




Analysis of Software Vulnerabilities Introduced in Programming Submissions Across Curriculum at Two Higher Education Institutions

Andrew Sanders
Computer and Cyber Science
University 1
Augusta, USA
asanders4@augusta.edu

Gursimran Singh Walia
Computer and Cyber Science
University 1
Augusta, USA
gwalia@augusta.edu

Andrew Allen
Computer Science
University 2
Statesboro, USA
aallen@georgiasouthern.edu

Abstract—This full research paper describes the analysis of common software vulnerabilities that are introduced by students enrolled in four-year computing and cybersecurity majors from two different higher education institutions in Georgia.

As the demand for secure coding education continues to grow, pedagogical improvements need to be made in identifying key software vulnerabilities students commit during code development (from the first programming course to the exit senior design capstone) which in turn can be analyzed to inform the pedagogical interventions focused at preparing students with skill sets for writing secure code and entering the professional workforce. While code security is emphasized throughout the computing curriculum, this research is focused on training individuals to be aware of common vulnerabilities and tailoring programming concept knowledge that has been shown to have a positive effect on code security.

Existing research has mainly focused on developing vulnerability analysis tools rather than collecting data (and subsequently analyzing) regarding the types of vulnerabilities produced by students at their institutions.

In this paper, we analyzed student code across different courses and reported the types of vulnerabilities produced by students in their assignment submission code from two different higher education institutions across different levels of the four-year curriculum. The reported vulnerabilities are subsequently grouped by CWE-ID, which is a standard and common way to categorize and identify software vulnerabilities. The resulting CWE-IDs are then grouped per student submission and per semester (across curriculum levels) to discover the common types of software vulnerabilities committed across cross sections of students.

Our results from the analysis of vulnerabilities (ranging from CS1 courses to capstone courses) are organized around the following research questions: 1) What are the most common software vulnerabilities produced by computing majors at different levels through the computing curriculum?; and 2) Do these vulnerabilities persist throughout their curriculum as they advance into higher-level courses?

We report that students commonly make mistakes related to variable usage, null pointer checks, hard-coding sensitive information, and improperly validating input. Vulnerabilities such as CWE-489 ("Active Debug Code") and CWE-215 ("Insertion of Sensitive Information Into Debugging Code") tend to persist across multiple course levels and may need to be focused in the computing curriculum. The number of vulnerabilities introduced in assignment code increases as course complexity increases. We also find that vulnerabilities produced by students have little overlap with what software vulnerability researchers commonly

study, potentially leading to a mismatch in priority for secure coding topics. Our findings have implications for computer science and cybersecurity curriculum design and delivery.

Index Terms—Computing skills, Computer science, Engineering curriculum, Undergraduate

I. INTRODUCTION

Software vulnerability exploitation is one of the primary methods in which attacks access organizations [4] and is one of the main sources of reported security incidents [5]. Integrating secure coding principles throughout the curriculum is recommended by ACM/IEEE Curriculum Guidelines [10] and recent ABET standards. [1]. Even with this growing focus on secure coding education, the systematic process of integrating secure coding knowledge into the overall computing curriculum is lacking [9]. To help guide this, we believe a focus should be placed on the types of vulnerabilities introduced by students in their code so a more targeted, practiced approach could be used. However, the existing literature on student-produced vulnerabilities is limited [14, 12], and the limited literature, seems to indicate that the types of vulnerabilities produced by students have little overlap with the types that are commonly researched in software vulnerability detection literature [6, 12].

Analysis of student code submissions at different knowledge levels is needed to gain better insights. We used the Common Weakness Enumeration (CWE) framework to label software vulnerabilities [3] found in student code and to compare the results with prior work. To aid readers' understanding of the work presented in this research, we define weakness and vulnerability in code consistent with the CWE framework definitions:

- A weakness is a condition that, under certain circumstances, could contribute to the introduction of vulnerabilities.
- A vulnerability is a weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

CWE is a list of weakness types for software and hardware. It is common to refer to a weakness using its CWE-ID. The most

common types of weaknesses studied by software vulnerability researchers, referred to by their CWE-IDs, are as follows [6]:

- CWE-78 ("OS command injection") - Not sanitizing externally influenced input that is used in part of an OS command
- CWE-79 ("Cross-site scripting") - Not neutralizing user-controllable input before it is used as part of a web page that is served to other users
- CWE-89 ("SQL injection") - Not sanitizing externally influenced input that is part of an SQL command
- CWE-119 ("Buffer errors") - Reading/Writing to a memory location outside of the intended buffer boundary
- CWE-120 ("Buffer overflow") - Copies an input buffer to an output buffer without verifying the size difference of the buffers
- CWE-190 ("Integer Overflow") - Performs a calculation that can produce an integer overflow/wraparound
- CWE-306 ("Missing auth. for critical function") - Does not perform authentication for functionality that requires a provable user identity

Each of these weaknesses is mentioned in the 2022 CWE Top 25 Most Dangerous Software Weaknesses [2], which is based on data from officially submitted Common Vulnerabilities and Exposures (CVE) [7] and the National Vulnerability Database (NVD) [11].

