LP-3 (ISC)

Mini Project

Title: SQL injection attack and cross site scripting

Group Members:

1. Kasturi Patil (41149)
2. Mayuresh Pingale (41151)
3. Piyush Kulkarni (41152)

Problem Statement:

SQL Injection attacks and Cross -Site Scripting attacks are the two most common attacks on web application. Develop a new policy based Proxy Agent, which classifies the request as a scripted request or query based request, and then, detects the respective type of attack, if any in the request. It should detect both SQL injection attack as well as the Cross-Site Scripting attacks.

Abstract:

This **SQL injection attack** is one of the common means for hackers to **attack** database. Users can submit a database query code and get some data they want to know according to the results of the program. **Cross site scripting** (**XSS**) is a common attack vector that injects malicious code into a vulnerable web application.

Objectives:

- To transfer the information securely.

Hardware Requirements:

- 2 GB RAM
- 500 GB HDD
- GPU
- Working PC

Software Requirements:

- Jupyter Notebook

Introduction:

SQL injection is a technique used to exploit user data through web page inputs by injecting SQL commands as statements. Basically, these statements can be used to manipulate the application's web server by malicious users.

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

Cross Site Scripting (XSS) is a vulnerability in a web application that allows a third party to execute a script in the user's browser on behalf of the web application. Cross-site Scripting is one of the most prevalent vulnerabilities present on the web today. The exploitation of XSS against a user can lead to various consequences such as account compromise, account deletion, privilege escalation, malware infection and many more.

Depending on the context, there are ***two types*** of XSS –

1.**Reflected XSS:**
If the input has to be provided each time to execute, such XSS is called reflected.
2.Stored XSS:
When the response containing the payload is stored on the server in such a way that the script gets executed on every visit without submission of payload, then it is identified as stored XSS.

Preventing SQL Injection

- User Authentication: Validating input from the user by pre-defining length, type of input, of the input field and authenticating the user.
- Restricting access privileges of users and defining as to how much amount of data any outsider can access from the database. Basically, user should not be granted permission to access everything in the database.
- Do not use system administrator accounts.

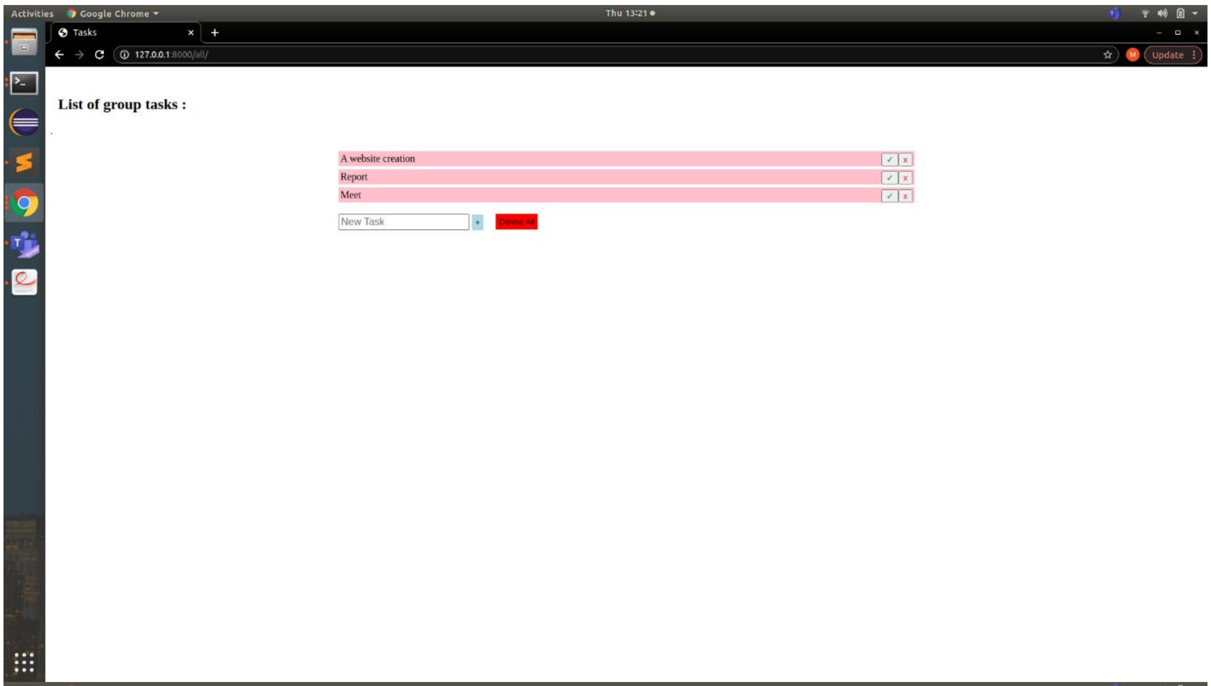
Preventing XSS

- **Escaping**: The first method you can and should use to prevent XSS vulnerabilities from appearing in your applications is by escaping user input. Escaping data means taking the data an application has received and ensuring it's secure before rendering it for the end user.

