

WEB APPLICATION PENETRATION TESTING REPORT

Prepared by:

Odoyi Faith Chizubem (odoyichizubem@yahoo.com)

Presented to:

DEMO CORP

Date: 13th / May / 2024

Testing Summary

Manual and automated methods were deployed for the assessment and various vulnerabilities were found ranging from SQL injection which summary can be found in (findings 001), Burp Suite, a proxy tool, and sqlmap were used to carry out this vulnerability exploitation. The web application is vulnerable to SQL injection attacks, which can allow malicious actors to execute arbitrary SQL queries and potentially gain unauthorized access to the underlying database. The database information was revealed with sqlmap showing database column and table names, also revealing username, password, and vault password's in hash which can be cracked and users' accounts compromised.

Stored cross-site scripting (findings 005) which is a critical vulnerability was also found when updating / adding vault information. The presence of stored cross-site scripting vulnerabilities enables attackers to inject malicious scripts into the web application, which can then be executed within the context of other users' sessions; this vulnerability can spread across the database by just sharing the vault key to different victims which can lead to theft of sensitive information or unauthorized actions, depending on the intent of the malicious actor. Some other vulnerabilities of critical, high, medium, low, and informational rating were also discovered which can be found in the technical findings of this report.

Test Note and Recommendations

Secure the web application against SQL injection attacks by using parameterized queries or prepared statements to handle user input securely.

Implement strict input validation and output encoding to mitigate stored cross-site scripting vulnerabilities and prevent malicious script injection.

Address the username confusion vulnerability by improving the authentication mechanism to prevent bypass techniques such as username manipulation.

Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.

Install and configure a Web Application Firewall (WAF) to provide additional security controls and mitigate various web-based threats, including SQL injection and cross-site scripting attacks.

Require users to provide non-empty passwords during authentication and enforce strong password policies to enhance security against brute force attacks and unauthorized access.

Implement measures such as consistent error messages to mitigate username enumeration vulnerabilities and prevent attackers from identifying valid user accounts.

Configure the web server to enforce HTTPS encryption to protect sensitive data and user credentials transmitted over the network, thereby preventing interception and unauthorized access.

Key Strength and Weakness

The following identifies the key strength identified in the assessment

1. No information leakage on the comment of the source code
2. File directory traversal vulnerability was not found but is not guaranteed that the web application is free from directory traversal vulnerability
3. No disclosure of secret key in the source code.

The following identifies the key weakness identified in the assessment

1. No input validation
2. No input sanitization
3. Rate limiting was not implemented
4. 2MFA was not implemented to mitigate account brute-force attack
5. Username confusion vulnerability was present
6. Web Application Firewall was not implemented.
7. No https encryption security
8. Broken access control

Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediation

Web Penetration Test Findings

4	8	2	N/A	1
Critical	High	Medium	Low	Information

Findings	Severity	Recommendation
SQL Injection Vulnerability	Critical	Input Validation and Sanitization: Validate and sanitize all user-supplied data before incorporating it into SQL queries.
Broken Authentication via cookie manipulation	Critical	Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login.
Broken access control	Critical	Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
Broken access control	Critical	Implement access control checks whenever accessing or modifying sensitive objects or resources.
Stored Cross Site Scripting	high	Input Validation and Sanitization: Validate and sanitize all user-supplied input before storing it on the server or displaying it to other users.
DOM Based Cross Site Scripting	high	Context-Aware Escaping: Escape user input based on its context within the DOM to prevent script execution
JWT Token Bypass	high	Implement strict validation of JWT tokens, including checking the token's integrity and expiration
Password Authentication Check Bypass	High	Implement validation checks to prevent users from setting their password to an empty value during account creation, password reset, or password change processes.

Username Confusion Vulnerability	High	Enforce uniqueness constraints for usernames to prevent users from registering accounts with similar usernames.
No HTTPS Encryption Security	High	Obtain an SSL/TLS certificate and configure your web server to use HTTPS encryption.
Lack of Rate Limiting	High	Implement multi-factor authentication (MFA) and CAPTCHA challenges to strengthen authentication and deter brute force attacks.
Clear text submission of password	High	Transport Layer Security (TLS): Encrypt network traffic using HTTPS (HTTP over SSL/TLS) to ensure data confidentiality and integrity during transmission.
Weak Password Policy	Medium	Requiring passwords to meet complexity requirements.
Username Enumeration Vulnerability	Medium	Ensure that the application displays generic error messages for both valid and invalid usernames to prevent attackers from distinguishing between them.
No WAF Protection on the Webserver	Informational	Consider leveraging cloud-based WAFaaS solutions offered by various providers.

TECHNICAL FINDINGS

Web Penetration Test Findings

Findings 001: SQL Injection Vulnerability

Description:	SQL Injection vulnerability was found during the web application penetration testing which allowed the dumping of database information's
Risk:	<ul style="list-style-type: none"> • Data Leakage: Attackers can extract sensitive data such as usernames, passwords, credit card numbers, and personal information. • Data Manipulation: Attackers can modify, delete, or insert unauthorized data into the database.
Affected Functionality	http://XX.XX.XX.XX/api/vault/:token
Tool Used	Burp Suit, Sqlmap and Firefox browser
References	https://cwe.mitre.org/data/definitions/564.html https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Proof of Concept

To carry out this injection, you have to generate your token by using the following curl command as specified on the API documentation

```
curl -X POST -d '{"username":"alex","password":"alex"}' -H 'Content-Type: application/json' http://<host>/api/token
```



```
$ curl -X POST -d '{"username":"", "password":""}' -H 'Content-Type: application/json' http://10.0.0.10/api/token
{"token":"cc97955f-c74c-5b4d24b90401"}
```

Figure 1 screenshot showing how account token is generated using the provided api endpoint

With the access token generated, access the endpoint as shown below

Burp Suite Request and response on the end point <http://XX.XX.XX.XX/api/vault/token>

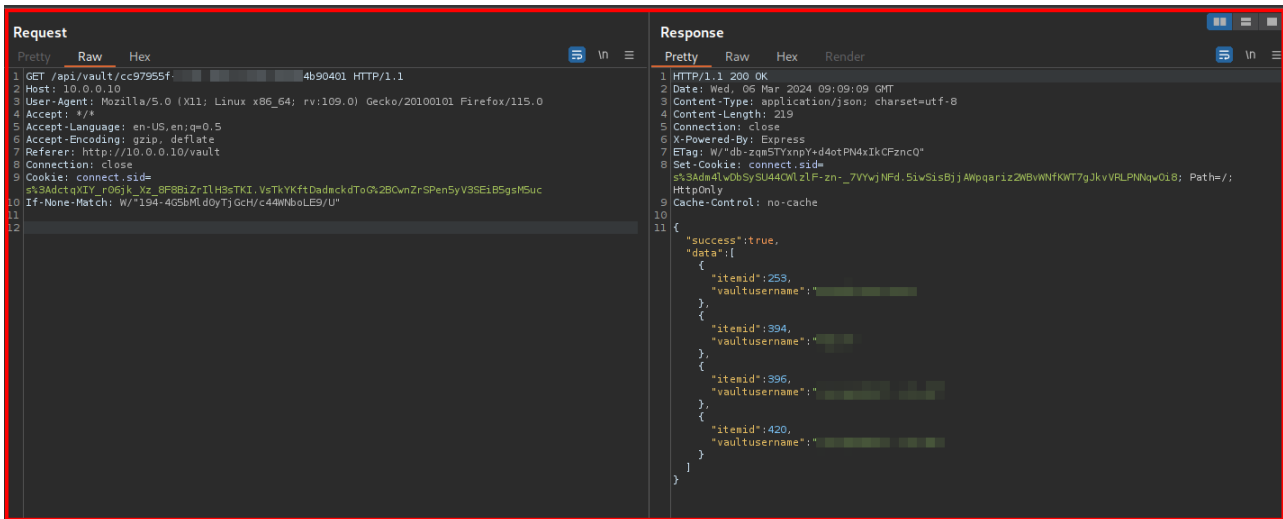


Figure 2 Screenshot showing request and response from the above mentioned api endpoint

Save the request in a file so can use sqlmap to carry out the injection attack as show below

