Journal of Computer and Communications, 2014, 2, 25-36
Published Online December 2014 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2014.214003

Scientific Research

A Case Study of Adopting Security Guidelines in Undergraduate Software Engineering Education

Yen-Hung Hu[1], Charles Scott[2]

[1]Department of Computer Science, Norfolk State University, Norfolk, Virginia, USA
[2]Department of Computer Science, Hampton University, Hampton, Virginia, USA
Email: yhu@nsu.edu, chazzscott15@gmail.com

Received 20 September 2014; revised 22 October 2014; accepted 18 November 2014

Abstract

Security plays a large role in software development; simply without its existence the software would be vulnerable to many different types of attacks. Software security prevents leaks of data, alternation of data, and unauthorized access to data. Building a secure software involves a number of different processes but security awareness and implementation are the most important ones among them. To produce high quality software security engineers need to meet today's cybersecurity demands, security awareness and implementation must be integrated in undergraduate computer science programming courses. In this paper, we demonstrate the importance of adopting security guidelines in undergraduate software engineering education. Thus, this paper focuses on integrating secure guidelines into existing applications to eliminate common security vulnerabilities. An assessment table, derived from several existing Java security guidelines, is developed to provide in depth critiques of the selected capstone project. Potential security vulnerabilities in the capstone project are identified and presented in a form showing the degree of threats against the three security characteristics: confidentiality, integrity, and availability addressed in the McCumber Cube model. Meanwhile, vulnerability density of the capstone project is calculated to demonstrate the performance of this research.

Keywords

Software Security, Security Guidelines, McCumber Cube Model, Vulnerability Density

1. Introduction

Security is a term that you simply cannot get rid of. As computers become more and more prevalent in our daily

How to cite this paper: Hu, Y.-H. and Scott, C. (2014) A Case Study of Adopting Security Guidelines in Undergraduate Software Engineering Education. *Journal of Computer and Communications*, **2**, 25-36.
http://dx.doi.org/10.4236/jcc.2014.214003

lives it becomes even more important to safeguard these technologies. This is just as important as in the topic of software engineering. A vast majority of attacks make use of software vulnerabilities that are entirely preventable. It is stated by a pair of two leading experts, John Viega and Gary McGraw, "*behind every computer security program and malicious attack lies a common enemy—bad software*" [1]. Within the topic of software engineering, security is defined as the effort to create software in a secure computing platform. Software security prevents the following: leaks of confidential data, alternation of data without the knowledge of the system, and unauthorized access to the system [2].

Software must be designed as well as implemented so that the secured users can perform actions that are needed and have been allowed. Building secure software involves many requirements and polices which thus produce a set of laws, rules, and practices that users have to abide by. These policies would address the following: data security, information security, and content security [2]. Data security refers to protective digital privacy measures that are applied to prevent unauthorized access to computer, databases and websites [3]. Some examples of data security technologies include software/hardware disk encryption, backups, data masking and data erasure [3]. Information security is designed to protect the confidentiality, integrity and availability of computer system data from those with malicious intentions [4]. Content security or sometimes referred to as digital rights management is the use of software or other computer technology to manage the conditions under which copyrighted material in digital form can be accessed [5].

Almost all computers are connected to a local network or the Internet so every piece of the software deployed in a system is subject to potential adversaries. Several approaches in literatures have aimed atreducing potential software breaches [6]-[8] and the ones that integrate security aspects into Software Development Life Cycle (SDLC) are adopted most [6] [9]-[11]. However, they may not be good resources for educating computer science students because complicated knowledge and procedures are involved in every phase of the SDLC.

This paper focuses on using existing products and selected security guidelines to explain and verify how to implement a secure application. We have selected 13 critical questions from several Java security guidelines to reflect most common security errors in students' codes. We then use these critical questions to assess every component in the selected capstone project. Each component will be scored using McCumber's Cube model which evaluates information security in terms of data confidentiality, integrity, and availability [12]. Possible solutions for each vulnerability are also suggested.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 brings up our motivation. Section 4 defines software security metrics adopted by this paper. Section 5 explains the approach of adopting security guidelines to identify software vulnerabilities. Section 6 presents background of the selected capstone project. Section 7 assesses every component in the capstone project and discusses our observations and suggestions. Section 8 lists our observations and suggestions. Section 9 concludes this paper and points out future work.