This research uses an existing static analysis tool, SonarQube Community Edition, to analyze the types of software vulnerabilities produced by students in their assignment submission code from two major universities in Georgia, referred to hereafter as University 1 and University 2 to maintain anonymity. SonarQube, as discussed further in section III, is used to scan software projects for errors and provides a mapping to CWE-ID weaknesses if the mapping is appropriate. The results found are used to further establish the most common types of student-produced software vulnerabilities so teaching methods and tools can be developed to help students and developers develop more secure software. Our previous work analyzed software vulnerabilities from student submissions in a CS2 course and compared the types of vulnerabilities discovered with what is commonly researched by software vulnerability researchers and what was previously reported in student software vulnerability studies [12]. With this work, we build upon our previous work by attempting to answer the following research questions.

- *RQ1*: What are the most common software vulnerabilities produced by computing majors at different levels through the computing curriculum?
- *RQ2*: Do these vulnerabilities persist throughout their curriculum as they advance into higher-level courses?

This research is formatted as follows. Section II reviews the related work in the research area. Section III discusses the dataset and methodology used to answer the research questions. Section IV reviews the results of each research question. Section V discusses the findings and conclusion.

II. RELATED WORK

In our previous work, we used a static analysis tool to analyze the types of software vulnerabilities produced by students in an undergraduate Programming II (CS2) course [12]. The dataset was created from assignment submissions from a University 2 Programming Principles II course over the 2017-2023 academic years. The course is taught using the Java programming language and the students are required to use the language for their assignment submissions. The course used GitHub to host some of the assignments and required the students to push their finished code to their Git repositories. These GitHub repositories were collected and organized by year and semester. In total, there were 3537 assignment submissions over the 2017-2023 school years. The static analysis tool analyzed each submission for vulnerabilities and produced CWE-ID classifications for each vulnerability found. We found that the following CWE-IDs were most common:

- CWE-546 ("Suspicious Comment")
- CWE-581 ("Object Model Violation")
- CWE-476 ("NULL Pointer Dereference")
- CWE-563 ("Assignment to Variable without Use")
- CWE-489 ("Active Debug Code")
- CWE-215 ("Insertion of Sensitive Information Into Debugging Code")
- CWE-459 ("Incomplete Cleanup")
- CWE-772 ("Missing Release of Resource after Effective Lifetime")
- CWE-1241 ("Use of Predictable Algorithm in Random Number Generator")
- CWE-326 ("Inadequate Encryption Strength")

We concluded that our findings, as with Yilmaz et al. [14], differed from what was commonly researched by vulnerability researchers [6]. The only common CWE-IDs found in our previous dataset and those reported in the research were CWE-190 ("Integer Overflow") and CWE-259 ("Use of Hard-Coded Password"). We also concluded that more work would need to be done to establish the common types of vulnerabilities produced by students by analyzing data from multiple institutions, including cybersecurity programs, and at different levels of the computing curriculum.

Yilmaz and Ulusoy used a source code vulnerability analysis tool to analyze vulnerabilities produced by students in a Database Management Systems course [14]. Their private dataset was sourced from two assignments over six semesters and the students used PHP, HTML, and JavaScript for their assignment submissions. The assignments required the students to develop a web application with a database using PHP/MySQL and were required to publish their application online. One of their findings was that students who focused on functionality tended to also have an increased amount of vulnerabilities, as shown in figure 1. This may have been due to students being concerned with receiving full credit on their assignment and leaving security as an afterthought. When including an incentive of bonus points if students completed a security-related portion of the assignment, they found that it

TABLE I
VULNERABILITIES OF STUDENT CODE PER CWE TYPE [14]

CWE-ID	CWE Name	Count
CWE-259	Use of Hard-coded Password	829
CWE-20	Improper Input Validation	761
CWE-564	SQL Injection: Hibernate	751
CWE-943	Improper Neutralization of Special Elements in Data Query Logic	751
CWE-489	Active Debug Code	714
CWE-315	Cleartext Storage of Sensitive Information in a Cookie	23
CWE-117	Improper Output Neutralization for Logs	17
CWE-532	Insertion of Sensitive Information into Log File	17
CWE-778	Insufficient Logging	17
CWE-521	Weak Password Requirements	15
CWE-311	Missing Encryption of Sensitive Data	14
CWE-614	Sensitive Cookie in HTTPS Session Without “Secure” Attribute	14

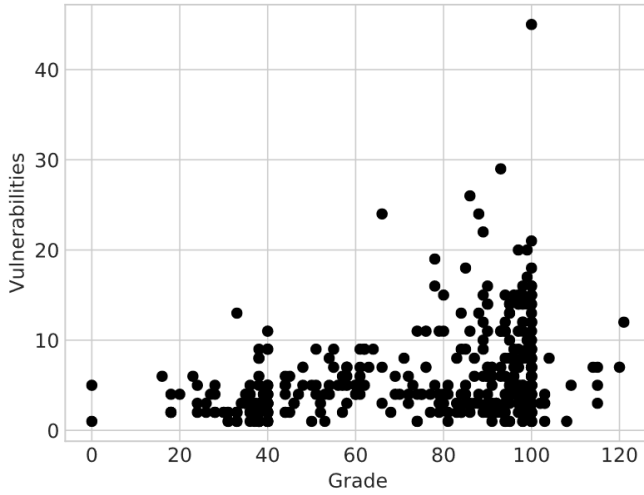


Fig. 1. Vulnerabilities of Student Code and Grades Received [14]

had an effect of reducing the number of vulnerabilities for the submission. This can potentially imply that emphasis, either through incentives or making it part of the assignment description, should be used to enhance the security of assignment submissions. Their major vulnerabilities reported, which we compare to in section IV, are shown in table II.

