Report for ABC pvt. Ltd.

Report Submitted by : Vastnet

Code assigned to organization :xx

Report Submitted On : xx.xx.2026

Test Bed details: XYZ Webserver

Contact Number: +91-XXXXXXXXXX

EXECUTIVE SUMMARY

Vastnet conducted a comprehensive **Vulnerability Assessment and Penetration Testing (VAPT)** engagement for the XXXX web application to evaluate its security posture prior to production deployment. The assessment was performed using a combination of automated tools, manual testing techniques, and in-depth source code review, ensuring a thorough examination of the application's attack surface. The objective of the engagement was to identify security weaknesses that could potentially be exploited by malicious actors and provide actionable remediation guidance to strengthen the overall security posture of the application.

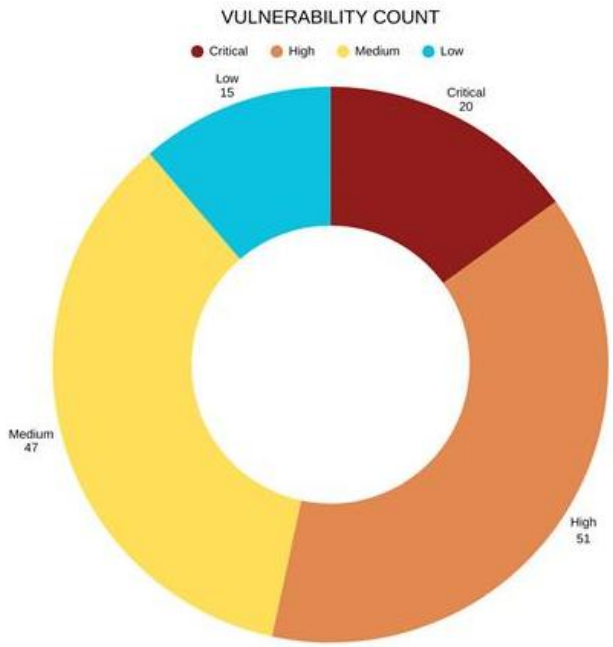
During the course of the assessment, the testing team performed extensive validation across all accessible components of the application, including authentication mechanisms, input parameters, APIs, session management, access controls, business logic, file handling mechanisms, and server-side implementations. Special emphasis was placed on parameter manipulation, privilege escalation scenarios, insecure data handling, injection vulnerabilities, authentication bypass vectors, and misconfiguration issues. The testing approach ensured that no accessible parameter, endpoint, or functional module was left unassessed, including detailed inspection of underlying source code logic wherever available.

A total of **133** security vulnerabilities were identified during the assessment, categorized based on severity and contextual impact. These included **20 Critical, 51 High, 47 Medium,** and **15 Low** severity vulnerabilities. While industry-standard scoring frameworks such as CVSS were considered during severity classification, the final risk ratings were determined by also incorporating the real-world operational context of the application, including its intended production deployment, potential exposure, privilege boundaries, and the sensitivity of the data processed by the platform.