2. Related Work

Several literatures have studied and addressed strategies with regards to how to implement security within the software development process. For example Computer Emergency Response Team (CERT) began new research in 2009 in software security measures that build on CERT'S core competence in software and information security. The overall purpose of this research was to address the following questions which include: "*how do I establish and specify the required/desired level of security for a specific software application, set of application software-reliant system, system of systems, supply chains, and other multi-systems environments*" [13]. Also, "*how do I measure, at each phase of the development or acquisition life cycle that the required and/or desired level of security has been achieved*?" [13].

In addition, Jorge A. *et al.* have also researched this topic [14]. They both believe that most of the software developed in the world use classic methods which include: the classic Waterfall Model, Spiral Model, Capability Maturity Model Integration (CMMI), Team Software Process (TSP) and Personal Software Process (PSP) [6] [14]. All of these classic methods with regards to software development are mainly dedicated to the quality and consistency of the software development process and not security. They believe it is necessary to modify these classic evolutionary deployment models by adding computer science security. Their paper propose to add security through all the phases of the process. They have concluded that everything within the process of software development must be reviewed thoroughly [14].

The McCumber Cube methodology offers a structured approach to assessing while also managing security risk in IT systems [12] [15] [16]. This model relies on the implementer to identify information assets and then think of risk management in a deconstructed view across 3 different characteristics which include as follows: confidentiality, integrity and availability of critical information [12] [16]. The strength in this model is in the multidimensional view required to implement robust Information Assurance (IA) programs. Well-executed systems will include this IA model during all phases of the SDLC.

3. Motivation

We have observed several cases showing the consequences of software vulnerabilities [17] [18]. For instance, a report from Capers Jones reveals that there are a number of vulnerabilities in most commercial software with10,000 function point size range and it will cost almost 50 cents out of every dollar to find and fix bugs [17].

Another report, released by B2B International and Kaspersky Lab in 2013, reveals that the average financial loss of IT security breaches suffered by large companies from a single serious incident was $649,000. For small and medium size companies, the average financial loss was $50,000. Meanwhile, they also indicated that vulnerabilities in the software ranked highest among all internal threats faced by companies during 2011-2013 and about 10% of these threats caused leaks of important business data [18].

These instances bring up our motivation of studying and engaging software security and quality knowledge into software engineering education. We make the assumption that with strong security knowledge gained during their undergraduate programming courses, our future software engineers will be able to eliminate most vulnerabilities within applications in the earlier stages of their SDLC process. As a result, many security breaches happened in today's software could be eliminated and financial loss associated with them could be saved and re-invested.

4. Software Security Metrics

Several literatures have proposed the metrics of measuring software quality or security [19] [20]. In this paper, we assume that a potential vulnerability exists if a defect in the code leads to the violation of data confidentiality, integrity, and availability. Where, defect is defined as any issue that causes incomplete executions of the software. Thus, this assumption could be rewritten as:

A defect $z \in \{$*vulnerability*$\}$ *if z leads to the violation of data confidentiality, integrity, and availability.*

We also define the vulnerability density (V_D) as the number of vulnerabilities in one line of code (LOC). Therefore, we have

$$V_D = \frac{V}{S}$$

where, V is the number of vulnerability and S is the size of the software in the unit of one LOC. Vulnerability density could be used to measure the programming in terms of how secure the code is. If other conditions are the same, a program with a higher vulnerability density is likely to be compromised more often.

5. Adopting Security Guidelines to Identify Software Vulnerabilities

To identify potential vulnerabilities in the application, this paper adopts security guidelines from several resources which emphasize on Java security design and implementation. As shown in **Table 1**, we create a 3-column table to address the criteria for identifying software vulnerabilities. A list of 13 critical questions derived from several existing Java security guidelines [2] [7] [8] [21] [22] is presented in the first column. These 13 questions are used to assess every component in the application to see whether there exist vulnerabilities in such a component. Explanations of every critical question are presented in the second column. Last, potential vulnerabilities of every component against the three security characteristics (*i.e.* confidentiality, integrity, and availability) addressed in the McCumber Cube model are listed in the third column. We examine most components including variables, methods, and classes in the application thoroughly to see whether any of them violates these security characteristics and to count the number of vulnerabilities as well as the vulnerability density of the application.

Table 1. Critical questions, explanations, and potential software vulnerabilities.

