Programming Project - Damn Vulnerable Web Application

Kai Depweg

December 2024

Contents

1 Setup

Note The fallowing instructions required full screenshots this means that the images are smaller comparative to if I had just taken the necessary info as a screenshot. Please zoom in to the the images.

1.1 Setup Guide

For the setup I followed the installation guide on GitHub provided to us though GitHub (https://github.com/digininja/DVWA)

1.2 Technologies Used

The following technologies were utilized during the setup process:

- **Operating System:** Ubuntu (running in a VirtualBox VM)
- **Virtualization Software:** VirtualBox (to isolate the environment)
- **Web Server:** Apache (provided as part of DVWA setup)
- **Database:** MySQL (used to store DVWA data)
- **Programming Language Support:** PHP (required for DVWA functionality)
- **Tools for Verification:** `tree` command (to display directory structure)
- **Proxy and Packet Interception:** Burp Suite and FoxyProxy (configured for vulnerability testing)

1.3 File setup

The fallowing is screenshots of the file structure.

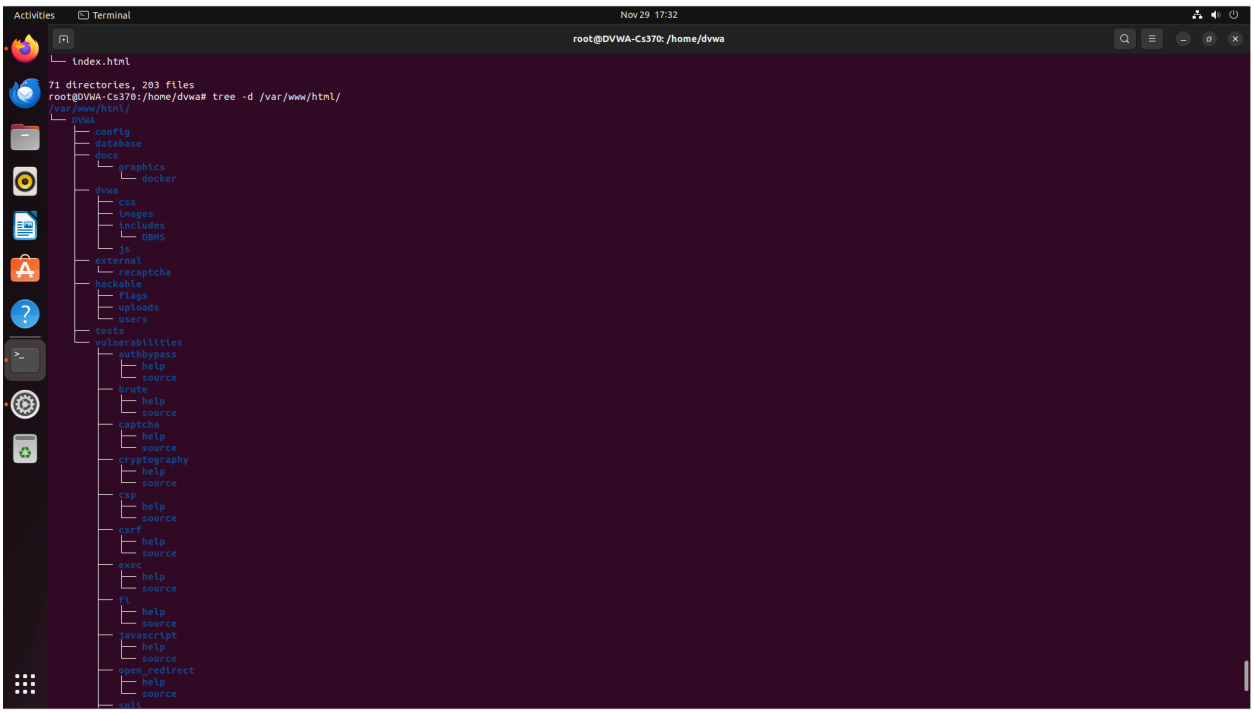

Figure 1: Folder Structure

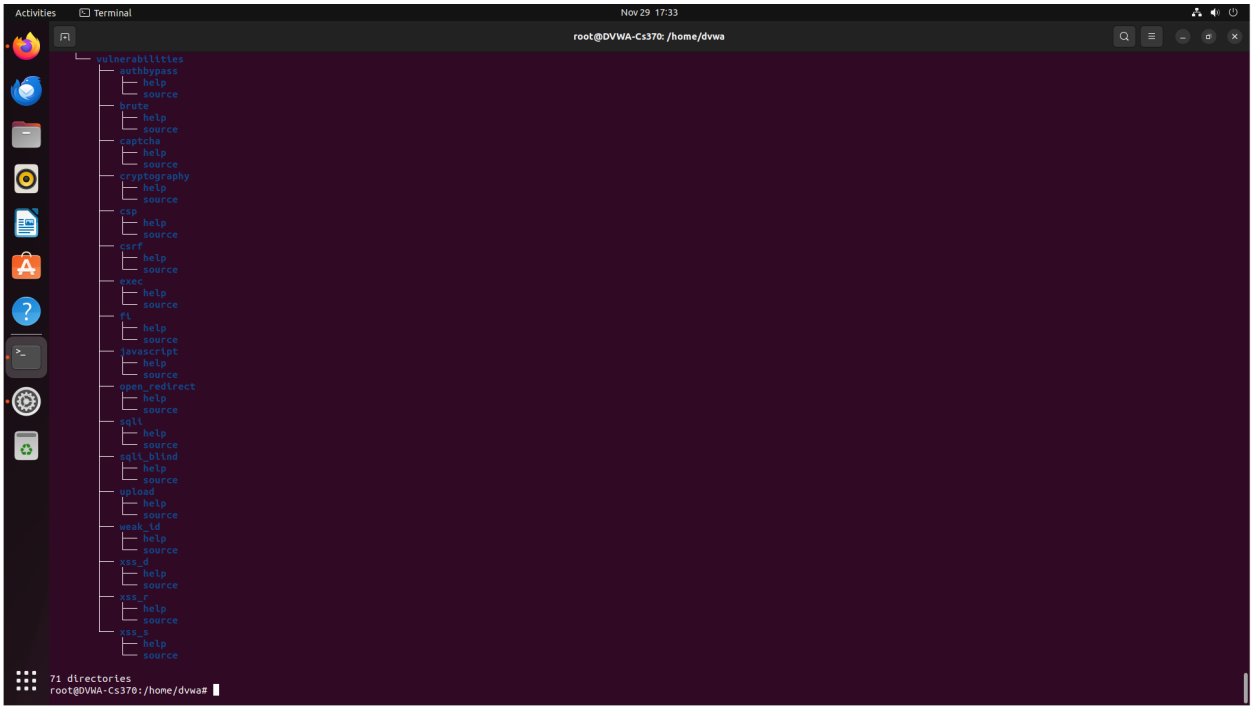

Figure 2: Folder Structure

1.4 In depth setup

The following is the In-depth setup

For the setup I followed the installation guide on GitHub provided to us though GitHub (https://github.com/digininja/DVWA). Due to the guide and canvas recommending an isolated environment I am running a Ubuntu VM through Virtual Box Figure 3). Due to the screen being small the following screenshots will be of the full screen of Ubuntu instance within the Virtual Box.

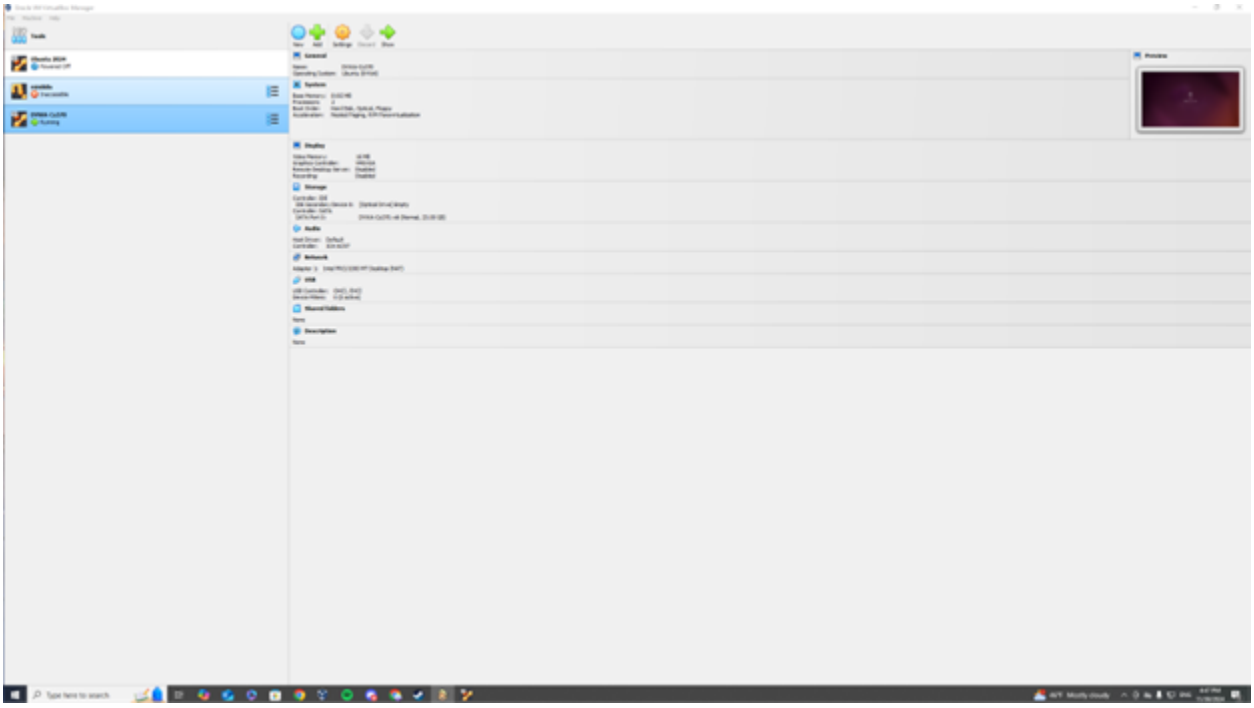

Figure 3: Virtual box

Fallowing the setup guide I opted to run the automated script provided to me within the GitHub page:

```
sudo bash -c "$(curl␣--fail␣--show-error␣--silent␣--location␣\
https://raw.githubusercontent.com/IamCarron/DVWA-Script/main/Install-DVWA.sh)"
```

When executing this, the following setup was ensured and default parameters were set (Figure 4). Additionally, within this step, I installed ‘tree‘ using:

```
sudo apt install tree -y
```

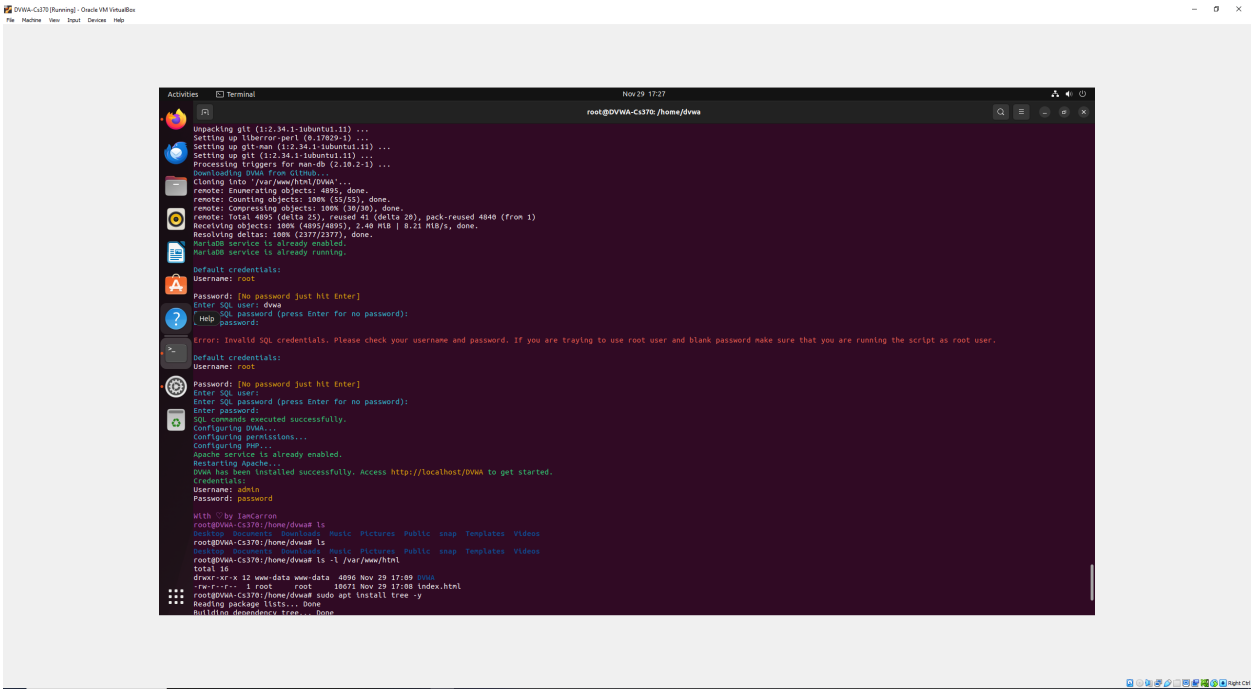

Figure 4: Setup

This resulted a default file structure as seen bellow (Figure 1, Figure 2) when running the command:

```
tree -d /var/www/html/
```

Additionally a proxy and Burp Suite was installed to the VM by fallowing the instructions found in the fallowing link: (How to configure Burp Suite with Firefox FoxyProxy on Kali Linux). This allowed for my stem to be setup with a proxy and a packet intercept system for me to be using in attacks like brute force (Figure 5, Figure 6).

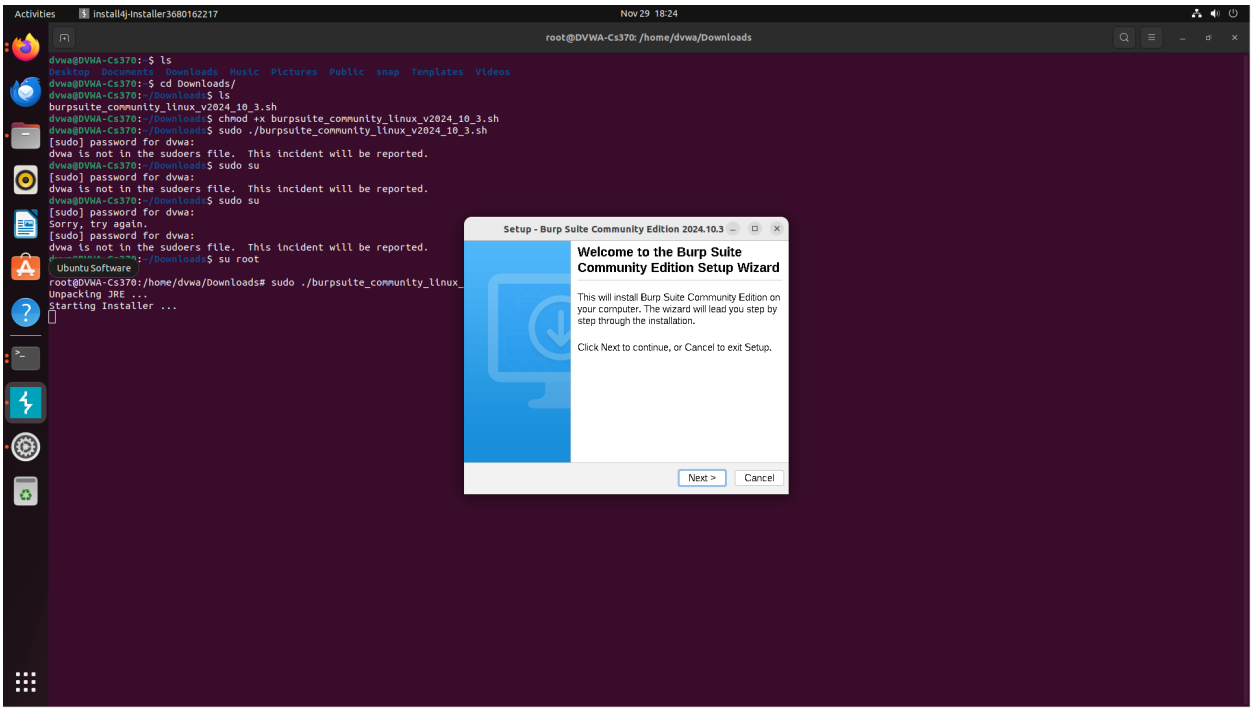

Figure 5: Burp Suite

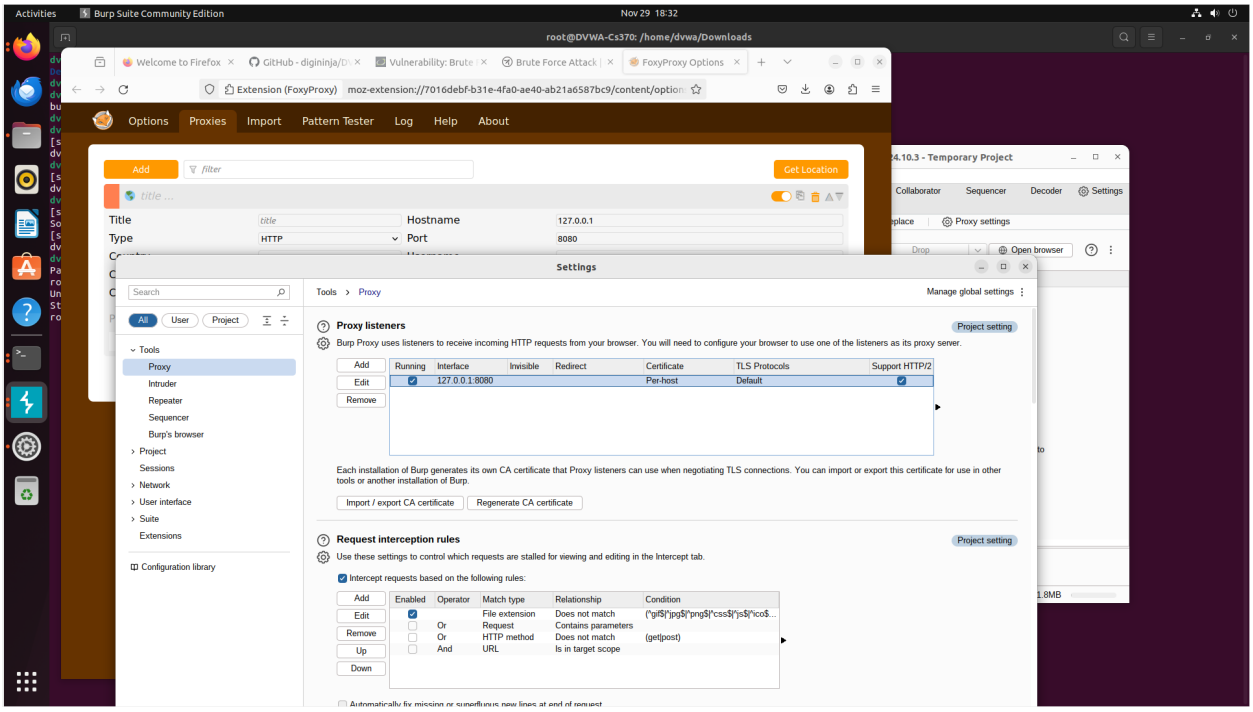

Figure 6: FoxyProxy

2 Brute Force

- **How does this feature normally work?**
  In a normal environment, a login feature securely allows a user to access their account by providing valid credentials. The system then will compare the submitted credentials against a stored database containing your user login info. If they match it grands access to the service if not they send back a access deny message or a timeout upon too many attempts. Systems like these normally had added security like CAPTCHA, login attempt limits and encryption added to them to prevent brute force.

- **What does it take to exercise the vulnerability?**
  It takes a continues number of attempts with differing passwords and usernames until a success is found. It also requires a system to have lacking security to multiple attempt without a lockout mechanism. The use of a proxy and a brute force tool like Burp Suite is convenient but a script can be wrote to just loop though and input common passwords.

- **How did the feature work differently than normal use?**
  In the DVWA it accepts unlimited attempts without triggering any lockouts or delays. On medium difficulty the only delay added is a 2 second delay between attempts this only slows the attack slightly.

- **Why did this work differently?**
  The system lacked lockout mechanisms, a rate-limiting mechanism, or CAPTCHA making it simple to brute force. Additionally there was no logging or monitoring mechanism added to detect automated tools.

- **Why should we care about this vulnerability?**
  This is in a way a simplistic attack, but if this vulnerability is present it allow for any body to find login credentials of any user that its comparing to in the secure database. This means that it would allow access to user accounts and information once login information was found and logged.

- **How can we fix the vulnerability?**

  1. Implement Logging all login attempts and monitor for unusual patterns.
  2. Add meaningful rate-limiting to slow down repeated attempts.
  3. Add CAPTCHA to limit the amount of automated tools available to use.
  4. Add account lockout after a set number of login attempt failures.

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** Have the correct setup for Burp Suite and Foxy Proxy outlined in setup or fallowing video: (Setup). Enter a recognizable entry into the password and username in my case "123" in both, and turn on intercept on Burp Suite via

Proxy $\Rightarrow$ *Intercept* $\Rightarrow$ *Intercept On* (Figure 7)

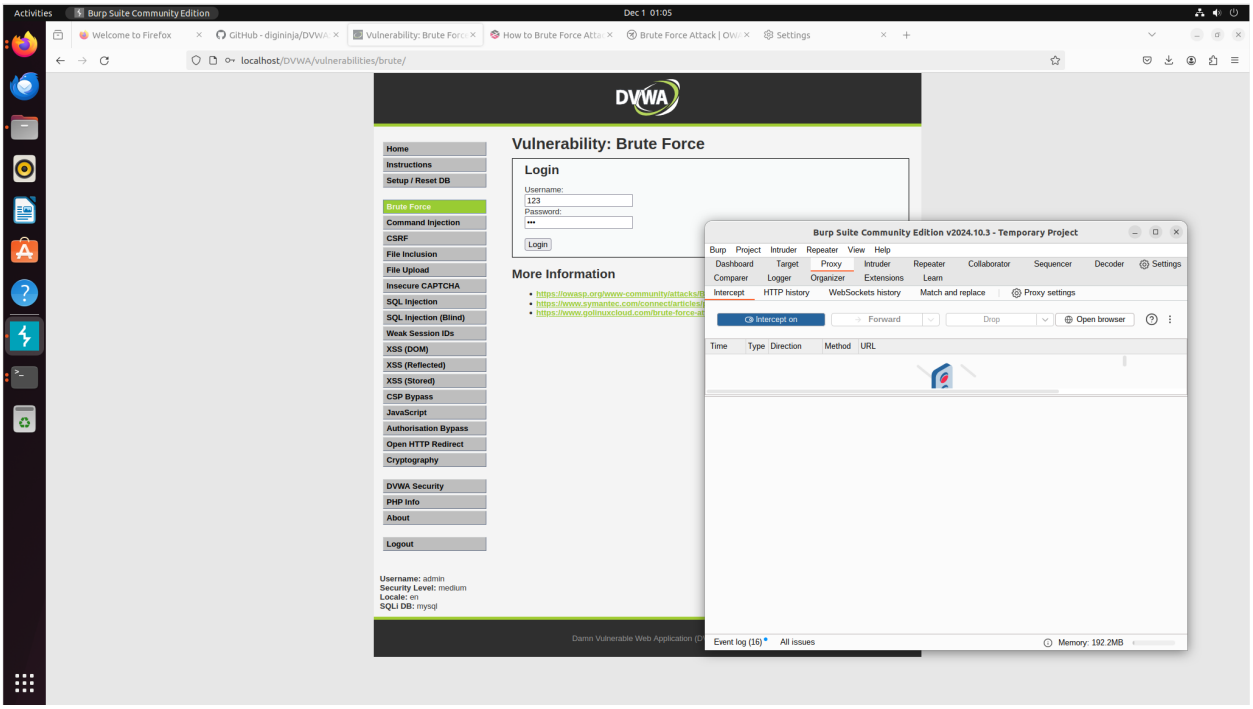

Figure 7: Step1

2. **Step 2:** Start Proxy via chrome extension by clicking the local host as seen in (Figure 8). Once you have set the proxy click login on the DVWA.

