

LEWIS KAMAU NDERU

lewiskamau@gmail.com / kamau2325@yahoo.com

CS-SA10-25029

Link: <https://academy.hackthebox.com/module/75/section/763>

INTRODUCTION TO WEB APPLICATIONS MODULE

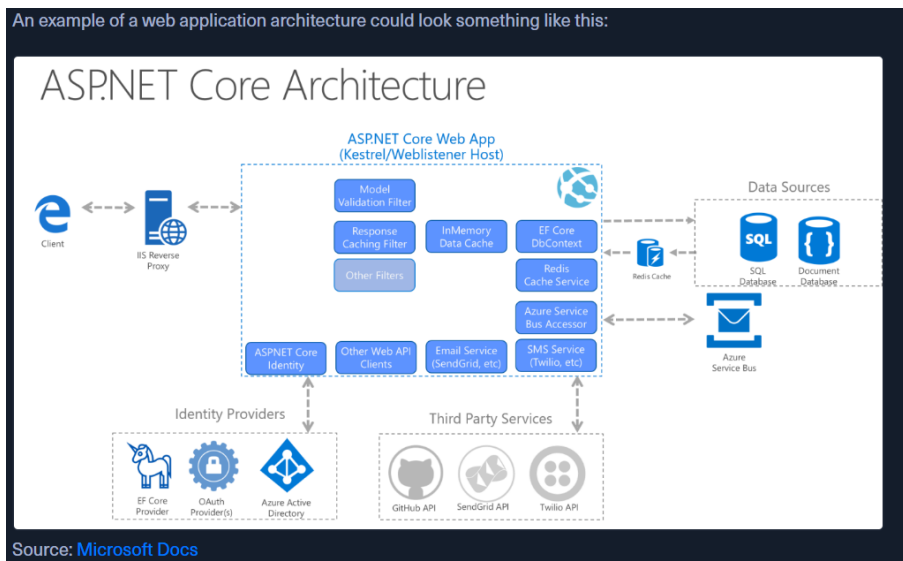
1. INTRODUCTION

Web applications are dynamic, interactive programs that operate on a client-server model, with front-end interfaces running in browsers and back-end processes on servers, enabling real-time functionality and global accessibility. Unlike static websites (Web 1.0), web applications (Web 2.0) offer modular, platform-independent features without requiring installation, though they face security risks like SQL injection or file upload vulnerabilities due to their vast attack surface. Organizations must prioritize penetration testing and secure coding to mitigate threats, as web app flaws can lead to data breaches or system compromises.

2. WEB APPLICATION LAYOUT

Web applications vary widely in design, infrastructure, and architecture, tailored to different business needs and audiences. Common setups range from simple single-server models, which have a high-risk if compromised, to distributed architectures, that have enhanced security and redundancy. Modern approaches like **microservices**, modular, reusable functions, and **serverless computing**, cloud-managed infrastructure, improve scalability and flexibility.

Security risks arise from poor access control or architectural flaws, emphasizing the need for penetration testing and secure design throughout development. Web application layout is crucial for effective security assessments and vulnerability mitigation.

