Digital Forensics

Instructor: Muhammad Waqar

Muhammad Murtaza

24109110

Syed Muhammad Khizar Hussain

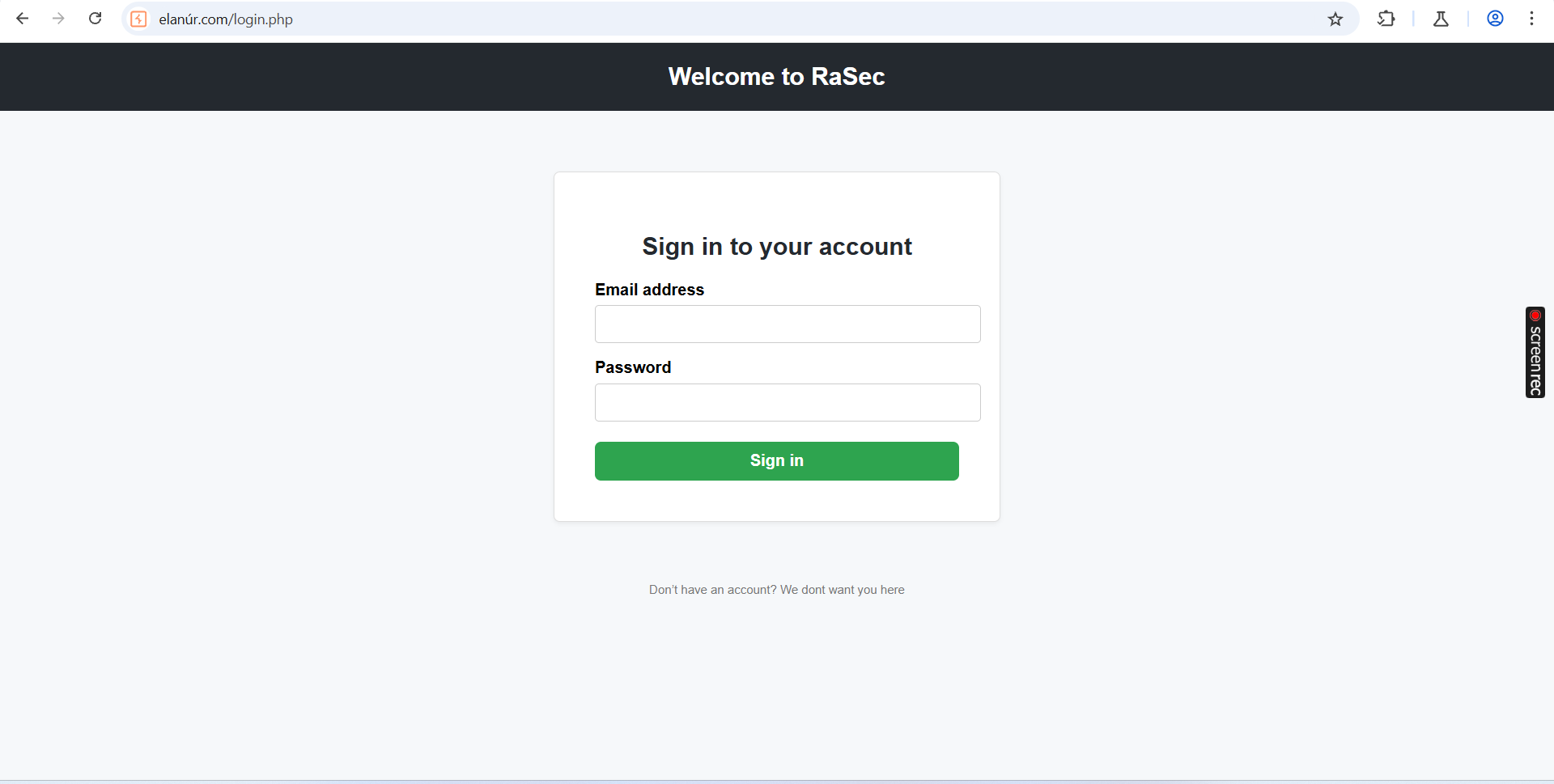
24109125

**Black Box Penetration Test Methodology**

**1. Initial Credential Attack Attempt**

**Summary:**As part of the initial reconnaissance phase of this black box penetration test, a login page was discovered on the target web application. Without any prior knowledge of valid usernames or passwords, an authentication attempt was made using a set of commonly known default credentials. This technique is frequently leveraged in real-world attacks to identify weakly protected systems where developers or administrators have neglected to remove or change default access configurations. Although this initial attempt did not succeed in granting unauthorized access, it served a crucial role in establishing the baseline security posture of the authentication mechanism and confirmed that no basic credential misconfigurations were in place.

