الأكاديمية العربية للعلوم والتكنولوجيا والنقل البحري
Arab Academy for Science, Technology & Maritime Transport

DIFR Project Documentation

Digital Forensics & Incident Response

Dr. Mostafa Ammar

Eng: Haya Yasser

Team:

- Mawada Ayman ~ 221003907
- Nada Ibrahim ~ 221003011
- Ahmed Walid Ibrahim ~ 221011183
- Ahmed Mohamed Mahmoud ~ 221010720
- Omar Sherief ~ 221010339

May 2024

Table of Contents

Introduction

This document outlines the setup, execution, and analysis of a web security simulation project. It covers the website and infrastructure configuration, offensive security testing, and log analysis performed as per the project requirements. The primary goal was to establish a vulnerable web application, attempt to exploit its weaknesses, and analyze the resultant logs to understand attack vectors and defense mechanisms.

Team Structure:

Offensive Security Specialist:

- Omar Shrief

System Administrator:

- Ahmed Mohamed

Log Analysts/Defenders:

- Nada Ibrahim
- Mawada Ayman
- Ahmed Walid

1. Website & Infrastructure

This section details the configuration of the virtualized web server environment and the Splunk instance used for log analysis.

1.1. Web Server VM Configuration

The web server was hosted on a Kali Linux virtual machine running XAMPP as backend.

Configuration Details:

- **Operating System:** Kali Linux (VM)
- **Web Server Software:** Apache (via XAMPP)
  - XAMPP Version: 8.2.12-0 (as per web_server.txt)
  - Apache Version: 2.4.58 (Unix)
  - The server was managed using commands like sudo /opt/lampp/lampp start, stop, and restart.
- **Server-Side Scripting:** PHP 8.2.12
- **Database:** MySQL (MariaDB fork, version 5.1 or similar, as indicated by SQLMap output and php_error_log.txt)
  - Managed via phpMyAdmin.
  - phpMyAdmin Access: http://localhost/phpmyadmin
  - phpMyAdmin Credentials: myadminuser / PHPadminAhmed221010720$

- **Document Root:** /opt/lampp/htdocs/
- **Application Directory:** /opt/lampp/htdocs/dfir/
- **Apache User/Group:** daemon / daemon
- **Listening Port:** HTTP on port 80, HTTPS on port 443.

Core Apache Configuration (httpd.conf highlights):

- ServerRoot "/opt/lampp"

- Listen 80
- User daemon, Group daemon
- DocumentRoot "/opt/lampp/htdocs"
- For the /opt/lampp/htdocs directory:
  - AllowOverride All (enabling the use of .htaccess files)
  - Require all granted (allowing access)
- Directory Index: index.html index.html.var index.php index.php3 index.php4
- Error Log: logs/error_log
- Access Log: CustomLog "logs/access_log" common
- SSL/TLS was configured via etc/extra/httpd-ssl.conf.
  - Certificate: mysite.crt
  - Key: mysite.key
  - Both stored in /opt/lampp/ssl/ (as per web_server.txt and SSL configuration snippets). The certificate and key files (mysite.crt.txt, mysite.key.txt) were provided.

Application Files:

The web application consisted of the following core files located in /opt/lampp/htdocs/dfir/:

- index.php: Main landing page.
- login.php: User login functionality.
- contact.php: Contact form and message display.
- flag.php: Flag checking mechanism.
- style.css: CSS for styling the web pages.
- Hello.jpg: Image displayed on the homepage.

1.2. Splunk VM Setup and Dashboard

A Splunk instance was configured for log collection and analysis. The setup involved a Splunk server and a Universal Forwarder on the web server VM.

Splunk Configuration:

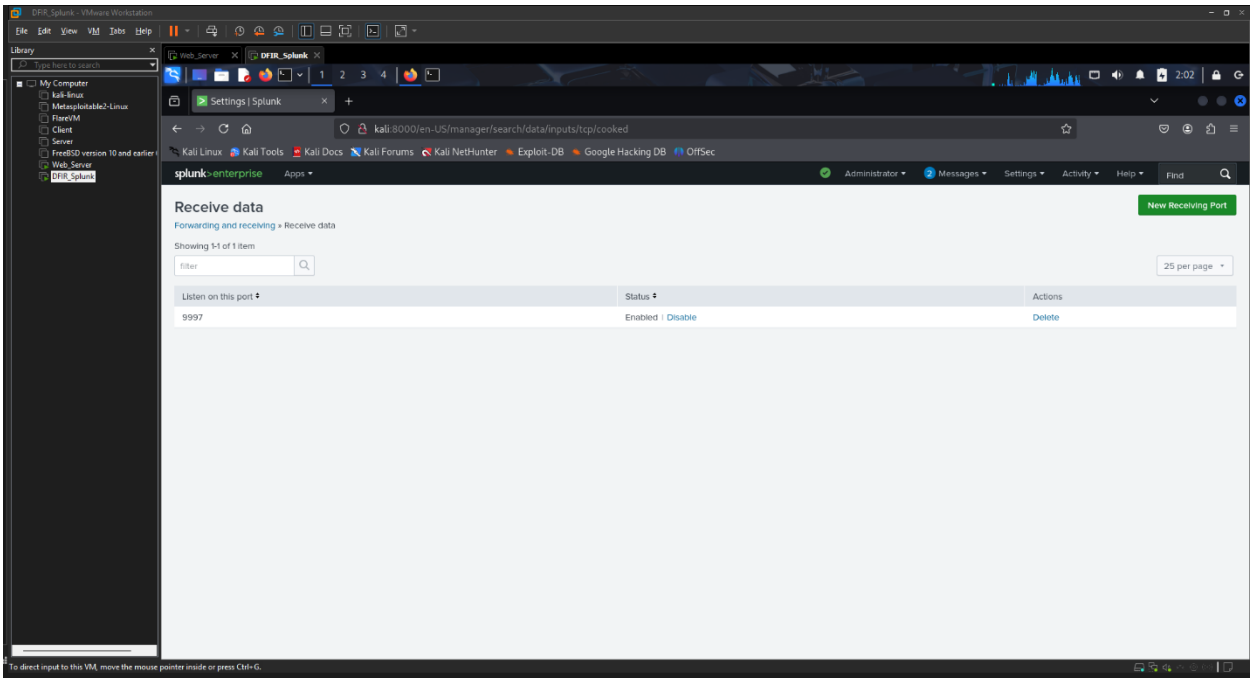
- **Splunk Server:** Accessed via http://kali:8000
- **Splunk Credentials:** admin / aastadmin.
- **Splunk Forwarder:** Installed on the web server VM.

