جامعة القاهرة

Supervised By:

DR.Basheer Abdel Fatah Youssef

TA.Khloud Khaled Attia

Prepared & Implemented By:

Abdelrahman Ashraf ELMahdy	**(20186012)**
Amro Adel Farid	**(20206145)**
Dalia Gamal Abdelhamed	**(20206023)**
George Tadros Emil	**(20206014)**
Mootaz Medhat Ezzat Abdelwahab	**(20206074)**

Academic Year: 2023/2024

Second Semester

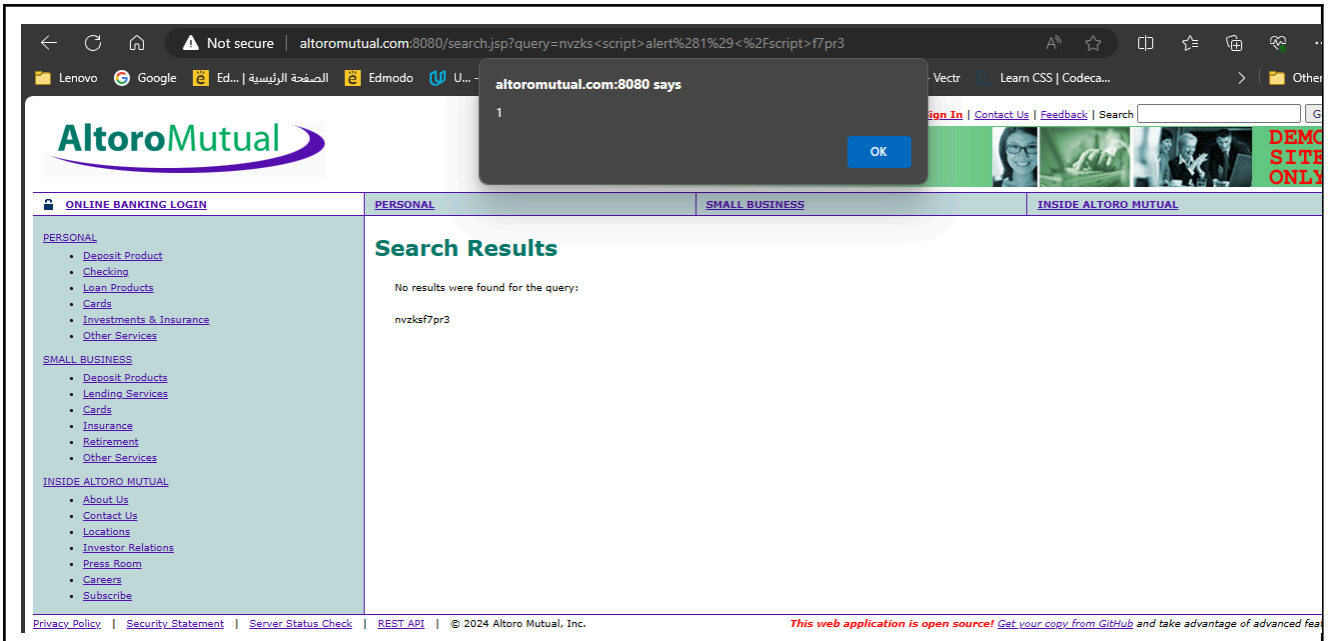
SCS359-Software Security **[Assignment - 2]**

