Lab 1.2: Web Programming **2**

Lab 1.3: Broken Authentication **5**

Write ups for Lab 1.3: **9**

Lab 1.5: Broken Access Control **10**

Lab 1.5.1 Write Up: **12**

Lab 1.5.2 Write Up: **15**

Lab 1.5.3 Write Up: **18**

Labs 1.5.5-1.5.11 write up: **28**

Labs 1.6: SSRF **30**

Labs 1.6: SSRF Write Up: **32**

Labs 1.7: XXE **33**

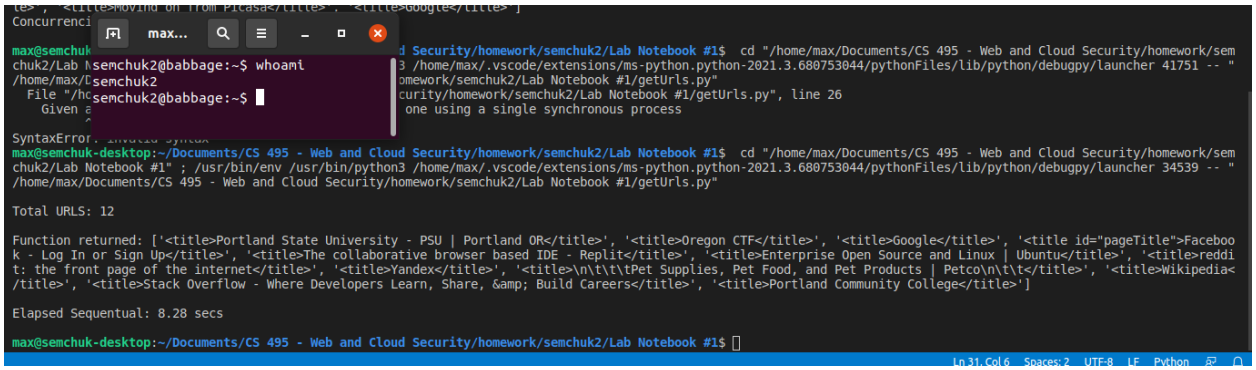
Labs 1.7: XXE Write Up: **35**

03/31/2021

Lab 1.2: Web Programming

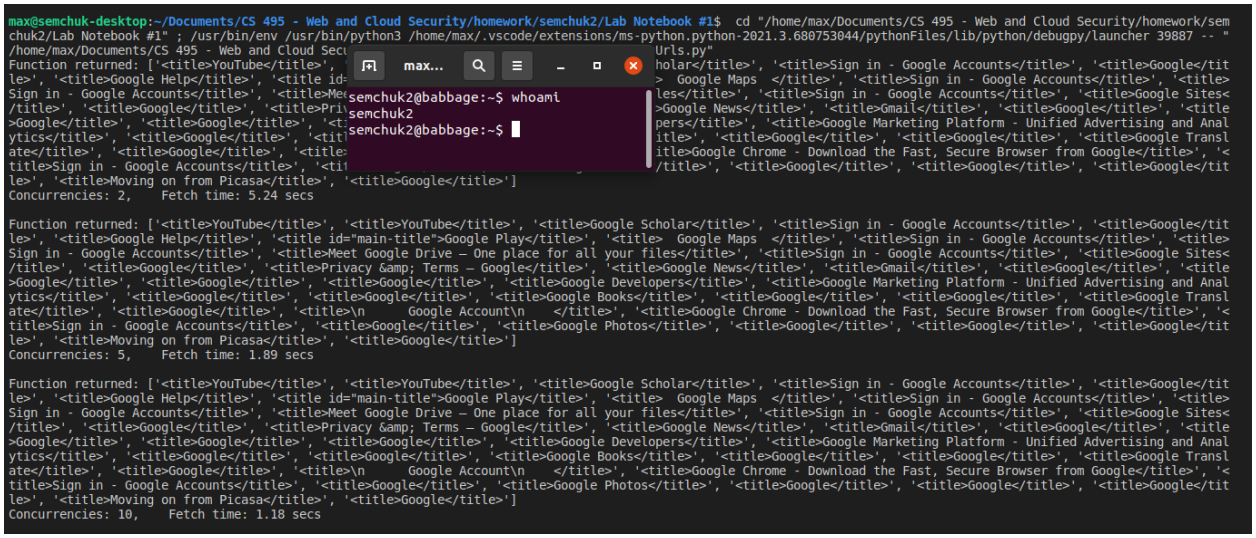
CS 495 – Intro to web and cloud security Notebook
Note: I have included screenshots with me logged into my linux at psu account to show I am doing the work, and not copying. I am doing the work locally on my own machine since everything works here, instead of ssh'ing / or using a dedicated VM.

#4
Sequential program:



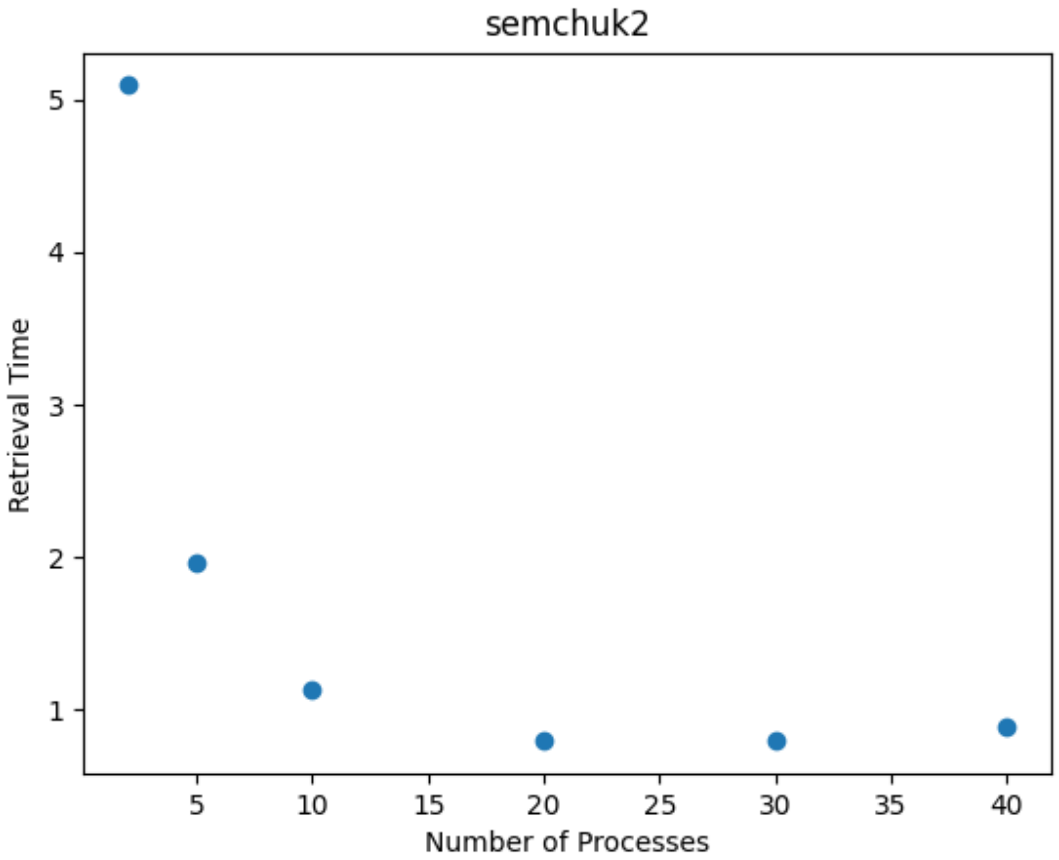
it took .35 seconds to run on a sequential run over 12 URL's

#6
Multiprocessing program:

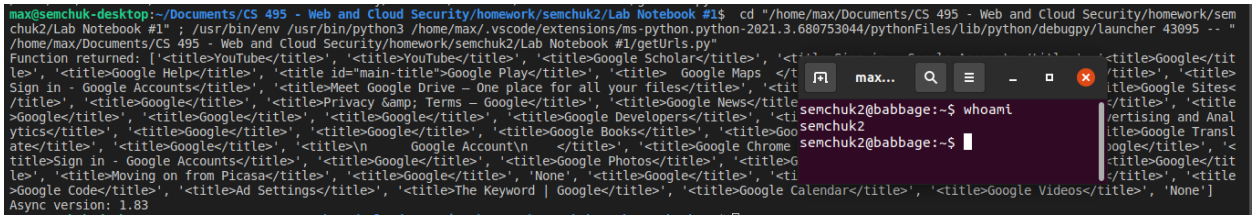


here, it shows 12 URLS being fetched with 2,5,10 concurrences in 5.24, 1.89, and 1.18 seconds respectively.

#8
matplotlib on repl.it



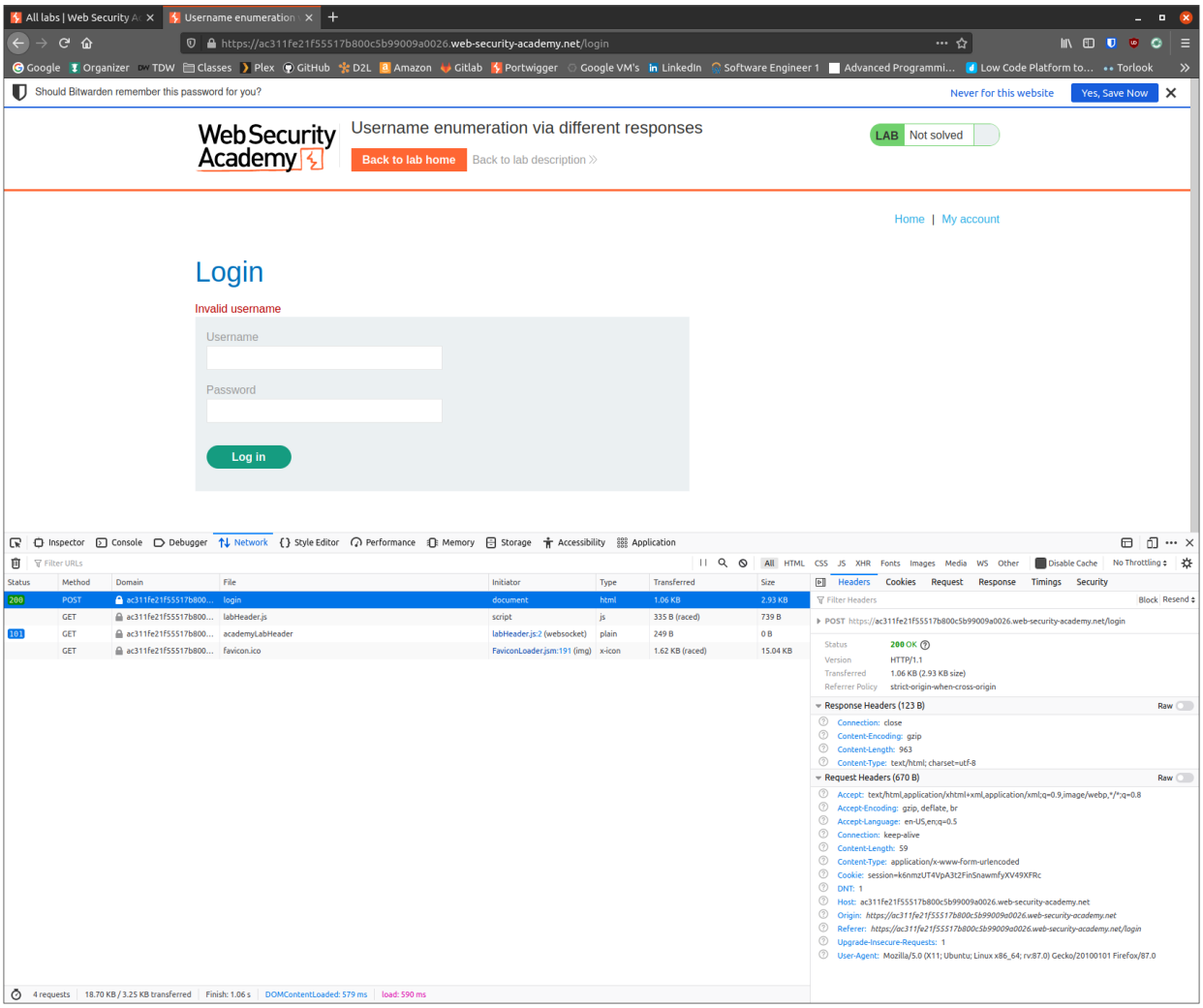
#11



Description of Learning Objective: To better understand some basics of python, as well as some web scraping techniques. The lab also goes over how to set up asynchronous methods to get work done fast.