Critical Question	Explanation	Potential Vulnerabilities against Security Characteristics in the McCumber Cube Model: Confidentiality, Integrity, Availability
Q.1. Limit the accessibility of classes, interfaces, methods, and fields	Use an access modifier to limit their accessibility. The four access levels are: • Default: visible to the package. No modifiers are needed. • Private: visible to the class only • Public: visible to the world • Protected: visible to the package and all subclasses.	If wrongly declared, data confidentiality, integrity, and availability may be violated.
Q.2. Use a try-with-resources statement to safely handle closeable resources.	The try-with-resources statement ensures that each resource is closed at the end of the statement.	When resources are not closed properly, data confidentiality, integrity, and availability may be violated.
Q.3. Avoid using try-catch-finally block.	Ordinary try-catch-finally block can raise some issues: such as failing to close a resource because an exception is thrown as a result of closing another resource, or masking an important exception when a resource is closed.	When resources are not closed properly, data confidentiality, integrity, and availability may be violated.
Q.4. Use the same type for the second and third operands in conditional expressions.	Use different types in a conditional expression may cause unintended type conversions.	When types are not the same, data integrity may be violated.
Q.5. Avoid using static field variables.	Static variables are class variables, not instance variables. Science any other class in the same scope can access the static variables, it is very difficult to secure them.	Since static variables can be modified by other classes in the same scope, data integrity may be violated.
Q.6. If possible make public static fields final	Otherwise, attacker may change the value.	If value changed, data integrity may be violated.
Q.7. If possible, use immutable objects.	Contents of the mutable object can be changed.	If contents changed, data integrity may be violated.
Q.8. Avoid storing user-given mutable objects directly.	Contents of the mutable objects can be changed. Clone the objects before processing them internally.	If contents changed, data integrity may be violated.
Q.9. Avoid using inner classes.	After compilation, any class in the package can access the inner class. Meanwhile, private filed of the inner class will be converted into non-private to permit access by the inner class.	If other classes in the package can access the inner class, data confidentiality and integrity may be violated. If private filed of the inner class changed, data integrity may be violated.
Q.10. Avoid using the clone() method to copy untrusted method parameters	Inappropriate use of the clone() method can allow an attacker to exploit vulnerabilities by providing arguments that appear normal but subsequently return unexpected values. Such objects may consequently bypass validation and security checks [22].	Data confidentiality and integrity may be violated.
Q.11. Make secure classes un-cloneable.	Otherwise, malicious developers can instantiate a class without running its constructors [2].	This may violate data confidentiality and integrity.
Q.12. Avoid embedding sensitive information	Malicious developers can obtain such a sensitive information	This may violate data confidentiality.
Q. 13. Prevent constructors from calling methods that can be overridden	Constructors that call override methods give attackers a reference to the object being constructed before the object has even been fully initialized	This may violate data integrity

6. Capstone Project Background

Since focusing on software engineering education, we choose a computer science capstone project as the sample to examine common security vulnerabilities appearing in students' projects. The application is selected from last year's senior computer science capstone course. The goal of this project was to create an application for computerized stock trading. The student team developed their own algorithms to implement their trading strategy with how they buy or sell stocks. The algorithms are based on the trading strategies including high-frequency trading,

day-trading, and investment. High-frequency trading strategy is where stocks are only held for a fraction of second or a few seconds. Day trading is a strategy where stocks may be purchased and sold several times in the same day. Finally investment strategy; where the holding period may be a few days, weeks, or even a holding period for a long-term capital gain.

The capstone project consists of 5 Java classes: M3_Main, Wallet, Stock, Transactions, and Trade_Decision. M3_Main class is the main class that interacts with other classes in the product. Wallet class handles all transactions that happen within the wallet whenever the application buys or sells stock based on their own algorithm. Stock class handles all information regarding the stocks depending on the application buying or selling based on the algorithm. Transactions class is used for transactions made between the automated buying and selling of stocks while also allocating enough money within the wallet when the application buys a stock. Trade_Decision class is used for the trade decision between how stocks are bought and sold within the application. The project has totally about 2036 lines of code.

7. Vulnerability Assessment for the Capstone Project

Below these 3-columntables (**Tables 2-11**) provide in depth critiques of this capstone project which includes its classes, variables, and methods. In these tables, column 1 displays the details of variables or methods in each class; column 2 explains the meanings of variables or methods in the 1st column; and column 3 indicates the critical questions that compromise variables or methods in the 1st column.

Table 2. Vulnerability assessment for variables in the M3_Main class.