- **Validating Input:** Validating input is the process of ensuring an application is rendering the correct data and preventing malicious data from doing harm to the site, database, and users.

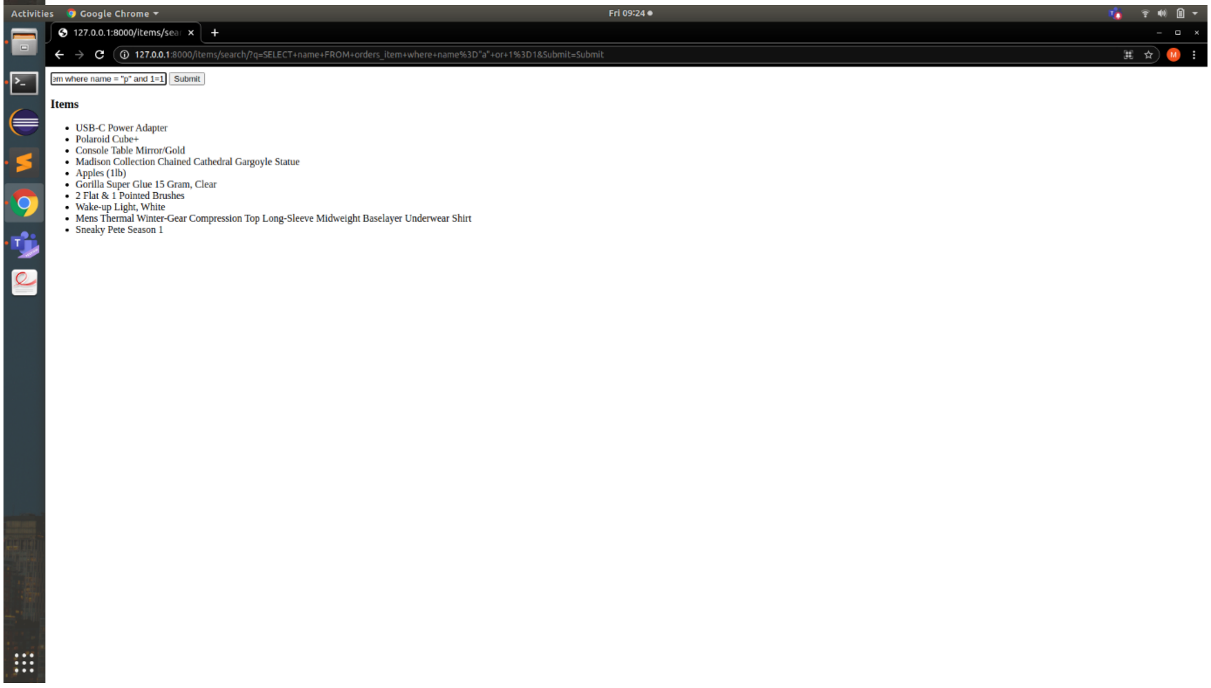
- **Sanitizing**: A third way to prevent cross-site scripting attacks is to sanitize user input. Sanitizing data is a strong defence, but should not be used alone to battle XSS attacks.

Code and output:



List of group tasks :

