Embedding Security Concepts in Introductory Programming Courses*

Ajay Bandi, Abdelaziz Fellah, Harish Bondalapati
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, MO 64468
{ajay,afellah,s530741}@nwmissouri.edu

Abstract

The challenges of teaching and learning introductory computer programming tend to cumulate over the years. Students should achieve a chain of ideal accomplishments including problem-solving skills, language features, structures, semantics, and coding. However, current programming practices consider security as a low-priority task and students rarely embed the predefined security requirements in their coding and software development process which may lead to insecure programs. Changing this trend in the classroom is quite challenging. As a result, a program which appears to be secure at the source code level may equate to security vulnerabilities that are often exploited by hackers. In this paper, we focus on the language-based security main features and standard security mechanisms in developing secure programs and avoiding security loopholes. This paper takes a step forward in this direction and contributes to filling the security gap.

1 Introduction

A software vulnerability is a security flaw within a software product that can be exploited by hackers and can cause cascading and recurring negative impacts in the software industry. Many software developers believe that software vulnerabilities should be addressed after the product is released, hacked, or once

errors and defects are discovered. They refer to such a policy as "release early and fix later". However, security vulnerabilities and flaws in software could happen at any point in time when the application goes live and available for the users. Several studies [15, 4] have confirmed that vulnerability disclosure may adversely affect in the long run the market performance of a software and the reputation of a firm whose software has been breached.

Developers have a misconception that security is someone else's responsibility [9]. They mostly concentrate on implementing the functionality of software systems and later releasing security updates sometimes called patches while many other users are still running old versions that might have flaws and make them prone to failure. Overall, a patch means there is one less vulnerability and access to the patch may give security advantages to an attacker. It is the responsibility of all the teams who gather requirements, design, write code, test, deploy and maintain the software [12]. Thus, the consequences of the security flaws are a spectrum of various defects that are manifested in either requirement, design, architecture, or implementation. Removing such defects and security flaws is a major challenge for software developers. One of the primary reasons that developers do not focus on security tasks is because they lack the expertise of using secure programming practices [3, 11]. Research has shown that most software [13] vulnerabilities are related to programming errors that are well understood. Such vulnerabilities could be avoided and prevented by injecting/incorporating coding standards, developing guidelines, adopting the best practices that define the programming language itself, and understanding how the language is interpreted and compiled on various runtime platforms. Moreover, research has shown [13, 16, 7] that adding security through testing is not a feasible solution and it is an incessant task for software developers. Even with automated testing tools, errors still occur in lines of code at a significant rate. Most of the current researches were focused on exploration of security by designing specific computer security courses rather than including the concepts in CS introductory programming courses.

In this paper, we present effective ways of introducing the most fundamental security-related vulnerability topics, characteristics and sources of vulnerability such as sources, severe flaws, severity and the best practices of developing secure programs in introductory computer programming courses.

Security vulnerabilities affect different types of software such as operating systems, networks, e-mail servers, software libraries, browsers, and coding, just to name a few. In the work, we focus on secure programming concepts in general and in particular on secure coding in Java. These concepts are incorporated in the introductory Java programming labs. Specifically and in terms of learning outcomes, students should be able to

- Understand common software flaws that lead to vulnerabilities.

- Identify coding mistakes and eliminate coding errors which may lead to software vulnerabilities.
- Understand how common coding mistakes can be exploited.
- Incorporate best practices and enforce a coding standard which helps programmers circumvent pitfalls and avoid vulnerabilities

From the programming perspective, the course covers the following common types of software flaws that lead to vulnerabilities: input validation errors, error handling & exceptions, memory safety violations and hashing.

2 Literature Review and Background

Some of the malicious attacks and hacking strategies are data breaches (SQL injections, XML injections), buffer overflow attacks, phishing, and sniffing, etc. Some of the security breaches negatively impacted several firms. For example, Guess.com was vulnerable to an SQL injection, allowing any user to construct a customized URL to access approximately 200,000 names and their credit card information from the website's customer database. Recently, the phishing WannaCry Ransomware attack is suspected in 150 countries by encrypting the users' hard disks impacting more than 230,000 people [3]. Heartland Payment System was attacked, and the customers' data was stolen using sniffing. The Code Red (computer worm) infected over 359,000 computers by exploiting a buffer overflow vulnerability in Microsoft Internet Information Services [17].