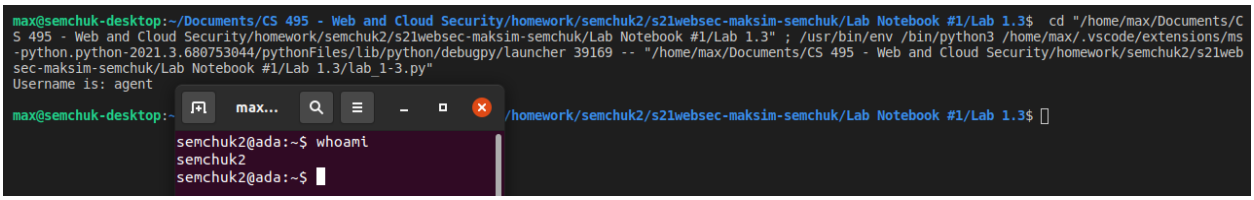
Lab 1.3: Broken Authentication

Step #2: Login form:

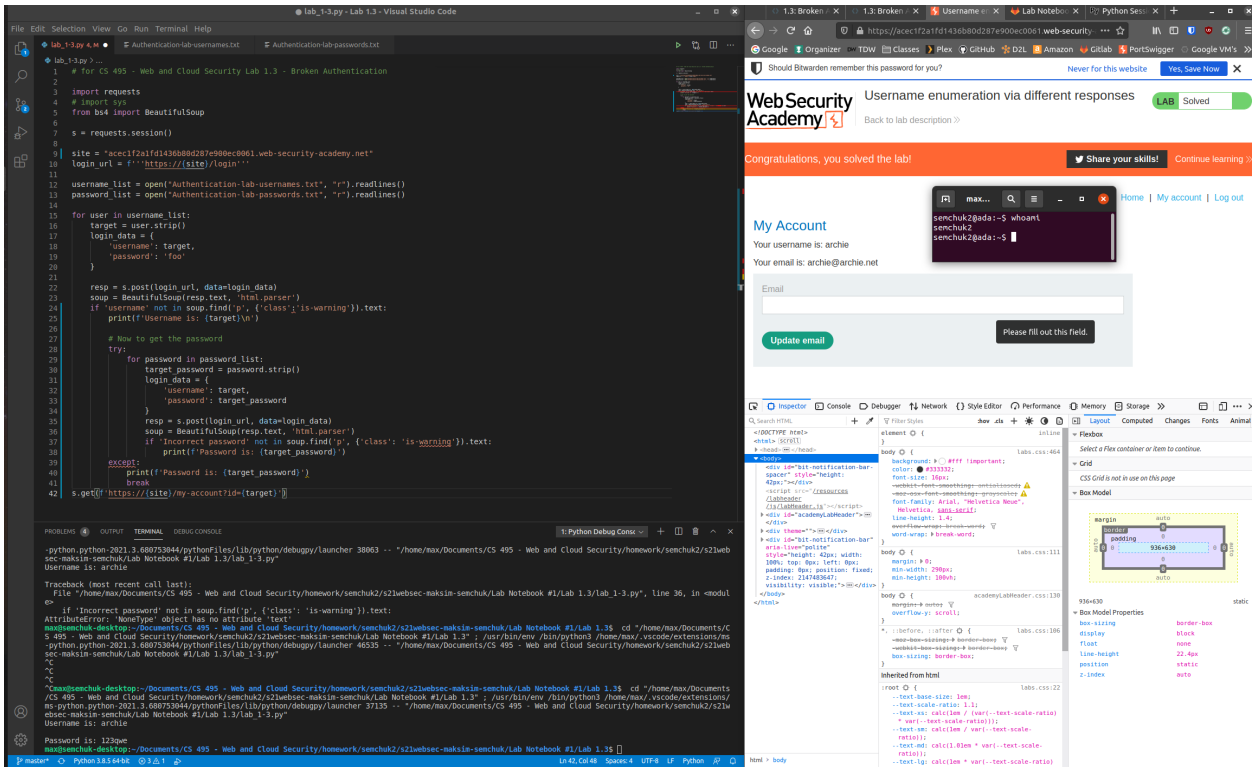


Shows the submitted username and password portion of the lab.

Step 3:

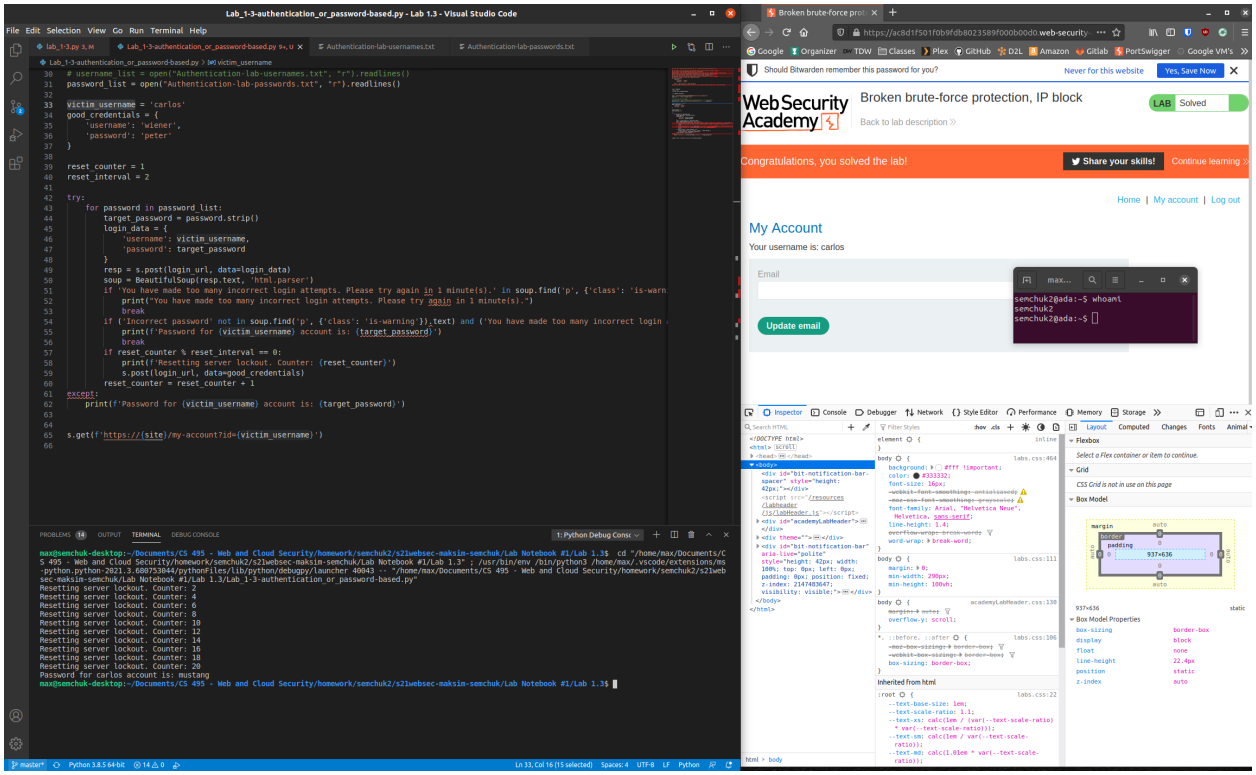


Here, the brute force attack worked, it was able to find the username.



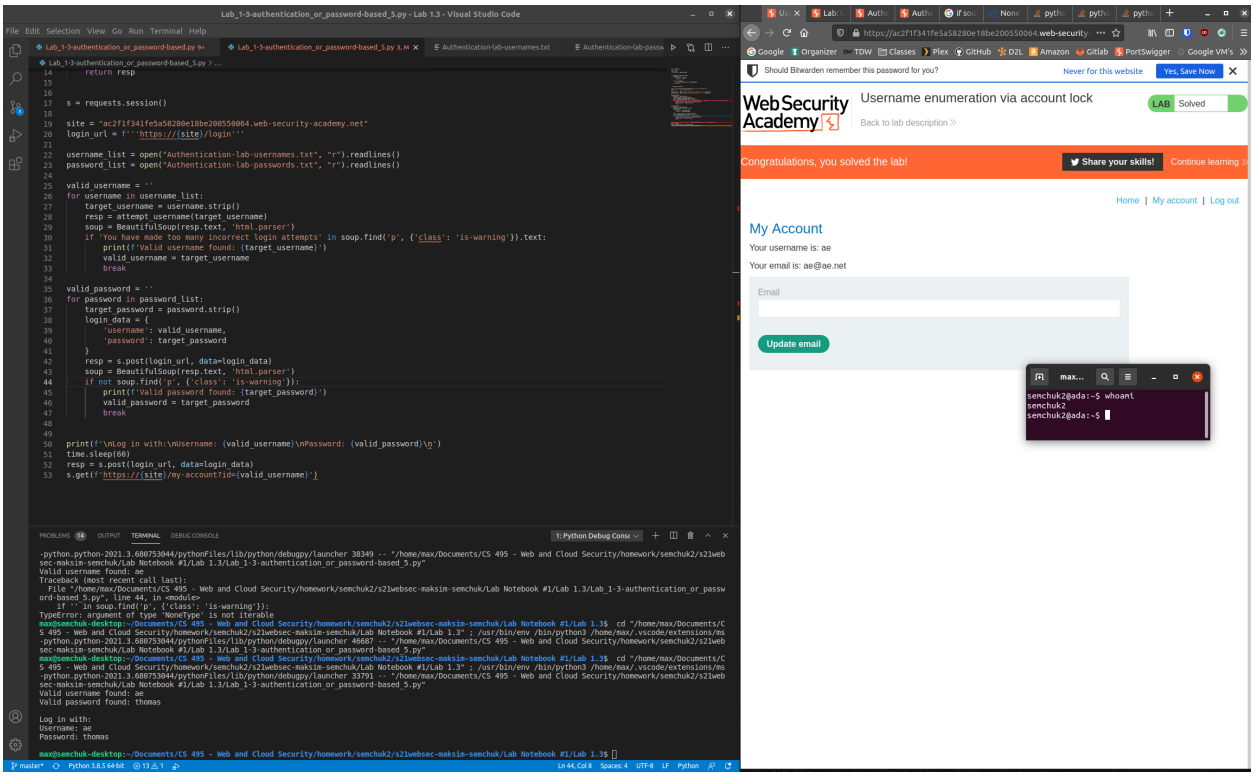
This screenshot shows the username and password I used to log into the user account.
I had to do a try except block to be able to get the password to display.

Step 4: authentication/password-based



This displays that I was able to log into Carlos' account with the password the brute force attack found. To accomplish this, I logged in with the good known credentials every other time, as any more would lock me out.

Step 5:



Here it shows that I was able to log into ae's account using the password thomas as seen in the code.

Write ups for Lab 1.3:

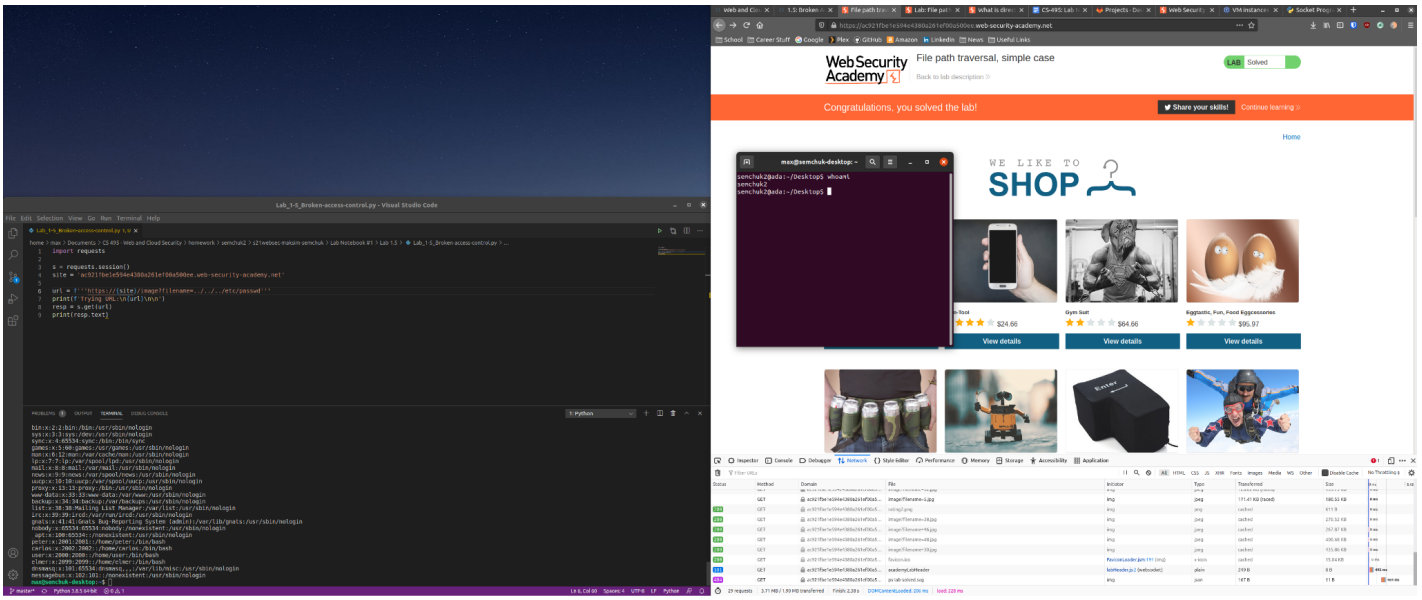
For step #3:
This is a vulnerability comes from the fact that the server has a different response if the username is valid or not, this is bad since a hacker may not even need a list, but rather randomly attempt different usernames and see if they are valid. If they are, then they can begin to test passwords.
This can be remediated by returning the same message to the server, and simply sending a notification of “have you forgotten your username?” with some information in regards to the login like IP and such, since gmail is much more likely to be secure then a random forum where they do not have the same funding google does for security.
The same goes for the password field. It should never say which is wrong, and have the same message of “Error logging in”. This ambiguity at least gets rid of the less talented hackers/script kiddies type of deal.