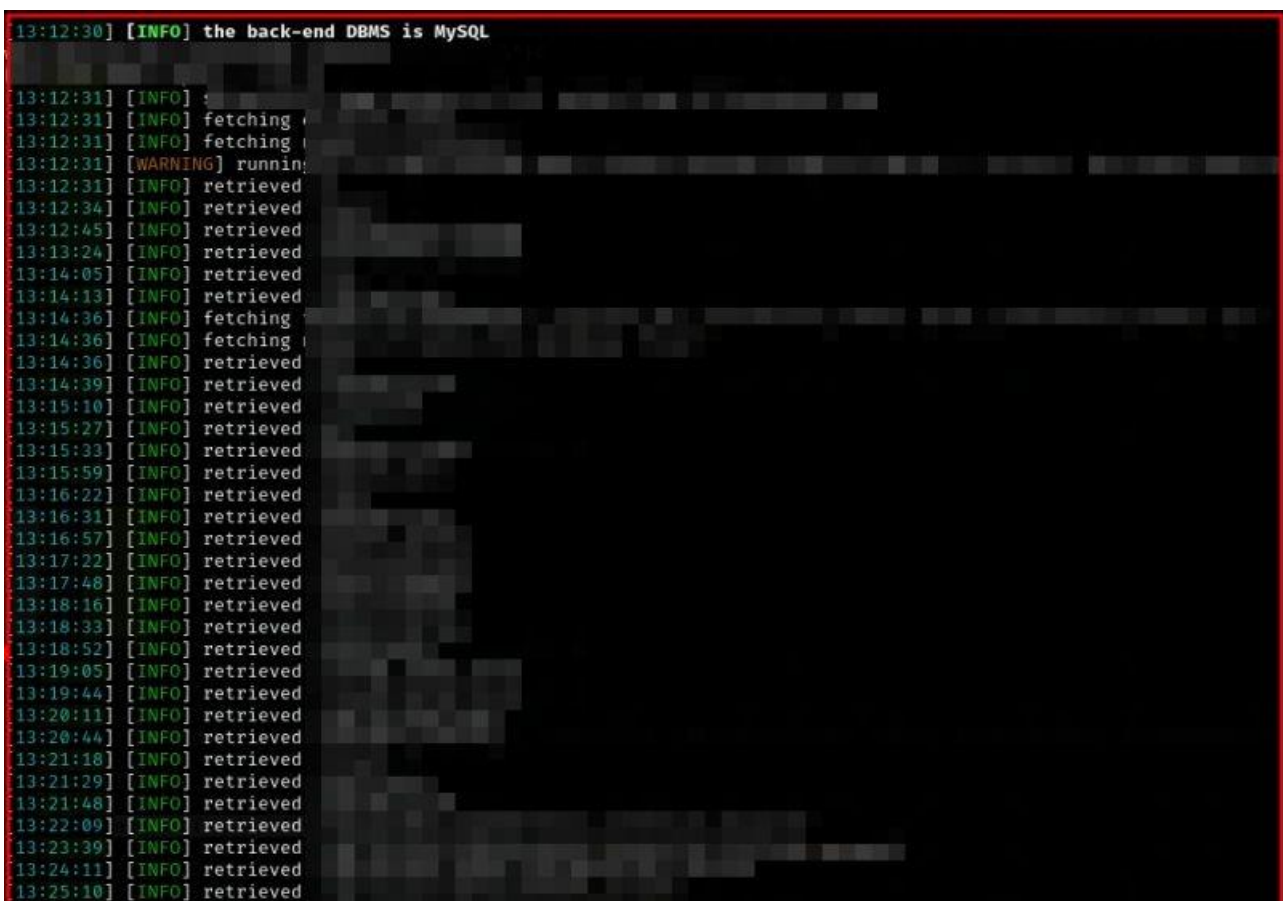
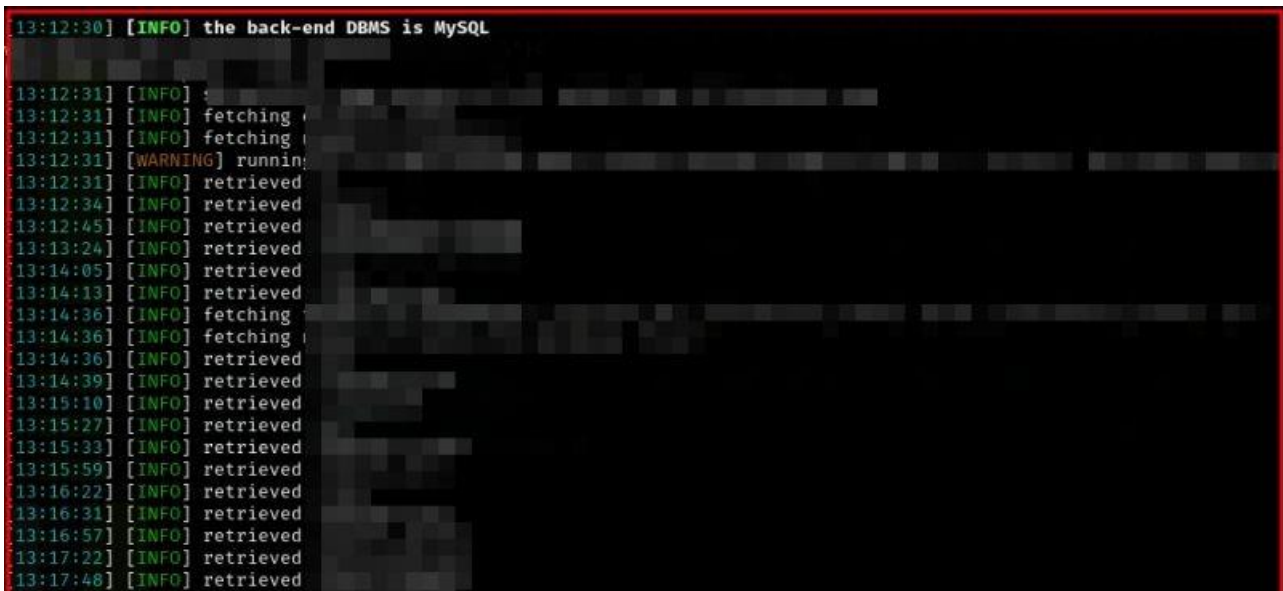


Figure 3 Using sqlmap to automate sql injection attack on the api endpoint as specified above

Sqlmap tool has done the job by dumping the database information

```
13:12:30] [INFO] the back-end DBMS is MySQL
13:12:31] [INFO] :
13:12:31] [INFO] fetching
13:12:31] [INFO] fetching
13:12:31] [WARNING] running
13:12:31] [INFO] retrieved
13:12:34] [INFO] retrieved
13:12:45] [INFO] retrieved
13:13:24] [INFO] retrieved
13:14:05] [INFO] retrieved
13:14:13] [INFO] retrieved
13:14:36] [INFO] fetching
13:14:36] [INFO] fetching
13:14:36] [INFO] retrieved
13:14:39] [INFO] retrieved
13:15:10] [INFO] retrieved
13:15:27] [INFO] retrieved
13:15:33] [INFO] retrieved
13:15:59] [INFO] retrieved
13:16:22] [INFO] retrieved
13:16:31] [INFO] retrieved
13:16:57] [INFO] retrieved
13:17:22] [INFO] retrieved
13:17:48] [INFO] retrieved
13:18:16] [INFO] retrieved
13:18:33] [INFO] retrieved
13:18:52] [INFO] retrieved
13:19:05] [INFO] retrieved
13:19:44] [INFO] retrieved
13:20:11] [INFO] retrieved
13:20:44] [INFO] retrieved
13:21:18] [INFO] retrieved
13:21:29] [INFO] retrieved
13:21:48] [INFO] retrieved
13:22:09] [INFO] retrieved
13:23:39] [INFO] retrieved
13:24:11] [INFO] retrieved
13:25:10] [INFO] retrieved
```

Figure 4 SQL injection POC on the <http://api/vaul/token> endpoint



```
13:12:30] [INFO] the back-end DBMS is MySQL
13:12:31] [INFO] :
13:12:31] [INFO] fetching
13:12:31] [INFO] fetching
13:12:31] [WARNING] running
13:12:31] [INFO] retrieved
13:12:34] [INFO] retrieved
13:12:45] [INFO] retrieved
13:13:24] [INFO] retrieved
13:14:05] [INFO] retrieved
13:14:13] [INFO] retrieved
13:14:36] [INFO] fetching
13:14:36] [INFO] fetching
13:14:36] [INFO] retrieved
13:14:39] [INFO] retrieved
13:15:10] [INFO] retrieved
13:15:27] [INFO] retrieved
13:15:33] [INFO] retrieved
13:15:59] [INFO] retrieved
13:16:22] [INFO] retrieved
13:16:31] [INFO] retrieved
13:16:57] [INFO] retrieved
13:17:22] [INFO] retrieved
13:17:48] [INFO] retrieved
```

Figure 5: More proof of concept of database information as dumped by sqlmap tool

Remediation:

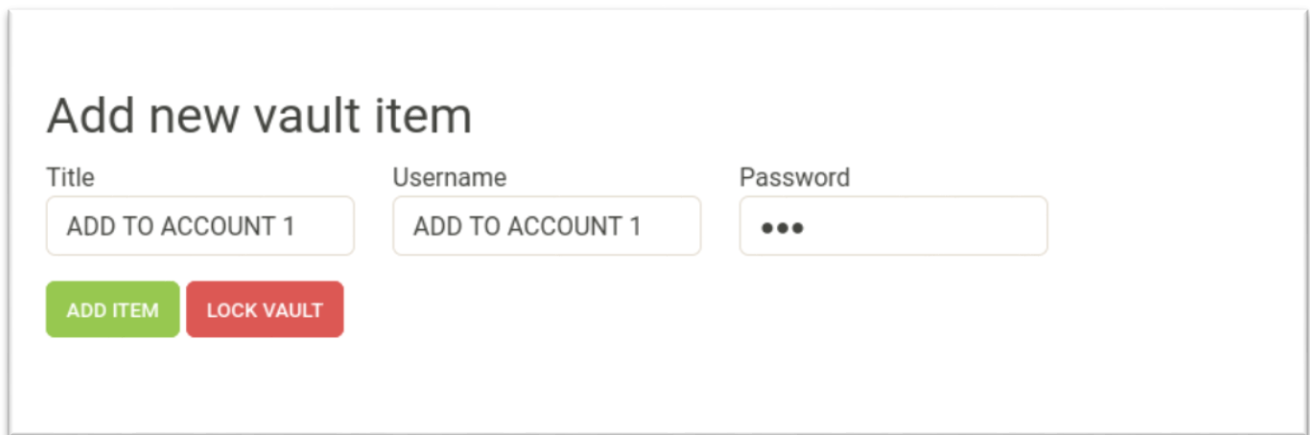
- Use Parameterized Queries: Utilize prepared statements or parameterized queries with bound parameters.
- Input Validation and Sanitization: Validate and sanitize all user-supplied data before incorporating it into SQL queries.
- Principle of Least Privilege: Limit database user privileges to only those required for specific tasks.
- Web Application Firewall (WAF): Employ WAFs to detect and block SQL injection attempts.
- Regular Security Audits: Conduct routine security assessments and penetration testing to identify and mitigate vulnerabilities.

