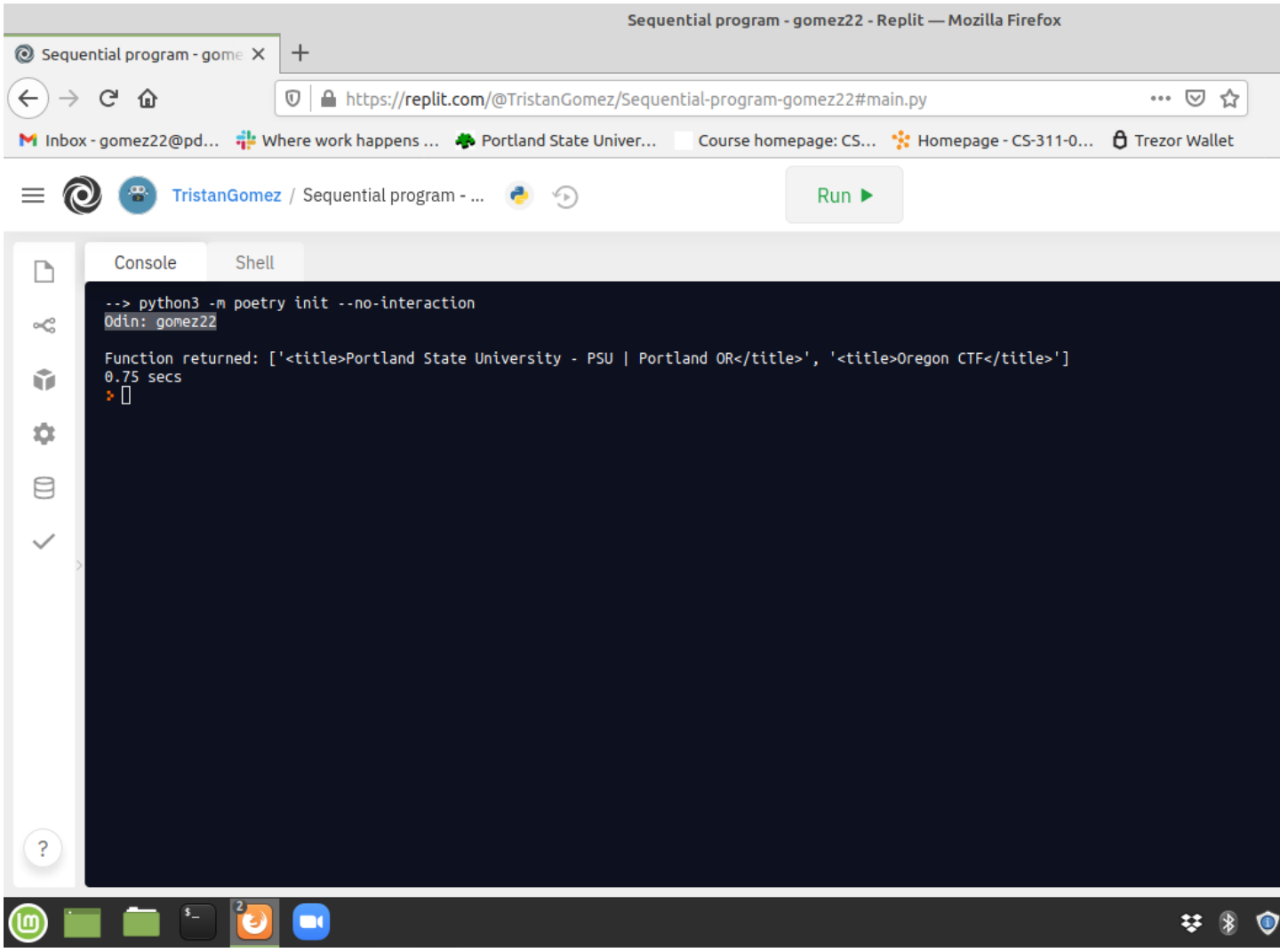
Tristan Gomez
CS 595

Table of Contents

1.2 Web Programming

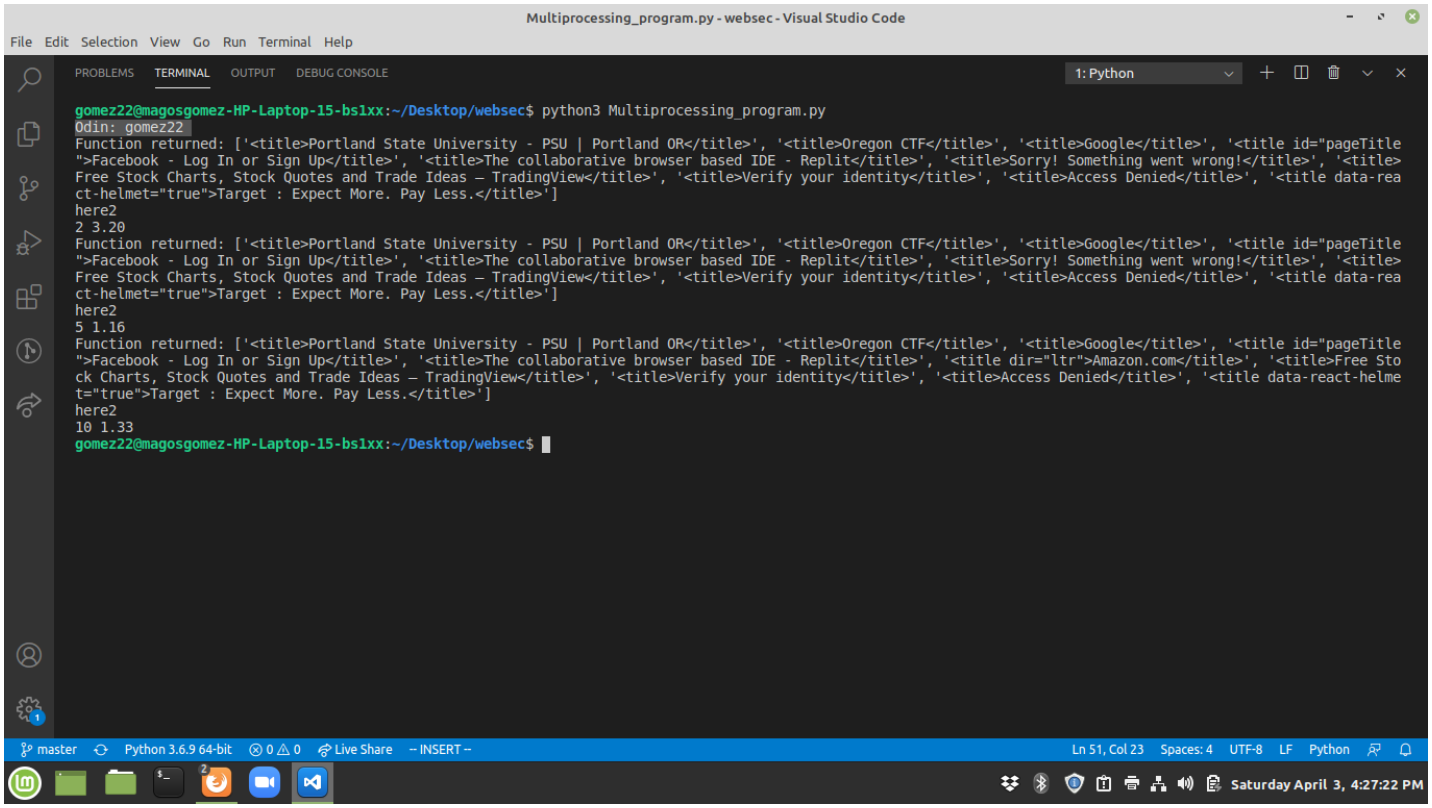
Sequential Program

-My repl account is from when I took CS161 years ago, so I cannot change my username to my Odin name. I will print my Odin name with the output of each program. As you can see, my Odin name is highlighted in the program output. Also my username/real name @TristanGomez is in the URL for the repl page.

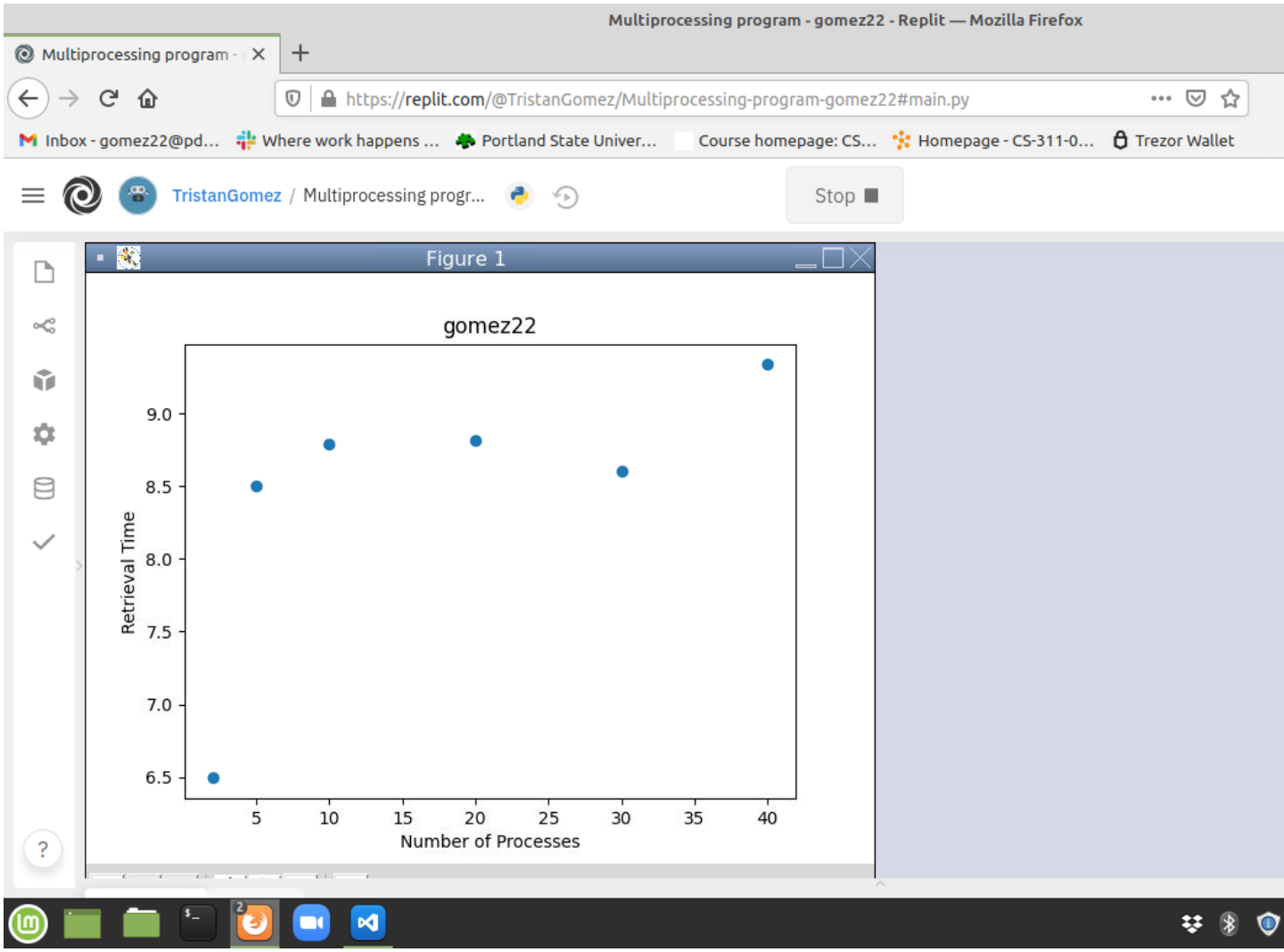


Multiprocessing program

-My Odin ID is highlighted at the top and is also present as my username on the file path.

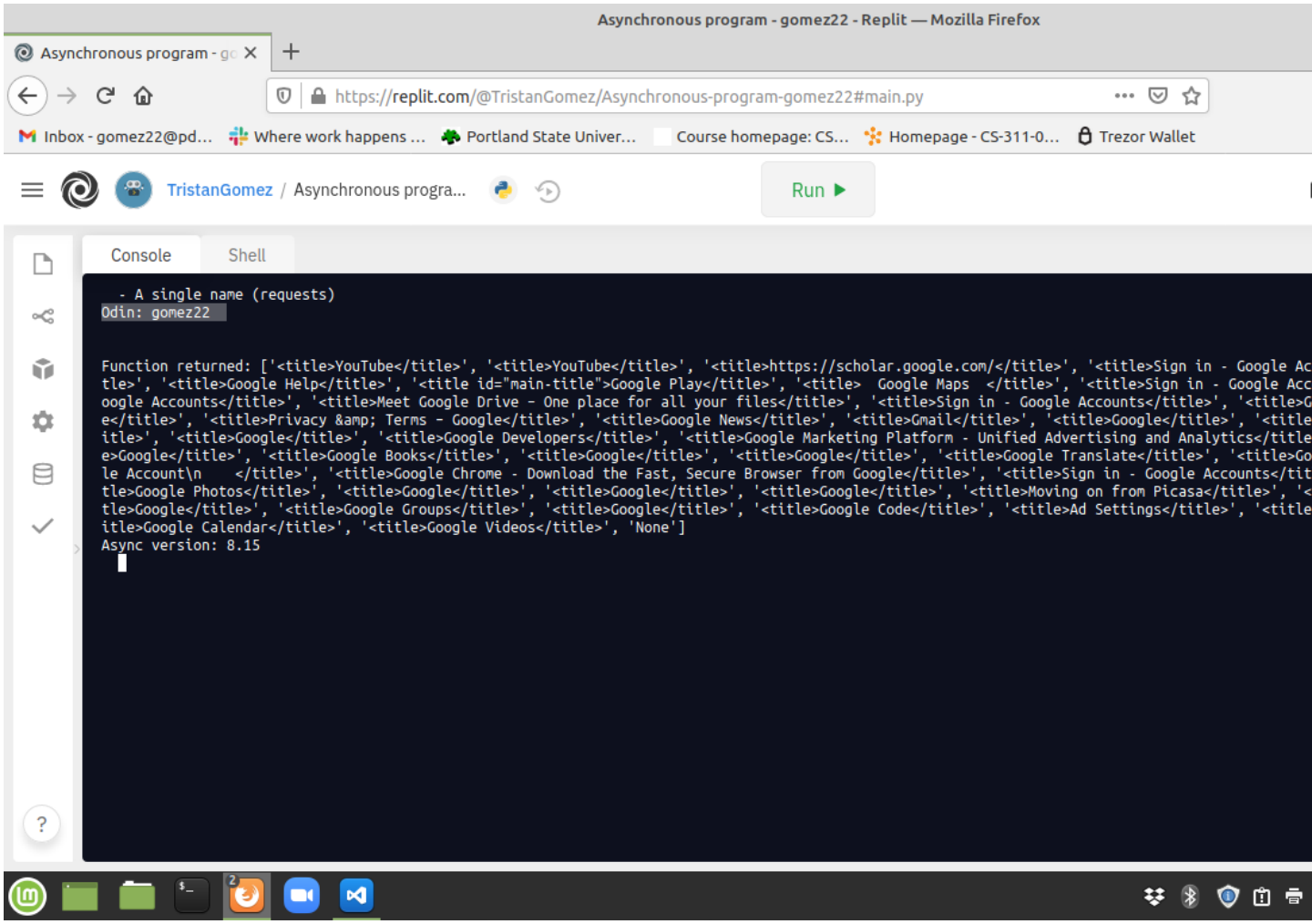

Multiprocessing_program.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
1: Python
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ python3 Multiprocessing_program.py
Odin: gomez22
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>', '<title>Google</title>', '<title id="pageTitle
">Facebook - Log In or Sign Up</title>', '<title>The collaborative browser based IDE - Replit</title>', '<title>Sorry! Something went wrong!</title>', '<title>
Free Stock Charts, Stock Quotes and Trade Ideas — TradingView</title>', '<title>Verify your identity</title>', '<title>Access Denied</title>', '<title data-rea
ct-helmet="true">Target : Expect More. Pay Less.</title>']
here2
2 3.20
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>', '<title>Google</title>', '<title id="pageTitle
">Facebook - Log In or Sign Up</title>', '<title>The collaborative browser based IDE - Replit</title>', '<title>Sorry! Something went wrong!</title>', '<title>
Free Stock Charts, Stock Quotes and Trade Ideas — TradingView</title>', '<title>Verify your identity</title>', '<title>Access Denied</title>', '<title data-rea
ct-helmet="true">Target : Expect More. Pay Less.</title>']
here2
5 1.16
Function returned: ['<title>Portland State University - PSU | Portland OR</title>', '<title>Oregon CTF</title>', '<title>Google</title>', '<title id="pageTitle
">Facebook - Log In or Sign Up</title>', '<title>The collaborative browser based IDE - Replit</title>', '<title dir="ltr">Amazon.com</title>', '<title>Free Sto
ck Charts, Stock Quotes and Trade Ideas — TradingView</title>', '<title>Verify your identity</title>', '<title>Access Denied</title>', '<title data-react-helme
t="true">Target : Expect More. Pay Less.</title>']
here2
10 1.33
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master Python 3.6.9 64-bit 0 0 Live Share -- INSERT -- Ln 51, Col 23 Spaces: 4 UTF-8 LF Python
Saturday April 3, 4:27:22 PM

matplotlib



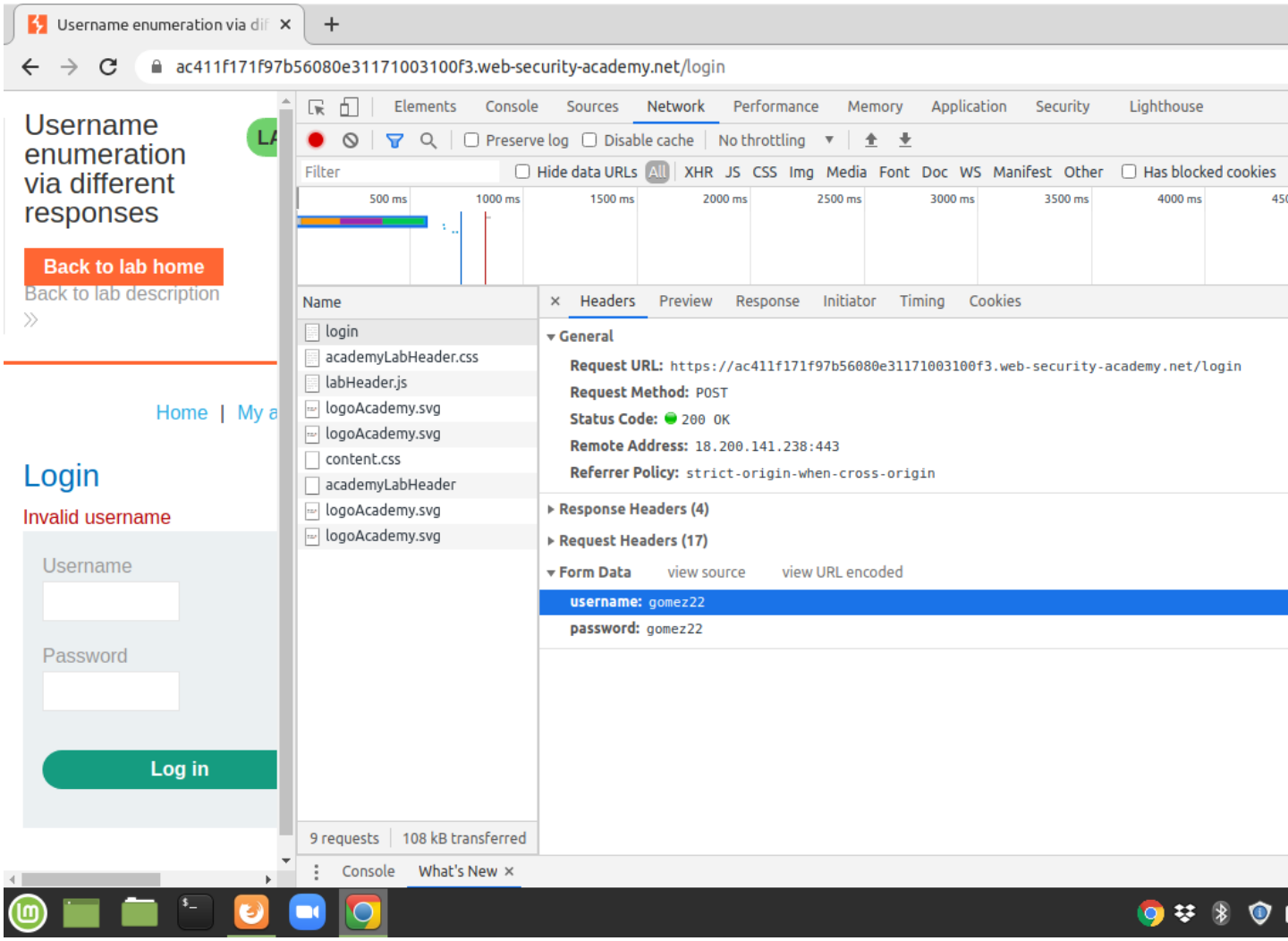
Asynchronous program

-My Odin ID is printed at the top left corner of the screenshot.



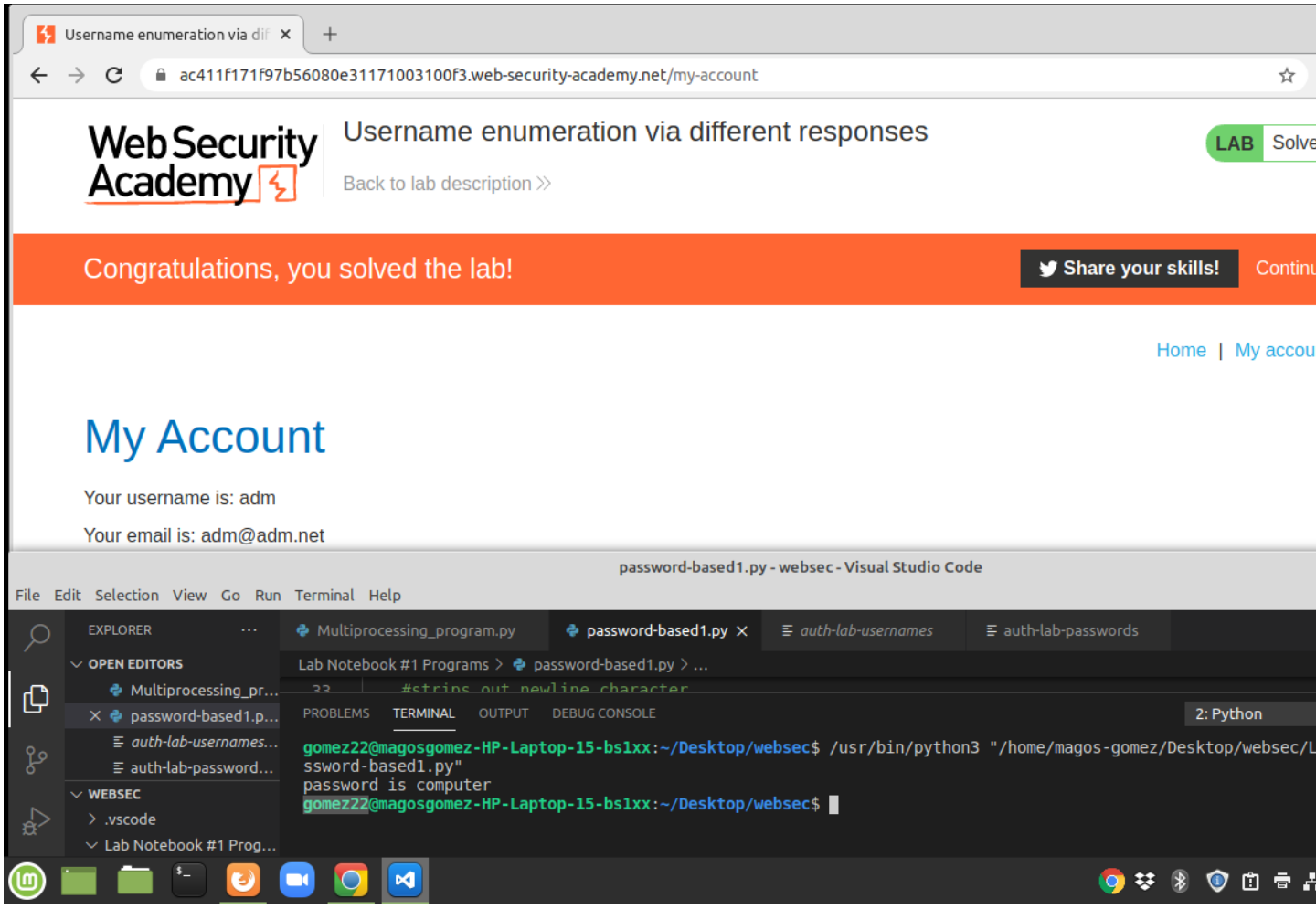
1.3 Broken Authentication

-Screenshots should include the unique URL you are given for the lab (in the URL bar), the completion message (in Orange), and your OdinID (shell window) as shown below.