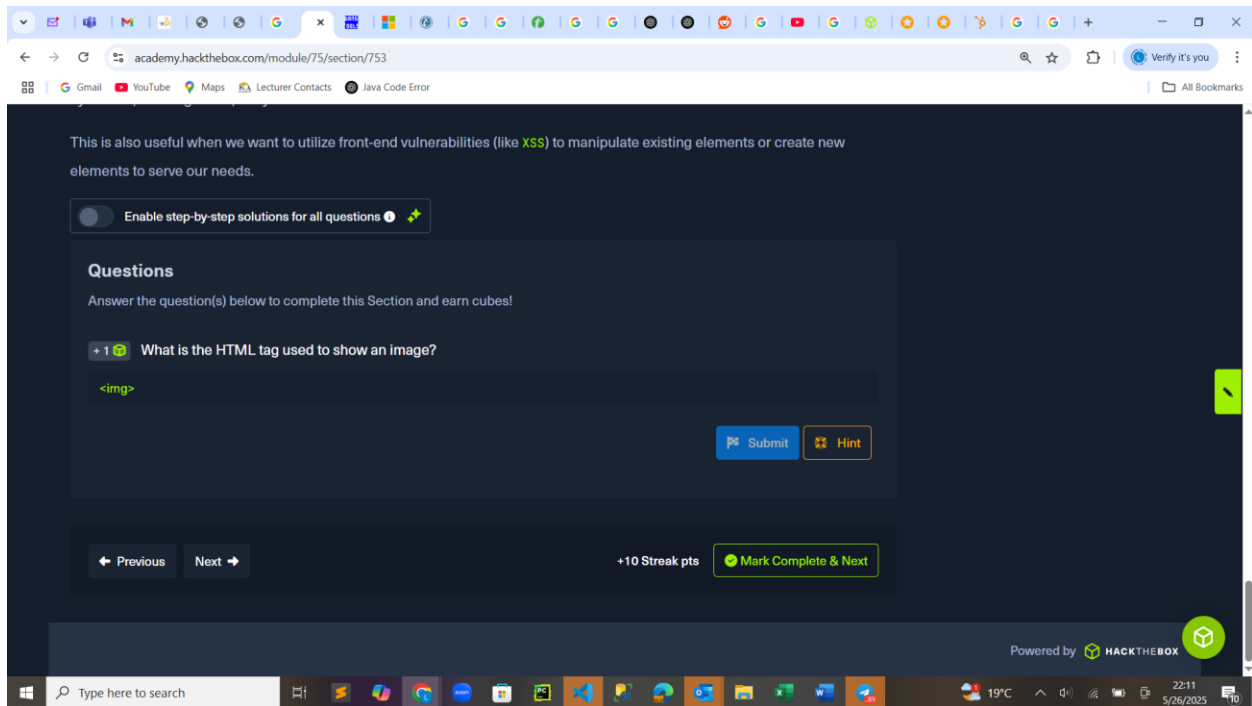


3. Front End vs. Back End

Web applications consist of front-end (client-side) and back-end (server-side) components. The front end, built with HTML/CSS/JavaScript, handles what users see and interact with, while the back end processes data and business logic using servers, databases, and frameworks.

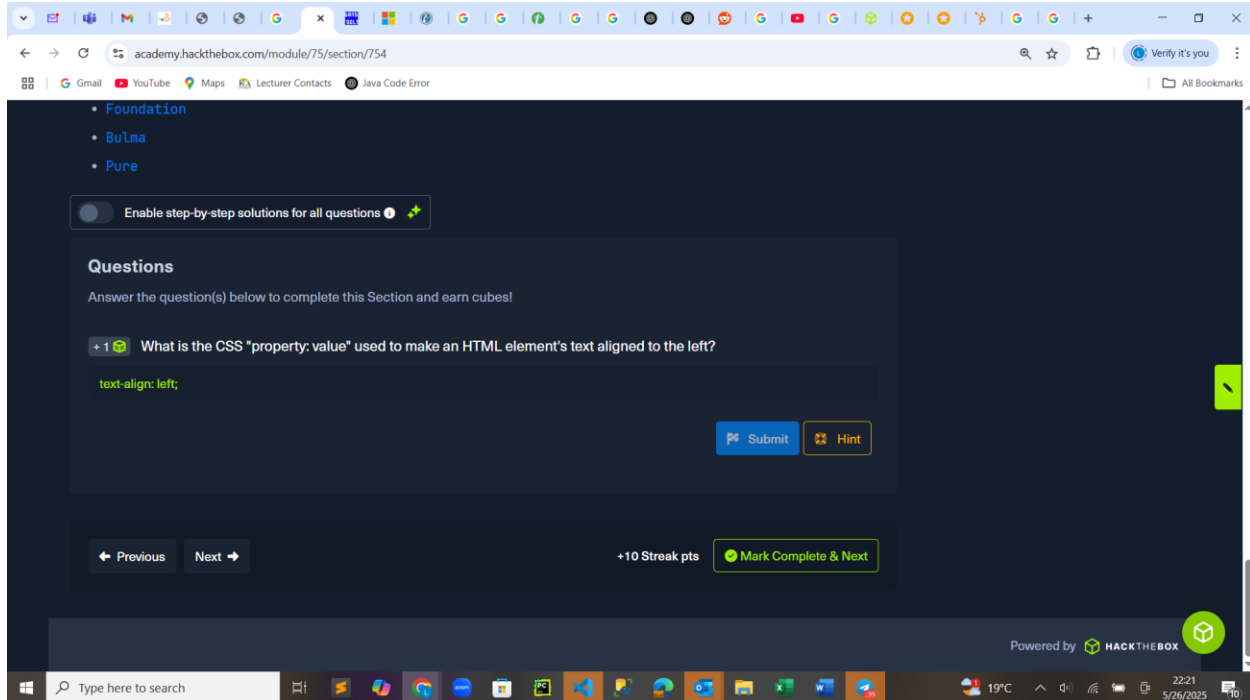
Both layers face security risks like XSS for front end and SQL injection for back end, often faced because of common developer mistakes like permitting invalid data entry, plain text password storage, weak passwords, unencrypted data storage, excessive client-side dependence, hard-coded backdoor accounts, and unverified SQL injections. These mistakes directly enable OWASP's top 10 vulnerabilities that include: Broken Access Control, Cryptographic Failures, Injection flaws, Insecure Design, Security Misconfigurations, Vulnerable Components, Identification and Authentication Failures, Software Integrity Issues, Insufficient Monitoring, and Server-Side Request Forgery.

