CS 5343 Secure Systems and Software Spring 2015

Assignment#3

Submitted by: **Viswanath Nuggu (tef771)**

Submitted to: **Dr. Gregory B. White**

PART 1: Code Examination

Code sample 1:

"HelloWorld.java" program was written to print a message based on summation of two double values d1 and d2. As per coding should print a message "Hello World!" if value of (d1+d2) > 0.04699753441986 or "Good bye World!" If value of (d1+d2) < 0.04699753441186. But I noticed in the code review the basic purpose of this program is just to print "Hello World!" message.

Static Code Review comments:

1) Code obfuscation was implemented just to print "Hello World!" message.
   Obfuscation is the deliberate act of creating obfuscated code, i.e. source or machine code that is difficult for humans to understand [1].

   Programmers may deliberately obfuscate code to conceal its purpose (security through obscurity) or its logic, in order to prevent tampering, deter reverse engineering, or as a puzzle or recreational challenge for someone reading the source code [1].

   Code obfuscation can be implemented by implementing recursions or iterations.

   In "HelloWorld.java" program, code obfuscation was implemented by using unnecessary conditions, addition and iterations just to make sure that an attacker won't be able to decompile the program and tamper source code or claim ownership on source code.

2) As the values of d1, d2 were hard coded every time you run this program it will give you a message "Hello World!" as (d1+d2) is always > 0.04699753441986 that means else if block will never execute. So else if block is not required. Purpose may be just to print "Hello World!" message. So, else bock will not get executed forever.

3) Using System.out.println() statements is not a good practice, instead use logging.

   Reason:
   The reason is that sending messages to stdout is usually inappropriate in a production environment. If you're coding a library, the library should return information to its caller, but not printing to stdout. If you're coding a GUI app, the information should be presented to the user, not to where stdout happens to be pointing (which might be nowhere). If you're coding a server (or something that runs in a server-side container) you should be using whatever logging facility the framework has provided.

   System.out.println() is an IO-operation and therefore it is time consuming. The Problem is, that your program will wait until the println has finished. This may not be a problem in small applications but as soon as you get load or having more no of iterations it will become clumsier.

   The better approach is to use logging framework. They use a message queue and write only if no other output is going on.

4) For calculating d1 and d2, program used unnecessary iterations which will consume processing resources (this will cause wastage of CPU cycles).

Code Sample 1 can be written in below simple way:

```
public class Helloworld{

public static void main(String args[]){

system.out.println(“Hello World!”);

}

}
```

Code sample 2:

The purpose of this c program is just to print numbers from 0 to 9.

Static Code Review Comments:

Program was obfuscated by using recursions in order to prevent tampering, deter reverse engineering or as a puzzle or recreational challenge for someone reading the source code.

This code uses if loop and a recursive function for which an array, index variable and termination conditions were passed as parameters. It also performed casting to the address of an array as int in main(), recasting it as a pointer in recursiveFunction() and uses pointer arithmetic for array access.

Simple format of this code should like as below code:

```
#include”stdio.h”

int main()

{

int i=0, array[]={0,1,2,3,4,5,6,7,8,9};

for(i=0;i<10;i++)

{

Printf(“%d”,array[i]);

}

return 0;

}
```

Part 2: Zune Failure Lab

Zune was a brand of digital media products and services marketed by Microsoft, it included a line of portable media players, digital media player software for Windows PCs, a music subscription service known as a "Zune Music Pass", music and video streaming services for the Xbox 360 game console via the Zune Software, music, TV and movie sales, and desktop sync software for Windows Phone [2].

Zune Problem:

First-generation Zunes went silent everywhere on December 31, 2008. The cause was soon traced to be a calendrical code in the device's firmware. The problem was caused by a 3rd-party driver written by Freescale for their MC13783 PMIC processor. The bug also froze up Toshiba Gigabeat S media players that share the same Freescale device and driver. The exact cause was first discovered by itsnotabigtruck of Zune Boards [3].