Hanif et al. created a taxonomy of research interests for software vulnerability detection methods [6]. The research interests they categorize are methods, detection, feature, code, and dataset. They reported a large interest in research papers focusing on addressing methods and detection problems and papers using machine learning to detect vulnerabilities. Relevant to this research is Hanif et al. found that most works targeted specific types of vulnerabilities for detection because the vulnerabilities are frequently targeted by vulnerability detection systems and not necessarily because they are commonly introduced in code. Another finding by the authors was that there is a lack of a large, gold-standard dataset for software vulnerability detection outside of the National Vulnerability Database (NVD). The authors note that

Our Measurement	Book 1	Book 2	Book 3	Book 4
Security Vulnerability	17	13	17	26
Quality Vulnerability	82	36	121	106

Fig. 2. Types of Bugs Reported by FindBugs in Four Java Textbooks Categorized by the Authors’ Vulnerability Criteria [8]

the NVD dataset involves manual source code extraction from repositories, potentially leading to mislabeling of data. Of the 83 cited papers in their study, 20 papers source the NVD, 18 papers source the Software Assurance Reference Dataset (SARD), 49 papers source Open Source Software (OSS), 2 papers source from Common Vulnerabilities and Exposures (CVEs), 3 papers source from code competitions, and 12 papers source from private datasets. Rolling up the data shows that it predominately comes from the National Institute of Science and Technology (NIST), Open Source Software, or private datasets. As a result of this concentration, student code may be underrepresented. This may lead to an insufficient focus on targeted instruction for the types of vulnerabilities commonly introduced by students in their assignments.

Hu et al. reviewed software vulnerabilities in multiple Java programming textbooks for an undergraduate Java programming course [8]. Figure 2 shows their breakdown of software vulnerabilities detected in Java programming textbook examples using the authors’ vulnerability criteria. FindBugs categorizes bugs into 9 different categories: Bad practice, correctness, experimental, internationalization, malicious code vulnerability, multithreaded correctness, performance, security, and dodgy code. Hu et al. categorize all 9 bugs as “Quality Vulnerability”, and categorize bad practice, correctness, malicious code vulnerability, multithreaded correctness, and security under “Security Vulnerability”. The authors used the open-source vulnerability analysis tool, FindBugs, to analyze the byte code of sample source code contained in four Java textbooks. They found many common bugs, as reported by FindBugs, in the source code that went undetected by the textbook authors. Since these Java textbooks targeted un-

dergraduate programming courses, this has implications for teaching students to produce insecure code, if the students were to adopt the coding styles of the textbooks.

This section indicates that while much work has been placed into vulnerability detection systems and highlighting the need to enforce secure coding practices, there is still a lack of focus on analyzing the types of software vulnerabilities produced by computing students in their assignment code. This lack of focus can be paired with a lack of directed pedagogy toward preventing students from producing the types of vulnerabilities they commonly create in their assignment submissions.

III. METHODOLOGY

This section reviews the methodology used in this work to answer the research questions. There are subsections for Dataset, Research Question 1, and Research Question 2.

A. Dataset

Our heterogeneous dataset of 7,969 assignment submissions was generated from several courses at University 1 and University 2 (major higher-ed institutions in Georgia).¹ Each data source was cleaned of incompatible data, such as assignments that required images (i.e., png, jpg, etc.).

From University 1, the following sources of data were used:

- 10 assignment submissions from a software engineering capstone course for the 2022-2023 school years, collected through the LMS (learning management system)
- 1688 assignment submissions from a CS1 course from the 2018-2023 academic years, collected through the LMS
- 70 assignment submissions from a Database System course from the 2020-2021 school years, collected through the LMS

From University 2, the following sources of data were used:

- 25 assignment submissions from a Fall 2018 CS2 course, collected through the Learning Management System (LMS)
- 3,563 assignment submissions from a CS2 course from the 2017-2023 school years, collected through GitHub
- 75 assignment submissions from a software engineering (SE) capstone course from the 2017-2023 school years, collected through GitHub
- 2,538 assignment submissions from a CS2 course from the 2021-2022 school years, collected through Gradescope (an auto-grading tool)

The assignment submissions were anonymized and de-identified. The University 2 CS2 assignment requirements were focused on object-oriented-related topics and all submissions used the Java programming language. The University 1 CS1 assignment focused on beginner programming topics and all submissions used the C# programming language. The University 1 Database Systems course used the Java programming language for the assignments. The SE submissions were from the semester-long capstone projects. Each SE submission was varied in programming language and design, as they were

¹Full dataset located at (Hidden for review purposes)

