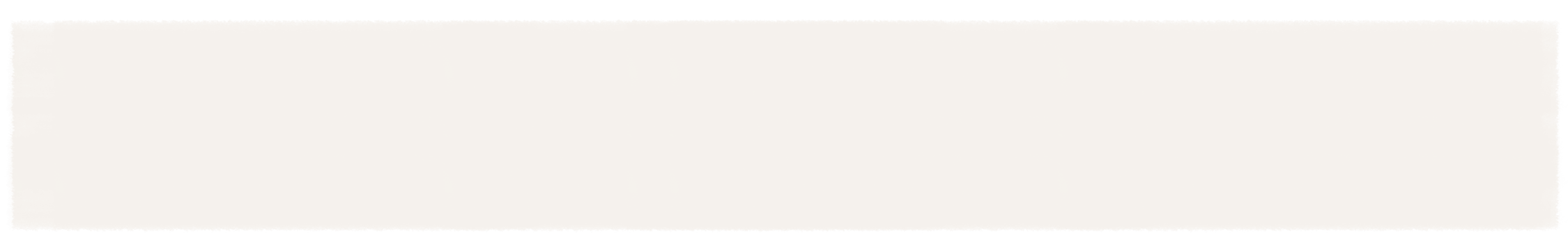
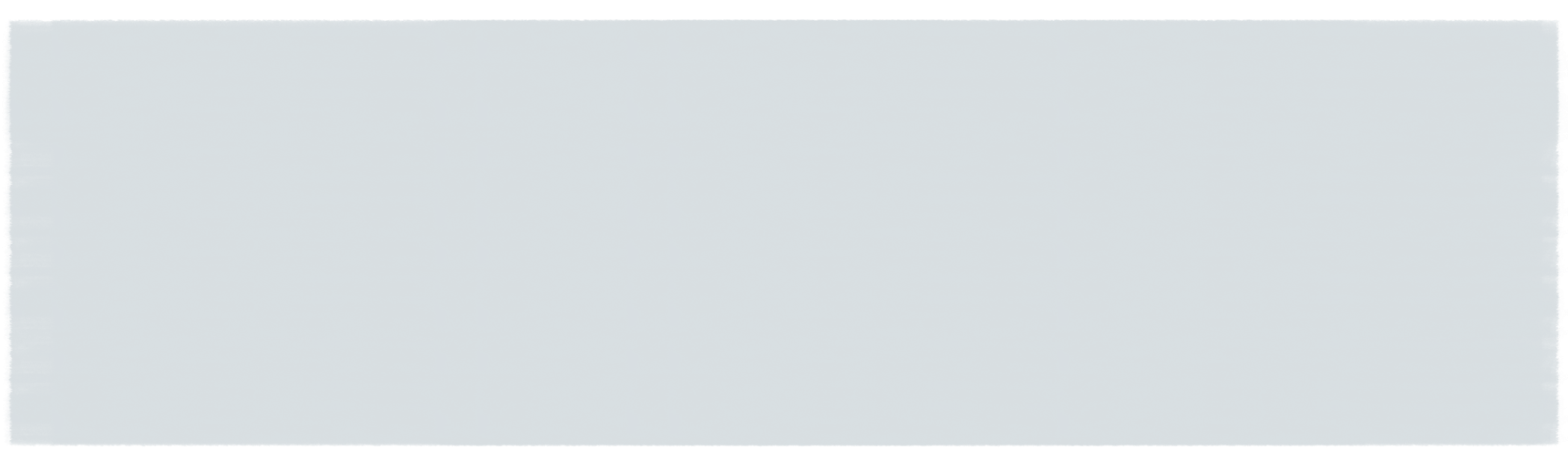