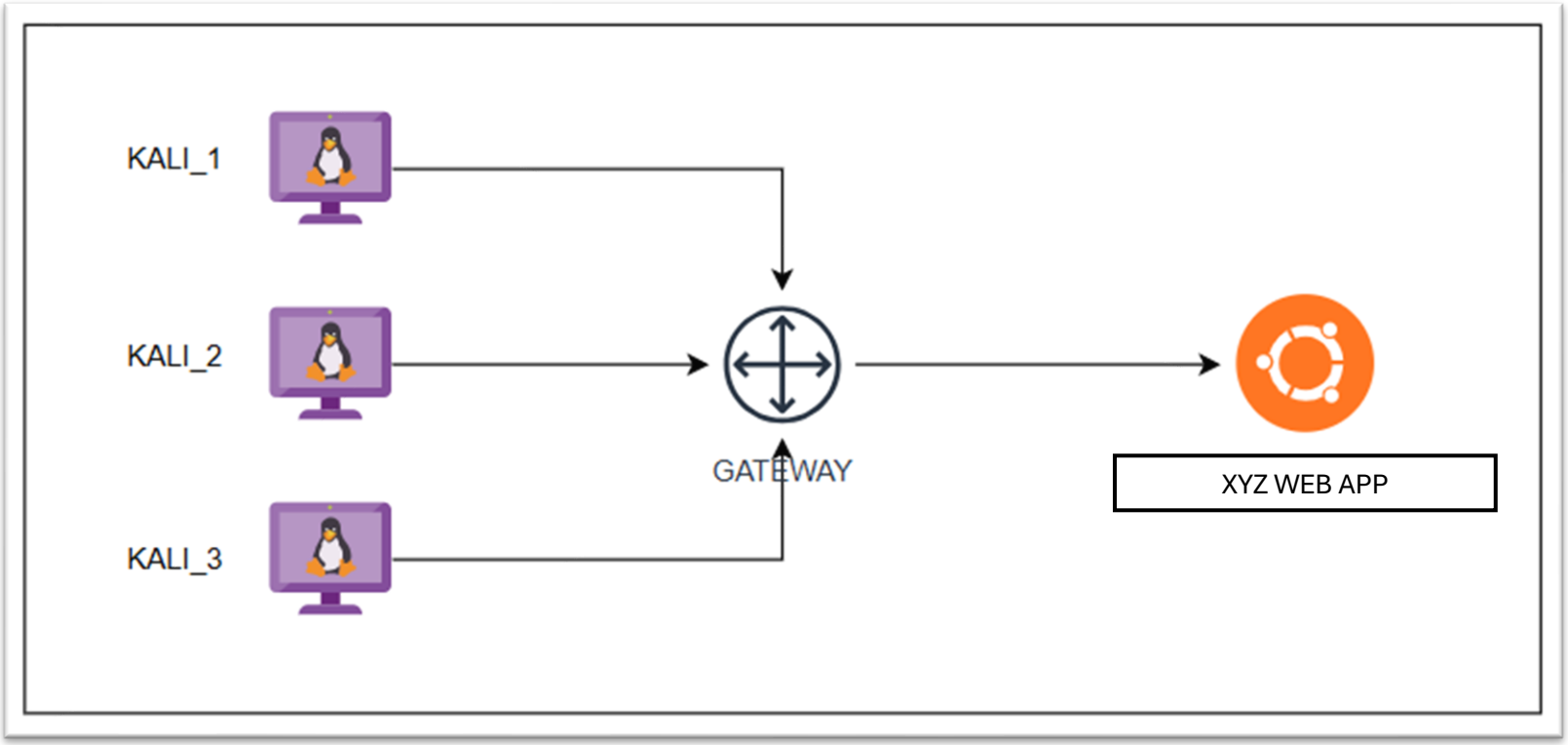
Several identified vulnerabilities present significant security risks that could potentially lead to unauthorized access, data exposure, account compromise, privilege escalation, and disruption of application functionality if exploited by an attacker. Given that the application is intended for a production environment, certain vulnerabilities were prioritized as Critical or High severity due to their direct impact on confidentiality, integrity, and availability of the system and its data.

Vulnerability Wise Colour Coding:-

Critical	High	Medium	Low



1. **OVERVIEW OF ARCHITECTURE SETUP**



2. LIST OF STANDARDS, FRAMEWORKS, SEVERITY SCORING SYSTEMS, AND OTHER REFERENCES:

S. No.	Standard / Framework used	Severity Score System Used	Other References
1	OWASP (Top 10, ASVS, WSTG)	Owasp Risk Rating Methodology	-
2	NIST	CVSS (Common Vulnerability Scoring System) version 4.0	-

