

# **AI FOR CYBERSECURITY WITH IBM QRADAR**

## **ASSIGNMENT – 1**

### **PERFORMING VULNERABILITIES ON WEBSITES**

**NAME: SHAIK MUHAMMED FAIZAAN ALI**

**BRANCH: ELECTRONICS AND  
COMMUNICATION ENGINEERING**

**CAMPUS: VIT-VELLORE**

**EMAIL ADDRESS:**

**shaikmuhammed.faizaan2021@vitstudent.acin**

## **BROKEN ACCESS CONTROL:**

Broken access control refers to a security vulnerability that occurs when an application's access control mechanisms are improperly configured or not effectively enforced. Access control is the process of regulating who can access what resources and perform specific actions within a system. Broken access control vulnerabilities can lead to unauthorized users gaining access to restricted resources, performing actions beyond their privileges, and potentially compromising the security of the application and its data.

There are several ways in which broken access control can manifest:

1. **\*\*Improper Authorization:** This occurs when an application fails to properly verify a user's authorization before granting access to certain resources or functionality. This can allow unauthorized users to perform actions they shouldn't have access to.
2. **Insecure Direct Object References (IDOR):\*\*** Insecure direct object references occur when an attacker is able to manipulate input parameters, such as URLs or form fields, to access resources they shouldn't have access to. This could allow them to access sensitive information or perform unauthorized actions.
3. **\*\*Privilege Escalation:** Privilege escalation happens when a user with lower privileges is able to gain access to resources or perform actions that are normally restricted to users with higher privileges. This can occur due to improper validation or lack of authorization checks.
4. **\*\*Horizontal and Vertical Privilege Escalation:** Horizontal privilege escalation occurs when a user can access resources belonging to other users with the same level of privilege. Vertical privilege escalation occurs when a user gains access to resources intended for users with higher levels of privilege.
5. **\*\*Insecure API Endpoints:** If APIs do not properly validate and authorize requests, attackers can exploit these vulnerabilities to gain unauthorized access to sensitive data or perform actions.
6. **\*\*Broken Session Management:** If an application's session management is flawed, attackers could hijack legitimate user sessions or bypass authentication mechanisms.

To mitigate broken access control vulnerabilities:

**\*\*Proper Authorization:** Implement strict and proper authorization mechanisms that check a user's privileges before allowing access to resources or functionality.

**\*\*Use Role-Based Access Control (RBAC):** Implement RBAC to ensure that users have only the privileges necessary for their roles and responsibilities.

**\*\*Use Principle of Least Privilege:** Ensure that users are granted the minimum privileges required to perform their tasks and no more.

**\*\*Secure Object References:** Avoid exposing direct references to sensitive resources in URLs or other accessible parts of the application.

**\*\*Access Control Testing:** Regularly test the application for access control vulnerabilities, both manually and through automated security testing tools.

**\*\*Secure API Design:** Implement strong authentication and authorization for APIs, ensuring that only authorized users can access them.

**\*\*Regular Security Audits:** Conduct regular security assessments, code reviews, and penetration testing to identify and address access control vulnerabilities.

By addressing broken access control vulnerabilities, you can help prevent unauthorized access and maintain the confidentiality and integrity of your application's data and resources.

### Unprotected admin functionality with unpredictable URL:

The screenshot displays a web application interface. At the top, the header includes the 'WebSecurity Academy' logo, the title 'Unprotected admin functionality with unpredictable URL', and a 'LAB Not solved' status indicator. Below the header, there are links for 'Home' and 'My account'. The main content area features a section titled 'WE LIKE TO SHOP' with a shopping bag icon. Below this, four product cards are displayed, each with an image, title, price, star rating, and a 'View details' button. The products are: 'Caution Sign' (\$8.62), 'Six Pack Beer Belt' (\$83.77), 'Lightbulb Moments' (\$39.37), and 'Sprout More Brain Power' (\$89.44). The browser's address bar at the bottom shows the URL '0a9200b5031ca4f080ae6cda00200017.web-security-academy.net'.

