A picture containing text, clipart

Description automatically generated

**Secure Software Development**

Year 2 (2022/23), Semester 3

**SCHOOL OF INFOCOMM TECHNOLOGY**

Diploma in Cybersecurity & Digital Forensics

Assignment

**Secure Web Application**

**Duration:** Weeks 11 - 15

**Weightage:** 40%

**Individual/Team/Both:** Both

**Deadline: 31 July 2022, 11:59pm (Sunday)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Group Name** | **Team BeanCave** | | | | |
| **Student IDs** | **Student**  **Dominic Lee** | **S10222894** | **Student**  **Han Xihe** | **S10222998** |  |
| **Student Names** | **Dominic Lee, Han Xihe** | | | | |
| **Module Group** | **P02** | | | | |

Table of Contents

[Heading 1 2](#_Toc1391709524)

[Heading 2 2](#_Toc1432387504)

[Heading 3 2](#_Toc407265522)

[1. Project Title 2](#_Toc1770775274)

[2. Project Vision 2](#_Toc466534969)

[3. Project Description 2](#_Toc1632937969)

[4. Project Features 3](#_Toc1524214662)

[5. Members & Features Worked On TAG TENTATIVE ref sample 5](#_Toc1299483323)

[6. 3-Tier Application Architecture TAG DIAGRAM TENTA. TO FEATURES 5](#_Toc657049828)

[7Feature 1 (Login/Registration/Authentication) 6](#_Toc407823138)

[7.1Secure Software Requirements 7](#_Toc2060460698)

[7.1.1Use Case and Misuse Case Modelling 8](#_Toc57756310)

[7.1.2Data Labelling & Classification 9](#_Toc1096652601)

[7.1.3 Subject/Object Matrix 10](#_Toc1821859149)

[7.2Secure Software Design 11](#_Toc1563396575)

[7.2.1 Threat Modeling 11](#_Toc2038648089)

[7.3Secure Software Design Coding 17](#_Toc630662584)

[7.3.1 Feature related Defensive Coding Practices 18](#_Toc282852495)

**TAG convert all hand-drawn diagrams to PowerPoint diagrams**

# Heading 1

## Heading 2

### Heading 3

#### Heading 4

# 1. Project Title

The team was engaged by OPhone Pte Ltd to develop a secure web application for selling smartphones online. Thus, the theme of the project’s web application is the sales of smartphones.

# 2. Project Vision

We aim to securely design a website that can withstand all attack vectors and yet be intuitive enough that the user experience is not compromised because of said security features.

# 3. Project Description

**TAG Project description is tentative to the code**

Our company, OPhone Pte Ltd, maintains and operates a website that sells brand-new or refurbished smartphones. Profit is earned as the company takes a percentage of each phone sale sold by sellers.

Customers and sellers are only allowed to buy or sell phones after registering for an account and logging in. The only payment method is cashless payment (credit/debit) and the price must be paid in full (no monthly instalments.) Furthermore, the phones will be shipped from the seller to the location that the customer provides. The customer will then be charged accordingly for the smartphone that they purchased.

There are 3 kinds of users: customers, sellers, and admins. Admins accounts have extra privileges and are reserved for the staff of OPhone. Customers are restricted to activities that facilitate the purchase of a smartphone, while sellers are restricted to activities that facilitate the listing and selling of smartphones.

Before purchasing a smartphone, customers should be allowed to view pictures of the smartphone provided by the seller and **TAG read reviews and ratings on the seller**. Sellers should be allowed to put up listings a relevant photo, modify the listings, and remove listings to facilitate the selling of phones.

Customers can search for mobile phone models and be presented with a list of phones that match their search query and brand filter. This page can only be edited by Admins or Sellers. The customers can then select which phones to purchase and then add them to a shopping card, before checking out with their shopping cart to purchase their items.