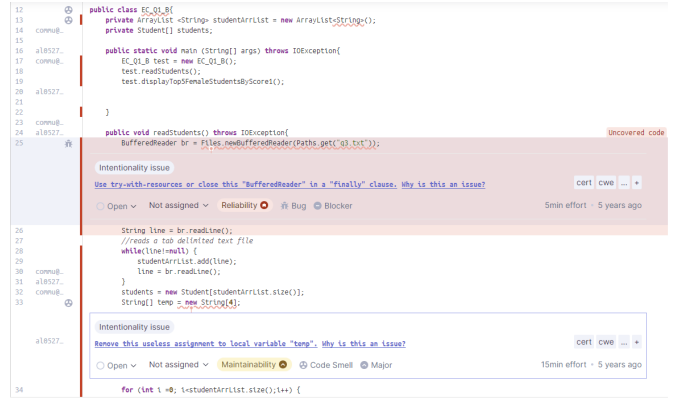


Fig. 3. SonarQube Analysis Example

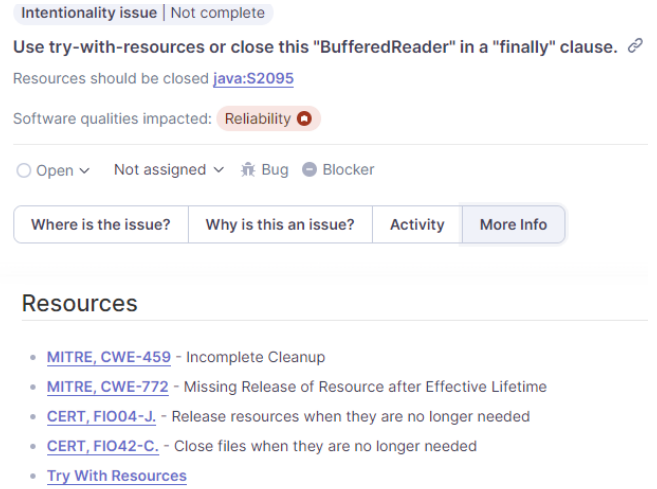


Fig. 4. SonarQube Issue Information Example

based on the capstone project topics for the current semester. Each CS1, CS2, and Database Systems course submission was produced by a single student. The number of students per SE assignment submission is unknown but could be produced by up to five students. To account for SE student number variability, whenever rates of vulnerabilities produced per SE assignment are discussed, we provide an additional 0.2-scaled version for the five-person capstone team scenario. The 3,563 CS2 GitHub assignment submissions and the 75 SE GitHub assignment submissions were included as the dataset for our previous work [12].

B. Research Question 1

To answer *RQ1*, we applied the self-managed static analysis tool, Sonarqube Community Edition (Version No. 10.2.0.77647), on our dataset to analyze student assignment submission codes for vulnerabilities and weaknesses. SonarQube has a quality model with four different types of rules: reliability (bug), maintainability (code smell), and security (vulnerability and security hotspot) [13]. A security hotspot highlights a sensitive piece of code that requires the review

of a developer to determine the impact. A vulnerability is a problem that impacts the security of the program and needs to be fixed. A bug is a coding mistake that can lead to an error at runtime. A code smell is an issue that makes code difficult to maintain and potentially confusing for other developers.

The SonarQube tool scanned each repository on our dataset and generated a report of each bug, security hotspot, code smell, and vulnerability it found. Each reported issue contained tags for identification. For our research, only the tags with “cwe” were used to determine the vulnerabilities of the submissions, as they contained a CWE-ID mapping that is used for comparison with prior research. These CWE-IDs were associated with the assignment submission they originated from and used for later analysis. Figure 3 shows an example SonarQube analysis of an assignment submission. The figure shows both a bug and a code smell on separate lines. Each issue has a “cwe” tag, indicating that the issue has a CWE-ID mapping. Figure 4 shows a description of the bug issue from the highlighted area in Figure 3. The issue in this figure is “Use try-with-resources or close this ‘BufferedReader’ in a ‘finally’ clause”. This issue has a CWE-ID mapping to both CWE-459, “Incomplete Cleanup” and CWE-772, “Missing Release of Resource after Effective Lifetime”. When analyzing a submission, we aggregate all CWE-IDs. In the example, that means including CWE-IDs 459, 772, and any other CWE-IDs that were mapped to in the other issues.

These findings are also compared with the types of vulnerabilities most commonly studied [6] and what has been reported previously [14, 12] to bolster the understanding of which types of vulnerabilities are most commonly produced by students. The most commonly studied vulnerabilities by software vulnerability researchers are CWE-IDs 78, 89, 119, 120, 190, and 306 [6]. The most common vulnerabilities reported by [14] are CWE-IDs 259, 20, 564, 943, 480, 315, 117, 532, 778, 521, 311, and 614. The most common vulnerabilities reported in our previous work are CWE-IDs 546, 581, 476, 563, 489, 215, 459, 772, 1241, and 326 [12]. Our results are presented in section IV-A.

C. Research Question 2

To answer *RQ2*, we used the findings from *RQ1* and grouped the dataset sources into three categories: introductory courses, intermediate courses, and advanced courses. For the introductory courses, we analyzed the (University 1) CS1 course assignment submissions. For the intermediate courses, we analyzed the (University 2) CS2 courses and (University 1) Database course submissions. For the advanced courses, we analyzed the (University 1) and (University 2) SE course submissions.

In total, the introductory-level courses had 1,688 total assignment submissions. The intermediate-level courses had 6,196 total assignment submissions. The advanced courses had 85 total assignment submissions.