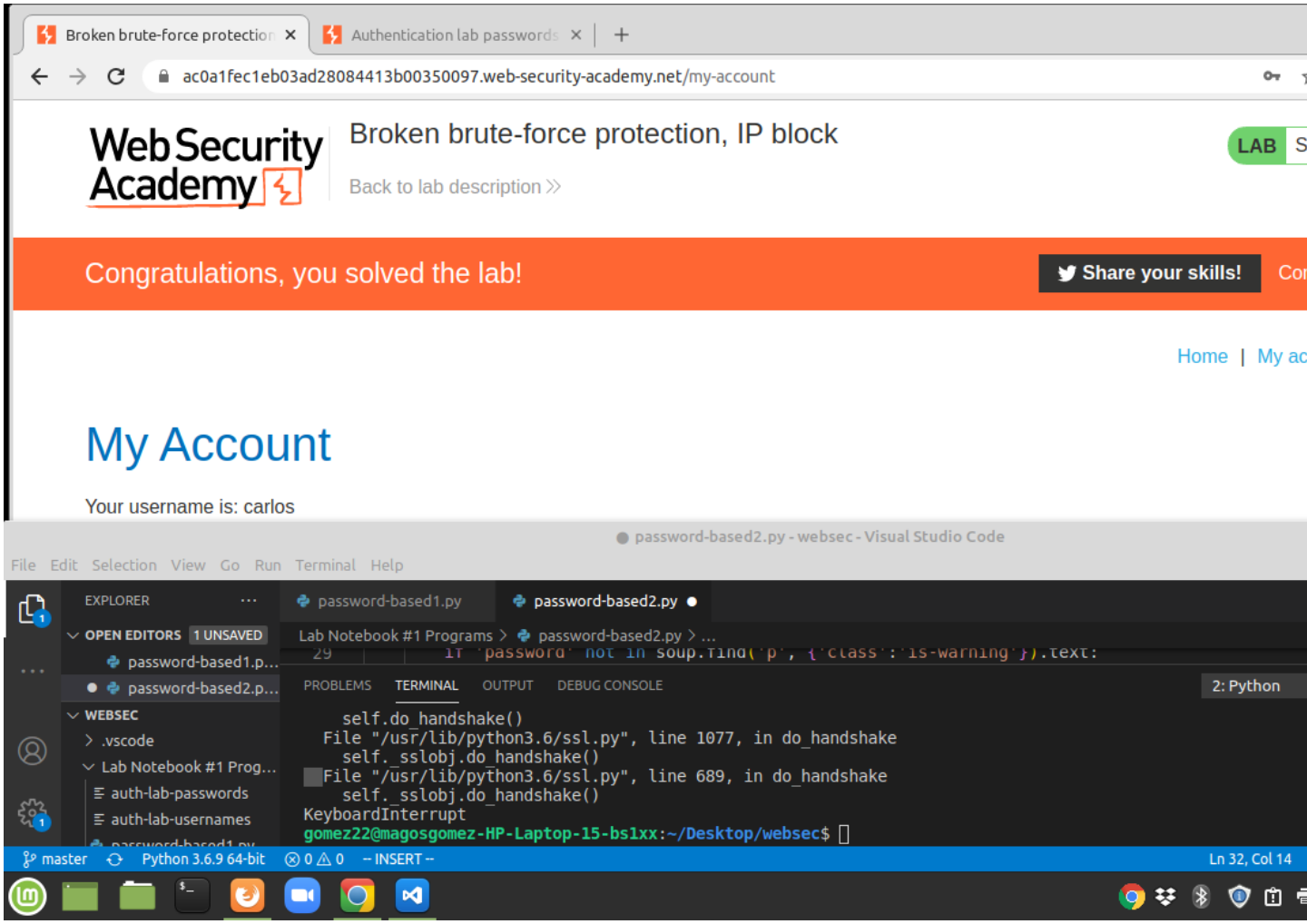
3. authentication/password-based (1)

-This vulnerability leverages the fact that the login form displays different warning/error messages for invalid passwords and usernames instead of using one generic message. This allows an attacker to iterate through a given list of usernames to find a valid username because the “invalid username” message will disappear if a correct username is entered. Then an attacker can quickly iterate through a given list of passwords along with the correct username. When the invalid password message stops appearing, you know that you have found the correct username/password combination. To remediate the vulnerability, use a more generic error message like “invalid credentials” entered. Then as an attacker, I would not know if either the username or password is correct.



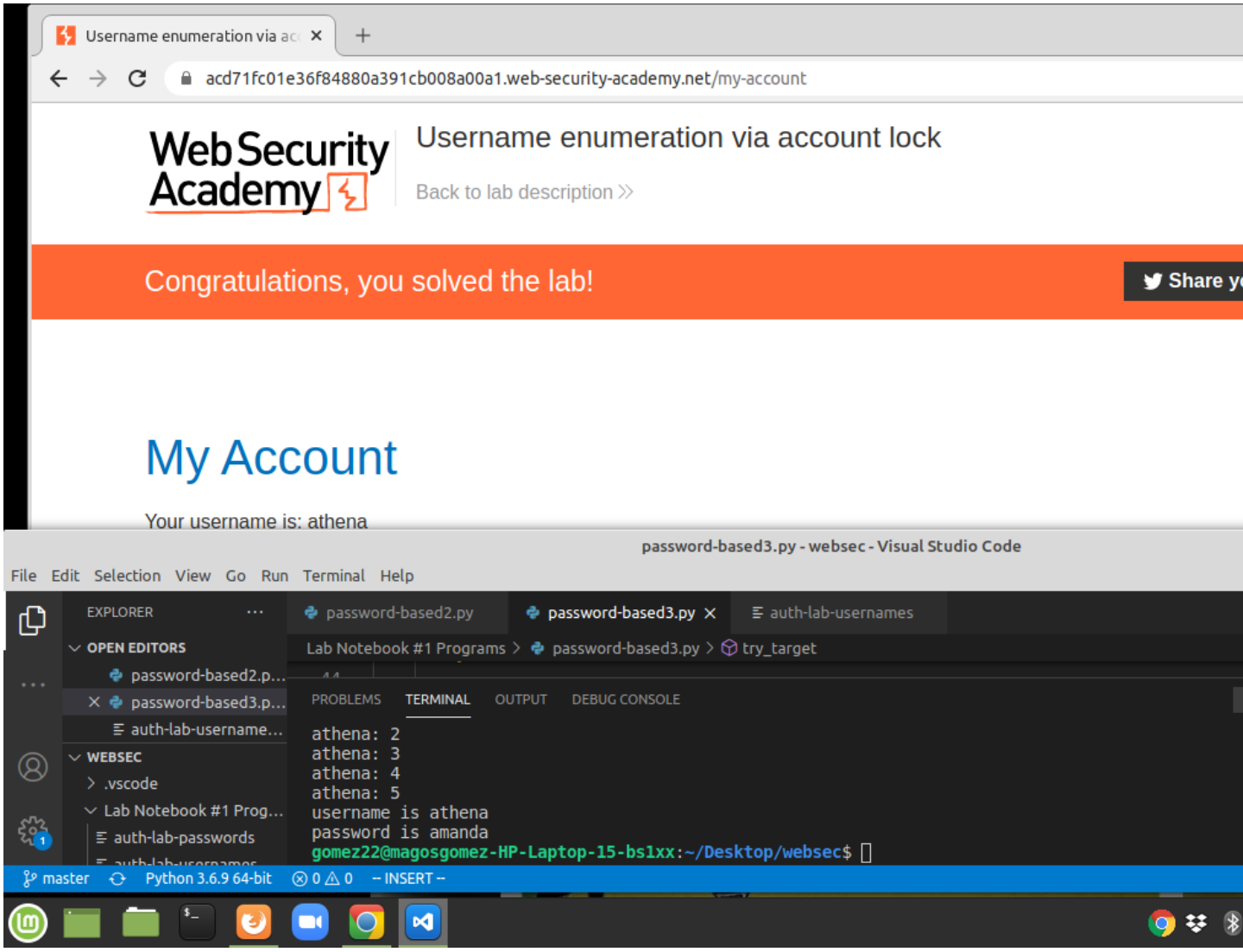
4. authentication/password-based (2)

This vulnerability leverages the idea that a valid login attempt will reset the number of remaining login attempts. Given a valid username but no password, I can then attempt to brute force passwords until I find the right one, periodically logging in with another valid username/password combination to reset my number of attempts. To remediate this vulnerability, you can link the login attempts to one set of credentials. For example, If I am trying to brute force the password for Carlos's account, then link the number of attempts to only Carlos's account. Then if I attempt to login to another account with valid credentials, it doesn't resent the number of attempts for a different account like Carlos's/



5. authentication/password-based (3)

This vulnerability leverages the idea that there is no lockout period for entering too many invalid usernames. Also, it leverages that the error message for logins can change after a certain number of tries. Specifically, if the username is incorrect then “Incorrect username or password” appears, but if the username is valid then on the 5th try, a new message appears “You have made too many incorrect login attempts”. I can then brute force usernames until I get this new message at which point I have just found a valid username. Then I can brute force passwords. If I get “You have made too many incorrect login attempts” then I know that the password is incorrect. If nothing is returned then I have a correct password. To remediate this vulnerability, they should stick with “incorrect username or password” on all attempts because then I wouldn’t be able to narrow down the correct username/password combination.

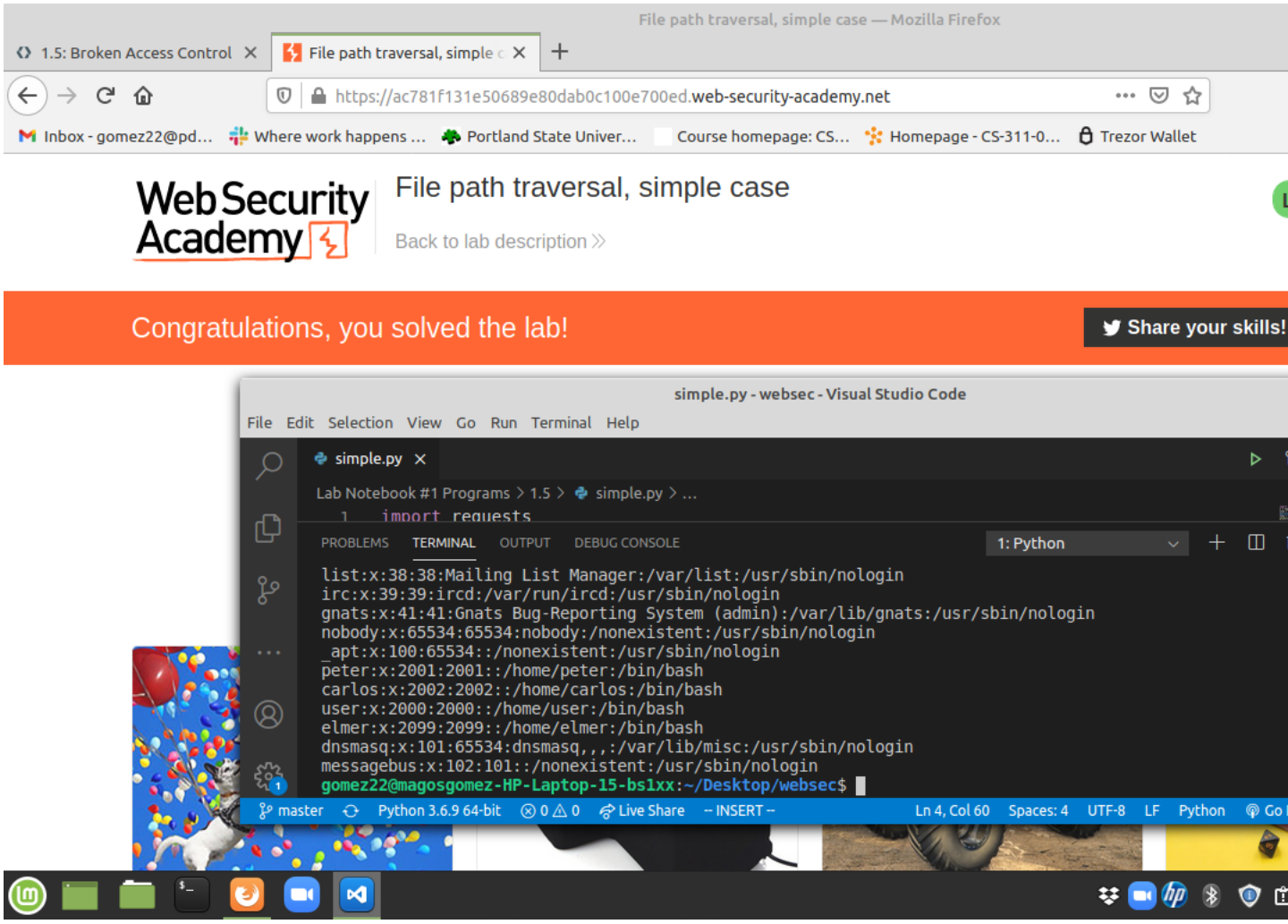


1.5 Broken Access Control

1. file-path-traversal (1)

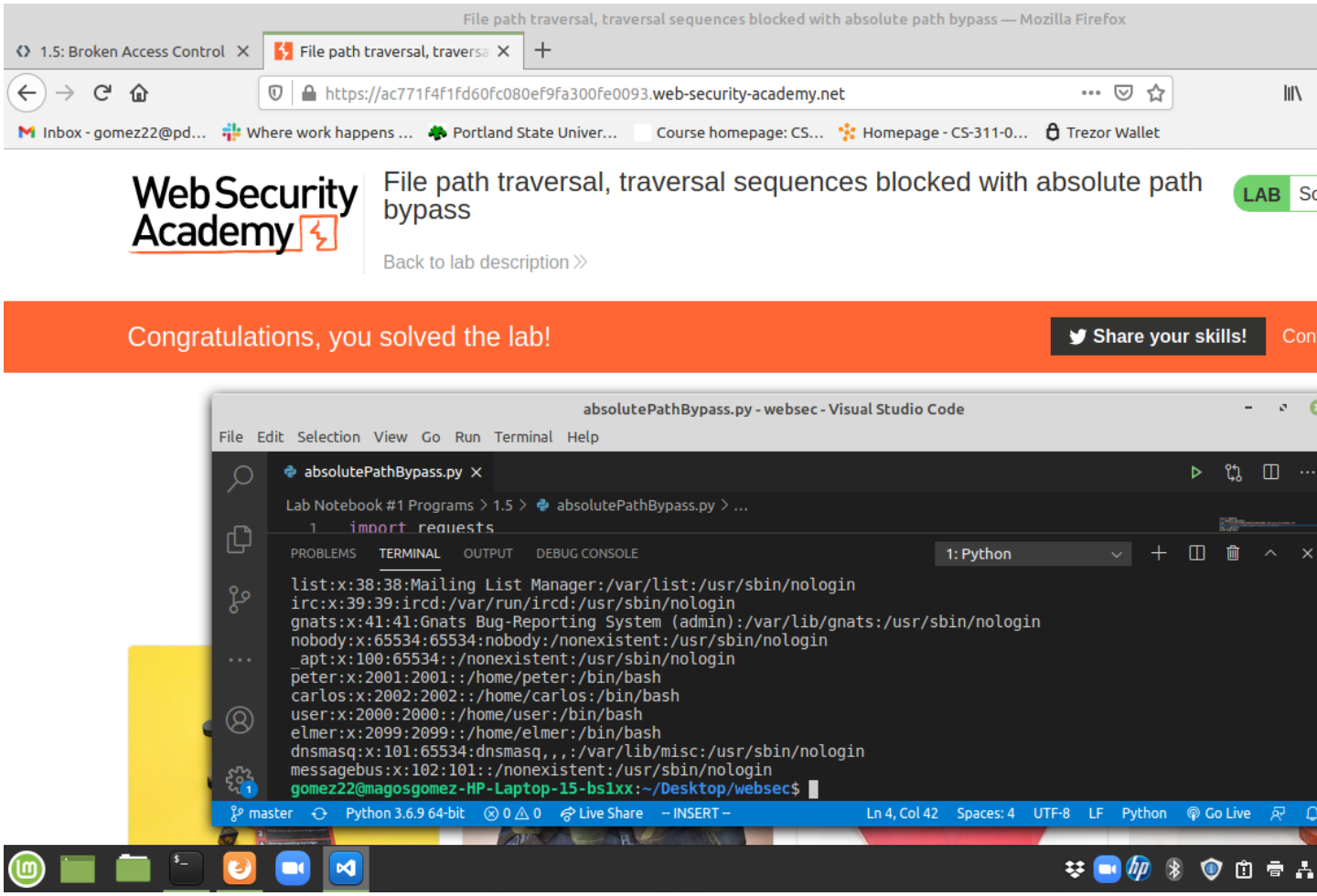
Simple:

This vulnerability leverages the “bug” that if a URL uses input that is used by the site to navigate the file system then as an attacker, I can inject file paths into the url to hopefully get access to data on the server. What makes this attacker even easier to perform is that relative file paths are not filtered in anyway. To remediate this, one could potentially not use URL input to navigate the filesystem but rather, use some other scheme for retrieving content.



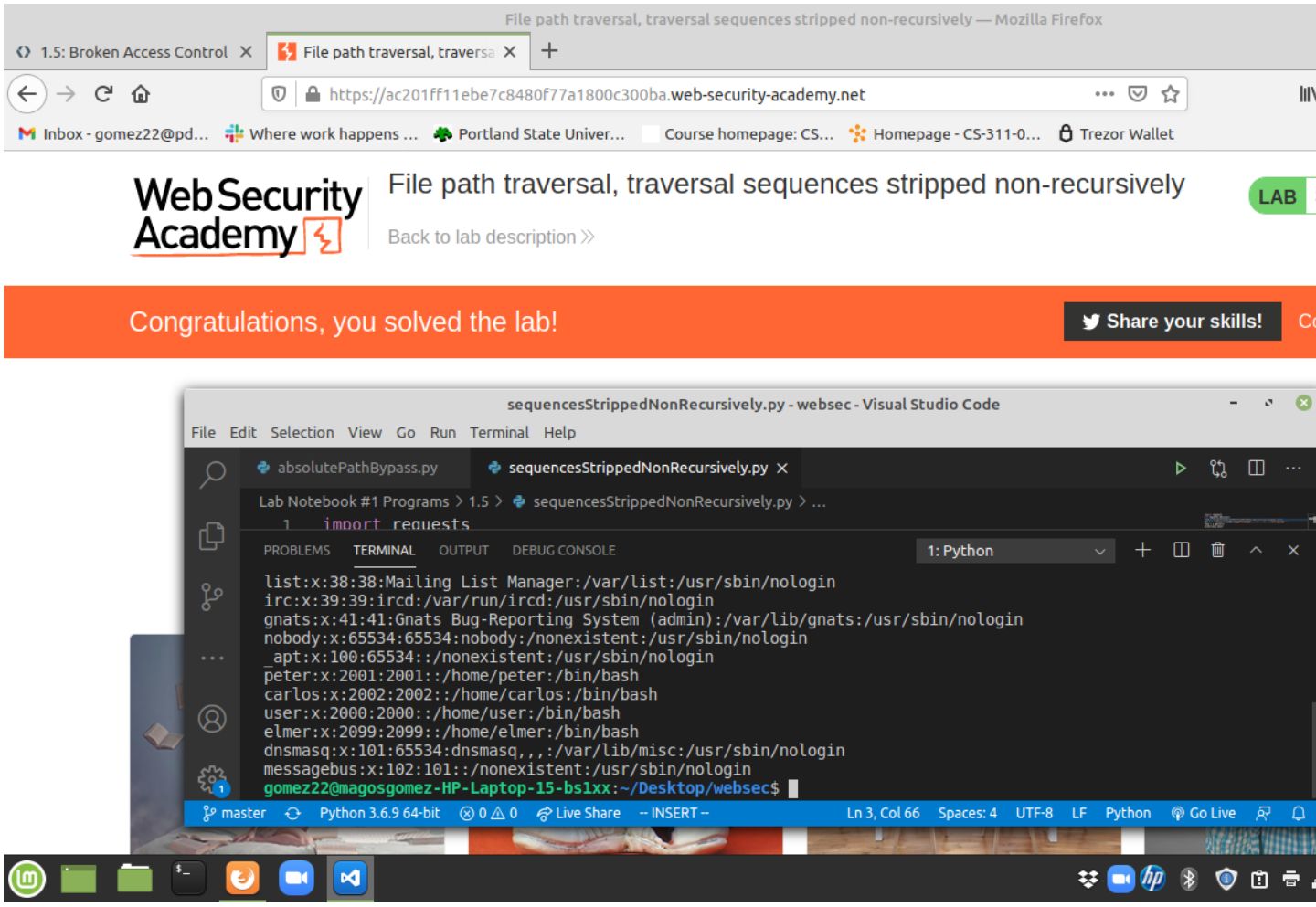
Absolute path bypass

This vulnerability leverages the idea that I can insert file paths into the url to allow me to directly navigate a server's file system to extract data. This "challenge" filters out relative path traversals, but absolute path traversals are still allowed. As an attacker, I can guess potential absolute paths since I am familiar with the Unix file system scheme and I can narrow down file paths until I find what I am looking for. To remediate this issue, one should use a different scheme to retrieve content rather than using file paths.