Step#4:
Here, the account does not lock if an IP logs into a valid user. This is a vulnerability because, just because a person is logging into multiple accounts successfully, should not reset the count for attempted logins into another user. This may even indicate a hacker with leaked identities since it is not often that a user is using multiple different peoples accounts on a single device, especially a cell phone etc.
This can be remediated, by simply tracking how many attempted logins have been made to an account. If the number is exceeded, then lock the account, and each attempt up to a certain amount will be a long and longer lock, after which the account needs to be unlocked by resetting a password by sending the user a text or email on file.

Step #5:
Here, the issue is that if the username is correct, but the password is incorrect, it will return the incorrect code for the password. If this is done several times a return code indicates that the account is real, but the password is incorrect. This is a vulnerability since this allows the hacker to use a list to quickly go through the password list from a leaked database to then find out which of the credentials work. All values should be the same return types, and the user does not need to be notified via browser, rather via email, text, push notification. If a new IP is logging in, then it should lock the account after several attempts to log in, as that will more then likely be a hacking attempt. Combatting this, is easy. Do not send different responses to the user. Always send the same ones, as this could tip off a hacker.

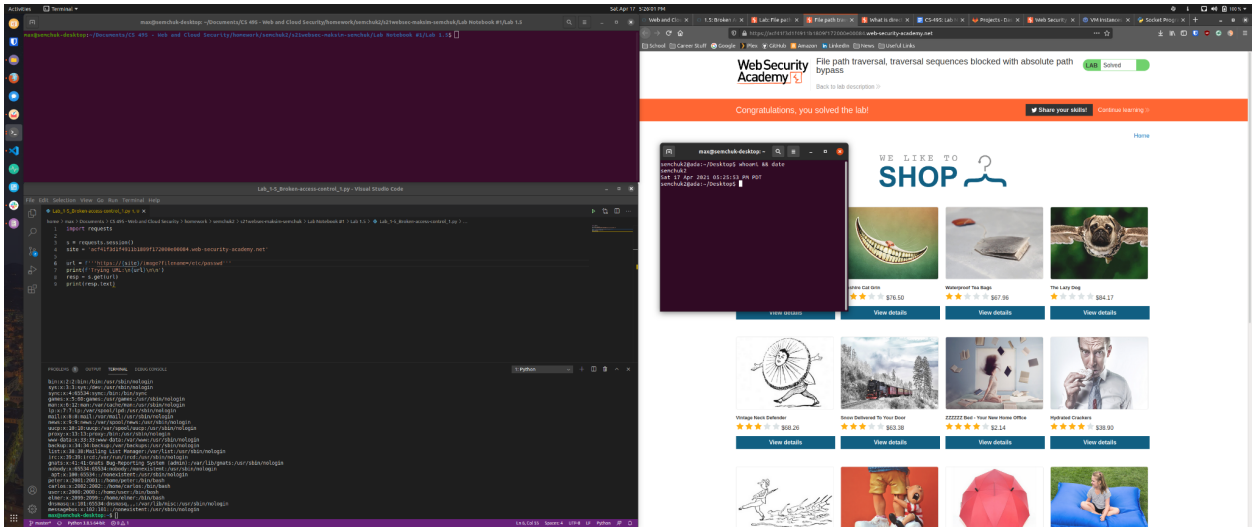
Lab 1.5: Broken Access Control

simple:



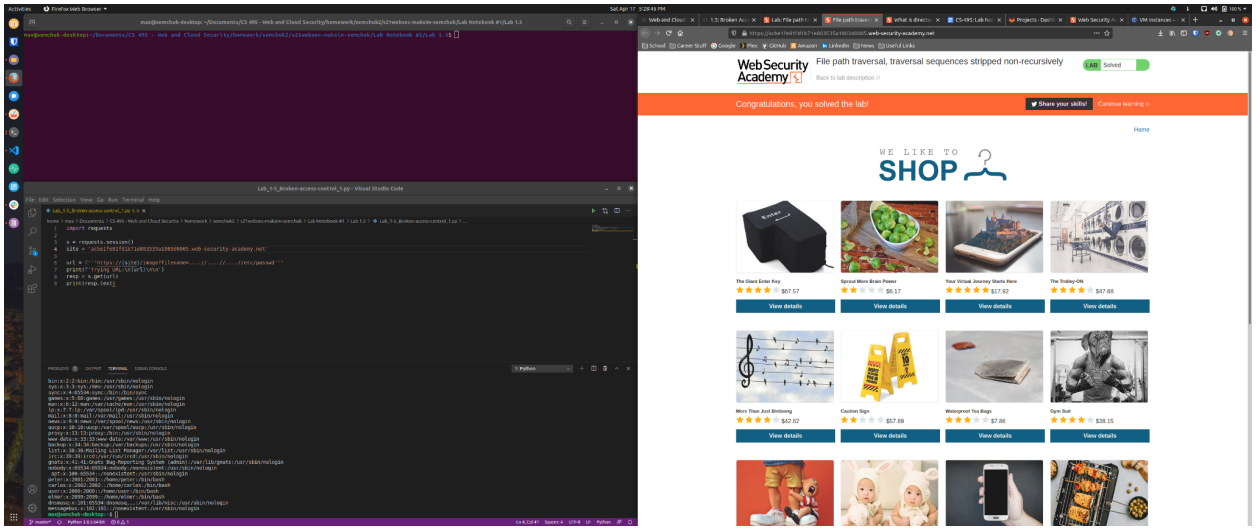
This screenshot includes the output of the /etc/passwd file in vscode, and the logged in to PSU servers with my username. The lab page shows that I was able to complete the lab, as well as the snippet of code.

absolute-path-bypass:



This screenshot shows that I was able to simply provide the file name to the URL and was able to solve the level. This completely negates the traversal process and goes from root.

sequences-stripped-non-recursively:

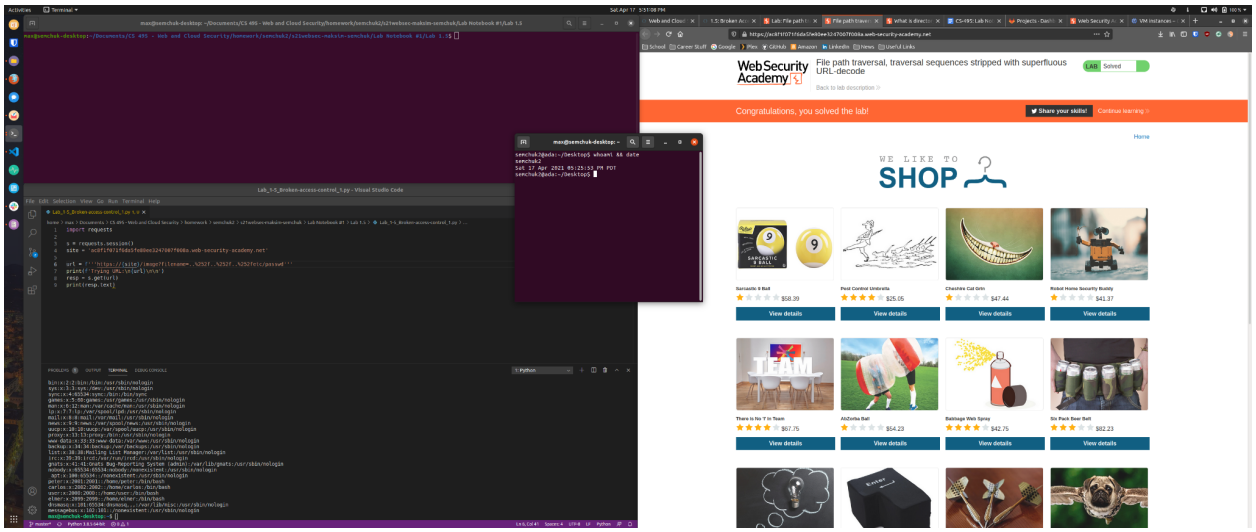


In this screenshot, I was able to get the /etc/password file by bypassing the stripping. It seems that the vulnerability is through the fact that the pattern system only gets rid of ../ first, and then combines the rest, assuming it is safe. This is not true since what is left over is: ../ which is still unsafe and allows traversal.

Lab 1.5.1 Write Up:

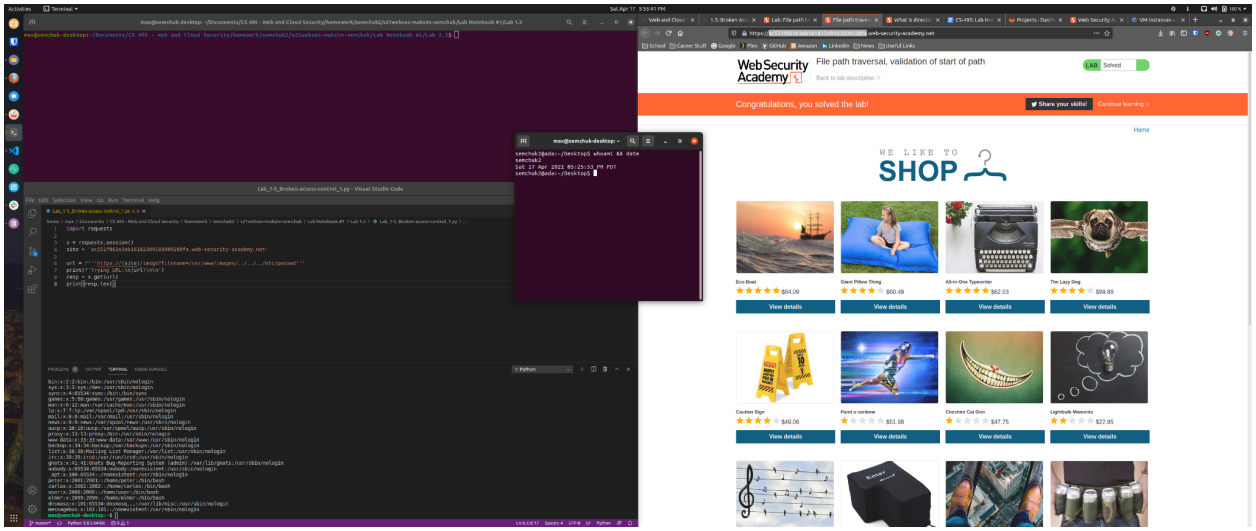
These vulnerabilities come from the fact that they do not have any way to check if the URL being passed in is not in fact checked for if the URL can be traversed in the server. This is a huge vulnerability, as it basically allows a user to move up and down the directory structure and get whatever they want, like passwords, and api keys.
This can be protected from completely, as the server can implement a regex to go over the input url to ensure that it does not contain any movements. Another thing to be done is to simply take in everything as a string and not really run it as code inside the server. Another way is to separate out each request into a sandbox, though this is heavy.

2.
Superfluous-url-decode:



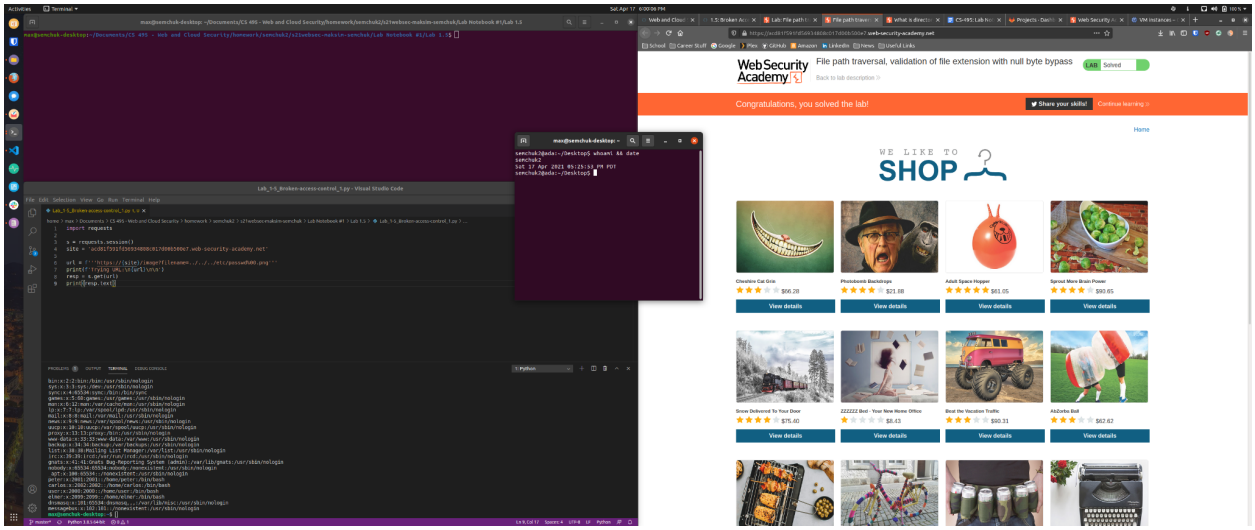
Here, I was able to get the etc/passwd by using: ..%252f , which is an issue with the way the browser handles encoding and decoding, leading to a vulnerability.

