

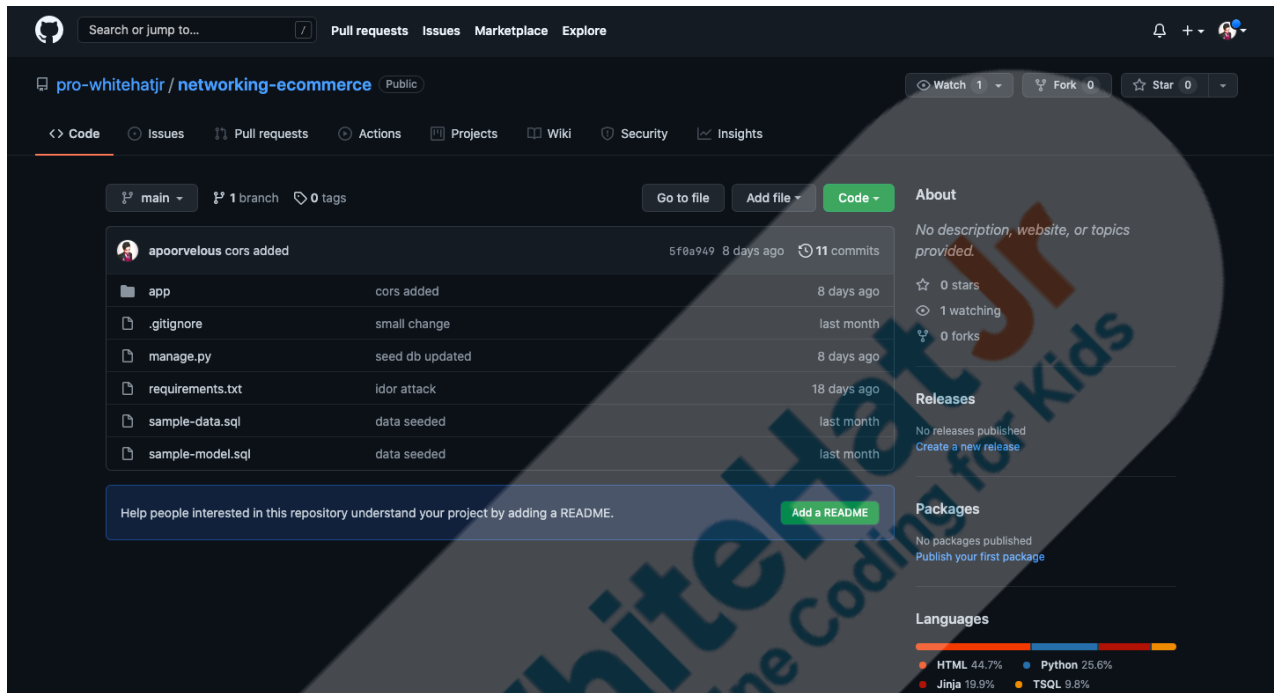
Topic	Identifying Vulnerabilities	
Class Description	Students will go through the code and try to identify different vulnerabilities and discuss/implement their solutions.	
Class	C-238	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> <li>Understanding the ecommerce website code</li> </ul>	
Resources Required	<ul style="list-style-type: none"> <li>Teacher Resources:               <ul style="list-style-type: none"> <li>Laptop with internet connectivity</li> <li>Earphones with mic</li> <li>Notebook and pen</li> <li>Visual Studio Code</li> </ul> </li> <li>Student Resources:               <ul style="list-style-type: none"> <li>Laptop with internet connectivity</li> <li>Earphones with mic</li> <li>Notebook and pen</li> <li>Visual Studio Code</li> </ul> </li> </ul>	
Class structure	<b>Warm-Up</b> <b>Teacher and Student led Activity</b> <b>Wrap-Up</b>	<b>10 mins</b> <b>30 mins</b> <b>5 mins</b>
<b>WARM-UP SESSION - 10mins</b>		
<b>Teacher Action</b>		<b>Student Action</b>
<i>Hey &lt;student's name&gt;. How are you? It's great to see you!</i> <i>Are you excited to learn something new today?</i>  In the last session, we again performed a phishing attack		<b>ESR:</b> Hi, thanks, yes, I am excited about it!

