De La Salle University
College of Computer Studies
Software Technology Department

SECURE DEVELOPMENT

Threat Model Document for the TALARIA FOOTWEAR site

Section	S20
Team Members	Fernandez, Ryan Austin
	Hade, Alden Luc R.
	Poblete, Clarisse Felicia M.
	Syfu, Jonah E.

Date Submitted	August 25, 2016

Table of Contents

1. Security Objectives

The Talaria Footwear Company's online site features tight security against a variety of malicious online attacks. The site implements best practices regarding authentication, authorization, session management, cross-site scripting, SQL injection, data validation, cross site request forgery, clickjacking, and handling e-commerce payments. The following paragraphs represent a list of the security objectives that the Talaria Footwear Company site strives to address.

Authentication must be strictly enforced. Accounts must be secured with a well-formed password immune to rainbow table attacks. Compromised sessions must be invalidated after set amounts of time. Repeated failed attempts at logging in must be diverted by lockout. Authentication must be reinforced for important transactions.

Authorization must be centralized and not be hardcoded. Authorization must be checked for each action. Each user type must only have the least privilege necessary to perform their task.

Session Management must be done strictly through HTTPS. Any attempt to access the site through HTTP must be completely stateless, with no cookies or sessions available. Sessions must be untamperable through URL rewriting or by XSS. A hijacked unauthenticated session must not lead to a hijacked authenticated session. Session ID's should be unpredictable.

Regarding cross site scripting, multiple layers of security must be placed against this. This includes data validation, HTTP headers, and output encoding. The former would also protect against SQL and interpreter injection.

Also, CSRF attacks must be mitigated, and this is done by using hidden tokens. Finally, clickjacking must be mitigated using HTTP headers.

E-Commerce payments must be handled securely i.e. no sensitive information is stored in the database.

2. Application Profile

This section is a profile of the Talaria Footwear Online Portal. This section is divided into four parts, the Data Flow diagram, the ERD Model, and User Roles.

Data Flow Diagram

Figure 2.1. shows the Data Flow diagram of the site.

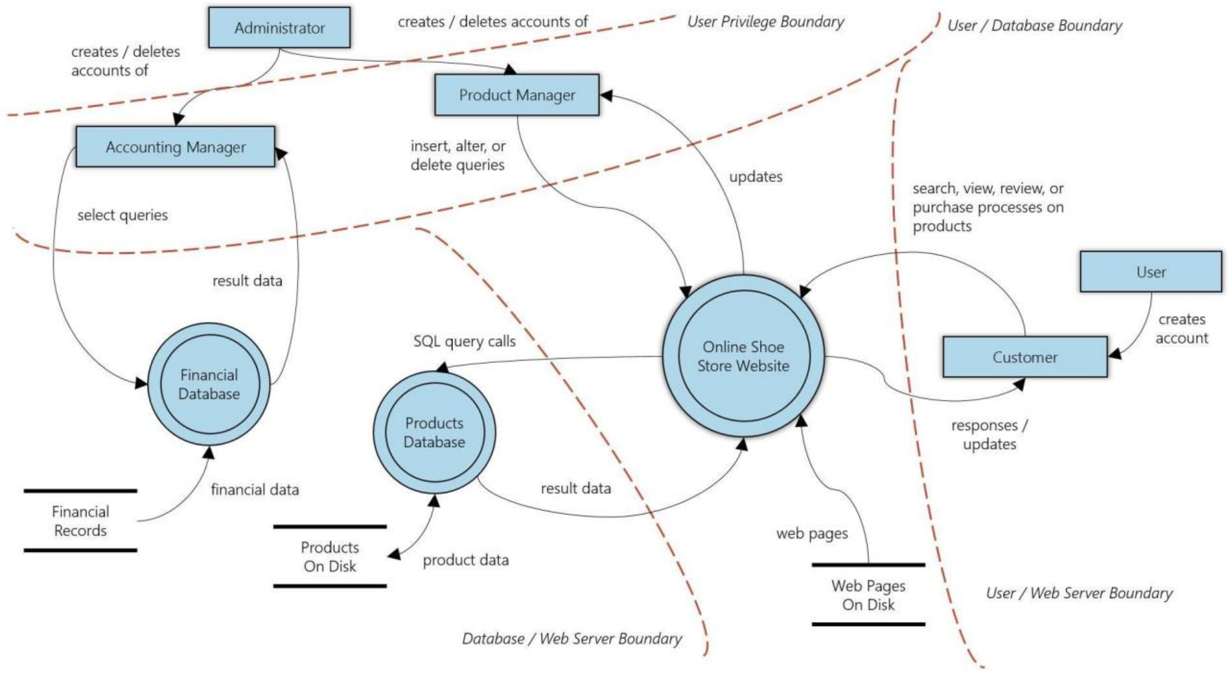


Figure 2.1. Data Flow Diagram of Talaria Footwear online portal.

First, an administrator creates accounting managers and product managers. The product manager then adds, edits, or deletes products on the website, building a set of purchasable items. A user then creates an account and becomes a customer. A customer can now search, view, and purchase a product. After purchasing a particular product, a user can now review the product, which is reflected in the database. The accounting manager can then view the total sales for the entire store, as well as the sales per item type and sales per product.

ERD Model

Figure 2.2. shows the ERD of the Talaria Footwear online portal.

