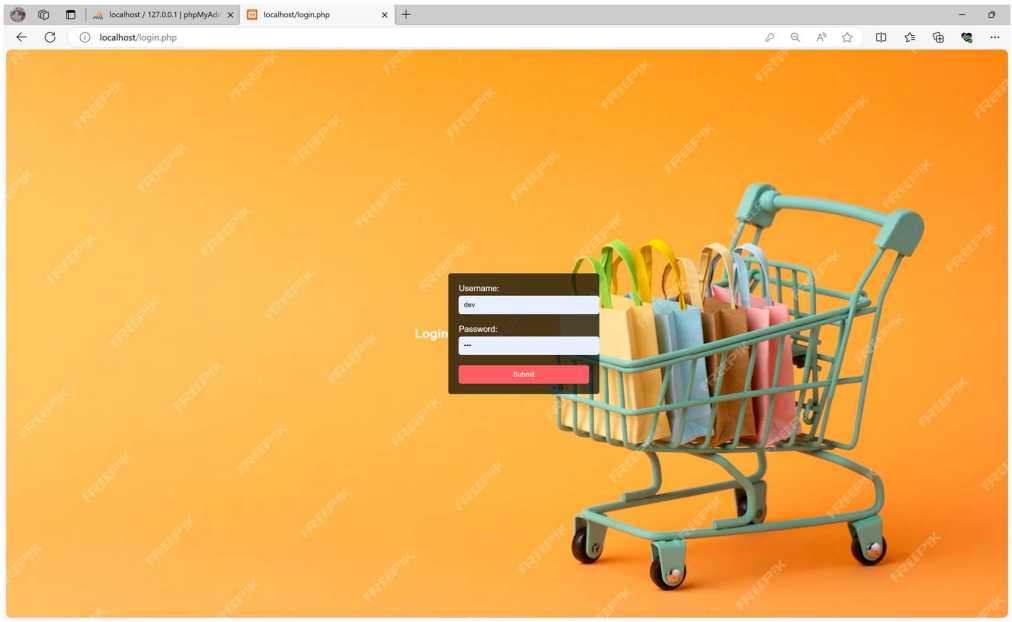
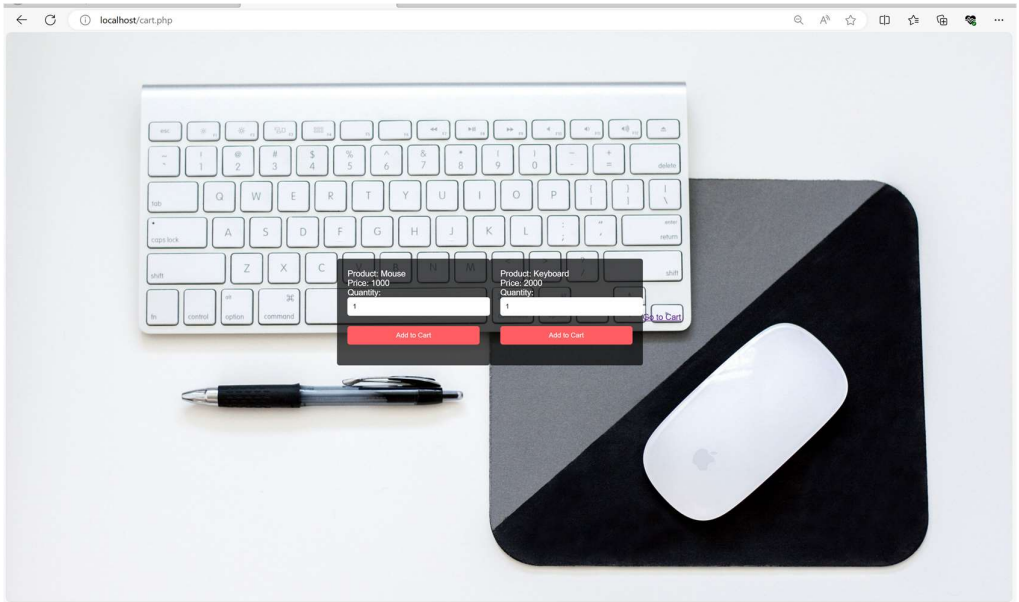
Information Security Management

Name: V. Saidev

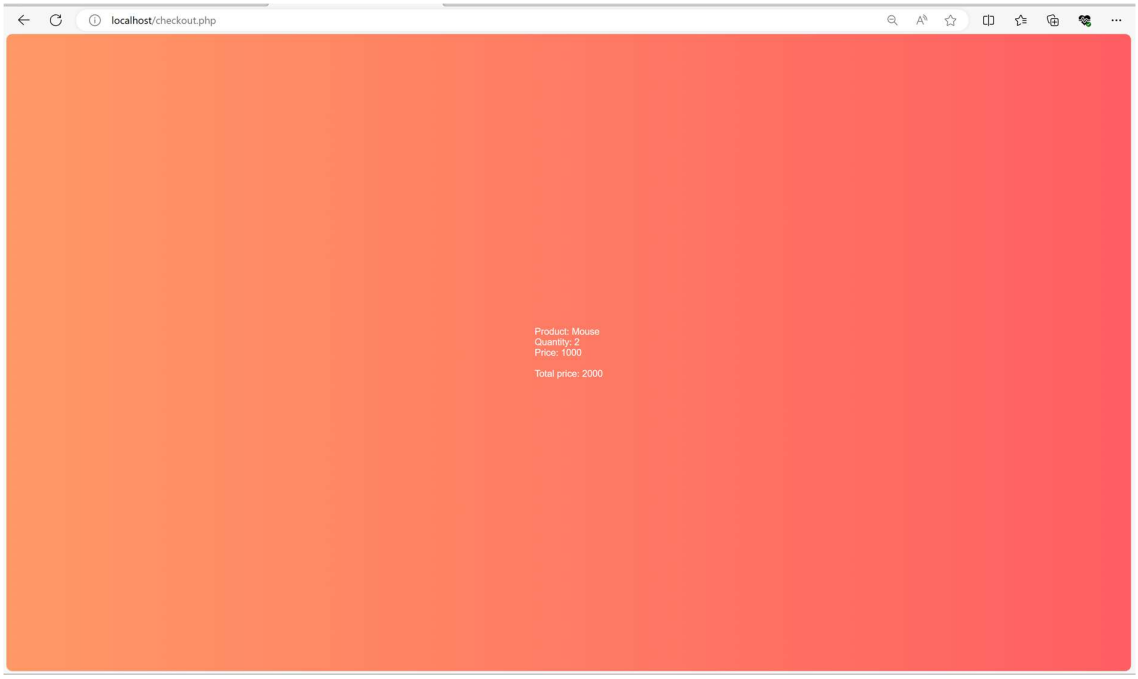
At Start, Login page and Cart page have to be created



Cart page



Checkout Page



Experiment 1

Objective:

Client-Side Attack, specifically a type of attack called Manipulating Hidden Fields. In this case, the client-side (i.e., the user's browser) is manipulated to **change the price of a product**. This is a common security flaw where the server trusts the client-side to send the correct price.

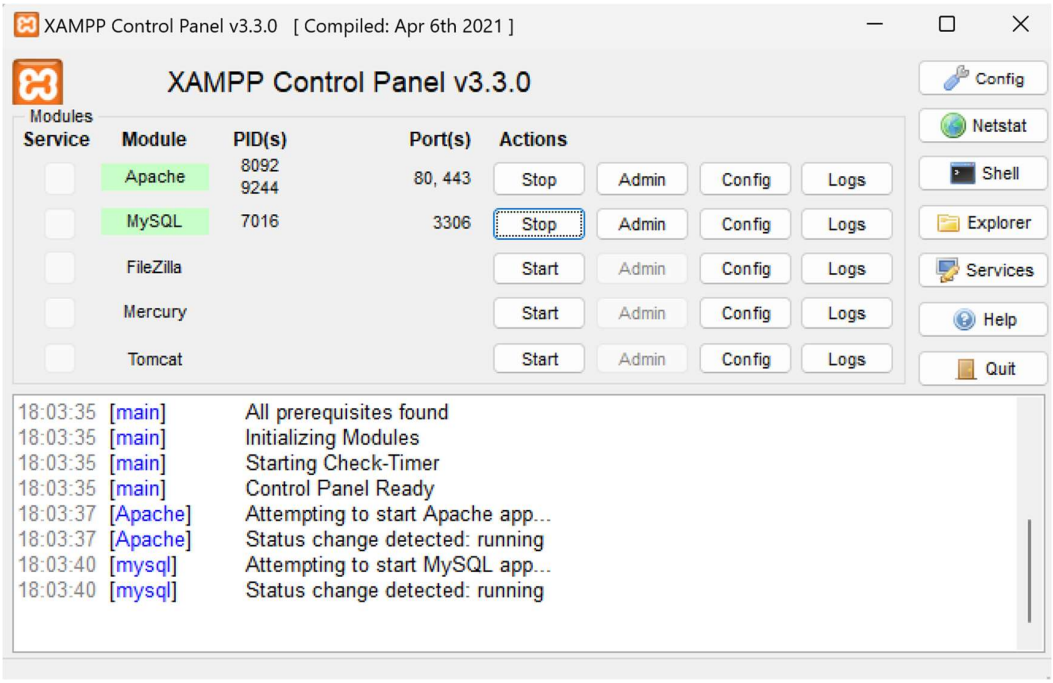
Introduction:

The exercise we're doing is essentially a demonstration of this attack and how to prevent it. The prevention involves modifying the server-side code to always determine the price of a product from the server-side, where it can't be manipulated by the client, rather than trusting the price sent from the client-side. This is a fundamental principle of web security: never trust the client, always verify everything on the server side.

Setup:

We would need to have a server environment that supports PHP and MySQL. Here are the steps:

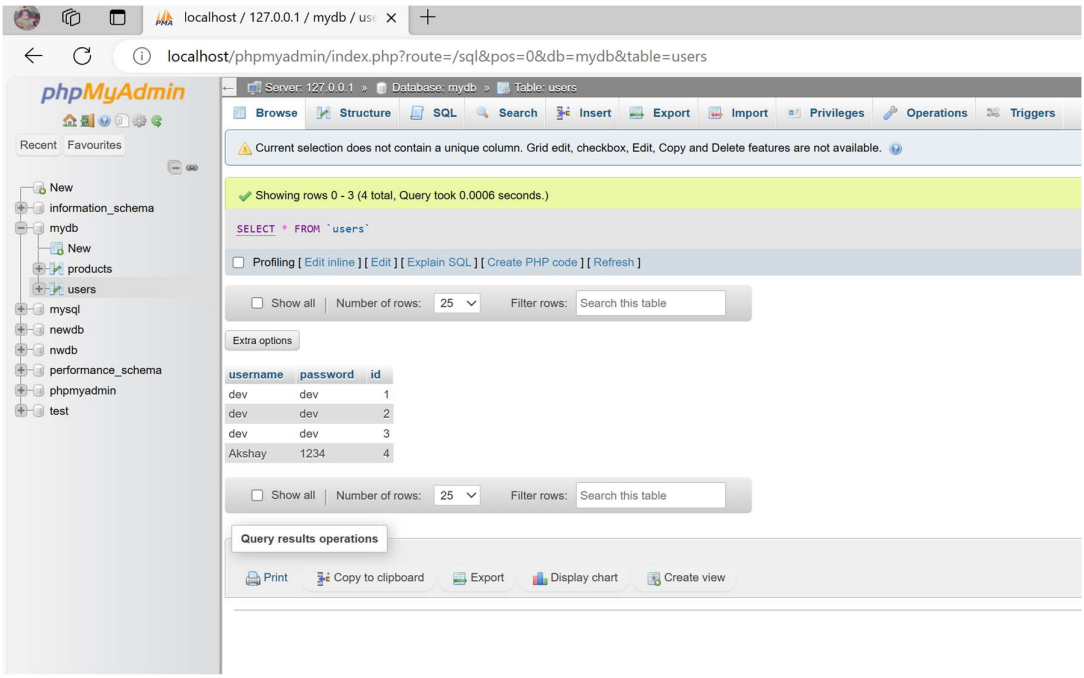
1. **Install a local server environment**: We are using XAMPP software. These software packages include Apache (a web server), MySQL (a database), and PHP.



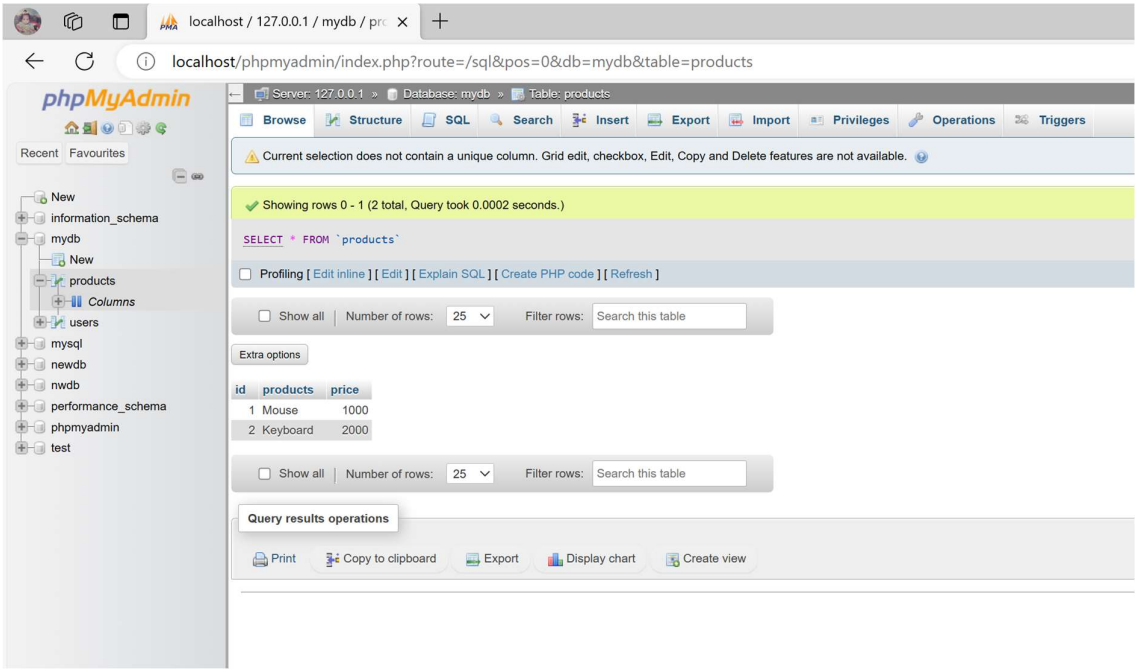
2. **Create a MySQL Database and User**: Using phpMyAdmin (which comes with XAMPP), create a new database and a user with full privileges for that database.

3. **Create the necessary tables in our database**: We have Created a user table to store usernames and passwords and Products tables to store product details and prices.

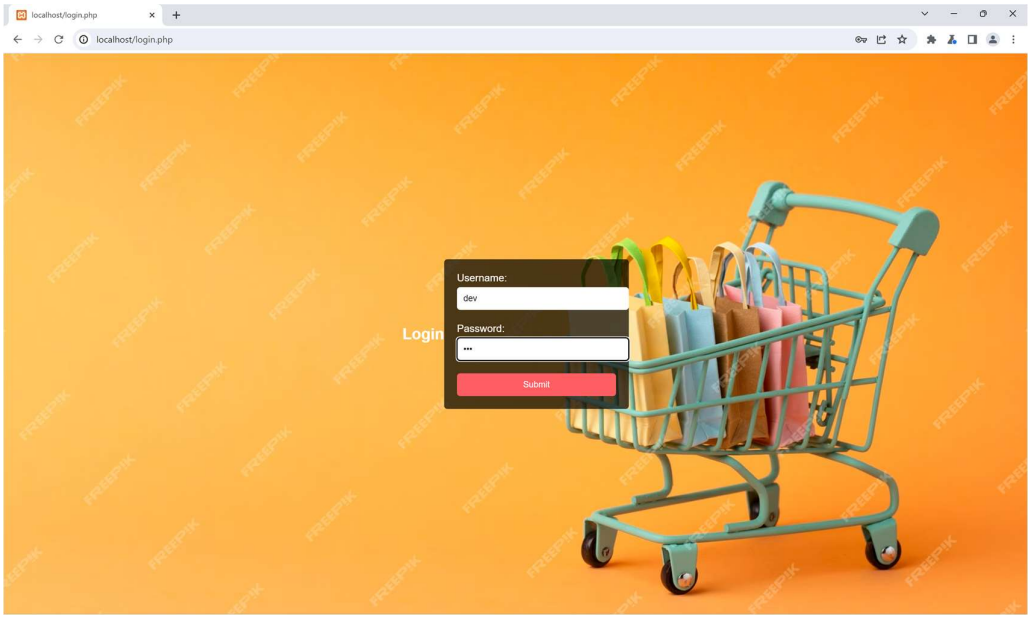
Users Table :



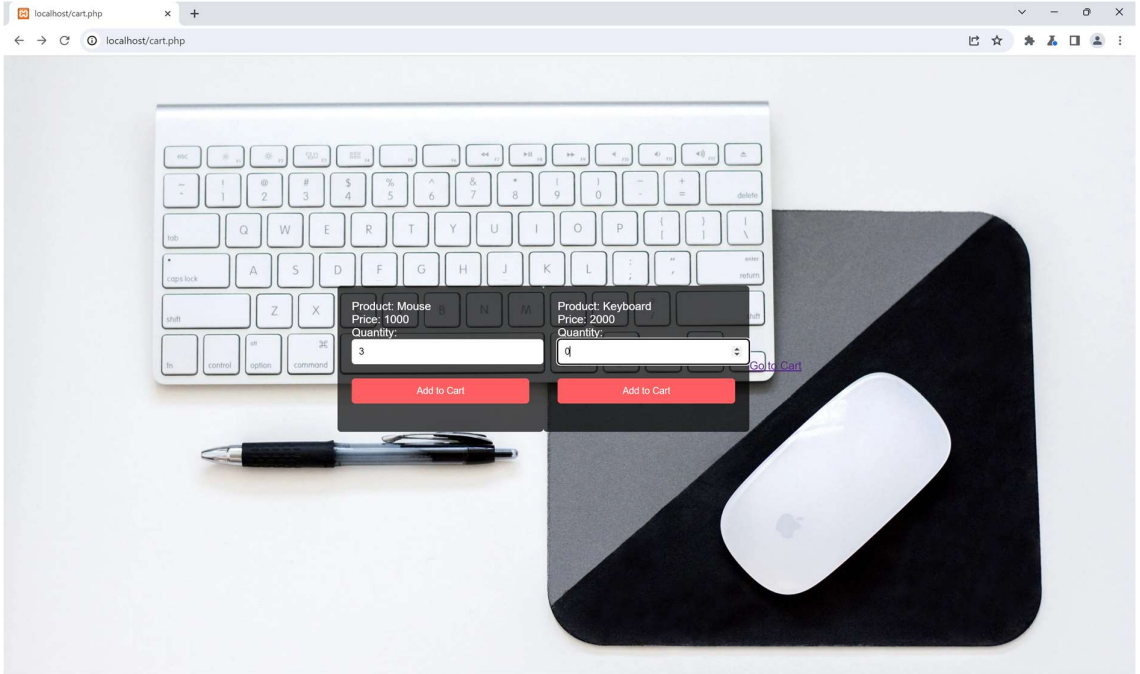
Products Table :



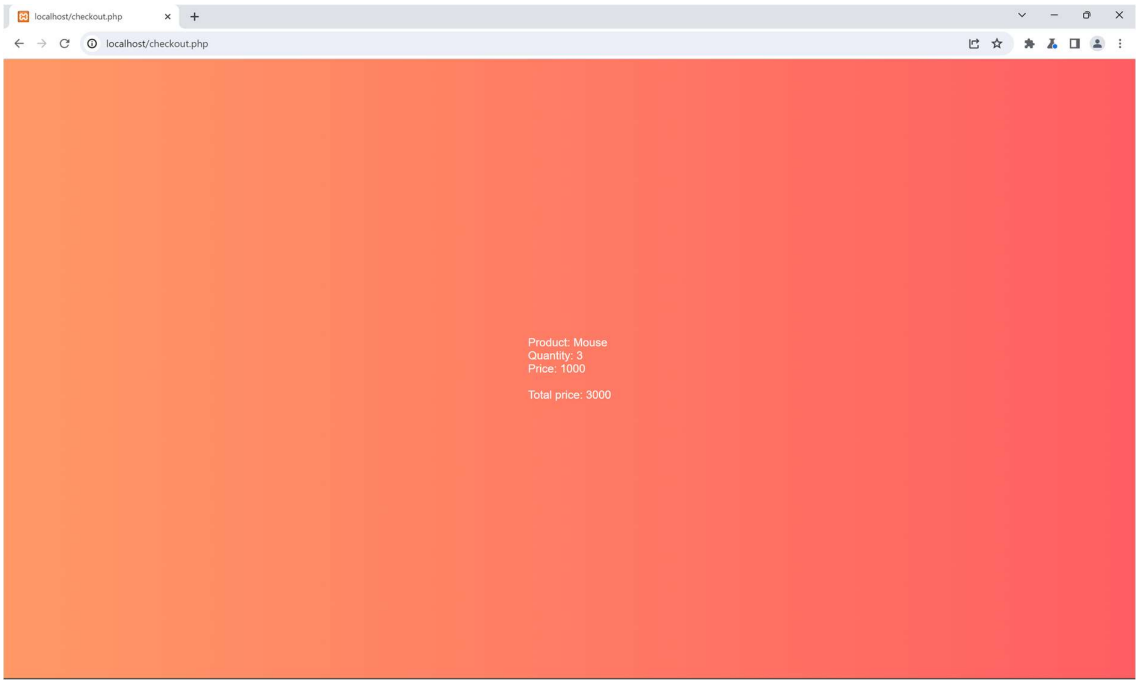
4. **Place our PHP files in the correct directory**: In XAMPP, we store php files in htdocs folder.
5. **Start our server and open the PHP files in a web browser**: We can do this by navigating to localhost/file.php in our web browser.
6. **Configure our browser to work with Burp Suite:** We need to set up your browser to use Burp Suite as its proxy. The default address is localhost and port is 8080.



