Report on Research and Learnings

In-Band SQL Injection

This is the most common type, where the attacker uses the same communication channel to both launch the attack and retrieve data.

- **Union-Based SQL Injection:**

Working: Uses the UNION SQL operator to combine the results of a malicious query with the results of a legitimate query.

Example: The payload

```
' UNION SELECT null, version() --
```

appends a query to fetch the database version.

- **Error-Based SQL Injection:**

Working: Leverages error messages thrown by the database to extract information.

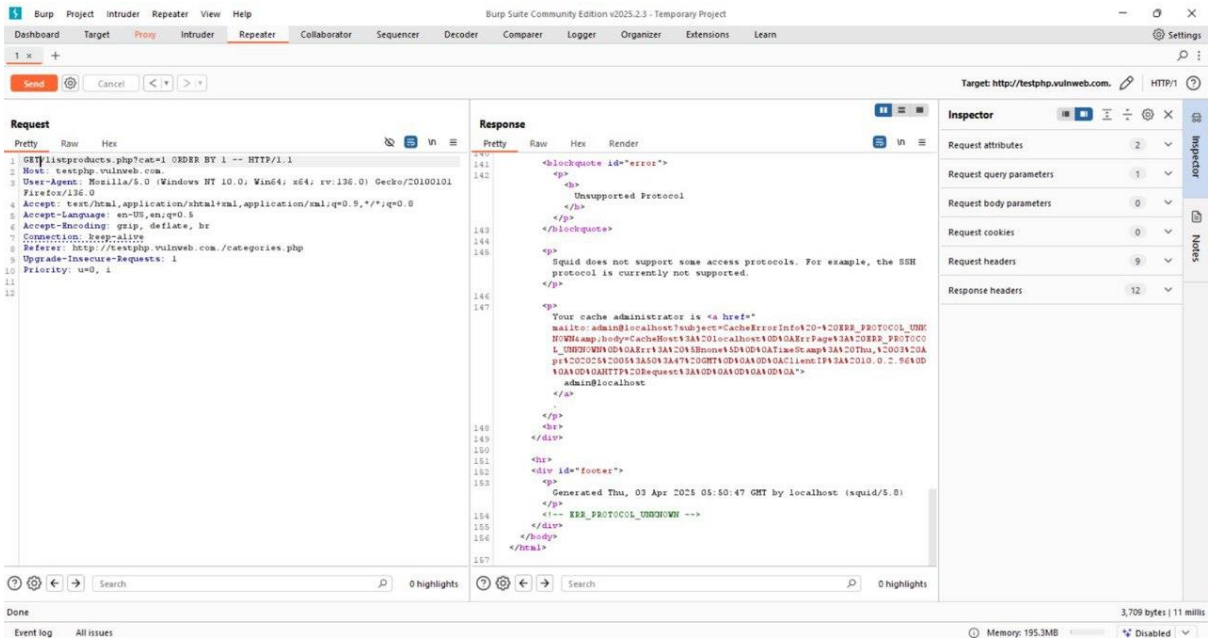
Usage: An attacker deliberately causes an error (for example, by feeding incorrect syntax) to see detailed error output that might reveal database details.

Description:

Error-Based SQL Injection is a technique where an attacker manipulates an SQL query to force the database to generate an error message. These error messages often reveal crucial details about the database structure, such as table names, column names, and database versions.

On **testphp.vulnweb.com**, it was found that injecting ORDER BY clauses with incorrect column numbers can generate database errors, exposing the internal database structure.

1] After inserting payload



Inferential (Blind) SQL Injection

In this type, the attacker doesn't get direct output from the database. Instead, they infer data based on how the application responds to different inputs.