WE LIKE

SHO



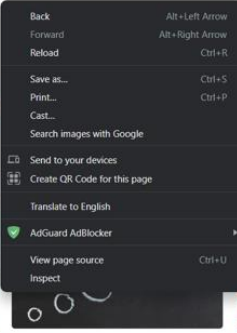
### Caution Sign

★★★★☆ \$8.62

[View details](#)

### Six Pack Beer Belt

★★★★☆ \$83.77

[View details](#)

### Lightbulb Moments

★★★★☆ \$39.37

[View details](#)

### Sprout More Brain Power


★★★★☆ \$89.44

[View details](#)

```

14 <script>
15   <!--unprotected admin functionality via unprotected url-->
16   <a class=link back href="https://portswigger.net/web-security/access-control/lab-unprotected-admin-functionality-with-unpredictable-url">
17     <button>go to lab</button></a>
18   </script>
19   <svg version=1.1 id=layer_1 xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x=0px y=0px viewBox="0 0 28 30" enable-background="new 0 0 28 30" xml:
20     <g>
21       <polygon points="1.4,0 0,1.2 12,0.15 0,28.8 1.4,30 15.1,15"></polygon>
22       <polygon points="14.3,0 12.9,1.2 25.6,15 12.9,28.8 14.3,30 28,15"></polygon>
23     </g>
24   </svg>
25 </a>
26 </div>
27 <div class="widgetcontainer-lab-status is-notsolved">
28   <span>LAB</span>
29   <span>Not solved</span>
30   <span class="lab-status-icon"></span>
31 </div>
32 </div>
33 </section>
34 </div>
35 <div theme="ecommerce">
36   <section class="maincontainer">
37     <div class="container">
38       <header class="navigation-header">
39         <section class="top-links">
40           <a href="/Home"><p></p>
41         </script>
42
43 var isAdmin = false;
44 if (isAdmin) {
45   var topLinksTag = document.getElementsByClassName("top-links")[0];
46   var adminPanelTag = document.createElement('a');
47   adminPanelTag.setAttribute('href' , '/admin-ahdb45');
48   adminPanelTag.innerHTML = 'Admin panel';
49   topLinksTag.append(adminPanelTag);
50   var pTag = document.createElement('p');
51   pTag.innerHTML = '1';
52   topLinksTag.appendChild(pTag);
53 }
54 </script>
55   <a href="/My-account">My account</a><p></p>
56 </section>
57 </header>
58 <header class="notification-header">
59 </header>
60 <section class="ecommerce-pageheader">
61   

```

Academy  [Back to lab description >>](#)

WE LIKE TO

**SHOP** 

WebSecurity Academy

Unprotected admin functionality with unpredictable URL

LAB Not solved

Back to lab description >>

[Home](#) | [My account](#)

Users

wiener - Delete

carlos - Delete

0a9200b5031ca4080a6cda00200017.web-security-academy.net/admin-ahwb45

WebSecurity Academy

Unprotected admin functionality with unpredictable URL

LAB Solved

Back to lab description >>

Congratulations, you solved the lab!

[Share your skills!](#) [Continue learning >>](#)[Home](#) | [My account](#)

User deleted successfully!

Users

wiener - Delete

## Unprotected Admin Functionality:

WebSecurity Academy


Unprotected admin functionality

LAB Solved

Back to lab description >>


[Home](#) | [My account](#)

WE LIKE TO SHOP




Padding Pool Shoes

★ ★ ★ ★ ★ \$71.65




Couple's Umbrella

★ ★ ★ ★ ★ \$40.20



The Giant Enter Key

★ ★ ★ ★ ★ \$69.95



Photobomb Backdrops

★ ★ ★ ★ ★ \$67.54

0a8200fb03fa0d6681fb89d6001700c3.web-security-academy.net/administrator-panel

WebSecurity Academy

Unprotected admin functionality

LAB Not solved

Back to lab description >>

[Home](#) | [My account](#)

Users

wiener - Delete

carlos - Delete

## **SQL INJECTION:**

SQL Injection is a type of security vulnerability that occurs when an attacker can manipulate or inject malicious SQL (Structured Query Language) code into an application's input fields or parameters that interact with a database. This can lead to unauthorized access, data manipulation, data exfiltration, or even complete control over the underlying database.