Sequences stripped non-recursively

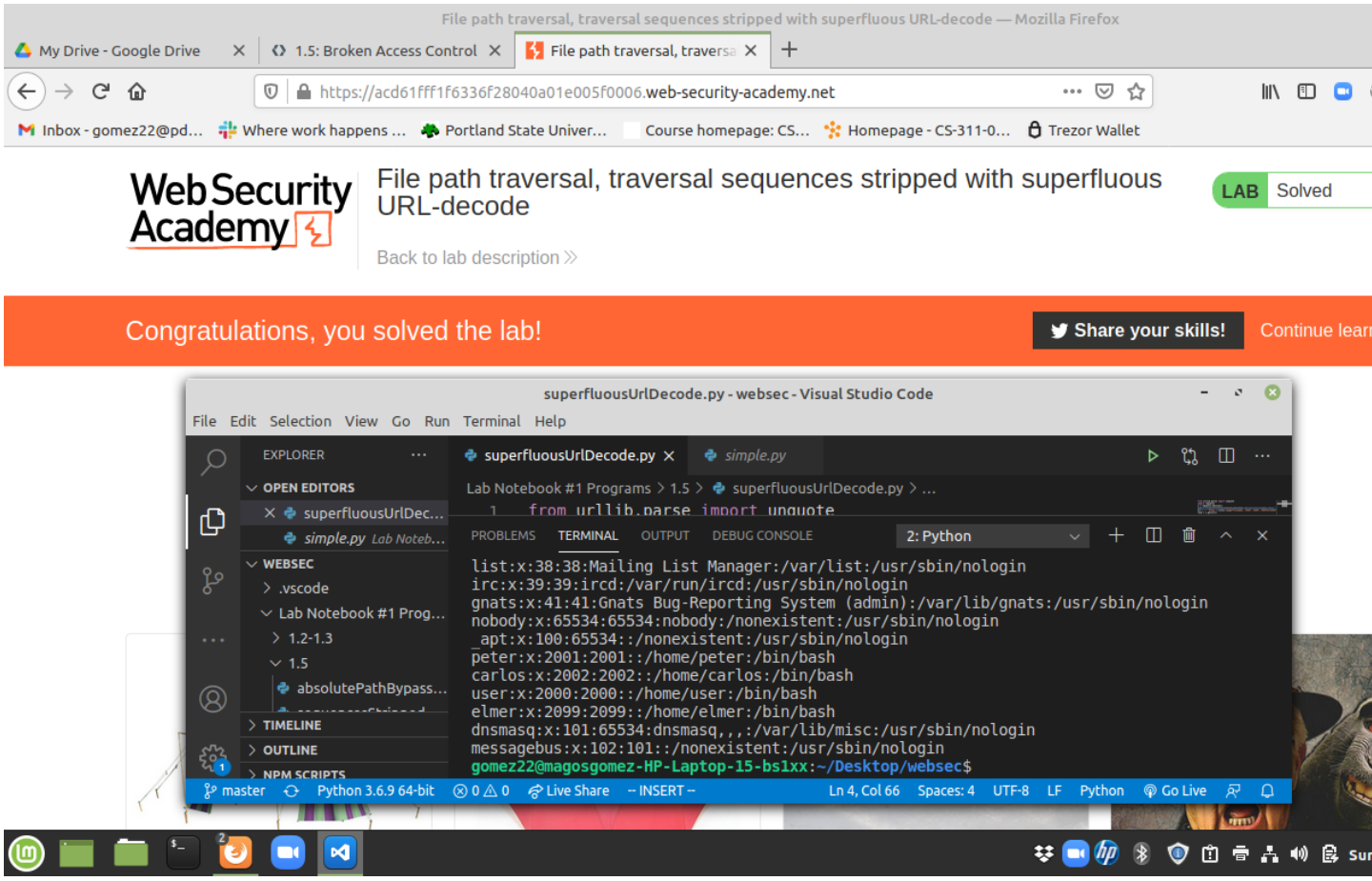
This vulnerability leverages the idea that some URLs take input which is then used to navigate a server's file system. As an attacker, I can insert an unintended file path and extract information that I am not supposed to have. This particular level attempts to filter out '../' in hopes of preventing navigation of the server's filesystem by preventing relative path traversal. However, this filter doesn't prevent an attacker from entering '..../' as part of a relative file path. The filter only removes half of the '.' and '/' characters resulting in a valid relative file path of "../". Chaining these together allows an attacker to successfully bypass the filter and get into the file system. Again, to remediate this vulnerability, the victim needs to use some other scheme to retrieve web content instead of allowing file path navigation through a URL.



2. file-path-traversal (2)

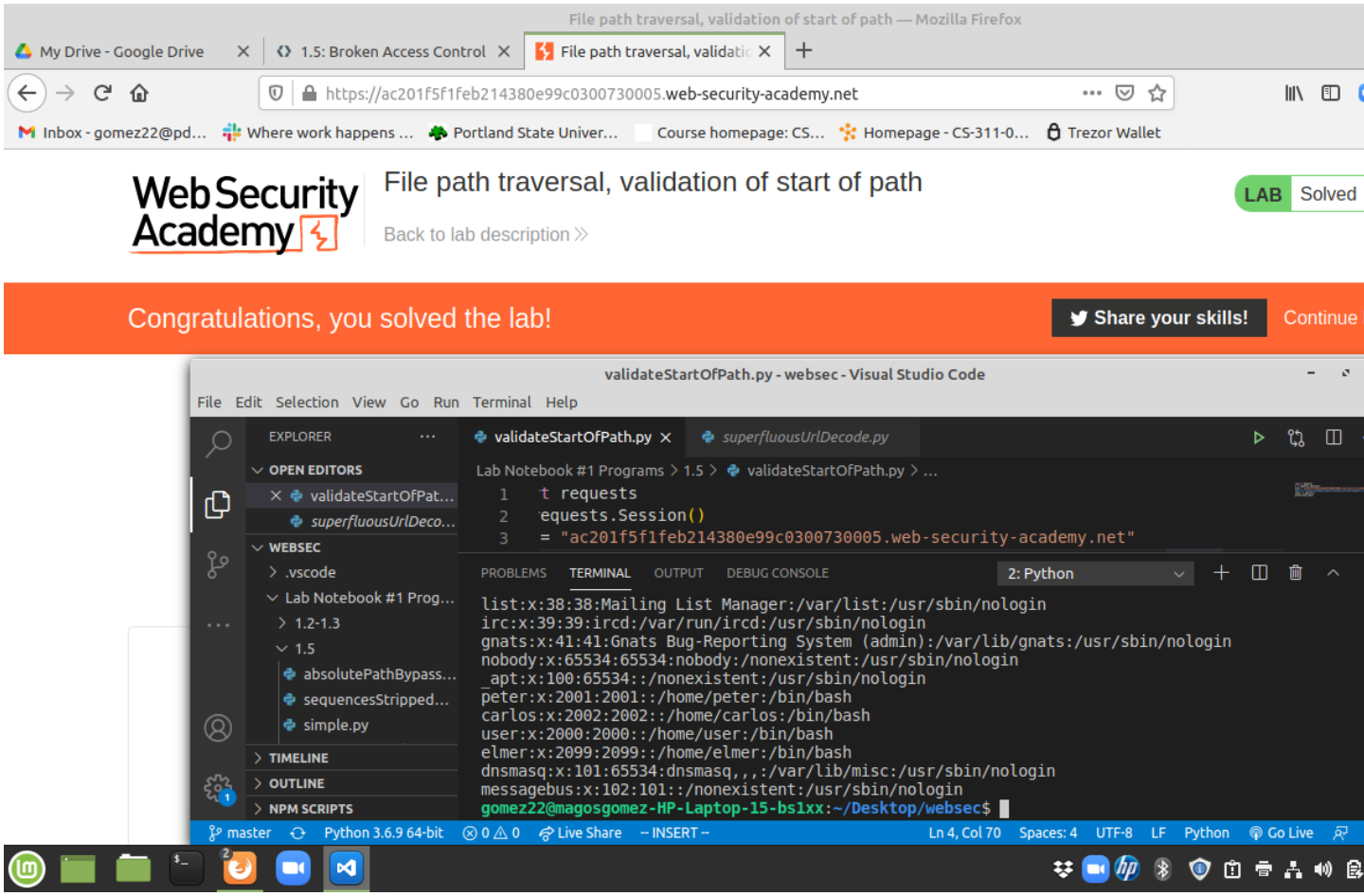
Superfluous url decode

This vulnerability leverages confusion between front end frameworks and back end servers as to which party is responsible for decoding URLs. As an attacker I can insert special characters to take advantage of this vulnerability to allow me to traverse the server's file system. Remediation for this vulnerability would be to fully shift the burden of URL decoding onto one party and focus on hardening that party's ability to decode URLs and handle special cases.



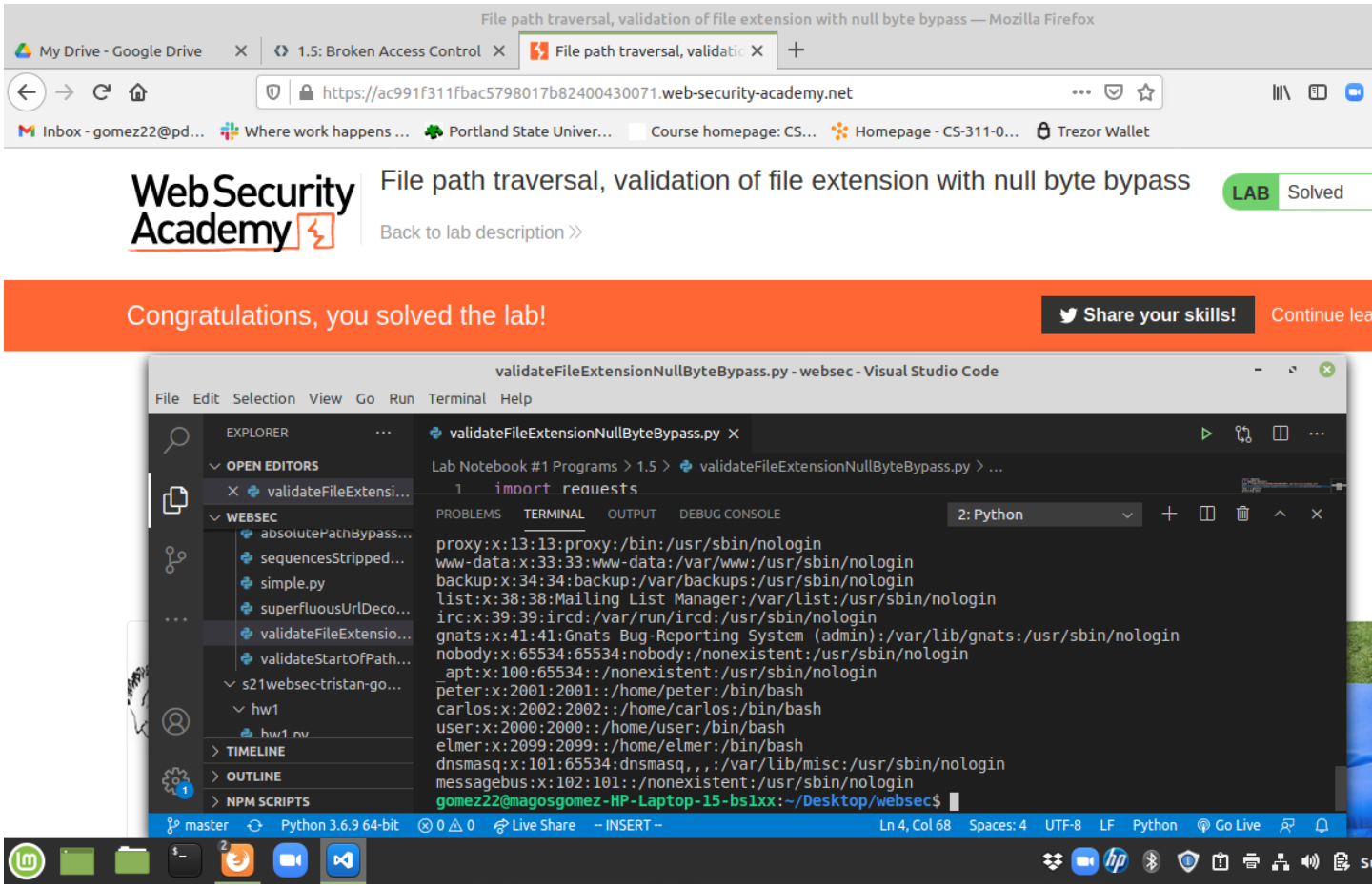
Validate start of path

This vulnerability attempts to limit traversal attacks by forcing a filename to have a particular starting pattern. As an attacker, I can use this starting pattern but then append a relative file path to the end of the pattern, allowing me to bypass it and perform the traversal attack. To remediate this, It would be best if a different scheme for retrieving content was introduced that didn't use file paths. If that is not possible then hardening the URL processing method to account for possible malicious inputs would be the next best option.



Validate file extension null byte bypass

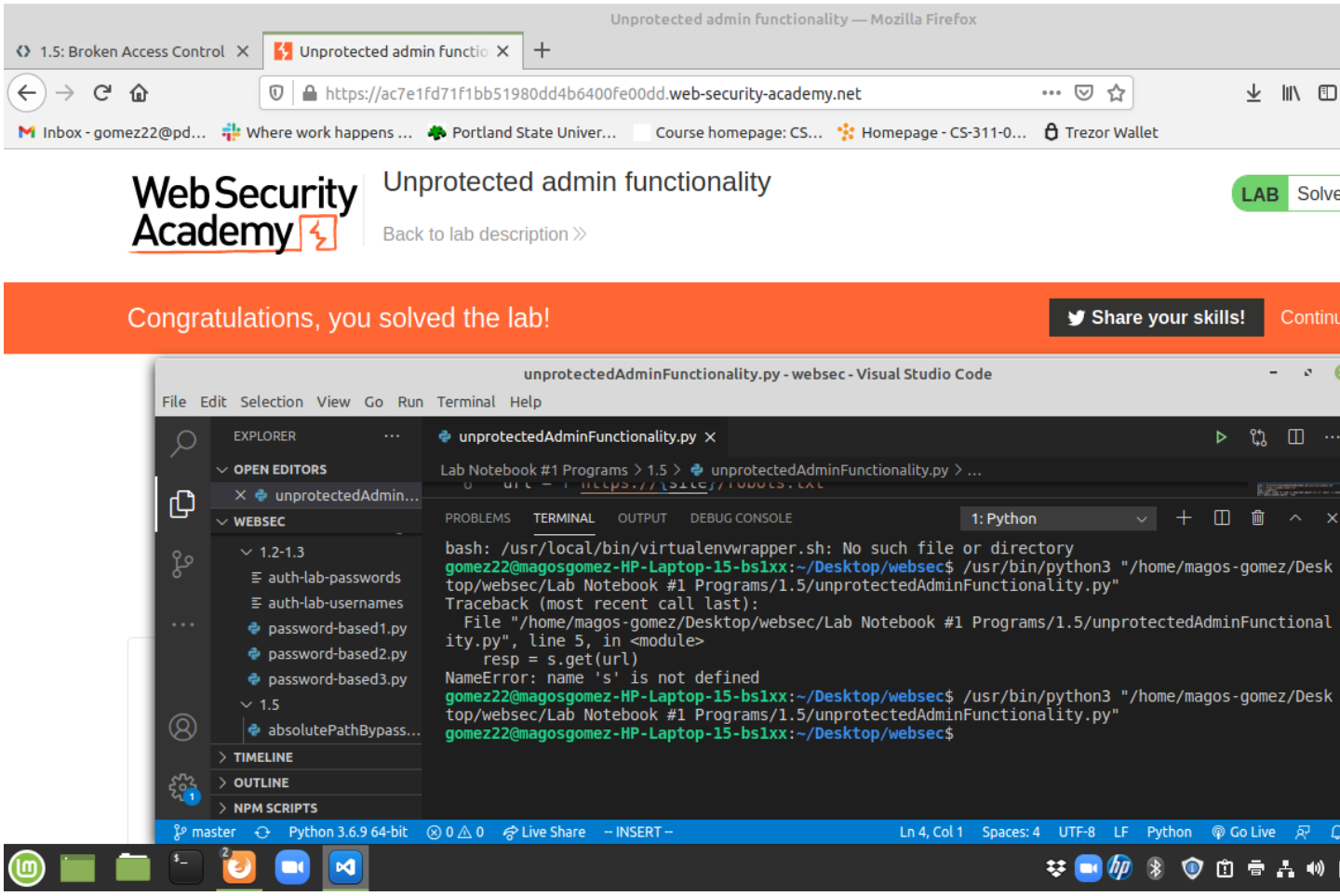
The defender in this exercise is attempting to thwart my traversal attacks by appending a file extension to all content retrieved. Since what I am trying to recover is not a .png then I should be prevented from accessing the password directory. However, I can insert a NULL character “%00” at the end of the url. The .png extension will then be appended to the string after the NULL character. When this string is processed, the processing will end at the NULL character and will not read the .png extension, allowing me to still perform traversal attacks. Possible remediation would be to not process NULL characters and reject requests containing them.



3. access-control (1)

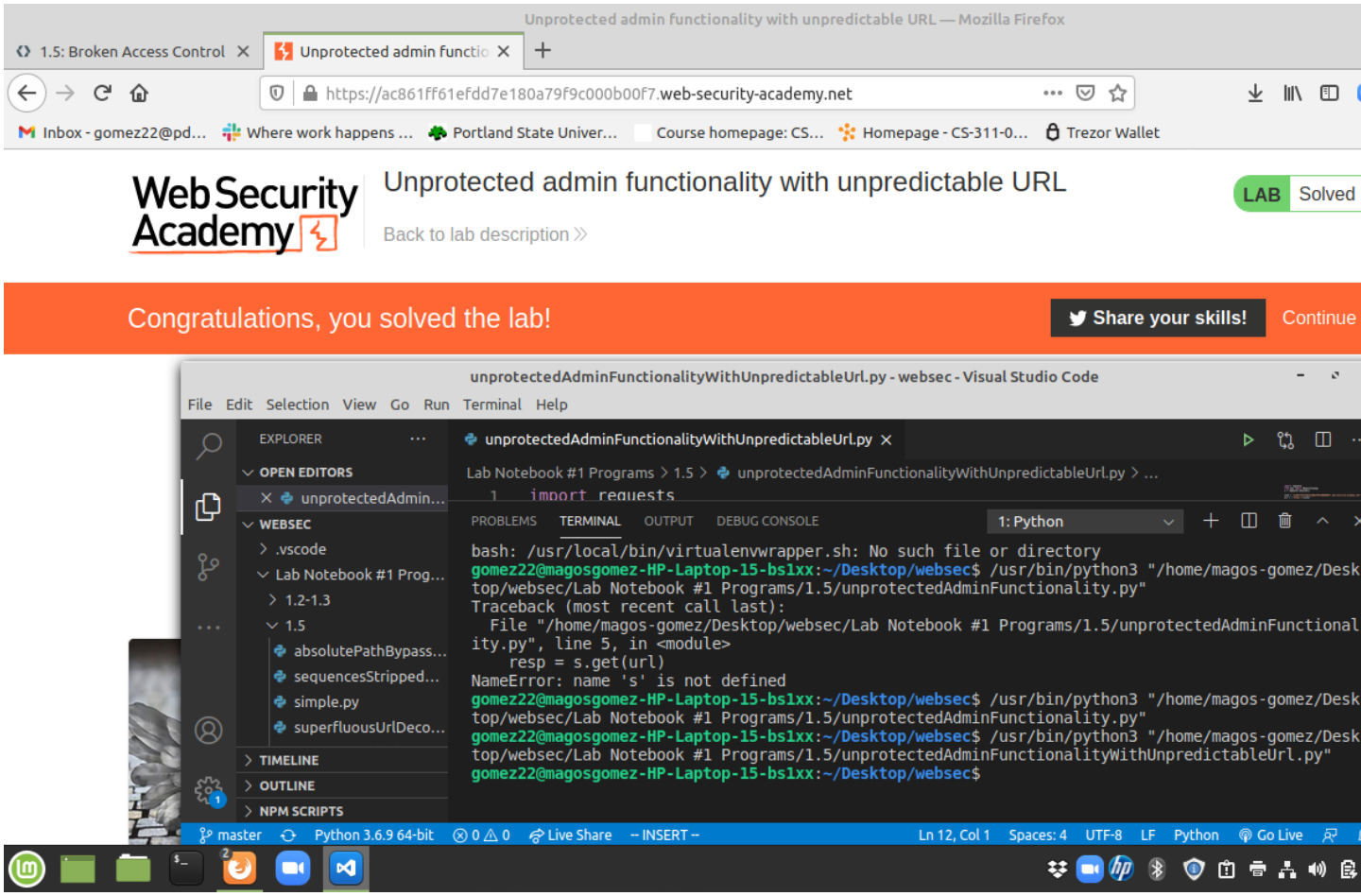
Unprotected admin functionality

This exercise has a vulnerability where I can examine a .txt file from the site which contains a URI to an administrator panel. From there I can append the URI to the site's URL to get admin access and delete carlos' account. To remediate this attack, not leaving privileged URIs around in plain text would be a good start. Then protecting those paths with some sort of authentication upon access would also be best.



Unprotected admin functionality with unpredictable url

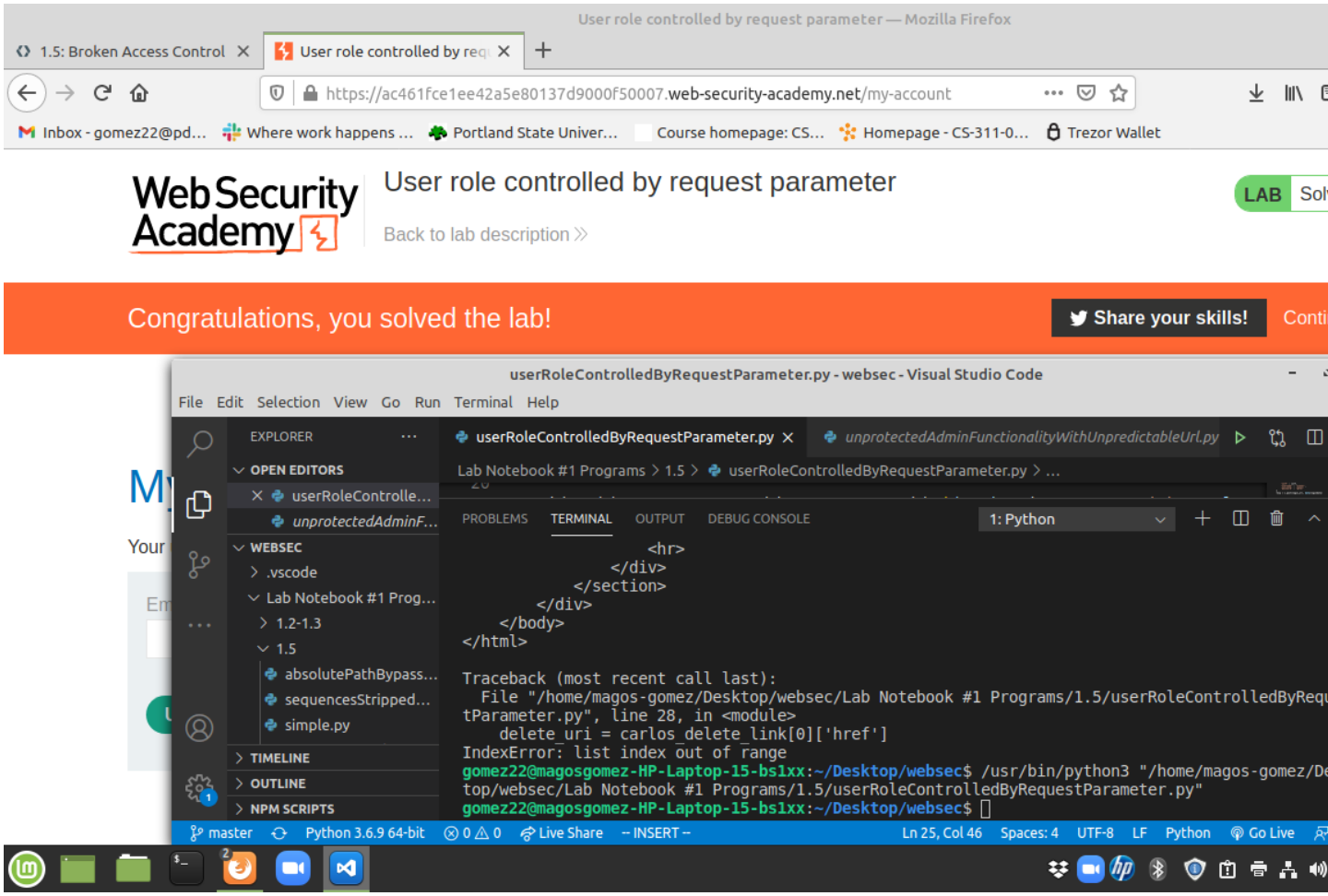
This level attempts to hide the administrator URI inside of the site's page source code. By parsing the source code, I can extract the hidden URI, append it to the site's URL and then get access to admin functionality. To remediate this vulnerability, not hiding plain text privileged URIs in the source code of a page is a good start. Then having some sort of authentication upon trying to access a privileged URL would be best.