- Forwarding was configured to the Splunk server at 192.168.179.153:9997 using the same admin credentials.

- **Data Inputs:**
  - All logs were indexed into web_logs.
  - Monitored files and their sourcetypes:
    - /opt/lampp/logs/php_requests.log
    - /opt/lampp/logs/access_log
    - /opt/lampp/logs/error_log
    - /opt/lampp/logs/php_error_log
    - /opt/lampp/logs/ssl_request_log
    - /opt/lampp/var/mysql/kali.err

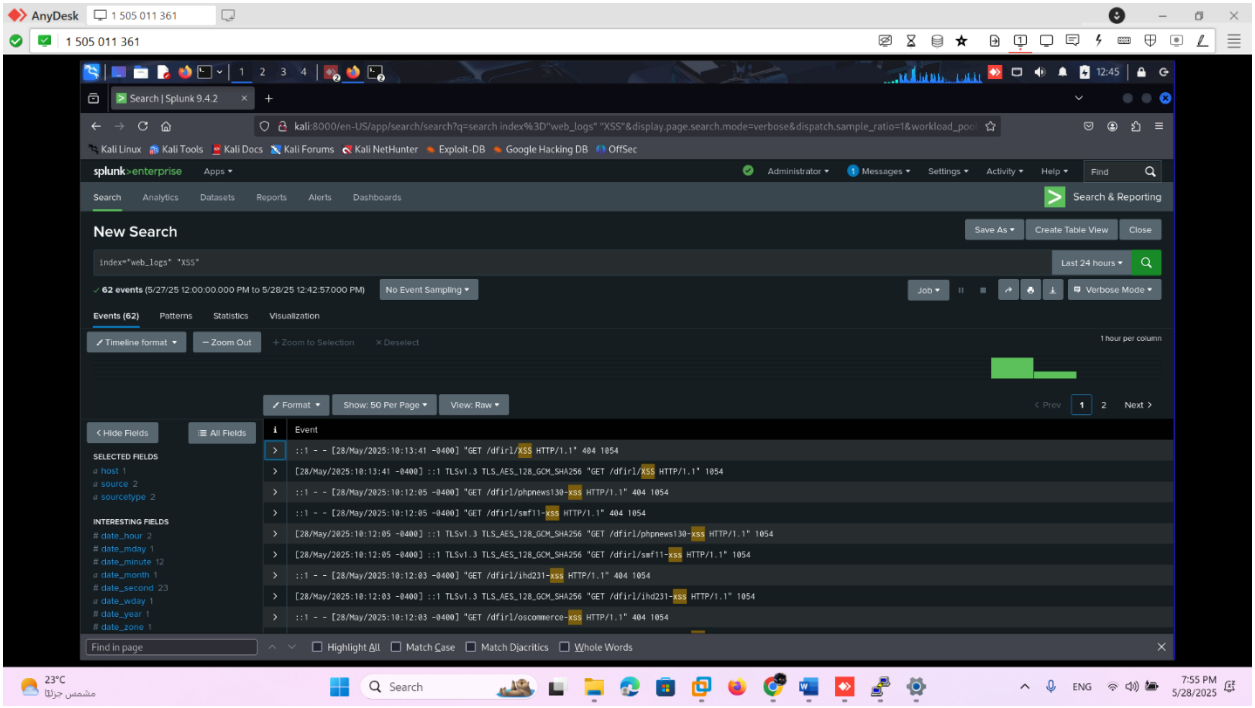
- **Splunk Receiving Port:** Port 9997 was enabled on the Splunk server to receive forwarded data