A website creation ✓ x
Report ✓ x
Meet ✓ x

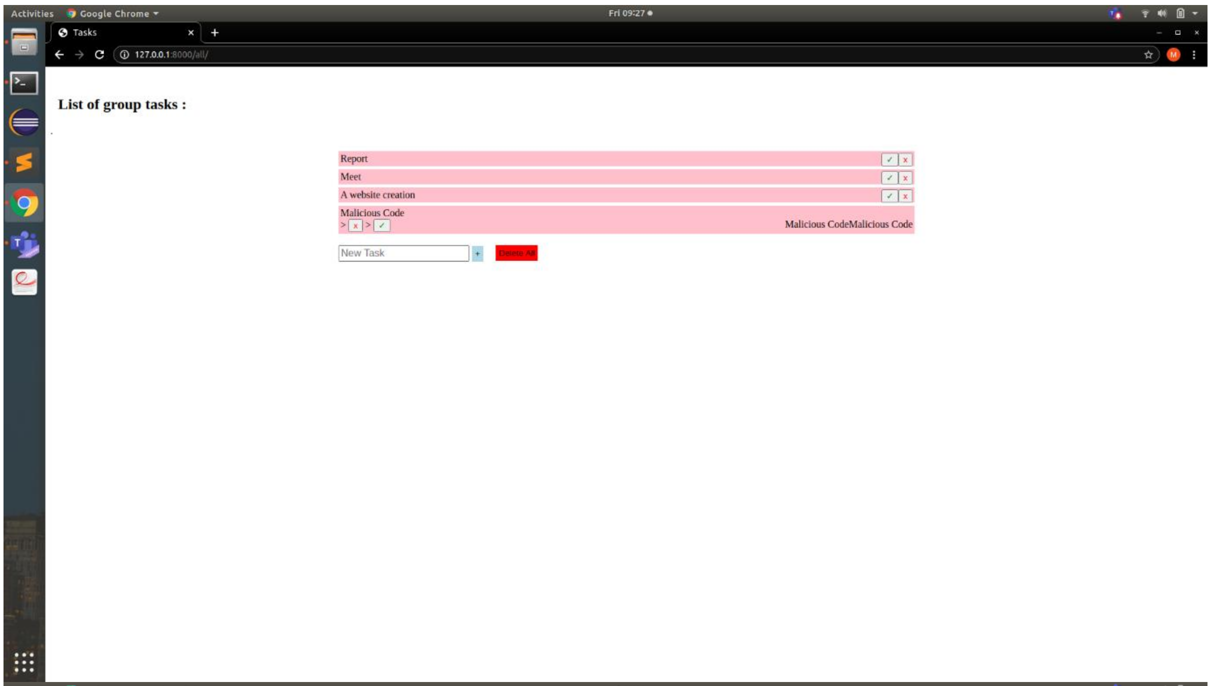


```
127.0.0.1:8000/items/search/?q=SELECT+name+FROM+orders_item+where+name%3D"a"+or+1%3D1&Submit=Submit
```

```
em where name = "p" and 1=1
```

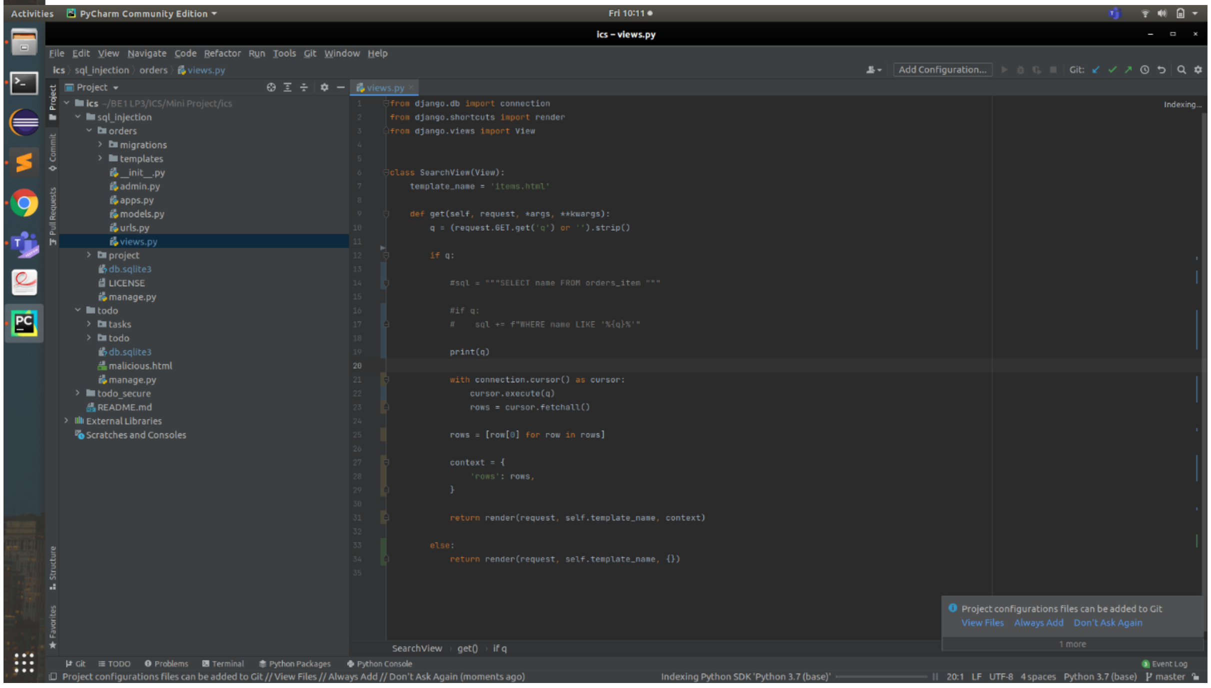
Items

- USB-C Power Adapter
- Polaroid Cube+
- Console Table Mirror/Gold
- Madison Collection Chained Cathedral Gargoyle Statue
- Apples (1lb)
- Gorilla Super Glue 15 Gram, Clear
- 2 Flat & 1 Pointed Brushes
- Wake-up Light, White
- Mens Thermal Winter-Gear Compression Top Long-Sleeve Midweight Baselayer Underwear Shirt
- Sneaky Pete Season 1