The primary target of the hackers and cyber-criminals is to routinely exploit the insecure code. The hacking strategies usually happen after the software is released into the market, causing a threat to the privacy of the users. When these threats are raising and discovered, organizations invest extra millions of dollars in securing their applications [8]. This also increases the cost to fix the insecure code in the existing versions as well as future releases of the software. The effects of social media and social networking sites on marketing and business have become an excellent vehicle for effective communications and have been integrated in the marketplace. Social media websites or apps allow third-party applications to collect feedback using opinion polls and track the users' activity to sponsor ads. For example, Facebook uses pixels to track the users' activity and sponsor ads. This raises a concern about users' data privacy and security. However, social media platforms are offering privacy controls, but users ignore them because of lack of knowledge and awareness. This is also applicable to the other insecure websites and apps because of lack of (adherence to) security principles. Software organizations must scrutinize the apps designed by individuals/businesses to identify any vulnerabilities or any security and policy violations. A recent example of a data firm Cambridge

Analytica which gained access to personal information of millions of Facebook users. Even though this incident is not considered to be a breach of databases but a violation of terms of use of data, privacy of the users was compromised.

3 Research Goal

Vulnerable programs are one of the leading causes of computer insecurity. Our contributions focus on developing In this research we focus on the threats to security and users' privacy in software applications that arise due to lack of secure coding practices. The best way to prevent these vulnerabilities is by improving the understanding and learning abilities of developers and users at the student level by teaching security concepts [2]. However, it is a challenge for educators to increase the number of credit hours for computer science or data science [6] degrees or to add new courses. To leverage the number of credit hours, we propose embedding secure programming concepts during the introductory programming courses. We embedded these security concepts in our lab sessions that are conducted over 2 hours weekly without slimming down or modifying the original description of the course. This helps to improve the students' ability to develop secure software from their introductory programming courses.

4 Language-Based Security Concepts

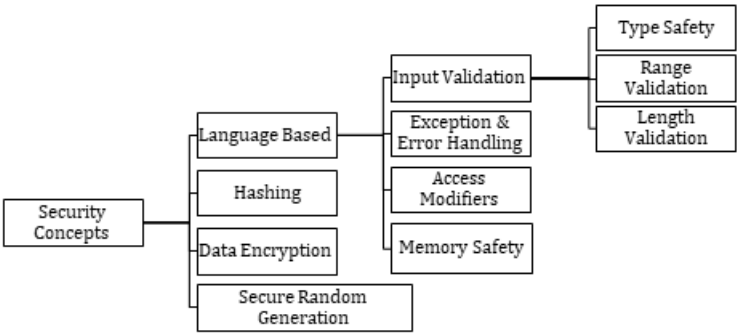

Figure 1: Classification of Security Concepts

We emphasized and embedded security concepts in the Object-Oriented Programming course (Java). We incorporated security concepts during class

time while balancing the teaching of regular material on programming concepts. Then, we embedded the implementation of security concepts in the lab sessions. The classification of security concepts are shown in Figure 1. We broadly classified the security concepts as language-based, hashing, data encryption, and secure random generation. The rationale for this classification is to cover all the basic security issues that relate to the concepts of the introductory programming course for graduate students. We further classify the language-based concepts into input validation, exception & error handling, access modifiers, and memory safety. The rationale for the classification of language-based security concepts is because students do not require any additional knowledge of these concepts as they are related to the general programming concepts. Other than the language-based security concepts, students can make use of existing Java APIs to implement in the labs. We did not concentrate those concepts in the regular class time. This gave students the similar experience of learning and using regular Java APIs. The rest of this section presents a brief description of these concepts with a sample source code that we used in labs.

4.1 Input Validation Vulnerability

This class of vulnerability is called input validation vulnerability and it occurs when a program does not check or validate user inputs. The unsafe domain of inputs may trigger a vulnerability that are commonly exploited by hackers and can arise in any programming language. The lack of validation in the input fields such as text boxes, radio buttons, checkboxes, and dropdowns, to name few are the primary reason of vulnerabilities that may lead to potential injection attacks such as SQL and XML injections. In addition to injection attacks, the invalid inputs could sometimes cause the defects in the functionality of the software [14, 5]. Typically, the class of input validation vulnerabilities includes buffer overflows, string format, and integer overflows. From the programming language perspective, the choice of the type, strong or weak, and the type checking, static or dynamic, can contribute to security problems and may affect the robustness of the software. We classified input validations into three different categories.

