1. Cross-site Request Forgery

The CSRF vulnerability was identified in the profile and self-service page of the application. When a user changes their password, the POST method used for this action was observed in the tools section of the browser. Exploiting this vulnerability allows an attacker to create a webpage with a hidden form connected to the POST method. When an unsuspecting user clicks the link, their password is automatically changed to "water" without their knowledge.

This CSRF vulnerability is critical as it enables unauthorized changes to user passwords, potentially leading to account compromise and unauthorized access.

The following steps and observations were made to discover the CSRF vulnerability:

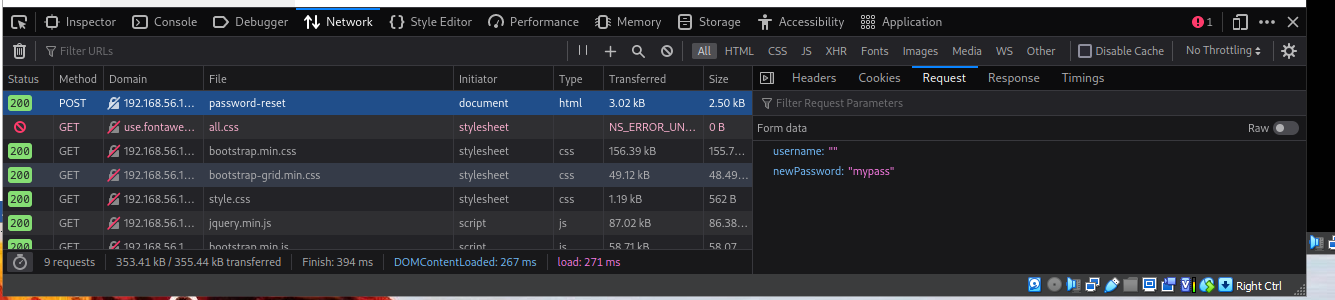
1. Navigate to “Profile and Self-Service” Page: Access the profile or self-service page through the application interface.
2. Password Change Action: Observe the use of the POST method when changing the password.

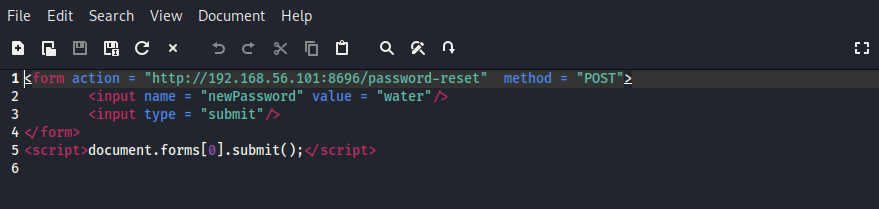
The steps and details of exploiting the CSRF vulnerability are as follows:

1. Create Malicious Webpage: Develop a webpage with a hidden form connected to the observed POST method for password change.
2. Link Distribution: Persuade the victim to click on a link leading to the malicious webpage.

Potential Damage: Exploiting this vulnerability allows an attacker to change a user's password to "water" without their knowledge, compromising the security of their account.

To mitigate the CSRF vulnerability and prevent unauthorized password changes, the following countermeasures are recommended:

1. Anti-CSRF Tokens: Implement anti-CSRF tokens within forms to validate that the request originates from a legitimate source.
2. Referer Header Check: Verify the Referer header to ensure that the request is coming from an authorized domain.

