



MONTCLAIR STATE
UNIVERSITY

Montclair State University
**Montclair State University Digital
Commons**

Theses, Dissertations and Culminating Projects

1-2023

A Functioning Code May Not Be a Secure Code : A Preliminary Study on the Students' Complacency with Secure Coding

Jeremiah Niiquaye Kotey

Follow this and additional works at: <https://digitalcommons.montclair.edu/etd>



Part of the [Computer Sciences Commons](#)

Abstract

Eleanor Roosevelt once said: "Learn from the mistakes of others. You can't live long enough to make them all yourself". Mistakes are almost inevitable while coding or designing a system. Therefore, patches are created to fix the issues in the code either by a manual review, or through a static analysis tool. Oftentimes, mistakes in programming emanate from lack of skills thus, competence with a particular programming language but negligence also plays a role in other instances. A functioning code that solves a particular problem does not guarantee that the code is secure, hence the code should be structured to meet secure programming guidelines and principles. Most students tend to stop at a functioning code, paying less attention to the security aspects of programming. This has an ultimate impact on the industries where software security gets the priority. Therefore, students should be motivated for practicing secure programming in their academic levels. It will grow their interests in writing professional code from the beginning and raise their values as novel developers to the competing world. How do we bridge the gap between common mistakes made by new developers and professional developers? Strict coding practices must be enforced in academia and an updated database of common errors in programming must be kept as a guide to enrich rookie programmers for the software development industry. New developers also tend to make light of security when writing programs and this becomes a habit that negatively affect software industries. The primary objective of this study is to determine how negligent students are in writing secure code, analyze their complacency and understand the effect it has on new developers in the software development industry. To achieve this objective, two surveys were created. The first survey was to understand students' views about secure coding and collected code samples from students. The second survey was structured to collect senior managers' view about new developers

programmers when they first get started in the programming industry. Codes samples were then analyzed to find frequently occurring common mistakes and then compared students' common mistakes to Common Vulnerabilities and Exposures database (CVE). Professional developers were also asked about the common mistakes these new developers make to understand what the industry expects from them. The results suggest that students rarely care about security while programming. 60 participants out of 98 focused more on the proper functioning of code as compared to the security aspects of code. About 30% of the participants have never considered the security of a program they developed and 93% of the participants among them intend to pursue a career in a software programming field in the future. Based on these findings, it is essential to strengthen security education at the academic levels so that the students can be conscientious programming professionals. The results of the second survey shows that most managers are concerned about security and expect entry-level programmers to know a thing or two about software security. Close to 90% of managers suggest it will be a good idea for programming students to be knowledgeable about secure programming before they enter the industry.

Keywords: vulnerability, secure code, software security, functioning code, programming

MONTCLAIR STATE UNIVERSITY

A functioning code may not be a secure code: A Preliminary study on the
students' complacency with secure coding

by

Jeremiah Niiquaye Kotey

A Master's Thesis Submitted to the Faculty of
Montclair State University

In Partial Fulfillment of the Requirements

For the Degree of

Master of Science

January 2023

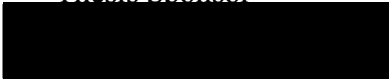
College of Science and Mathematics

Department of Computer Science

Thesis Committee:


Dr. Kazi Zakia Sultana

Thesis Sponsor

 12/16/22

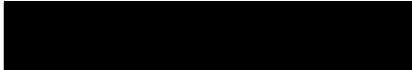
Dr. Bharath Samanthula

Committee Member

 12/18/22

Dr. Vaibhav Anu

Committee Member

 12/16/22

A FUNCTIONING CODE MAY NOT BE A SECURE CODE: A PRELIMINARY STUDY ON
THE STUDENTS' COMPLACENCY WITH SECURE CODING

A THESIS

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

by

Jeremiah Niiquaye Kotey

Montclair State University

Montclair, NJ

2023

Copyright@2022 by Jeremiah Niiquaye Kotey. All rights reserved.

Acknowledgements

First and foremost, I want to thank The Lord Jesus Christ, for the inspiration I got to do my master's and making this possible. Glory to God! Secondly, I am extremely grateful to my supervisor, Dr. Kazi Zakia Sultana for her invaluable advice, continuous support, and patience during my master's study. Her immense knowledge and plentiful experience encouraged me in all the time of my academic research. I want to express my gratitude to Dr. Bharath Samanthula for his advice and support from day one, when I started my master's program at Montclair State University, not forgetting Dr Vaibhav Anu for his support during my final parts in the program. I would like to thank the management at R.Seelaus and Co. for their financial support and the opportunity to learn from them. It is their kind help and support that has made my study successful. Finally, I would like to express my gratitude to my parents, my wife, and my child. Without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Contents

Introduction.....	10
Related Work.....	13
Methodology.....	14
<i>Research Questions.....</i>	<i>15</i>
<i>Data Collection.....</i>	<i>18</i>
<i>Tools.....</i>	<i>18</i>
<i>Survey Participants.....</i>	<i>19</i>
<i>Survey Questions.....</i>	<i>21</i>
<i>Data Analysis.....</i>	<i>22</i>
Results and Discussions.....	24
Threats to Validity.....	35
Conclusion.....	37
References.....	39

List of Tables

Table 1: Growth in Software Developers by Year.....	20
Table 2: Project Demographics: Level of education.....	20
Table 3: Project Demographics: Rating of programming skill	20
Table 4: Project Demographics: Country of college	20
Table 5: Project Demographics: Number of students that know a specific language	20
Table 6: Table 2: Survey (S1) Questions	21
Table 7: Table 2: Survey (S2) Questions	21
Table 8: Table 2: Survey (S2) Questions	21
Table 9: Table of CVEs & CWEs.....	33

List of Figures

Figure 1: Sample Code.....	16
Figure 2: Determining factor for programming mistakes by students.....	23
Figure 3: Determining factor for programming mistakes by new developers.....	23
Figure 4: Responses for S1.2.....	25
Figure 5: Responses for S1.4.....	26
Figure 6: Responses for S2.2.....	26
Figure 7: Responses for S2.3.....	27
Figure 8: Responses for S2.4.....	27
Figure 9: Improper Initialization	28
Figure 10: Improper use of functions.....	29
Figure 11: Responses for S2.5.....	30
Figure 12: Error Handling.....	32
Figure 13: Knowledge in application domain.....	33

I. INTRODUCTION

The concept of software security must be tackled at early stage, preferably when students begin the journey of learning a programming language. It aims to enforce a set of practices that makes systems more secure and preventive from the cyber-attacks. In today's technological world, programming secure systems has become a top priority for developers. The encouraging thing is that many potential exploits and attacks can be averted through secure and attentive coding practices. Therefore, writing secure code should be the first line of defense for every application. If students become professionals, why shouldn't software security be a priority at the most fundamental level? Learning a particular programming language and knowing what it takes to write secure codes in that programming language must be closely associated. Even though many potential exploits and attacks can be averted through different code review processes, the goal is to avoid making mistakes. Vulnerability in a program is almost inescapable, but a system that follows good programming conventions is more defensive and harder to break for the attackers. According to the National Vulnerability Database (NVD), a repository of Common Vulnerabilities and Exposures (CVEs), there's an average of more than 50 different kinds of vulnerabilities, ranging from critical to low, logged each day which takes the vulnerability count to at least 1800 in a month and 20,000 in a year (NVD Dashboard). These figures represent reported vulnerabilities yet, there are some others that are not reported owing to "zero days". Students do not care about secure programming yet; these students are the same professionals who graduate in computer science and still struggle with the security aspects of their code. Vulnerability education tackles the realization that things can go wrong. A user might enter a string when the program expects an integer. A function that opens a file might fail. The program must check for these possibilities and handle them correctly. If these students are educated on secure programming guidelines and conventions,

there will be less time between software development phase, testing phase and deployment phase of a software development life cycle.

Vulnerabilities in code usually result from unmitigated oversight or lack of skills of the programmers in properly using logic functions, arithmetic calculations, and variable assignment (Kotey et al, 2021). Wrong variable declaration is another fundamental cause of most programming vulnerabilities (Kotey et al, 2021). There are many static analysis tools which can locate specific vulnerabilities in the source code. These tools often suffer from high false positive rates and fail to detect all the vulnerable files (Nadeem et al, 2012). This might occur if a new vulnerability is discovered in a system or if the analysis tool has no understanding of the run-time environment (Dewhurst, 2022). The aim of this study is to observe common students' mistakes that lead to vulnerabilities so that future tools can curtail these mistakes in a form of vulnerability education. This study will assist to improve these static analysis tools to better understand human behavior pertaining to secure programming and thus ensure effective vulnerability education.