Variable	Description	Critical Questions Associated With This Variable
public static String accountPassword = “hello”;	Setting static string variable to the word string “hello”	Q.1, Q.5, Q.6, Q.12
public static String secQuestion1Answer = “”;	Public static String variable setting itself to empty quotes	Q.1, Q.5, Q.6
public static String secQuestion2Answer = “”;	Public static String variable setting itself to empty quotes	Q.1, Q.5, Q.6
public static boolean accountInitialized = false;	Public static Boolean variable names “accountInitialized” which is then set to false	Q.1, Q.5, Q.6
public static boolean policyChecked = false;	Public static Boolean variable names “policyChecked” which is then set to false	Q.1, Q.5, Q.6
public static Wallet myWallet = new Wallet(1000000);	Sets static object Wallet for the wallet within the application. Set value of 1 million.	Q.1, Q.5
Timer timer = new timer();	Creating object called timer	
Calendar now;	Using calendar class for market times of operation	
int hour;	integer variable named “hour” used for time for market hours of operation	Q.1
int minute;	Integer variable named “minute” used for time for market hours of operation	Q.1
int day;	Integer variable named “day” used for time for market hours of operation	Q.1
Stock stock;	A stock object	
public static boolean marketIsOpen = false;	Boolean variable that sets “MarketIsOpen” to false	Q.1, Q.5, Q.6
public boolean stockIsOwned = false;	Boolean variable that sets “stockisOwned” to false	Q.1, Q.5
public int x = 0;	Public integer variable “x” set to 0	Q.1
public static int rowCount = 0;	Public static integer variable “rowCount” to 0	Q.1, Q.5, Q.6

Table 3. Vulnerability assessment for methods in the M3_Main class.

Method	Description	Critical Questions Associated with This Method
public void start(Stage stage) throws Exception	Method that requests the PolicyPrompt FMXL which is shown to the user before the application is started. SetTitle object for the title application which is presented at the beginning of the application.	Q.1, Q.2, Q.3
public static void makeSectorTable()	Public method to make sector table within database. Pulls this information from a webpage named YahooFinanaceScript. This information is pulled and updates in 15 minute intervals.	Q.1, Q.2, Q.3
public static void startPhp()	Public static method that starts the PHP which is a scripting language used for website development. The connection is opened and then the stock information is pulled line by line and then printed out.	Q.1, Q.2, Q.3
public static void restartDatabase()	This method restarts the database which is pulled from the Yahoo Finance Script which now has been named Finesse.php.	Q.1, Q.2, Q.3

class startTask extends TimerTask
{
@Override
public void run() {
…….
}
}
	This startTask class extends the TimerTask of the program and it called every certain amount of minutes to basically check if the market is open or not and then runs on another thread. A run method determines if the market is open or close based on the times of 9:30 am - 4:00 pm M-F.	Q.9

public static Stock connectToDatabase (String query, int t)	This public static method purpose is to connect to the database within the application.vMethod pulls necessary information: Symbol, Name, Percent_Change, Price, Price2, Price3, Days_High, and the Days_Low.	Q.1, Q.2, Q.3
public void strategy()	This is the strategy method that runs the trading strategy of the application. The stock is updated from the sandp500 and if the stock is in portfolio (the stock that we own) and if the stock is in the watchlist (the stocks we are watching, not owned) each strategy is performed.	Q.1, Q.2
public static void main(String[] args)	The main method within this application because this application is deployed by JavaFX. The main serves only as fallback in case the application cannot be launches through deployment artifacts.	Q.1
public void updateAccount()	This method simply updates the account. The method updates watch list tab, portfolio tab, and also update past tab method	Q.1

Table 4. Vulnerability assessment for variables in the wallet class.

Variable	Description	Critical Questions Associated with This Variable
double cash	Double variable for cash within application	Q.1
double startValue	Double variable for the started value within the wallet of application	Q.1
double annualReturn = 0	Double variable for annual return value which is initialized to 0	Q.1
double AccountValue = 0	Double variable for the account value which is initialized to 0	Q.1
double marketValue = 0	Double variable for market value which is initialized to 0	Q.1
double spMoney	Double variable for amount of money allocated to spend per day	Q.1
double avMoney	Double variable for amount of money available to spend per stock	Q.1

8. Observations and Suggestions

In this research, we observe that there are several common security vulnerabilities appearing in the capstone project. We have analyzed them and addressed our suggestions below:

- Accessibility of class, method, and variable should be addressed properly to protect data confidentiality, integrity, and availability.
- Resources must be closed properly after released to protect data confidentiality, integrity and availability.
- Resources must be monitored with special syntax and identifier to protect data confidentiality, integrity and availability.
- Inner classes should be avoided to protect data confidentiality, integrity and availability.

Table 5. Vulnerability assessment for methods in the wallet class.