When using the app in the real world, the OPhone web application will be implemented in a 3-Tier architecture, going from a web server to an application server to a database server. Asp.net will be used for the application server and SQL will be used for the database server. **TAG additional details?**

# 4. Project Features

**TAG Furnish checkout page features once checkout page is done**

We will be implementing and securing 6 features for the web application. They consist of authentication, authorization, auditing, a CRUD database, a checkout page where the customers can checkout with their chosen purchases, and a search & filter feature for customers to look for specific devices.

Authentication consists of the signing up and logging in of users. If a user does not have an account created, they will have to use the sign-up page to create an account. Otherwise, a login page will be shown to them to sign in. To sign in they will need their username and password.

Authorization prevents users from performing actions that do not have rights too. An example of this would be a customer being allowed to edit the price of the phones. It will check if a user is authorized to perform a certain action.

Auditing is the feature that tracks and logs all user’s actions. This way, users will have to take responsibility for their actions ensuring non-repudiation. This can be done by monitoring significant user actions such as logging in. It can also detect hackers trying to crack user accounts.

Next, a CRUD database allows for user data to be stored and secured. This will be reserved only for admin users.

Lastly, a promotion page consists of the items that are on promotion. This page acts as an independent page from the page which sells items without discounts. Likewise, only admin users can update the page and regular users can only view and buy items from the page.

All in all, the OPhone web application aims to provide ease-of-use, while establishing security to protect against security threats by fulfilling the following quality attributes of secure software:

* **Confidentiality:** protect the information of all users against any type of unauthorized leakage or disclosure
* **Integrity:** ensure that the software and its data is protected against unauthorized or inappropriate changes, such that the resiliency of the software is ensured. This is so that the data of the software is stored, sent and processed as intended by the author of the password, and that the software performs reliably as expected
* **Availability:** by ensuring that the software stays operational, and the data of the software is handled and stored as expected, the availability of the software to authorized parties is very high, such that little to no downtime, if it ever happens, will be low-impact and the software will be available as soon as possible
* **Authentication:** ensures that the software verifies that the user is who they claim to be, such that the credentials of a user requesting access will have to be verified. This lends to authorization, accountability, and session management. Multi-factor authentication can be implemented for additional layers of security with the following factors combined:
  + **Knowledge:** what the user knows (username, plaintext password, identification details)
  + **Ownership: what** the user owns (tokens, smart cards, hardware-based authentication keys)
  + **Characteristic:** what the user is (biometrics of the user’s face, fingerprint, iris)
* **Authorization:** ensures that different types of users at different levels of an organizational hierarchy with different needs are given different levels of access and denial. This is such that lower-level users in an organization or service will have different privilege levels, so that low-level users do not perform actions that they are not intended to perform.
* **Accountability:**
* **Non-repudiation**
* **Session Management:**
* **Error/Exceptions Management:**

# 5. Members & Features Worked On **TAG TENTATIVE ref sample**

|  |  |
| --- | --- |
| **Member** | **Features** |
| Dominic |  |
| Xihe |  |

# 6. 3-Tier Application Architecture **TAG DIAGRAM TENTA. TO FEATURES**

An overview of the web application’s architecture is diagrammed, such that each component of the application can be determined. **TAG INSERT DIAGRAM**

**TAG take some non-implementable features from the diagram, e.g. SQL off-site backup slave server, cannot be implemented as it is not technically possible for us to simulate this backup in this assignment**

**We decided against a 3rd party backup, as 3rd parties may compromise our data without our knowledge**

**Talk about how it is not possible to have everything in this diagram**

Timeline

Description automatically generated

Figure 1 - OPhone's 3-tier application architecture

The design of the web application will be logically split into a 3-tier architecture, as shown in Figure 1. To make the design of the software scalable and easy to maintain for all operational variables and fulfilments, splitting the design of the software into multiple tiers introduces logical layering (Paul, 2014) that makes the entire application easier to work on, especially if multiple parties are involved.