**What Was Done:**The login interface was manually tested using typical default credentials such as admin:admin, admin:password, root:toor, and user:user. These attempts were performed without automated brute-force tools to avoid triggering any potential rate-limiting or IP-blocking mechanisms. The server consistently returned authentication failures, indicating the application had not been left vulnerable to such basic flaws.



**2. Source Code Review via Page Inspection**

**Summary:**  
Further reconnaissance involved analyzing the client-side code of the web application through page source inspection. This type of passive analysis is a standard procedure in black box penetration testing, aiming to uncover any sensitive information that may have been inadvertently exposed by developers. It is not uncommon for developers to leave behind debug comments, endpoint references, or even hardcoded data within HTML, JavaScript, or hidden form elements. In this case, viewing the page source revealed multiple clues that hinted at the existence of backend directories and possibly additional application functionality that was not directly linked from the main interface. These hints were instrumental in shaping the next phase of the assessment.

**What Was Done:**  
Using the browser’s “View Page Source” feature, the entire HTML structure of the login page was reviewed. Embedded comments, form actions, hidden fields, and JavaScript logic were analyzed to gather indirect information. References to internal directory paths, suspicious endpoints, and parameter structures were noted for further exploration.

A screenshot of a login form

AI-generated content may be incorrect.

**3. Initial Directory Brute-Force Attempt**

**Summary:**  
After collecting clues from the source code, a directory enumeration attack was initiated to identify potentially unprotected or hidden directories on the web server. Directory brute-forcing is a common tactic used by attackers to uncover files and folders that have not been properly secured or indexed. These directories often contain administrative tools, configuration files, backups, or test pages that are not intended for public access. The results of this first enumeration effort helped map out the initial layout of the server’s directory structure, providing actionable paths for deeper analysis and exploitation.

**What Was Done:**  
A directory brute-force scan was launched using tools such as dirb and ffuf with a medium-sized wordlist derived from SecLists (e.g., common.txt). The tool was configured to target the root of the domain and analyze HTTP response codes for each requested path. The goal was to identify directories returning 200 OK or 403 Forbidden responses, which typically indicate the presence of real files or folders.

A screenshot of a computer

AI-generated content may be incorrect.

**4. Further Directory Brute-Forcing — Valid Paths Identified**

**Summary:**  
A second, more extensive directory brute-forcing phase was conducted, using a refined wordlist and adjusted scanning parameters. This yielded three valid directories that were previously undiscovered and likely not intended for public access. The existence of these directories indicates a lack of proper access control and directory indexing prevention on the web server, which could expose the application to unauthorized file browsing and further exploitation. The discovery of these directories significantly expanded the attack surface and provided new avenues for manual exploration and vulnerability identification.