Strongly Typed: Strongly typed languages such as Java requires developers to infer the data type during the compilation time, i.e., developers must validate user inputs by checking the type of the data. For example:

Listing 1: Example for Strongly Typed

```
Scanner sc = new Scanner(new File(``file.txt''));
if(sc.hasNextInt()){
    //Read integer data
    int data = sc.nextInt();
```

```
}else{
    //Handle mismatch
}
```

Range Validation: Input data such as age, dates, quantity, and price need to be validated to prevent miscalculations. Range validation is not only for numeric data values but also for characters and string data types. For example,

Listing 2: Example for Range Validation

```
Scanner sc = new Scanner(System.in);
int semester;
do{
    semester = sc.nextInt();
}while(semester>=1 && semester<=4);
```

Length Validation: User inputs such as phone numbers, social security numbers, credit/debit card details are of fixed length. Validating these values are important in any software to avoid human errors. For example,

Listing 3: Example for Length Validation

```
Scanner sc = new Scanner(System.in);
long mobile = sc.nextLong();
while(new String(mobile).length() != 10){
    mobile = sc.nextLong();
}
```

4.2 Exception & Error Handling

When an error occurs such as invalid user inputs, software act in an unexpected manner in response to such an exceptional event and the code may raise an exception. Appropriate exceptions and error handling techniques are an important software component of software security. In programming languages such as C++, C# or Java, error handling is considered a form of exception handling. Let us consider an application when exceptions and errors are not handled; the complete error stack is displayed to the users. This error stack may contain all the details like which database is used or threads that reveals the implementation details. Attackers can take advantage of these error messages to create vulnerabilities [16]. For example,

Listing 4: Example for Exception & Error Handling

```
Scanner sc = new Scanner(System.in);
int semester;
try {
    semester = sc.nextInt();
} catch(InputMismatchException ex){
```

```
//User Message: Invalid semester, Enter valid semester#
    semester = sc.nextInt();
}
```

4.3 Access Modifiers

Access modifiers of classes, methods, and attributes in Java and other programming languages provide a mechanism to restrict access but may have a negative impact on the security of an application since they do not guarantee information flow control of a code. The absence of a well-defined access modifier in a Java code may break the encapsulation of that code which may lead to unexpected program behaviors. The goal of security is to prevent unauthorized access to any data. If an inappropriate access specifier is used, the attackers can change the state of the objects by accessing its variables or by calling methods. In the Java programming language, there are four access modifiers (private, protected, public, package). If global access is not required, usage of public access modifier to the state variables of a class should be avoided as it allows any other object to change its state. Alternatively, it is better to use public setter methods to mutate the state of instance variables by performing any required validations.

4.4 Buffer/Integer Overflow

Java language is considered to be memory safe regarding array bounds and pointer dereferences. However, it is required to explicitly handle integer overflows while calculating and assigning values that involve large numerals. A buffer overflow occurs when the user is trying to store data into the buffer than its limit. It is one of the most common vulnerabilities. In every programming language, a datatype is pre-defined to hold certain bytes (for instance, integer type in Java, can hold 4 bytes with any value in the range of -2,147,483,648 to 2,147,483,647. When assigning a value to a variable, Java would throw a syntax error during the compile time if the value is out of the range [7, 17].

4.5 Hashing

Confidential data like passwords and digital signatures are used to verify the authenticity of an account or a document. If these details are stored as clear text in the database, a hacker could hack the passwords during the data breach. Instead, use a hash function to generate a hash code and store in the database. Later, this hash code is compared with the new hash code generated from the data provided by the user. Java API provides MD5, SHA-1, SHA-256

algorithms to create the hash values. An example source code snippet for SHA-1 is shown below.