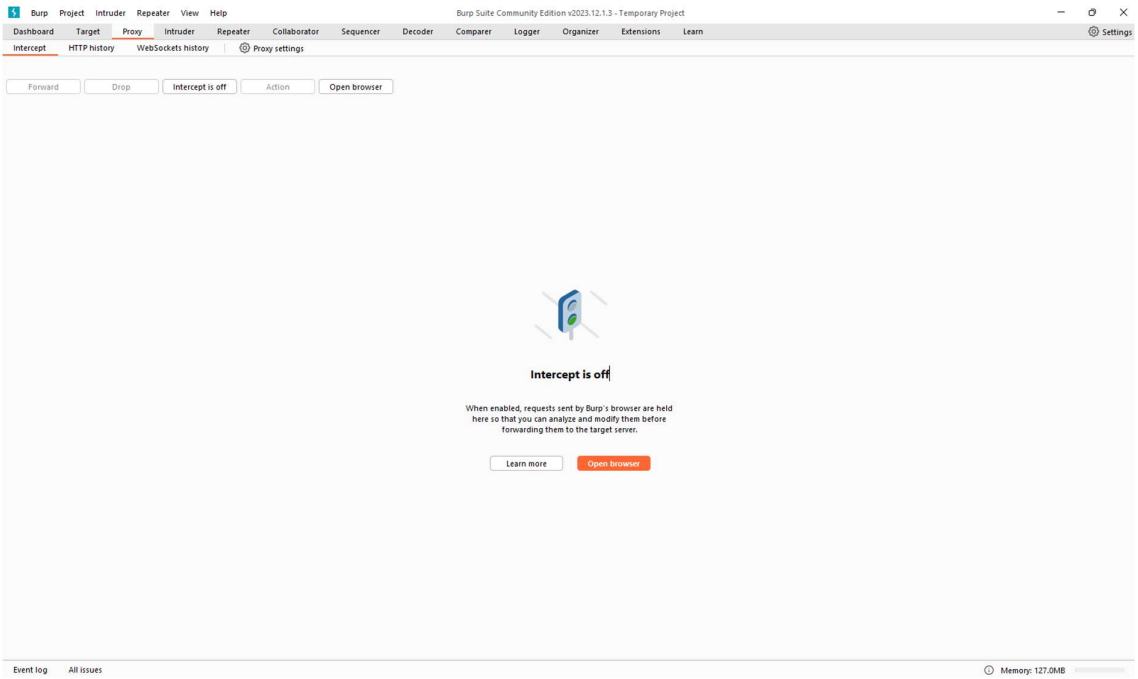
After logging in will be redirected to cart page so that we can select items and add to the cart.



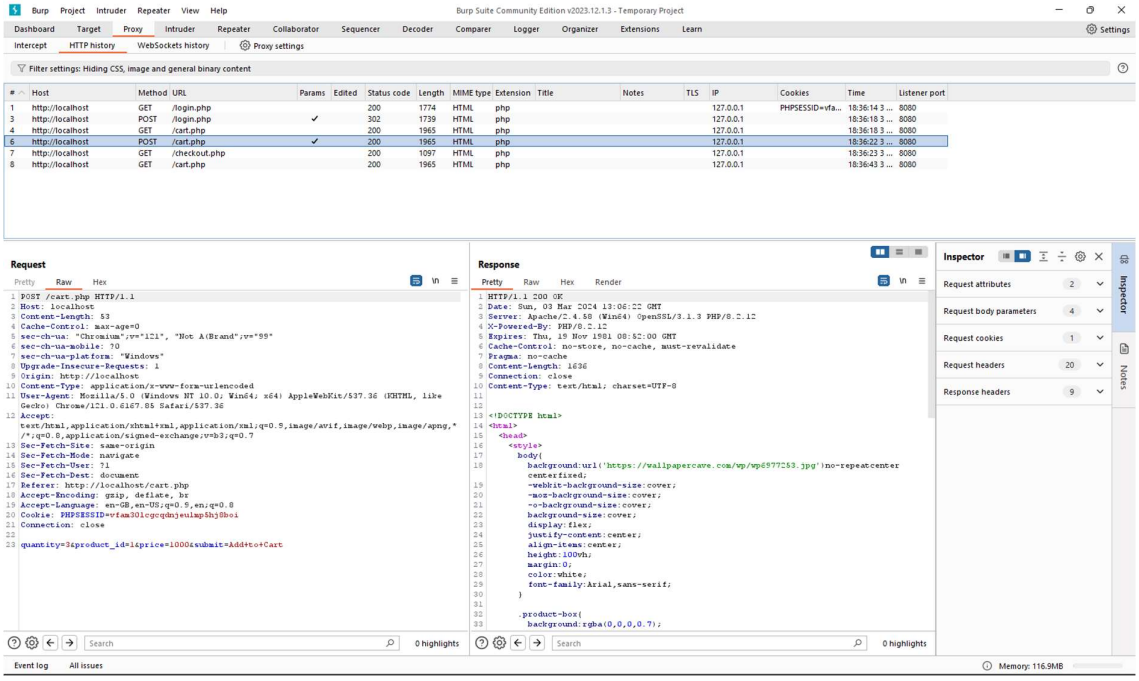
Items in cart with total price will be displayed in Checkout.php



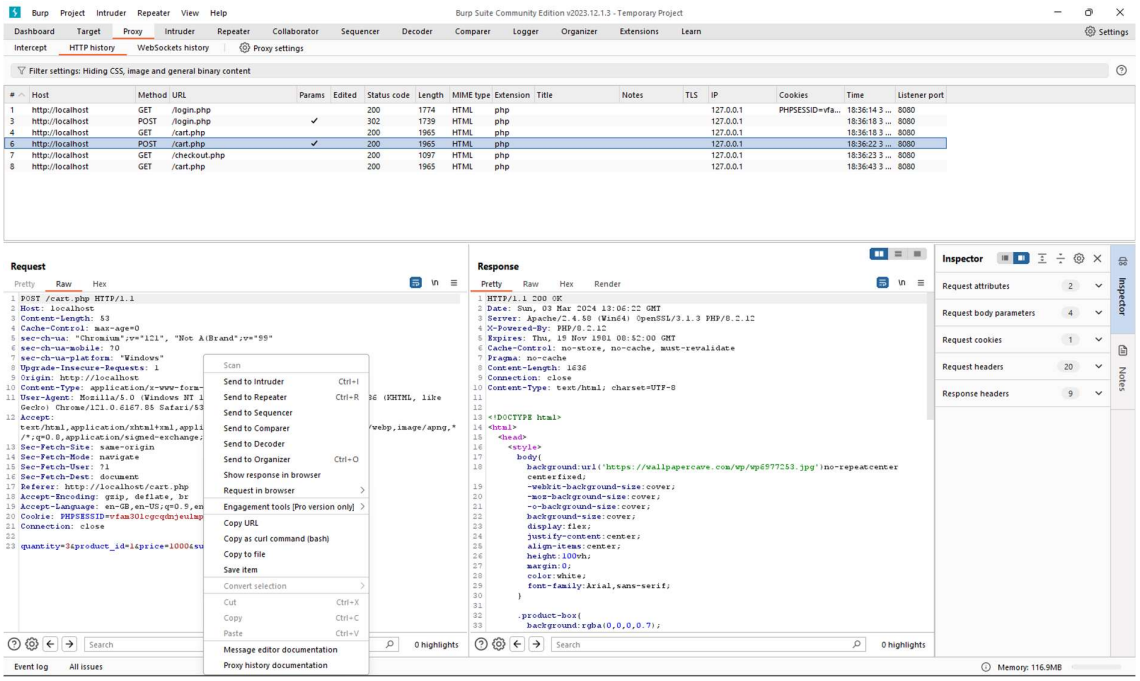
7. **Turn off intercept mode in Burp Suite:** When we first start Burp Suite, the intercept mode is ON by default. We need to turn it off to allow the traffic to flow freely.



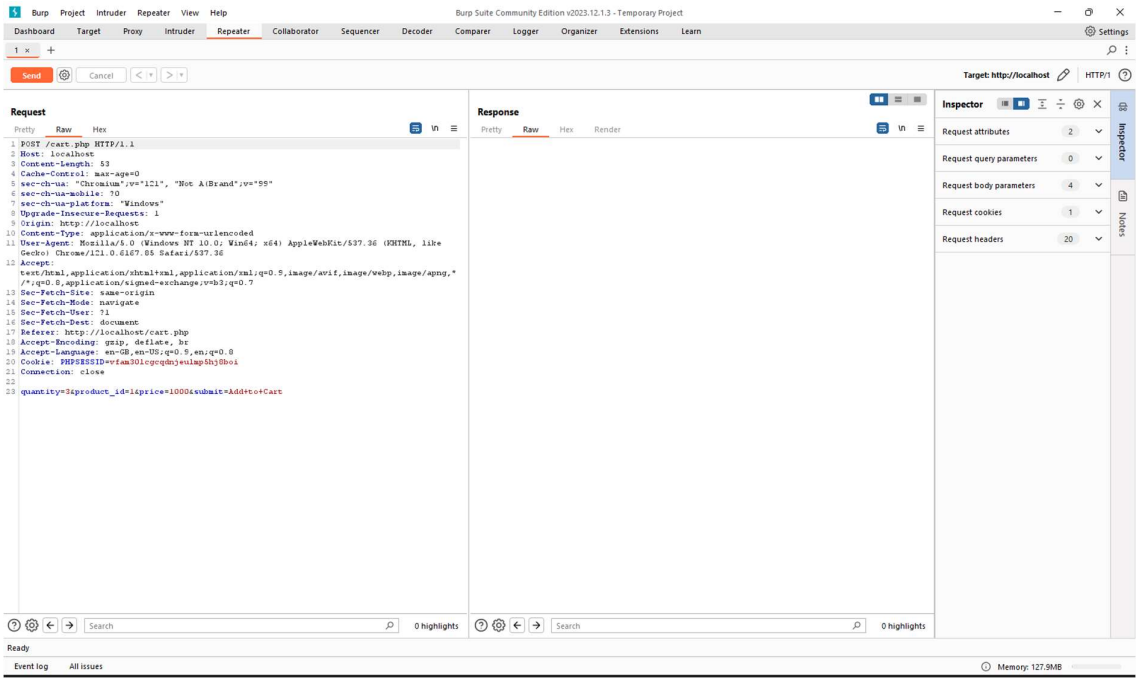
8. **Access our PHP application through the browser:** Now, if we access your PHP application through our browser, the traffic should be captured in Burp Suite under the "HTTP history" sub-tab in the "Proxy" tab.



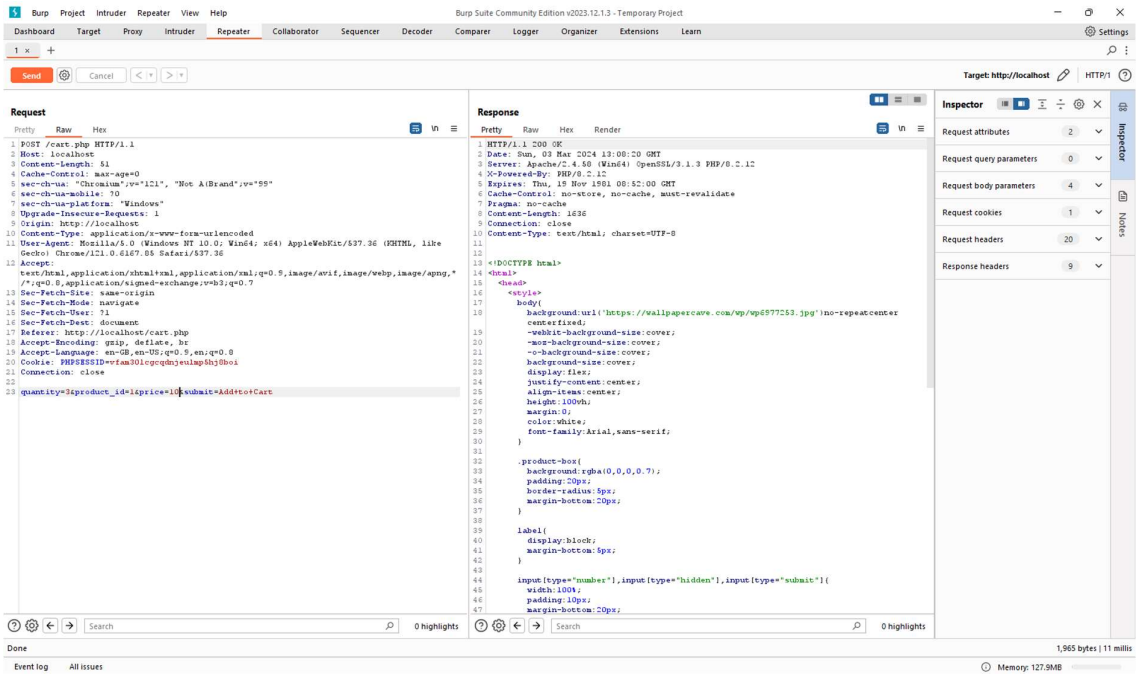
9. **Analyze the traffic:** We can now analyze the HTTP requests and responses, modify them, replay them, and do various other things as part of our security testing.



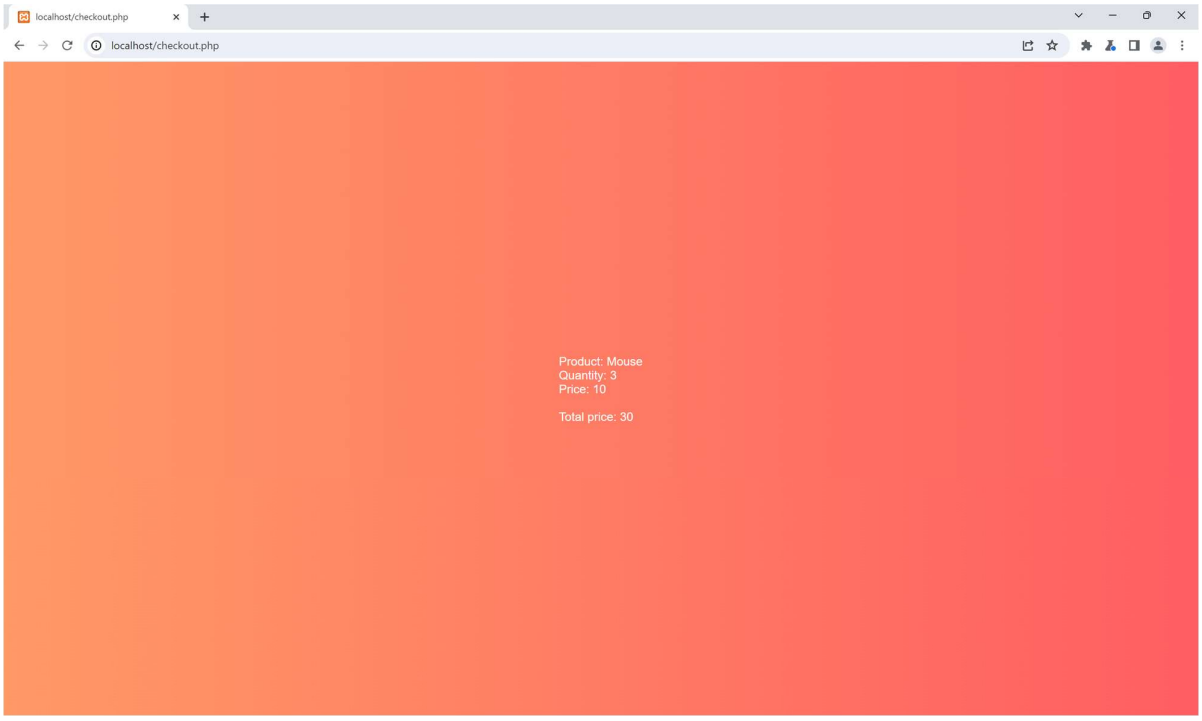
Here as shown in above Screenshot When we click on Post method (URL /Cart.php) quantity, Product id and price of item will be displayed in Request. And then Right click and select send to Repeater.



Now we changed the price from 1000 to 10 and click on Send to get response.



Now refresh the page to check whether the attack performed or not i.e., price is changed from 3000 to 30 as shown in below Screenshot.



We performed the attack Succesfully.

Solution :

In our original code, we were accepting the price of a product from the client-side (i.e., from the user's browser). This is a security vulnerability because a user can manipulate the data sent from their browser. In our case, a user could potentially change the price of a product in their cart to any value they want.

To fix this, we need to always determine the price of a product on the server-side, where it can't be manipulated by the user. Here's how the modified code achieves this:

1. When a user adds a product to their cart, our code accepts the product ID and quantity from the user's POST request.
2. Instead of also accepting the price from the POST request, our code ignores any price sent from the client-side. Instead, it queries your database to find the correct price of the product.
3. Our code uses a prepared statement to query the database, which helps prevent SQL injection attacks. The product ID is used as a parameter in this query.
4. Once the correct price is retrieved from the database, it's used to calculate the total price of the items in the user's cart.

This way, even if a user tries to manipulate the price on the client-side, our server will always use the correct price from the database. This is a fundamental principle of web security: never trust the client, and always verify everything on the server side.

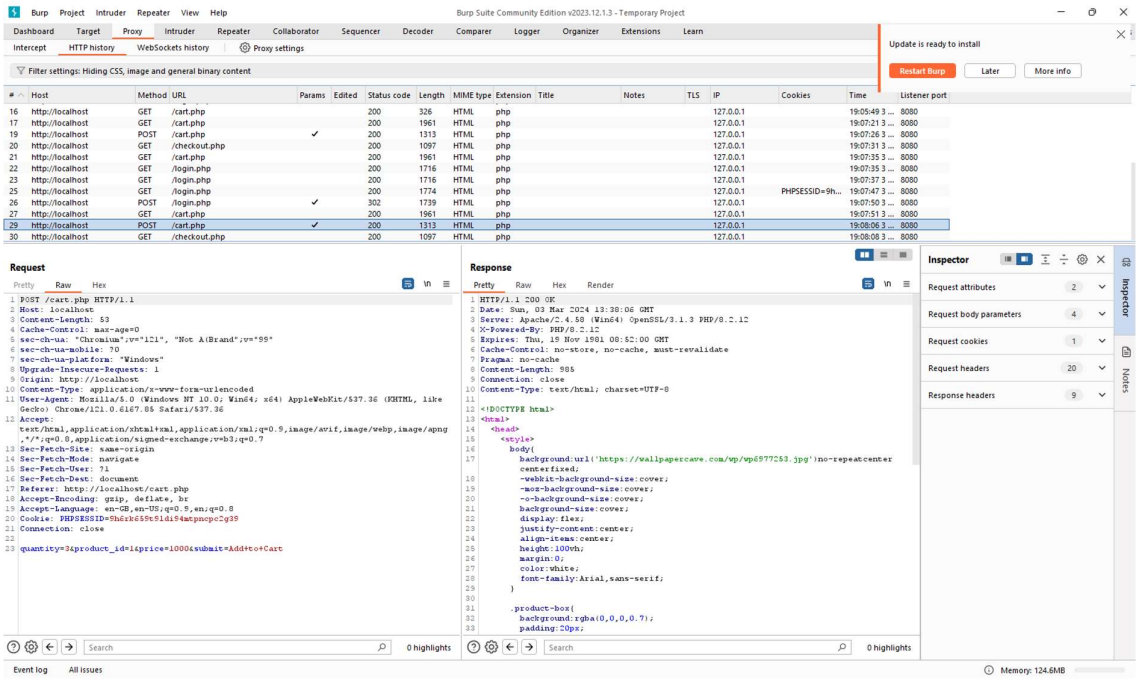
<u>Modified part of the Code :</u>

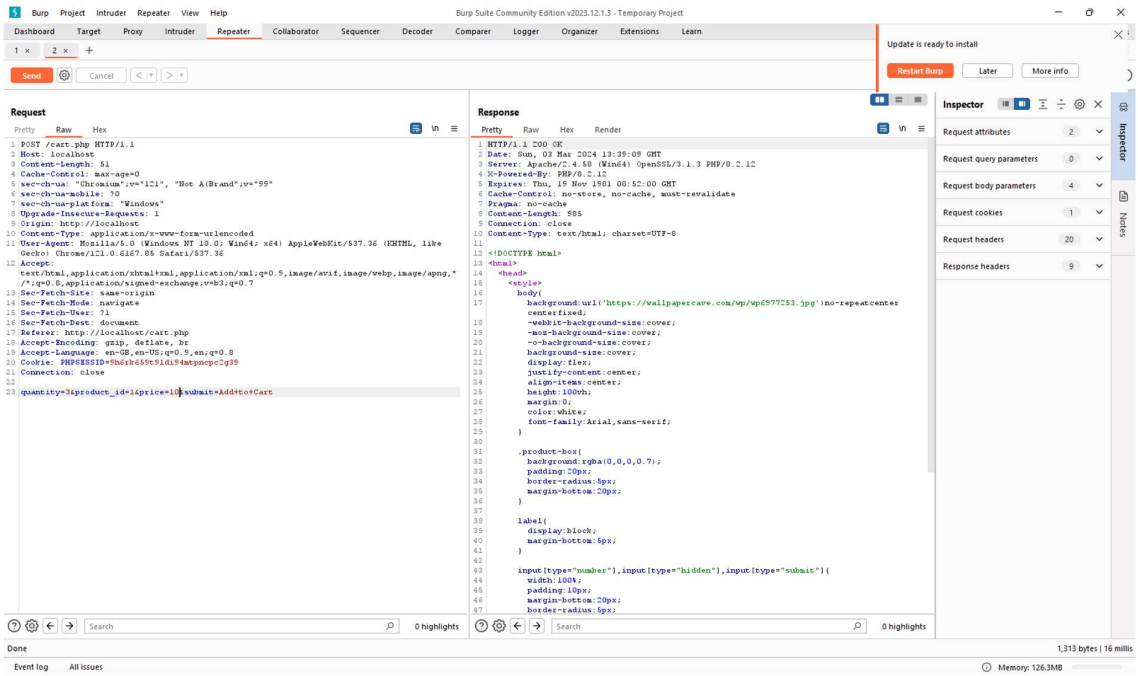
```
// Add product to cart
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $product_id = $_POST["product_id"];
  $quantity = $_POST["quantity"];

  // Get the price from the database, not from the client-side
  $stmt = $conn->prepare("SELECT price FROM products WHERE id = ?");
  $stmt->bind_param("i", $product_id);
  $stmt->execute();
  $result = $stmt->get_result();
  $row = $result->fetch_assoc();
  $price = $row["price"];

  $_SESSION["cart"][$product_id] = array("quantity" => $quantity, "price" => $price);
}
```

```php
<?php
// Start the session
session_start();

// Check if user is logged in
if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true){
    header("location: login.php");
    exit;
}

// Connect to the database
$servername = "localhost";
$username = "dev";
$password = "dev";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Fetch product details
$sql = "SELECT id, products, price FROM products";
$result = $conn->query($sql);

// Initialize cart if it's not set
if(!isset($_SESSION["cart"])) {
  $_SESSION["cart"] = array();
}

// Add product to cart
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $product_id = $_POST["product_id"];
  $quantity = $_POST["quantity"];
  $price = $_POST["price"]; // Get the new price from the client-side
  $_SESSION["cart"][$product_id] = array("quantity" => $quantity, "price" => $price);
}

// Calculate total price
$total_price = 0;
foreach ($_SESSION["cart"] as $id => $item) {
  $total_price += $item["price"] * $item["quantity"];
}
?>

<!DOCTYPE html>
<html>
```

