

Security Assessment Findings Report - CodeNimbus Systems

Name	Title	Contact Information
CodeNimbus Systems		
John Smith	Global Information Security Manager	Email: jsmith@CodeNimbusSystems.com
SudoZain		
Muhammad Zaindin	Penetration Tester	Email: muhzaindin03@gmail.com

Business Confidential

Date: July 7, 2025
Project: CNS-01
Version: 1.0

Tables of Contents

1. Image Table	3
2. Confidentiality Statement	4
3. Disclaimer	4
4. Assessment Overview	5
5. Assessment Components	5
6. Executive Summary	5
7. Scoping and Time Limitations	6
8. Testing Summary	6
9. Tester Notes and Recommendations.....	6
10. Key Strengths and Weaknesses	7
11. Introduction	7
12. Objective	7
13. Scope	8
14. Summary	9
15. Technical Findings	10
16. SQL Injection – GET Parameter id	10
a. Steps to Reproduce.....	11
b. Proof of concept	11
17. Stored XSS in the Admin profile	13
a. Steps to Reproduce.....	14
b. Proof of concept	14
18. Broken Access Control - Unauthenticated Admin Page Access	17
a. Steps to Reproduce	18
b. Proof of concept	18
19. Server-Side Request Forgery (SSRF) in URL Parameter	20
a. Proof Of Concept	21
20. Reflected XSS via k304 GET Parameter	24
a. Steps to Reproduce	24
b. Proof of concept	25
21. Insecure Design (Password Reset Mechanism)	26
a. Steps to Reproduce	27
b. Proof of concept	27
22. IDOR - View Other Users' Notes	29
a. Steps to Reproduce	29
b. Proof of concept	30
23. Autocomplete-enabled Password Field	32
a. Steps to Reproduce	32
b. Proof of concept	33

Image Table

Image	Title	Description	Page
Image 1.1	Exploring the website	main domain page for SQL Injection vulnerable application	10
Image 1.2	SQL error during manual testing	Backend show error due to the input error 2'	10
Image 1.3	Automation testing using sqlmap	Sqlmap injecting various payloads	11
Image 1.4	SQLMap discloses the version of the database	Results generated by sqlmap	11
Image 2.1	Exploring the domain	Main domain page for stored XSS vulnerable application	13
Image 2.2	Manual testing Contact Us	Filling the user field of contact us	13
Image 2.3	the results of sent message in the Queries		14
Image 2.4	Send a malicious payload in the message field	Filling the input field with javascript code	14
Image 2.5	empty pop up after sending the malicious XSS payload	Result of the previous step	14
Image 2.6	pop up proving the previous pop up was resulting of XSS vulnerability	Result of injecting a malicious javascript code in the input field	15
Image 3.1	Authenticate as test	Registration a new account	17
Image 3.2	Enumerate directories	Using dirbuster to automate the enumeration	18
Image 3.3	After navigating to /admin.php	The ability to access sensitive file	18
Image 4.1	Exploring the domain	Main domain page for SSRF vulnerability application	20
Image 4.2	giving the parameter in the URL a different value	The parameter looks suspicious to SSRF	20
Image 4.3	Using Dirbuster automate values for url parameter		21
Image 4.4	Confirm manual from dirbuster	After finding set of results we can test them manually	21
Image 4.5	Exposing sensitive information from the server like credentials	While testing manually we could retrieve sensitive info from /config	22
Image 5.1	Exploring the domain website	Main domain page for reflected XSS vulnerability application	24
Image 5.2	Finding a Vulnerable Parameter to reflected XSS	Found a vulnerable parameter to reflected XSS	24
Image 6.1	Found a login form	During exploration of the domain	26
Image 6.2	Testing the I forgot my password option		26
Image 6.3	Trying to guess security questions	Explored more than one security questions	27

Image 6.4	more investigating from the previous step		27
Image 6.5	Getting new password	After answering correctly	27
Image 6.6	any account is accessible by guessing the security question	Represents the result of the attack	27
Image 7.1	Login form to note server		29
Image 7.2	after authentication shows my note		29
Image 7.3	other notes are exposed	As a result of IDOR vulnerability	30

1. Confidentiality Statement

This document contains confidential information intended solely for the use of the recipient organization (CodeNimbus Systems) and authorized personnel. Unauthorized access, copying, disclosure, or distribution of any part of this document is strictly prohibited. If you are not the intended recipient, please notify the sender and destroy this document immediately.

2. Disclaimer

The findings and recommendations in this report are based on the security assessment conducted during the specified period. The assessment was performed on a best-effort basis and is limited to the scope defined. The tester cannot guarantee the identification of all vulnerabilities or be held liable for any future incidents resulting from untested or undisclosed areas.

3. Assessment Overview

This penetration test focused on identifying security vulnerabilities in CodeNimbus Systems' web applications. The goal was to simulate real-world attack scenarios, assess exposure to threats, and provide actionable recommendations to improve the overall security posture. The engagement included:

- Manual and automated vulnerability discovery
- Proof-of-concept exploitations (where safe and approved)
- Analysis of system configuration, access controls, and input validation

4. Assessment Components

- **Target Scope:** <https://app.codenimbus-systems.com>, supporting services and endpoints
- **Testing Type:** Black-box / External Web Application Penetration Testing
- **Tools Used:** Burp Suite, OWASP ZAP, Nikto, dirsearch, sqlmap, browser-based testing, manual fuzzing
- **Vulnerability Categories:** OWASP Top 10 (XSS, SQLi, Broken Access Control, SSRF, etc.)

5. Executive Summary

The assessment identified multiple security vulnerabilities that could be exploited by an external attacker. Key issues include:

- Stored Cross-Site Scripting (XSS)
- SQL Injection
- Broken Access Control
- Server-Side Request Forgery (SSRF)
- Insecure direct object references (IDOR)
- Insecure Design

Several of these findings fall under the **High/Critical** severity category due to their potential to impact confidentiality and system control. Immediate remediation is recommended to mitigate risk.

6. Scoping and Time Limitations

- **Assessment Window:** July 1-7, 2025
- **Environment Access:** Public-facing web interface only (no internal infrastructure or API testing included)
- **Limitations:** No source code access, no DoS attacks, no exploitation beyond PoC level
- **Tester Role:** External, unauthenticated black-box (with test accounts where applicable)

7. Testing Summary

#	Vulnerability Title	Description	Severity
1	SQL Injection – GET Parameter id	Time-based and union-based SQL injection allows DB access via id parameter.	Critical
2	Stored XSS in the Admin Profile	Malicious script stored by a receptionist user triggers in the admin panel.	Medium
3	Broken Access Control – Unauthenticated Admin Page	Non-admin users can access and modify admin panel functionalities without authorization.	Critical
4	SSRF – URL Parameter	Server-side request forgery enables access to internal resources via crafted url.	Critical
5	Reflected XSS – k304 Parameter	Payloads in k304 parameter are reflected and executed client-side.	Medium
6	Insecure Design – Password Reset via Security Questions	Password reset mechanism uses guessable answers, allowing account takeover.	Critical
7	IDOR – View Other Users' Notes	Users can view others' notes by modifying the note_id in the URL.	High
8	Autocomplete-Enabled Password Field	Login form allows browsers to store autofill data, posing risks on shared or infected devices.	Low

8. Tester Notes and Recommendations

1. Ensure **input sanitization** and output encoding is enforced across all endpoints.
2. Implement **strict access controls** based on user roles and enforce server-side checks.
3. Validate and whitelist **URL inputs** to prevent SSRF attacks.
4. Protect **configuration files** and sensitive directories via proper access controls.
5. Conduct **regular internal code reviews** and **automated security testing** in CI/CD.

9. Key Strengths and Weaknesses

Strengths:

- User interface is clean and user-friendly
- Session expiration and logout mechanisms work as expected
- Some access restrictions (403) are partially in place

Weaknesses:

- Input is not validated or sanitized on several endpoints
- Access control mechanisms are improperly enforced
- Sensitive internal requests are exposed via SSRF
- SQL injection is possible through login bypass techniques

10. Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against **CodeNimbus Systems**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

11. Objective

The objective of the assessment was to assess the state of security and uncover vulnerabilities in **CodeNimbus Systems Web Application** and provide with a final security assessment report comprising vulnerabilities, remediation strategy and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

12. Scope

This section defines the scope and boundaries of the project.

Application Name	CodeNimbus Systems Web Application
URL	https://staging.codenimbus-systems.com https://app.codenimbus-systems.com/admin https://beta.codenimbus-systems.com https://cdn.codenimbus-systems.com

12.1. Assessment Attribute(s)

Parameter	Value
Starting Vector	External
Target Criticality	Critical
Assessment Nature	Cautious & Calculated
Assessment Conspicuity	Clear
Proof of Concept(s)	Attached wherever possible and applicable.

