

**SECURDE**

Threat Model Document

|  |  |
| --- | --- |
| **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

2. Application Profile

3. Threat Documentation

1. Threat List
2. STRIDE/DREAD Model

4. References

**1. Security Objectives**

The Talaria Footwear Company’s online site must security against the most common attacks. This includes best practices regarding authentication, authorization, session management, cross-site scripting, SQL injection, data validation, cross site request forgery, clickjacking, and handling e-commerce payments.

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 not cookies or sessions available. Sessions must be untamperable through url rewriting or 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.

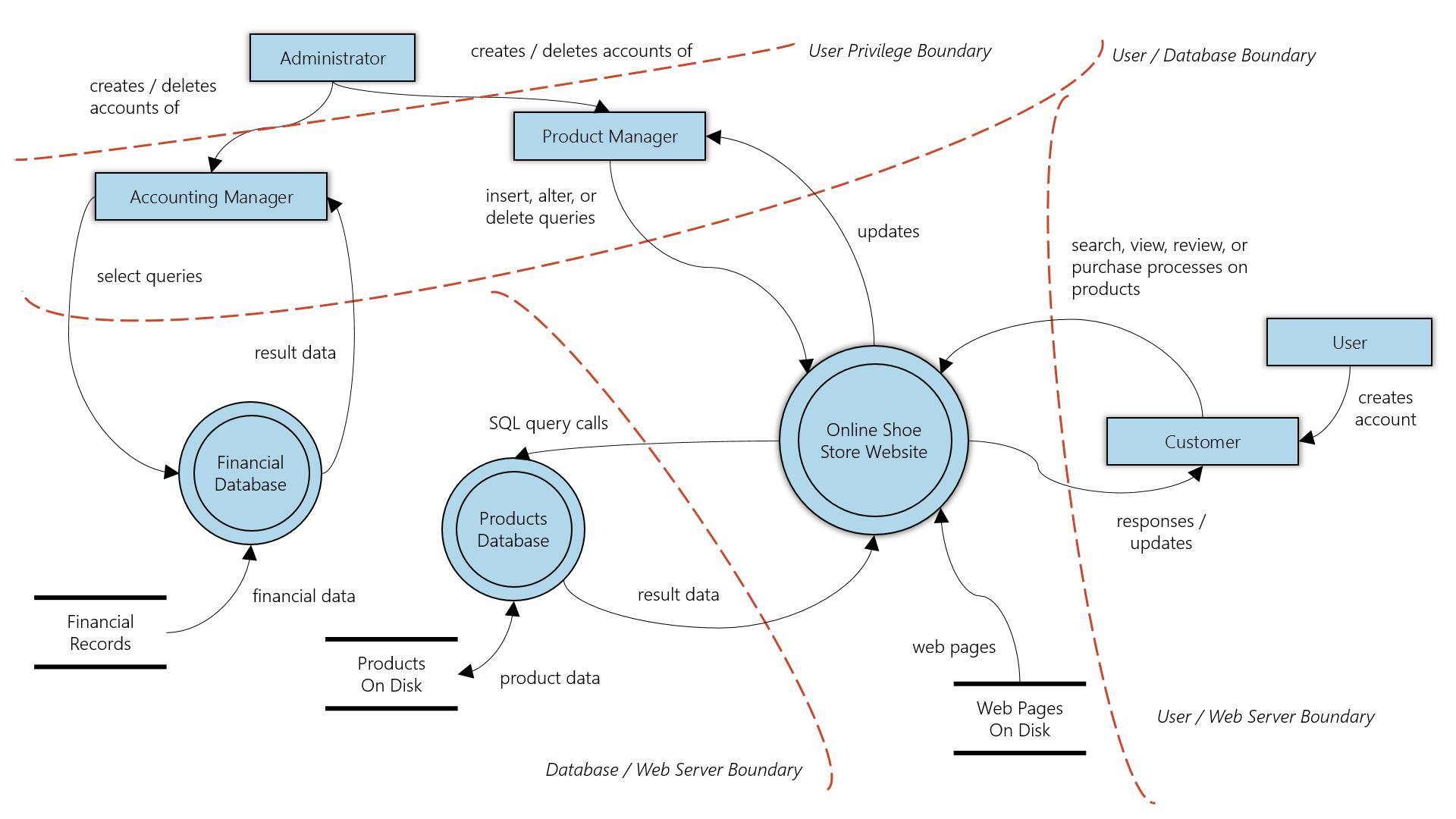
CSRF attacks must be mitigated using hidden tokens. Clickjacking must be mitigated using HTTP headers. E-Commerce payments must be handled securely i.e. no sensitive information is stored in the database.

**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**

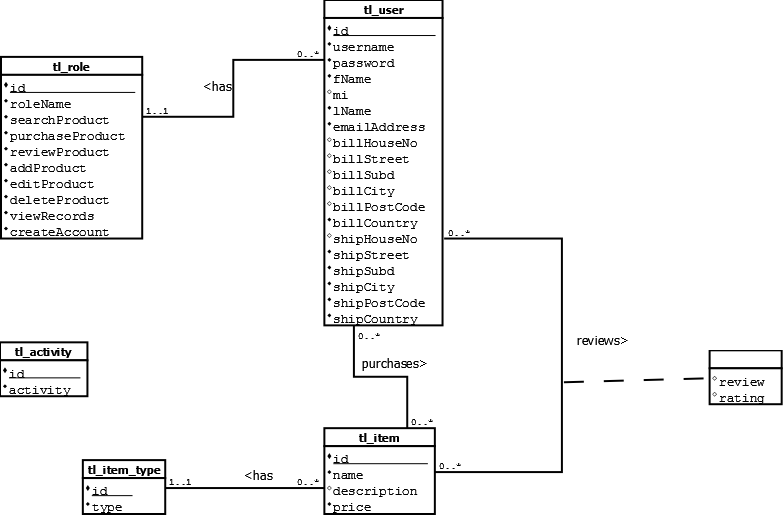
Figure 2.1. shows the Data Flow diagram of the site.