Validate-start-of-path:



The vulnerability comes from the fact that it is assumed that nothing malicious can come after the good path to the images. This is a wrong way to validate as the complete path should be validated and not just the start.

Validate-file-extension-null-byte-bypass:

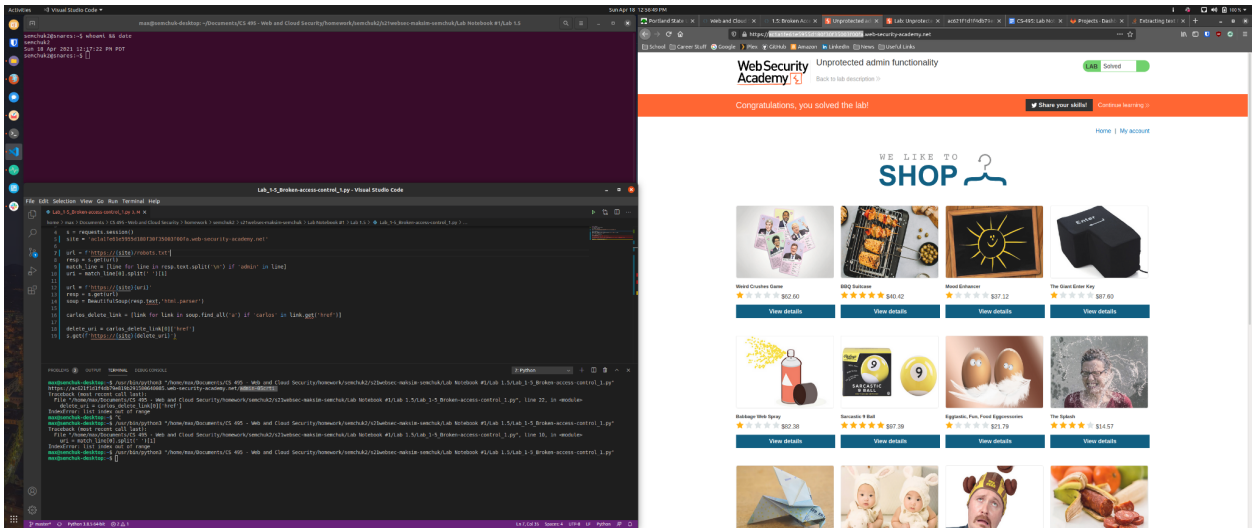


This is another vulnerability that only validates that the file ended in the permitted way, but does not check the encoding and decoding. This is easy to bypass due to that fact.

Lab 1.5.2 Write Up:

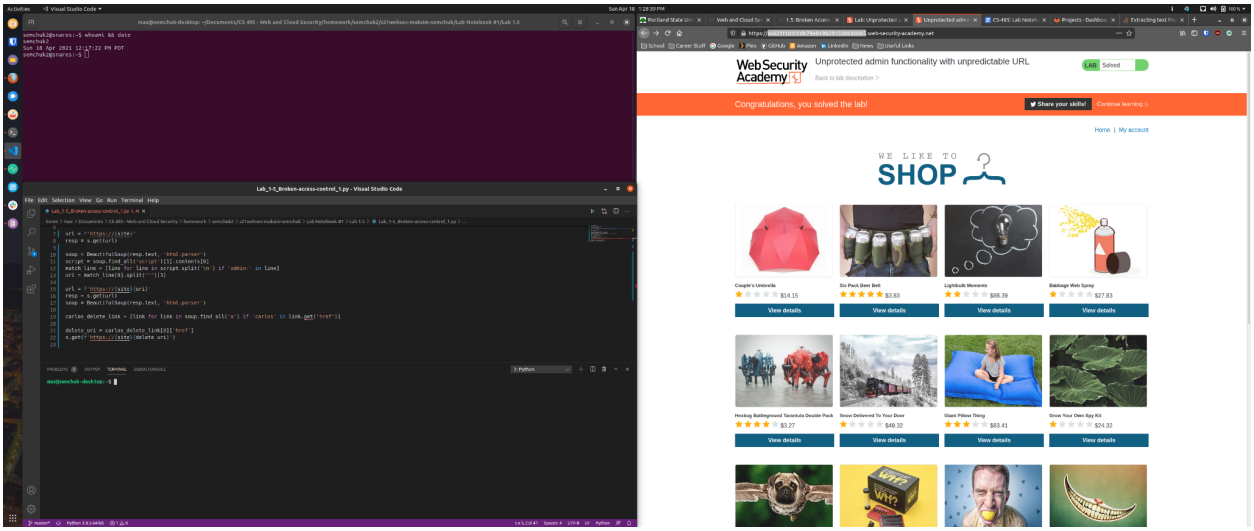
These labs have a weakness from url encoding and decoding and the way the server sees that. When a server gets an encoded URL, it then may again encode/decode it, and that leads to unexpected results. This can be fixed by doing a regex on the code to figure out if is encoded, then it needs to be decoded but only in that spot as otherwise it will do unexpected things in the URL upon decode.
It can be also used to ensure that only certain users are able to get to certain parts of the directory structure. This is a huge issue since otherwise unrestricted access could be had for anyone just by messing with the URL.
THe same applies to the null byte, as things like this should always be filtered out unless they are occupied by valid areas otherwise.

3.
unprotected-admin-functionality:



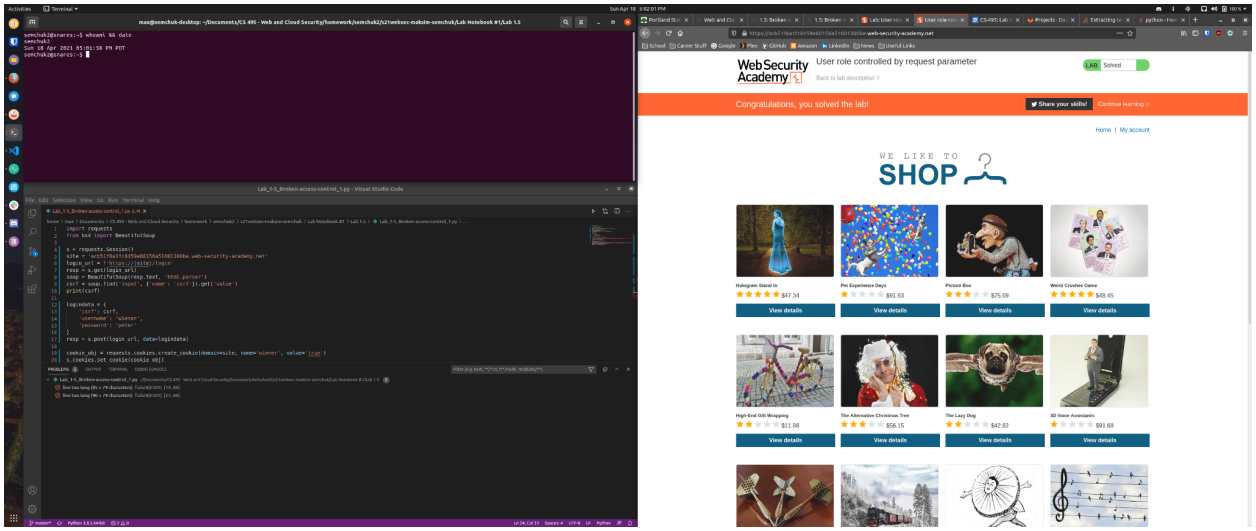
Here the panel does not check for the input of admin into the command line. Obviously dangerous as the admin could be locked out in the best case.

unprotected-admin-functionality-with-unpredictable-url:



This demonstrates that security through obscurity does NOT in fact work. This has a JS on the page that will check for if there is an admin user using it, and you can get the hidden path to the directory folder and delete users.

User-role-controlled-by-request-parameter:



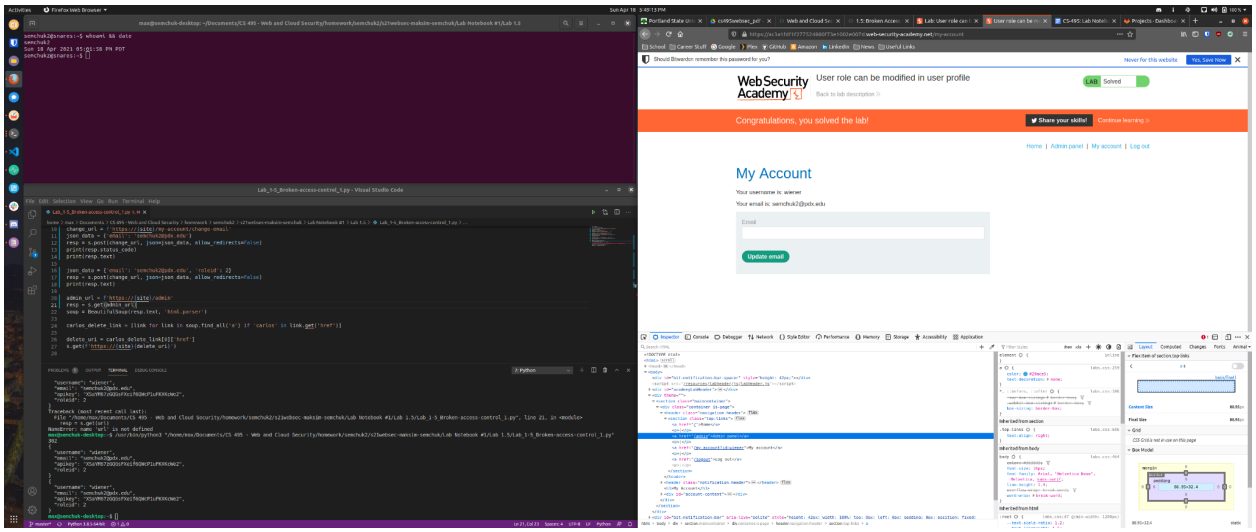
This vulnerability comes from the fact that the server accepts modified cookies and trusts the user will NOT tamper with the cookies.

Lab 1.5.3 Write Up:

The vulnerability here comes from the fact that users are able to get access to the admin areas of the directory structure, and this allows the attacker to get into the administrator panel and either download or delete users.This is a huge issue since if a regular users then makes himself a admin, or gains access to an admin panel.

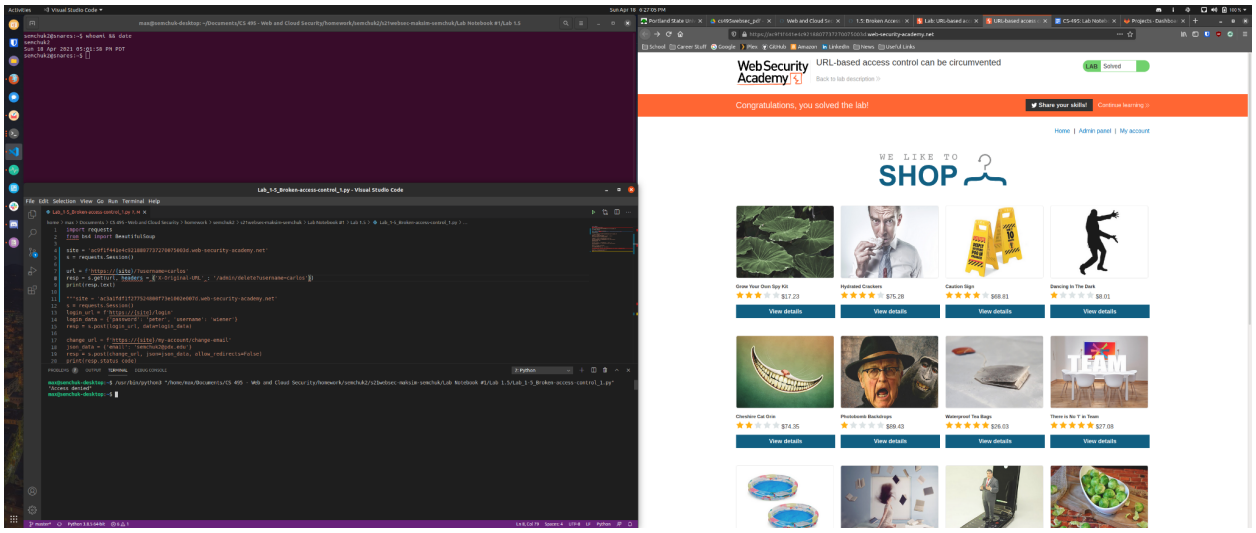
This can be fixed through explicit monitoring of admin containing urls and getting rid of them, unless properly authenticated prior. The address could also be banned if it attempts to access restricted systems by editing urls or other such areas of the system.