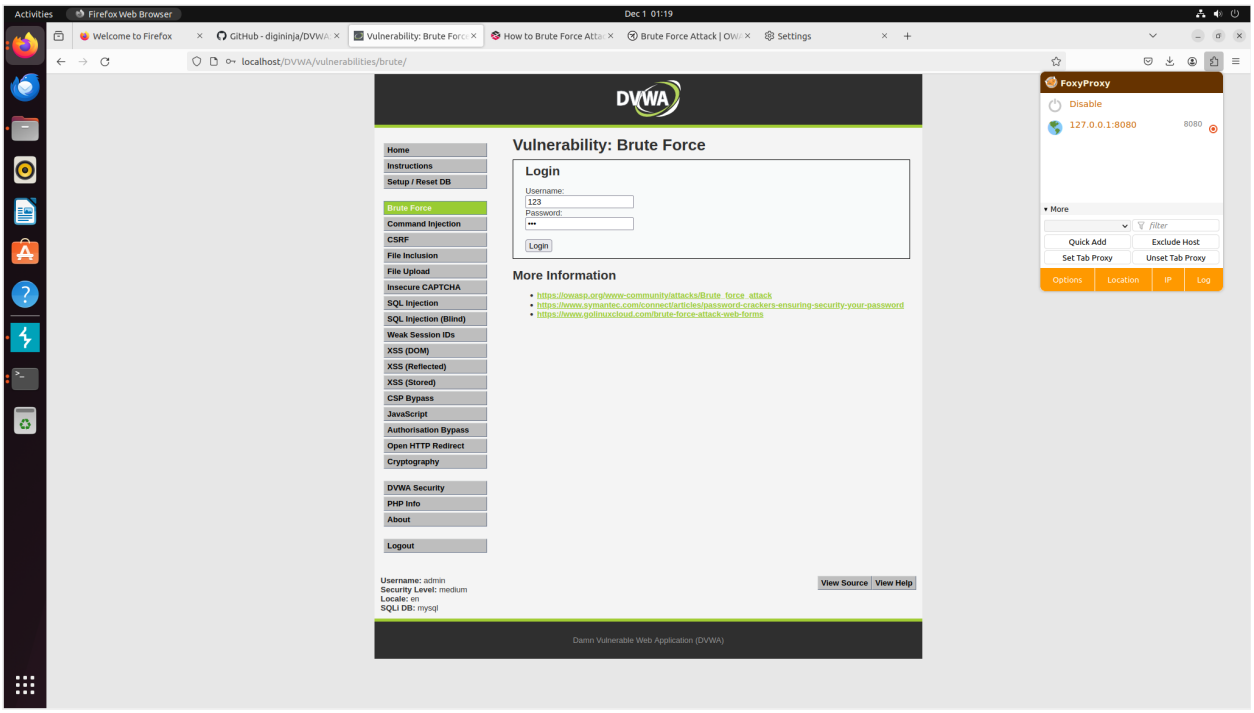

Figure 8: Step2

3. **Step 3:** Open Burp Suite and click the local host link. This should open the Get info within the raw bellow. Right click and click "Send to Intruder" (Figure 9)

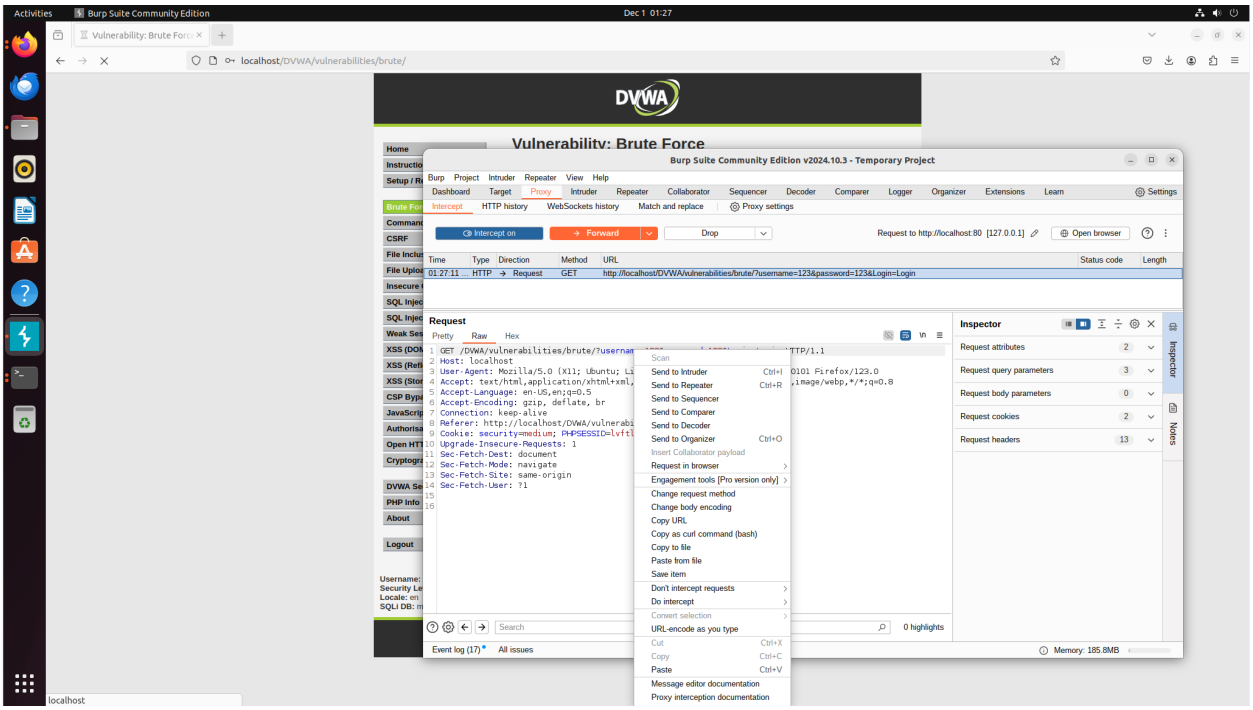

Figure 9: Step3

4. **Step 4:** Now click Intruder on the top nav bar and it should open something like (Figure 10). Now click (Clear $). Double click for the username "123" and click (Add $) repeat this for the password. Now click the drop down menu next to start attack select "Cluster Bomb Attack". After this is done your Window should look like (Figure 11)

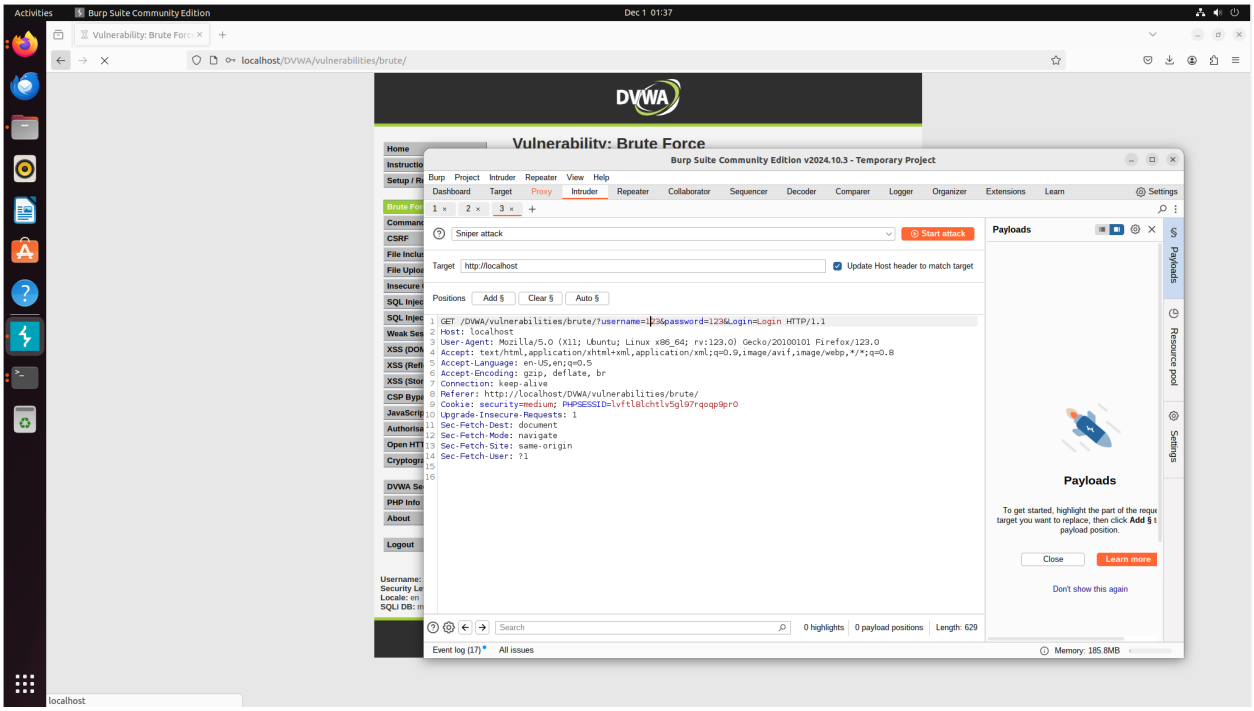

Figure 10: Step4

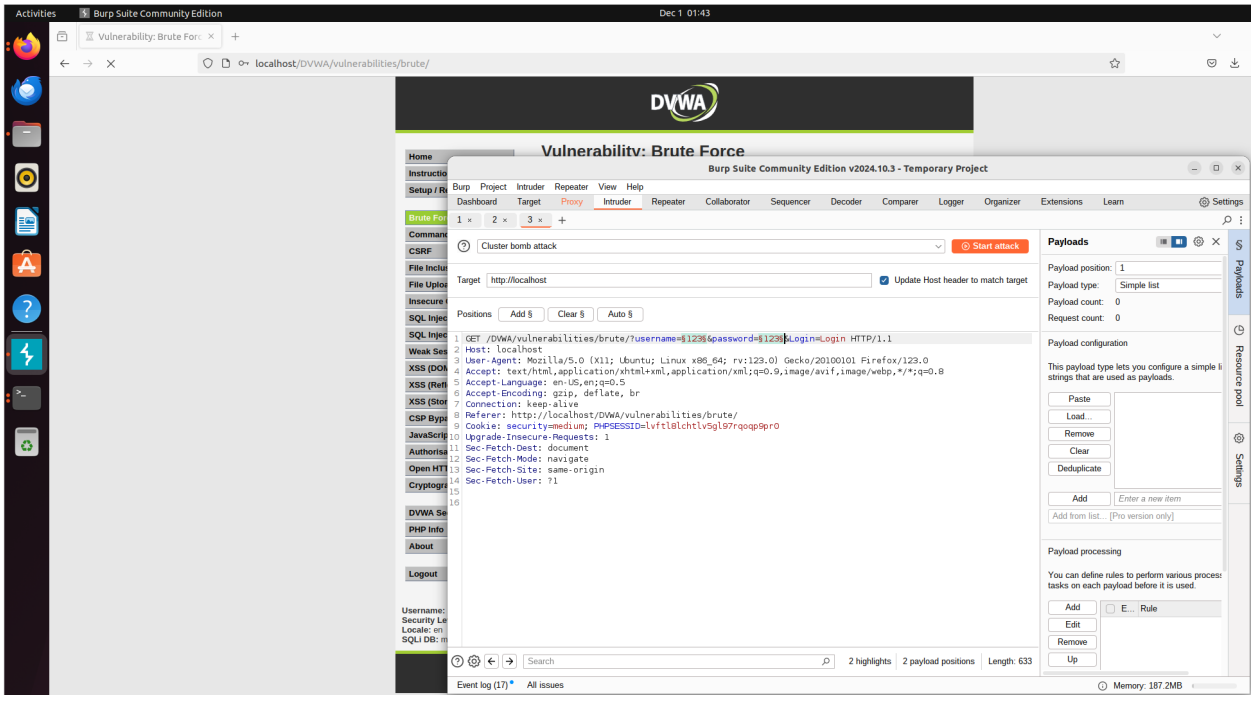

Figure 11: Step4

5. **Step 5:** Now we will move on to added in the payload. On the right we should see payloads, under payload configuration add the fallowing word; admin, manager, root, cisco, apc, pass, security, user, system, sys, wampp, newuser, xampp-dav-unsecure, vagrant (Figure 12). Now select 2 under payload postion and add the fallowing words; admin, password, manager, letmein, cisco, default, root, apc, pass, security, user, system, sys, none, xampp, wampp, ppmax2011, turnkey, vagrant (Figure 13). Now click start attack

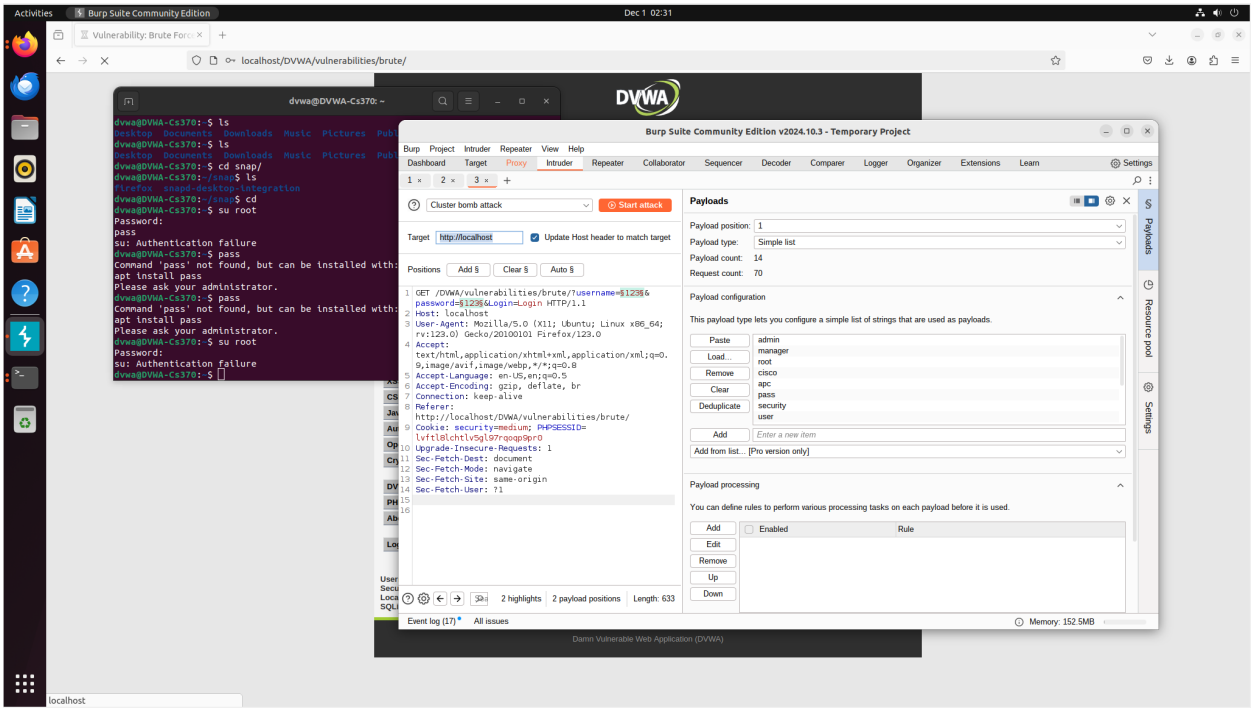

Figure 12: Step5

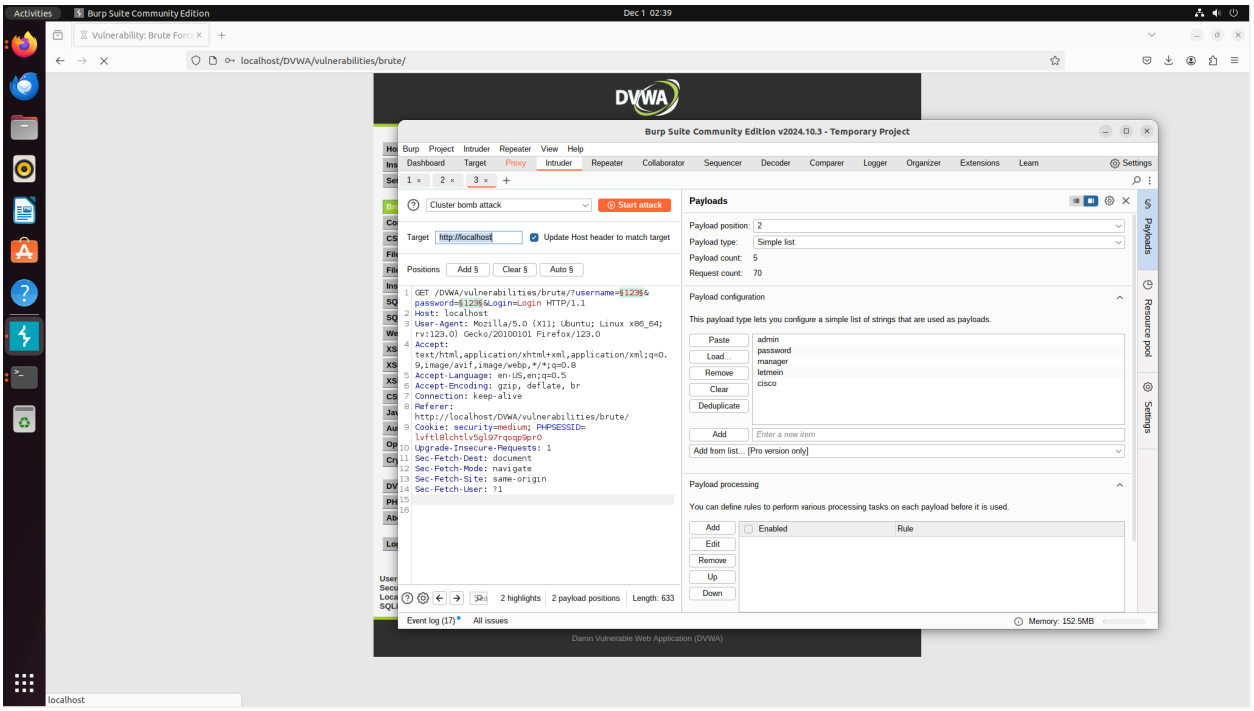

Figure 13: Step5

6. **Step 6:** Now after letting the code run though the passwords we can notice something interesting in (Figure 14). We can see that once of the response received is differing for the other within the 2000's additionally we see that the content length is much longer in this one. This means that its most likely the correct password and username.

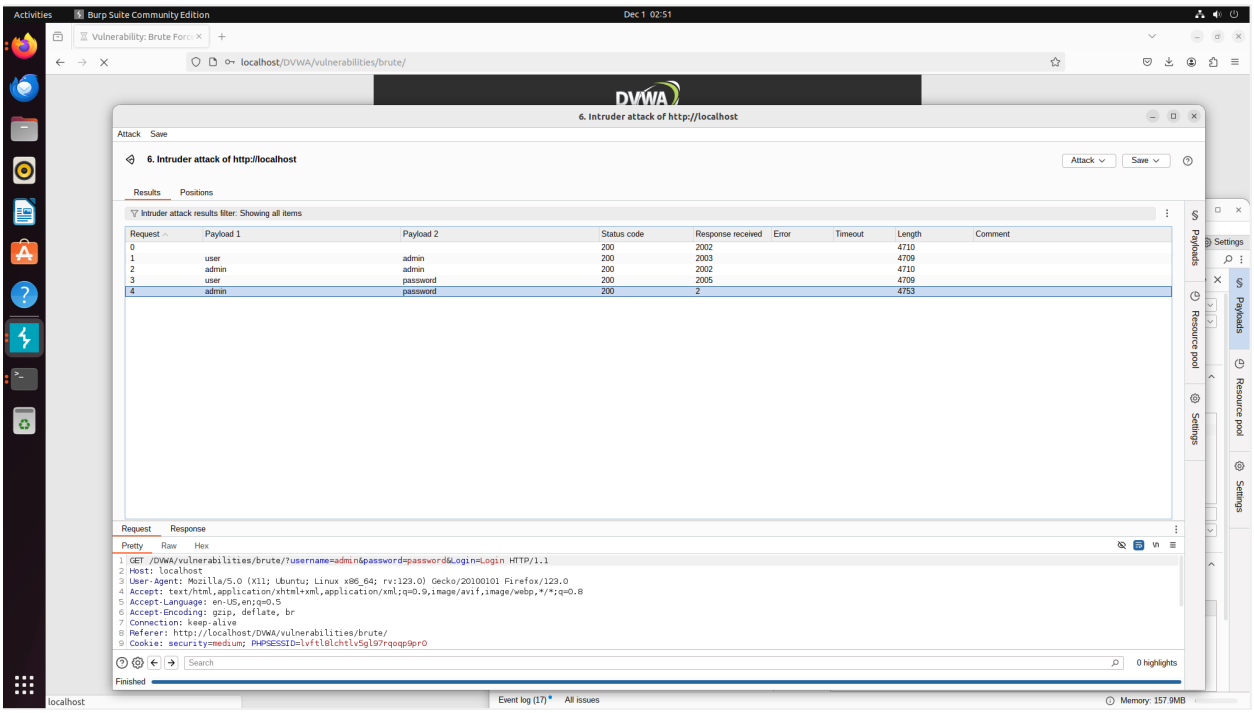

Figure 14: Step6

- **Related OWASP Top 10 Vulnerabilities:**
  A07:2021 - Identification and Authentication Failures

- # References

[1] [HackHunt], "How to configure Burp Suite with Firefox FoxyProxy on Kali Linux," YouTube, 2021. `https://www.youtube.com/watch?v=MyJnuw7afX8`.

[2] [HackHunt], "DVWA - Brute Force (Low — Medium — High)" YouTube, 2024. `https://www.youtube.com/watch?v=pSBD9cgwgk0`.

3 Command Injection

- **How does this feature normally work?**
  In a secure environment a system accepts the user input and is validated before any input or execution. This means that it prevents direct execution of input as a command due to validation rejecting it.

- **What does it take to exercise the vulnerability?**
  It requires the attacker to submit a crafted input that allows them to inject and execute commands on the host system. This uses characters like `|`, `;` , `&&`, `||`, or interacting with the URL or form that interacts with the backend of a system.

- **How did the feature work differently than normal use?**
  in DVWA the application directly executes without any validation. This allows for malicious users to inject commands that can be executed on the system.

- **Why did this work differently?**
  This exits because the web application is tasking the user input and inserting it directly into a system shell command. This combination with no validation means that user user input is treated as code rather then data allowing for attackers to manipulate the command injection flow to run commands.

- **Why should we care about this vulnerability?**
  This can lead to many many downsides if it exits. A person that is able to run a command on your system could lead to data leaks or theft. It would also lead to an escalation of privileges and allow for the user to access even more deeper in the system, and it could lead to a full system compromise where the attacker gains complete control allowing for installations of packets deletion of logs and so on.

- **How can we fix the vulnerability?**

  1. Implement input validation (ensure that it does not contain special character that can be used for injection)
  2. Use safe API calls
  3. Limit User Privileges
  4. Add Use of Whitelist to ensure only specific commands are allowed to be run by the system at the web endpoint

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** Now for this its important to first look at what the code is doing, on medium we can see that it is removing `&&`, and `;`.(Figure 15)

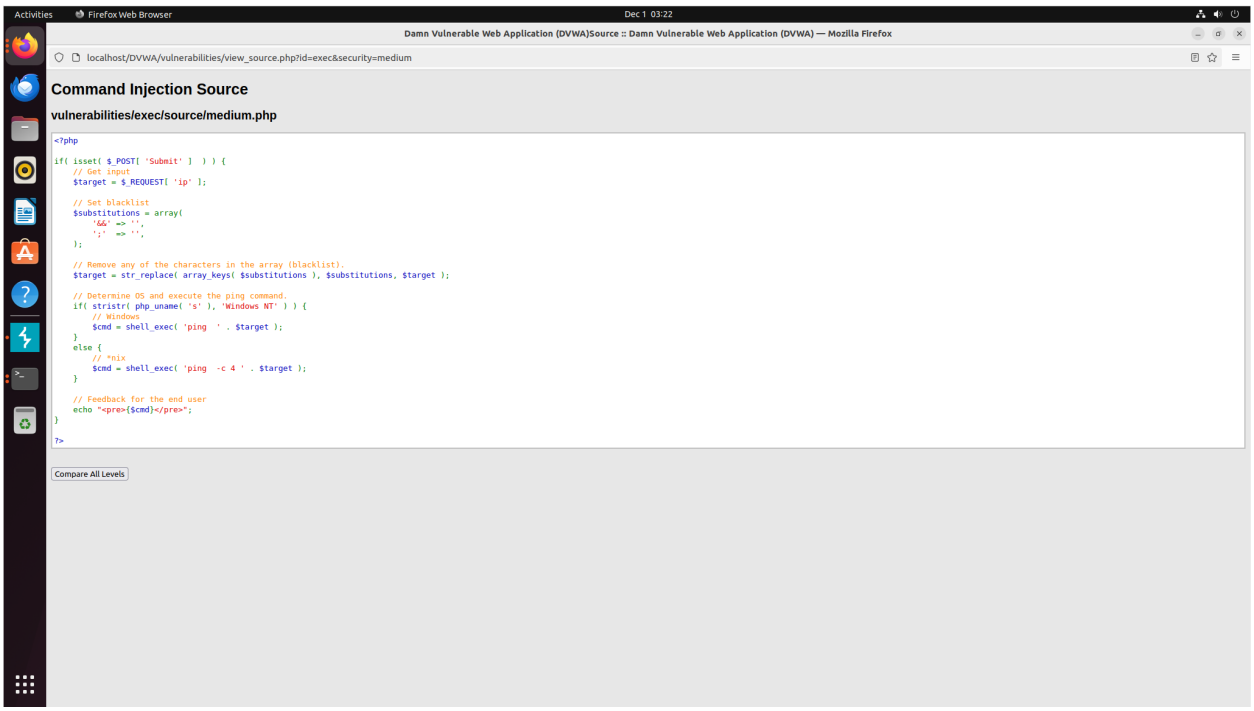

Figure 15: Step1

2. **Step 2:** Now that we have this thought we can run a simple command

```
1                    127.0.0.1 | pwd
```

When pushing enter we get the fallowing output in (Figure 16) showing our current location and thus running a command

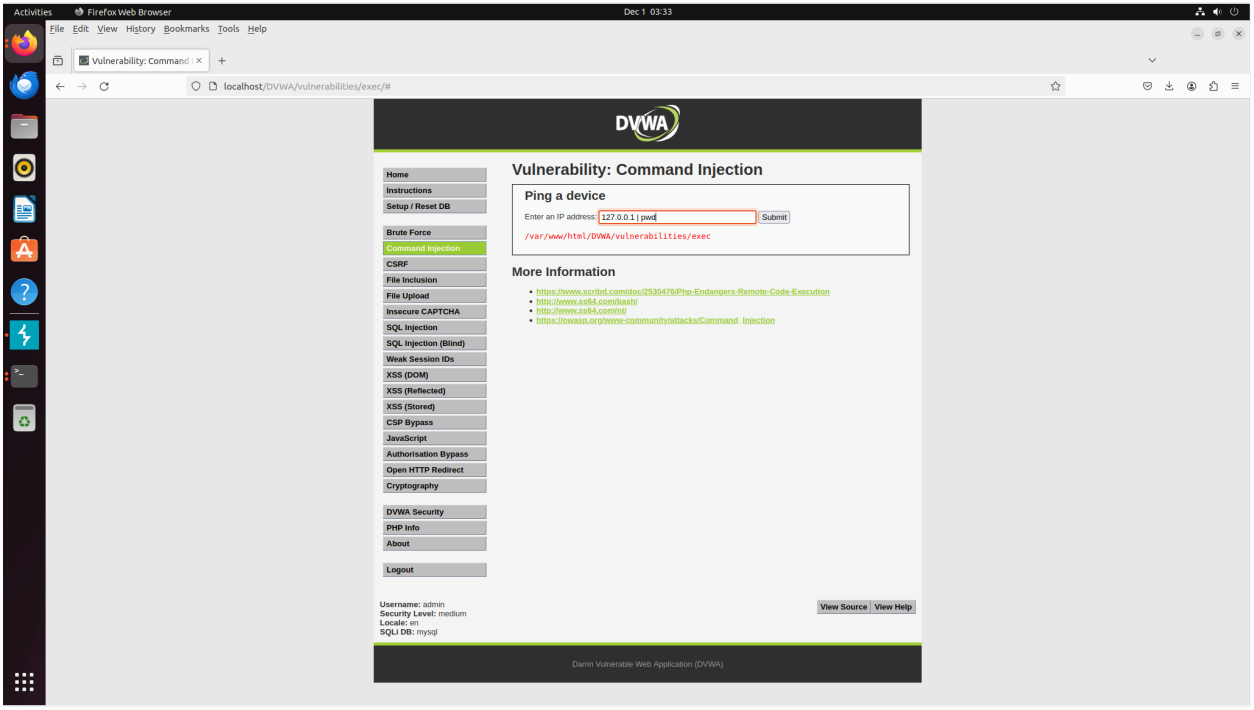

Figure 16: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A01:2021 - Broken Access Control
  - A05:2021 - Security Misconfiguration
  - A07:2021 - Identification and Authentication Failures

References

4 CSRF

- **How does this feature normally work?**
  CSRF works by doing a bit of social engineering. It revolves around tricking the user into submitting an unintended request. This can come in the form of a false link via an embed. This link could then send a request to the target website that the victim is authenticated to and for example change the password for their account making it look like the victim sent the request.

- **What does it take to exercise the vulnerability?**
  For this it requires the user to be logged into the website and the attacker to craft

a malicious link. When the user clicks the link the execution is ran if there is not authentication setup.

- **How did the feature work differently than normal use?**
  In CSRF attacks work without the users consent or knowledge. Attackers can trigger actions by embedding malicious request within a background operation, making it unnoticed by a victim until its too late.

- **Why did this work differently?**
  CSRF work because the server does not verify the origin of the request or if the user meant to submit the request. Websites use cookies or tokens to authenticate the user and these tokens and cookies are are sent by the browser and the server will treat these requests as valid even if the user did not do it intentionally.

- **Why should we care about this vulnerability?**
  These attacks and lead to changes of sensitive information, unauthorized transactions or changing of passwords. It could lead to accounts being taken.