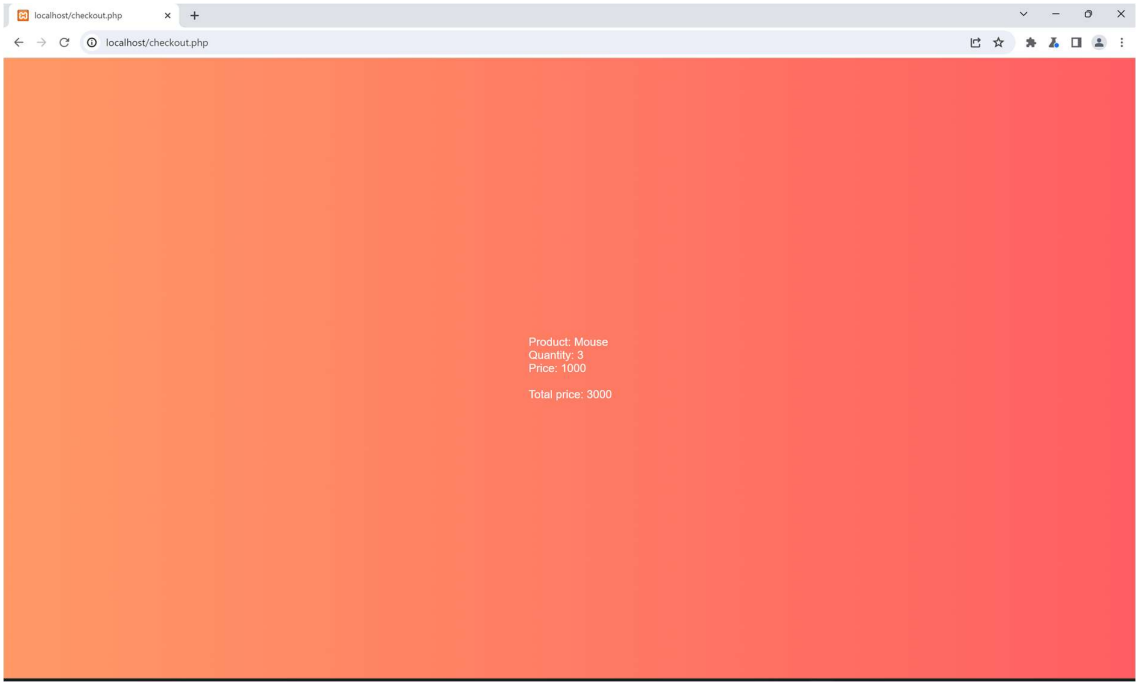
```php
<?php
// Start the session
session_start();

// Check if user is logged in
if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true){
    header("location: login.php");
    exit;
}

// Connect to the database
$servername = "localhost";
$username = "dev";
$password = "dev";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Fetch product details
$sql = "SELECT id, products, price FROM products";
$result = $conn->query($sql);

// Initialize cart if it's not set
if(!isset($_SESSION["cart"])) {
  $_SESSION["cart"] = array();
}

// Add product to cart
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $product_id = $_POST["product_id"];
  $quantity = $_POST["quantity"];

  // Get the price from the database, not from the client-side
  $stmt = $conn->prepare("SELECT price FROM products WHERE id = ?");
  $stmt->bind_param("i", $product_id);
  $stmt->execute();
  $result = $stmt->get_result();
  $row = $result->fetch_assoc();
  $price = $row["price"];

  $_SESSION["cart"][$product_id] = array("quantity" => $quantity, "price" => $price);
}

// Calculate total price
```

Now, let’s try to change price again after updating the code as shown in above Screenshot.





The Price is not changed after performing the attack.



We successfully prevented form the attack, even if a user tries to manipulate the price on the client-side, our server will always use the correct price from the database.

Experiment 2

Experimental Objective:

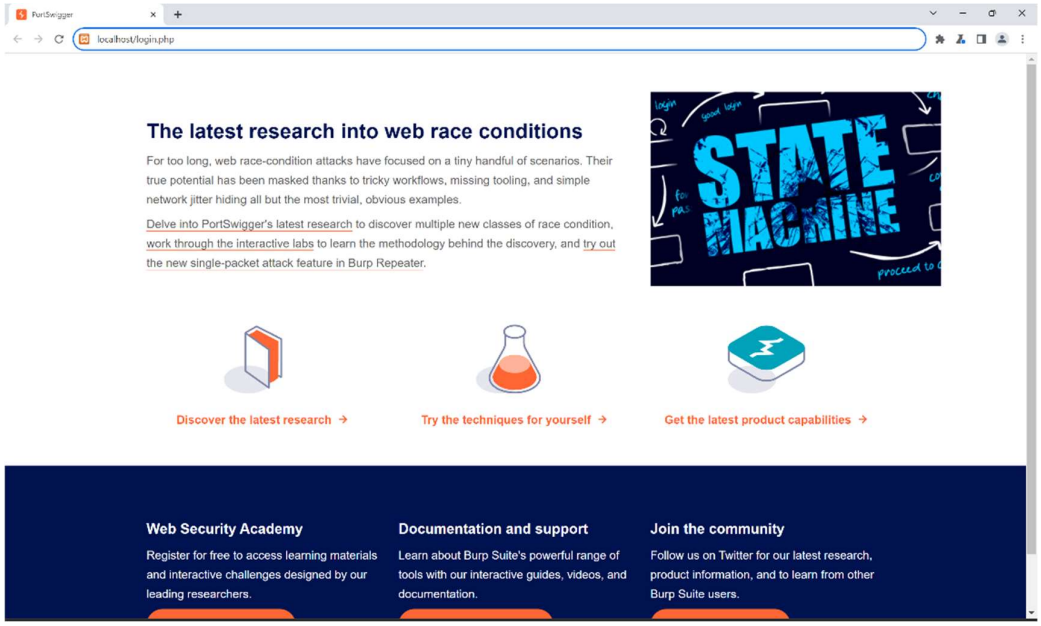
The main Objective is to find the password and the username of a website using Brute Force method. To understand and apply the four different attack types in Burp Suite's Intruder tool to find the username and password of a web application.

Types of Attacks :

1. **Sniper:** This attack places each payload into each payload position in turn. It uses a single payload set. The total number of requests generated in the attack is the product of the number of positions and the number of payloads in the payload set.
2. **Battering Ram:** This attack places the same payload into all of the defined payload positions simultaneously. It uses a single payload set. The total number of requests generated in the attack is the number of payloads in the payload set.
3. **Pitchfork:** This attack iterates through a different payload set for each defined position. Payloads are placed into each position simultaneously. The total number of requests generated in the attack is the number of payloads in the smallest payload set.
4. **Cluster Bomb:** This attack iterates through a different payload set for each defined position. Payloads are placed from each set in turn, so that all payload combinations are tested. The total number of requests generated in the attack is the product of the number of payloads in all defined payload sets.

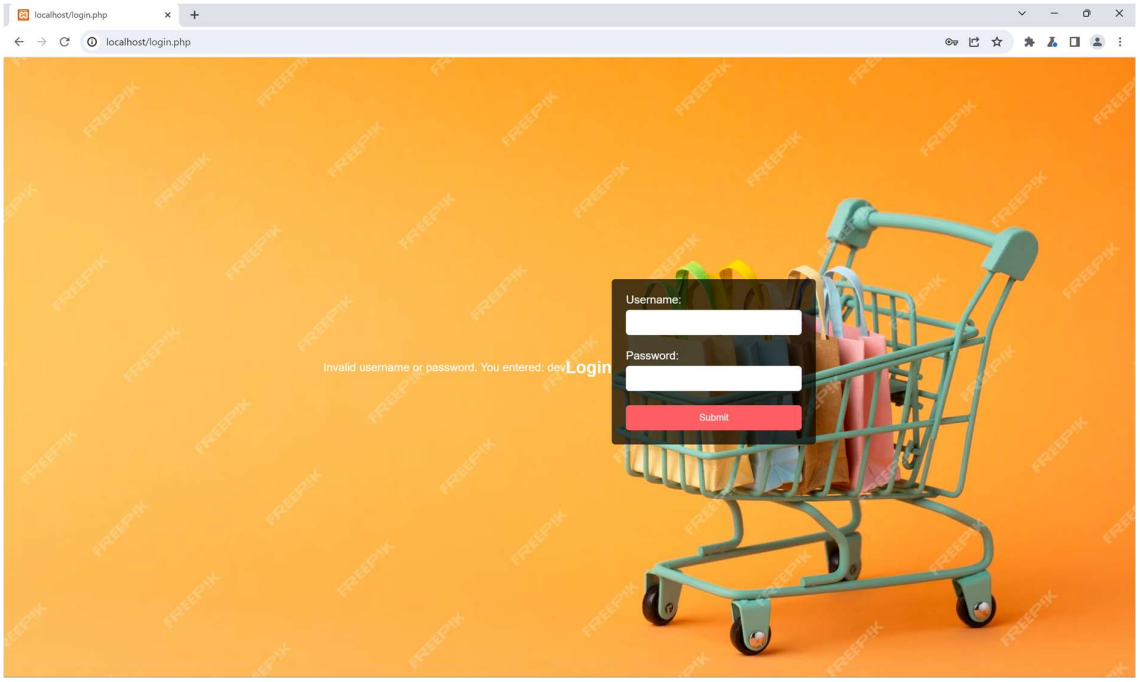
Setup:

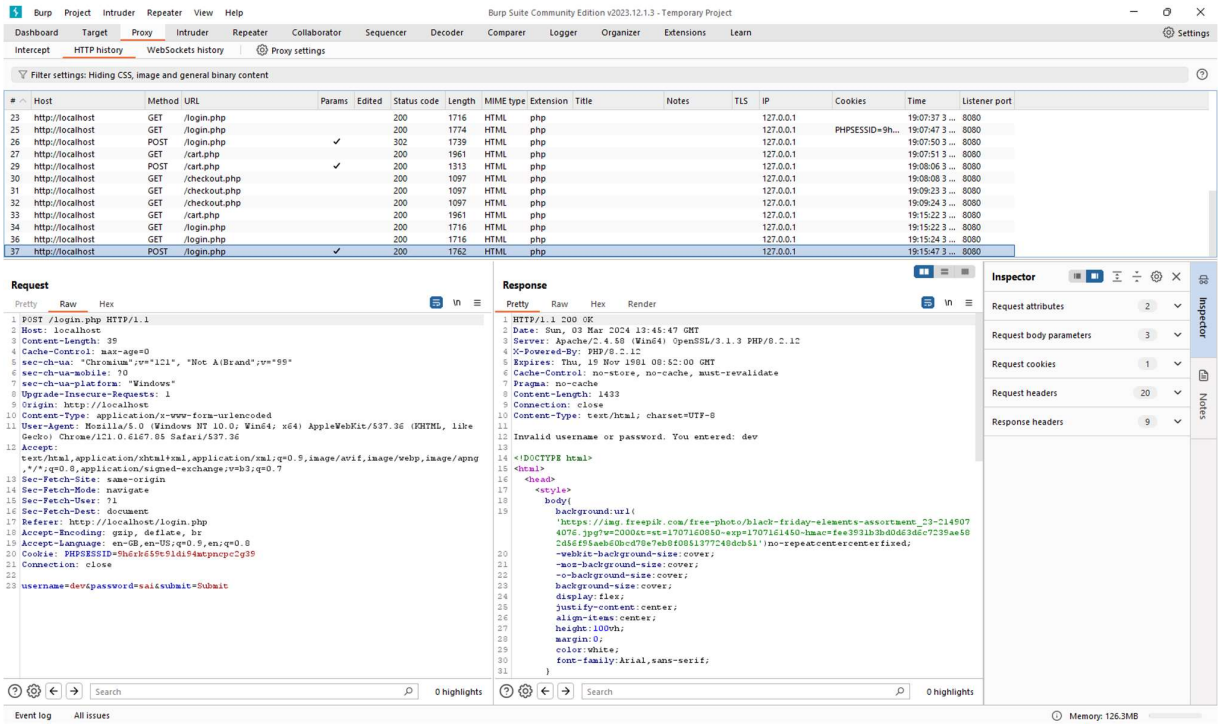
1. Install and run Burp Suite.
2. Navigate to the site we want to test. (localhost/login.php)



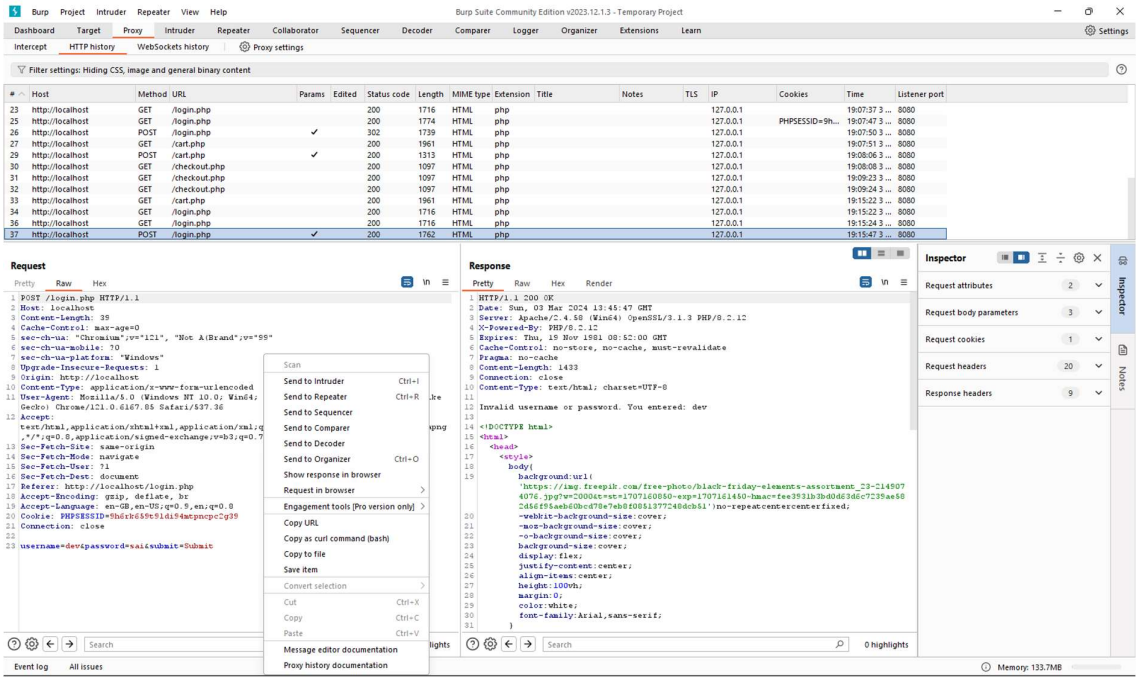
Experimental Steps :

- Intercept the login request: Use the Burp Suite Proxy tool to intercept the HTTP request sent when attempting to log in

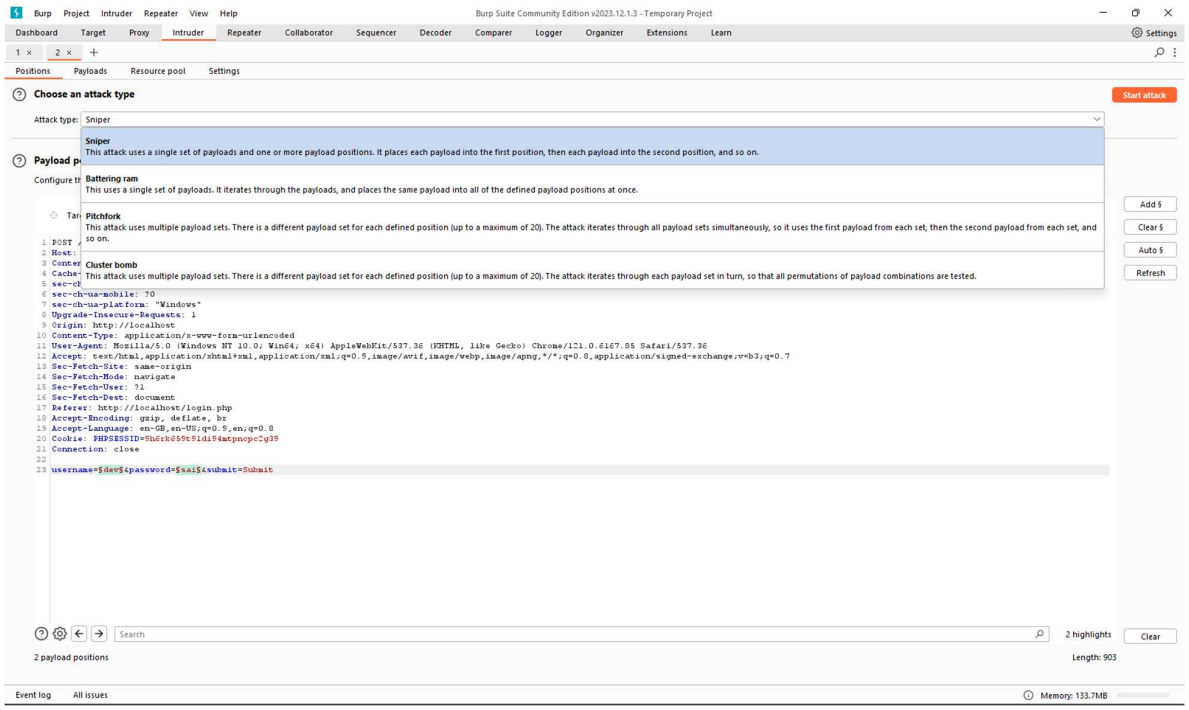




- Send to Intruder: Right-click on the intercepted request and click on "Send to Intruder".

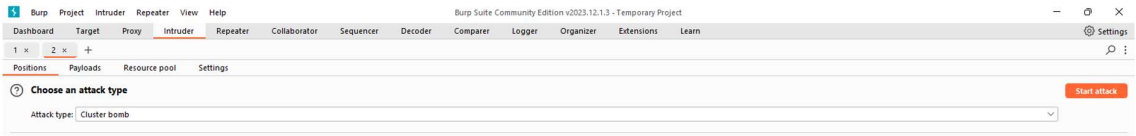


- In intruder tab we can see four attack types as shown in below scrrenshot, Let's perform all four attacks.



Cluster bomb :

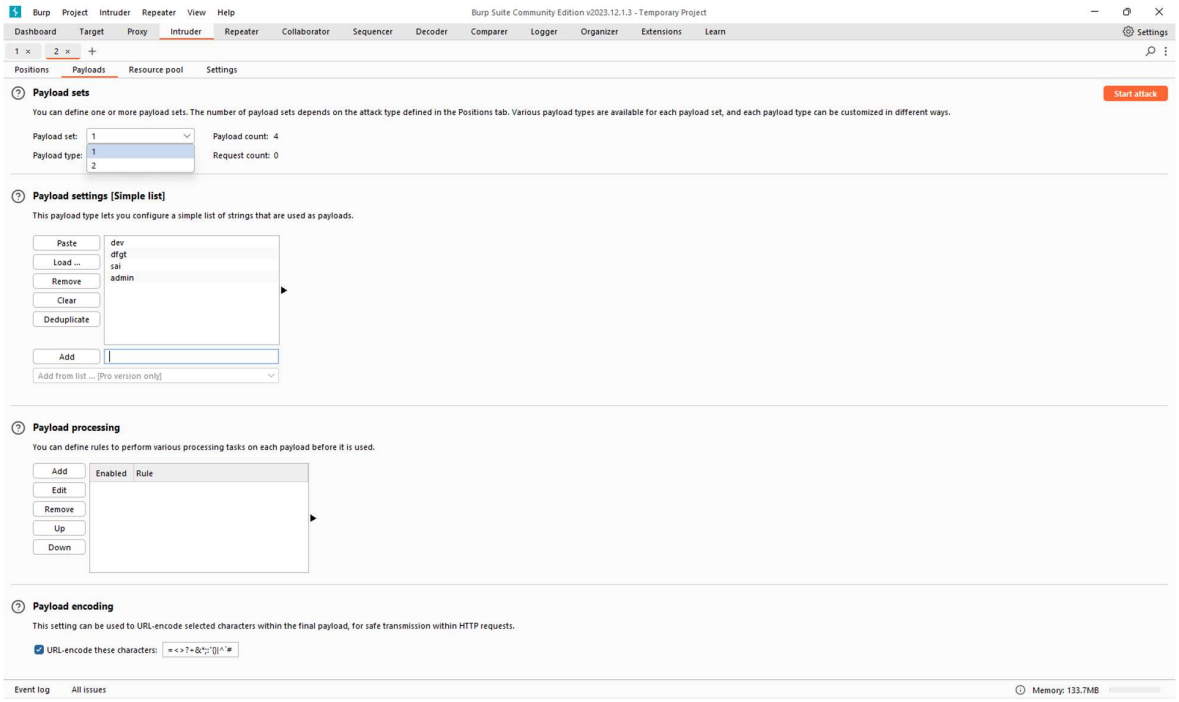
- Set the attack type: In the Intruder tab, set the attack type to "Cluster Bomb".



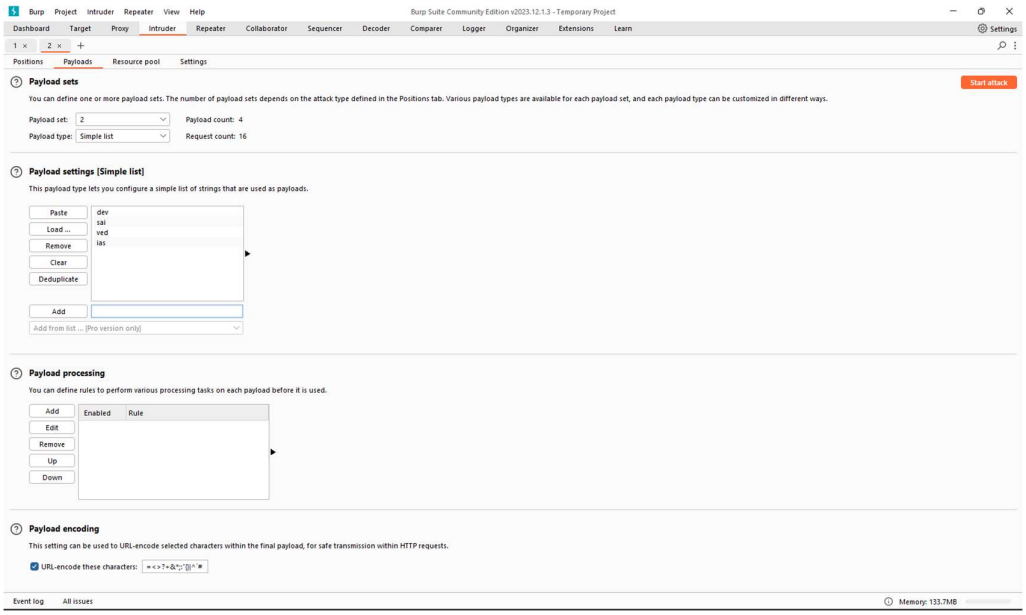
- Set the payload positions: Highlight the values you want to brute force (e.g., username and password fields) and click “Add $” to mark them as payload positions.