Method	Description	Critical Questions Associated with This Method
public Wallet(double start)	This method starts the value of the wallet while also setting the start value to the double value "Cash". The amount of money allocated to spend per day (spMoney) and the amount of money avaible to spend per stack (avMoney) is run through a series of calculations.	Q.1
public void setStartDayValue()	Public method that sets the start day value of the wallet.	Q.1
public double getStartDayValue()	Public double method to get the start day value of the wallet.	Q.1
public double getCurrentValue()	Public double method to get current value of the wallet. The current value is then returned.	Q.1
public String getDailyReturn()	This String method returns the daily return within the wallet which is how much money you have successfully made in the same day. The daily return is current value subtracted by the start day value.	Q.1
public double getCash()	Public double method to return the amount of cash within the wallet.	Q.1
public void setCash(double c)	Public void method which sets the amounts of cash.	Q.1
public double getStartValue()	Public double method that returns the start value within the wallet	Q.1
public double getAnnualReturn()	Public double method that will return the annual value which is calculated by account value divided by start value raised to the (365/1), thus the number of days in a year divided by the number of days since account started.	Q.1
public double getAccountValue()	Public double method that returns the account value which is calculated by adding the cash to the market value of the stock.	Q.1
public void setMarketValue(double mv)	Public void method that sets the market value. This market value is expecting a double.	Q.1
public double getMarketValue()	Public double method that returns the market value.	Q.1
public double getSP()	Public double method to return spending money per day	Q.1
public void setSP(double s)	Public void method to set spending money per day.	Q.1
public double getAV()	Public double method that returns the amount of money to spend per transaction when buying	Q.1
public void setAV(double d)	Public void method to set the amount of money to spend per buy transaction,	Q.1

- Static variables should be avoided to protect data integrity.
- Sensitive information should not be included in any part of codes and should be encrypted to protect data confidentiality.

At the end, 123 potential vulnerabilities (51 variables and 72 methods) are identified and classified in the capstone project with total of about 2036 lines of code. Therefore, the vulnerability density of this project can be estimated as:

$$V_D = \frac{123}{2036} \approx 6\%$$

This 6% vulnerability density is relatively high while comparing with most commercial applications [19]. However, we have to take into account that this represents students from last year's computer science capstone course, who lack in software security awareness and quality knowledge. After the software security and quality knowledge have been introduced to a new group of students in this year's senior computer science capstone course. Students in the new group are capable of identifying and rewriting these 51 variables and 72 methods. The revised project has eliminated most vulnerabilities listed on the **Table 12** and has the vulnerability density close to 0%. That is $V_D \approx 0\%$.

Table 6. Vulnerability assessment for variables in the stock class.

Variable	Description	Critical Questions Associated with This Variable
SimpleStringProperty rec;	Simple String rec variable that gets the buy time stamp of the stock	Q.1
SimpleStringProperty rec2;	Simple String rec2 variable that gets the sold time stamp of the stock	Q.1
SimpleStringProperty type;	Simple String type variable that is used for the type of stock	Q.1
SimpleStringProperty name;	Simple String name variable that is used for the name of stock	Q.1
SimpleStringProperty symbol;	Simple String symbol variable that is used for the symbol of the stock	Q.1
SimpleStringProperty sector;	Simple String sector variable that is used for sector of the stock	Q.1
SimpleDoubleProperty percentChange;	Simple Double percentChange variable used to find percent change for each stock	Q.1
SimpleDoubleProperty curPrice;	Simple Double curPrice variable used for the current price of each stock	Q.1
SimpleDoubleProperty prevPrice1;	Simple Double prevPrice1 variable used for getting the price of the stock each time the application refreshes	Q.1
SimpleDoubleProperty averagePrice;	Simple Double averagePrice variableused for the average price of the stock throughout the day	Q.1
SimpleIntegerProperty count;	Simple Double count variable used for counting of all the stock price updates	Q.1
SimpleDoubleProperty sumPrice;	Simple Double sumPrice used to get the average which is sumPrice/count = averagePrice	Q.1
SimpleDoubleProperty prevPrice2;	Simple Double prevPrice2 variable used for getting another price of stock after refreshing	Q.1
SimpleDoubleProperty purchasePrice;	Simple Double purchasePrice variable used for the purchase price of the stock	Q.1
SimpleDoubleProperty accVal;	Simple Double accVal variable used for the account value	Q.1
SimpleIntegerProperty quantity;	Simple Integer quantity variable used for the quantity of the stocks	Q.1
SimpleIntegerProperty quantity2;	Simple Integer quanity2 variable used for the quantity of the stocks	Q.1
SimpleBooleanProperty owned;	Simple Boolean owned variable used for the owned stocks that the application has bought	Q.1
SimpleDoubleProperty totalValue;	Simple Double totalValue variable used for total value of the stock	Q.1
SimpleDoubleProperty net, daysHigh, daysLow	Simple Double net, daysHigh, and daysLow variables used for the net of the stock, also used to get the day's high and day's low of the stocks	Q.1
SimpleDoubleProperty purchase Value, sellPrice, profit, total Cost	Simple Double purchaseValue, sellPrice, profit, and totalCost. Variables used for the purchased price of the stock, the sell price of the stock, the profit of the stock, and the total cost of each stock	Q.1

