**DSE - Ethical Hacking**

**Practical File**

**Hemant Kr Singh**

**22058570006**

**B.Sc. (Hons.) Computer Science**

**6th Sem**

**Practicals**

**Q1) Introduction to Vulnerabilities and OWASP Top 10:**

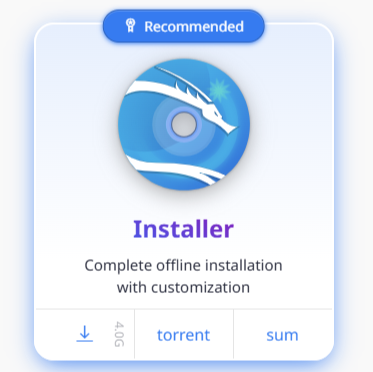
**Set up a local installation of OWASP Juice Shop or DVWA (Damn Vulnerable Web Application) on a virtual machine.Identify vulnerabilities such as SQL Injection or Cross-Site Scripting (XSS).**

**Tools: VirtualBox/VMware, OWASP Juice Shop/DVWA.**

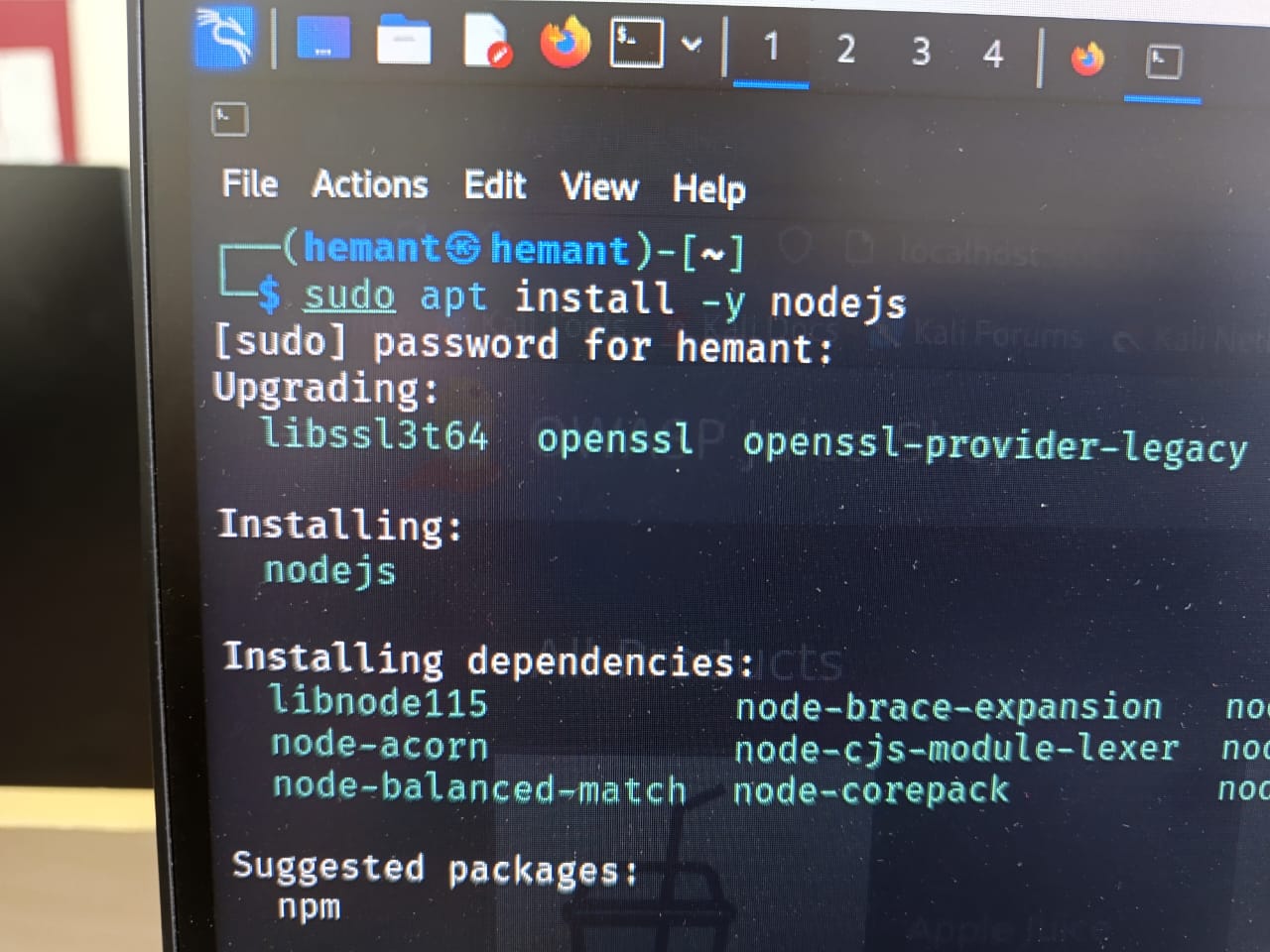
**Practical:-**

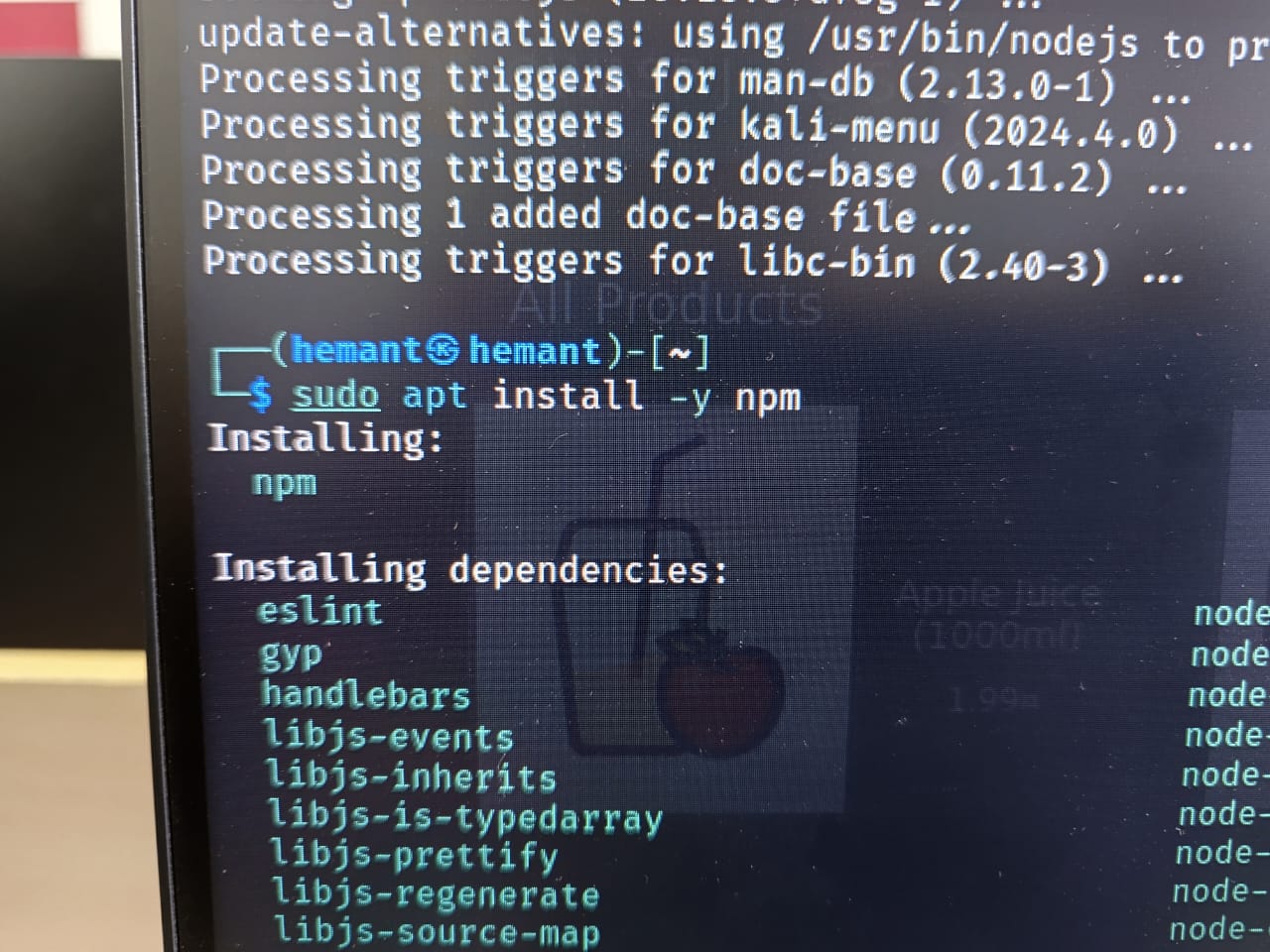
**Install Kali Linux ISO from-** [**https://www.kali.org/get-kali/#kali-installer-images**](https://www.kali.org/get-kali/#kali-installer-images)