4. access-control (2)

User role controlled by request parameter

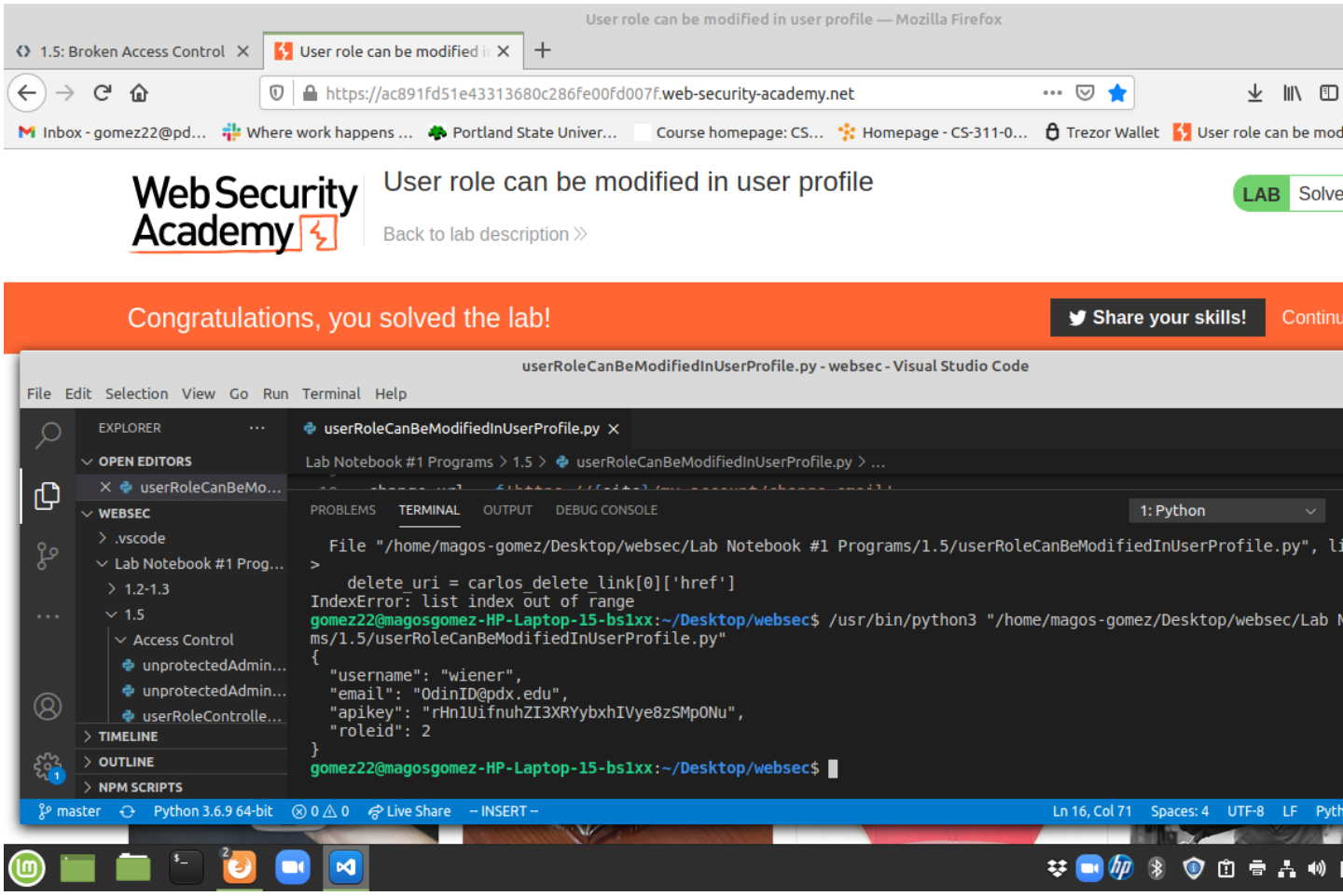
The vulnerability in this exercise is that the website trusts client input allowing me as an attacker to use malicious input to exploit the website. I am able to login with a valid account and check the cookies associated with that login request. In the cookies, I can see a field “Admin” with a value of “false”. Since the site trusts my input, I can alter my cookie to have a value of “true”, giving me privileged access to the site. This allows me to delete Carlos’ account and finish the exercise. To remediate this vulnerability, the site should perform backend input validation to detect if the client is attempting to escalate their privileges.



5. access-control (3)

User role can be modified in user profile

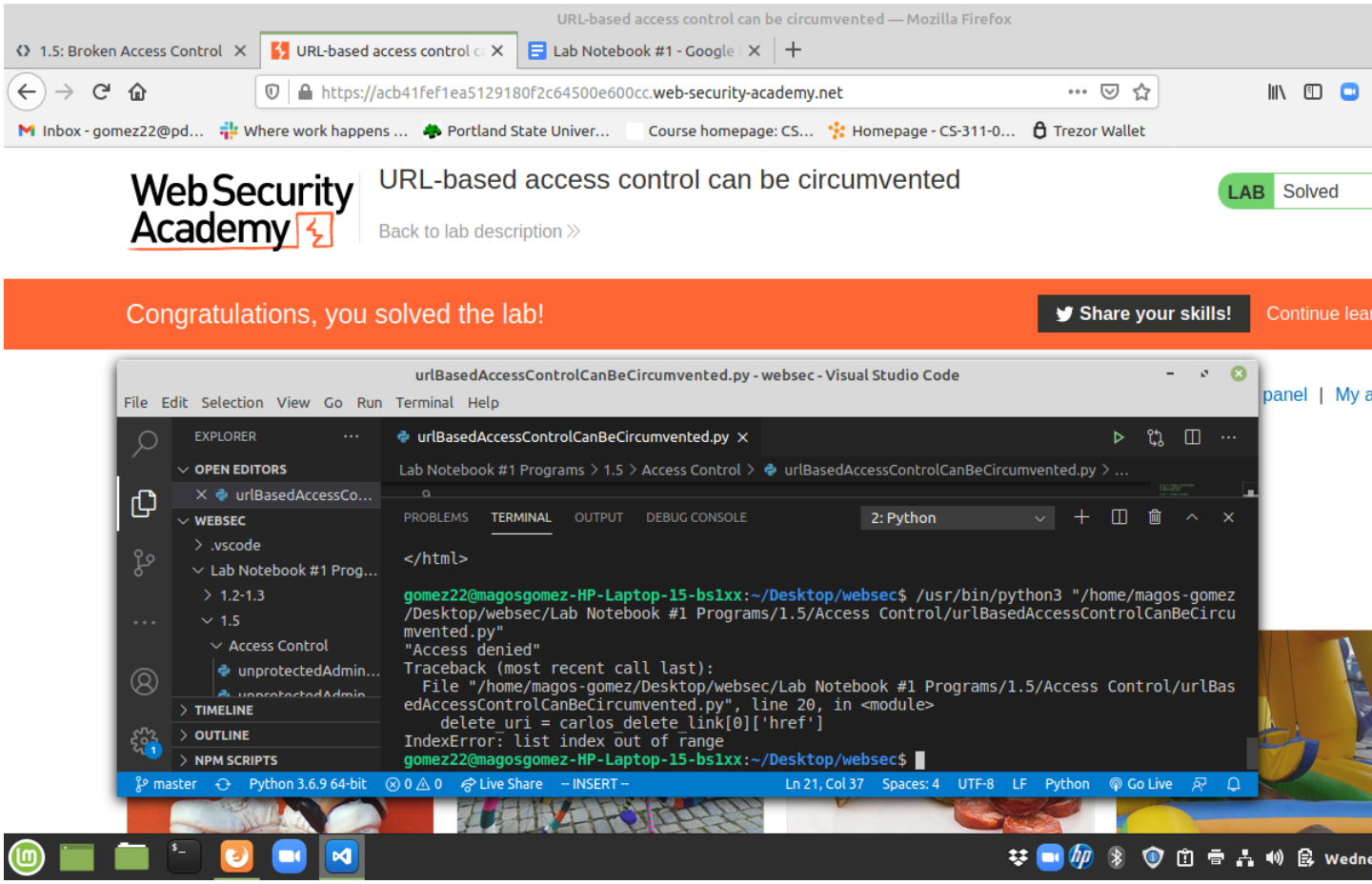
The vulnerability in this exercise is one where I can attach a JSON payload to my request and the site trusts the payload without verifying it. This lets me change my ‘roleid’ field to a ‘roleid’ with administrator privileges, allowing me to finish the level. To remediate this vulnerability, the site shouldn’t trust JSON payloads sent by the client. There should be some backend verification of the JSON payload.



6. access-control (4)

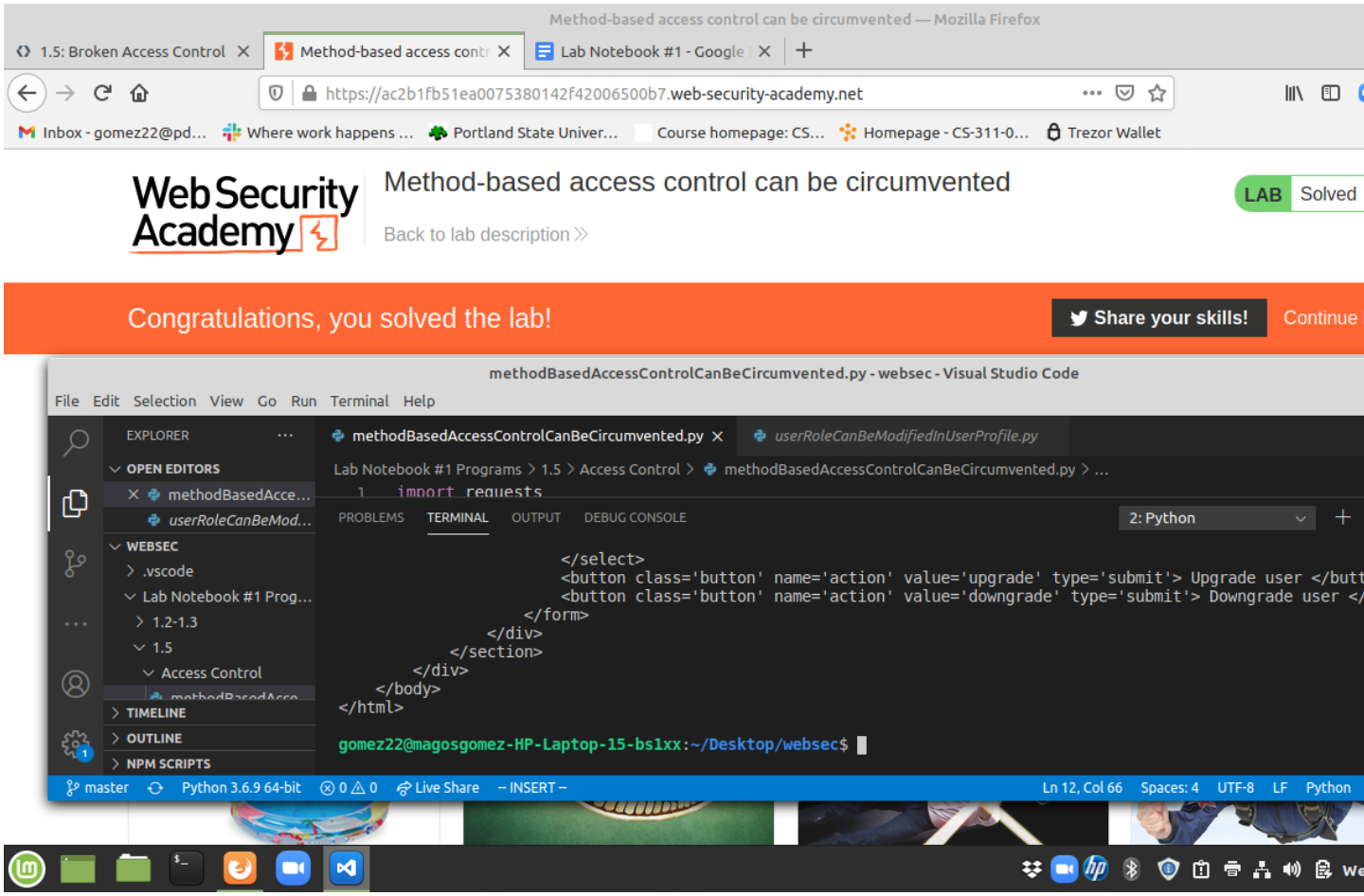
URL based access control can be circumvented

This vulnerability is where the front-end blocks invalid/unauthorized requests for web content, but it doesn't check the headers. As an attacker I can use the 'X-Original-URL' header to specify that I desire the "/admin" page. My non-malicious request is accepted by the filter and passes through. Then, the header I used allows me to get access to the privileged resources. To remediate this vulnerability, checking headers on the back-end would be best.



Method based access control can be circumvented

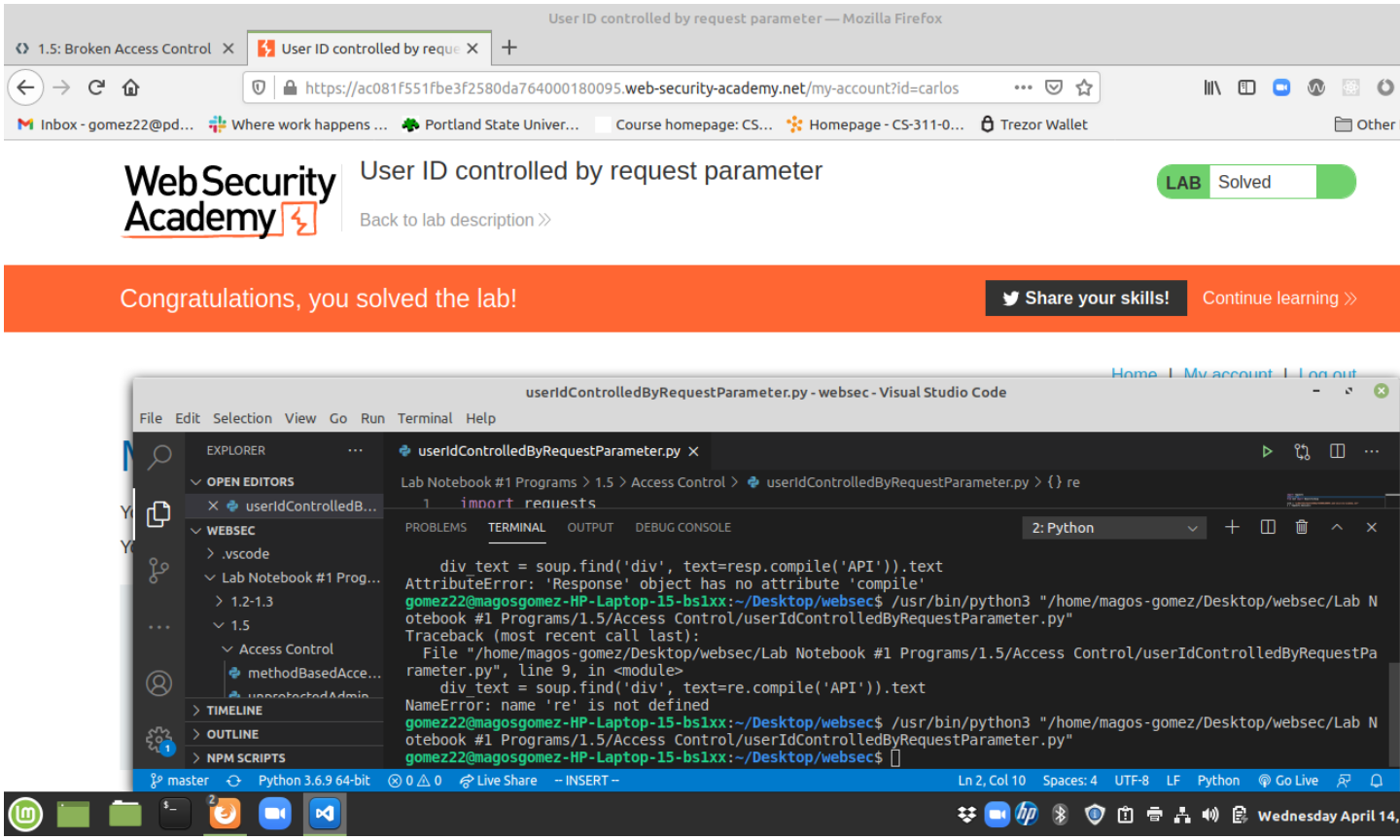
This vulnerability restricts non-authorized users from accessing the administrator panel. Since I am not an administrator, I cannot POST to the /admin-roles form to escalate privileges. However, I can use a GET request to submit the vulnerable form by encoding the parameters into the url. For example “<site url>/admin-roles?username=wiener&action=upgrade”. This let me submit the form using my username and escalate my privileges to admin level. To remediate this vulnerability, one should prevent critical forms from being submitted using GET requests.



7. access-control (5)

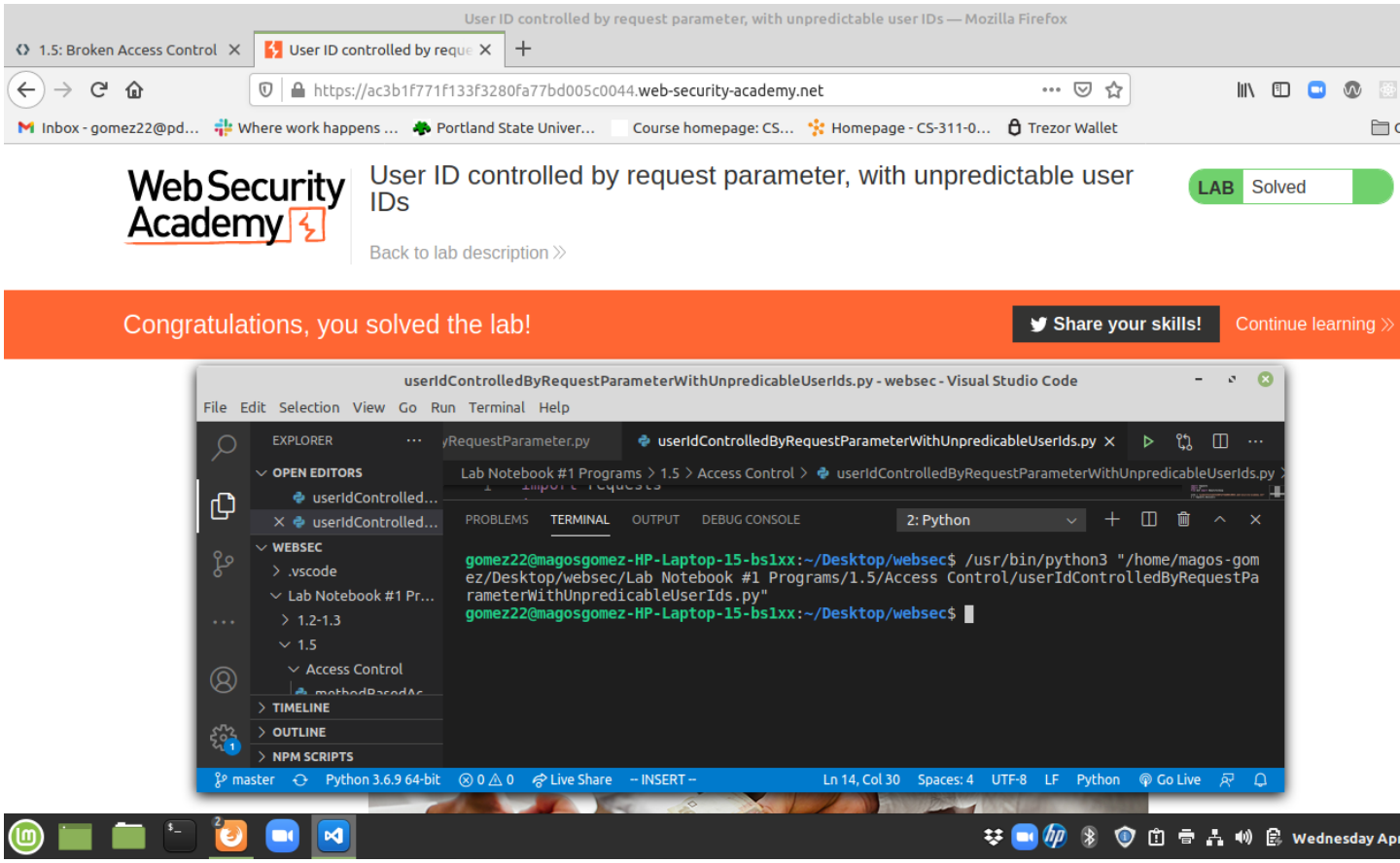
User id controlled by request parameter

This level is vulnerable because the user Id is encoded in the URL. As an attacker I can change the user ID to another user's ID and view their private information. To remediate this vulnerability, the site should either not encode user IDs in the URL or prevent users from changing the ID in the URL.



User id controlled by request parameter with unpredictable user ids

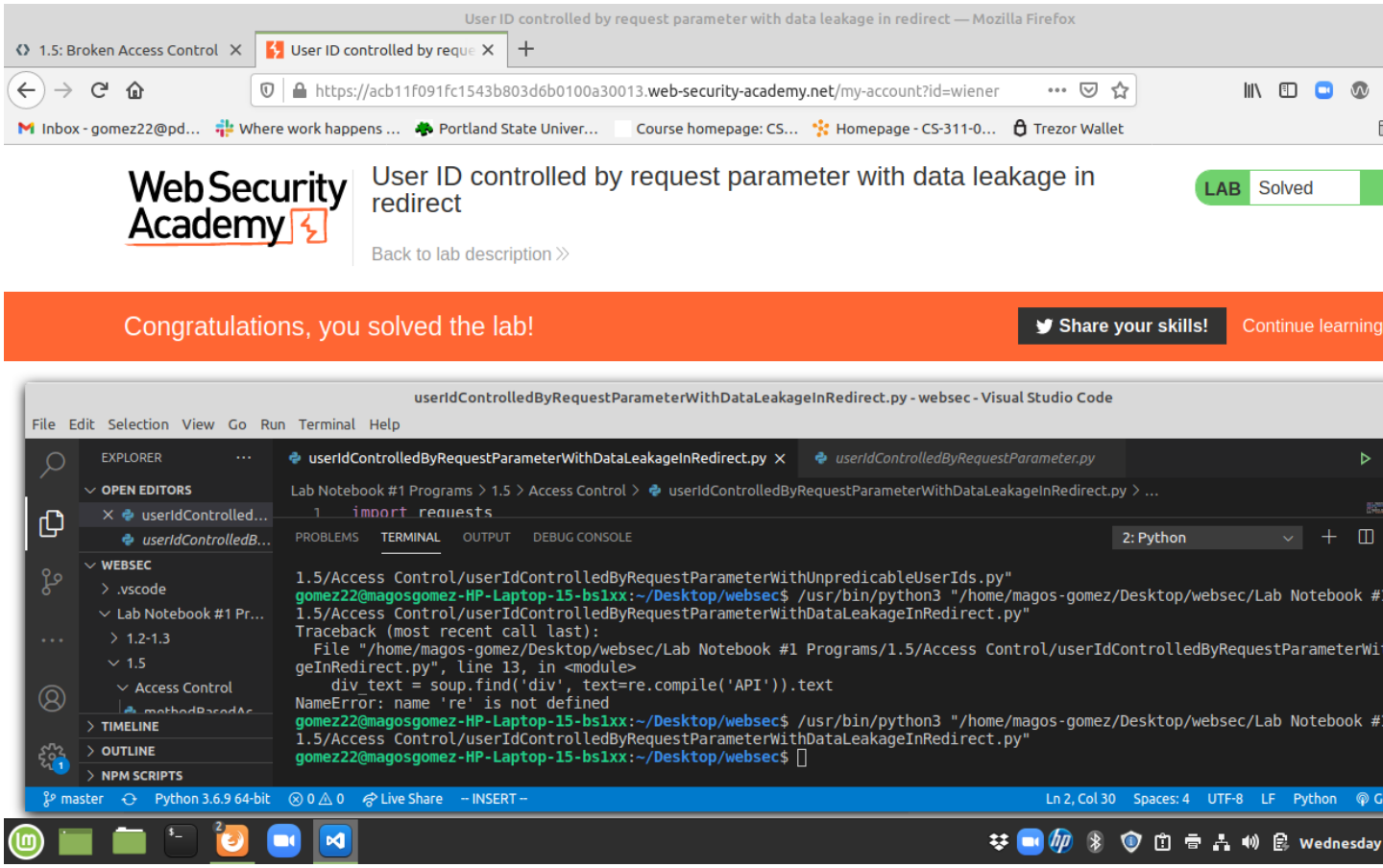
This level is vulnerable because it allows a user to view the private information of other users by passing in another user's ID into the URL as a parameter. The problem with this level is that user IDs are unpredictable. As an attacker, I had to search through posts until I found one from my target, Carlos. Then I could parse the HTML of his post's web page and find his user ID. Then I could add it to the "/my-account?userId=<insert_id_here>" and get access to the private information. To remediate this vulnerability, the site should prevent users from being able to alter user IDs in the URL or just not encode user IDs into the URL at all.