Listing 5: Example for Hashing

```
MessageDigest md = MessageDigest.getInstance("SHA");
byte[] digest = md.digest(password.getBytes());
String encryptedPassword = new String(digest);
```

4.6 Data Encryption

Encryption can secure the transmission of data through a network. Java Cryptography Architecture (JCA) [11] allows implementing several prominent symmetric and asymmetric algorithms for verification of messages and encryption/decryption of data. Java API includes encryption algorithms like AES, Blowfish, DES, RSA, DESede, etc. Sample source code for data encryption using AES algorithm is provided below.

Listing 6: Example for Encryption

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
SecretKey aesKey = keygen.generateKey();
Cipher aesCipher;
// Create the cipher
aesCipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
// Initialize the cipher for encryption
aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);
// Our cleartextbyte[] cleartext =
    "This is just an example".getBytes();
// Encrypt the cleartext
byte[] ciphertext = aesCipher.doFinal(cleartext);
```

4.7 Secure Random Number Generation

Pseudo random number algorithms are used to generate a sequence of random values. While the sequence generated by these algorithms may appear to be random, at their core they are not. Having an insecure or deterministic random generator cause vulnerabilities in applications. Java API provides a SecureRandom class to generate a secure pseudo random number generator [11]. A sample code snippet is shown below.

Listing 7: Example for Secure Random