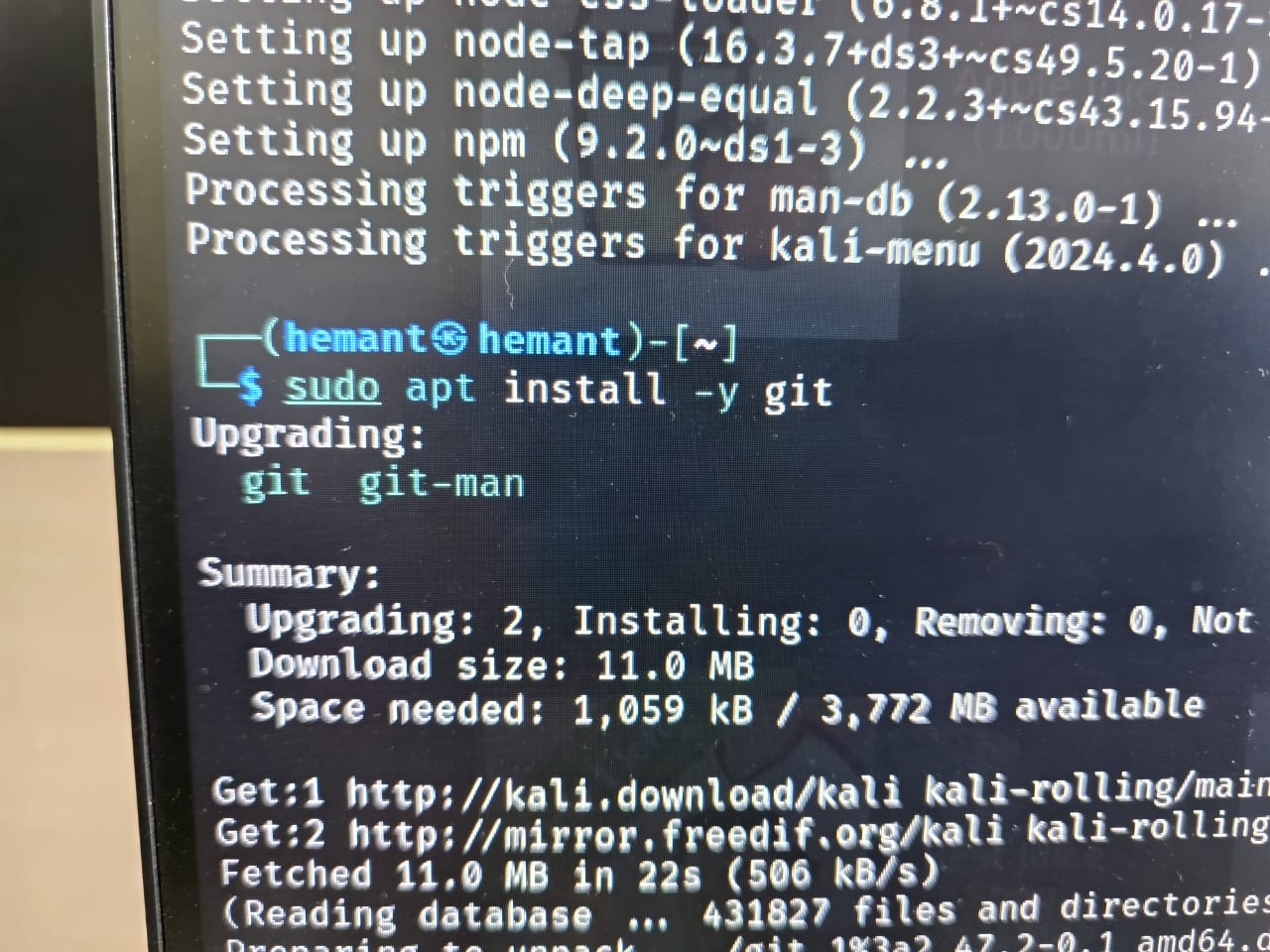
**Click on 4.0G install button**

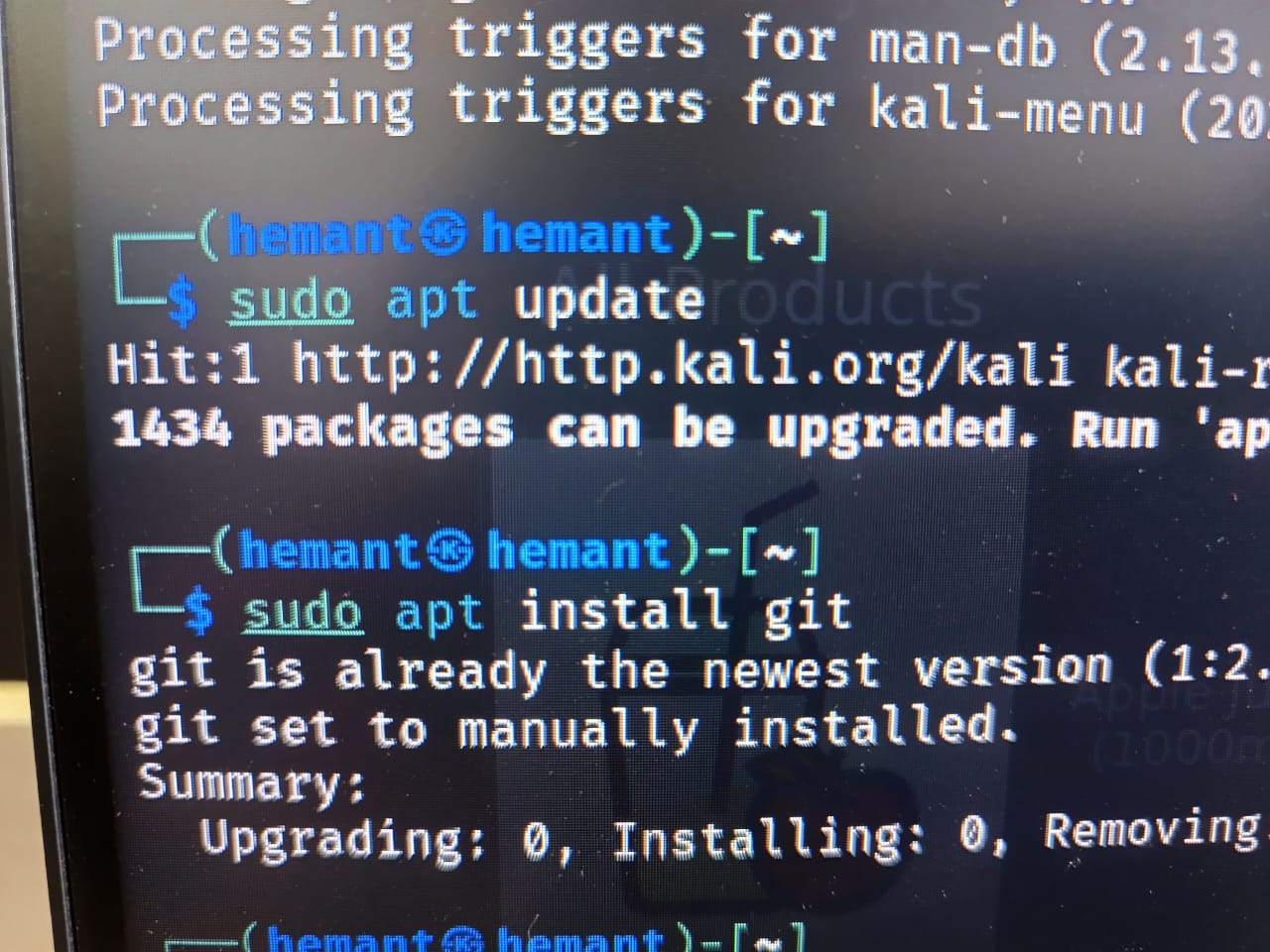
****

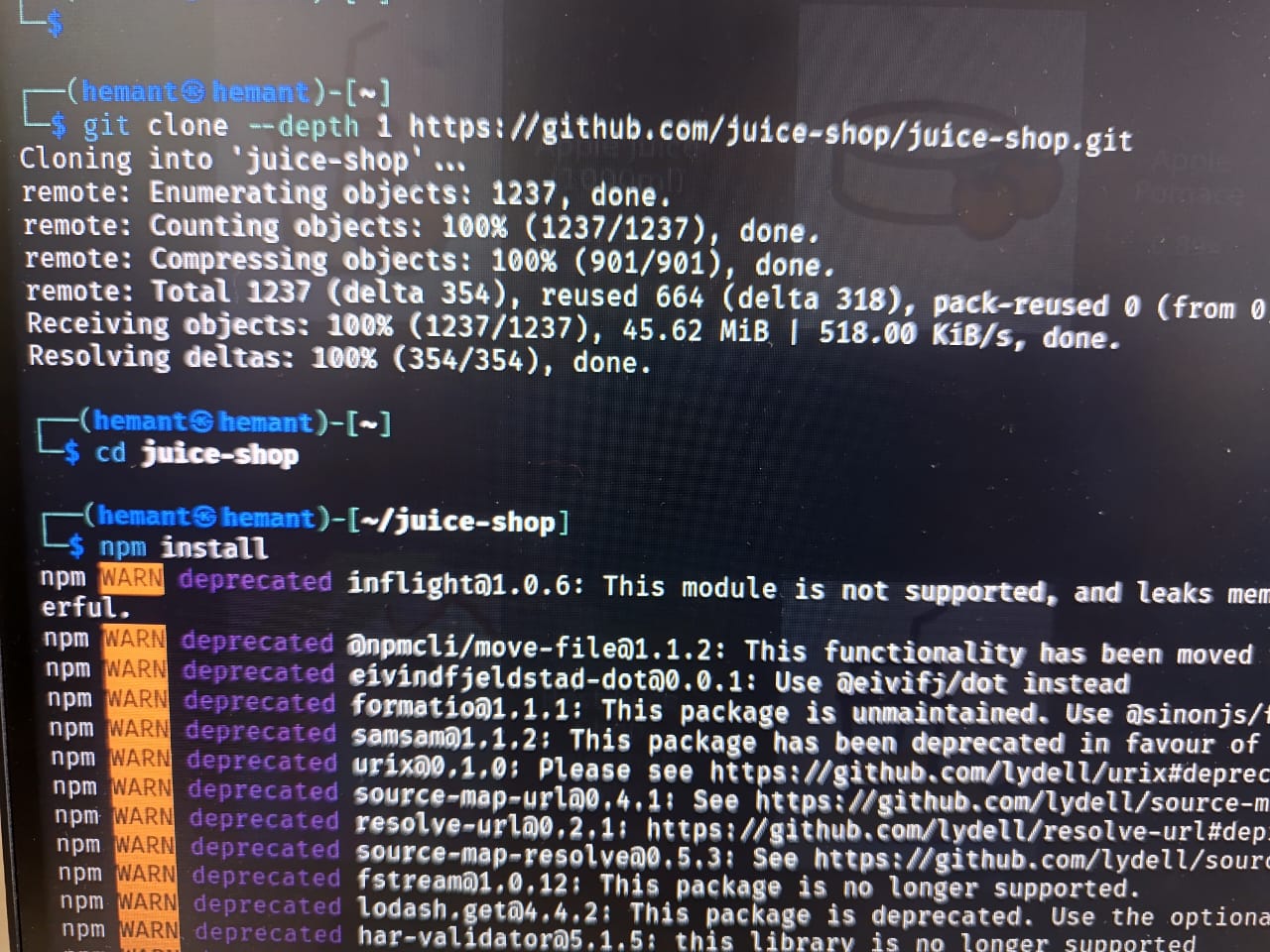
**Setup Kali Linux on Virtual Box as a New Machine then follow the following steps:-**