### **Here's how SQL injection works:**

1. User Input: Many web applications take user input and use it in SQL queries to interact with a database. This input can come from form fields, URL parameters, or other sources.
2. Lack of Validation/Sanitization: If the application doesn't properly validate or sanitize user input before using it in SQL queries, an attacker can manipulate the input to inject their own malicious SQL code.
3. Malicious SQL Code: The attacker can insert SQL code that alters the intended behaviour of the query. For example, they might inject code that always evaluates to true, thereby bypassing authentication, or code that retrieves sensitive data.
4. Query Execution: The manipulated SQL code is executed by the application's database engine. This can lead to unauthorized access, data leakage, data manipulation, or even the ability to execute administrative commands on the database.

For example, consider a login form where a user provides their username and password. If the application doesn't properly validate and sanitize the input, an attacker might input something like:

```
```Username: ' OR '1'='1
```

```
Password: somepassword
```

```The attacker's input gets incorporated into the SQL query used for authentication, resulting in a query like:

```
```sql
```

```
SELECT * FROM users WHERE username = " OR '1'='1' AND password  
= 'somepassword';  
...
```

Because ``1'='1" is always true, the attacker successfully bypasses the authentication mechanism and gains access to the application.

To prevent SQL injection vulnerabilities:

**Parameterized Queries:** Use parameterized queries (prepared statements) instead of directly embedding user input into SQL queries. Parameterization ensures that user input is treated as data, not code.

**Input Validation:** Validate and sanitize user input before using it in SQL queries. Reject or sanitize inputs that contain unexpected characters or patterns.

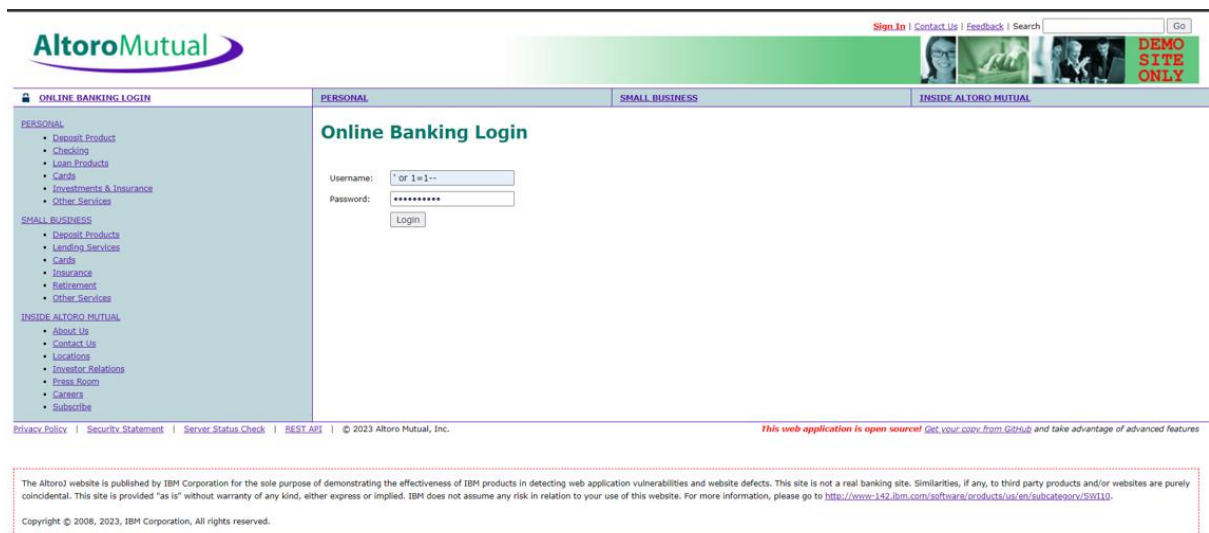
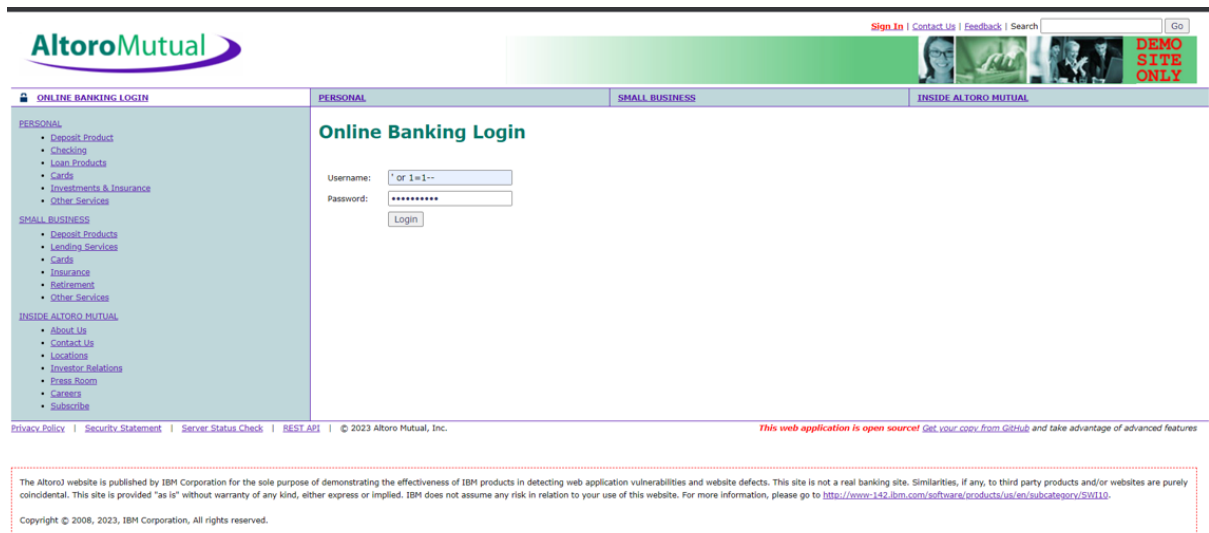
**-Escaping:** If you must include user-generated content in dynamic SQL queries, make sure to escape it properly to neutralize any special characters that might be interpreted as SQL code.

**- Least Privilege:** Limit the database user's permissions to only what's necessary. This reduces the potential impact of an SQL injection attack.

**ORMs and Frameworks:** Use Object-Relational Mapping (ORM) tools and web frameworks that handle SQL interactions for you, as they often have built-in protections against SQL injection.

**- \*\*Regular Security Testing:** \*\* Perform regular security assessments, including penetration testing and code reviews, to identify and address potential SQL injection vulnerabilities.

Preventing SQL injection is crucial for ensuring the security and integrity of your application's data and preventing unauthorized access.





# **CROSS SITE SCRIPTING VULNERABILITY**

## **CHECKING:**

Checking for Cross-Site Scripting (XSS) vulnerabilities is essential to ensure the security of web applications. XSS is a type of security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can execute in the context of the victim's browser, potentially leading to data theft, session hijacking, defacement, or other security risks.

### **Here's a step-by-step guide on how to check for XSS vulnerabilities in your web application:**

#### 1. Manual Code Review:

- Examine your application's source code, especially input fields, and look for instances where user-provided data is inserted into the HTML, JavaScript, or other dynamic content without proper escaping or validation.