When vulnerabilities are exploited, the impact can be insignificant or detrimental, depending on the attack's nature. New developers make mistakes that experienced programmers might have come across before. In the survey, some managers claim new programmers rush into coding without creating a plan. And also, new programmers do not modify modules and functions which might contain bugs or skip to observe how modules will affect their programs. Some technology organizations take new developers employees through rigorous on-the-job training before they are allowed to work on live programs but is that enough?

Negligence is one human behavior that contributes to vulnerabilities in programs. Negligence relating to ignorance or negligence pertaining to disregarding security. The point is,

if software security is not prioritized at fundamental stages of programming education, most students, hence new developers will not regard security with utmost importance. Before a program is considered secure, it must follow secure programming guidelines and conventions. Most programs go through several review processes before being considered secure given that programmers fail to follow the right guidelines and conventions and attackers are always getting creative thus finding ways to exploit vulnerabilities. During software testing, testers mostly ensure that the program functions correctly. They do not tackle security to the core. A program needs to be tested thoroughly in order to reduce any hangs or lags in processing, as majority of coding errors occur in data processing with out-of-bounds read and out-of-bounds write being top of the list (Hladun, 2022). These mistakes cause buffer overflow which makes up to more than 35 percent of vulnerabilities (Hladun, 2022). Research found that buffer overflow is one of the most widespread and frequently reported vulnerabilities that result in system crashes (Kotey et al, 2021). Therefore, vulnerability education should not be limited to organizational practices in software production environments. If students are educated on secure programming guidelines during their academic years, they will become good developers and testers in the future. Organizations will need to spend less effort and time during software testing.

This research study will contribute to the software security and training in the following ways:

- It will help understand the need for software security education at the academic levels.
- It will make students professionally prepared for industry and help them to write secure code as a professional developer.
- It will motivate software security education and thereby will contribute to lessening the need for training resources, cost, and effort for new developers in the industry.

II. RELATED WORK

This section will focus on some related works on vulnerability education and software security. Earlier studies focused on creating awareness on the frequent causes of vulnerabilities in code. Kotey et al (2021) found that lack of input sanitization, improper checking of array bounds and parameters, and the lack of value and range checks on variables are the most common programming issues that lead to a buffer overflow. They also reported that improper use of “If” and “While” loop conditions frequently contribute to the errors in bounds and variable checks. Taeb & Chi (2021) proposed a model by preparing a set of hands-on labs that will introduce students to secure programming practices. They used source code and log file analysis tools to predict, identify, and mitigate vulnerabilities. There are some other research on developing curriculum for theory and hands-on security education for the students. These courses were developed to train students as skilled users so that they do not become the victims of cyber-attacks. According to a study Pothamsetty (2005), disseminating knowledge in security technologies, attack techniques and tools, cryptographic mechanisms cannot contribute to demolishing vulnerability, rather students need to be trained on mitigating vulnerabilities through secure programming in the undergrad level courses. When it comes to cyber security, prevention is better than control thus solutions or fixes, because the least of vulnerabilities, when exploited by an attacker can cause massive damages to an organization.

Imtiaz et al, (2021) also proposed a model based on association rule mining to discover the relationship between a security flaw and a corresponding fix, by deploying the classical Apriori algorithm. Their proposed model will work on the basis of X implies Y. In the context of secure software engineering, X could be an attack type in which case Y would denote the relevant security flaw. More commonly, X could be a flaw in which case Y would be a fix. The

model is based on a logic that, attack type leads to flaw and flaw leads to a fix.

Online vulnerability databases and awareness creation also plays a role. CVE (Common Vulnerabilities and Exposures) and CWE (Common Weakness Enumeration) are both publicly useful online databases that aid vulnerability education by sharing information and resources. The database records act as vulnerability dictionary and helps programmers to make proactive cautions and remediation to avoid these vulnerabilities. It also standardizes weakness across different programming languages and makes programming conventions easy to understand. Before CVE was created in 1999, there was no centralized list of common identifiers that made it possible to share information across multiple information sources, databases, tools, and services (Jelen, 2019). These security databases have helped programmers and organizations in general to avoid weaknesses in their systems. Both CWE and CVE use a score rating system to help programmers understand the significance of a reported vulnerability. Also, both CVE and CWE are recognized using identifiers or entries, usually CVE or CWE followed by “year added” and unique number. For example, CVE-2022-26809 - "Remote Procedure Call Run-time Remote Code Execution Vulnerability".

This study is different from past studies because it seeks to analyze students’ programs to find and understand the common mistakes that students make that lead to a vulnerability. Other studies generalized this assertion. Also, this study will aim to find the differences and similarities having to do with the common mistakes that lead to vulnerabilities between students (new developers) and professional developers.

III. METHODOLOGY

In this section, the research questions, survey questions, mode of data collection and data analysis will be discussed.

1. Research Questions

The goal of this study is to determine whether students care about security in programming and if software security education at the academic levels is related to the professional development of the software programmers? The objective of this study is to analyze students' code in addition to some secure programming questions and to ask professional developers about their views of new developers with regards to secure programming. To better develop research questions, these hypotheses were formed:

H.1: *Students do not care about the security in their code, they only care whether the code solves the problem it is designed for.*

H.2: *Entry level students lack the standard software security education required to be successful in the software programming industry.*

The subsections briefly describe the questions that were asked to get the goal of this study.

A. Importance of software security education at academic levels and benefits in programming industry

The software industry is a rigorous and fast-paced environment that expects accuracy, efficiency and effectiveness from professionals but are entry-level programmers ready when hired? Most managers expect new developers to have a fundamental idea about security. Most new developers tend to focus more on the functions of the program rather than its security. Also, they tend to try and beat timelines of projects, thereby ignoring or being eluded by all security measures and guidelines. Generally, students work to write a functioning code which may or may not be secure. But do students pay attention to the security of their code too? Students tend to stop at a functioning code without reviewing to see if it meets proper secure coding conventions. The SQL code below shows a submission from a student during the survey.

Figure 1: Sample Code

```
SELECT k.studentId, k.firstName, k.Lastname, k.dob
FROM StudentData k
WHERE k.hashValue = ''
AND k.student = 'nhinkson'
AND k.passWordx = 'HoldidayDet4'
```

For the code above, when "--" is added to the username, it becomes nhinkson "--". This will comment out the last part of the authentication process, giving an attacker privileges to the system. Double hyphen when used in a SQL statement or PL/SQL block, adds trailing comments to a line. In this common case of SQL Injection, the double hyphen will comment out the last line of code, there skipping that code during the authentication process. This was evident in the NoseRub protocol attack in 2007 (Groebert, 2007). To understand how important security education is to the programming industry, this first question was asked:

RQ.1: *To what extent is software security education at the academic levels related to the professional development of the software programmers?*

B. Most frequent programming mistakes made by students that lead to a vulnerability

Software developers make some mistakes in their code that can lead to vulnerable code resulting in security breaches. For example: a wrong logic, unchecked array bounds, incorrect and non-descriptive variable definition, or an arithmetic calculation done incorrectly can occur frequently in code due to an oversight or lack of knowledge. Every programming language has its own conventions and practices for ensuring secure code and ignoring them can end up with developing a vulnerable system. It is contingent upon the programmer to have the knowledge required to make a program secure. If students are enforced to follow security conventions while coding, they can build the practice of writing secure code from the earlier stages which will make a significant impact on secure software development in the industry. Therefore, in order to

better understand common coding mistakes made by the new developers, the second question was:

RQ.2: *What are the most frequent programming mistakes by the students that lead to a vulnerability?*

C. Most frequent programming mistakes made by new developers that lead to a vulnerability

Software companies expect new developers to be equipped with the fundamental knowledge to make a program functional while regarding security. Just like students, new developers have lapses when it comes to secure programming. With this in mind, the third question was asked:

RQ.3: *What are the most frequent programming mistakes (irrespective of Programming Languages) made by new developers that lead to a vulnerability?*

D. Programming mistakes of students and new developers vs. professional programmers

Every software developer is bound to make mistakes at some point, but experience is key regardless of a programming language. Experienced programmers are used to programming conventions and good programming habits which new developers are not used to. This could be because professional developers have firsthand experience of what a programming mistake cause. Also, professional developers tend to pay close attention to security when programming because security is a widely used concept in the programming industry. Yet, there are some similarities in programming mistakes made by new developers and professional developers. This subsection aims to compare students' mistakes vs. professional programmers' mistakes that lead to vulnerable code. A comparative study of the students' programming mistakes with those by the professional developers will help us to better design vulnerability education courses in the academic levels. This leads to the next research question:

RQ.4: *Are there any similarities and variations between programming mistakes by the students(new developers) and those by the professional developers as recorded in CVEs?*

E. Skills, experience, and educational requirements for good developers

Every programmer needs some form of programming education whether informal or formal. Formal in the sense of being taught and informal in the form of self-taught. Formal education guides a programmer through the basics of programming. Beyond the applied knowledge a person acquires from learning to code, a degree in computer science related field will give a programmer a near-complete background and affiliation to a community in computer science to use as a foundation for your knowledge and this involves principles of secure programming which self-taught programmers lack. Owing to lack of community, most self-taught programmers lack the use of industry-wide conventions and principles of secure coding. This brought us to the final question which is:

RQ.5: *What kinds of professional skills could overcome common programming mistakes, leading to software vulnerability, made by new developers?*

2. Data Collection

A. Tools

Survey planet, an open-source online survey solution tool that provides an easy-to-use interface for data collection and analysis was used for the surveys. Two surveys were developed for this study. For the first survey, data was collected from ninety-eight (98) students from seven (7) different countries and for the second survey, data was collected from thirty-seven (37) Professional developers from eighteen (18) companies in the United States, India, and Ghana. Survey planet provides a sharing tool that made it easy to share survey with participants all over the world via a secure link and results are already segmented making it easy for analysis and drawing results. Data was then imported into an online graph representation platform called VISME to make the data more readable and presentable.