Findings 002: Broken Authentication via cookie manipulation

Description:	Ability to add data on another users vault by intercepting and manipulating the cookie ssid
Risk:	<ul style="list-style-type: none"> Once authenticated, attackers may exploit flaws in the authentication process to escalate their privileges within the system, gaining access to sensitive functionalities or data they're not authorized to see. Data Manipulation: Attackers can modify, delete, or insert unauthorized data into another users account
Affected Functionality	http://XX.XX.XX.XX/vault/add
Tool Used	Burp Suit, and Firefox browser
References	https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/README

Proof of concept

- Create two different account
- login and take note of the both cookie ssid
- authenticate to both account in the same browser using Firefox multi account container addon
- On account 2 unlock the vault to add vault item details



Add new vault item

Title: ADD TO ACCOUNT 1

Username: ADD TO ACCOUNT 1

Password: ...

ADD ITEM LOCK VAULT

Figure 6: Screenshot showing vault account details being added

Intercept The Request using Burp suite

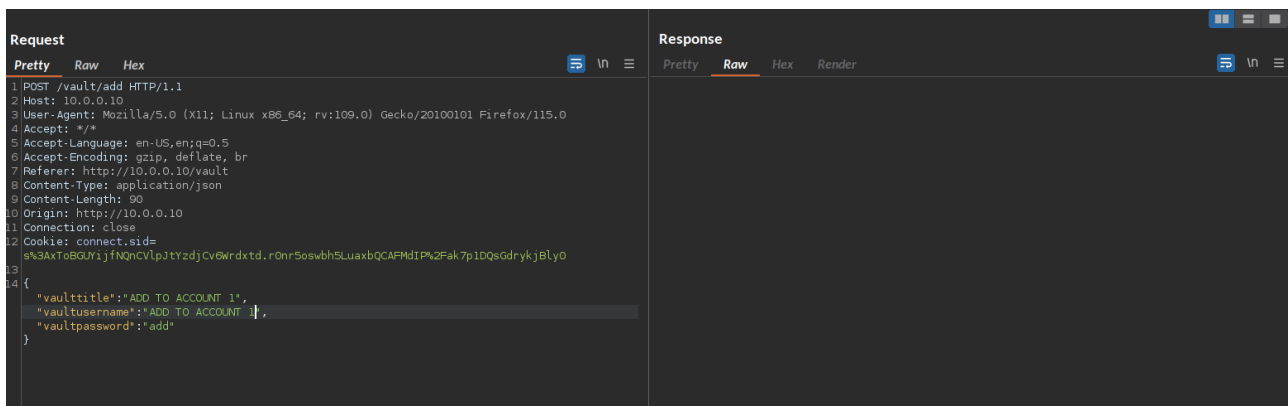


Figure 7: Screenshot showing intercepted POST request to 10.0.0.10/vault/add endpoint

Send the request to repeater to manipulate the cookie value by changing it to account one cookie

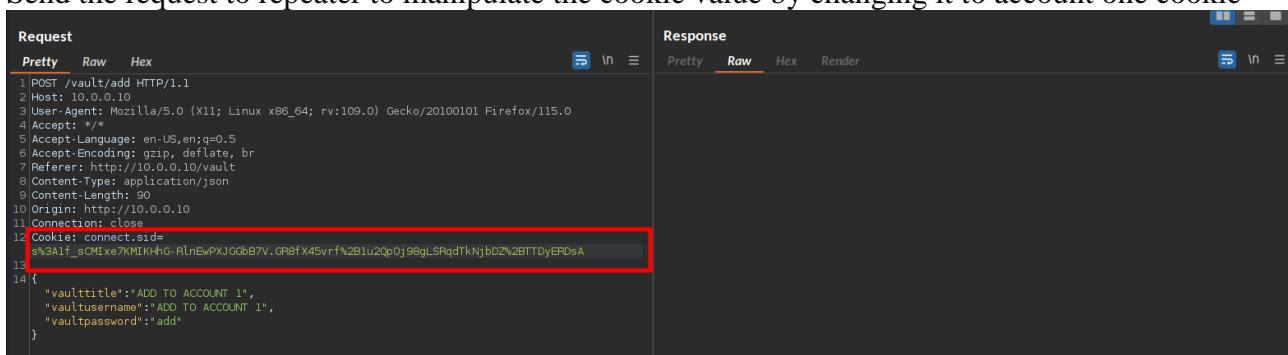


Figure 8: Screenshot show manipulated cookie value from account two to account one cookie

Forward The Request to get a 200 Ok Response

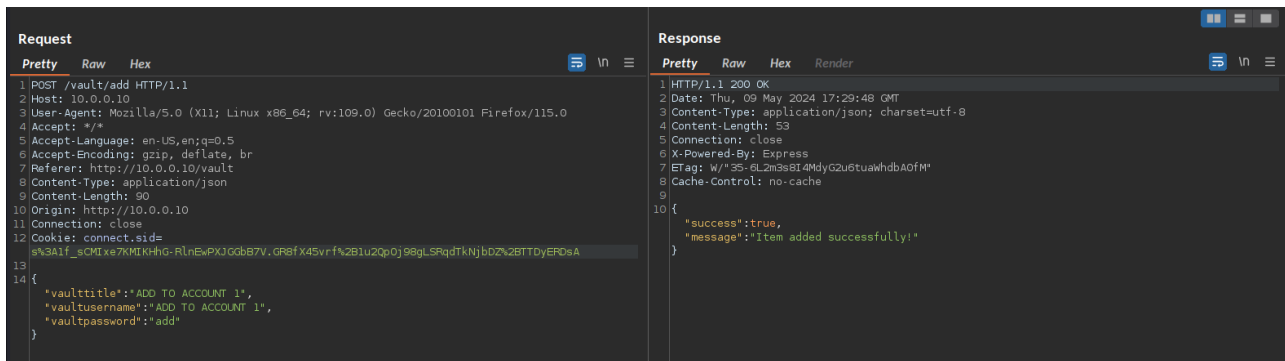


Figure 9: Screenshot showing forwarded request after manipulating the cookie value

The broken access control vulnerability was exploited... view your account one vault to see the added vault item from account 2

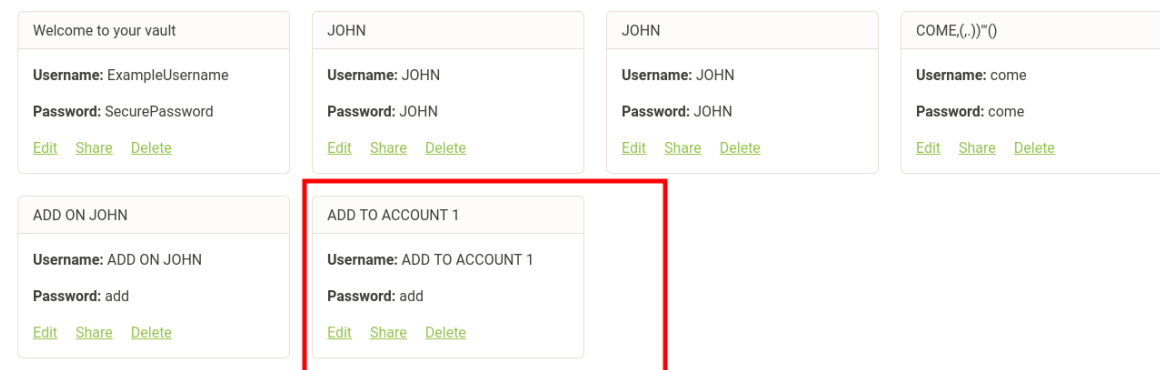


Figure 10: Screenshot showing successful exploitation

Remediation

Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

Implement proper access control lists to restrict access to resources based on user roles and permissions. Ensure that users only have access to the resources and functionalities they need to perform their tasks.

Findings 003: Broken access control on <http://XX.XX.XX.XX/vault/add>

Description:	This vulnerability allows at attacker to delete vault item from another users vault
Risk:	<ul style="list-style-type: none"> • Unauthorized Data Access: Attackers can exploit broken access controls to gain access to sensitive data they are not authorized to view, such as personally identifiable information (PII), financial records, or intellectual property. • Data Tampering: Without proper access controls, attackers may be able to manipulate or modify data stored within the system, leading to data corruption, integrity breaches, or fraudulent activities.
Affected Functionality	http://XX.XX.XX.XX/vault/add
Tool Used	Burp Suit, and Firefox browser
References	https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README

Proof of concept

create two different account and login to both account using Firefox multi account container

On account two add a vault item and note the item id via burp request and response on <http://XX.XX.XX.XX/vault/add>

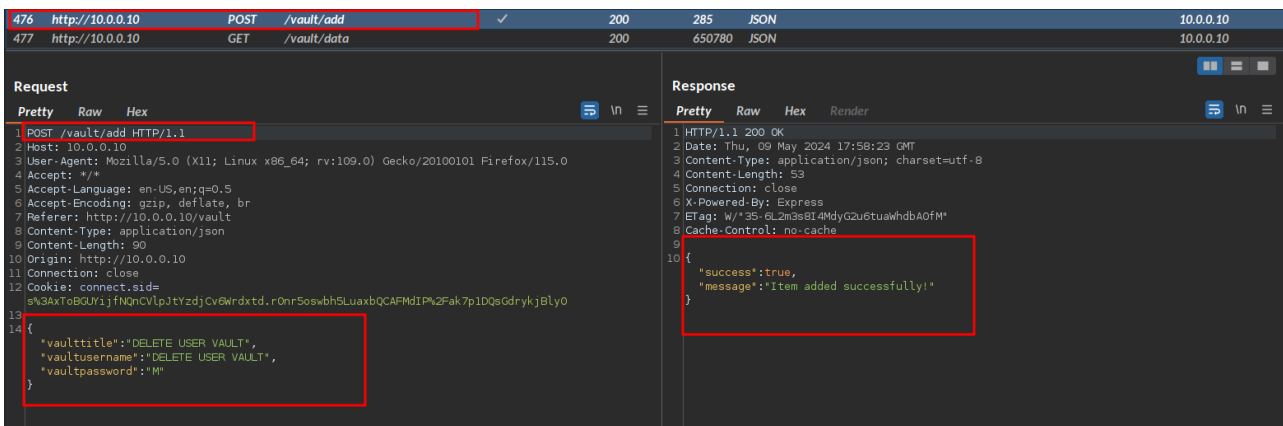


Figure 11: Screenshot show successful added vault item

Observe the next request and response to take note of the “itemid” via <http://XX.XX.XX.XX/vault/data>