3. LIST OF VULNERABLE PARAMETER, LOCATION DISCOVERED

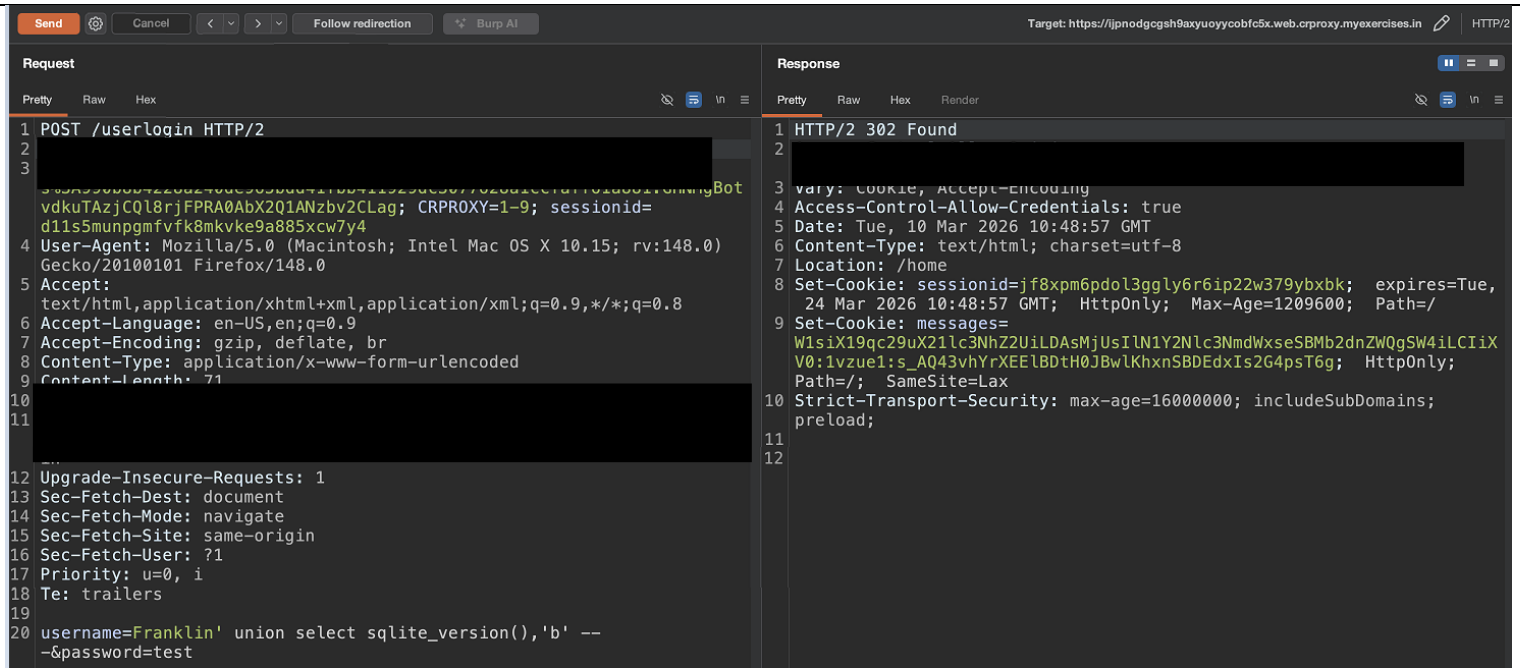
S.NO.	Path / Parameter	Name of the vulnerability	References (CWE-ID, CVE, CVSS, OWASP TOP10, etc.)	Severity
1	User Name	SQL Injection Leading to Authentication Bypass	CWE-89	Critical
2	/password_reset/	Sensitive Information Leakage Forgot Password OTP Leak	CWE-200	Critical
3	/controllerlogin/	SQL Injection Leading to Authentication Bypass	CWE-89	Critical
4	/password_reset/otp/	Missing Rate Limiting on OTP Validation Brute Force	CWE-307	Critical