B. Survey Participants

For the first survey, participants were chosen from seven different countries as shown in table 4. The project population was not just limited to one country to help understand how students perceive secure programming in different countries around the world. According to a google report, Africa now has 716,000 software developers, a 3.8 percent rise in 2022 and a figure that will rise even more in the coming years (David et al, 2021). This stresses the fact that more people are taking on programming and also stresses the need for secure programming. The participants are either enrolled in regular classes or self-taught students and they have knowledge about programming at different levels (Beginner or Intermediate or Pro as shown in table 3). In this project, a *Beginner* was defined as a student who has started writing programs or learning how to write programs. An *Intermediate* level student can write a functional program that will somehow serve the purpose; a *Semi-pro* student can write a fully functional program and also can find out any problem happened inside it through debugging; a *Professional* student can write a fully functional and secure program and can also teach other students about different aspects of programming. An *expert* is a student who is capable of being a part of an organization and writing live programs. A participant could be knowledgeable in more than one programming language as specified in table 5.

Thirty-seven (37) Professional developers for survey from eighteen (18) different companies in the survey process. The project population was not just limited to one country to help us understand how other nationalities perceive secure programming. For the criteria for selection, participants were expected to be all managers or supervisors with some knowledge about entry-level developers and be an expert in at least one (1) modern programming language.

The software industry is fast growing with more than 4 million professional developers in

the United States, a number which is expected to double by 2030 according to Statista.com and there are more than 27 million professional developers currently in the world, a number that's expected to grow to 45 million by 2030 (Team, 2021).

Table 1: *Growth in Software Developers by Year*

Year	Number of Software Developers
2018	23.9 million
2019	26.4 million
2021	26.9 million
2023	27.7 million
2024	28.9 million
2030	45.0 million

Table 2: *Project Demographics: Level of education.*

Bachelor's Degree	62
Master's Degree	19
Diploma	12
Self-taught	5

Table 3: *Project Demographics: Rating of programming skill*

Beginner	18
Intermediate	26
Semi Pro	39
Professional	12
Expert	3

Table 4: *Project Demographics: Country of college*

United States	33
Ghana	52
India	2
Nigeria	2
Tanzania	1
Liberia	1
Netherlands	1
Ivory Coast	1
South Africa	1
United Kingdom	4

Table 5: *Project Demographics: Number of students that know a specific language*

Python	59
C++	24
Java	31

SQL	58
Perl	4
PHP	27
Other	18

C. Survey Questions

For survey one (S1), demographics of the students were collected while designing the survey. The core questions were based on identifying their concerns about secure programming. In this study, a secure code was defined as a functioning code that follows security guiding principles and written with security in consideration. Secure code can defend against cyber-attack. On the other hand, a functioning code just solves the problem. Therefore, a functioning code may or may not be concerned with security. It focuses more on “if the program works”.

Below are examples of some survey questions:

Table 6: Survey (S1) Questions

S1.1: <i>How would you rate your programming skill?</i>
S1.2: <i>What is more important to you when completing assignments or projects?</i>
S1.3: <i>Have you ever considered the security of a program/code you wrote?</i>
S1.4: <i>Do you intend to pursue a career in a software programming field in the future?</i>
S1.5: <i>Which programming languages are you knowledgeable about?</i>
S1.6: <i>If you had rules to guide you when completing school programming assignments/projects to avoid little mistakes that could cause a vulnerability, would you consider these rules?</i>
S1.7: <i>Have you ever used a static analysis tool?</i>

For Survey two (S2), while designing the survey questions, the location and name of organization of the participants to get an understanding of nation-specific or organization-specific perspective pertaining to secure programming was collected. Below are the examples of some survey questions:

Table 7: Survey (S12) Questions

S2.1: <i>Which programming languages does your company work with?</i>
S2.2: <i>How would you rate the importance of security education at the academic levels for</i>

<i>professional success?</i>
S2.3: <i>How is a computer program reviewed for ensuring security during development in software industry?</i>
S2.4: <i>Will it be useful if new developers are trained for using the automated review tools before they join the industry?</i>
S2.5: <i>Which of the following are the most frequent vulnerabilities you observe in the new developer's code?</i>
S2.6: <i>Which of the following common weaknesses related to security are frequently observed in new developer's code?</i>
S2.7: <i>Which of the following vulnerabilities are common both in the code of new developer and code of the professional developers?</i>
S2.8: <i>Which of the following vulnerabilities are found only in the code of new developers?</i>
S2.9: <i>New programmers from which of these education levels make the fewest coding errors?</i>
S2.10: <i>Which of these academic majors/minors can provide good training to mitigate potential programming errors?</i>
S2.11: <i>What is the correlation between a given programming language proficiency and the skills needed to overcome common mistakes? (e.g., C, Python, Java etc.)</i>
S2.12: <i>Is a good knowledge of the application domains useful to reduce programming errors? (e.g., Real Estate, Banking, Medical Robots, Patent Law, Stock Markets etc.)</i>

D. Data Analysis

Both quantitative (for numerical data) and qualitative (logical data) methods were used to analyze the survey data. By process of cross tabulation, data was broken into subgroups to obtain variables. ANOVA (Analysis of Variables), an analysis tool used in statistics that splits an observed aggregate variability found inside a data set was used to divide data into parts (Kenton, 2022). This helped to statistically model the relationship between all variables and analyzed the relationship between all variables using the level of education as key.

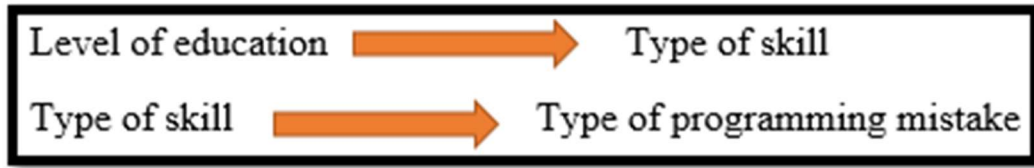
For the first survey (S1) thus survey of students, the level of education is the deterministic factor for the level of skill and the level of skill determines the kind of error that the student is likely to make. These three (3) variables were formed, and the model below was adopted as shown in figure 2.

V.1: Level of education

V.2: *Type of skill*

V.3: *Type of programming mistake*

Figure 2: *Determining factor for programming mistakes by students*



For the second survey (S2) thus survey of professional developers, the idea was to find the relationship between new developers and the common mistakes they make that lead to a vulnerability. To understand this, a relationship had to be created between these variables according to the data in the survey. For this to be effective, these five (5) variables were formed. Also, having proven knowledge in the application domain determines if a new developer has what it takes to mitigate programming mistakes. This model was adopted as shown in figure 3.