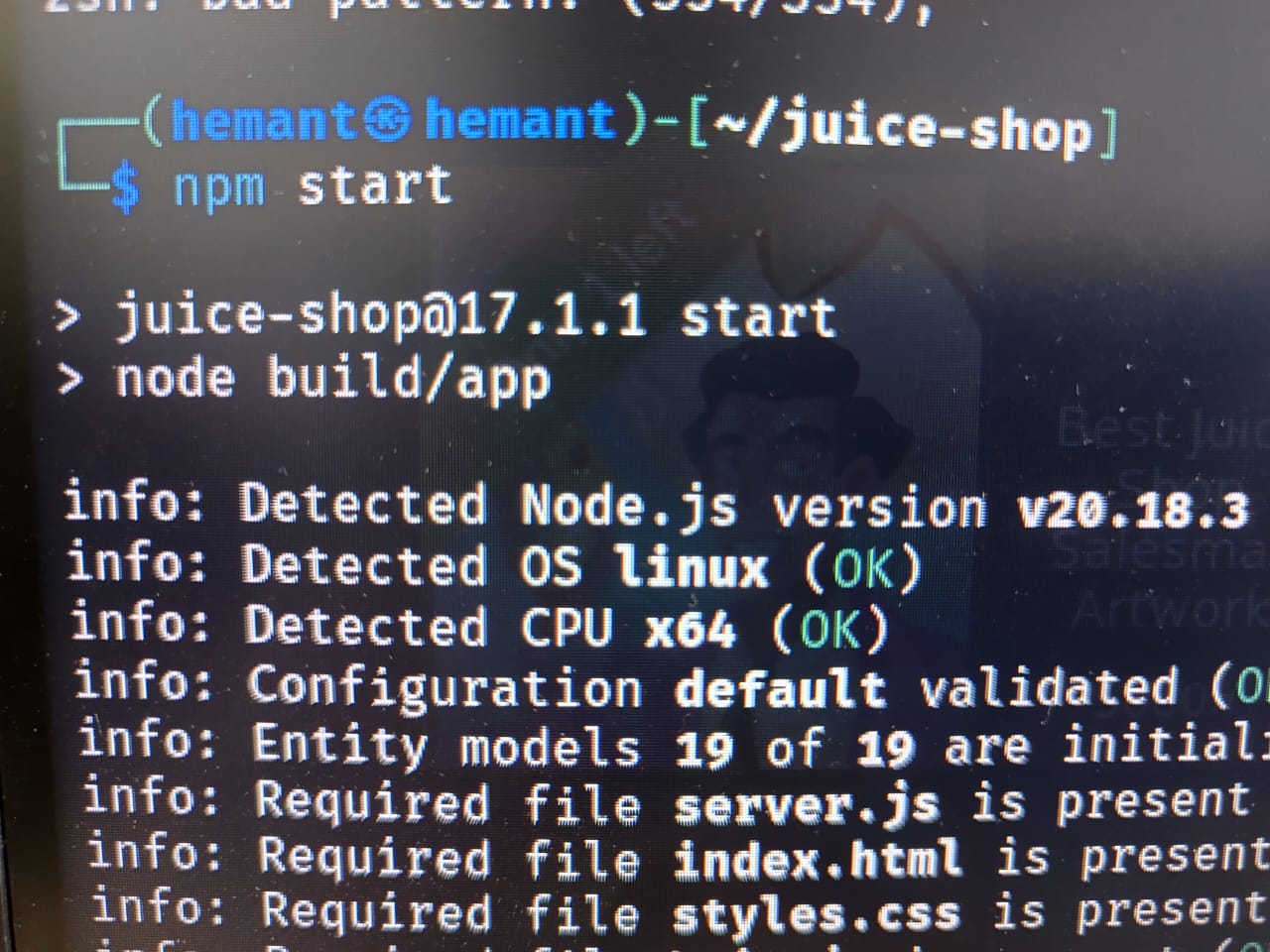


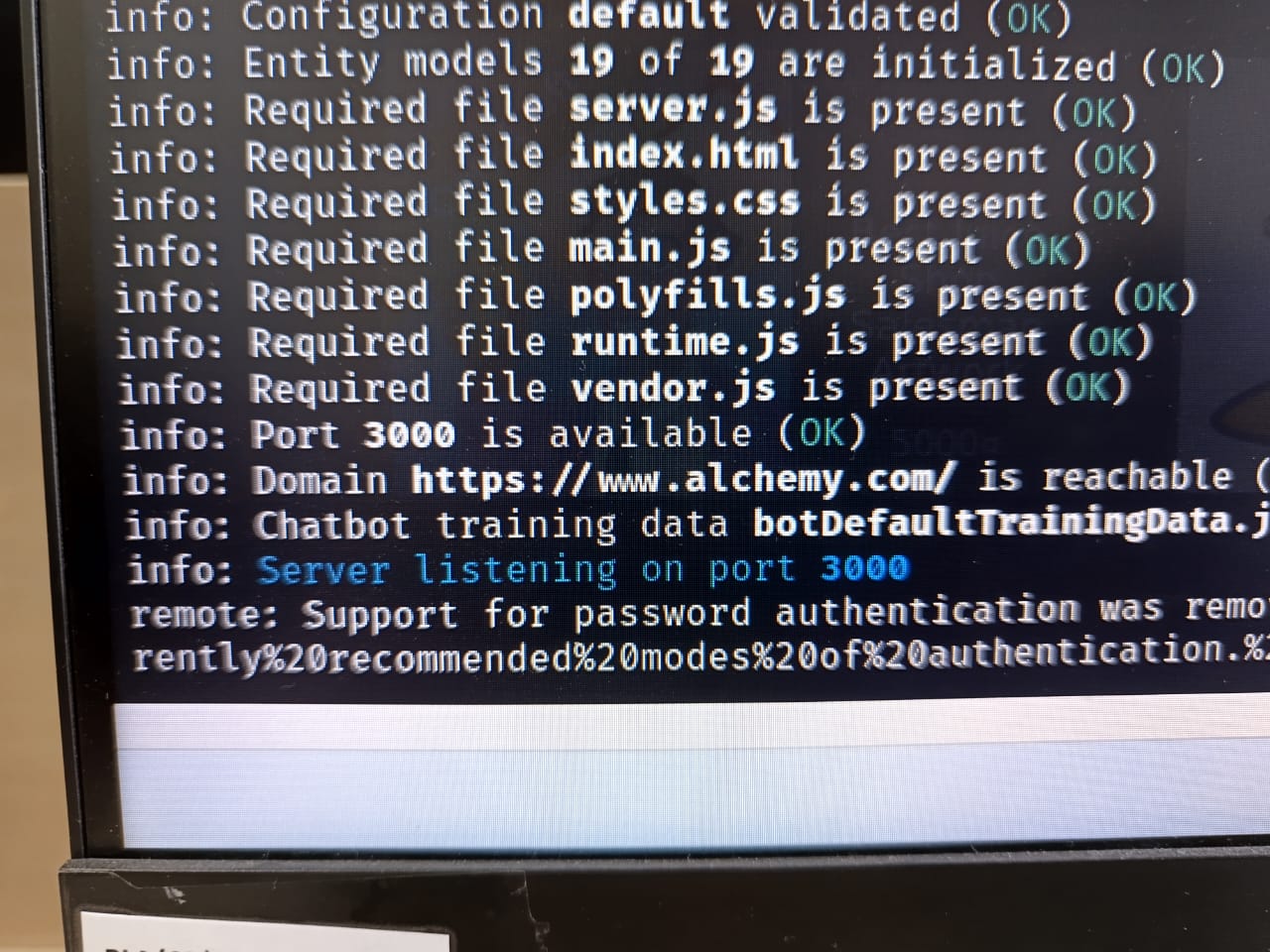


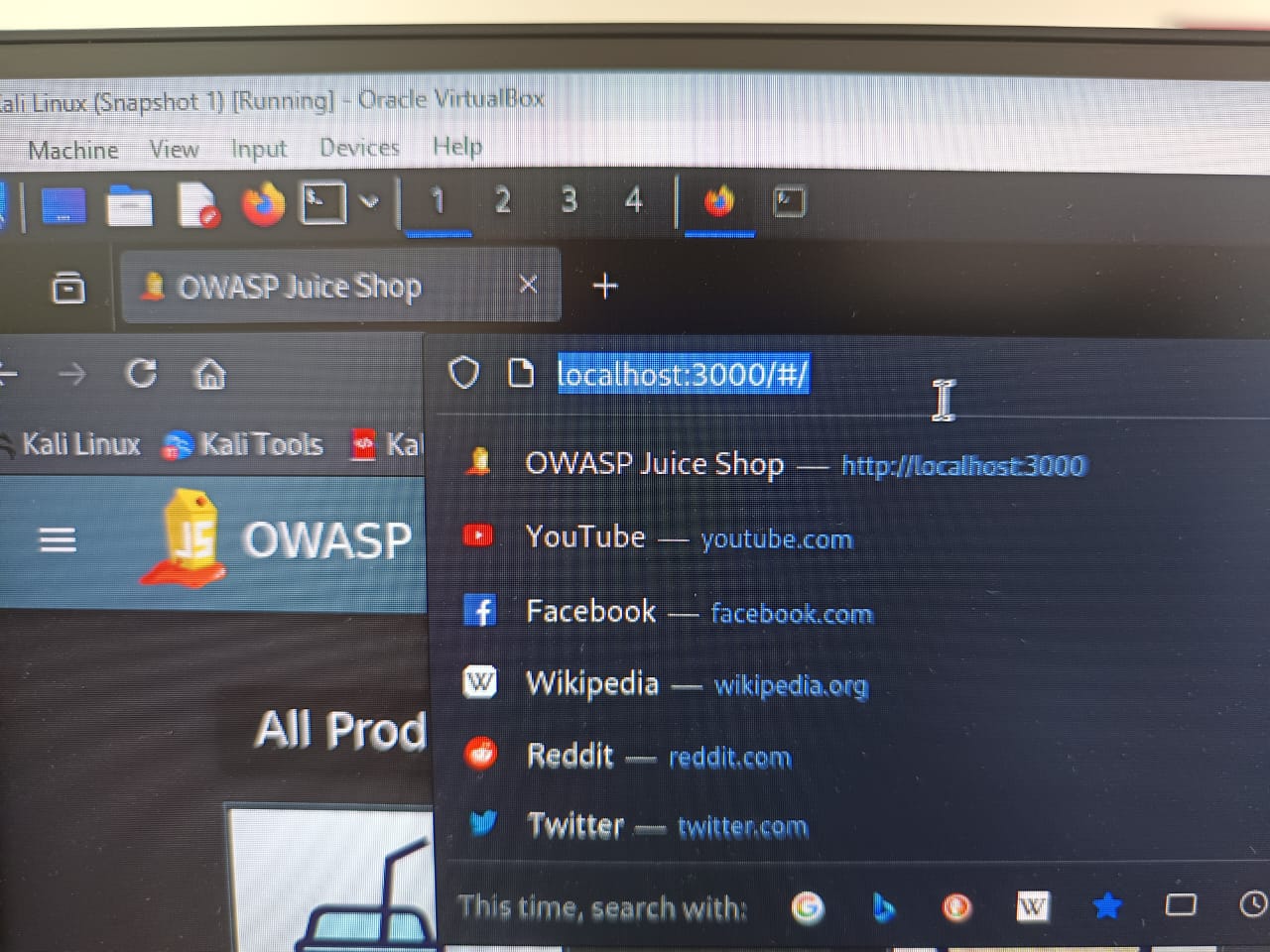


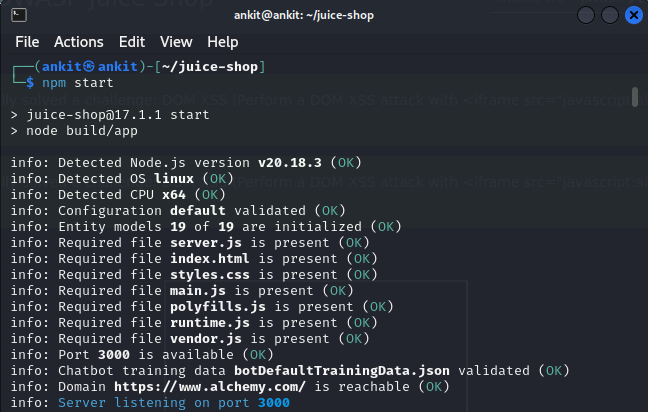






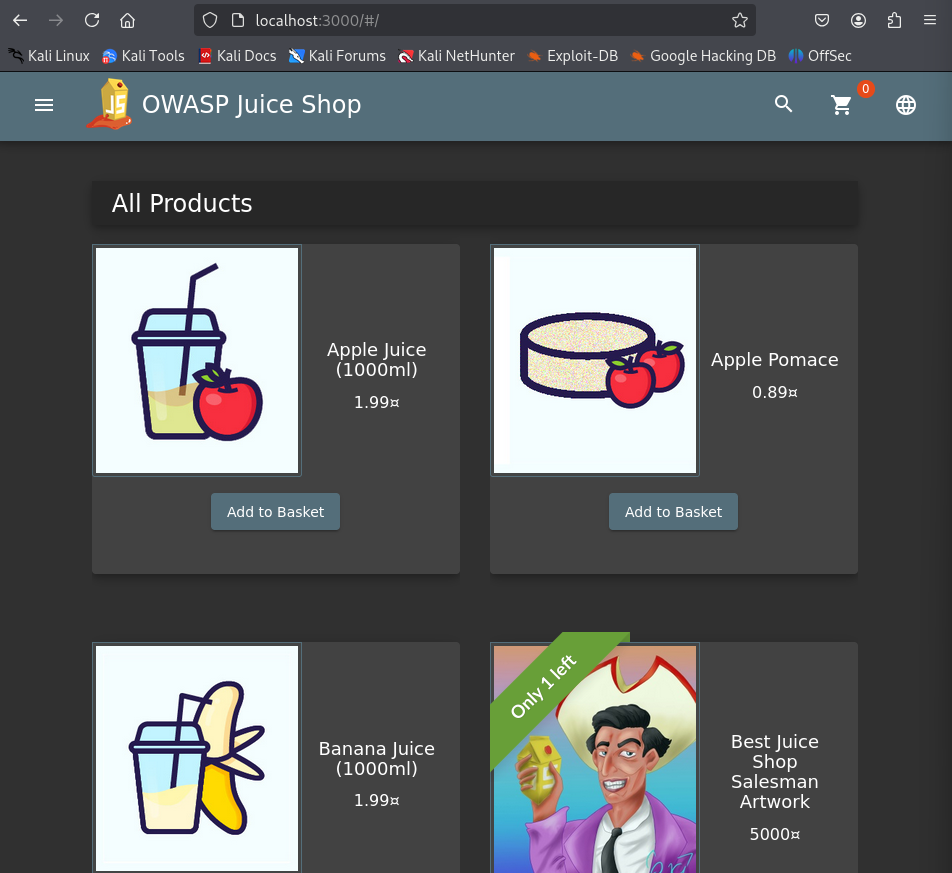






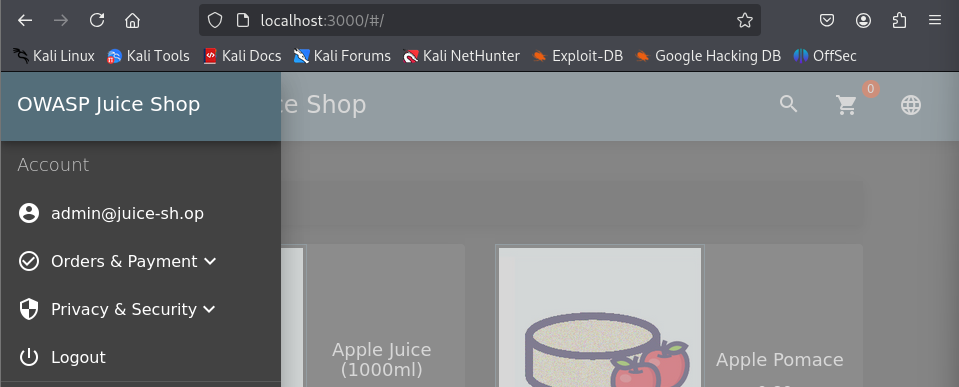
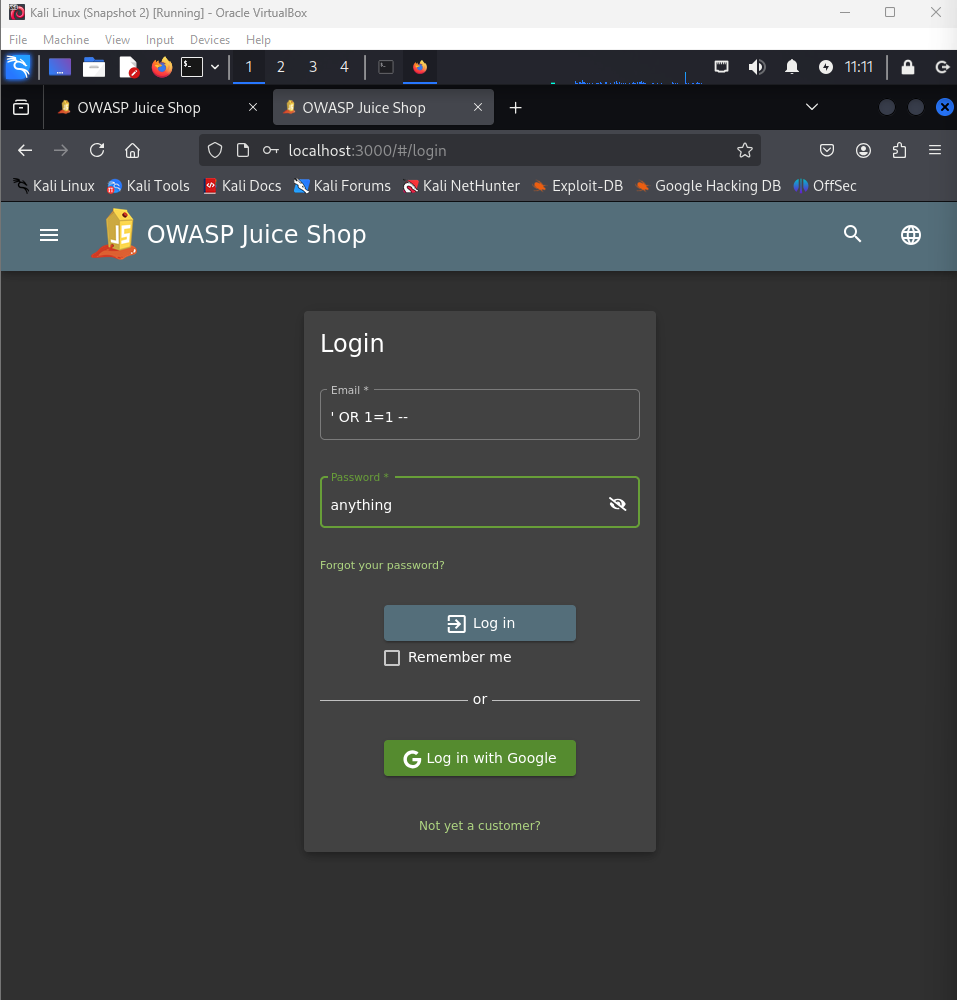
**Now visit the following link using the Mozilla Firefox(or any other available) search engine:-**

****

****

**This page shows that owasp juice-shop is running successfully.**

**Now click on Account button on top right corner of the website and click on Login. Then on the login page type the following text on the Email textbox and type anything you want in the password textbox.**