12.2. Risk Calculation and Classification

Following is the risk classification:

Info	Low	Medium	High	Critical
No direct impact. Typically used for informational findings only.	Minor impact or low probability of exploitation. Usually local or hard to exploit.	Moderate impact. Exploitable with some effort or partial access. May need user interaction.	Major impact. Exploitable remotely or by low-privileged users. Fix or patch usually available.	Severe impact. Easy to exploit, remote, no patch available. May lead to full system compromise or data breach.

Table 1: Risk Rating

13. Summary


This penetration test was conducted on the CodeNimbus Systems web applications to assess the overall security posture, identify vulnerabilities, and recommend mitigation strategies. The testing focused on authenticated and unauthenticated user roles and covered key areas including input validation, access control, session management, and server-side logic.

Total: 8 Vulnerabilities

Critical	High	Medium	Low	Info
4	1	2	1	0

14. Technical Findings

1. SQL Injection – GET Parameter id

Reference No:	Risk Rating:
WEB_VUL_01	Critical 
CVSS:	CWE:
9.4 - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Tools Used:	
Browser, SQL Map	
Vulnerability Description:	
Unsanitized user input in the id GET parameter is directly embedded into the SQL query, enabling SQL injection. This allows execution of error-based, UNION, time-based blind, and stacked queries via MySQL.	
Vulnerability Identified by / How It Was Discovered	
Manual testing and automated validation using sqlmap.	
Vulnerable URLs / IP Address	
<a href="https://staging.codenimbus-systems.com/post?id=<number>">https://staging.codenimbus-systems.com/post?id=<number>	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">Confidentiality breach: Attacker can extract sensitive data (e.g., user credentials) via UNION/time-based queries.Integrity/data loss: Possibility to modify or delete data (e.g., DROP TABLE).Account takeover: Bypassing authentication logic (e.g., OR 1=1).System compromise: Execution of stacked queries can lead to RCE or OS-level attacks with sufficient privileges.	
Suggested Countermeasures	
It is recommended to implement below control for mitigating the SQLi: <ul style="list-style-type: none">Use prepared statements / parameterized queries (e.g., WHERE id = ?)Enforce strict input validation: ensure id is numeric only.Configure DB user privileges minimallySuppress detailed SQL error messages in user-facing responsesConduct security code reviews and periodic testing, including sqlmap or similar DAST, and follow OWASP SQLi prevention recommendations.	
References	
https://owasp.org/www-community/attacks/SQL_Injection https://logz.io/blog/defend-against-sql-injections/	

Steps to Reproduce:

1. Open browser to: `https://staging.codenimbus-systems.com/post?id=2'`
2. Observe syntax error in response.
3. Run:
`?id=2 AND SLEEP(5)`
4. Response delay confirms time-based injection.

Proof of concept:

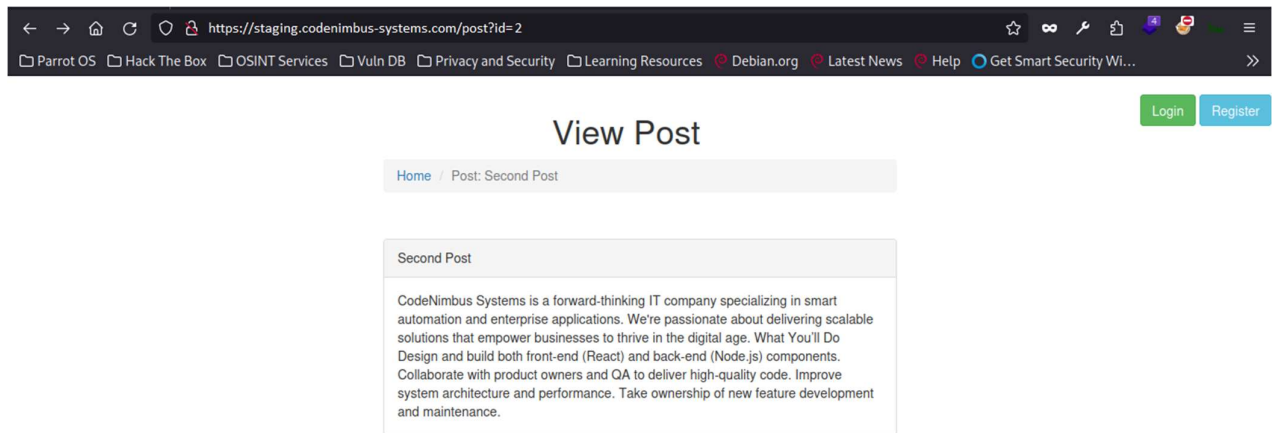


Image 1.1: Exploring the website

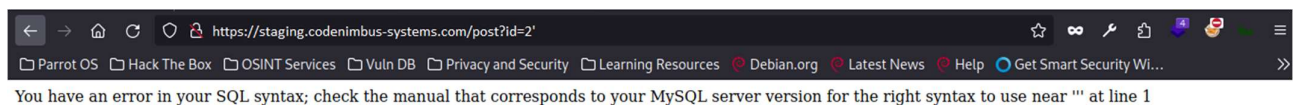


Image 1.2: SQL error during manual testing

The results of SQL error are a sign for SQL Injection

```

[14:21:43] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[14:21:43] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'% near "" at line 1
[14:21:43] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[14:21:44] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[14:21:44] [INFO] GET parameter 'id' is 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)' injectable
[14:21:44] [INFO] testing 'MySQL inline queries'
[14:21:44] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[14:21:44] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
[14:21:56] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 stacked queries (comment)' injectable
[14:21:56] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[14:22:06] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[14:22:06] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[14:22:06] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[14:22:06] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[14:22:07] [INFO] target URL appears to have 4 columns in query
[14:22:08] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable

```

Image 1.3: Automation testing using sqlmap

As sqlmap says that the 'GET parameter id is injectable' and looking at the payloads it discovered we know that we can dump the database. But for confidence reasons we will not do that and enough with knowing the exact version of the database.

```


[14:22:44] [INFO] the back-end DBMS is MySQL
[14:22:45] [WARNING] potential permission problems detected ('command denied')
web server operating system: Linux Ubuntu
web application technology: Nginx 1.18.0
back-end DBMS: MySQL >= 5.6

```

Image 1.4: SQLMap discloses the version of the database

Sqlmap also shows other information like the web server OS and web application technology.

2. Stored XSS in the Admin profile.

Reference No:	Risk Rating:
WEB_VUL_02	Medium 
CVSS:	CWE:
4.6: CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Tools Used:	
Browser	
Vulnerability Description:	
<p>User input in the message submission form is stored in the database and rendered without sanitization on the admin queries page.</p> <p>This allows malicious HTML/JS (e.g. <code><script>alert('XSS')</script></code>) to execute for any admin or staff viewing that page. This is a classic stored XSS scenario.</p>	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://app.codenimbus-systems.com/messages/submit https://app.codenimbus-systems.com/admin/queries	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Account/session takeover: Attackers could steal cookies/tokens and hijack admin sessions.• Privilege escalation & unauthorized actions: Malicious code can perform admin actions on behalf of users.• Malware distribution & phishing: Inject scripts to redirect or present fake login prompts.• Brand and data integrity risk: Persistent XSS harms user trust and can leak sensitive client info.	
Suggested Countermeasures	
<p>It is recommended to:</p> <ul style="list-style-type: none">• Input validation & sanitization: Clean input or restrict to only expected content (e.g. plaintext).• Output encoding/escaping: HTML-encode user-provided data before display.• Strict Content Security Policy: Restrict script sources and disallow inline scripts.• HttpOnly session cookies: Prevent JavaScript access to authentication tokens.• Use secure templating/frameworks: Use templating libraries that auto-escape input (e.g., React, Angular).• Periodic audits: Regular code reviews, automated scanning (SAST/DAST), and input/output flow analysis.	
References	
https://portswigger.net/web-security/cross-site-scripting https://blog.sqreen.com/stored-xss-explained/	

Steps to Reproduce:

1. Log in as receptionist, navigate to: <https://app.codenimbus-systems.com/messages/submit>
2. Fill out fields, including payload in message: `<script>alert('XSS')</script>`
3. Submit form.
4. As admin, visit: <https://app.codenimbus-systems.com/admin/queries>
5. Observe JavaScript dialog (`alert('XSS')`) shown for each message entry. (Screenshot evidence)

Proof of concept:

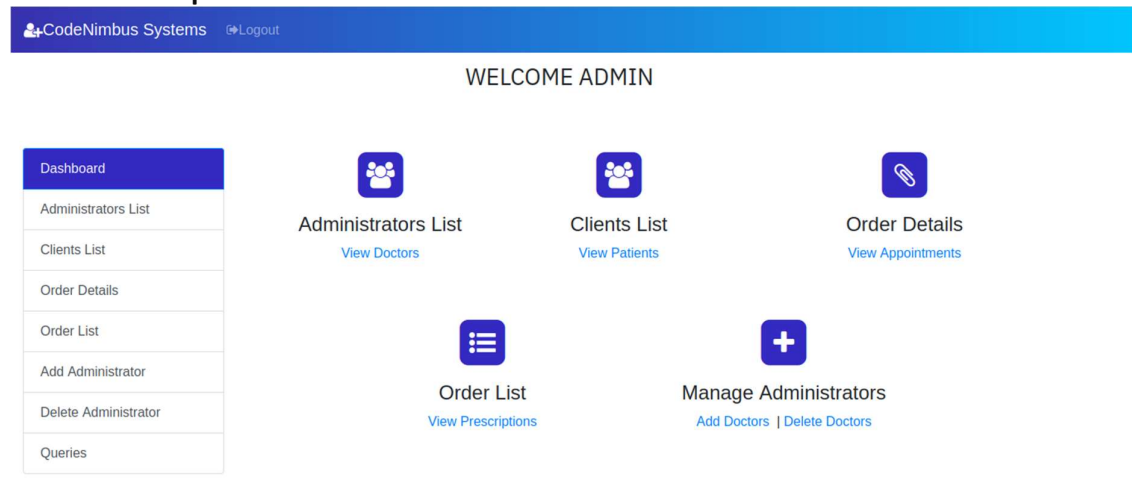


Image 2.1 : Exploring the domain

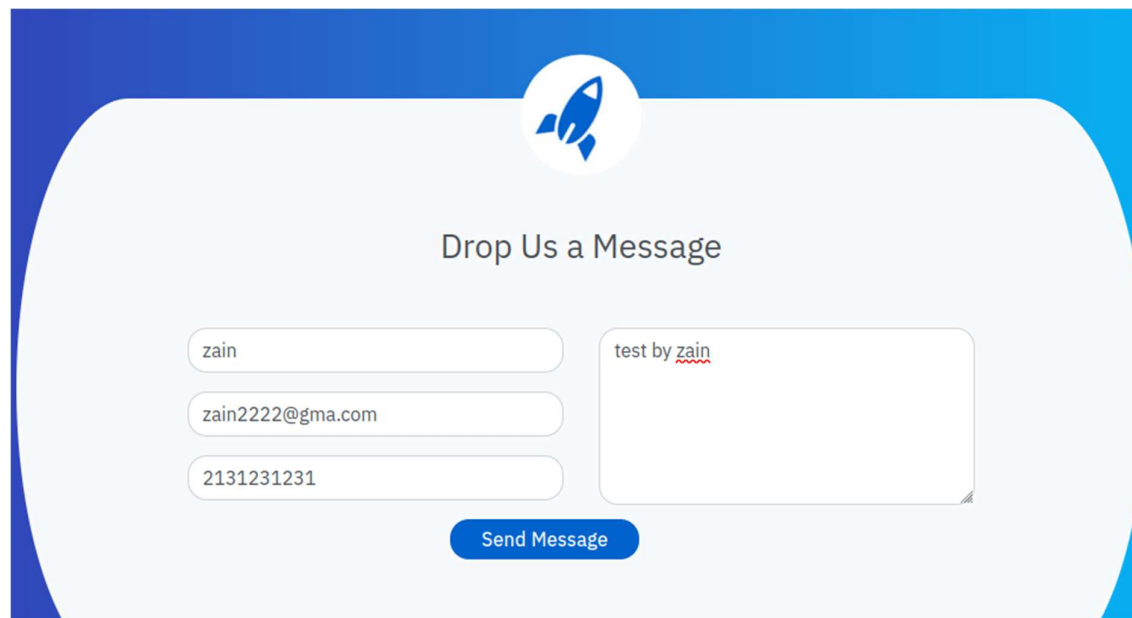


Image 2.2 : Manual testing Contact Us

CodeNimbus Systems Logout				
Delete Administrator	Mani	mani@gmail.com	8977768978	Want some coffee?
Queries	Karthick	karthi@gmail.com	9898989898	Good service
	Abbis	abbis@gmail.com	8979776868	Love your service
	Asiq	asiq@gmail.com	9087897564	Love your service. Thank you!
	Jane	jane@gmail.com	7869869757	I love your service!
		sad@dc.com	2123123213	
	zain	zain2222@gma.com	2131231231	test by zain
	zain	zain2222@gma.com	2131231231	test by zain

Image 2.3 : the results of sent message in the Queries

The screenshot shows a web form titled "Drop Us a Message". It contains four input fields and a "Send Message" button. The first three fields contain malicious XSS payloads: the first has `<script>alert()</script>`, the second has `zain2222@gma.com`, and the third has `2131231231|`. The fourth field is empty. The "Send Message" button is blue and located at the bottom right of the form.

Image 2.4 : Send a malicious payload in the message field

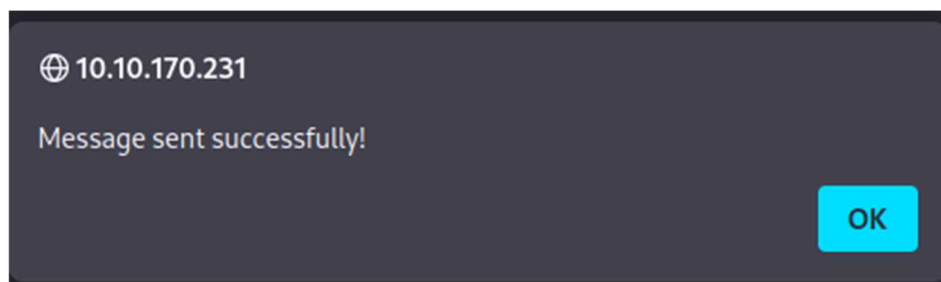


Image 2.5: empty pop up after sending the malicious XSS payload

Here we need to login as admin to see the execution of the tested JavaScript payload.