V2.1: *Level of education*

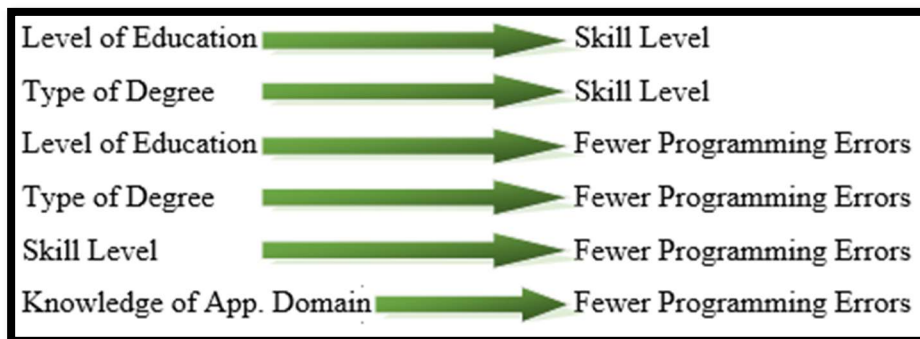
V2.2: *Type of degree(major)*

V2.3: *Knowledge in application domain*

V2.4: *Skill level*

V2.5: *Fewer programming errors*

Figure 3: *Determining factor for programming mistakes by new developers*



Sample codes were collected from students for analysis to determine some common mistakes they make when programming. The sample codes were compiled using another tool named as “SonarQube”, an open-source static analysis tool to better understand and make the code review easier. SonarQube is a web-based platform in Java and can analyze and manage code of more than 20 programming languages including C/C++, PL/SQL, Cobol etc. through plugins (Vizteck, 2016). In order to avoid the false positives, which most static analysis tools tend to report, a manually review was done to verify the results of the tool.

IV. RESULTS AND DISCUSSIONS

In this section, results from both surveys will be discussed. Secure programming is strictly enforced in the software programming industry so, security in programming must be valued analogous to the manner in which program is designed to function to help equip students and new developers with the skills they need to be successful in the software industry because students grow to become professionals. This survey-based analysis corroborates the hypotheses that students do not focus on security when coding and new developers lack the security skills to program secure and industry-standard applications. In this section, the findings for each of the research questions designed for this study will be discussed.

RQ.1: *To what extent is software security education at the academic levels related to the professional development of the software programmers?*

The responses of five (5) survey questions(S1.2, S1.4) and (S2.2, S2.3, and S2.4) from S1 and S2 respectively, tackles this research question. Responses have been displayed in figures 4, 5, 6, 7, and 8 respectively. Out of 98 students, 62% claimed a functioning code is more important to them than a secure code. Of the 38% that cared about security, after analyzing their sample codes, it was clear that most of them made basic mistakes in programming especially variable assignment. Out of the 98 students, 93%intended to pursue a career in a software programming

field in the future as shown in figure 5. Secure programming practices and conventions must be enforced in the basic programming courses to build the skills among the students. Most students, (60%) find the vulnerability education as helpful because they feel the need for secure programming guidelines.

More than 90% of professional developers deem security education at academia level very important as it equips new developers with everything they need to be successful the industry. Having knowledge in security when programming means developing secure systems which is the number one issue currently in the industry. Also, students have to be exposed to static analysis tools and different automated code review tools and techniques to help detect vulnerable codes. This will also help students to develop good analytical skills, which every programmer needs.