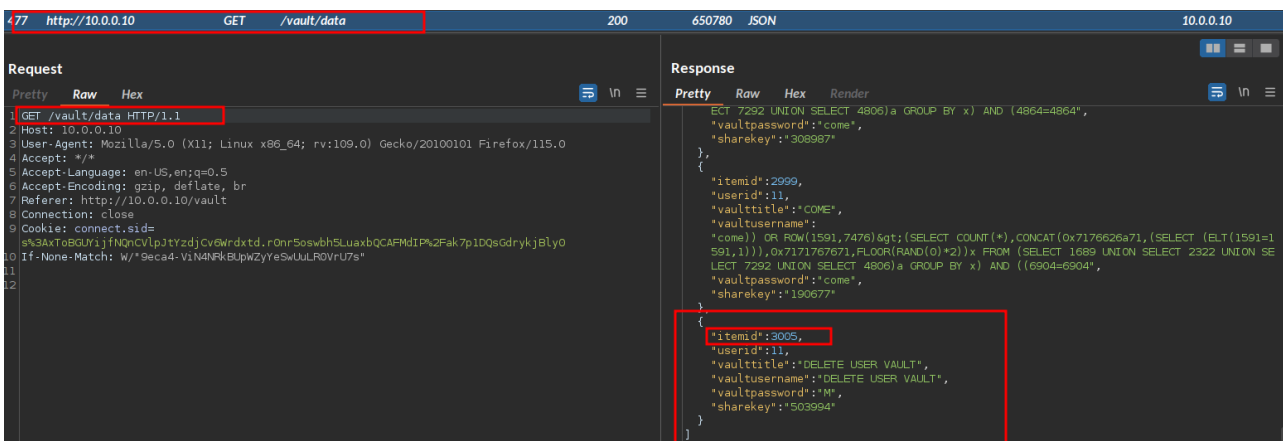


Figure 12: Screenshot showing added vault details and taking note of "itemid"

The Screenshot Below Shows Successful Added Vault Item On Account Two

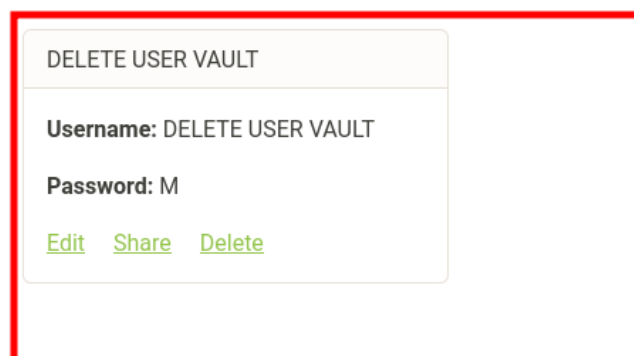


Figure 13: Screenshot showing the above successful added vault item

Login to account one and initiate a delete request on any vault item of your choice, intercept the request and send it to repeater

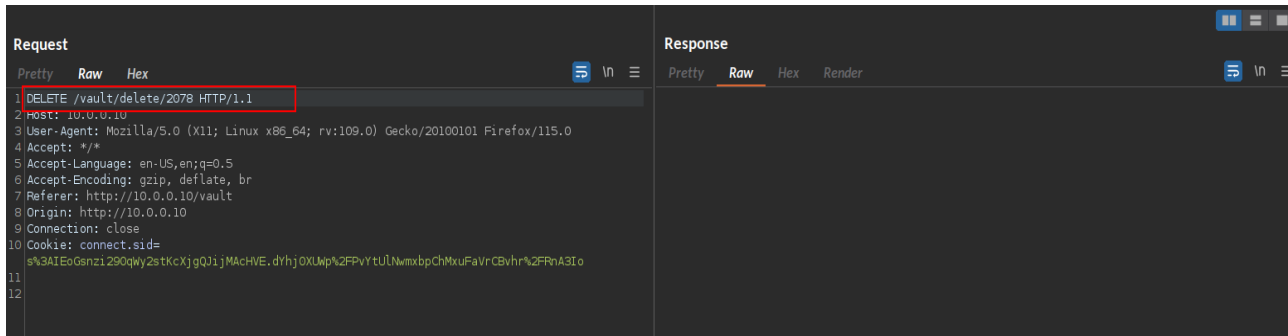


Figure 14: Screenshot showing intercepted delete request initiated

The above delete request was initiated with account one, intercepted and send to Burp suite repeater.

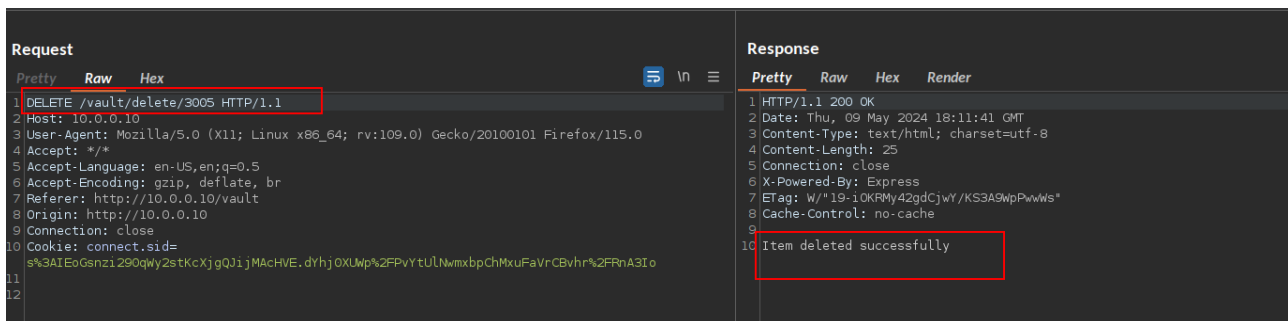


Figure 15: Screenshot showing successful manipulated vault "itemid" between two

Manipulate the **“itemid”** on the URL from account one **“itemid”** to account two **“itemid”**

The ability to delete any vault item using item id using any account was successful

You can do a brute-force attack and delete all vault item in the database with this vulnerability

Remediation

Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.

Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.

Findings 004: Broken access control on <http://XX.XX.XX.XX/share/id>

Description:	This vulnerability allows an attacker to view vault items of other users via brute-force dictionary attack
Risk:	<ul style="list-style-type: none"> • Unauthorized Data Access: Attackers can exploit broken access controls to gain access to sensitive data they are not authorized to view, such as personally identifiable information (PII), financial records, or intellectual property. • Data Tampering: Without proper access controls, attackers may be able to manipulate or modify data stored within the system, leading to data corruption, integrity breaches, or fraudulent activities. •
Affected Functionality	http://XX.XX.XX.XX/share/id
Tool Used	Burp Suite, and Firefox browser
References	https://owasp.org/www-project-application-security-verification-standard https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README

Proof of concept

Initiate a share request on <http://XX.XX.XX.XX/share/id>, on any vault item of your choice send the request to burp intruder

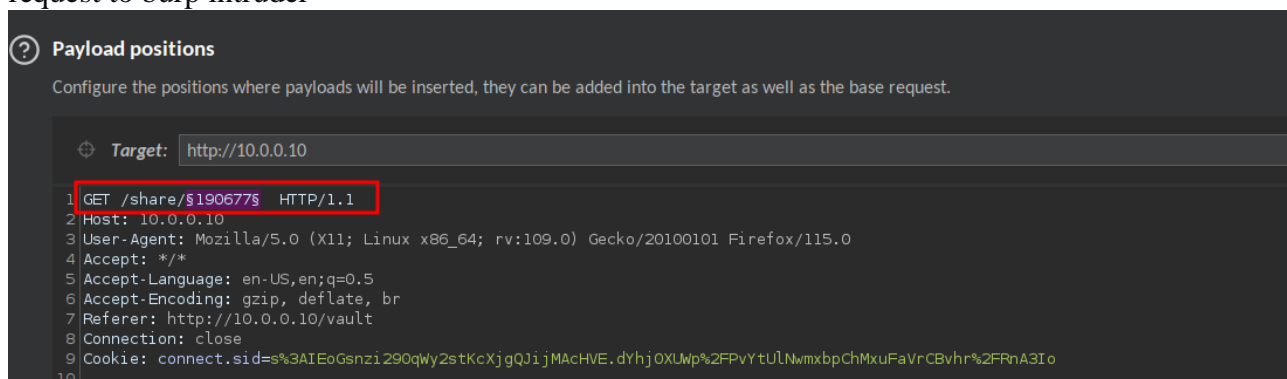


Figure 16: Screenshot showing vault share key selected as the payload position

Highlight The Vault Share Code, As The Payload Position

On the payload set, choose number you can add from 0 - 999999 but in this case i added 888888 - 999999

Payload sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 111,112
Payload type: Numbers Request count: 111,112

Payload settings [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: ☒ Sequential ☐ Random

From: 888888
To: 999999
Step: 1
How many: 1

Number format

Base: ☒ Decimal ☐ Hex

Min integer digits: 0
Max integer digits: 6
Min fraction digits: 0
Max fraction digits: 0

Examples

1
654321

Figure 17: Screenshot showing payload set

With the 200 status code from the screenshot below, i was able to view other vault item secrets via brute force and dictionary attack

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	618			290	
68	888955	200	351			290	
392	889279	200	309			290	
730	889617	200	389			290	
838	889725	200	373			290	
901	889788	200	392			290	
1339	890226	200	393			290	
1664	890551	200	300			290	
1896	890783	200	355			290	