Compared to lower-tiered architectures, the application will be less complex and intertwined, making development more efficient and less resource-intensive. Developers, especially those who were not involved in the initial development of the application, will be able to easily implement or upgrade components in the application, as they can easily understand what they must change, where they must do it, and how one component in one tier may affect another in another tier in the architecture. Integrating existing solutions to the application will be easier as well, thus speeding up development.

When there is separation between different components of the application into tiers, some components will become isolated from one another – for example, in Figure 1, the SQL database in Tier 3 does not directly meet any component of Tier 1. This reduces the attack surface of all components of both tiers, thus reducing the vulnerabilities available for malicious actors to exploit (IBM Cloud Education, 2020). Security gaps are also easier to close due to the reduced number of attack vectors in such architectures.

# 7 Feature 1 (Login/Registration/Authentication)

Since this web application services multiple users, it is imperative to implement some form of authentication to identify users who are requesting to use the web application and its services. It authenticates a user’s claim to check if the user requesting access is who they claim to be (Paul, OFFICIAL (ISC)^2 GUIDE TO THE CSSLP CBK, 2014). This is a basic element of security to ensure that potentially malicious but unskilled attackers do not perform any type of malicious act under the pretense of another identity claim, or without anything to identify them at all.

**TAG add security reasoning for authentication (login + registration) + copy paste from vision board**

**TAG for extra features related to authentication, consider the use of SSO and MFA with the existing login/registration features in place**

**TAG check for CSRF tokens**

## 7.1 Secure Software Requirements

Since OPhone’s web application will host a business that involves several entities performing different actions (i.e., seller who creates phone listings to sell phones, admins who have the authorization to perform any action on the site for the purposes of maintenance and administration, and users who look to buy phones), this web application will require authentication for various users and registration for new user accounts.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 2 - Login Page

### 7.1.1 Use Case and Misuse Case Modelling

Use case and misuse case modelling visualizes the intended way the web application is meant to be used with an order of actions or “use cases”. This maps out the functions of the web application clearly without confusion. Actors are identified in the use case with actions that can be performed, where misuse cases can be developed from. The misuse cases determine cases of abuse that an attacker or any other malicious actor can potentially perform, providing an understanding of potential threats to the web application.

Text

Description automatically generated

Figure 3 – Login/Registration: Use and Misuse Case for OPhone

Here we see that there are 3 misuse cases. To deal with brute force attacks, we can enable account lockout which locks the account after a certain number of failed attempts. We can use SSL to protect the password and username in transit and lastly, we can encrypt session IDs or use multi-factor authentication to prevent session hijacking (impersonate identity).

### 7.1.2 Data Labelling & Classification

Every web application will deal with several types of data in different mediums, formats, and levels of importance. Due to the differing nature of the types of data that OPhone will deal with, it is important to determine the levels of importance that each type of data must be treated with.

This can be done with data labeling & classification, where data is classified in a way that organizes its impact on the web application and the business under “Low, Medium, High” impacts, where the proper security measures can then be implemented to the data (Paul, OFFICIAL (ISC)^2 GUIDE TO THE CSSLP CBK, 2014)

In an ideal situation, instead of performing data labelling, organizations with large funds can treat all pieces of abusable data with the same levels of importance, and therefore the same levels of security. This approach will work well, but in the real world, an organization will have limited funds and resources for such measures. Placing equal importance on all data types will not give well-valued yields, so it is important to place different levels of security on differing data types to reduce the wastage of funds and resources. Moreover, not all data types have to be strongly protected, such as publicly available usernames. Failure to label data appropriately will lead to important data being leaked, as well as the company overspending on protecting data that is not so important.