4. TOOLS USED FOR CARRYING OUT THE TESTS:-

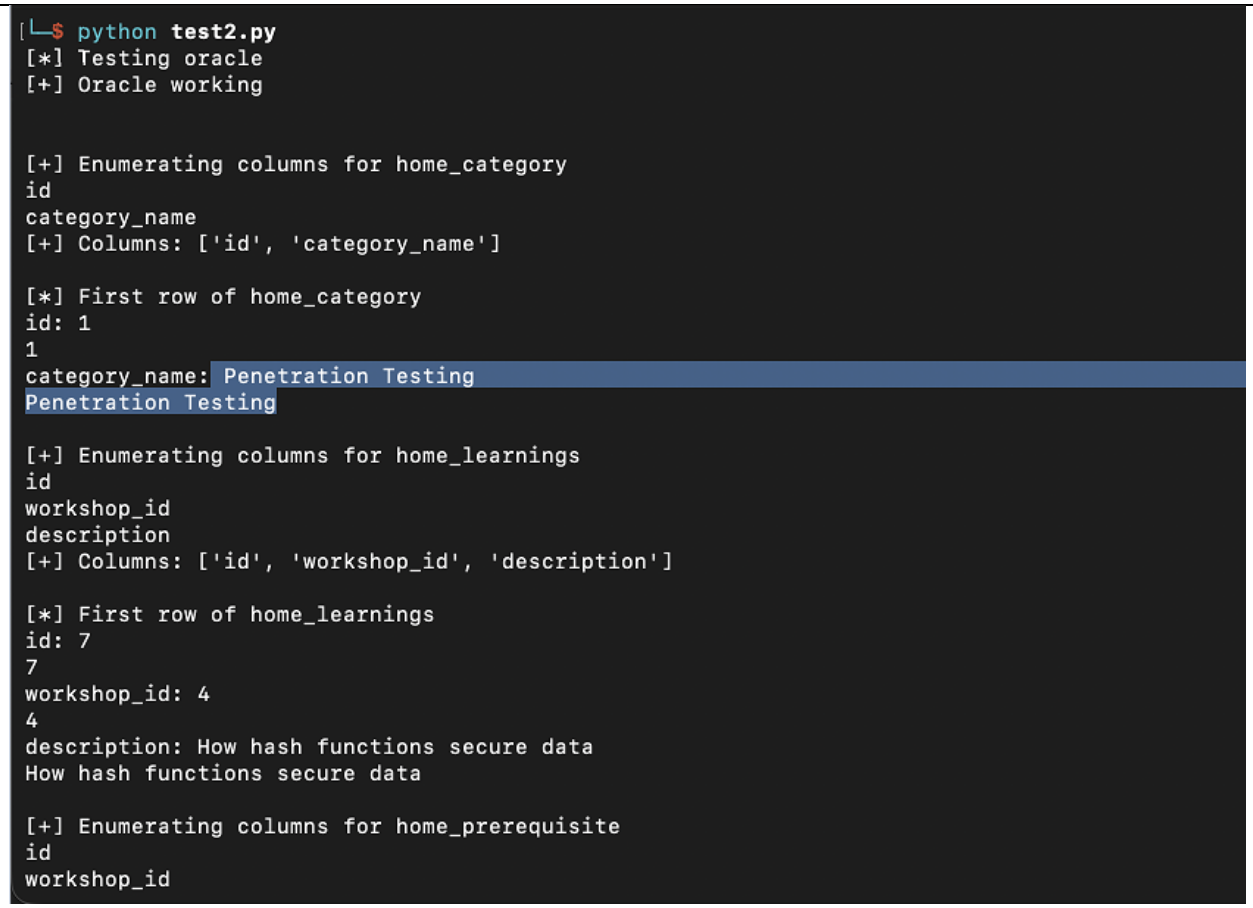
Description	Tools Used
Vulnerability Scanning and Management	➢ Nessus Expert
	➢ Tenable
	➢ Nessus
Network Scanning and Enumeration	➢ Smbclient
	➢ Nmap
	➢ Crackmapexec
	➢ Wireshark
Web Application Security	➢ Burp Suite
	➢ Dirbuster
	➢ Nikto
	➢ Owasp Zed Attack Proxy (ZAP)
	➢ SQL MAP
Penetration Testing Frameworks and Exploitation	➢ Metasploit
	➢ Impacket
	➢ Secret Dump
Operating Systems for Security	➢ Kali Linux
	➢ ParrotOS
Monitoring and Logging	➢ Splunk

VULNERABILITY_1 SQL Injection Leading to Authentication Bypass

S. No.	1	
Name of Vulnerability:	**SQL Injection Leading to Authentication Bypass**	
Vulnerable Point / Error Location	Vulnerable Location	http://XXX.XXX.XXX.XXX
	Vulnerable Path/ Port/ URL	/userlogin
	Vulnerable Parameter	“username”
CVE/CWE	**CWE-89**	
CVSS / EPSS Score	**9.8**	
Description	The login interface fails to sanitize user-supplied input before incorporating it into a database query. By injecting malicious SQL syntax (e.g., ' OR 1=1--), an attacker can manipulate the query logic to return a "true" result, effectively bypassing the authentication mechanism and gaining unauthorized access to the application.	

Proof of concept and Steps to Reproduce **with** clearly **visible screenshots**	


└─$ python test.py
[[*] Testing oracle...
[+] Oracle working
[*] Extracting SQLite version
3.45.1
[+] SQLite Version: 3.45.1
[*] Enumerating tables
[+] Table 0: django_migrations
[+] Table 1: auth_group_permissions
[+] Table 2: auth_user_groups
[+] Table 3: auth_user_user_permissions
[+] Table 4: django_admin_log
[+] Table 5: django_content_type
[+] Table 6: auth_permission
[+] Table 7: auth_group
[+] Table 8: auth_user
[+] Table 9: django_session
[+] Table 10: home_category
[+] Table 11: home_learnings
[+] Table 12: home_prerequisite
[+] Table 13: home_tag
[+] Table 14: axes_accessattempt