➤ **Findings List with severity, CVSS risk score and risk categorized:**

ID	Name	Original Severity	Snyk score
ID01	Cross Site Scripting **(XSS)** in Search Functionality	High	
ID02	**SQL** Injection in Login	High	
ID03	Cross Site Scripting **(XSS)** in Send Feedback Form	High	
ID04	Cross Site Scripting **(XSS)** queryxpath.jsp	High	
ID05	Sql injection AdminServlet.java	High	
ID06	Cross site scripting in transaction.jsp	High	
ID07	Open redirect disclaimer.htm	High	**558**
ID08	Cross site scripting serverStatusCheck.html	High	**558**
ID09	Open redirect cusomize.jsp	High	**552**
ID10	Improper Neutralization of CRLF Sequences in HTTP Headers LoginServlet.java	High	**552**
ID11	Code injection serverstatuscheck.html	High	**552**
ID12	Use of hardcoded credentials	High	**504**
ID13	Observable Timing Discrepancy (Timing Attack)	High	**502**
ID14	XML External Entity (XXE) Injection	High	**502**
ID15	Trust Boundary Violation surveyservlet,java	High	**402**
ID16	Sensitive Cookie Without 'HttpOnly' Flag loginservlet.java	High	**402**
ID17	Cross-site Scripting (XSS) balance.jsp	High	**839**
ID18	SQL injection in transaction.jsp	High	**829**

➢ Finding details (test and retest):

Name (ID01)	Reflected Cross site scripting (XSS) in Search				
Test Severity	High	Test Score	9.4 / Critical	Retest Severity	
Description:					
Upon testing the search functionality of the application located at http://localhost:8080/altoroj-main/search.jsp, it was discovered that the value of the query request parameter is directly reflected back in the HTML response without proper sanitization. This allows an attacker to inject arbitrary JavaScript code into the application's response, potentially leading to XSS attacks (such as the execution of malicious scripts within the context of other users' sessions).					
Definition of Cross-Site Scripting (XSS):					
Cross-Site Scripting vulnerabilities occur when untrusted data is incorporated into a web application's output in an unsafe manner, allowing an attacker to inject malicious scripts into the application's web pages. These scripts can then be executed in the browsers of other users who view the affected pages, potentially leading to various security threats.					
Impact:					
The impact of this vulnerability is severe. An attacker can exploit this flaw to perform various malicious actions, such as stealing session tokens or login credentials, performing unauthorized actions on behalf of users, or even logging their keystrokes. Depending on the application's functionality and the privileges of the affected users, the consequences could range from compromising sensitive data to complete system takeover.					
Recommendations:	To remediate this vulnerability, the following actions are recommended:				
1. Input Validation: Validate all user-controllable input on arrival, enforcing strict criteria based on the expected content. Reject any input that does not meet the validation criteria, rather than attempting to sanitize it.					
2. HTML Encoding: Encode user input whenever it is included in application responses. Replace all HTML metacharacters, such as <, >, ", ', and =, with their corresponding HTML entities (<, >, etc.).					
3. web application firewalls (WAFs): Implement WAFs to provide an additional layer of defense against XSS attacks by filtering and blocking malicious input.					
Finding Test Steps:					
1. Open the the following page located at http://localhost:8080/altoroj-main/search.jsp					
2. Inject the payload **"nvzks<script>alert(1)</script>f7pr3"** into the query parameter.					
3. Observed that the injected script executed successfully in the application's response, confirming the presence of an XSS vulnerability.					


Not secure
altoromutual.com:8080/search.jsp?query=nvzks<script>alert%281%29<%2Fscript>f7pr3
altoromutual.com:8080 says
1
OK
AltoroMutual
ONLINE BANKING LOGIN
PERSONAL
SMALL BUSINESS
INSIDE ALTORO MUTUAL
Search Results
No results were found for the query:
nvzksf7pr3
Privacy Policy | Security Statement | Server Status Check | REST API | © 2024 Altoro Mutual, Inc.

Name (ID02)	SQL Injection in Login				
Test Severity	High	Test Score	9.4 / Critical	Retest Severity	

Description:

Upon testing the search functionality of the application located at http://localhost:8080/altoroj-main/login.jsp, it was discovered that the value of the query request parameter is directly reflected back in the HTML response without proper sanitization. This allows an attacker to inject arbitrary JavaScript code into the application's response, potentially leading to XSS attacks (such as the execution of malicious scripts within the context of other users' sessions).