- **Boolean-Based Blind SQL Injection:**

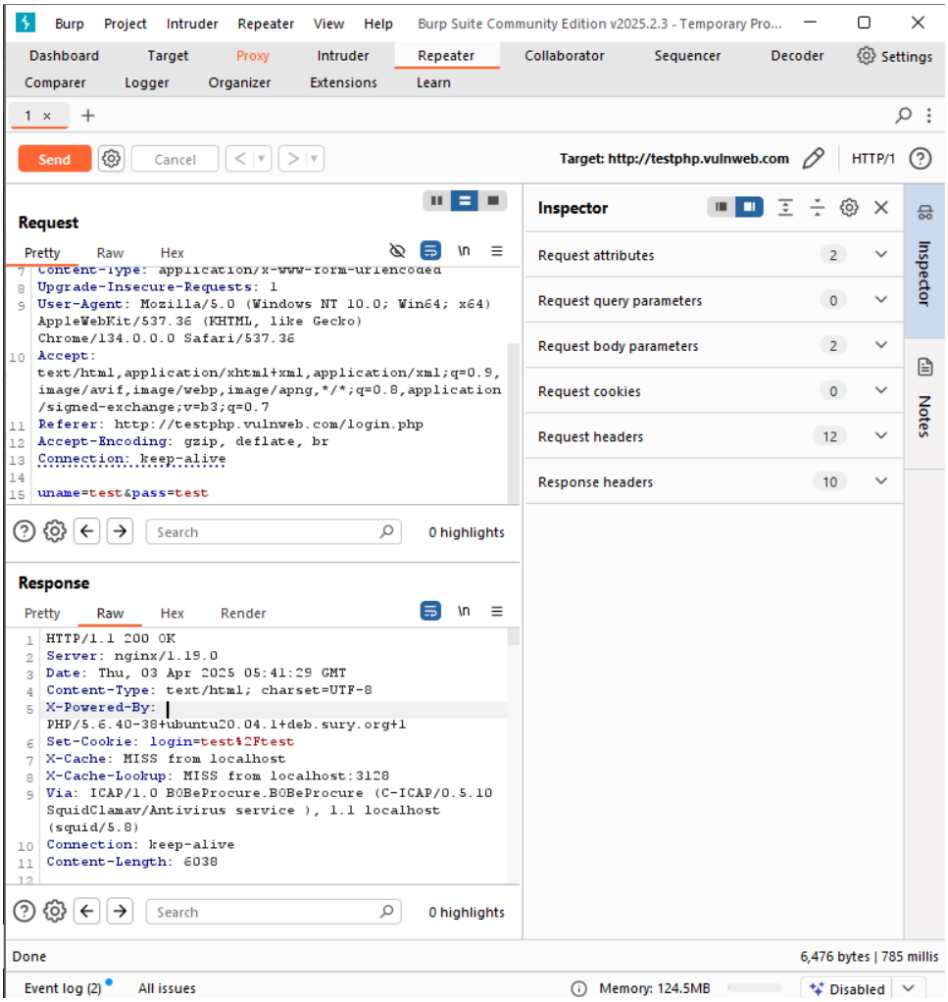
- **Working :** The attacker injects SQL code that results in a true/false condition, and then observes changes in the page (like content or behavior) to infer if the condition is true or false.
- **Example:**
  `' OR '1'='1`
  might be used to bypass a login by forcing the condition to always be true.
- If the application's response is not visibly different, but we deduce the database state by observing subtle changes, it's considered blind injection.

Description:

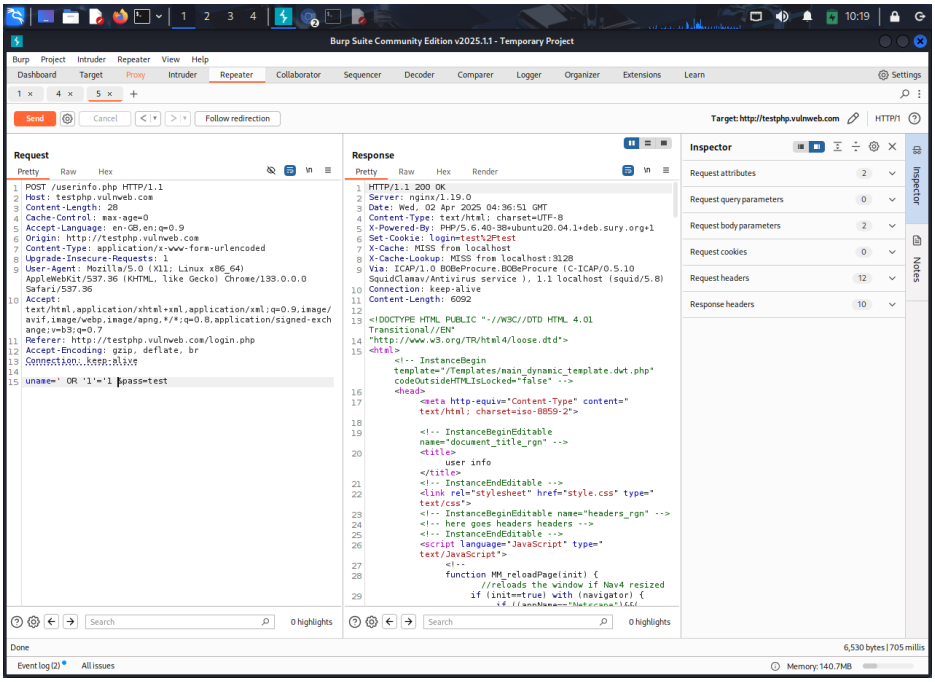
SQL Injection (SQLi) is a critical web application vulnerability that occurs when an application fails to properly sanitize user input before including it in SQL queries. **Boolean-Based SQL Injection** occurs when an attacker manipulates SQL queries using conditional logic (e.g., `OR '1'='1'`) to extract sensitive information or bypass authentication.