9. Conclusions

After the study conducted on this capstone project we can now see and understand the underlying vulnerabilities that lie within this Java code. It is evident that the variables and methods within each of these classes are feasible to exploit within the application. It is also evident that the developers of the capstone project, while implementing, they were not aware of the threats they posed on themselves while developing. The very focus of this paper is to conduct a vulnerability assessment of this Java code to reveal its vulnerabilities. As you can see by using the guidelines above it is apparent that this application can pose many threats which include the following: integrity, confidentiality and even the availability of the entire application.

For example the use of public variables being created can pose threats of the following just listed. This was exemplified throughout the application time and time again. Another constant vulnerability that was found throughout the Java code included using static field variables. Since static variables can be modified by other

Table 7. Vulnerability assessment for methods in the stock class.

Method	Description	Critical Questions Associated with This Method
public Stock (String sym, String n, double perCha, double cPrice, double pPrice1, double pPrice2, double dH, double dL)	Constructor for normal stocks. This method takes in the symbol, the percent change, current price, price1, price2, day's low and also day's high.	Q.1
public Stock (String sym, String n, double perCha, double cPrice, int q, double pPrice, String time)	Constructor for Portfolio transactions. Takes in information about stocks which includes: the symbol, percent change, current price, quantity, price, and the time when it was bought.	Q.1
public Stock (String s, String t1, int q, double buy, double tc, String t2, int noSell, double sell, double prof, double n)	Constructor for History of all stocks when transactions are made. Method takes in symbol, quantity, net, purchase price, totalCost, quantity, sold price, and profit of stock.	Q.1
public SimpleBooleanProperty watchlist Property()	Method within Watch-list to return the stocks owned	Q.1
public double getDaysHigh()	Double method to return the day's high of the stock	Q.1
public double getDaysLow()	Double method to return the day's low of the stock	Q.1
public String getRec()	String method to return the time of which the stock was bought.	Q.1
public void setRec(String d)	Void method to set the sell time of which a stock was sold at.	Q.1
public void setRec2(String d)	Void method (setRec2) to set the sell time of which a stock was sold at.	Q.1
public String getRec2()	String method (getRec2) to return the time of which the stock was bought at.	Q.1
public String getType()	String method to return trade type of stock	Q.1
public void setTotalValue()	Void method that sets the total value of the stock. User quantity return method multiplied by current price return method.	Q.1
public double getTotalValue()	Double return method to return the total value of stock	Q.1
public int getQuantity()	Int return method to return the quantity of each stock	Q.1
public SimpleIntegerProperty quantityProperty()	Return SimpleIntegerProperty method to return the quantity value	Q.1
public void setQuantity(int q)	Setter method to set the quantity of the stocks using the Simple InetegerProperty	Q.1
public String getSymbol()	Return method to get the symbol of stock	Q.1
public SimpleStringProperty symbolProperty()	Method using SimpleStringProperty to return the symbol of stock	Q.1
public String getName()	String method to return the name of the stock	Q.1
public SimpleStringProperty nameProperty()	Method using SimpleStringProperty to return the name of the stock	Q.1
public String getSector()	String method to return the sector of the stock	Q.1
public SimpleStringProperty sectorProperty()	Method using SimpleStringProperty to return the symbol of the stock	Q.1
public double getPercentChange()	Double method to return the value of the percent change over time with stock	Q.1
public SimpleDoubleProperty percent ChangeProperty()	Method using SimpleStringProperty to return the percent change of the stock	Q.1
public double getCurPrice()	Double method to return the current price	Q.1
public SimpleDoubleProperty curPriceProperty()	Method using SimpleStringProperty to return the current price of the stock	Q.1
public double getPrevPrice1()	Double method to return the previous price of a stock	Q.1
public SimpleDoubleProperty prevPrice1()	Method using SimpleStringProperty to return the previous price of the stock	Q.1
public double getPrevPrice2()	Double method to return the 2nd previous price of a stock	Q.1
public SimpleDoubleProperty prevPrice2()	Method using SimpleStringProperty to return the 2nd previous price of a stock	Q.1
public void setPurchasePrice()	Method to set the purchase price of the stock	Q.1