<p>where we uploaded our HTML code bypassing an image upload and made our cloned page a part of the website.</p> <p>Any doubts from the last session?</p> <p><i>The teacher clarifies doubts (if any)</i></p> <p><i>We have studied so many vulnerabilities so far. We saw how there is no data authorisation i.e. anyone could access anyone's data. We also saw how SQL Injection, IDOR attack or image bypassing techniques could also be used by any of the hackers. In today's class, we will be looking closely at the code of this website and see how exactly this is happening and how we can avoid it.</i></p> <p><i>Let's get started!</i></p>	<p><b>ESR:</b> Varied!</p>
<p><b>Q&amp;A Session</b></p>	
Question	Answer
<p>Which of the following libraries can be used to convert python code to <b>exe</b> file?</p> <ol style="list-style-type: none"> <li>1. installer</li> <li>2. pyinstall</li> <li>3. pyinstaller</li> <li>4. pythoninstaller</li> </ol>	<p><b>C</b></p>
<p>Which of the following commands is used to change the permission of the file?</p> <ol style="list-style-type: none"> <li>1. sudo</li> <li>2. chmod</li> <li>3. chpermission</li> </ol>	<p><b>B</b></p>

4. sudo permission	
<b>TEACHER &amp; STUDENT LED ACTIVITY - 30 mins</b>	
<b>Teacher Initiates Screen Share</b>	
<p align="center"><b><u>ACTIVITY</u></b></p> <ul style="list-style-type: none"> <li>Understanding the Ecommerce Code</li> </ul>	
Teacher Action	Student Action
<p><i>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.</i></p> <p>Over the past few classes, we have exploited quite a lot of vulnerabilities while trying to hack this platform.</p> <p>Can you tell me what all we did?</p> <p>Okay, now those are quite a lot of vulnerabilities that we have explored!</p> <p>Now, let's look at this website's code once and see how it works!</p>	<p><b>ESR:</b></p> <ol style="list-style-type: none"> <li>1. We learnt SQL and performed SQL Injection</li> <li>2. We learnt about the IDOR attack and performed it.</li> <li>3. We performed a phishing attack bypassing image</li> </ol>

*Teacher opens the Teacher Activity 1*


*Student opens Student Activity 1*



Let's understand how this code works first. It is not the conventional flask code we have written so far, but rather it is written in a different way because of Databases that we are using and a mix of APIs and Views.

Do not be worried yet! Let's take this piece by piece.

What you are seeing right now is the following -

 <b>apoorvelous</b> cors added			5f0a949 8 days ago	🕒 11 commits
📁 app	cors added			8 days ago
📄 .gitignore	small change			last month
📄 manage.py	seed db updated			8 days ago
📄 requirements.txt	idor attack			18 days ago
📄 sample-data.sql	data seeded			last month
📄 sample-model.sql	data seeded			last month

There are following things -

1. **app** folder
2. **.gitignore** file
3. **manage.py**
4. **requirements.txt**
5. **sql files**

Now, out of these, the SQL files are irrelevant. They are just used to seed the data into the database. It's like having some data filled already into the database.

We also know what the **.gitignore** file is. Can you tell me what it is?

That's right! Similarly, we have a **requirements.txt** while, which contains the names of all the libraries required for this application to run. All of these libraries can be installed at once using a single command -

**`pip install -r requirements.txt`**

Here is the content of the file -

#### ESR:

It is a file that let's github know which folders and files we do not want to upload to github

5 lines (5 sloc) 59 Bytes

```
1 flask
2 jinja2
3 flask-sqlalchemy
4 flask-migrate
5 psycopg2-binary
```

Raw Blame

Next, comes the ***manage.py*** file.

This is one of the most important files that manage how this application is run. This is where our normal application that we have built so far using Flask becomes different from this.

Let's take a look at the ***manage.py*** first -

255 lines (239 sloc) 5.9 KB

```
1 from flask.cli import FlaskGroup
2 from app import create_app, db
3 from flask import current_app
4
5 from datetime import datetime
6 import csv
7 import os
8
9 from app.models.users import Users
10 from app.models.products import Products
11
12 from app.models.editor.customer import Customer
13 from app.models.editor.supplier import Supplier
14 from app.models.editor.company_products import CompanyProducts
15 from app.models.editor.company_orders import CompanyOrders
16 from app.models.editor.order_item import OrderItems
```

Raw Blame

At the top of the file, we have the regular imports. Do notice the ***app.models.\**** syntax that we have at the very beginning too. Let's ignore these for now, and just look at what the code of this file contains further.