5.
User-role-can-be-modified-in-user-profile:

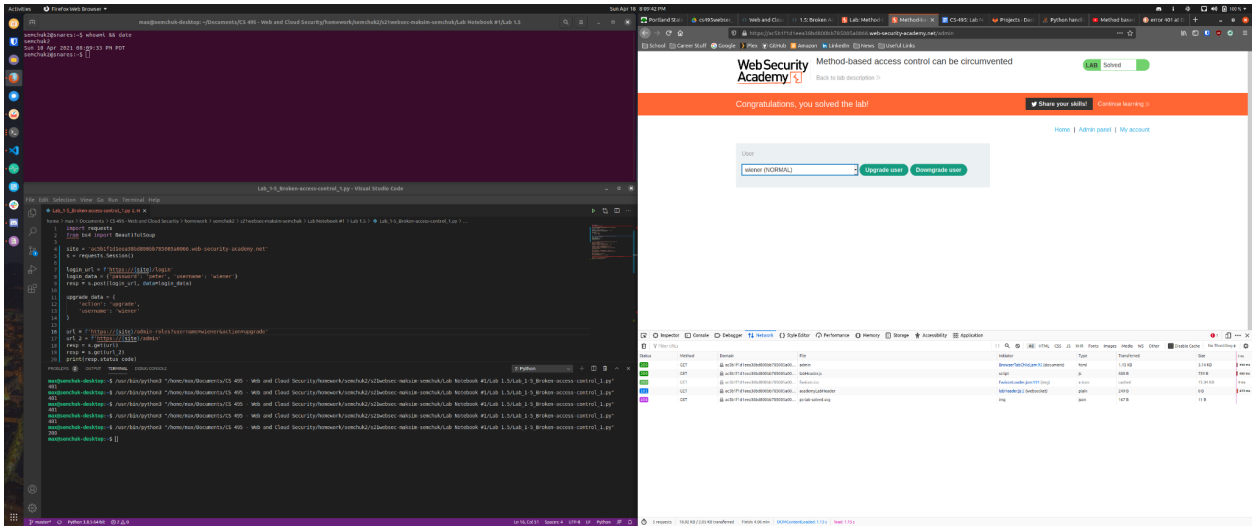


This lab has a vulnerability from the fact that it allows the user to set certain things during the changing of emails or other various operations without checking from the user.

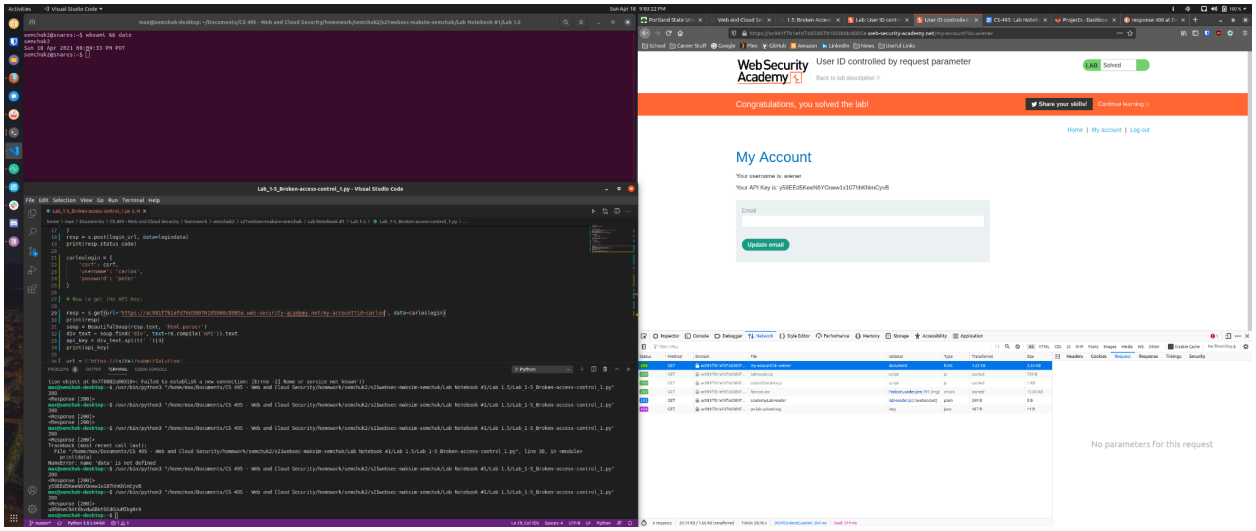
6.
URL-based access control can be circumvented:



Method-based-access-control-can-be-circumvented:

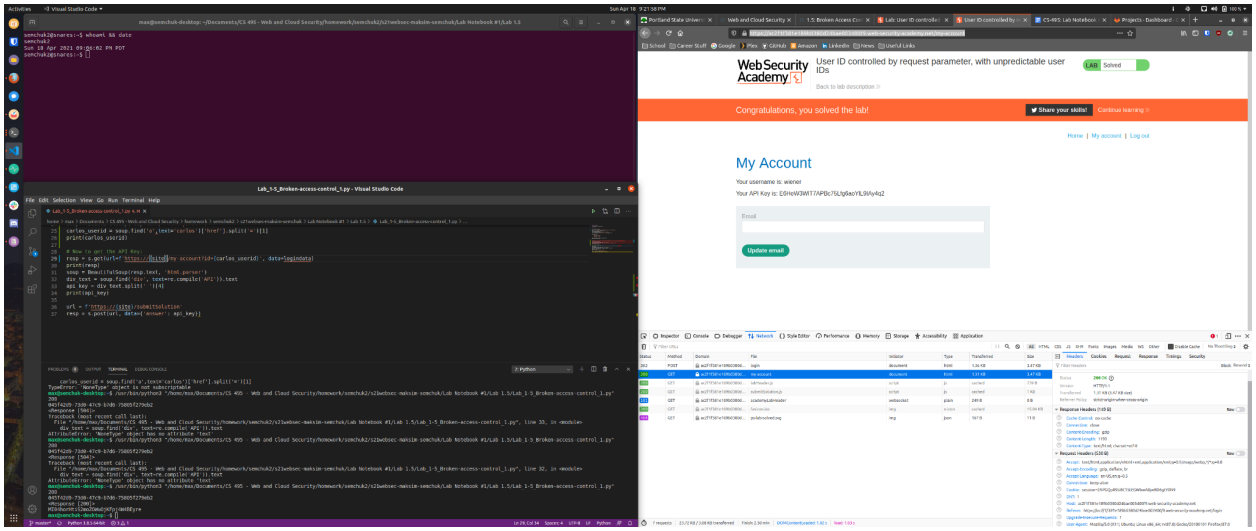


7.
User-id-controlled-by-request-parameter:



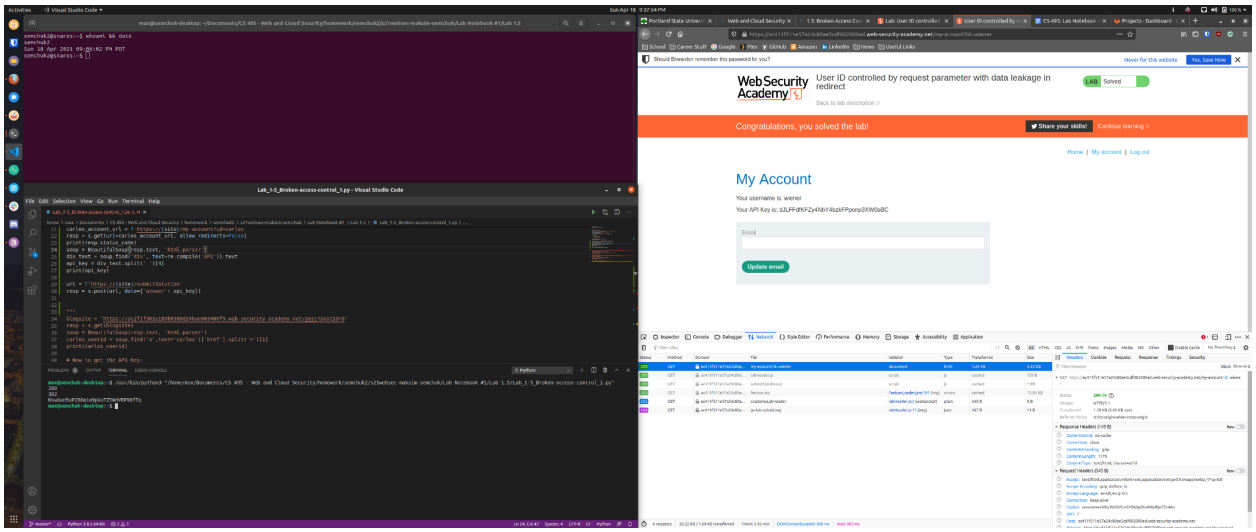
Here, you are able to log in as a valid user, and then you are allowed to modify the URL.

user-id-controlled-by-request-parameter-with-unpredictable-user-ids:



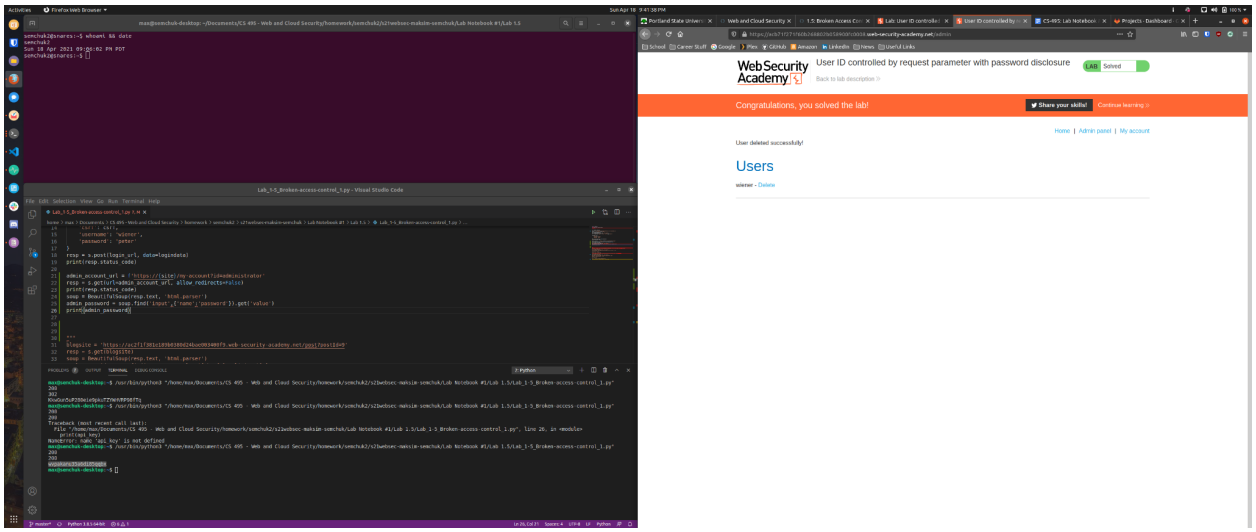
This allows you to get a users ID by looking at their randomized username, then log in using that random username and get the API key.

8.
User-id-controlled-by-request-parameter-with-data-leakage-in-redirect:

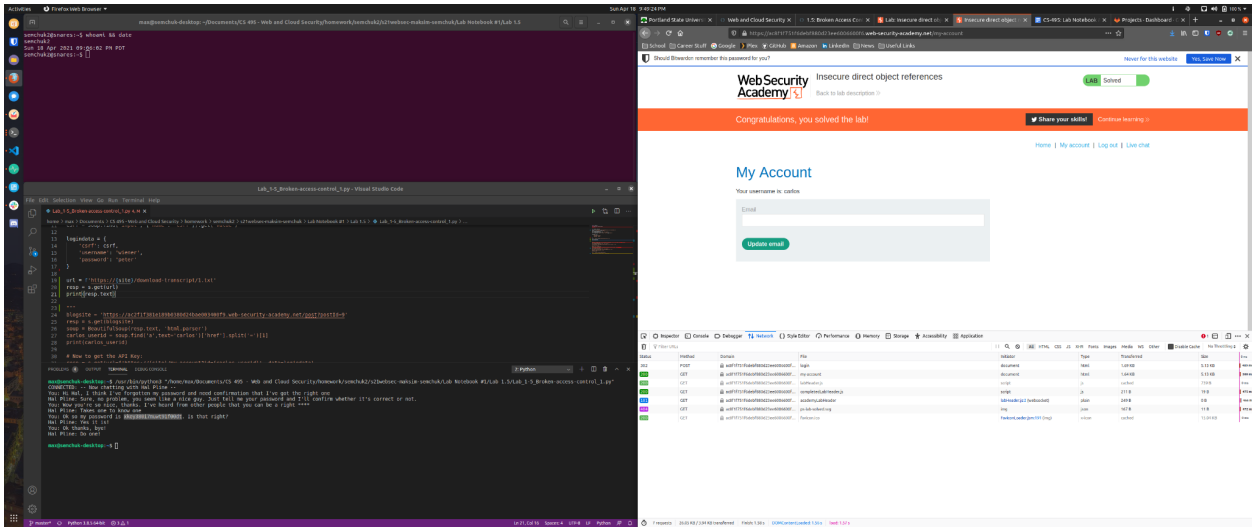


Here, we are again able to go through a redirect since the PHP code does not exit, but rather continues execution. This is a weakness in PHP.

user-id-controlled-by-request-parameter-with-password-disclosure:

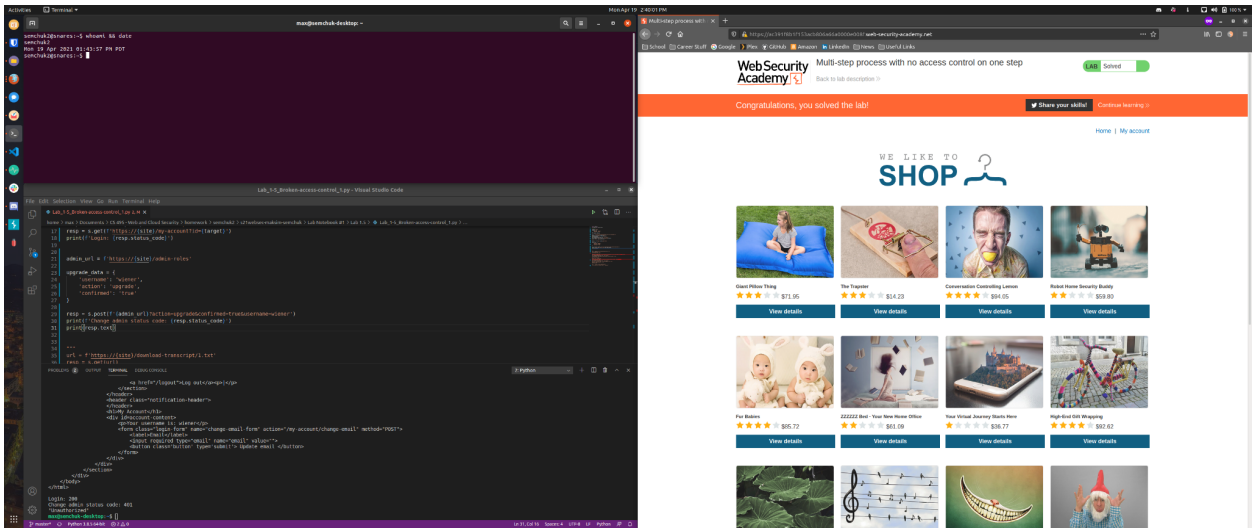


insecure-direct-object-references:

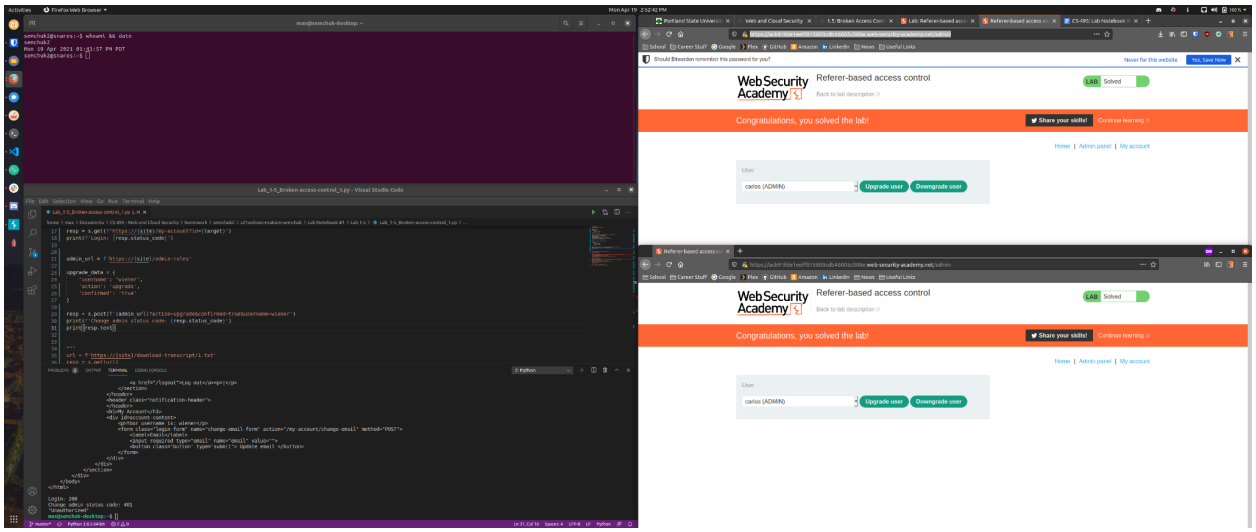


This vulnerability comes from the fact that anyone is able to download any chat, and the user is able to specify which chat, and no checking is done to ensure it is correct and belongs.

Multi-step process with no access control on one step:

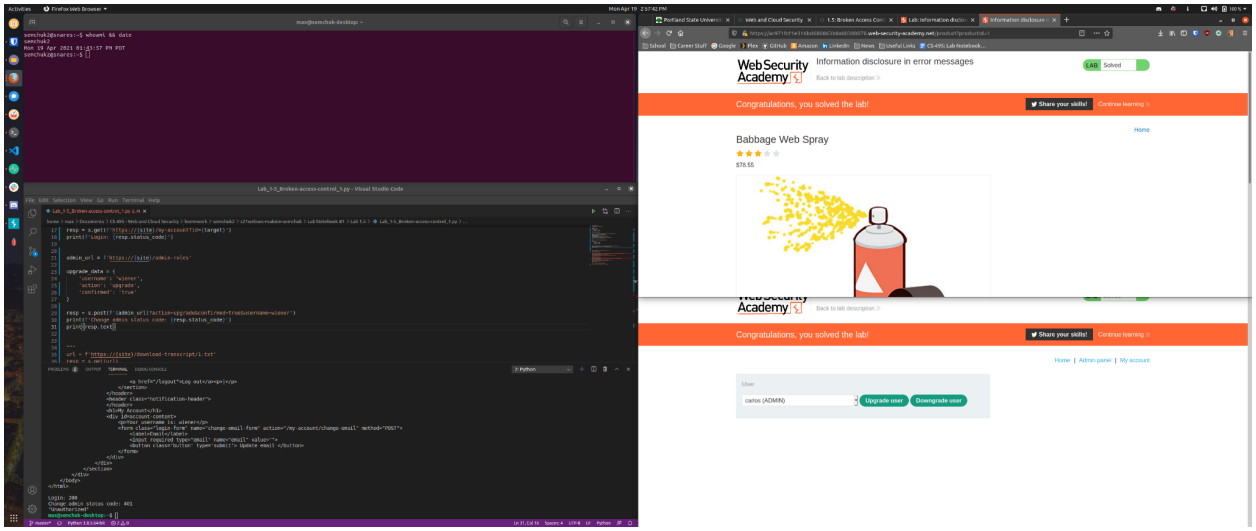


referer-based-access-control:

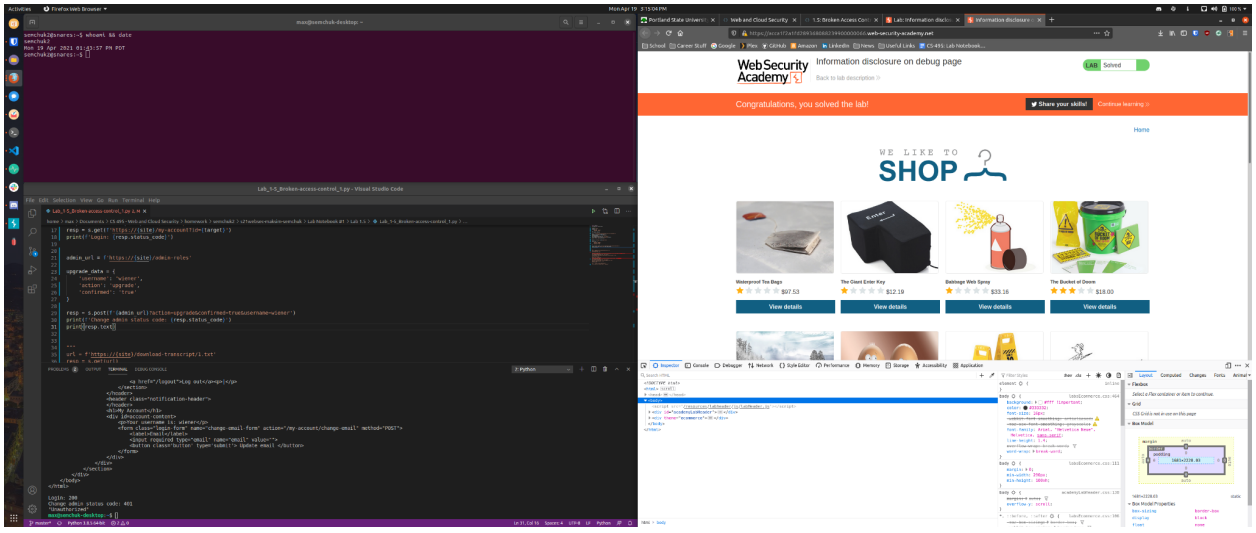


#10

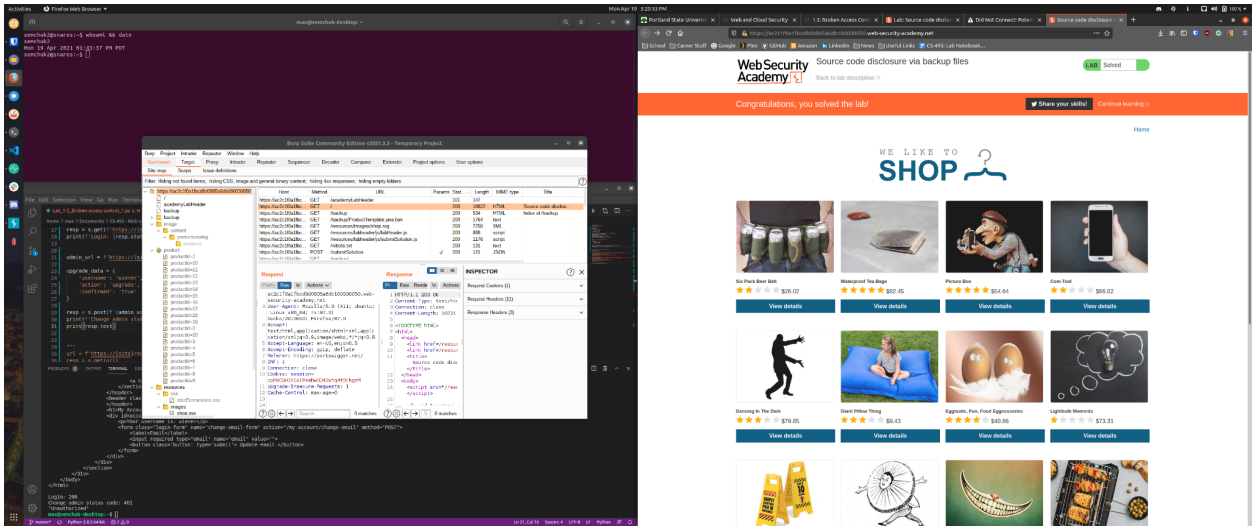
Lab-infoleak-in-error-messages:



Lab-infoleak-on-debug-page:



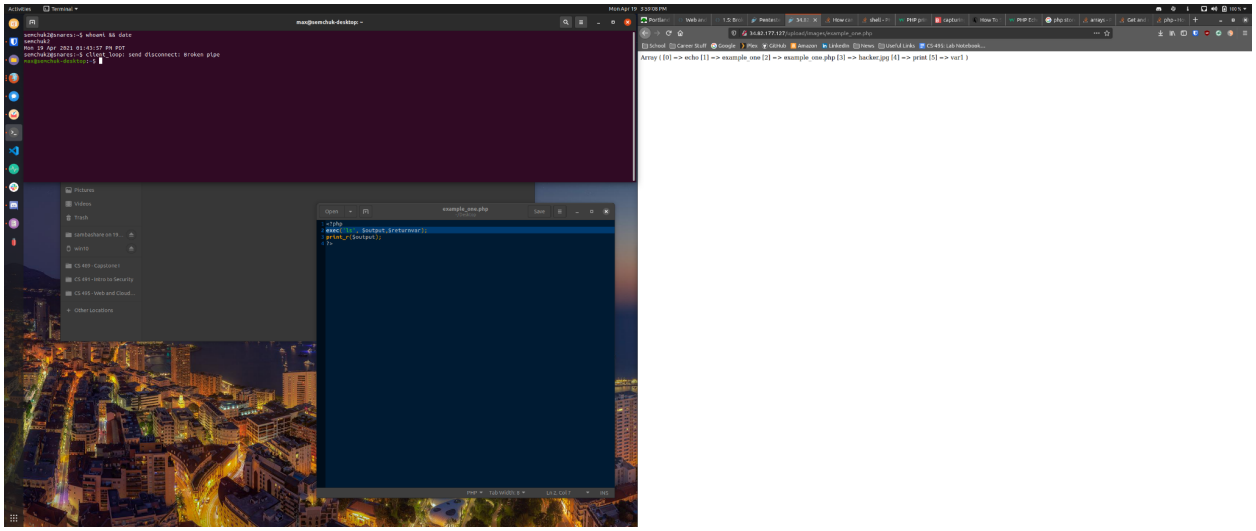
Lab-infoleak-via-backup-files:



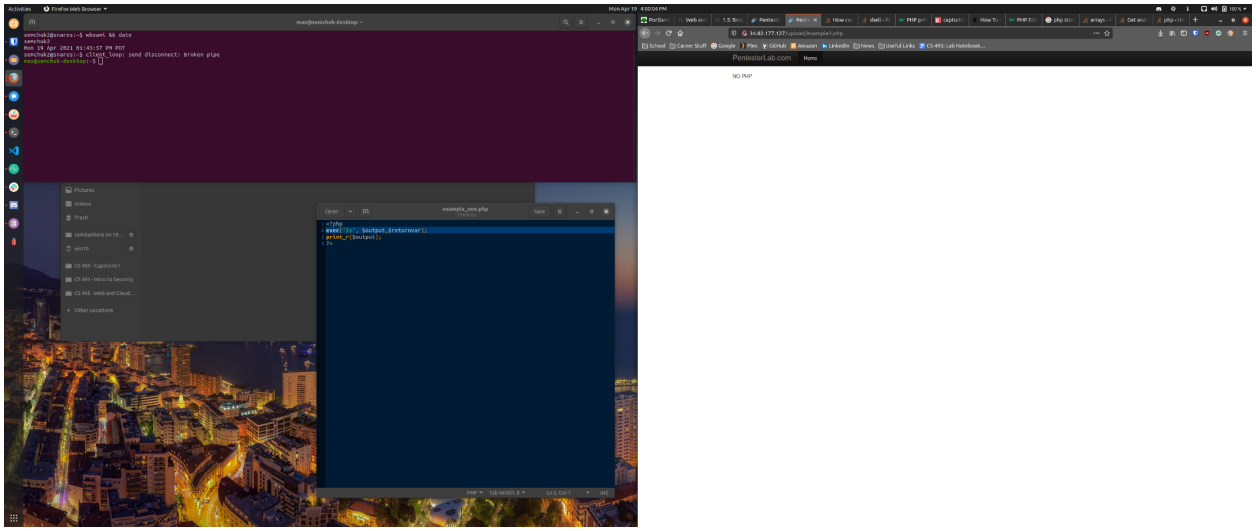
11.