- **How can we fix the vulnerability?**

  1. Use of Anti CSRF Tokens
  2. implementation of checking if the request is coming form the correct domain
  3. Use of HTTPS

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** The first step is creating the malicious link for this we must know the web input when a password is changed. To set this up we could run an intercept but out of convenience change the password and copy the link. Now create a text file containing the fallowing code. Make sure to replace [copied link] with your copied link (Figure 17). Now change your password back to what it was, and Open the html website that you just created.

```
<html>
<body>
<h3>Test</h3>
<a href="[copied␣link]"=Change#>Newlink</a>
</body>
</html>
```

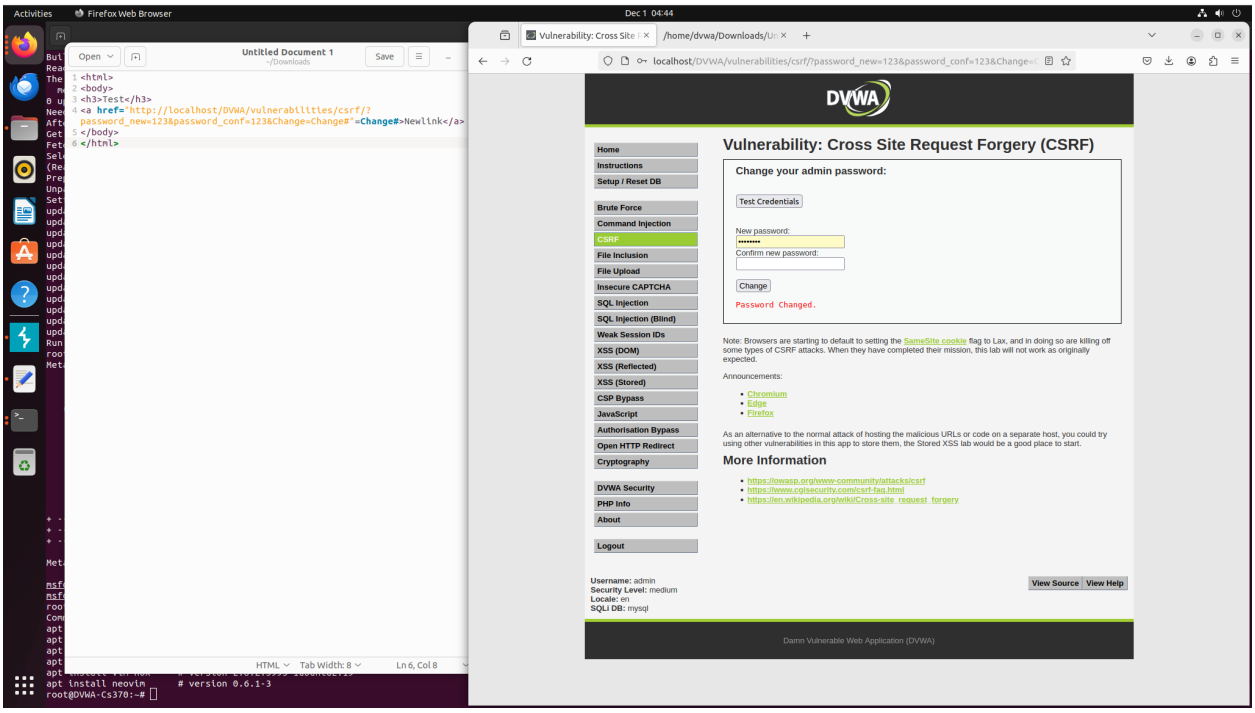

Figure 17: Step1

2. **Step 2:** Now enter pass into the password change boxes. Enable foxy proxy and Burp Suite Intercept (Shown in Brute Force Step 1). Now click change, we should now see a URL containing conf=pass. Now click the link on your own website. This will then pull up a second HTTP request with conf=123 representing the attackers desired password change. Your Burp Suite should look like (Figure 18)

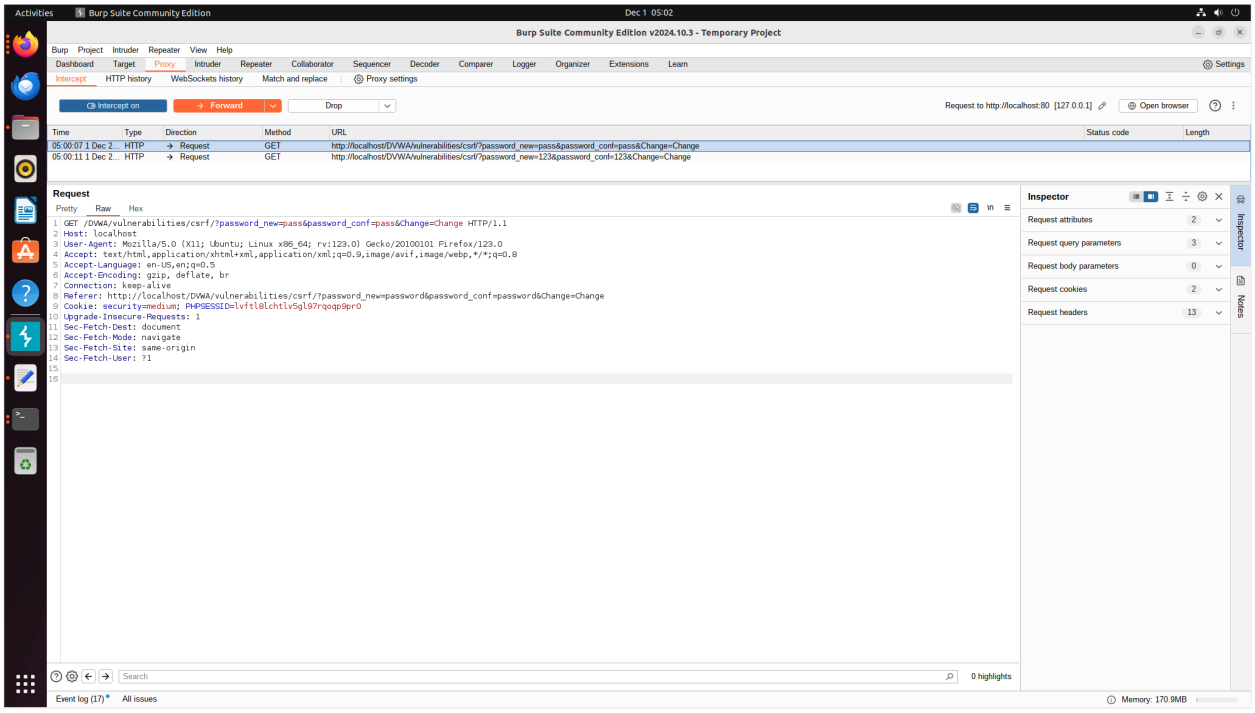

Figure 18: Step2

3. **Step 3:** Now when clicking between the two requests we see something interesting on our created website we do not have a Referer. To fix this we simply need to copy the Referer form the first one and add it to our newlink request and change the pass to 123 (Figure 19). Now click forward for the newlink request. We can now see that the password has changed(Figure 20).

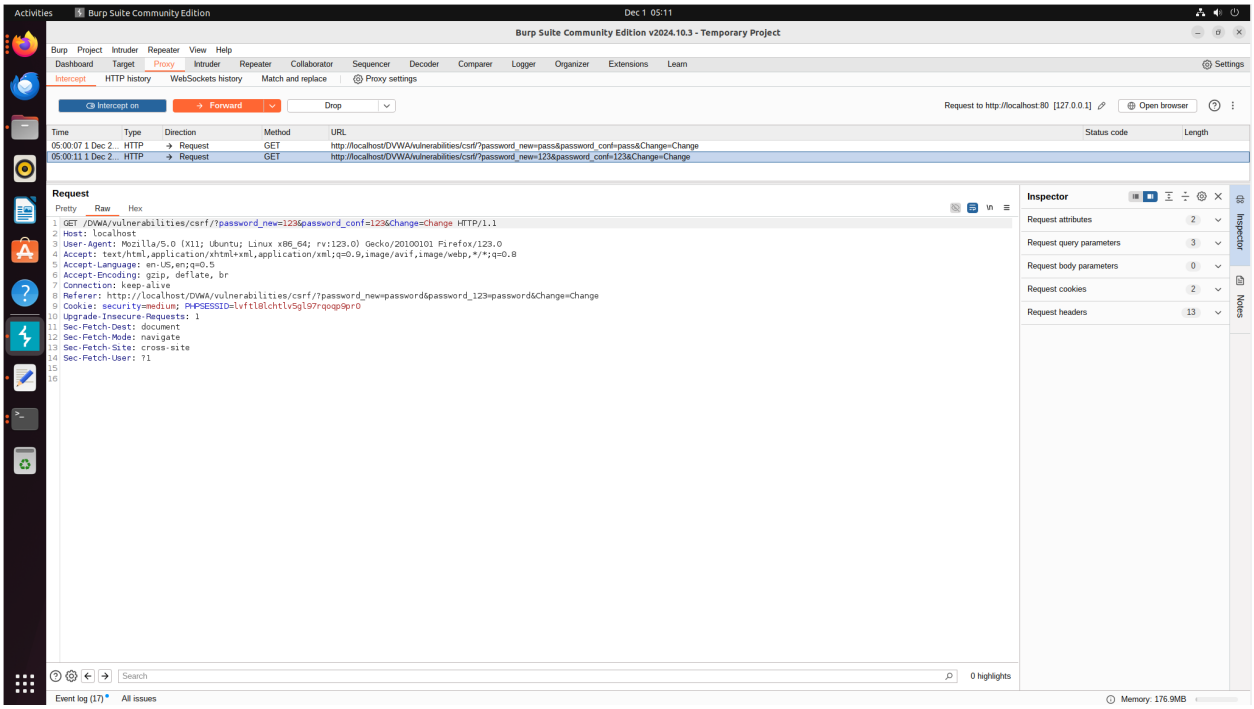

Figure 19: Step3

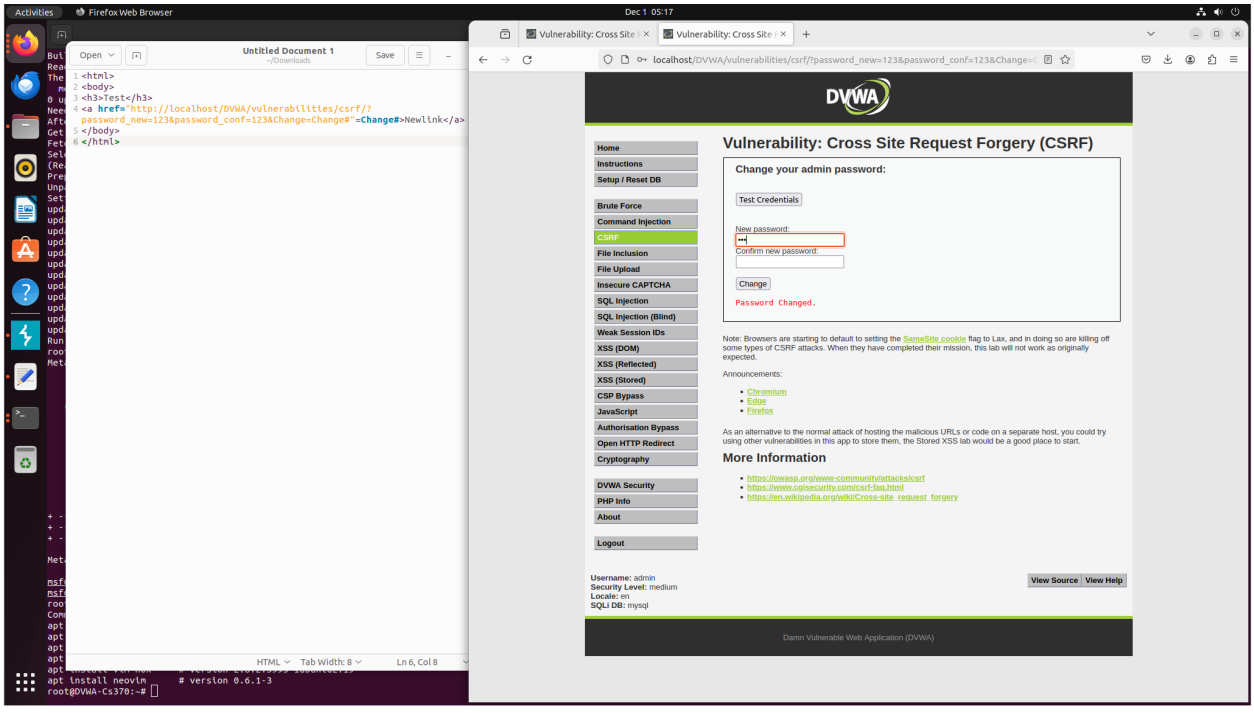

Figure 20: Step3

- **Related OWASP Top 10 Vulnerabilities:**

  - A07:2021 - Identification and Authentication Failures

• References

[1] [How2Hack], "DVWA CSRF LOW , MEDIUM — HOW TO HACK — CYBER SECURITY TUTORIAL BEGINERS — ETHICAL HACKING — OWASP10" YouTube, 2022. https://www.youtube.com/watch?v=-GfLZkGdc_w.

5 File Inclusion

- **How does this feature normally work?**
  In a default application file inclusion is used to dynamicly load the files on the server side. A good example is a web app having home.html and page1.html.

- **What does it take to exercise the vulnerability?**
  It requires the attacker to input and manipulate the file parameter to included unintended files. This can be done though ./ ../ and .../ . This can lead to an execution of malicious scripts by including external files.

- **How did the feature work differently than normal use?**
  The server processes includes any file specified in the input. This means that the file parameter is not sanitized correctly.

- **Why did this work differently?**
  This happens because lack of input validation and lack of functions like include or require in PHP. This also means that there is an absence of configuration settings to prevent remote file inclusion.

- **Why should we care about this vulnerability?**
  This could lead to passwords being exposed or other data being exposed via access to a unsecured sensitive file. It could lead to code executions on the server.

- **How can we fix the vulnerability?**

  1. Set PHP configurations to allow certain url's
  2. Restrict file access
  3. Validate user input

- **Steps to Exercise the Vulnerability:**

1. **Step 1:** When looking at the code we can seee that its removing ../ ..
but we can use .../ and ./ still. Using this knowledge we can switch from for example file 1 to the PHP info page file. By clicking on file1.php we can see that we have a url=?page=file1.php(Figure 21)

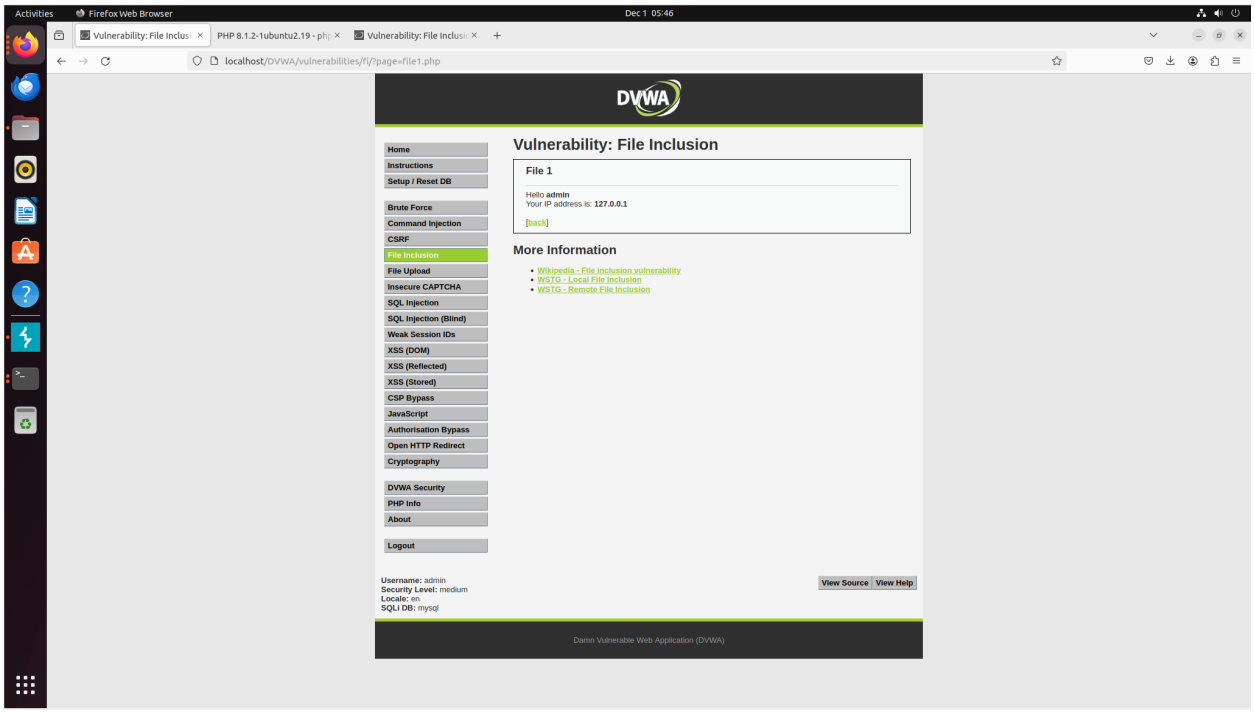

Figure 21: Step1

2. **Step 2:** we can now change the page=file1.php to page=..././..././phpinfo.php (Figure 22). This allows us to access the PHP info from the file inclusion page

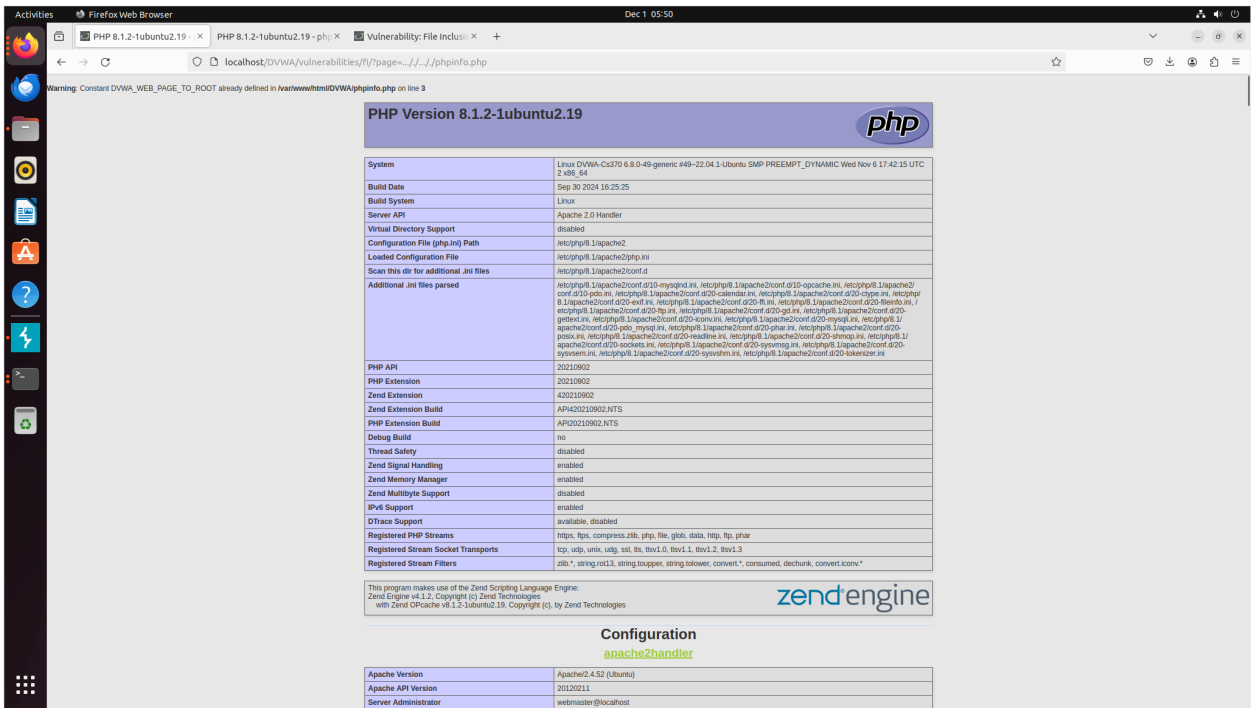

Figure 22: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A01:2021 - Broken Access Control
  - A06:2021 - Vulnerable and Outdated Components

- # References

[1] OWASP, "Testing for File Inclusion," OWASP Web Security Testing Guide, `https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_File_Inclusion`. Accessed December 1, 2024.

6 File Upload

- **How does this feature normally work?**
  File upload allow users to upload files like images documents and other resources to a server. The server validates the uploaded file making sure the application requirements are met.

- **What does it take to exercise the vulnerability?**
  To exploit this vulnerability an attacker can upload a malicious file like a web shell or

a script descied as a regular file. This restrictions the bypassing the restrictions of the file type.

- **How did the feature work differently than normal use?**
  In DVWA the file upload features looks for JPG and restricts PHP files form being uploaded this being said you can insert scripts using a php.jpg or a changing of the content type within the http request.

- **Why did this work differently?**
  The vulnerability exists because the server does not enforce validation rules and its only done on the front end side. File contents are not inspected and files are uploaded within the same directory of other files with little access control limitations.

- **Why should we care about this vulnerability?**
  This could lead to code execution meaning that the same goes here as the rest data breaches, system compromises can be a thing. Along with this attackers could escalate privileges due to access to the back-end.

- **How can we fix the vulnerability?**

  1. Strictly validate file types
  2. Validate file content
  3. Reject double extensions

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** To start we will be making a php file to be uploading instead of the inteded jpeg. For this we will be writing some simple php.

```
1 <?php system($_REQUEST["cmd"]); ?>
```

  code(Figure 23)

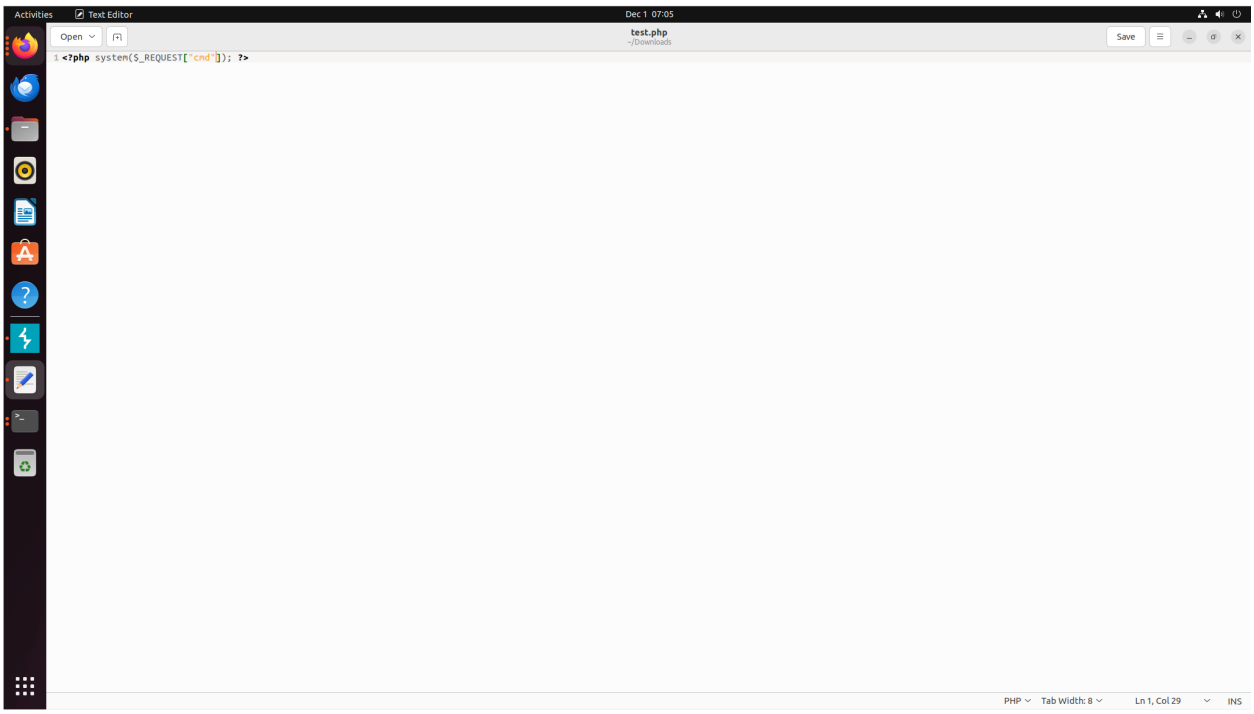

Figure 23: Step1

2. **Step 2:** We can now start foxy proxy, and Burp and upload our new php file. When doing this we can see a Post request and when clicking on it we real our request.(Figure 30).

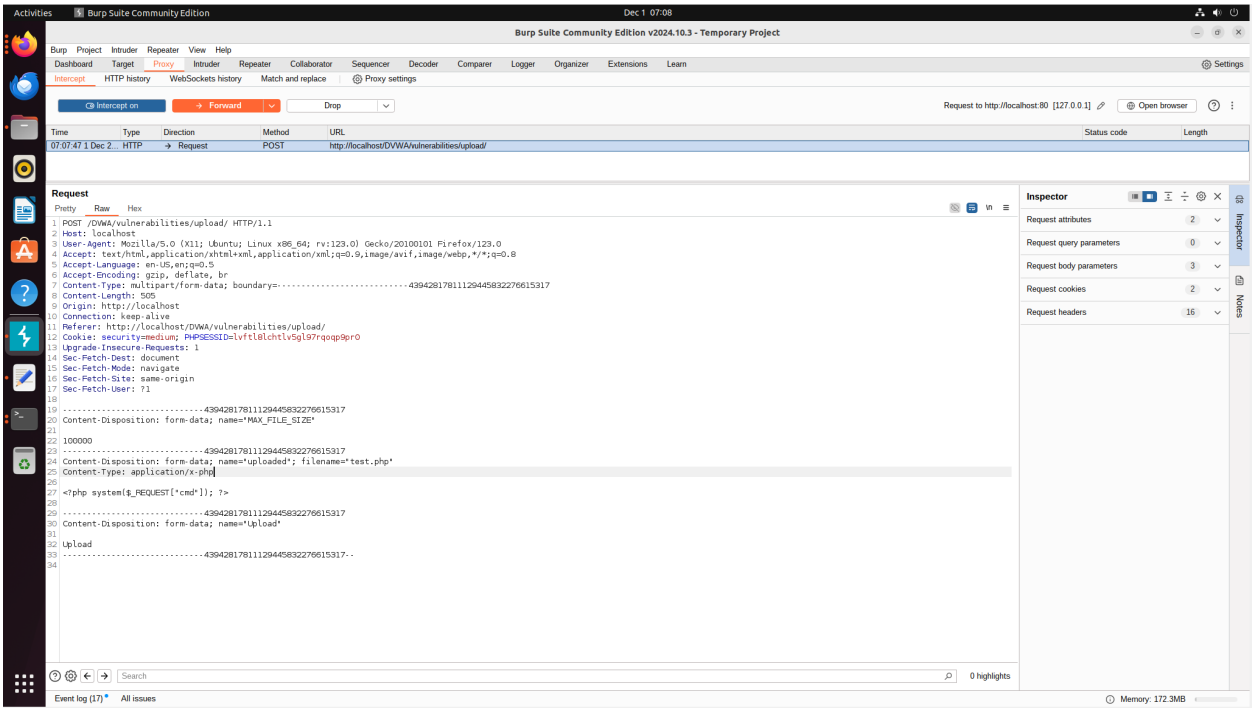

Figure 24: Step2

3. **Step 3:** We then will edit the Content-Type: form application/x-php to image/jpeg to override the jpeg check (Figure 32). When clicking forward and clicking back to our homepage we see that the php file has been uploaded as seen in (Figure 26)