Using these groupings, we calculate the frequency of each CWE-ID for each category and summarize our findings. We then compare the common CWE-IDs found between levels

to determine which, if any, vulnerabilities persist throughout different computing academic levels. We also observe any changes in the types of CWE-IDs produced as students advance through the curriculum.

Our results are presented in section IV-B.

IV. RESULTS

Our results for answering *RQ1* and *RQ2* are discussed in this section. In total, there were 25,583 vulnerabilities detected by SonarQube across 7,969 submissions.

A. Research Question 1

To address *RQ1*, we utilized the SonarQube static analysis tool to scan assignment submissions across different courses and institutions. We provide a comprehensive overview of the types and frequencies of vulnerabilities introduced in the code.

Table II provides the most common CWE-IDs produced by students in the computing curriculum.

- CWE-563, “Assignment to Variable without Use”, was the most common CWE-ID detected. This weakness was consistently the most frequent across multiple datasets. This indicates that students often create variables that are not used in their code, which can lead to confusion and potential errors in their code.
- CWE-546, “Suspicious Comment”, was the second most common CWE-ID detected. This indicates that students commonly leave comments in their code that could potentially reveal sensitive information, which could lead to a security risks.
- The third most commonly found CWE-ID, CWE-581, refers to “Object Model Violation: Just One of Equals and Hashcode Defined”. This is related to students not implementing both the equals method and the hashcode method in their Java programs. Depending on the requirements of the Java programming course, this weakness could potentially be ignored as it may be out of scope for the programming level of the students.
- The fourth most commonly found CWE-ID was CWE-476, which is “NULL Pointer Dereference”. This is related to students not checking for NULL before attempting to use an object. This can lead to runtime errors in programs in which the programmer does not check each object for NULL.
- The fifth most commonly found CWE-ID was CWE-215, which is “Insertion of Sensitive Information Into Debugging Code”. This indicates that students often leave sensitive information in debugging code, such as revealing critical information through `System.out.println` in Java code. This exposure can leak sensitive information about the program, which may result in exploitation.

These vulnerabilities persisted across various courses and levels. This suggests that these issues are pervasive and may need targeted pedagogical interventions.

Table III provides the summary of the top five weaknesses found for every Non-SE dataset. Over the entire Non-SE dataset, we found 21,829 vulnerabilities, as detected by

TABLE II
MOST COMMON CWE-IDS PRODUCED BY COMPUTING MAJORS

CWE-ID	Count	Rate (per submission)	CWE Name
CWE-563	2118	0.266	Assignment to Variable without Use
CWE-546	1795	0.225	Suspicious Comment
CWE-581	1584	0.199	Object Model Violation: Just One of Equals and Hashcode Defined
CWE-476	1329	0.167	NULL Pointer Dereference
CWE-215	1304	0.164	Insertion of Sensitive Information Into Debugging Code
Total	25583	3.210	–

TABLE III
SUMMARY OF TOP FIVE CWE-IDS FOUND IN ASSIGNMENT SUBMISSIONS FROM NON-SE DATASETS.

Dataset	CWE-ID	Count	Rate (per submission)	CWE Name
(University 1) CS1 LMS	CWE-563	489	0.290	Assignment to Variable without Use
	CWE-571	195	0.116	Expression is Always True
	CWE-570	195	0.116	Expression is Always False
	CWE-338	137	0.081	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
	CWE-330	137	0.081	Use of Insufficiently Random Values
	Total	1575	0.933	–
(University 2) CS2 Gradescope	CWE-563	447	0.176	Assignment to Variable without Use
	CWE-581	362	0.143	Object Model Violation: Just One of Equals and Hashcode Defined
	CWE-595	258	0.102	Comparison of Object References Instead of Object Contents
	CWE-597	258	0.102	Use of Wrong Operator in String Comparison
	CWE-476	233	0.092	NULL Pointer Dereference
	Total	3371	1.328	–
(University 2) CS2 LMS	CWE-338	11	0.440	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
	CWE-330	11	0.440	Use of Insufficiently Random Values
	CWE-326	11	0.440	Inadequate Encryption Strength
	CWE-1241	11	0.440	Use of Predictable Algorithm in Random Number Generator
	CWE-493	10	0.400	Critical Public Variable Without Final Modifier
	Total	72	2.880	–
(University 2) CS2 GitHub	CWE-546	1502	0.422	Suspicious Comment
	CWE-581	1222	0.343	Object Model Violation: Just One of Equals and Hashcode Defined
	CWE-476	1089	0.306	NULL Pointer Dereference
	CWE-563	1049	0.294	Assignment to Variable without Use
	CWE-489	870	0.244	Active Debug Code
	Total	15444	4.335	–
(University 1) Database LMS	CWE-89	179	2.557	Improper Neutralization of Special Elements used in an SQL Command
	CWE-564	179	2.557	SQL Injection: Hibernate
	CWE-20	179	2.557	Improper Input Validation
	CWE-943	179	2.557	Improper Neutralization of Special Elements in Data Query Logic
	CWE-459	113	1.614	Incomplete Cleanup
	Total	1367	19.529	–
Overall Total	–	21829	–	–

SonarQube. For the (University 1) CS1 course, we see 1,575 vulnerabilities over 1,688 assignment submissions for a rate of 0.933 vulnerabilities per submission. Since this course is an introductory computing course, many of the vulnerabilities found, such as CWE-563 (“Assignment to Variable without Use”), CWE-571 (“Expression is Always True”), and CWE-570 (“Expression is Always False”) can be closely associated with beginner programmers.