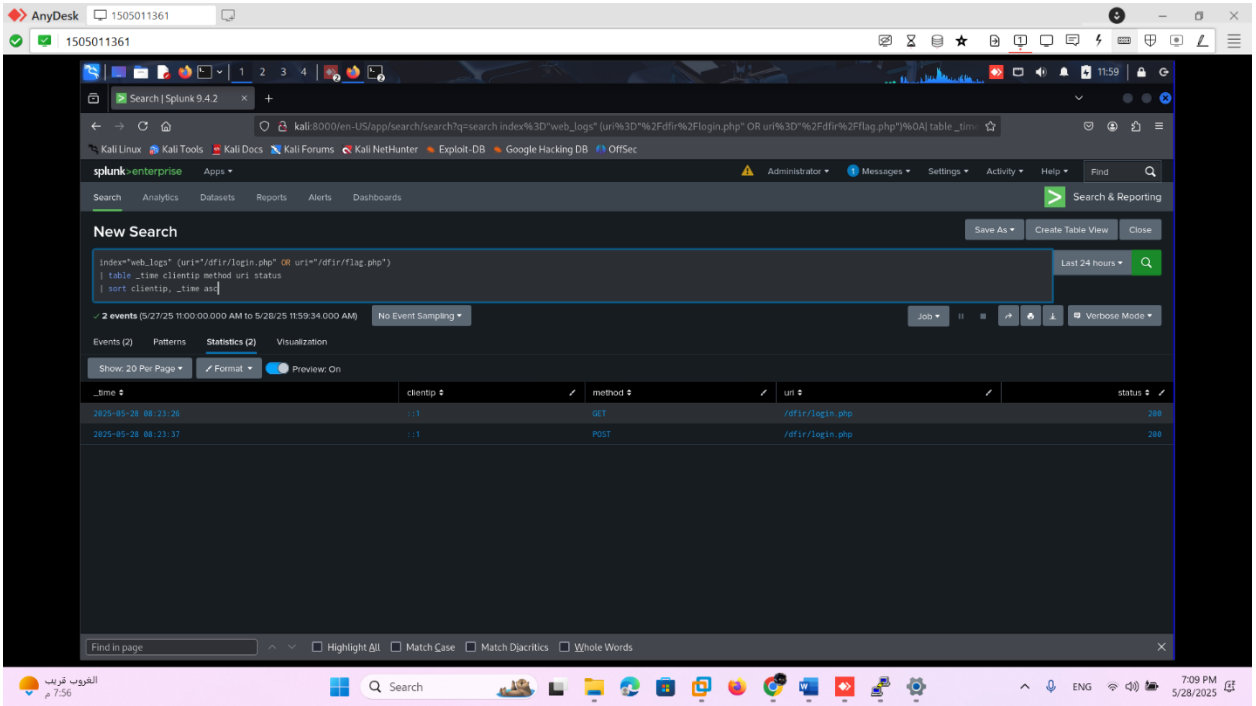
Splunk Settings Screenshot

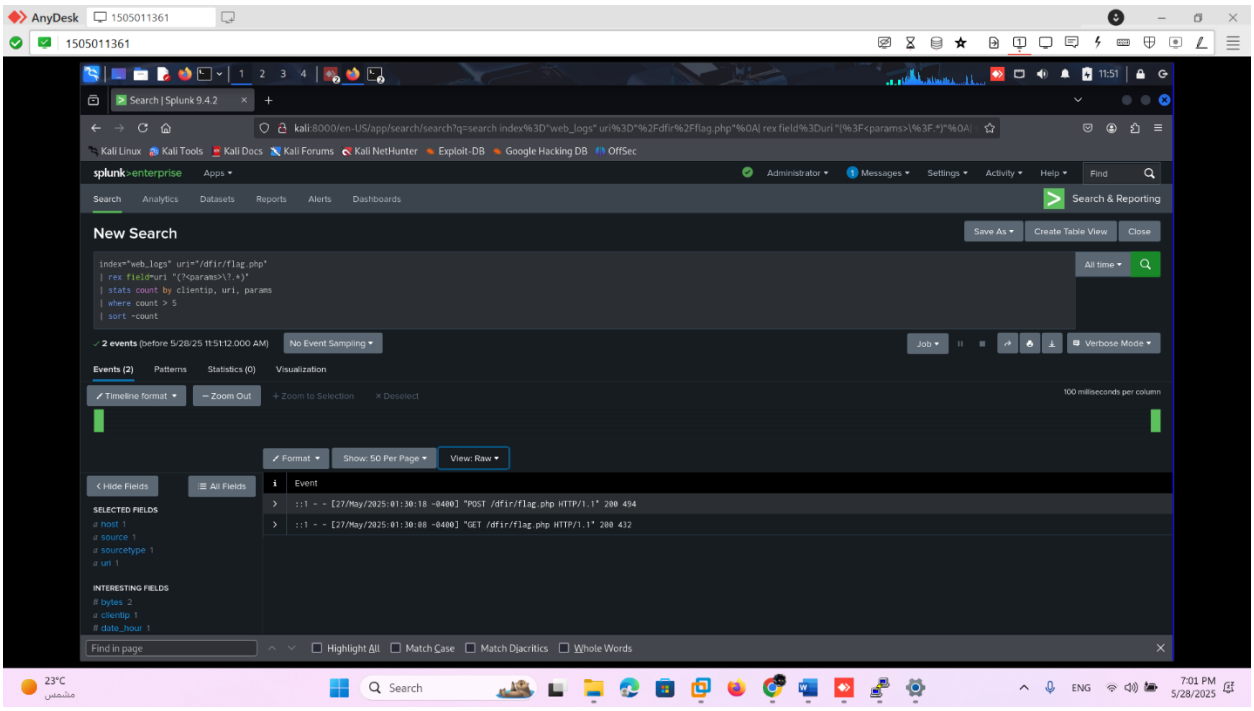


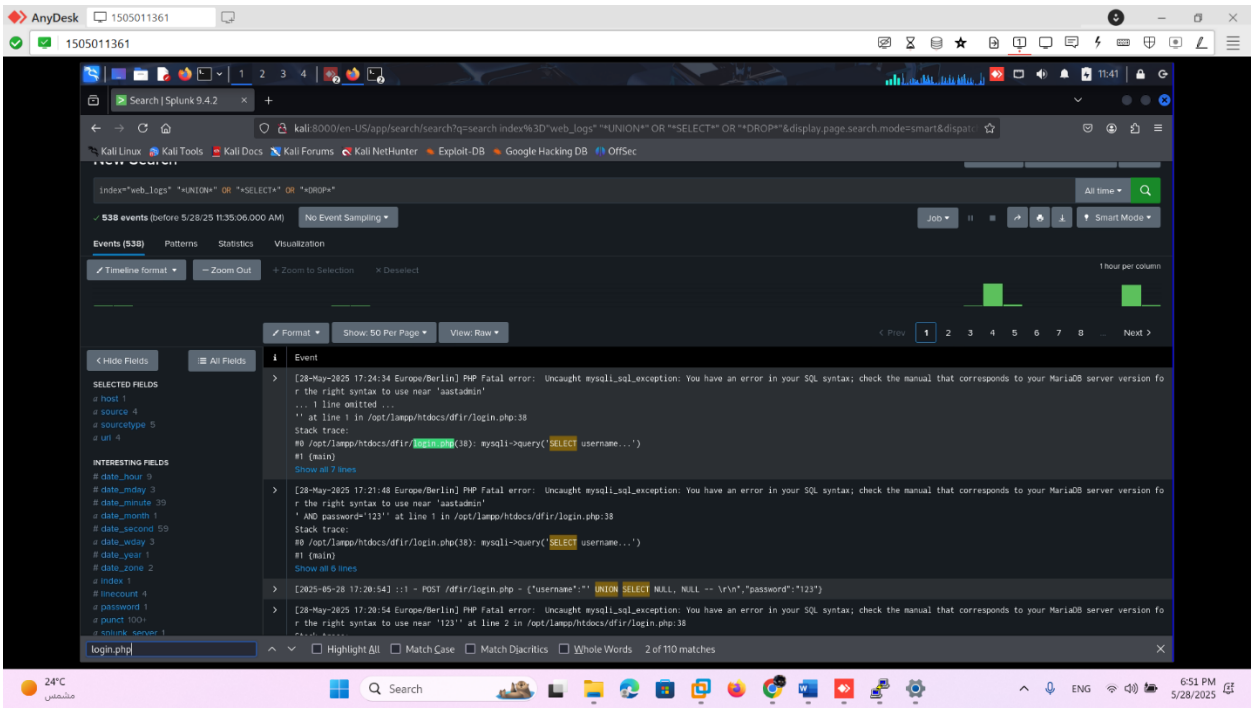
Splunk Dashboards & Usage:

Splunk was used to monitor and analyze incoming logs in real-time. Dashboards and search queries were utilized to detect various attack attempts. (Visual evidence from Splunk screenshots:









These screenshots demonstrate searching for specific keywords like "XSS", viewing overall traffic patterns, and identifying specific log sources.