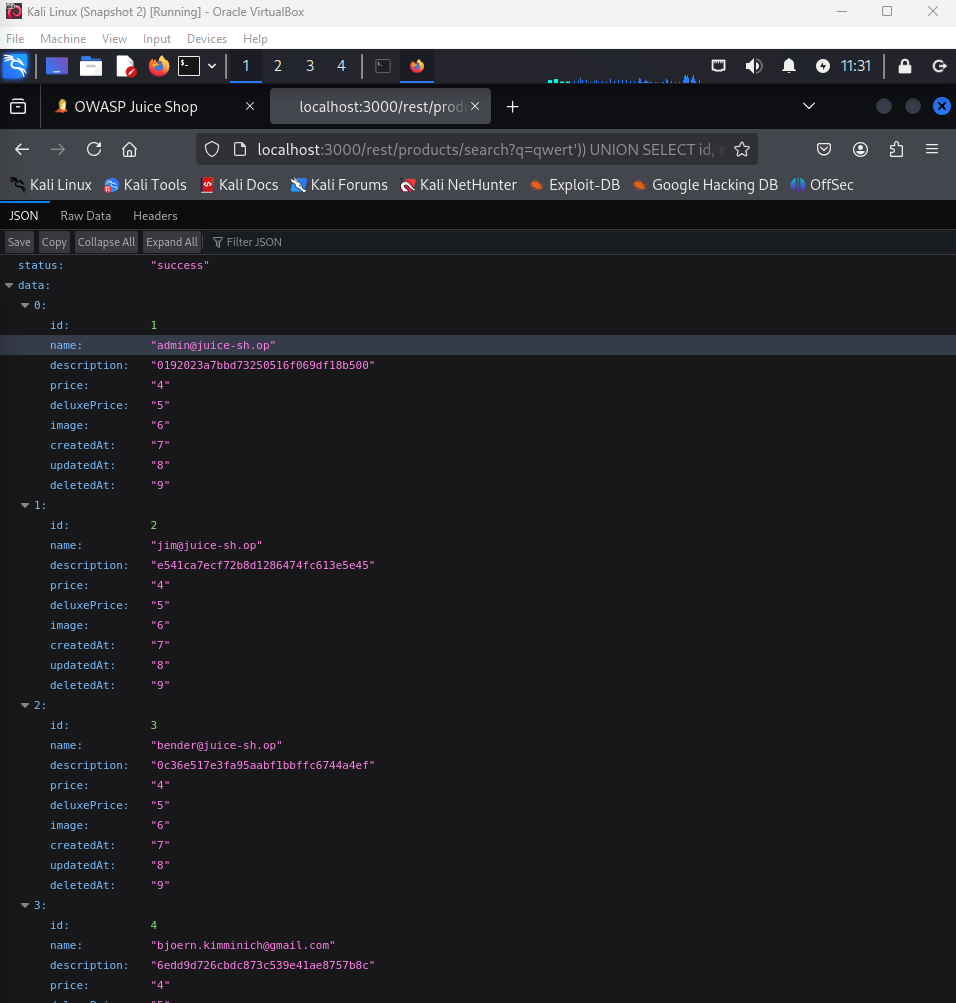
****

**If you see this page after clicking Log in then you have successfully used sql injection to get admin access in owasp.**

**Now write the following in the search bar of Juice Shop:- UNION SELECT id, email, password, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’ FROM Users–**

**OR type the following link in the search engine:-**

[**http://localhost:3000/rest/products/search?q=qwert%27))%20UNION%20SELECT%20id,%20email,%20password,%20%274%27,%20%275%27,%20%276%27,%20%277%27,%20%278%27,%20%279%27%20FROM%20Users--**](http://localhost:3000/rest/products/search?q=qwert%27))%20UNION%20SELECT%20id,%20email,%20password,%20%274%27,%20%275%27,%20%276%27,%20%277%27,%20%278%27,%20%279%27%20FROM%20Users--)

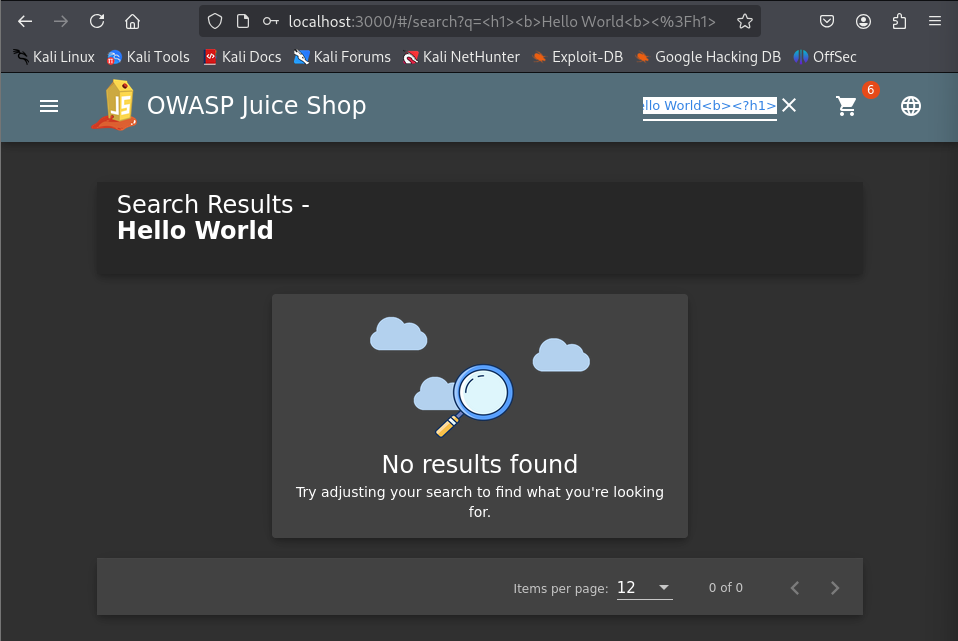
****

**If this page shows up then you have successfully used SQL Injection to get the database containing all user details.**

**Now to check XSS vulnerabilities.**

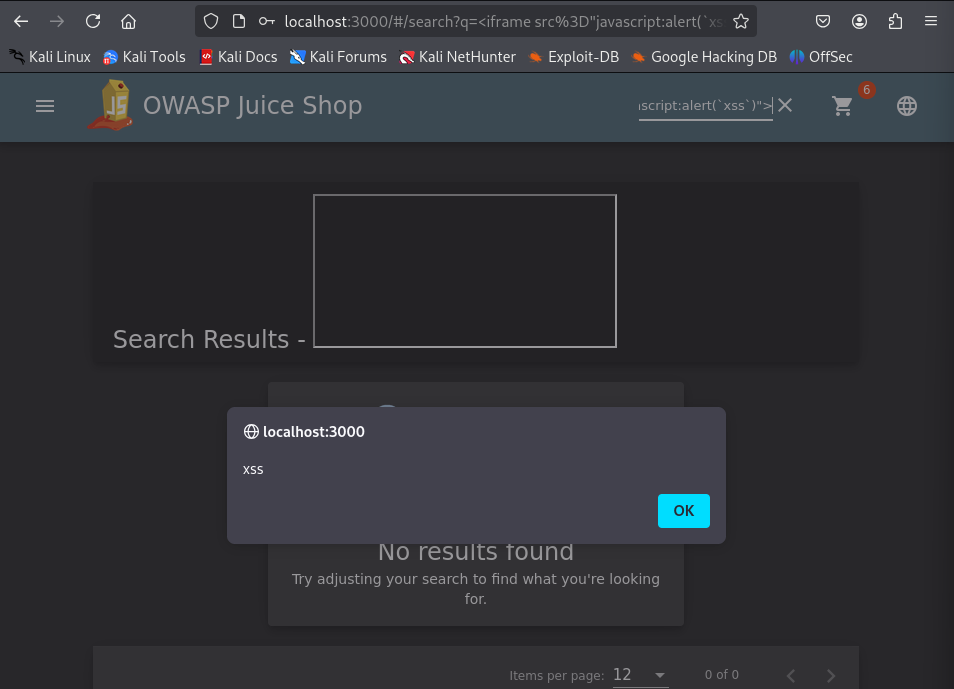
**Write the following command in the search bar of Juice Shop:-**

**<h1><b>Hello World<b></h1>**

****

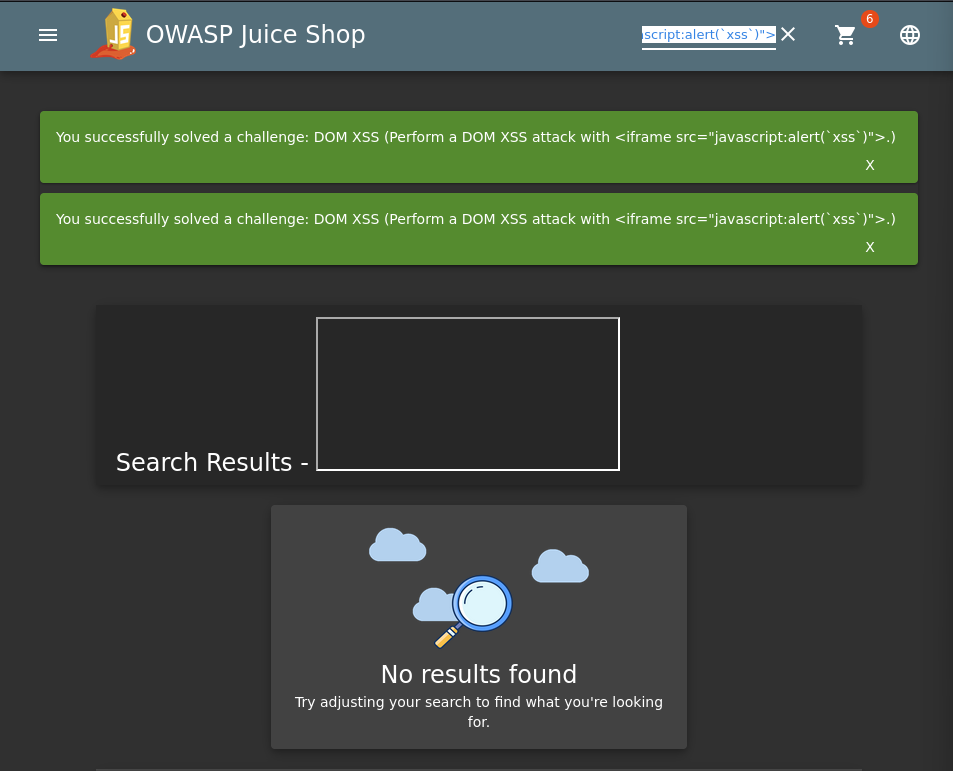
**This shows that we can inject HTML in Juice shop.**

**Next write the following command in the search bar:-**

**<iframe src="javascript:alert(`xss`)">**

**The following popup will show that DOM XSS attack can also be done on Juice Shop.**

**This page after clicking Ok on the popup shows a successful DOM XSS vulnerability.**



**Ethical hacking practical 2**

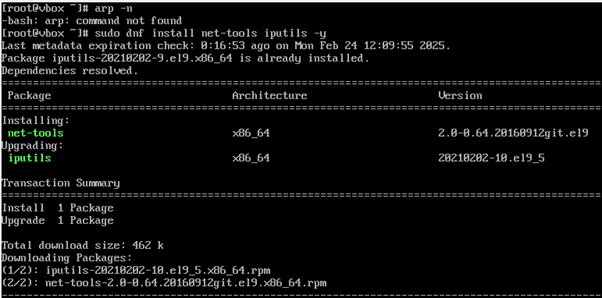
**Q2) Use tools like ipconfig, ifconfig, arp, and ping to explore network configurations on a test VM.Perform network scanning using Nmap on a local network (ensure permission is obtained).**