4. HTML



5. CSS

CSS (Cascading Style Sheets) is the stylesheet language used alongside HTML to format and set the style of HTML elements. Like HTML, there are several versions of CSS, and each subsequent version introduces a new set of capabilities that can be used for formatting HTML elements. Browsers are updated alongside it to support these new features.



6. JAVASCRIPT

JavaScript is a programming language primarily used for web development. It runs in web browsers to add interactivity and dynamic behavior to websites. Unlike HTML (for structure) and CSS (for styling), JavaScript handles functionality like form validation, content updates, and user interactions.

Developing complex web applications with pure JavaScript can be inefficient. JavaScript frameworks simplify this process by providing reusable libraries and tools for common features like user authentication and dynamic content rendering.

7. SENSITIVE DATA EXPOSURE

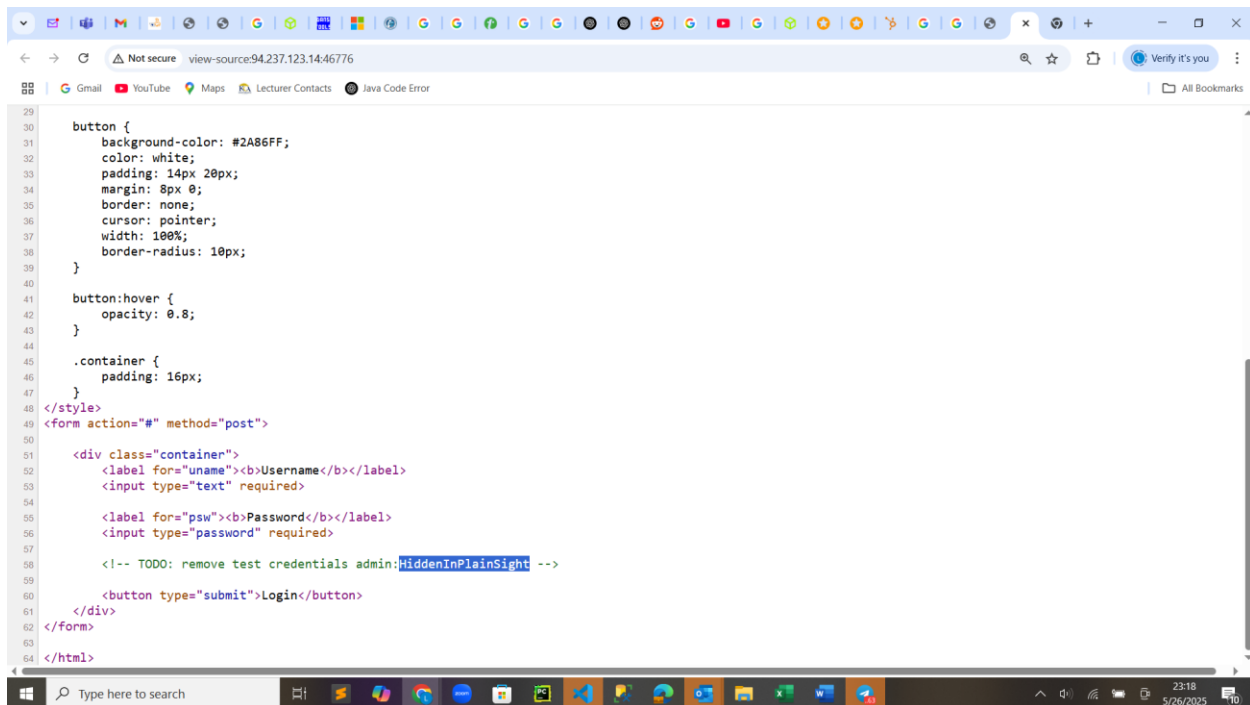
Front-end components execute client-side and don't directly threaten the core back-end systems, but they can endanger end-users when vulnerabilities exist. Attacks exploiting front-end weaknesses may lead to unauthorized access, sensitive data exposure, or service disruption.

During penetration testing, examining page source code (accessible via right-click or Ctrl+U) is critical, as developers sometimes leave sensitive information like test credentials ("test:test" in HTML comments), hidden directories, or debugging parameters exposed.

While back-end testing remains primary, front-end vulnerabilities shouldn't be overlooked since they might provide access to admin functionality or server compromise pathways. Prevention requires removing unnecessary code/comments, classifying data types to control client-side exposure, and implementing code obfuscation to hinder automated scanning tools. Regular code reviews should verify no sensitive data remains in front-end components before deployment.

Check the above login form for exposed passwords. Submit the password as the answer.

spawned the machine and used the IP address and port to access the login page via a browser. Navigated to the source file using ctrl + U and the password was included in a comment on the HTML page as shown below.