1.3. Screenshot of Vulnerable Code with Embedded Flag

Several vulnerabilities and embedded flags were intentionally included in the web application.

- **SQL Injection Vulnerability (login.php):**
  - The login page directly incorporates user input into the SQL query:
    // From login.php
    $sql = "SELECT username, flag FROM users WHERE username='$user' AND password='$pass'";

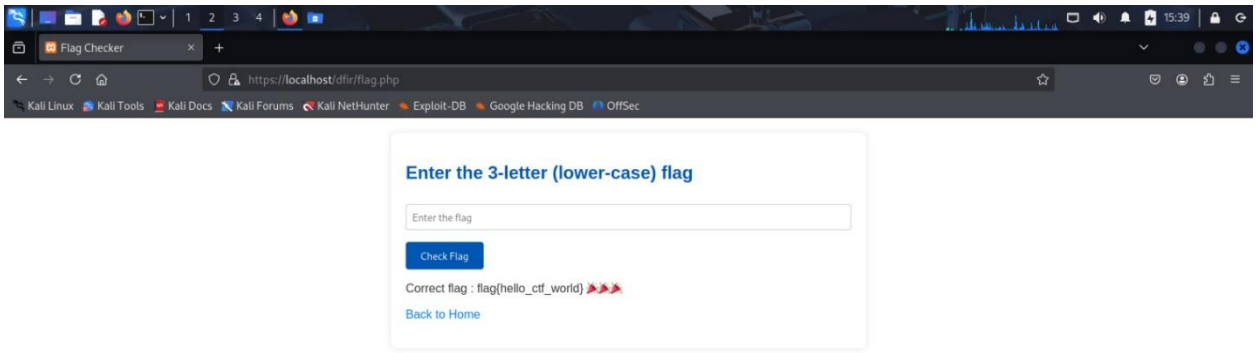
  - This allows an attacker to manipulate the SQL query. The flag flag_{AAST-SQL-INJECT-42} was embedded in the database for the aastadmin user and could be retrieved via SQL injection. (Evidence: login.php code, Screenshot 2025-05-28 at 10.25.25 PM.jpeg showing successful SQLi).
- **Hidden Flag in HTML (contact.php):**
  - The contact form contains a hidden input field with a flag:
    <input type="hidden" name="flag" value="flag_p1{x9k2m1v">

- **Flag in CSS Comment (style.css):**
  - A flag was embedded as a comment within the CSS file:
    /* From style.css, also viewable in Screenshot 2025-05-28 at 10.43.49 PM.jpeg */
    /*flag_p2-> w0-q1r2} */

  - This combines with the HTML flag to form: flag_p1{x9k2m1vw0-q1r2} (as noted in web_server.txt).

- **Flag in flag.php (Brute-Force Target):**
  - The flag.php page reveals flag{hello_ctf_world} upon successful submission of a 3-letter code. The code itself is not directly embedded but is discoverable through brute-forcing the 3-letter input.

Evidence:



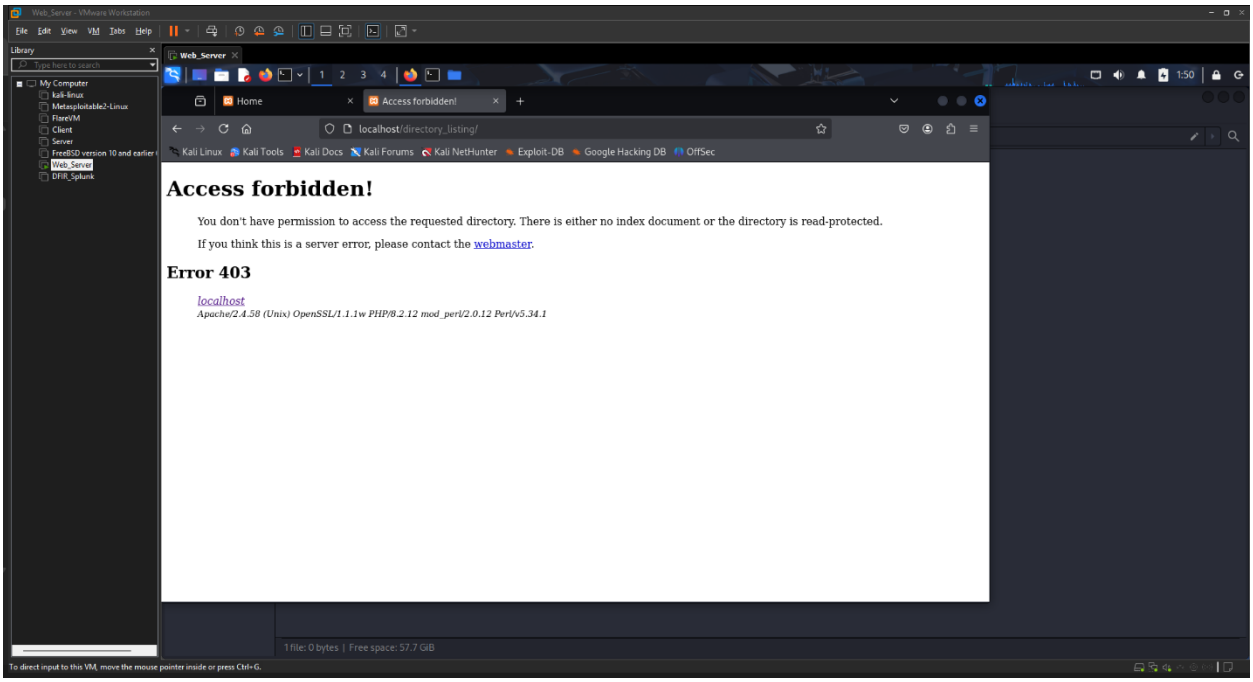
showing successful flag check

1.4. Evidence of Hardening and Defense Measures

Several steps were taken to harden the web server and application:

1. **Disabled Directory Listing:**
   - In /opt/lampp/etc/httpd.conf, the Options Indexes FollowSymLinks directive was modified or removed for the document root to prevent directory listing (as noted in web_server.txt and httpd.conf.txt).