Definition of Cross-Site Scripting (XSS):

Cross-Site Scripting vulnerabilities occur when untrusted data is incorporated into a web application's output in an unsafe manner, allowing an attacker to inject malicious scripts into the application's web pages. These scripts can then be executed in the browsers of other users who view the affected pages, potentially leading to various security threats.

Impact:

The impact of this vulnerability is severe. An attacker can exploit this flaw to perform various malicious actions, such as stealing session tokens or login credentials, performing unauthorized actions on behalf of users, or even logging their keystrokes. Depending on the application's functionality and the privileges of the affected users, the consequences could range from compromising sensitive data to complete system takeover.

Recommendations:

To remediate this vulnerability, the following actions are recommended:

1. Input Validation:
Validate all user-controllable input on arrival, enforcing strict criteria based on the expected content. Reject any input that does not meet the validation criteria, rather than attempting to sanitize it.

2. HTML Encoding:
Encode user input whenever it is included in application responses. Replace all HTML metacharacters, such as <, >, ", ', and =, with their corresponding HTML entities (<, >, etc.).

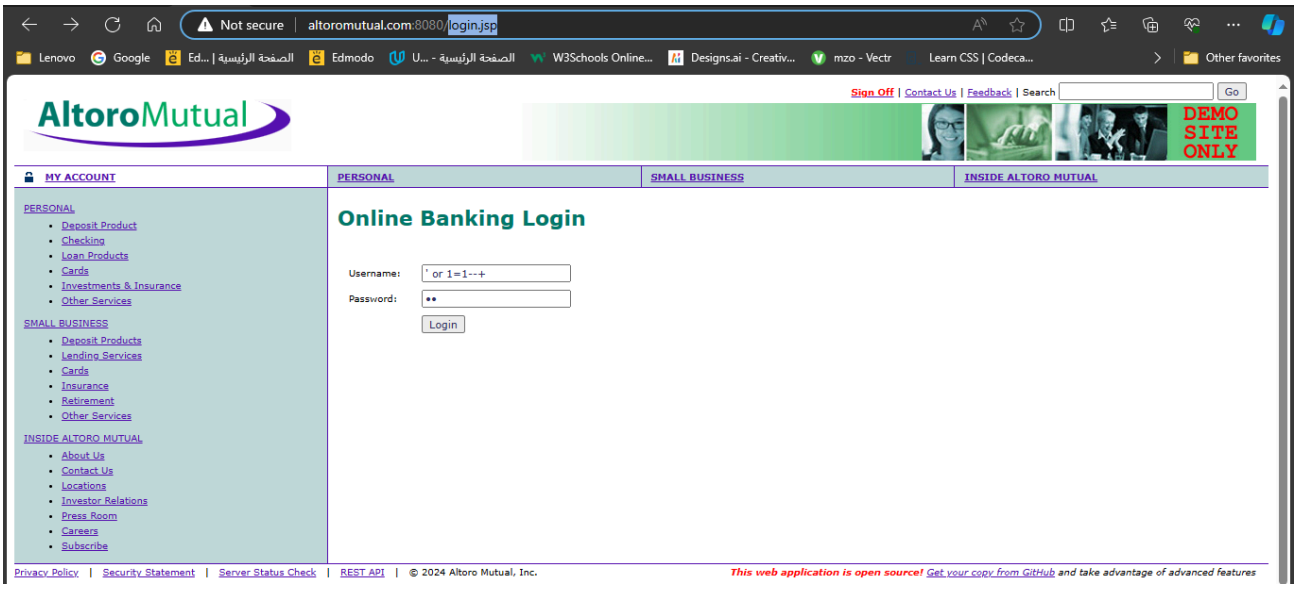
3. web application firewalls (WAFs):
Implement WAFs to provide an additional layer of defense against XSS attacks by filtering and blocking malicious input.