Continued

public double getPurchasePrice()	Double method to return the purchase price of the stock	Q.1
public SimpleDoubleProperty purchase PriceProperty()	Method using SimpleDoubleProperty to return the purchase price of the stock	Q.1
public double getSellPrice()	Double method to return the sell price of the stock	Q.1
public SimpleDoubleProperty purchase SellProperty()	Method using SimpleDoubleProperty to return the sold price of the stock	Q.1
public SimpleDoubleProperty accountValue()	Method using SimpleDoubleProperty to return the account value	Q.1
public void setPurchaseValue()	Method using the getQuantity and getPurchasePrice methods to set the purchase value of the stock.	Q.1
public void setNet(double n)	Method to set the Net value of the stock	Q.1
public double getNet()	Double method to return the net of the stock	Q.1
public double getTotalCost()	Double method to return the total cost of a stock	Q.1
public double getProfit()	Double method to return the profit	Q.1

Table 8. Vulnerability assessment for variables in the transactions class.

Variable	Description	Critical Questions Associated with This Variable
static Stock tStock	Static tStock variable used throughout the Transaction class	Q.1
Double totalCost	Double totalCost variable used for total cost of stocks	Q.1
Double acctValue	Double acctValue variable used for account value of transactions	Q.1
Int rowCount = 0	Int rowCount variable set to 0	Q.1
boolean flag = false;	boolean flag variable set to false	Q.1
protected double cost = 0;	protected double variable named cost set to 0	Q.1
static boolean bought = false;	static boolean variable named bought set to false	Q.1, Q.5
static boolean held = false;	static boolean variable named held set to false	Q.1, Q.5
int maxNumshares;	integer variable named maxNumshares for the maximum number of share for each stock	Q.1

Table 9. Vulnerability assessment of methods in the transactions class.

Method	Description	Critical Questions Associated with This Method
public Transaction(Stock s)	Transaction constructor which sets s to the variable made tStock	Q.1
protected void buy(double c)	This protected method is for purchasing stocks. When stock is bought it is ran through a number of if statements that allocates and set the available money to spend per buy transactions. The number of shares is set and if there is not enough money available in the wallet the user is presented with an “insufficient funds”.	Q.2
protected void sell(double c, int q, double purch, String t)	This method is for selling stocks. It retrieves the stock name and calculates total cost of the stock. The cash is then updated within the wallet based on transaction.	
protected void hold()	When the decision is made hold a stock the name of the stock is returned and the variable held is set to true this holding the stock.	
protected void log(String t, String sym, double p, int qty, double total)	This method records all data within transactions. It records the time, date, and year and also the name of the stock. This includes: symbol of the stock, current price, percent change, and the quantity of each stock which is all added to the portfolio.	Q.2
private static void p(String s)	Private method that is used for printing the variable s which is made within the constructor above in the class	
public void updateTable(String query)	This method updates table which is connected to the database and is made to return the necessary data. If statements check connection.	Q.1, Q.2, Q.3

Table 10. Vulnerability assessment for variables in the Trade_Decision class.

Variables	Description	Critical Questions Associated with This Variable
public static int d = 0;	public static integer d variable set to 0	Q.1, Q.5, Q.6

Table 11. Vulnerability assessment for methods in the Trade_Decision class.

Methods	Description	Critical Questions Associated with This Method
public static int w_Strategy(Stock stock)	This method is run through for stocks within the watch list. Here we use three different prices that are refreshed constantly which are then ran through a number of if statements to determine if the stock should be bought, or left alone.	Q.1
public static int p_Strategy(Stock stock)	This method is run through when stocks are in the portfolio. Again they have used three different prices that are ran through a number of if statements; each refreshed 3 different times to determine if the stock should be sold or held within the portfolio.	Q.1

Table 12. Suggestions for eliminating common software security vulnerabilities.

Critical Question	Reasons Causing Vulnerabilities	Suggested Solutions
Q.1	Classes are public	The product should make all classes package-private, since they are in the same package (M3Application) and not served as an API or interface for external classes.
Q.1	Methods are public	This implementation is not appropriate. Methods in the product should obtain at least default access modifier privilege. Most of them should be in private access modifier privilege, since they are used internally.
Q.1	Variables are public	Variables should limit the accessibility. In this product, most variable should be in private access modifier privilege.
Q.2	Resource is not proper closed	This product should use try-with-resources statement to ensure each resource is closed at the end of the statement.
Q.2	More than one resource operations in the try-catch-finally block	Since several exceptions may be thrown, some exceptions will be masked.
Q.5	Static variables	Since other classes in the same scope may be able to access and modify a static variable, this variable should better be claimed as final static.
Q.9	Inner class	Since there are some security issues related to inner class. It is better to move inner classes to outer classes.
Q.12	Sensitive information	Use Java security APIs to handle sensitive information.

classes in the same scope, data integrity can be violated throughout the application. This vulnerability was exemplified in almost every class throughout the application, which can have serious impact to the core of the program. Constantly understanding and being aware of the threats that may occur throughout developing is very important in all applications and languages no matter what you're programming in.