Figure 4: Responses for S1.2

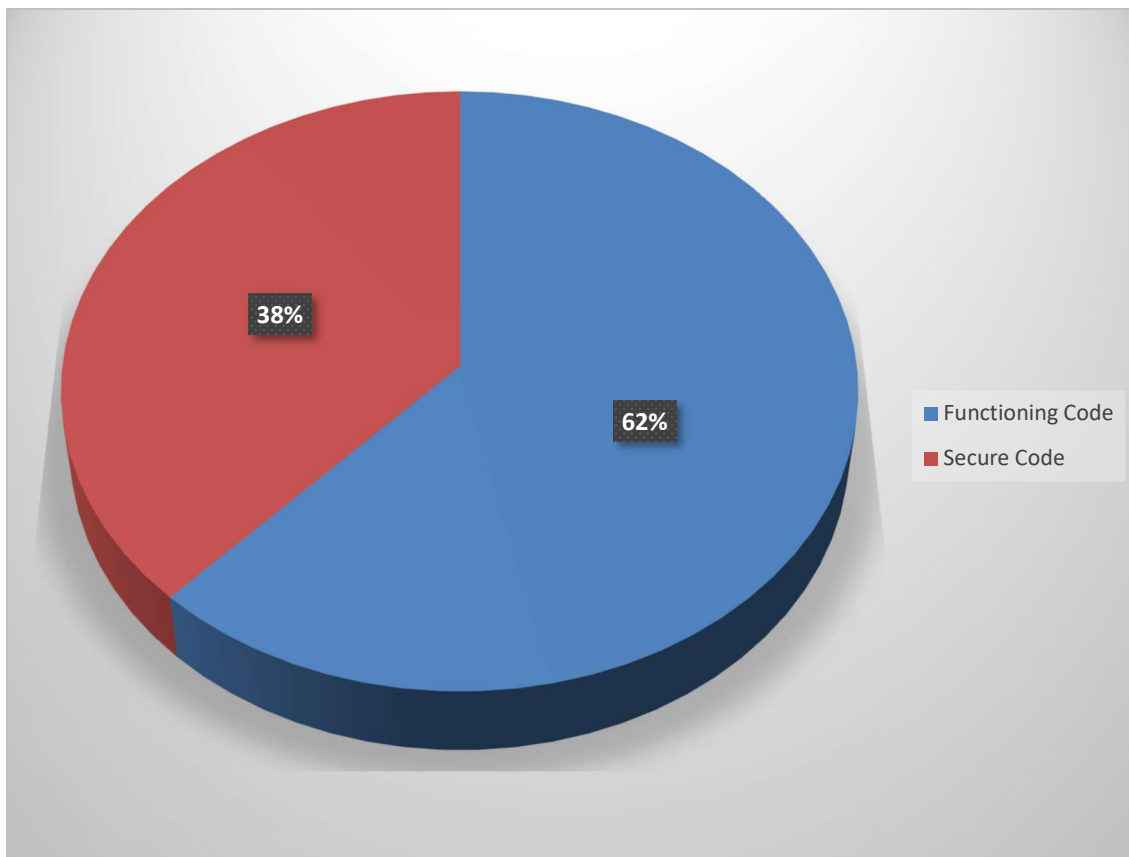


Figure 5: Responses for S1.4

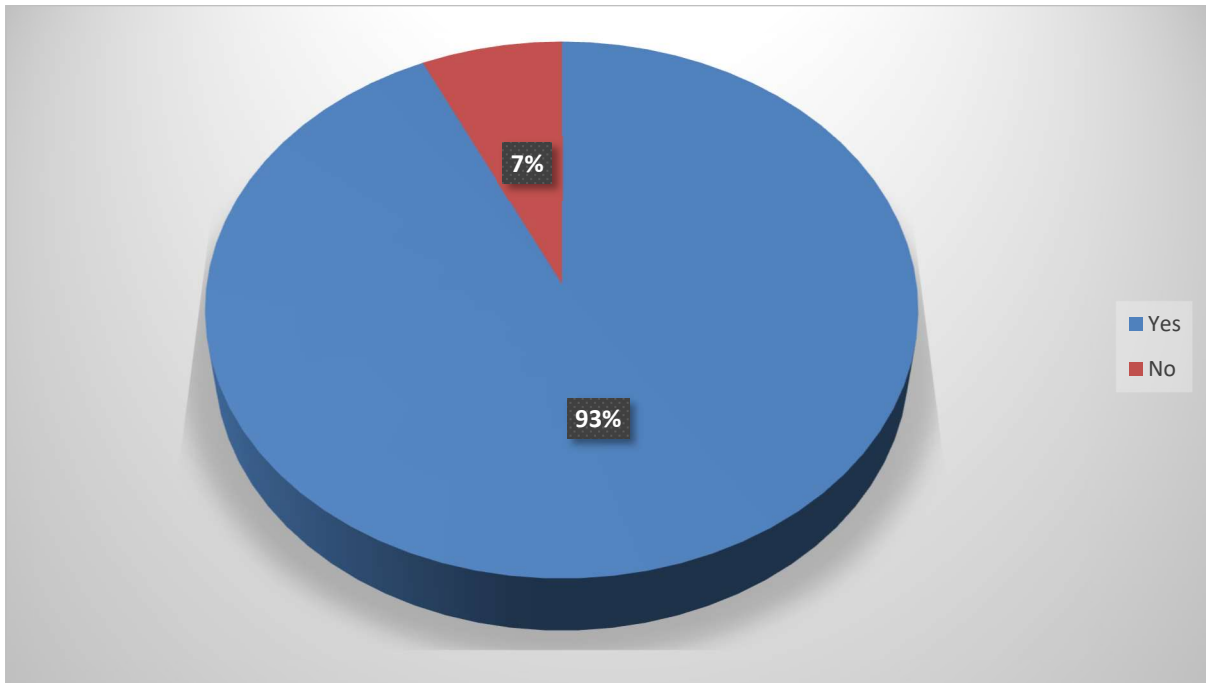


Figure 6: Responses for S2.2

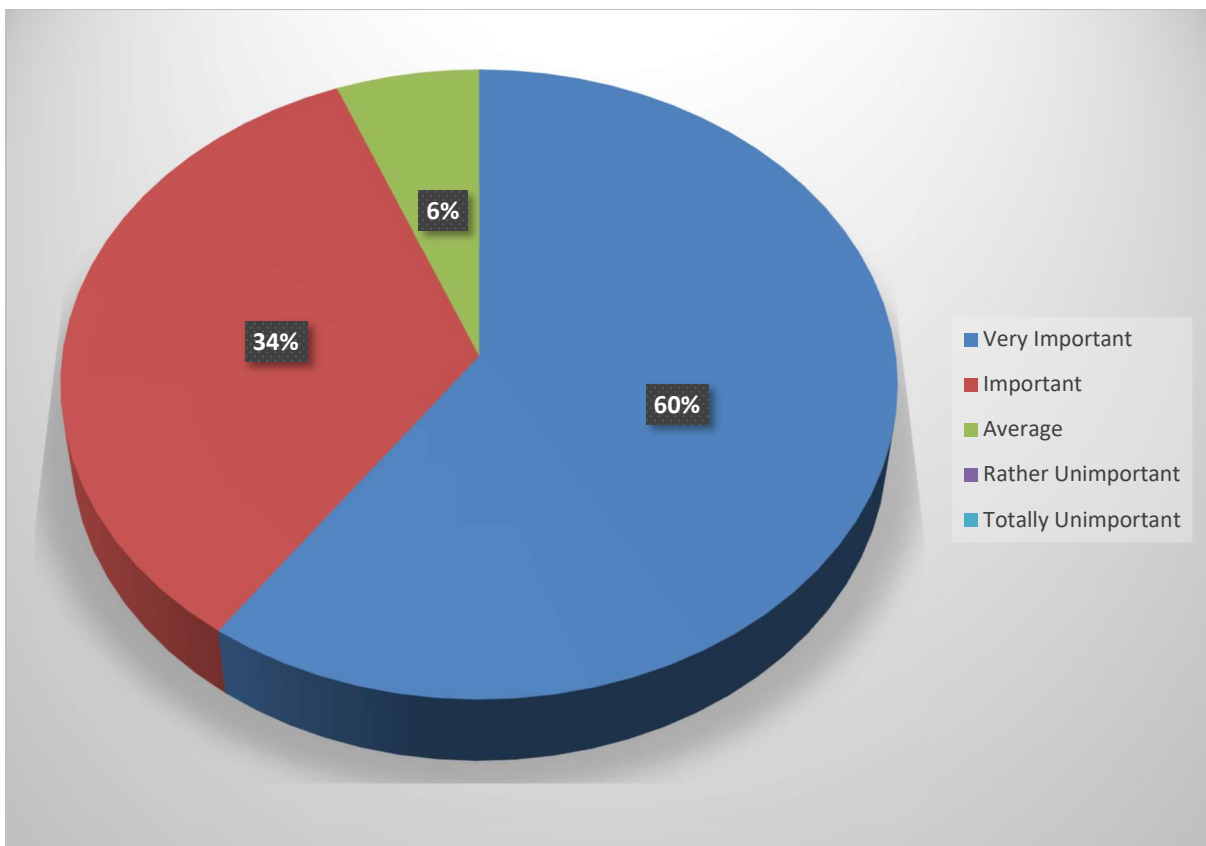


Figure 7: Responses for S2.3

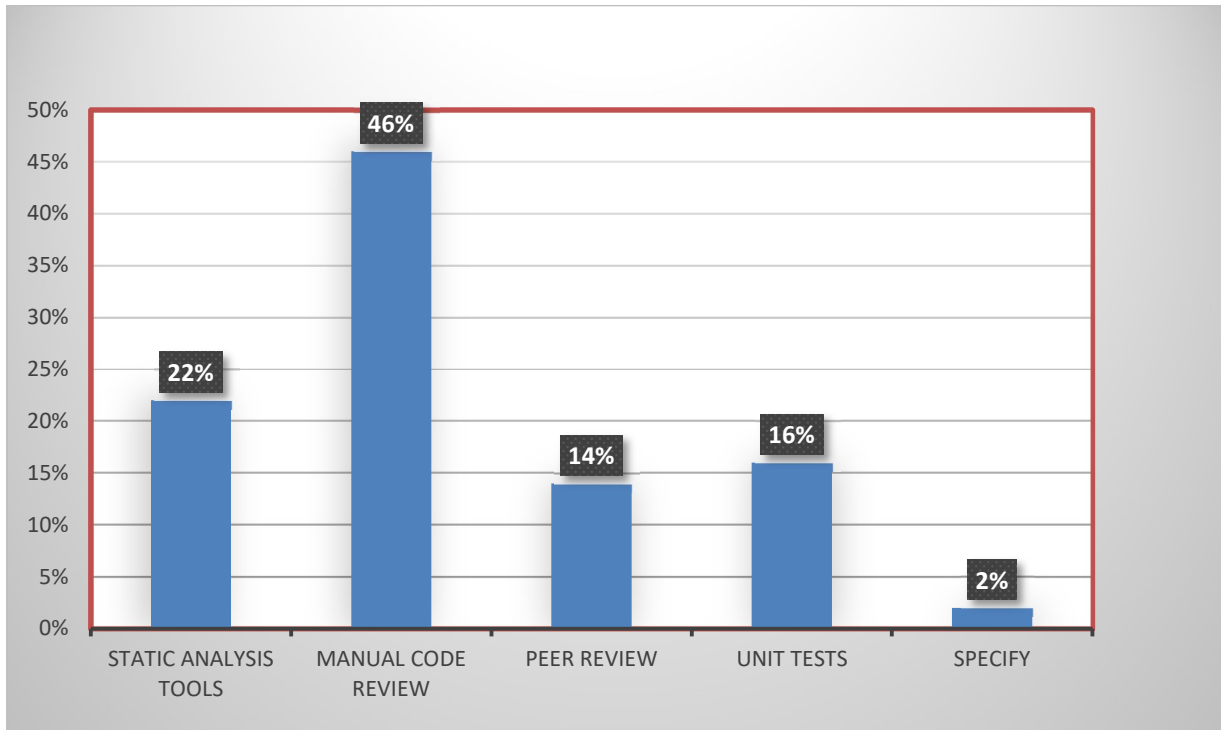
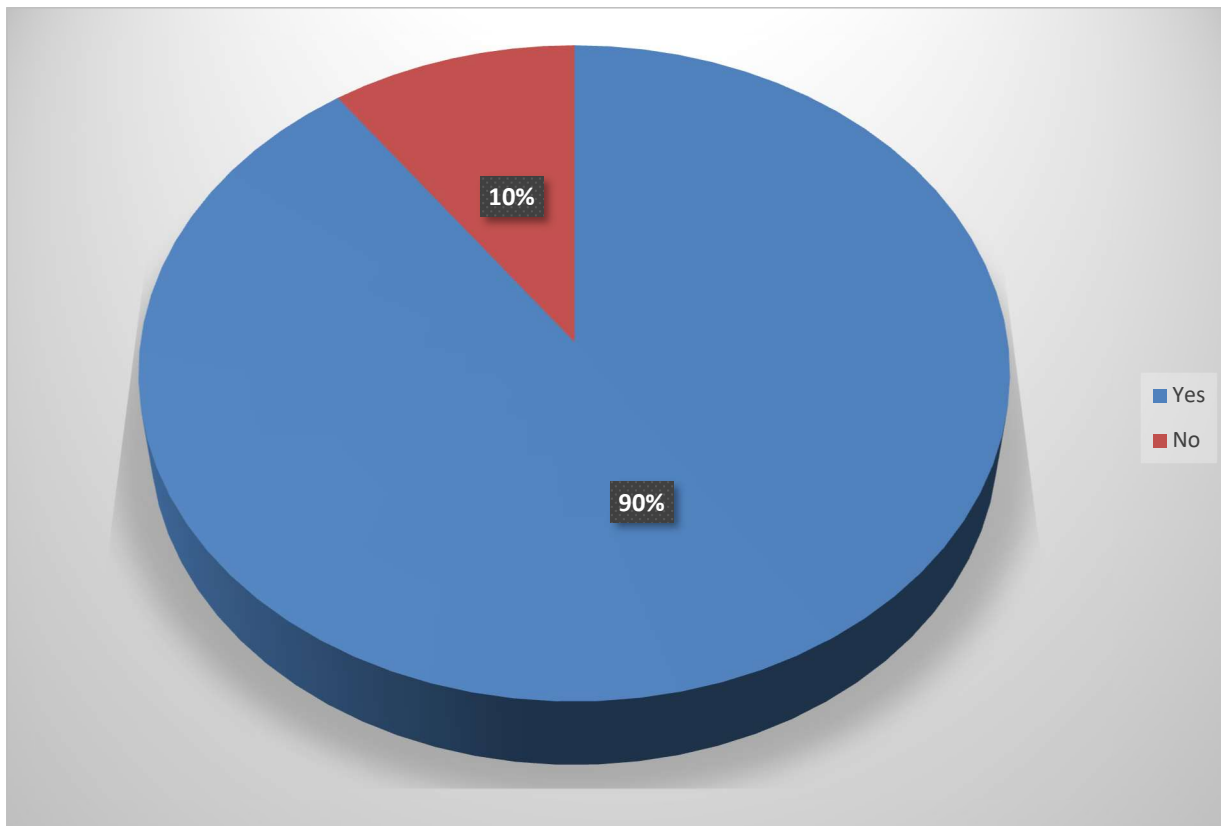


Figure 8: Responses for S2.4



RQ.2: *What are the most frequent programming mistakes by the students that lead to a vulnerability?*

RQ.2 focused on four (4) survey questions (S1.1, S1.3, S1.5, and S1.7). and sample codes collected from the students. There were a couple of mistakes pertaining to students' code.

