1. Account Enumeration and Guessable User Account

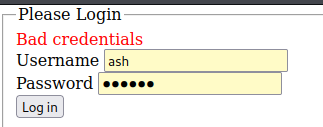
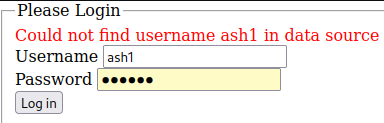
* Description and Severity Assessment:

During the penetration testing, I identified a vulnerability related to Account Enumeration and Guessable User Account. The system exhibited distinct error messages for valid and invalid user accounts, inadvertently disclosing information about the validity of usernames.

This vulnerability poses a significant risk by providing attackers with a clear path to identify valid user accounts. Successful exploitation could lead to unauthorized access, potentially compromising sensitive information and system integrity.

* Steps to Discover:

During the initial phase of exploration, I pinpointed the login page as a possible vulnerability. To investigate, I tested error messages resulting from login attempts. When using correct usernames with incorrect passwords, the system returned a "Bad credentials" error. In contrast, entering invalid usernames with incorrect passwords triggered a "Could not find username ... in data source" error, indicating potential user enumeration. This structured approach unveiled a vulnerability in the system's response patterns during login attempts, potentially exposing valid usernames.



* Damage it Could Cause:

The potential damage includes the risk of malicious actors using enumerated valid usernames for targeted attacks, unauthorized access, or social engineering. Information about non-existent usernames could be exploited for reconnaissance in crafting more sophisticated attacks. Addressing this vulnerability promptly is crucial to mitigate these risks.

* Countermeasures to Fix the Vulnerability:
* More Generic Error Messages: The system can be modified to show the same generic error message for both correct and incorrect usernames. This is going to prevent the attackers of guessing the username using the error message.
* Account lockout: The system can be modified in a way that locks the user out of it for a period of time after a certain amount of login attempts. This is going to deter the brute-force attacks and increase the system security.

1. Cross-site Scripting

* Description and Severity Assessment:

While exploring the web application under the Ash account (username: "ash" and password: "mypass"), a significant security vulnerability was identified—Cross-site Scripting (XSS). The vulnerability resides in the "List My Properties" page, exposing the application to potential exploitation.

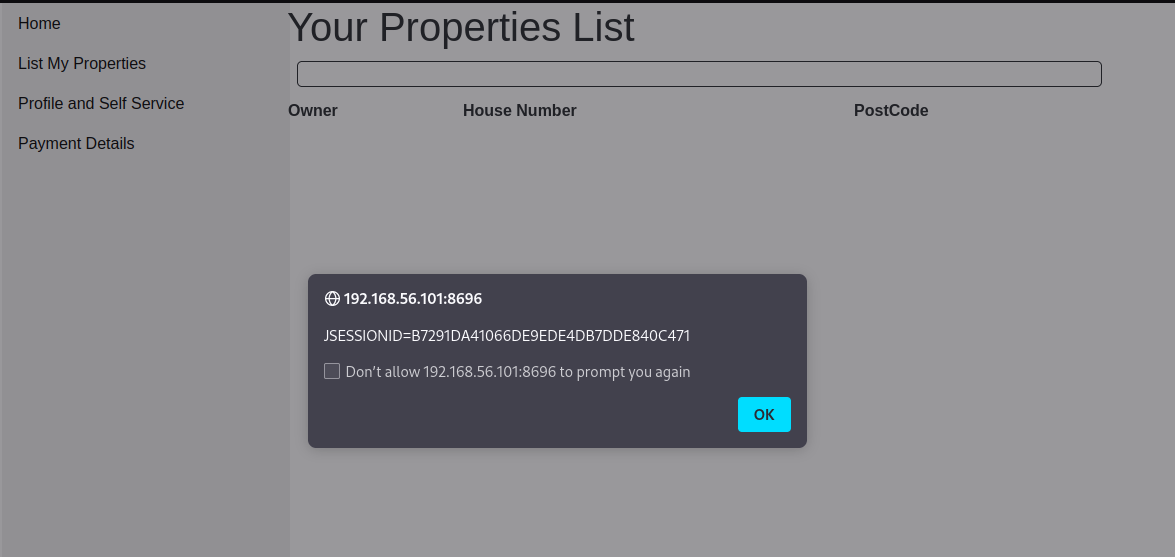
Rated as high severity, this XSS vulnerability poses a serious threat by enabling the execution of arbitrary scripts within the user's session, creating avenues for potential session hijacking, data theft, and the compromise of sensitive information.