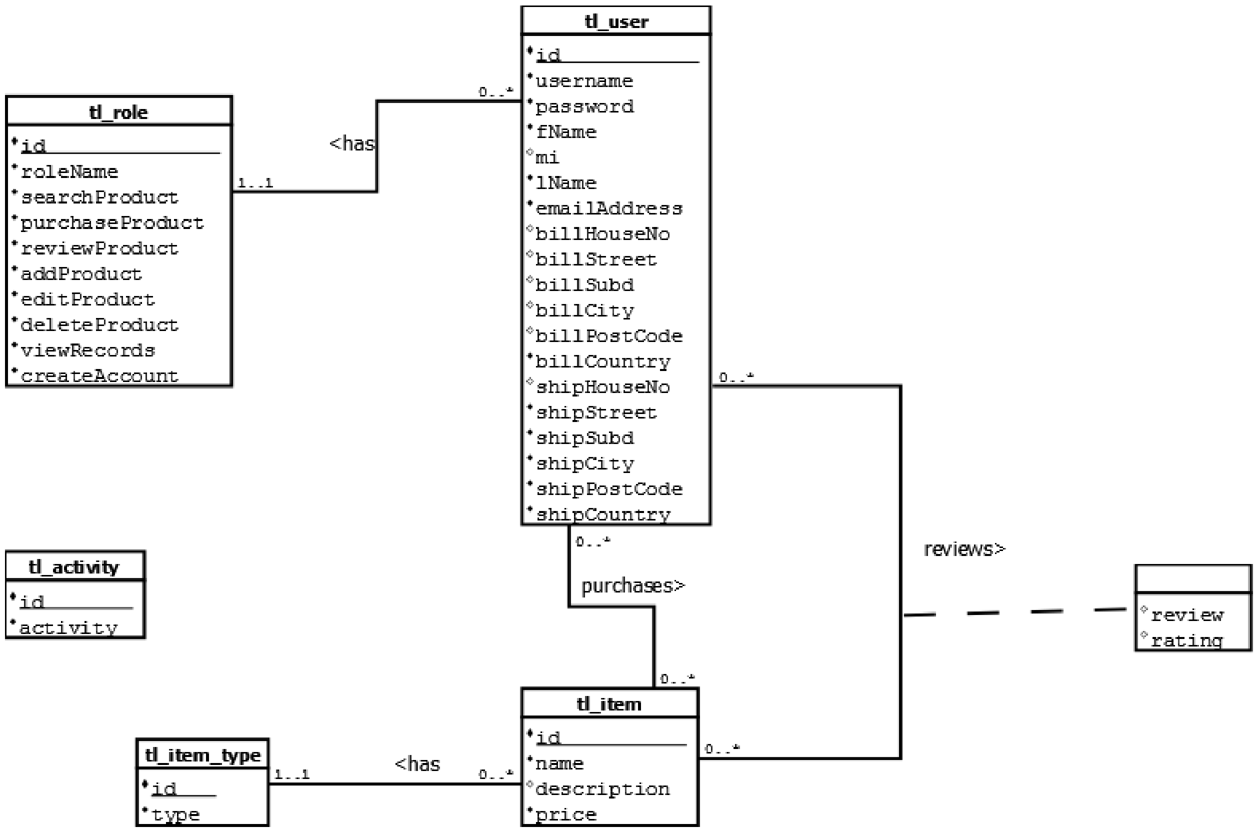


Figure 2.2. Entity-Relationship Diagram of the Talaria Footwear online portal.

Users have a username, password, first name, middle initial, last name, and email address. Customer type users have a billing address (consisting of a house number., street, subdivision, city, post code, country), and shipping address (also consisting of the same fields as the billing address: house number, street, subdivision, city, post code, country).

Items have a name, description, price, and type. Type can be either shoes, sandals, boots, or slippers. Users can buy items then submit a review.

Figure 2.3. shows the relational model of these rules.