**You may use Nmap, Command-line tools, Virtual Machines.**

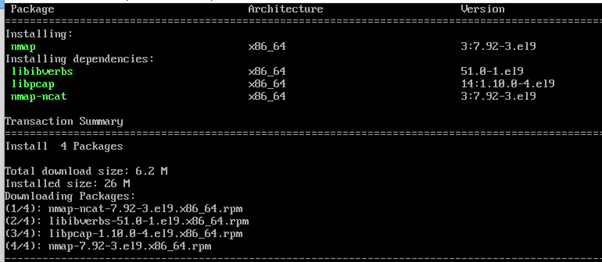
login to your account



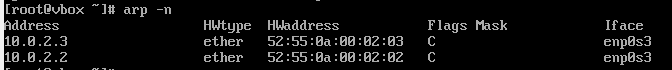
Install nmap





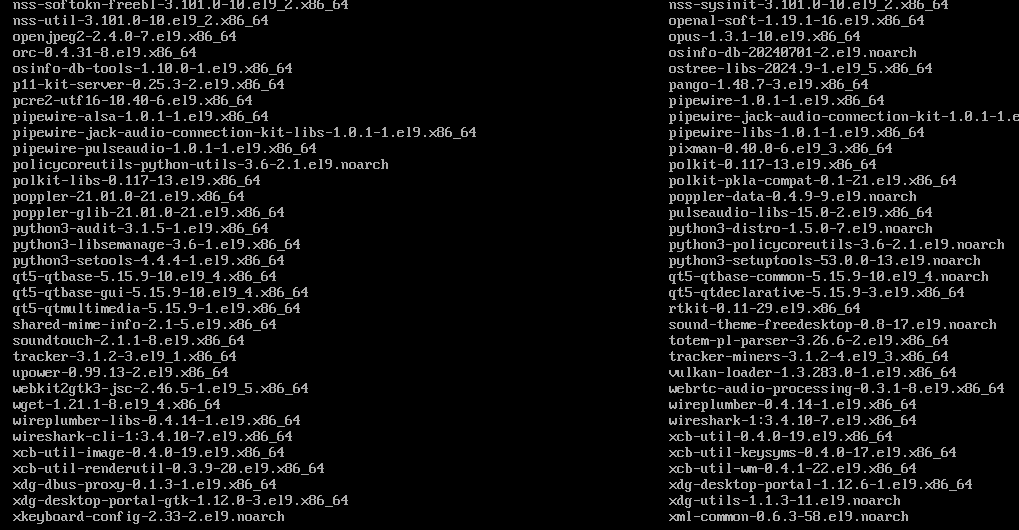


arp

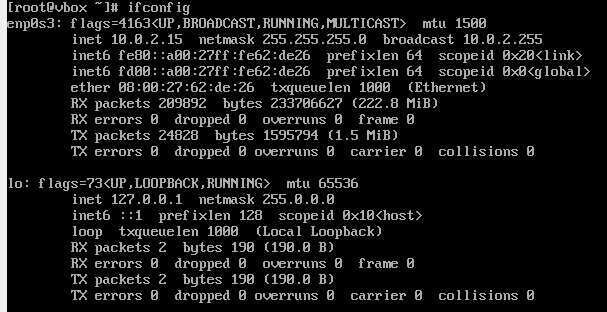


Install wireshark



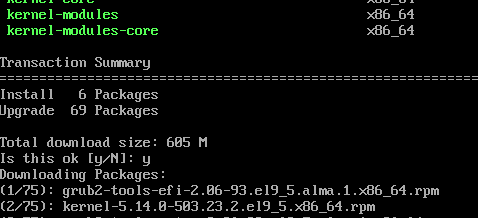


Ifconfig

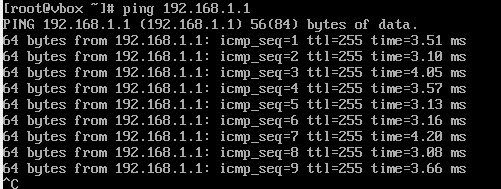


Updating

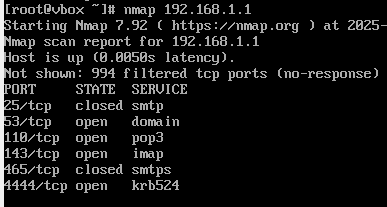




ping



Nmap

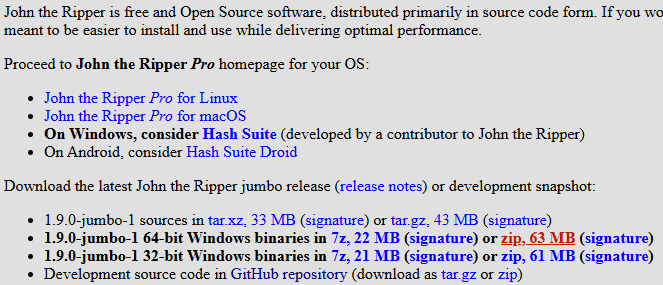


**Q3) Use John the Ripper or Hashcat to crack simple password hashes generated using online tools.**

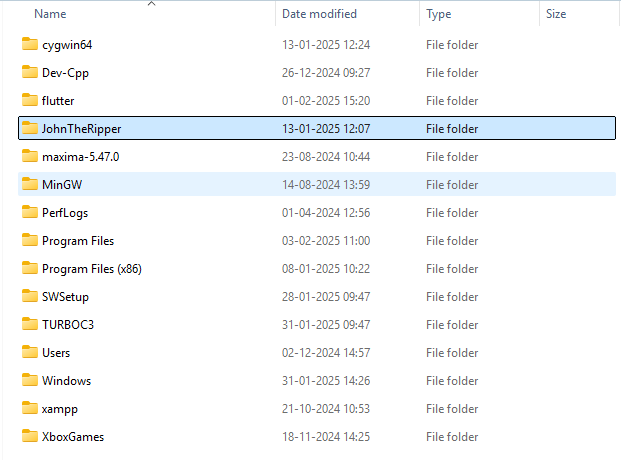
**Practical:-**

**Step1) Download John the Ripper from google from the following website**

<https://www.openwall.com/john/>

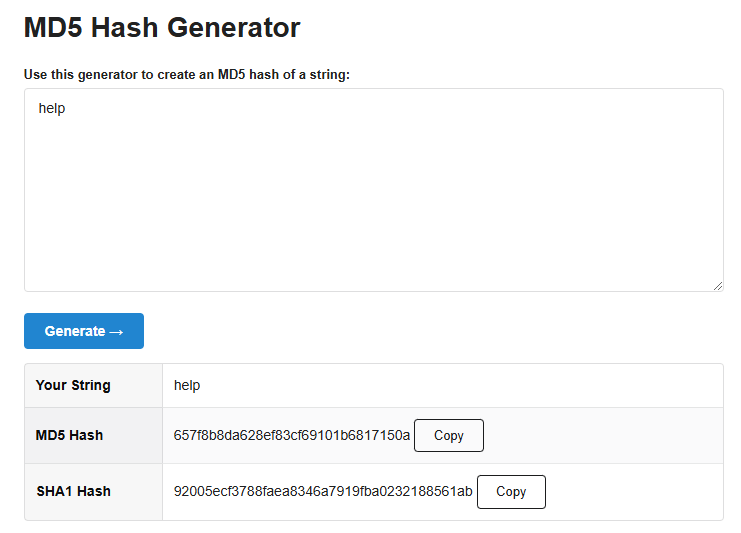


**Step2) Extract the downloaded file and save the file in C drive, then change the name of the file in C drive to JohnTheRipper**

****

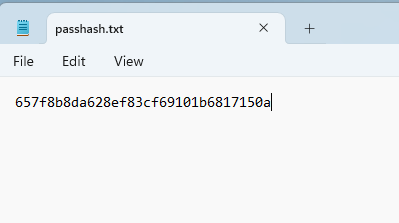
**Step3) Now open an MD5 hash generator on google and generate hash value for any password of length of upto 4 letters.**

<https://www.md5hashgenerator.com/>

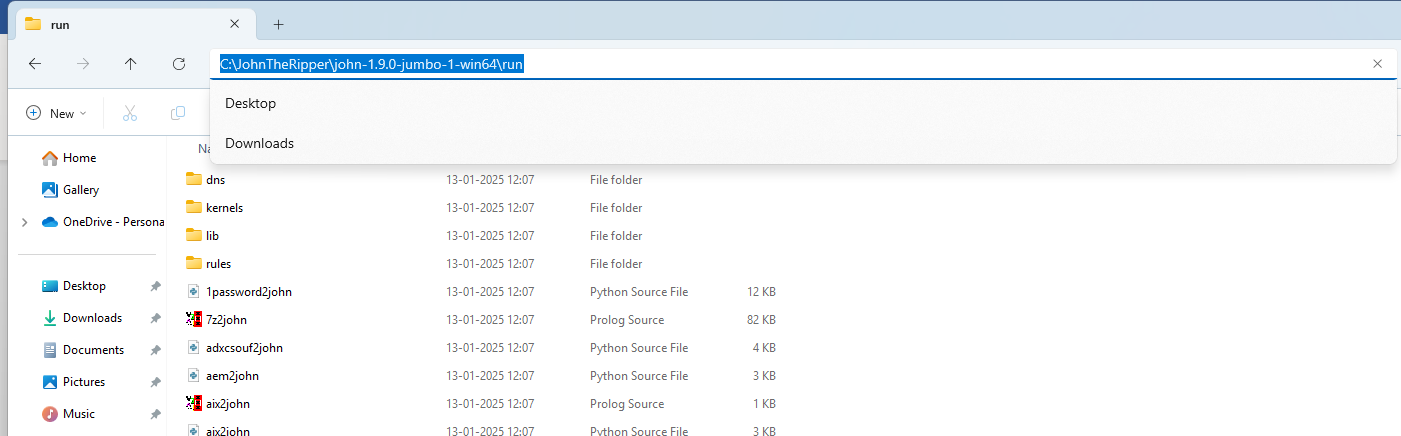


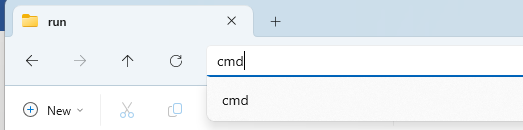
**Step4) Now copy the MD5 hash value**

**Step5) Open a new Notepad file and save the copied MD5 hash value in this file and save the file with any name.**

****

**Step6) Now open C:\JohnTheRipper\john-1.9.0-jumbo-1-win64\run folder and open a CMD by typing cmd on the top in place of the file path name and click Enter.**