Search for common XSS attack vectors, such as ``<script>``, ``onmouseover``, ``javascript:``, and other similar patterns.

#### 2. Input Validation:

- Validate and sanitize all user inputs, including form fields, URL parameters, and cookies.
- Reject or sanitize inputs that contain potentially harmful characters or patterns.

#### 3. Output Encoding:

- Encode user-generated content before inserting it into HTML, JavaScript, or other contexts. Use appropriate encoding functions like HTML entity encoding or JavaScript escaping.

- Libraries like OWASP's Java Encoder or JavaScript's `encodeURIComponent()` can help with this.

#### 4. Content Security Policy (CSP):

- Implement a CSP to restrict which sources of content are allowed to be loaded and executed in the context of your web application.

- Define a strict CSP policy that prevents the execution of inline scripts and limits the sources of executable content.

#### 5. Use Frameworks and Libraries:

- Many modern web frameworks and libraries provide built-in protection against XSS vulnerabilities. Utilize their features for input validation and output encoding.

#### 6. Browser Developer Tools:

- Use browser developer tools to inspect the network traffic and the rendered HTML of your application.

- Look for unusual behaviour or unexpected script execution.

#### 7. Security Testing Tools:

- Utilize automated security testing tools like OWASP ZAP, Burp Suite, or Akinetic to scan your application for potential XSS vulnerabilities.

- These tools can help identify both reflected and stored XSS vulnerabilities.

#### 8. Manual Testing:

- Craft test inputs that contain malicious scripts and attempt to inject them into various parts of your application.

- Observe the behaviour to see if the injected script is executed or blocked.

#### 9. Context-Aware Escaping:

- Be aware of the context in which user-generated content is used (HTML, attributes, JavaScript, etc.), and apply appropriate escaping for each context.

#### 10. Regular Security Audits:

- XSS vulnerabilities can appear as your application evolves. Perform regular security audits and penetration tests to identify and fix new vulnerabilities.

Remember that XSS vulnerabilities can have serious consequences, and it's important to address them promptly. Preventing XSS requires a multi-layered approach involving secure coding practices, input validation, output encoding, and the use of security mechanisms like Content Security Policies.

The screenshot shows the AltoroMutual website with a search bar at the top right. The search bar contains the payload `><script>alert('hacked')<` and a "Go" button. An alert box is visible on the right side of the page, displaying the text `><script>alert('hacked')</script>`. The website layout includes a navigation menu on the left with links for "PERSONAL", "SMALL BUSINESS", and "INSIDE ALTORO MUTUAL". The main content area features sections for "Online Banking with FREE Online Bill Pay", "Real Estate Financing", "Business Credit Cards", "Retirement Solutions", "Privacy and Security", and "Win a Samsung Galaxy S10 smartphone".

The first screenshot shows a search query `><script>alert('hacked')<` in the search bar, resulting in a message that says "testfire.net says hacked". The second screenshot shows a search query `><script>alert('hello + this is a vulnerability')<` in the search bar, resulting in a message that says "testfire.net says hello this is a vulnerability". Both screenshots show the search bar with the payload and the "Go" button.