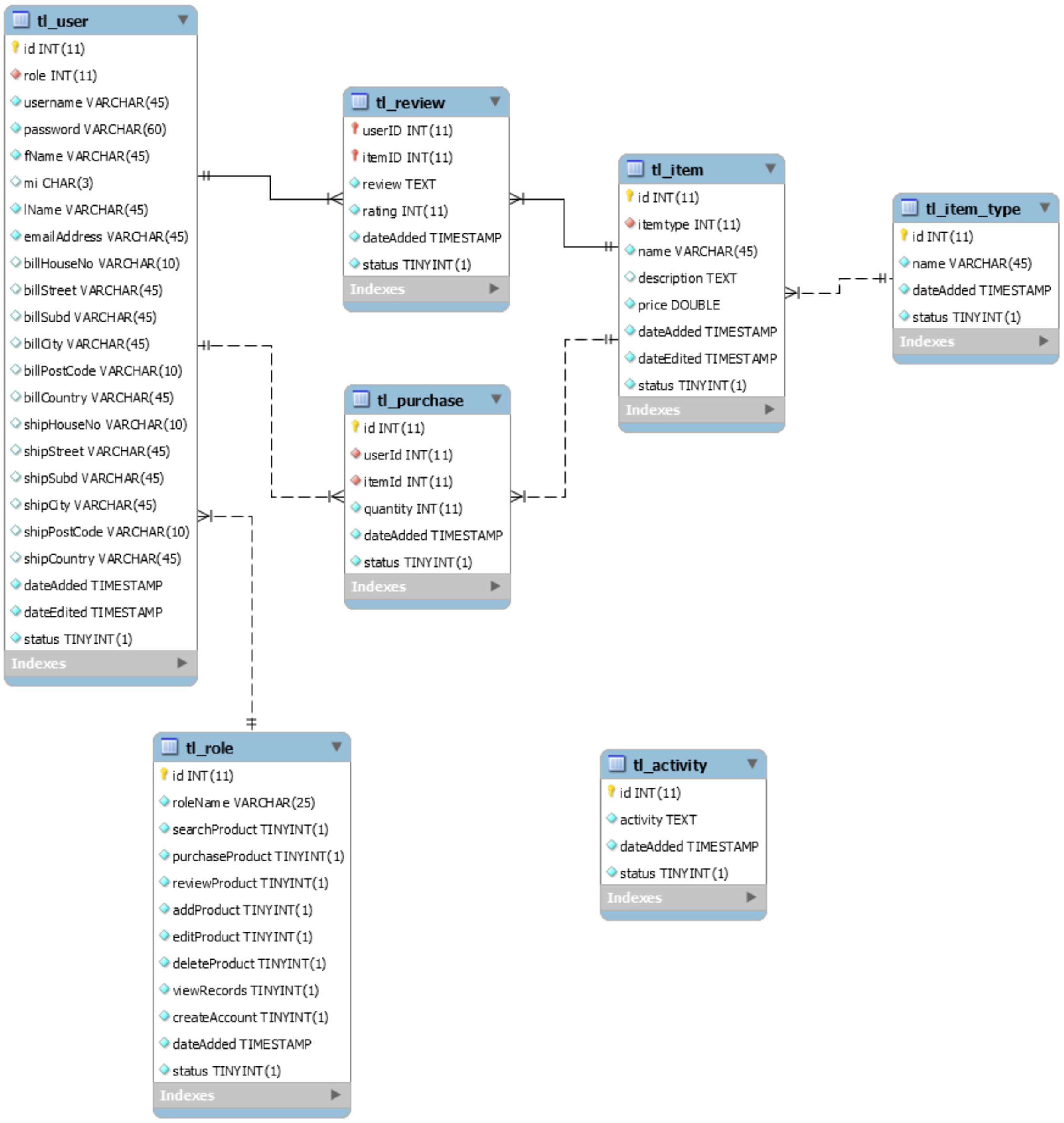


Figure 2.3. Relational Model of the Talaria Footwear online portal.

User Roles

Role	Description of Tasks
Anonymous User	● Search Product
Customer	● Search product ● Purchase Product ● Review product (that they have purchased)
Product Manager	● Search product ● Add Product ● Edit Product ● Delete Product
Financial Manager	● Search product ● View purchase records: total sales, sales per product type, or sales per product.
Admin	● Search product ● Create product manager or financial manager account

3. Threat Documentation

This chapter is divided into two parts: (1) the threat list, which documents each threat in detail; and (2) the DREAD Model, which prioritizes each threat by using the PxI scoring method.

3.1. Threat List

3.1.1. Session Fixation Attack

STRIDE Classification: Spoofing Identity, Information Disclosure

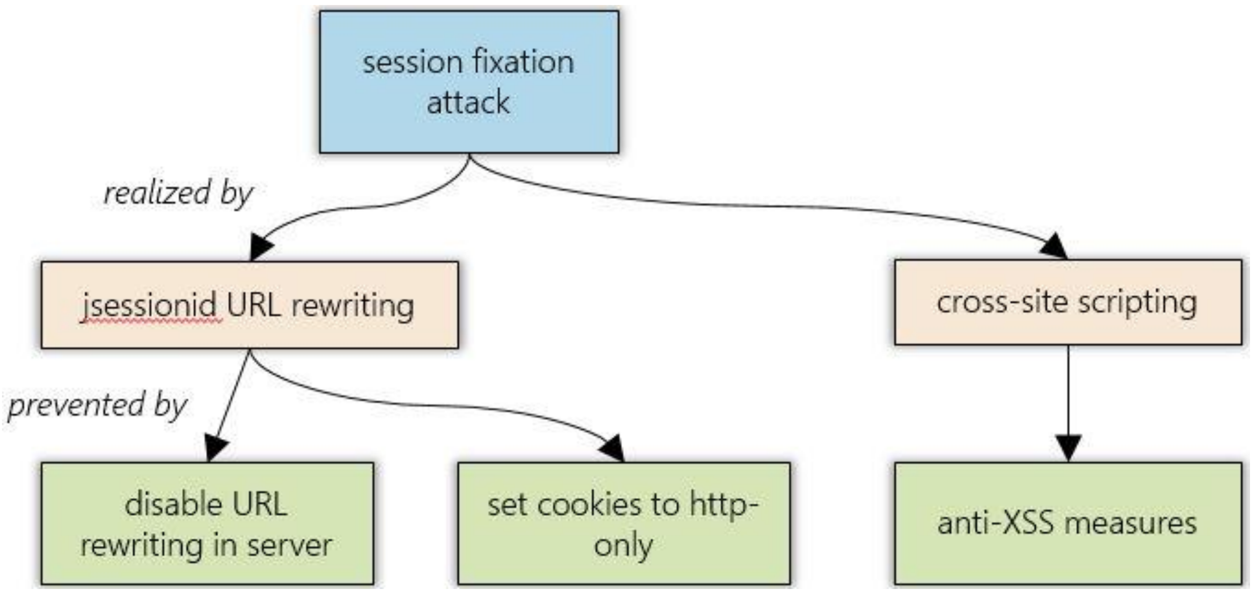


Figure 3.1. Attack graph for a session fixation attack.

Session fixation on a J2EE site would normally done by either URL rewriting, which is rewriting the session ID of the user by putting `;jsessionid=<new session id>` in the URL, through XSS, or by rewriting the cookie using injected JavaScript.