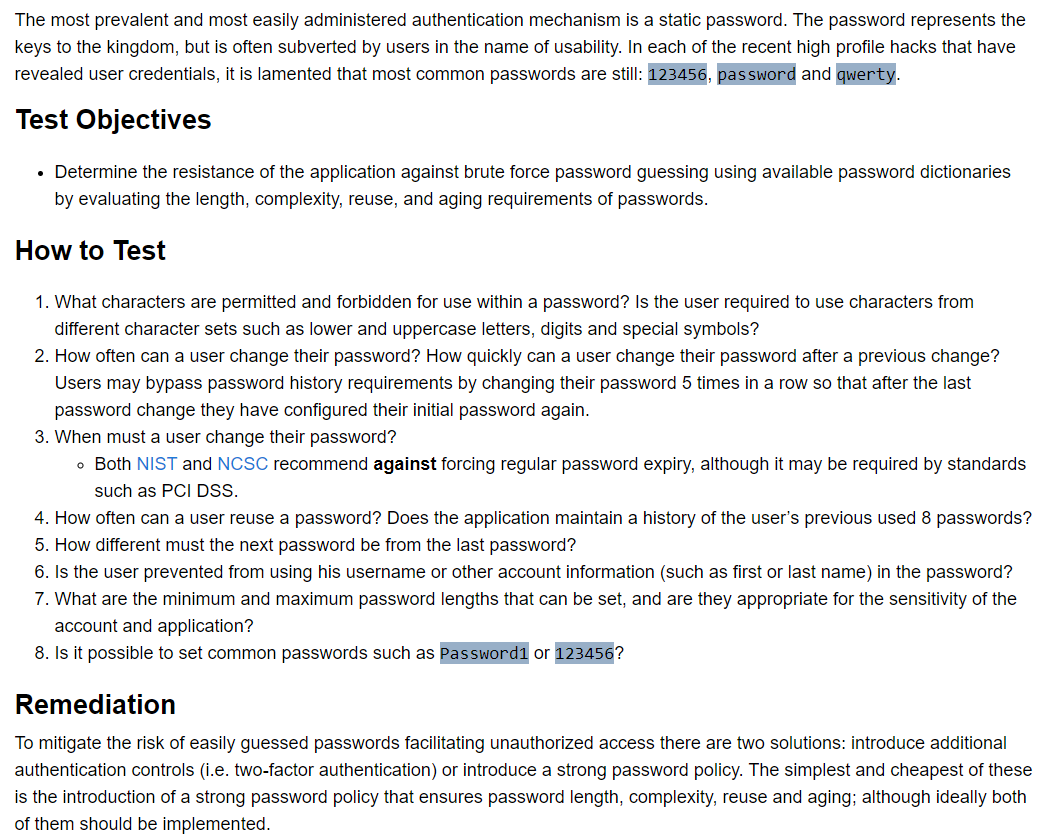


**EDIT THE CODE SCREENSHOT {</SCRIPT>}**

1. Bad Cookie practice

Something to do with HttpOnly. The flag might not be set for all cookies. Probably a vulnerability.

1. Bad Password Policy



1. Cros-Site Scripting

The XSS vulnerability was identified in the “Properties” webpage of the application. By appending “?q=<script>alert(document.cookie)</script>” to the URL and clicking "Enter," an alert popup appeared, displaying the jsessionid in the victim's browser. This XSS vulnerability is significant as it allows an attacker to execute arbitrary scripts in the context of other users' browsers, potentially leading to session hijacking or unauthorized access.

The following steps and observations were made to discover the XSS vulnerability:

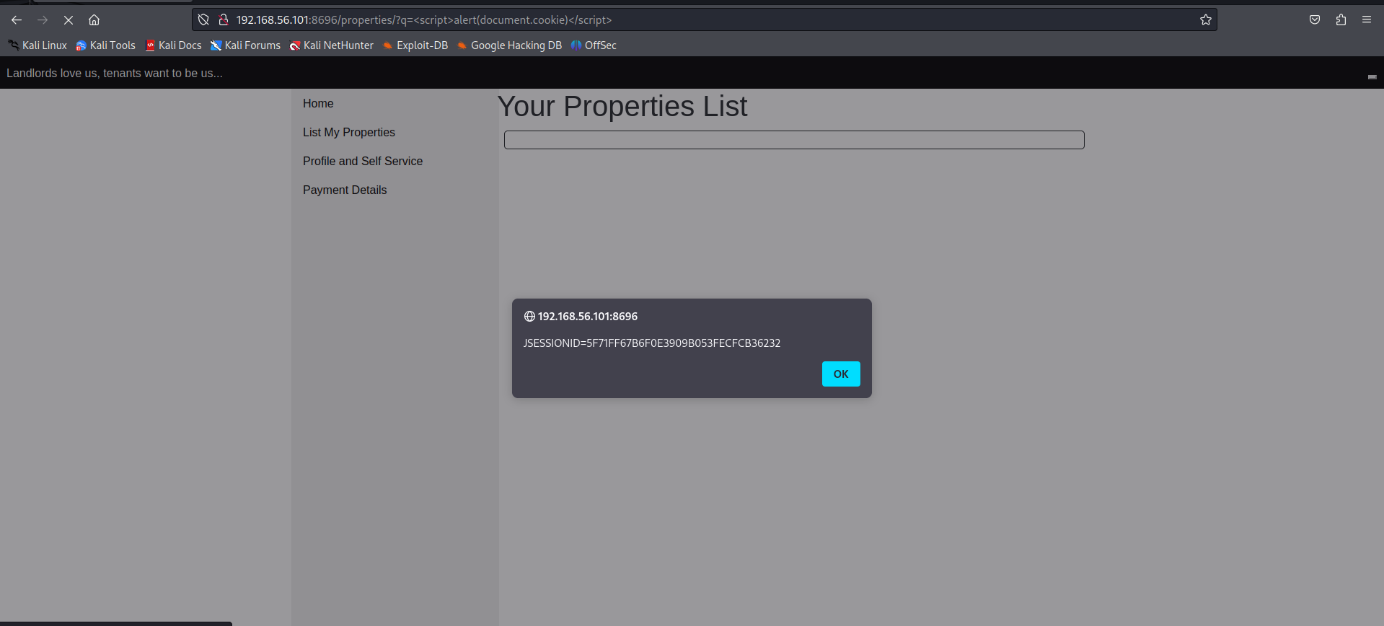
1. Access Properties Webpage: Navigate to the properties webpage through the application interface.
2. URL Payload: Append “?q=<script>alert(document.cookie)</script>” to the URL to inject a script.