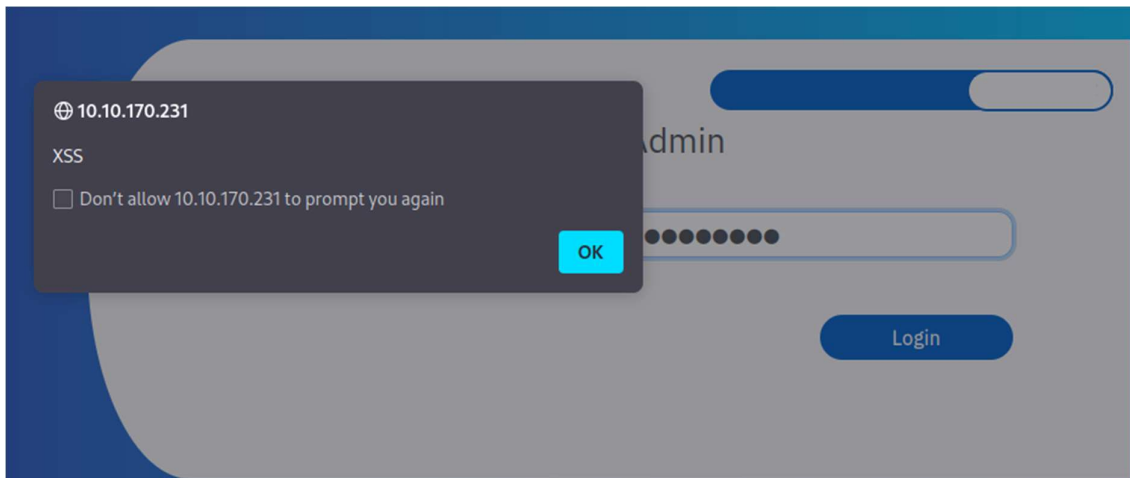



Image 2.6: pop up proving the previous pop up was resulting of XSS vulnerability

3. Broken Access Control – Unauthenticated Admin Page Access

Reference No:	Risk Rating:
WEB_VUL_03	Critical 
CVSS:	CWE:
9.9 CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:L	CWE-284: Improper Access Control
Tools Used:	
Browser	
Vulnerability Description:	
<p>The application fails to enforce proper access controls on sensitive endpoints. Specifically, the admin.php page is accessible to low-privilege or unauthenticated users by directly navigating to the URL. No server-side verification of user roles or session data is performed before rendering admin-only functionality (e.g., user management, privilege changes). This is a clear case of Broken Access Control and violates OWASP A01:2021.</p>	
Vulnerability Identified by / How It Was Discovered	
<ul style="list-style-type: none">• Manual testing (forced browsing)• Automation testing: Gobuster directory enumeration• Verified manually	
Vulnerable URLs / IP Address	
https://cdn.codenimbus-systems.com/	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Privilege Escalation: Any authenticated user can elevate privileges, assign themselves admin rights, or modify user roles.• Unauthorized Data Access: Attackers can view, edit, or delete user records or system settings intended for admins only.• Complete Application Compromise: If additional controls (like audit logging or input validation) are also missing, attackers can potentially modify business-critical data or create backdoor accounts.• Non-compliance Risks: Exposure of administrative functions to unprivileged users can lead to violations of GDPR, HIPAA, or other data protection standards.	
Suggested Countermeasures	
<p>It is recommended to:</p> <ul style="list-style-type: none">• Implement Role-Based Access Control (RBAC) on the server-side for all sensitive endpoints like admin.php. Validate roles on every request.• Deny by default: Make all sensitive functions inaccessible unless explicitly allowed per role.• Session Validation: Ensure session tokens are tied to role-specific permissions before loading admin content.• Hide and protect admin endpoints: Do not rely solely on obscurity. Apply authentication and authorization checks regardless of whether the endpoint is linked in the UI.• Security Testing: Include forced browsing tests in regular pentest cycles using tools like Burp Suite, OWASP ZAP, or Gobuster.	
References	
https://owasp.org/Top10/A01_2021-Broken_Access_Control/	
https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html	
https://portswigger.net/web-security/access-control	

Steps to Reproduce:

1. Authenticate as a low-privilege user (e.g., "test") at: <https://cdn.codenimbus-systems.com/login.php>
2. Perform directory enumeration using a tool like Gobuster or Dirb: `gobuster dir -u https://cdn.codenimbus-systems.com/ -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt`
3. `admin.php` is discovered as an accessible file, along with restricted directories such as `.htpasswd/`.
4. Access the admin interface directly using the discovered endpoint: <https://cdn.codenimbus-systems.com/admin.php>
5. Observe behavior: Despite not being an admin, the page loads. The interface allows the user to view, edit, and toggle admin-level access for other users. No server-side session or privilege verification is triggered.
6. Modify user privileges using the admin panel UI: Toggle checkboxes for "Admin access" and click "Save Changes". Changes are applied even though the user lacks authorization. Confirm privilege escalation or unauthorized user manipulation.

Proof Of Concept:

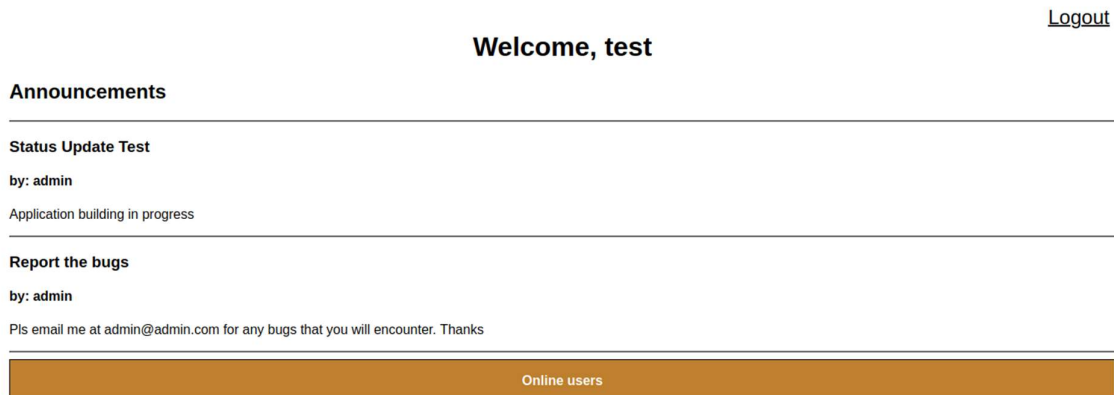


Image 3.1 : Authenticate as test

Type	Path	Response	Size
Dir	/	200	3724
Dir	/.htaccess/	403	448
File	/.htpasswd.php	403	448
File	/.htaccess.php	403	448
File	/login.php	200	2246
File	/index.php	200	3726
File	/script.js	200	1057
File	/jquery.min.js	200	89747
Dir	/.htpasswd/	403	448
File	/admin.php	302	307
Dir	/assets/	403	448
Dir	/assets/.htaccess/	403	448
File	/assets/.htpasswd.php	403	448
Dir	/assets/.htpasswd/	403	448
File	/assets/.htaccess.php	403	448

Image 3.2 : Enumerate directories

Here we find sensitive endpoints like admin.php, /.htpasswd.php

Welcome To Your Admin page, User

Logout

You can view the list of users who use VulnerableApp here. Select the respective checkboxes to delete a user or change their authorization. Click 'Save changes' to save changes made & 'Undo Changes' to reset.

Email	First Name	Last Name	Auth level	Delete	Admin access
			Admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>
			Admin	<input type="checkbox"/>	<input checked="" type="checkbox"/>


Save Changes

Undo Changes

Image 3.3 : After navigating to /admin.php

It's enough to navigate to /admin.php and it will give anyone the privilege as admin

4. Server-Side Request Forgery (SSRF) in URL Parameter

Reference No:	Risk Rating:
WEB_VUL_04	Critical 
CVSS:	CWE:
9.9: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:L	CWE-918: Server-Side Request Forgery (SSRF)
Tools Used:	
Browser	
Vulnerability Description:	
<p>The web application endpoint <code>http://hrms.thm/?url=</code> allows user input to be directly fetched and included server-side, without proper validation or filtering. This results in a Server-Side Request Forgery (SSRF) vulnerability, where an attacker can manipulate the server to make requests to internal resources (localhost, 127.0.0.1, or cloud metadata IPs like 169.254.169.254).</p> <p>This vulnerability is present in how PHP handles includes or file reads via URLs, particularly when used with functions like <code>file_get_contents()</code> or <code>include()</code> with external sources.</p>	
Vulnerability Identified by / How It Was Discovered	
<ul style="list-style-type: none">Manual testing: The url parameter was fuzzed with internal values like localhost/credit, localhost/config, etc.Behavioral confirmation: The server returned internal file contents (details, config, header.php) including hardcoded credentials in plain text.File enumeration: List of reachable files with status 200 (e.g., /constant, /details, /logout, /config) further confirmed backend access to files not exposed publicly.	
Vulnerable URLs / IP Address	
<code>https://Hrms.thm/</code>	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">Information Disclosure: Internal configuration files with sensitive information like admin credentials, tokens, and database connection strings are exposed.Pivoting for Internal Recon: An attacker can use SSRF to map the internal network or reach services not otherwise accessible from the internet (e.g., Redis, local APIs, AWS metadata).Bypass of Firewall or Whitelisting: Attackers can access otherwise protected services due to the trust placed on server-originated requests.Potential for RCE: If combined with other misconfigurations (e.g., internal service injection), SSRF may lead to full Remote Code Execution.Credential Exposure: As shown in localhost/config, hardcoded database credentials were revealed without authentication.	
Suggested Countermeasures	
<p>It is recommended to:</p> <ul style="list-style-type: none">Input Validation and WhitelistingDisallow Internal Address RangesDisable URL-based File IncludesUse Network SegmentationSet Up SSRF Protections at the Web Server LevelObfuscate or Move Sensitive FilesMonitor and Log Server-Side Requests	
References	
<p>https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html</p> <p>https://portswigger.net/web-security/ssrf</p>	

Proof Of Concept:

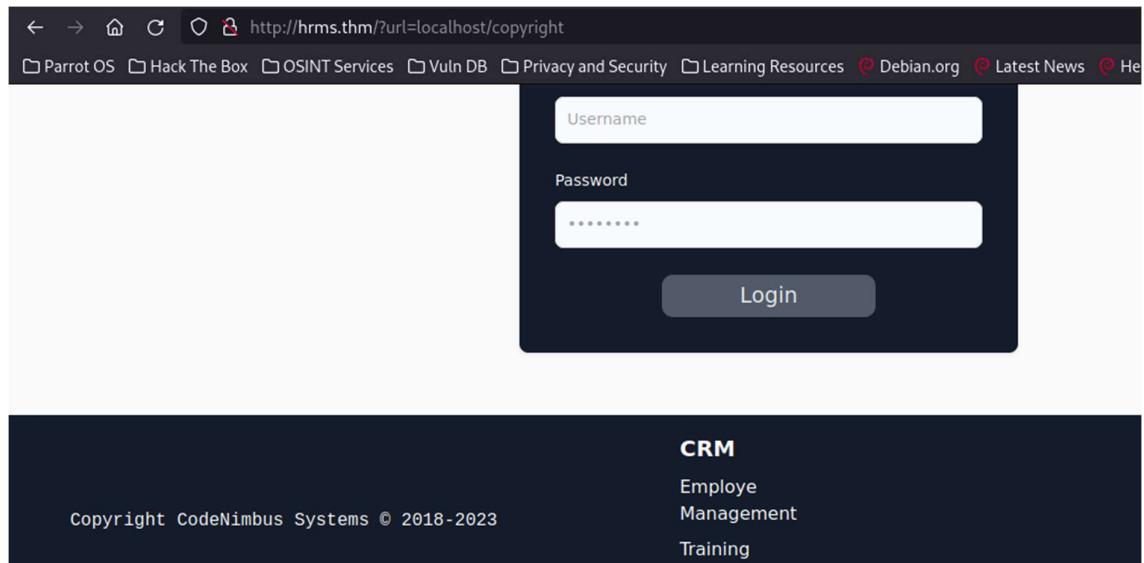


Image 4.1 : Exploring the domain

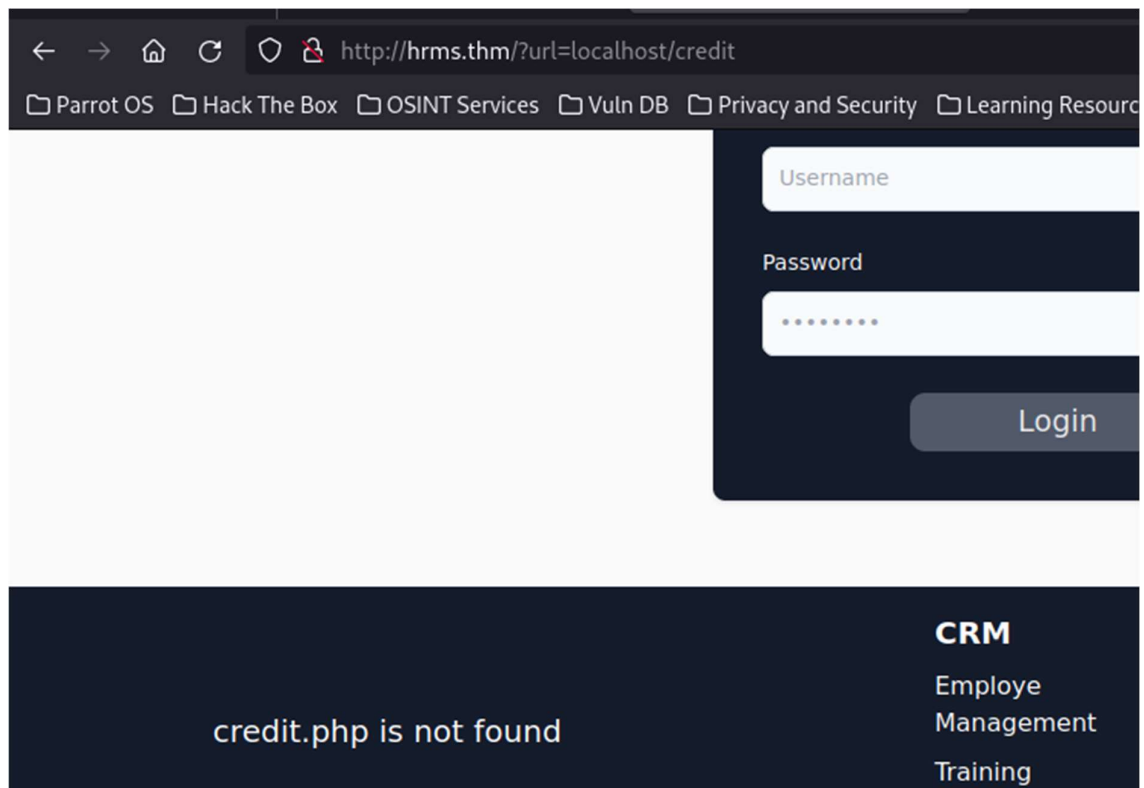


Image 4.2 : giving the parameter in the URL different value

Request	Payload	Status code	Response received	Error	Timeout	Length
7808	footer	200	86			13529
5995	details	200	101			12878
9563	index	200	105			12470
8808	header	200	79			11797
11077	logout	200	82			10642
5151	constant	200	88			10622
5117	config	200	146			10602
929	Database_Administration	200	89			10558
5733	database_administration	200	103			10558
932	Documents and Settings	200	71			10557
1897	administratoraccounts	200	91			10556
1030	ServerAdministrator	200	88			10554

Image 4.3 : Using Dirbuster automate values for url parameter

Then sort them by the length of the response

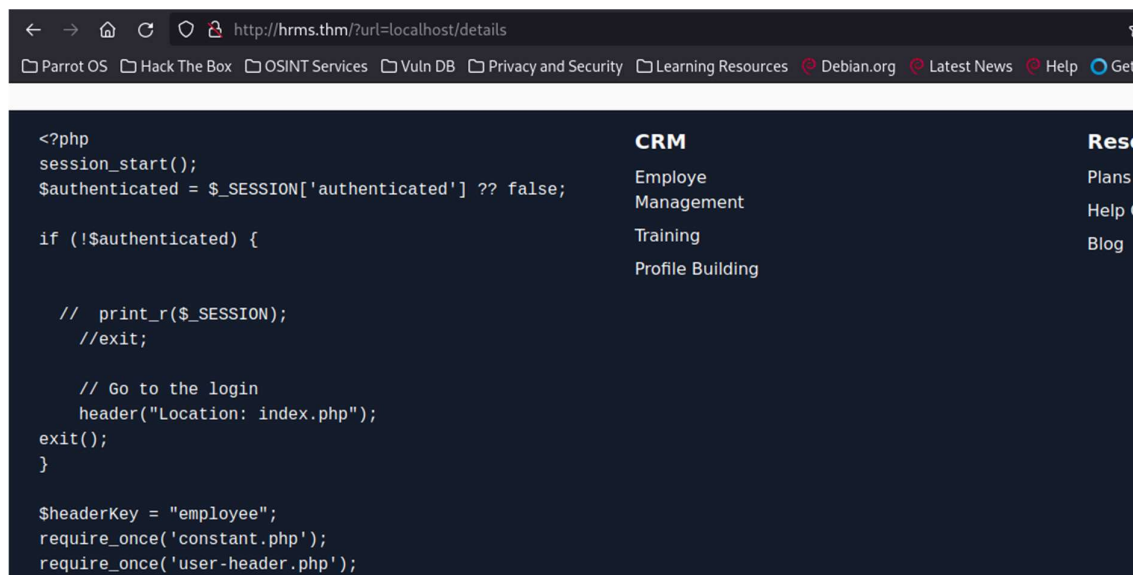


Image 4.4 : Confirm manual from dirbuster

Confirming the values that dirbuster showed us we can confirm the website is reflecting data from the local server by testing for details and it found it.