Although some of the mistakes may not be directly causes for vulnerabilities, they can indirectly make the code prone to vulnerabilities. Most students' codes were copied from open-source code libraries. These codes are not peer reviewed and thus do not follow security policies. Attackers are usually familiar with open-source code and know how to exploit these codes. Some of the most frequent mistakes made by students are as follows:

- Students do not initialize their variables which presents an unexpected behavior of their programs when certain conditions are not met. A variable that is not initialized does not have a defined value, hence it cannot be used. *Improper Initialization* has security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not (CWE-665). An attacker can manipulate an uninitialized value which can cause denial of service vulnerability. An example can be seen in figure 9; the string “city” needs to be initialized.

Figure 9: *Improper Initialization*

```
public class Assignment6 {
    public static void main(String[] args) {
        String city;
        System.out.println(line);|
        System.out.println(line);
    }
}
```

- Students struggle in using functions properly. They do not take care when using functions like `gets()` and `strcpy()`, which is one of the most frequent causes for buffer overflow attack(Owasp). One sample code as shown in figure 10 presents a student's authentication system that can cause buffer overflow. The `gets()` function does not check the array bounds and can even write string of length greater than the size of the buffer which will cause a buffer overflow. For example, if "xxxxxxxxaaabbbxxxxxxxx" is entered as the password, root privileges will be granted because the input length is greater than what the memory can hold so, the value of "correct" does not become zero. Another common mistake that evident in students' code is writing logic for "If" loops.

Figure 10: *Improper use of functions*

```
int main(void)
{
    char checkLength[15];
    int correct = 0;

    printf("\n password : \n");
    gets(checkLength);

    if(strcmp(checkLength, "assignment3"))
    {
        printf ("\n Password incorrect\n");
    }
    else
    {
        printf ("\n Password accepted\n");
        correct = 1;
    }

    if(correct)
    {
        printf ("\n Root user successfully logged in \n");
    }

    return 0;
}
```

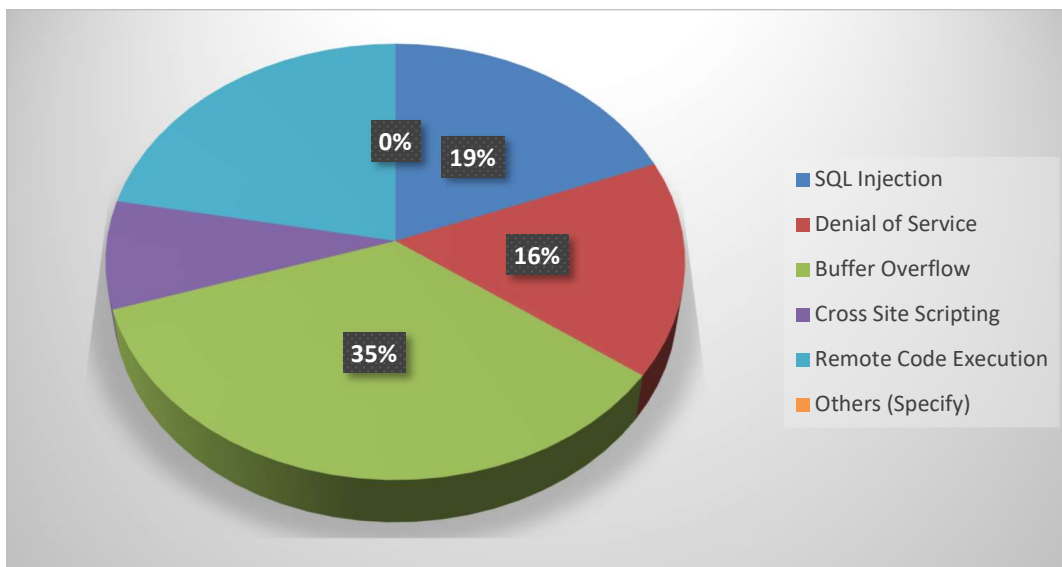
RQ.3: *What are the most frequent programming mistakes (irrespective of Programming Languages) made by new developers that lead to a vulnerability?*

For RQ.3 focused on three (3) survey questions (S2.1, S2.5, and S2.6). Professional developers reported that, new developers make lots of mistakes that lead to buffer overflow as shown in figure 11. Some of the common mistakes were, unchecked return value, out of bounds RW, integer overflow, weak password management, improper validation of array index,

improper/missing initialization, as represented. All these mistakes especially out of bounds RW and integer overflow can lead to buffer overflow which can in turn lead to a "Denial of Service". Unintended integer overflows can cause memory corruption or information disclosure vulnerabilities in variables associated with memory accesses or memory allocations (Hamilton, 2016). Also, new developers tend to extract codes from open-source code libraries which might contain a vulnerability. New developers are not familiar with or knowledgeable about input validation and code sanitization which deals with deciding what input to accept, and filtering and modifying data to meet certain criteria. Input validation and code sanitization helps mitigate cross-site scripting which the second frequent mistake made by new developers in figure 11, according to the second survey.

Some managers reported that, new developers lack analytical skills and fail to plan the structure of their code thereby making code review very difficult. It will be difficult to detect vulnerabilities with Static analysis tools when the code is unstructured and even more difficult for peer and manual reviews. It also takes experience to be used to industry-wide programming conventions which most new developers lack.

Figure 11: Responses for S2.5



RQ.4: *Are there any similarities and variations between programming mistakes by the new developers and those by the professional developers?*

For RQ.4 focused on two (2) survey questions (S2.7, and S2.8) and analysis of the sample codes collected from the students and also analyzed data on “Common Vulnerabilities and Exposures” (CVE) databases. Whether an individual is a student or professional developer, all programmers must follow the same conventions to enhance security and all-round good programming. Programming is like a language and a common language has to be spoken by all programmers of the same language. For the first survey (S1), observations showed that, there are some similarities and differences between professional programmers' and students' codes. For example, CVE-2003-0968 (Stack-based buffer overflow in SMB-Logon-Server of the rlmsmb experimental module for FreeRADIUS 0.9.3 and earlier allows remote attackers to execute arbitrary code via a long User-Password attribute) can be compared to the buffer overflow as shown in figure 10. In this vulnerability, there exists a stack buffer overflow in rlm-smb module which can be triggered by a long User-Password attribute (greater than 128 bytes)(S-Quadra Advisory, 2003).

Another difference between professional developers and students is error handling. Professional developers handle errors in code properly by throwing exceptions as shown in figure 12. Throwing exceptions promotes complex error handling code that is more likely to contain security vulnerabilities (CWE-397). Improper handling of errors can cause denial of service by causing the system to crash or use important resources, ultimately denying or reducing service to authorized users. Figure 12 shows an example of error handling by student verses error handling by professional developer in CVE-2022-28463 (ImageMagick 7.1.0-27 is vulnerable to Buffer Overflow)~CWE-252: Unchecked Return Value. In this case, the developer inserted a "ThrowReaderException" to check any issue of a crash that could lead to a denial of service.

Also, professional programmers maintain the habit of inserting a return value in method of function which students do not. Failing to insert a return statement could leave the program in an unexpected state. CWE-252: Unchecked return value confirms this assertion. Kotey et al (2021) found that unchecked value is one of the common cause of buffer overflow.