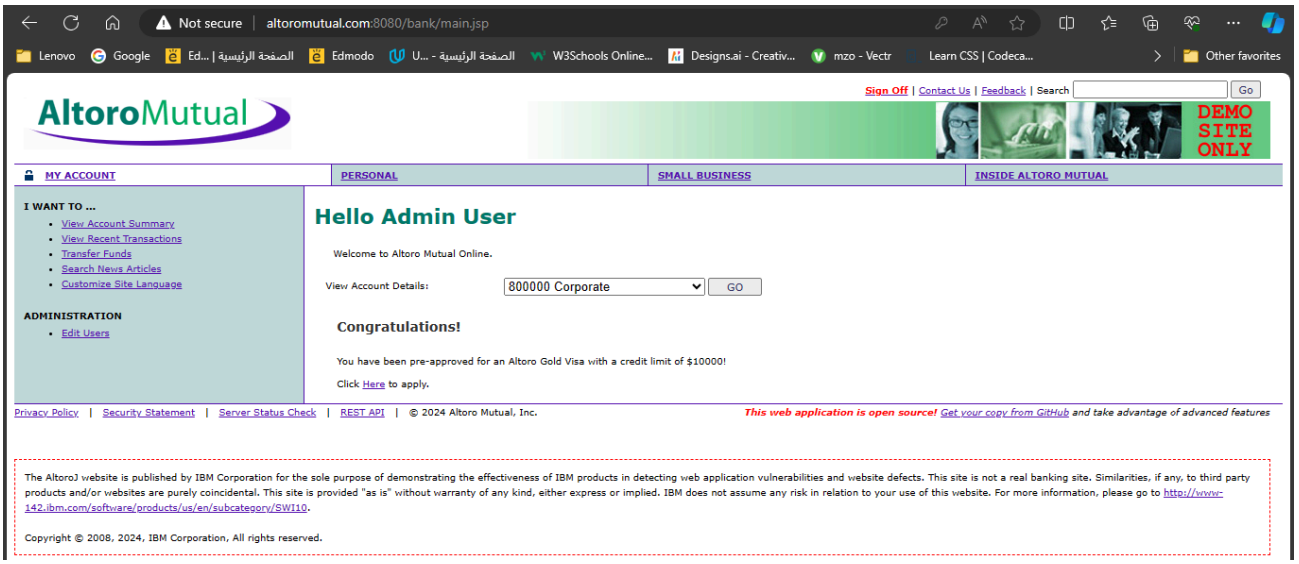
Finding Test Steps:

1. Open the the following page located at http://localhost:8080/altoroj-main/login.jsp

2. Inject the payload **"' or 1=1--+"** into the query parameter.

3. Observed that the injected script executed successfully in the application's response, confirming the presence of an XSS vulnerability.


Not secure | altoromutual.com:8080/login.jsp
AltoroMutual
Sign Off | Contact Us | Feedback | Search Go
DEMO SITE ONLY
MY ACCOUNT
PERSONAL
SMALL BUSINESS
INSIDE ALTORO MUTUAL
Online Banking Login
Username: ' or 1=1--+
Password: ••
Login
Privacy Policy | Security Statement | Server Status Check | REST API | © 2024 Altoro Mutual, Inc.
This web application is open source! Get your copy from GitHub and take advantage of advanced features


Not secure | altoromutual.com:8080/bank/main.jsp
AltoroMutual
Sign Off | Contact Us | Feedback | Search Go
DEMO SITE ONLY
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
ADMINISTRATION
Edit Users
Hello Admin User
Welcome to Altoro Mutual Online.
View Account Details: 800000 Corporate 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 | © 2024 Altoro Mutual, Inc.
This web application is open source! Get your copy from GitHub and take advantage of advanced features
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/SWI10.
Copyright © 2008, 2024, IBM Corporation, All rights reserved.

Name (ID03)	Reflected Cross site scripting (XSS) in Send Feedback Form				
Test Severity	High	Test Score	9.4 / Critical	Retest Severity	

Description:

Upon testing the feedback form functionality of the application located at http://localhost:8080/altoroj-main/sendFeedback, it was observed that the value of the **"name"** request parameter is directly reflected back in the HTML response without proper sanitization. This allows an attacker to inject arbitrary JavaScript code into the application's response, potentially leading to the execution of malicious scripts within the context of other users' sessions.

