Topic	CODE REVIEW	
Class Description	**Students will go through the code of a secure app and understand how certain vulnerabilities could have been avoided.**	
Class	**C-239**	
Class time	**45 mins**	
Goal	• Review and compare the code of a secure v/s non secure app • Understand how to secure an application in various ways	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Visual Studio Code
	
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Visual Studio Code	

Class structure	**Warm-Up** **Teacher and Student led Activity** **Wrap-Up**	**10 mins** **30 mins** **5 mins**
WARM-UP SESSION - 10mins		
Teacher Action		**Student Action**

Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? In the last session, we went through the code of the ecommerce application that we have been hacking in various ways from the past few classes. Any doubts from the last session? *The teacher clarifies doubts (if any)* *With all the vulnerabilities we have seen so far, we are now ready to understand how to avoid them. Today we will be reviewing the code of a secure application, comparing it with the non secure application and understanding how it is avoiding certain vulnerabilities.* *Let's get started!*	**ESR**: Hi, thanks, yes, I am excited about it! **ESR**: Varied!
Q&A Session	
Question	**Answer**
Which method is used to close the socket? A. flush() B. send() C. close() D. accept()	**C**
When using the scrollbar() widget in Tkinter, why do we use the yview property? A. To move content from top to bottom	**A**

B. To move content from left to right C. To move content from right to left D. None of the above	

TEACHER & STUDENT LED ACTIVITY - 30 mins

Teacher Initiates Screen Share

ACTIVITY

- **Understanding the secure application's code for ecommerce app**
- **Comparing it with non-secure code**

Teacher Action	Student Action
Note - *The student and the teacher are expected to have a discussion in this class. Teachers should engage the student more and ask questions about if they can understand the code first or not, or if they recall doing something similar or using similar code elsewhere. Let the student first see the secure application's code for themselves, compare it with non secure application's code and help them understand where and how it's different and how it's making the application secure.* ***Note -*** *The non secure application's code from the last class is available in Teacher and Student Activity 1 and the secure application's code is available in Teacher and Student Activity 2.* Now that we have seen how a large application works, we are ready to see how to make it secure. Most of the large scale applications are written this way, with their views and apis separated from each other. Can you tell me what Flask uses to separate these 2 different types of functions?	**ESR:** Blueprints

Excellent! Now, let's get started with the code. ***Note -*** *For simplicity, both the activities can be opened side-by-side in the browser as well, and can be reviewed from Github instead of cloning it and opening it in VS Code.* You can clone the secure version of the ecommerce application from Student Activity 1. It's original non secure version from the last class is available in Student Activity 2. *Teacher helps the student in cloning the repositories from Student Activity 1 and 2 and opening them separately in VS Code.* Okay, now let's try to understand how we could have avoided the SQL Injection Attack! Can you see any differences in code where the SQL queries are implemented in both the secure and non-secure version of the code? *Help the student explore the differences. Let them spend a good 5 minutes trying to explore it by themselves and engage in a discussion with them.*	*Student clones the repositories from Student Activity 1 and 2 and opens them separately in VS Code.*
In the non-secure version, the way we query SQL is this -	

```python
@views.route('/order')
@cross_origin()
def order():
    try:
        product_id = request.args.get("id")
        if not product_id:
            return jsonify({
                "message": "No product for purchase!",
                "status": "error"
            }), 400
        query = f"select * from products where id={product_id};"
        product = db.engine.execute(query).first()
        address_query = f"select * from address where user_id='{session.get('user_id')}'"
        addresses = db.engine.execute(address_query).all() or []
        return render_template("/order/order.html", product=product, addresses=addresses, user_id=session.get('user_id'))
    except Exception as e:
        return jsonify({
            "message": str(e),
            "status": "error"
        }), 400
```

(This is the *"/order"* function in *views.py*)

Now, if we take a look at the same code snippet in the secure code, it is -

```python
@views.route('/order')
@cross_origin()
def order():
    try:
        product_id = request.args.get("id")
        if not product_id:
            return jsonify({
                "message": "No product for purchase!",
                "status": "error"
            }), 400
        product = Products.query.filter_by(id=product_id).first()
        addresses = Address.query.filter_by(user_id=user.id).all()
        return render_template("/order/order.html", product=product, addresses=addresses, user_id=session.get('user_id'))
    except Exception as e:
        return jsonify({
            "message": str(e),
            "status": "error"
        }), 400
```