* Steps to Discover:

While using the Ash account, I explored the system, logging in and navigating to the "List My Properties" page. In the process of looking for security issues I found that the URL might be vulnerable to Cross-site Scripting (XSS). This discovery unfolded as I interacted with the application, identifying potential risks during my exploration.

* Exploiting the Vulnerability:

Seeking to exploit the identified Cross-site Scripting (XSS) vulnerability, I injected a script into the URL parameter using "?q=<script>alert(document.cookie)</script>". When I pressed "Enter," the page reloaded, triggering an alert box that displayed the JSESSIONID of the active session. This step-by-step process demonstrated the successful execution of an XSS attack, exploiting the vulnerability for potential session hijacking and data theft.



* Damage it Could Cause:

The XSS vulnerability presents a significant threat, potentially allowing malicious actors to hijack user sessions and gain unauthorized access to sensitive accounts and data. Moreover, the exploitation of confidential information stored in user sessions could lead to privacy breaches and the compromise of sensitive data.

* Countermeasures to Fix the Vulnerability:
* Content Security Policy (CSP): Enforce a robust CSP to restrict the sources from which scripts can be executed.
* Output Encoding: Apply proper output encoding to user inputs displayed on webpages.
* Input Validation: Implement strict input validation on user inputs to prevent the execution of scripts.

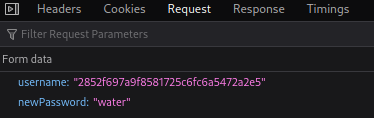
1. Cross-site Request Forgery

* Description and Severity Assessment:

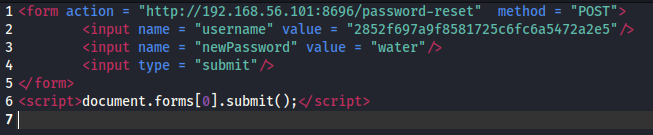
While navigating to the "Profile and Self Service" page under the Ash account (username: "ash" and password: "mypass"), a substantial security vulnerability was uncovered—specifically, Cross-site Request Forgery (CSRF). This vulnerability is apparent in the "Change Password" form, where the "newPassword" field is integral to the request sent to the database after a password change. This flaw poses a significant risk to the application's security.

Rated as high severity, the CSRF vulnerability exposes the system to potential exploitation, allowing unauthorized password changes without user consent. This could lead to severe consequences, including unauthorized access, manipulation of user accounts, and a compromise of sensitive information. The severity of this vulnerability is accentuated by the profound impact it could have on both system integrity and user privacy.

* Steps to Discover:

During the routine inspection of the "Profile and Self Service" page, my attention was drawn to the "Change Password" form. To understand the inner workings of the application, I utilized the "Developer Tools" to inspect the request sent to the database after a password change. This investigation brought to light the presence of the "newPassword" field, revealing the CSRF vulnerability and its potential for unauthorized password changes. Another thing I noticed is that the username is md5 hashed.

* Exploiting the Vulnerability:

With the CSRF vulnerability identified, I proceeded to exploit it by creating an HTML form designed to change a user's password without any authorization. The crafted form manipulated the "newPassword" field, and a user only needed to click the link to the page. Upon clicking, the user's password was automatically changed to "water" without their knowledge. This step-by-step exploitation demonstrated the real-world risk associated with CSRF vulnerabilities, showcasing the potential for unauthorized manipulation of user accounts.

* Damage it Could Cause:

The potential damage resulting from CSRF is notable. Attackers could force users to unknowingly change their passwords, leading to unauthorized access and potential account compromise. This could have cascading effects on the user's other accounts if password reuse is common.

* Countermeasures to Fix the Vulnerability:
* Anti-CSRF Tokens: Introduce anti-CSRF tokens within forms to validate the authenticity of requests.
* Authorize Password Changes: Require additional authentication or authorization for sensitive actions like password changes.