If we scroll down a little, we will see a list of dictionaries that contains some of the seeded data of users and products that we have already seen on the website -

```
user_json = [  
    {  
        "name": "John Doe",  
        "email": "john.doe@gmail.com",  
        "password": "hello_john",  
        "contact": "+1 (1234) 123 123"  
    },  
    {  
        "name": "James Bond",  
        "email": "james.bond@gmail.com",  
        "password": "bond007",  
        "contact": "+37 (1234) 567 890"  
    },  
    {  
        "name": "Amy Brown",  
        "email": "amy.brown@gmail.com",  
        "password": "brownamy3",  
        "contact": "+83 (4456) 285 847"  
    },  
    {  
        "name": "John Wick",  
        "email": "john.wick@gmail.com",  
        "password": "johndog_1",  
        "contact": "+49 (8712) 419 063"  
    },  
    {  
        "name": "Helene Sebi",  
        "email": "helene.sebi@gmail.com",  
        "password": "helenelove1",  
        "contact": "+33 (9331) 740 903"  
    },  
]
```

We also have a **FlaskGroup** function, that can be used to create commands to manage this flask application -

```
cli = FlaskGroup(create_app=create_app)
```

We are saving it in a variable called **cli**.

Next, we have created a few functions called **recreate\_db** and **seeder()**.

**recreate\_db()** is used to reset the database, and **seeder()** is used to again add some data into the database that is just resetted -

```
def recreate_db():
    db.drop_all()
    db.create_all()
    db.session.commit()

def seeder():
    for user in user_json:
        Users.create(user.get("name"), user.get("email"), user.get("password"), user.get("contact"))

    for product in product_json:
        Products.create(product.get("name"), product.get("image"), product.get("rating"), product.get("marked_price"), product.get("selling_price"))

    #Seeding the editor
    with open("app/editor_data/customer.csv", "r") as f:
        csvreader = csv.reader(f)
        for row in csvreader:
            try:
                Customer.create(int(row[0]), row[1], row[2], row[3], row[4], row[5])
            except:
                pass
```

Finally, we are using the **cli** variable to declare a command called **rsd()** which calls both **recreate\_db()** and **seeder()** functions -

```
@cli.command()
def rsd():
    # if current_app.config.get('ENV') not in ('development', 'test', 'testing'):
    #     print("ERROR: seed-db only allowed in development and testing env.")
    #     return
    recreate_db()
    seeder()

if __name__ == '__main__':
    cli()
```

The way this command can be used is in the following way -

**python manage.py rsd**



Here, **rsd** is the name of the function. This way, manage.py can be used to create command line tools that can be used to create commands and manage applications.

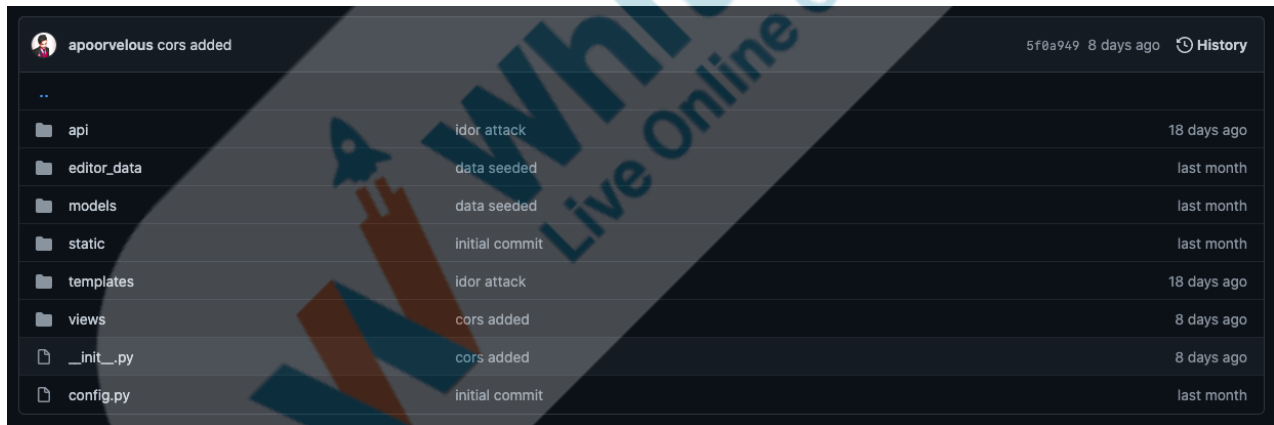
When we use the **FlaskGroup()** function, we are usually generating a new command for the terminal too, which is -