Here, you will notice that the code has become short. Earlier in the non-secure version, we were first constructing the query and then we were executing it, but this time it's different.	**ESR:**

You may wonder, how does this still work? The answer is SQLAlchemy. It's a library that provides short and secure syntaxes for building and executing queries. Let's take a look at the ***users.py*** in the ***models*** folder to understand this better -	Yes!

```
class Users(db.Model):
    __tablename__ = "users"
    id = db.Column(db.Integer, primary_key=True)
    guid = db.Column(db.String, nullable=False, unique=True)
    name = db.Column(db.String(64))
    email = db.Column(db.String(64))
    password = db.Column(db.String(64))
    contact = db.Column(db.String(64))
    addresses = db.relationship(Address, lazy=True, backref="user")
    orders = db.relationship(Orders, lazy=True, backref="user")
    tickets = db.relationship(Tickets, lazy=True, backref="user")
```

Here, you would notice that we have defined the schema of our User's table through a class that takes ***db.Model*** as an argument.

This means that this class ***Users*** is a SQL model for a table whose **tablename** would be ***users*** and then we have listed all the columns with their types.

Here, ***backref="user"*** means that if someone is querying the ***address*** or ***order*** or ***tickets***, they can know which user it belongs to.

Now since this is a class, which is a concept of object oriented programming, it has some functions of it's own.

When we pass ***db.Model*** as an argument, it takes on the functions of SQLAlchemy too which then enables us to construct queries by using its function ***query***.

All the models and tables in our application are constructed in a similar way!

Therefore, when we look at the following line -

```
product = Products.query.filter_by(id=product_id).first()
```

We are actually using SQLAlchemy's function ***query*** to query the table ***Products*** with a ***where*** condition defined in the ***filter_by()*** function and the ***first()*** means that we want to get just the first value, since we are expecting and needing only one result here.

Now SQLAlchemy itself provides a secure layer where SQL Injection cannot be performed on the database.

Also, when we fetch the product this way, we are getting the ***object*** of a product it found for us, which means that all the columns in the product's table can now be accessed as attributes while earlier we were receiving a list of values.

Therefore, with SQL Alchemy, if we want to get the id of the product that we have just received, then we can simply do -

product.id

Instead of

product['id']

Take a look at the profile page's view function in the secure code -

```python
@views.route('/profile')
@cross_origin()
def profile():
    try:
        user_id = session.get('user_id')
        user = Users.query.filter_by(id=user_id).first()
        order_query = f"select p.image, p.name, o.amount from products p right join orders o on o.user_id={user['id']} and p.id=o.product_id;"
        orders = db.engine.execute(order_query).all()
        tickets = Tickets.query.filter_by(user_id=user.id).all()
        addresses = Address.query.filter_by(user_id=user.id).all()
        return render_template("/profile/profile.html", user=user, orders=orders, addresses=addresses, tickets=tickets, user_id=user.id)
    except Exception as e:
        return jsonify({
            "message": str(e),
            "status": "error"
        }), 400
```

Here, when querying tickets or addresses, we are simply using ***user.id*** instead of ***user['id']*** because we now have user as a class object whose attributes we can directly access since we are now using SQLAlchemy. Explore the ***api.py*** file too to see how SQLAlchemy is used in various places! *Teacher helps the student in exploring the **api.py** file and discusses how SQLAlchemy is being used in the secure version of the app in comparison to the non secure version.*	
Okay, now our next vulnerability was an IDOR attack that occurs because the URL can be manipulated to get unauthorized data of other users. Can you tell me how it was happening in code and how it got fixed? For it, let's take a look at the ***login*** function in the ***api.py*** -	**ESR:** Varied!

```
if user:
    session["email"] = email
    session["user_id"] = user[0]
    return jsonify({
        "status": "success",
        "id": user[0]
    }), 200
```

Here, after we are validating that the user has entered the correct credentials, we are using a ***session*** we have imported from flask and we are saving the user's ***email*** and ***user_id*** inside of it.

What is a ***session***?

Sessions are browser sessions for when a user has logged in. It works like a dictionary and can hold values for each and every user for further use.