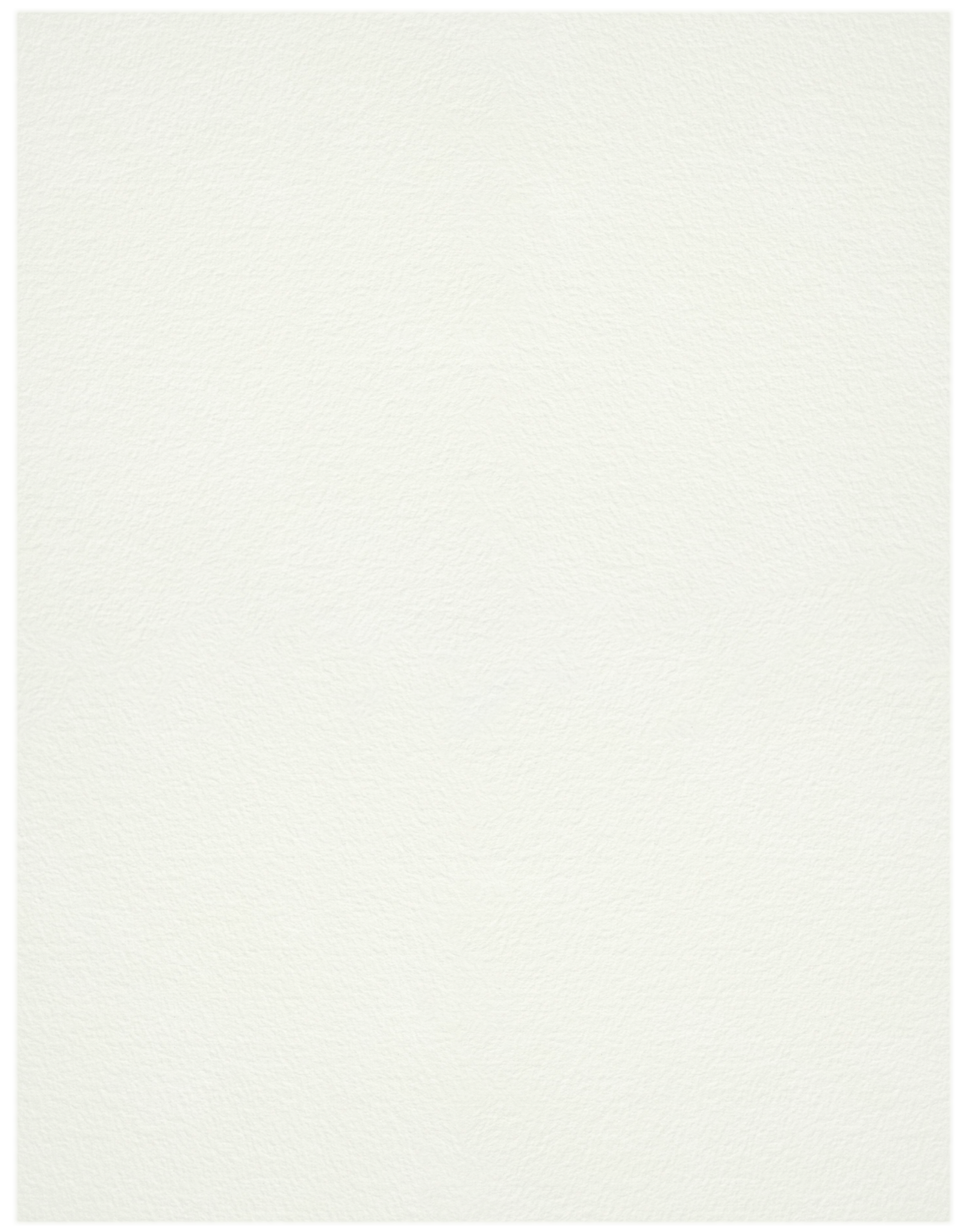
**Jonathan O'Brien – Nidhu Gopalakrishnan – Cathal Cronin**