For the (University 2) CS2 courses, we see a rate of vulnerabilities per assignment of between 1.328 and 4.335, indicating a higher amount of vulnerabilities introduced than compared to the CS1 course. While CWE-563 (“Assignment to Variable without Use”) can be seen in the top five CWE-IDs for two of these datasets, the types of vulnerabilities found are more closely related to CS2-related concepts. For instance, we can see CWE-595 (“Comparison of Object References Instead of Object Contents”) and CWE-476 (“NULL Pointer Dereference”). Both of these vulnerabilities are related to

object-oriented concepts that CS2 students are commonly introduced to.

For the (University 1) Database course, we see a much higher rate of vulnerabilities per submission at 19.529, indicating that programs created during this course have much more vulnerabilities than lower-level courses. As might be intuitively guessed, the major types of vulnerabilities produced by students in this course are related to input validation (CWE-20) and SQL Injection (CWE-89/564). As students are introduced to databases, they are also introduced to SQL and integration with other programming languages. Within this course, students were required to connect and communicate with their database using Java. As SonarQube scanned the Java code for vulnerabilities, it found many instances of SQL-injected and input validation-related vulnerabilities.

Table IV shows the summary of the top five weaknesses found for every SE dataset. Over the entire SE dataset, we found 3,754 vulnerabilities, as detected by SonarQube. As SE

TABLE IV
SUMMARY OF TOP FIVE CWE-IDS FOUND IN ASSIGNMENT SUBMISSIONS FROM SE DATASETS.

Dataset	CWE-ID	Count	Rate (per submission)	Rate (*0.2)	CWE Name
(University 1) SE LMS	CWE-215	36	3.600	0.720	Insertion of Sensitive Information Into Debugging Code
	CWE-489	36	3.600	0.720	Active Debug Code
	CWE-493	32	3.200	0.640	Critical Public Variable Without Final Modifier
	CWE-563	26	2.600	0.520	Assignment to Variable without Use
	CWE-1333	24	2.400	0.480	Inefficient Regular Expression Complexity
	Total	289	28.900	5.780	–
(University 2) SE GitHub	CWE-489	282	3.760	0.752	Active Debug Code
	CWE-215	282	3.760	0.752	Insertion of Sensitive Information Into Debugging Code
	CWE-400	262	3.493	0.699	Uncontrolled Resource Consumption
	CWE-1333	260	3.467	0.693	Inefficient Regular Expression Complexity
	CWE-353	250	3.333	0.667	Missing Support for Integrity Check
	Total	3465	46.200	9.240	–
Overall Total	–	3754	–	–	–

courses may have between one to five students working on each assignment, we provide the rate column multiplied by 0.2 to act as the scenario where five students are working on the assignments. Without this column, it may incorrectly indicate that assignment submissions in SE courses have nearly 30 times as many vulnerabilities per submission as earlier-level courses.

For the (University 1) and (University 2) SE courses, we can see a dramatic increase in the number of vulnerabilities produced per assignment submission. The (University 1) SE course had a rate of 28.9 (or 5.78 if scaled to five students), and the (University 2) SE course had a rate of 46.2 (or 9.24 if scaled to five students). This indicates, along with what was found by Yilmaz and Ulusoy [14], that as code complexity increases, the chance for vulnerabilities to be produced also increases. Though over two institutions, the two main vulnerabilities found were CWE-215 (“Insertion of Sensitive Information into Debugging Code”) and CWE-489 (“Active Debug Code”). These indicate that students commonly introduce vulnerabilities related to leaving development-related code in their programs. These, if introduced to a malicious actor, could potentially lead to the exploitation of their programs. As program complexity increases, it may be important to teach students to avoid leading development code in their program builds.

When compared to existing research, we find some similarities with what is commonly studied by software vulnerability researchers, according to Hanif et. al [6], and what was previously reported by Yilmaz and Ulusoy [14]. When compared to what is commonly studied by software vulnerability researchers, we find similarities with just CWE-89 (“SQL Injection”) when considering the (University 1) Database course. The other CWE-IDs commonly researched are not shared with the major CWE-IDs found within our dataset. This indicates that the types of vulnerabilities commonly studied by software vulnerability researchers are not reflective of the types of vulnerabilities commonly produced by students. This may lead to an over-emphasis in security course curriculums towards less-commonly produced vulnerabilities than other types, such as CWE-489 (“Active Debug Code”) When compared to what was previously reported by Yilmaz and Ulusoy, we see much

more commonality. CWE-IDs such as CWE-20, CWE-564, CWE-943, and CWE-489 are all shared but some of our datasets. CWE-20 (“Improper Input Validation”) and CWE-564 (“SQL Injection: Hibernate”) are contained within our (University 1) Database course. As their dataset comes from a Database Management Systems course, this makes sense as to why there is a strong overlap. As their assignment requirements had the students create web programs that communicate with a database, having CWE-489 (“Active Debug Code”) being shared with our (University 1) and (University 2) SE courses also makes sense, as they both require large programs being developed with high complexity.