1. Insecure direct object references

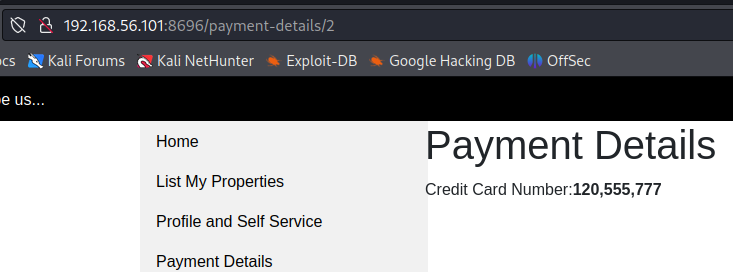
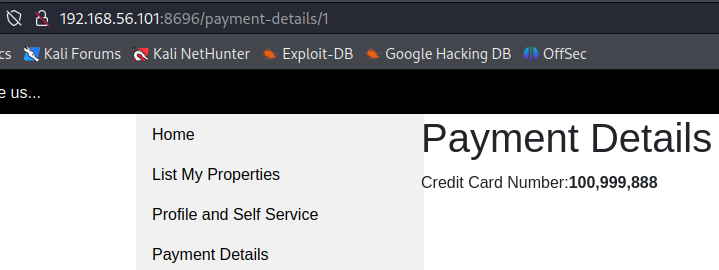
* Description and Severity Assessment:

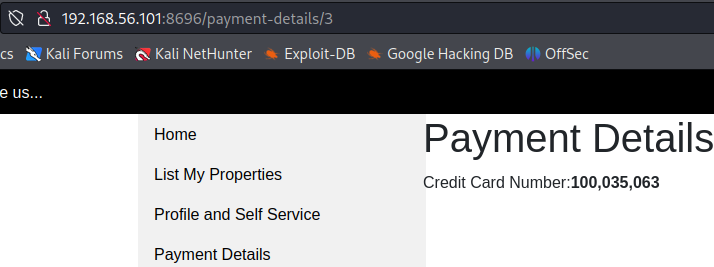
While inspecting the "Payment Details" page, a significant security vulnerability was identified—specifically, Insecure Direct Object References (IDOR). The vulnerability becomes apparent in the URL structure, where a numerical identifier is used to reference individual accounts. In this instance, the URL "http://192.168.56.101:8696/payment-details/2" caught attention, and altering the numerical value at the end of the URL revealed other users' payment details.

Categorized as a moderate to high severity, this IDOR vulnerability allows unauthorized access to sensitive information, potentially compromising user privacy. The impact is significant as it enables attackers to access data belonging to other users within the application.

* Steps to Discover:

While exploring the "Payment Details" page, I decided to inspect its structure. What immediately caught my attention was the URL format: "http://192.168.56.101:8696/payment-details/2". The numerical value at the end of the URL seemed like a potential vulnerability. To test this, I changed it to "1" and observed the page changing accordingly. This revealed another user's "Payment Details." Continuing the investigation, I changed the number to "3" and encountered yet another account's details. While other numeric values didn't yield results, it became apparent that at least two other accounts were accessible through this URL manipulation.





* Damage it Could Cause:

The potential damage resulting from the IDOR vulnerability is significant. Unauthorized access to other users' payment details compromises privacy and confidentiality. Attackers could leverage this information for malicious purposes or use it in conjunction with other attacks, such as phishing or social engineering.

* Countermeasures to Fix the Vulnerability:
* Proper Authorization Checks - implement access control checks to ensure users can only access their own data.
* Use Randomized Identifiers: Employ randomized and unique identifiers in URLs to prevent easy enumeration.

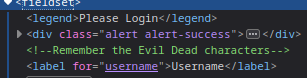
1. Information Leakage

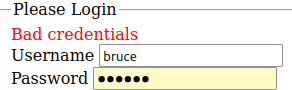
* Description and Severity Assessment:

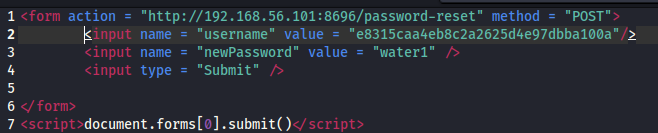
While reviewing the application, a notable security vulnerability emerged—specifically, Information Leakage. This flaw was identified through a comment in the login page code, potentially exposing details about the application's internal workings.

This vulnerability could provide attackers with insights into valid usernames, streamlining the process of username enumeration and facilitating unauthorized access attempts. The severity is moderate to high, posing a tangible risk to the overall security of the authentication system.

* Steps to Discover:

While closely examining the web pages, I looked for possible information leakage. This began with a discovery in the login page code, where a comment mentioning "Evil Dead" characters raised suspicion.

Curiosity led me to research the show, revealing possible clues about the application's inner workings. I manually brute-forced the username, taking advantage of the bad error messages, until I found that username “bruce” gave me a “Bad Credentials” error message, meaning it existed in the database.