On **testphp.vulnweb.com**, it was observed that the application is vulnerable to **Boolean-Based SQL Injection** in the **search functionality**, allowing attackers to manipulate database queries by injecting Boolean conditions.

1] Inserting real data



2]Inserting Boolean based payload



- **Time-Based Blind SQL Injection:**

  - **Working:** The attacker uses SQL commands that force the database to wait (using functions like `sleep()`) if a condition is true. By measuring the delay, the attacker can infer information.
  - **Example:**
    `' OR IF(1=1, sleep(5), 0)--`
    would delay the response by 5 seconds if the condition is true.

Out-of-Band SQL Injection

This type is used when in-band techniques are not effective or feasible. Data is retrieved using a different channel, such as a DNS or HTTP request.

- **Working:**

  - The injected SQL query causes the database to send data to an attacker-controlled server (e.g., via DNS requests).
  - **Usage:** Often a fallback method when error messages or content-based responses are disabled.

Cross-Site Scripting (XSS) Using Burp Suite

Cross-Site Scripting (XSS) is a web vulnerability that allows attackers to inject **malicious JavaScript** into a website, which is then executed in another user's browser.

Identify a Reflected or Stored XSS Vulnerability

1. Open Burp Suite and start the **Intercept Mode** under the **Proxy** tab.
2. Navigate to a web page that has a **search box, comment box, or login form** (anything that accepts user input).
3. Enter a basic XSS payload like:
4. `<script>alert('XSS')</script>`
5. If the page **returns input in the response**, it may be vulnerable to XSS.

Capture & Modify the Request in Burp Suite

1. **Enable Intercept** in Burp Suite Proxy.
2. Enter an XSS payload in the vulnerable input field (e.g., a comment box).
3. Click **Submit** and let Burp Suite capture the request.
4. Send the captured request to **Burp Repeater** (Right-click → **Send to Repeater**).
5. Modify the request by injecting a more advanced payload:
6. `<img src=x onerror=alert('XSS')>`
7. Click **Send** and check the Response tab.
   - If the payload **executes in the browser**, the site is vulnerable.
   - If it's blocked, try bypassing it using **URL encoding** or **event handlers** (`onmouseover`, `onerror`).

Types of Intruder Attacks in Burp Suite

Burp Suite **Intruder** is a powerful tool used for automating customized attacks on web applications. It allows penetration testers to send multiple requests with different inputs to identify vulnerabilities such as **authentication weaknesses, SQL injection, cross-site scripting (XSS), and other injection-based attacks**.

1.Sniper Attack (Single Parameter Testing)

- **Description:**
  - The **Sniper** attack method iterates through a **single** payload list and injects values into **one** parameter at a time.
- **Use Case:**
  - Used for **input validation testing**, including **XSS, SQL Injection, and command injection**.
  - Ideal for testing **each field individually** to identify vulnerabilities.

3.Pitchfork (Parallel Payload Injection in Multiple Positions)

- **Description:**
  - Uses **two or more payload lists** and injects values **side by side** into different positions.
  - Each request contains values from the same row of different payload lists.
- **Use Case:**
  - **Brute-force username-password combinations** during login testing.
  - Useful for testing **SQL Injection** across multiple parameters.