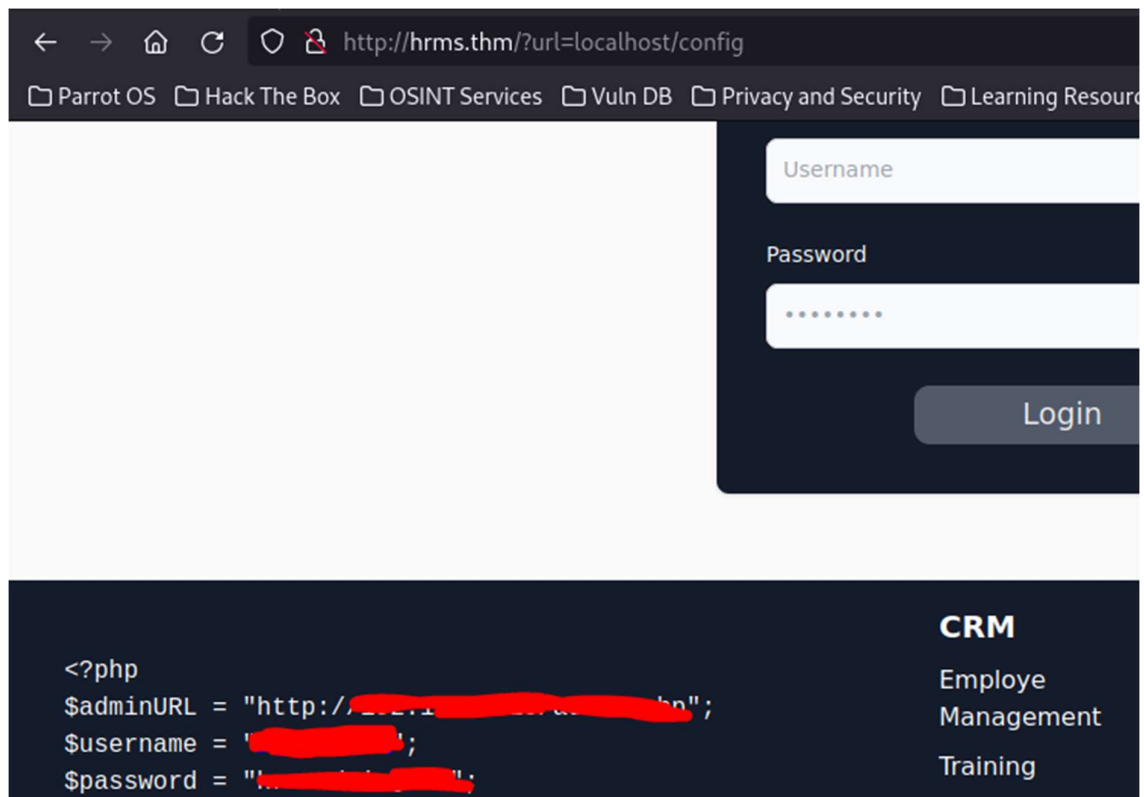



Image 4.5: Exposing sensitive information from the server like credentials

Also, it shows the admin login url inside the local server

5. Reflected XSS via k304 GET Parameter

Reference No:	Risk Rating:
WEB_VUL_05	Medium 
CVSS:	CWE:
5.4: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Tools Used:	
Browser	
Vulnerability Description:	
The k304 parameter in the URL is directly reflected into the page HTML or link without proper encoding. This enables attackers to inject JavaScript via payloads like <code></code> , causing scripts to execute in a victim's browser. This matches a known reflected XSS pattern	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://beta.codenimbus-systems.com	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Session hijacking: Attackers can steal cookies and impersonate users.• Phishing and fraud: Malicious overlays or redirects can capture credentials.• Malware delivery: Can inject hidden iframes or scripts for drive-by downloads.• Reduced customer trust: In-browser attacks erode user confidence and company reputation.	
Suggested Countermeasures	
It is recommended to: <ul style="list-style-type: none">• Output encoding: HTML-encode user input before insertion into the page (e.g., convert <code><</code> to <code>&lt;</code>).• Input validation: Reject or sanitize unexpected characters (<code><</code>, <code>></code>, <code>"</code>); whitelist allowed characters.• CSP headers: Implement a Content Security Policy to restrict inline script execution.• HttpOnly cookies: Ensure session cookies cannot be accessed via client-side scripts• Use secure templating frameworks: Employ frameworks that auto-escape user-generated content and reduce manual errors.	
References	
https://portswigger.net/web-security/cross-site-scripting	

Steps to Reproduce:

1. Navigate to the vulnerable URL: https://beta.codenimbus-systems.com/?k304=
2. The payload is rendered and executed, resulting in an alert popup as evidence.
3. This confirms the vulnerability is reflected.

Proof of concept:

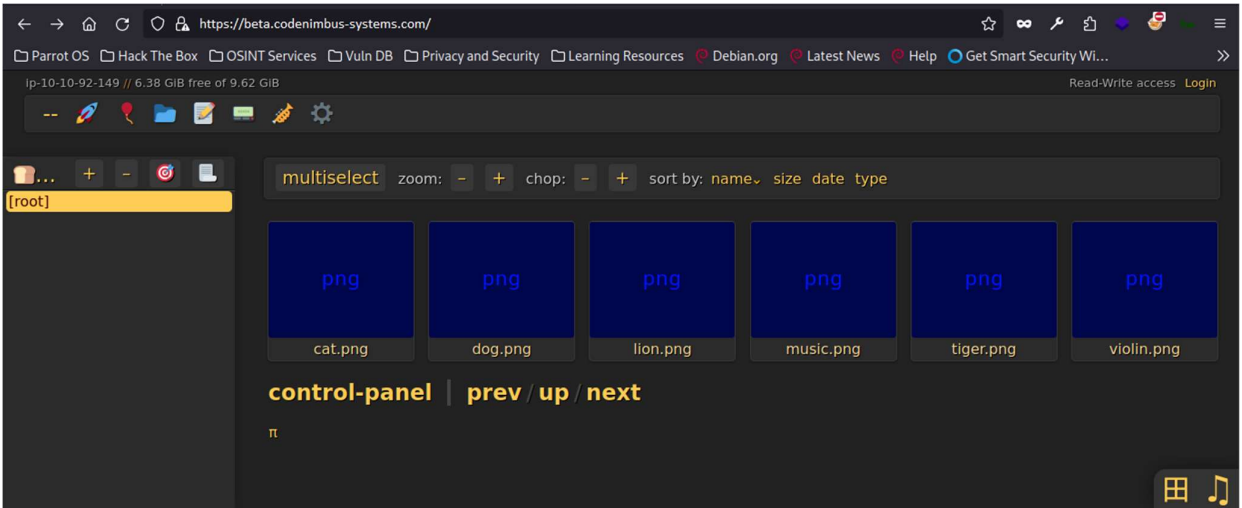


Image 5.1: Exploring the domain website

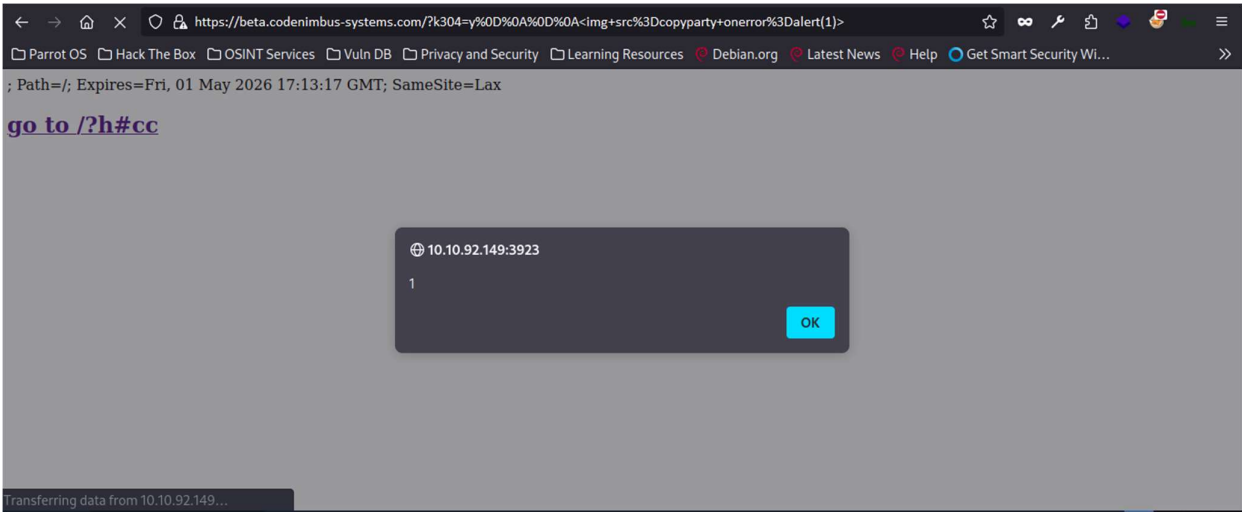



Image 5.2: Finding a Vulnerable Parameter to reflected XSS

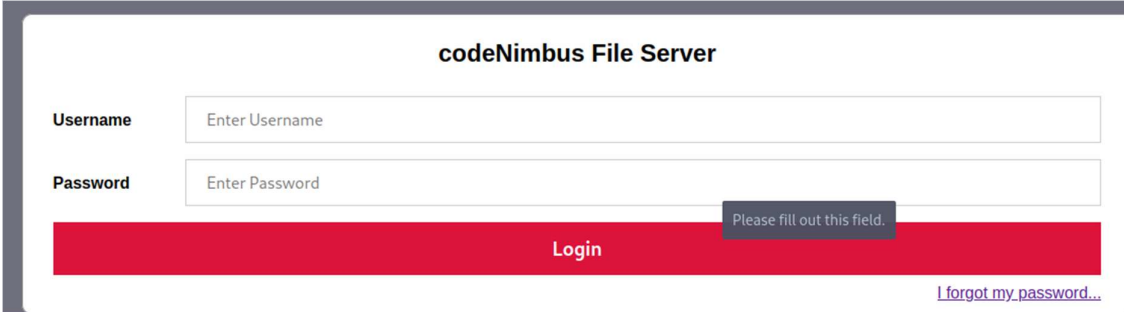
6. Insecure Design (Password Reset Mechanism)

