Lab 2: Injection, Accessibility and Extensibility, Input Validation and Mutability

# Problem 1: Simple SQL Injection Attack

Compile the **DynSQLLogin.java** program. This program prompts the user to enter a username and password and then creates a SQL statement to see if a user with that username exists in the **users** database (the same database created and used in the previous problem).

To run the **DynSQLLogin** program use:

Run a terminal window

Copy the **users.sql** script into the current directory of the terminal window in the Kali VM (if you haven’t done this already)

At the command line type in: **mysql -uroot -ppassword < users.sql** and hit return. This will run the script and create the database

**java -classpath .:./mysql-connector-java-5.1.45-bin.jar DynSQLLogin**

The program will output “**Valid**” if the username and password provided by the user are in the database, and “**Invalid**” if not.

Your task is to attempt to fool the program into thinking that an invalid username and/or password is valid when it is not. This involves “injecting” SQL code into the SQL statement being prepared by the program to create a SQL command that is not as intended by the program’s developer.

Find one way to subvert the program via SQL injection.

# Problem 2: Extensibility

You are being given a file called **Person.java** that defines a Person class with the following methods:

* Person()
* Person(name, phone)
* setName
* setPhone
* getName
* getPhone

Both the name and the phone values are stored as **String** objects in the class. Create a file called **UsePerson.java** that contains a **main()** method. In this **main()** method construct an instance of a **Person** object and initialize it to hold a name of “***betty***” with a phone number of “***2345***”.

In **UsePerson.java** create another method with the following signature:

**private static void showPerson(Person p)**

This method should print out the data from the provided Person object in the following format:

***<name>, <phone>***

Call this method in **main()** and give it the instance of the **Person** object you instantiated earlier in main(), make sure that the program works to this point.

Now create another file called **EvilPerson.java**. This implements a class that inherits from **Person**. The **EvilPerson** class doesn’t want you to be able to set a phone number for a user, so it always stores the text “***n/a***” in the phone attribute. You should define a constructor for **EvilPerson** that takes in a name and a phone number, stores the name correctly but throws away the phone number values and stores “***n/a***” there instead. The **setPhone()** method from **Person** should also be overridden in **EvilPerson** to do the same thing (store “***n/a***” instead of the provided phone number).

In the **UsePerson.java** file instantiate an **EvilPerson** object after you instantiate the **Person** object. Try and give a reasonable name and phone number for **EvilPerson**. Add another call to **showPerson()** and give the **EvilPerson** object as the argument to that call. You should see “***n/a***” output for the **EvilPerson**’s phone number.

Finally, create another version of the **Person** class called **PersonSafer**. This class should not be subclassable, and should not allow any changes to the attribute data once they have been set, i.e. it should be *immutable*. Try the same process as before but using the **PersonSafer** class instead of **Person**/**EvilPerson** and ensure that it cannot be misused as the **Person** class can be.

# Problem 3: Reuse via Composition, not Inheritance

You must create a file called **Student.java**. This class will define a **Student** object which is the same as a **Person** object (as defined for problem 1 above), but with the addition of an **gpa** attribute.

The **Student** class must ***NOT*** use inheritance to reuse the functionality from **Person**. Instead you must include a reference to a **Person** object as one of the attributes of **Student** (so that Student “contains” a **Person** object inside itself, this is what we mean by *composition*). The “internal” **Person** object will take care of the name and the phone number for the **Student**, and the **Student** will specifically take care of the **gpa** attribute.

The **Student** class must not allow subclassing, nor changes to any attribute data. Calls to **Student** to get the name or phone number for the **Student** should be “passed through” to the internal **Person** object’s methods, which will return the data to **Student** who then returns it to the code that called the method in **Student**.

You are being given a file called **UseStudent.java** which contains a **main()** method you should use to test that your composed class design is working correctly.

# Problem 4: Problems with Inheritance

You are being given four files for this problem: **People.java**, **UsePeople.java**, **Students.java** and **UseStudents.java**.

**People.java** defines a class containing a “generic” (templated) **ArrayList** that can hold objects of different types (defined by the “**<T>**” in the class definition). The class provided methods for adding and retrieving object references to/from the **ArrayList**, and for returning a reference to the “inner” **ArrayList** as a whole.

In **UsePeople.java** a **People** object is instantiated that can work with **Person**-type objects. Two **Person** objects are added to the **People** object and then their data is retrieved and displayed on-screen.

The **Students** class defined in **Students.java** inherits from the **People** class. It adds two methods used simply to “clean up” the API (they mention “**Student**” rather than “**Person**” in their names), and a method for returning the number of **Student** objects currently stored in the **Students** object (**getNumStudents()**). Otherwise it works the same as **People**.

In the **UseStudents.java** file there is a **main()** method. In there the program instantiates a **Students** object that can store and manage **Student**-type objects. Two **Student** objects are added to the **Students** object. The program then retrieves a reference to the “inner” **ArrayList** from **Students** and calls the “**clear()**” method to delete everything from the **Students** object. This is not supposed to be allowed for **Students**!

Why was this possible to do, and how would you change the classes to stop this from being possible in future?

# Problem 5: Trusted code working with Untrusted code

For this problem you are being given a pre-compiled .class file called **Untrusted.class**. This file contains the bytecode for a Java class called **Untrusted**, with a method called **concatName(String firstname, String lastname)** which takes two arguments, a first name and a last name (as **String**s) and returns a **String** consisting of the first name followed by a space followed by the last name.

The **Untrusted** class has an unpredictable odd behaviour. Discover what it is. You need to write a **main()** class that will instantiate and use an **Untrusted** object, calling the **concatName()** method in it and checking the return value for anything strange. Reject any strange values.

# Problem 6: Copying objects by Cloning and via Copy Constructor

Examine CopyByCloning.java and UseCopyByCloning. Execute UseCopyByCloning. Why do the values change as they do?

Examine CopyByCopyConstructor and UseCopyByCopyConstructor. Execute UseCopyByCopyConstructor. What is the difference between this version of the program and the previous one and why?

# Problem 7: Copy Input and Output values when dealing with Untrusted code

## Part 1:

**CopyInput** that contains two methods:

1. **main()**
2. **private static void printValue(StringBuffer sb)**

The **main()** method instantiates a **StringBuffer** object initialized to “***Hello!***” and passes it to the **printValue()** method to be displayed. The **printValue()** method prints out the **StringBuffer**’s text, and then should sneakily append the word “***HACKED***” to the end of the **StringBuffer**. When the program returned to **main()** after calling **printValue()** **call printValue()** a second time and see what the program displays. Change the code in the main method so that it prints the string twice *without* displaying “HACKED”.

## Part 2:

The class called **CopyOutput** that contains a main() method and an inner class called **OutputValue**.

The **OutputValue** class contains a **StringBuffer** as an attribute, initialized to “***Hello!***”. It also has a method called **getText()** which returns the reference to the **StringBuffer** when called.

The **main()** method in **CopyOutput** instantiates an **OutputValue** object and calls **getText()** on it to print out the text stored in it. Then the **main()** method appends the text “***HACKED***” to the end of the text stored in the **StringBuffer** stored in the **OutputValue** object. Again, print out the text returned by calling **getText()** on the **OutputValue** object. Why has the value changed? Change the code in getText() in the OutputValue class to prevent “HACKED” being displayed.