Request: Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Date: Thu, 09 May 2024 18:28:11 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 53
Connection: keep-alive
X-Powered-By: Express
ETag: W/"35-6L2m3eBI4MdyG2u6tuaWhdbA0FM"
Cache-Control: no-cache

{
  "success": true,
  "message": "Item added successfully!"
}
```

Figure 18: Screenshot showing vault share key with 200 status code

6775	895662	200	586
6883	895770	200	472
7322	896209	200	518
7347	896234	200	463
7447	896334	200	326
7882	896769	200	346
8525	897412	200	356
8987	897874	200	539
9347	898234	200	335
9671	898558	200	350
10069	898956	200	307
10085	898972	200	359
10465	899352	200	329
10532	899419	200	990
11356	900243	200	337
11536	900423	200	294
11662	900549	200	380
1	888888	500	604
2	888889	500	499
3	888890	500	604
4	888891	500	506
5	888892	500	490
6	888893	500	604
7	888894	500	507
8	888895	500	607
9	888896	500	602
10	888897	500	359

Figure 19: More POC

Remediation

vault share key should be encrypted to avoid being passed on clear text.

Use encryption to protect sensitive data both at rest and in transit.

Implement real-time alerts for suspicious activities or access patterns.

Implement access control checks whenever accessing or modifying sensitive objects or resources.

Findings 005: Stored Cross Site Scripting Vulnerability

Description:	Stored Cross-Site Scripting (Stored XSS) is a type of Cross-Site Scripting (XSS) vulnerability where malicious scripts are injected into a vulnerable web application and stored persistently on the server
Risk:	<ul style="list-style-type: none">• Data Theft: Attackers can steal sensitive information from other users, such as session cookies, authentication tokens, or personal data.• Account Compromise: By stealing session cookies or tokens, attackers can hijack user sessions and impersonate legitimate users, gaining unauthorized access to their accounts.
Affected Functionality	http://XX.XX.XX.XX/vault/add
Tool Used	Firefox browser
References	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting

Proof of Concept

Use the following payload on the username input field when adding a new vault item

``

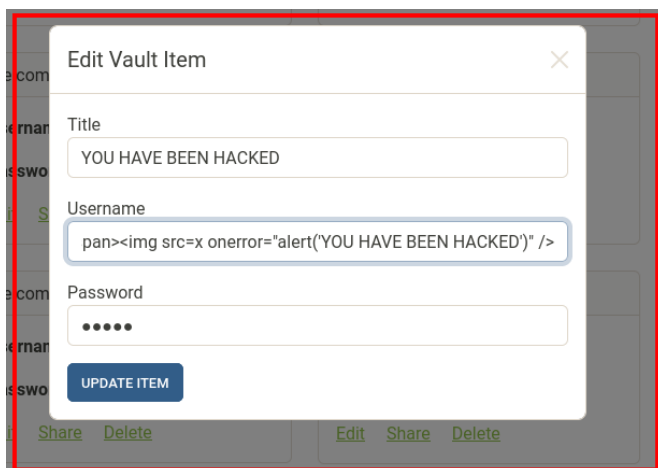


Figure 20: Screenshot showing payload inserted in input field

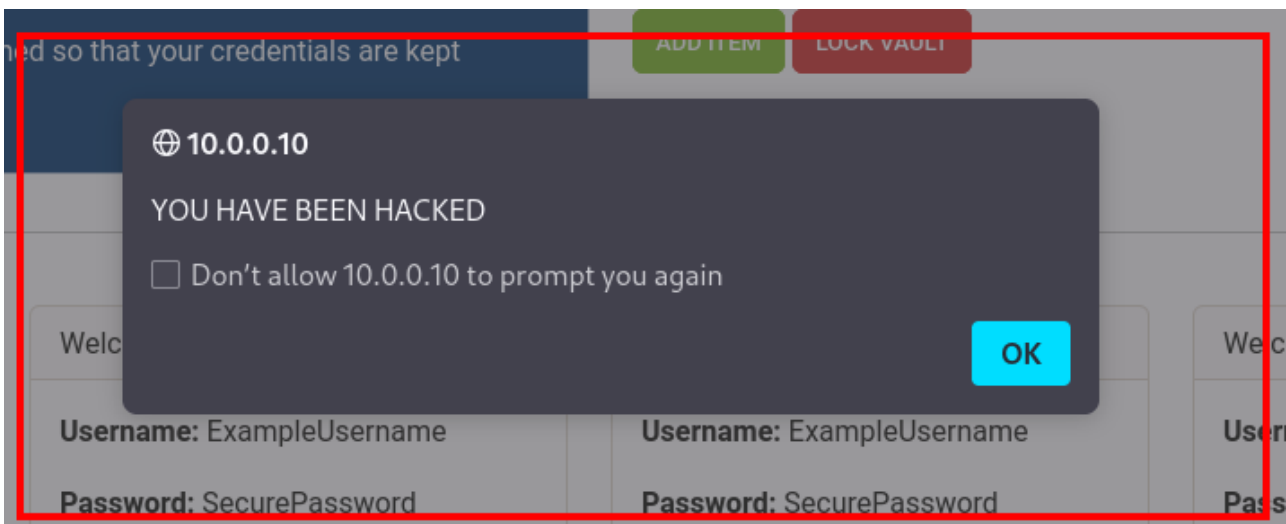


Figure 21: Screenshot showing payload execution

Remediation

- **Input Validation and Sanitization:** Validate and sanitize all user-supplied input before storing it on the server or displaying it to other users.
- **Content Security Policy (CSP):** Implement CSP directives to restrict the sources from which scripts can be executed, mitigating the impact of XSS attacks.
- **Output Encoding:** Encode user-generated content properly before rendering it in HTML context to prevent script execution.
- **Web Application Firewall (WAF):** Employ WAFs to detect and block malicious XSS payloads before they reach the application server.
- **Regular Security Audits:** Conduct routine security assessments and penetration testing to identify and remediate XSS vulnerabilities in the application code.

Findings 006: DOM Based Cross Site Scripting Vulnerability

Description:	A user can share a vault key containing a malicious DOM Based cross site scripting leading to the execution of malicious scripts in a victim's browser.
Risk:	<ul style="list-style-type: none"> • Data Theft: Attackers can steal sensitive information stored within the victim's browser, such as session cookies, authentication tokens, or personal data. • Session Hijacking: By stealing session cookies or tokens, attackers can impersonate legitimate users and perform unauthorized actions on their behalf. • Malicious Redirects: Attackers can redirect users to phishing sites or malware-infected pages, leading to further exploitation or compromise.
Affected Functionality	http://XX.XX.XX.XX/vault/add
Tool Used	Firefox browser
References	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client_Side_Testing/01-Testing_for_DOM-based_Cross_Site_Scripting

Proof of Concept

Edit or add a new vault item, on the title input.. insert the following code

``

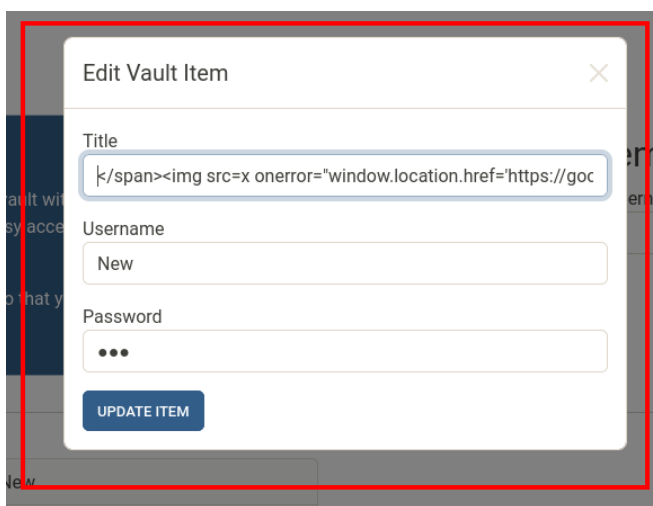


Figure 22: Screenshot showing DOM XSS payload in input field

Immediately you click add / update item, the page will refresh and redirect to the specified endpoint on the payload

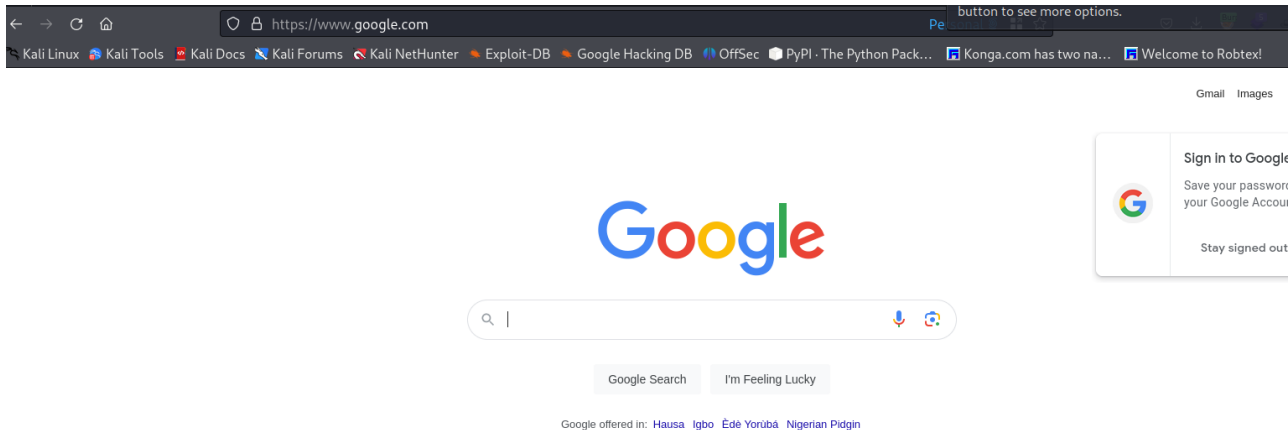


Figure 23: Screenshot showing DOM XSS exploitation

Remediation

- **Input Sanitization:** Validate and sanitize all user-supplied input before manipulating the DOM or executing client-side scripts.
- **Context-Aware Escaping:** Escape user input based on its context within the DOM (e.g., HTML, JavaScript, URL) to prevent script execution.
- **Secure Coding Practices:** Follow secure coding guidelines and best practices when developing client-side JavaScript code to minimize the risk of introducing DOM XSS vulnerabilities.

Findings 007: JWT Token Bypass

Description:	A JSON Web Token (JWT) bypass vulnerability occurs when an attacker is able to manipulate or tamper with the components of a JWT token to gain unauthorized access, which I was able to use Autorize: a burp suite extension to check for JWT manipulation.
Risk:	<ul style="list-style-type: none"> • Unauthorized Access: Attackers may gain access to restricted resources or perform actions on behalf of other users by forging or modifying JWT tokens. • Privilege Escalation: Manipulating JWT tokens may allow attackers to escalate their privileges within the application, gaining access to administrative features or sensitive data. • Data Tampering: By modifying the payload of a JWT token, attackers can tamper with user data, session information, or other sensitive attributes.
Affected Functionality	http://XX.XX.XX.XX/api/vault:id
Tool Used	Firefox browser, Burp Suite Autorize Extension
References	https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html https://tools.ietf.org/html/rfc7519

Proof of Concept

Create two account (bob and james), generate your token as specified in the API documentation

