kam Tse. 2001. The effects of information request ambiguity and construct incongruence on query development. *Decision Support Systems* 32, 1 (2001), 3–25.

[11] Stefan Brass and Christian Goldberg. 2006. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software* 79, 5 (2006), 630–644.

[12] Gretchen I Casterella and Leo Vijayasarathy. 2013. An experimental investigation of complexity in database query formulation tasks. *Journal of Information Systems Education* 24, 3 (2013), 6.

[13] William Crumpler and James A Lewis. 2019. *The cybersecurity workforce gap*. Center for Strategic and International Studies (CSIS) Washington, DC, USA.

[14] Simon P Davies. 1991. The role of notation and knowledge representation in the determination of programming strategy: a framework for integrating models of programming behavior. *Cognitive Science* 15, 4 (1991), 547–572.

[15] Françoise Détienne. 1995. Design Strategies and Knowledge in Object-oriented Programming: Effects of Experience. *Hum.-Comp. Interact.* 10, 2 (1995), 129–169.

[16] Masoud I El Agha, Abdallah M Jarghon, and Samy S Abu-Naser. 2018. SQL Tutor for Novice Students. (2018).

[17] CC2020 Task Force. 2020. *Computing Curricula 2020: Paradigms for Global Computing Education*. Association for Computing Machinery, New York, NY, USA.

[18] Brendan J. Frey and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science* 315 (2007), 972 – 976.

[19] Steven Furnell. 2021. The cybersecurity workforce and skills. *Computers & Security* 100 (2021), 102080.

[20] Vanessa E Ghosh and Asaf Gilboa. 2014. What is a memory schema? A historical perspective on current neuroscience literature. *Neuropsych.* 53 (2014), 104–114.

[21] Juha Helminen, Petri Ihantola, Ville Karavirta, and Lauri Malmi. 2012. How do students solve parsons programming problems? an analysis of interaction traces. In *Proceedings of the ninth annual international conference on International computing education research*. 119–126.

[22] Roya Hosseini, Arto Vihavainen, and Peter Brusilovsky. 2014. Exploring problem solving paths in a Java programming course. (2014).

[23] Alison Hull and Benedict du Boulay. 2015. Motivational and metacognitive feedback in SQL-Tutor. *Computer Science Education* 25, 2 (2015), 238–256.

[24] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proc. of the 2015 ITiCSE on Working Group Reports* (Vilnius, Lithuania) *(ITICSE-WGR '15)*. ACM, 41–63.

[25] H Laine. 2001. SQL-trainer. In *Proceedings of Kolin Kolistelut/Koli Calling–First Annual Baltic Conference on Computer Science Education, Report A-2002*, Vol. 1. 13–17.

[26] Juho Leinonen, Nea Pirttinen, and Arto Hellas. 2020. Crowdsourcing Content Creation for SQL Practice. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 349–355.

[27] Robert L Leitheiser and Salvatore T March. 1996. The influence of database structure representation on database system learning and use. *Journal of Management Information Systems* 12, 4 (1996), 187–213.

[28] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin* 38, 3 (2006), 118–122.

[29] Salil Maharjan and Amruth Kumar. 2020. Using Edit Distance Trails to Analyze Path Solutions of Parsons Puzzles.. In *EDM*.

[30] Russell Mosemann and Susan Wiedenbeck. 2001. Navigation and comprehension of programs by novice programmers. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*. IEEE, 79–88.

[31] Joint Task Force on Cybersecurity Education. 2018. *Cybersecurity Curricula 2017: Curriculum Guidelines for Post-Secondary Degree Programs in Cybersecurity*. Association for Computing Machinery, New York, NY, USA.

[32] Benjamin Paaßen, Bassam Mokbel, and Barbara Hammer. 2015. A Toolbox for Adaptive Sequence Dissimilarity Measures for Intelligent Tutoring Systems. In *Proceedings of the 8th International Conference on Educational Data Mining (EDM 2015)* (2015-06), Olga Christina Santos, Jesus Gonzalez Boticario, Cristobal Romero, Mykola Pechenizkiy, Agathe Merceron, Piotr Mitros, Jose Maria Luna, Christian Mihaescu, Pablo Moreno, Arnon Hershkovitz, Sebastian Ventura, and Michel Desmarais (Eds.). International Educational Datamining Society, 632–632.

[33] Allen Parrish, John Impagliazzo, Rajendra K Raj, Henrique Santos, Muhammad Rizwan Asghar, Audun Jøsang, Teresa Pereira, and Eliana Stavrou. 2018. Global perspectives on cybersecurity education for 2030: a case for a metadiscipline. In *Proceedings Companion of the 23rd annual aCM conference on innovation and technology in computer science education*. 36–54.

[34] David N Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of learning in novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 37–55.

[35] Jean Piaget. 1971. Biology and knowledge: An essay on the relations between organic regulations and cognitive processes. (1971).

[36] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 153–160.

[37] Chen Ping, Wang Jinshuang, Yang Lanjuan, and Pan Lin. 2020. SQL Injection Teaching Based on SQLi-labs. In *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*. IEEE, 191–195.

[38] Robert S Rist. 1989. Schema creation in programming. *Cognitive Science* 13, 3 (1989), 389–414.

[39] Josep Soler, Ferran Prados, Imma Boada, and Jordi Poch. 2006. A Web-based tool for teaching and learning SQL. In *International Conference on Information Technology Based Higher Education and Training, ITHET*.

[40] Laszlo Solymar et al. 1999. *Getting the message: A history of communications*. Oxford University Press.

[41] Valdemar Švábenskỳ, Jan Vykopal, and Pavel Čeleda. 2020. What are cybersecurity education papers about? a systematic literature review of sigcse and iticse conferences. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2–8.

[42] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[43] John Sweller and Graham A Cooper. 1985. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and instruction* 2, 1 (1985), 59–89.

[44] Toni Taipalus and Piia Perälä. 2019. What to Expect and What to Focus on in SQL Query Teaching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 198–203.

[45] Toni Taipalus, Mikko Siponen, and Tero Vartiainen. 2018. Errors and complications in SQL query formulation. *ACM Transactions on Computing Education (TOCE)* 18, 3 (2018), 15.

[46] Cynthia Taylor and Saheel Sakharkar. 2019. '); DROP TABLE textbooks;– An Argument for SQL Injection Coverage in Database Textbooks. In *Proceedings of the 50th ACM technical symposium on computer science education*. 191–197.

[47] Arto Vihavainen, Juha Helminen, and Petri Ihantola. 2014. How novices tackle their first lines of code in an ide: Analysis of programming session traces. In *Proc. of the 14th Koli Calling Int. Conf. on Comp. Ed. Research*. ACM, 109–116.

[48] Leon E Winslow. 1996. Programming pedagogy - a psychological overview. *ACM Sigcse Bulletin* 28, 3 (1996), 17–22.

[49] Xiaohong Yuan, Imano Williams, Tae Hee Kim, Jinsheng Xu, Huiming Yu, and Jung Hee Kim. 2017. Evaluating hands-on labs for teaching SQL injection: a comparative study. *Journal of Computing Sciences in Colleges* 32, 4 (2017), 33–39.