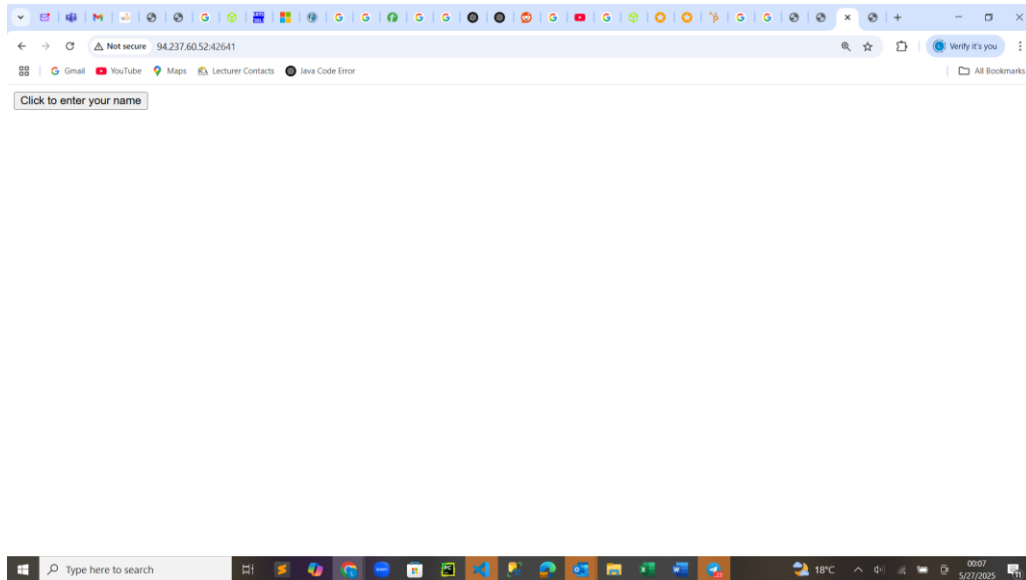
```
29
30 button {
31   background-color: #2A86FF;
32   color: white;
33   padding: 14px 20px;
34   margin: 8px 0;
35   border: none;
36   cursor: pointer;
37   width: 100%;
38   border-radius: 10px;
39 }
40
41 button:hover {
42   opacity: 0.8;
43 }
44
45 .container {
46   padding: 16px;
47 }
48 </style>
49 <form action="#" method="post">
50
51   <div class="container">
52     <label for="uname"><b>Username</b></label>
53     <input type="text" required>
54
55     <label for="psw"><b>Password</b></label>
56     <input type="password" required>
57
58     <!-- TODO: remove test credentials admin:HiddenInPlainSight -->
59
60     <button type="submit">Login</button>
61   </div>
62 </form>
63
64 </html>
```

8. HTML INJECTION

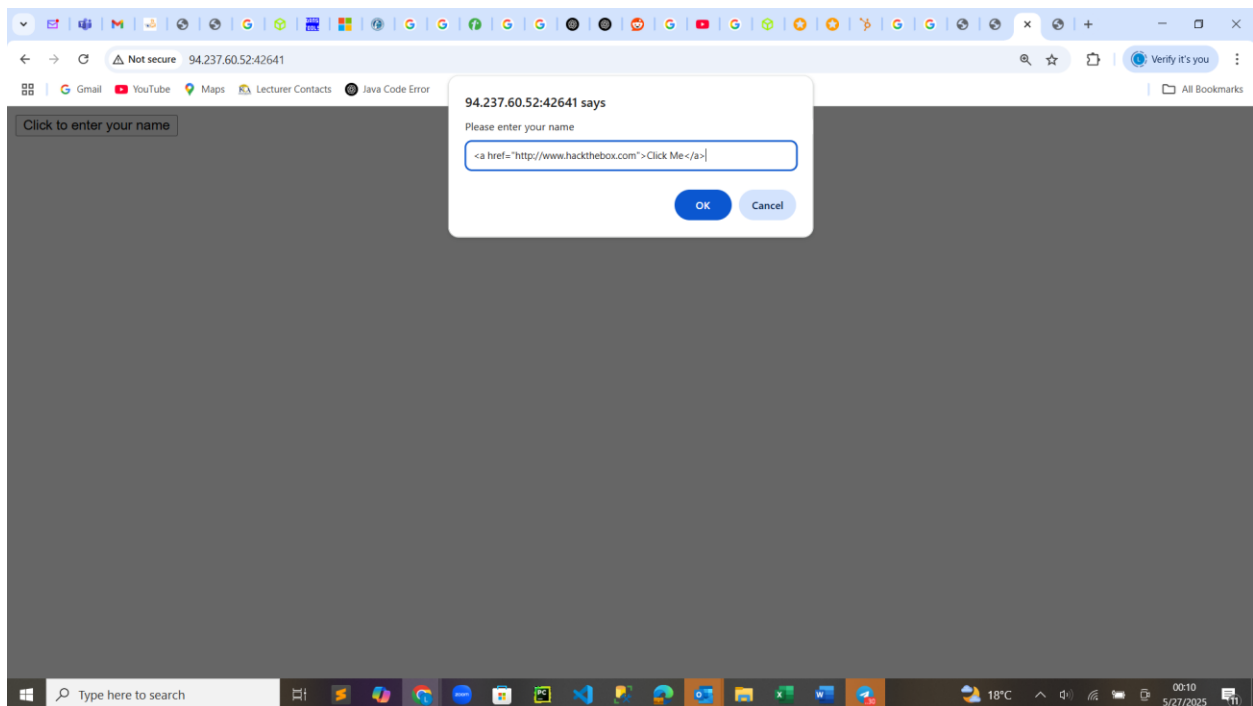
Front-end input validation is equally important as back-end validation since some user input is processed entirely on the client-side. When applications fail to properly sanitize this input, HTML injection vulnerabilities emerge, allowing attackers to inject and execute malicious code directly in the page. This can lead to several serious consequences including the insertion of fake login forms to harvest credentials, complete webpage defacement through injected HTML/CSS modifications, and potential cross-site scripting (XSS) attacks.