The Screenshot



shows an "Access forbidden!" error, demonstrating that directory listing was indeed disabled for at least some paths.

2. **File Permission Hardening:**
   - Ownership of the web root (/opt/lampp/htdocs) was set to the Apache user (daemon:daemon): sudo chown -R daemon:daemon /opt/lampp/htdocs.
   - File permissions were restricted:
     - Files: sudo find /opt/lampp/htdocs -type f -exec chmod 644 {} \; (read/write for owner, read-only for group/others).
     - Directories: sudo find /opt/lampp/htdocs -type d -exec chmod 755 {} \; (read/write/execute for owner, read/execute for group/others).

3. **Basic Input Validation/Sanitization:**
    - **contact.php:** Uses htmlspecialchars() to encode messages before storing and displaying them, mitigating basic XSS from user-submitted messages. It also attempts to filter out messages containing the word "script".
    - **flag.php:** Uses preg_match('/^[a-z]{3}$/', $input_flag) to validate that the input is exactly three lowercase letters. It also uses prepared statements for database queries, preventing SQL injection on this page.

4. **HTTPS (Self-Signed Certificate):**
    - SSL/TLS was enabled using a self-signed certificate.
    - The certificate (mysite.crt) and private key (mysite.key) were generated using OpenSSL: sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mysite.key -out mysite.crt (from web_server.txt).
    - Apache SSL configuration (/opt/lampp/etc/extra/httpd-ssl.conf) was updated to use these files for localhost:443 (details in web_server.txt, httpd.conf.txt, mysite.crt.txt, mysite.key.txt).

5. **Restricted Database User Privileges:**
    - Specific MySQL users (webuser, bruteuser) were created with limited privileges, rather than using the root user for web applications (commands in web_server.txt).
        - webuser has permissions on the dfir database.
        - bruteuser has SELECT permissions on the bruteflag database.

6. **Prevention of .htaccess Viewing:**
    - The Apache configuration (httpd.conf.txt) includes:
      <Files ".ht*">
        Require all denied
      </Files>

      This prevents clients from viewing the content of .htaccess and .htpasswd files.

7. **Application-Level Request Logging:**
    - All PHP pages (login.php, contact.php, flag.php) implement a logRequest() function that records details of incoming GET/POST requests to /opt/lampp/logs/php_requests.log. This provides an additional layer of application-specific logging.

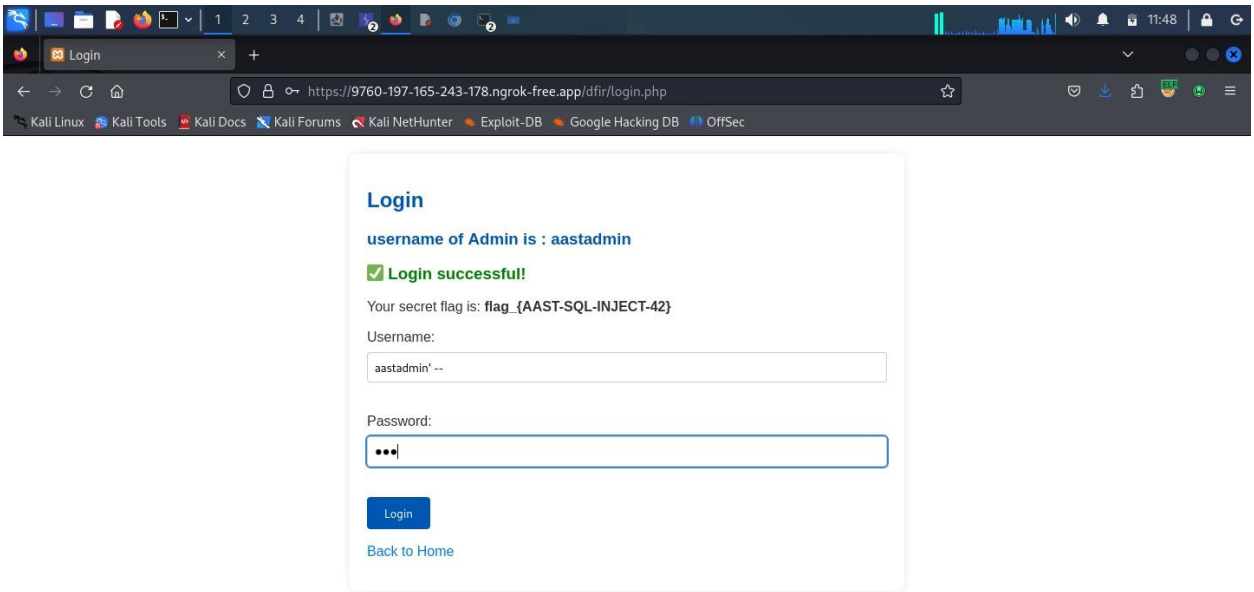
2. Offensive Report

This section details the attacks performed against the web application, the tools used, and the flags captured.

2.1. Attacks Performed

- **SQL Injection (SQLi):**
  - **Target:** login.php
  - **Method:** The username field was exploited due to direct concatenation of user input into the SQL query.
  - **Payload Example (conceptual, derived from login.php vulnerability and successful login screenshot):** aastadmin' -- (or similar variations to bypass password check). The php_requests.log.txt and php_error_log.txt show various SQLi payloads being attempted.
  - **Result:** Successful extraction of the flag associated with the aastadmin user: flag_{AAST-SQL-INJECT-42}.