The countermeasure here is to disable URL rewriting on the server. This is done by adding

```
<session-config>
      <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

in web.xml.

As for the XSS, the site is protected by XSS, as will be mentioned later. As an additional layer of defense, cookies are set to http only, which means they can only be changed on the server through response data, and not using Javascript.

For the session cookies, this is done by adding

```
<session-config>
      <cookie-config>
            <http-only>TRUE</http-only>
      </cookie-config>
</session-config>
```

to the web.xml file.

In the other cookies, this is done by calling `Cookie#setHttpOnly(boolean)` and passing `true`.

As another line of defense, just in case a session is compromised, the session has an idle timeout of 10 minutes and an absolute timeout of 30 minutes. This was done by adding

```
<session-config>
      <session-timeout>10</session-timeout>
</session-config>
```

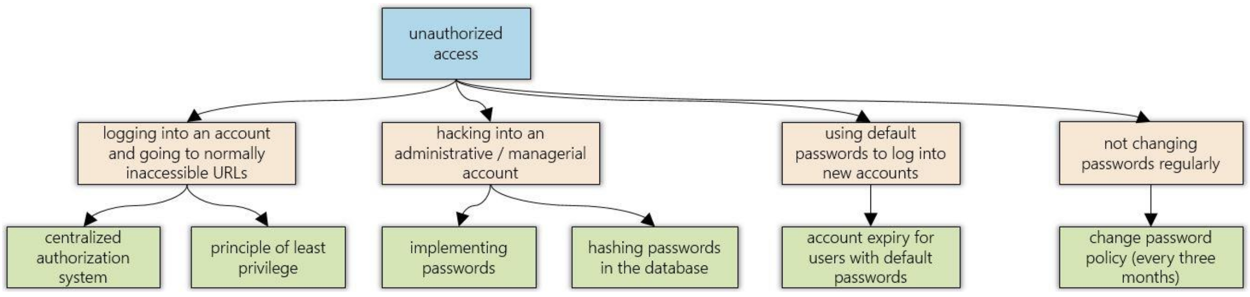
to the web.xml file. The absolute timeout was done every time before authorization is checked.

If an admin account is compromised, adding new accounts requires the user to input their password again. In case the client is not logged in when a session fixation attack occurs, a new session id is generated on login.

3.1.2. Unauthorized Access in

- Purchase Product
- Review Product
- Add Product
- Edit Product
- Delete Product
- View Records
- Create Account

STRIDE Classification: Elevation of Privilege



unauthorized access
logging into an account and going to normally inaccessible URLs
hacking into an administrative / managerial account
using default passwords to log into new accounts
not changing passwords regularly
centralized authorization system
principle of least privilege
implementing passwords
hashing passwords in the database
account expiry for users with default passwords
change password policy (every three months)

Figure 3.2. Attack graph for the threat of unauthorized access.

Unauthorized access can be done by just logging into any account and exploiting the faulty authorization controls (such as by going to specific URLs supposedly accessible only by elevated accounts). It can also be done by logging in as an admin or manager.

Of course, passwords are implemented. Passwords are also hashed in the database to protect user passwords in case of a leak (manually copying the DB file).

The site is well protected because, first, it has a centralized authorization system. Each request passed through an authorization module that checks the database, where the permissions are stored. Also, principle of least privilege is strictly followed. Whereas multiple accounts under the same email are possible, each account can only have one role and therefore, one restricted set of privileges.