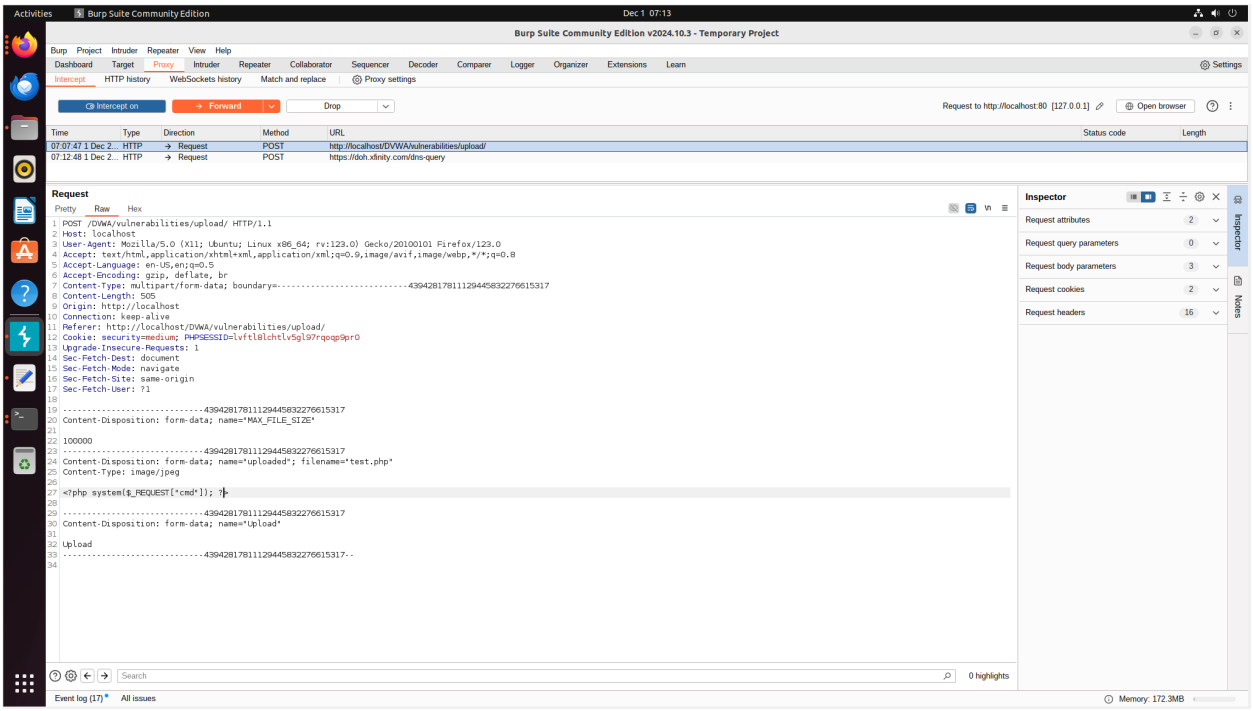

Figure 25: Step3

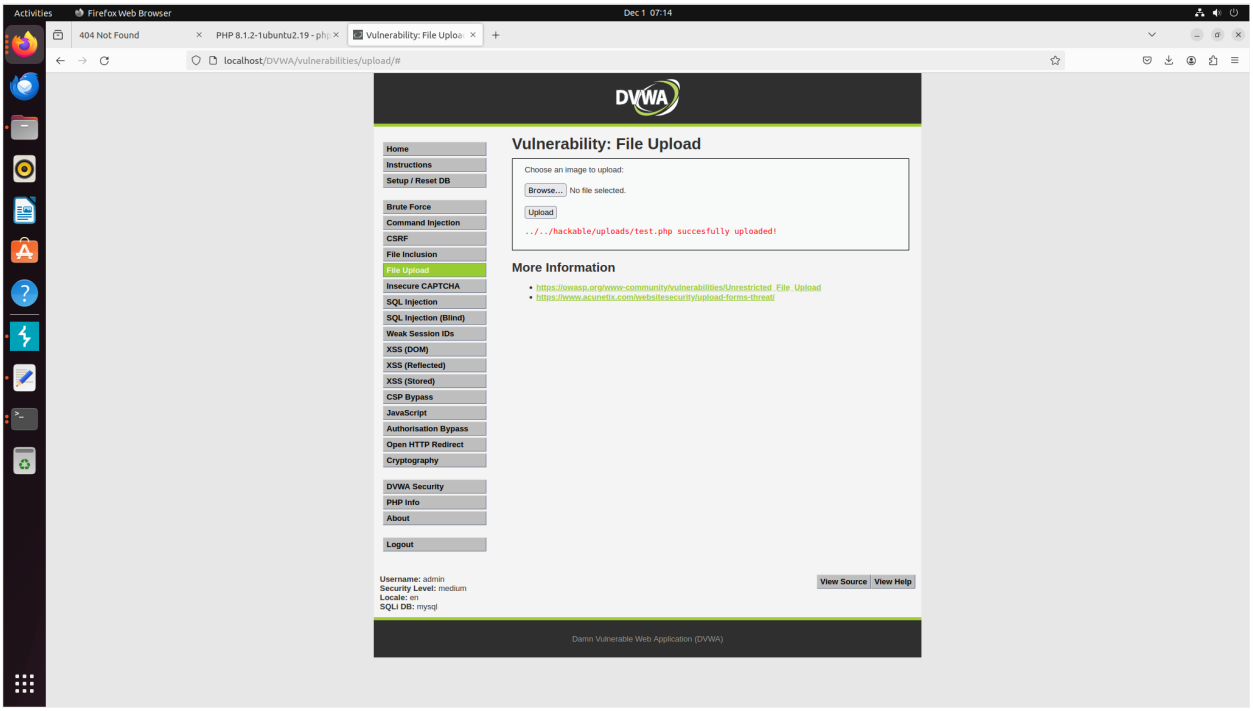

Figure 26: Step3

4. **Step 4:** Finally we are able to access the site though localhost/DVWA/hackable/uploads/test.php?cmd=ls where we can see it running a shell command though a file we uploaded (Figure 27)

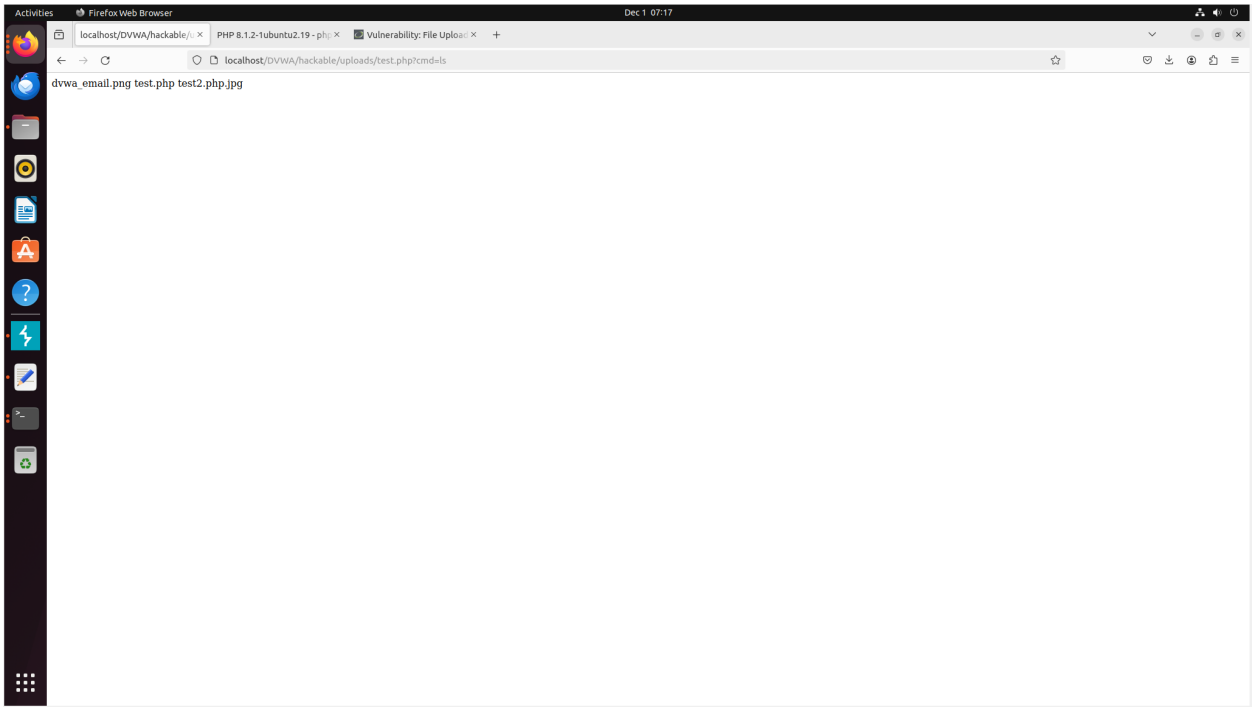

Figure 27: Step1

- **Related OWASP Top 10 Vulnerabilities:**
  - A01:2021 - Broken Access Control
  - A06:2021 - Vulnerable and Outdated Components
  - A03:2021 – Injection
  - A05:2021 – Security Misconfiguration

• References

[1] [CyptoCat], "5 - File Upload (low/med/high) - Damn Vulnerable Web Application (DVWA)" YouTube, 2021. `https://www.youtube.com/watch?v=K7XBQWAZdZ4&list=PLHUKi1UlEgOJLPSFZaFKMoexpM6qhOb4Q&index=6`.

7 Insecure CAPTCHA

- **How does this feature normally work?**
  A CAPTCHA is typicaly used to prevent bots from interacting with pages. It is a key that is easy for humans but is hard for bots.

- **What does it take to exercise the vulnerability?**
  To exploit a CAPTCHA it require a few thing a simple predictable CAPTCHA, a lack of rate limit within the CAPTCHA or a weak CAPTCHA algorithm, Additionally we can go around the CAPTCHA by capturing if the CAPTCHA was competed or not and just starting an input form there.

- **How did the feature work differently than normal use?**
  Insecure CAPTCHA can be bypassed via validation or automatically though submission forms. It really comes down to a weak implementation of the CAPTCHA more then that of a weak CAPTCHA usually.

- **Why did this work differently?**
  The CAPTCHA in this DVWA failed because we are able to access the completion token after the CAPTCHA is competed and just trick the system into thinking the CAPTCHA was computed for a separate request.

- **Why should we care about this vulnerability?**
  This security measure could be useless if not implemented correctly allowing for bypass of the CAPTCHA allows for other attacks to happen like the ones we talked about previously.

- **How can we fix the vulnerability?**

  1. Proper Implementation using session ID's or States not being on the user side
  2. Upgraded CAPTCHA
  3. other User verification

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** To start we will be completing the CAPTCHA and entering a new password your screen should look like (Figure 28). Now click Change and you should get a screen like (Figure 29)

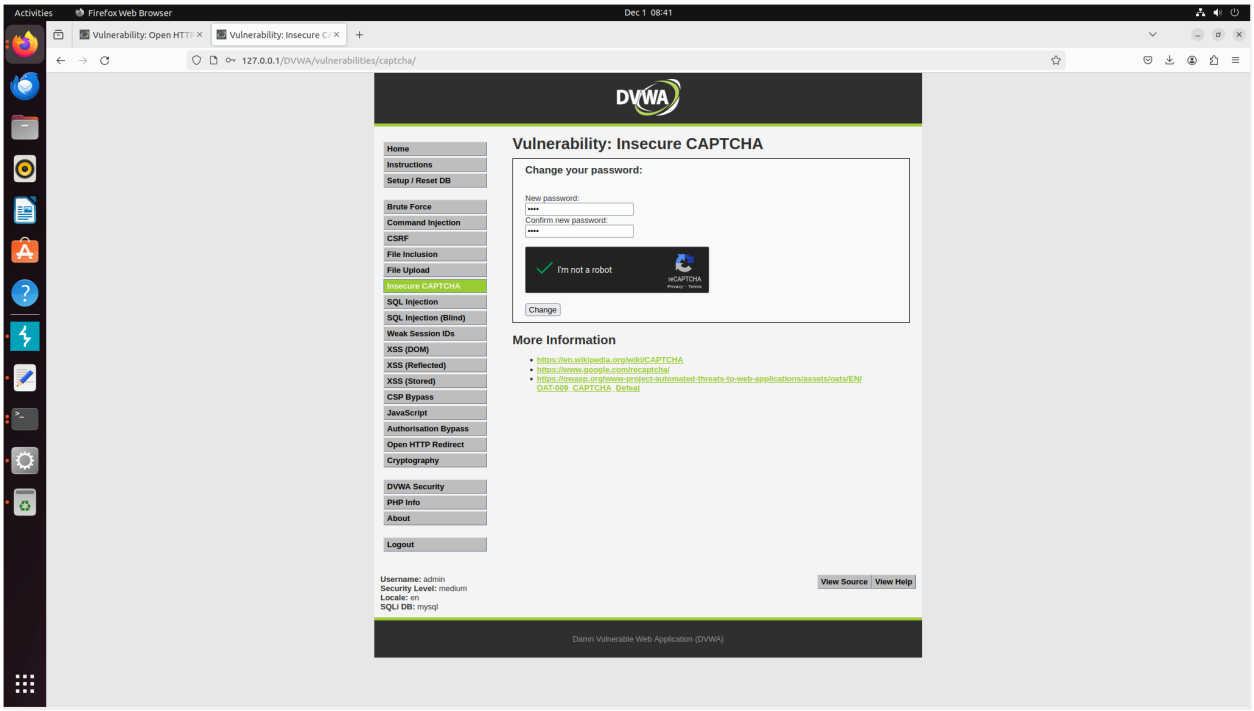

Figure 28: Step1

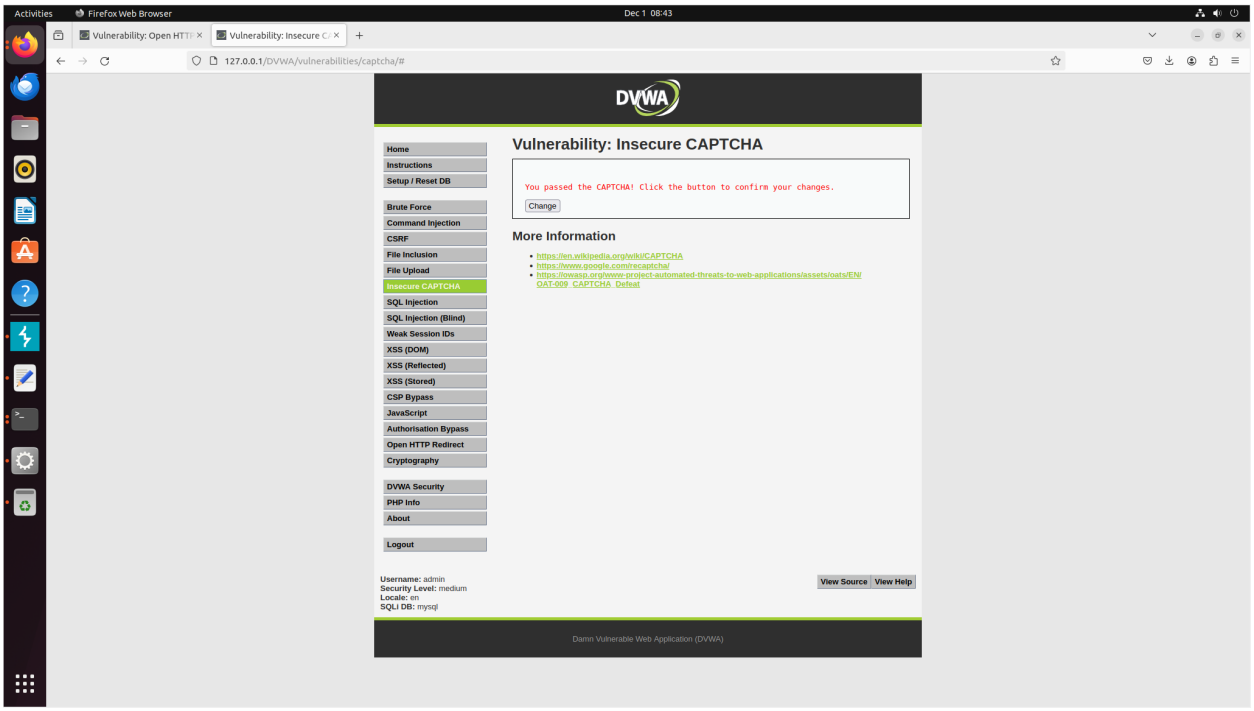

Figure 29: Step1

2. **Step 2:** We can now start foxy proxy, and Burp. After starting the two click change and navigate back to Burp and see the POST Request (Figure 30). When looking at this we can see a few things within this POST request. A varibale new holding the new password, and a conf holding the conformation of the new password a captcha bool holding if we have passed the captcha. Using this we can then just change the variables in our new and conf to match a password we desire like (Figure 31)

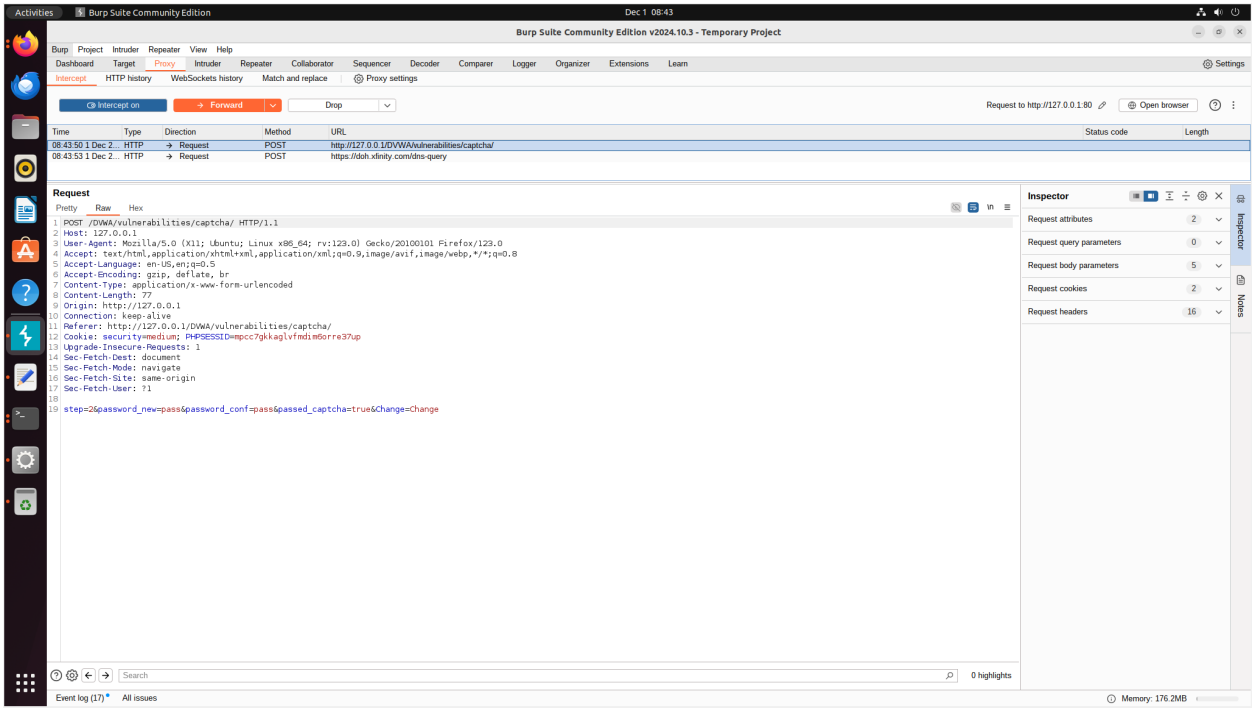

Figure 30: Step2

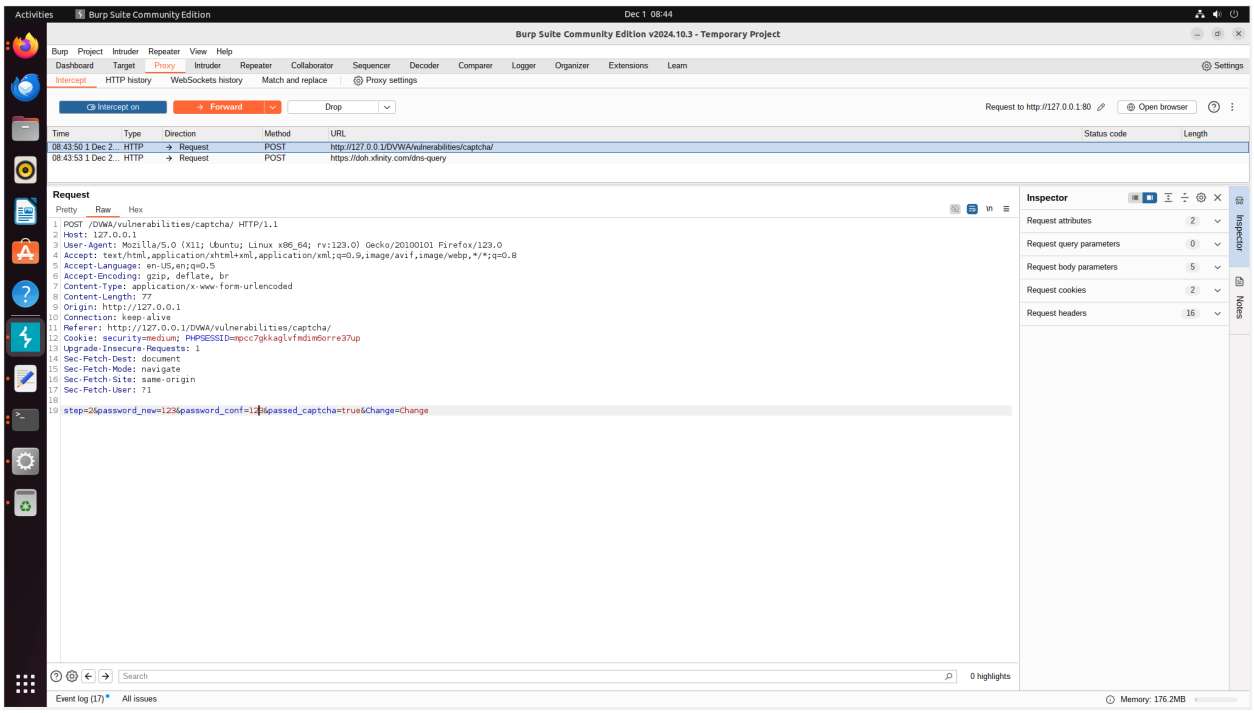

Figure 31: Step2

3. **Step 3:** Now when clicking forward and looking back on our website we can see (Figure 26). When confirming with logging in we can see that our password has been changed to 123 competently circumnavigating the CAPTCHA

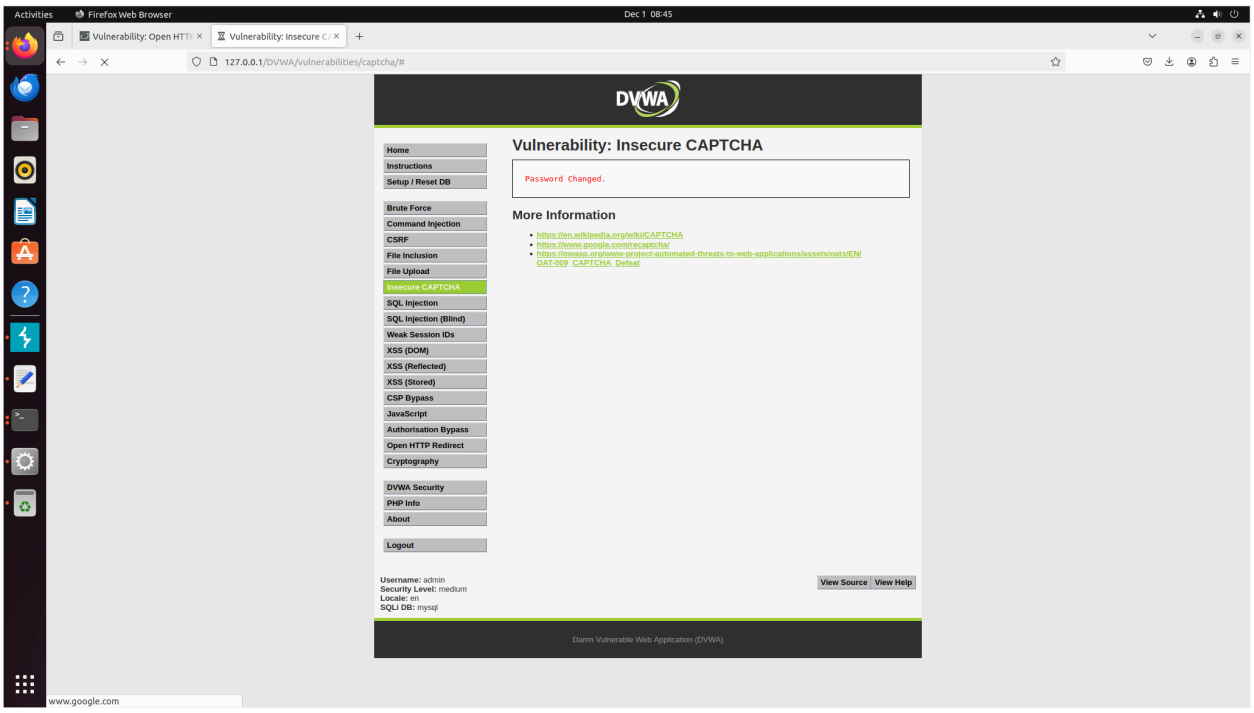

Figure 32: Step3

- **Related OWASP Top 10 Vulnerabilities:**
  - A7: Insufficient Security Configurations
  - A5: Security Misconfiguratio

- **References**

8 SQL Injection

- **How does this feature normally work?**
  drop-down fields or search bar within web application allow input data. This input data is then process by the backend and used to retrieve or manipulate the database.

- **What does it take to exercise the vulnerability?**
  To exploit SQL Injection it involves injecting SQL into the input field so that that the backend application runs the SQL that you injected. Adding 1 or 1 = 1 could allow for users to input SQL.

- **How did the feature work differently than normal use?**
  Normally drop down menus or inputs pass values after a check to the backend to query though a database using SQL. This vulnerability of injecting SQL code and making it run is not intended behavior.