Having awareness throughout implementation of an application by using the guidelines and taking the time to cautiously implement will have great benefits, while also providing a more secure application. Again, the vulnerability assessment results confirm that the capstone project can be exploited to be able to carry out intellectual and also component penetrations. And this result also confirms the importance of our approach for adopting security guidelines into undergraduate software engineering education.

Acknowledgements

Dr. Yen-Hung Hu is an Associate Professor in the Department of Computer Science at Norfolk State University. Prior to joining the NSU, he was the Director of the Information Assurance Center at Hampton University and PI of the NSF CyberCorps: SFS HU GETS-IA program. Charles Scott is a graduate student in the Department of Computer Science at Hampton University and a scholarship recipient of the NSF CyberCrops: SFS: HU GETS-IA program.

References

[1] Raman, J. (2006) Regulating Secure Software Development. University of Lapland Printing Centre, Rovaniemi.

[2] Sinn, R. (2008) Software Security Technologies: A Programmatic Approach. Thomson Course Technology, Boston.

[3] Janssen, C. (2010) Data Security. Techopedia. http://www.techopedia.com/definition/26464/data-security

[4] Janssen, C. "Information Security (IS)". Techopedia. http://www.techopedia.com/definition/10282/information-security-is

[5] "Digital Rights Management". The Free Dictionary. http://www.thefreedictionary.com/Content+security

[6] Grembi, J.C. (2008) Secure Software Development: A Security Programmer's Guide. Thomson Course Technology, Boston.

[7] Oracle, "Java SE Security Documentation". http://www.oracle.com/technetwork/java/index-139231.html

[8] Long, F., Mohindra, D., Seacord, R.C., Sutherland, D.F. and Svoboda, D. (2011) The CERT Oracle Secure Coding Standard for Java, Addison-Wesley Professional.

[9] What Are the Software Development Life Cycle Phases? http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/

[10] Howard, M. and Lipner, S. (2004) The Trustworthy Computing Security Development Lifecycle. *IEEE* 2004 *Annual Computer Security Applications Conference*, Tucson.

[11] Davis, N. (2006) Secure Software Development Life Cycle Process. Carnegie Mellon University, Pittsburgh. https://buildsecurityin.us-cert.gov/articles/knowledge/sdlc-process/secure-software-development-life-cycle-processes.

[12] Maconachy, W.V., Schou, C.D., Ragsda, D. and Welch, D. (2001) A Model for Information Assurance: Integrated Approach. Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, New York, 5-6 June 2001.

[13] Allen, J. (2010) Measuring Software Security. Carnegie Mellon University, Pittsburgh.

[14] Ruiz-Vanoye, J.A., Díaz-Parra, O., Arias, M.D.I.Á.B. and Saenz, A.C. (2013) A Model for Evolutionary Software Development with Security (MESS) Applied to an Electrical Research Institute. *Mexican Journal of Scientific Research*, **2**, 2-22.

[15] Whitman, M.E. and Mattord, H.J. (2012) Principle of Information Security. 4th Edition, Thomson Course Technology, Boston.

[16] "Review: McCumber Cube Methodology," Protect Your Bits, 5 October 2009. http://protectyourbits.wordpress.com/2009/10/05/review-mccumber-cube-methodology/

[17] Jone, C. (2012) Software Quality Metrics: Three Harmful Metrics and Two Helpful Metrics.

[18] Lab, K. (2013) Global Corporate IT Security Risks: 2013.

[19] Alhazmi, O.H., Malyiya, Y.K. and Ray, I. (2006) Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems. *Computer & Security*, 26, 219-228.

[20] Mohagheghi, P., Conradi, R., Killi, O.M. and Schwarz, H. (2006) An Empirical Study of Software Reuse vs. Defect-Density and Stability. *Proceedings of the* 26*th International Conference on Software Engineering*, Edinburgh, 23-28 May 2006, 282-292.

[21] CWE, CWE-844: Weaknesses Addressed by the CERT Java Secure Coding Standard. Common Weakness Enumeration. https://cwe.mitre.org/data/definitions/844.html

[22] Oracle, Secure Coding Guidelines for the Java Programming Language, Version 4.0. http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.