Definition of Cross-Site Scripting (XSS):

Cross-Site Scripting vulnerabilities occur when untrusted data is incorporated into a web application's output in an unsafe manner, allowing an attacker to inject malicious scripts into the application's web pages. These scripts can then be executed in the browsers of other users who view the affected pages, potentially leading to various security threats.

Impact:

The impact of this vulnerability is severe. An attacker can exploit this flaw to perform various malicious actions, such as**:**

- Theft of sensitive information such as session tokens, cookies, or login credentials.
- Session hijacking, allowing the attacker to impersonate legitimate users.
- Performing unauthorized actions on behalf of the victim user.
- Potentially compromising the security of other applications within the same domain or organization.
- Exploiting users' trust in the application to conduct phishing attacks or distribute malware.

Recommendations: To remediate this vulnerability, the following actions are recommended:

1. Strict Input Validation:
Validate and sanitize all user-controllable input upon arrival, ensuring that it adheres to expected formats and does not contain any malicious content. Reject any input that fails validation rather than attempting to sanitize it.

2. HTML Encoding:
Encode user input whenever it is included in application responses. Replace all HTML metacharacters, such as <, >, ", ', and =, with their corresponding HTML entities (<, >, ", ', &).

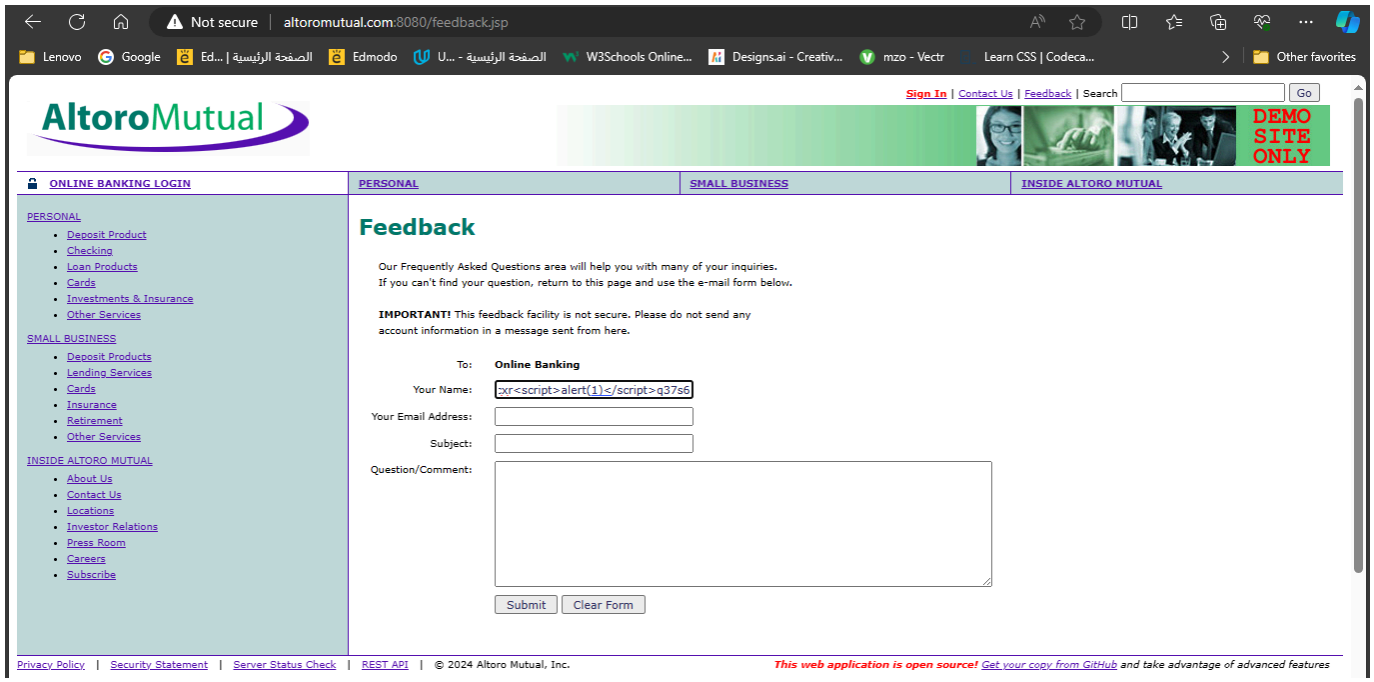
3. Restricted HTML Parsing:
If the application allows users to input HTML content using a restricted subset of tags and attributes (e.g., in blog comments), implement a thorough parsing mechanism to validate that the supplied HTML does not contain any potentially dangerous syntax.