I md5 hashed the “bruce” username and used the CSRF vulnerability to change the password of the account.

* Damage it Could Cause:

The Information Leakage vulnerability could result in unauthorized access, privacy breaches, and compromised user accounts. Exploited details might be used for credential stuffing or phishing attacks, potentially leading to data manipulation and unauthorized actions. Swift mitigation is essential to safeguard the application and user privacy.

* Countermeasures to fix the vulnerability:
* Comment cleanup in source code - Remove unnecessary comments in the code that provide insights into the application's internal structure or potential vulnerabilities.
* Source code maintenance – check the code regularly to mitigate the exposure of unnecessary comments that may contain sensitive information.

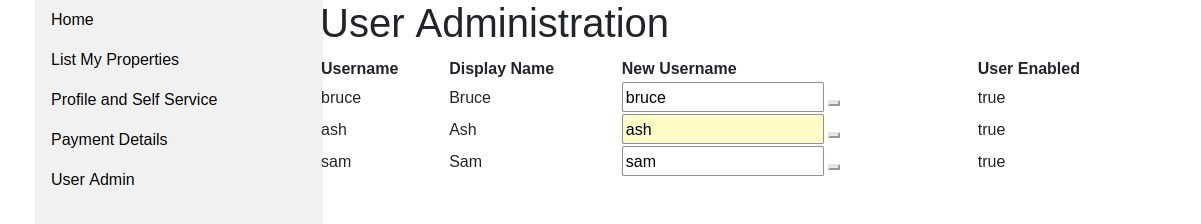
1. SQL Injection

* Description and Severity Assessment:

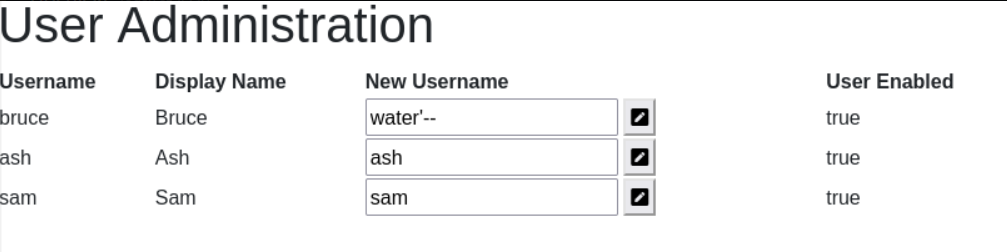
The SQL Injection vulnerability was identified while exploring the application under the "bruce" account. A crucial observation was made upon discovering an additional page labelled "User Admin," suggesting administrative privileges for the "bruce" account. Accessing this page revealed fields intended for modifying other user accounts' usernames.

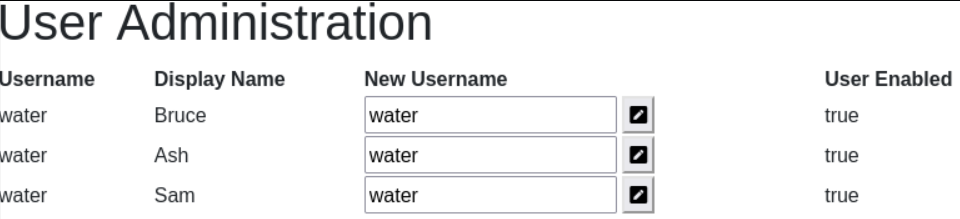
Rated as a high-severity vulnerability, the SQL Injection allows unauthorized manipulation of user account data. The ability to change usernames indiscriminately poses a significant threat to the application's integrity, potentially leading to unauthorized access and data compromise

* Steps to Discover:

Upon successfully logging into the "bruce" account, I initiated an examination of the pages within the application. The discovery of an additional page labeled "User Admin" caught my attention. Upon navigating to this page, I observed a form displaying the three user accounts along with fields to change their usernames. This observation raised suspicions about a potential vulnerability in the website, prompting further investigation.

* Exploiting the Vulnerability:

I took a closer look at the “New Username” field. I decided to try an SQL injection to change the accounts’ usernames. In the first field, I wrote “water’--” and clicked the “Enter” button. The page reloaded and the three usernames were changed to “water”.



* Damage it could cause:

Potential damage includes unauthorized access to user accounts, compromising sensitive information and threatening data integrity. This manipulation raises concerns about identity misrepresentation, operational disruptions, and the potential for reputational damage.