- **Why did this work differently?**
  User inputs were directly inserted into the SQL queries. There were no checks to ensure user-provided data was in the expected format.
- **Why should we care about this vulnerability?**
  It can lead to data loss exposure or data modification and is a popular way to exploit unguarded doors to gain information stored in the DB's.
- **How can we fix the vulnerability?**

  1. Sanitize and Validate Inputs
  2. Change database permissions to allow the least privilege for user accounts
  3. Use of Prepared statements that switch input to data not code

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** For this SQL injection we will be using Inspect element. Under the drop down right click and click inspect. Navigate though the drop-down div to find

```
1          <option value="1">1</option>
```

  this is where we will be adding our SQL code (Figure 33)

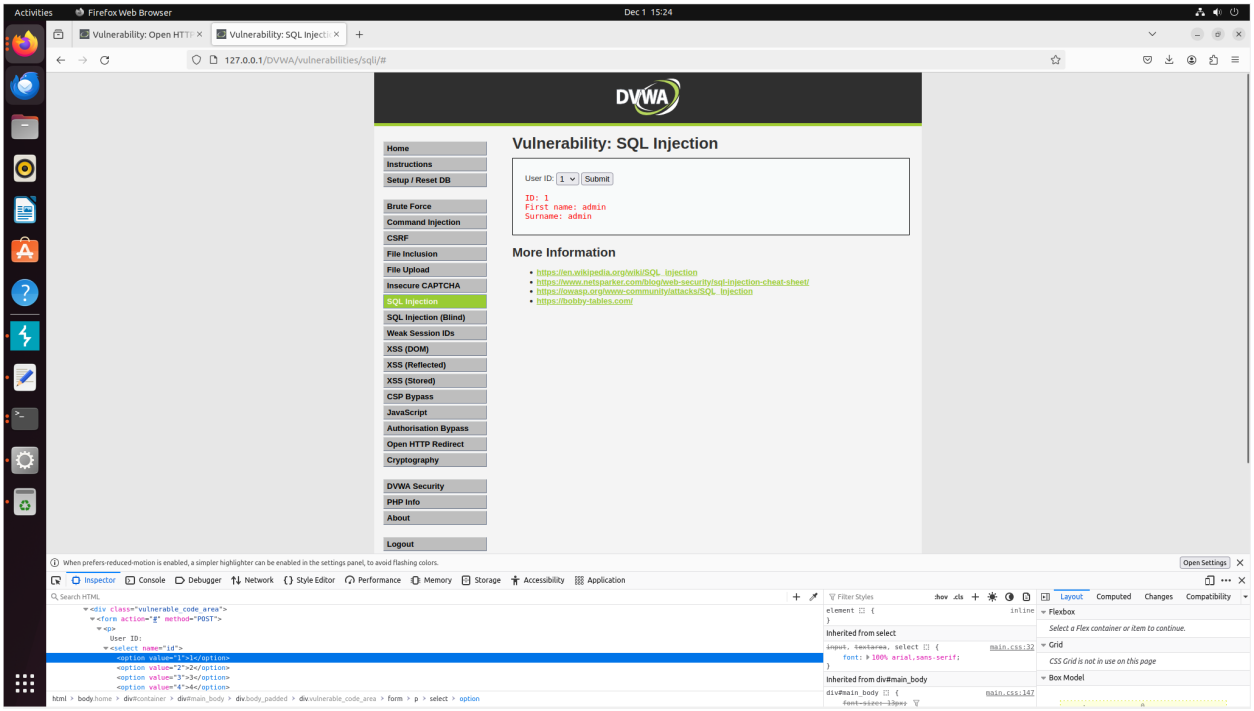

Figure 33: Step1

2. **Step 2:** Replace the previous code with

```
<option value="1␣or␣1=1␣UNION␣SELECT␣user,␣password␣FROM␣users#">UNION SELECT user, password FROM users#</option>
```

(Figure 34)

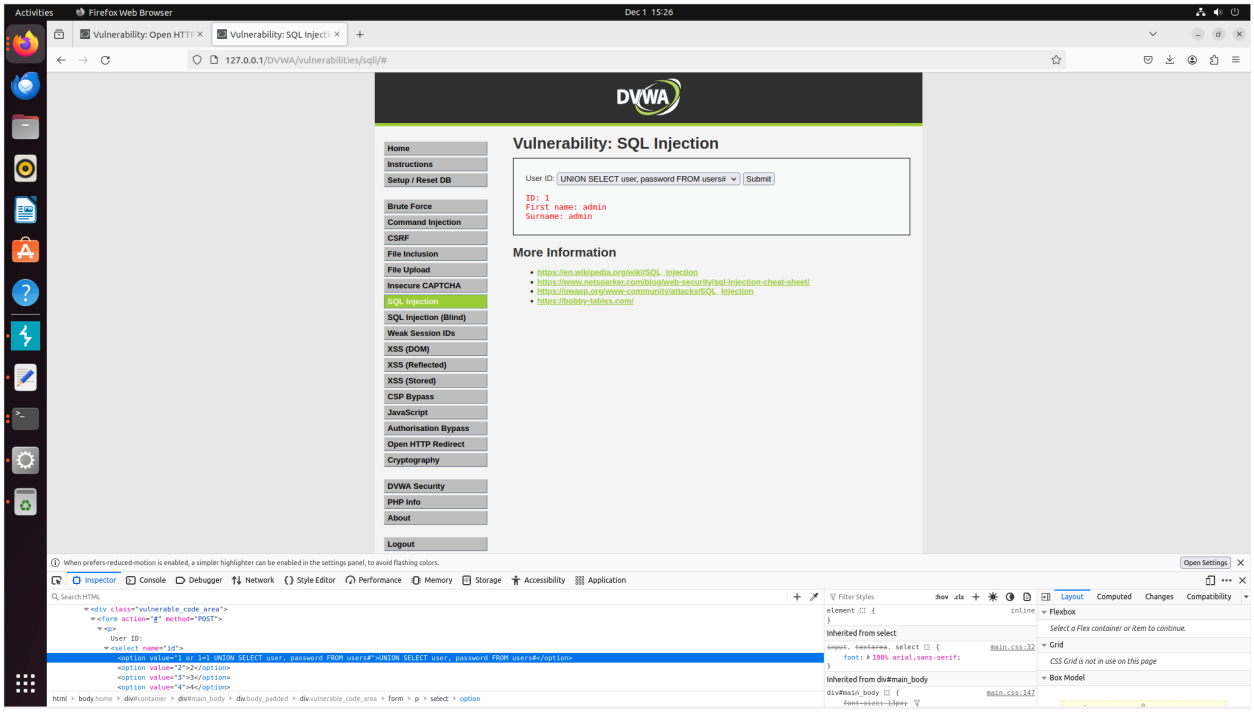

Figure 34: Step2

3. **Step 3:** Click submit and we can see that the SQL injection has worked displaying out all the passwords for users (Figure 35)

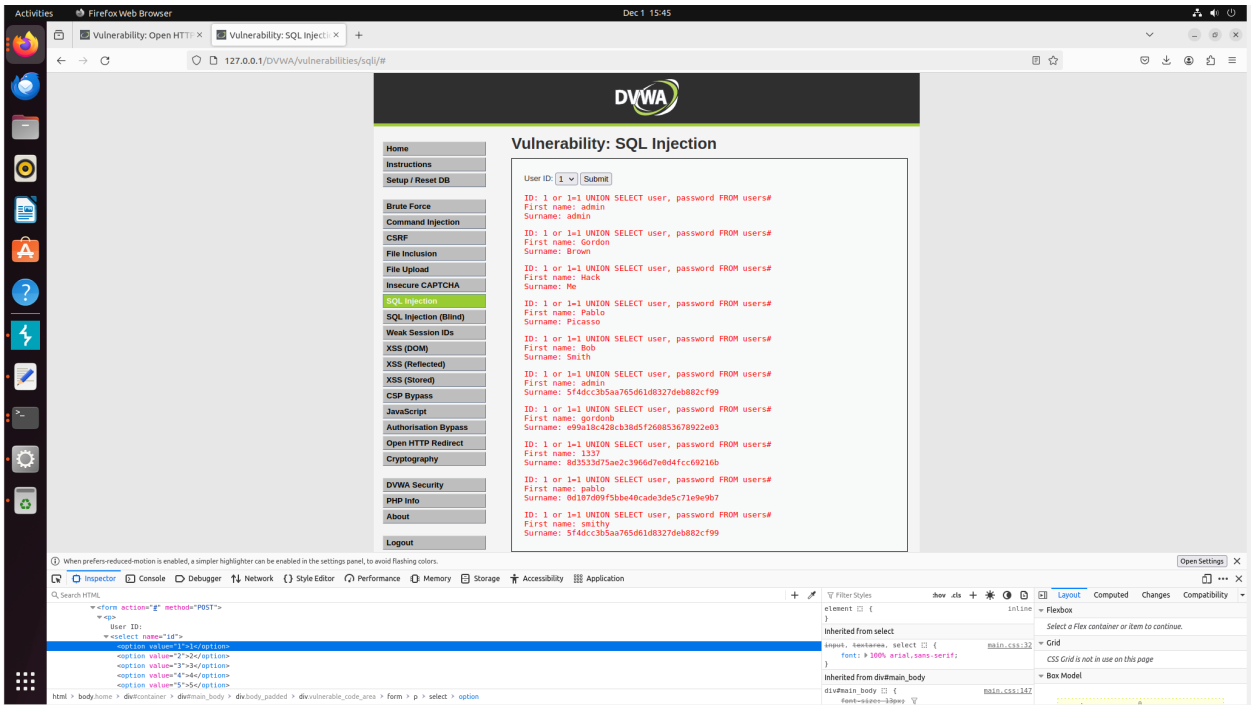

Figure 35: Step3

- **Related OWASP Top 10 Vulnerabilities:**
  - A3: Injection: Specifically, SQL injection vulnerabilities.

- # References

[1] [CyptoCat], "7 - SQL Injection (low/med/high) - Damn Vulnerable Web Application (DVWA)" YouTube, 2021. `https://www.youtube.com/watch?v=5bj1pFmyyBA`.

9 SQL Injection (Blind)

- **How does this feature normally work?**
  The web application uses the id parameter to query the database. This means that code like

```
1          SELECT * FROM users WHERE id = 1;
```

  would display on the webpage.

- **What does it take to exercise the vulnerability?**
  We are going to using a time based SQL injection meaning that wee will be injecting a payload like

```
?id=1 AND sleep(3)&Submit=Submit
```

This would delay the response by 3 seconds if the condition is true. This daly then can be used to indicate if the SQL query executed injecting the command successfully

- **How did the feature work differently than normal use?**
  The application would retrieve our data and then we would see a noticeable delay between the result.

- **Why did this work differently?**
  it does not sanitize the parameterize correctly or at all within the medium to low security options allowing for SQL commands to execute.

- **Why should we care about this vulnerability?**
  This with the combination with other exploits could lead to unauthorized access or could lead to attacker to see sensitive information.

- **How can we fix the vulnerability?**

  1. Use of prepare statements to parameterize inputs to prevent concatenation
  2. Allow for only expected inputs within the fields
  3. Limiting privileges.

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** Start your foxy proxy and turn on intercept on Burp and then click submit in the webpage. You should see a

```
id=1&Submit=Submit
```

  located at the bottom of the request like (Figure **??**)

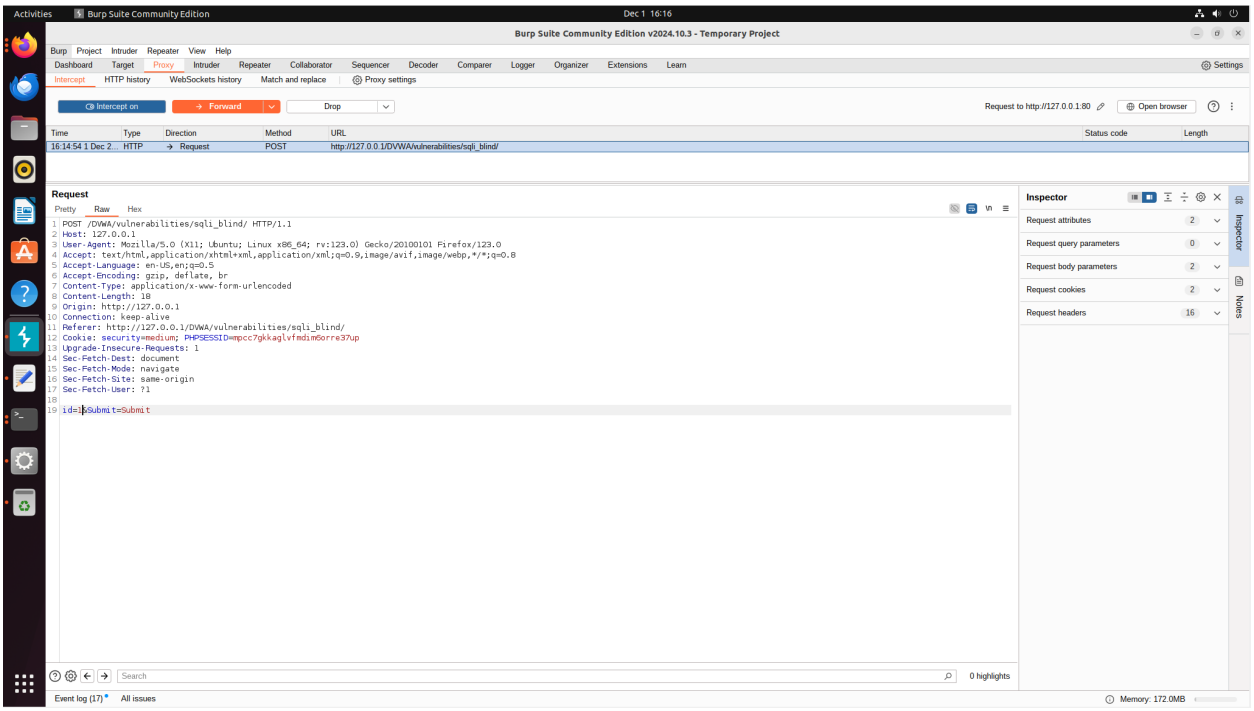

Figure 36: Step1

2. **Step 2:** Replace the previous code with

```
id=1 and sleep(5)&Submit=Submit
```

like (Figure 37)

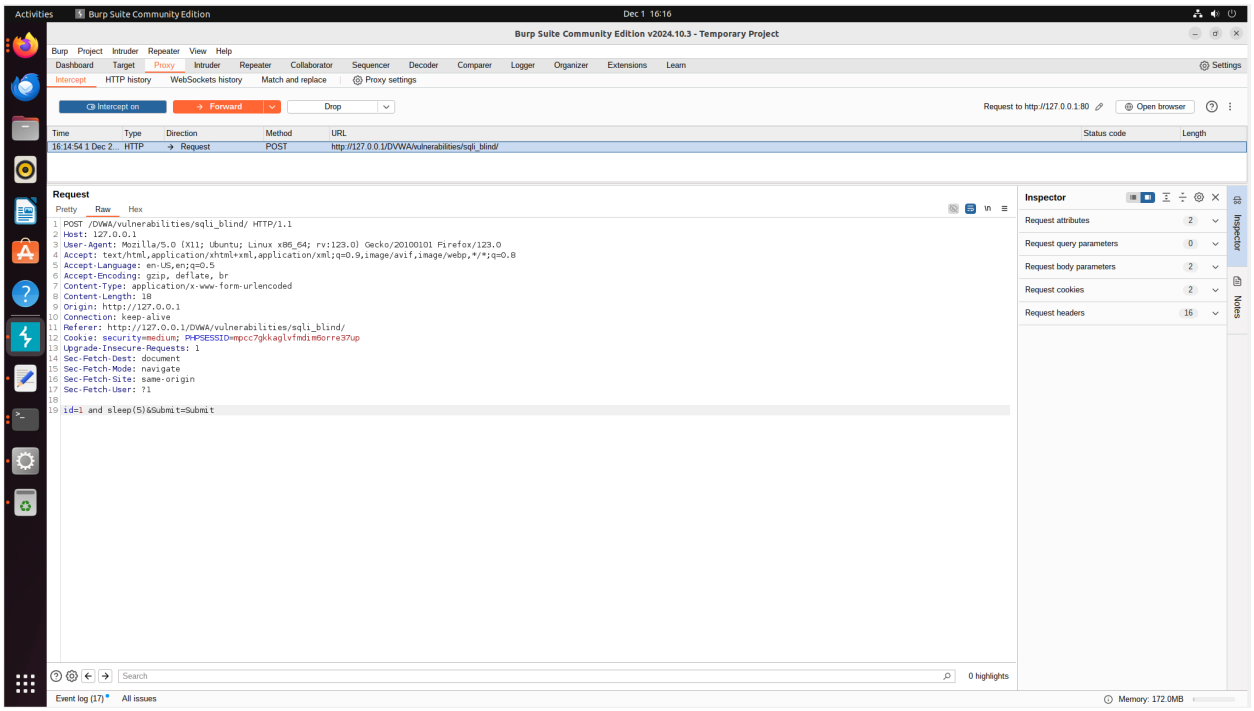

Figure 37: Step2

3. **Step 3:** Click submit that the program has a 5 second delay then outputs User ID is Missing from the database (Figure 38)

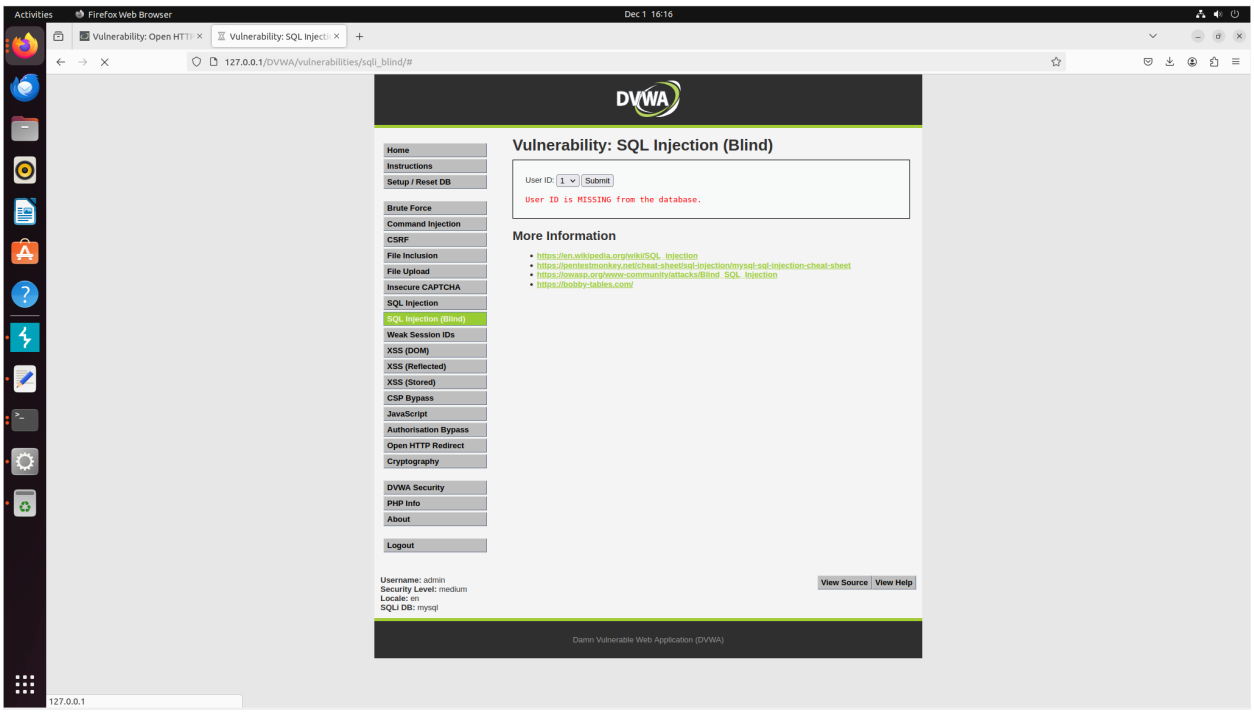

Figure 38: Step3

- **Related OWASP Top 10 Vulnerabilities:**
  - A3: Injection

- # References

[1] [CyptoCat], "8 - Blind SQL Injection (low/med/high) - Damn Vulnerable Web Application (DVWA)" YouTube, 2021. `https://www.youtube.com/watch?v=uN8Tv1exPMkA`.

10 Weak Session ID

- **How does this feature normally work?**
  Session Ids are used to track and maintain the users state including if they are logged. Session IDs are usaly random and unique making it difficult to predict,

- **What does it take to exercise the vulnerability?**
  Analyzing the session ID in DVWA we can see that its using epoch time but a weak session ID with a pattern allow the attacker to guess and forge a session ID.

- **How did the feature work differently than normal use?**
  Instead of ensuring security though randomness, we have a predictable session ID. This allows an attacker to access user session by guessing their ID's.

- **Why did this work differently?**
  The system does not encode session ids and does not use a secure random generator to create session IDs. We are relying on a deterministic value for the session allowing for brute force and hijacking.

- **Why should we care about this vulnerability?**
  It allows for a few things including session hijacking where a attacker and impersonate a real user. This could lead to Unauthorized access and actions within the system.

- **How can we fix the vulnerability?**

  1. cryptographically secure session ID's
  2. Using session timeout limits
  3. Regeneration of session id when doing actions
  4. Secure cookies

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** Access the website and push generate. Then push F12 and open up storage (Figure 39) now click the generate button again, and refresh the storage and we can see that it has incremented (Figure 40). Just to make sure we can repeat this process and we can see that its incrementing and not random

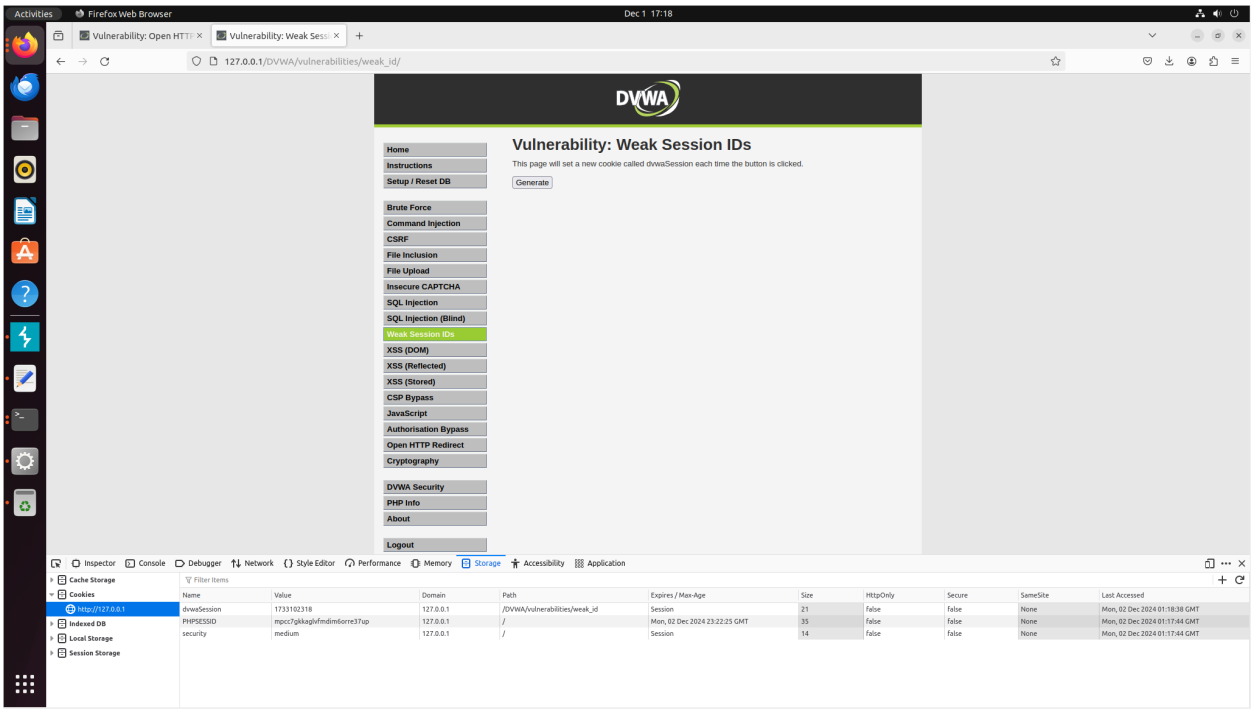

Figure 39: Step1

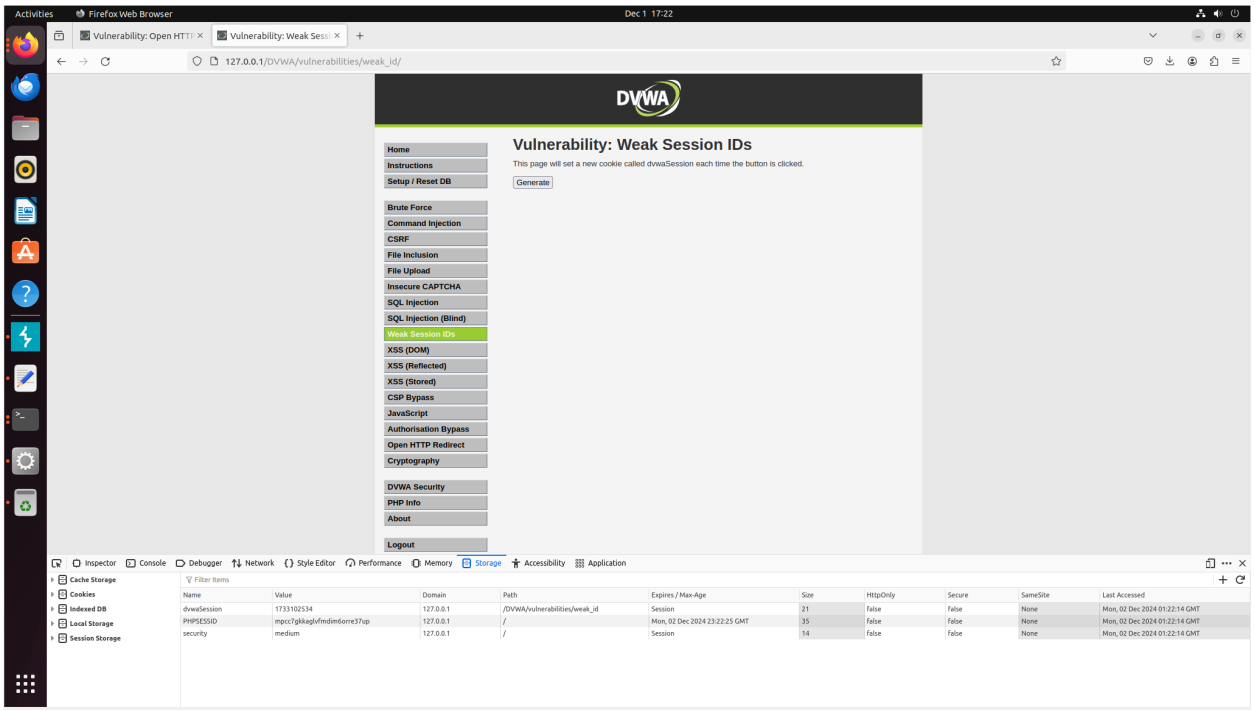

Figure 40: Step1

2. **Step 2:** Now that we can see its not random we can try to figure out what system its using. Using a simple good search we can find that its using Epoch. We can check this by entering one of our generated cookies into an Epoch Converter and we get a output that makes sense (Figure 41)

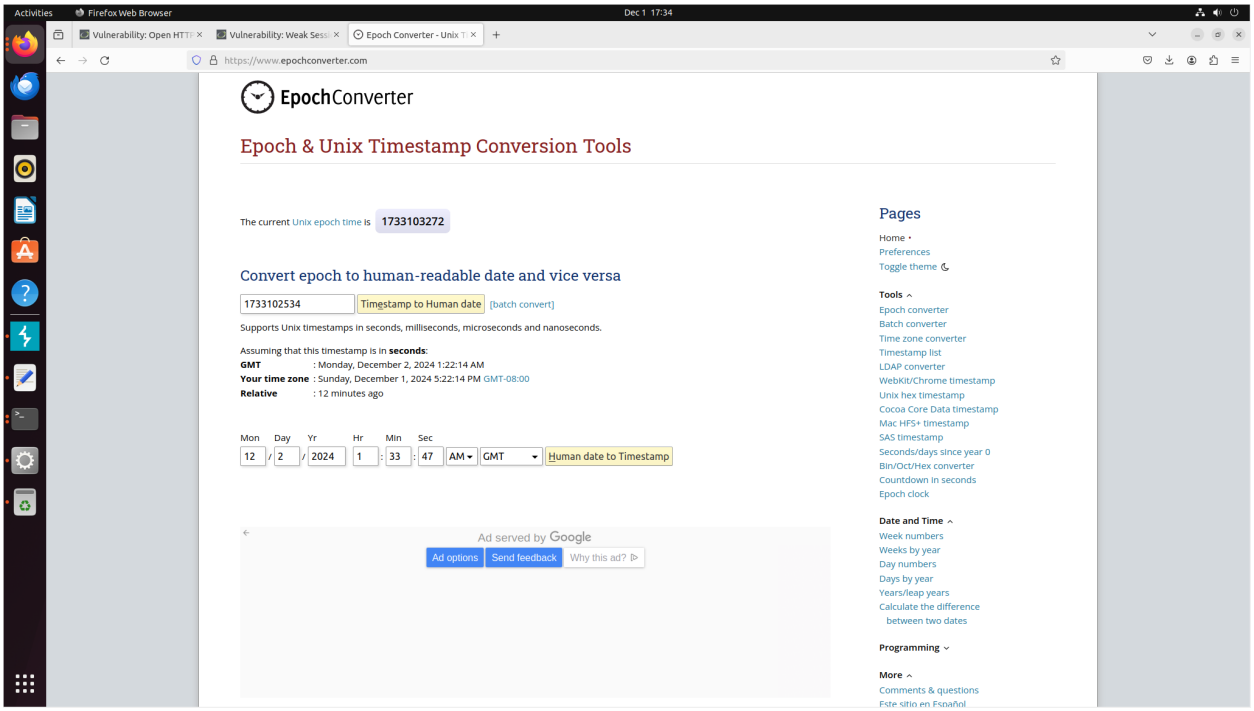

Figure 41: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A2: Broken Authentication

- **References**

11 XSS (DOM)

- **How does this feature normally work?**
  Language section the allows users to select their preferred language. In a secure application this process is on the client side. This means its not exposing its self to external inputs or execution.