B. Research Question 2

As shown in figure V, the rate of vulnerabilities per assignment submission for introductory courses was 0.93, indicating a relatively small amount of introduced vulnerabilities in the earlier-level courses of the computing curriculum. Per figure VI, the rate of vulnerabilities per assignment submission for intermediate courses was 3.27, indicating a moderate amount of introduced vulnerabilities as students approach more advanced courses. Figure VII shows the rate of vulnerabilities per assignment submission for advanced courses being 44.165 (or 8.833, if scaled by 0.2 to account for five-person teams). Advanced courses, as mentioned in section III, consist of the (University 2) and (University 1) computer science capstone courses. The assignment submissions for these courses can be for teams of up to five members. Due to this, the vulnerabilities introduced in advanced-level classes should be considered to be split into up to five ways. This means the actual rate of vulnerabilities introduced is between 8.83 (five members) and 44.16 (one member).

These results indicate that as students advance through the computing curriculum, they are more likely to introduce software vulnerabilities in their code. This may be due to increasing complexity and new types of content being learned. For example and as discussed in IV-A, SQL and input-related vulnerabilities, such as CWE-89 and CWE-20, are first introduced in the Database course. This means this whole class of vulnerabilities is most likely to be untouched by CS1 and CS2 students as they are unaware of databases and

TABLE V
SUMMARY FOR TOP FIVE CWE-IDS FOUND IN INTRODUCTORY COMPUTING COURSES ((UNIVERSITY 1) CS1)

CWE-ID	Count	Rate (per submission)	CWE Name
CWE-563	489	0.290	Assignment to Variable without Use
CWE-571	195	0.116	Expression is Always True
CWE-570	195	0.116	Expression is Always False
CWE-338	137	0.081	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
CWE-330	137	0.081	Use of Insufficiently Random Values
Total	1575	0.933	–

TABLE VI
SUMMARY FOR TOP FIVE CWE-IDS FOUND IN INTERMEDIATE COMPUTING COURSES ((UNIVERSITY 2) CS2 AND (UNIVERSITY 1) DATABASE)

CWE-ID	Count	Rate (per submission)	CWE Name
CWE-546	1735	0.280	Suspicious Comment
CWE-581	1584	0.256	Object Model Violation: Just One of Equals and Hashcode Defined
CWE-563	1509	0.244	Assignment to Variable without Use
CWE-476	1323	0.214	NULL Pointer Dereference
CWE-459	1001	0.162	Incomplete Cleanup
Total	20254	3.269	–

TABLE VII
SUMMARY FOR TOP FIVE CWE-IDS FOUND IN ADVANCED COMPUTING COURSES ((UNIVERSITY 2) AND (UNIVERSITY 1) SE)

CWE-ID	Count	Rate (per submission)	Rate (*0.2)	CWE Name
CWE-215	318	3.741	0.748	Insertion of Sensitive Information Into Debugging Code
CWE-489	318	3.741	0.748	Active Debug Code
CWE-1333	284	3.341	0.668	Inefficient Regular Expression Complexity
CWE-400	274	3.224	0.645	Uncontrolled Resource Consumption
CWE-353	250	2.941	0.588	Missing Support for Integrity Check
Total	3754	44.165	8.833	–

SQL. Furthermore, as students advance through the computing curriculum, they must handle many more types of vulnerabilities and be prepared to knowingly handle them to not leave their programs open to exploitation. This difficult balance can potentially help explain why later-level courses contain many more vulnerabilities in their assignment submissions than earlier-level courses.

These results indicate that the assignment and course type may change the types of software vulnerabilities produced by students, though some vulnerabilities like CWE-563 (“Assignment to Variable without Use”) and CWE-489 (“Active Debug Code”) can be seen in multiple course levels. The results also reinforce the notion that the types of vulnerabilities commonly studied by vulnerability researchers are not reflected in the types of vulnerabilities commonly produced by students. Further research is needed to determine how assignment and course requirements influence the vulnerabilities introduced by students and therefore their practical exposure to resolving them.

V. DISCUSSION AND CONCLUSION

This research paper presented the analysis of software vulnerabilities in multiple computing courses from two universities. We used the SonarQube static analysis tool to examine the types of vulnerabilities produced by students in their assignment code and categorized the data by course level. We also provide a comparison of our results with prior research in section IV-A.

We find that students tend to produce fewer vulnerabilities in introductory programming courses as compared to higher-level courses like the Software Engineering Capstone course. The types of vulnerabilities produced also change depending on the course. Beginner-level courses, like introductory CS1 programming courses, tend to have more simplistic vulnerabilities, such as CWE-563 (“Assignment to Variable without Use”) and CWE-570 (“Expression is Always True”). This may be due to the limited complexity of earlier-level courses and students making simple programming mistakes due to inexperience.

Intermediate-level courses like CS2 and Database tend to vary in vulnerability type, but in general, contain more overall vulnerabilities than earlier-level courses. CS2 courses tend to contain more object-oriented related vulnerabilities, such as CWE-595 (“Comparison of Object References Instead of Object Contents”) and CWE-476 (“NULL Pointer Dereference”). This may be due to the introduction of object-oriented related concepts that are not taught in CS1 courses. Database courses tend to contain more SQL Injection and Input Validation-related vulnerabilities such as CWE-89 (“SQL Injection”) and CWE-20 (“Improper Input Validation”). This may be due to students being introduced to database communication in their programming language and not being familiar with security-related concepts in these courses.