8. access-control (6)

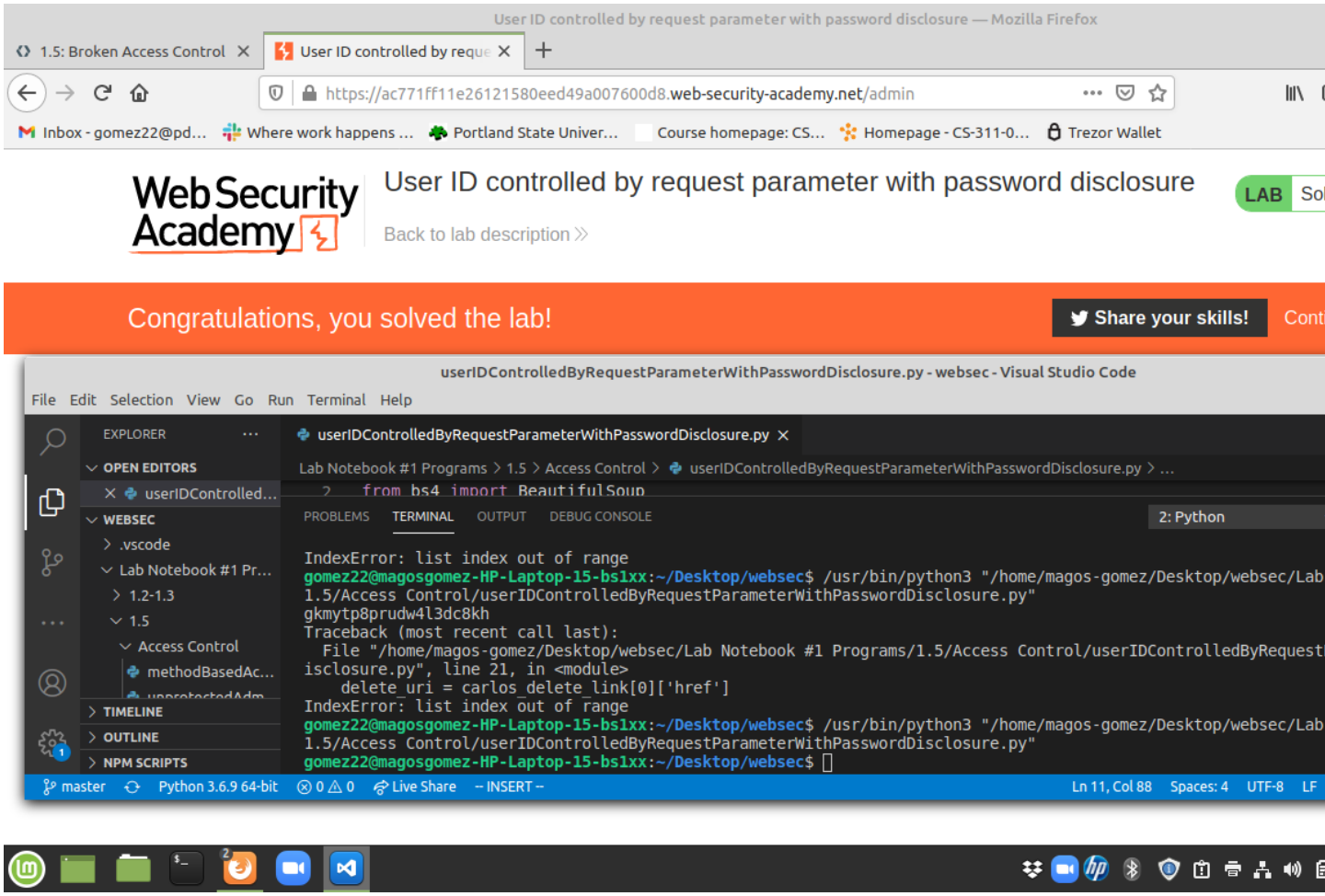
User Id controlled by request parameter with data leakage in redirect

The vulnerability in this lab was an Execute-After-Redirect vulnerability which results in information being leaked after a request is redirected. The information is leaked because of continued code execution. As an attacker, I can change the user Id in the URL to another user's id. Then I prevent redirects and extract the leaked information which allows me to login to another user's profile. There are two remediations for this vulnerability. The first one is to properly prevent code from continuing executing upon a redirect. Second, prevent users from changing the user ID in the URL.



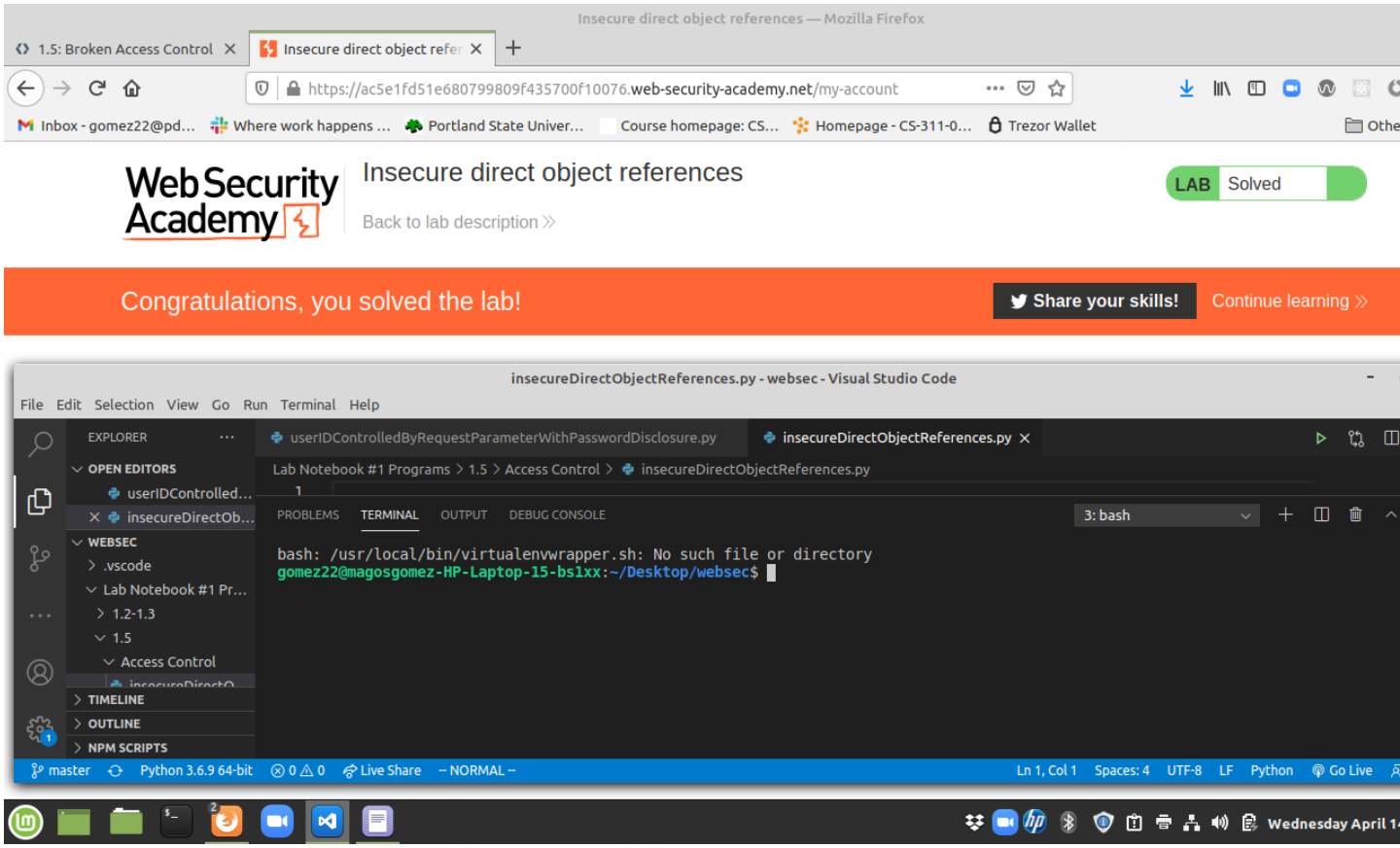
User Id controlled by request parameter with password disclosure

This vulnerability consists of two parts. The first part is that the site allows a user to change the user ID field in a url to be that of another user. This allowed me to change the user id to “administrator” which took me to the /login page. Since I had the “administrator” user id, the site didn’t clear the password input element after the last administrator login. I was able to parse the page to extract the administrator’s password and use it to login. To remediate this vulnerability, the site should always clear input elements upon submission and the site should prevent users from changing the user id field in the URL.



Insecure direct object references

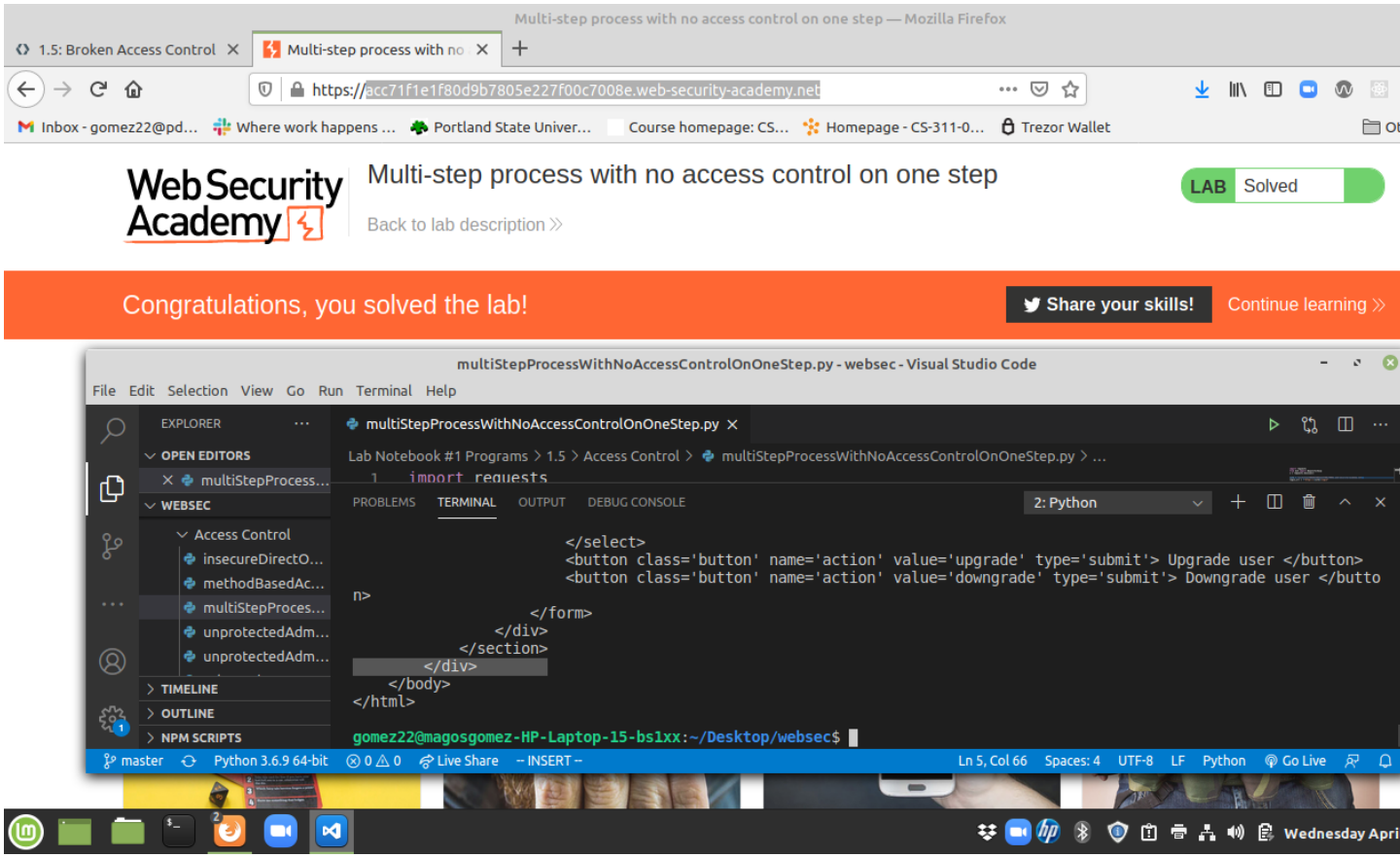
This level lets users chat with each other and then download the transcripts of the chat. This is vulnerable to attack because the transcripts use predictable file names. As an attacker, this let me download old chat logs and extract a password within one of them. To remediate this vulnerability, the site should randomize file names to make it hard or almost impossible to guess the names of chat files.



9. access-control (7)

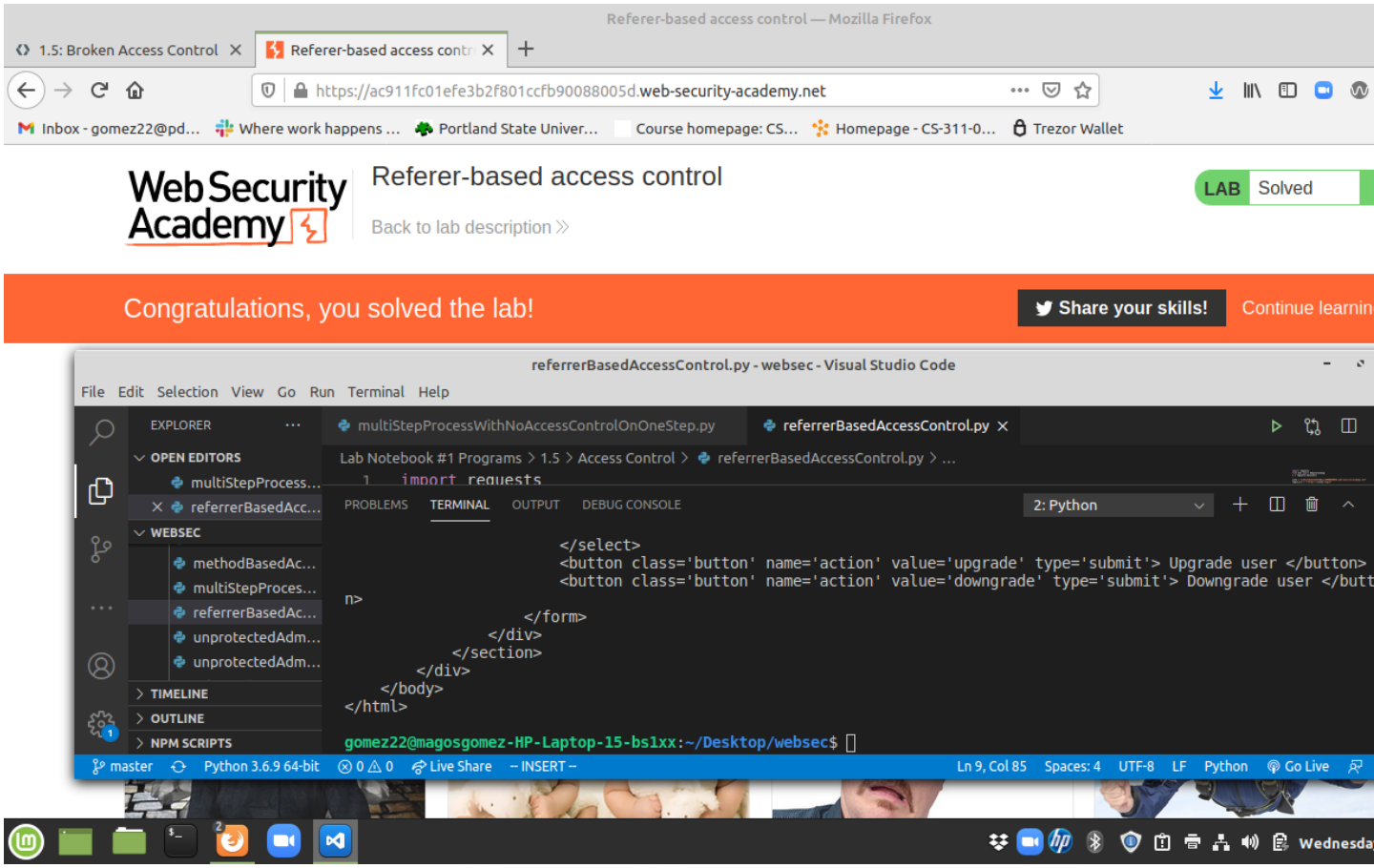
Multi-step process with no access control on one step

This lab is vulnerable because it only protects against GET requests to the /admin and /admin-roles pages, but it does not protect those pages against POST requests. As an attacker, I am able to use a POST request to escalate my privileges. To remediate this vulnerability, the site should prevent unauthorized POST requests from going through.



Referrer based access control

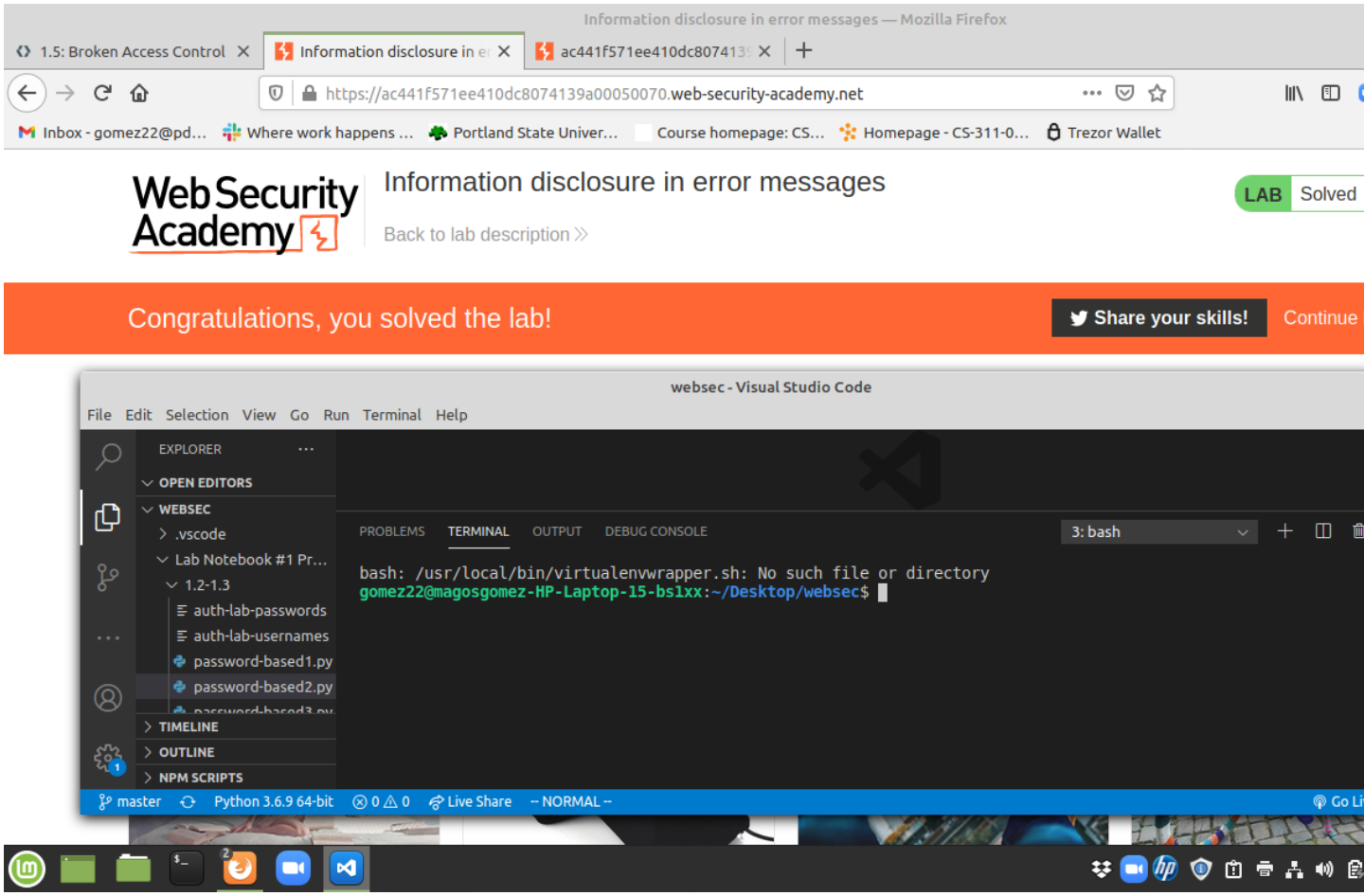
This lab is vulnerable because it allows a user to control the Referer header. As an attacker, I can set the Referer header to the site + /admin to make it appear as though my request is coming from an internal page. Specifically from a page that only administrators can access, making it seem as though I have admin access. To remediate this vulnerability, the site should not allow clients the ability to edit the Referer header.



10. information-disclosure

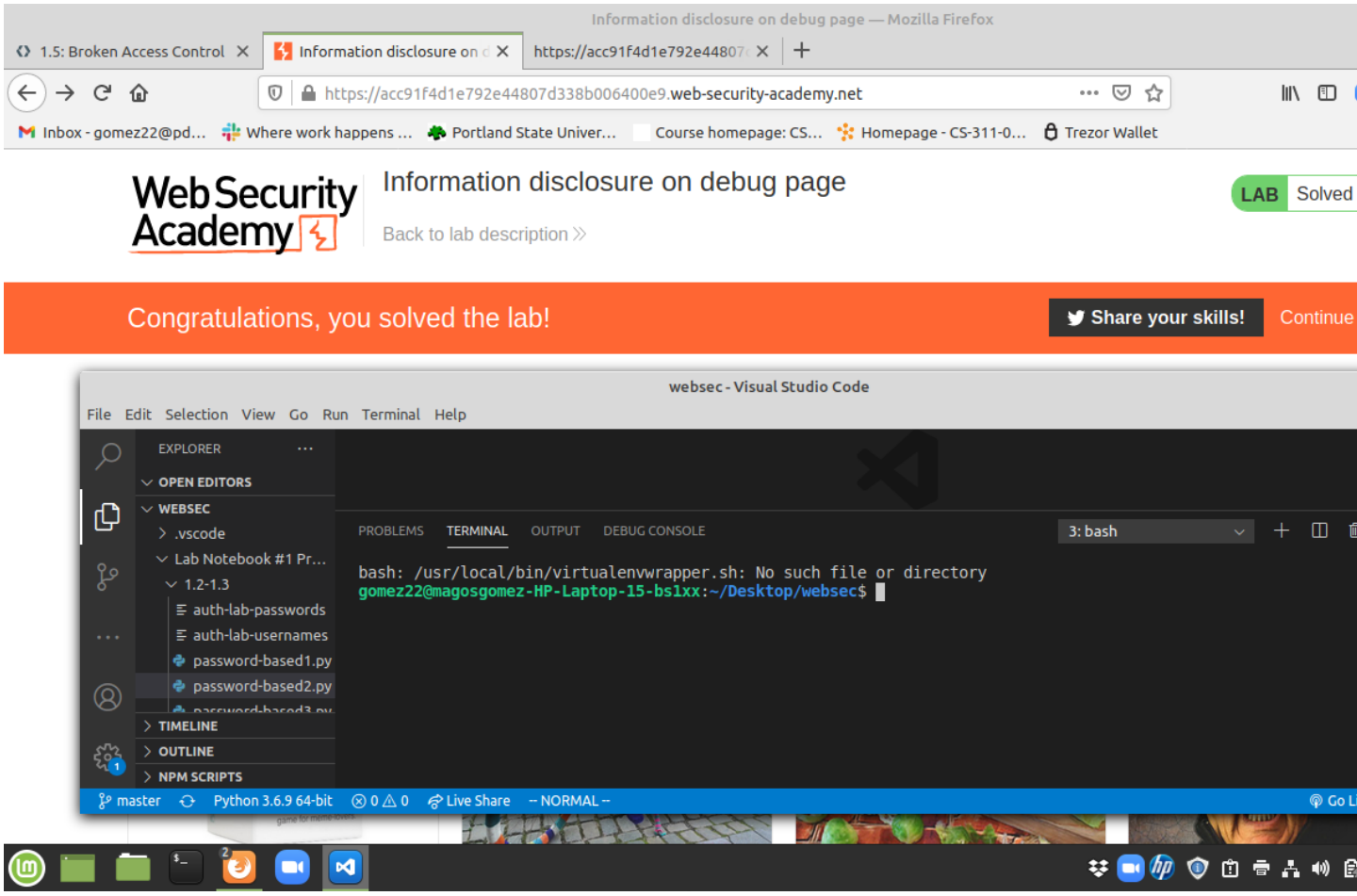
Lab infoleak in error messages

The vulnerability in this lab is that when you pass unexpected parameters in the URL, the site will reply with very informative error messages. As an attacker, I am able to use the excessive error information to find the server's version number. I could then lookup vulnerabilities for that software version and continue attacking the site. To remediate this, the site should provide less informative error message(i.e. Ones without potentially sensitive information within them).



