Cyber Defenses

Cyber Defenses: Evaluating Application Vulnerabilities with SQL and XSS

Jai Vang

ITIS 4221-091

Secure Programming and Penetration Testing

October 8th, 2023

VULNERABILITY ASSESSMENT AND SYSTEMS ASSURANCE REPORT

TABLE OF CONTENTS

1.0 General Information

1.1 Purpose

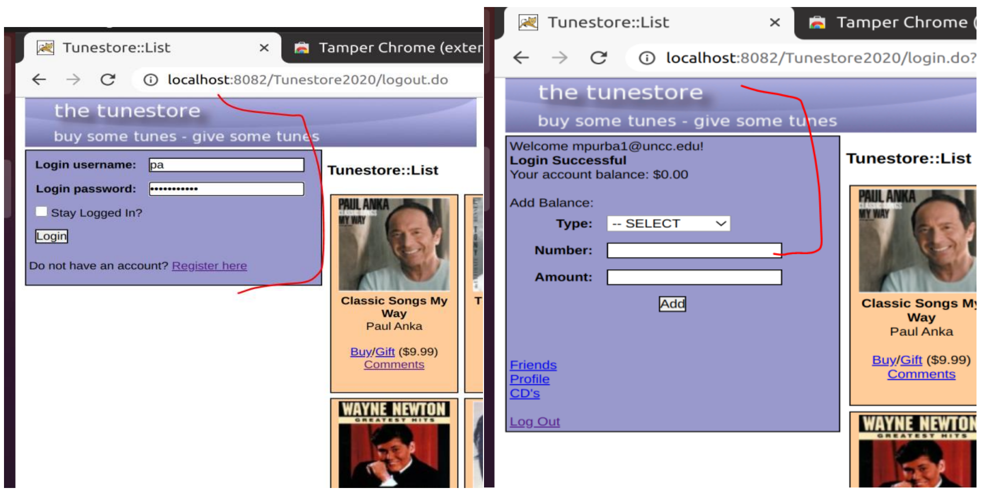
The objective for this assessment is to exploit the existent of vulnerabilities within the application. The overall security of the application is vital and therefore will be tested with SQL injections and cross-site scripting (XSS) to ensure the integrity and security of the application.

2.0 SQL Injection

Tunestore is vulnerable to SWL injection attacks. These attacks occurs when hackers enter SQL statement in the form's input box to access the database server. With SQL injection, the attacker can obtain the credentials of others and impersonate these the users. In Tunestore, SQL injection could be used to login in as a random user or as a specific user.

2.1 SQL Injection – Logging in as a random user

The Tunestore application is poorly designed and there are a lot of vulnerabilities. We will use SQL injections to log in as a random user. What will take place is some input of a random username. This user may or may not exist. Furthermore, some SQL injection will be inputted in the password. An attacker can type a random name like "pa' into the username input. Afterward, they can type ' OR 'a'='a in the password input as seen below and click 'login':


Tunestore::List
Tamper Chrome (exter
localhost:8082/Tunestore2020/logout.do
the tunestore
buy some tunes - give some tunes

Login username: pa
Login password: •••••••••••
Stay Logged In?
Login
Do not have an account? Register here

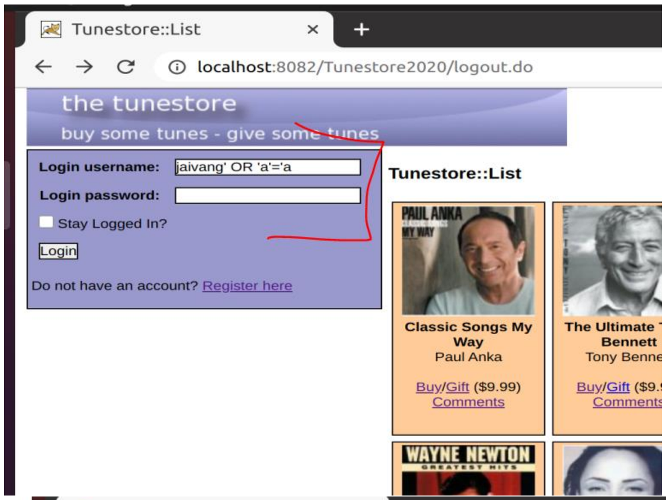
Welcome mpurba1@uncc.edu!
Login Successful
Your account balance: $0.00

Add Balance:
Type: -- SELECT
Number:
Amount:
Add

The image to the right show that the attacker was able to log in successfully without any issue.

2.2 SQL Injection – Logging in as a specific user

An attacker can log in as a random user by typing in the username name of an exist user. I created an account with username as *jaivang* and with a password as *1234*. The hacker can exploit the vulnerable application by typing the known username into the Login Username input follow by ' OR '1' = '1'. All together this is how it look like: jaivang' OR '1'='1'. This will allow access to this specific account. The images below demonstrate this attack:

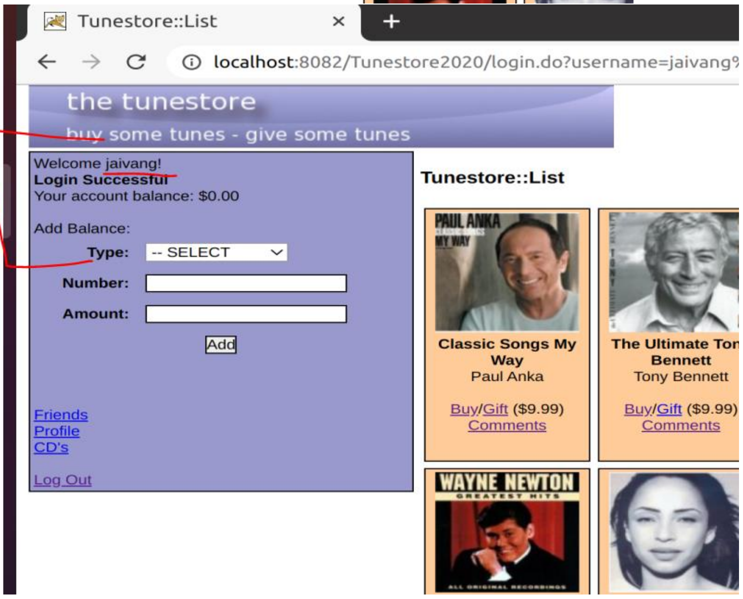

Tunestore::List

localhost:8082/Tunestore2020/logout.do

Login username: jaivang' OR 'a'='a

Login password:

Stay Logged In?


Tunestore::List

localhost:8082/Tunestore2020/login.do?username=jaivang%

Welcome jaivang!
Login Successful
Your account balance: $0.00

Add Balance:

Type: -- SELECT

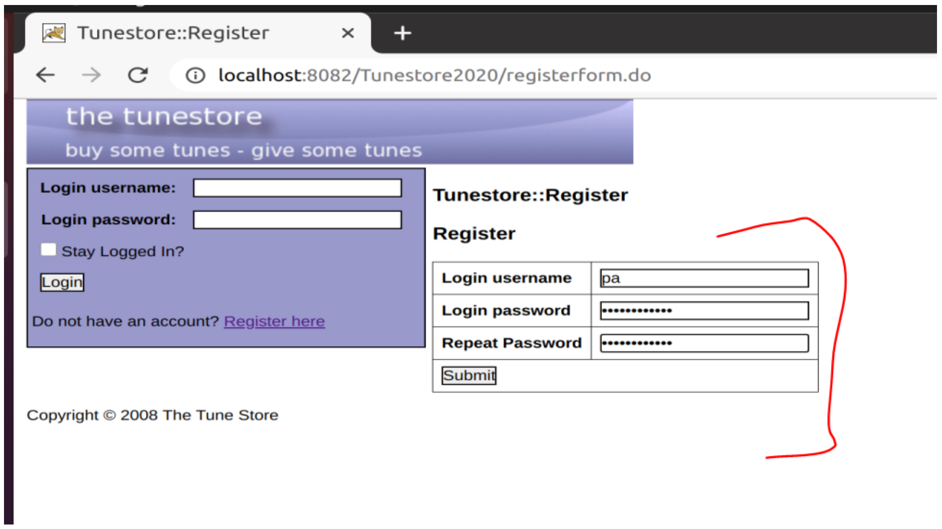
Number:

Amount:

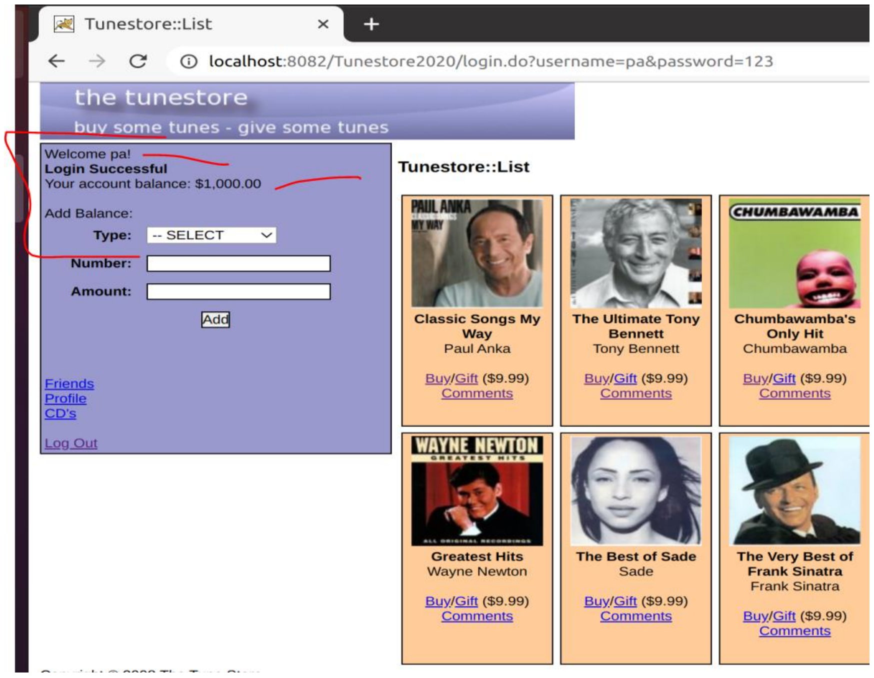
No password is needed since in the database the result will return 'true' because 'a' is always equal to 'a'. Since they are equal, it will automatically accept the username.

2.3 Register a new user with lots money in account without paying for it

An attacker can start by registering a new user with username as *pa* and password as *123',1000)*--. This will create account with an amount of $1,000.00. After creating the account, the attacker will log in again with the username and password.



Login username: pa
Login password: ••••••••••••
Repeat Password: ••••••••••••



Welcome pa!
Login Successful
Your account balance: $1,000.00

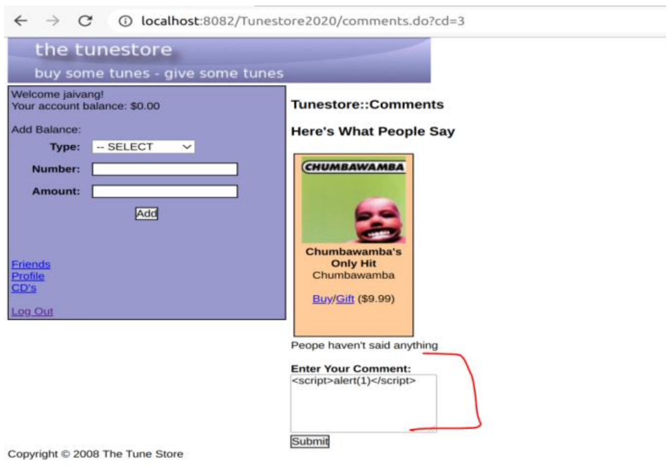
The double dash overrides the database default of $0.00 by commenting it out and the new balance is added. Thus, $1000 is added to the new account.

3.0 XSS Vulnerabilities

Tunestore is vulnerable to cross-site scripting (XSS) since it does not properly sanitize user input. With this, a malicious user can inject JavaScript code into input field. Once another user access the page, they activate the malicious code without any knowledge. Two types of XSS we will look at here is the stored (Persistent) XSS and the reflected (Non-Persistent) XSS. The stored XSS is most critical type of XSS, which occurs when user input is stored on the back-end database and then displayed upon retrieval (e.g., posts or comments). A reflected XSS occurs when user input is displayed on the page after being processed by the backend server, but without being stored (e.g., search result or error message)

3.1 Stored XSS and Reflected XSS

An attacker invoked a stored XSS <script>alter(1)</script> on this comment input after logging in as a user:


localhost:8082/Tunestore2020/comments.do?cd=3

the tunestore
buy some tunes - give some tunes

Welcome jaivang!
Your account balance: $0.00

Add Balance:
Type: -- SELECT
Number:
Amount:
Add

Friends
Profile
CD's

Log Out

Tunestore::Comments

Here's What People Say

Chumbawamba's Only Hit
Chumbawamba
Buy/Gift ($9.99)

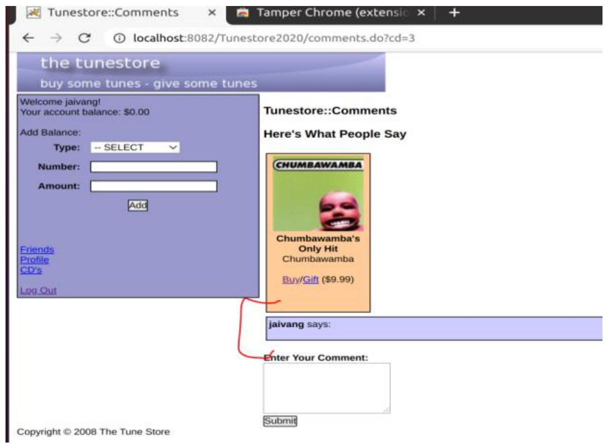
Peope haven't said anything

Enter Your Comment:
<script>alert(1)</script>

Submit

Copyright © 2008 The Tune Store

After clicking 'submit', it saved the comments. The malicious code are saved in the database. As you can see there is no comments:


Tunestore::Comments | Tamper Chrome (extensio

localhost:8082/Tunestore2020/comments.do?cd=3

the tunestore
buy some tunes - give some tunes

Welcome jaivang!
Your account balance: $0.00

Add Balance:
Type: -- SELECT
Number:
Amount:
Add

Friends
Profile
CD's

Log Out

Tunestore::Comments

Here's What People Say

Chumbawamba's Only Hit
Chumbawamba
Buy/Gift ($9.99)

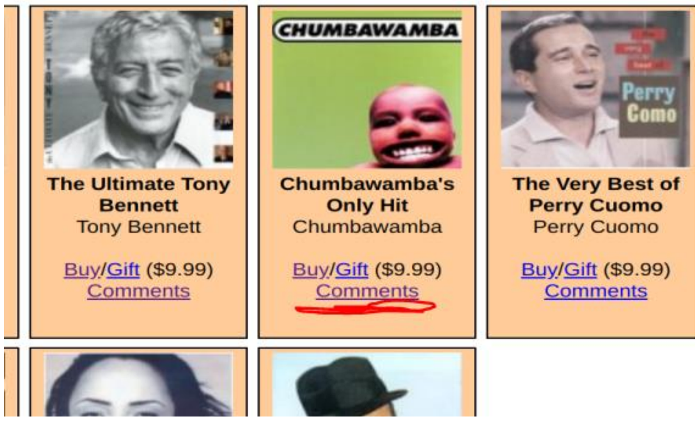
jaivang says:

Enter Your Comment:

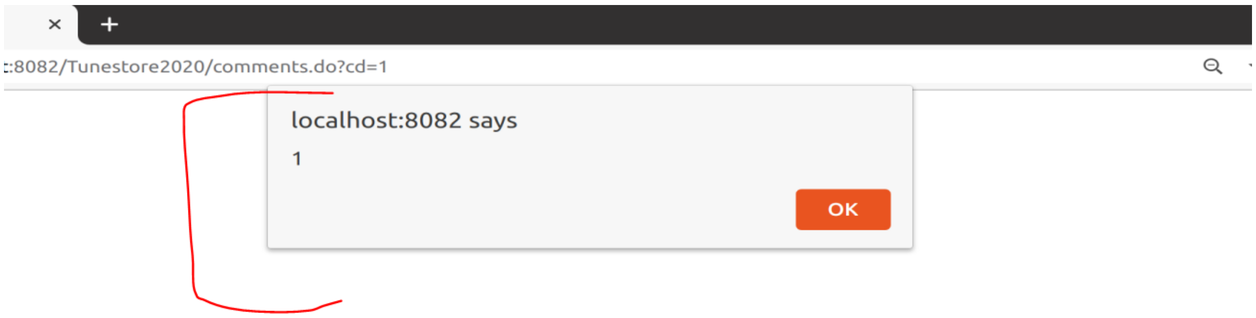
Submit

Copyright © 2008 The Tune Store

Whenever another user click the ‘comment’ URL to see what other users have posted:



The malicious code activate. This appeared:



Dangerous codes can be inputted into the database with another user’s knowledge. The above are examples of stored and reflective XSS.