```
(venuskali@kali)~$ curl -x proxy 127.0.0.1:8080 -X POST -d '{"username":"james","password":"james"}' -H 'Content-Type: application/json' http://10.0.0.10/api/token
{"token":"f7fc65ed-8a4c-43db-973a-c426d32a00cc"}

(venuskali@kali)~$ curl -x proxy 127.0.0.1:8080 -X POST -d '{"username":"bob","password":"bob"}' -H 'Content-Type: application/json' http://10.0.0.10/api/token
{"token":"55206812-f3af-49df-b81f-b921a23049a8"}
```

Figure 24: Screenshot showing how user token is been generated

add bob token to authorize as showing below

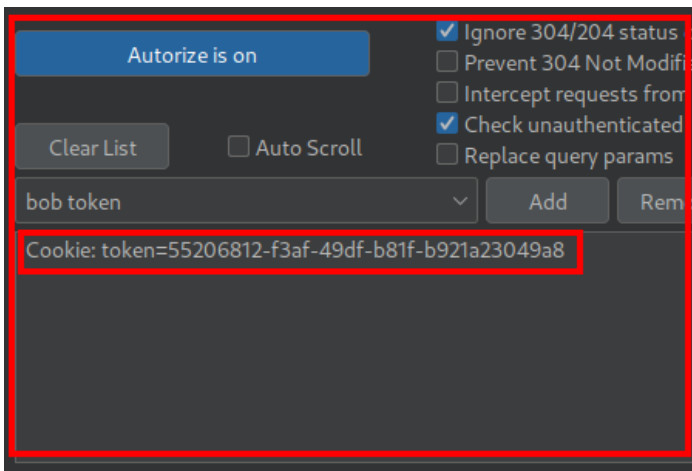


Figure 25: bob account token inputted on Authorize burp extension

Proxy a post request to burp suite with james account token while the token of bob has been inserted on burp suite Authorize extension

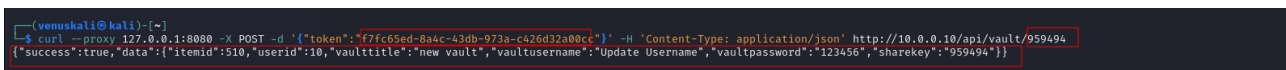


Figure 26: Screenshot showing a POST request on James account

From the below screenshot, It shows bypassed which means there is a token bypassed vulnerability which needs further investigation

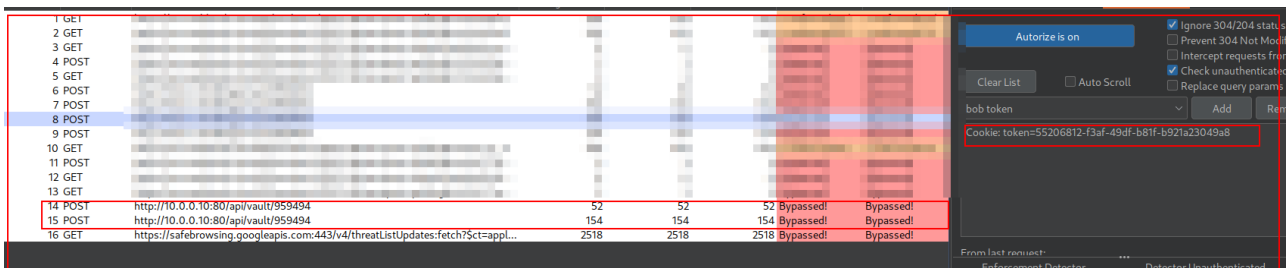


Figure 27: Screenshot showing token bypassed vulnerability

Remediation

- **Strong Key Management:** Use strong cryptographic keys and algorithms for signing JWT tokens, and securely manage and rotate these keys.

- **Algorithm Restrictions:** Limit the acceptable algorithms for JWT token signatures to only those considered secure and trusted.
- **Strict Validation:** Implement strict validation of JWT tokens, including checking the token's integrity, expiration, issuer, and audience claims.
- **Secure Token Storage:** Store JWT tokens securely on the client side, using mechanisms such as HTTP-only cookies or local storage with appropriate safeguards.

Findings 008: Password Authentication Check Bypass: Ability to update empty password field during account update

Description:	This vulnerability occurs when an application allows users to set or update their password to an empty value, effectively bypassing password-based authentication altogether.
Risk:	<ul style="list-style-type: none"> • Unauthorized Access: Attackers can gain unauthorized access to user accounts by setting the password to an empty value, bypassing authentication checks. • Data Breach: Empty passwords weaken the security posture of the application, potentially leading to data breaches and unauthorized access to sensitive information.
Affected Functionality	http://XX.XX.XX.XX/account/update
Tool Used	Firefox browser, Burp Suite
References	https://cwe.mitre.org/data/definitions/319.html

Proof of Concept

- On account password update, insert a password and intercept the request, using burp suite
- Remove the password to leave the input empty
- Forward the request and you will have a 200 Ok response code
- Sign out
- Sign in with the same username and an empty space password

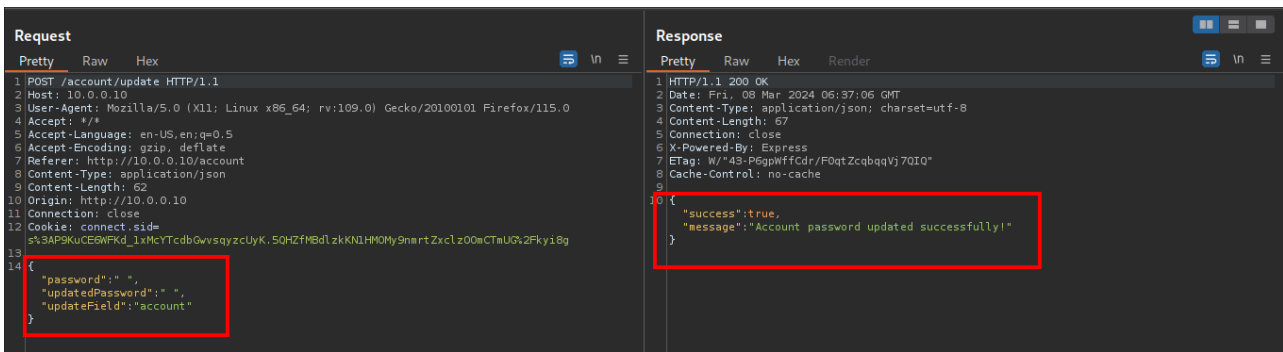


Figure 28: Screenshot showing ability to update empty field in account update

Screenshot showing login successful with empty password field

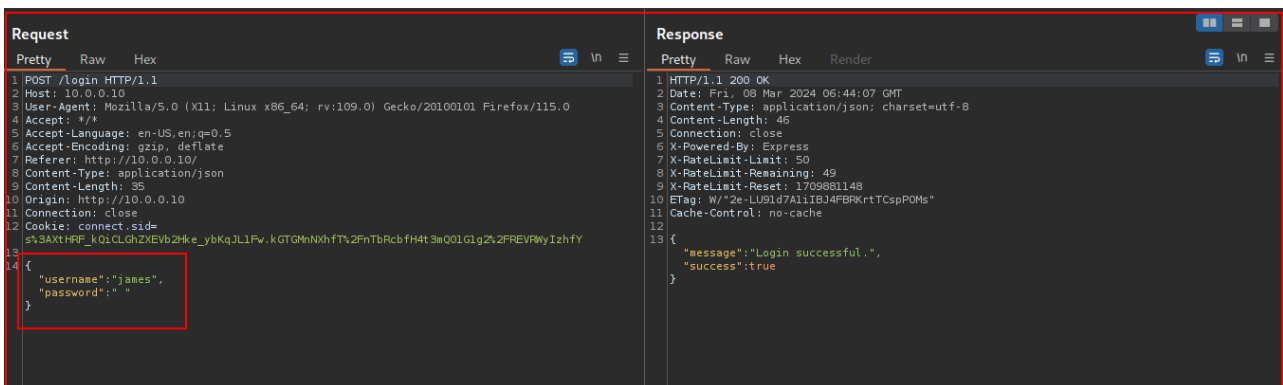


Figure 29: Screenshot showing ability to update and login with empty password field

Remediation

- **Enforce Password Complexity:** Require users to create strong passwords with minimum length requirements, character diversity, and complexity rules.
- **Disallow Empty Passwords:** Implement validation checks to prevent users from setting their password to an empty value during account creation, password reset, or password change processes.
- **Secure Defaults:** Ensure that accounts are created with secure default settings, including non-empty passwords, to prevent initial account compromise.
- **Password Expiration:** Implement password expiration policies to prompt users to update their passwords periodically, reducing the likelihood of empty passwords persisting over time.

Findings 009: Username Confusion Vulnerability, allowing an attacker to add a space in front of username during account creation

Description:	The ability to add a space before the username during account creation is a vulnerability that can lead to various security issues, including user confusion, account enumeration, and potential bypass of security measures
Risk:	<ul style="list-style-type: none"> • User Confusion: Users may encounter login issues or errors if they unintentionally include or omit the space character when entering their usernames. • Security Risks: Account enumeration can facilitate targeted attacks, such as credential stuffing or brute force attacks, by identifying valid usernames for exploitation. • Data Integrity: Inconsistent handling of usernames may lead to data integrity issues; such as duplicate or conflicting user records within the system.
Affected Functionality	http://XX.XX.XX.XX/register
Tool Used	Firefox browser
References	https://owasp.org/www-project-web-security-testing-guide/v42 https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf

Proof of Concept

From the screenshot it can be observed that there is a space before the username on the request and the response was “Registration Successful”

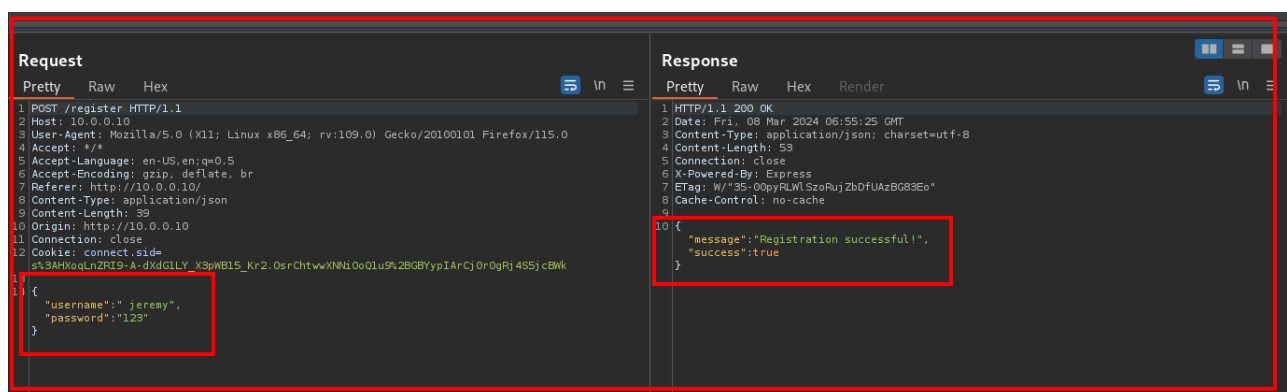


Figure 30: Screenshot showing a space before a username in account Reg.

Remediation

- **Input Sanitization:** Implement strict input validation and sanitization to prevent users from adding leading or trailing spaces to usernames during account creation.
- **Normalization:** Normalize usernames to remove extraneous whitespace characters or other non-standard characters to ensure consistency and prevent confusion.
- **Unique Username Enforcement:** Enforce uniqueness constraints for usernames to prevent users from registering accounts with similar or conflicting usernames.
- **Security Testing:** Conduct thorough security testing, including vulnerability assessments and penetration testing, to identify and remediate vulnerabilities related to username manipulation and account enumeration.

Findings 010: Lack of Rate Limiting

Description:	It was observed that the login page has no rate limiting security implemented. The lack of rate limiting vulnerability occurs when an application fails to enforce restrictions on the number of requests a user can make within a specified time period. Without rate limiting, attackers can exploit the application by sending a large volume of requests in a short amount of time, leading to various security issues
Risk:	<ul style="list-style-type: none"> • Service Disruption: DoS attacks can disrupt the availability of the application, causing inconvenience to users and potential financial losses for businesses. • Unauthorized Access: Brute force attacks and account takeover attempts can lead to unauthorized access to sensitive information, compromise user accounts, and result in data breaches. • Resource Consumption: Excessive request volume can consume excessive server resources, leading to increased operational costs, decreased performance and disrupt webserver performance
Affected Functionality	http://XX.XX.XX.XX/login http://XX.XX.XX.XX/vault/unlock
Tool Used	Firefox browser, Burp Suite and FFUF
References	https://nvd.nist.gov/vuln/detail/CVE-2023-20155

Proof of Concept

Screenshot showing brute force attack with ffuf tool on login page with different wordlist.