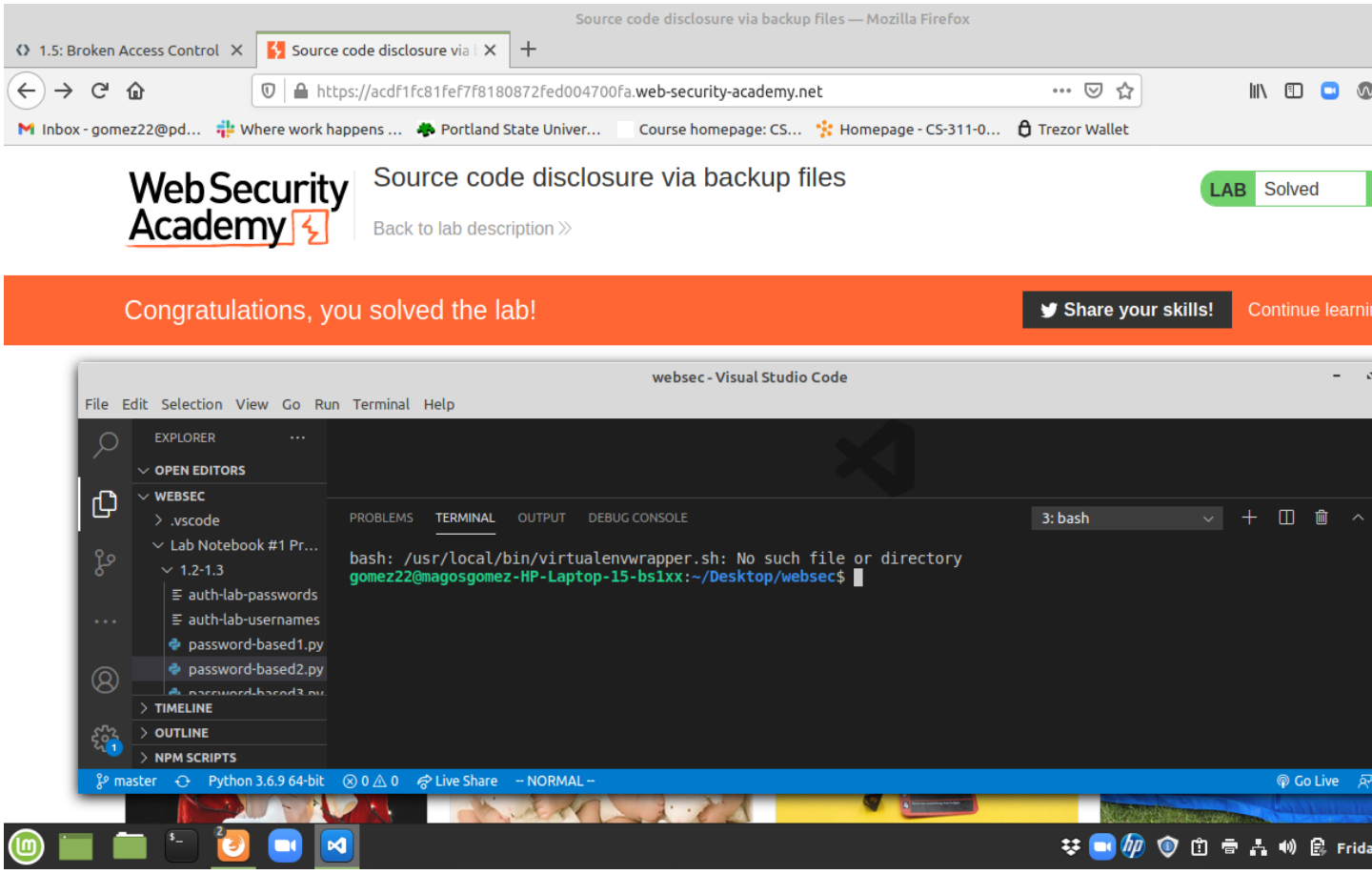
Lab infoleak on debug page

The vulnerability in this lab is that the developers left commented out debugging code in the page source. In the commented out code is a URI for a debug info page. Following the URI, I was able to locate API keys in the page. As an attacker, I could use those found credentials to more easily access critical/secure areas of the site. To remediate this attack, the site should remove all debugging code in each page to hide potentially accessible URLs.



Lab infoleak via backup files

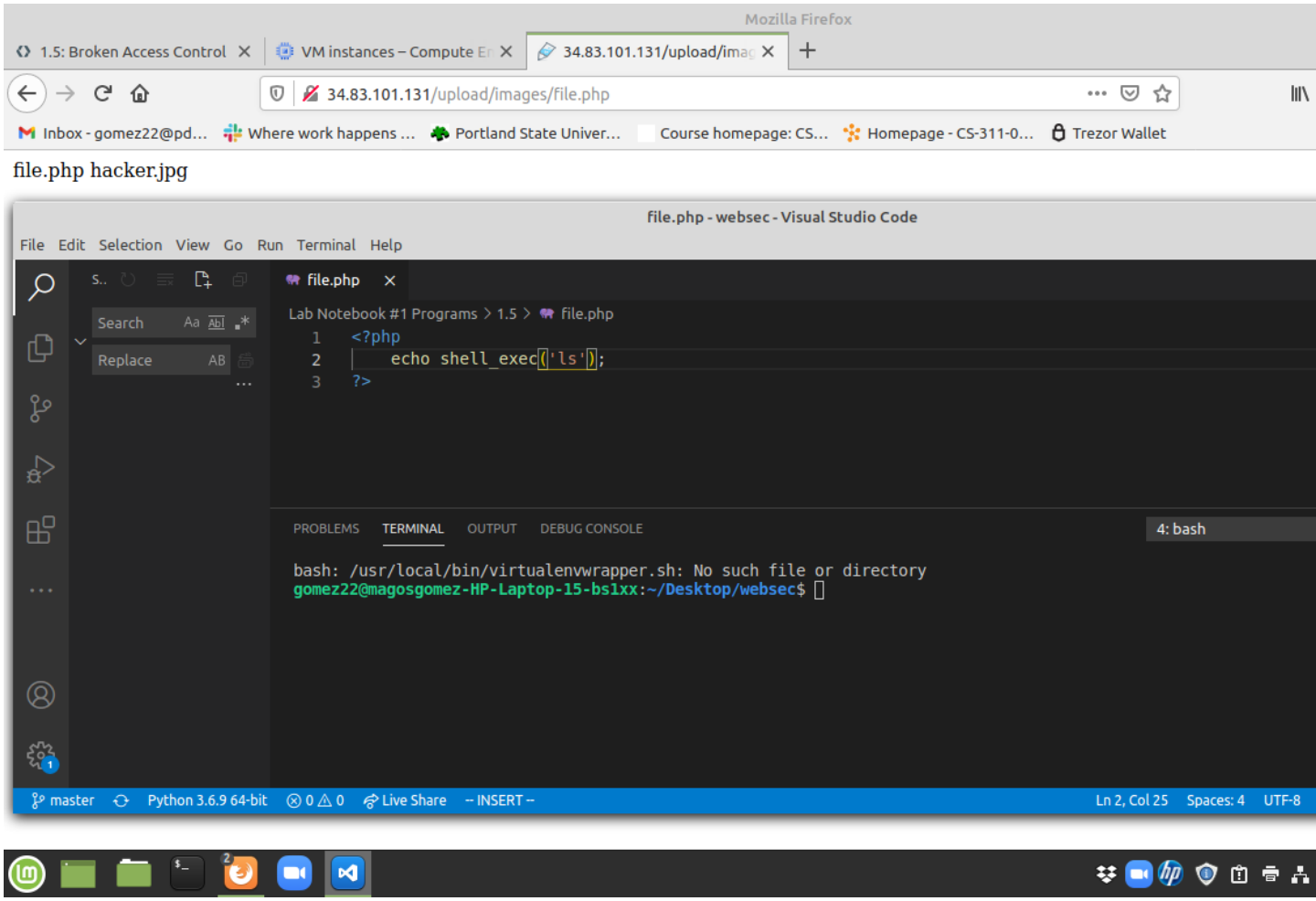
In this lab, I was able to find the /robots.txt resource which contained a “hidden” path of /backup. I followed this path and was able to locate a file containing secret information like passwords. I was then able to use those passwords to solve the level. To remediate this, the developers should’ve better looked through their potentially accessible resources and clean out any references to privileged information.



11. WFP1: File upload

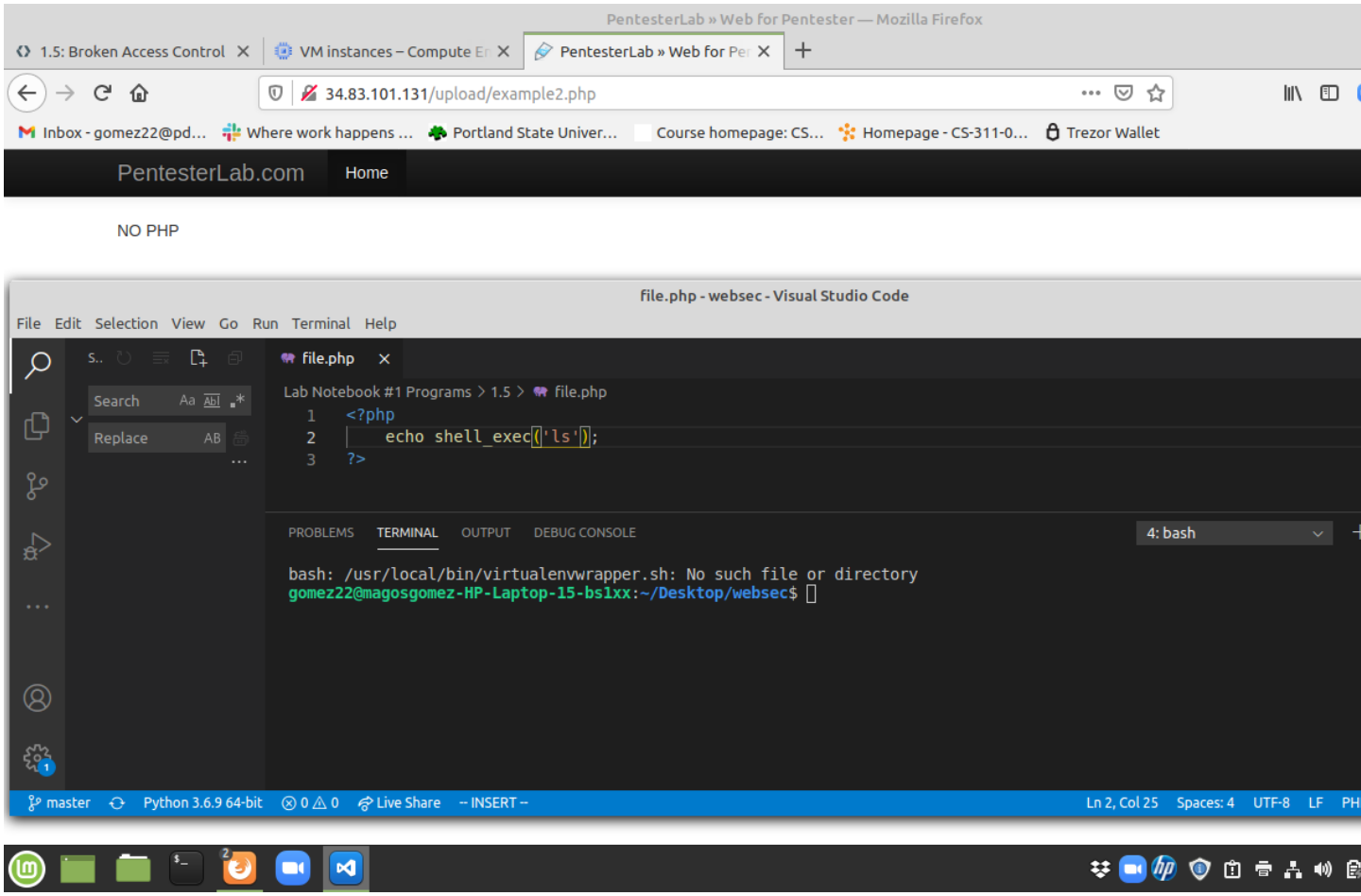
Example #1

Based on the URL, the programming language used to implement this web app is PHP. If I upload a PHP file and access it by clicking on the file link then I am pretty sure that I could get the file to execute. Since the website is implemented in PHP, I was able to upload a PHP file and get it to execute by accessing it. To remediate this, the site should prevent running uploaded files or run checks to ensure the uploaded file isn't a script.

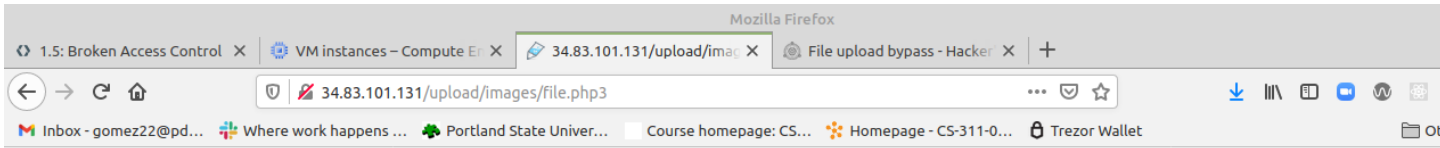


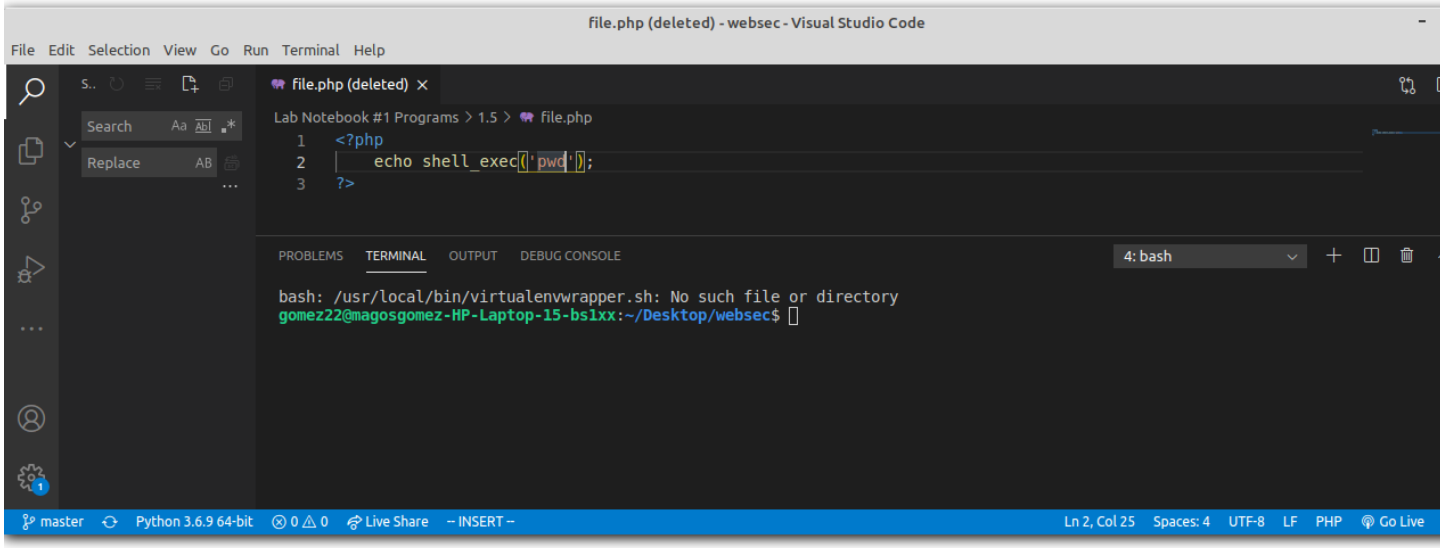
Example #2

Try uploading the same file used in the previous exercise.



The extension I found which was able to bypass the filter was “.php3” which successfully executed my “echo shell_exec(‘pwd’) command. To remediate this vulnerability, all potential php extensions should be filtered out.



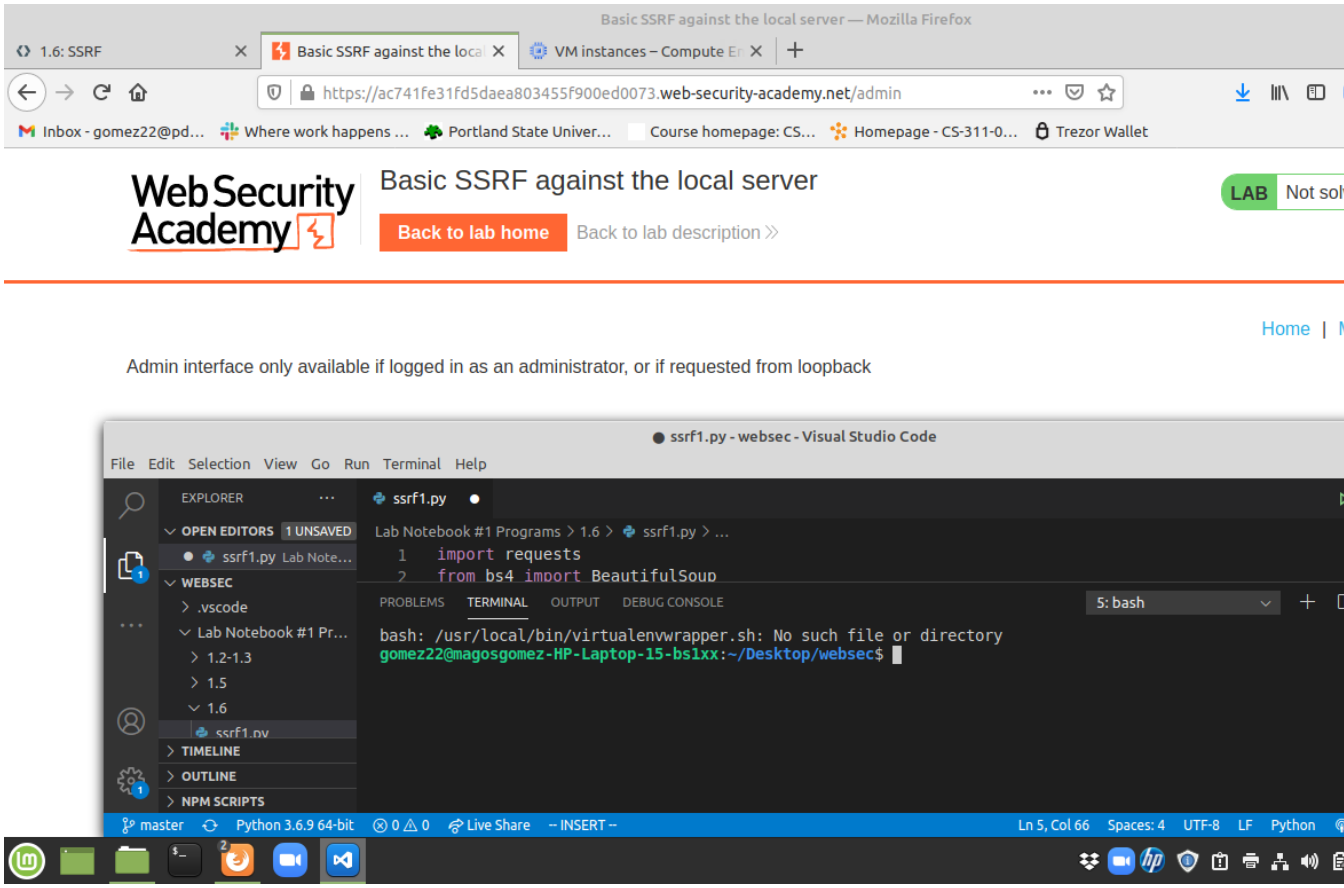


1. ssrf (1)

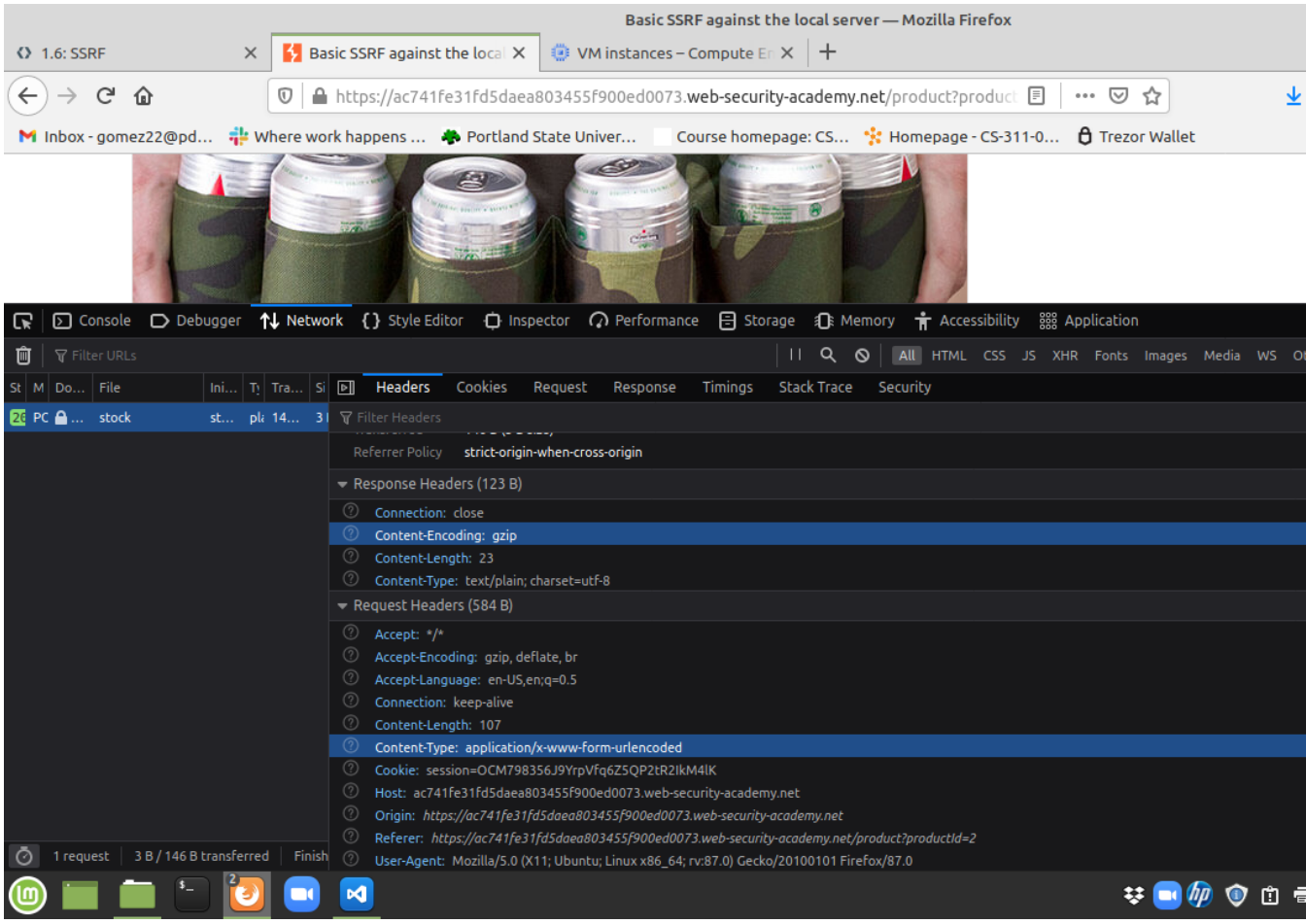
Basic ssrf against localhost

The vulnerability in this lab is that I can use an SSRF attack to request privileged information using a server as my proxy. The internal network trusts the server to request information so I modified my request to have the server make the request for the admin page. To remediate this, there should be some sort of filter to check if the payload is requesting privileged resources and authenticate that the requester is valid.