- Configure the payloads: Go to the Payloads tab and configure your payloads. You’ll need to set up two payload sets - one for each payload position.

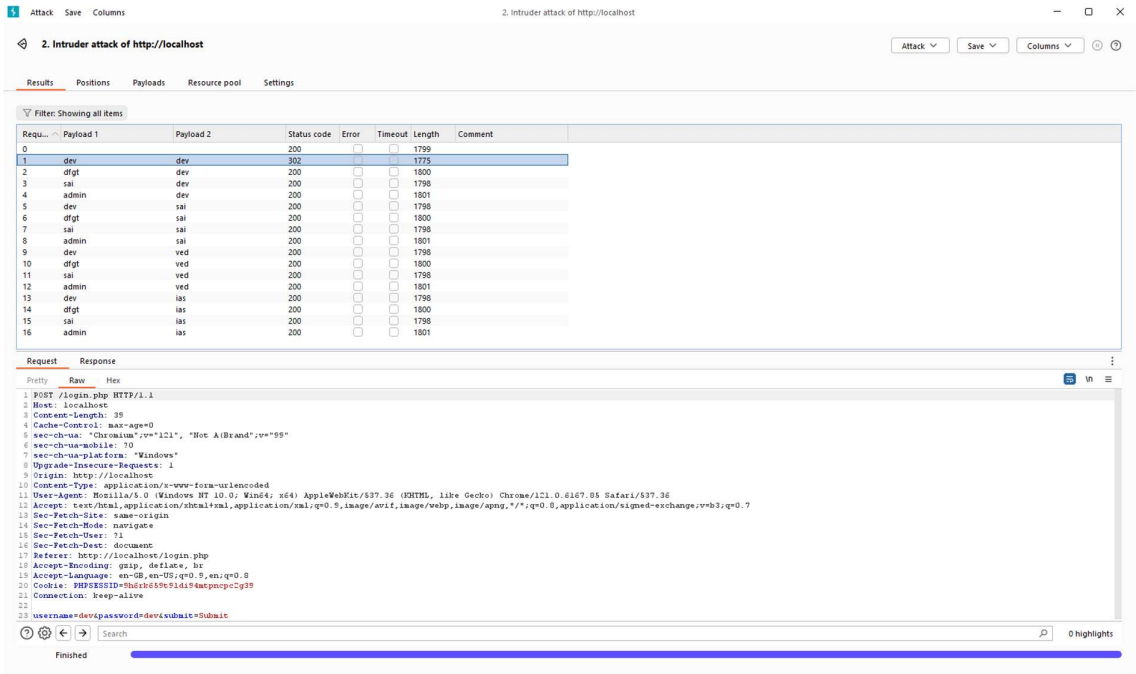
Payload Set – 1



Payload Set - 2



- Start the attack: Click the “Start attack” button.

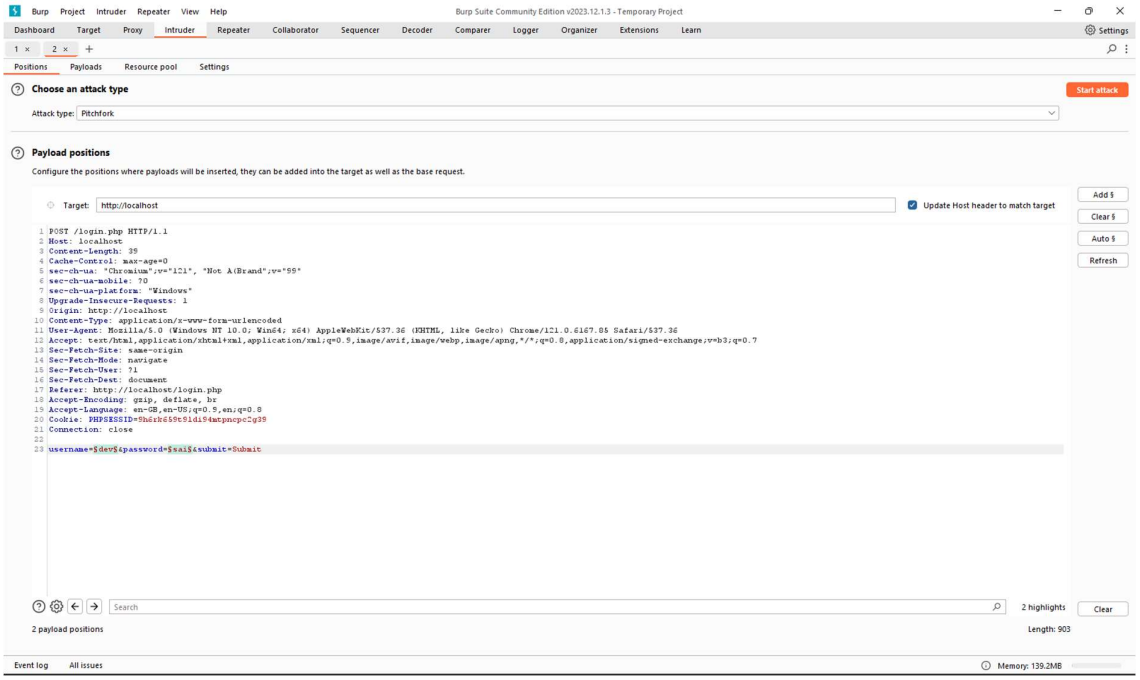


Observations :

We observed that when username and password are correct then the **length section element is different from other lengths or the Status code is different from the other status codes**, hence it is right password and username.

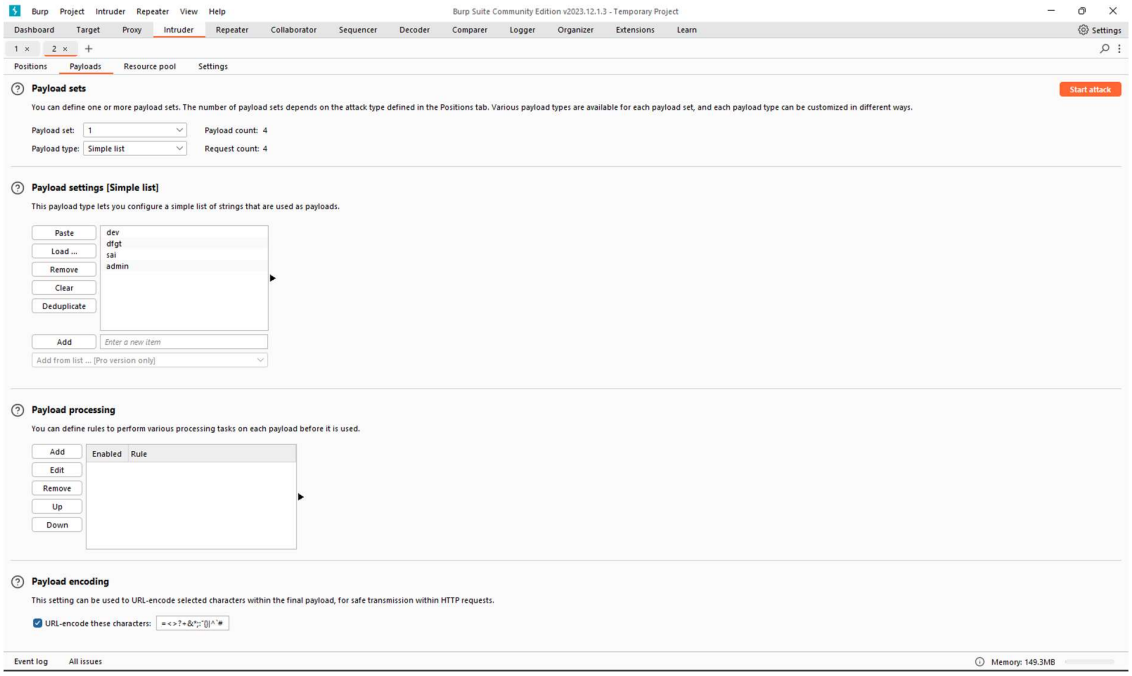
Pitchfork :

- Set the attack type: In the Intruder tab, set the attack type to "Pitchfork".
- Set the payload positions: Highlight the values you want to brute force (e.g., username and password fields) and click “Add $” to mark them as payload positions.

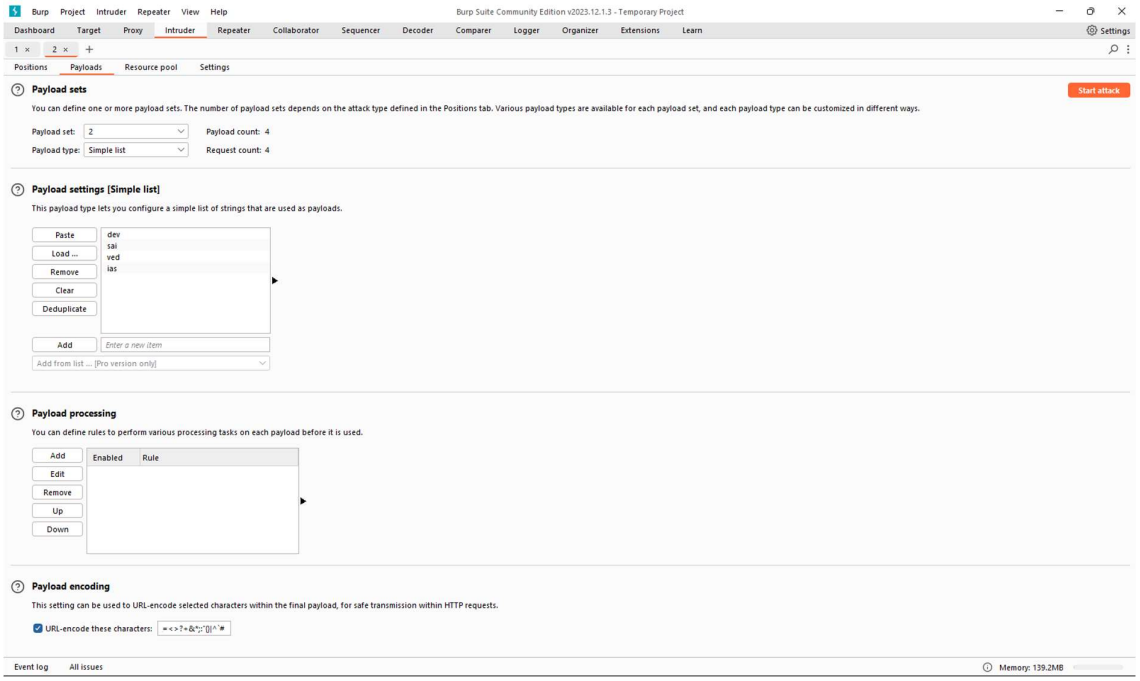


- Configure the payloads: Go to the Payloads tab and configure your payloads. You’ll need to set up two payload sets - one for each payload position.

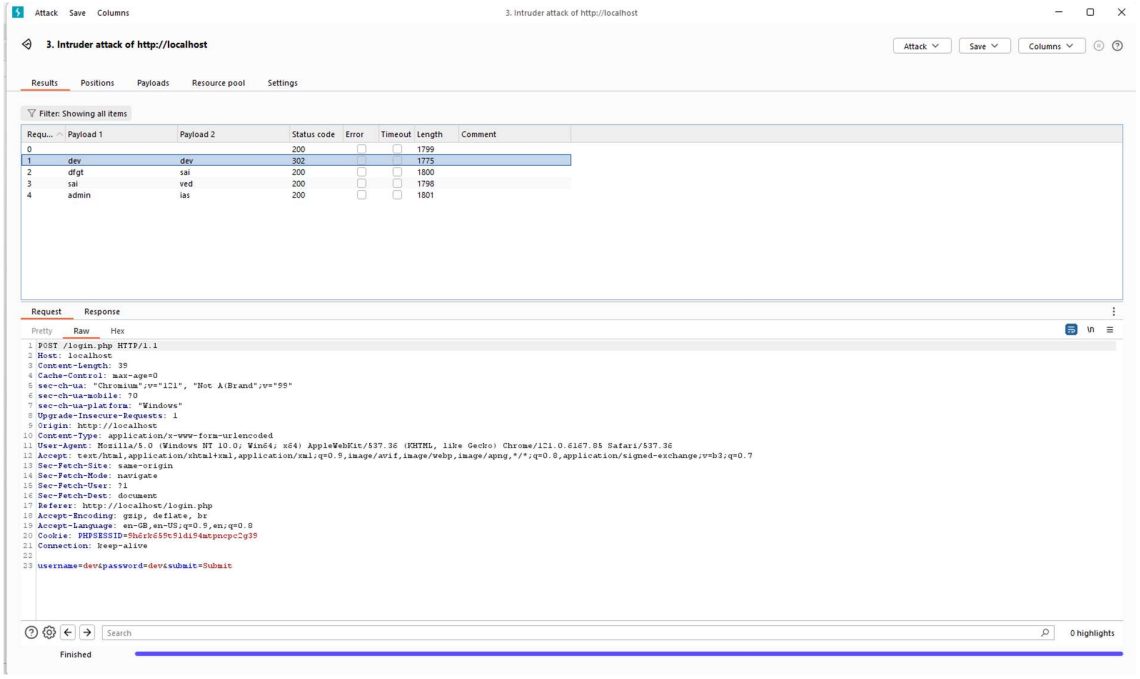
Payload Set - 1



Payload Set - 2



- Start the attack: Click the “Start attack” button.

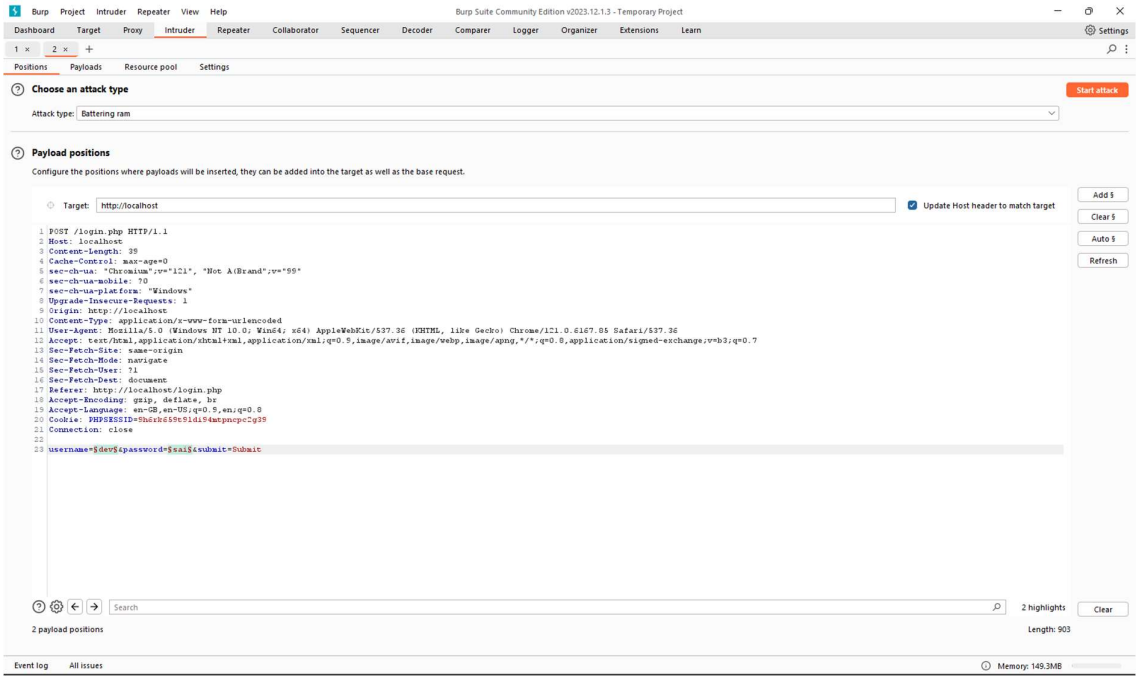


Observations :

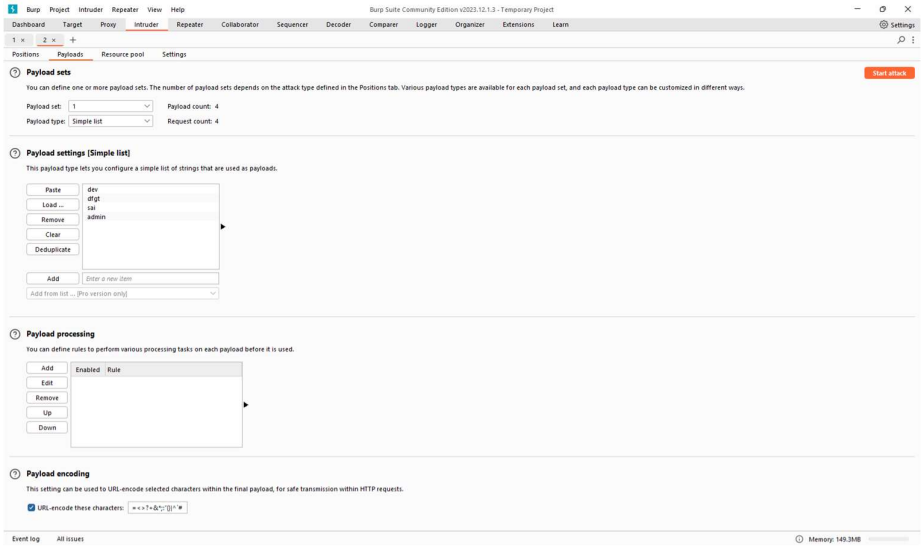
We observed that when username and password are correct then the **length section element is different from other lengths or the Status code is different from the other status codes**, hence it is right password and username.

Battering Ram :

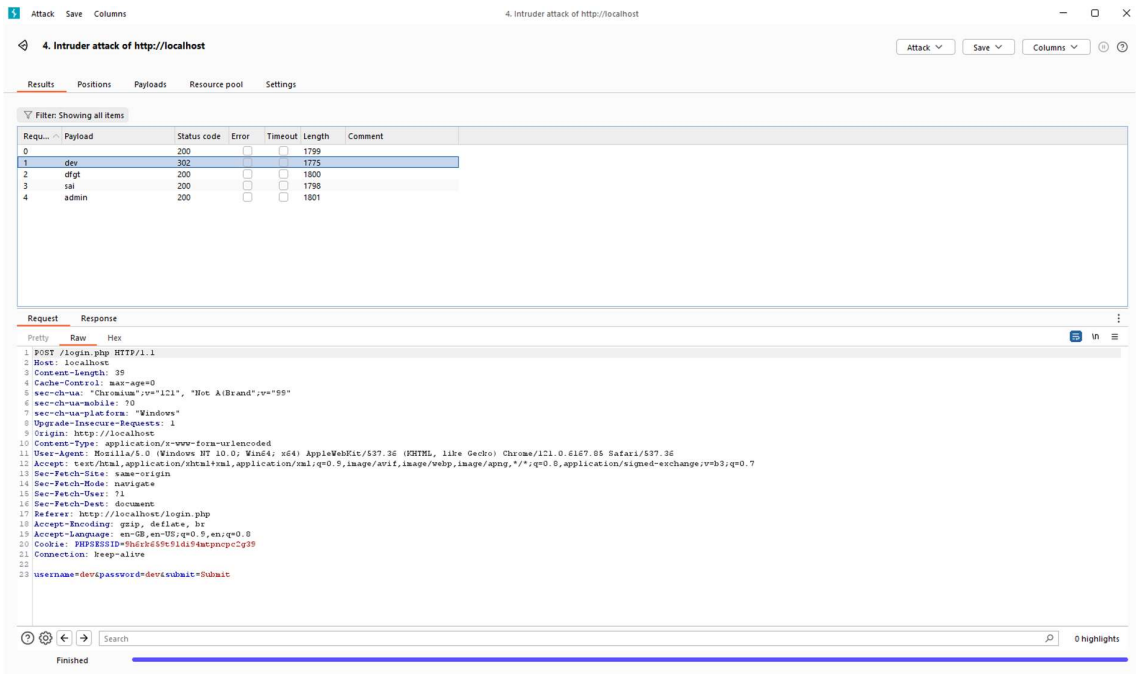
- Set the attack type: In the Intruder tab, set the attack type to "Battering Ram".



- Configure the payloads: Go to the Payloads tab and configure your payloads. You'll need to set up only one payload set.



- Start the attack: Click the “Start attack” button.