## **BROKEN AUTHENTICATION:**

According to the OWASP Top 10 ranking, broken authentication is the second most serious vulnerability. An attacker can take over user accounts in a system using this vulnerability. In the worst-case scenario, it may enable them to take total control of the system.

Verifying a user's identity through authentication entails using a username and password combination. When authentication is compromised, it implies that attackers can leverage security holes to go around the authentication process entirely or obtain access to user accounts without authorization.

Here are a few typical instances of weak authentication flaws:

**Weak Password Policies:** Attackers can use methods like brute force or dictionary attacks to guess passwords and obtain unauthorised access if an application permits users to establish weak passwords (short, simple, or obvious ones) or does not enforce password complexity standards.

**Predictable Credentials:** Attackers can quickly guess a user's credentials if a programme produces predictable passwords or utilises default credentials for new users.

**Credential Leakage:** If an application saves passwords insecurely (for example, in plaintext or with shoddy encryption), attackers who discover the passwords can use them to get in without needing a working login and password.

**Issues with session management:** If an application fails to log users out after a certain amount of inactivity or improperly manages user sessions, attackers may take legitimate session tokens or reuse old ones to get unauthorised access.

Developers should adhere to security best practises like the following to avoid broken authentication vulnerabilities:

putting in place strong password standards, such as demanding frequent password changes and mandating password complexity.

using safe password storing techniques, such as salt-based strong hashing methods.

putting in place multi-factor authentication (MFA) to offer an additional security layer.

ensuring secure session management, which includes making use of temporary session tokens, suitable logout procedures, and defence against session fixation threats.

employing methods like penetration testing and code reviews to test the programme often for flaws, such as broken authentication problems.

Unauthorised access to critical systems and user accounts is a severe security risk caused by failed authentication flaws. To safeguard their code, developers must use strong authentication systems and follow the most recent security guidelines.

**Insecure Authentication Flows:** Attackers may be able to influence the authentication process by intercepting requests for authentication or by taking use of logical weaknesses to get around authentication checks.

**Password Reset Vulnerabilities:** If a programme enables hackers to quickly change user account passwords without sufficient validation, they can seize control.

The screenshot displays the AltoroMutual Online Banking Login page. The browser address bar shows the URL `testfire.net/login.jsp`. The page features the AltoroMutual logo at the top left. A navigation menu on the left lists categories: MY ACCOUNT, PERSONAL, SMALL BUSINESS, and INSIDE ALTORO MUTUAL, each with sub-links. The main content area is titled "Online Banking Login" and contains a login form with fields for Username (containing "jsmith@--") and Password (masked with "\*\*\*"), and a "Login" button. The footer includes links to Privacy Policy, Security Statement, Server Status Check, and REST API, along with a copyright notice for Altoro Mutual, Inc. and a statement about the application being open source.