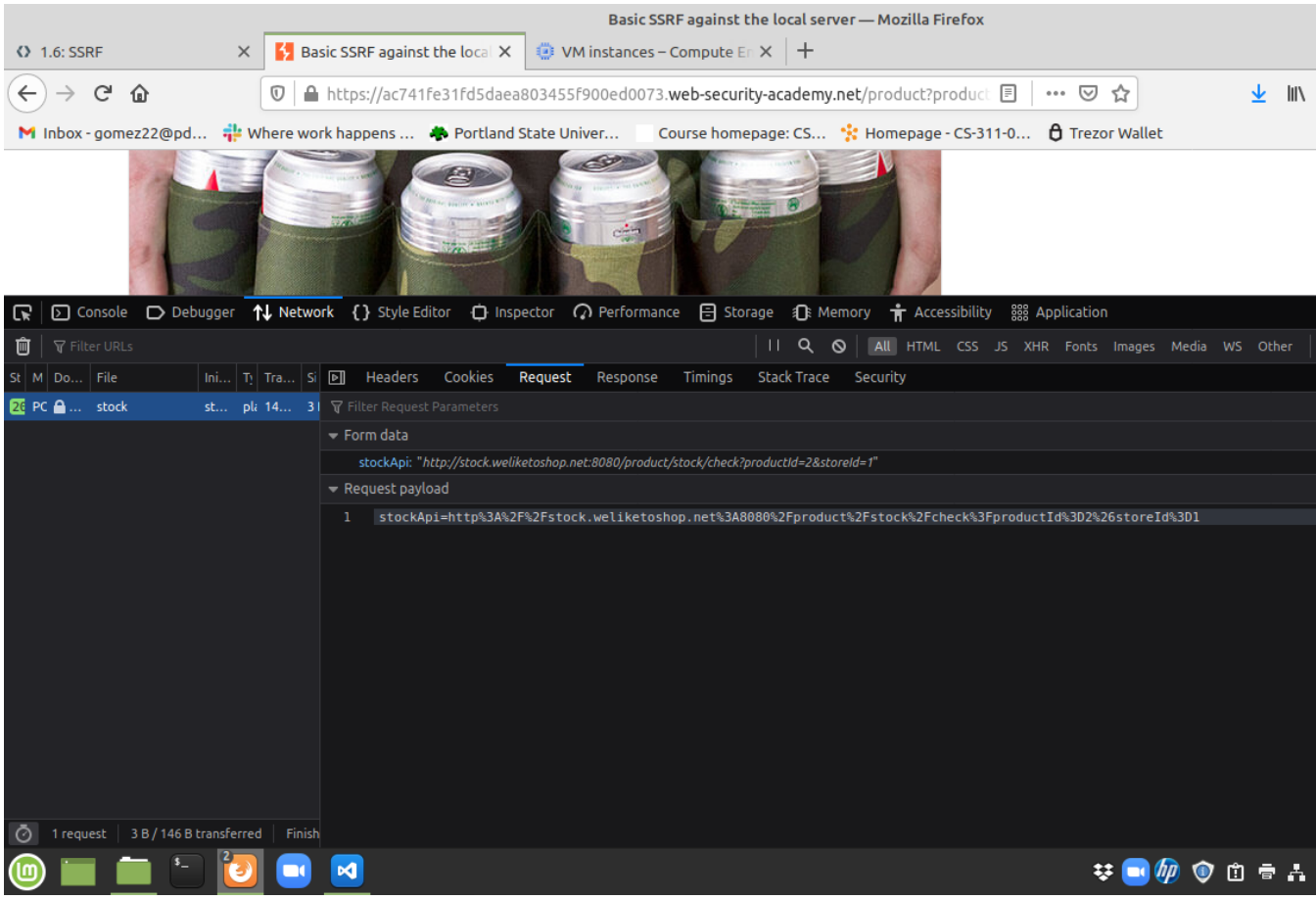
Take a screenshot of the message that is returned when you attempt to visit the /admin path of your site.

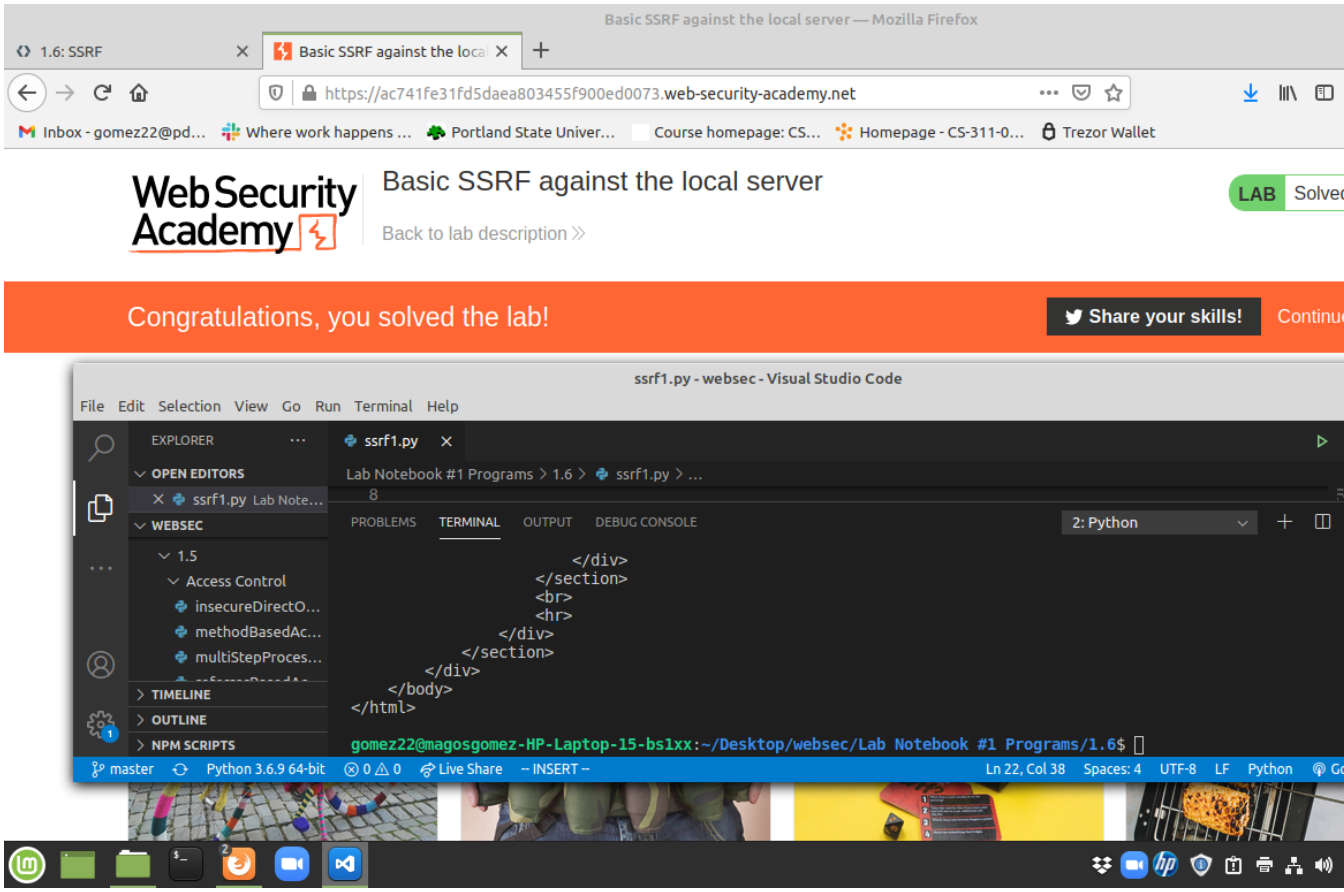


Take a screenshot that shows the Content-Type: request header that specifies the format of the form submission



Then, take a screenshot of the form submission payload.

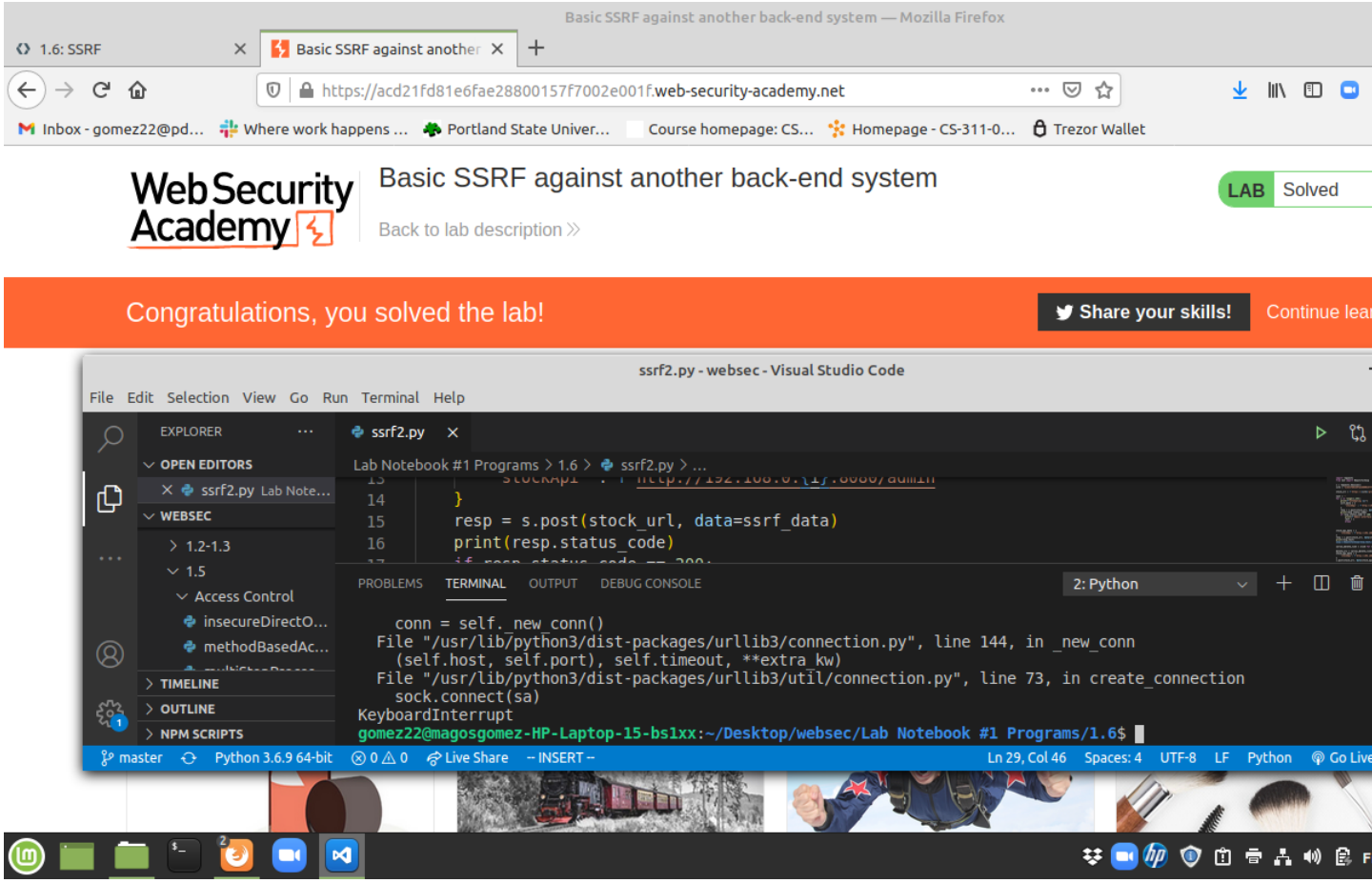

Basic SSRF against the local server — Mozilla Firefox
1.6: SSRF
Basic SSRF against the loca
VM instances – Compute Er
https://ac741fe31fd5daea803455f900ed0073.web-security-academy.net/product?product
Inbox - gomez22@pd...
Where work happens ...
Portland State Univer...
Course homepage: CS...
Homepage - CS-311-0...
Trezor Wallet
Console
Debugger
Network
Style Editor
Inspector
Performance
Storage
Memory
Accessibility
Application
Filter URLs
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
St
M
Do...
File
Ini...
Ty
Tra...
Si
PC ...
stock
st...
pla
14...
Headers
Cookies
Request
Response
Timings
Stack Trace
Security
Filter Request Parameters
Form data
stockApi: "http://stock.weliketoshop.net:8080/product/stock/check?productId=2&storeId=1"
Request payload
1
stockApi=http%3A%2F%2Fstock.weliketoshop.net%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D2%26storeId%3D1
1 request
3 B / 146 B transferred
Finish



2. ssrf (2)

Basic ssrf against backend system

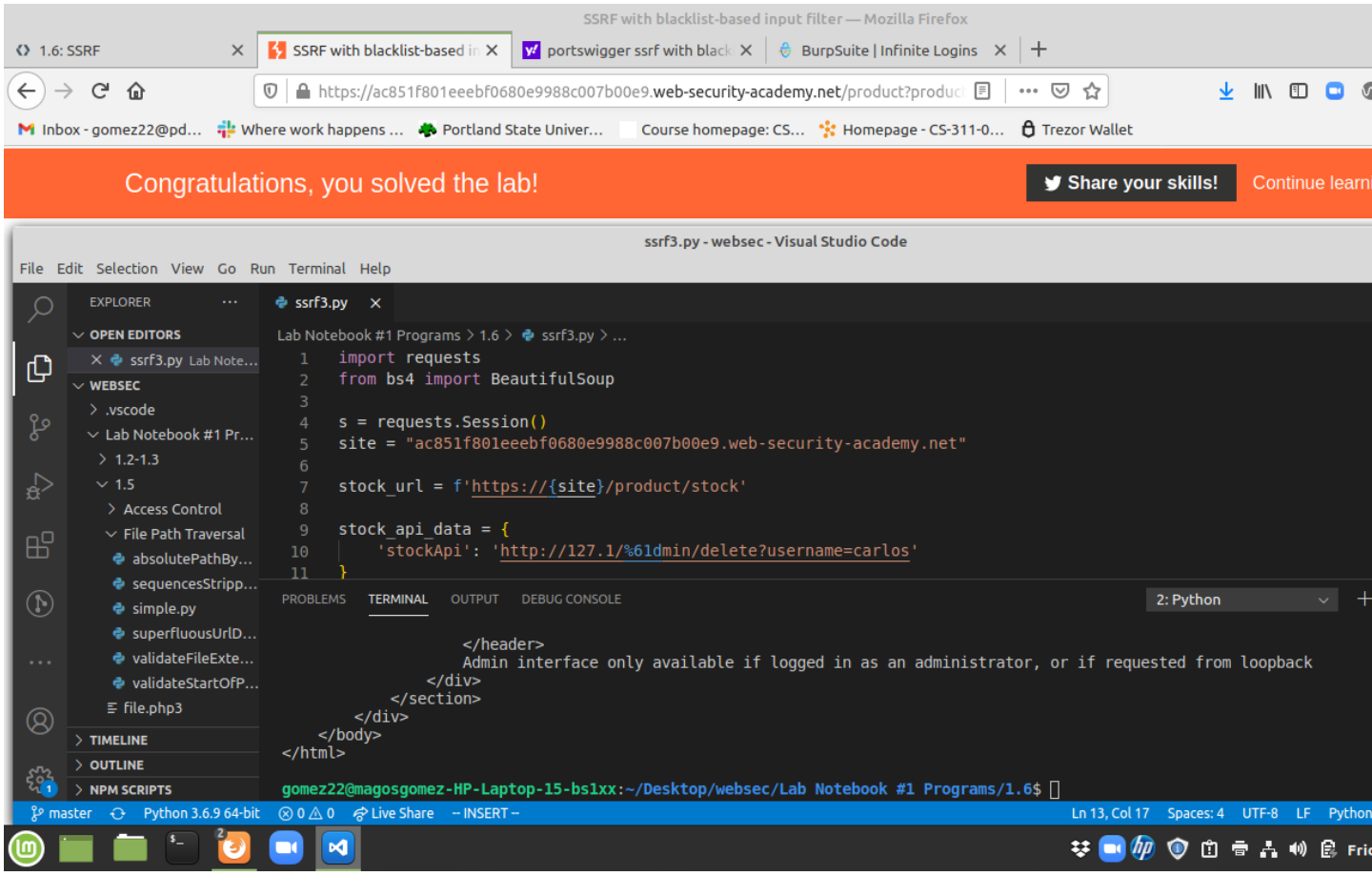
This lab revolves around probing for an admin interface on a private IP address that I am requesting through an SSRF attack. Once I have found the admin interface, I can continue to use it through an SSRF attack to access privileged information like deleting carlos' profile. To remediate this, the admin interfaces should be secured in a way to prevent access from servers which are used by outside clients. This way only internal, authenticated serves can request the privileged information.



3. ssrf (3)

Ssrf with blacklist filter

This lab prevents ssrf attacks using ‘localhost’ however, as an attacker, I can attempt using aliases for localhost in the hopes that the developers forgot to filter out one of the aliases. They did forget so I used “127.1”. Then the site filters out the /admin path. So I had to double encode the /admin path in the url in order to bypass the filter. To remediate this vulnerability, accounting for all of the localhost aliases would be the best bet.



4. ssrf (4)

Ssrf filter bypass via open redirection

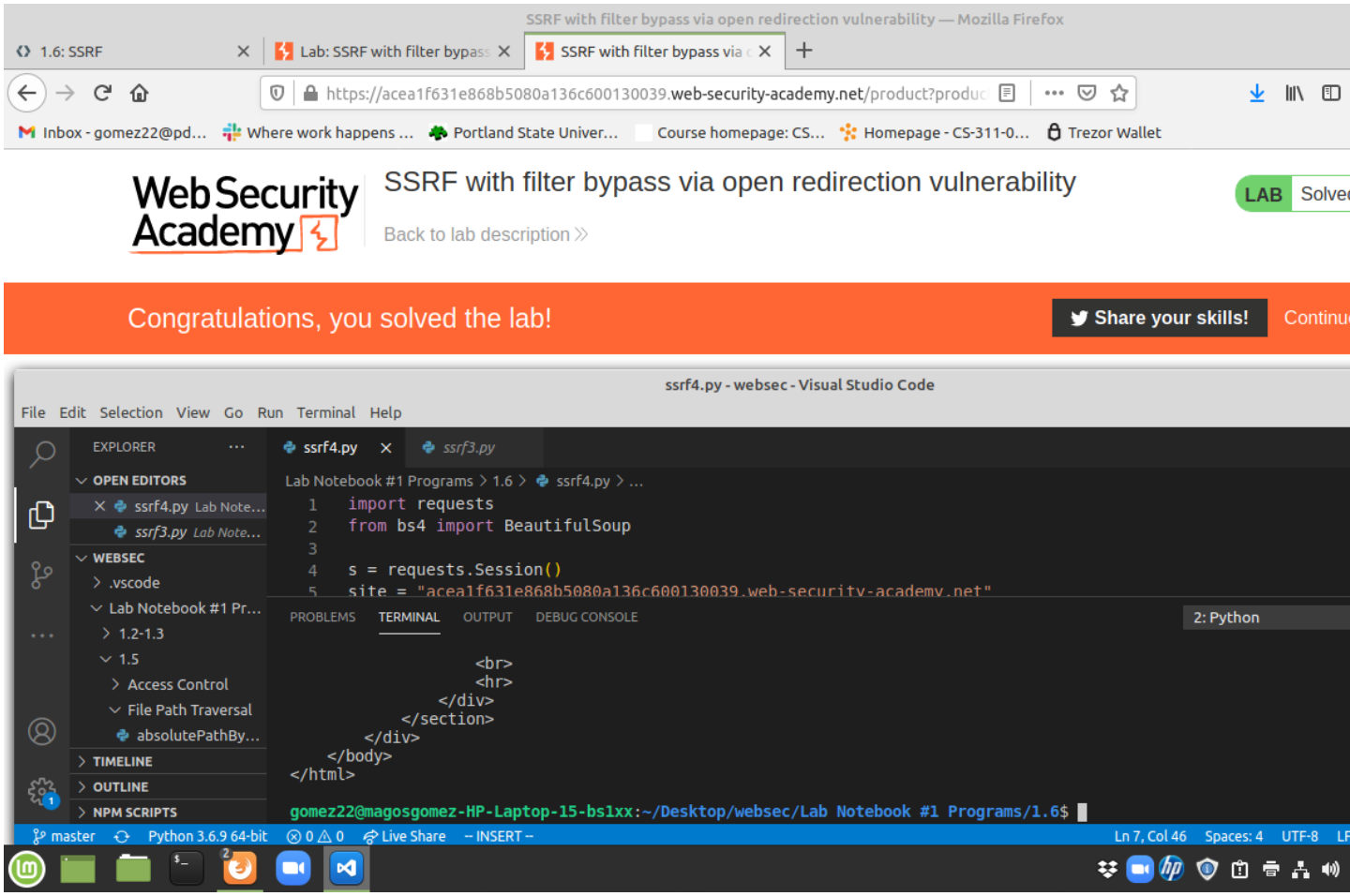
This vulnerability is that there exists an unvalidated redirect. As an attacker, I can change the redirect path parameter to the admin interface. To remediate this vulnerability, the site should have validation on redirect paths.

What is the page that implements the redirect?

The page is /nextProduct

What parameter specifies the URI that the page uses to redirect the browser to?