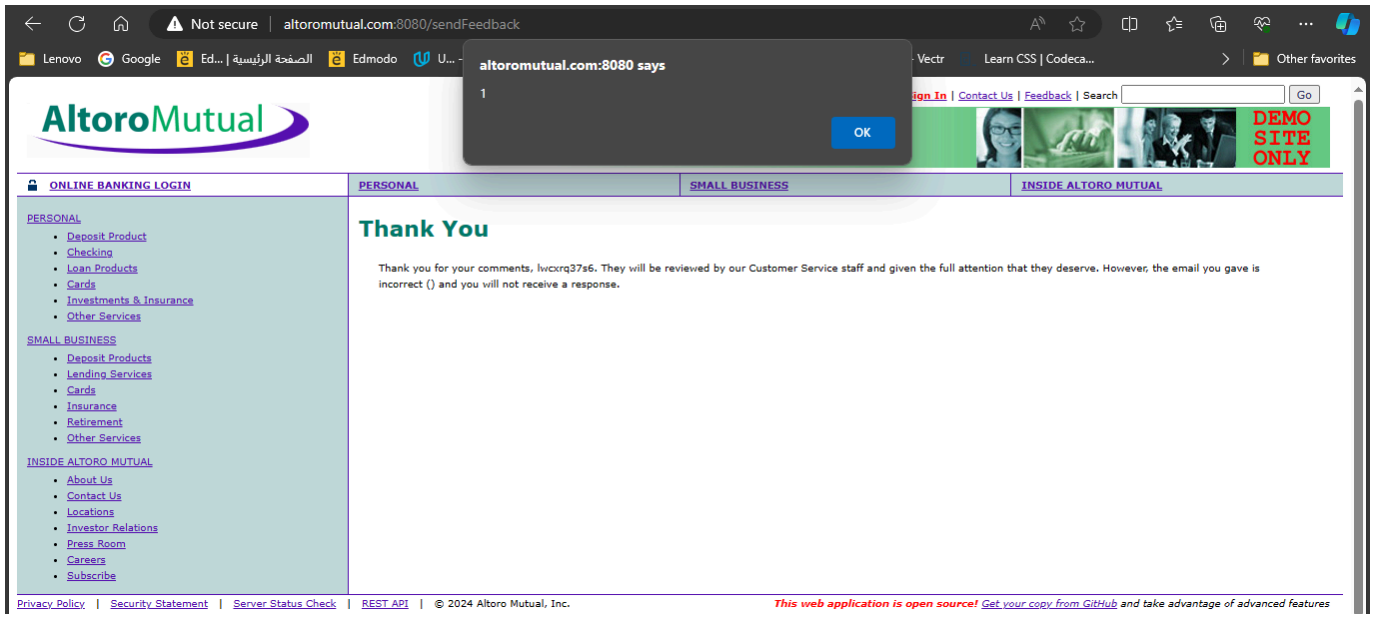
Finding Test Steps:

1. Open the the following page located at http://localhost:8080/altoroj-main/feedback.jsp

2. Inject the payload **"lwcxr<script>alert(1)</script>q37s6"** into the query parameter.

3. Observed that the injected script executed successfully in the application's response, confirming the presence of an XSS vulnerability.





Id05

Unsanitized input from *an HTTP parameter flows* into *execute*, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

Severity high

Id06 Unsanitized input from *an HTTP parameter flows* into *print*, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS).
Severity high

Id07 Unsanitized input from *the document location flows* into *window.location*, where it is used as an URL to redirect the user. This may result in an Open Redirect vulnerability.
Severity high

Id08
Unsanitized input from *data from a remote resource flows* into *eval*, where it is executed as JavaScript code. This may result in a Code Injection vulnerability.
Severity high

Id09
Unsanitized input from *data from a remote resource flows* into *eval*, where it is executed as JavaScript code. This may result in a Code Injection vulnerability.
Severity high

Id10
Unsanitized input from *a database flows* into *addCookie* and reaches an HTTP header returned to the user. This may allow a malicious input that contain CR/LF to split the http response into two responses and the second response to be controlled by the attacker. This may be used to mount a range of attacks such as cross-site scripting or cache poisoning.
Severity high

Id12
Do not hardcode passwords in code. Found hardcoded password used in *Password*.

Severity high

Id13

An attacker can guess the secret value of *password* because it is compared using *equals*, which is vulnerable to timing attacks. Use java.security.MessageDigest.isEqual to compare values securely.
Sserity high

Id14

A file is loaded by *parse*, which allows expansion of external entity references. This may result in an XXE attack leading to the disclosure of confidential data or denial of service.

Id15

Unsanitized input from *an HTTP parameter flows* into *setAttribute* where it is used to modify the HTTP session object. This could result in mixing trusted and untrusted data in the same data structure, thus increasing the likelihood to mistakenly trust unvalidated data.
Medium

Id17

Unsanitized input from *an HTTP parameter flows* into *print*, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS).
Severity high

```
Make all changes
        through the admin page. -->

        <h1>Account History - <%=accountName%></h1>

        <table width="590" border="0">
          <tr>
            <td colspan=2>
              <table cellSpacing="0" cellPadding="1" width="100%" border="1">
                <tr>
                  <th colSpan="2">
                    Balance Detail</th></tr>
                <tr>
                  <th align="left" width="80%" height="26">
                    <form id="Form1" method="get" action="showAccount">
                        <select size="1" name="listAccounts" id="listAccounts">
                            <%
                                for (Account account: accounts){
```

Fix:

<h1>Account History - <%= org.apache.commons.lang.StringEscapeUtils.escapeHtml(accountName) %></h1>

<%@ page import="org.apache.commons.lang.StringEscapeUtils" %>

In this fix, I used StringEscapeUtils.escapeHtml() from Apache Commons Lang library to HTML encode the accountName. This will ensure that any potentially harmful HTML characters in accountName are properly encoded and displayed as plain text in the HTML response, preventing XSS attacks.

Id18 Unsanitized input from an HTTP parameter flows into executeQuery, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

Severity high

```
390
391             if (startDate != null && startDate.length()>0 && endDate != null && endDat
392                 dateString = "DATE BETWEEN '" + startDate + " 00:00:00' AND '" + endDa
393             } else if (startDate != null && startDate.length()>0){
394                 dateString = "DATE > '" + startDate +" 00:00:00'";
395             } else if (endDate != null && endDate.length()>0){
396                 dateString = "DATE < '" + endDate + " 23:59:59'";
397             }
398
399             String query = "SELECT * FROM TRANSACTIONS WHERE (" + acctIds.toString() +
400             ResultSet resultSet = null;
401
402             try {
403                 resultSet = statement.executeQuery(query);
404             } catch (SQLException e){
405                 int errorCode = e.getErrorCode();
406                 if (errorCode == 30000)
407                     throw new SQLException("Date-time query must be in the format of y
408
```