- **What does it take to exercise the vulnerability?**
  We are able to modify the URL to include malicious inputs this includes a simple script like

```
<script>alert(1)</script>
```

or it allow for HTML injections like

```
?default=English>/option></select><img src='x'
   onerror='alert(1)'>
```

Both of these exploit the unsafe handling of the inputs within the DOM making the code to execute

- **How did the feature work differently than normal use?**
  Instead of changing to the selected language the application executes the code
- **Why did this work differently?**
  The application includes the user input in the DOM with sanitation.
- **Why should we care about this vulnerability?**
  There are a number of reasons it could lead to session hijacking, phishing, and data theft]
- **How can we fix the vulnerability?**

  1. sanitize user input before its sent to DOM
  2. Use Content security polices
- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** When opening the web app and selecting and submitting english we can see the HTML link is

```
link\?default=English
```

  (Figure 42) We can now try to input after the English to add a script.

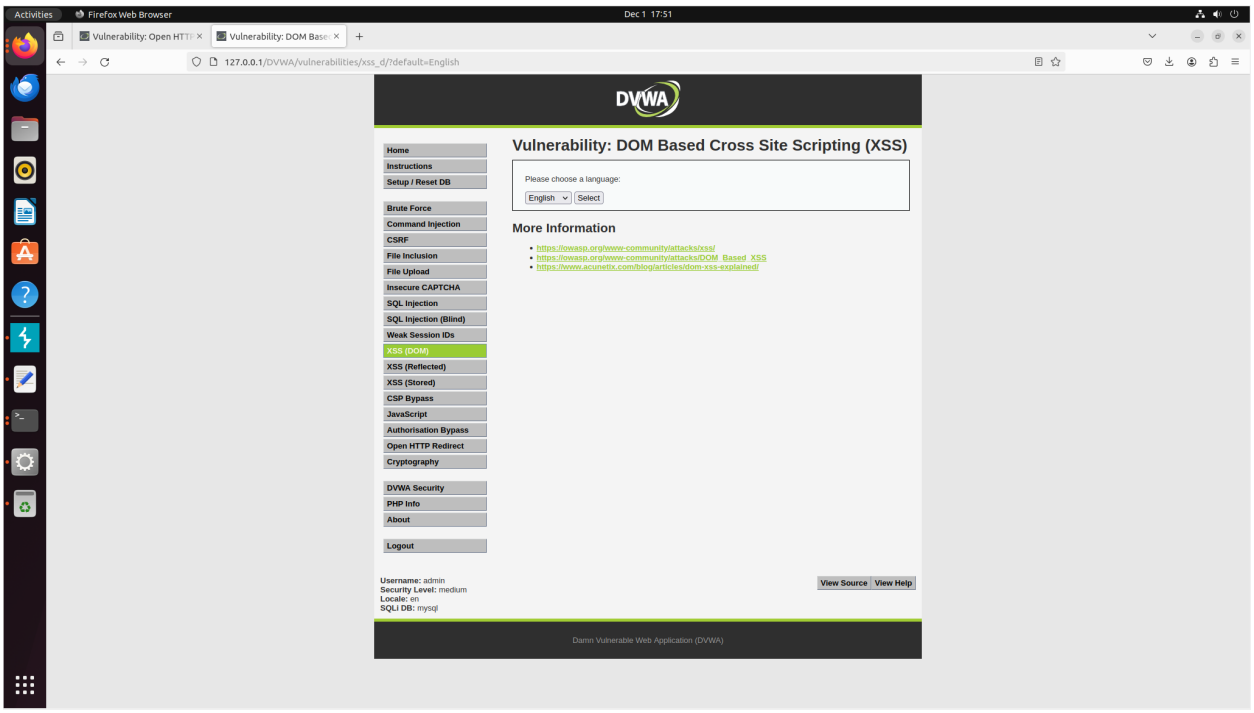

Figure 42: Step1

2. **Step 2:** This can be done by adding

```
1        ?default=English>/option></select><img src='x'
            onerror='alert(1)'>
```

like (Figure 43) When clicking enter we get the alert shown in (Figure 44) meaning we had a successful code execution.

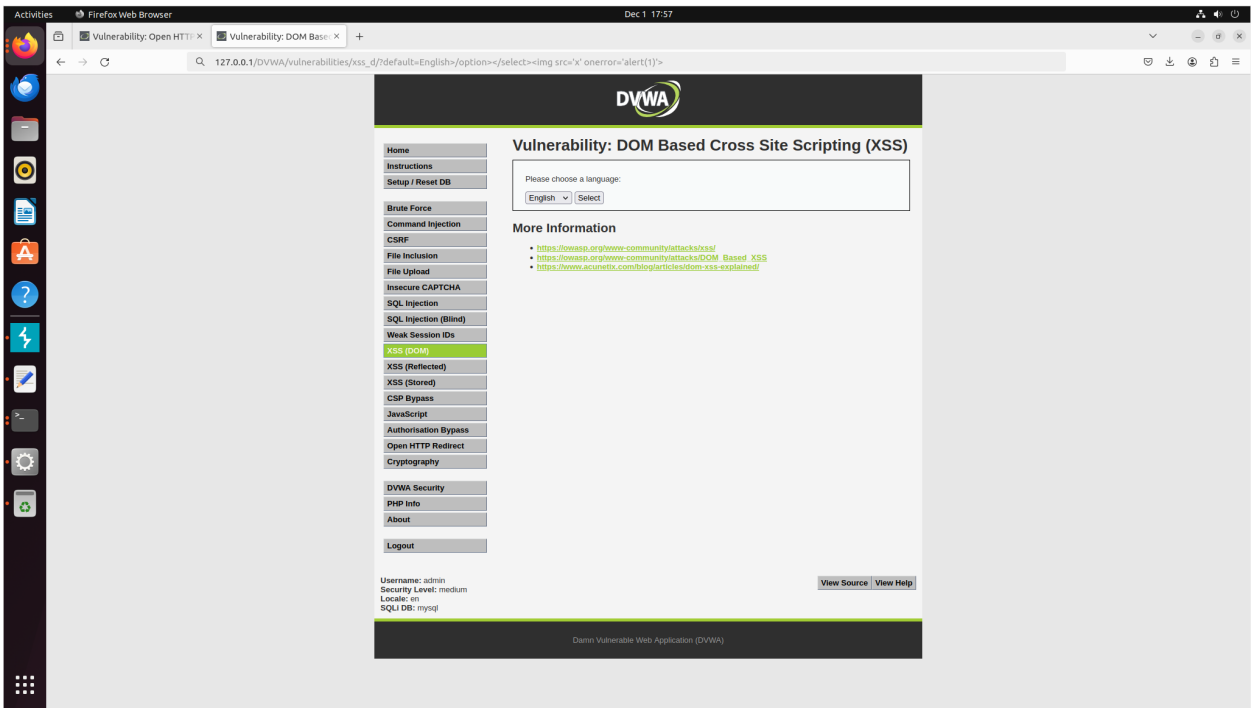

Figure 43: Step2

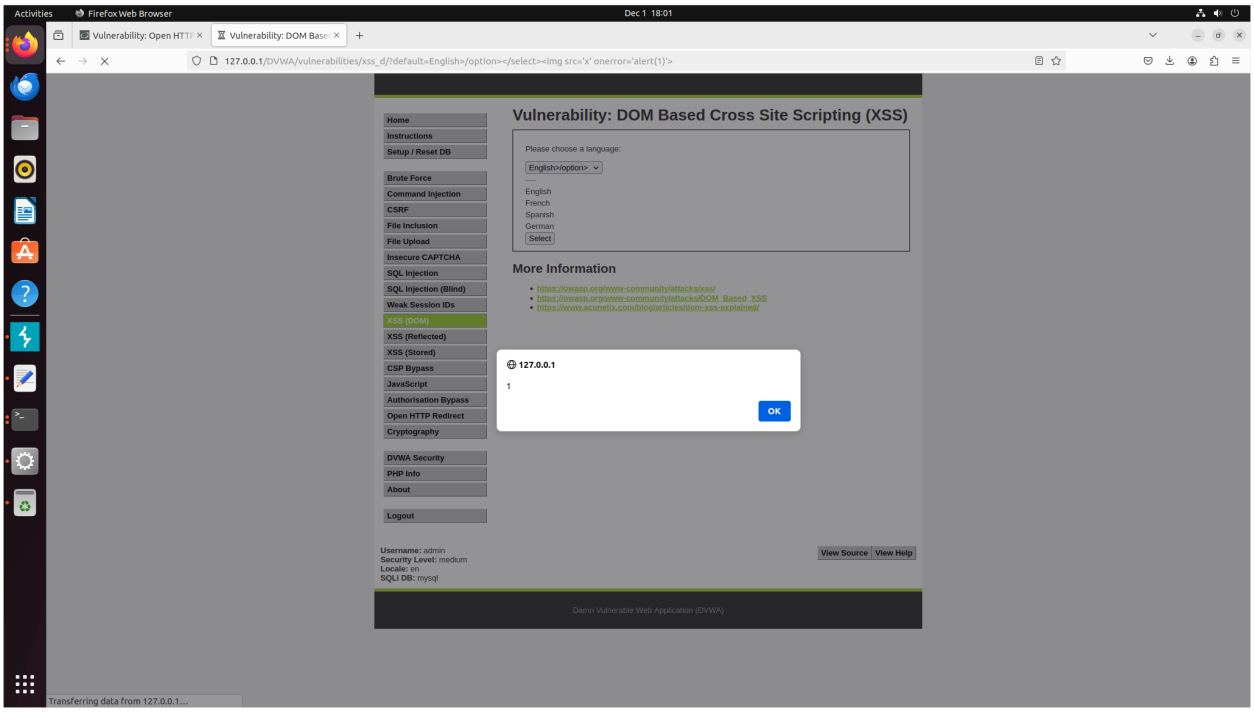

Figure 44: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A7: Cross-Site Scripting (XSS)

- **References**

[1] [CyptoCat], "10 - XSS (DOM) (low/med/high) - Damn Vulnerable Web Application (DVWA)" YouTube, 2021. `https://www.youtube.com/watch?v=X87Ubv-qDm4`.

12 XSS (Reflected)

- **How does this feature normally work?**
  The application expects the user to enter their name into a input filed. IT then displays back hello, [Name]."

- **What does it take to exercise the vulnerability?**
  We can input JavaScript code into the whats your name input.

```
1        <Script>alert("hello world")</sCript>
```

  the code then is executed resulting in the alert popping up

- **How did the feature work differently than normal use?**
  Normally the input would just display the text but it lakes sensitization so when the script is executed it runs the script opening the alert box with hello world.

- **Why did this work differently?**
  This vulnerability exists because the application failed to sanitize the user inputs. The script tag is processes correctly but not edge cases like sCript. This means its case sensitive.

- **Why should we care about this vulnerability?**
  Using this I would run malicious JavaScript to a number of things steal tokens or sensitive information.

- **How can we fix the vulnerability?**

  1. Input Validation
  2. Use Content security polices

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** Input the

```
1          <Script>alert("hello world")</sCript>
```

into the text box (Figure 45)

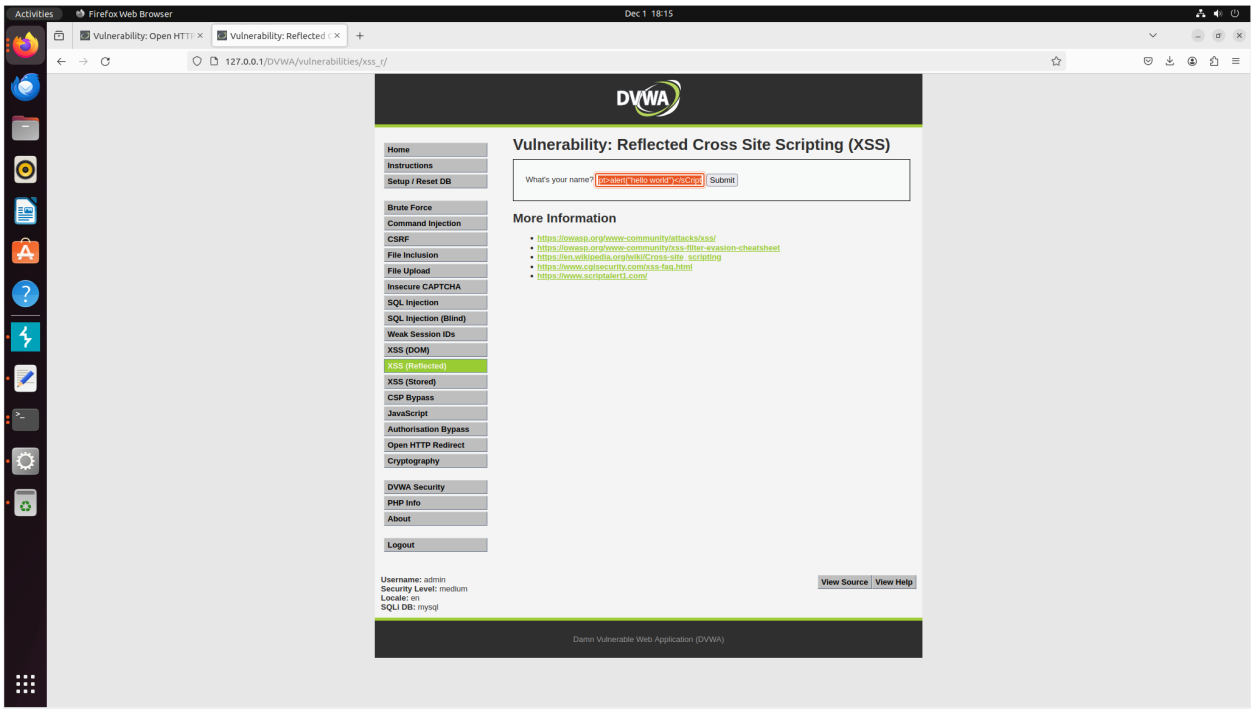

Figure 45: Step1

2. **Step 2:** Click submit and we should get the fallowing display (Figure 46) showing the script was ran

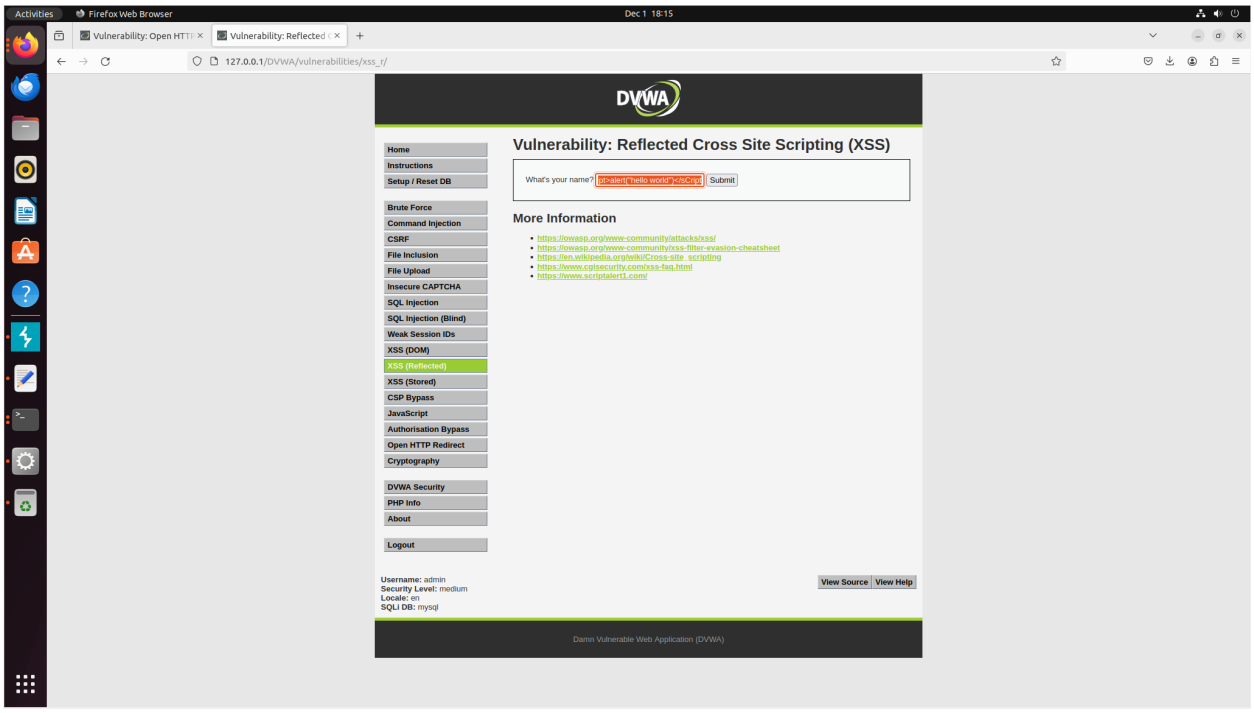

Figure 46: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A7: Cross-Site Scripting (XSS)

- **References**

13 XSS (Stored)

- **How does this feature normally work?**
  The application has a name and message field allowing users to input their name and message to be stored and displayed within the page. Noticeably the name box is a lot less char max size then that of the message box.

- **What does it take to exercise the vulnerability?**
  First inspect eh page and change the HTML for max length for the name box to be larger then 10. This then allows us to insert scripts into the name fields where no checks are happening.

- **How did the feature work differently than normal use?**
  Normally the name name and message would be stored as plain text and sanitized but in this case it seems there was an oversight with no sanitation within the name box allowing for us to change the max length and input scripts.

- **Why did this work differently?**
  The vulnerability works because the application failed to sanitize the user input before storing it rendering it as code.
- **Why should we care about this vulnerability?**
  It all for injection of malicious scripts that can again steal cookies and preform actions. When scripts are injected it allows for opening of attack vectors.
- **How can we fix the vulnerability?**

  1. Input Validation
  2. Use Content security polices
  3. stricter limit of input lenght
- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** After a bit of testing and sending scripts though the message section we can change our focus to the name section. Here we can see that there is a max length. To get around this we will right lick and inspect element and change the maxlength value from 10 seen in (Figure 47), to some larger value allowing for larger inputs. (Figure 48)

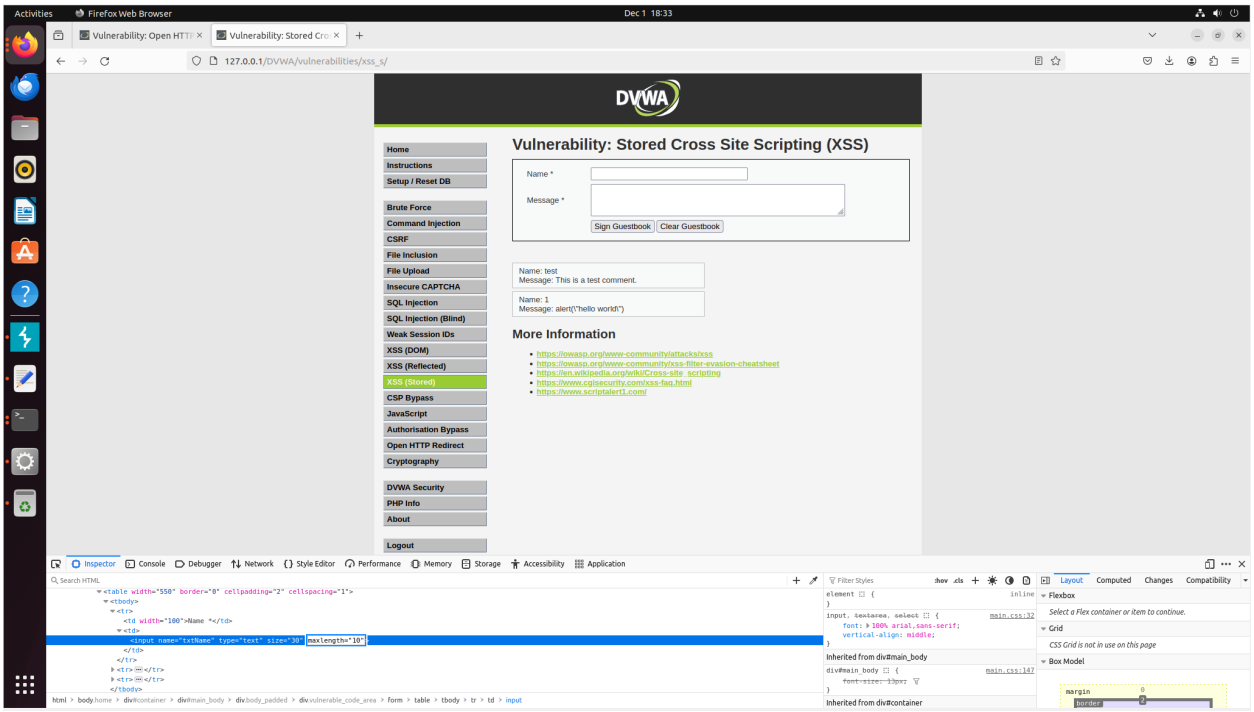

Figure 47: Step1

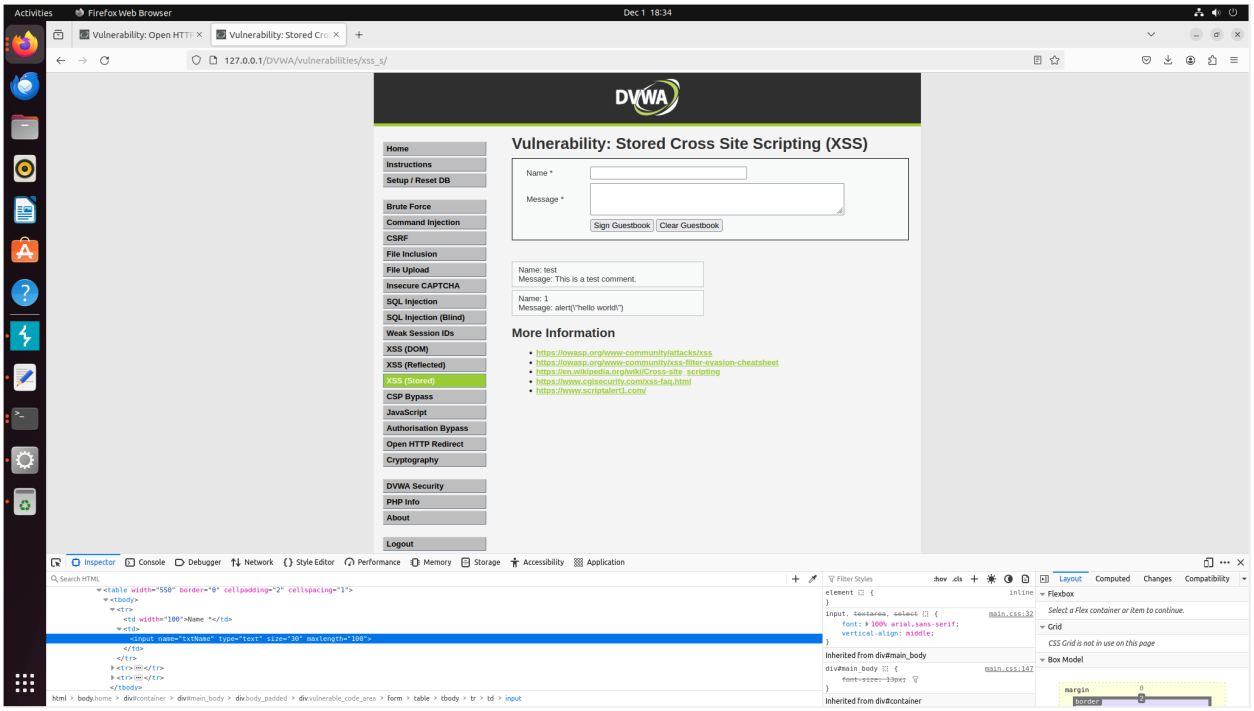

Figure 48: Step1

2. **Step 2:** This then allow for us to input a script much like we did in XSSReflect

```
1          <Script>alert("hello world")</sCript>
```

seen in (Figure 49). Now when clicking submit we now see our script has been run (Figure 50)

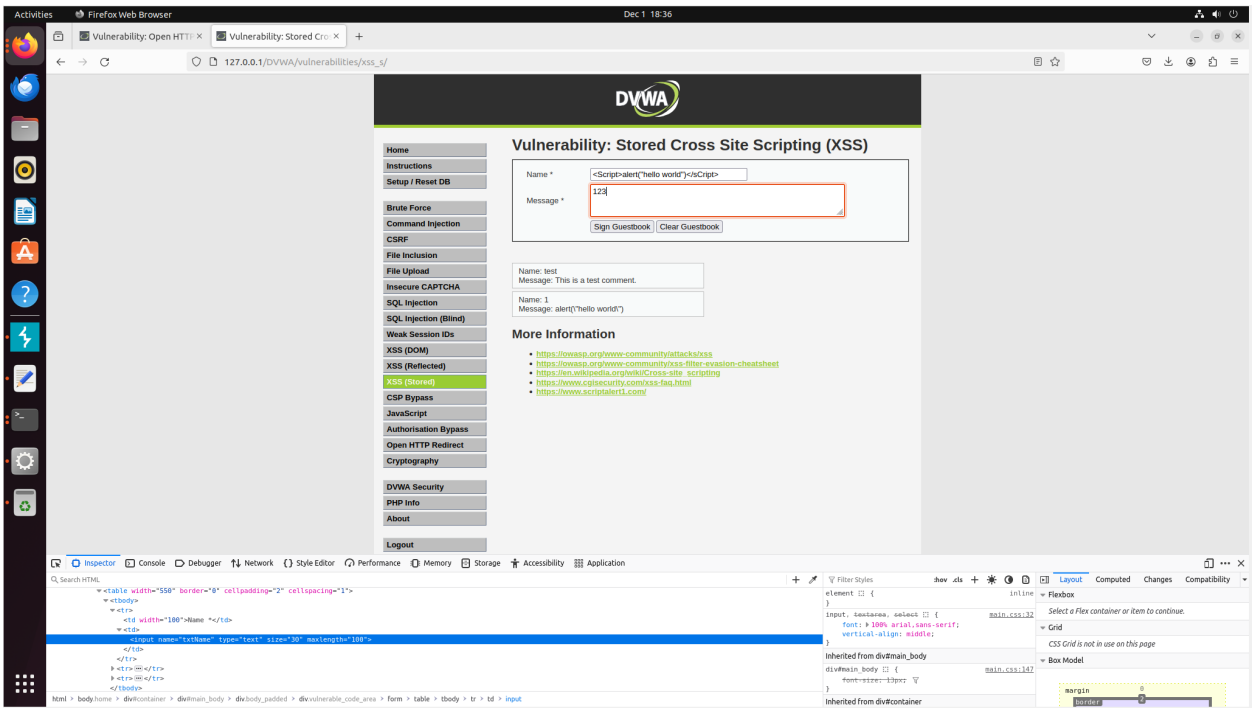

Figure 49: Step2

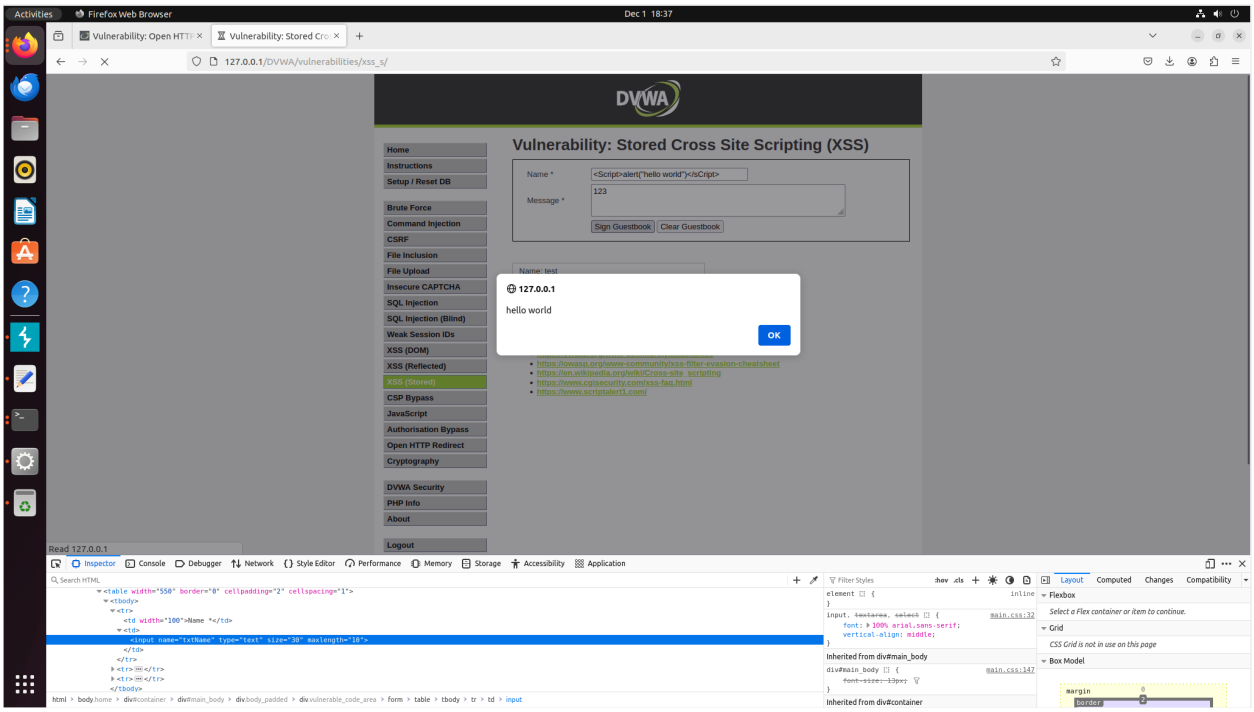

Figure 50: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A7: Cross-Site Scripting (XSS)

- **References**

14 Content Security Policy (CSP)

- **How does this feature normally work?**
  CSP is designed to prevent XSS by restricting h sources form where the scripts can be loaded and executed. This is done by setting the http headers, this means that only trusted sources can run scripts.