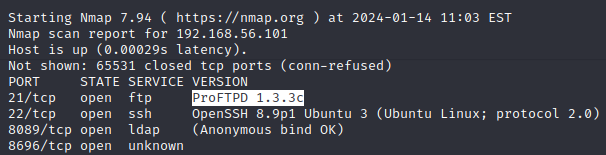
* Countermeasures to fix the vulnerability:
* Parameterized Queries - Utilizing parameterized database queries with bound, typed parameters and careful use of parameterized stored procedures in the database.
* Input Validation - Validate and sanitize user inputs to ensure that they adhere to expected formats and patterns, minimizing the risk of malicious injections.

1. FTP Server Backdoor

* Description and Severity Assessment:

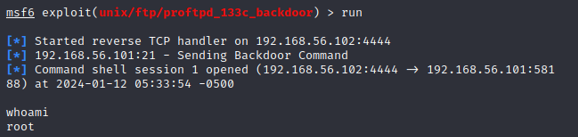
The FTP Server vulnerability stems from exploiting a ProFTPD 1.3.3c server discovered in an Nmap scan on port 21. Categorized as a critical security issue, this vulnerability provides unauthorized access to the server, granting the attacker root privileges. This heightened access poses serious risks, allowing potential system shutdown and unauthorized entry to sensitive data.

* Steps to Discover:

While conducting an Nmap scan on the vulnerable machine's IP address, my attention was drawn to port 21, which revealed the presence of a ProFTPD 1.3.3c FTP server. Recognizing the significance of this discovery, I inferred a potential vulnerability associated with the identified FTP server version. This inference was based on the understanding that outdated or unpatched versions often harbor security flaws that could be exploited.

* Exploiting Vulnerability:

While conducting online research to explore potential exploits for the identified ProFTPD 1.3.3c FTP server on port 21, I followed specific steps based on the insights gathered. The exploitation process successfully unfolded, culminating in the execution of a "whoami" command on the compromised server. The server's response, indicating "root," unequivocally confirmed the success of the exploitation, granting me root-level access.



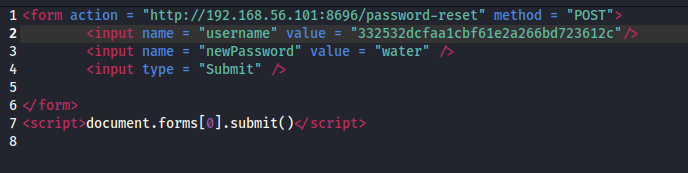
* Damage it Could Cause:

The FTP server vulnerability, when exploited to gain root access, poses severe risks. This includes unauthorized access, potential system shutdown, and the manipulation or exfiltration of sensitive data. The consequences extend to service disruptions, compromised data integrity, and the risk of further exploitation.

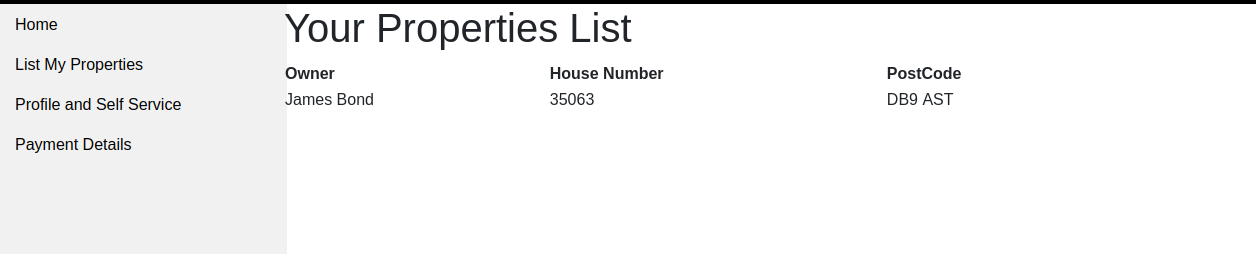
* Countermeasures to fix the vulnerability:
* Software Update – Ensure software is updated regularly with the latest security patches
* Isolate System – Keep critical systems isolated to minimize the impact of potential attacks, protecting the overall network from harm.

**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”.

****

Encrypts the files in the system by AES enctyption method, meaning the key for encryption and the decryption is the same.

Purpose: perform ransom

CSRF vulnerability exposes the system to potential exploitation, which allows unauthorised password changes without any sort of user consent. This could lead to very big problems, such as unauthorized access, user accounts manipulation, and exposure to sensitive information.