WFP1: File upload:

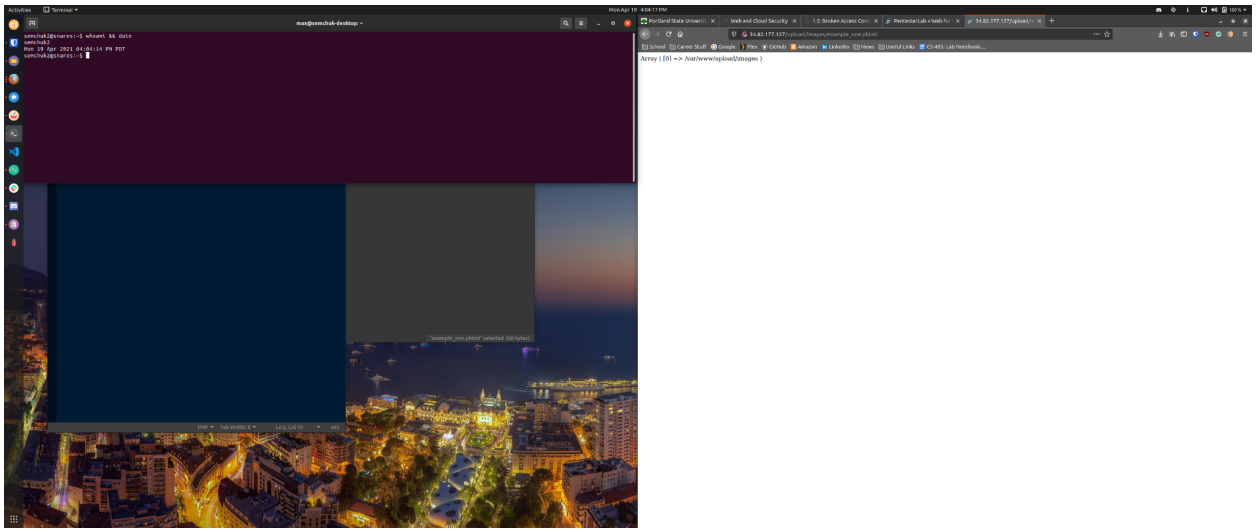
Example 1:



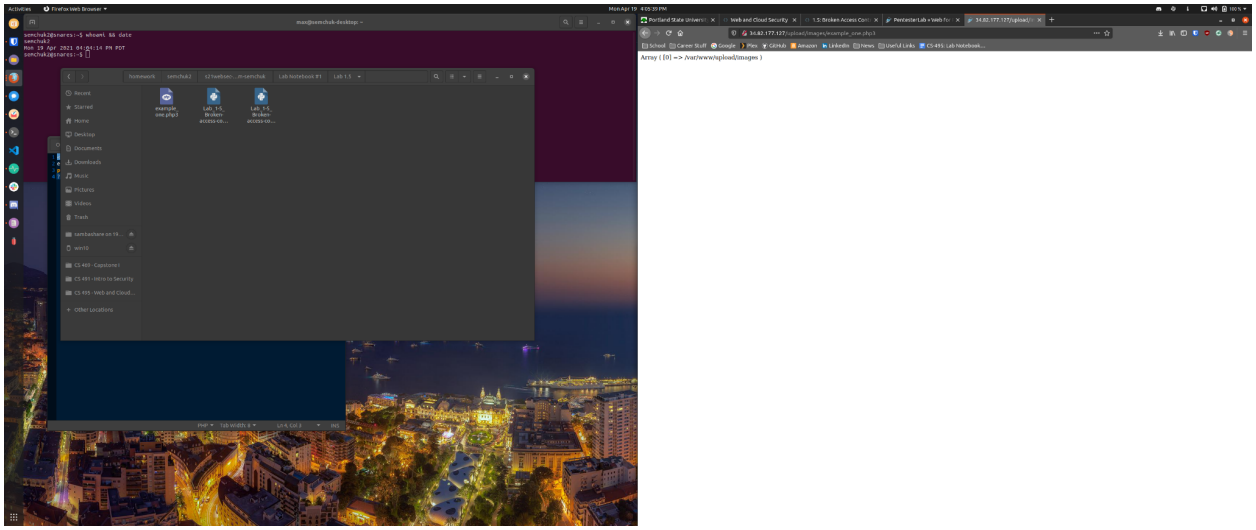
Example 2:



Php did NOT pass



Phtml passed



Php3 passed

Labs 1.5.5-1.5.11 write up:

For #5: This is called by editing the crcf part of the login data script. This allows getting access to the user account by changing the email information. This is really dangerous as it exposes the email, and the user account to do whatever the attacker wants. This is a huge issue since you can get whatever user account using this method. This means the the user can change to whatever user account to whatever email they want, even automating the process.
This can be addressed by making sure the crcf tokens are not modified by the user by comparing all data fields within the crcf.

For #6: With CDN's and firewards you have to make sure that the URL is not modified and that the user does not have a modified url from the original with CDN's. This allows the user to redirect to wherever they want by putting through various ways for the user to get into whatever system they want.
This can be fixed by not allowing redirects through areas where they were not allowed or redirected from the area where they were originally going to. This is implemented through the cdns and servers. They have to make sure that the redirect is valid and the request is valid as well. The crcf tokens have to also match the ones that were received and sent.

For #7: So when a user logs in properly, and get authenticated there must be measures to prevent one authenticated user from getting access to another authenticated person's account. This lab has the vulnerability of not protecting the user accounts. The randomized user ids are also pointless if they are not protected. The way a user gets access to someone else's account is getting the API key, which is unique to every account. This can be prevented by not allowing users who have differing API's keys. The weakness is the ability to switch the user account usernames to be entered and gotten from the user. The user can then get the API key to gain access to the account.

For #8: Redirects can leak data if they are not done properly. This allows for potential weaknesses like redirects happening to places for unauthorized users. When this occurs the user can specify which account they want to go to. This can be fixed by allowing redirects from authorized sessions. The other way this can be fixed is by making sure that certain redirects only go to certain areas. Then also making sure that the certificates match up in between redirects.

For #9: altering the url can change the redirect and modify the final redirect. This is vulnerable as it allows someone to change where they get the final redirect from. This can be fixed by making sure that the url's addresses are not altered by making sure the server and redirect CDN are talking to each other and only sending URL's. This is a huge deal since it allows the users to potentially get access to many users and the redirect servers logs as well.

For #10: information leaks from modifying certain requests in the request and http gets. The apache framework has issues if you simply input something it does not expect. The other issue

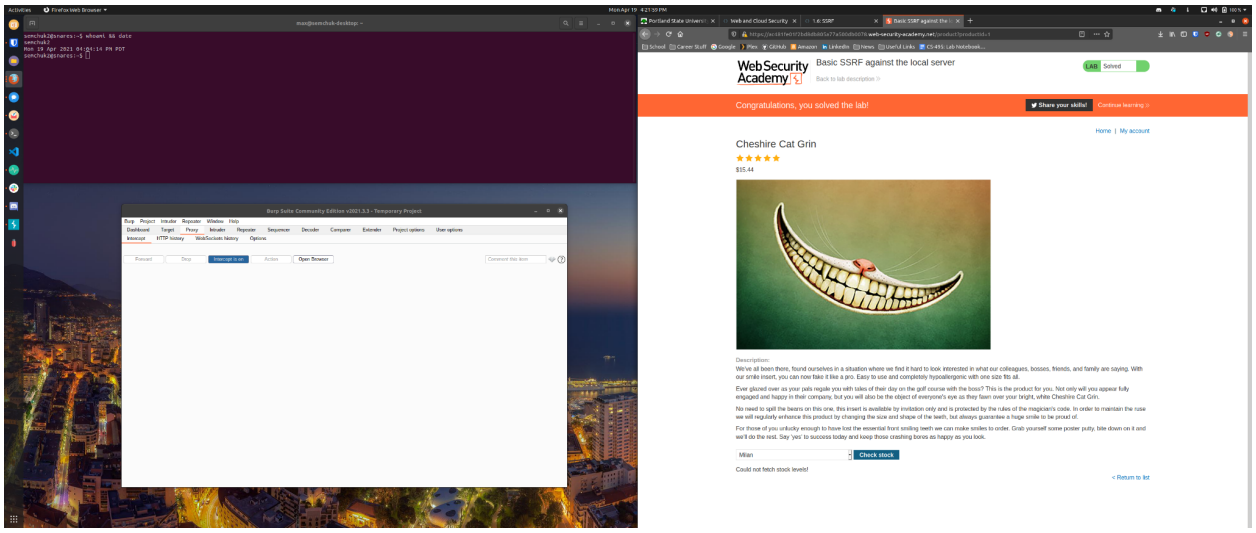
is if there is an insecure php code in the html, and if it can be exposed, it can be located by visiting that directory. This can be countered by making sure they do not appear in the debug area.

For #11: This will exploit a weakness that has embedded php code inside of a picture file and it assumes that that input is good. The expected input is a picture but since it has an embedded code it will then also run that code without verification. The same goes for the second example as well, just it expects a different file extension.

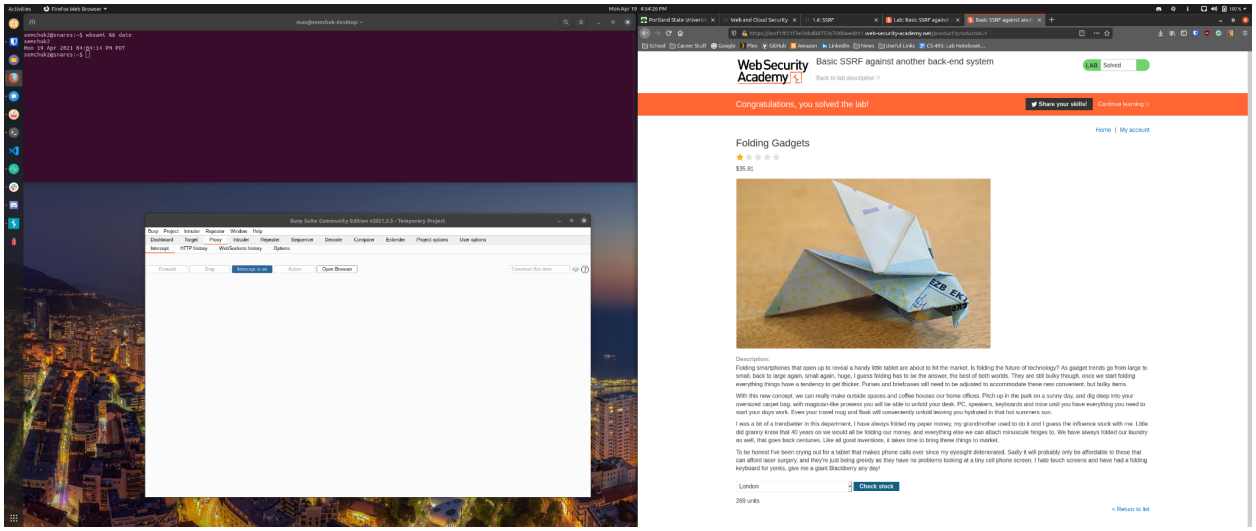
Labs 1.6: SSRF

1.