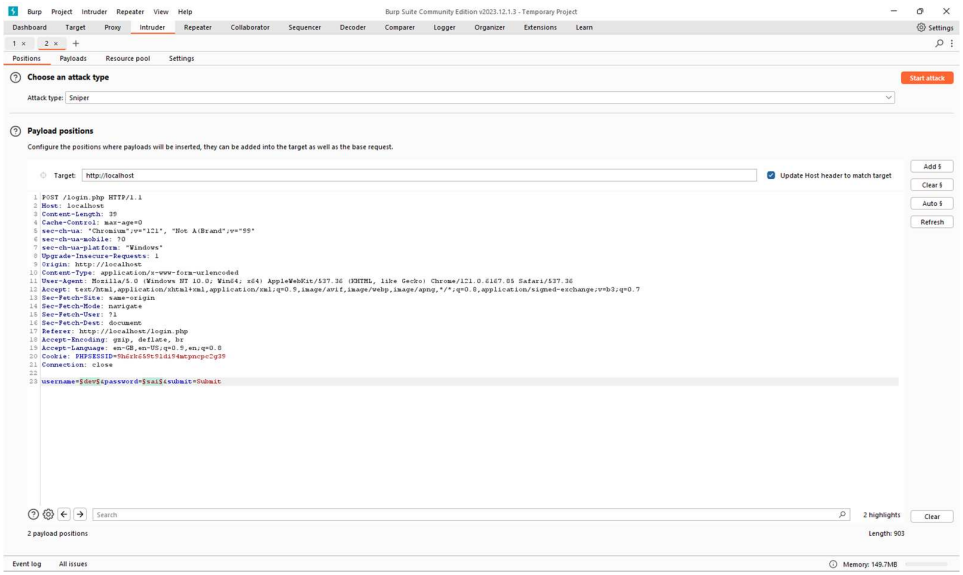


Observations :

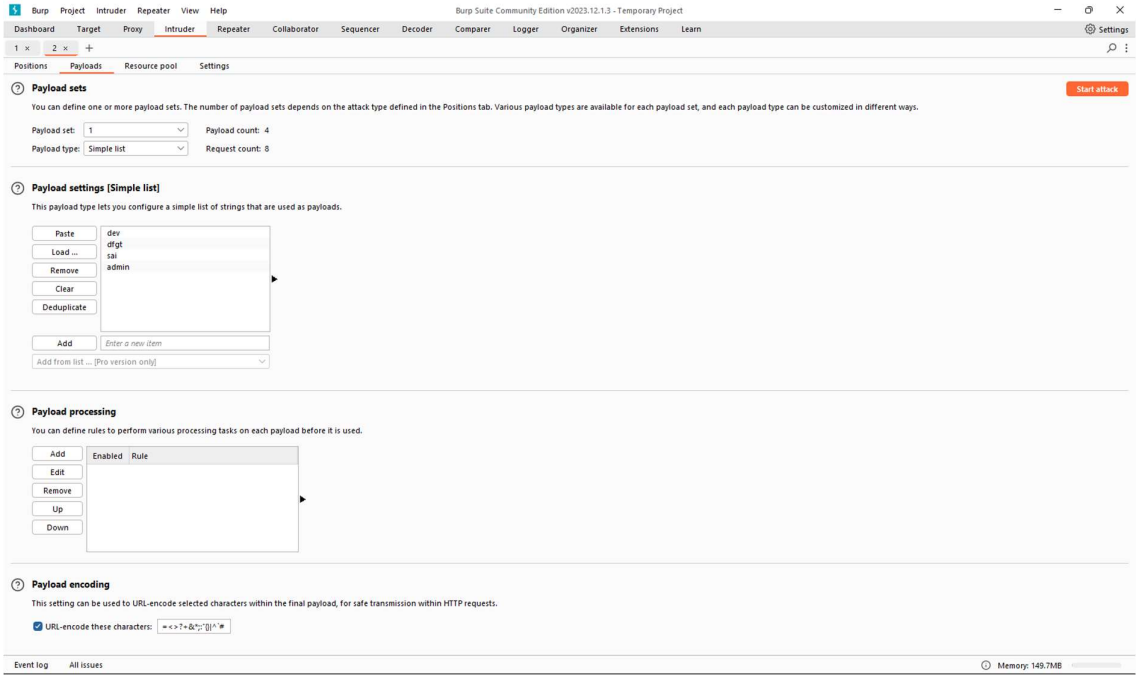
We observed that when username and password are correct then the **length section element is different from other lengths or the Status code is different from the other status codes**, hence it is right password and username.

Sniper :

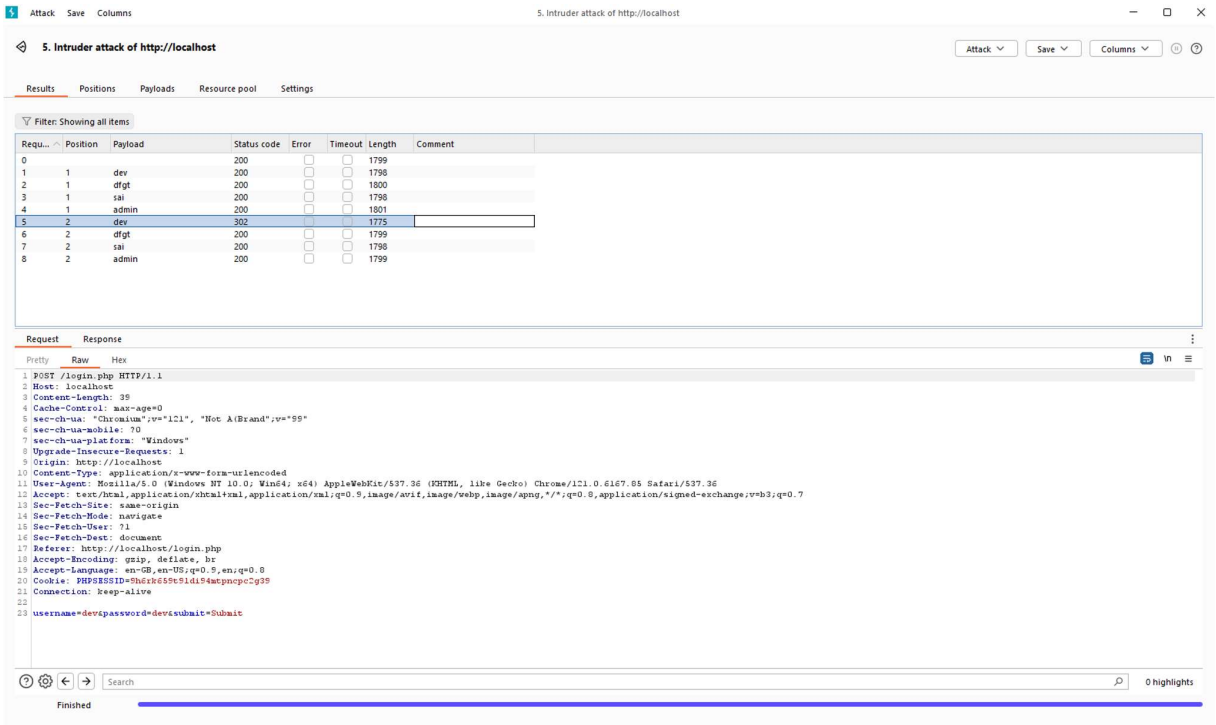
- Set the attack type: In the Intruder tab, set the attack type to "Sniper".



- Configure the payloads: Go to the Payloads tab and configure your payloads. You'll need to set up only one payload set.



- Start the attack: Click the "Start attack" button.



Observations :

We observed that when username and password are correct then the **length section element is different from other lengths or the Status code is different from the other status codes**, hence it is right password and username.

Experiment 3

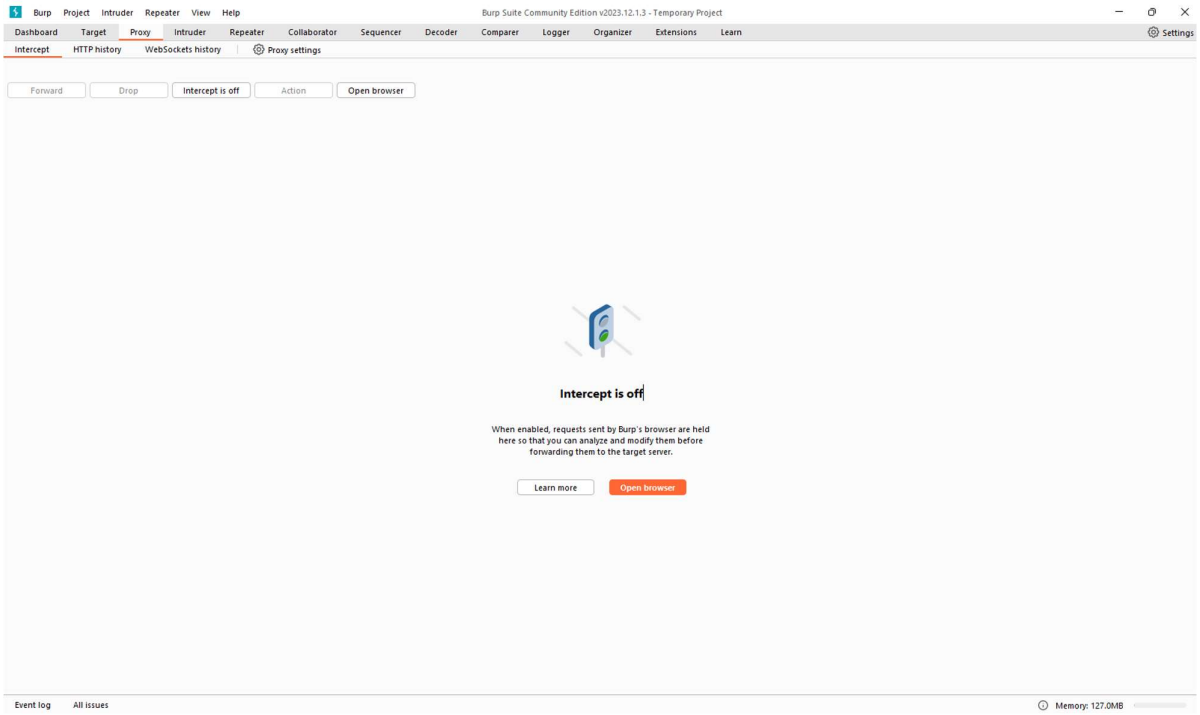
SQL Injection

Objective :

The primary objective of this experiment is to evaluate the vulnerability of the application to SQL Injection attacks. This is achieved by inputting special characters in the login page and observing the system's response. If the system throws an error, it may indicate a potential security flaw that could be exploited by malicious users to manipulate the SQL queries.

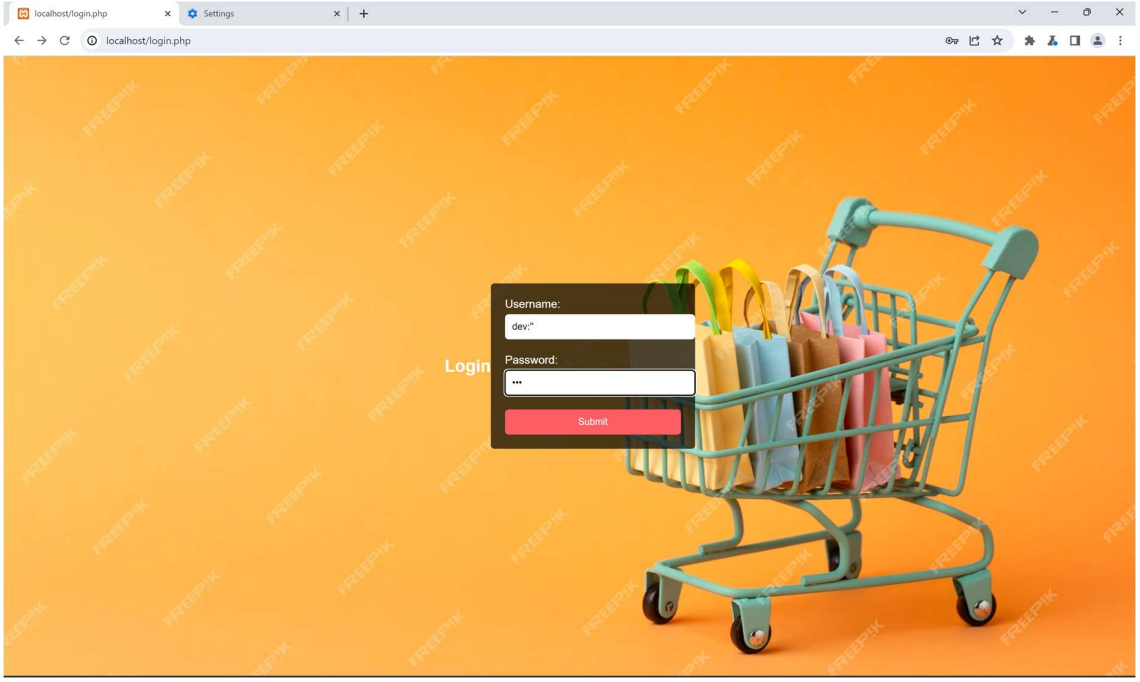
Setup :

Open burpsuite and in Dashboard->Proxy click on open browser and open the application in the browser on which we want to perform attack.



On our login page we are going to perform SQL Injection attack.

In here I gave Special Characters in username **dev:"** as shown in below Screen shot



When we clicked on login we gat ERROR as shown in below Screenshot, Which means it indicate a potential security flaw that could be exploited by malicious users to manipulate the SQL queries.

The issue we're facing is due to SQL Injection, which is a code injection technique that attackers can use to exploit vulnerabilities in a web application's database query. When we

directly insert user input into an SQL query, the user can manipulate the query by inserting special characters like '.

This is our original code :

```
<?php
// Start the session
session_start();

// Connect to the database
$servername = "localhost";
$username = "dev";
$password = "dev";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Check if form is submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Fetch user from the database
    $sql = "SELECT id FROM users WHERE username = '$username' and password = '$password'";
    $result = $conn->query($sql);
    $row = $result->fetch_assoc();
```

Solution :

To prevent this, we should use prepared statements with parameter binding. This ensures that the user input is not treated as part of the SQL command, but as a parameter that doesn't interfere with the query structure.

<u>Modified Code :</u>

```php
<?php
// Start the session
session_start();

// Connect to the database
$servername = "localhost";
$username = "dev";
$password = "dev";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Check if form is submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Fetch user from the database
    // $sql = "SELECT id FROM users WHERE username = '$username' and password = '$password'";
    // $result = $conn->query($sql);
    // $row = $result->fetch_assoc();

    // Prepare SQL statement
    $stmt = $conn->prepare("SELECT id FROM users WHERE username = ? and password = ?");
    $stmt->bind_param("ss", $username, $password);

    // Execute the statement
    $stmt->execute();

    // Get the result
    $result = $stmt->get_result();
    $row = $result->fetch_assoc();

    // If user exists, set session variable and redirect to cart page
    if($row){
        $_SESSION['loggedin'] = true;
        $_SESSION['username'] = $username;
        header("location: cart.php");
    } else {
        // Display the username without encoding
        echo "Invalid username or password. You entered: " . $username;
    }
}
?>
```

In this modified code, We've replaced the $sql variable with a prepared statement that uses ? placeholders for the username and password. The bind_param function binds the actual values to these placeholders. This way, even if the user input contains special characters, they will not affect the SQL query structure. This change will prevent SQL Injection attacks and solve our issue.

In the previous code, we were directly inserting user input into an SQL query. This can lead to SQL Injection attacks when special characters are used in the input.

Here’s what I changed:

1. **Prepared the SQL statement with placeholders**: Instead of directly inserting user input into the SQL query, I prepared the SQL statement with ? placeholders for the username and password.

   PHP

   $stmt = $conn->prepare("SELECT id FROM users WHERE username = ? and password = ?");

2. **Bound the actual values to the placeholders**: I used the bind_param function to bind the actual username and password values to these placeholders.

   $stmt->bind_param("ss", $username, $password);

3. **Executed the statement and fetched the result**: I then executed the prepared statement and fetched the result.

   PHP

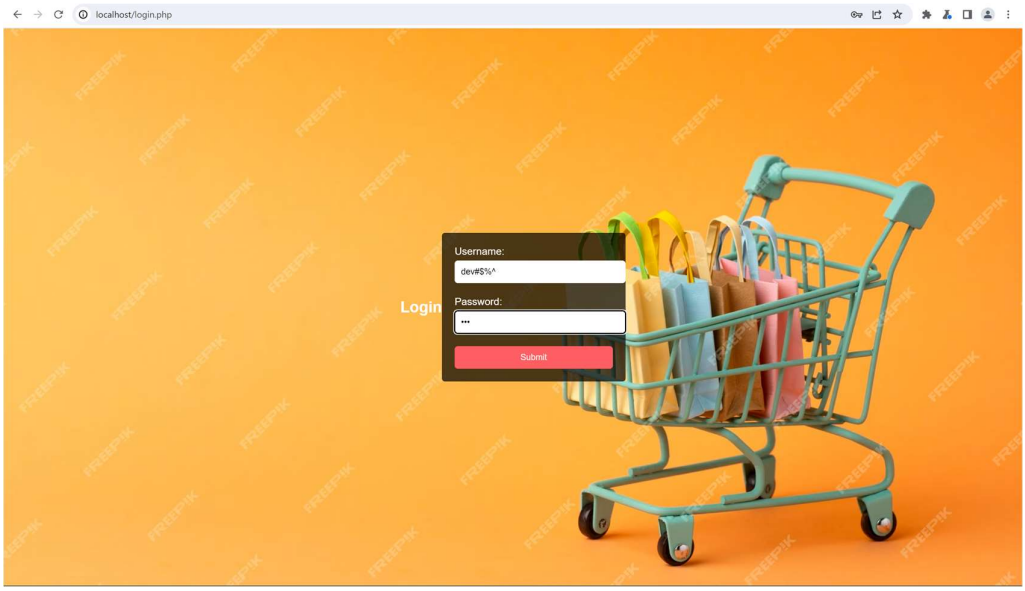
   $stmt->execute();

   $result = $stmt->get_result();

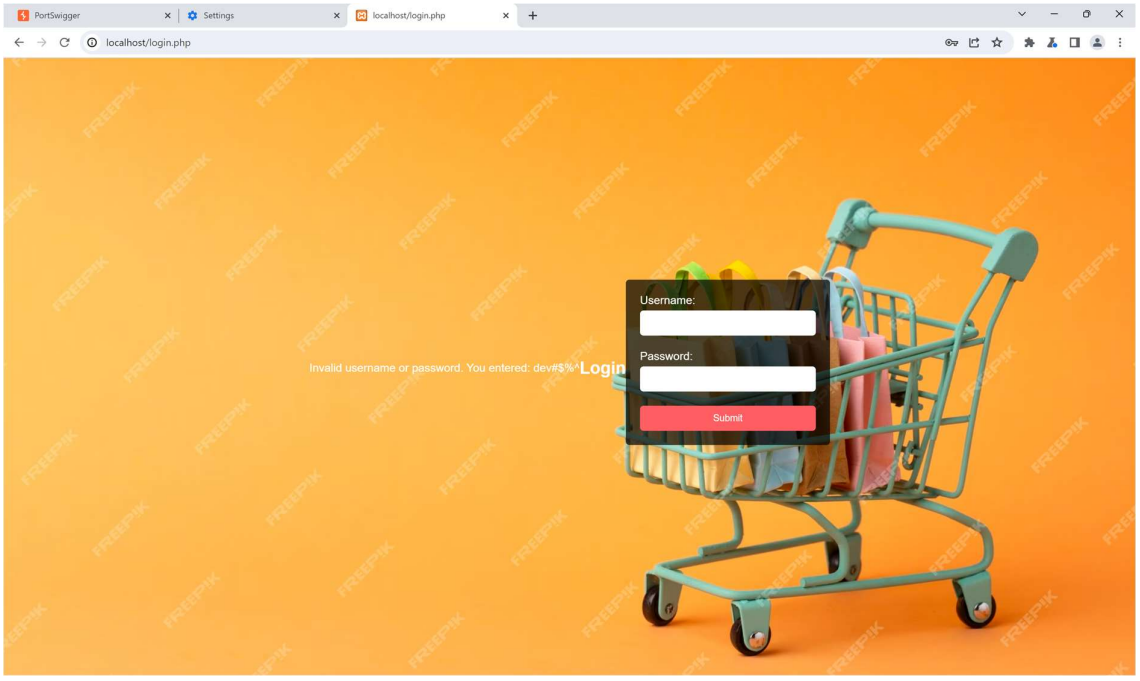
   $row = $result->fetch_assoc();

These changes ensure that the user input is not treated as part of the SQL command, but as a parameter that doesn’t interfere with the query structure. This prevents SQL Injection attacks and allows special characters to be used in the username and password without causing any issues.

Now Let’s login again using Special Characters



Now, it’s showing invalid even if we entered Special Characters and not showing any error, hence SQL Injection is prevented.



Experiment 4

Reflected Cross Site Scripting (XSS)

Objective of the Exercise :

The objective of this exercise is to understand the potential security risks associated with unsanitized user inputs in web applications. By echoing back the username without any sanitization or encoding, we expose our application to Cross-Site Scripting (XSS) attacks.

Introduction :

Cross-Site Scripting (XSS) is a type of security vulnerability typically found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. In this exercise, we intentionally introduce an XSS vulnerability to understand its implications and the importance of sanitizing user inputs.

Setup :

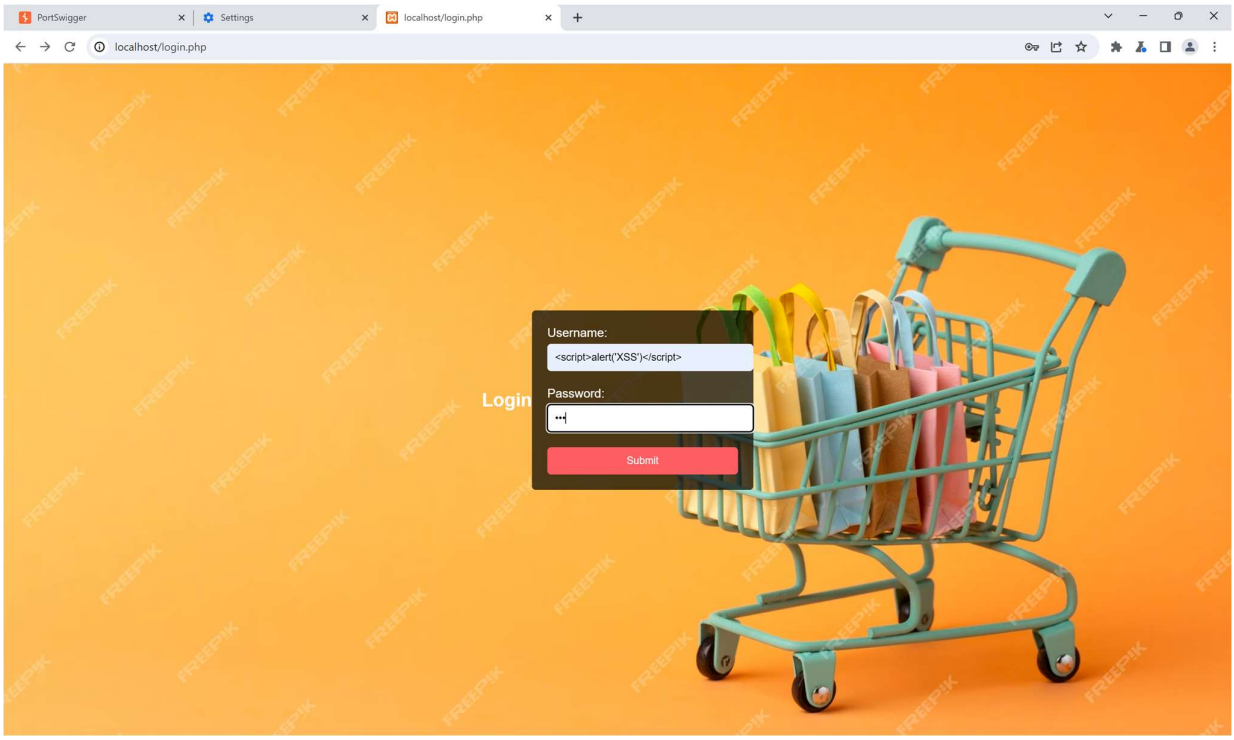
- A web server that supports PHP (such as Apache or Nginx)
- A code editor or IDE
- A web browser that supports JavaScript (such as Chrome or Firefox)
- Optionally, a database server (such as MySQL or PostgreSQL) if you want to store user data

The setup involves a simple PHP application with a login form. The form includes fields for the username and password. In the original code, upon unsuccessful login, the application simply echoes “Invalid username or password.” In the modified code, the application echoes back the entered username along with the invalid login message. This is done without sanitizing the user input, thereby introducing an XSS vulnerability.