****

****

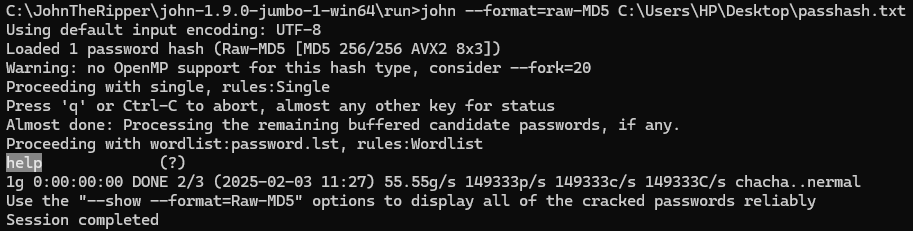
**Step7) In CMD write the following command**

john --format=raw-MD5 “hash\_file\_path”

In my case the command is:- john --format=raw-MD5 C:\Users\HP\Desktop\passhash.txt

****

**Step8) Click Enter**

****

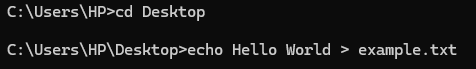
**The cracked password will be visible in the CMD.**

**Q5) Alternate Data Streams (ADS): Practice creating ADS on a Windows VM to hide files. Explore how to detect and counter such techniques.**

**• Tools: Windows VM, Command Prompt**

**Practical:-**

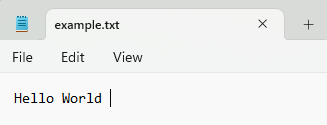
**Create a file using cmd with some text in it using the following command:**

****

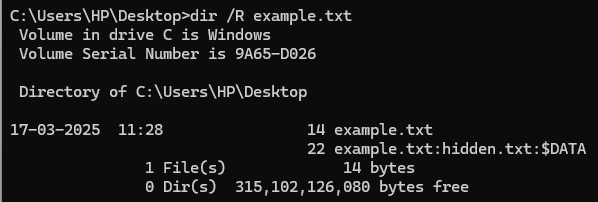
**Now add some hidden text in the file using the following command:**

****

**The file looks like this:**

****

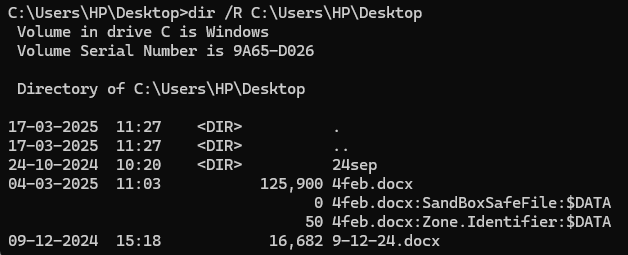
**Now write the following command to list alternate data streams (ADS) of the file with the following output:**

****

**To read the ADS text use the following command:**

****

**To detect ADS using CMD write the following command:**

****

****

**To restrict permissions for non-administrative users to prevent them from creating or modifying ADS write the following command:**

****

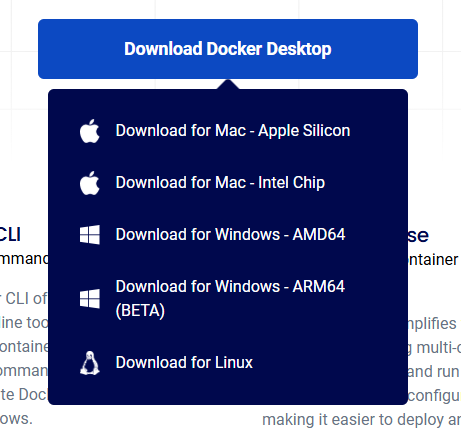
**Q6) SQL Injection (Test Environment): Perform SQL Injection on DVWA or Juice Shop to extract sample data. Understand how to mitigate such vulnerabilities.**