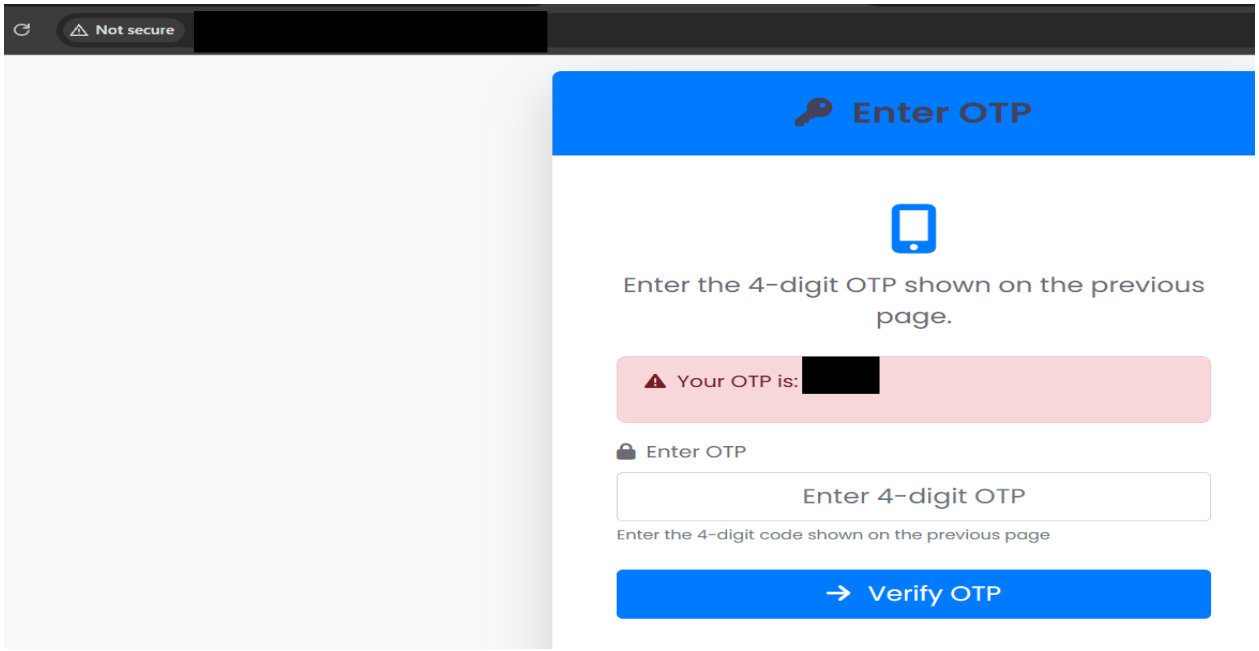


a. Intercepted the `/userlogin` request using Burp Suite and identified the `username` parameter.
b. Injected a SQL payload in the `username` field to test for SQL injection.
c. The application executed the injected query and revealed the SQLite database version.
d. Using the same injection point, database tables and columns were enumerated confirming the SQL Injection vulnerability.

Workaround / Solutions / Recommendations	**Primary Fix:** Use **Parameterized Queries (Prepared Statements)** for all database interactions to ensure input is treated as data, not executable code. **Secondary Fix:** Implement an Object-Relational Mapper (ORM) and enforce strict input validation using allow-lists. Ensure the database user operates under the **Principle of Least Privil**
References (Optional)	OWASP: SQL Injection Prevention Cheat Sheet PortSwigger: SQL Injection Authentication Bypass
Additional observations / information w.r.t this vulnerability (Optional) *	During testing, it was observed that the database error messages are verbose, providing clues about the underlying table structure. It is highly recommended to disable detailed error reporting in the production environment to prevent further reconnaissance.

VULNERABILITY_2 Sensitive Information Leakage Forgot Password OTP Leak

S. No.	2	
Name of Vulnerability:	**Sensitive Information Leakage: Forgot Password OTP Leak**	
Vulnerable Point / Error Location	Vulnerable Location	http://XXX.XXX.XXX.XXX
	Vulnerable Path/ Port/ URL	/password reset/
	Vulnerable Parameter	NA
CVE/CWE	CWE-200	
CVSS / EPSS Score	9	
Description	The application leaks the generated One-Time Password (OTP) within the HTTP response body during the password reset request. An attacker can intercept this response using a proxy tool, bypass the need for email access, and successfully reset any user's password, leading to full account takeover.	
Proof of concept and Steps to Reproduce **with** clearly **visible screenshots**	a. Just enter username in forget passowrd b. We can see otp is reflected In webpage	

	
Workaround / Solutions / Recommendations	**Immediate Fix:** Ensure that OTPs are never returned in the API response or client-side logs. The OTP should only be sent via out-of-band channels (Email/SMS). **Further Fix:** Implement strict rate limiting on the "Forgot Password" endpoint and ensure the OTP has a short expiration time (e.g., 5 minutes).
References (Optional)	OWASP: Forgotten Password Cheat Sheet CWE-200: Exposure of Sensitive Information https://cwe.mitre.org/data/definitions/200.html
Additional observations / information w.r.t	The application does not currently implement any rate-limiting or "lock-out" mechanism on the OTP entry field. This significantly increases the risk, as even if the leak is patched, the short OTP length remains vulnerable to brute-force attacks.

this vulnerability (Optional) *	