SQL Injection Vulnerability

```
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
```

It can be rectified by using SQL alchemy syntax:

```
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
```

IDOR Vulnerability

Here the email and user_id is saved in *Sessions* of the user if the credentials are correct. Hence user_id can be accessed directly from the session instead of URL

```
def login():
    try:
        return render_template("/login/login.html")
    except Exception as e:
        return jsonify({
            "message": str(e),
            "status": "error"
        }), 400
```

Authorisation Issue

```
def profile():
    try:
        user_id = request.args.get("id")
        user_query = f"select * from users where id='{user_id}';"
```

Hence the secure version of the code is:

```
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
        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()
```

Phishing Attack

In the following code, there is no check for the type of file that can be uploaded.

```
@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
        )
```

Hence the rectified code would be:

```
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 = Users.query.filter_by(email=user_email).first()
    Tickets.create(user.id, title, description, filename)
    return jsonify(
            {
                "status": "success",
            }, 201
        )
```