Also, for newly created product manager and accounting manager accounts, the account is set to expire after 24 hours if the password is unchanged. After changing passwords, each login pushes back the account's expiry by three months. That way, unused admin accounts have less chance of being compromised since inactive accounts as expired.

In addition, all default accounts used in the testing of the site have been removed, as per OWASP's recommendation.

3.1.3. DDoS Attack in

- Register Page
- Login Page

STRIDE Classification: Denial of Service

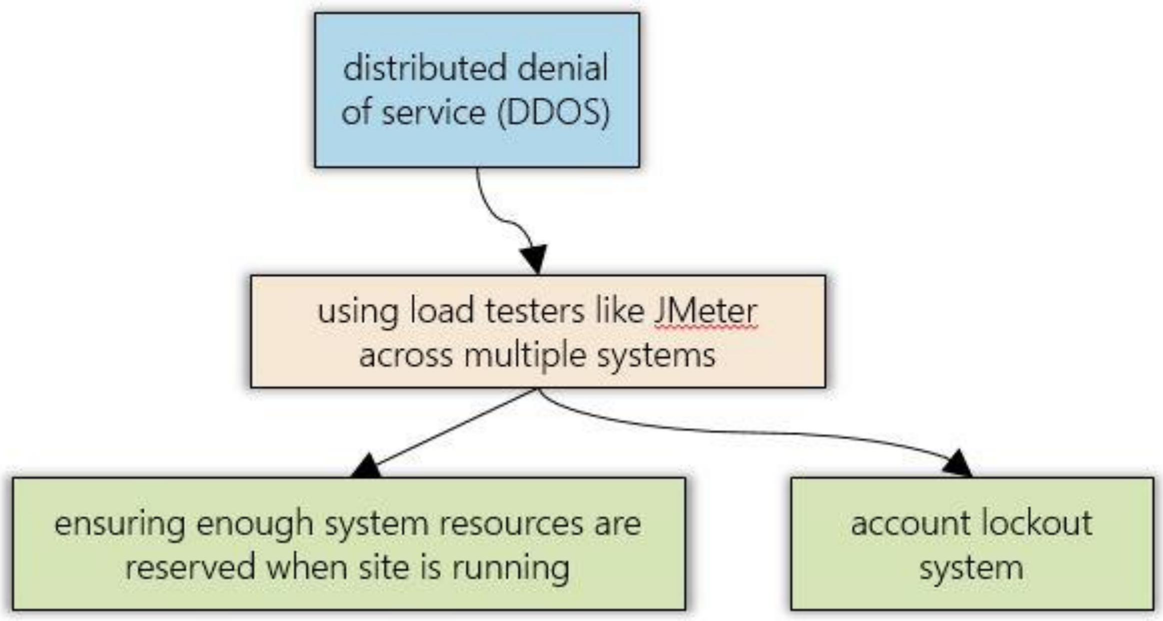

distributed denial of service (DDOS)
using load testers like JMeter across multiple systems
ensuring enough system resources are reserved when site is running
account lockout system

Figure 3.3. Attack graph for a DDOS attack.

This can be exploited by using JMeter on multiple PC's to start multiple threads attempting to login to multiple random accounts simultaneously.

This is countered by issuing a lockout after five failed attempts at login. This was done by adding a loginAttempts field in the user table in the database. Whenever a failed login attempt occurs, this field is incremented by 1. Whenever a user successfully logs in, this counter is reset. Once the loginAttempts field reaches 5, the account is locked out for fifteen minutes using a timestamp stored in the database for unlockTime. Also, when running the site, the server hosting the site must reserve enough system resources for the upkeep of the site, so that it takes much more stress to crash the system.

3.1.4. Faulty Auditing System

STRIDE Classification: Repudiation

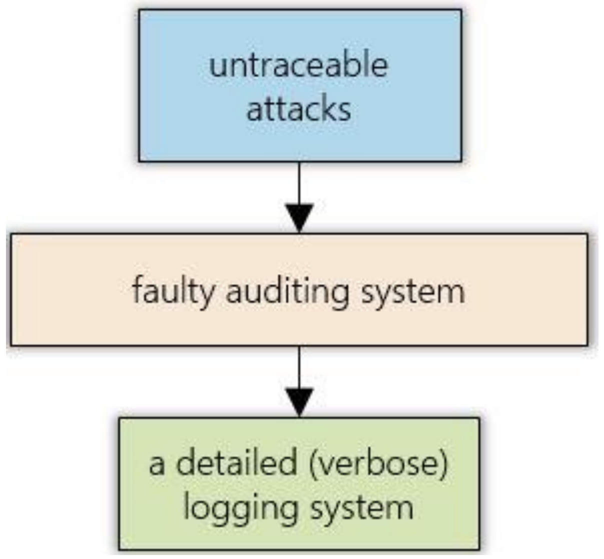


Figure 3.4. Attack graph for attacks that may become untraceable due to not having logging.

Any hacker that can find a vulnerability in this site can perform an attack without detection if the auditing system is faulty.

This is countered by a verbose auditing system, which notes the action, the parameters of the action, the user, the IP address of the user, the timestamp, and any errors the user encountered. All this is stored in a database where the only access permission for adding audits belongs to the controller module. The only way to add an audit is by performing an action on the site, for which only the audit for that action is added.