***python manage.py run***

This command is created by default using the **FlaskGroup()** function, and it is used to just run or start the app.

Now, you may wonder, where is the code of the application?

It's in the **app** folder! Let's take a look at the contents of the app folder now -



File/Folder	Commit Message	Time Ago
api	ldor attack	18 days ago
editor_data	data seeded	last month
models	data seeded	last month
static	initial commit	last month
templates	ldor attack	18 days ago
views	cors added	8 days ago
__init__.py	cors added	8 days ago
config.py	initial commit	last month

Here, we have 2 files, called **\_\_init\_\_.py** and **config.py**

**config.py** is used to set some configurations for the file. Most of it is unimportant stuff and some that is used frequently is the environment selection (if it is in development, stage or production) or the URL of the database, if you are using a SQL database.

<p>Do you understand what <b>development, stage and production</b> environments mean?</p> <p>Development environment is when the application is still in development by developers. They may use less security checks so that they can access stuff easily and make changes to the code.</p> <p>Stage environment is for testing the application. It can be used by project managers to see if everything is working fine and as per their needs, and let developers know if there are any changes.</p> <p>Production environment is when the application is live for the users to use.</p> <p>Now comes the <code>__init__.py</code> file.</p> <p>This is one of the most important files in Flask, used for initialising a folder.</p> <p>Sometimes, you have folders inside of folders and we want the folders to act as an application. In these scenarios, the <code>__init__.py</code> file is used to initialise a folder.</p> <p>Let's look at the contents of the file -</p>	<p><b>ESR:</b> Varied!</p>
<pre>import os from flask import Flask, jsonify from flask_cors import CORS from flask_sqlalchemy import SQLAlchemy from flask_migrate import Migrate  # instantiate the extensions db = SQLAlchemy() migrate = Migrate()</pre>	

Here, we are importing some modules that we are using in this App. **Cors** is used to allow cross origin in an application.

Similarly, **SQLAlchemy** is used to write SQL queries and execute SQL queries through Flask.

**Migrate** is used to migrate data into the database. If there is any change in the database models or anything, then migrate can be used to change the database models without any loss of data.

Next, we have the usual initialisation of applications that we do in the normal applications as well -

```
def create_app(script_info=None):
    # instantiate the app
    app = Flask(__name__)
    cors = CORS(app)

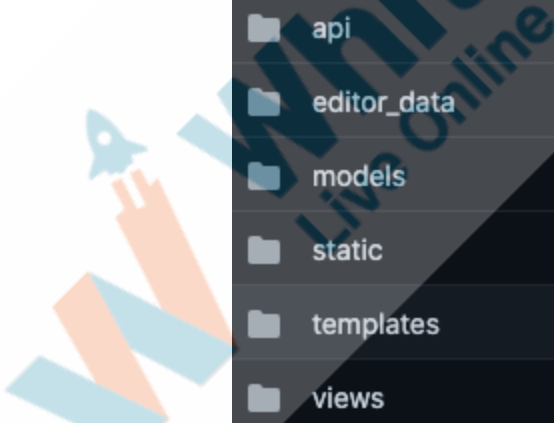
    # set configz
    app_settings = os.getenv('APP_SETTINGS')
    app.config.from_object(app_settings)
    app.config['CORS_HEADERS'] = 'Content-Type'

    # set up extensions
    db.init_app(app)
    migrate.init_app(app, db)




    # register blueprints
    from .views.views import views
    from .api.api import api

    app.register_blueprint(views)
    app.register_blueprint(api)
```