```
SecureRandom random = new SecureRandom();
 byte bytes[] = new byte[20];
 random.nextBytes(bytes);
```

5 Case Studies

The methodology of the case studies is adopted from [10]. In this research, we proposed emphasizing security concepts to students and embedding security-related scenarios in lab assignments. We have conducted our case studies in a graduate course of a total of 90 students, where students are taking Object-Oriented Programming course (Java emphasis). Approximately 30% of the graduate students who enrolled in this course are computer science undergraduate majors and the remaining 70% are non-computer science undergraduate majors. The steps involved were as listed below:

1. Students are pretested for their current level of understanding security concepts using survey [1].
2. Emphasize on security concepts to the students was done while teaching the course while balancing the regular programming concepts.
3. Students are given a total of 12 lab assignments embedding scenarios related to security concepts along with regular programming language concepts. Different kinds of input validation scenarios are included in the Control Structures, Interfaces & Abstract Classes, and Sorting & Equals labs. Exception & Error handling scenarios are included in the Casting & Exceptions and Sorting & Equals labs. Hashing and Random Generators are included in the Interfaces & Abstract Classes. Buffer Overflow scenario are included with Recursion tracing problems.
4. Students are post-tested for their level of understanding security concepts using survey [1].

6 Results and Analysis

After collecting the data from our case studies, we presented our results in a comprehensive tabular format shown in Tables I and II.

Level of Understanding	1 (Never heard of)	2 (Exposure)	3 (Familiarity)	4 (Knowledgeable)	5 (Mastering)
Pretest	23%	36%	22%	16%	4%
Post-test	0%	1%	7%	42%	51%
Difference	-23%	-34%	-15%	26%	47%

Table I: Students' Level of Understanding of Security Concepts

Table I shows the results for the overall level of understanding of all the security concepts. The pretest results show that around 59 students out of 73 were below knowledgeable. Later, we emphasized the security concepts in the class, embedded those concepts in labs and asked students to solve the

lab exercises. After the post-test, all the students have substantially strengthened their technical knowledge of security concepts mentioned in this paper. Both students pretest and post-test results were proved by the instructor. The "Mastering" level did improve by 47%. These students in level 5 moved from other levels. The number of students who are knowledgeable about the security concepts is also increased by 26% and these students moved from lower levels. The increased percentage in green color shows the substantial positive increase of understanding level. The red color graphs are considered as positive decrease highlighting the number of students moved from the lower three categories (never heard of, exposure, familiarity) to the knowledgeable and mastering categories. Overall there is no negative impact of introducing security concepts in the introductory programming class for graduate students.

Table II shows the comprehensive pretest and post-test results of individual security concepts. The "Mastering" level of students did improve in all the individual security concepts. From the results, most of the students' understanding level increase in the input validation, category.

Level of Understanding		1 (Never heard of)	2 (Exposure)	3 (Familiarity)	4 (Knowledgeable)	5 (Mastering)
Input Validation	Pretest	1%	21%	30%	29%	19%
	Post-test	0%	1%	6%	32%	62%
	Difference	-1%	-19%	-25%	3%	42%
Exception & Error	Pretest	10%	16%	33%	34%	7%
	Post-test	0%	1%	8%	48%	43%
	Difference	-10%	-15%	-25%	14%	36%
Access Modifiers	Pretest	7%	14%	29%	37%	14%
	Post-test	1%	0%	6%	38%	55%
	Difference	-6%	-14%	-23%	1%	41%
Hashing/ Data Encryption	Pretest	14%	21%	33%	23%	10%
	Post-test	0%	1%	8%	44%	47%
	Difference	-14%	-19%	-25%	21%	37%
Random Generators	Pretest	8%	29%	25%	22%	16%
	Post-test	0%	1%	7%	48%	44%
	Difference	-8%	-27%	-18%	26%	27%
Buffer Overflow	Pretest	12%	22%	33%	22%	11%
	Post-test	1%	4%	18%	38%	38%
	Difference	-11%	-18%	-15%	17%	27%

Table II: Students' Level of Understanding of Individual Concepts

In all the security concepts, there is a definite increase in the understanding levels of students in mastering and knowledgeable categories. The red color graph in each category shows that the positive decrease highlighting the students in those levels move students to upper levels. In our study, none of the

students move from higher level to lower levels. Therefore, there is no declining performance of any student.

7 Conclusion

The proposed approach of embedding security concepts in introductory programming courses and the practical implementation of those concepts in labs for graduate students has provided evidence in strengthening the students understanding and learning considering the security concepts without compromising the programming skills. The empirical evidence shows that students' understanding and learning improved from lower levels to higher levels. Students are also comfortable in handling security issues such as input validations, hashing, exceptions, usage of access modifiers, and generating secure random numbers. Our study also highlights providing insights on how to teach security concepts without increasing the number of credit hours in the computer science curriculum. As future research, we propose to identify any shortcomings and pitfalls in this approach by extending this study to larger computer science undergraduate majors for a more extended period.

References

[1] Pre and post survey. `https://goo.gl/forms/rBcJuHWDSt21hv012`.

[2] Java cryptography architecture, 1993,2018. `https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html`.

[3] *OWASP Secure Coding Practices Quick Reference Guide*. 2010.

[4] Ashish Arora, Ramayya Krishnan, Rahul Telang, and Yubao Yang. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, 21(1):115–132, 2010.

[5] C. Warren Axelrod. *Engineering Safe and Secure Software Systems*. Artech House, 2012.

[6] Ajay Bandi and Abdelaziz Fellah. Crafting a data visualization course for the tech industry. *Journal of Computing Science in Colleges*, 33(2):46–56, 2017.

[7] Jon Brodkin. The top 10 reasons web sites get hacked. *Network World*, 24(39):1–20, 2007.

[8] Brian Chess and Jacob West. *Secure Programming with Static Analysis*. Pearson Education, 2007.

[9] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. pages 119–129. DARPA Information Survivability, 2000.

[10] Abdelaziz Fellah and Ajay Bandi. The essence of recursion: Reduction, delegation, and visualization. *Journal of Computing Science in Colleges*, 33(5):115–123, 2018.

[11] Katerina Goseva-Popstojanova and Andrei Perhinschi. On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*, 68(C):18–33, 2015.

[12] Samanvay Gupta. Buffer overflow attack. *IOSR Journal of Computer Engineering*, 1(1), 2012.

[13] Elisa Heymann, Barton P Miller, and Jim Kupsch. 10 common programming mistakes that make you vulnerable to attack. *Condor Week*, 2012.

[14] Clifton Phua. Protecting organisations from personal data breaches. *Computer Fraud & Security*, 2009(1):13–18, 2009.

[15] Rahul Telang and Sunil Wattal. An empirical analysis of the impact of software vulnerability announcements on firm stock price. *IEEE Transactions on Software Engineering*, 33(8):544–557, 2017.

[16] Jason Earl Thomas. Individual cyber security: Empowering employees to resist spear phishing to prevent identity theft and ransomware attacks. *International Journal of Business and Management*, 13(6), 2018.

[17] Kenneth A Williams, Xiaohong Yuan, Huiming Yu, and Kelvin Bryant. Teaching secure coding for beginning programmers. *Journal of Computing Sciences in Colleges*, 29(4):91–99, 2014.