With a detailed (or verbose) auditing system, administrators can easily detect suspicious behavior, and be able to more easily take action against specific perpetrators of potential attacks against the site. Of course, IP addresses can be spoofed or zombified, but with a detailed auditing system in addition to iron-clad security in other areas, attackers may think twice about attempting more attacks against the system.

3.1.5. Rainbow Table Attack

STRIDE Classification: Spoofing Identity, Information Disclosure

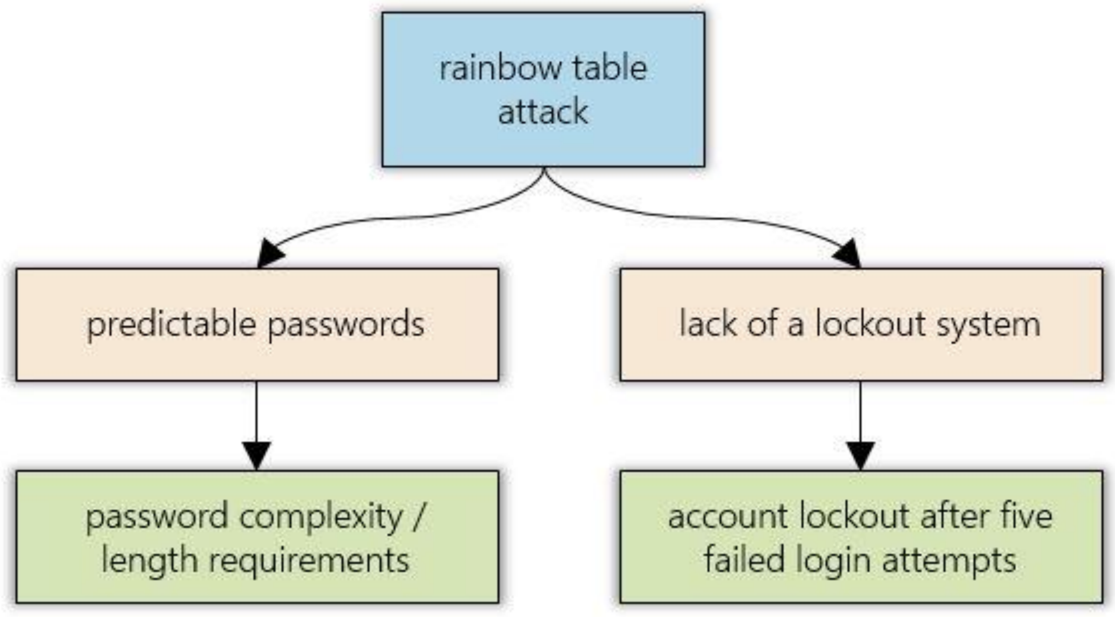

rainbow table attack
predictable passwords
lack of a lockout system
password complexity / length requirements
account lockout after five failed login attempts

Figure 3.5. Attack graph for a rainbow table attack.

A hacker may attempt to brute force the password of a user using a rainbow table attack, by writing a simple automated script to try multiple passwords. The vulnerabilities here would be a predictable password (or a commonly-used password) pattern, as well as a lack of any measures that prevent a user from retrying logging in over and over again.

This is countered in multiple ways. After five failed login attempts, the account is locked out for fifteen minutes, which increases the difficulty for the attack. Also, passwords are required to be complex, i.e. have at least 8 characters, one lowercase letter, one uppercase letter, one number, and one special character, a check that is performed in the view and the controller.

3.1.6. Session Hijacking

STRIDE Classification: Spoofing Identity

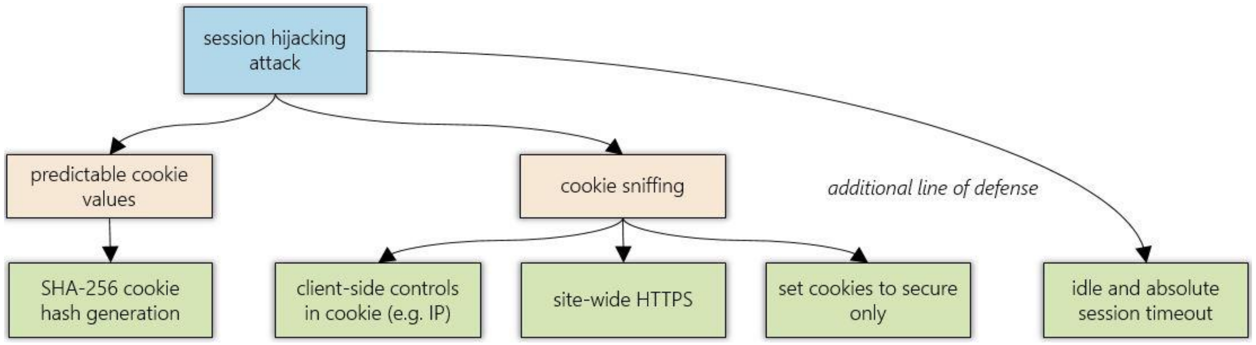

session hijacking attack
predictable cookie values
cookie sniffing
additional line of defense
SHA-256 cookie hash generation
client-side controls in cookie (e.g. IP)
site-wide HTTPS
set cookies to secure only
idle and absolute session timeout

Figure 3.6. Attack graph for session hijacking.

The application stores a cookie on the client side that allows sessions to persist for up to 30 minutes after closing the browser or restarting the machine. An attacker can simply predict this cookie's value to hijack the session. A client's session can also be sniffed using Firesheep or some other third party software.

This is countered by having client side controls in the custom session cookie. This includes the user id, user name, and the client's IP address. If an attacker can successfully guess or steal a user's cookie, unless they're on the same machine, the cookie is invalid. This cookie is generated using a SHA256 hash of all the components so it has enough complexity to prevent brute force attacks.

To prevent sniffing of cookies, the entire site is forced to HTTPS. This is done by configuring Tomcat to run with SSL/TLS encryption.

Session cookies and all other cookies are also set to secure. This is done by adding