- **What does it take to exercise the vulnerability?**
  In this case the vulnerability is in the fact that there is some dev code left within the application meaning that we can just use this key to act as a trusted source and run the code. This can be seen in the commented (Figure 51)

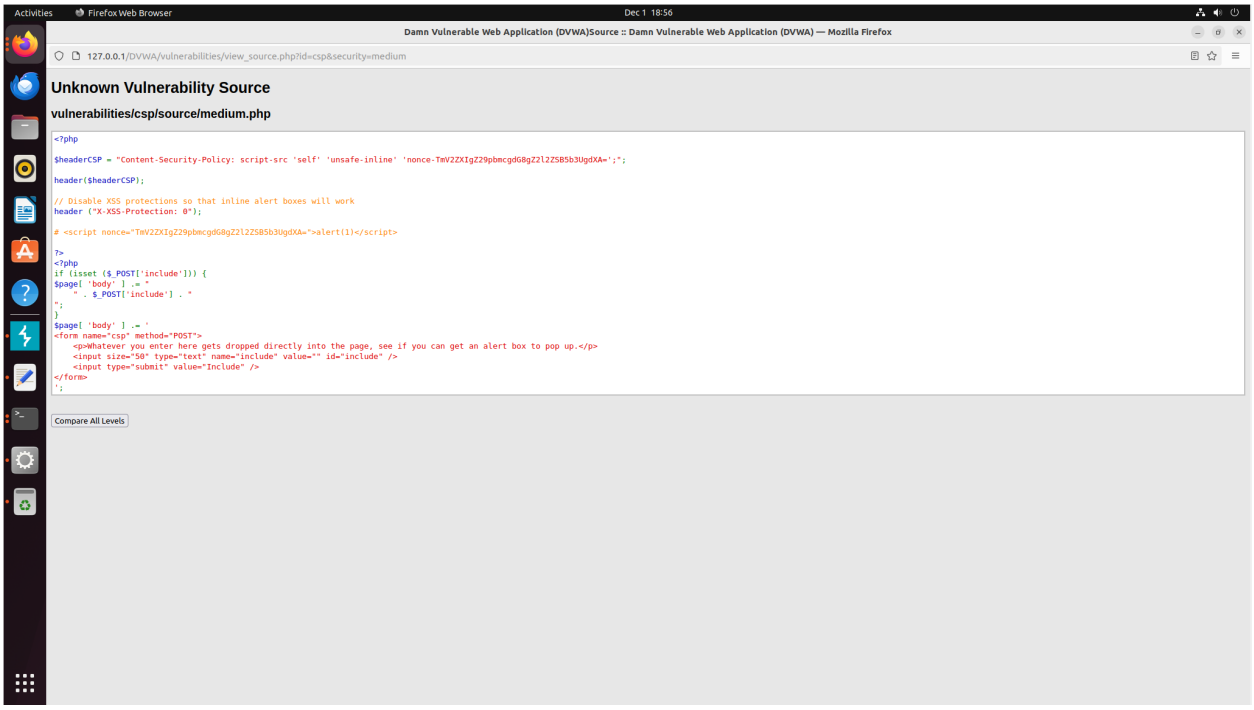

Figure 51: Commented key

it should also be noted that it was also added in the headder allowing access to this key

- **How did the feature work differently than normal use?**
  In this case the CSP should have prevented the inline javascript but when running with the nonce we are able to get it to execute scripts.
- **Why did this work differently?**
  This worked because the CSP did not block our inline script. Normally it would be we have the bypass key in a way allowing for us to execute code.
- **Why should we care about this vulnerability?**
  The bypassing of CSP allows for XSS attacks to happen
- **How can we fix the vulnerability?**

  1. Removal of dev comments
  2. Use of strict-dynamic
- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** When looking at the code we can see that there is the value for the nonce in the header and in the code as a comment (Figure 51). This means that if we run a script with this nonce we are able to bypass it acting as a developer and running a script.

  2. **Step 2:** To do this I copied the script in the comments in the and pasted it into the box like so (Figure 52). Now when clicking submit we now see our script has been run we get the code to run like (Figure 53)

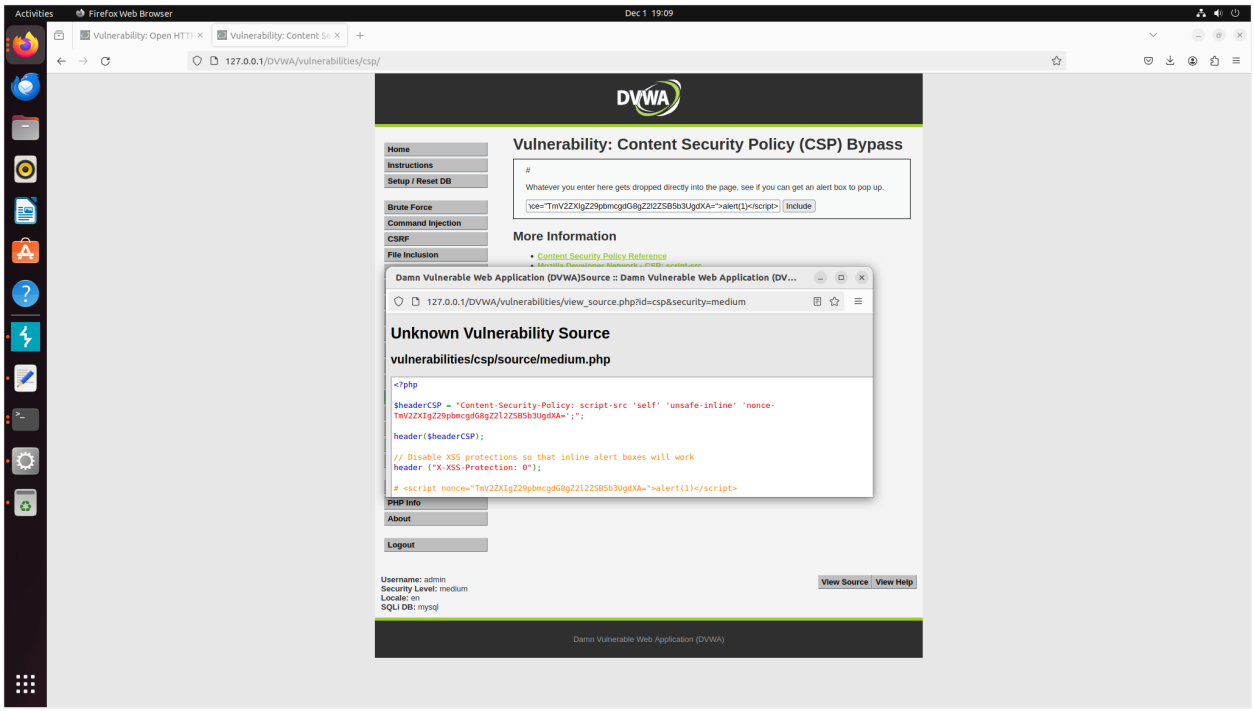

Figure 52: Step2

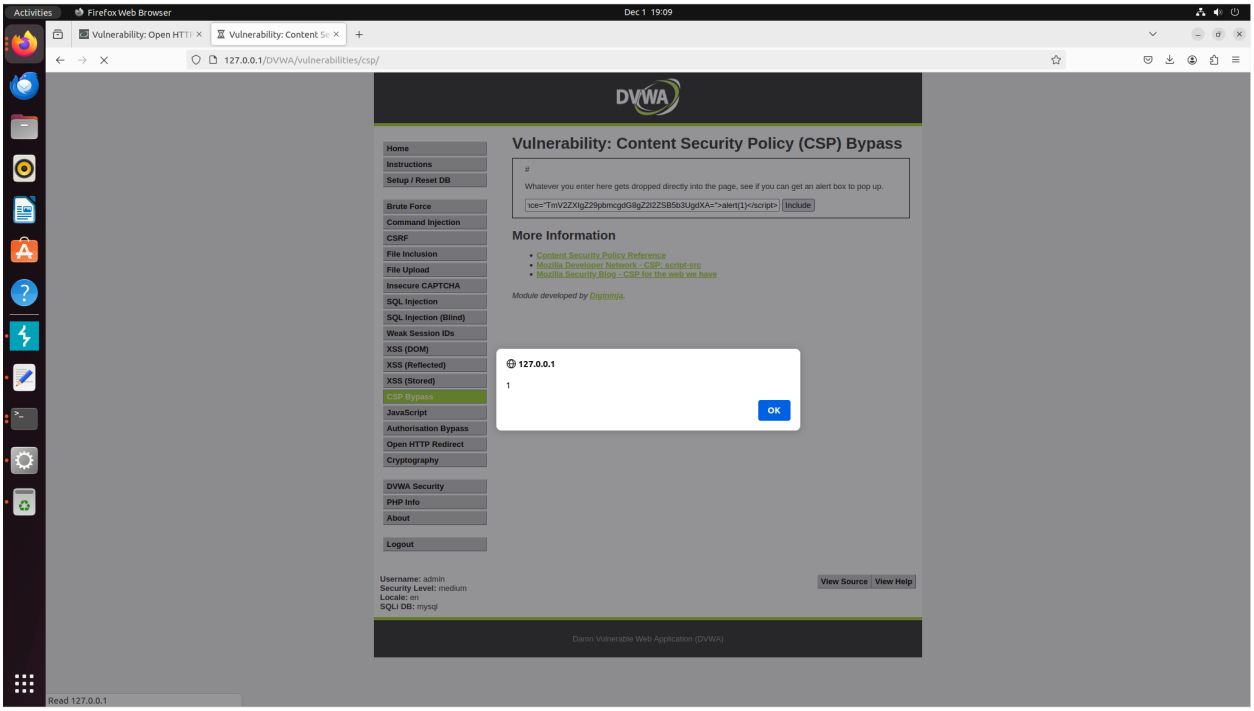

Figure 53: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A7: Cross-Site Scripting (XSS)

- **References**

15 JavaScript

- **How does this feature normally work?**
  Normally JavaScript encryption or functions work by taking text and changing it then its inspected by the backend. This means that manual inspection of the source code would not tell you everything you needed to know.
- **What does it take to exercise the vulnerability?**
  In this case we can open the browser debugger and inspect medium.js to look at the code and reverse engineer the desired inputs for check to get back success. This requires you to intercept the Http request and some simple reverse engineering.
- **How did the feature work differently than normal use?**
  This worked differently because the input validation meaning that the input was not assumed to be modified at any point. Never trust the user fully.
- **Why did this work differently?**
  In this code we are doing a lot client side with no manipulation checks meaning that logic being exposed in the browser can mean that any change to the input for a check can just be replicated.
- **Why should we care about this vulnerability?**
  Bypassing the input validation means that it could lead to unauthorized actions or data manipulation.
- **How can we fix the vulnerability?**

  1. Secure data transmission.
  2. Never trust the user
  3. Secure functions not on the same page as the JavaScript file facing outwards.

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** We will first try to look at the code and figure out what is being done within the code. This is done by pushing F12 and navigating to the to the debugger and looking at the medium.js file. Here we can see something interesting our input is being changed using the do something function and being outputted

(Figure 54). Now using some reverse engineering knowing the desired input we can tell that XX and XX are added at the beginning and end then the words are scrambled. Now the scrambling part workings thought he for loop we can figure out that output of this .js file would be XXsseccusXX

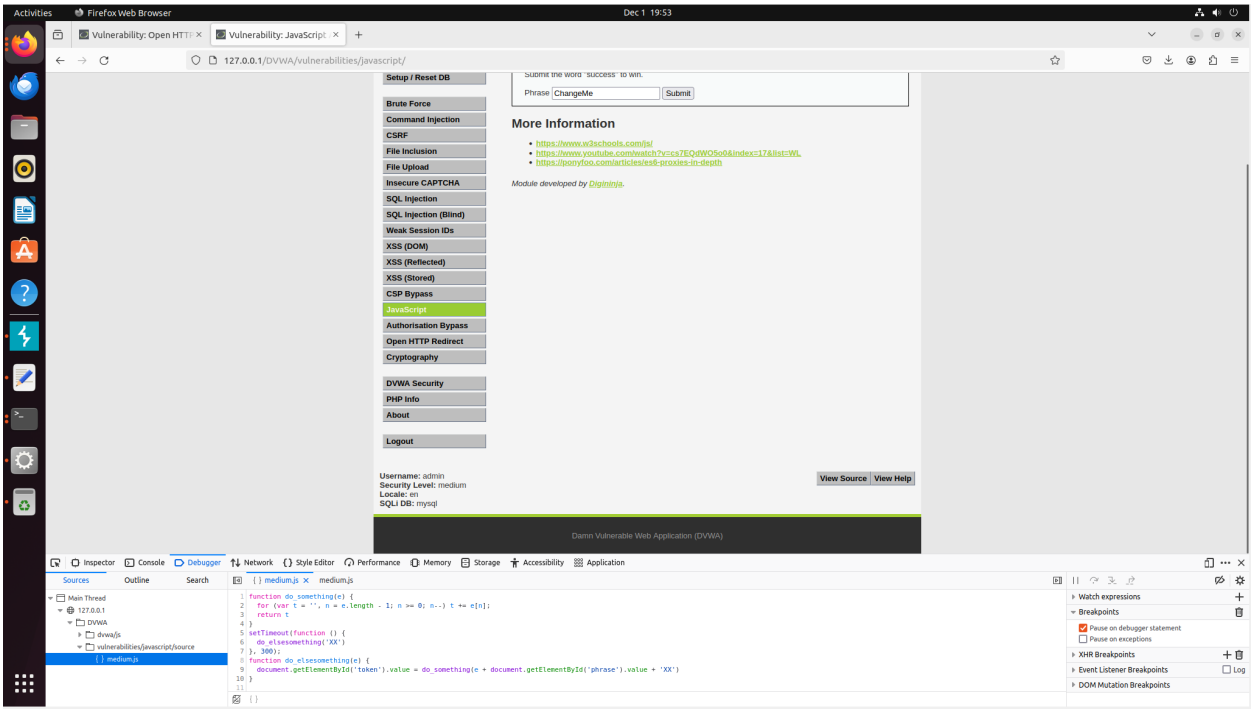

Figure 54: Step2

2. **Step 2:** To check this logic we need to input Success into the input field and start foxy proxy and Burp and push submit. When doing this we can see that the packet is intercepted like (Figure 55)

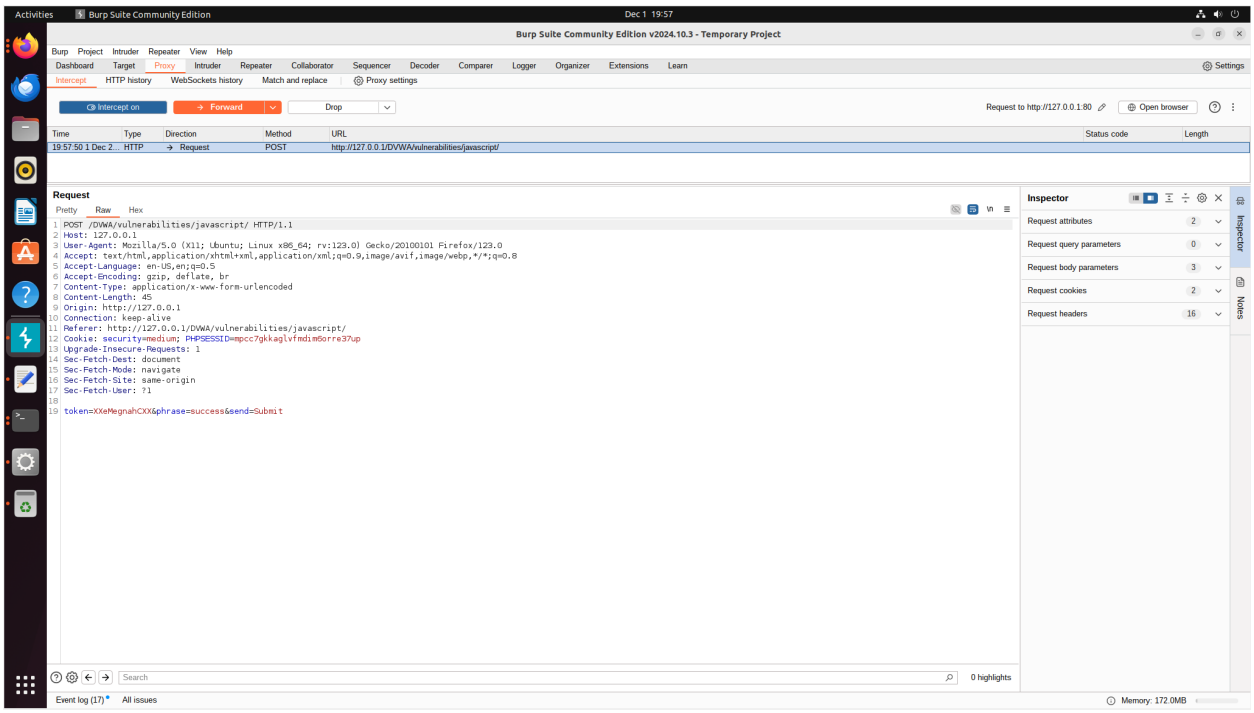

Figure 55: Step2

3. **Step 3:** Now we want to input the output our new scrambled word instead of the existing token like in (Figure 56). Doing this and forwarding the packet we can see a success message on our website now (Figure 57)

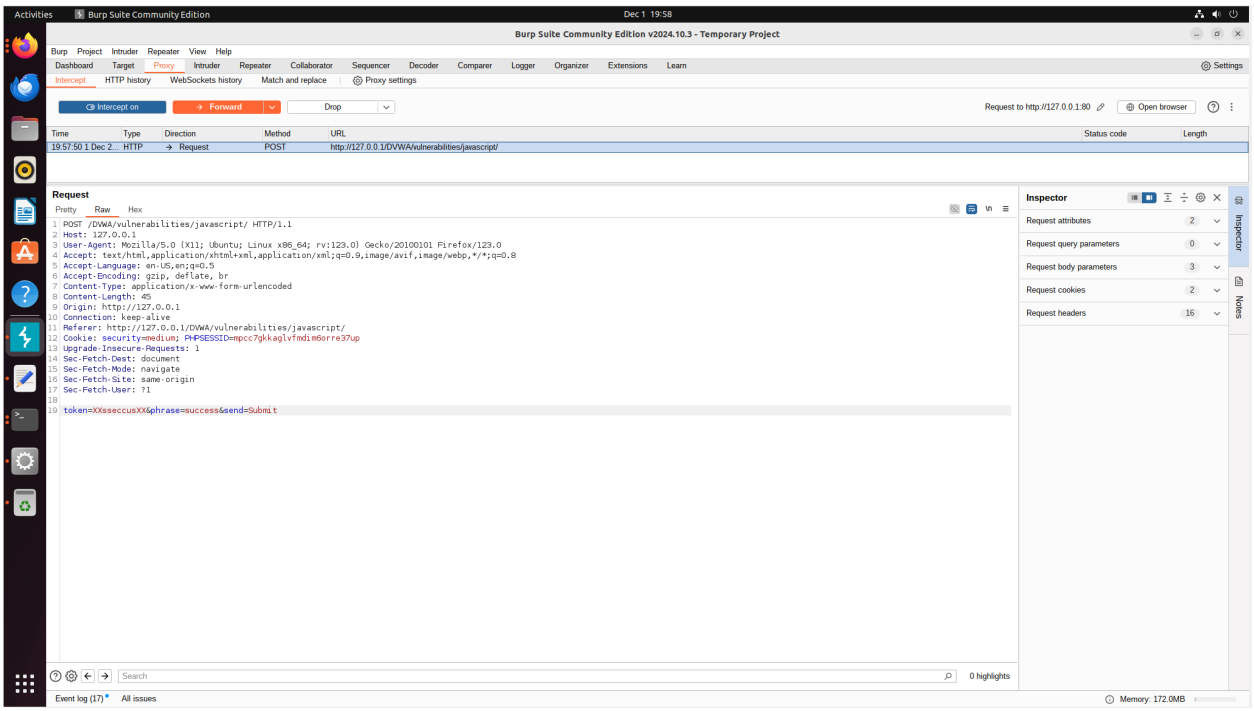

Figure 56: Step3

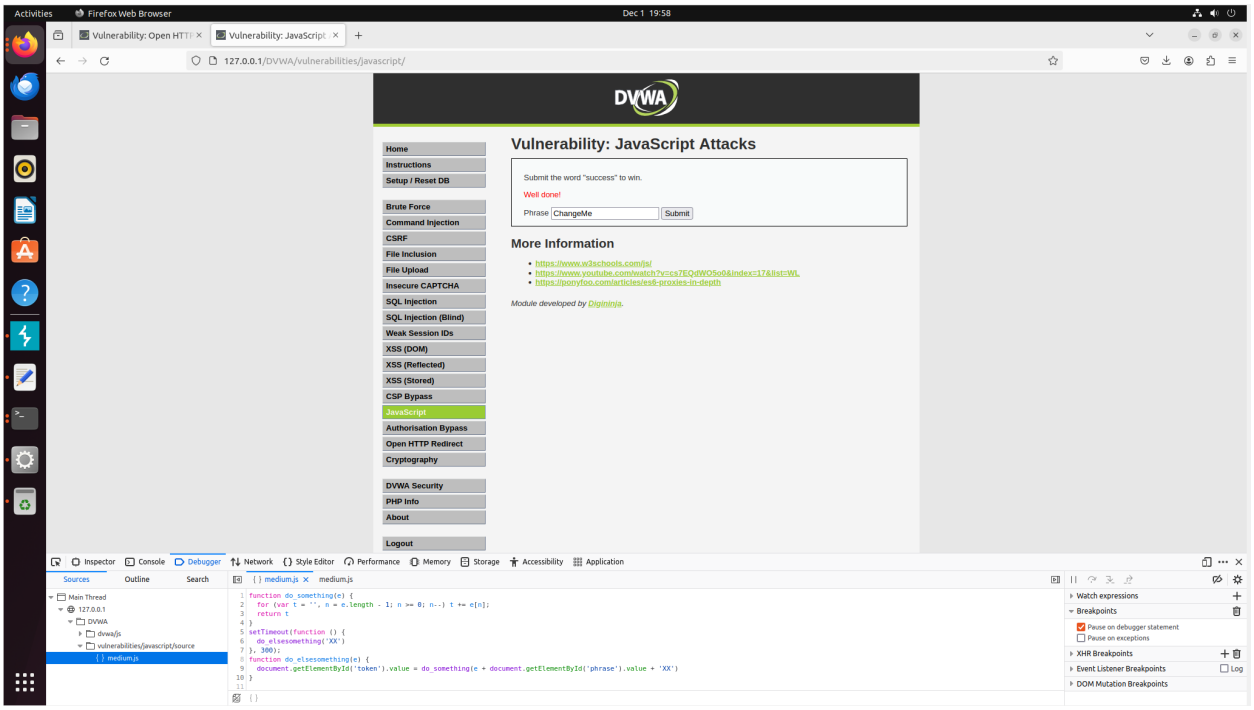

Figure 57: Step3

- **Related OWASP Top 10 Vulnerabilities:**
  - A3: Sensitive Data Exposure
  - A6: Security Misconfiguration

- **References**

16 Authorization Bypass

- **How does this feature normally work?**
  Normally authorization mechanism are used to ensure that users only have access resources and pages they are permitted to view. When a user is logged in it should not have access to the development pages.

- **What does it take to exercise the vulnerability?**
  This requires a log in for a no admin user, and the knowledge of the link that the devtool is linked to (link to restricted page). It also requires you to know the file directly for the PHP file or file directory of the file you wish to access.

- **How did the feature work differently than normal use?**
  This features works differing because appending the path to he PHP file bypasses the access control check within the directory. This means that there is a miss-configuration in the rules allowing unrestricted read access within the dev directory.

- **Why did this work differently?**
  This worked differently due to a failure in access control mechanism allowing for users to read files in a directory they should have access to.

- **Why should we care about this vulnerability?**
  It can lead to Expose sensitive data, Privilege escalation.