## **SENSITIVE DATA EXPOSURE (DUE TO BROKEN ACCESS CONTROL):**

Sensitive Data Exposure due to Broken Access Control is another critical security vulnerability identified by OWASP in its OWASP Top Ten list. This vulnerability occurs when an application fails to properly enforce access controls, allowing unauthorized users to access sensitive data.

Access control refers to the mechanisms and policies that dictate who is allowed to access what resources within an application. Broken access control vulnerabilities can lead to unauthorized users gaining access to sensitive information, such as personal user data, financial records, proprietary information, and more.

### **Here are some scenarios that can lead to sensitive data exposure due to broken access control:**

1. Inadequate Authorization Checks: If an application does not properly check whether a user has the appropriate permissions to access certain resources, an attacker might be able to bypass these checks and access sensitive data.
2. Direct Object Reference (DOR) Attacks: In some cases, applications use predictable or sequential identifiers to reference resources (like URLs or database keys). If these references are not properly validated, an attacker could manipulate them to access resources they should not have access to.
3. Privilege Escalation: If an application does not correctly validate a user's privileges when they perform actions or access resources, an attacker might be able to elevate their privileges and gain access to sensitive data or perform actions beyond their authorized scope.
4. Horizontal and Vertical Access Control Bypass: Horizontal access control bypass occurs when an attacker gains access to data belonging to other users with similar privileges. Vertical access control bypass occurs when an attacker with lower privileges gains access to data intended for users with higher privileges.
5. Insecure Data APIs: If APIs (Application Programming Interfaces) lack proper access controls, attackers can abuse them to access sensitive data directly.

**To prevent sensitive data exposure due to broken access control, developers should follow these best practices:**

**Implement Proper Authorization:** Ensure that every access to sensitive data is properly authorized and validated. Implement role-based access controls (RBAC) and least privilege principles, where users are granted only the minimum access they need.

**Use Indirect Object References:** Avoid using direct identifiers, such as database keys or sequential IDs, in URLs or API calls. Instead, use indirect references that are mapped to actual resources on the server side.

**Regularly Test Access Controls:** Regularly test your application's access controls using techniques like penetration testing, security code reviews, and automated tools to identify vulnerabilities and weaknesses.

**Apply Principle of Least Privilege:** Limit user access to only the resources and functionality they need to perform their tasks and avoid granting unnecessary privileges.