Figure 12: *Error Handling*

Student Code	Professional Developer code CVE-2022-28463
<pre>if (rate < 0) { System.out.println ("The rate is: " + rate) }</pre>	<pre>cin.file.image_offset=ReadBlobLong(image); if (cin.file.image_offset < 712) ThrowReaderException (CorruptImageError,"ImproperImageHeader");</pre>

Table 9: *Table of CVEs & CWEs.*

TABLE OF CVEs and CWEs		
CVE or CWE	Vulnerability Title	Reference
CVE-2003-0968	Stack-based buffer overflow	https://nvd.nist.gov/vuln/detail/CVE-2003-0968
CWE-665	Improper Initialization	https://cwe.mitre.org/data/definitions/665.html
CWE-397	Declaration of Throws for Generic Exception	https://cwe.mitre.org/data/definitions/397.html
CVE-2022-28463	ImageMagick - Buffer Overflow	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-28463
CWE-252	Unchecked return value	https://cwe.mitre.org/data/definitions/252.html

RQ.5: *What kinds of professional skills could overcome common programming mistakes, leading to software vulnerability, made by new developers?*

For RQ.5 focused on two (2) survey questions (S2.10, and S2.12) as well as verbal comments. Results were broken into subsections based on data collected and professional developers' view about reasons why new developers make mistakes. Professional skills that new developers require to overcome common mistakes that lead to software vulnerabilities include:

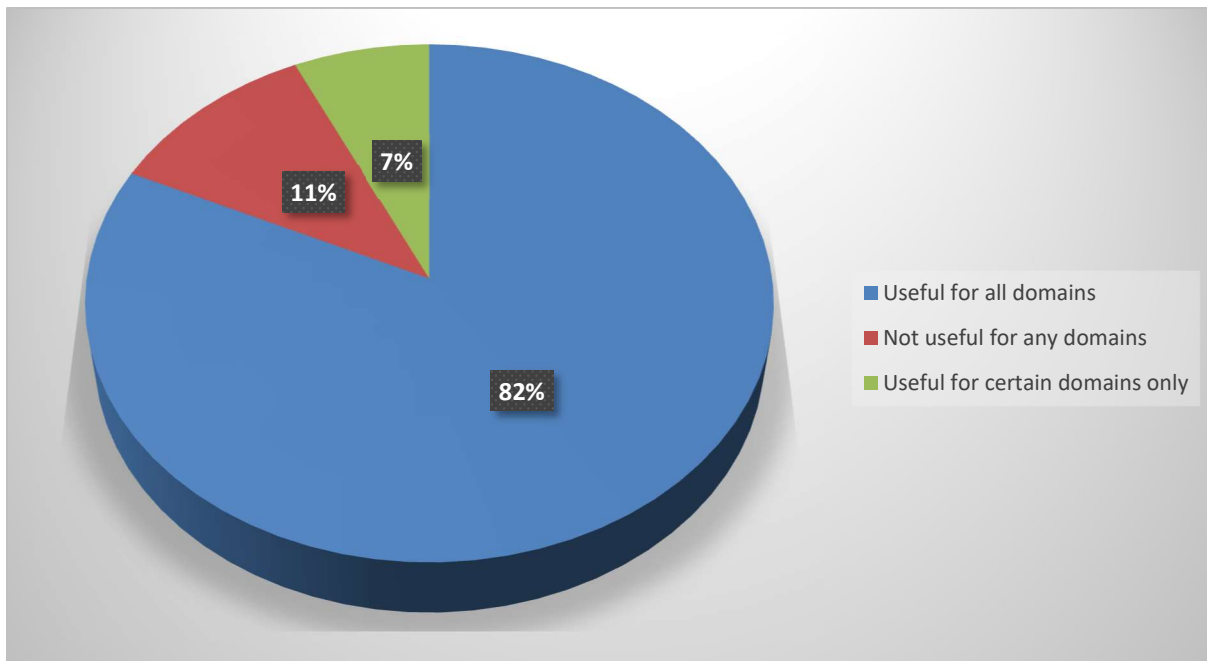
A. Good analytical skills

Good analytical skills involves critical thinking, in-depth analysis, creativity, problem solving and communication. Every software developer needs these qualities to be a good software developer.

B. Knowledge in application domain

Survey showed that developers need to have a background knowledge in the domain they program for. More 80% of the managers claimed that a good knowledge in application domain is good for reducing errors in programming as shown in figure 13.

Figure 13: *Knowledge in application domain*



C. Not being Negligent; Role of Negligence (The major difference between students & professional developers)

Students code for fun while professional developers write code for live and standard industry applications with no room for error or negligence. After observations in this study, it was clear that negligence was the number one factor that cause vulnerability in students' code. Negligence of secure coding in this study was defined as when students or programmers in general fail to follow programming conventions. With negligence of secure coding, a student may or may not care about the security of the program but cares about the functionality of the program. Software vulnerabilities have great variation among them, but in the end, nearly all come from preventable human error. Whether due to laziness, negligence, or a simple lack of knowledge, small errors in writing systems can have major problems when exploited (Hamilton, 2015). Negligence was broken down into primarily two parts thus, negligence pertaining to disregarding security and negligence pertaining to lack of knowledge and skill.

i. Negligence pertaining to disregarding security

Most students write programs with the aim to make it functional there by foregoing all security conventions and rules. The following are examples of negligence pertaining to disregarding security and programming conventions:

- **Bad variable naming:** it is helpful to be clear and concise when naming variables. Most students tend to choose unnatural variables that doesn't clearly define the code. For example, using "nad" to represent a names and addresses variable or using "let rates = 0.20" instead of "let interestRates = 0.20".
- **Using strings to code mathematical calculations instead of basic integers.** This issue is significantly supported by (Kotey et al, 2021) findings. Lack of input sanitization,

improper checking of array bounds and parameters, and the lack of value and range checks on variables are the most common programming issues that lead to a buffer overflow in these systems.

- Not planning the structure of the code. Students tend to start coding without proper planning thus, fixing things as they go. Writing an effective and secure code must start with good planning and design, knowing what to achieve with a single line of code and also consequences.

ii. Negligence due to lack of knowledge and skill

This is the most common form negligence. Most student programmers lack the knowledge and skill to be able to detect a vulnerable code. This was evident in question ten of the first survey. 90% of the participants could not detect buffer overflow in a short code. Students cannot write a secure code unless they understand the need for a secure code. At the most basic level, this means they need to understand the rules of their programming language well. Also, owing to the fact that most students consider themselves to be self-taught programmers, thus learning a programming language outside the school environment by themselves, and at their own pace, they tend to lack conventional and secure coding updates that are usually common in programming community. However, building relationships in a programming community is crucial in numerous aspects such that it helps programmers to keep up with the latest secure coding trends while sharing learning resources and building knowledge.

V. THREATS TO VALIDITY

In the following subsections, the three common threats to this study will be addressed.

A. Internal Validity

Internal validity is the concept of how much credible the results of research are. If there is

a possibility of any confounding factors, the results suffer from the threats of internal validity. In the experiments, code samples were collected from the students irrespective of their proficiency in respective programming language. That means the programming language used in a collected code sample has not been mapped to the language the student is skilled at. Therefore, if the student makes any programming mistake from security perspective in his/her code, there is a possibility that it was due to the lack of proficiency in that particular language, not due to the lack of security concerns. Also, new developers could study a particular programming language for a long time and not have to use it when they first start in the industry. They could be introduced to a new programming language they are not skilled at thereby making mistakes with security concerns.

B. External Validity

Threats to external validity are any factors within a study that reduce the generality of the results. In other words, it is the extent to which the results of a study can be generalized to and across other contexts. The study only included 98 students from less than 20 schools in 10 different countries, hence, the population size is not representative of the entire STEM student population in the world. To have a strong case of students' complacency with secure programming, a larger population has to be sampled from different schools and likely from different countries and also interview a lot more students to understand their perspective about secure programming. As this is a preliminary study of the importance of vulnerability education in the academic levels, the participants targeted were the ones reachable in the time constraints. With more time, a larger population could have been reached to make this study a lot more definitive. This study gave us a perception about the need of this type of education and common mistakes by the students which will help in the next endeavor of broad range study.

C. Construct Validity

Construct Validity defines the correctness of all observations and analysis made. Not every student submitted a code for analysis. Only about 60% of students willingly submitted a code. This reduced the success rate of determining if students really cared about securing coding practices. Also, one assignment or project from a student does not fully define the student's view about secure programming. Having at least 5-10 assignments or projects from a student will clearly create a pattern, thus giving you a signature of how that particular student writes codes.

VI. CONCLUSION

In this study, the complacence level in secure coding among 98 students explored and the views of 37 professional developers about the common mistakes that new developers make pertaining to programming were analyzed. Results found showed that there is complacency among students pertaining to secure programming. The results suggest that the lack of security proficiency in programming among students is an ongoing issue that must be paid close attention to as more that 60% of students did not care about security or have the knowledge to make their code secure. The major cause vulnerability in students' program is due to negligence, thus either negligence due to lack of skill or negligence due to laziness or disregarding security. Most students were honest and claimed that they have never considered the security of a program they developed, and that security is not a consideration when writing programming assignments or developing a system. The problem with this is that most of these students will eventually become professionals and not considering security when programming at that level could be drastic for developers.