4.Cluster Bomb (All Possible Payload Combinations)

- **Description:**
  - Uses **two or more payload lists** and tests **every possible combination**.
  - More thorough than **Pitchfork**, but also significantly slower.
- **Use Case:**
  - Used for **brute-forcing login credentials** when username-password pairs are unknown.
  - Useful for multi-field injection attacks where different parameters interact.

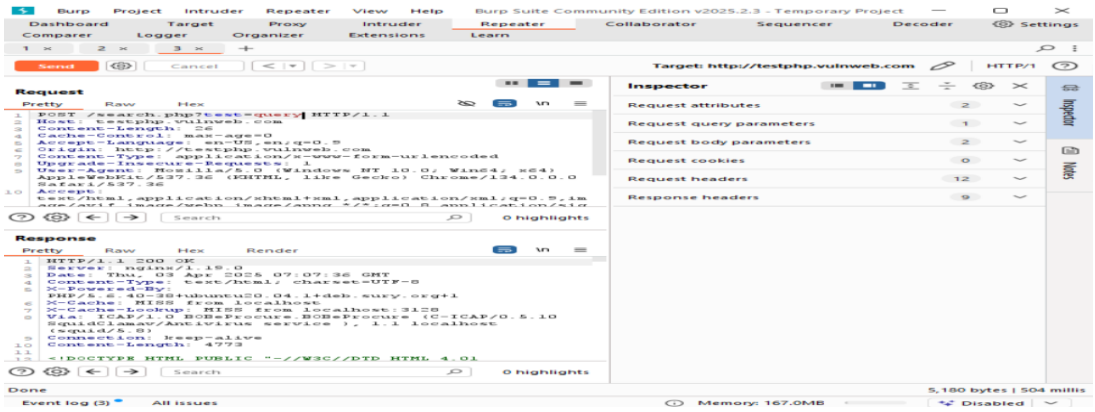
Steps to Reproduce (Proof of Concept - PoC):

1.**Navigate to a search input field** on the target website(testphp.vulnweb.com).
2. **Enter and submit the following payload:**

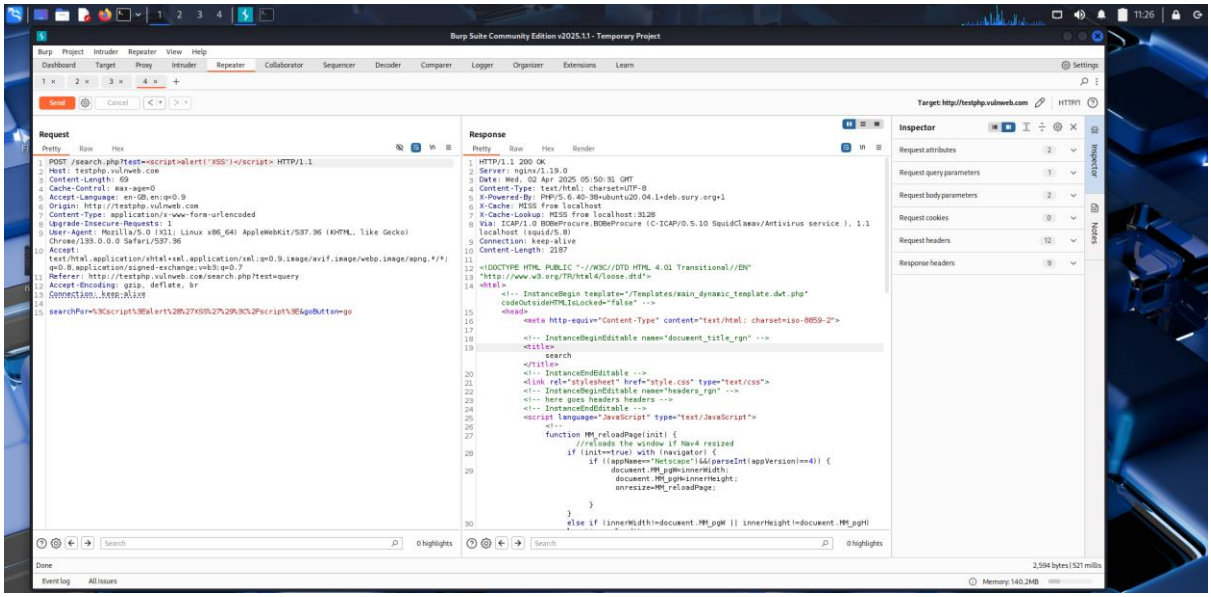
```
<script>alert('XSS')</script>
```

3. **Check if the script gets stored and executed** when the page reloads.
4. **If another user visits the affected page**, their browser will execute the script, displaying the alert box.

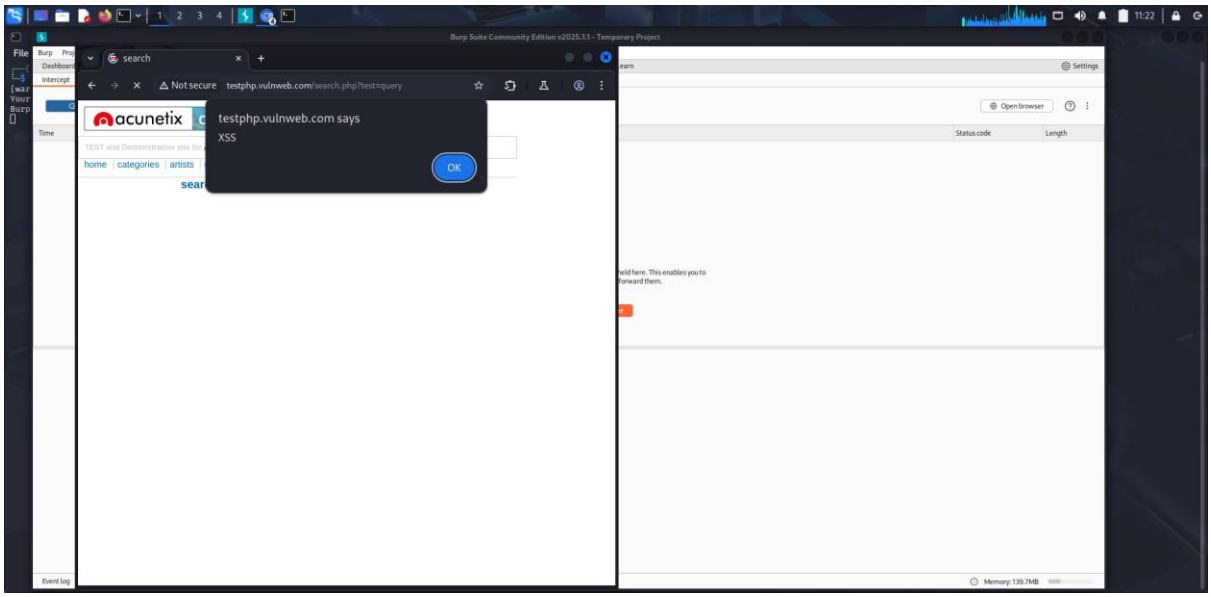
1] Before adding payload



2] Inserting payload



3]Results



Broken Authentication Using Burp Suite Intruder

A web application is vulnerable to **Broken Authentication** if an attacker can:

- Brute-force user logins using common credentials.
- Hijack valid sessions using stolen **Session IDs**.
- Bypass authentication mechanisms (weak password policies, no rate limiting).

Attack 1: Brute Force Login Using Burp Suite Intruder

Capture the Login Request

1. Open Burp Suite and **turn on Intercept** under the **Proxy** tab.
2. Go to the website's login page and enter **any random username/password**.
3. Click **Login** and **Burp Suite** will capture the request.