**• Tools: DVWA, OWASP Juice Shop.**

**Setup the test environment**

1. **Download DVWA using docker**

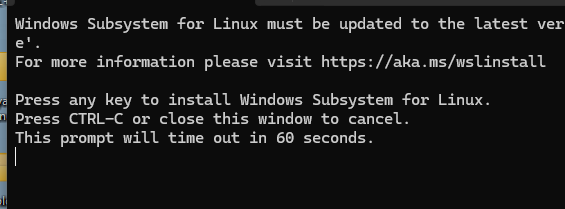
**Download docker from the internet**

****

**Check if docker is installed or not on cmd**

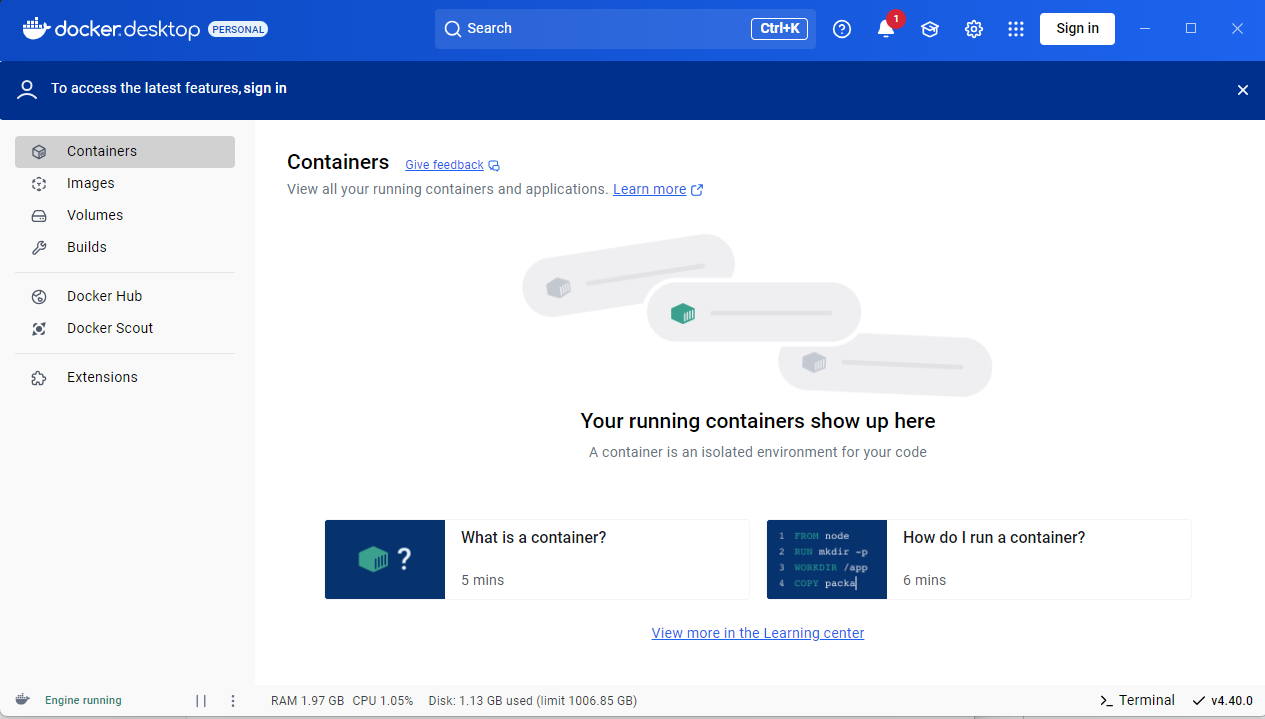
****

**Then run docker application**

****

**Then press any key and install subsystem**

**Then run docker**

****

**Check if docker is running**

****

**Then run this**

**Pull dvwa docker image**

****

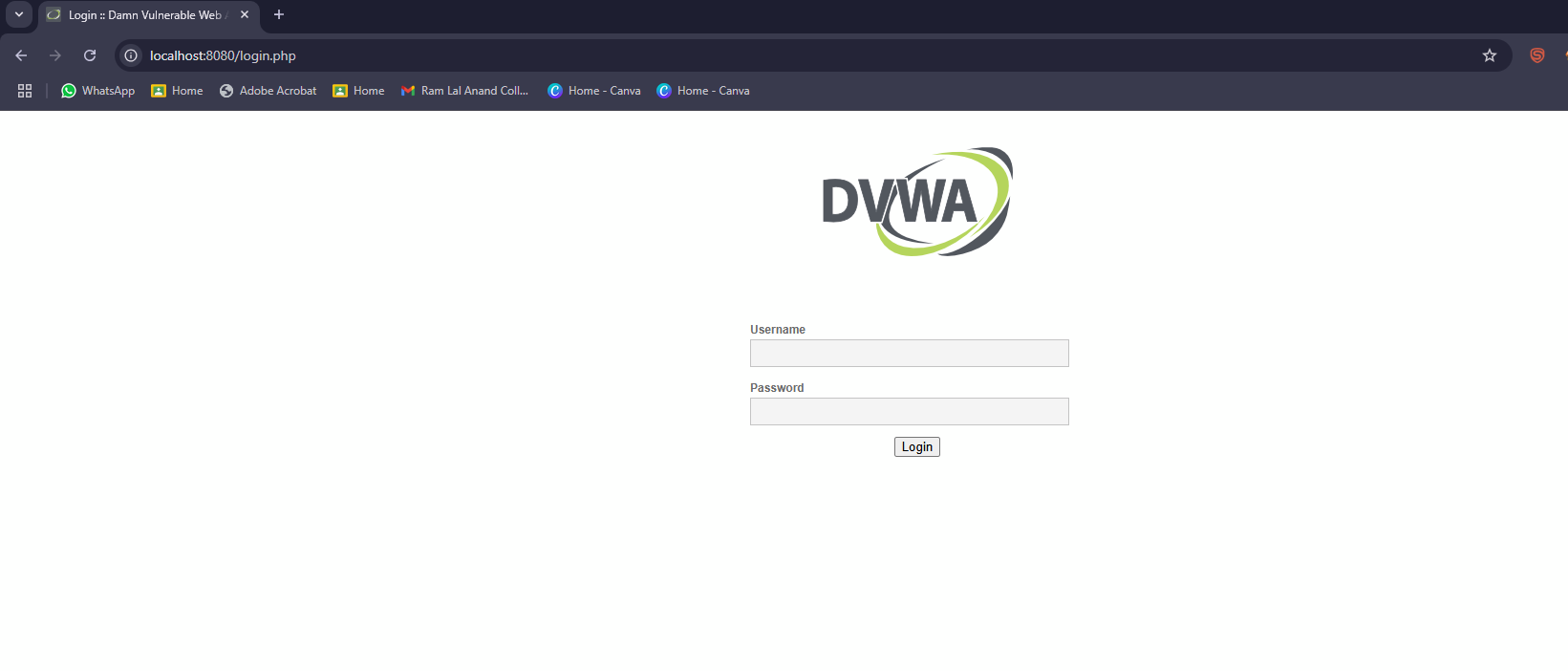
**Run dvwa container**

****

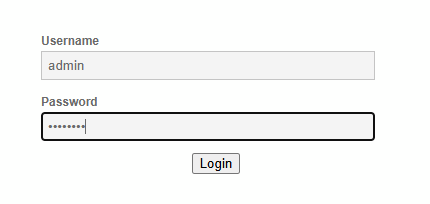
**Then allow backend**

**Then search on chrome**

****

****

**Enter**

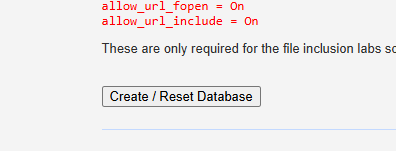
****

**Username - admin**

**Password - password**

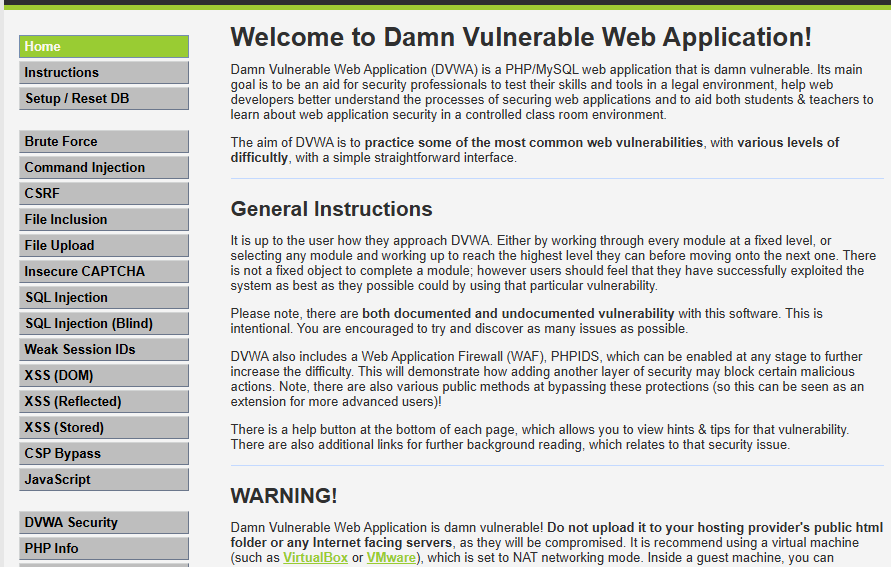
****

**Then**

****

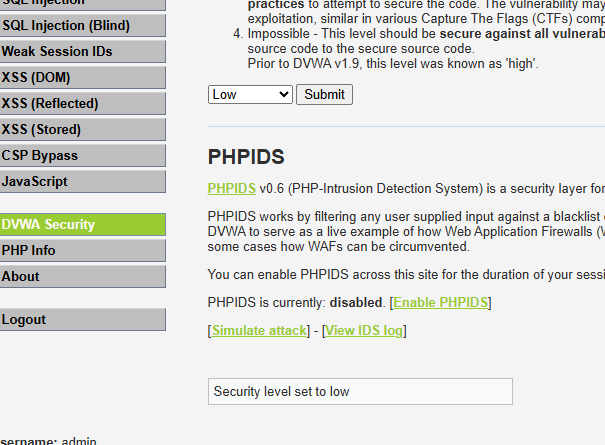
**Click on this create/reset database**

**Then login again**

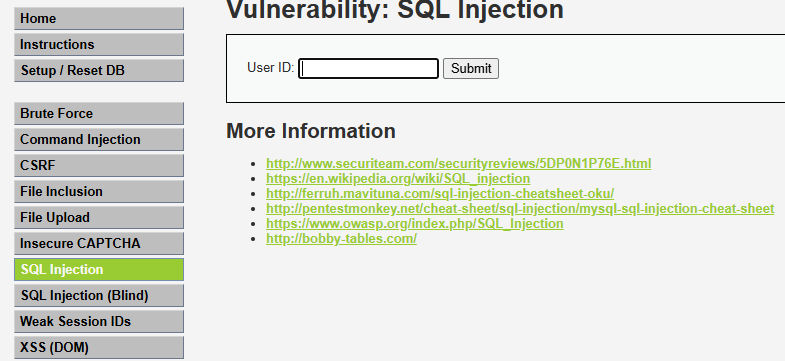
****

**Then**

**Go to dvwa security and submit security as low**

****

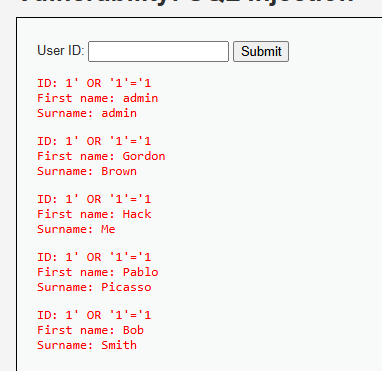
**Now perform sql injection**

****

**Then write**

****

**List of users info is displayed**

****

**SQLi Prevention and Mitigation**

**The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.**

**If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.**

**1. Use Prepared Statements (Parameterized Queries)**

**Example (Python with SQLite):**

**cursor.execute("SELECT \* FROM users WHERE username = ? AND password = ?", (username, password))**

**2. Input Validation and Whitelisting**

* **Validate input for expected format (e.g., integers, emails).**
* **Whitelist known-safe values instead of blacklisting dangerous ones.**

**if (!is\_numeric($input)) {**

**die("ID must be a number.");**

**}**

**Mitigating SQL Injection Attacks**

**OWASP lists the following four methods to mitigate SQL Injection Attacks-**

**1) Use of Prepared Statements (with Parameterized Queries)-** Prepared statements with parameterized queries are one of the most effective ways to prevent SQL injection. This technique involves creating a SQL query template where placeholders (such as ? or named parameters) are used for user inputs, rather than directly inserting user-supplied data into the query. The database engine then treats the user input as data, not executable code, preventing malicious input from altering the query's structure. This ensures that user input cannot inject arbitrary SQL code.

**2) Use of Properly Constructed Stored Procedures-** Stored procedures are precompiled SQL statements stored within the database. By using stored procedures, you can centralize logic on the server side, and avoid directly embedding user inputs into dynamic SQL queries. When constructed properly, stored procedures can protect against SQL injection by using parameters instead of concatenating user input directly into queries. However, it's important to avoid dynamic SQL within stored procedures, as it can still be vulnerable to injection if not handled carefully.

**3) Allow-list Input Validation-** Allow-list input validation (also known as whitelisting) involves checking user input against a set of predefined, acceptable values (such as alphanumeric characters or specific formats). This reduces the risk of malicious input being passed into SQL queries by ensuring that only expected, safe data is accepted. This technique doesn't entirely eliminate the risk of SQL injection but helps mitigate it by reducing the possibility of dangerous input.

**4) STRONGLY DISCOURAGED: Escaping All User Supplied Input-** Escaping user input involves adding escape characters to potentially dangerous characters (like single quotes or semicolons) to neutralize their threat. This was once a common method to prevent SQL injection but is now considered ineffective and error-prone. If you improperly escape input or fail to escape every possible special character, you leave your application vulnerable to injection. Modern database systems and frameworks support safer alternatives like prepared statements, so relying on escaping input is discouraged.

While it can be effective when done carefully, escaping input does not provide the same level of safety as other methods like parameterized queries, and it’s much harder to get right. Therefore, relying on it alone can be risky.

**Q7) Framework Mapping: Map a given security incident to the NIST Cybersecurity Framework or Cyber Kill Chain. Prepare a simple report outlining the stages of the attack.**

**Practical:-**

**Cybersecurity Incident Overview- Domino’s India Data Breach of 2021**

On March 24, 2021, one of the most popular pizza chains of the world **Domino’s Pizza** suffered a massive cyber attack in India where 18 crore order details were stolen and put up for sale on the dark web by hackers. The leaked data included sensitive information like name, mobile number, email address, and GPS location of users.

**Mapping to NIST Cybersecurity Framework**

**1) Identify (Risk Management, Asset Management)-** Lack of strong data security policies for stored customer data, inadequate third-party security audits.

**Evidence:** Domino's India acknowledged a security incident but did not provide specifics about their data security policies or audit practices.

**2) Protect (Data Security, Access Control)-** Possible weak database security (misconfigurations, unencrypted data), inadequate API security controls.

**Evidence:** While the breach involved unauthorized access to customer data, the exact vulnerabilities exploited (e.g., database misconfigurations or unsecured APIs) were not publicly disclosed.

**3) Detect (Security Monitoring, Anomalies & Events)-** The breach was discovered externally (on the dark web), indicating lack of real-time breach detection.

**Evidence:** The breach became public when hackers advertised the stolen data on the dark web, suggesting that Domino's India may not have detected the intrusion internally.

**4) Respond Function (Incident Analysis, Mitigation)-** Engaged global forensic experts post-breach, lodged cybercrime complaints, public disclosure was delayed.

**Evidence:** Domino's India stated they were investigating the incident with the help of experts and had reported the matter to authorities.

**5) Recover Function (Recovery Planning, Security Improvements)-** Strengthened security infrastructure, enhanced data access controls, and collaborated with cybersecurity professionals.

**Evidence:** Specific post-breach security enhancements were not detailed in public statements.

**Conclusion**

In conclusion, while certain aspects of the breach were documented, specific technical details about the vulnerabilities exploited and the organization's internal detection capabilities have not been publicly disclosed.