We have a function called **create\_app()** which is used to initialise the Flask Application and allow it for Cross Origin

<p>Requests.</p> <p>Then, we are using Config.py to get different environment variables that we have to use.</p> <p>Finally, we are initialising the Database and then registering Blueprints with the <b>register_blueprint()</b> function.</p> <p>This initialises the application for us.</p> <p>Now, you may wonder, what is a blueprint? Any guesses?</p> <p>Blueprints can be used to set the different types of routes differently. For example, in this application, there are different routes for Frontend and Backend. The <b>views</b> contains the routes for Frontend and the <b>api</b> contains the routes for backend.</p> <p>Now, let's take a look at the folders -</p>	<p><b>ESR:</b> Varied!</p>
	
<p>As we discussed, the <b>api</b> folder has the backend routes while the <b>views</b> folder has the frontend routes.</p> <p>Similarly, we have the <b>static</b> and the <b>templates</b> folder that contains the static data and the HTML/CSS data in the <b>templates</b> folder.</p>	

Then, we have the models folder which is used to create models in the database. It is used to define datatypes of the columns of the SQL table.	
<i>Let the student explore the Views, API and Model files to understand the code and have a discussion if they can understand it or not. In the next class, we will be diving deep into this code and see how to resolve the issues.</i>	
In the next class, let's pinpoint the vulnerabilities, their causes and the solutions to these problems and vulnerabilities.	
<b>Teacher Guides Student to Stop Screen Share</b>	
<b>WRAP UP SESSION - 5 Mins</b>	
<b>Quiz time - Click on in-class quiz</b>	
Question	Answer
Which function is used to create a command to manage the flask application?  A. Flaskmanage() B. Flask() C. FlaskGroup() D. Flaskmanager()	
What's the purpose of recreate_db?  <pre>def recreate_db():     db.drop_all()     db.create_all()     db.session.commit()</pre> A. reset the database B. insert new values in the database C. update the database	<b>A</b>

D. delete the database	
Which function should be used to insert some data into a newly reset database?  A. seeder() B. reader() C. feeder() D. creator()	<b>A</b>
<b>End the quiz panel</b>	
<b>FEEDBACK</b> <ul style="list-style-type: none"> <li>• Appreciate the students for their efforts in the class.</li> <li>• Ask the student to make notes for the reflection journal along with the code they wrote in today's class.</li> </ul>	
Teacher Action	Student Action
<p>You get Hats off for your excellent work!</p> <p>In the next class we will learn about SQL Union</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div>           Creatively Solved Activities  +10         </div> <div>           Great Question  +10         </div> <div>           Strong Concentration  +10         </div>
<b>Project Discussion</b> <p>Based on all the code files and folders you've seen in the ecommerce application, your task is to document the</p>	

code and write comments on which block of code does what.	
<div>Teacher Clicks</div> <div>✕ End Class</div>	
<b>ADDITIONAL ACTIVITIES</b>	
<p><b>Additional Activities</b></p> <p><i>Encourage the student to write reflection notes in their reflection journal using markdown.</i></p> <p>Use these as guiding questions:</p> <ul style="list-style-type: none"> <li>• What happened today?           <ul style="list-style-type: none"> <li>◦ Describe what happened.</li> <li>◦ The code I wrote.</li> </ul> </li> <li>• How did I feel after the class?</li> <li>• What have I learned about programming and developing games?</li> <li>• What aspects of the class helped me? What did I find difficult?</li> </ul>	<p><i>The student uses the markdown editor to write her/his reflections in the reflection journal.</i></p>

<b>ACTIVITY LINKS</b>		
Activity Name	Description	Link
Teacher Activity 1	Ecommerce Code	<a href="https://github.com/pro-whitehatjr/networking-ecommerce">https://github.com/pro-whitehatjr/networking-ecommerce</a>
Student Activity 1	Ecommerce Code	<a href="https://github.com/pro-whitehatjr/networking-ecommerce">https://github.com/pro-whitehatjr/networking-ecommerce</a>

© 2021 - WhiteHat Education Technology Private Limited.

Note: This document is the original copyright of WhiteHat Education Technology Private Limited.

Please don't share, download or copy this file without permission.