**Implement Secure APIs:** Ensure that APIs used to access sensitive data are properly authenticated and authorized and validate inputs to prevent unauthorized access.

**Use Strong Session Management:** Protect user sessions with strong authentication and session management to prevent attackers from hijacking legitimate users' sessions.

In conclusion, addressing sensitive data exposure due to broken access control is crucial to maintaining the confidentiality of user data and sensitive information within an application. Developers must implement proper authorization checks and access controls to prevent unauthorized access to sensitive resources.



← → ↻ 🔒 Not secure | testfire.net/bank/main.jsp

Sign Off | Contact Us | Feedback | Search

Go

AltoroMutual

MY ACCOUNT

PERSONAL

SMALL BUSINESS

INSIDE ALTORO MUTUAL

I WANT TO ...

- View Account Summary
- View Recent Transactions
- Transfer Funds
- Search News Articles
- Customize Site Language

Hello John Smith

Welcome to Altoro Mutual Online.

View Account Details: 800002 Savings GO

Congratulations!

You have been pre-approved for an Altoro Gold Visa with a credit limit of \$10000!

Click [here](#) to apply.

Privacy Policy | Security Statement | Server Status Check | REST API | © 2023 Altoro Mutual, Inc.

This web application is open source! Get your copy from GitHub and take advantage of advanced features

The Altoro3 website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to <http://www-142.ibm.com/software/openproducts/us/en/subcategory/SW110>.

Copyright © 2008, 2023, IBM Corporation. All rights reserved.

← → ↻ 🔒 Not secure | testfire.net/bank/showAccount?listAccounts=800002

Sign Off | Contact Us | Feedback | Search

Go

AltoroMutual

MY ACCOUNT

PERSONAL

SMALL BUSINESS

INSIDE ALTORO MUTUAL

I WANT TO ...

- View Account Summary
- View Recent Transactions
- Transfer Funds
- Search News Articles
- Customize Site Language

Account History - 800002 Savings

Balance Detail	
800002 Savings	Select Account
Ending balance as of 8/28/23 5:40 AM	-\$220446287585044070000.00
Available balance	-\$220446287585044070000.00

Date	Description	Amount
2023-08-28	Withdrawal	-\$1000.00
2023-08-28	Withdrawal	-\$1000000.00
2023-08-28	Withdrawal	-\$1000.00
2023-08-28	Withdrawal	-\$1000.00
2023-08-28	Withdrawal	-\$10.00
2023-08-28	Withdrawal	-\$10.00
2023-08-28	Withdrawal	-\$10.00

Account	Date	Description	Amount
1001160140	12/29/2004	Paycheck	1200
1001160140	01/17/2005	Paycheck	1200

← → ↻ ⚠ Not secure | testfire.net/bank/showAccount?listAccounts=800002

Credits

Account	Date	Description	Amount
1001160140	12/29/2004	Paycheck	1200
1001160140	01/12/2005	Paycheck	1200
1001160140	01/29/2005	Paycheck	1200
1001160140	02/12/2005	Paycheck	1200
1001160140	03/01/2005	Paycheck	1200
1001160140	03/15/2005	Paycheck	1200
1001160140	03/29/2005	Paycheck	1200

Debits

Account	Date	Description	Amount
1001160140	01/17/2005	Withdrawal	2.85
1001160140	01/25/2005	Rent	800
1001160140	01/25/2005	Electric Bill	45.25
1001160140	01/25/2005	Heating	29.99
1001160140	01/29/2005	Transfer to Savings	321
1001160140	01/29/2005	Groceries	19.6
1001160140	01/29/2005	Entertainment	22.12

[Privacy Policy](#) | [Security Statement](#) | [Server Status Check](#) | [REST API](#) | © 2023 Altoro Mutual, Inc.

**This web application is open source!** [Get your copy from GitHub](#) and take advantage of advanced feat

The AltoroJ website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to <http://www-142.ibm.com/software/products/us/en/subcategory/SW110>.

Copyright © 2008, 2023, IBM Corporation, All rights reserved.