```

:: Method      : POST
:: URL         : http://10.0.0.10/login
:: Wordlist     : FUZZ: /usr/share/wordlists/seclists/SecLists-master/Passwords/xato-net-10-million-passwords-100000.txt
:: Header      : User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
:: Header      : Accept-Language: en-US,en;q=0.5
:: Header      : Content-Type: application/json
:: Header      : Origin: http://10.0.0.10
:: Header      : Connection: close
:: Header      : Cookie: connect.sid=s%3AORnEBxCAXaP7RMu_ZWd76_xeINWnw3Wj.MVgnLjGHUcotdqJKRSvx%2BM695ozG1%2BZ3juQkOCZQoJ0
:: Header      : Host: 10.0.0.10
:: Header      : Accept: */*
:: Header      : Accept-Encoding: gzip, deflate
:: Header      : Referer: http://10.0.0.10/
:: Data        : {"username":"jeremy","password":"FUZZ"}
:: Follow redirects: false
:: Calibration : false
:: Timeout      : 10
:: Threads      : 40
:: Matcher      : Response status: 200-299,301,302,307,401,403,405,500

```

Figure 31: Screenshot showing abruteforce attack on login page over 10 million username and password

Remediation

- **Implement Rate Limiting:** Enforce rate limits on user actions, such as login attempts, API requests, or resource accesses, to restrict the number of requests a user can make within a specified time period.
- **Distributed Denial of Service (DDoS) Protection:** Deploy DDoS protection mechanisms, such as web application firewalls (WAFs) or cloud-based DDoS mitigation services, to mitigate large-scale attacks.
- **Strong Authentication Mechanisms:** Implement multi-factor authentication (MFA) and CAPTCHA challenges to strengthen authentication and deter brute force attacks.
- **Monitoring and Logging:** Monitor application traffic, track request rates, and log suspicious activities to detect and respond to potential rate limiting bypass attempts or abuse.

Findings 011: Clear text submission of password on XX.XX.XX.XX/login

Description:	The clear text submission of passwords vulnerability occurs when sensitive information, such as passwords, is transmitted over the network without encryption, making it susceptible to interception by malicious actors.
Risk:	<ul style="list-style-type: none"> • Password Theft: Attackers can capture users' passwords and potentially gain unauthorized access to their accounts. • Credential Stuffing: Stolen passwords can be used in automated attacks to compromise other accounts where users have reused passwords. • Data Breaches: Sensitive data transmitted in clear text can be intercepted and used for malicious purposes, leading to data breaches and loss of trust.
Affected Functionality	http://XX.XX.XX.XX/login http://XX.XX.XX.XX/vault/unlock
Tool Used	Firefox browser, Burp Suite
References	https://cwe.mitre.org/data/definitions/319.html

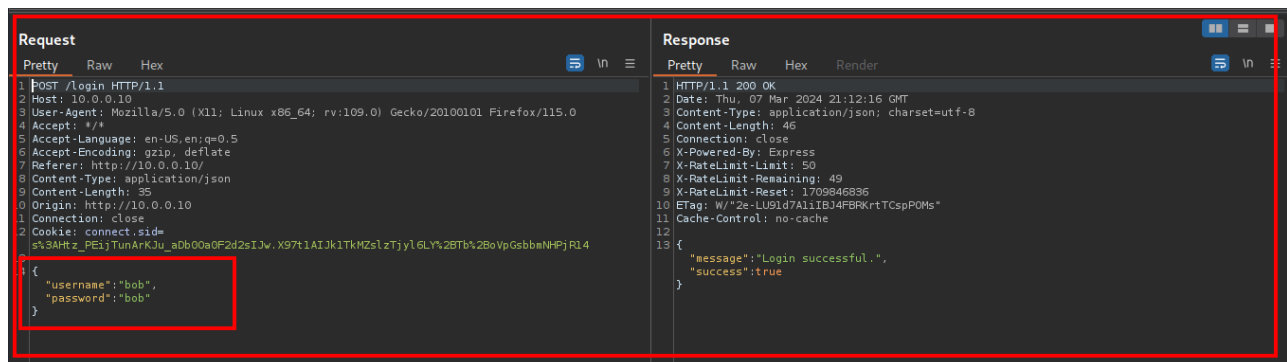
Proof of Concept

Figure 32: Screenshot showing clear text submission of password in a POST Request

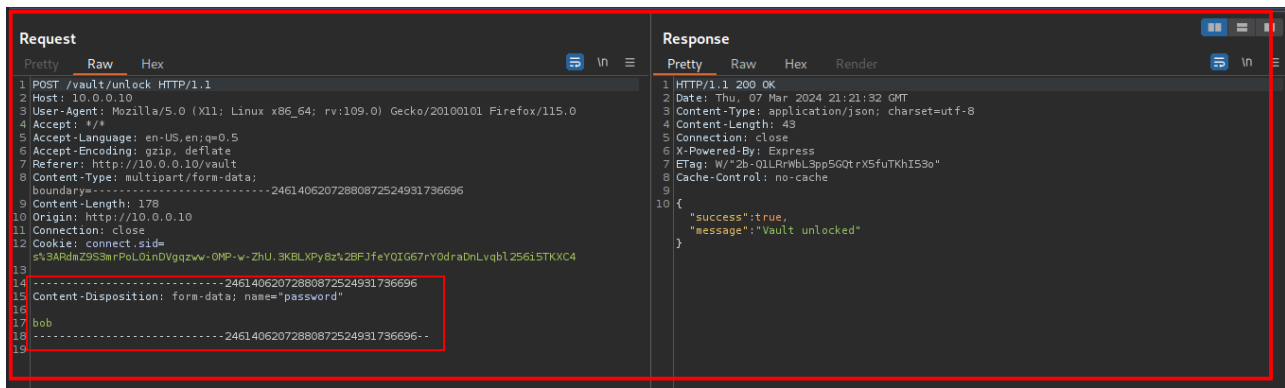


Figure 33: Figure 32: Screenshot showing clear text submission of password in a POST Request

Remediation

- **Transport Layer Security (TLS):** Encrypt network traffic using HTTPS (HTTP over SSL/TLS) to ensure data confidentiality and integrity during transmission.
- **Secure Password Hashing:** Store passwords securely on the server-side using strong cryptographic hash functions (e.g., bcrypt, Argon2) to prevent password disclosure even if intercepted.
- **Password Managers:** Encourage users to use password managers to generate and securely store complex passwords, reducing the risk of password interception.