List of group tasks :

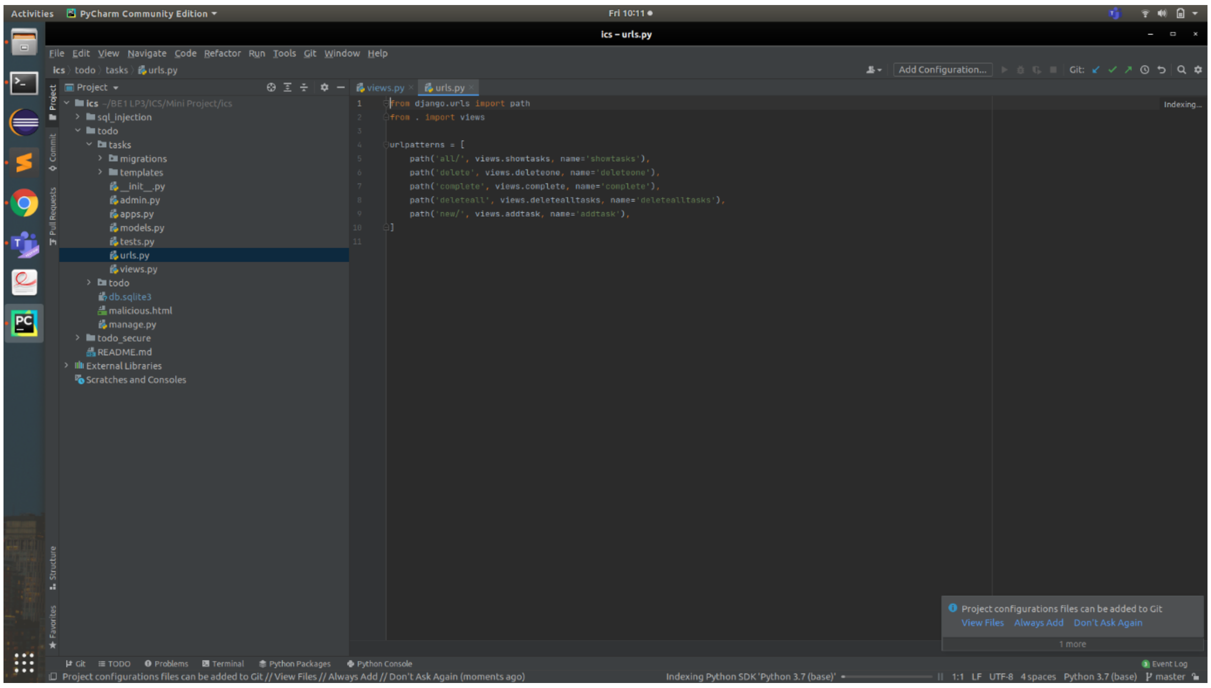
Report
Meet
A website creation
Malicious Code
Malicious CodeMalicious Code



```python
from django.db import connection
from django.shortcuts import render
from django.views import View


class SearchView(View):
    template_name = 'items.html'

    def get(self, request, *args, **kwargs):
        q = (request.GET.get('q') or '').strip()

        if q:

            #sql = """SELECT name FROM orders_item """

            #if q:
            #    sql += f"WHERE name LIKE '%{q}%'"

            print(q)

            with connection.cursor() as cursor:
                cursor.execute(q)
                rows = cursor.fetchall()

            rows = [row[0] for row in rows]

            context = {
                'rows': rows,
            }

            return render(request, self.template_name, context)

        else:
            return render(request, self.template_name, {})
```

Project configurations files can be added to Git
View Files Always Add Don't Ask Again
1 more

Project configurations files can be added to Git // View Files // Always Add // Don't Ask Again (moments ago)   Indexing Python SDK 'Python 3.7 (base)'   20:1 LF UTF-8 4 spaces Python 3.7 (base) master



```python
from django.urls import path
from . import views

urlpatterns = [
    path('all/', views.showtasks, name='showtasks'),
    path('delete', views.deleteone, name='deleteone'),
    path('complete', views.complete, name='complete'),
    path('deleteall', views.deletealltasks, name='deletealltasks'),
    path('new/', views.addtask, name='addtask'),
]
```

Project configurations files can be added to Git // View Files // Always Add // Don't Ask Again (moments ago)

Conclusion:

We successfully classified the request as a scripted request or query-based request, and then, detected the respective type of attack, if any in the request.