Analysis of the Zune code and Problem cause:

Below is the code that caused problem:

```
year = ORIGINYEAR;//origin year is 1980

while (days > 365) {
  if (IsLeapYear(year))  {
    if (days > 366)  {
      days -= 366;
      year += 1;
    }
   } else {
    days -= 365;
    year += 1;
  }
}
```

In the above code, the variable "days" is the number of days that have elapsed since January 1, 1980. This code is supposed to figure out which year it is, and how many days have elapsed since January 1 of the current year.

If we debug this code on December 31, 2008, "days" variable will contain 10592 as 10592 days had passed since January 1, 1980 (22 normal years and 7 leap years). It follows that 10226 days had passed since January 1, 1981. (Because there were 366 days in 1980, and (10592-366) =10226) applying the same logic repeatedly, we can figure out how many days had passed since January 1 of each subsequent year. We can stop doing this when the number of remaining days is less than a year then we'll know which year it is, and which day within that year [4].

This is the logic used by the Zune code above. It has two variables, days and year, and it maintains the rule that days no of days have passed since January 1 of year. The procedure continues as long as there are more than 365 days remaining (while (days > 365)). If the current year is a leap year ("if

(IsLeapYear(year))“), it subtracts 366 from days and increments year by one, otherwise it subtracts 365 from days and increments year by one.

Consider the code behavior on December 31, 2008, starting with days=10592 and years=1980, the code would eventually reach the point where days=366 and year=2008, which means that 366 days had elapsed since January 1, 2008. This is where things went wrong. The code decided it wasn't time to stop yet, because days were more than 365. (while (days > 365)) It then asked whether year was a leap year, concluding correctly that 2008 was a leap year. (if (IsLeapYear(year))) It next determined that days was not greater than 366 (if (days > 366)), so that no arithmetic should be performed. The code had gotten stuck and it couldn't stop, because days were greater than 365, but it couldn't make progress, because days were not greater than 366. This section of code would keep running forever and making Zune players silent on that day.

The only way out of this mess was to wait until the next day i.e. January 1, 2009, when the computation would go differently and work correctly.

Fix to Zune code:

The problem with this Zune code is it did not handle when days=366, in this case The code is failing because the while part of the code keeps on iterating forever if the year given in the input is a leap year. so we need to handle this situation by adding a break statement and come out of loop in else block.

Below is the modified code:

```
year = ORIGINYEAR;//origin year is 1980

while (days > 365) {
  if (IsLeapYear(year))  {
    if (days > 366)  {
      days -= 366;
      year += 1;
    }
    else{
      break;
    }
   } else {
    days -= 365;
    year += 1;
  }
}
```

Other solution can be adding an if block to handle days in leap year “if(days==366)” and calculating days and year. We can also modify the existing if condition to (days>=366).

Microsoft's comment on bug and official fix:

Microsoft posted a comment on the support front page stating the issue is because 2008 is a leap year, and a firmware clock driver used by the Zune 30 improperly hadles the last day of a

leap year, causing the player to freeze. The driver is for a part used only in the Zune 30 model, which is why the bug didn't affect the other Zune models. The official fix was to drain the device battery and then recharge after midday GMT on 1 January 2009 [3].

From this analysis I noticed, designing a trustworthy application is not easy. Even the applications developed by best companies in the world can also get vulnerabilities at any time.

What happened to the Microsoft SDL process?

I think it was not the problem with Microsoft SDL process but it was due to improper implementation of SDL process. It was the problem due to the use of driver from "Freescale" which was customized and used by Microsoft. So most likely the bug wasn't even Microsoft's, and other Freescale hardware using this same boilerplate code suffers the same fate. And, despite good QA, obviously no one tested this scenario. I suspect they didn't test this code as much as what they engineered themselves, and I think it's an important thing to consider: code reuse is often a good idea but it has security and performance implications.

Microsoft QA team should have tested third party driver also thoroughly. Finding the bug before production would not have defamed Microsoft and face a huge loss in the next quarter.

During that console crisis, the company offered to fix faulty machines free. Microsoft set aside a reported $1.1 billion for the repairs, a figure that suggested to industry analysts that the problem could affect a third of the 11.6 million 360s already in the hands of consumers [5].