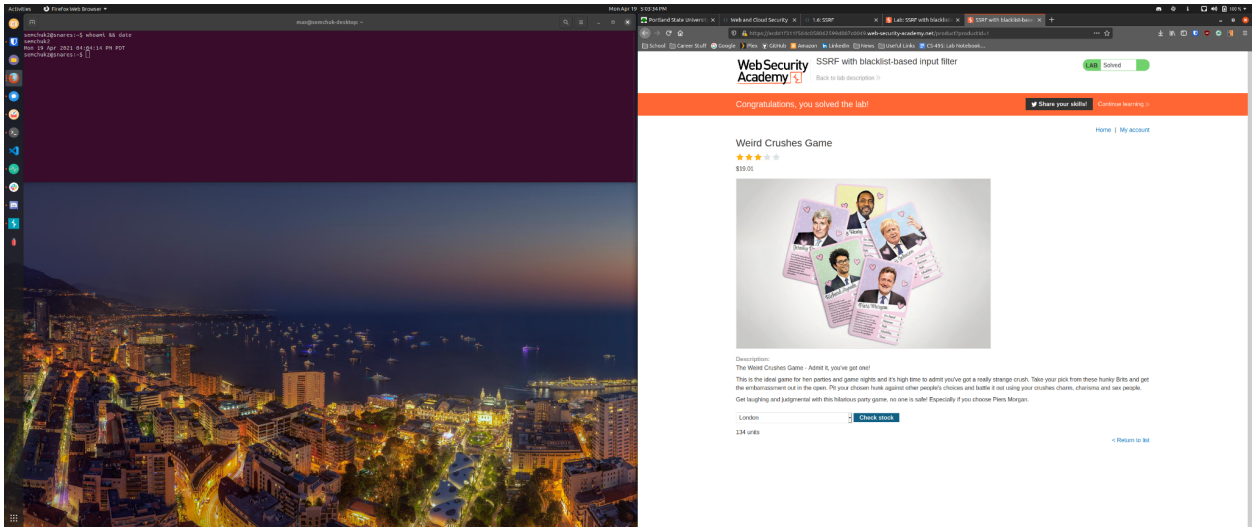
Basic-ssrf-against-localhost:



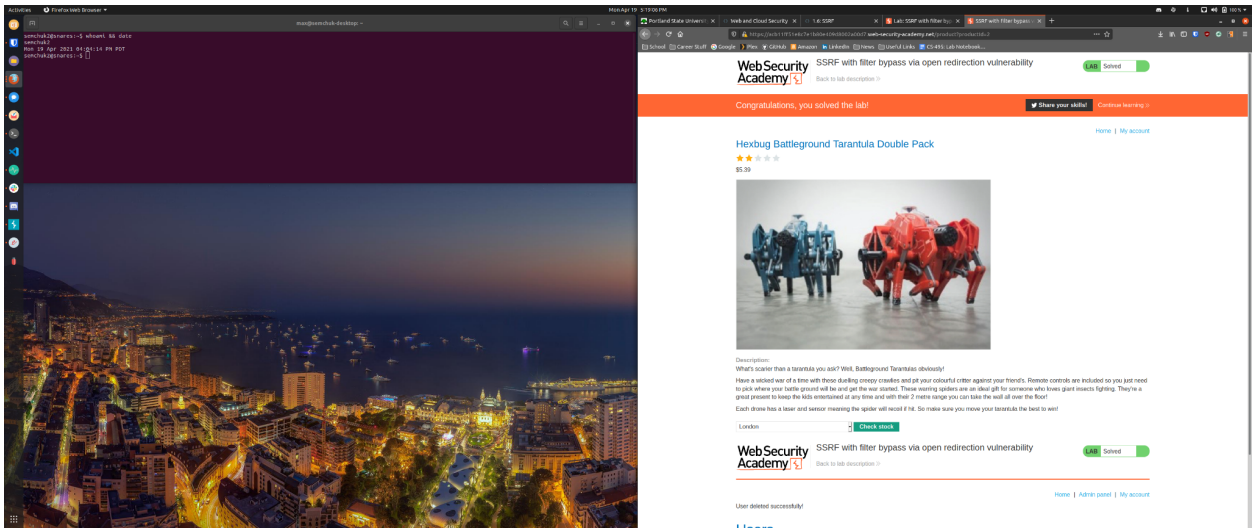
Basic SSRF against another back-end system:



SSRF with blacklist-based input filter:



SSRF with filter bypass via open redirection vulnerability:

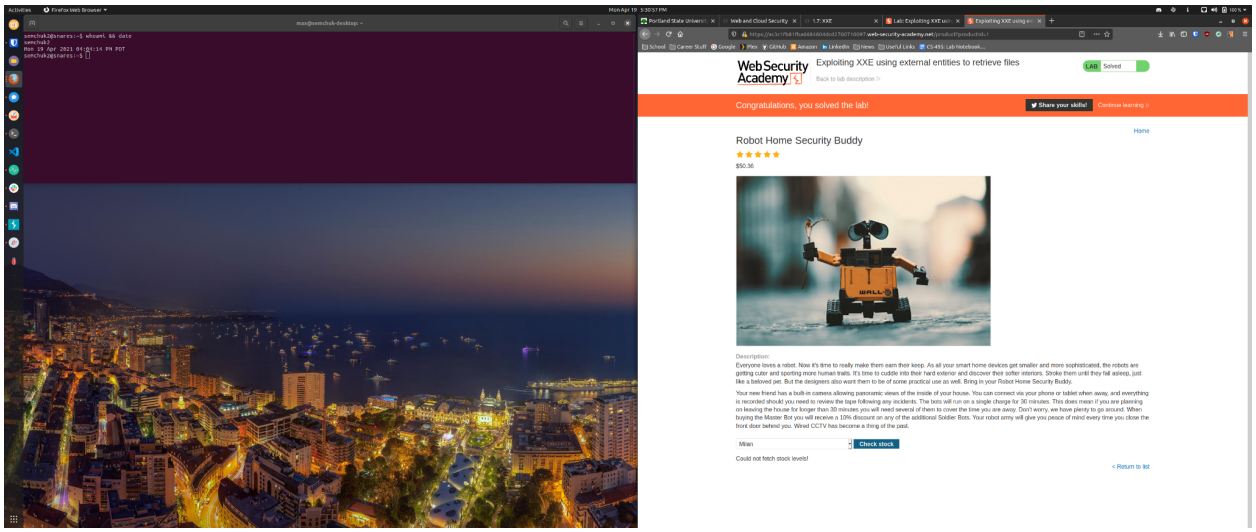


Labs 1.6: SSRF Write Up:

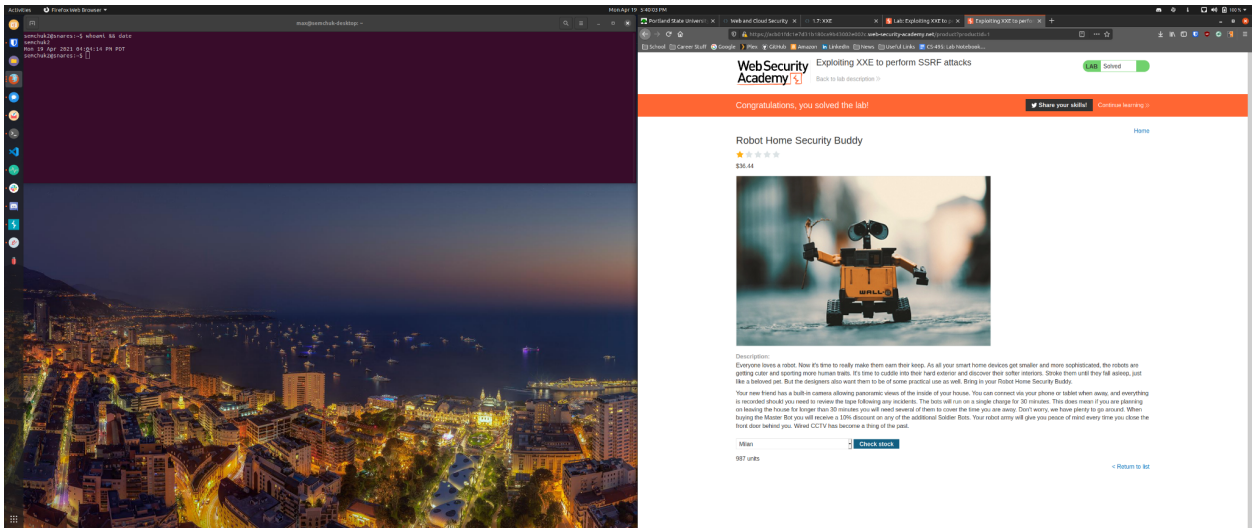
The vulnerability here, shows that if a user is able to get the basic backend information they will be able to modify the path in order to receive additional data. Localhost is a default area that most areas will have. The localhost admin is an area which will modify the user accounts. A remote server should not be able to get access to a localhost area without special permissions.

Labs 1.7: XXE

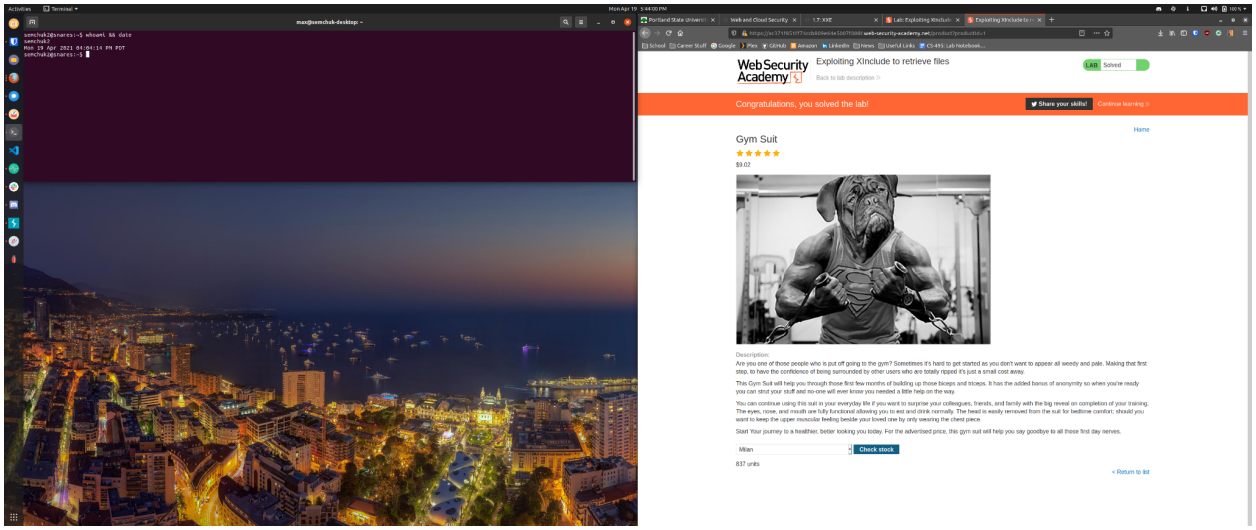
exploiting-xxe-to-retrieve-files:



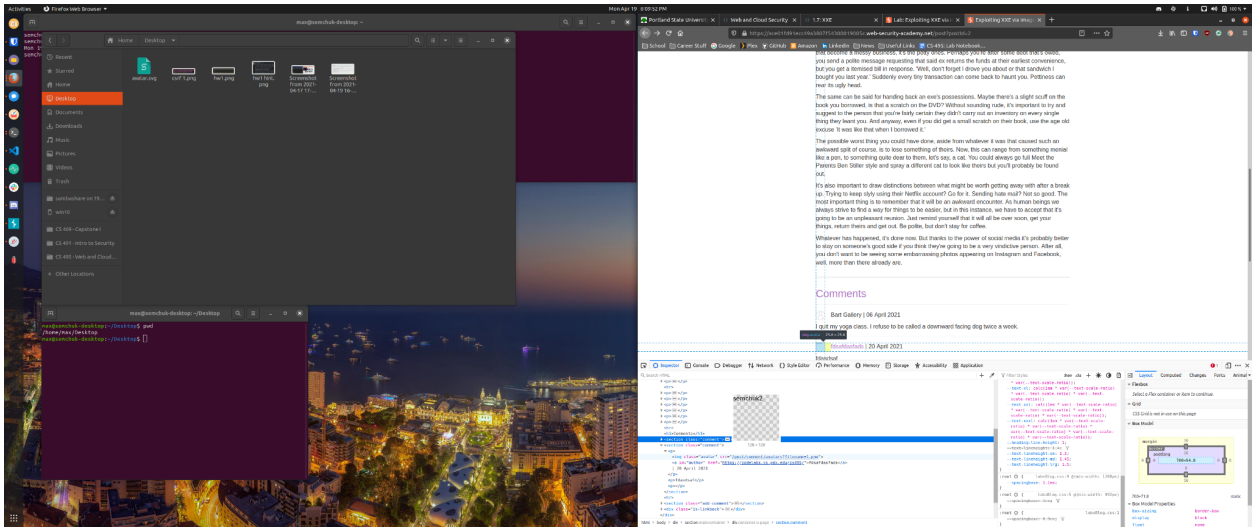
Exploiting XXE to perform SSRF attacks:



Exploiting XInclude to retrieve files:



xxe-via-file-upload:



Labs 1.7: XXE Write Up:

Xml data can be used to exploit the web servers, and this can be used to get both, unrestricted access, and also data on the web server itself. This even goes for insecure AWS - instances. These can be used to exploit bad things inside the server to and to get the server information. This can be countered with, not allowing any user scripts to run on the server. That is to much of a risk and shouldnt ever be allowed.