First, an administrator creates accounting managers and product managers. The product manager then adds, edits, or deletes products into the website, building a set or 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.

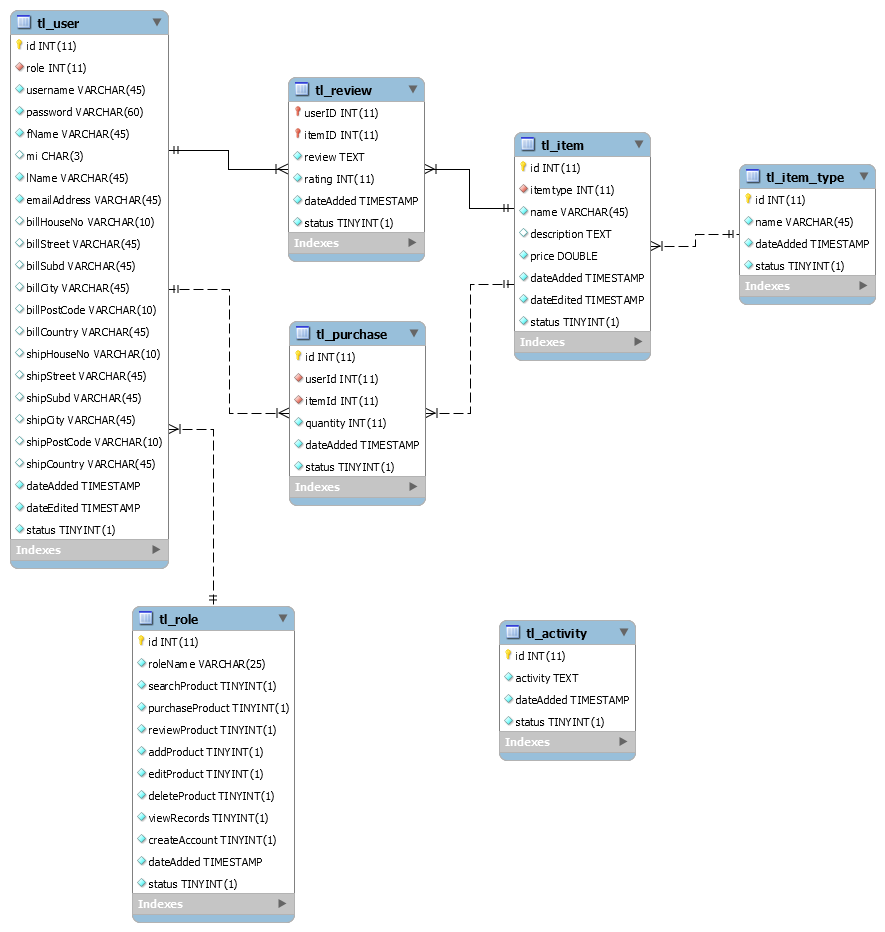


**Figure 2.2. – Entity-Relationship Model of Talaria Footwear Online Portal**

Users have a username, password, first name, middle initial, last name, and email address. Customer type users have a billing address (house no., street, subdivision, city, post code, country),and shipping address(house no., 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 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: the threat list documents each threat in detail; the DREAD Model prioritizes each threat by using the PxI scoring method.

**3.1 Threat List**

**Session Fixation Attack**

STRIDE Classification: Spoofing Identity, Information Disclosure

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 that url rewriting is disabled on the server. This is done by adding

<session-config>

<tracking-mode>COOKIE</tracking-mode>

</session-config>

in the 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. 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.

**Unauthorized Access in**

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

STRIDE Classification: Elevation of Privilege

Unauthorized access can be done by just logging into an account and exploiting the faulty authorization controls.

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, if the user has permission to access that page. 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.

**DDoS Attack in**

* Register Page
* Login Page

STRIDE Classification: Denial of Service

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.

**Faulty Auditing System**

STRIDE Classification: Repudiation

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.

**Rainbow Table Attack**

STRIDE Classification: Spoofing Identity, Information Disclosure

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.

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 one lowercase letter, one uppercase letter, one number, and one special character.

**Session Hijacking**

STRIDE Classification: Spoofing Identity

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 fixation attack occurs, a new session id is generated on login.

**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

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

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

**Mixed Content Attack**

STRIDE Classification: Tampering of Data

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 later. 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.

**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

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, which means any authentication or status variables for the script cannot be accessed through console scripting.

**CSRF Attack in**

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

STRIDE Classification: Spoofing Identity, Tampering of Data

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.

**Clickjacking Attack in**

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

STRIDE Classification: Spoofing Identity, Tampering of Data

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 Model**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |
| Login Bypass | 5 | 1 | 5 | 5 | 10 | 16 | 10 | 160 | 5 |
| 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 OWASP: https://www.owasp.org/index.php/OWASP

\_Guide\_Project

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

\_Testing\_Guide\_v3.pdf