Experimental Steps :

The username entered by the user is echoed back without any sanitization or encoding. If a user enters a string like <script>alert('XSS')</script> as their username, the script will be executed when the page is loaded.

Enter a string like <script>alert('XSS')</script> in username



Click on Login

The script is executed when the page is loaded.

The conclusion for the above attack is that echoing user input directly without sanitization or encoding can lead to Cross-Site Scripting (XSS) attacks. If a user enters a string like <script>alert('XSS')</script> as their username, the script will be executed when the page is loaded. This can have serious consequences, such as stealing session cookies, redirecting users to phishing sites, or installing malware on their devices.

Solution :

Original Code:

PHP

```
// Display the username without encoding

echo "Invalid username or password. You entered: " . $username;
```

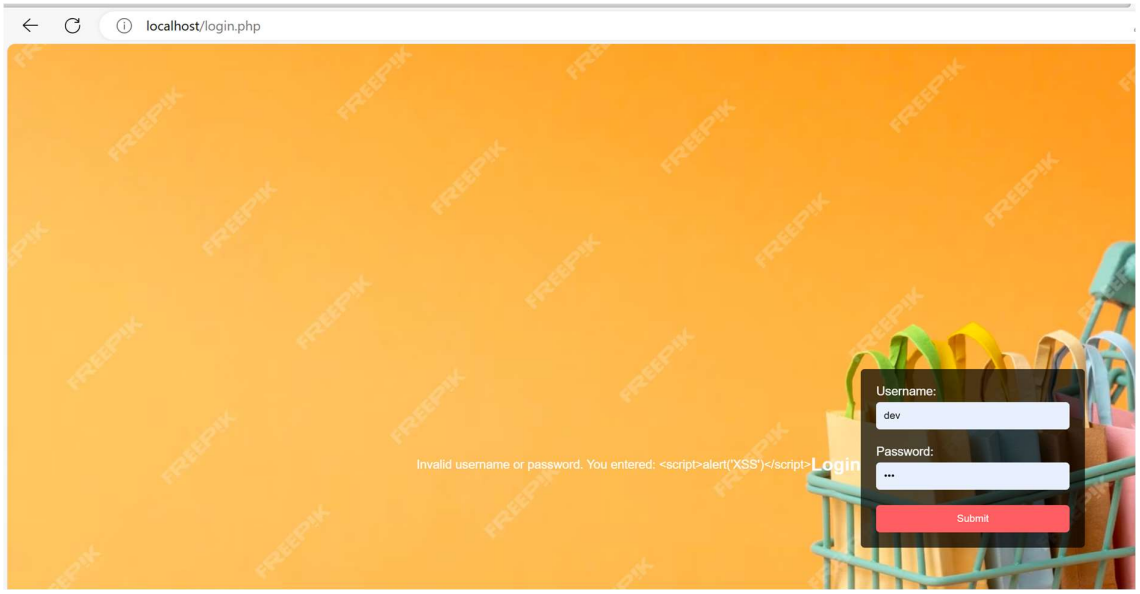
Modified Code (Solution) :

```
// Display the username with encoding

echo "Invalid username or password. You entered: " . htmlspecialchars($username, ENT_QUOTES, 'UTF-8');
```

To prevent XSS attacks, it’s crucial to sanitize and encode user inputs before displaying them on the web page. In PHP, this can be done using function like **htmlspecialchars**. This

function convert special characters to their HTML entities, preventing them from being interpreted as HTML tags or JavaScript code.

Now if try login again as we can see in below screenshot it shows invalid.



Stored Cross Site Scripting :

Introduction :

Stored Cross-Site Scripting (XSS) is a type of security vulnerability that affects web applications. It occurs when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

In a stored XSS attack, the attacker injects malicious scripts into a target website, which are then permanently stored and displayed to other users. For example, suppose a website allows users to submit comments on blog posts. An attacker can submit a comment containing a malicious script, like <script>alert('XSS')</script>. This comment would then be stored on the server and displayed to any user who views the blog post.

When an unsuspecting user views the blog post, their browser executes the embedded JavaScript, thinking that it is the legitimate code of the webpage. The script supplied by the attacker will then execute in the victim user's browser, in the context of their session with the application.

Stored XSS attacks are particularly dangerous because they allow the attacker to attack many victims with a single payload. The attacker does not need to find an external way of inducing other users to make a particular request containing their exploit. Rather, the attacker places their exploit into the application itself and simply waits for users to encounter it.

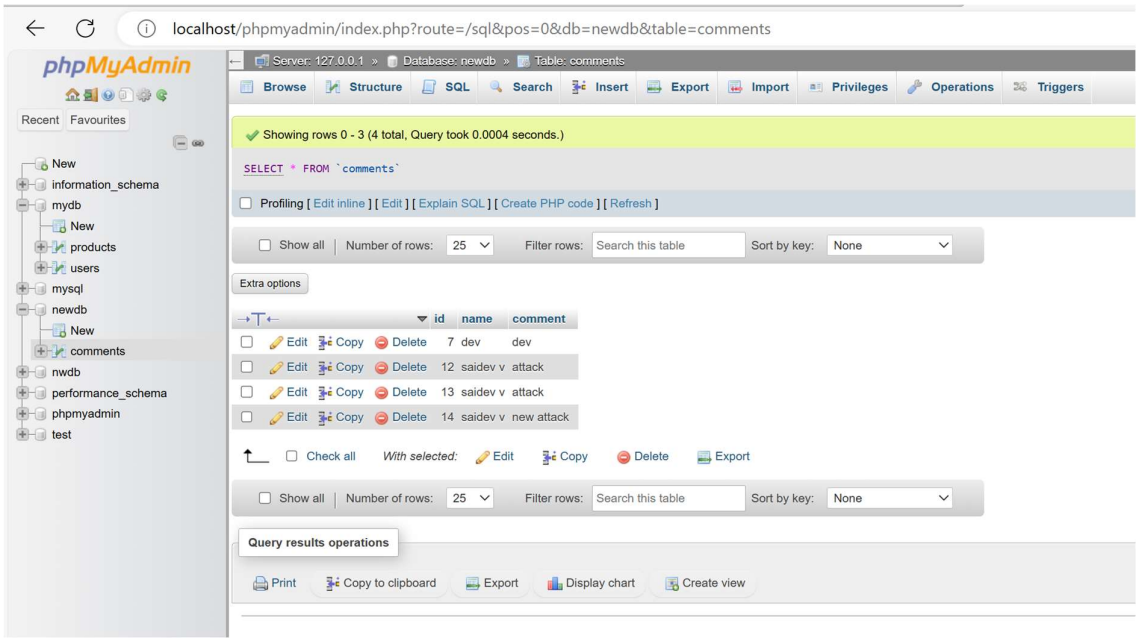
The impact of stored XSS attacks can be severe. If an attacker can control a script that is executed in the victim's browser, then they can typically fully compromise that user. They can steal session cookies, leak personal information, change site contents, or redirect the user to a malicious site. It's important to take measures to prevent such attacks in any web application.

Setup :

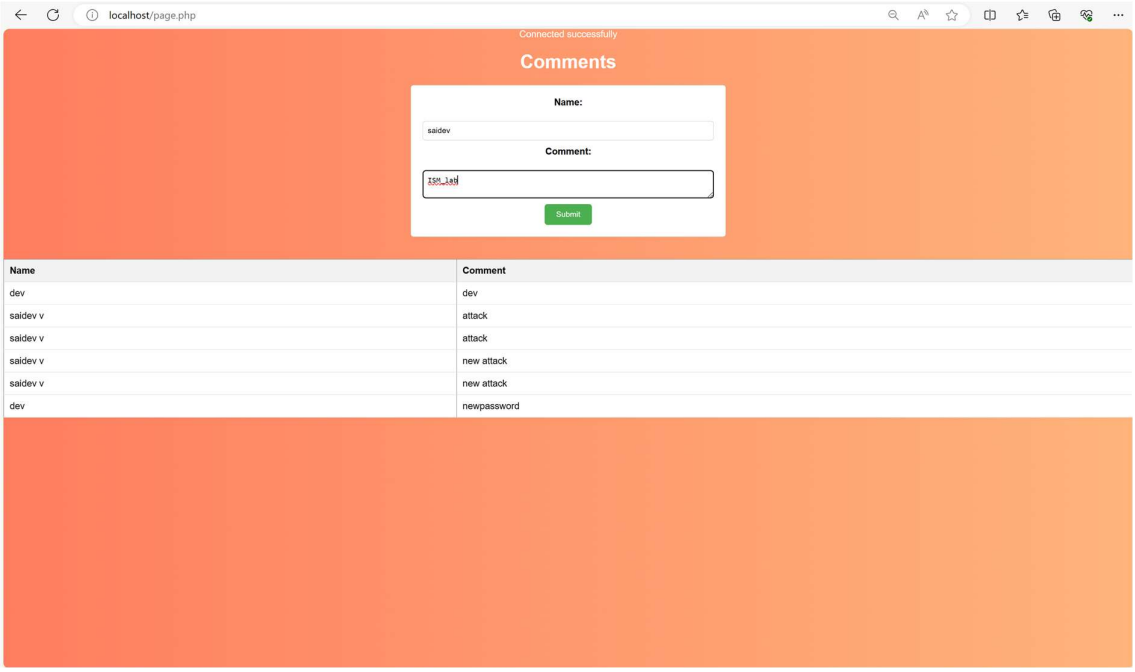
- A web server that supports PHP (such as Apache or Nginx)
- A code editor or IDE
- A web browser that supports JavaScript (such as Chrome or Firefox)
- Optionally, a database server (such as MySQL or PostgreSQL) if you want to store user data

Experimental Steps :

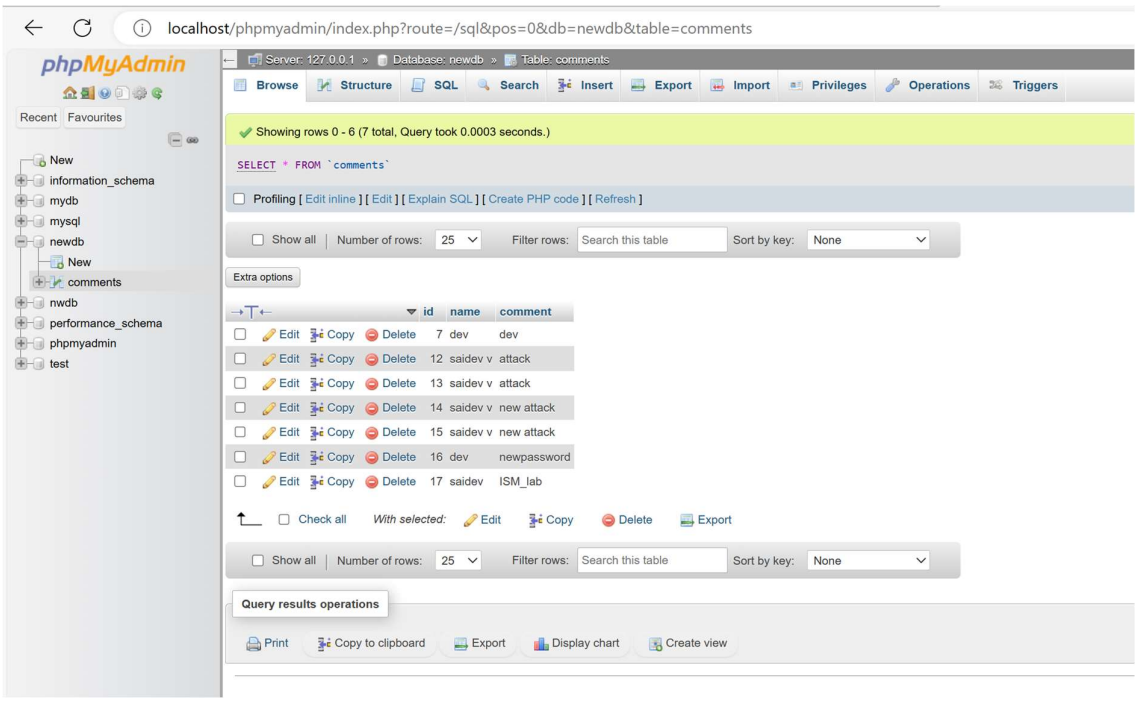
To perfrorm Stored XSS we have created a new dabase named newdb as shown in below Screenshot

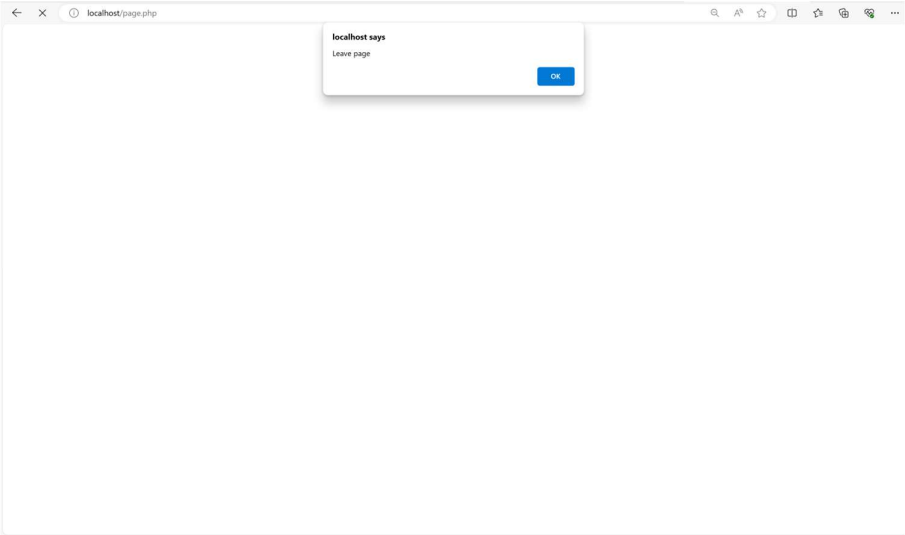


Also We created page.php



Now in here we can see that the username and password are getting stored in the database.

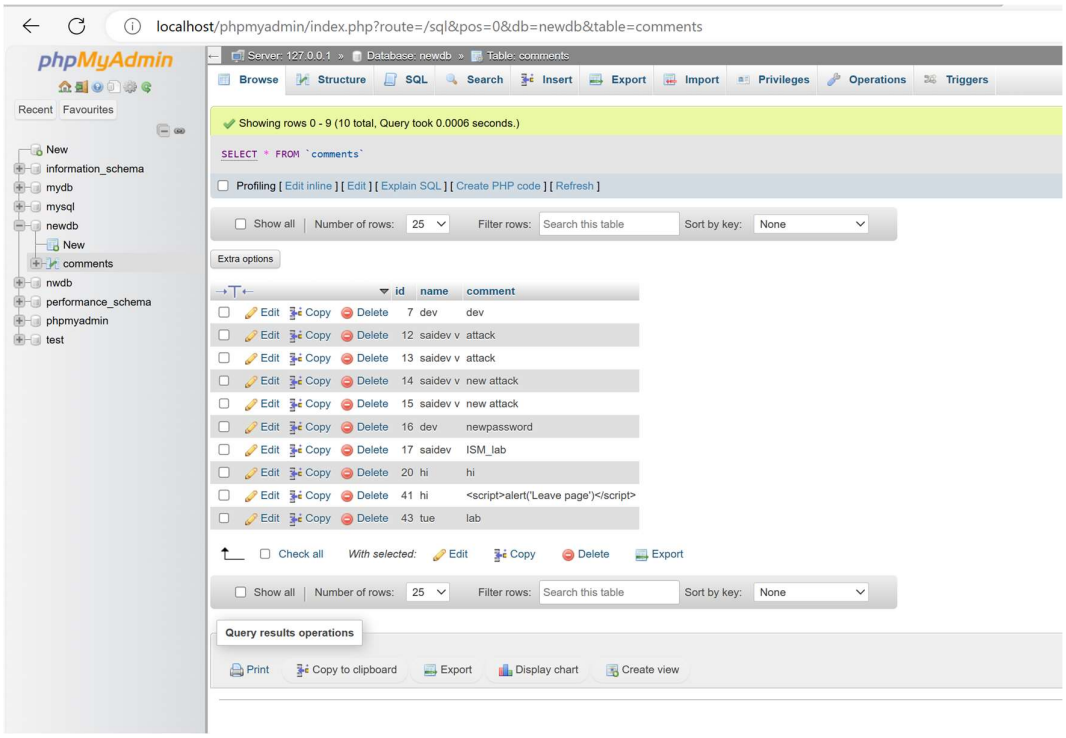




.

To demonstrate an XSS attack, we can input a JavaScript snippet in the name or comment field, like <script>alert('XSS')</script>. When the page reloads, the script will execute and show a popup alert with the message 'XSS'. This is a simple demonstration of an XSS attack.

To prevent this attack, you should sanitize the user input before outputting it to the webpage. You can use the htmlspecialchars function in PHP to convert special characters to their HTML entities. This will prevent the browser from interpreting the user input as HTML or JavaScript code.



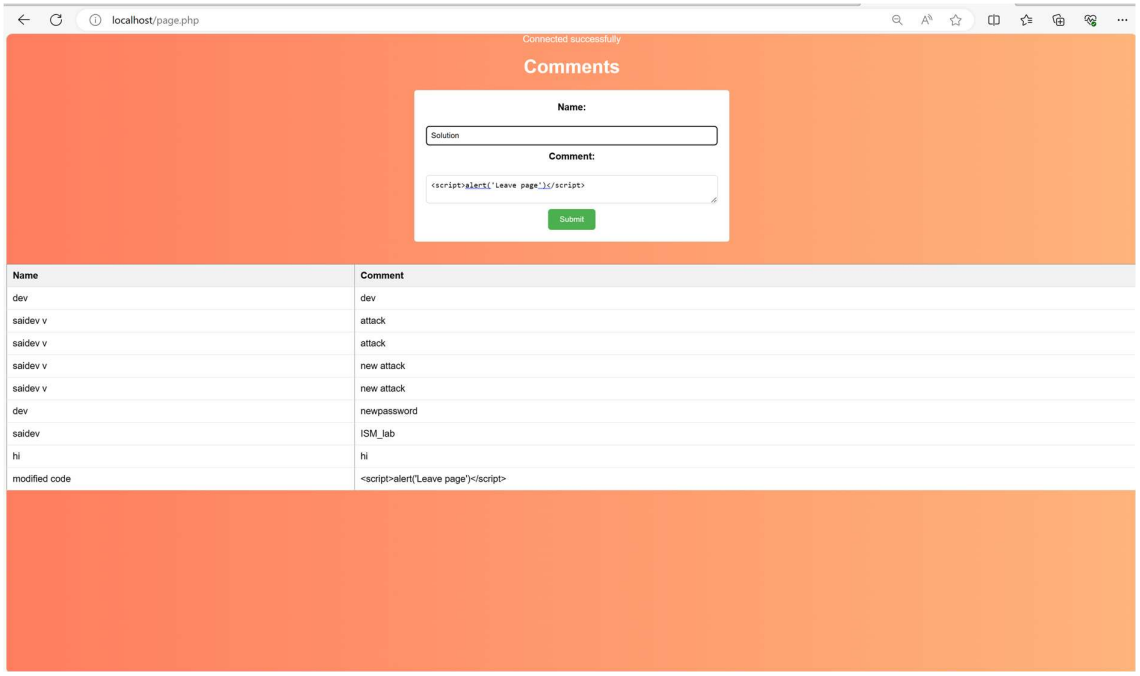
Solution :

<u>Here's how we modified our code to prevent XSS attacks:</u>

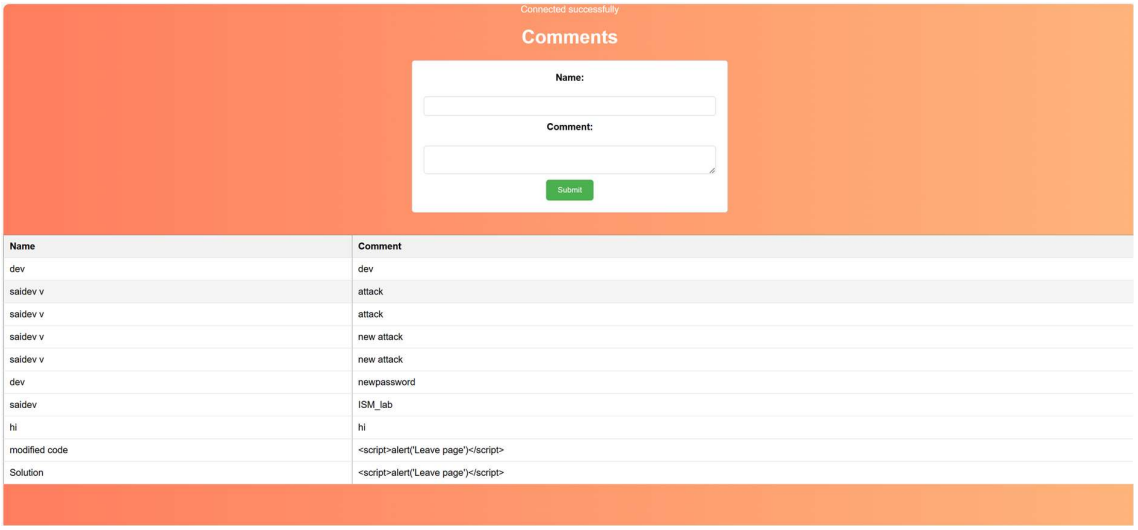
```
<table border="1">
    <tr>
        <th>Name</th>
        <th>Comment</th>
    </tr>
    <?php foreach ($comments as $comment) : ?>
        <tr>
            <!-- Sanitize user input before outputting -->
            <td><?= htmlspecialchars($comment['name'], ENT_QUOTES, 'UTF-8') ?></td>
            <td><?= htmlspecialchars($comment['comment'], ENT_QUOTES, 'UTF-8') ?></td>
        </tr>
    <?php endforeach; ?>
</table>
```

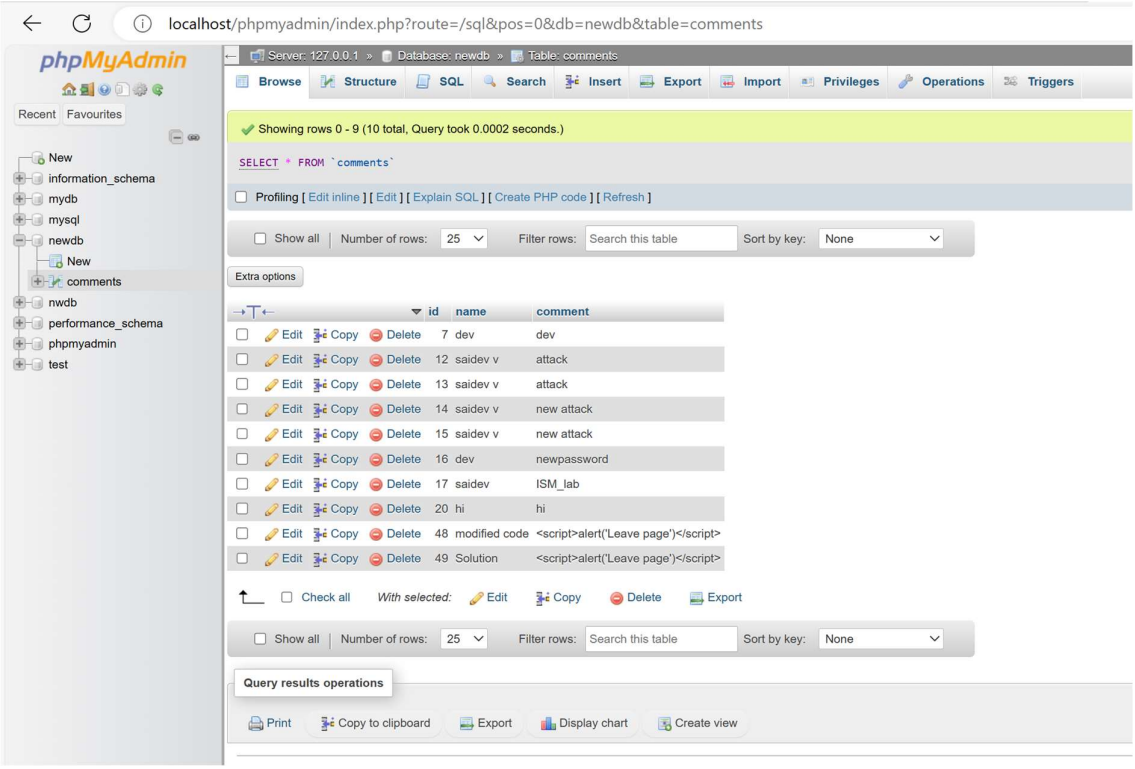
```
    <table border="1">
    <tr>
        <th>Name</th>
        <th>Comment</th>
    </tr>
    <?php foreach ($comments as $comment) : ?>
        <tr>
            <!-- Sanitize user input before outputting -->
            <td><?= htmlspecialchars($comment['name'], ENT_QUOTES, 'UTF-8') ?></td>
            <td><?= htmlspecialchars($comment['comment'], ENT_QUOTES, 'UTF-8') ?></td>
        </tr>
    <?php endforeach; ?>
</table>
```

This modification ensures that the user's name and comment are sanitized before being displayed on the webpage, preventing potential XSS attacks.