- **Security Education:** Educate users about the importance of using secure connections (HTTPS) and avoiding transmitting sensitive information over unsecured networks.

Findings 012: Weak Password Policy

Description:	Weak password policies increase the risk of users creating weak passwords that could allow attackers to steal easily through generic techniques such as brute force attacks, authentication challenge theft, etc.
Risk:	<ul style="list-style-type: none"> • Weak password policies increase the risk of unauthorized access, data breaches, and account compromises. They undermine the confidentiality, integrity, and availability of systems and sensitive information.
Affected Functionality	http://XX.XX.XX.XX/login http://XX.XX.XX.XX/vault/unlock
Tool Used	Firefox browser, Burp Suite

References	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/07-Testing_for_Weak_Password_Policy
------------	---

Proof of Concept

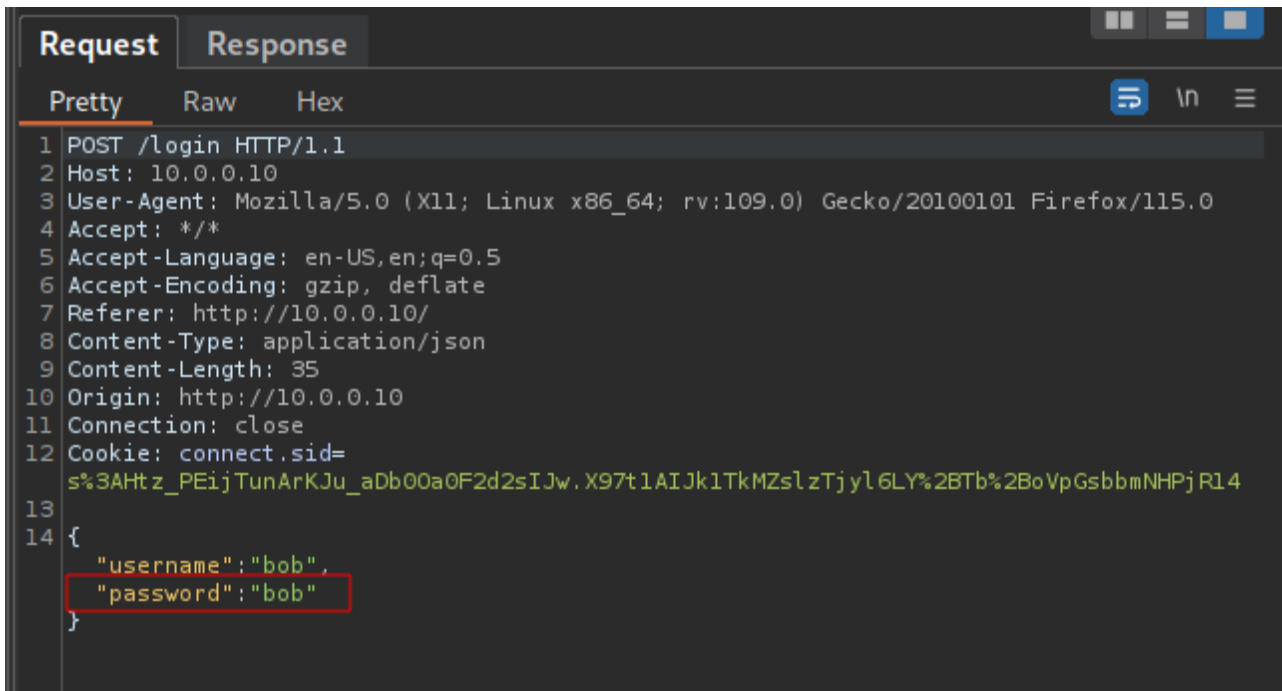


Figure 34: Screenshot showing a weak password being used during account registration

From the screenshot above the password can be easily brute forced and the password hash easily cracked by attacker

Remediation

- Requiring passwords to meet complexity requirements.
- Enforcing minimum password length.
- Mandating regular password changes.
- Implementing account lockout mechanisms.
- Conducting regular security awareness training for users.
- Implementing multi-factor authentication where possible.

Compliance and Standards: Various compliance regulations and standards, such as PCI DSS, HIPAA, and NIST guidelines, provide recommendations and requirements for implementing strong password policies to protect sensitive data and systems.

Findings 013: Username Enumeration Vulnerability

Description:	Username enumeration vulnerability is a security flaw that allows attackers to determine valid usernames on a system or application through various methods, such as error messages, timing differences, or different responses to valid and invalid inputs
Risk:	<ul style="list-style-type: none"> • User Privacy: Attackers can collect valid usernames, which may be used for targeted attacks, spam, or phishing campaigns. • Account Takeover: Knowing valid usernames can aid attackers in launching credential stuffing or brute force attacks to gain unauthorized access to user accounts.
Affected Functionality	http://XX.XX.XX.XX/register
Tool Used	Firefox browser
References	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account

Proof of Concept

From the screenshot... I was to create an account with the username Jeremy and the response was very specific that the username already exists.

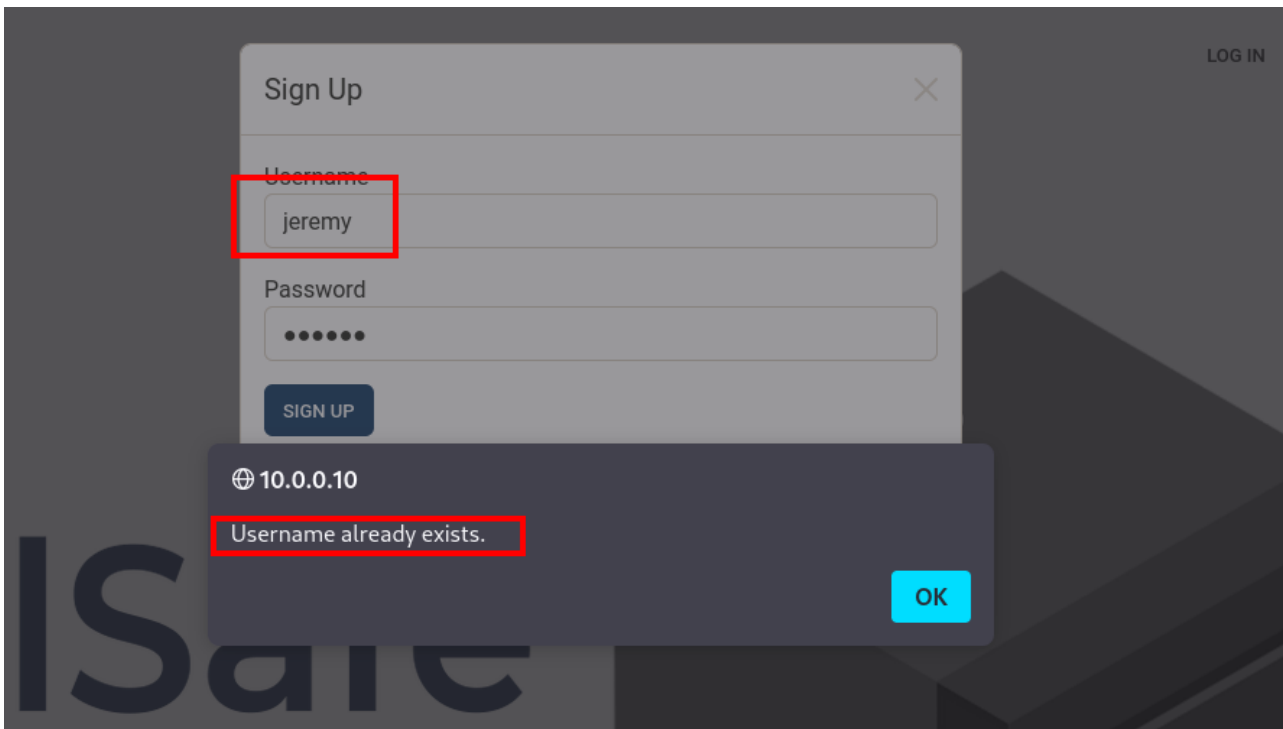


Figure 35: Screenshot showing a response of username already exists

Remediation

- **Consistent Error Messages:** Ensure that the application displays generic error messages for both valid and invalid usernames to prevent attackers from distinguishing between them.
- **Rate Limiting:** Implement rate limiting on authentication and account recovery endpoints to mitigate the risk of brute force attacks.
- **CAPTCHA Challenges:** Introduce CAPTCHA challenges or other anti-automation mechanisms to deter automated username enumeration attempts.
- **Security Awareness Training:** Educate users and administrators about the risks of username enumeration and the importance of strong authentication practices.

LAST PAGE