**TAG insert explainer for data classification**

Diagram

Description automatically generated

Figure 4 – Login/Registration: Data Classification and Labelling

In the login and authentication feature, passwords and usernames are the data that will be stored in the database. From figure 4, passwords are confidential in nature as the attacker will be able to login to the user account with the password, however, there is insignificant impact on altercation and destruction. For both altercation and destruction, the attacker will not be able to access the user account, also, username used in conjunction with other forms of authentication such as email address/phone number should be enough to prove to the company’s support team to authenticate the user and change the user’s password.

Hence as disclosure of password brings high damage to the company, it is crucial that the company uses techniques to prevent disclosure of the user’s passwords. This includes hashing of passwords when storing them in the database.

On the other hand, the username has minimal impact on disclosure, altercation, and destruction. Even if the hacker knows the username, without the password they cannot login to the user account. And should the username be altered or deleted, the user can also use other means to prove his identity and recover his account from the company's support team.

As such, there are not any necessary measures to put in place to protect usernames that are stored in the database.

## 7.2 Secure Software Design

**Feature related security design**

**Secure design features**

1. **Single Sign on (no need screenshots)**
2. **Multi factor authentication (Screenshots)**
3. **Account Lockout (screenshots)**

**Secure design principles**

1. Defense in depth
2. Fail Safe
3. Economy of mechanisms
4. Psychological acceptability (Account lockout)

**Attack Surface Evaluation**

The entry point for the registration/login feature would be the login and registration pages. Those are the pages user inputs their username and password. There are no exit points for this feature as users will be directed to the home page after logging in.

After gathering the requirements for the software, it is imperative to derive the protections needed for the web application in the Design stage from the requirements that were identified. An architectural design can be created, where according to the logical flow of the business, the business’s workflows and priorities, and the severity of the threats at hand, a security-oriented architecture can be designed from the ground up during the development of the application.

Implementing secure software design, where security is integrated with development from the ground-up, has a few key benefits:

* **Ease of maintenance:** When the software is made with a clear design in mind, less design errors are made during initial development, increasing the overall reliability and quality of the software. This makes software easier to maintain while it is less error prone (Paul, OFFICIAL (ISC)^2 GUIDE TO THE CSSLP CBK, 2014).
* **Culture of security:** A habit or culture of security is promoted, where developers are encouraged or motivated to find security flaws early. This reduces the number of problems faced later, reducing development efforts later in the software’s lifecycle (Cypress Data Defense, 2020).
* **Design consistency:** When a clear design architecture is established, the software will be more consistent as developers design the software towards similar goals. The software will also go through minimal redesigns as it is handed from one development team to another.

### 7.2.1 Threat Modeling

From the requirements gathered for OPhone, the software faces several security threats that can be addressed. Measures to protect against such threats will then be implemented based on how severe the threats are to OPhone. The first step is to identify what possible attacks (using STRIDE) are followed by taking a deeper look at how these attacks could be carried out (using Attack Tree).

#### 7.2.1.1 STRIDE List

The STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Escalation of Privileges) list identifies threats and categorizes them according to what kind of threat it is. The threats are derived from the use case and misuse case modeling process from requirements. The following STRIDE structure is implemented by Microsoft, where they created it to make the threat modeling process easy to understand and as a cost-effective method of mapping out potential threats (Microsoft, 2022).

|  |  |
| --- | --- |
| **S – Spoofing** | **Impersonate Identity** |
| **T – Tampering** | **Modify account information**  **Change user access** |
| **R – Repudiation** |  |
| **I – Information Disclosure** | **Intercept password** |
| **D – Denial of Service** | **Flood system** |
| **E – Escalation of Privileges** | **Brute-force password**  **SQL-Injection** |

Table 1 – Login/Registration: STRIDE List

#### 7.2.1.2 Attack Tree

The attack tree sees the web application from the perspective of the attacker and adds upon the STRIDE list. It serves as an objective approach to branching out the methodologies or methods that an attacker may use to perform a certain objective or to use a certain attack vector.