Lesson to be learnt from this is "identify the bug as early as possible and fix it". QA teams should make sure they test all the scenarios while integration too. Time lines also need to be allocated during project planning to test third party software.

Suggestions for Testing Process:

1. Testers and developers should be aware of all the business requirements and their implications.
2. Automating Unit testing and integration testing would have helped in catching this kind of bugs in the early stages by mocking with different types of test data.
3. Code review should have been conducted more accurately.
4. I felt that code was not easy to understand, so this might by the reason for not catching this during code review. Follow KIS (Keep it simple) principle to make things easier.
5. QA team should create all the possible test scenarios based on requirement to catch this kind of bugs. They might have missed this scenario, which caused a great loss to Microsoft.
6. All the third party software must be tested before and after integrating in our application.

Part3: Static Code Review and Code Test

Code Review also known as white-box testing is a part of static code analysis and is carried out at the Implementation phase of a Software Development Lifecycle (SDL). It will help in identifying the bugs in the early stages.

Code Review for Medical Application

Below are my code review comments, I mentioned class names and corresponding code review observations for the classes where I noticed an action to be taken.

LoginForm.cs:

1. In line 74 "passlen" variable is calculating length of the password that were stored in "PasswordFile.txt" but not calculating the password length taking as input from the user.
2. Line 72 compares the password length from password.txt and it calculates passlen if it is less than 8 otherwise it is taking passlen as 8. But what if the password is greater than 8? It will consider only first 8 characters from user input and compare this with stored password in PasswordFile.txt. So attackers who can guess first 8 characters of user's password will get unauthorized access.
3. In line 76 it is comparing password from user and from "PasswordFile.txt" but it uses substring functionality which is not required to compare passwords. Instead, code should be able to compare input password and stored password will improve security.
4. In line 79 catch block was written, but it was not defined what to do when an exception is caught.
5. ReadPasswordData() will contain sensitive information and its access specifier is mentioned as "public" which will allow unauthorized access to this data, instead using a "private" access specifier will improve security for this kind on methods.

LoginForm.Design.cs:

1. For password text box in the properties password character was not set so the password will be visible, which is a major security issue.
2. Password textbox property was not set to read only, if it is not set as read only it will allow copying the password and using it by unauthorized user.

OptionScreenForm.cs:

1. btnScheduling_click() was not defined, this is related to scheduling information button, so when user click this button no response will be seen by the user.
2. Instead of hard coding roles (patient, doctor, admin etc…) every time defining constants and using them would be better for maintainability.
3. SetPermissions() should have used protected access specifier, instead of public as this method needs to be more secure.

frmAddCharge.cs:

1. In line 45, 68 pre-constructed SQL queries were used which may lead to security issues like SQL injection.

2. The connection object for data source was opened, but did not close in the code, it may lead to nested connection problems and if we don't close the connection, it will lead to connection memory leakage unless until application server/web server is shut down, the connection will remain activated even though the user logs out.
3. The connection object was created as global but I feel it should be created and closed locally in the local method itself.

manageUsers.cs:

In line 37 StreamReader is being used to read data from "PasswordFile.txt", StreamReader is not thread safe.

manageUsers.Design.cs:

Edit user button's visibility property was set to false so it will not be available to the user and its functionality is not implemented.

frmAddCreditCard.cs

1. The connection object for data source was opened, but did not close in the code, it may lead to nested connection problems and if we don't close the connection, it will lead to connection memory leakage unless until application server/web server is shut down, the connection will remain activated even though the user logs out.
2. In frmAddCreditCard_Shown() method code was missing it was just opening connection, so functional requirement to Add credit card will not work as it is not implemented.
3. In btnSave_Click() pre construced SQL query is being used which may cause SQL injection vulnerability.
4. Connection object was created as global but I feel it should be created and closed locally in the method itself for security issues and for no connection problems.

frmPatientBilling.cs:

1. In FillDataGrid() pre-constructed SQL query is being used which may cause SQL injection vulnerability.
2. The connection object for data source was opened, but did not close in the code, it may lead to nested connection problems and if we don't close the connection, it will lead to connection memory leakage unless until application server/web server is shut down, the connection will remain activated even though the user logs out.
3. In line 70 FindUserInformation() is being called but return value may be null or user value it is not being validated after returning value from called method.
4. FindUserInformation() should have used private access specifier, instead of public as this method needs to be more secure.