```
<session-config>
    <cookie-config>
        <secure-only>TRUE</secure-only>
    </cookie-config>
</session-config>
```

to the web.xml file.

For other cookies, Cookie#setSecure(boolean) is called, passing true. This means that if the site is run through an HTTP protocol and not an HTTPS protocol, it is completely stateless and unusable.

As another line of defense, just in case a session is compromised, as mentioned in the session fixation section, the session has an idle timeout of 10 minutes and an absolute timeout of 30 minutes. If an admin account is compromised, adding new accounts requires the user to input their password again. In case the client is not logged in when a session hijacking attack occurs, a new session id is generated on login.

3.1.7. SQL Injection in

- Login Page
- Search Page
- Purchase Product URL
- Register Page
- Add Product Page
- Edit Product Page
- Delete Product Page
- Create Special Account

STRIDE Classification: Tampering of Data, Information Disclosure

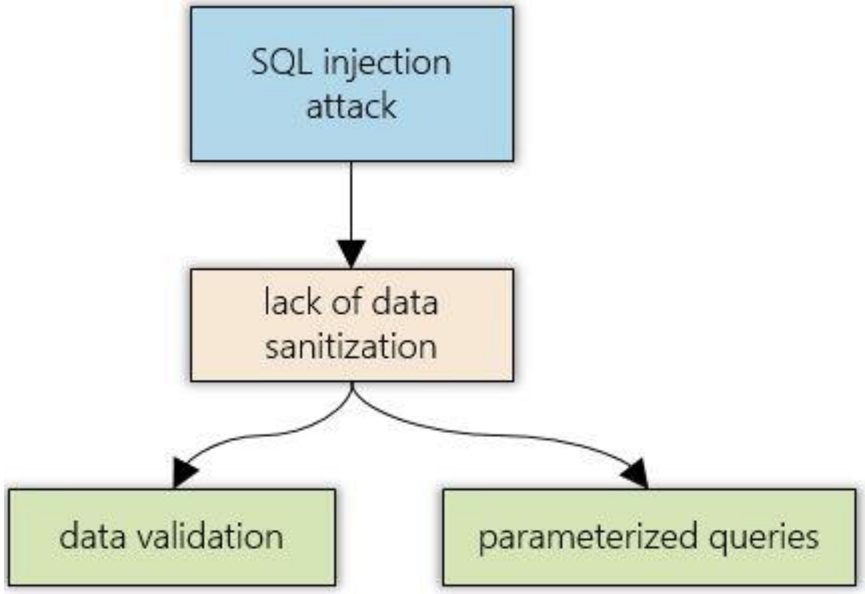


Figure 3.7. Attack graph for an SQL injection attack.

Data can be disclosed, changed, or even deleted if SQL Injection is done on the fields.

This is prevented by two layers of security. Data validation is performed on each field via Javascript and in the Controller. Parameterized queries are used in the model.

3.1.8. Mixed Content Attack

STRIDE Classification: Tampering of Data

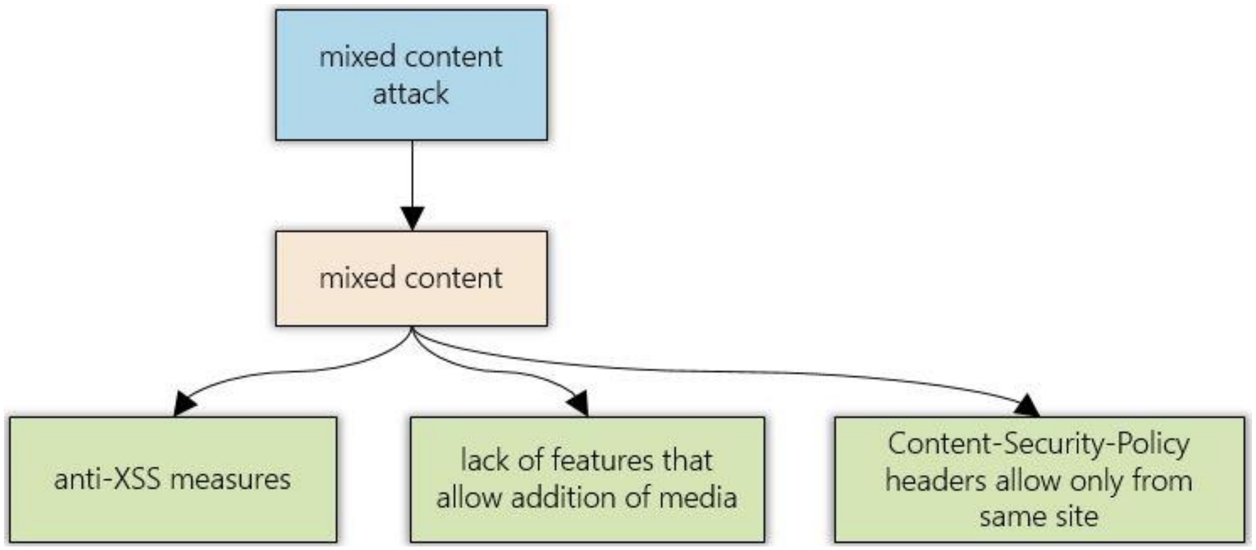

mixed content attack
mixed content
anti-XSS measures
lack of features that allow addition of media
Content-Security-Policy headers allow only from same site

Figure 3.8. Attack graph for mixed content.

An attacker may inject scripts or other malicious data into the HTTPS site by linking to an HTTP link via XSS.

This is prevented by protecting against XSS, as will be mentioned in section 3.1.9. Also, there is no functionality on the site that allows addition of media like images, audio, or video. All resources come from the server. As another layer of security, Content-Security-Policy headers are set to only allow content from the site itself.

3.1.9. XSS Attack in

- Login Page
- Search Page
- Register Page
- Add Product Page
- Edit Product Page
- Delete Product Page
- Create Special Account

STRIDE Classification: Tampering of Data

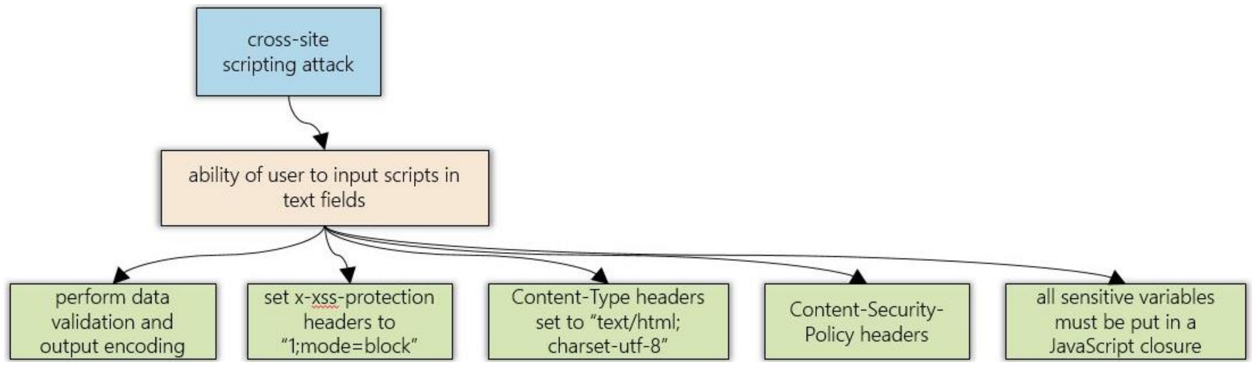

cross-site scripting attack
ability of user to input scripts in text fields
perform data validation and output encoding
set x-xss-protection headers to "1;mode=block"
Content-Type headers set to "text/html; charset-utf-8"
Content-Security-Policy headers
all sensitive variables must be put in a JavaScript closure

Figure 3.9. Attack graph for cross-site scripting attacks.

An attacker may inject a malicious script in the review of an item or in the username of a user or a search query or a product name or description for a compromised product manager. This script may call restricted functions or redirect the page or other possible attacks.

The site protects against these in multiple ways. As with SQL injection, data validation was performed via Javascript and the Controller module. Output encoding is performed using the Java Standard Tag Library's <c:out value=""/> function. For JS generated html, a custom escape function was used.