The steps and details of exploiting the XSS vulnerability are as follows:

1. Script Injection: Inject the script into the URL to trigger an alert displaying the jsessionid.
2. User Interaction: Convincing the victim to access the manipulated URL.

Potential Damage: Exploiting this vulnerability allows an attacker to execute scripts in the user's browser, potentially leading to the disclosure of sensitive information such as session identifiers.

To mitigate the XSS vulnerability and prevent script injection, the following countermeasures are recommended:

1. Input Validation: Implement strict input validation to filter and sanitize user inputs, especially in URLs.
2. Output Encoding: Encode user-generated content before rendering it in the browser to prevent script execution.
3. Content Security Policy (CSP): Utilize CSP headers to control which scripts are allowed to run on the webpage.



1. Weak Passwords and Insecure Hashing **IDOR**

An additional vulnerability was discovered related to weak password management practices. Another user account with the username "bruce" was identified. To exploit this, the username "bruce" was MD5 hashed, and the password for the account was changed to "water." This vulnerability is critical as it exposes a weakness in password storage and management, potentially leading to unauthorized access to user accounts.

The following steps were taken to discover and exploit the weak password management vulnerability:

1. Identification of Duplicate Username: Identify the existence of another user account with the username "bruce."
2. MD5 Hashing: MD5 hash the username "bruce" to manipulate the account.

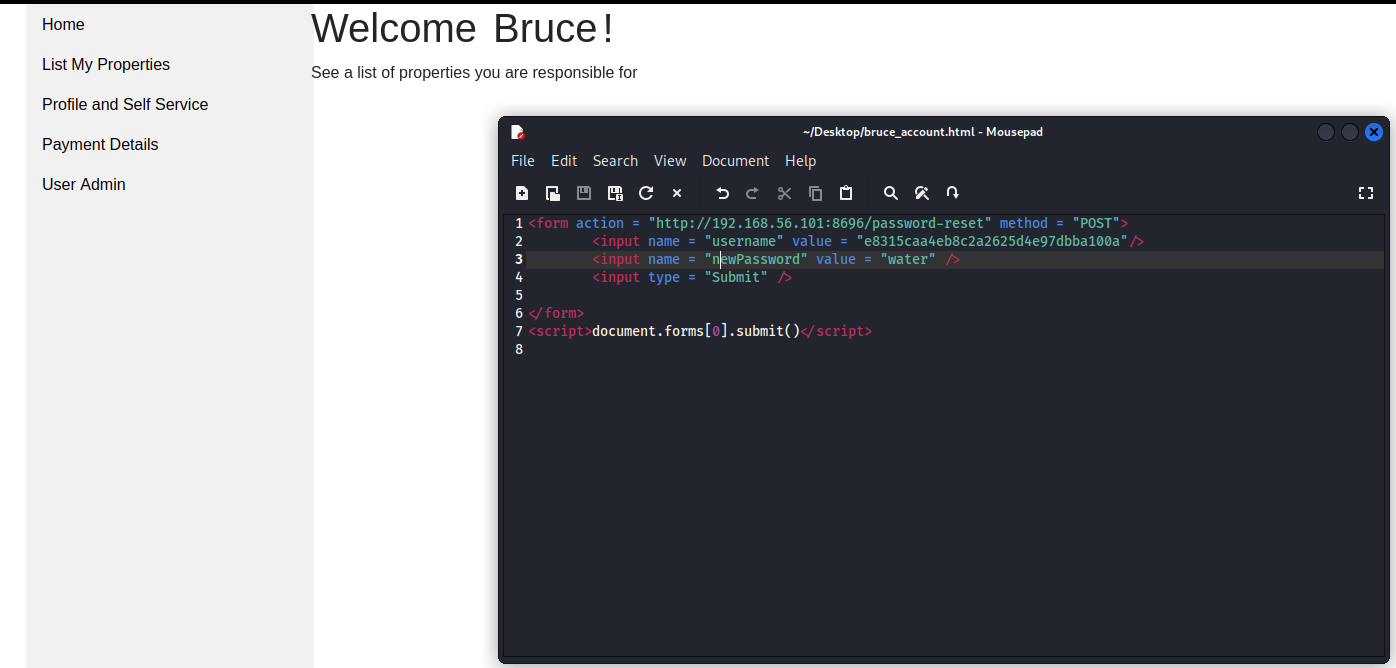
The steps and details of exploiting the weak password management vulnerability are as follows:

1. MD5 Hashing of Username: Convert the username "bruce" to its MD5 hash.
2. Change Password: Update the password for the identified account to "water."

Exploiting this vulnerability allows an attacker to change the password of another user account, potentially leading to unauthorized access.

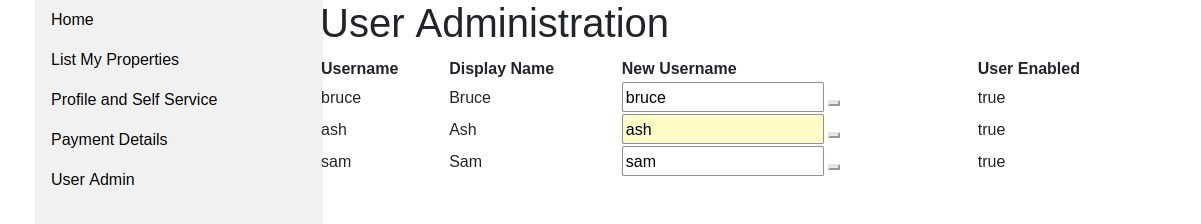
To mitigate the weak password management vulnerability, the following countermeasures are recommended:

1. Strong Password Policies: Enforce strong password policies, requiring users to choose complex passwords.
2. Salting and Secure Hashing: Implement secure password hashing techniques, including salting, to enhance the security of stored passwords.



1. SQL Injection

Navigated to “User Admin” page in Bruce account.



This vulnerability poses a significant threat as it allows unauthorized users to manipulate the usernames associated with accounts, potentially leading to unauthorized access or confusion within the system.

The following steps and commands were followed to discover the SQL injection vulnerability:

1. Log in to Bruce account
2. Navigate to ‘User Admin’ Page

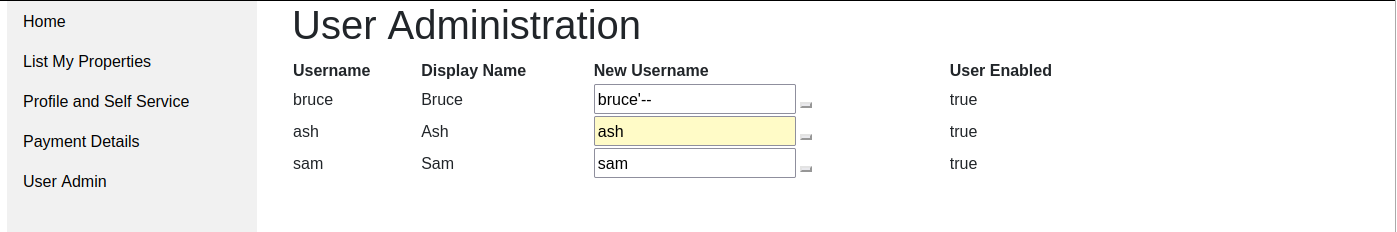
The steps and commands to exploit the SQL injection vulnerability are as follows:

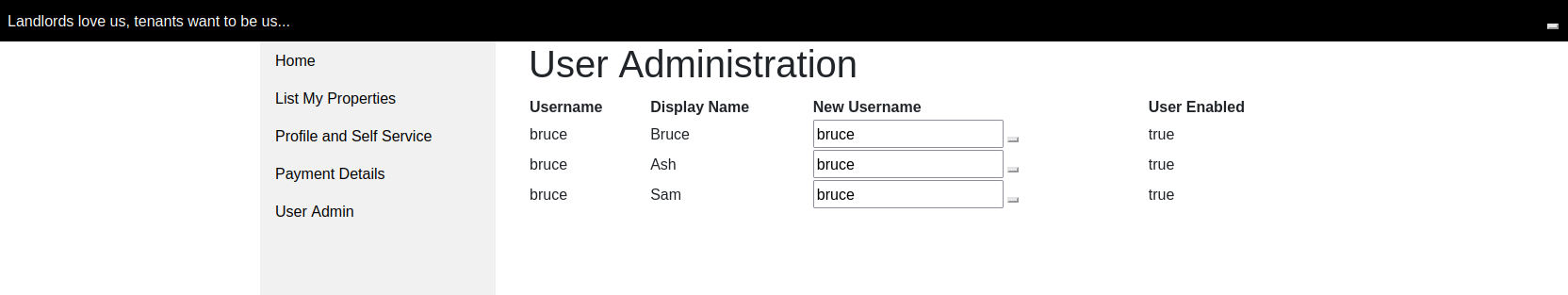
1. Click on one of the “New Username” fields
2. Type in ‘bruce--’ and click “Enter”
3. The page refreshes and changes all the account usernames to “bruce”.

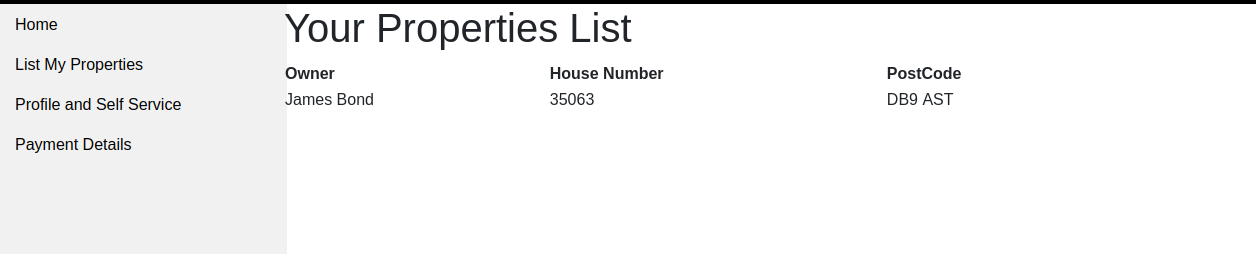
Potential Damage: Exploiting this vulnerability could lead to unauthorized changes in usernames, causing confusion, identity theft, or unauthorized access to sensitive information.

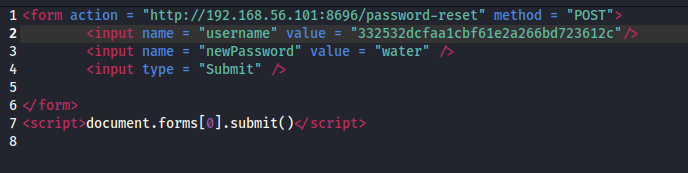
To mitigate the SQL injection vulnerability and prevent unauthorized username changes, the following countermeasures are recommended:

1. Input Validation: Implement strict input validation to filter and sanitize user inputs.
2. Parameterized Queries: Use parameterized queries to prevent SQL injection attacks by separating SQL code from user input.





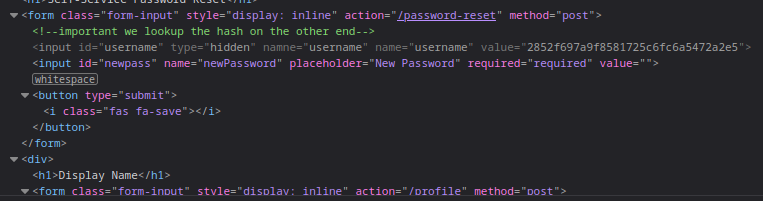
**WHERE DOES JAMES BOND LIVE  
**



I successfully found James Bond’s address by accessing Sam’s account. Firstly, I navigated to the “User Admin” webpage in the Bruce account. I saw that there is a third account with username “sam”. I logged in the “sam” account the same way I did in the “bruce” one. I MD5 hashed the username so it could match it in the database, then changed the account’s password to “water”. When I logged in the account, I navigated to the “List My Properties” page and saw that James Bond’s house number is “35063” and his post code is “DB9 AST”.

Vulnerability- switching pages 1, 2, 3 on payments.

Vulnerability- different message when (right user wrong pass) and (wrong user wrong pass).

After inspecting password reset code. Check comment.

**ftp**