What text would be displayed on the page if we use the following payload as our input: `Click Me`

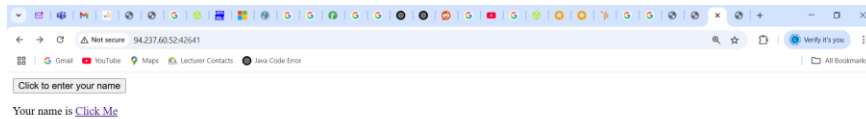
After spawning the machine, I copied the IP address and port in a new tab and a page with button appears as shown below:



Clicked on the button and on the textfield that appears, I input the HTML tag as the question instructs:



A line is displayed and it says “**Your name is Click Me**” whereby Click Me is a clickable link, And that was the answer



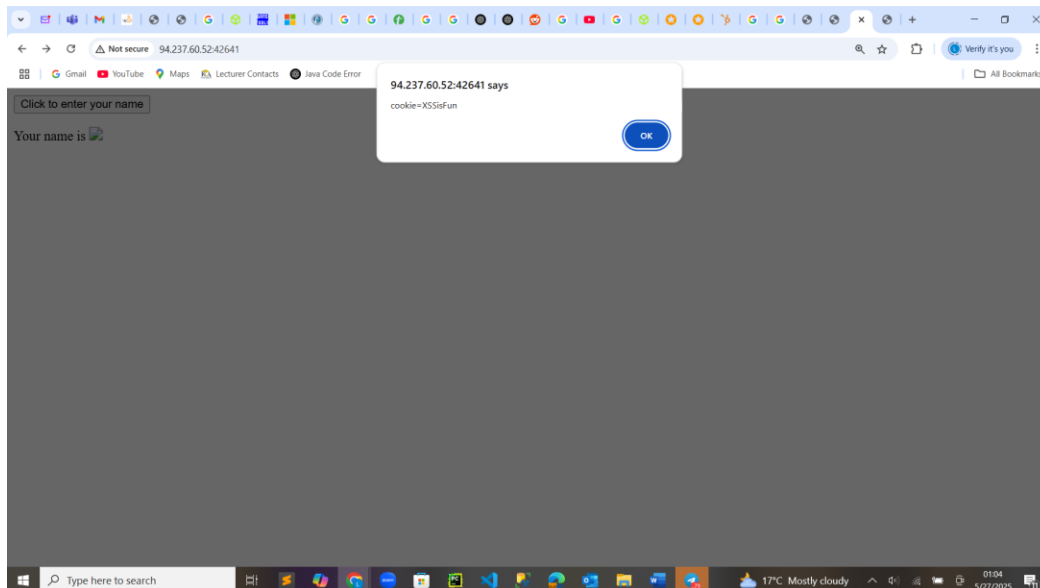
9. CROSS-SITE SCRIPTING (XSS)

HTML Injection vulnerabilities can often be utilized to also perform Cross-Site Scripting attacks by injecting JavaScript code to be executed on the client-side. Once we can execute code on the victim's machine, we can potentially gain access to the victim's account or even their machine.

XSS is very similar to HTML Injection in practice. However, XSS involves the injection of JavaScript code to perform more advanced attacks on the client-side, instead of merely injecting HTML code.