Let's say that both you and I logged into an application. The application now needs to remember you and me in order to keep us logged in right?

Everytime we open the application from our computer, it needs to identify your computer and display your information to you and identify my computer and display my information to me.

Sessions help with that. They remember the computer and who logged in from that computer.

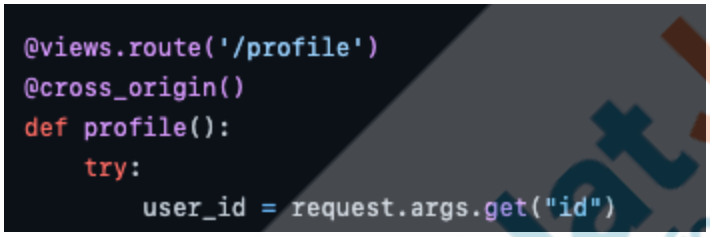
Therefore, in this case, if let's say, 5 users have logged in from 5 different computers, then the session would remember each of their emails and user ids with their computer so that we can later on display the right information to the right user.

Since we are taking the ***user_id*** and saving it inside of a

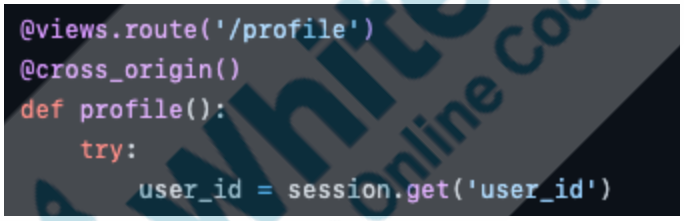
ESR:
Varied!

<table>
<tr><td>session, then we can access the user ID in the profile page from the session instead of from the URL and that's what we have done!

If you take a closer look in the profile function in view.py, you will notice that -</td><td></td></tr>
<tr><td colspan="2">Non Secure Version -

Secure Version -

 
</td></tr>
<tr><td>Earlier, we were using request.args.get() function to get the value of user_id from the URL, but now, we are simply using session.get() function to get the value.

This way, our application would always by default know which users data it needs to display based on which computer we are trying to access the data.

Amazing, isn't it?

This will also affect the cloning of the profile page that we did. We were using the user_id in the URL and were getting the information and content of the profile page based on that, but now since our app would see if our</td><td>ESR:
Yes!</td></tr>
</table>

computer has any user_id saved in it's session or not, we would have to ensure that in order to do the cloning, we are logged into the application from our computer as well. Not just ours, but our victim on whom we are performing the attack should also be logged into this application or the attack would not work. Now, let's move to our next vulnerability, which was bypassing the image upload and being able to upload some other files like HTML files or EXE files that we did. Do you think we would have been able to perform the phishing attack if we were not able to upload these files in the first place? In order to avoid other files getting uploaded into the server, we should always validate the extension of the file that is being uploaded. For example, we know that the image file we are expecting would always be one of ***png, jpg, jpeg*** or maybe even a ***.gif*** Let's take a look at the upload function in the ***api.py*** -	**ESR:** No!

```python
@api.route("/submit-help", methods=["POST"])
def submit_help():
    title = request.form.get("title")
    description = request.form.get("description")
    attachment = request.files.get("attachment")
    if attachment:
        filename = secure_filename(attachment.filename)
        attachment.save(os.path.join(UPLOAD_FOLDER, filename))
    user_email = session.get("email")
    user_query = f"select * from users where email='{user_email}';"
    user = db.engine.execute(user_query).first()
    Tickets.create(user["id"], title, description, filename)
    return jsonify(
            {
                "status": "success",
            }, 201
        )
```

Here, if we were validating the extension of the file before calling the ***attachment.save()*** function, then we would have avoided the image bypass that anyone could attempt!

We could do this in the following way -

```python
@api.route("/submit-help", methods=["POST"])
def submit_help():
    title = request.form.get("title")
    description = request.form.get("description")
    attachment = request.files.get("attachment")
    if attachment:
        filename = secure_filename(attachment.filename)
        extension = filename.split(".")[1]
        if extension.lower() not in [".png", ".jpg", ".jpeg", ".gif"]:
            return jsonify({
                "status": "error",
                "message": "Invalid file!"
            }), 400
        attachment.save(os.path.join(UPLOAD_FOLDER, filename))
    user_email = session.get("email")
    user_query = f"select * from users where email='{user_email}';"
    user = db.engine.execute(user_query).first()
    Tickets.create(user["id"], title, description, filename)
    return jsonify(
            {
                "status": "success",
            }, 201
        )
```

Here, we are splitting the value of the filename with a ***dot "."*** and taking the second value.