Reference No:	Risk Rating:
WEB_VUL_06	Critical 
CVSS:	CWE:
10.0 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N	CWE-640: Weak Password Recovery Mechanism for Forgotten Password
Tools Used:	
Browser	
Vulnerability Description:	
The application implements an insecure password reset mechanism based on static security questions. By answering predefined questions (e.g., "What's your favourite colour?"), an attacker can reset the password of any known user without verifying identity through secure means such as email or 2FA.	
Vulnerability Identified by / How It Was Discovered	
Manually discovered during black-box testing. After initiating a password reset for a known user, the application allowed password reset by submitting trivial or guessable answers to security questions, bypassing proper identity validation.	
Vulnerable URLs / IP Address	
https://cdn.codenimbus-systems.com/	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Unauthorized access to user accounts and private data• Full compromise of any user account, including potential admins• Violation of data privacy and trust• Regulatory non-compliance (e.g., GDPR breach)	
Suggested Countermeasures	
It is recommended to: <ul style="list-style-type: none">• Completely remove insecure security questions from the reset flow• Implement password reset via secure email token-based verification• Add multi-factor authentication (MFA) where possible• Log and alert password reset attempts• Use rate limiting to prevent brute-force guessing of answers	
References	
https://owasp.org/Top10/A04_2021-Insecure_Design/ https://cwe.mitre.org/data/definitions/640.html https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html	

Steps to Reproduce:

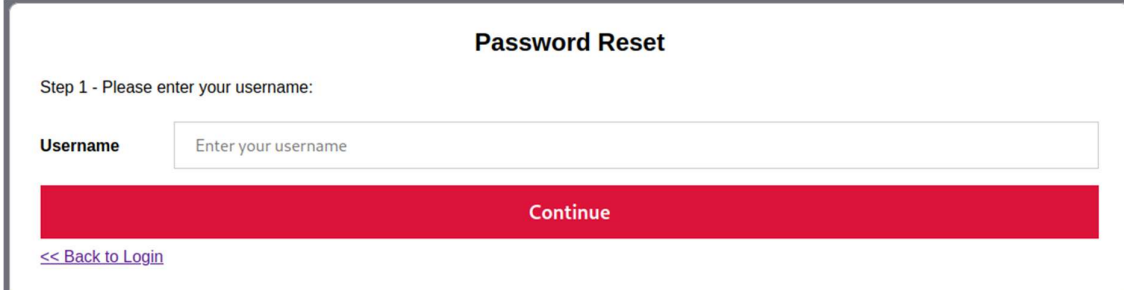
1. Navigate to the File Server Login Page, Open the login page:
<https://cdn.codenimbus-systems.com/login>
2. Click on “I forgot my password...” This will take you to the password reset flow.
3. Enter a Target Username Provide a valid username (e.g., joseph) in the username field to trigger the security question step.
4. Enumerate and Guess Security Questions Observe that the system uses static or easily guessable security questions like: “What’s your favourite colour?” “What’s your mother’s sister’s son’s neighbour’s friend’s name?”
5. Submit Common Answers (e.g., “blue”) to guess the correct answers.
6. Reset the Target’s Password, upon correct submission, the system displays the new password in plain text on screen: The password for user joseph has been reset to xxxxxx
7. Login as the Victim User Return to the login page, enter the victim’s username and the newly displayed password.

Proof of concept:



The screenshot shows the 'codeNimbus File Server' login interface. It features a title 'codeNimbus File Server' at the top. Below the title are two input fields: 'Username' with a placeholder 'Enter Username' and 'Password' with a placeholder 'Enter Password'. A red 'Login' button is positioned below the password field. A tooltip 'Please fill out this field.' is visible over the password field. At the bottom right, there is a link 'I forgot my password...'. The entire form is enclosed in a light gray border.

Image 6.1: Found a login form



The screenshot shows the 'Password Reset' page. The title 'Password Reset' is at the top. Below the title, it says 'Step 1 - Please enter your username:'. There is a 'Username' label and an input field with a placeholder 'Enter your username'. A red 'Continue' button is located below the input field. At the bottom left, there is a link '<< Back to Login'. The entire form is enclosed in a light gray border.

Image 6.2: Testing the I forgot my password option

Password Reset

Step 2 - Please answer one of your security questions to confirm your identity:

Security Question

What's your mother's sister's son's nephew's neighbour's friend name? ▾

Answer

Continue

Image 6.3: Trying to guess security questions

Password Reset

Step 2 - Please answer one of your security questions to confirm your identity:

Security Question

What's your favourite colour? ▾

Answer

Continue

Image 6.4: more investigating from the previous step

Password Reset

Success: The password for user joseph has been reset to CzTH1Nm7LeXhg

[<< Back to Login](#)