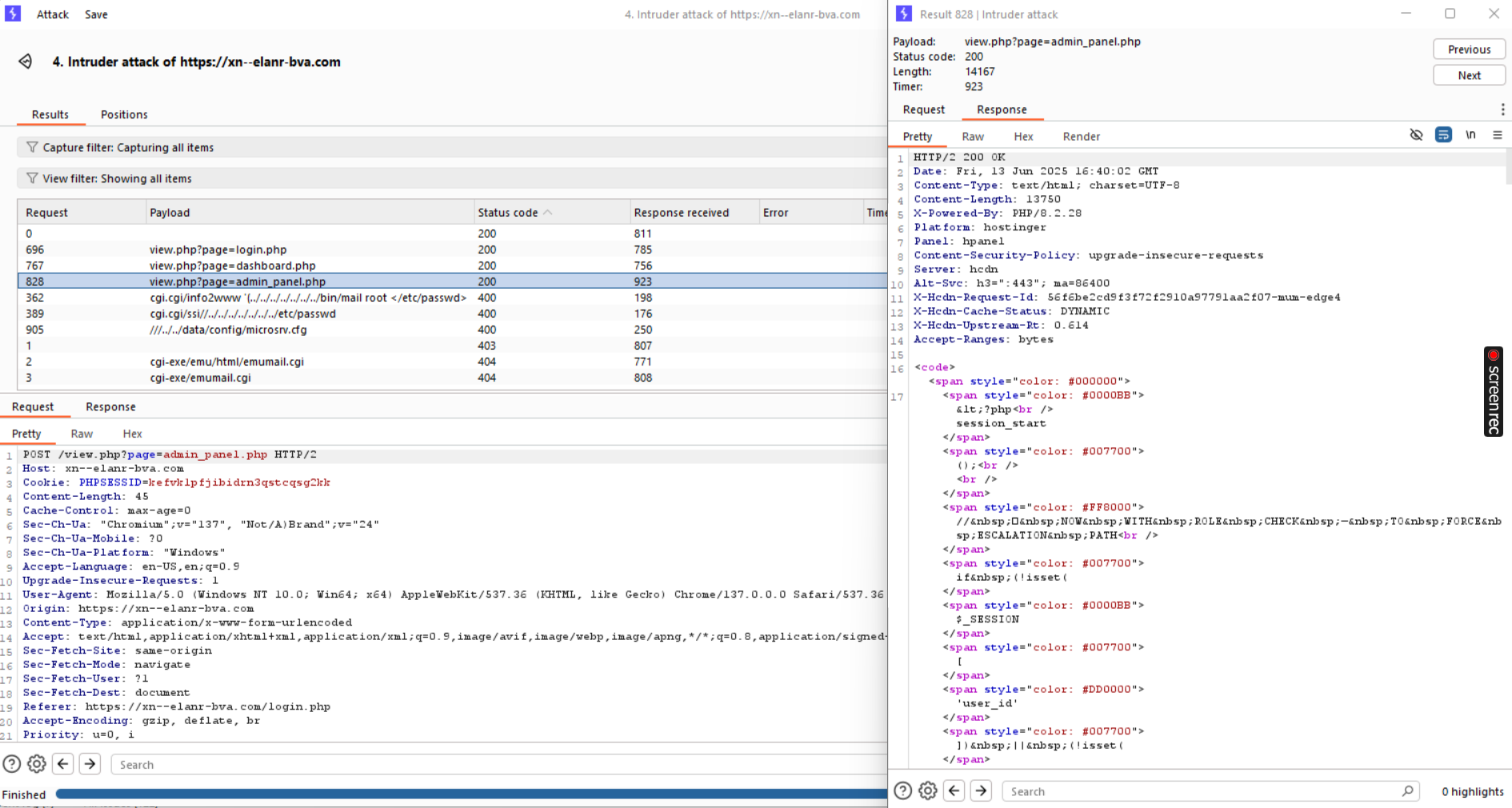
**What Was Done:**  
I increased the depth of scanning using recursive brute-forcing and leveraged a more comprehensive wordlist (e.g., directory-list-2.3-medium.txt). By changing the scanning configuration to include trailing slashes and check for directory-specific HTTP headers, I successfully identified three accessible directories that returned HTTP 200 responses. These were manually tested using a browser to explore their contents, which revealed files of interest, including source code and logs.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

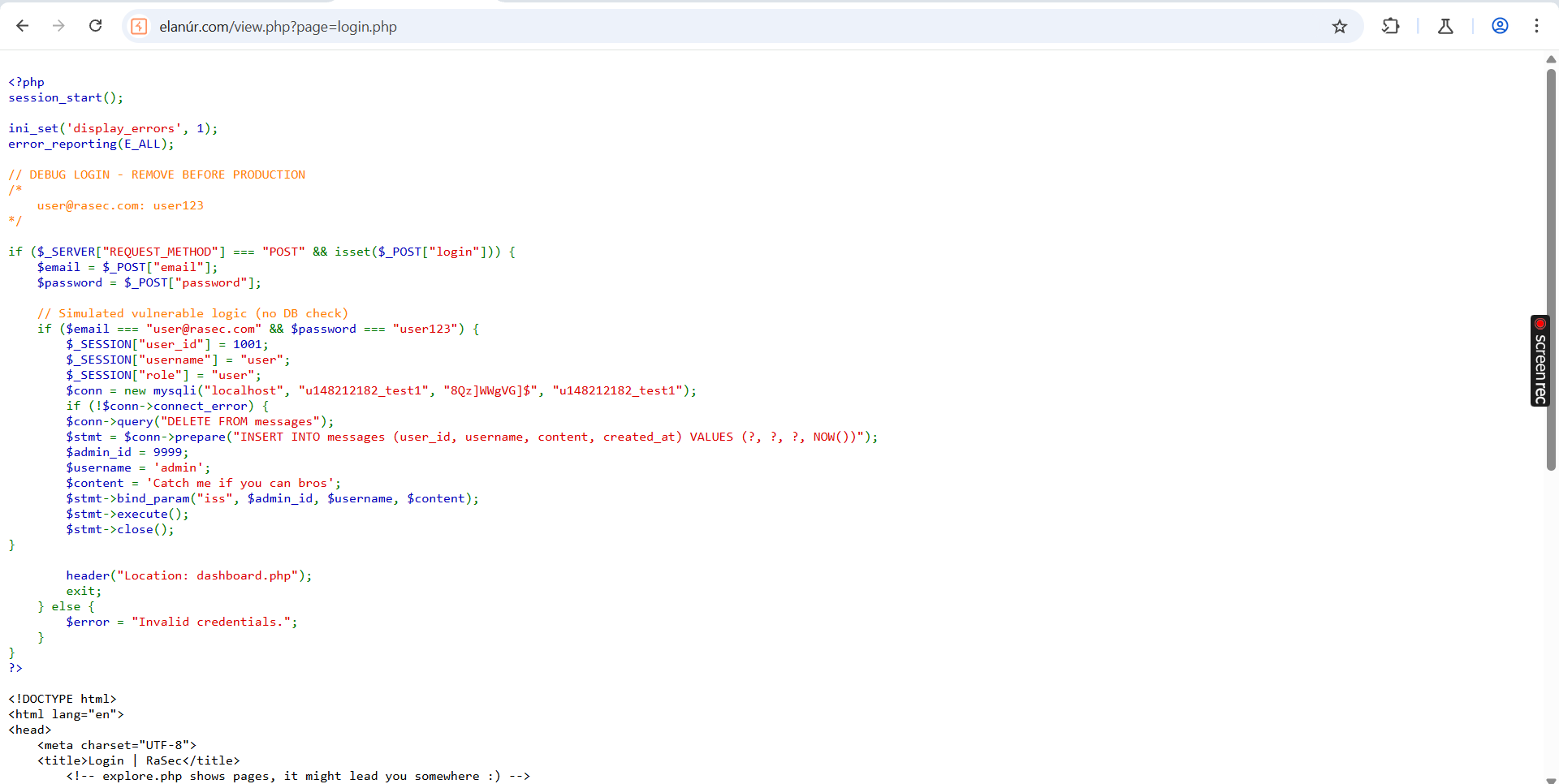
AI-generated content may be incorrect.