Try to use XSS to get the cookie value in the above page

On the previous question's text field, I injected the javascript code: `` then a window with the cookie appeared as shown below:



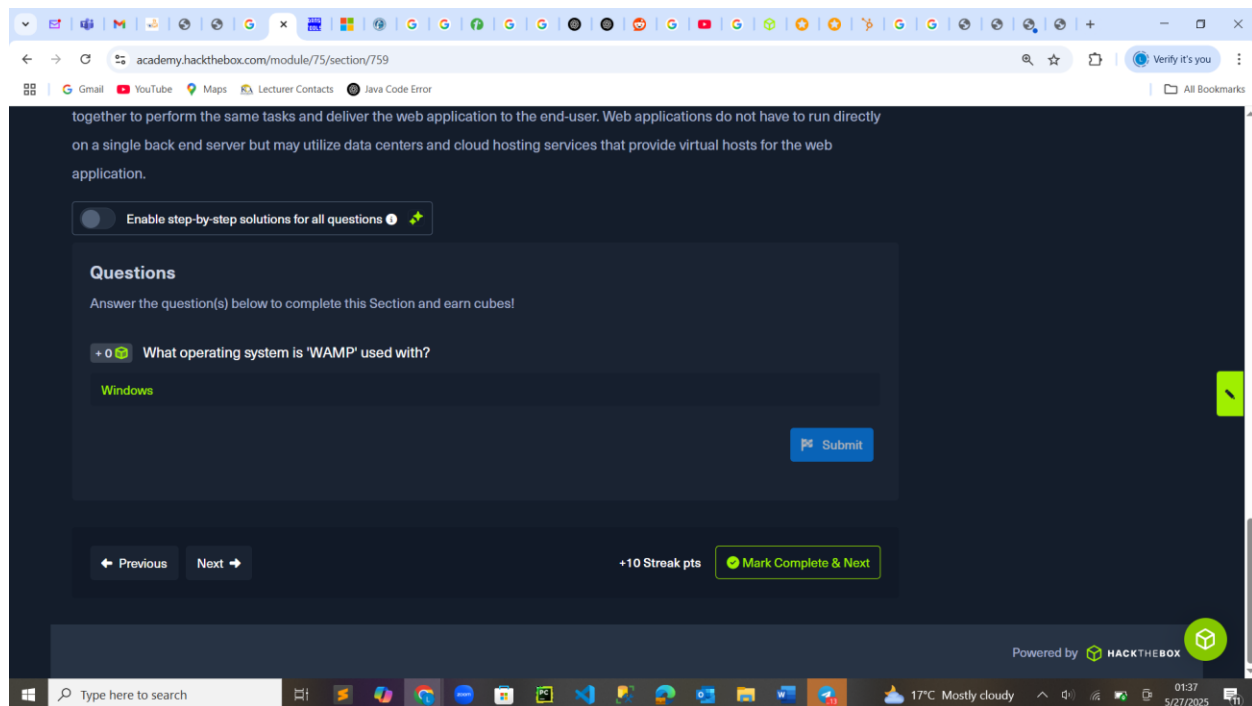
10. CROSS-SITE REQUEST FORGERY (CSRF)

CSRF attacks may utilize XSS vulnerabilities to perform certain queries, and API calls on a web application that the victim is currently authenticated to. This would allow the attacker to perform actions as the authenticated user. It may also utilize other vulnerabilities to perform the same functions, like utilizing HTTP parameters for attacks.

it is also always important to filter and sanitize user input on the front end before it reaches the back end, and especially if this code may be displayed directly on the client-side without communicating with the back end. Two main controls must be applied when accepting user input: sanitization and validation.