Higher-level courses like Software Engineering Capstone tend to have much more vulnerabilities than earlier-level courses and have more development-related vulnerabilities

than other courses. Vulnerabilities such as CWE-489 (“Active Debug Code”) and CWE-215 (“Insertion of Sensitive Information Into Debugging Code”) are much more common in higher-level courses. This may be due to students leaving in development-related code to reach the deadlines of the course rather than considering the security implications of leaving sensitive information in the code. Additionally, as multiple components may be interacting, the likelihood of vulnerabilities being introduced may increase and oversights may occur.

For each course level, it may be important to target pedagogical interventions to prevent common types of vulnerabilities from appearing. For CS1-level courses, this may be instructing students to regularly clean up their code and checking their boolean conditions to prevent their code from becoming more unreadable and less secure. For CS2-level courses, this may come in the form of making sure students always check for NULL before working with objects and helping them to understand how object references and object contents differ so they do not use “==” for inappropriate comparison. For Database-level courses, this probably comes in the form of instructing students to use more secure methods of handling and sanitizing input when accessing databases in programs. For SE-level courses, this may come in the form of teaching students industry-type methodologies, like having production and development builds and making sure they understand to not leave sensitive information hard-coded in their programs.

Poorly understood concepts such as proper variable usage, null pointer checks, avoiding hard-coded sensitive information, and ensuring proper input validation seem to be major trends throughout the assignment submissions. It may be important to integrate secure coding practices early, emphasize security as complexity grows, and provide incentives to practice secure coding at all levels. More research will need to be done to design effective pedagogy to address software vulnerability-related issues with the current computing curriculum.

REFERENCES

- [1] *Accreditation Changes — ABET*.
URL: <https://www.abet.org/accreditation/accreditation-criteria/accreditation-changes/> (visited on 02/02/2023).
- [2] *CWE - Common Weakness Enumeration*.
URL: <https://cwe.mitre.org/index.html> (visited on 02/22/2023).
- [3] *CWE - Frequently Asked Questions (FAQ)*.
URL: <https://cwe.mitre.org/about/faq.html> (visited on 03/08/2023).
- [4] *DBIR Report 2023 - Master's Guide*.
Verizon Business. URL: <https://www.verizon.com/business/resources/reports/dbir/2023/master-guide/> (visited on 07/06/2023).
- [5] Department of Homeland Security, US-CERT. *Software Assurance*.
URL: https://www.cisa.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf (visited on 07/11/2023).
- [6] Hazim Hanif et al. “The Rise of Software Vulnerability: Taxonomy of Software Vulnerabilities Detection and Machine Learning Approaches”.
In: *Journal of Network and Computer Applications* 179 (Apr. 1, 2021), p. 103009. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2021.103009.
URL: <https://www.sciencedirect.com/science/article/pii/S1084804521000369> (visited on 01/11/2023).
- [7] *Home — CVE*.
URL: <https://www.cve.org/> (visited on 02/22/2023).
- [8] Yen-Hung Hu and Thomas Kofi Annan. “Assessing Java Coding Vulnerabilities in Undergraduate Software Engineering Education by Using Open Source Vulnerability Analysis Tools”.
In: *Journal of The Colloquium for Information Systems Security Education* 4.2 (2 Feb. 19, 2017), pp. 33–33. ISSN: 2641-4554. URL: <https://cisse.info/journal/index.php/cisse/article/view/60> (visited on 02/01/2023).
- [9] John Zorabedian. *Veracode Survey Research Identifies Cybersecurity Skills Gap Causes and Cures*. Veracode.
URL: <https://www.veracode.com/blog/security-news/veracode-survey-research-identifies-cybersecurity-skills-gap-causes-and-cures> (visited on 07/12/2023).
- [10] Association for Computing Machinery (ACM) and IEEE Computer Society Joint Task Force on Computing Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*.
New York, NY, USA: Association for Computing Machinery, 2013. 518 pp. ISBN: 978-1-4503-2309-3.
- [11] *NVD - Home*.
URL: <https://nvd.nist.gov/> (visited on 02/22/2023).
- [12] Andrew Sanders, Gursimran Singh Walia, and Andrew Allen. “Assessing Common Software Vulnerabilities in Undergraduate Computer Science Assignments”.
In: *Journal of The Colloquium for Information Systems Security Education* 11.1 (1 Feb. 27, 2024), pp. 8–8. ISSN: 2641-4554. DOI: 10.53735/cisse.v11i1.179.
URL: <https://cisse.info/journal/index.php/cisse/article/view/179> (visited on 05/01/2024).
- [13] SonarSource. *Issues*. URL: <https://docs.sonarsource.com/sonarqube/latest/user-guide/issues/> (visited on 07/12/2023).
- [14] Tolga Yilmaz and Özgür Ulusoy. “Understanding Security Vulnerabilities in Student Code: A Case Study in a Non-Security Course”.
In: *Journal of Systems and Software* 185 (Mar. 1, 2022), p. 111150. ISSN: 0164-1212. DOI: 10.1016/j.jss.2021.111150.
URL: <https://www.sciencedirect.com/science/article/pii/S0164121221002430> (visited on 09/18/2022).