```
function escapeHtml(text) {
  var map = {
    '&': '&amp;',
    '<': '&lt;',
    '>': '&gt;',
    '"': '&quot;',
    "'": '&#039;'
  };

  return text.replace(/[&<>"']/g, function(m) {
    return map[m];
  });
}
```

This protects against even nested Reflected XSS attacks. Additionally, X-XSS-Protection headers are set to “1;mode=block”. Content-Type headers are set to “text/html; charset-utf-8”. Content-Security-Policy headers are also set to prevent loading of external scripts, stylesheets, or any foreign malicious content.

Also, to prevent against tampering with the JS variables, all sensitive variables are put within a Javascript closure, e.g.

```
var closure = (function() {
      var secureVar1 = null;
      var secureVar2 = null;

      function secureFunc1() {
            //body
      }

      function secureFunc2() {
            //body
      }

      return {
            openFunc1 : function() {
                  //body
            },
            openFunc2 : function() {
                  //body
            }
      };
})();
```

which means any authentication or status variables for the script cannot be accessed through console scripting. In the sample code, the only accessible parts of the closure are the return variables, openFunc1 and openFunc2. secureFunc1, secureFunc2, secureVar1, and secureVar2 cannot be accessed via console.

3.1.10. CSRF Attack in

- Login
- Register
- Review Product
- Add Product
- Edit Product
- Delete Product
- Create Special Account

STRIDE Classification: Spoofing Identity, Tampering of Data

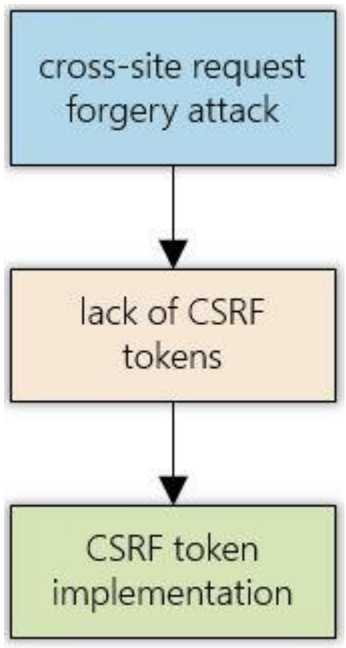


Figure 3.10. Attack graph for cross-site request forgeries.

An attacker may send a link to a CSRF site which pretends to be one of the post requests mentioned above. That means they can perform actions on behalf of a user against the user's will.

This is countered by introducing CSRF tokens in all forms across the site. These tokens are checked for each request if they match with the session's token. If the token is missing or incorrect, the request fails and is redirected to an appropriate page.

3.1.11. Clickjacking Attack in

- Login
- Register
- Review Product
- Add Product
- Edit Product
- Delete Product
- Create Special Account

STRIDE Classification: Spoofing Identity, Tampering of Data

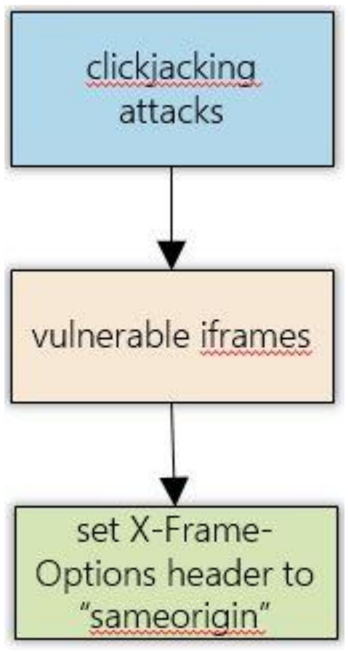


Figure 3.11. Attack graph for clickjacking attacks.

An attacker may include part of the site in an iframe on their own site and send it to a logged in user, making them think it's another site, which may lead to them performing an action that they do not wish to do. This is a textbook clickjacking attack.

To protect against this, an X-Frame-Options header is set to "sameorigin" to prevent the site from being included in other sites' iframes.

3.2. DREAD Ranking

Threat	D	R	E	A	DI	P	I	P x I	Rank
Session Fixation Attack	5	1	5	5	10	16	15	240	2
Unauthorized Access	5	1	1	10	10	12	15	180	4
DoS Attack	10	3	1	10	10	14	20	280	1
Faulty Audit System	10	3	1	5	10	14	15	210	3
Rainbow Table Attack	5	1	1	5	10	12	10	120	6
Session Hijacking	5	1	1	5	10	12	10	120	6
SQL Injection	10	1	1	10	10	12	20	240	2
Mixed Content Attack	10	1	1	10	10	12	20	240	2
XSS Attack	10	1	1	10	10	12	20	240	2
CSRF	5	1	1	5	10	12	10	120	6
Clickjacking	5	1	1	5	10	12	10	120	6

4. References

The OWASP Development Guide (2005). From https://www.owasp.org/index.php/OWASP_Guide_Project.

The OWASP Testing Guide v3 (2008). From https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf.

Appendix A: Distribution Of Work

Member	**Contribution**
Fernandez, Ryan Austin (Lead Developer)	Model Controller Load Testing Requirements Analysis Documentation
Hade, Alden Luc (Lead Designer)	Site Design (Wireframes and Mockups) Black Box Testing Load Testing Penetration Testing Documentation
Poblete, Clarisse Felicia (Lead Analyst)	User Interface Black Box Testing Functional Testing Penetration Testing Documentation
Syfu, Jonah (Lead QA)	Black Box Testing Functional Testing Penetration Testing Documentation