After executing above script like previous response it is not interpreting the user input as HTML or JavaScript code





That’s because the htmlspecialchars function in PHP is doing its job correctly. It converts special characters to their HTML entities, which prevents the browser from interpreting the user input as HTML or JavaScript code.

In our case, the <script> tags and their content are being converted to <script> and </script>, respectively. This means that when the user input is displayed on the webpage, it’s displayed as plain text, not as HTML or JavaScript code.

So, when you enter <script>alert("Leave Page");</script> in the comments, it’s displayed on the webpage as <script>alert("Leave Page");</script>. The browser doesn’t interpret this as a script, so no popup alert is shown.

Experiment 5
Cross Site Request Forgery

Experimental Objective :

The objective of this exercise is to understand how CSRF attacks work and how they can be mitigated. We will create a mock exploit page that will attempt a CSRF attack on our own application.

Setup :

Exploit Page code :

```
<!DOCTYPE html>
<html>
<body>
<h2>Exploit Page</h2>
<form method="post" action="http://localhost/login.php">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" value="attacker">
  <label for="password">Password:</label>
  <input type="password" id="password" name="password" value="attackerpassword">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

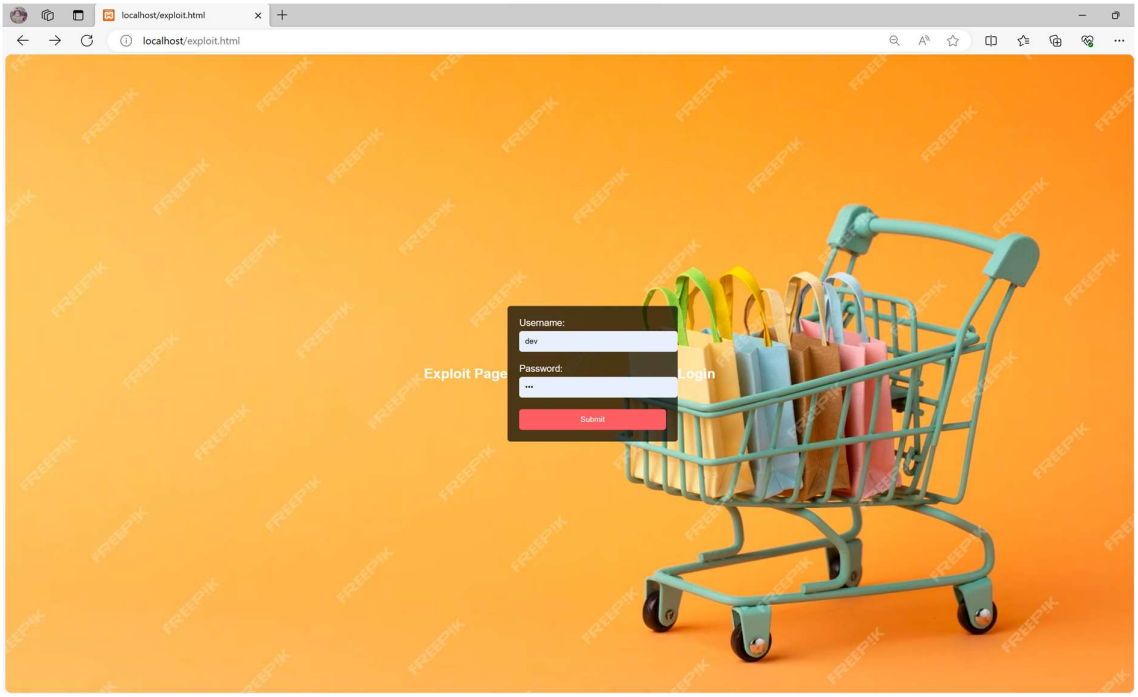
In this Exploit page code in action I have given my login page link to perfrom attack as shown in above Screenshot and after that I have updated the code with styles similar to look like my application.

Induce the Victim to Open the Exploit Page: We could do this by sending the victim a link to the exploit page. When the victim opens the page, the form will automatically submit, sending a POST request to our login.php page.

In the context of a CSRF attack, an exploit page would contain a form that mimics a form on our target page (in our case, login.php). When a user submits the form on the exploit page, it sends a request to our target page, triggering the vulnerability.

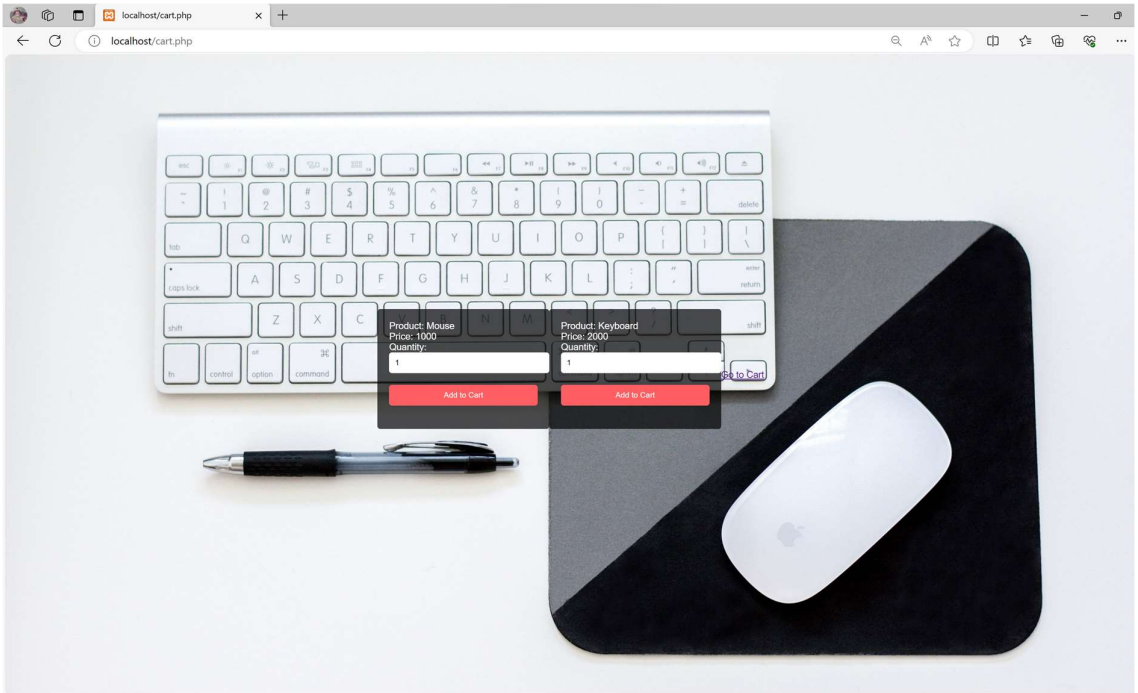
In here, the form action points to our login.php page. The username and password fields are pre-filled with the attacker's credentials. When a user visits this page and submits the form, it will send a POST request to our login.php page, logging them in as the attacker.

1. **Exploit Page:** Create an exploit page that contains a form identical to the one in our application. The action of the form should point to the form processing page (Login page) of our application.



2. **Attack Execution:** On the exploit page, submit the form. If the application is vulnerable to CSRF, the request will be processed as if it was made from the actual application.

After Logging in it is directly redirected to Cart page through the exploit page.



3. **Observation:** Observe the behavior of your application. If it processes the request, it confirms the presence of a CSRF vulnerability.

Since, it processed the request we can confirm the presence of a CSRF vulnerability.

4. **Mitigation:** Implement CSRF tokens in your application and repeat the process. The application should now reject the request made from the exploit page.

<u>Original Code :</u>

login.php | cart.php | checkout.php | exploit

File Edit View

```php
<?php
// Start the session
session_start();

// Connect to the database
$servername = "localhost";
$username = "dev";
$password = "dev";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Check if form is submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Fetch user from the database
    // $sql = "SELECT id FROM users WHERE username = '$username' and password = '$password'";
    // $result = $conn->query($sql);
    // $row = $result->fetch_assoc();

    // Prepare SQL statement
    $stmt = $conn->prepare("SELECT id FROM users WHERE username = ? and password = ?");
    $stmt->bind_param("ss", $username, $password);

    // Execute the statement
    $stmt->execute();

    // Get the result
    $result = $stmt->get_result();
    $row = $result->fetch_assoc();

    // If user exists, set session variable and redirect to cart page
    if($row){
        $_SESSION['loggedin'] = true;
        $_SESSION['username'] = $username;
        header("location: cart.php");
    } else {
        // Display the username without encoding
        echo "Invalid username or password. You entered: " . $username;
    }
}
?>
```

Solution :

I made the following changes to add Cross-Site Request Forgery (CSRF) protection:

1. **Added CSRF Token Generation**: At the start of the session, I added a check to see if a CSRF token is already set in the session. If not, a new token is generated and stored in the session. This token is a random string that will be used to verify that form submissions are coming from your site and not an external source.

```
if (!isset($_SESSION['token'])) {
   $_SESSION['token'] = bin2hex(random_bytes(32));
}
```

2. **Added CSRF Token to Form**: I added a hidden input field to our form that includes the CSRF token. This token will be sent along with the form data when the form is submitted.

```
<input type="hidden" name="token" value="<?php echo $_SESSION['token'] ?>">
```

3. **Checked CSRF Token on Form Submission**: When the form is submitted, I added a check to verify that the CSRF token in the POST data matches the token stored in the session. If they don't match, or if no token is provided, the script terminates with an error message. This prevents an attacker from submitting the form from an external site.

```
if (!isset($_POST['token']) || $_POST['token'] !== $_SESSION['token']) {
   die('Invalid CSRF token');
}
```

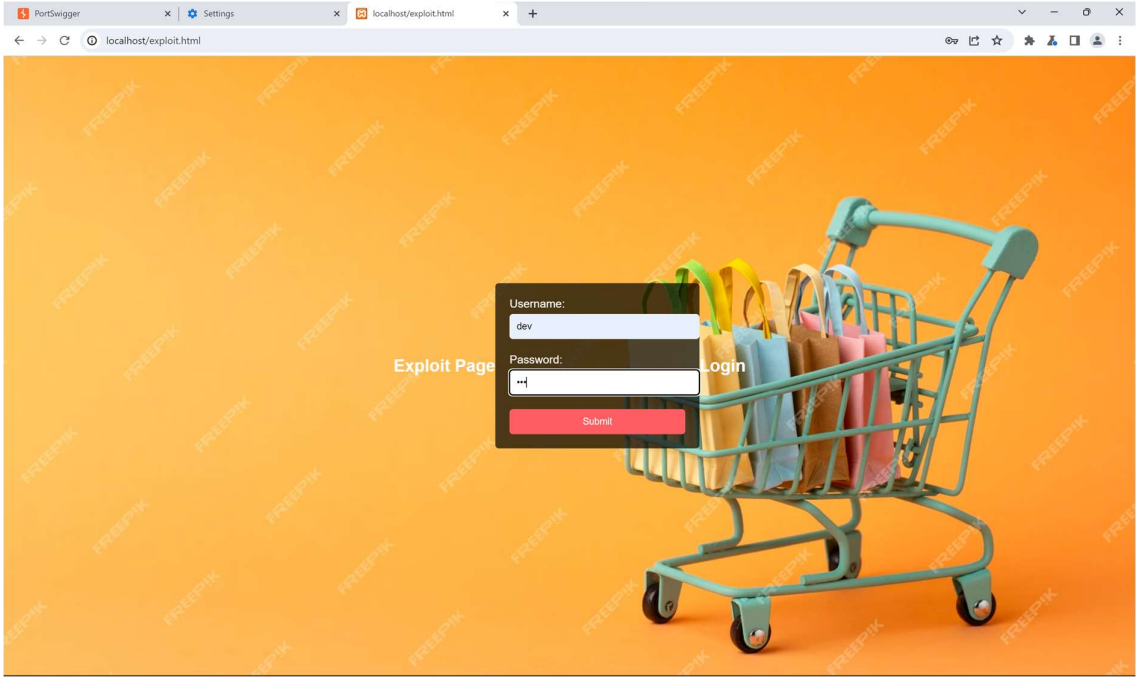
Updated Code :

login.php | cart.php | checkout.php | exploit

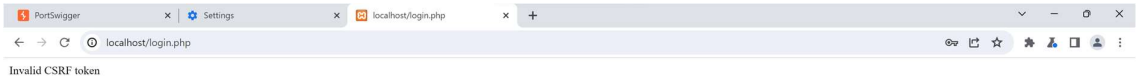
File Edit View

```php
<?php
// Start the session
session_start();

// Generate a CSRF token if one doesn't exist
if (!isset($_SESSION['token'])) {
    $_SESSION['token'] = bin2hex(random_bytes(32));
}

// Connect to the database
$servername = "localhost";
$username = "dev";
$password = "dev";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// Check if form is submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){
    // Check CSRF token
    if (!isset($_POST['token']) || $_POST['token'] !== $_SESSION['token']) {
        die('Invalid CSRF token');
    }

    $username = $_POST['username'];
    $password = $_POST['password'];

    // Prepare SQL statement
    $stmt = $conn->prepare("SELECT id FROM users WHERE username = ? and password = ?");
    $stmt->bind_param("ss", $username, $password);

    // Execute the statement
    $stmt->execute();

    // Get the result
    $result = $stmt->get_result();
    $row = $result->fetch_assoc();

    // If user exists, set session variable and redirect to cart page
    if($row){
        $_SESSION['loggedin'] = true;
        $_SESSION['username'] = $username;
        header("location: cart.php");
    } else {
        // Display the username without encoding
        echo "Invalid username or password. You entered: " . $username;
    }
}
?>
```

Now, Let’s login again using Exploit page



Here, it shows invalid CSRF token which means the session id created is not matched so the attacker unable to get the credentials of user.



We have Succesfully prevented the CSRF attack using tokens, Here we are verifying that the CSRF token in the POST data matches the token stored in the session. If they don't match, or if no token is provided, the script terminates with an error message. This prevents an attacker from submitting the form from an external site.

Experiment – 6

Objective: The objective of the process is to perform securely exchange information between two systems using Kleopatra. This involves:

1. **Signing Messages**: You create a digital signature for your message using your private key. This signature serves as a guarantee that the message originates from you and hasn't been tampered with.
2. **Sharing Public Key**: You share your public key with the receiver. They will use this key to verify your signature and ensure the message indeed comes from you.
3. **Certifying User ID**: On the receiver's system, they certify the user ID that you've shared. This step confirms the identity of the sender and establishes a trust relationship.
4. **Decrypting/Verifying Message**: The receiver then decrypts the message and verifies the signature using your public key. If the verification is successful, it confirms that the message was indeed sent by you and wasn't altered during transmission.

By following these steps, we're ensuring the secure exchange of information, where the receiver can trust the authenticity and integrity of the received message.

Introduction:

In the realm of digital communication, ensuring the authenticity and integrity of messages is paramount. One widely used tool for this purpose is Kleopatra, a certificate manager and GUI for GnuPG. The software is part of the Gpg4win initiative, which aims to provide an easy-to-install package for encryption and decryption using GPG on Windows.

Setup:

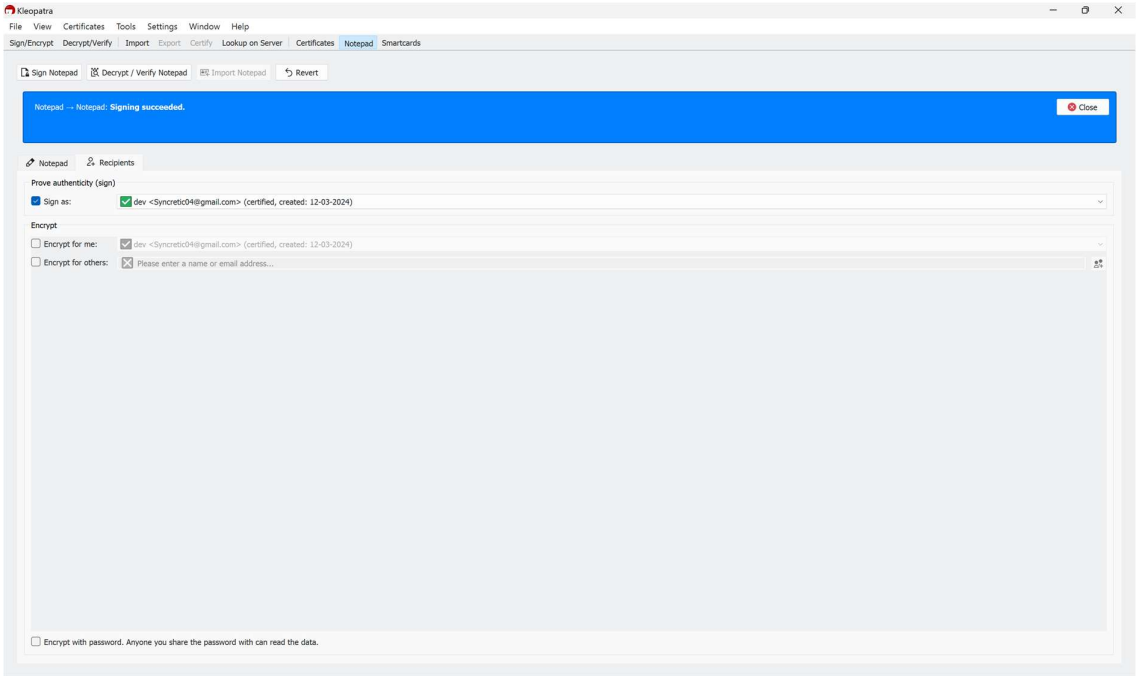
Hardware:

- A computer system with a network connection for exchanging messages.

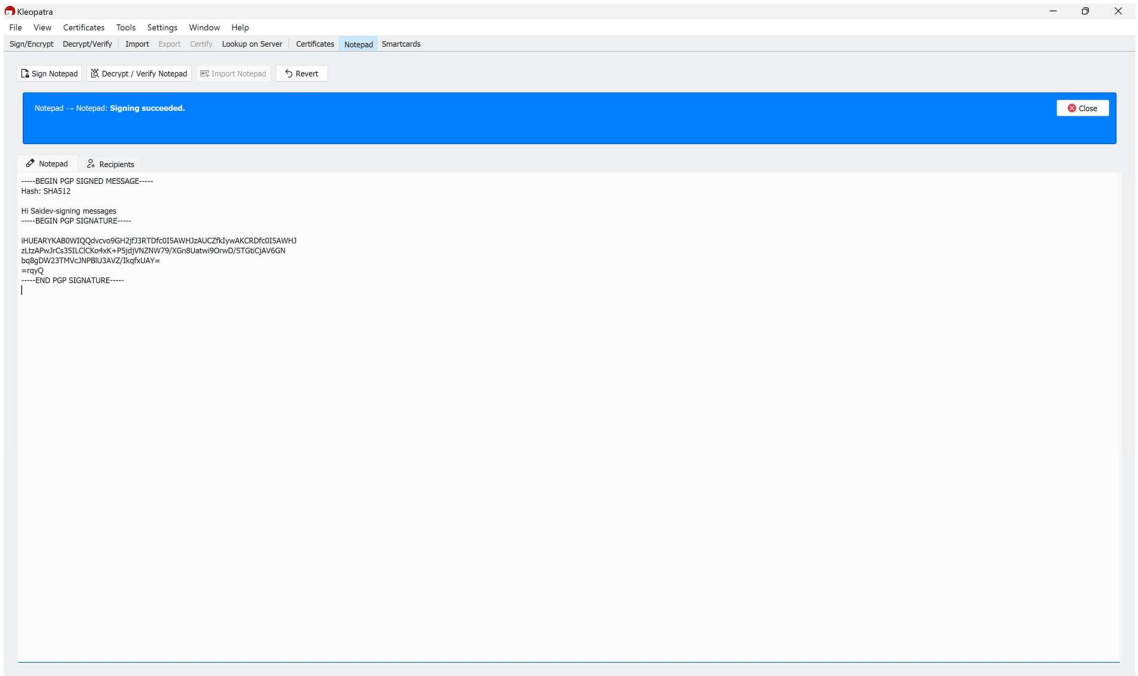
Software:

- **Operating System**: Windows, as Kleopatra is part of the Gpg4win package designed for Windows.
- **Gpg4win**: This is the main software package that includes Kleopatra. You can download it from the official Gpg4win website.
- **Kleopatra**: This is the certificate manager and GUI for GnuPG included in Gpg4win. It's used for creating and managing your keys, signing and encrypting your messages, and verifying and decrypting received messages.

Signing and Verifying Messages

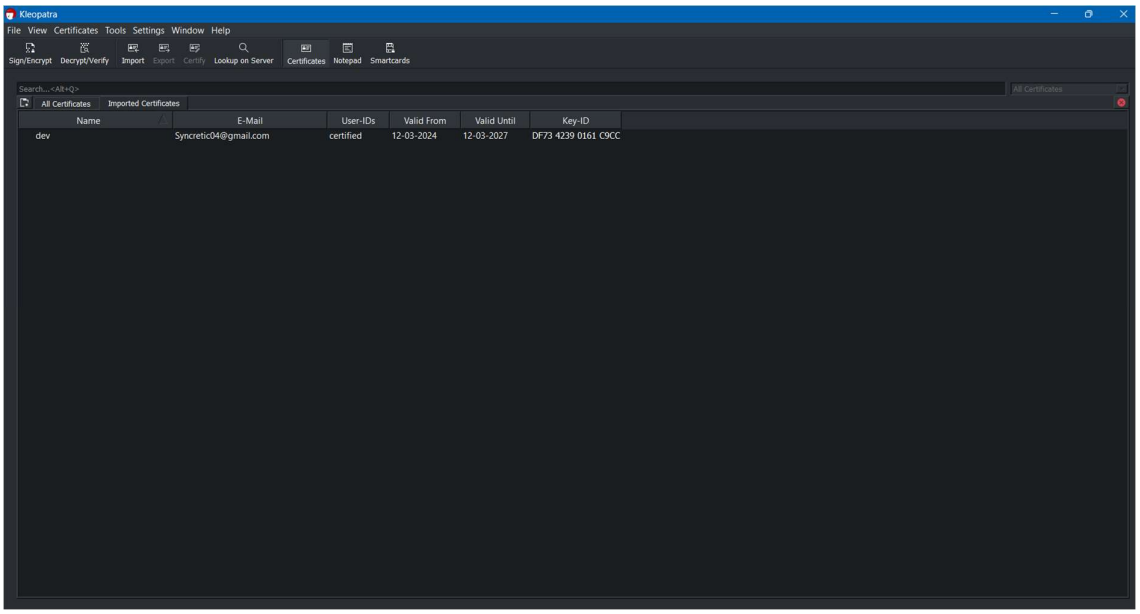


In below screenshot we can see the message with signature

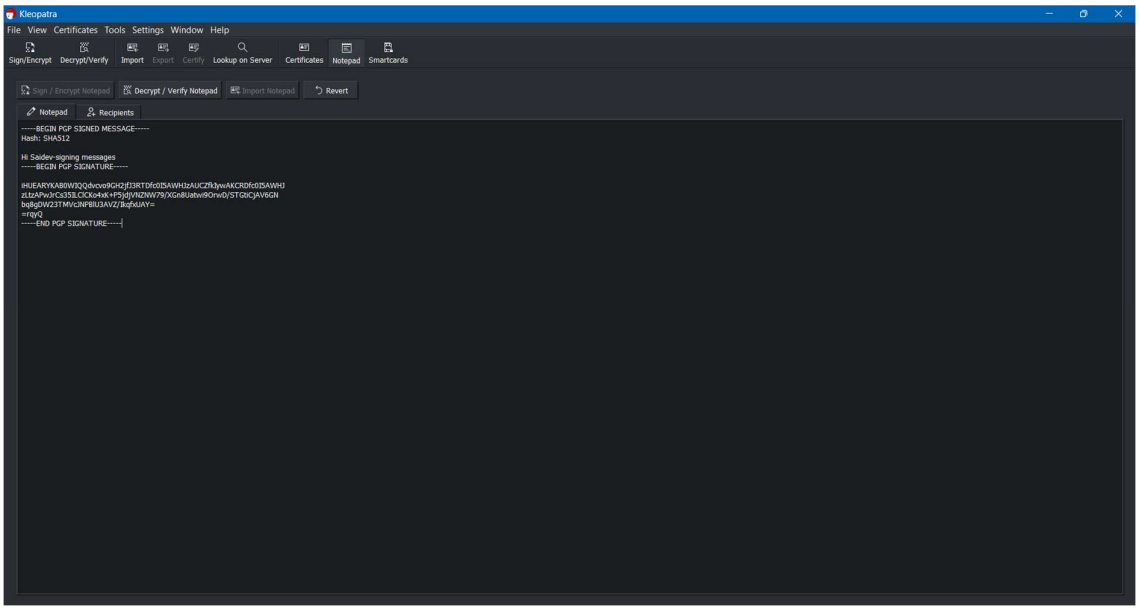


Now we have to share our public key to receiver so, that he can accesss the information that we are going to share

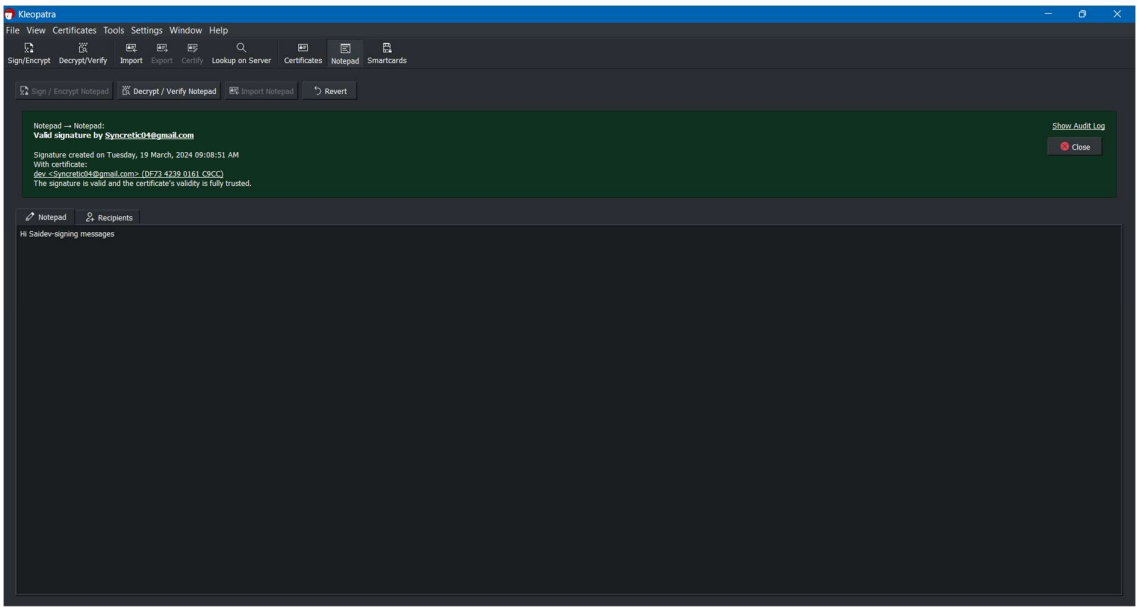
Now in another system, certify the user id that we have shared.



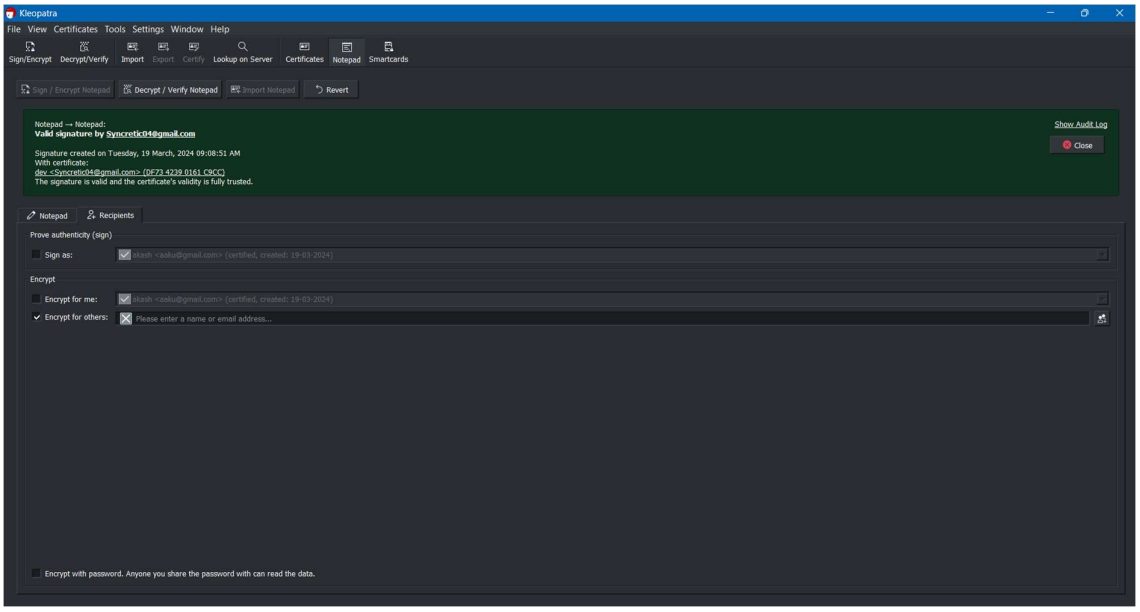
Now decrypt/verify Message containing with signature



Decrypted Message



Recipients



In conclusion, the process we've followed involves setting up and using Kleopatra, a certificate manager and GUI for GnuPG, to ensure secure message exchange. We've successfully signed messages, shared our public key, certified the user ID on the receiver's system, and decrypted/verified the message. This process has enhanced the security of our communication by ensuring the authenticity and integrity of the messages exchanged.

Experiment 7

Network Intrusion Detection using Snort

Objective: To set up and configure Snort, a network intrusion detection system, on a Windows operating system for the purpose of monitoring network traffic and detecting potential threats. The process involves the following steps:

1. **Initialization of Snort**: Start Snort and ensure it's properly installed and configured on your system.
2. **Selection of Network Adapter**: Identify and select the appropriate network adapter that Snort will monitor. This is done by reviewing the network adapter information displayed on the command-line interface.
3. **Setting up Local Rules**: Define local rules in Snort for detecting specific types of network traffic, such as different types of ping requests. These rules instruct Snort on what kind of traffic to look for and what actions to take when such traffic is detected.
4. **Monitoring Network Traffic**: With Snort set up and the rules defined, commence the monitoring of network traffic. Snort will analyze the network traffic, generate logs when it detects traffic that matches the rules, and alert you of any potential threats.

Introduction:

Intrusion Detection Systems (IDS) are crucial for maintaining network security in today's digital world. One such system is Snort, a free and open-source network intrusion detection system (NIDS) that is capable of performing real-time traffic analysis and packet logging on Internet Protocol (IP) networks.

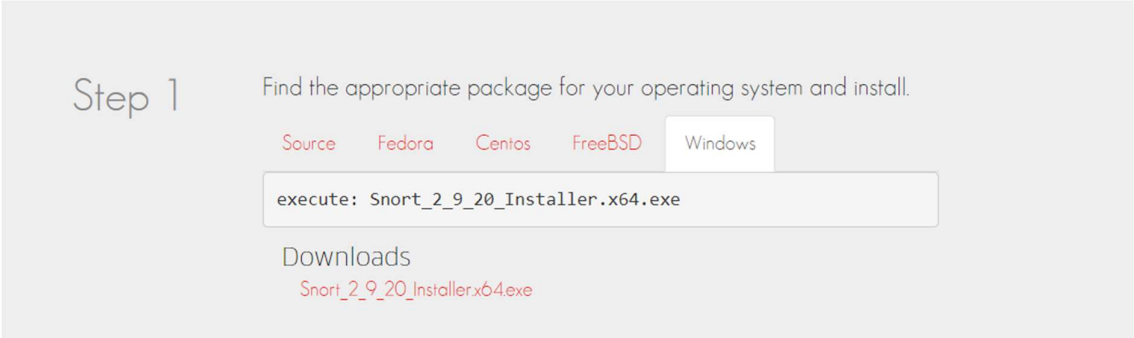
Setting up and configuring Snort on a Windows operating system. This includes initializing Snort, selecting the appropriate network adapter for monitoring, defining local rules for detecting specific types of network traffic, and finally, commencing the monitoring of network traffic.

Snort analyzes the network traffic and generates logs when it detects traffic that matches the rules we've set up. These logs provide valuable insights into your network activity and can alert us to potential threats in real time.

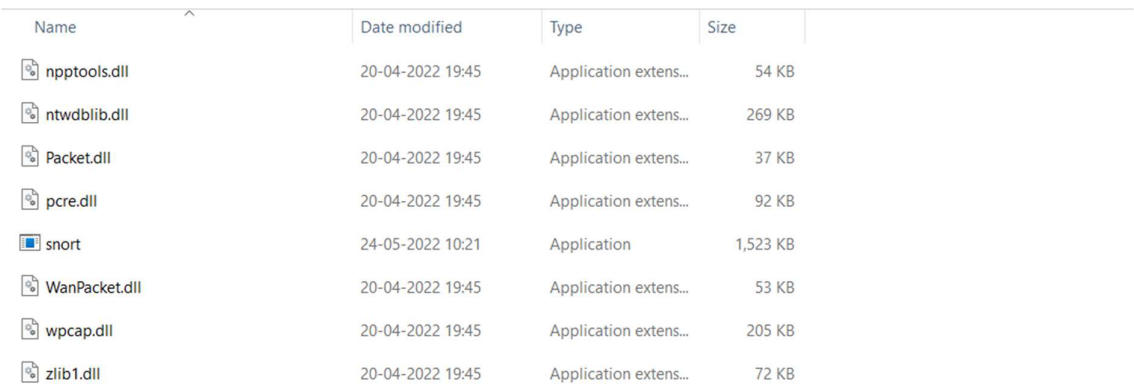
By successfully setting up and configuring Snort, we're taking a significant step towards enhancing the security of your network. This hands-on experience with Snort will also provide us with a deeper understanding of network security concepts and best practices.

Setup:

1. **Download and Install Snort**: Download Snort from the official website. Choose the version that is compatible with your Windows operating system. After downloading, run the installer and follow the prompts to install Snort.



2. **Snort Application**: This is the executable file for Snort, a network intrusion detection system. When we run this application, it starts Snort and begins monitoring network traffic according to the configuration and rules you've set up.



Name	Date modified	Type	Size
npptools.dll	20-04-2022 19:45	Application extens...	54 KB
ntwdblib.dll	20-04-2022 19:45	Application extens...	269 KB
Packet.dll	20-04-2022 19:45	Application extens...	37 KB
pcre.dll	20-04-2022 19:45	Application extens...	92 KB
snort	24-05-2022 10:21	Application	1,523 KB
WanPacket.dll	20-04-2022 19:45	Application extens...	53 KB
wpcap.dll	20-04-2022 19:45	Application extens...	205 KB
zlib1.dll	20-04-2022 19:45	Application extens...	72 KB

3. **Initialization and commencement of packet processing in Snort, a network intrusion detection system**

```
C:\Snort\bin>snort
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to passive.
The DAQ version does not support reload.
Acquiring network traffic from "\Device\NPF_{2D671199-1D92-4299-A1C4-81D4E3078515}".
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.20-WIN64 GRE (Build 82)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=6260)
```

During **initialization**, Snort prepares to monitor network traffic according to the configuration and rules we've set up.

After initialization, Snort begins processing packets. This means it starts **analyzing the network traffic** on our system, looking for any activity that matches the rules we've set up.

4. **List of network adapters available on our system**

After Snort initialization, we would typically select a network adapter for Snort to monitor. The image shows the list of network adapters available on our system. Each network adapter is associated with a specific network connection, and it has its own unique properties and settings

```
C:\Snort\bin>Snort -W

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.20-WIN64 GRE (Build 82)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.11

Index   Physical Address        IP Address      Device Name     Description
-----   ----------------        ----------      -----------     -----------
    1   00:00:00:00:00:00       disabled        \Device\NPF_{2D671199-1D92-4299-A1C4-81D4E3078515}      WAN Miniport (Network Monitor)
    2   00:00:00:00:00:00       disabled        \Device\NPF_{FE3CAE7E-5C62-4B8E-9982-37851F0D12ED}      WAN Miniport (IPv6)
    3   00:00:00:00:00:00       disabled        \Device\NPF_{82D7A781-1801-4E5D-818D-5CB6300333DC}      WAN Miniport (IP)
    4   00:15:5D:1D:C0:8D       172.21.208.1    \Device\NPF_{B9C2C904-E82D-45FF-ADEC-1F64A7369A1D}      Hyper-V Virtual Ethernet Adapter
    5   B4:8C:9D:08:27:FC       169.254.166.143 \Device\NPF_{542B0228-5BD9-4272-AD6D-D24221C7F4E3}      Bluetooth Device (Personal Area Network)
    6   B4:8C:9D:08:27:FD       192.168.26.9    \Device\NPF_{96E053D2-57EC-4C87-B83F-22E0AF84058D}      Realtek RTL8822CE 802.11ac PCIe Adapter
    7   00:50:56:C0:00:08       192.168.177.1   \Device\NPF_{B0DE984E-42EF-449F-8DE7-FE44C334597D}      VMware Virtual Ethernet Adapter for VMnet8
    8   00:50:56:C0:00:01       192.168.224.1   \Device\NPF_{E5820C30-5FC1-42ED-9A5C-6D8F11105A96}      VMware Virtual Ethernet Adapter for VMnet1
    9   F6:8C:9D:08:27:FD       169.254.107.218 \Device\NPF_{FFD1A830-FF8B-4656-819B-2BCED80A9A38}      Microsoft Wi-Fi Direct Virtual Adapter #4
   10   B6:8C:9D:08:27:FD       169.254.209.107 \Device\NPF_{E5585669-4F75-4C8E-B5C7-B48637069E96}      Microsoft Wi-Fi Direct Virtual Adapter #3
   11   00:00:00:00:00:00       0000:0000:0000:0000:0000:0000:0000:0000 \Device\NPF_Loopback    Adapter for loopback traffic capture
   12   C0:18:03:3E:F4:B2       169.254.173.199 \Device\NPF_{D94391AA-A21D-416D-80A8-6FBFAB2AF5A0}      Realtek PCIe GbE Family Controller
```

The output includes technical data such as:

- **Adapter Name**: This is the name of the network adapter. It's used by the operating system to identify the adapter.
- **Adapter Description**: This provides more details about the adapter, such as the manufacturer and model.
- **Adapter Address**: This is the hardware or MAC address of the adapter. It's a unique identifier assigned to the network adapter by the manufacturer.
- **IP Address and Mask**: These are the IP address and subnet mask associated with the adapter. They define the network and host parts of the IP address.

We would typically choose the network adapter that's associated with the network we want to monitor. Once we've selected an adapter, Snort can start monitoring the network traffic that goes through that adapter, according to the rules we've set up.

5. **Local Rules in network security:**

```
#-------
# LOCAL RULES
#-------

alert icmp any any -> 192.168.26.9 any (msg:"ICMP Ping Request"; sid:10000001;)
alert udp  any any -> 192.168.26.9 any (msg:"udp Ping Request"; sid:10000002;)
alert tcp  any any -> 192.168.26.9  any (msg:"tcp Ping Request"; sid:10000003;)
```

These rules are used by Snort, a network intrusion detection system, to monitor network traffic and detect suspicious activity.

Here's a breakdown of what the rules are doing:

This rule is looking for ICMP echo requests, also known as "pings".
alert icmp any any -> $HOME_NET any (msg:"ICMP PING"; itype: 8; icode: 0; sid:477; rev:3;)

This rule is looking for ICMP echo replies, which are the responses to pings.
alert icmp $HOME_NET any -> any any (msg:"ICMP PING reply"; itype: 0; icode: 0; sid:480; rev:3;)

Each rule starts with the action alert, which means Snort will generate an alert when it detects traffic that matches the rule. The icmp specifies the protocol. The any any -> $HOME_NET any part specifies the direction of the traffic. In this case, it's looking for pings from any IP address (any any) to any IP address within your home network ($HOME_NET any).

The msg part is the message that will be included in the alert. The itype and icode are the ICMP type and code, respectively. For echo requests (pings), the type is 8 and the code is 0. For echo replies (responses to pings), the type is 0 and the code is 0.

The sid is the Snort ID for the rule, and rev is the revision number of the rule. These are used to uniquely identify each rule.

These rules are important for detecting different types of ping requests, which can be a sign of a network scan or other suspicious activity. By adding these rules to our Snort configuration, we can enhance the security of our network.

6. **Final Output:**

The below screenshot of the output shows a series of log entries from Snort the network monitoring tool 've been using. These entries are showing UDP and TCP ping requests with timestamps, priorities, and IP addresses.

```
04/16-10:37:28.946169  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 34.230.200.85:443 -> 192.168.26.9:55541
04/16-10:37:28.946169  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 34.230.200.85:443 -> 192.168.26.9:55541
04/16-10:37:28.946169  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 34.230.200.85:443 -> 192.168.26.9:55541
04/16-10:37:29.356592  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 34.230.200.85:443 -> 192.168.26.9:55541
04/16-10:37:29.356592  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 34.230.200.85:443 -> 192.168.26.9:55541
04/16-10:37:29.356592  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 34.230.200.85:443 -> 192.168.26.9:55541
04/16-10:37:33.247424  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 52.2.241.103:443 -> 192.168.26.9:55486
04/16-10:37:33.247771  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 52.2.241.103:443 -> 192.168.26.9:55486
04/16-10:37:33.305375  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 108.158.251.110:443 -> 192.168.26.9:55535
04/16-10:37:33.656483  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 52.2.241.103:443 -> 192.168.26.9:55486
04/16-10:37:38.674034  [**] [1:10000003:0] tcp Ping Request [**] [Priority: 0] {TCP} 20.212.88.117:443 -> 192.168.26.9:54462
```

- **Timestamp**: This is the date and time when the log entry was created. It can help you determine when a particular network event occurred.
- **Priority**: This is the priority level of the log entry. In Snort, priority levels can be used to indicate the severity or importance of a detected event.
- **IP Addresses**: These are the source and destination IP addresses for the network traffic that triggered the log entry. The source IP is the address of the system that sent the network traffic, and the destination IP is the address of the system that received the traffic.
- **UDP and TCP**: These are the protocols used for the network traffic. UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) are two of the main protocols used in network communication.

The log entries are showing that Snort is successfully monitoring our network traffic and generating logs when it detects the specific types of traffic (in this case, UDP and TCP ping requests) that we set up rules for.