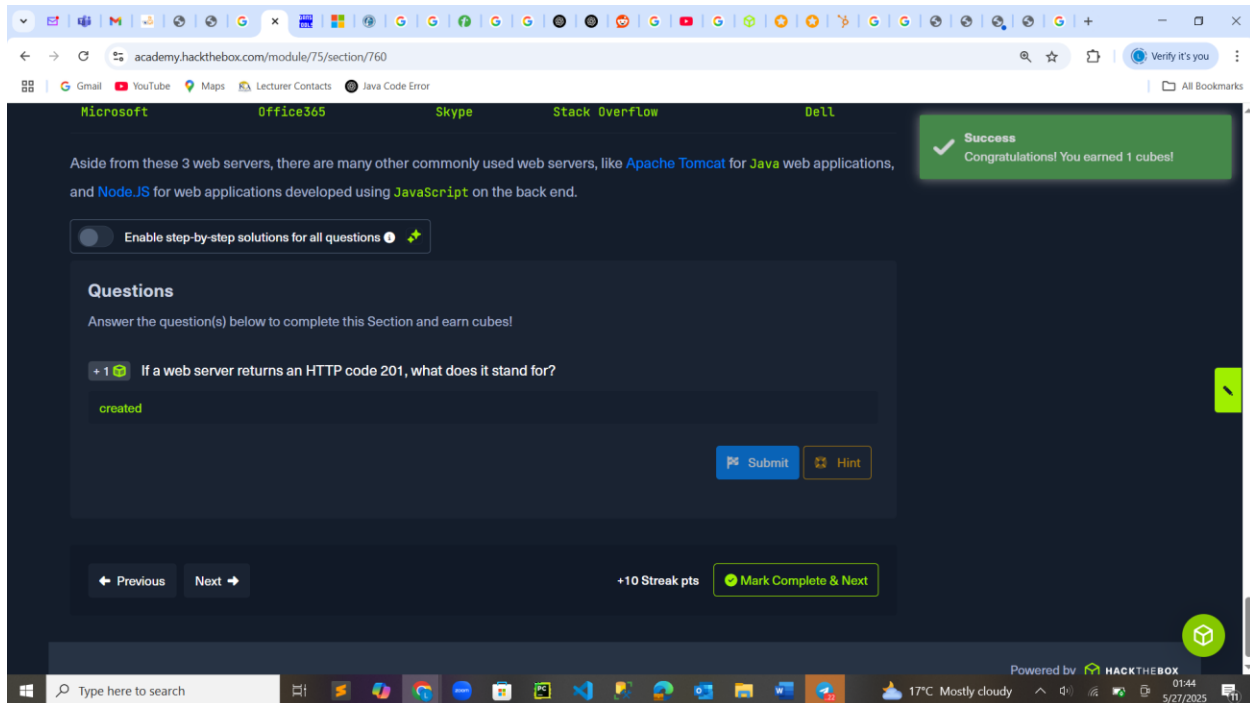
11. BACK END SERVERS

A back-end server is the hardware and operating system on the back end that hosts all of the applications necessary to run the web application. It is the real system running all of the processes and carrying out all of the tasks that make up the entire web application.



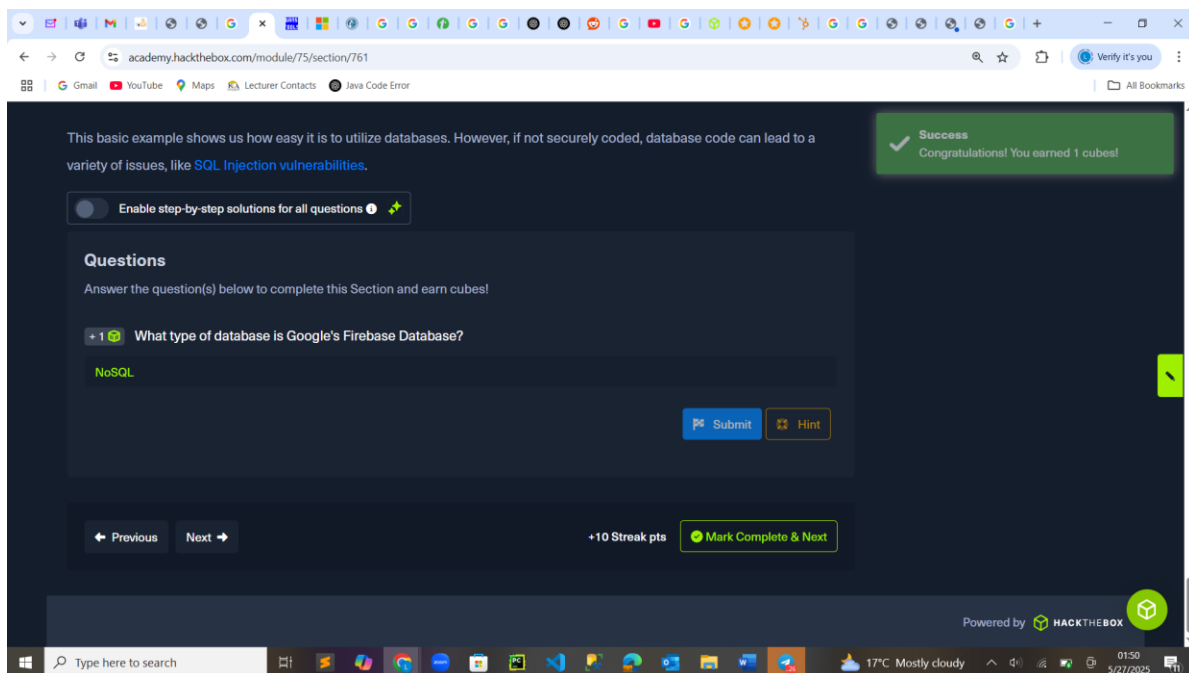
12. WEB SERVERS

A web server is an application that runs on the back end server, which handles all of the HTTP traffic from the client-side browser, routes it to the requested pages, and finally responds to the client-side browser. Web servers usually run on TCP ports 80 or 443, and are responsible for connecting end-users to various parts of the web application, in addition to handling their various responses.



13. DATABASES

Databases store various content and information related to the web application. SQL databases (MySQL, PostgreSQL) excel at relational data with fast queries, while NoSQL (MongoDB, Cassandra) offers flexibility for dynamic content. Proper integration ensures efficiency, but insecure coding risks SQL injection, making database choice and security critical for performance and safety.