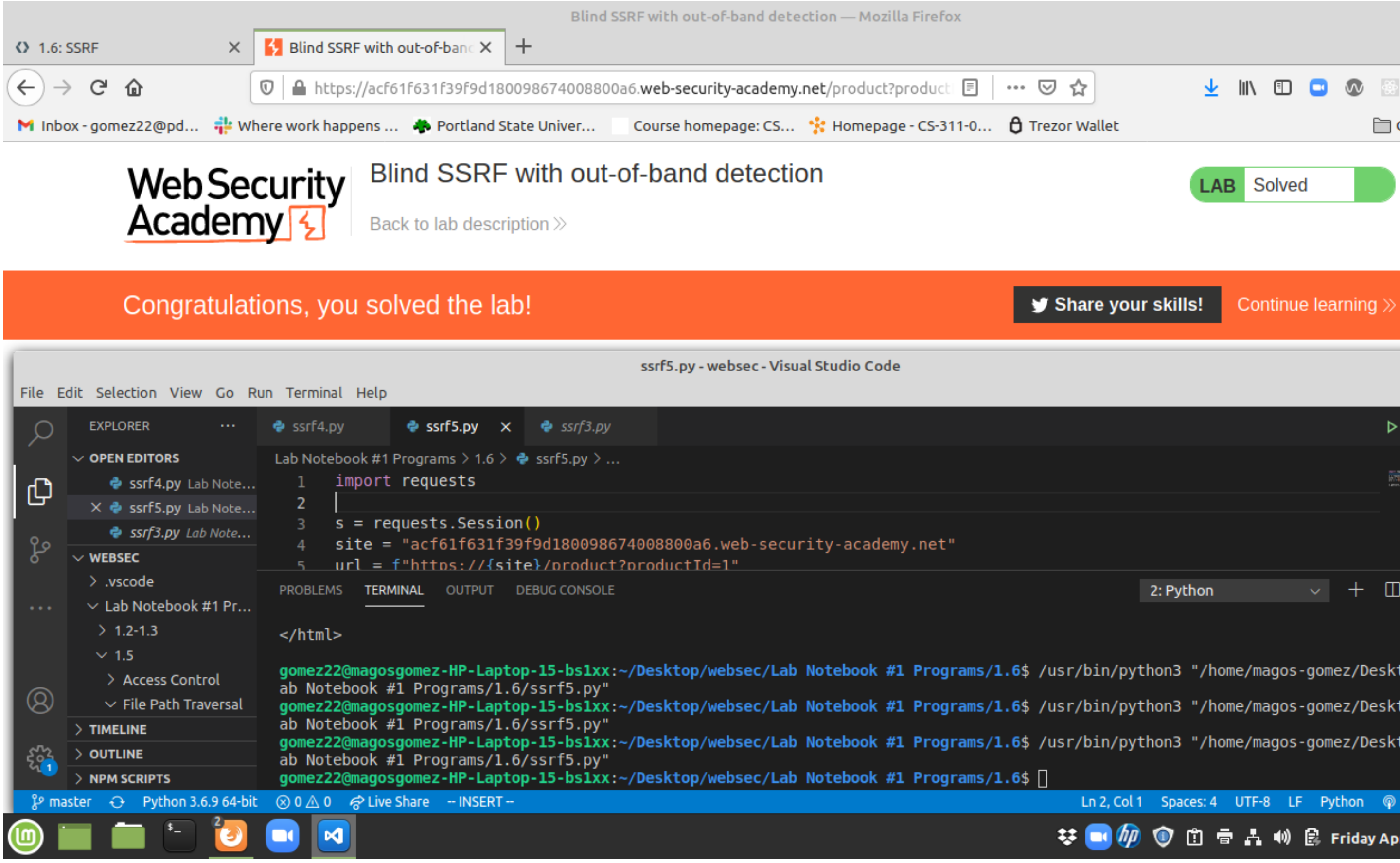
The parameter is “path=”



5. ssrf/blind

Out of band detection

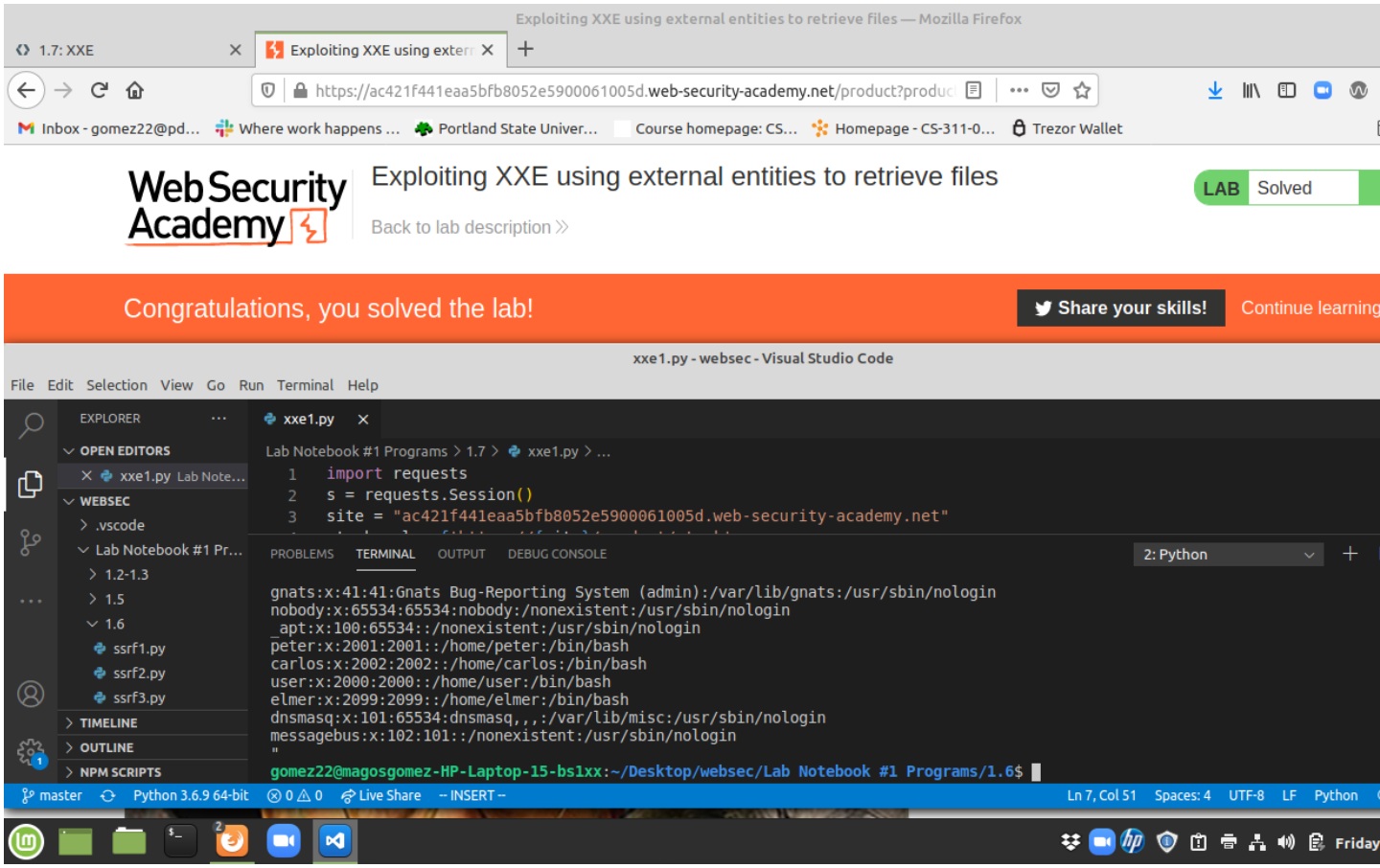
The vulnerability in this lab is that the site fetches the URL in the Referer header when a product page is loaded. As an attacker, I can insert the desired site into the referer header and hijack the request. To remediate this vulnerability, there should be validation of the referer header.



1.7 XXE

1. XXE(1)

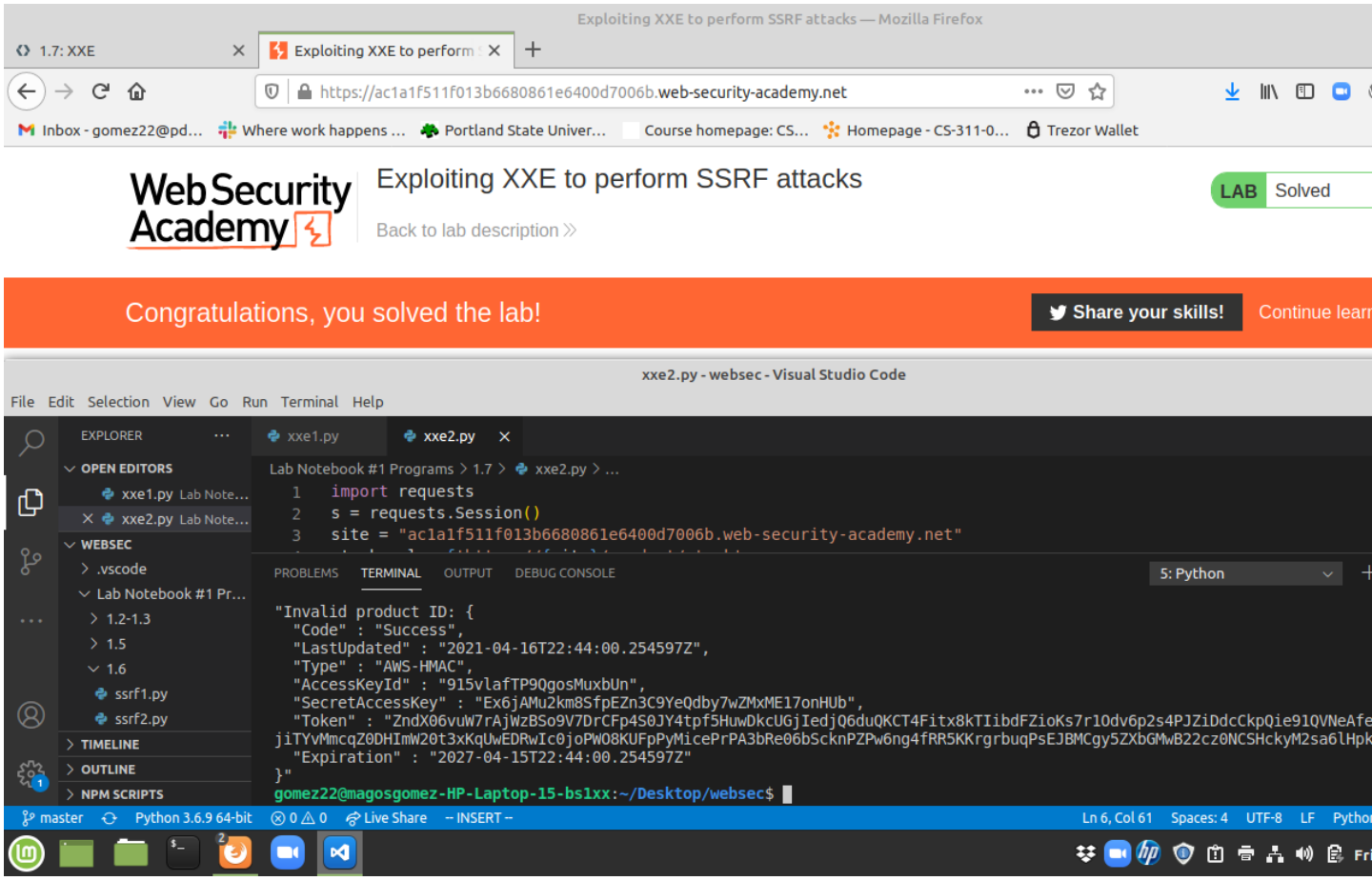
The vulnerability of this lab is that the site allows unsanitized XML input from the client, so I can include arbitrary files as the attacker. I am able to include a file that contains code which exfiltrates the contents of the etc/password. To remediate this vulnerability, the site should sanitize XML input from the client.



2. XXE (2)

Exploiting xxe to perform ssrf

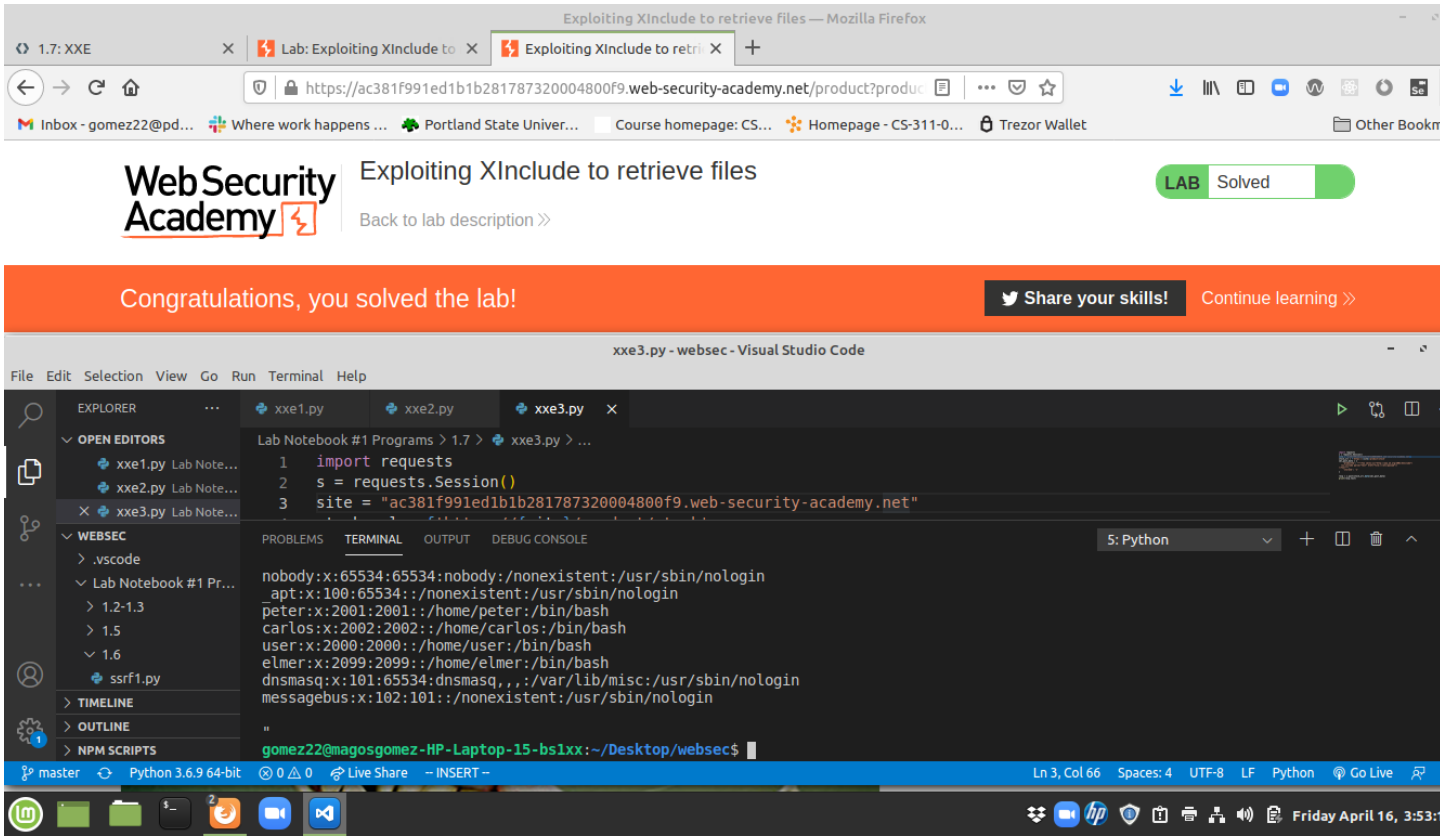
The vulnerability of this lab is that I can send an XML payload that is attempting to force the server to retrieve files from “protected” servers. To remediate this vulnerability, the site should sanitize XML input from the client.



3. XXE (3)

Xinclude-attack

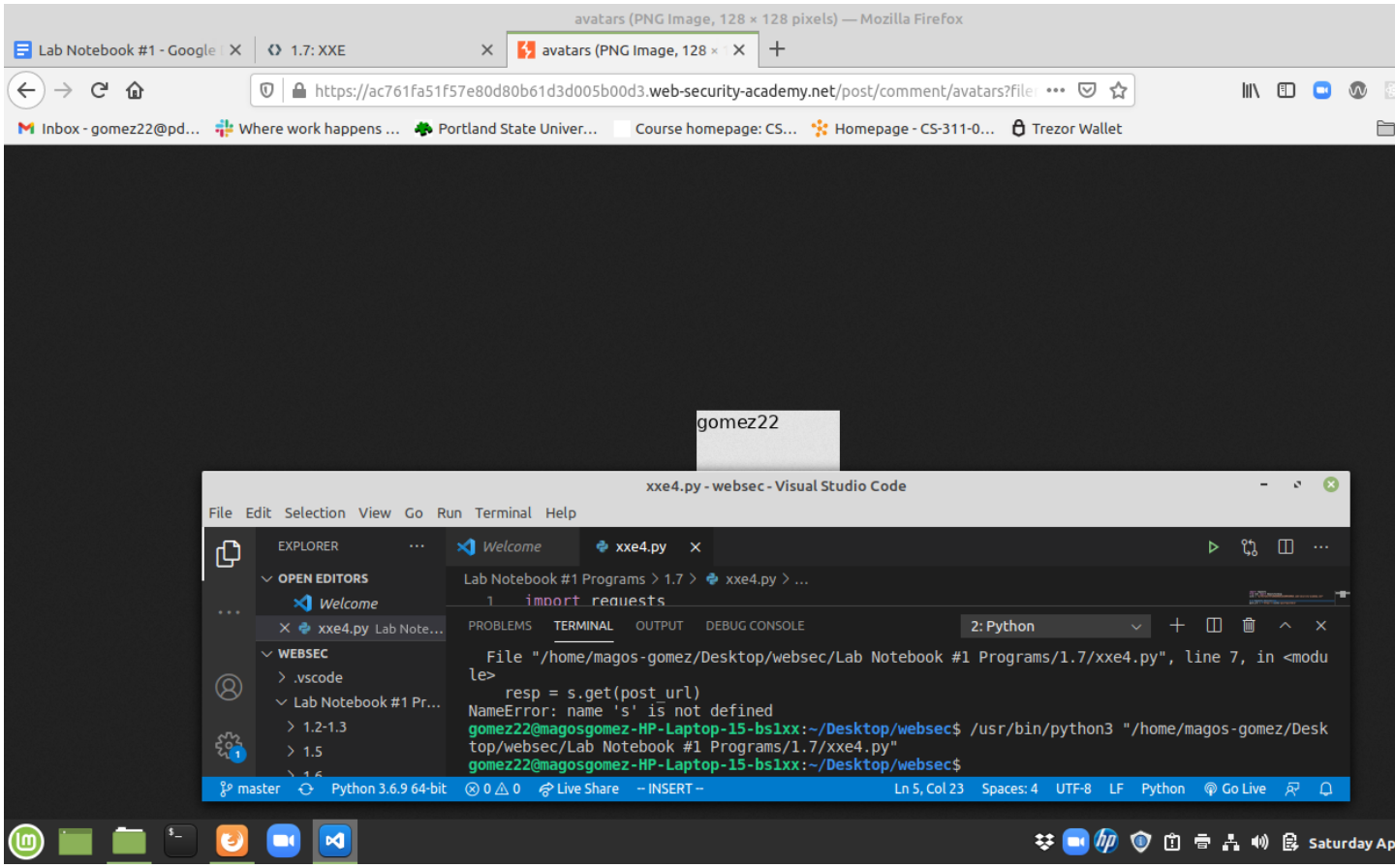
The vulnerability of this lab is that I can insert invalid XML payloads and the server has to enter an error handling state to process the payload. The site doesn't properly process the payload and leaks the contents of the sensitive file that I requested in the malformed XML payload. To remediate this vulnerability, the site should properly sanitize error messages.

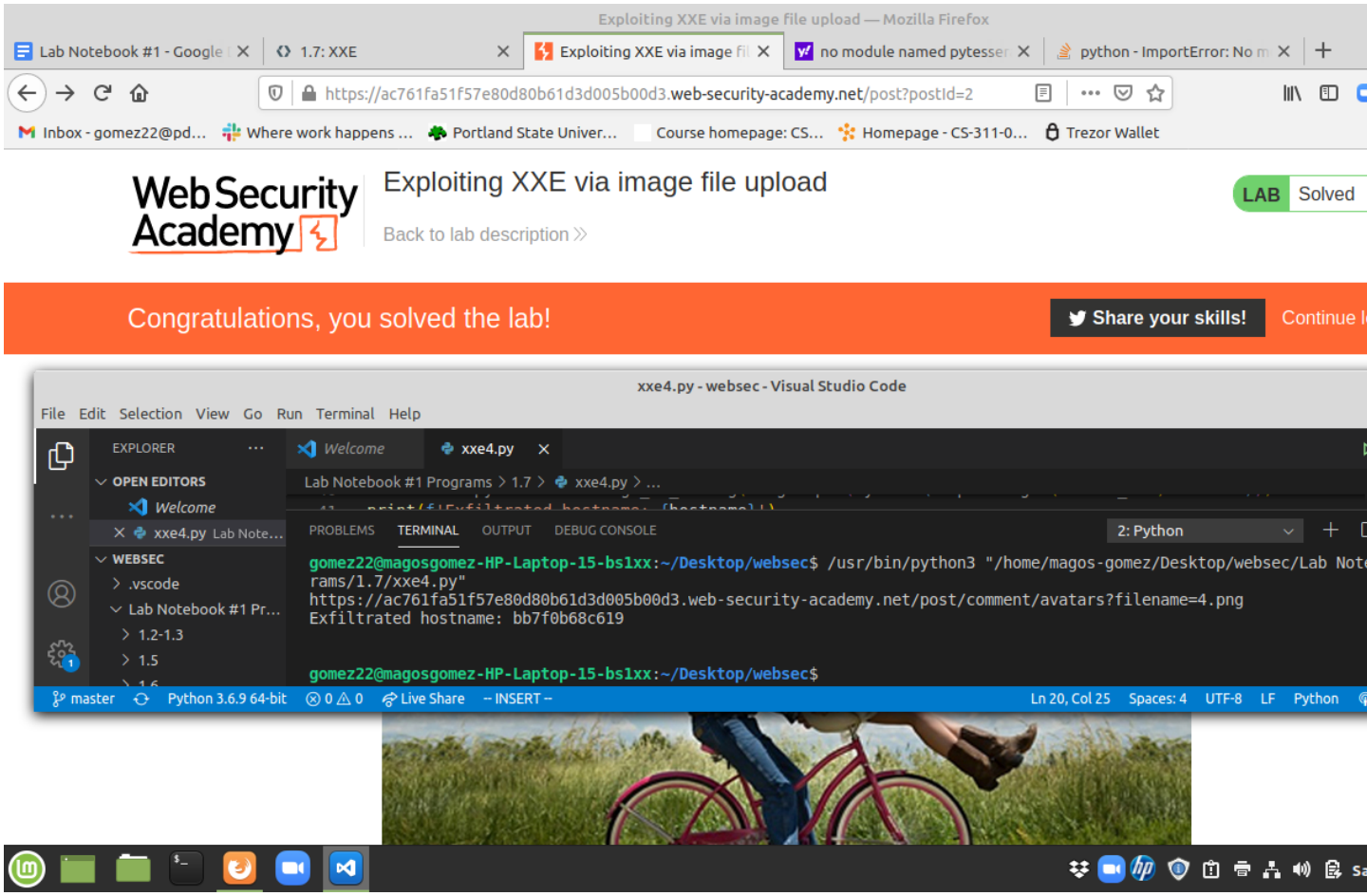


4 & 5 XXE

Xxe via file upload

The vulnerability of this lab is that I can upload a malicious SVG image that contains an XXE payload that causes sensitive data to be exfiltrated. To remediate this vulnerability, the site should have some sort of filtering for user provided uploads.


avatars (PNG Image, 128 × 128 pixels) — Mozilla Firefox
Lab Notebook #1 - Google
1.7: XXE
avatars (PNG Image, 128
https://ac761fa51f57e80d80b61d3d005b00d3.web-security-academy.net/post/comment/avatars?file
Inbox - gomez22@pd...
Where work happens ...
Portland State Univer...
Course homepage: CS...
Homepage - CS-311-0...
Trezor Wallet
gomez22
xxe4.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
Welcome
xxe4.py Lab Note...
WEBSEC
.vscode
Lab Notebook #1 Pr...
1.2-1.3
1.5
Welcome
xxe4.py
Lab Notebook #1 Programs > 1.7 > xxe4.py > ...
1 import requests
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
2: Python
File "/home/magos-gomez/Desktop/websec/Lab Notebook #1 Programs/1.7/xxe4.py", line 7, in <module>
resp = s.get(post_url)
NameError: name 's' is not defined
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab Notebook #1 Programs/1.7/xxe4.py"
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master
Python 3.6.9 64-bit
0 0
Live Share
-- INSERT --
Ln 5, Col 23
Spaces: 4
UTF-8
LF
Python
Go Live
Saturday Ap


Exploiting XXE via image file upload — Mozilla Firefox
Lab Notebook #1 - Google
1.7: XXE
Exploiting XXE via image fi
no module named pytesser
python - ImportError: No m
https://ac761fa51f57e80d80b61d3d005b00d3.web-security-academy.net/post?postId=2
Inbox - gomez22@pd...
Where work happens ...
Portland State Univer...
Course homepage: CS...
Homepage - CS-311-0...
Trezor Wallet
Web Security Academy
Exploiting XXE via image file upload
Back to lab description
LAB
Solved
Congratulations, you solved the lab!
Share your skills!
Continue
xxe4.py - websec - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
Welcome
xxe4.py Lab Note...
WEBSEC
.vscode
Lab Notebook #1 Pr...
1.2-1.3
1.5
Welcome
xxe4.py
Lab Notebook #1 Programs > 1.7 > xxe4.py > ...
PROBLEMS
TERMINAL
OUTPUT
DEBUG CONSOLE
2: Python
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$ /usr/bin/python3 "/home/magos-gomez/Desktop/websec/Lab Not
rams/1.7/xxe4.py"
https://ac761fa51f57e80d80b61d3d005b00d3.web-security-academy.net/post/comment/avatars?filename=4.png
Exfiltrated hostname: bb7f0b68c619
gomez22@magosgomez-HP-Laptop-15-bs1xx:~/Desktop/websec$
master
Python 3.6.9 64-bit
0 0
Live Share
-- INSERT --
Ln 20, Col 25
Spaces: 4
UTF-8
LF
Python