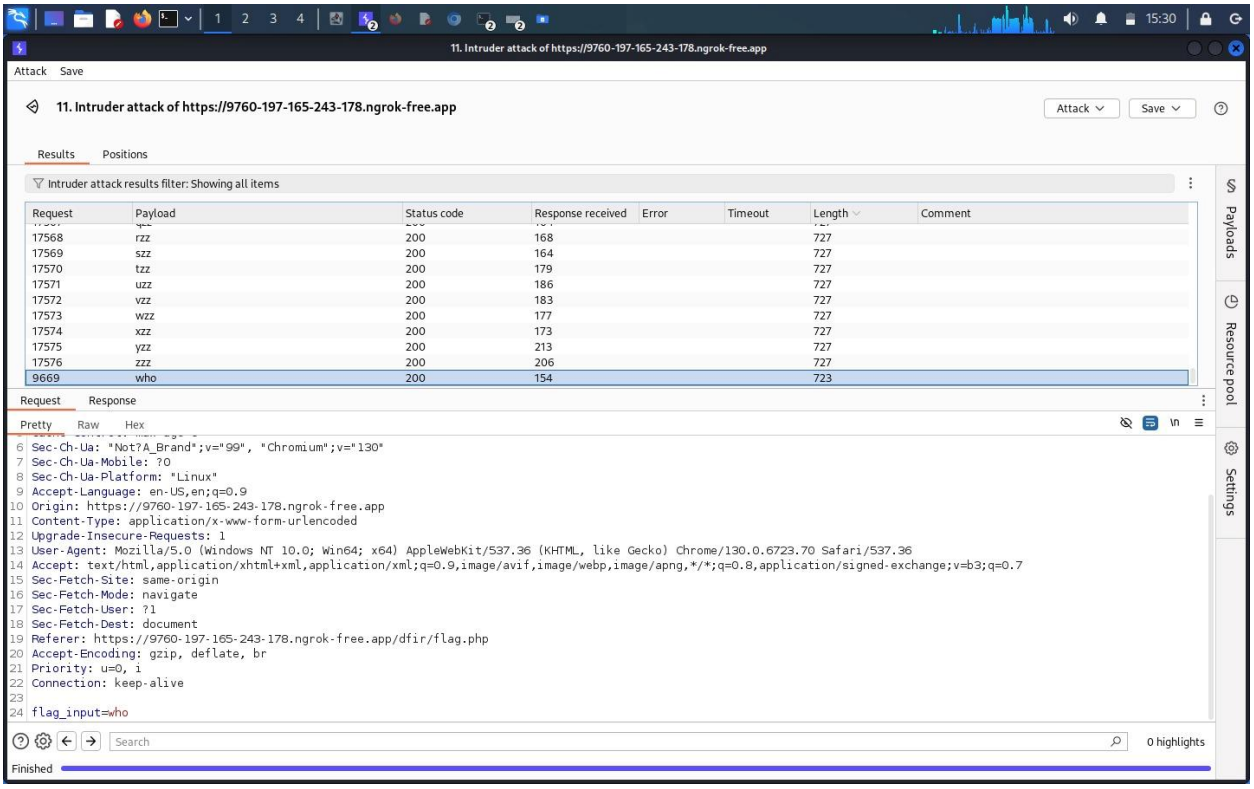
    Evidence:



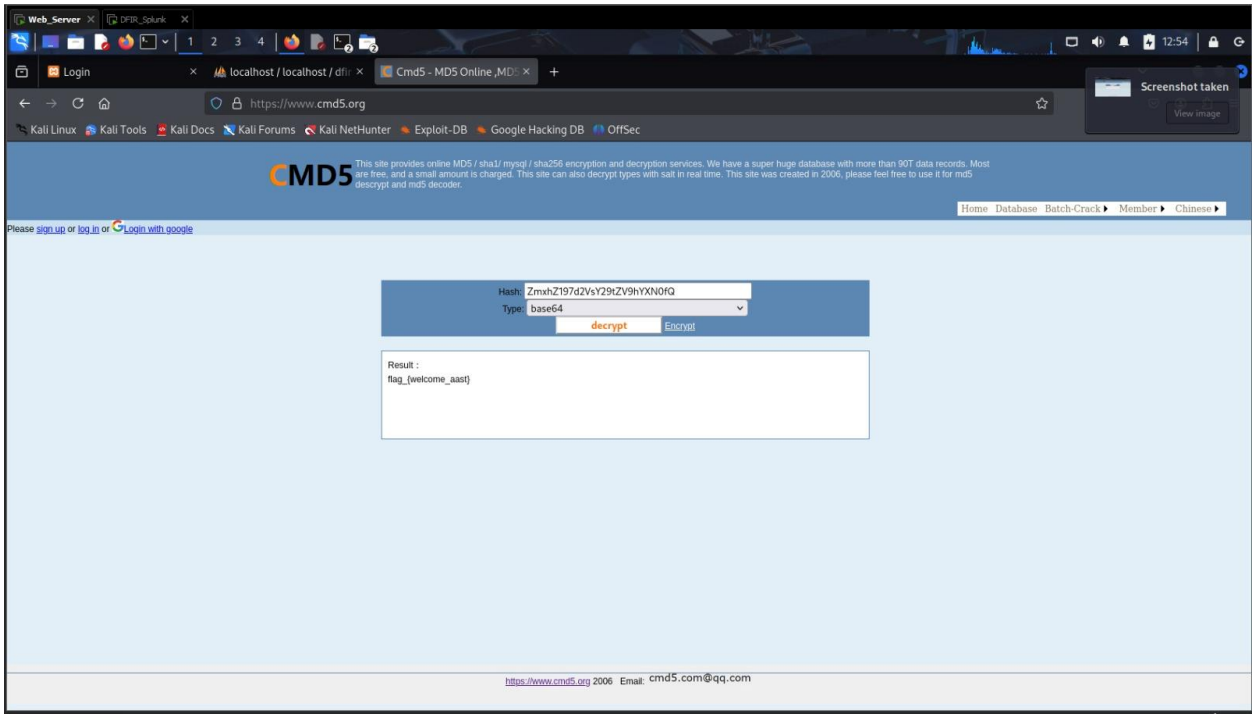
- **Brute Force Attack:**
  - **Target:** flag.php
  - **Method:** The 3-letter input field was subjected to a brute-force attack to find the correct combination.
  - **Result:** Successful discovery of the 3-letter code, revealing the flag: flag{hello_ctf_world}.

Evidence:

Burp Suite Intruder screenshot showing systematic attempts:



- **Source Code Analysis / Information Disclosure:**
  - **Target:** contact.php (HTML source), style.css (CSS source).
  - **Method:** Reviewing the page source and linked CSS file.
  - **Result:**
    - Discovery of flag_p1{x9k2m1v in a hidden field in contact.php. (Evidence: contact.php code, Screenshot 2025-05-28 at 10.43.50 PM.jpeg).
    - Discovery of flag_p2-> w0-q1r2} in a CSS comment in style.css. (Evidence: style.css code, Screenshot 2025-05-28 at 10.43.49 PM.jpeg).
    - Combined flag: flag_p1{x9k2m1vw0-q1r2}.
- **Base64 Decoding:**
  - **Target:** A Base64 encoded string found in the notes table of the dfir database.
  - **Encoded String:** ZmxhZ197d2VsY29tZV9hYXN0fQ== (from web_server.txt and SQLMap output screenshot Screenshot 2025-05-28 at 10.22.22 PM.jpeg).
  - **Method:** Using an online Base64 decoder like cmd5.org



  - **Result:** Decoded flag: flag_{welcome_aast}.

- **Cross-Site Scripting (XSS) - Attempted/Potential:**
  - **Target:** contact.php message submission.
  - **Method:** While htmlspecialchars() is used, which prevents stored XSS from rendering HTML tags directly, the project description mentions XSS as a possible vulnerability. The contact.php also filters the word "script". The Splunk screenshot XSS.jpg indicates searches for XSS attempts. Further investigation would be needed to confirm if any bypasses exist.

2.2. Tools Used

- **SQLMap:** Extensively used for detecting and exploiting SQL injection vulnerabilities, particularly on login.php. Commands included discovering databases, tables, columns, and dumping data (as listed in sql.txt and shown in SQLMap output screenshots like Screenshot 2025-05-28 at 10.21.30 PM.jpeg, Screenshot 2025-05-28 at 10.21.23 PM.jpeg, Screenshot 2025-05-28 at 10.20.50 PM.jpeg, Screenshot 2025-05-28 at 10.20.27 PM.jpeg, Screenshot 2025-05-28 at 10.19.09 PM.jpeg, Screenshot 2025-05-28 at 10.16.07 PM.jpeg).
- **Burp Suite:** Used for intercepting requests and performing automated attacks like brute-forcing. The Intruder tool was likely used against flag.php. (Evidence: Screenshot 2025-05-28 at 10.42.35 PM.jpeg showing Intruder results, Screenshot 2025-05-28 at 10.14.02 PM.jpeg showing a request in Burp).
- **Web Browser Developer Tools:** Used for inspecting page source (HTML, CSS) to find hidden flags. (Evidence: Screenshot 2025-05-28 at 10.43.50 PM.jpeg, Screenshot 2025-05-28 at 10.43.49 PM.jpeg).
- **Online Base64 Decoder:** Used to decode the Base64 encoded flag. (Evidence: Screenshot 2025-05-28 at 10.21.58 PM.jpeg).
- **Nmap (Mentioned in PDF):** Likely used for initial network scanning to discover web servers, although direct logs/screenshots of Nmap usage were not explicitly provided among the viewable files.

2.3. Flags Captured

A total of five distinct flags were captured:

1. **SQLi Flag:** flag_{AAST-SQL-INJECT-42}
    - Source: users table in dfir database, retrieved via SQLi on login.php.
2. **Base64 Decoded Flag:** flag_{welcome_aast}
    - Source: notes table in dfir database (sys_flag field, Base64 encoded as ZmxhZ197d2VsY29tZV9hYXN0fQ==), retrieved via SQLMap and decoded.
3. **Brute Force Flag:** flag{hello_ctf_world}
    - Source: Revealed by flag.php after submitting the correct 3-letter code ("who" as per Burp Intruder screenshot).
4. **Picture/Hidden Data Flag:** flag_{b7x9-Klm2_8zPq-A9tR}
    - Source: Mentioned in web_server.txt as "at picture". The screenshot WhatsApp Image 2025-05-28 at 10.12.47 PM.jpeg shows this flag embedded in what looks like non-UTF-8 text, possibly an image file opened as text.
5. **Combined Code Flag:** flag_p1{x9k2m1vw0-q1r2}
    - Source: Combination of:
        - flag_p1{x9k2m1v (from hidden field in contact.php HTML).
        - w0-q1r2} (from CSS comment in style.css, completing flag_p2).

2.4. Screenshots and Logs of Success/Failure

- **Successful SQL Injection:** Screenshot 2025-05-28 at 10.25.25 PM.jpeg shows the login page displaying "Login successful!" and the flag flag_{AAST-SQL-INJECT-42} after submitting aastadmin' -- as the username.
- **Successful Brute Force on flag.php:** Screenshot 2025-05-28 at 10.42.57 PM.jpeg shows the flag.php page displaying "Correct flag : flag{hello_ctf_world}". Screenshot 2025-05-28 at 10.42.35 PM.jpeg (Burp Intruder) shows the payload "who" resulting in a different response length/status, indicating success.
- **SQLMap Execution:** Screenshots like Screenshot 2025-05-28 at 10.22.22 PM.jpeg show SQLMap successfully dumping data from the notes table, including the Base64 encoded sys_flag.
- **Base64 Decoding Success:** Screenshot 2025-05-28 at 10.21.58 PM.jpeg shows

the successful decoding of ZmxhZ197d2VsY29tZV9hYXN0fQ== to flag_{welcome_aast}.

- **Source Code Flag Discovery:** Screenshot 2025-05-28 at 10.43.50 PM.jpeg (view-source of contact.php) and Screenshot 2025-05-28 at 10.43.49 PM.jpeg (CSS in browser inspector) show the embedded flags.
- **Log Evidence:**
  - php_requests.log.txt: Contains numerous entries showing SQLi payloads being sent to login.php and brute-force attempts on flag.php.
  - php_error_log.txt: Shows SQL syntax errors generated by some of the SQLi attempts, confirming the vulnerability.
  - access_log.txt: General web server access patterns during the attacks.
  - The other attack screenshots (which appeared black) would have provided further visual evidence of these processes.

3. Log Analysis Report

This section details the analysis of logs collected in Splunk, focusing on attack traces, detected activities, and potential responses.

3.1. Attack Traces in Splunk

Splunk was instrumental in monitoring and identifying malicious activities by analyzing the forwarded logs.