14. DEVELOPMENT FRAMEWORKS AND APIs

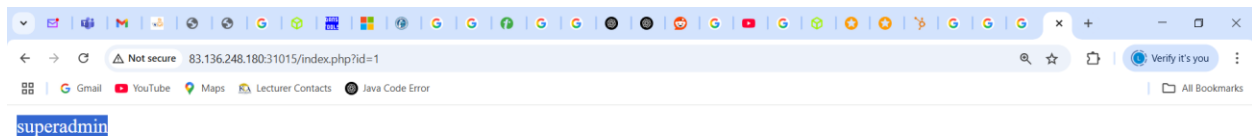
Web frameworks like Laravel and Django simplify development by handling common tasks. APIs connect front-end and back-end, allowing data exchange through requests and responses.

SOAP APIs use XML for structured data transfer, useful for complex operations. REST APIs rely on URLs and JSON, making them lightweight and scalable for web apps.

APIs enable features like weather data or social media feeds. They use HTTP methods - GET retrieves data, POST creates, PUT updates, and DELETE removes. Proper API design ensures secure and efficient communication between components. Frameworks and APIs together power modern web applications.

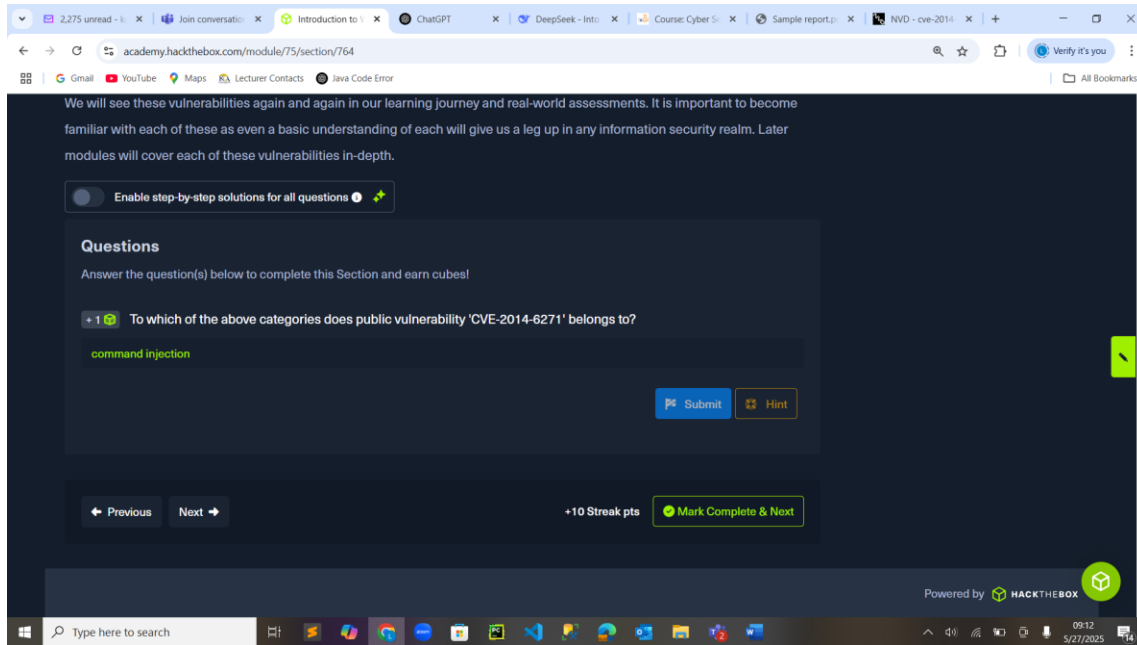
Use GET request '/index.php?id=0' to search for the name of the user with id number 1?

Spawning the machine and copying the address and port in a new tab on the browser and navigating to /index.php?id=0. Replaced the 0 with a 1 as the question suggests that we search for the name of the user with the id number 1. The name appears as superadmin as shown below:



15. COMMON WEB VULNERABILITIES

Researched the vulnerability from National Vulnerability database and got to understand that it processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, which is under **command injection** category



16. PUBLIC VULNERABILITIES

What is the CVSS v2.0 score of the public vulnerability CVE-2017-0144? 9.3

The screenshot shows the 'CVE-2017-0144 Detail' page on the NVD website. The page is titled 'DEFERRED' and states: 'This CVE record is not being prioritized for NVD enrichment efforts due to resource or other concerns.' The 'Description' section explains that the vulnerability affects the SMBv1 server in Microsoft Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8.1, Windows Server 2012 Gold and R2, Windows RT 8.1, and Windows 10 Gold, 1511, and 1607; and Windows Server 2016, allowing remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." The 'Metrics' section shows the CVSS Version 2.0 score as 9.3 HIGH. The 'References to Advisories, Solutions, and Tools' section provides links to NIST and other sources.

DEFERRED

This CVE record is not being prioritized for NVD enrichment efforts due to resource or other concerns.

Description

The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.

Metrics

CVSS Version 4.0 CVSS Version 3.x CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 2.0 Severity and Vector Strings:

NIST: NVD Base Score: 9.3 HIGH Vector: (AV:N/AC:M/Au:N/C:C/I:C/A:C)

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed,