**Web Application Security Assessment using Vulnerability Assessment and Penetration Testing(VAPT) Methodology**

**Target** : Altoro Mutual(http://testfire.net/)

1. **Overview**

* During the testing of Altoro Mutual, we focused on identifying vulnerabilities related to SQL injection, Cross-Site Scripting, and HTML injection. Our approach involved using both automated tools and manual techniques to identify potential vulnerabilities.
* VAPT is a structured and comprehensive approach to evaluating and enhancing the security of full-stack web applications. VAPT methodology combines two distinct yet complementary processes: Vulnerability Assessment, which identifies and prioritizes potential weaknesses, and Penetration Testing, which assesses the application's resilience to real-world attacks.
* In this exploration of Web Application Security Assessment using VAPT, we will delve into the intricacies of this methodology, from the initial preparations to the final reporting stage. We will examine how to identify and address vulnerabilities, simulate attacks, and ultimately fortify web applications against a multitude of threats. The aim of this methodology is not only to discover and remediate security flaws but also to enhance the overall robustness of web applications.

## **List of teammates**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Name** | **Roll No** |
| **1.**  **2.**  **3.** | **Y. Rudra Prakash**  **T. Tharuni**  **S. Anu** | **20481A05P5**  **20481A05M7**  **20481A05L9** |

# 

# **List of Vulnerable Parameter, Location discovered**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Vulnerability path** | **Name of the vulnerability** | **Reference CWE** |
| **1** | http://testfire.net/search.jsp?query= | **XSS** | **CWE - 79** |
| **2** | http://testfire.net/login.jsp | **SQL INJECTION** | **CWE-89** |
| **3**  **4**  **5** | <http://testfire.net/feedback.jsp>  <http://testfire.net/login.jsp>  http://testfire.net/login.jsp | **HTML INJECTION**  **IDOR Vulnerability**  **Brute Force** | **CWE-79**  **CWE-639**  **CWE-307** |

## **4.** **Other Information**

**· Tools Used :**

* 1. Kali Linux
  2. Firefox Browser
  3. Burp suite

**5 . Main vulnerability representation format :-**

1. **Vulnerability Name**: Cross-Site Scripting (Reflected)

**CWE** : CWE-79

**OWASP Category**: A03:2021 – Injection

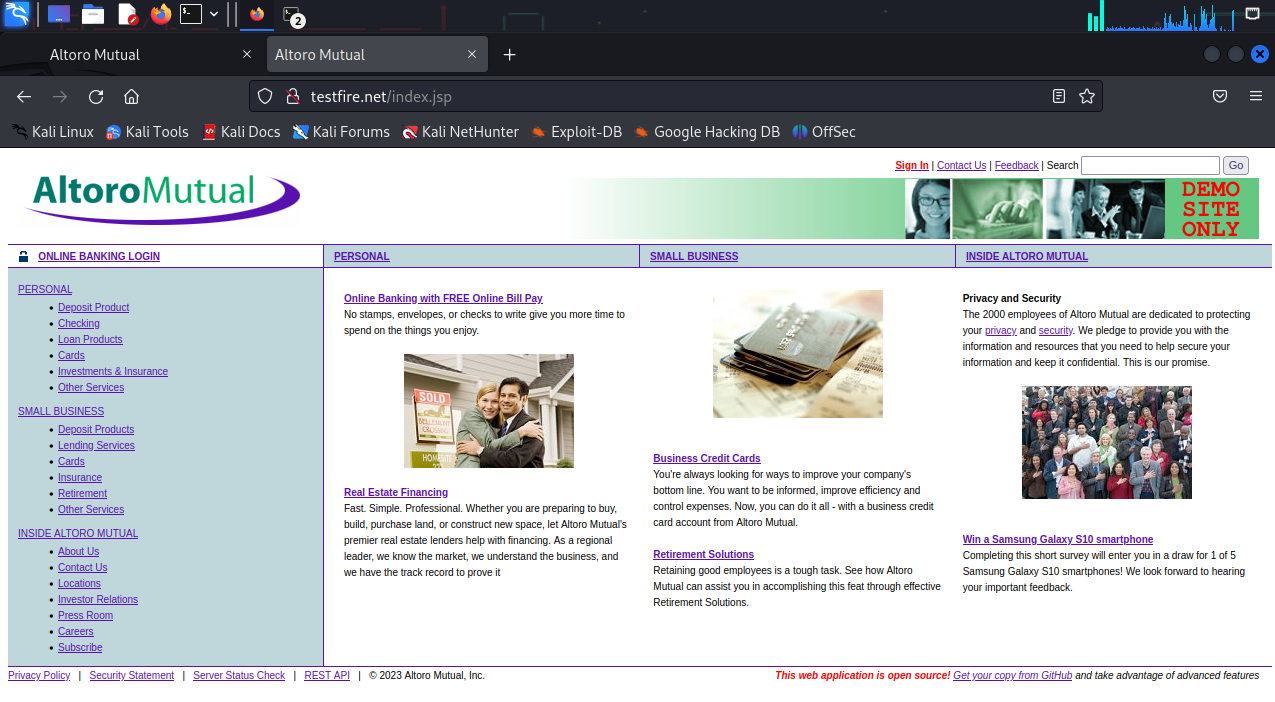
**Description**: Untrusted data enters a web application, typically from a web request.**Business Impact**: The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user. For example, the attacker might inject XSS into a log message, which might not be handled properly when an administrator views the logs.

**Vulnerability Path**: http://testfire.net/search.jsp?query=

**Vulnerability Parameter**:http://testfire.net/search.jsp?query=%3Cscript%3Ealert%28%22XSS+attack%22%29%3C%2Fscript%3E

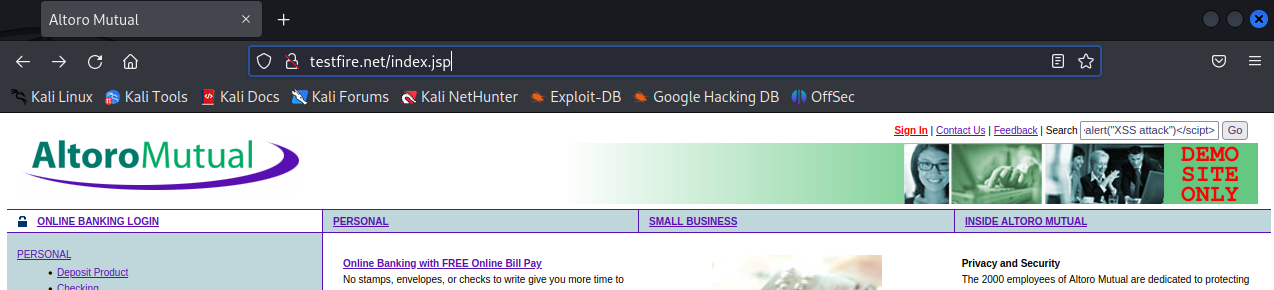
**Steps to Reproduce** :

Step 1: Access the URL testfire.net

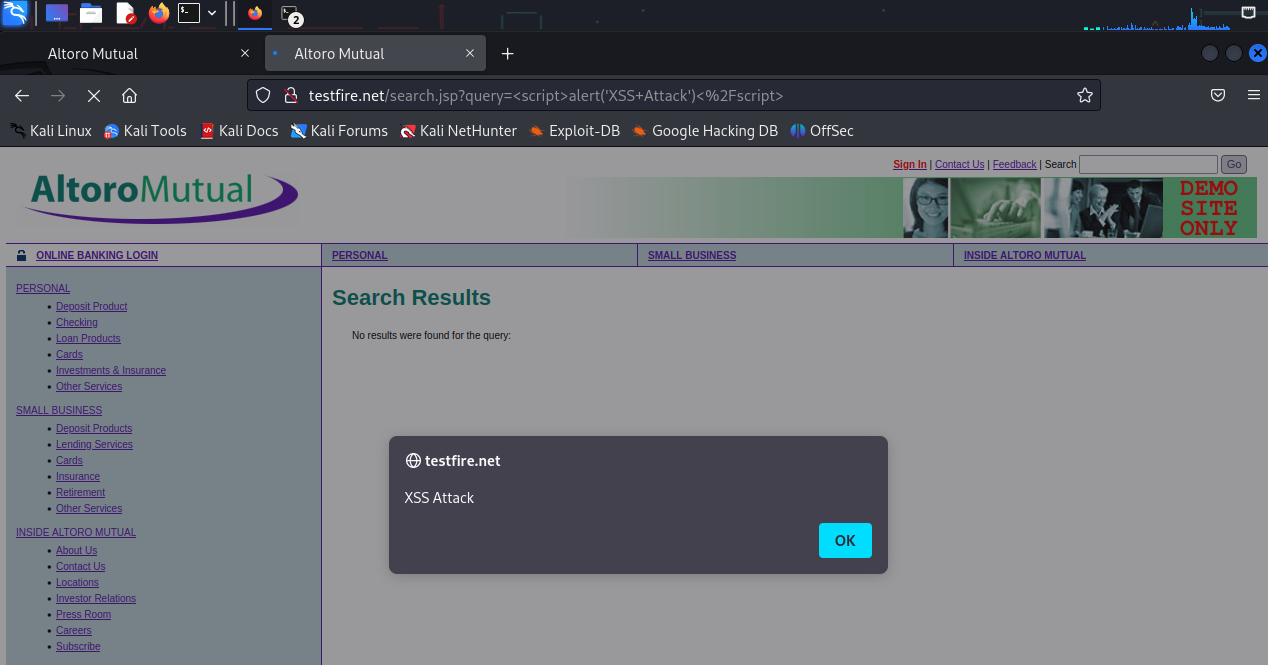


Step 2: Go to the Search and enter a xss payload

<script>alert(‘XSS Attack’)</script>



Step 3: Now there will be a pop up which is given in our xss payload.



**Recommendation**:

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth.

1. **Vulnerability Name :-** SQL Injection

**CWE : -** CWE-89

**OWASP Category :-** A03:2021 – Injection

**Description :-** Consider the following SQL query:

SELECT \* FROM Users WHERE Username='$username' AND Password='$password'

A similar query is generally used from the web application in order to authenticate a user. If the query returns a value it means that inside the database a user with that set of credentials exists, then the user is allowed to login to the system, otherwise access is denied. The values of the input fields are generally obtained from the user through a web form. Suppose we insert the following Username and Password values:

$username = 1' or '1' = '1

$password = 1' or '1' = '1

The query will be:

SELECT \* FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'

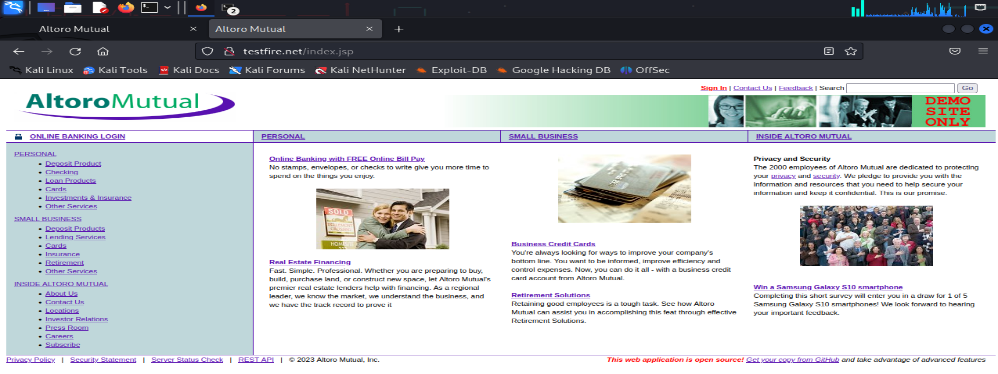
**Business Impact** :- A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into dataplane input in order to affect the execution of predefined SQL commands. SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server. SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces.

**Vulnerability Path:** <http://testfire.net/login.jsp>

**Vulnerability Parameter:** <http://testfire.net/bank/main.jsp>

**Steps to Reproduce :-**

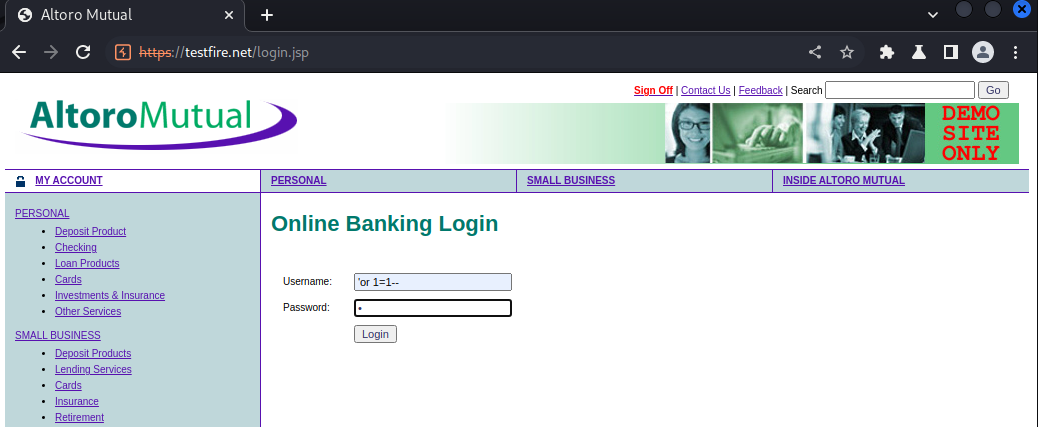
Step-1: Access the Url



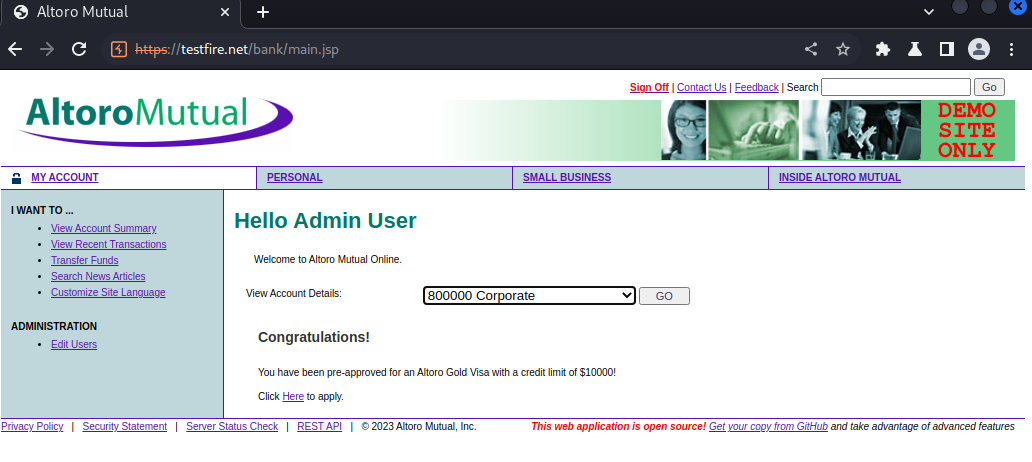
Step-2: Now give an sql payload as input

Username: ‘or 1=1--

Password: 1



Step-3: We have logged in as admin user.



**Recommendations** :-

* Do not rely on client-side input validation
* Use a database user with restricted privileges.
* Use prepared statements and query parameterization.
* **­­­­­­­­**Scan your code for SQL injection vulnerabilities.

1. **Vulnerability Name :-** HTML Injection-Reflected (POST)

**CWE :-** CWE-79

**OWASP Category :-** A03:2021 – Injection

**Description :-** HTML Injection also known as Cross Site Scripting. It is a security vulnerability that allows an attacker to inject HTML code into web pages that are viewed by other users.

**Business Impact:**-

A phony form might be used by the attacker to steal password information saved in the browser or to fool a user into entering their login information. Malicious actors may be granted administrative access to the online application if the targeted user has those rights.

By carrying out an assault that is visible to the public, the attacker might seriously damage the image of the business, organization, or even nation. Users or clients may make poor judgments and lose faith in your cybersecurity procedures if a high-value page is vandalized or exploited to propagate misinformation.

HTML injection might be used as a technique by the attacker to progress to more severe assaults like CSRF.

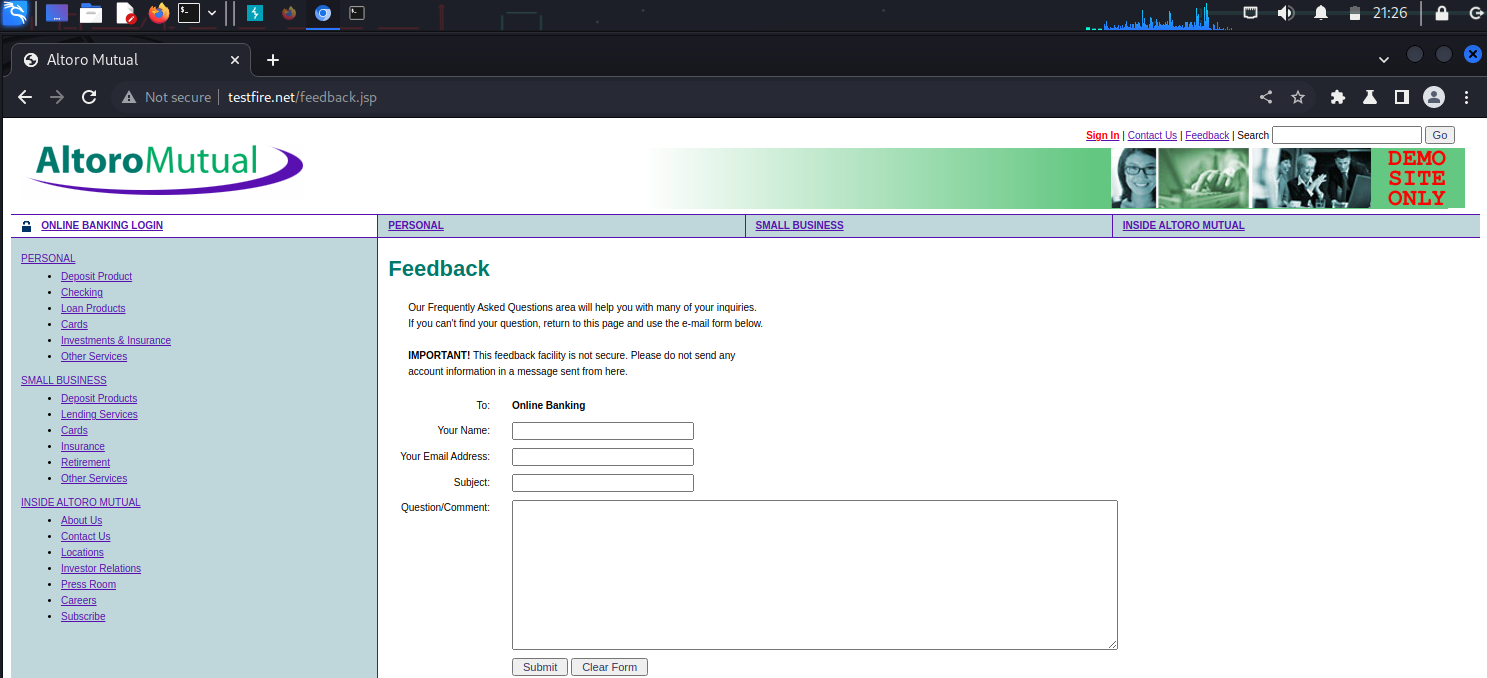
The attacker creates malicious links with his HTML content inserted into them, then emails the URLs to the victim. Because the page is hosted on a reputable domain, the user views it, which causes his identity to be stolen.

**Vulnerability Path**[:](http://10.0.2.4/001/) http://testfire.net/feedback.jsp

**Vulnerability Parameter**:- http://testfire.net/sendFeedback

**Steps to Reproduce :-**

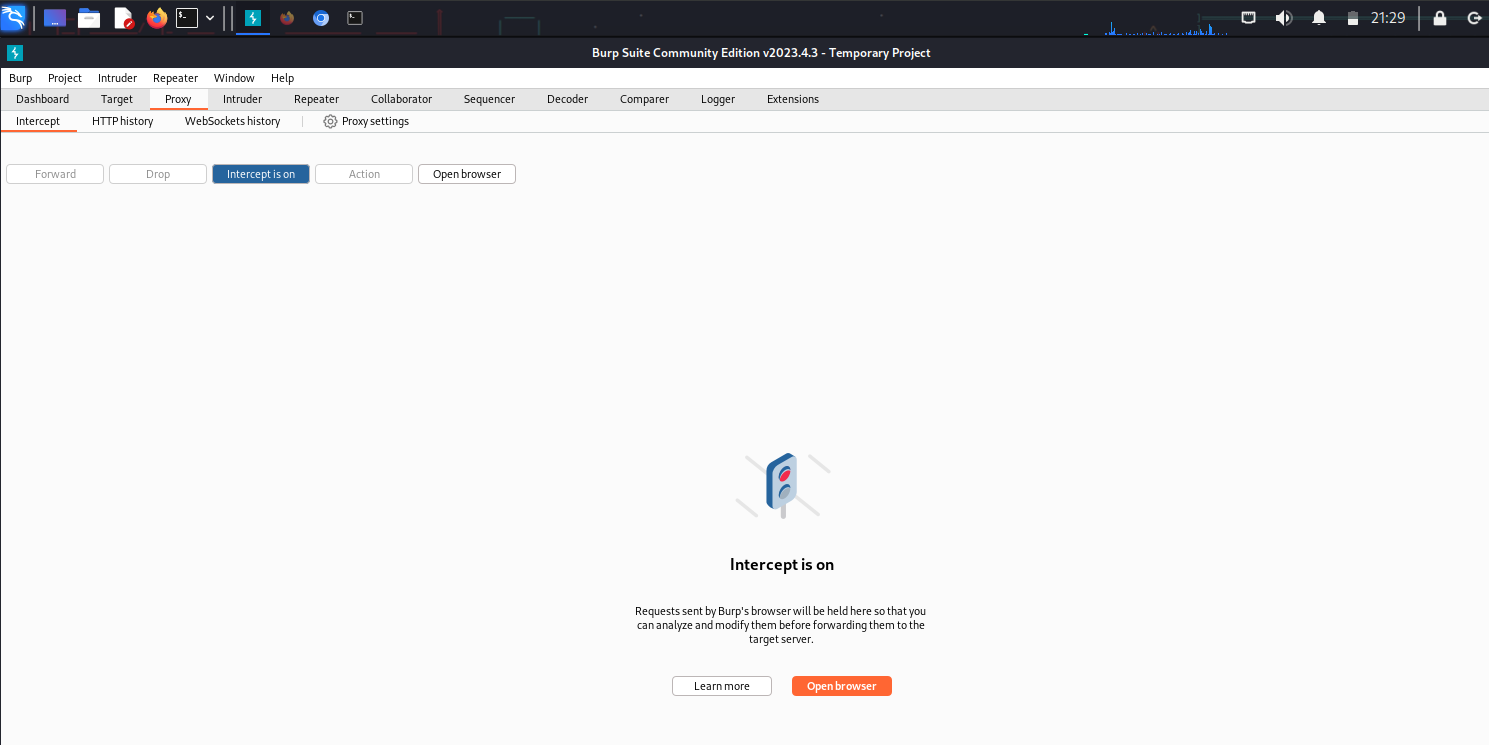
Step-1: Access the url:



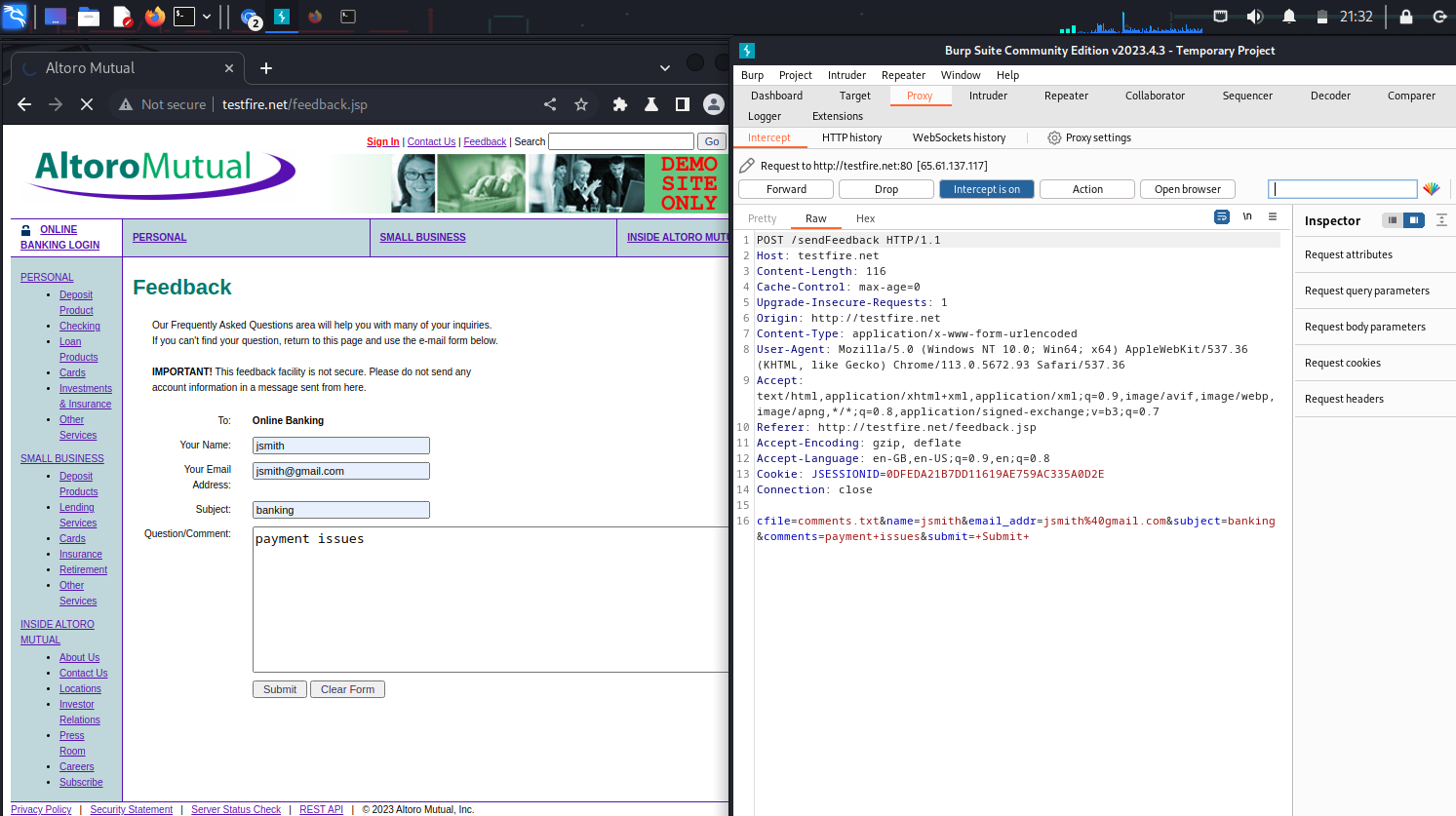
Step-2: Now give some dummy data and open burp suite and wait for the request while making the proxy on.



Step-3: Go to proxy and turn the intercept on

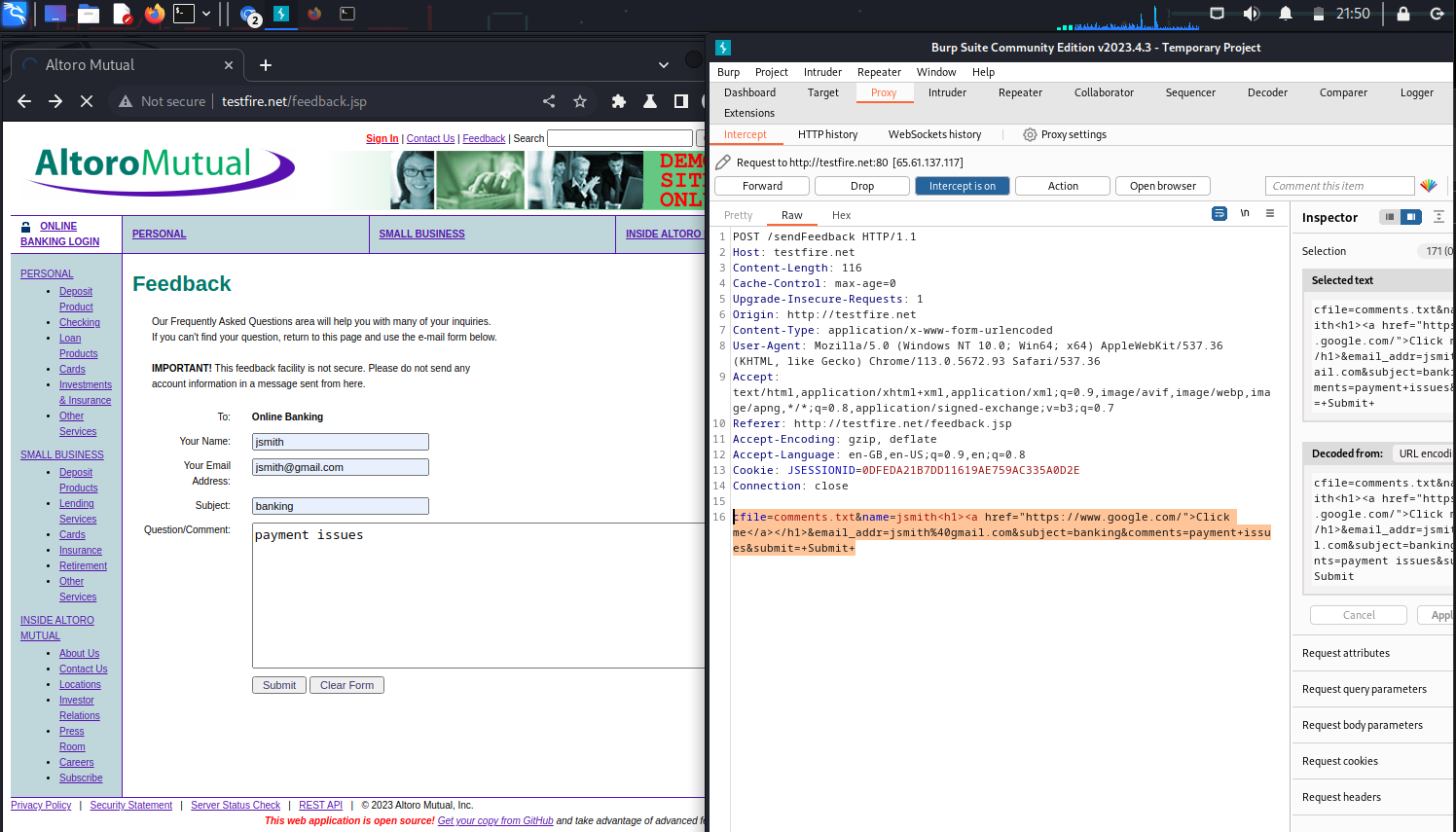


Step-4: We successfully intercepted the request and can see the request data in raw form.

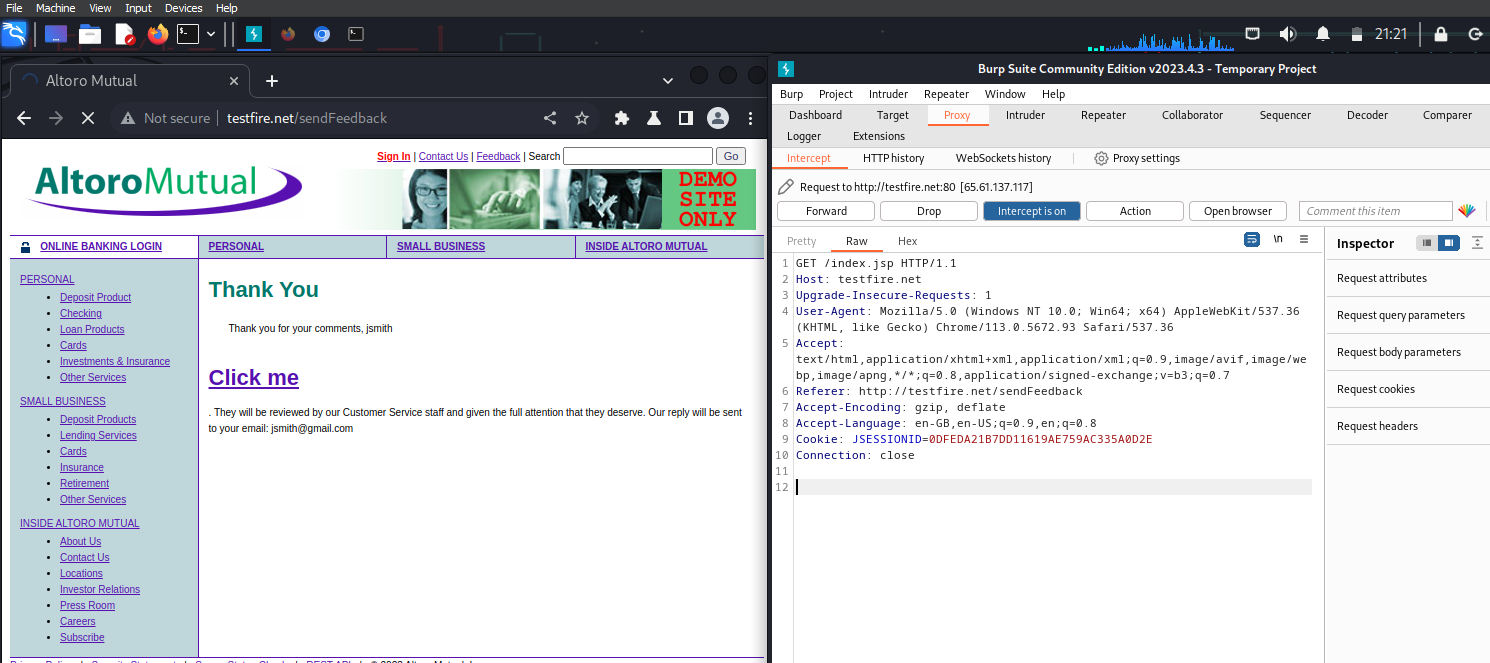


Step-5: Now you can see the given first name and email clearly and we inserted a anchor tag in between the first name value.

<h1><a href=”https://www.google.com/>Click me</a></h1>



Step-6: We successfully proved that there is no input validation ,as our input reflected on the site.



**Recommendation** :-

It is therefore essential to have appropriate data validation in place to prevent such attacks. Every input should be checked if it contains any script code or any HTML code. One should check, if the code contains any special script or HTML brackets – <script></script>, <html></html>.

The most common way of detecting HTML injection is by looking for HTML elements in the incoming HTTP stream that contains the user input. A naive validation of user input simply removes any HTML-syntax substrings (like tags and links) from any user-supplied text.

1. **Vulnerability Name :-** IDOR Vulnerability

**CWE :-** CWE-639

**OWASP Category :-** A04:2007 – Insecure Direct Object Reference

**Description :-**

Insecure Direct Object Reference (IDOR) is a critical web application security vulnerability that arises from improper authorization and access control. It is essential for developers and security professionals to be aware of this vulnerability and take proactive measures to prevent and mitigate it in their applications.

**Business Impact:**-

In the business context, an Insecure Direct Object Reference (IDOR) vulnerability can have serious and wide-ranging consequences. The primary concern is data security. IDOR vulnerabilities can lead to unauthorized access to sensitive data, which may include customer records, financial information, intellectual property, and confidential documents. Such unauthorized access can result in data breaches, causing significant reputational damage and undermining customer trust, leading to a potential loss of customers.

Furthermore, these vulnerabilities can have financial implications. They can be exploited to manipulate financial records or transactions, potentially leading to financial losses, such as unauthorized fund transfers or fraudulent activities. In cases involving personal data, organizations may face legal and regulatory consequences, including fines and legal actions, for non-compliance with data protection regulations.

The impact extends to the organization's reputation, which is a critical asset in business. News of a data breach or security incident can harm an organization's reputation, making it difficult to attract new customers and potentially causing customer churn. Additionally, competitors may take advantage of IDOR vulnerabilities to access sensitive business information, gaining a competitive edge.

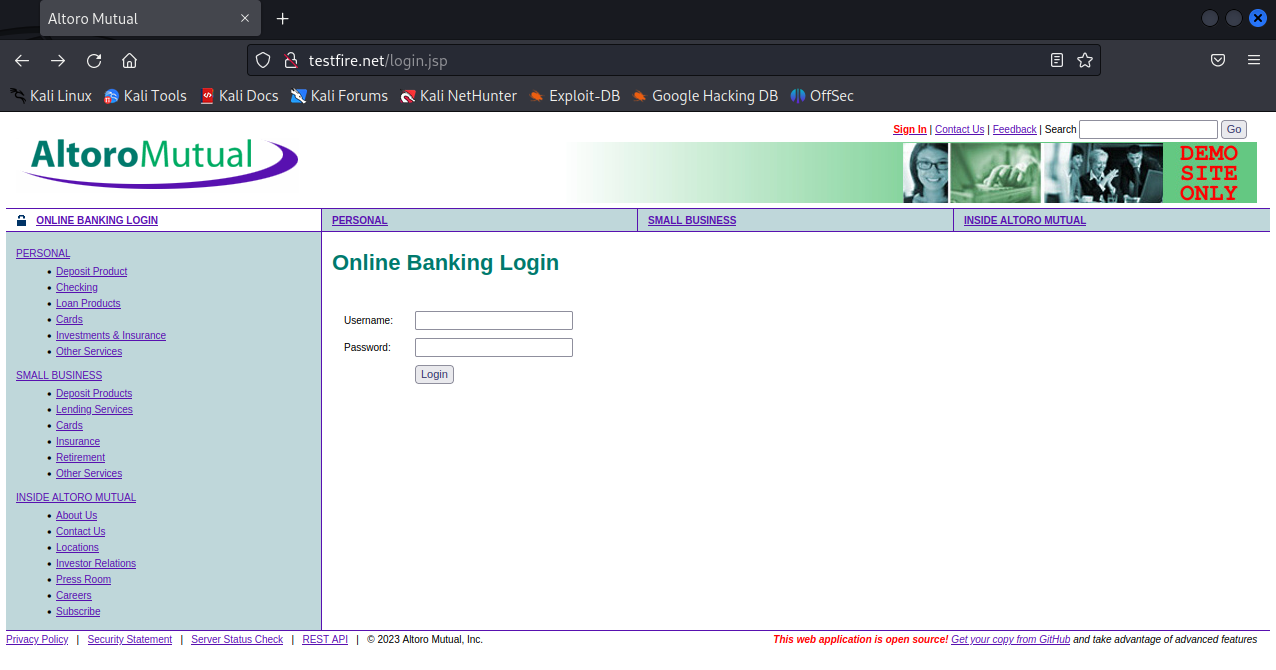
Overall, IDOR vulnerabilities can disrupt business operations, damage reputation, result in financial losses, and lead to legal and regulatory challenges. To mitigate these impacts, organizations must prioritize security measures, regularly assess and test their applications, and implement robust access controls and data protection mechanisms throughout their software development lifecycle. These proactive steps can help prevent IDOR vulnerabilities and minimize their potential consequences on a business.

**Vulnerability Path**[:](http://10.0.2.4/001/) http://testfire.net/login.jsp

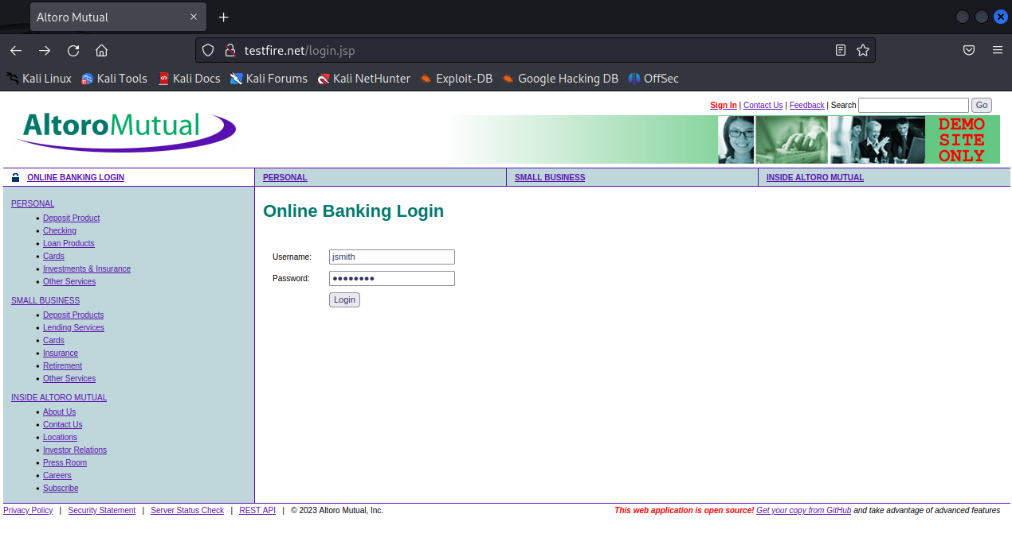
**Vulnerability Parameter**:- <http://testfire.net/sendFeedback>

**Steps to Reproduce :-**

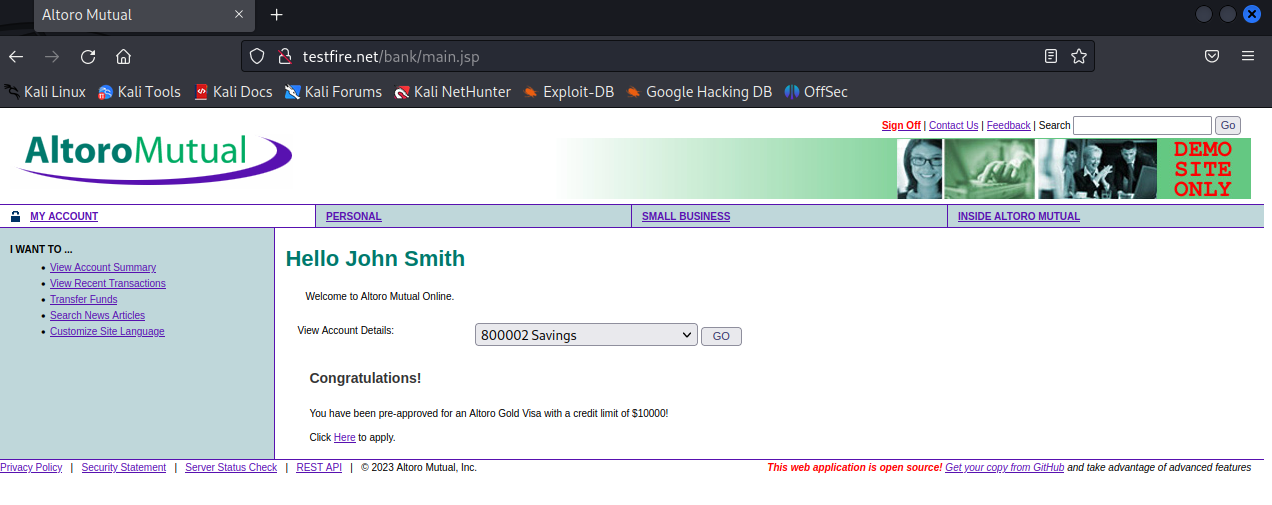
Step-1: Access the url:

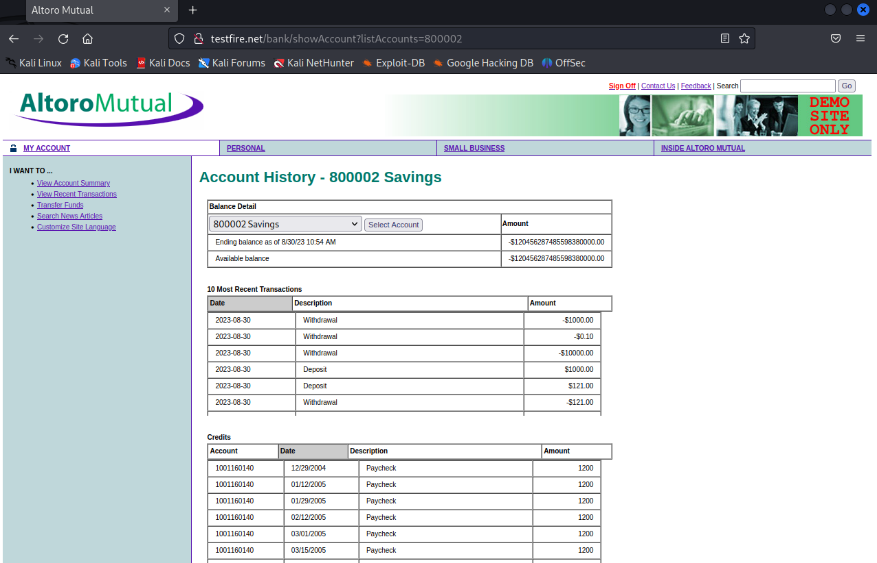


Step-2: Login into the bank account Username: jsmith password: demo1234

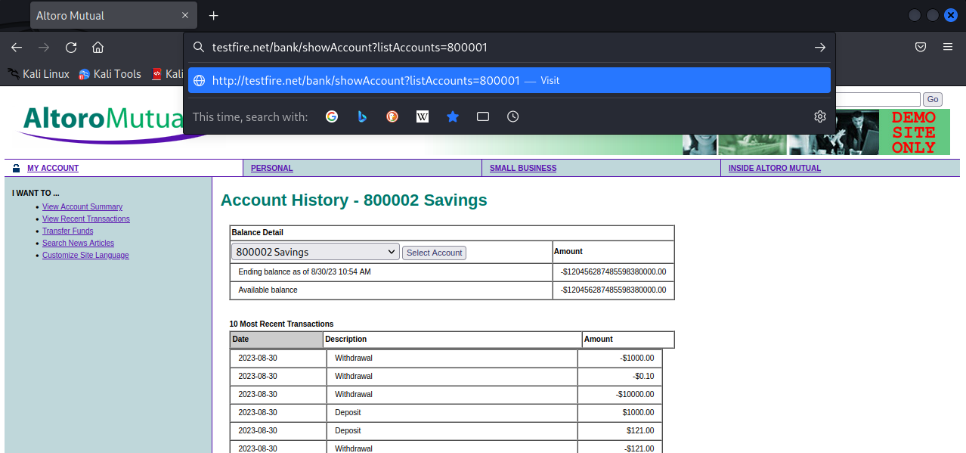


Step-3: Check the account details of jsmith

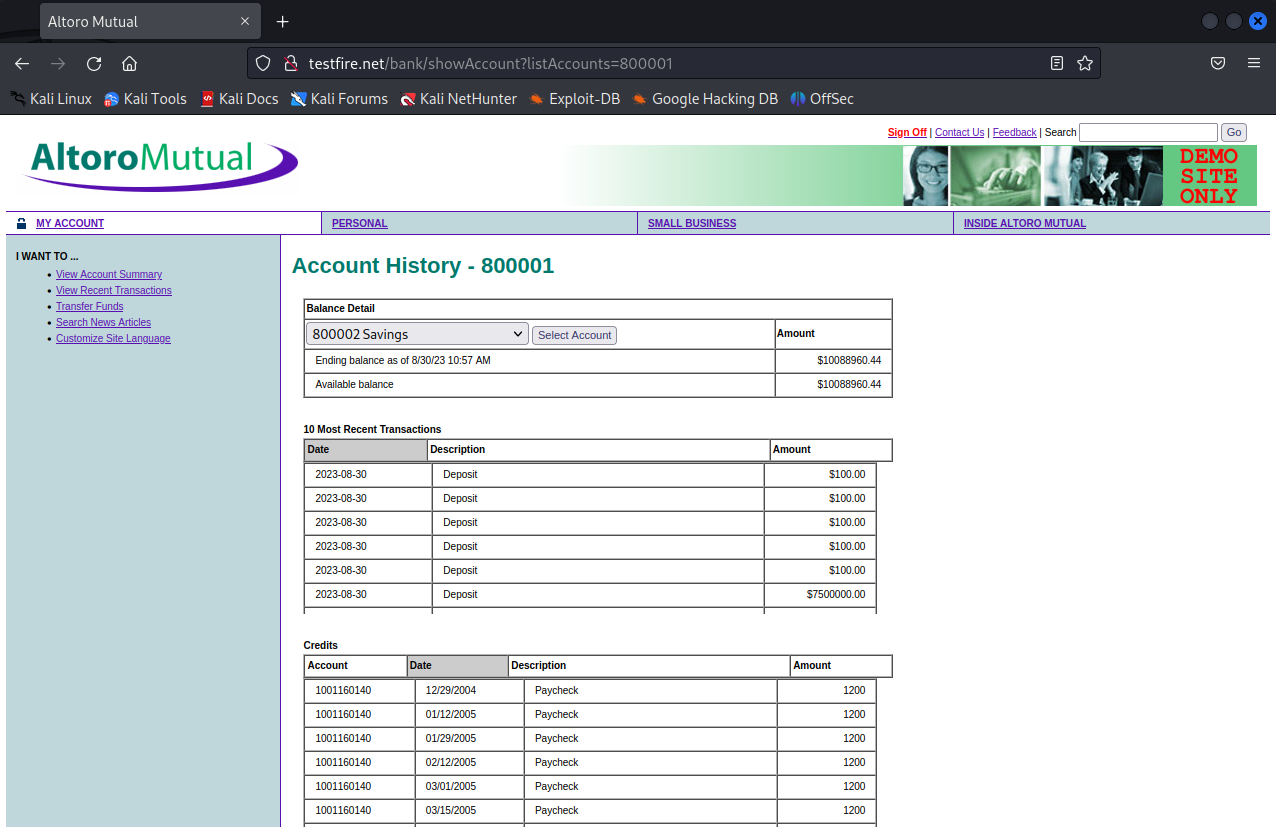




Step-4: In URL <http://testfire.net/bank/showAccount?listAccounts=800002> change the account to 800001



Step-5: Now you can access the deatils of the account 800001



**Recommendation** :-

* Implement Proper Access Controls
* Avoid Direct Object References in URLs or Parameters
* Apply Input Validation and Sanitization
* Use Unique and Randomized Identifiers

1. **Vulnerability Name :-** Brute Force

**CWE :-** CWE-307

**OWASP Category :-** A2:2017-Broken Authentication

**Description :-**

Brute force is a straightforward and systematic method of attempting to discover a password, encryption key, or access code by trying all possible combinations until the correct one is found. It is a common and often time-consuming technique used in cybersecurity attacks and password cracking.

**Business Impact:**

Brute force attacks can have significant business impacts, primarily due to the potential compromise of sensitive information and the resources required to mitigate and recover from such attacks. When successful, these attacks can lead to unauthorized access to systems, data, or accounts, resulting in a range of adverse consequences.

First and foremost, the unauthorized access to sensitive data can lead to data breaches, which may expose customer records, financial information, intellectual property, and confidential documents. This can have severe reputational consequences, eroding trust and confidence among customers and business partners. Loss of customer trust can result in customer churn, decreased business, and a negative impact on an organization's brand and market position.

Furthermore, the costs associated with responding to and recovering from a successful brute force attack can be substantial. Organizations may need to invest in incident response, forensic analysis, and remediation efforts, in addition to implementing enhanced security measures to prevent future attacks. Legal and regulatory consequences may also arise if data protection laws are breached, potentially leading to fines and legal actions.

In the case of downtime or operational disruption caused by a brute force attack, there can be direct financial losses due to interrupted services or lost business opportunities. Organizations must allocate resources to address these issues, incurring additional costs for remediation and recovery efforts.

To mitigate the business impact of brute force attacks, organizations should implement robust security measures, such as strong password policies, account lockout mechanisms, rate limiting, and multi-factor authentication. These measures can help deter attackers and reduce the likelihood of a successful brute force attack. Additionally, investing in proactive security practices and training for employees can further safeguard against the financial and reputational consequences of such attacks.

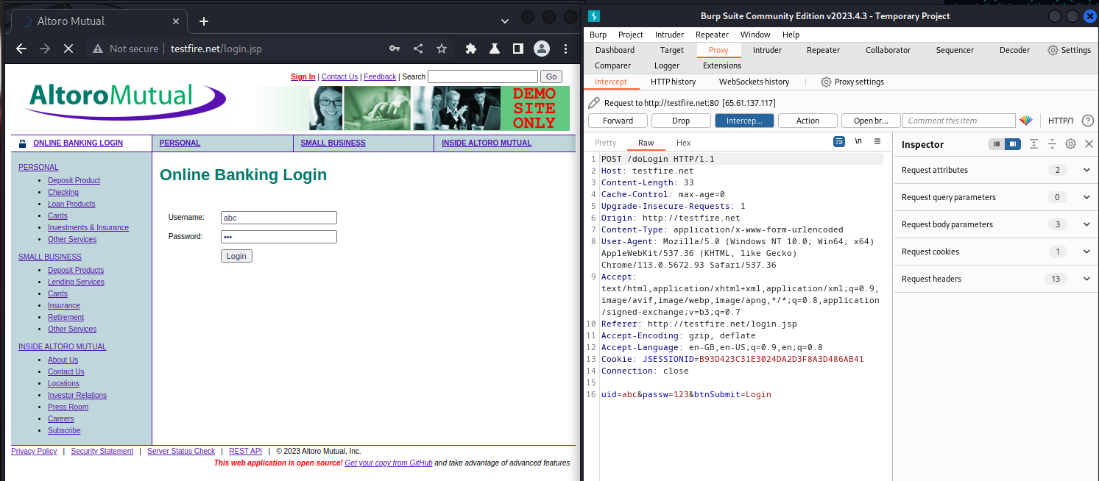
**Vulnerability Path**[:](http://10.0.2.4/001/) http://testfire.net/login.jsp

**Vulnerability Parameter**:- <http://testfire.net/sendFeedback>

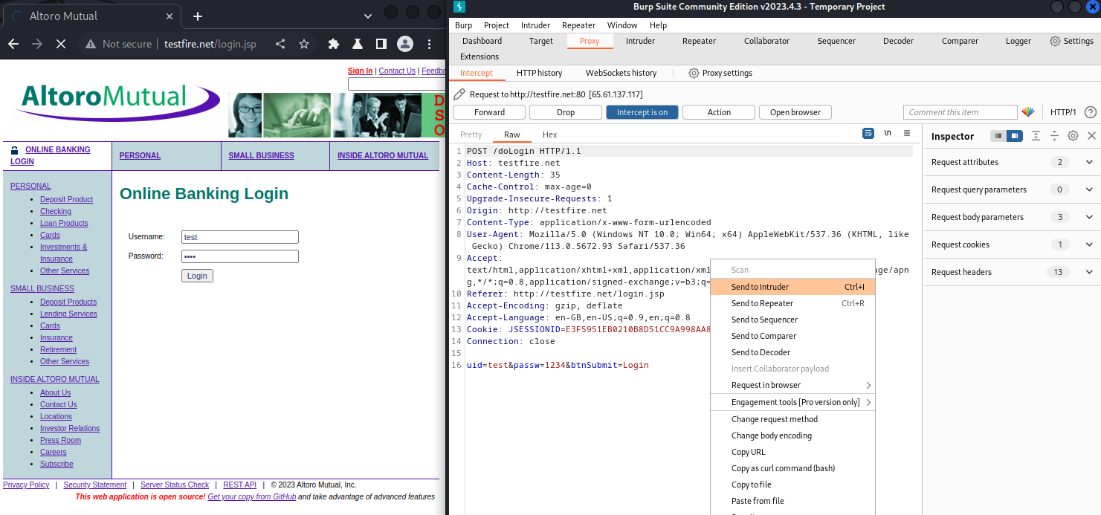
**Steps to reproduce:**

Step-1: Access the url

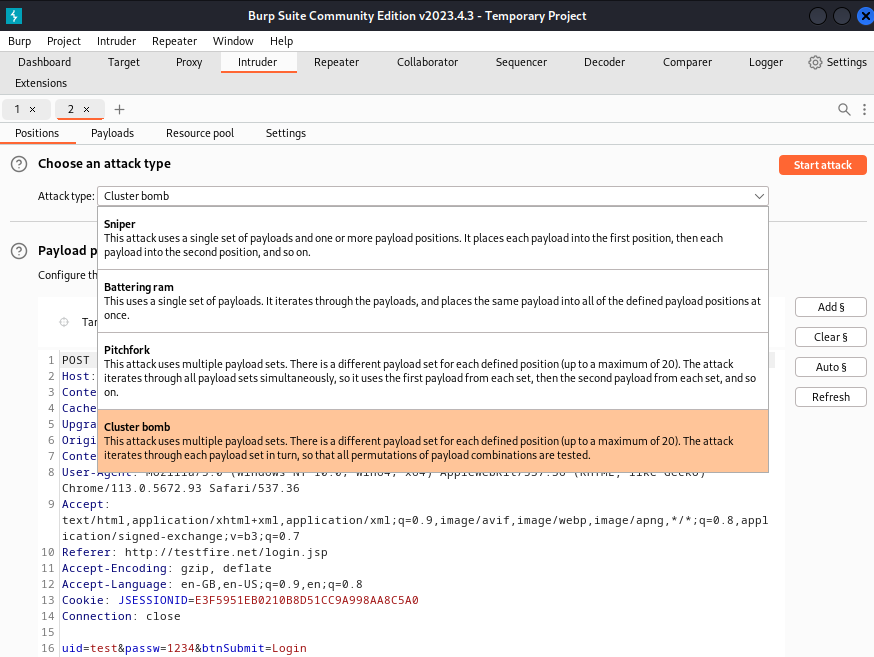
Step-2: Try to login with a random credentials. Using Burp suite Intercept the login request



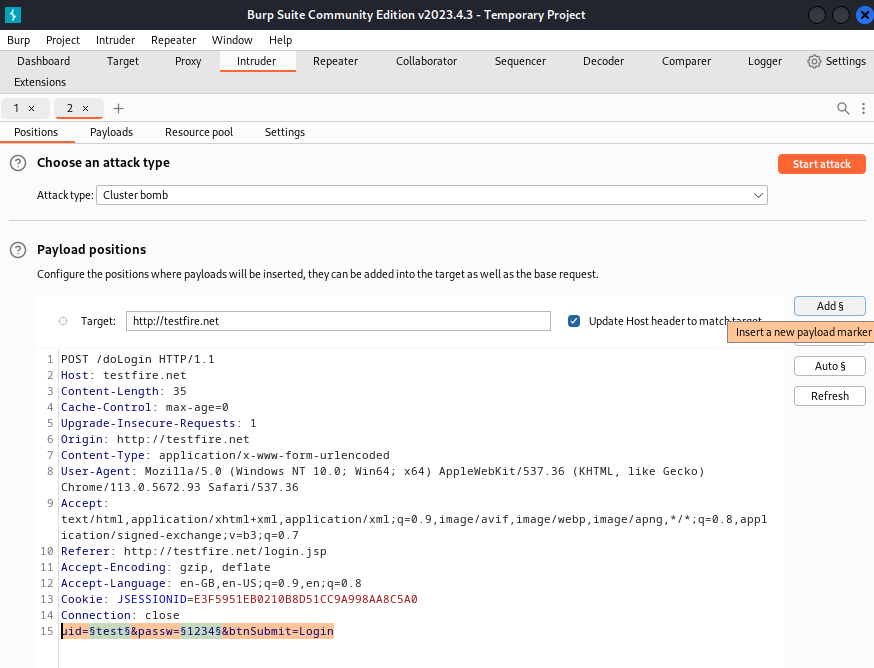
Step-3: Send the request to the intruder



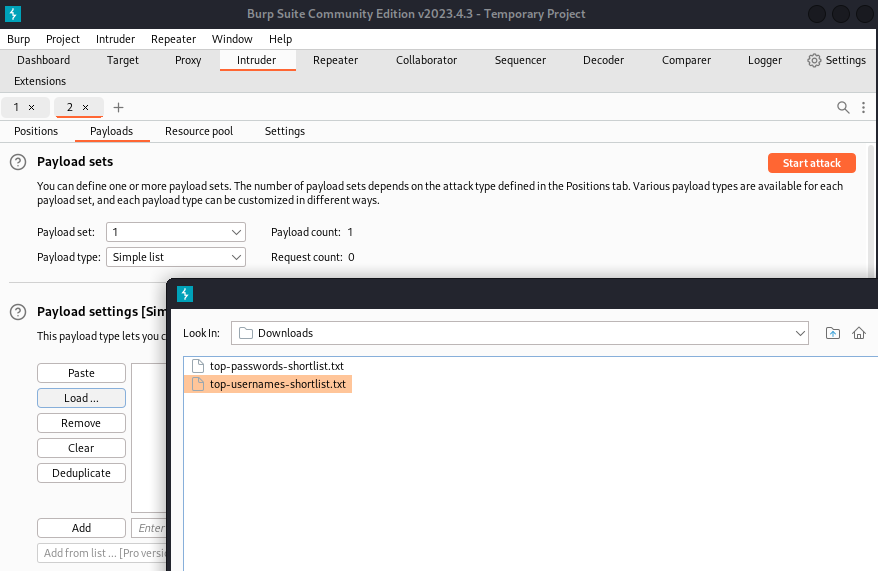
Step-4: Select the attack type as Cluster Bomb as we have to payload two values(username & password)