UserInformation.cs

In line 12 application used standard strings and byte arrays to store sensitive transient data such as passwords and cryptographic private keys instead of the more secure SecureString class.

public String strPassword;
This is Potentially Unsafe Code - Insecure Storage of Sensitive Information

PatientInformationForm.cs:

1. In dgUserHealth_CellContentClick() has not defined code when a condition is satisfied in the if block.
2. FindUserInformation() should have used private access specifier, instead of public as this method needs to be more secure.
3. In line 58 FindUserInformation() is being called but return value may be null or user value depending on input, but it is not being validated after returning value from called method. So, this may cause unhandled exceptions during run time.

PatientInformationForm.Designer.cs:

1. In line 200 parameters were not passed for dgUserHealth_CellContentClick(), it will create a null pointer exception during run time.
2. Included an Update button but text "update" was not set in the button properties.

Client side data validation is not implemented in any screen used in this application. This is a generic observation.

I suggest following after code review:

1. Placing sensitive information in a plain text format as followed in "PasswordFile.txt" instead such data needs to be either placed in an encrypted format or persist it in databases.
2. This application uses pre constructed SQL queries, building SQL strings in code is problematic. A simple way to remedy this is to leave the completion of the SQL string to the database and to not attempt the SQL string construction in your code. Pass the input of the user to a stored procedure and it should construct the queries in these procedures which is more secure.
3. Whenever your code opens a database connection you must close it after executing your queries, if not it will consume all the connections available.
4. Client validation must be implemented by using JavaScript or Jquery or by any related technology.
5. Writing unit test classes by using a unit test framework like NUnit [7] and automating them will help to identify bugs in the early stages.

Black Box Testing:

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings and this method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance [8].

Method used: manual testing

Black Box testing can be used to test both functional and non-functional requirements.

Non Functional Test Results:

Test Scenario	Non Functional type	Expected Result	Actual Result
on clicking BillingInformation in options view	Portability/ Usability	Application's all modules should work in all windows machines without any configuration changes required on executing machine.	Application throws an exception and popup is displayed showing run time exception if the machine that runs this application did not contain "System.Data.SQLServerce.dll"
On clicking BillingInformation in options view in a machine where SQL server assemblies are available.	Usability	Billing information screen should open without any exception messages or popup messages.	Application throws an exception and popup is displayed showing “Access to database file is denied”.
On clicking BillingInformation in options view in a machine where SQL server assemblies are available and running application as administrator.	Usability/Security	Billing information screen should open without any exception messages or popup messages.	It works as expected but it should not allow database applications to run as “sysadmin” privileges. If it allows it can cause security issues like SQL injection problems.
Searching for patients previous records in “Billing Information” screen	Usability	When a wrong from and to dates were given on click of search button should show an error message	No error message is being displayed. If no error message is displayed user will not trace why it is not displaying.

Password should be invisible while typing	Security	Password should be invisible while typing	Password is visible while typing
Copy the password from Password text box	Security	Password text box should not allow to copy the content in the textbox either by ctrl+c or by using right click option	It allows to copy password from this textbox by using both ctrl+c and by using right click option
authorized users try to login with incorrect passwords	Security	authorized users with incorrect passwords should not get access to resources	If a user uses only first 8 characters from his authorized password it is getting logged in. Example: user "JanePatient" was able to login if he gives "AReallyR" as his password altough his password is "AReallyReallySecurePassword".
Sensitive Information should be kept secret.	Security/Privacy	user information should be kept secret and encrypted	"PasswordFile.txt" is available in the installed folder which contains all user information without encryption, so every user information can be accessed by everyone who ever install this application.
Password policy	Security	good password policy should be implemented to make password guessing tough	No password policy is being followed it allows users with weak and no passwords also to login. Example: DrJones was able to login without giving any password.
Account Lockout	Security	Account should get locked after 'n' attempts unsuccessful attempts.	no account lockout policy implemented
when user logout	Security	when user logout it should close the window or it should go to home screen where credentials should not be visible	credentials are visible when user selects logout in the home screen