Diagram

Description automatically generated

Figure 5 - Login/Registration Attack Tree: Impersonate Identity

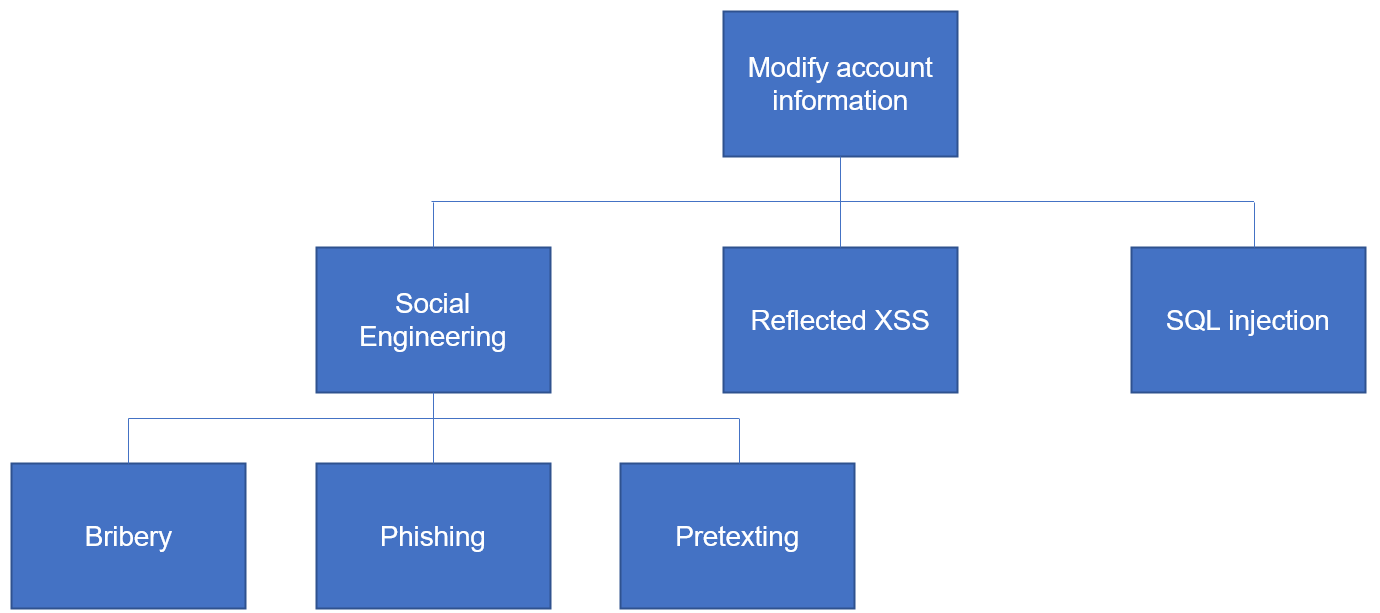
****

Figure 6 - Login/Registration Attack Tree: Brute-Force Password

**Graphical user interface, diagram, Teams

Description automatically generated**

Figure 7 - Login/Registration Attack Tree: Change user access

Diagram

Description automatically generated

Figure 8 - Login/Registration Attack Tree: Intercept password

**Diagram, timeline

Description automatically generated**

Figure 9 - Login/Registration Attack Tree: Flood system

Diagram

Description automatically generated

Figure 10 - Login/Registration Attack Tree: Brute-force password

Timeline

Description automatically generated

Figure 5 - Login/Registration Attack Tree: SQL injection

#### 7.2.1.3 Average Ranking

After determining the threats using STRIDE in 7.2.1.1, we can find the level of severity that the threats pose using average ranking. Average ranking quantifies the danger that each threat poses (Czagan, 2014) to the web application and its users, by assigning it to certain categories of damage and a numerical value based on the potential damage that the threat can cause (Paul, OFFICIAL (ISC)^2 GUIDE TO THE CSSLP CBK, 2014). The values assigned are then plugged into a set averaging formula to compute the risk (average rank) that each threat poses to the web application. This serves as a metric for us as we can have a score for each threat that is tangible and measurable. Which then allows us to compare the threats and rank them so that we know what to prioritize.