- **How can we fix the vulnerability?**

  1. File level authorization
  2. Access control mechanisms

- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** We will need to note down or capture the request to the restricted page. We then will need to login as a user and setup proxy and Burp to test. Now that we have login we can see that we don't see the dev page (Figure 58). Now as user we want to try to access the captured webpage. When doing this as a user with no auth we may get a message like (Figure 59)

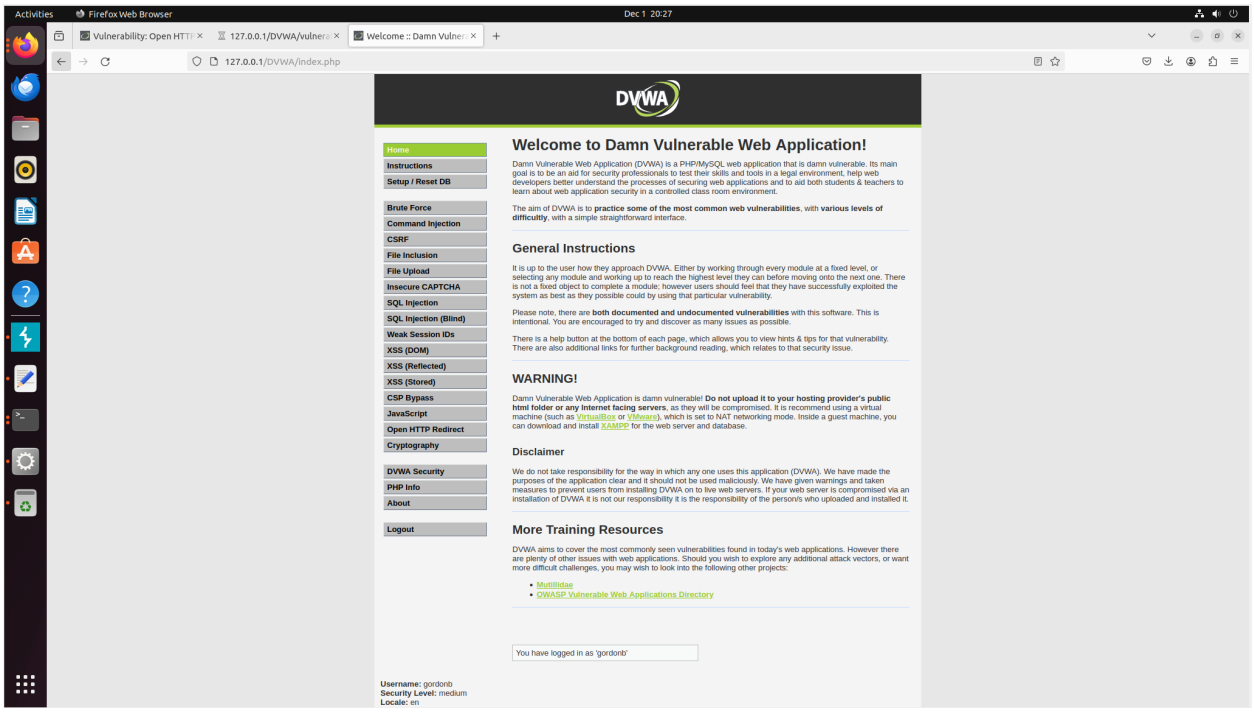

Figure 58: Step1

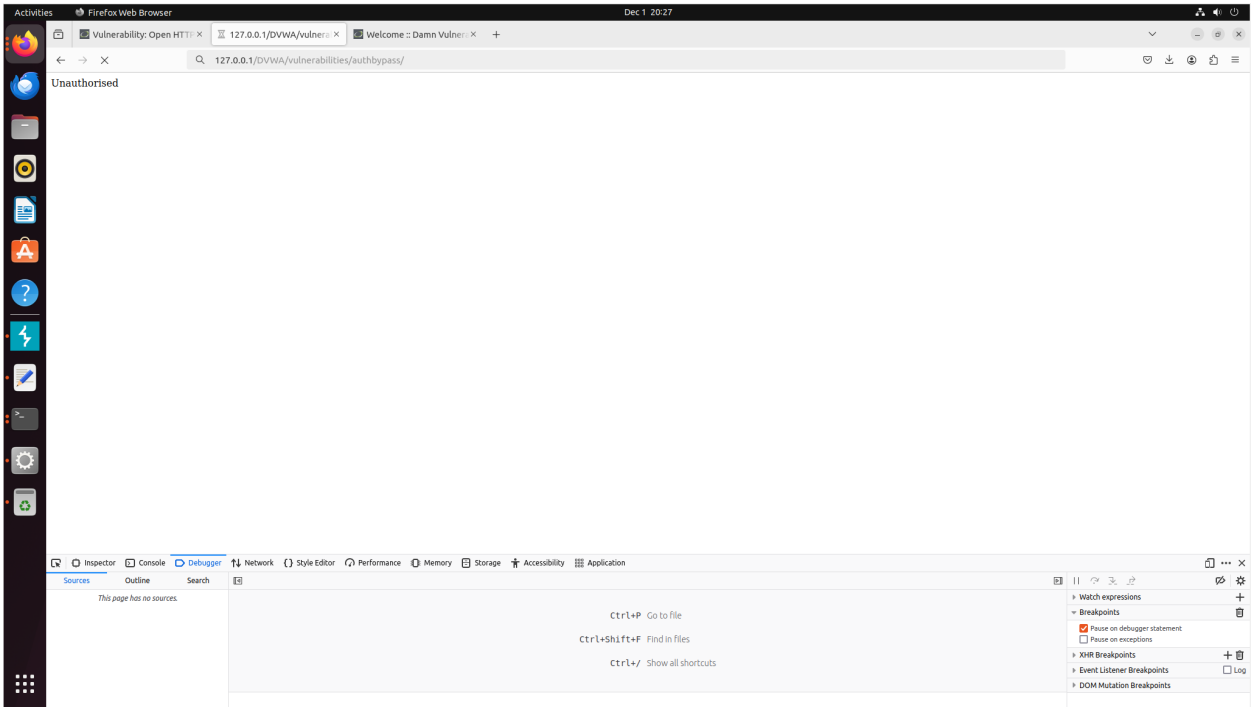

Figure 59: Step1

2. **Step 2:** To check this logic we need to input Success into the input field and start foxy proxy and Burp and push submit. When doing this we can see that the packet is intercepted we can try change the input form a post to a GET and so on. This does not do anything in this case so we can try another method. This method banks on the fact that we are able to read files within this directory without clearance. In order to this we can add

```
1          /get_user_data.php
```

to the end of the URL. Doing this shows us a readable output of the user data (Figure 60)

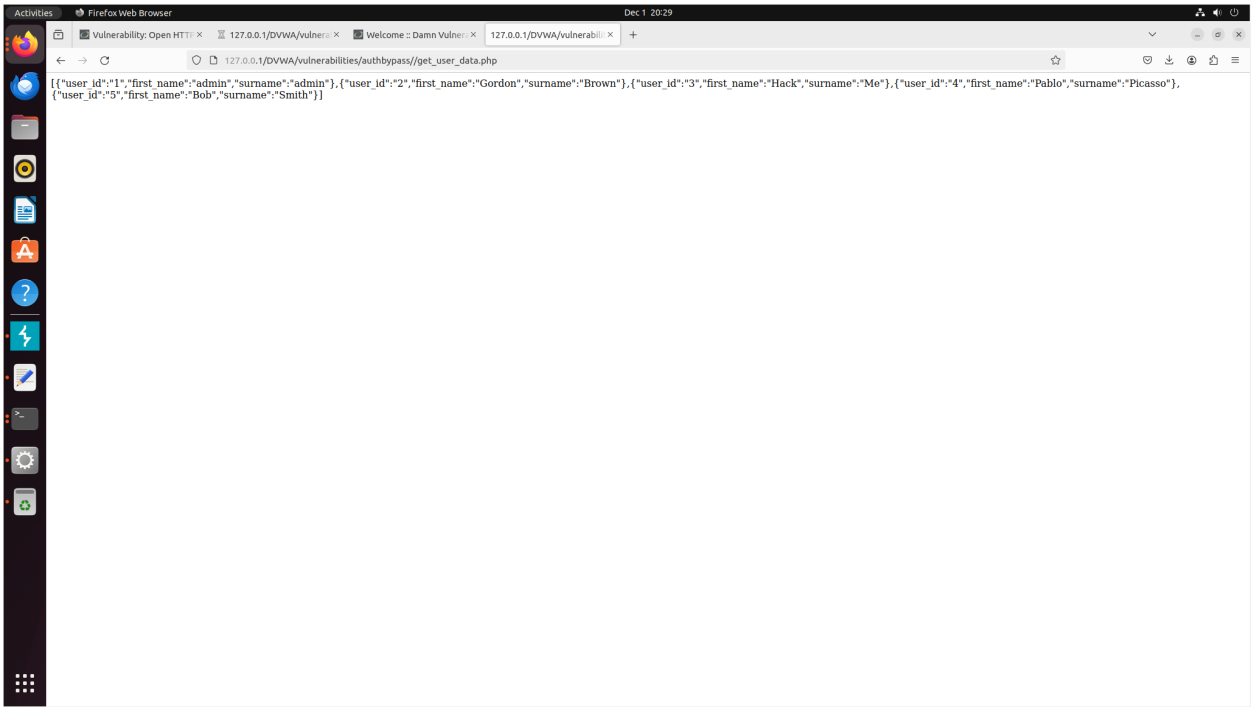

Figure 60: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A3: Sensitive Data Exposure
  - A5: Broken Access Control

- **References**

17 Open Redirect

- **How does this feature normally work?**
  The open redirect allow the user to direct to a different URL based on the application. Normally this redirects you for valid URL after authentication.
- **What does it take to exercise the vulnerability?**
  This exploit uses Burp to intercept the packet being sent to the web, it then changes the redirect parameter to the new redirect URL and then sends the packet.
- **How did the feature work differently than normal use?**
  This features allows the redirection to an external untrusted URL. It bypasses validation checks and instead redirects to the new url.
- **Why did this work differently?**
  This happened because the lack of validation within the application and improper input sanitation. We are able to directly edit the URL.
- **Why should we care about this vulnerability?**
  It could lead to Phishing attacks.
- **How can we fix the vulnerability?**

  1. Whitelist URLS
  2. Validate input
- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** For this we again will be using foxy proxy and Burp. Start both and open the and click quote 1 send the HTTP request. Now when opening the Burp intercept we can see (Figure 64)

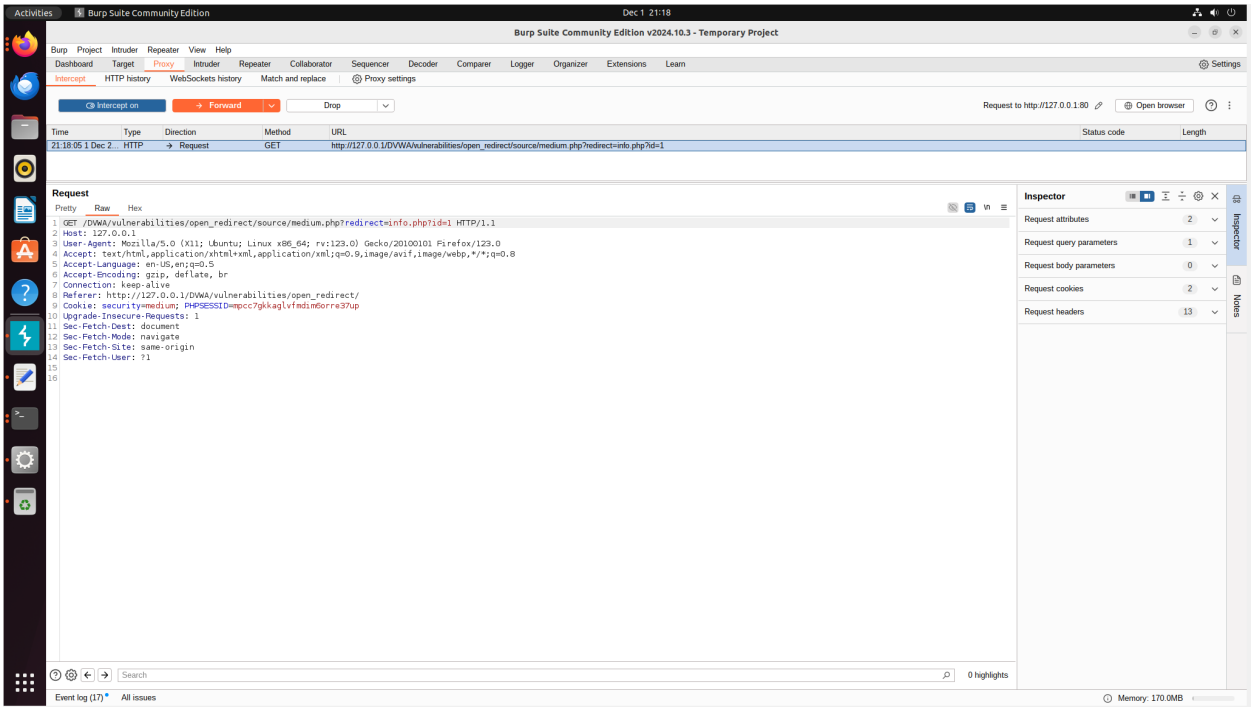

Figure 61: Step1

2. **Step 2:** Here we notice a

```
1          redirect=info.php
```

We now will change this to

```
1          redirect=//google.co.us
```

(Figure 65). Doing this and clicking forward we are no redirected to (Figure 66). Although this isnt a real website we still were able to redirect to a differing webpage.

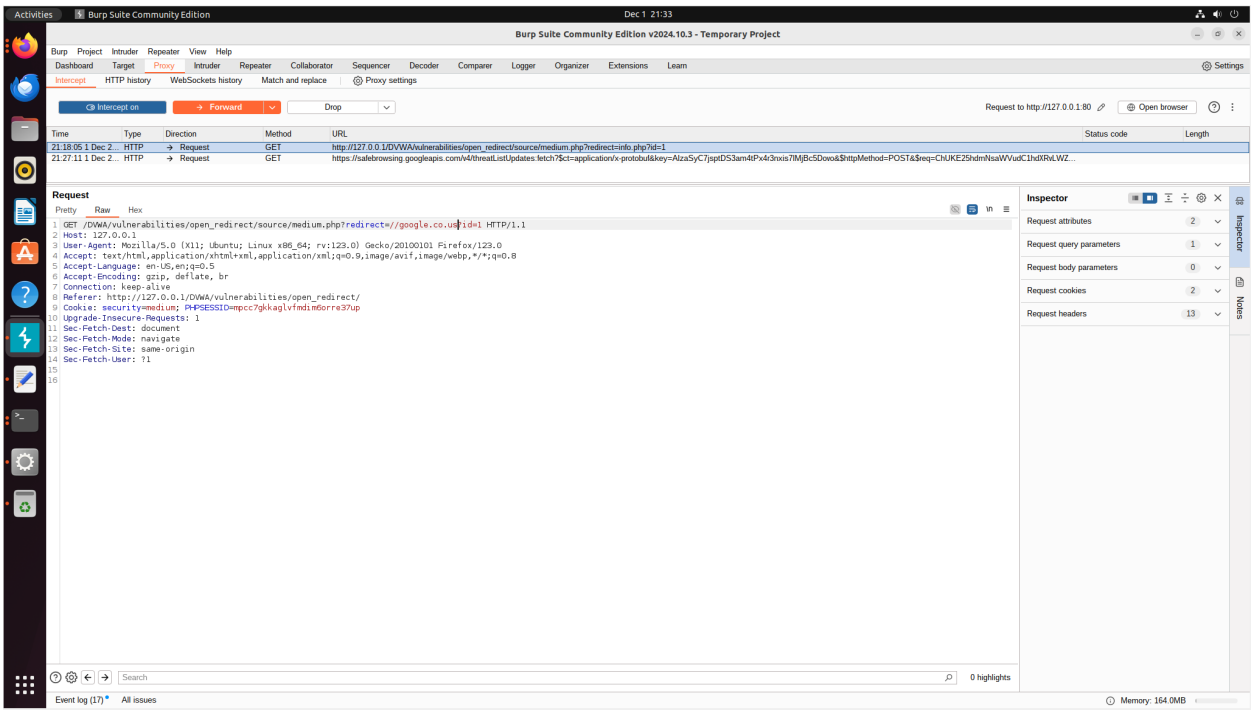

Figure 62: Step2

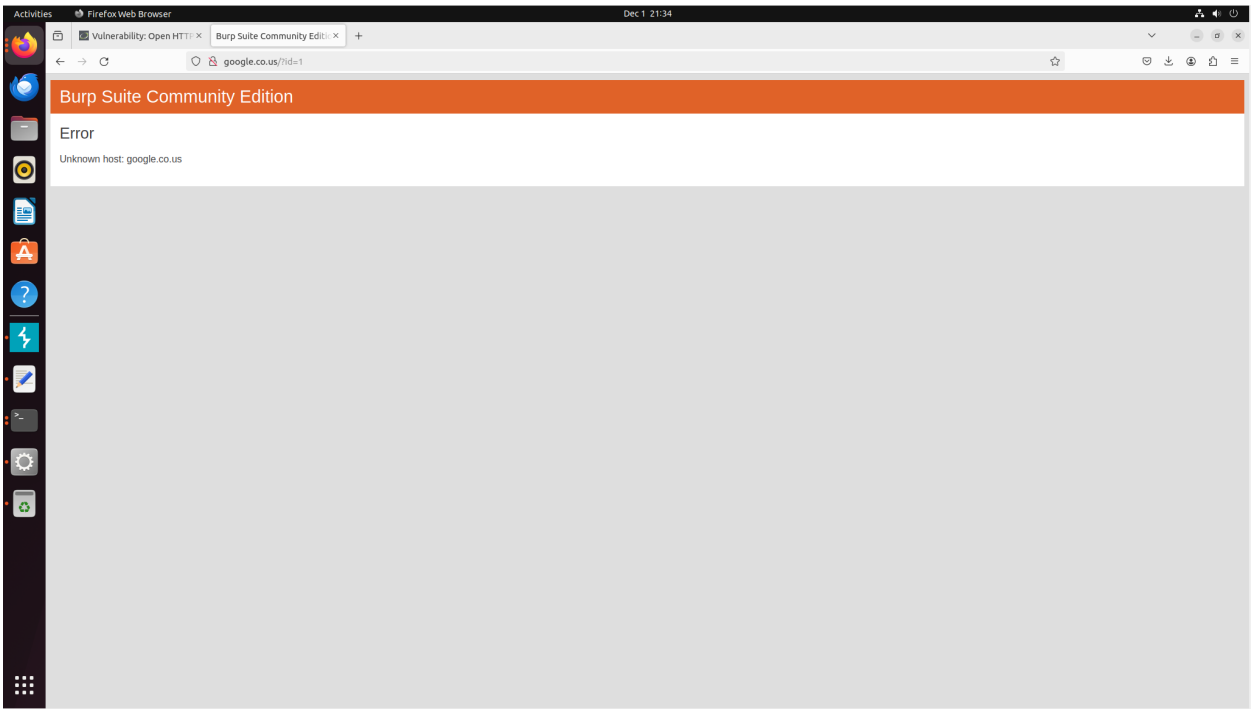

Figure 63: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A10:2021 – Server-Side Request Forgery (SSRF)

- **References**

18 Open Redirect

- **How does this feature normally work?**
  The open redirect allow the user to direct to a different URL based on the application. Normally this redirects you for valid URL after authentication.
- **What does it take to exercise the vulnerability?**
  This exploit uses Burp to intercept the packet being sent to the web, it then changes the redirect parameter to the new redirect URL and then sends the packet.
- **How did the feature work differently than normal use?**
  This features allows the redirection to an external untrusted URL. It bypasses validation checks and instead redirects to the new url.
- **Why did this work differently?**
  This happened because the lack of validation within the application and improper input sanitation. We are able to directly edit the URL.
- **Why should we care about this vulnerability?**
  It could lead to Phishing attacks.
- **How can we fix the vulnerability?**

  1. Whitelist URLS
  2. Validate input
- **Steps to Exercise the Vulnerability:**

  1. **Step 1:** For this we again will be using foxy proxy and Burp. Start both and open the and click quote 1 send the HTTP request. Now when opening the Burp intercept we can see (Figure 64)

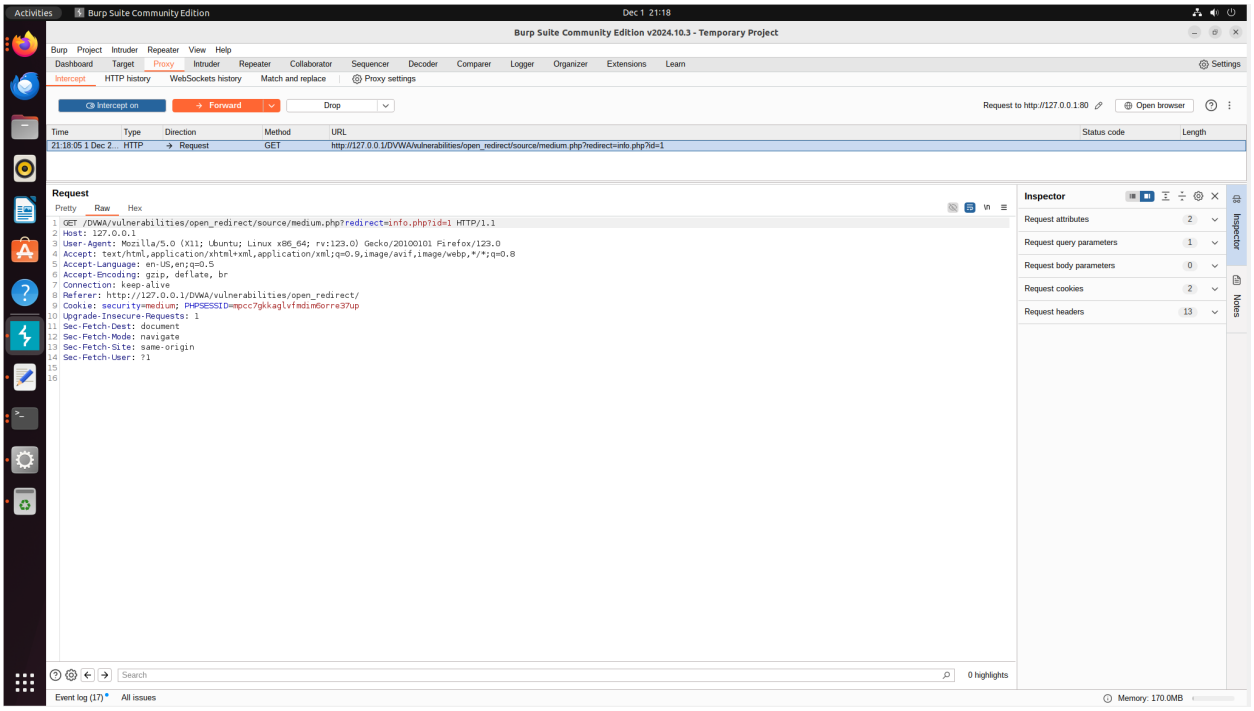

Figure 64: Step1

2. **Step 2:** Here we notice a

```
redirect=info.php
```

We now will change this to

```
redirect=//google.co.us
```

(Figure 65). Doing this and clicking forward we are no redirected to (Figure 66). Although this isnt a real website we still were able to redirect to a differing webpage.

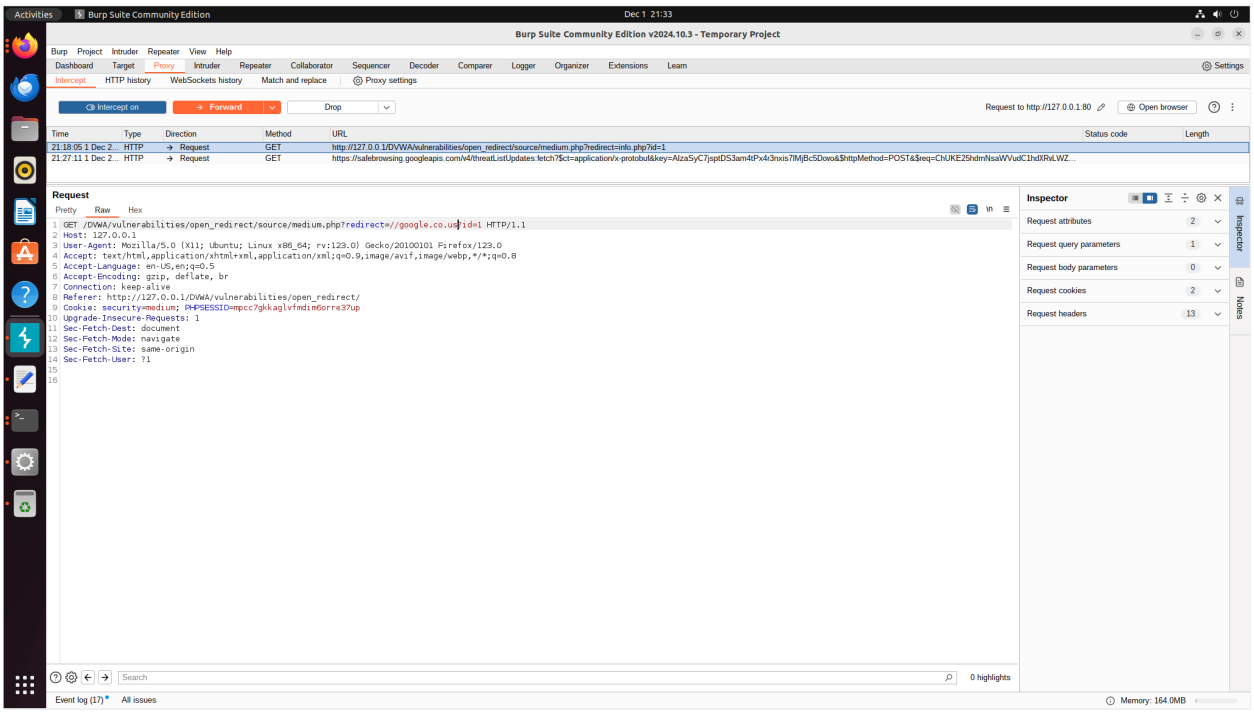

Figure 65: Step2

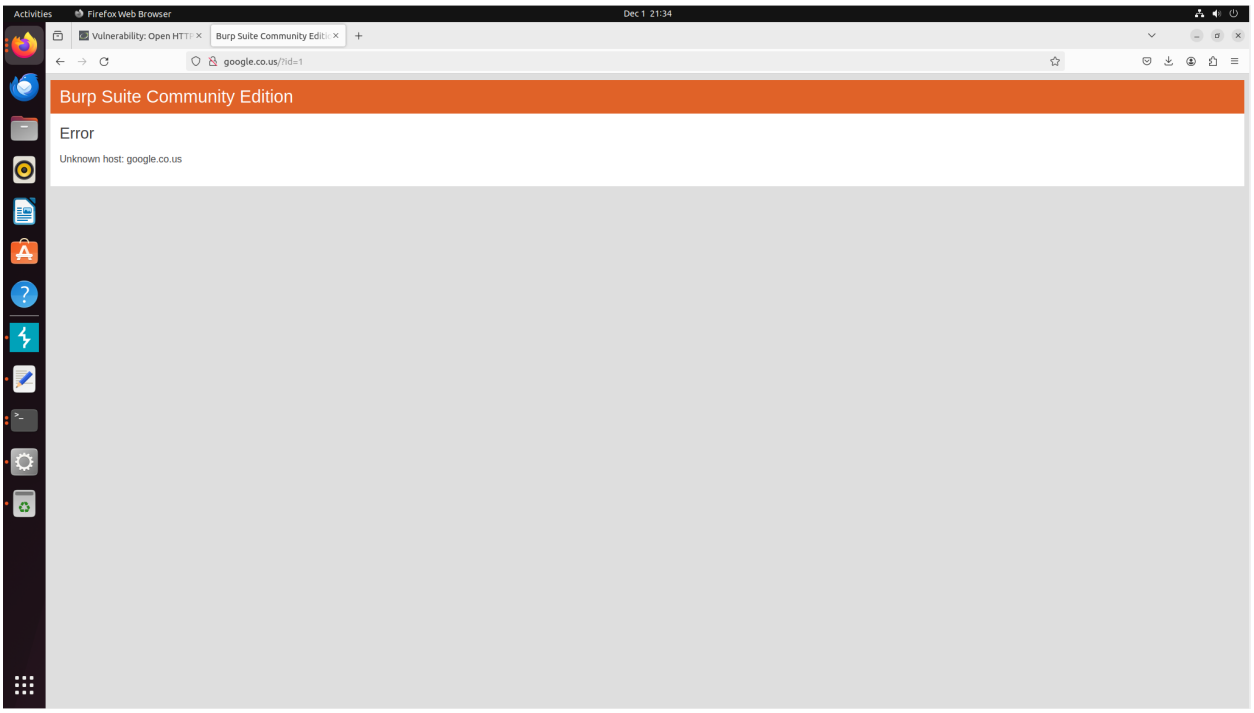

Figure 66: Step2

- **Related OWASP Top 10 Vulnerabilities:**
  - A10:2021 – Server-Side Request Forgery (SSRF)

- **References**

19 Conclusion

To automate the testing of vulnerability we need to have a system that would simulate real world attacks, while analyzing the web application responses. Some of these things already exist within Burp Suite but identifying all entry points is something of a fever dream. It still would be possible to find majority of them and this would most likely start by building out automation for the most common attack and exploit strategies. Some good ones to start with would be SQL injection, XSS, and insecure session ID's or cookie. Where we could mimic manual payload injections and validate responses. Additionally, we could automate entry point detection using web crawlers or tools like Burp Suite where we could take links in form pages and have scripts input into them and see the response. I think to automate the testing of vulnerability's it would be simplest to integrate existing tools like Burp or ZAP to streamline testing.