- **SQL Injection Attempts:**
  - Queries in Splunk on index=web_logs sourcetype=php_requests containing keywords like UNION, SELECT, '--, or common SQLi patterns in the params field for login.php would reveal these attempts.
  - sourcetype=php_error would show corresponding SQL syntax errors from the backend.
  - The Splunk screenshot SQL ingection.jpg shows events related to "UNION" or "SELECT" keywords, indicating SQLi detection.
- **Brute Force on flag.php:**
  - Analyzing index=web_logs sourcetype=php_requests for a high volume of POST requests to flag.php from a single IP with varying 3-letter payloads in a short time frame.
  - The Splunk screenshot bruteforce .jpg likely shows a query filtering for multiple attempts to flag.php.
- **XSS Attempts:**
  - Searching index=web_logs sourcetype=php_requests for common XSS payloads like <script>, onerror, etc., in the message parameter of contact.php.
  - The Splunk screenshot XSS.jpg demonstrates searching for "XSS" related events.
- **General Traffic Monitoring:**
  - The all traffic.jpg Splunk screenshot shows a general overview of web requests, which can be used to identify unusual request volumes or patterns.
  - The action of attacker .jpg screenshot likely filters for specific attacker IPs or known malicious URI patterns.

3.2. IPs, Payloads, Methods Detected

Analysis of the provided raw log files and Splunk capabilities allows for the detection of:

- **Attacker IP Addresses:**
  - The logs (e.g., access_log.txt, php_requests.log.txt) record the client IP for each request (mostly ::1 or 127.0.0.1 for local testing, but would be external IPs in a real scenario).
- **Payloads:**
  - **SQLi Payloads (from php_requests.log.txt and php_error_log.txt):**
    - 'aastadmin' --
    - 'aastadmin' OR '1'='1
    - Various UNION SELECT attempts.
    - Error-based payloads like EXTRACTVALUE.
    - Boolean-based blind payloads.
  - **Brute Force Payloads (from php_requests.log.txt and Burp screenshot):**
    - Systematic 3-letter combinations (e.g., "aaa", "aab", ..., "who", ..., "zzz") sent to flag.php.
  - **XSS Payloads (conceptual):**
    - <script>alert(1)</script> (though this would be sanitized by htmlspecialchars in contact.php and the word "script" filtered).
- **Methods Detected:**
  - **GET:** Used for accessing pages like index.php, login.php (initial load), contact.php (initial load), flag.php (initial load).
  - **POST:** Used for submitting data to login.php (credentials), contact.php (messages), flag.php (flag attempts).

3.3. Response Actions (Block IP, Patch, etc.)

Based on the detected attacks, the following response actions could be taken or were implicitly part of the setup:

- **Immediate Actions (Conceptual for a Live Scenario):**
  - **Block Attacker IP:** If a specific external IP is identified as malicious in Splunk (e.g., launching a high-volume brute-force or persistent SQLi), it could be blocked at the firewall level.

- **Proactive/Implemented Measures (Hardening):**
  - **SSL/TLS:** Encrypts data in transit, protecting against sniffing but not application-layer attacks.
  - **Disabled Directory Listing:** Prevents attackers from easily discovering files.
  - **File Permission Hardening:** Limits the web server's ability to write to unintended locations or attackers from modifying critical files if other vulnerabilities are exploited.
  - **Input Sanitization in contact.php:** htmlspecialchars mitigates basic stored XSS. Filtering "script" provides an additional, albeit simple, layer.
  - **Prepared Statements in flag.php:** This is a key defense against SQL injection for this specific functionality.
- **Patching Vulnerabilities (Recommendations):**
  - **login.php SQL Injection:** The primary vulnerability. This **must** be patched by rewriting the SQL query to use prepared statements with parameterized queries. This is the most critical fix.
    // Example of a patched query for login.php
    // $stmt = $conn->prepare("SELECT username, flag FROM users WHERE username=? AND password=?");
    // $stmt->bind_param("ss", $user, $pass); // Assuming passwords are not hashed, which is another major issue.
    // $stmt->execute();
    // $result = $stmt->get_result();

  - **Password Storage:** Passwords in the users table appear to be stored in plaintext or a very weak format. They should be securely hashed (e.g., using password_hash() and verified with password_verify()).
  - **Information Disclosure:** Remove flags from comments (style.css) and hidden HTML fields (contact.php) in a production environment. Flags in the database for CTF purposes are acceptable if access is properly controlled.

3.4. Improvements Implemented Post-Attack

The following improvements are recommended for a more secure system:

1. **Implement Parameterized Queries:** Universally adopt prepared statements for all database interactions, especially in login.php, to eliminate SQL injection vulnerabilities.
2. **Secure Password Hashing:** Store all user passwords using strong, modern hashing algorithms like Argon2 or bcrypt with unique salts.
3. **Web Application Firewall (WAF):** Consider implementing a WAF to provide an additional layer of defense against common web attacks, including SQLi and XSS.
4. **Enhanced Input Validation:** Implement more robust input validation on all user-supplied data, not just for specific fields. This includes length checks, character set restrictions, and type checking.
5. **Security Headers:** Implement security-related HTTP headers like Content Security Policy (CSP), X-Content-Type-Options, X-Frame-Options, and X-XSS-Protection to further mitigate XSS and other client-side attacks.
6. **Regular Security Audits & Code Reviews:** Conduct regular code reviews and security audits to identify and remediate vulnerabilities proactively.
7. **Update Software:** Keep all server software (OS, Apache, PHP, MySQL, Splunk) updated to the latest stable versions to protect against known vulnerabilities.
8. **Least Privilege for Database Users:** Ensure database users have only the minimum necessary permissions for their tasks.
9. **Centralized and More Granular Logging in Splunk:** While basic log forwarding is set up, create more specific Splunk dashboards and alerts for critical security events (e.g., multiple failed logins, SQLi detection, XSS detection).

The Conclusion

This project successfully simulated a web security environment, demonstrating the practical aspects of setting up a server, identifying vulnerabilities, performing attacks, and analyzing logs. The findings highlight common web application weaknesses like SQL injection and the importance of robust logging and monitoring for timely detection and response. The implemented hardening measures provided a baseline level of security, but the offensive testing revealed areas requiring significant improvement, primarily in how user input is handled in database queries and how sensitive information (like flags and passwords) is stored and exposed.

All Project files:
https://drive.google.com/drive/folders/18DKLd1SlYtQVu0Uxidf_HDS05HxYMnQW

The End :)