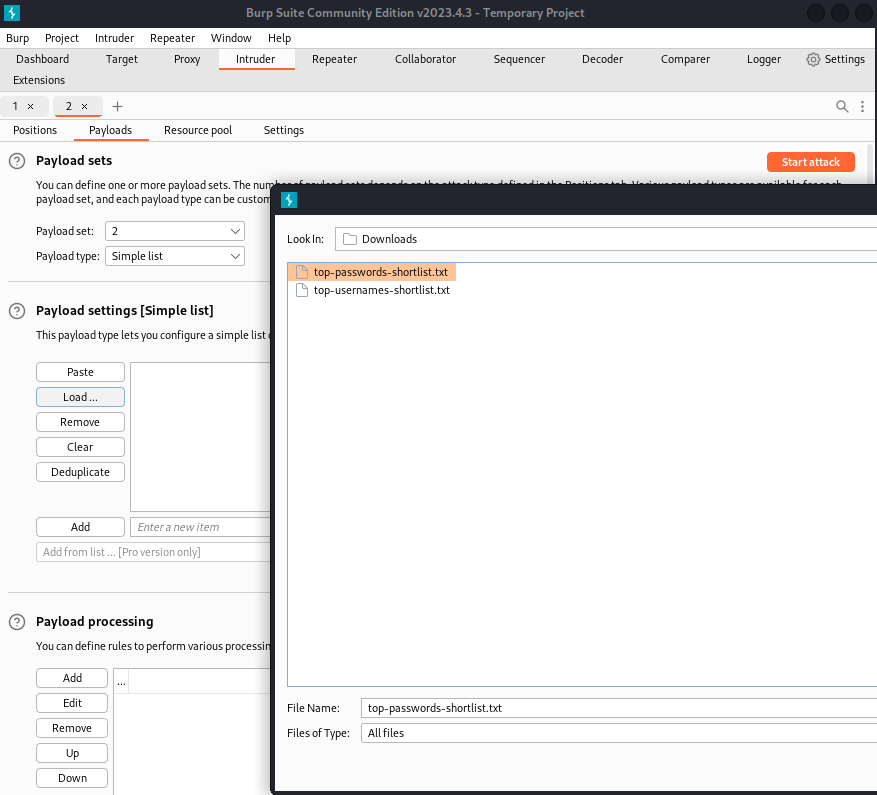
Step-5: Select the Username & password values and click ‘add $’ to brute force



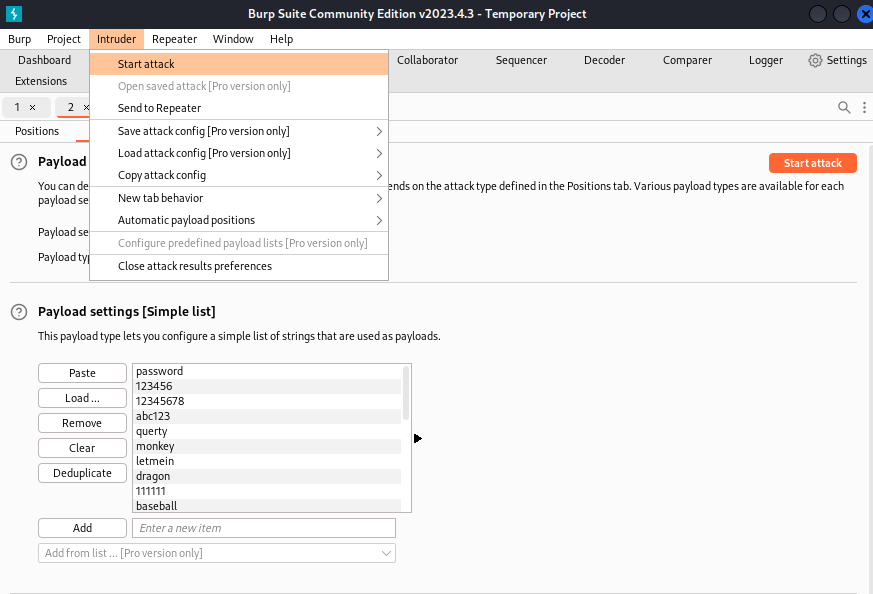
Step-6: For payload set 1 add username word list



Step-7: For payload set 2 add password word list

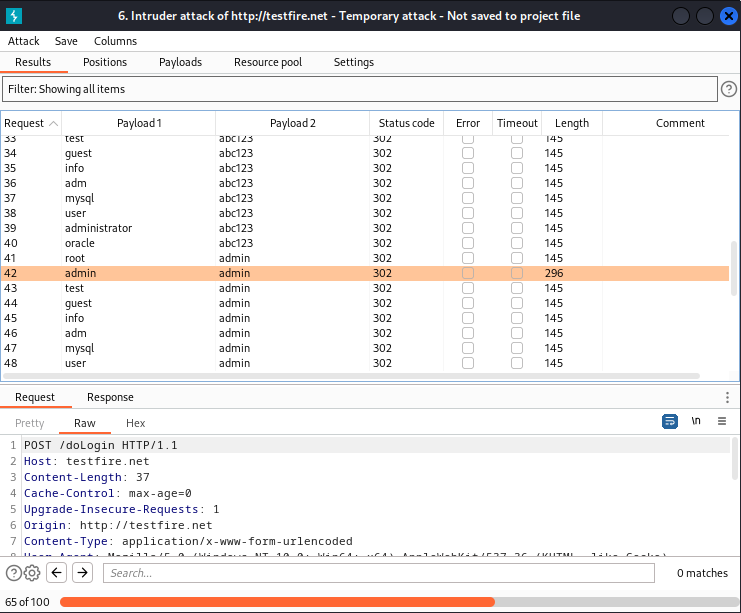


Step-8: Start the attack

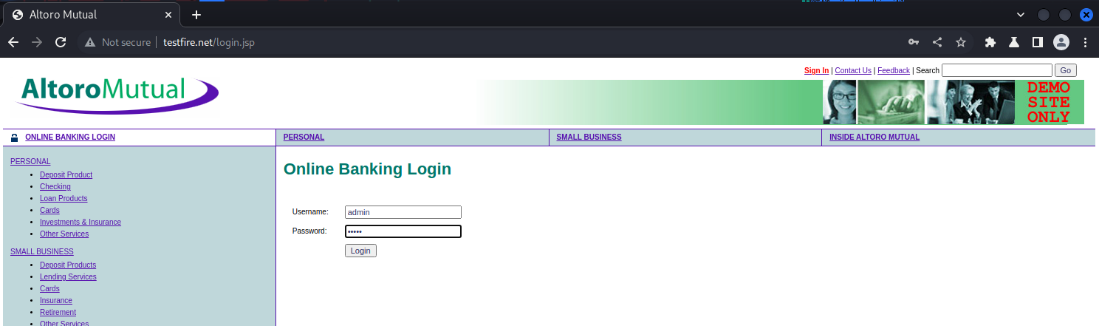


Step-9: Check the Length for a difference in value

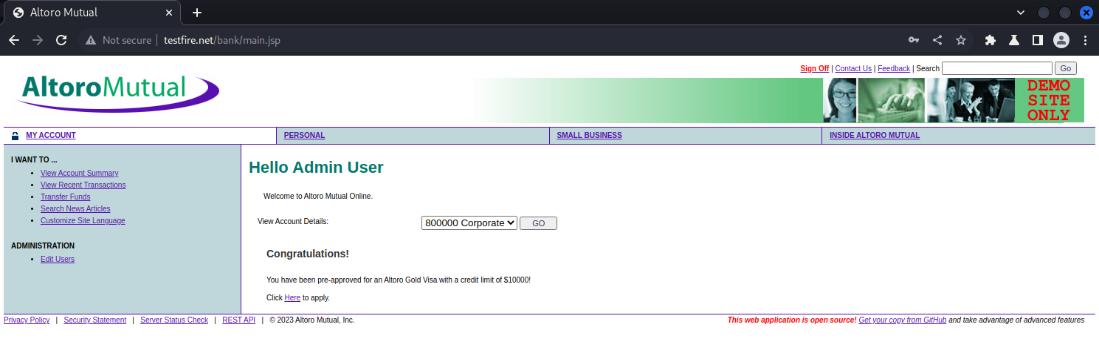
Here for username: admin and password: admin we got a Length value: 296



Step-10: Now Check the username and password



Step-11: We have successfully logged in



**Recommendations:**

* Use Strong and Unique Passwords
* Implement Multi-Factor Authentication (MFA)
* IP Blocking or Whitelisting
* Security Tokens and Session Management