services provided by application should be available	Availability	All the services should be available to authorized users to access them.	Application needs to be run as administrator to access "Billing Information" service otherwise an error message is populated indicating to run as system administrator indicating denial of access to database.
sensitive information of the user should not be displayed	Privacy	SSN of the users should not be displayed to others	patients SSN can be viewed by doctor, admin, nurse
data inconsistency in dropdowns in billininformation screen and patientinformation screen	Scalability/ Capacity	All the patients that were added should be available to select in drop down in billinginformation screen and patient information screen.	If total patients added are more than 20 in the drop down it is not listing all users.
No Exceptions should be shown to users.	Reliability	All the Exception scenarios should be handled	There are events/ scenarios where exceptions were not handled properly and system failure messages (unhandled exceptions) were shown to user.
Adding and deleting Administrators	Manageability / Security	Only a super administrator should be able to delete, add administrators	There is no super administrator. Any administrator can delete any other administrator. Even administrator who logged in can delete his own account. Super administrator will help in managing all users including administrators and improve security too.

Functional Test Results:

Test Scenario	Expected Result	Actual Result	Comments

Adding a new user	user without password should not be added	User without giving password can also be added.	Validation is not performed on this text box
On clicking Scheduling Information button	It should display scheduling information screen	It is not displaying requested screen or no error message is being shown.	All the functionalities needs to be implemented, this bug should have identified in the unit/integration testing
Deleting users	Delete user should delete only a single user at a time.	If multiple users were selected by using ctrl key and click on delete a user button it is allowing to delete all the selected users.	Ctrl key functionality should be disabled
Client Side Validation when adding a new user.	Client side validations should be implemented while taking input from user when adding a new user in "Administrative Information" screen	No validations were performed	Client side validations sometimes provide security
Client Side Validation when adding a card details in "Billing Information" screen	Client side validations should be implemented while taking input from user when adding a card details in "Billing Information" screen	No validations were performed	Client side validations sometimes provide security
Client side validations	All client side validations need to be implemented in the application.	Client side validations were not implemented in the application when taking data from user.	There are many places (almost everywhere) no client side data validation is implemented. So I am mentioning this in a generic manner.
Deletion of multiple users at a time	When administrator tries to delete multiple users at a time by selecting users using ctrl key it should not delete multiple users.	It is able to delete all the users selected.	This kind of implementation can result in loss of user data and accounts when admin is not aware of this.
Adding Credit card details	When user tries to save card details without entering any	It is displaying an exception message which should have	All the exceptions must be handled.

	card details it should perform client side validation.	been handled by developer.	
Multiple entries for same card no for same user	For a single user it should not accept same card no. Each card no should be unique for a user.	User can add same card no and related details any no of times.	Improper implementation
Transaction Date should be non editable by user.	While adding a charge user should not be able to edit transaction date	User can change transaction date	Transaction timestamp must be non editable.
When user types incorrect user name or password	An error message should be shown to user.	No error messages were displayed when user types incorrect user name or password.	User should be notified this kind of errors.
User name should be unique	Users with same user name should not be allowed to add.	Users with same user name can be added	This may lead to data inconsistency if SQL queries are using user name as parameter to retrieve user information

References

[1] http://en.wikipedia.org/wiki/Obfuscation_%28software%29

[2] http://en.wikipedia.org/wiki/Zune

[3] http://en.wikipedia.org/wiki/Zune_30

[4] https://freedom-to-tinker.com/blog/felten/debugging-zune-blackout/

[5] http://bits.blogs.nytimes.com/2008/12/31/the-day-microsoft-zunes-stood-still/?_r=0

[6] http://www.zuneboards.com/forums/showthread.php?t=38143

[7] http://nunit.org/

[8] http://en.wikipedia.org/wiki/Black-box_testing