**5. Source Code Exposure via LFI — Hardcoded Credentials Found**

**Summary:**  
Access to the discovered directories revealed a critical misconfiguration: the presence of source code files exposed directly to public access. Upon reviewing these files, it became evident that the application suffered from a Local File Inclusion (LFI) vulnerability, allowing unauthorized users to view the raw code of server-side scripts. One of these files contained hardcoded credentials embedded within the authentication logic, which is a serious security flaw. The exposure of these credentials could lead to unauthorized account access, lateral movement, and complete compromise of user accounts or internal systems.

**What Was Done:**  
I navigated to files like .php, .txt, and .bak within the newly discovered directories. Viewing the contents of a PHP script revealed user authentication logic, including a hardcoded username and password variable pair. This indicated either careless handling of secrets or an intentional backdoor left by the developer. I documented these credentials for further testing and potential use in authentication bypass attempts.



**6. Additional Source Code Review in Other Directories**

**Summary:**  
Continued inspection of the remaining discovered directories revealed further insecure development practices. Multiple files displayed internal logic, user access controls, or error messages that should not be exposed in a production environment. This level of transparency into the application’s backend structure provided a clearer understanding of how authentication and privilege checks were performed. More importantly, it highlighted additional coding flaws and the potential for privilege escalation.

**What Was Done:**  
Using the same method of manually browsing through exposed directories and reviewing .php, .html, and .js files, I identified several files that contained implementation details of session management and user roles. These were analyzed to determine whether they could be manipulated to gain unauthorized privileges or bypass security mechanisms.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**7. Successful Authentication Using Recovered Credentials**

**Summary:**  
Using the hardcoded credentials previously discovered in the source code, I attempted to authenticate via the main login interface. The login attempt was successful, confirming that the credentials were active and tied to a valid user account. This validated the severity of the earlier discovery and underscored the risk of hardcoded secrets in application logic. Successful authentication represented a major milestone in the assessment, granting access to potentially sensitive user functionalities.

**What Was Done:**  
I entered the recovered credentials (username and password) into the web application’s login form. Upon submission, I was redirected to a dashboard interface, indicating successful login. This confirmed that the credentials had not been disabled or deprecated, and that there were no protections in place to restrict login from unknown IP addresses or locations.

A screenshot of a computer

AI-generated content may be incorrect.

**8. Privilege Escalation to Admin**

**Summary:**  
Following successful login as a regular user, I initiated a privilege escalation assessment to determine whether administrative access could be obtained through manipulation of session tokens, parameter tampering, or endpoint abuse. This stage was successful — I identified a method to elevate my privileges and gain access to an administrative interface. This level of access gave complete control over the web application, including user management, system settings, and potentially access to underlying infrastructure. Such a privilege escalation pathway represents a critical risk, as it allows attackers to operate with full authority within the compromised environment.

**What Was Done:**  
After authenticating as a user, I inspected session cookies, user roles, and API requests using browser developer tools. I tested whether modifying cookie values (e.g., changing role=user to role=admin) would result in elevated access. Additionally, I probed restricted endpoints directly by guessing or fuzzing URLs (e.g., /admin, /manage, /settings). Eventually, I accessed an admin panel that was not properly protected, confirming successful privilege escalation. All administrative functionalities were available at this point.