In secure programming, the goal is to create a system that can withstand cyber-attacks. Preventing cybersecurity incidents, which can cause leaks of sensitive data and other personal

information, starts at the very beginning of the software development process with the source code itself (Morrow, 2022). Even though this is inevitable, following secure programming rules, practices and conventions could put developers on the bad side of attackers.

In a follow-up study, a larger population size will be the target and objective and reaching out to students in a lot more schools in different countries. This is because countries with less cyber-attacks could make the system developers care less about security. Also the plan is to create patterns and rules from the common mistakes that can be easily applied to machine learning. This will aid students to be more aware and create a platform for vulnerability education.

References

- Imtiaz, S. M., Sultana, K. Z., & Varde, A. S. (2021). Mining Learner-friendly Security Patterns from Huge Published Histories of Software Applications for an Intelligent Tutoring System in Secure Coding. *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021* (pp. 4869-4876). (Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021). Institute of Electrical and Electronics Engineers Inc.. <https://doi.org/10.1109/BigData52589.2021.9671757>
- Kotey, J., Ripley, M., George, G., Sultana, K. Z., & Codabux, Z. (2021). A preliminary study on common programming mistakes that lead to buffer overflow vulnerability. *Proceedings - 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021* (pp. 1375-1380). (Proceedings - 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021). Institute of Electrical and Electronics Engineers Inc.. <https://doi.org/10.1109/COMPSAC51774.2021.00194>
- Yilmaz, T., & Ulusoy, O. (2022) Understanding security vulnerabilities in student code: A case study in a non-security course. *Journal of Systems and Software*, vol. 185. <https://doi.org/10.1016/j.jss.2021.111150>
- NVD Dashboard. (n.d.). *National Vulnerability Database*. <https://nvd.nist.gov/general/nvd-dashboard>
- Hladun, I. (2022). Top 25 coding errors leading to software vulnerabilities. *Waverley*, <https://waverleysoftware.com/blog/top-software-vulnerabilities/>
- Dewhurst, R. (). Static code analysis. *Static Code Analysis | OWASP Foundation*. https://owasp.org/www-community/controls/Static_Code_Analysis
- Hamilton, C. B. (2015). Security in Programming Languages. *COMP 116: Introduction to Computer Security - Final Project Archive*. <https://www.cs.tufts.edu/comp/116/archive/>

Owasp Top Ten. *OWASP Top Ten* | *OWASP Foundation*. (n.d.).

<https://owasp.org/www-project-top-ten/>

Taeb, M., & Chi, H. (2021). A Personalized Learning Framework for Software Vulnerability Detection and Education. *2021 International Symposium on Computer Science and Intelligent Controls (ISCSIC)*, 2021, pp. 119-126.

<https://doi.org/10.1109/ISCSIC54682.2021.00032>

Du, W. (2011) SEED: Hands-On Lab Exercises for Computer Security Education. *IEEE Security & Privacy*, vol. 9, no. 5, pp. 70-73. <https://doi.org/10.1109/MSP.2011.139>

Jelen, S. (2019). Securitytrails," What is CVE? - Common Vulnerabilities and Exposures,

<https://securitytrails.com/blog>

David, A., Akinpelu, O., & Adebawale, A. (2021). Africa Now Has 716,000 Software Developers in 2021, a 3.8% Rise in 1 Year. *Technext*. <https://technext.ng/2022/02/22/africa-now-has-716000-software-developers-a-3-8-rise-in-one-year/>

Morrow, S. (2022). What Is Secure Coding and Why Is It Important?. *Vpnoverview*.

<https://vpnoverview.com/internet-safety/business/what-is-secure-coding/>

Sultana, K. Z., & Williams, B. J. (2017). Evaluating micro patterns and software metrics in vulnerability prediction. *Workshop on Software Mining*, vol. 00, 2017, pp. 40–47.

<https://doi.org/10.1109/SOFTWAREMINING.2017.8100852>

Morrison, P. J., Pandita, R., Xiao, X., Chillarege, R., & Williams, L. (2018). Are vulnerabilities discovered and resolved like other defects? *In Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 498. <https://doi.org/10.1145/3180155.3182553>

- Yang, J., Ryu, D., & Baik, J. (2016). Improving vulnerability prediction accuracy with secure coding standard violation measures. *International Conference on Big Data and Smart Computing (BigComp)*, pp. 115–122. <https://doi.org/10.1109/BIGCOMP.2016.7425809>
- Van den Berghe, A., Yskout, K., & Joosen, W. (2018). Security patterns 2.0: Toward security patterns based on security building blocks. *IEEE/ACM Intl. Workshop on Security Awareness from Design to Deployment*, pp. 45–48. <https://doi.org/10.23919/SEAD.2018.8472853>
- Pham, N. H., Nguyen, T. T., Nguyen, H. A., & Nguyen, T. N. (2010). Detection of recurring software vulnerabilities. *IEEE/ACM Intl. Conf. on Automated Software Engineering*, pp. 447–456. <https://doi.org/10.1145/1858996.1859089>
- Pothamsetty, V. (2005). Where security education is lacking. *2nd annual conference on Information security curriculum development (InfoSecCD '05)*, ser. *InfoSecCD '05*, pp. 54–58. <https://doi.org/10.1145/1107622.1107635>
- Fernández, E., Washizaki, H., Yoshioka, N., Kubo, A., & Fukazawa, Y. (2008). Classifying Security Patterns, 342-347. https://doi.org/10.1007/978-3-540-78849-2_35
- Hovemeyer, D., & Pugh, W. (2004). Finding bugs is easy. *SIGPLAN*, 92–106. <https://doi.org/10.1145/1052883.1052895>
- Nadeem, M., Williams, B. J., & Allen, E. B. (2012). High False Positive Detection of Security Vulnerabilities: *A Case Study*," *ACM-SE '12*, NY, USA, pp. 359–360. <https://doi.org/10.1145/2184512.2184604>
- Reynolds, Z. P., Jayanth, A. B., Koc, U., Porter, A. A., Raje, R. R., & Hill, J. H. (2017). Identifying and Documenting False Positive Patterns Generated by Static Code Analysis Tools. *IEEE/ACM 4th Int. Workshop on Software Engineering Research and Industrial Practice (SER IP)*, pp. 55–61. <https://doi.org/10.1109/SER-IP.2017..20>

- Du, W. (2011). SEED: Hands-on lab exercises for computer security education. *IEEE Security and Privacy*, 9(5), 70-73. [6029361]. <https://doi.org/10.1109/MSP.2011.139>
- Theisen, C., Williams, L., Oliver, K., & Murphy-Hill, E. (2016). Software security education at scale. 346-355. <https://doi.org/10.1145/2889160.2889186>
- Team, G. D. W. T. (2021). How many developers are in US and in the world [updated]. *Grid Dynamics Global Team*. <https://www.griddynamics.com/services/global-team/blog/development-trends/number-software-developers-world?ref=hireremote.io>
- Kenton, W. (2022). How analysis of variance (ANOVA) works. *Investopedia*. <https://www.investopedia.com/terms/a/anova.asp>
- Vizteck. (2016). Benefits of using Sonarqube for code reviews. *Vizteck Solutions*. <https://www.vizteck.com/post/benefits-of-using-sonarqube-for-code-reviews>
- S-Quadra Advisory. (2003) ``freeradius \<= 0.9.3 RLM-SMB module stack overflow vulnerability," <https://marc.info/?l=bugtraq&m=106986437621130&w=2>
- Groebert, F. (2007). NoseRub 0.5.2 - Login SQL Injection. <https://www.exploitdb.com/exploits/4805>
- Open Web Application Security Project. (n.d.). Buffer Overflow. https://owasp.org/www-community/vulnerabilities/Buffer_Overflow
- Common Weakness Enumeration. (n.d.). CWE-665: Improper Initialization. <https://cwe.mitre.org/data/definitions/665.html>
- Common Weakness Enumeration. (n.d.). CWE-397: Declaration of Throws for Generic Exception. <https://cwe.mitre.org/data/definitions/397.html>
- Common Vulnerabilities and Exposures. (2022). CVE-2022-28463: ImageMagick 7.1.0-27 is vulnerable to Buffer Overflow. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-28463>
- Common Weakness Enumeration. (n.d.). CWE-252: Unchecked Return Value. <https://cwe.mitre.org/data/definitions/252.html>