Image 6.5: After answering correctly on the question we've one-time use password

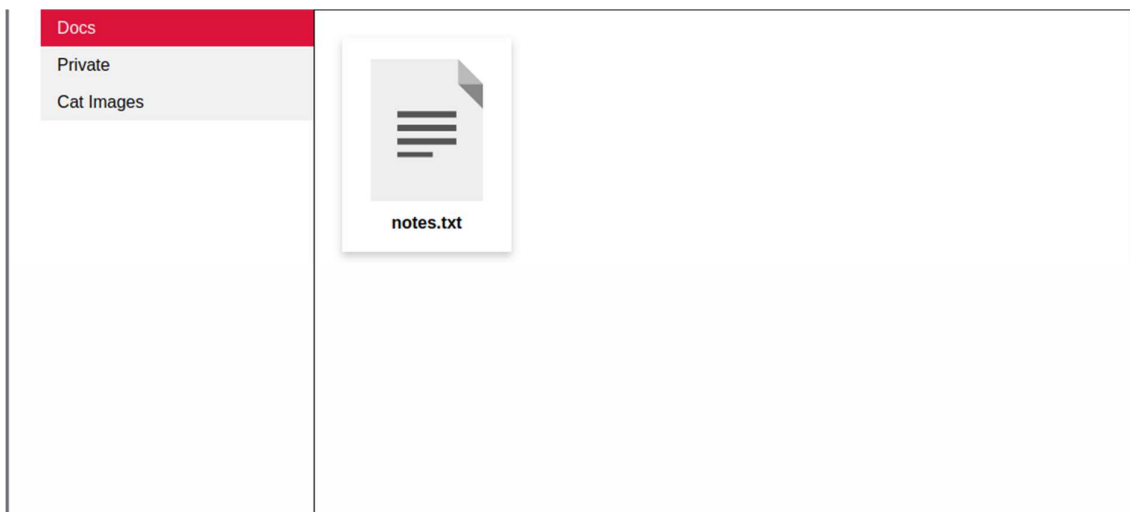



Image 6.6: any account is accessible by guessing the security question

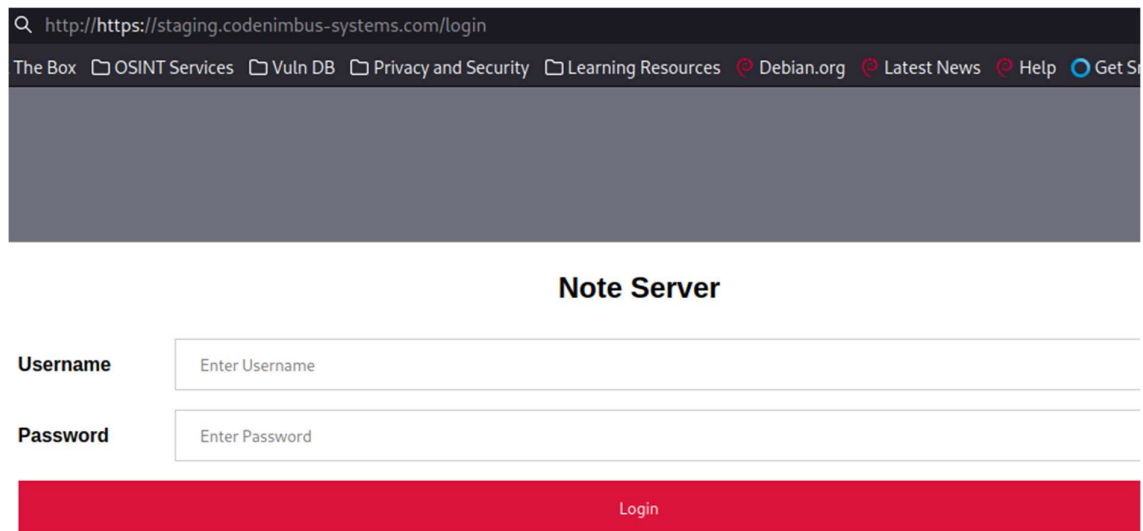
7. IDOR - View Other Users' Notes

Reference No:	Risk Rating:
WEB_VUL_07	High 
CVSS:	CWE:
7.1: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N	CWE-639: Authorization Bypass Through User-Controlled Key
Tools Used:	
Browser	
Vulnerability Description:	
The application fails to implement proper access control on the note_id parameter of the notes view page. This allows authenticated users to enumerate and access other users' private notes by incrementing or guessing note IDs, resulting in unauthorized data exposure.	
Vulnerability Identified by / How It Was Discovered	
Discovered manually during authenticated browsing of the application. By modifying the note_id in the URL, data belonging to other users was accessed, confirming an IDOR condition.	
Vulnerable URLs / IP Address	
https://staging.codenimbus-systems.com/	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Leakage of sensitive or personal information across user accounts• Violation of user privacy and potential GDPR or HIPAA non-compliance• Trust damage and legal exposure for the organization• May facilitate further privilege escalation or reconnaissance	
Suggested Countermeasures	
It is recommended to: <ul style="list-style-type: none">• Implement strict authorization checks on server side to ensure users can only access their own data• Avoid using incremental IDs in URLs; use UUIDs or access tokens when possible• Log and alert any unauthorized data access attempts• Include automated tests for object-level access control	
References	
https://cwe.mitre.org/data/definitions/639.html https://owasp.org/Top10/A01_2021-Broken_Access_Control/	

Steps to Reproduce:

1. Login to the Note Server at: <https://staging.codenimbus-systems.com/login>
2. Once logged in, access your own notes through a URL like:
https://staging.codenimbus-systems.com/note.php?note_id=1
3. Manually modify the note_id parameter in the URL (e.g., note_id=2, note_id=3).
4. Observe that notes belonging to other users are displayed without any authorization checks.
5. This confirms that access control is missing or improperly enforced on this endpoint.

Proof of concept:



http://https://staging.codenimbus-systems.com/login

The Box OSINT Services Vuln DB Privacy and Security Learning Resources Debian.org Latest News Help Get S

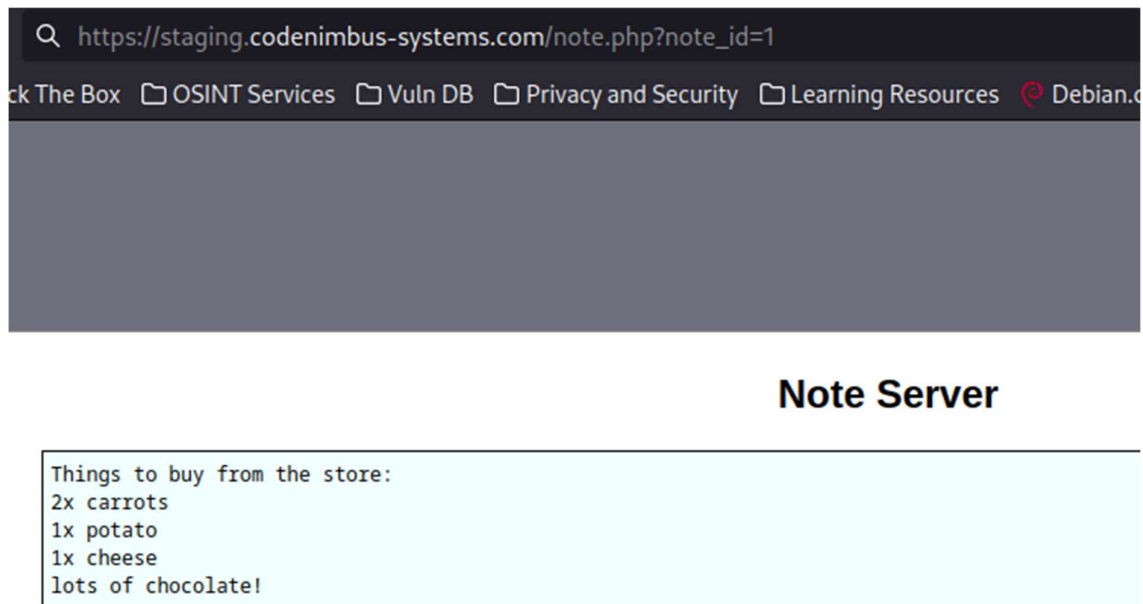
Note Server

Username

Password

Login

Image 7.1: Login form to note server



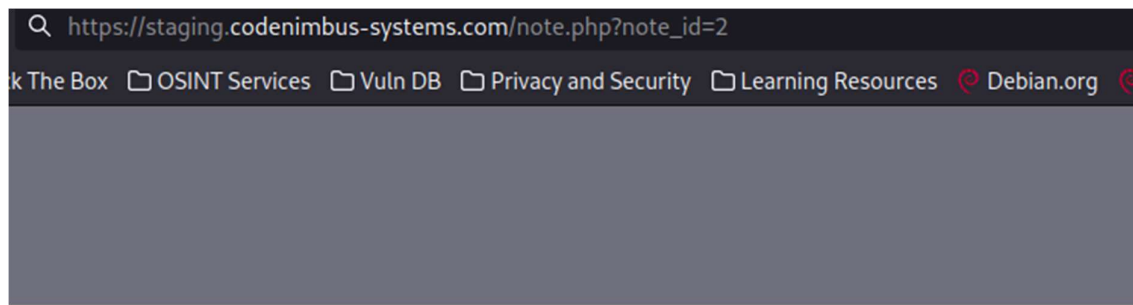
https://staging.codenimbus-systems.com/note.php?note_id=1

The Box OSINT Services Vuln DB Privacy and Security Learning Resources Debian.org

Note Server

Things to buy from the store:
2x carrots
1x potato
1x cheese
lots of chocolate!

Image 7.2: after authentication shows my note




Note Server

```
2022-09-10: Meeting with Sophie S.  
2022-09-12: Lunch at Manny's  
2022-09-27: Stu's birthday
```

Image 7.3: other notes are exposed

By changing the value of the parameter `note_id=<number>` I can expose other notes for other users without being authenticated

8. Autocomplete-enabled Password Field

Reference No:	Risk Rating:
WEB_VUL_08	Low 
CVSS:	CWE:
2.4: CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	CWE-524: Use of Web Browser Auto-Complete Functionality for Sensitive Input
Tools Used:	
Burp Suite	
Vulnerability Description:	
The application's login form does not disable browser autocomplete functionality for password fields. As a result, modern browsers may offer to store and autofill the password, potentially exposing stored credentials on shared or compromised machines.	
Vulnerability Identified by / How It Was Discovered	
Identified automatically using Burp Suite during static HTML analysis of the login form. The <input type="password"> field was found without an autocomplete="off" attribute.	
Vulnerable URLs / IP Address	
https://beta.codenimbus-systems.com	
Implications / Consequences of not Fixing the Issue	
<ul style="list-style-type: none">• Stored credentials may be exposed on shared computers or to malware with browser access.• May violate internal security policies or compliance standards (e.g., ISO 27001).• Enables credential theft through local access or info-stealer malware.	
Suggested Countermeasures	
It is recommended to: <ul style="list-style-type: none">• Add the attribute autocomplete="off" to both the <form> and <input> elements• Apply this to all login and sensitive forms across the application.• Educate developers on secure form design practices.	
References	
https://cwe.mitre.org/data/definitions/639.html https://owasp.org/Top10/A01_2021-Broken_Access_Control/	

Steps to Reproduce:

1. Access the login page (or any form with a password field).
2. View the HTML source or intercept with Burp Suite.
3. Locate the <input> tag: <input type="password" placeholder="Enter Password" name="pass" required>
4. Observe that there is no autocomplete="off" attribute set.
5. Login using any credentials and allow the browser to prompt for saving the password.
6. Log out, return to the login page, and verify that the browser autofills the credentials, confirming that autocomplete is enabled.

Proof of concept:

Request:

```
GET / HTTP/1.1
Host: https://beta.codenimbus-systems.com
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/122.0.6261.112 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response:

```
HTTP/1.1 200 OK
Date: Tue, 08 Jul 2025 10:45:43 GMT
Server: Apache/2.4.54 (Unix)
X-Powered-By: PHP/8.0.21
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 2673
Connection: close
Content-Type: text/html; charset=UTF-8
....
<div id="id01" class="modal">
```

```
<form class="modal-content animate" action="." method="post">
  <div class="imgcontainer">
    
    <h2>THM Note Server</h2>
  </div>
```

```
<div class="container">
  <label for="uname"><b>Username</b>
    <input type="text" placeholder="Enter Username" name="user" required>
  </label>
  <label for="psw"><b>Password</b>
    <input type="password" placeholder="Enter Password" name="pass" required>
  </label>
```

```
.....
```