Send the Request to Burp Intruder

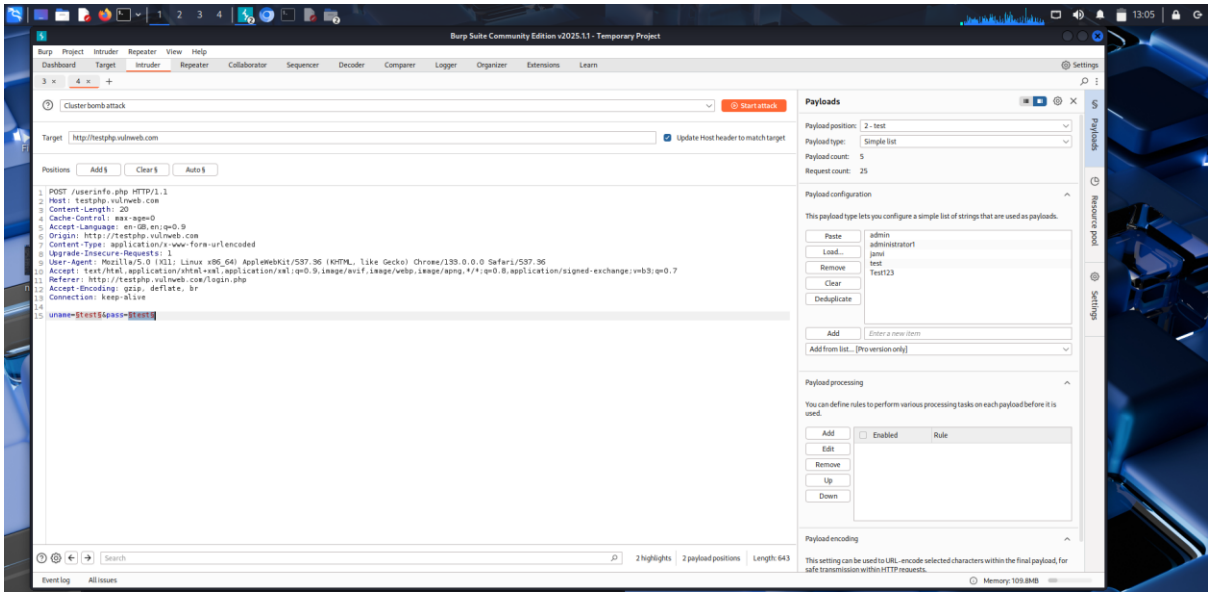
1. **Right-click** on the captured request → Click **Send to Intruder**.
2. Go to the **Intruder** tab → Select **Positions**.
3. Click **Clear** to remove all payload positions.
4. Highlight the **username** and click **Add** (set it as a payload position).
5. Highlight the **password** and click **Add** (set it as another payload position).

Load a Wordlist for Brute-Force Attack

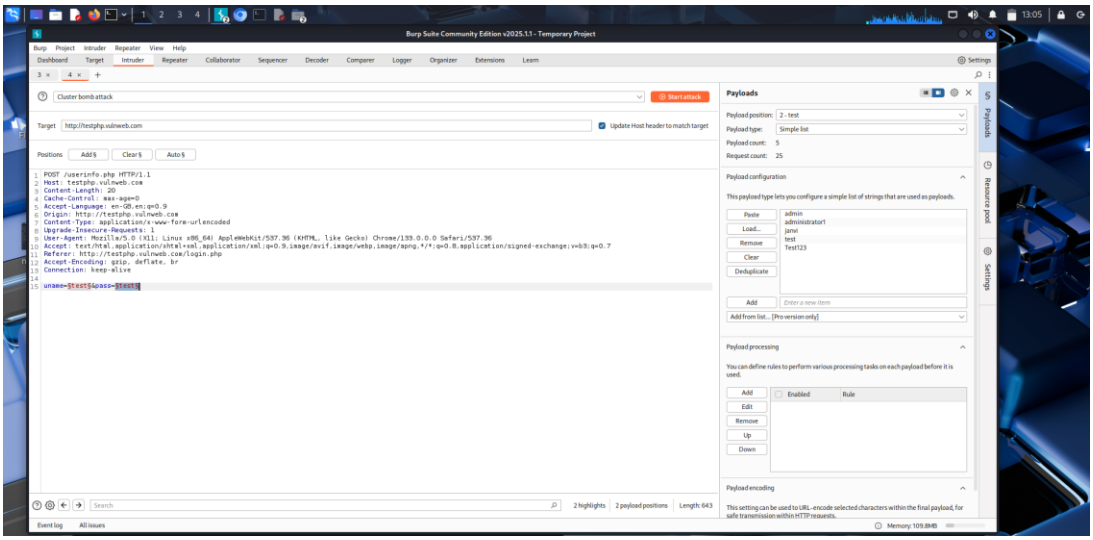
1. Go to the **Payloads** tab.
2. Under **Payload Set 1**, click **Load** and select a wordlist of common usernames (`admin`, `root`, `test`, etc.).
3. Under **Payload Set 2**, load a password list (e.g., `password`, `123456`, `admin123`).