This would also help if the filename is in the following format to avoid detection -

File.html.png

The split function in this case would return 3 elements, ***File, html*** and ***png*** and our second value would be ***html***.

Also, if the file is simple like -

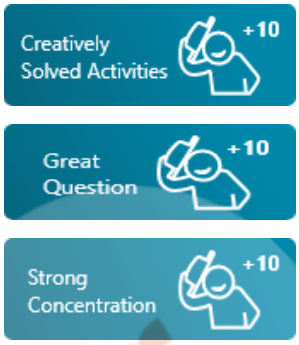
File.html

Again, the second value becomes ***html***

ESR:
Varied!

Next, we are saving this second value in a variable called ***extension*** and we are making sure that it is one of the following ***.png, .jpeg, .jpg or .gif***. We are using the ***lower()*** function on the extension to avoid missing any files that might have extensions in capital. If the extension is not one of these, we are giving a 400 Bad Request Error to the user with the message ***"Invalid File"***	
You may have noticed that just with the help of simple techniques and methods and logic, we were able to greatly improve the security of the ecommerce application. In the secure version, neither SQL Injection, nor IDOR Attack or Phishing Attack can be performed. A lot of hackers spend days and weeks trying to find vulnerabilities in existing websites, and when found, they report it to the concerned company officials. This is known as a ***Bug Bounty*** and it is rewarded handsomely by the companies depending on how big the vulnerability is. The hackers who report it for rewards are known as ***White Hats*** whereas those who misuse these vulnerabilities are known as ***Black Hats***. You can understand more about SQLAlchemy by referring to Teacher and Student Activity 3. It's a little bit high level, but looking at the code and the documentation would help greatly in understanding it.	
Teacher Guides Student to Stop Screen Share	

WRAP UP SESSION - 5 Mins	
Quiz time - Click on in-class quiz	
Question	**Answer**
What is the purpose of sm in Bootstrap? A. Scale of screen B. Column width in small screen C. Column width in desktop D. None of the above	**B**
How do you represent the closing of an HTML tag? A. / B. // C. \ D. /-	**A**
What is the purpose of loops? A. Repeat portion of code B. Repeat set of number of times C. Repeat until desired result D. All of the above	**D**
End the quiz panel	
FEEDBACK • **Appreciate the students for their efforts in the class.** • **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**	
Teacher Action	**Student Action**
You get Hats off for your excellent work! In the next class we will learn about SQL Union	*Make sure you have given at least 2 Hats Off during the class for:*



Project Discussion

The ecommerce company has approached you to create a detailed document of all the vulnerabilities that the application has and how it can be avoided.

They are happy with your bug-bounty skills and want to award you after getting the document. Your task is to create a detailed document on the vulnerabilities identified in the resolve for it.

Teacher Clicks ✖ End Class

ADDITIONAL ACTIVITIES

Additional Activities

Encourage the student to write reflection notes in their reflection journal using markdown.

Use these as guiding questions:

- What happened today?
  - Describe what happened.
  - The code I wrote.

The student uses the markdown editor to write her/his reflections in the reflection journal.

• How did I feel after the class? • What have I learned about programming and developing games? • What aspects of the class helped me? What did I find difficult?	

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity 1	Ecommerce Code	https://github.com/pro-whitehatjr/networking-ecommerce
Teacher Activity 2	Secure Ecommerce App	https://github.com/pro-whitehatjr/networking-ecommerce-secure
Teacher Activity 3	SQLAlchemy Documentation	https://docs.sqlalchemy.org/en/14/
Student Activity 1	Ecommerce Code	https://github.com/pro-whitehatjr/networking-ecommerce
Student Activity 2	Secure Ecommerce App	https://github.com/pro-whitehatjr/networking-ecommerce-secure
Student Activity 3	SQLAlchemy Documentation	https://docs.sqlalchemy.org/en/14/