**10002051 – 14013495 - 10131531**

|  |  |
| --- | --- |
| |  | | --- | | **WEB-BASED APPLICATION DESIGN** | |

**Project 2**

# OWASP Top 10 Defenses - Techniques

## A1: Injection

We used prepared statements in JPQL that are using bind variables for all SQL queries created from view input. The variable data will be escaped appropriately and it would be used as data and not as code.

## A2: Broken Authentication & Session Management

## 

When are new user creates a new account, the supplied password is hashed before it is stored in the database. When a user attempts to log in, the new supplied password is hashed and compared against the hashed password extracted from the database.

Checking if there are a certain length checks passwords for complexity, have uppercase or lowercase characters, symbols. A score is generated based on the latter values. If the score if not good enough the new account is denied from being made.

Running OWASP ZAP against the application showed that our application was vulnerable to URL rewriting. Adding a session timeout and setting the tracking mode to cookie in the “web.xml” file stopped OWASP from detecting this by forcing the session to be tracked in a cookie and with a timeout value.

## A3: Cross-Site Scripting (XSS)

We used OWASP anti-samy to clean input from the user views. When a user enters a username the name is filtered for reflexive XSS attacks from the name being displayed in an error.

Any comments that are added to a product are also cleaned by anti-samy before being stored in the database.

## A7: Missing Function Level Access Control

The main index page has a link to the admin page; this only shows when a user in an administrator role is logged into the system. There is a check done on the user role before that link is generated. Though if a standard user logs in and enters the admin URL they are brought directly to that page, so we have also incorporated this check on each admin based JSP page.

Each time a controller admin function is called via a URL pattern, the user role is checked before executing these functions. If they are not authorized the controllers redirect them automatically.

## A8: Cross-Site Request Forgery (CSRF)

A CSRF attacks are based on a user being logged in. An attacker will send forged HTTP requests to another site with that users session information and cookies as well as other authenticated and sensitive information.

The OWASP organization recommends a number of things to include in your web application to prevent and mitigate against this vulnerability. The first option is to add a hidden HTML field that is sent in the HTTP request and then validating this on the server side. If the token matches the one stored on the server, the request can be carried out. If the token does not match or is missing then the request is not carried out. A similar option for this is for the token to be displayed in the URL but this is riskier since the URL and token are exposed to the attacker.

The final option and the method we used is requiring the user to re-authenticate and prove they are a user. This is done through a CAPTCHA and it protects against a CSRF. Having the user provide this dynamic piece of information every time for a search prevents the CSRF vulnerability from taking place. If no CAPTCHA is entered the request is not processed.

## A10: Unvalidated Redirects

Unvalidated redirects and forwards are quite common for phishing and malware websites. These types of vulnerabilities bring customers to malicious pages that appear to be real and genuine and gain access to sensitive information.

The OWASP organization recommends to simply avoid redirects and forwarding requests. They recommend to not involve and user parameters in calculating the destination as attacker can exploit this. They recommend to always santise the destination URL, process it and validate it to ensure it is a genuine web page for the application. In our web application with our model-view-controller design pattern all URL forwarding and redirects are processed on the server side. The URLs are validated and mapped to the corresponding internal JSP page view that is then displayed for the user. This protects and avoids this vulnerability.

# Testing

We setup a ZAP proxy and captured the initial post request for login to the application and then used this with SQLmap to attempt to attack the server with injection attempts.

***sqlmap -r post.txt***

***http://50.203.224.9:8080/shopebean/login.jsp***

*[01:05:13] [WARNING] POST parameter 'login' is not injectable*

*[01:06:02] [WARNING] POST parameter 'login:username' is not injectable*

*[01:06:47] [WARNING] POST parameter 'login:password' is not injectable*

*[01:07:32] [WARNING] POST parameter 'login:submit' is not injectable*

*[01:08:09] [WARNING] POST parameter 'javax.faces.ViewState' is not injectable*

*[01:08:09] [CRITICAL] all tested parameters appear to be not injectable.*

From index page, captured a search post header and again ran SQLmap against the application.

[01:15:09] [WARNING] POST parameter 'type' is not injectable

[01:14:15] [WARNING] POST parameter 'search' is not injectable

[01:15:09] [CRITICAL] all tested parameters appear to be not injectable.

Also captured post from submitting a comment and then using this to run sqlmap.

[01:21:04] [WARNING] POST parameter 'productId' does not appear dynamic

[01:21:04] [ERROR] possible integer casting detected (e.g. "$productId=intval($\_REQUEST['productId'])") at the back-end web application

[01:23:09] [WARNING] POST parameter 'productId' is not injectable

[01:24:03] [WARNING] POST parameter 'comment' is not injectable

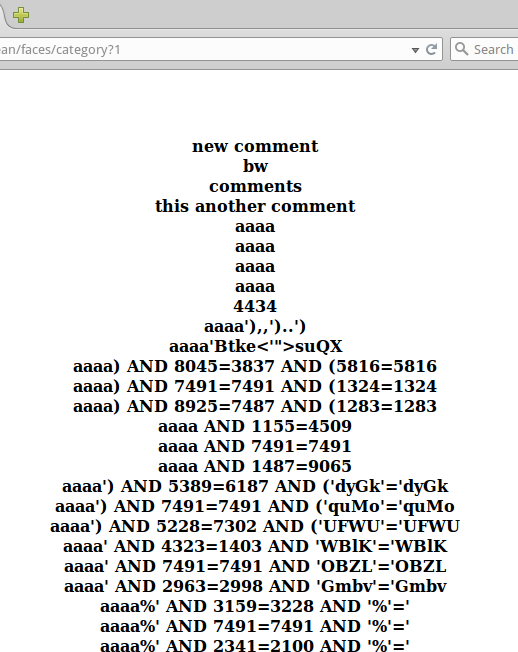
[01:24:55] [WARNING] POST parameter 'submit' is not injectable

***sqlmap -u http://50.203.224.9:8080/shopeBean/faces/category?1=1 --cookie=JSESSIONID=de1a33611af1697284bd5adf540c***

[01:32:07] [WARNING] GET parameter '1' does not appear dynamic

[01:28:13] [WARNING] GET parameter '1' is not injectable

[01:32:51] [WARNING] GET parameter '1' is not injectable



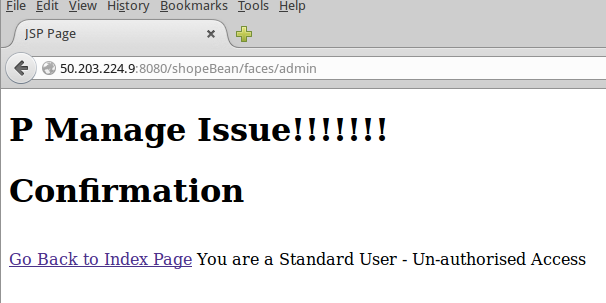
Figure

Below are 2 consecutive session ids captured via ZAP, they do not appear to be predicable.

JSESSIONID=de1a33611af1697284bd5adf540c

JSESSIONID=e0248f418ece20428c87dcbe3504

Below I figure 2, we signed in as a standard user an attempted to navigate directly to the admin pages via entering the URL’s into the browser. The system identified the role of the standard user and denied access.



Figure

Originally with we had reflexive XSS showing for the login page, once we used antisamy on the entered username this was eliminated. Any XSS attempt on the comments for products fails as the attacks are cleaned before being stored in the database.

'';!--"<XSS>=&{()}



Figure

## Code Quality

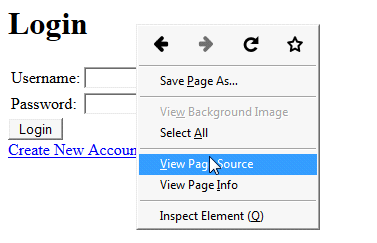
We checked the page source in the login page to check if there were any broken code,comments which were not necessary to be displayed but on examination of the page source page we were unable to find any broken links, unnecessary code or unnecessary comments.

Figure 4-Login page

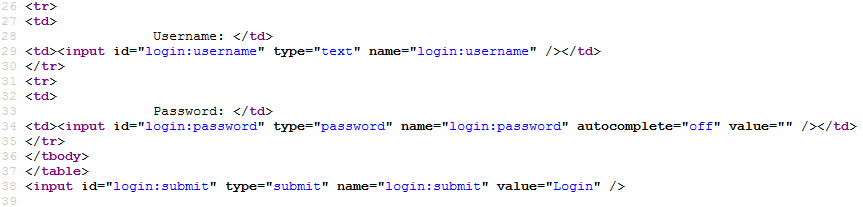


Figure 5-Page source info

We tested our application for any concurrency issues.We first opened a web page app in one tab while having another page of the app opened in another tab and added items to the cart from each of the separate tabs but on clicking the view cart link, the app displayed the synchronized information.



**Figure -Opened in Tab-1**



**Figure -Opened in Tab-2**

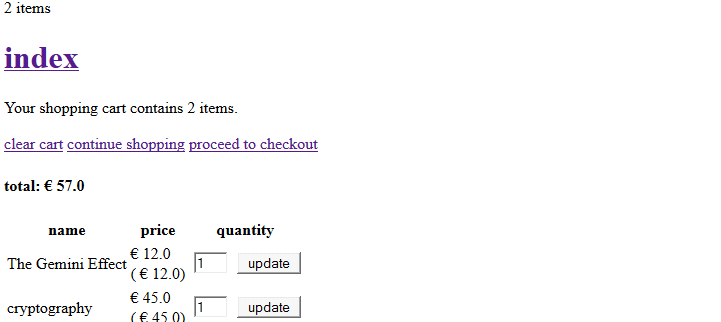


Figure 6-Synchronised display-Tab-1

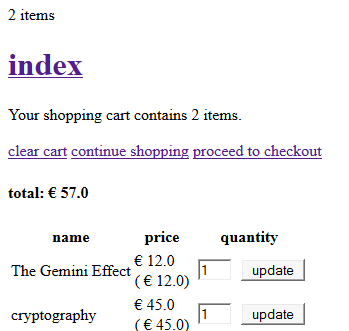
**

Figure 7-Synchronised Display-Tab-2

Next we tried to enter invalid entries in the quantity field to check for any unnatural behavior ,the system however did not accept the invalid entry and it displayed the previous valid entry that was selected.

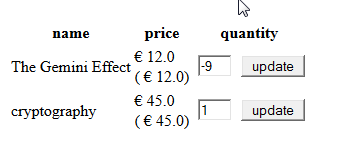


Figure 8-Validation



Figure 9

The quantity field was increased to “3”in the cart page of one tab and the quantity field was not changed and remained at “1” and then the proceed to checkout link was clicked from the tab where the quantity field was not changed, the checkout page however displayed the updated quantity and price and finally the thus the app does not appear to have concurrency vulnerabilities.



Figure 10-Increasing Quantity-Tab-1

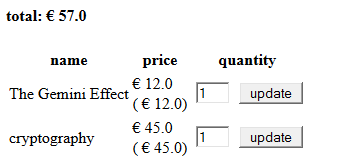


Figure 11-Quantity unchanged-Tab-2



Figure 12-Confirmation

## Exploiting Hidden Fields

We tried to exploit our app through tamper data.On reaching the checkout page tamper data was used to exploit the values entered in the price field but on selecting tamper no fields were displayed to change the price ,thus the app could not be exploited.

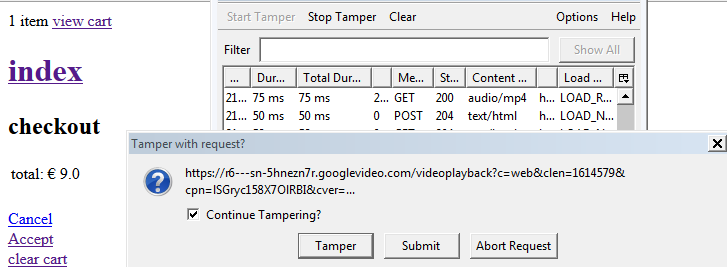


Figure 13-Tamper Data

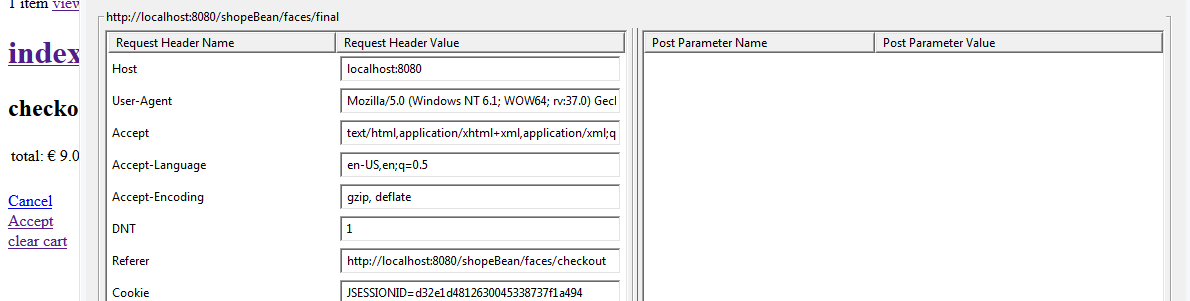


Figure 14-Tamper Data

Next we tried to do inject a script using tamper data,the script was entered into the comments fields and on pressing the ok button the confirmation message was displayed in our web app but the script could not execute as the app did not accept that script to actually run or process on the system.



Figure 15-Tamper comments section

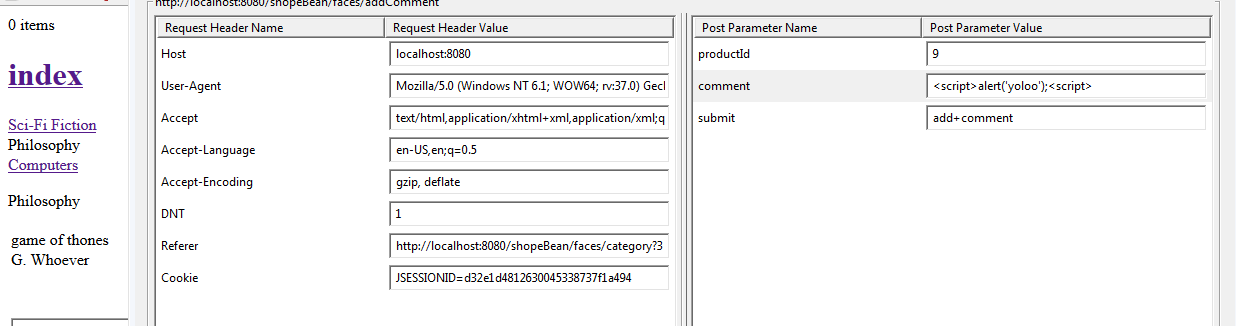


Figure 16-Injecting Script

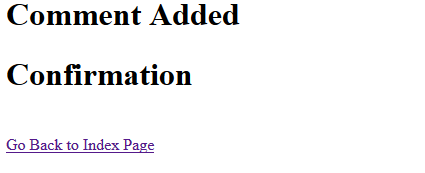


Figure 17-confirmation page



Figure 18-After script injection

## HTML Injection

We tried to do a phising attack from our search page by trying to inject a html script into the search field .however when the submit button was pressed the null exception was displayed.

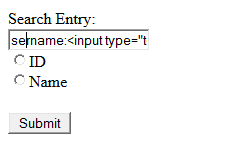


Figure 19-Inserting HTML Script

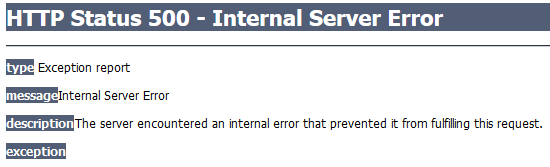


Figure 20-Exception

## Command Injection

We tried to test for any vulnerability in the app on injecting a command in the search field of our app using tamper data.On injecting the command the output that resulted was an unhandled exception with a dump of the stack trace.

****

Figure 21-command Injection-Search Field

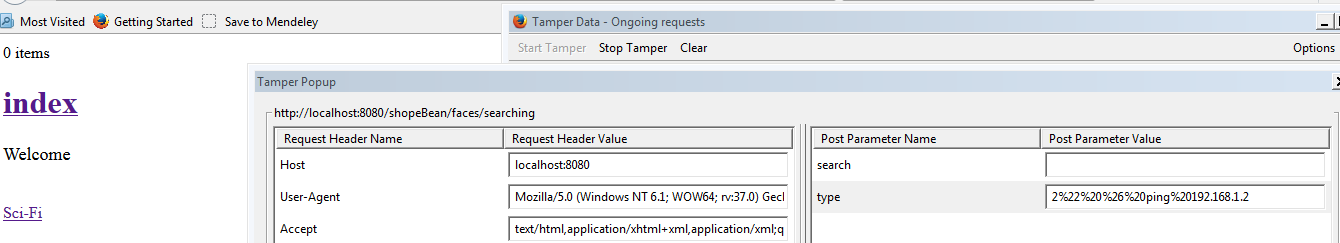
****

Figure 22-Command Injection-2

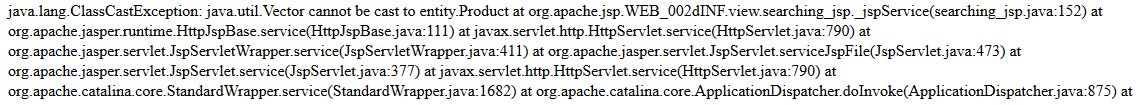


Figure 23-Command Injection-Exception-1

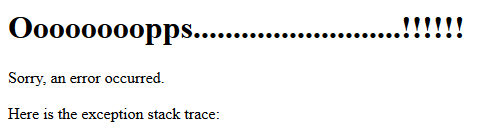


Figure 24-Command Injection-Exception-2

## Improper Error Handling-Login

We tried to exploit any vulnerability in error handling of the login screen.At first ,Loaded up the login screen and started tamper data and tried to delete element by right clicking the password field but was greeted with the proper error message.

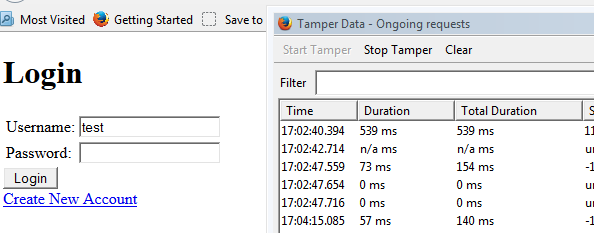


Figure 25-Login screen

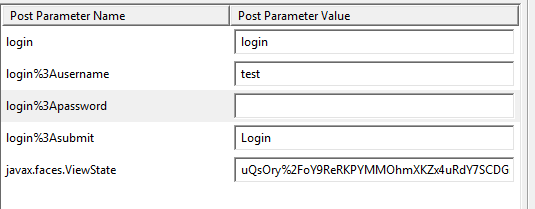


Figure 26-Login Password-Delete Element

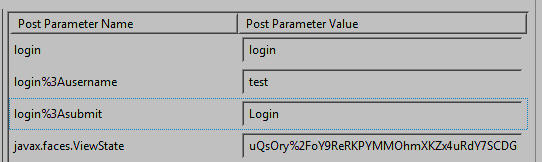


Figure 27-Password Field Deleted

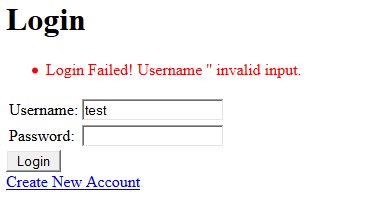


Figure 28-Error handling-Login

## Login-Sql Injection

We tried to do a sql injection in the login page to gain unauthorized access but were unable to gain unauthorized access as the system gave the proper error message thus was unable to exploit the sql injection in our app.

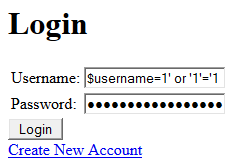


Figure 29-Login-Sql injection

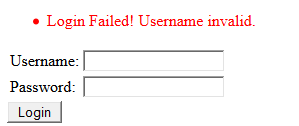


Figure 30-Invalid login

## Multiple Sessions

We tested the app for any vulnerabilities while handling multiple sessions of different users with different roles and we did find a vulnerability when more than one user is logged in. Firstly user1 logged in with normal customer role in tab 1 and user 2 with admin privileges logged in another tab, the issue was that the same session id was assigned to the other session(s) .

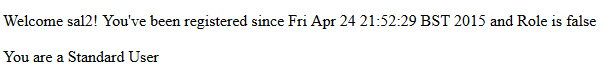


Figure 31-Customer login

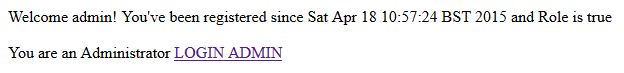


Figure 32-Admin login

But the moment the customer clicked on any of the categories or any other links, the customer session disappeared and was replaced with the session of the admin from the tab-2.

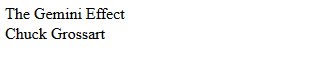


Figure 33-Customer views the product

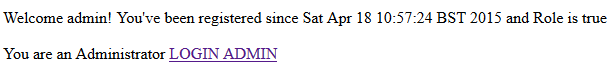


Figure 34-Session changed for Customer

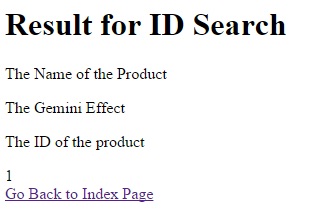
## Cross-Site Request Forgery

Captcha’s are a method of mitigating against these types of attacks. For our testing ensuring a catcha was entered was essential otherwise the POST requests are not being checked. Below you must enter the prodcut you are looking for as well as the generated Captcha.



Figure

If you enter all the information and the captcha is correct then the request is processed and your results are displayed for you if any are present.



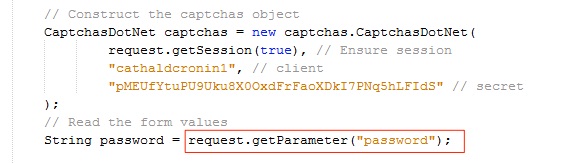
Figure

If however you do not enter the Captcha or it is invalid then the server will not process the search query and display an error page sating an incorrect Captcha.

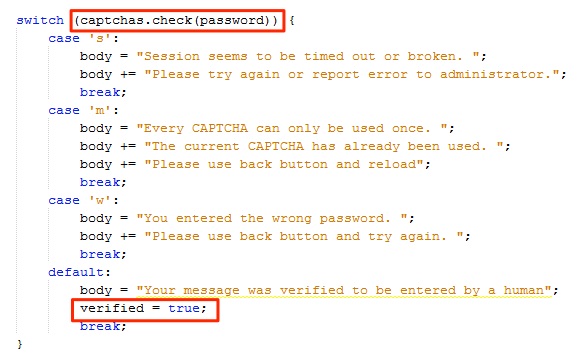


Figure

The code snippet below shows how this works. A Captcha object is generated and it used an ID and a secret session key. These were required as a part of the captcha library we use. The captcha is taken out of the form and then checked if it is valid, verified is set to true. This allows the code to proceed later on and carry out the request.

****

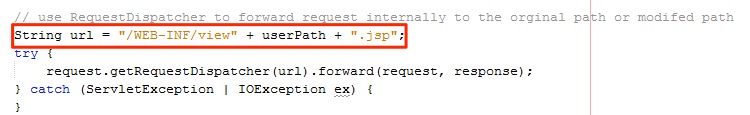
Figure

****

Figure

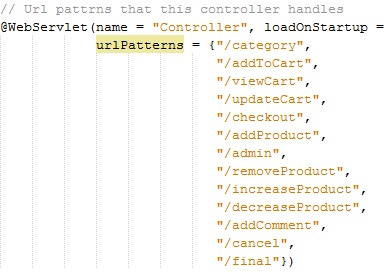
## Unvalidated Redirects and Forwards

All URL forwarding and redirecting occurs through our controller classes. There is a serach controller and general web controller. These controllers are configured to process pre-configured server side URLs. Other wise the request is not processed. For valid Urls the are not taken from the session directly and then forwarded. The URL path is taken and depending on the URL path, the redirect will be mapped to an internal URL that is created on the server side. This prevents against unvalidated redirects and forwards. From the screenshot below you can see the url being built. This is then used in the forward.



Figure

The controller handles a range of preconfigured URLS as seen below.



Figure

Throughout the controller there is a switch statement to handle the functionality that must happen for each of the pre configured URLs

Macintosh HD:private:var:folders:_v:t_d3tgc52ds8dfc6y7w6h7_m0000gn:T:DMD22903B0D-DA66-49BF-9168-1800E0241C17:Windows_7__Running_.jpg

Macintosh HD:private:var:folders:_v:t_d3tgc52ds8dfc6y7w6h7_m0000gn:T:DMD3B21FAFC-DDD8-48E1-92E0-98C6CF4EF3EC:Windows_7__Running_.jpg

Macintosh HD:private:var:folders:_v:t_d3tgc52ds8dfc6y7w6h7_m0000gn:T:DMDD9EC223E-1906-4F02-9B61-A66DF7D19198:Windows_7__Running_.jpg