Start the Attack

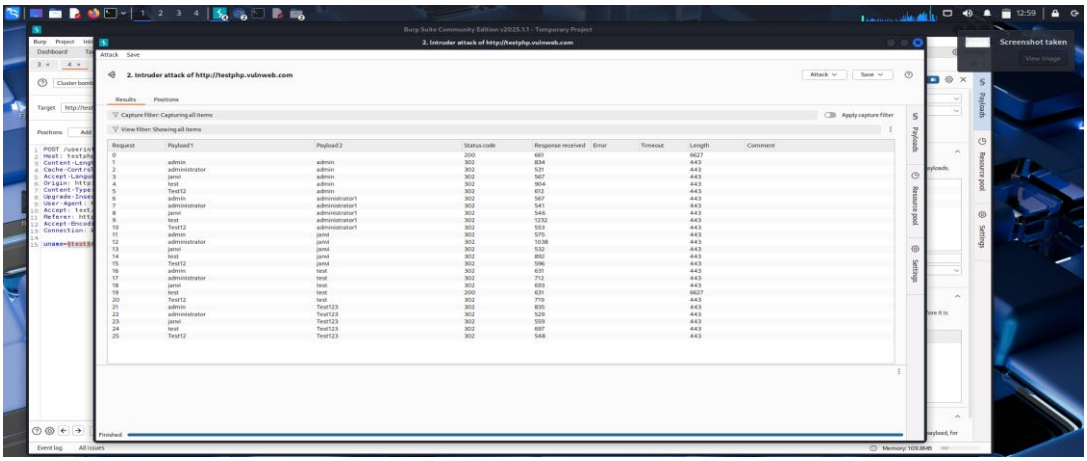
1. Click **Start Attack** in Burp Suite.
2. Watch for **successful responses** (HTTP 200 or a different response length).
3. If you get a different response for a username-password combination, **the credentials are valid.**
4. 1]Adding payload Markers



2]Adding payload 2



3]Results with status Code



Cross-Site Request Forgery (CSRF) in Search Functionality

Description:

Cross-Site Request Forgery (CSRF) is a **web security vulnerability** that allows an attacker to trick an authenticated user into unknowingly executing unwanted actions on a web application.

In this case, **the search functionality** of `testphp.vulnweb.com` is vulnerable to CSRF.

- Since the **search request is a POST request**, an attacker can craft a **malicious HTML form** to **silently submit a search request on behalf of a logged-in user**.
- If the user is authenticated and clicks on a **malicious link**, the search request gets executed **without their consent**.

Step 1: Identify the Search Request

1.Perform a **search query** on `testphp.vulnweb.com`.
2.Use **Burp Suite** to inspect the **POST request**.

Step 2: Create a CSRF Exploit Page

An attacker can create an HTML file (`csrf_exploit.html`) containing the **malicious CSRF form**:

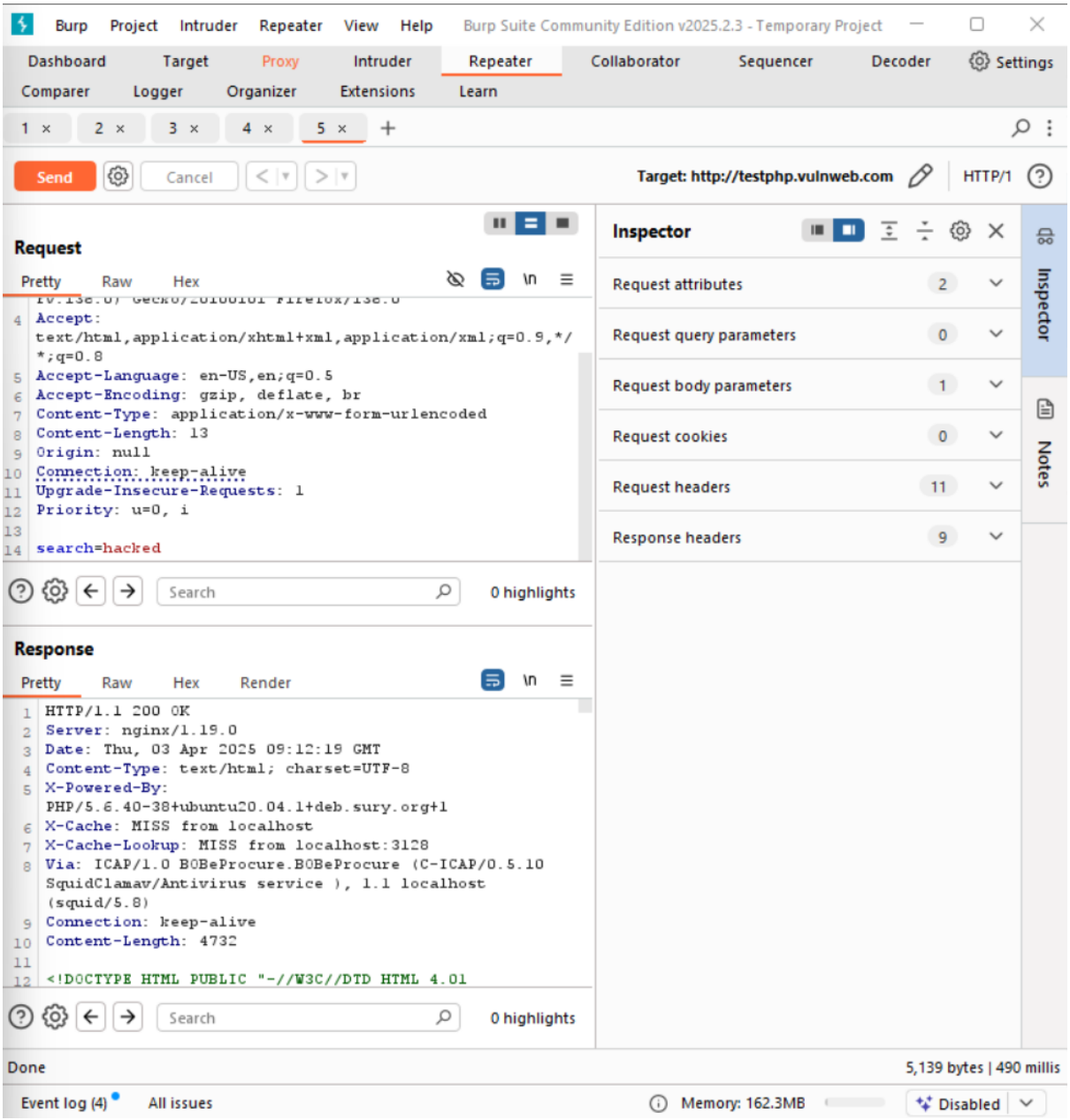
```
<!DOCTYPE html>
<html>
  <body onload="document.getElementById('csrfForm').submit()">
    <form id="csrfForm" action="http://testphp.vulnweb.com/search.php" method="POST">
      <input type="hidden" name="search" value="hacked">
    </form>
  </body>
</html>
```

Step 3: Trick the Victim

- Share the link with a logged-in victim through:
  a]Email (phishing attack)
  b]Social media posts
  c]Embedded in an **image** or **button**

Step 4: Observe the Request Execution

- If the victim clicks the link **while logged into** `testphp.vulnweb.com`, their browser automatically submits the **search request** without their knowledge.
- This **proves CSRF vulnerability** in the search functionality.



Version Disclosure Vulnerability

Description: Exposed software version details help attackers identify exploitable vulnerabilities.

Steps to Reproduce:

1. Inspect HTTP response headers or error pages.
2. Look for version details of PHP, MySQL, or other technologies.

Findings & Observations

- **High-Risk Vulnerabilities:** SQL Injection, Stored XSS, CSRF, Broken Authentication.
- **Medium-Risk Vulnerabilities:** Event-Based XSS, Version Disclosure.
- **Low-Risk Vulnerabilities:** Minor security misconfigurations.
- **Security Controls Missing:** Lack of proper input validation, no CSRF tokens, weak authentication mechanisms.

Conclusion & Learnings

- This research provided hands-on experience in identifying and exploiting web vulnerabilities.
- It highlighted the importance of secure coding practices.
- Future research could focus on automating vulnerability detection using advanced tools.

Appendix

- **Payloads Used:** SQL Injection queries, XSS scripts, CSRF exploit forms.
- **References:**
  - OWASP Top 10 (https://owasp.org/www-project-top-ten/)
  - SQL Injection Prevention (https://www.owasp.org/index.php/SQL_Injection)
  - XSS Protection Guidelines (https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))