The average ranking strategy to be used here is DREAD, where each threat is given a score on a scale of 1 to 5 (Not severe to extremely severe) based on the danger of each category:

**TAG transfer to an appendix**

* **Damage potential:** How much damage can be caused when the threat is exploited by attackers
  + **1:** No threat
  + **2:** Users minorly inconvenienced by small changes caused by damage
  + **3:** User data affected
  + **4:** Major damage and user data exposed to public facing platforms
  + **5:** Thorough damage and destruction
* **Reproducibility:** Ease and frequency of exploiting the threat via the same attack vector(s)
  + **1:** Very difficult or impossible for all users
  + **2:** Tedious steps required with some form of authorized access needed, on top of unconventional zero-day exploits
  + **3:** Tedious steps required with some unconventional exploits needed
  + **4:** Few steps required with pre-made tools available on the Internet
  + **5:** Only simple tools that are available to everyone are required, such as pre-written scripts and web browsers
* **Exploitability:** Amount of effort, resources and technical know-how/skillsets needed by an attacker to exploit the threat
  + **1:** In-depth custom technical tools and knowledge are required
  + **2:** Additional research and resources required to conduct an attack
  + **3:** Scripts and pre-made tools available on the internet, or very simple tools and exploits
  + **4:** Additional steps and scripts are required in addition to the app itself and the web browser
  + **5:** Only simple, common tools are required, such as the app itself and a web browser
* **Affected users:** Number of users who will be affected when the threat is exploited
  + **1:** No users affected
  + **2:** Only a specific subset of users is affected in an unnoticeable manner
  + **3:** Only a small group of users are affected in an easily recoverable manner/minor inconvenience
  + **4:** Large groups of users are affected with some damage
  + **5:** All users are affected with widespread damage
* **Discoverability:** How easy it is for the threat to be uncovered by third parties, such as security researchers, analysts and potential attackers conducting intelligence gathering.
  + **1:** Very difficult, even with white box analysis (internal access of application interfaces for admins, documentation, architecture)
  + **2:** Difficult, requiring large amounts of resources on top of intelligence gathering, unconventional zero-day exploits, and extensive reverse engineering
  + **3:** Reverse engineering is possible by monitoring network activity, exit points, errors and other outputs
  + **4:** A few steps are needed with pre-made tools without reverse engineering or complex intelligence gathering
  + **5:** Sensitive information relating to the architecture of the web application or other sensitive user information is directly exposed on the web site itself, by simply running the site with a web browser

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Threats** | **Damage potential** | **Reproducibility** | **Exploitable** | **Affected user** | **Discoverability** | **Average**  **score** |
| **Bribery** | **5** | **3** | **5** | **5** | **2** | **4** |
| **Phishing** | **3** | **5** | **4** | **4** | **4** | **4** |
| **Pretexting** | **4** | **3** | **4** | **4** | **2** | **3.4** |
| **CSRF** | **4** | **4** | **5** | **3** | **5** | **4.2** |
| **Credential stuffing** | **3** | **5** | **4** | **3** | **3** | **3.6** |
| **XSS** | **4** | **4** | **5** | **4** | **5** | **4.4** |
| **SQL injection** | **4** | **4** | **5** | **4** | **5** | **4.4** |
| **Stolen certificate** | **5** | **2** | **2** | **4** | **4** | **3.4** |
| **Keylogging** | **4** | **3** | **2** | **5** | **1** | **3** |
| **Tailgating** | **5** | **5** | **5** | **5** | **1** | **4.2** |
| **Packet analysis** | **4** | **4** | **3** | **4** | **4** | **3.8** |
| **Protocol flooding** | **2** | **5** | **4** | **3** | **4** | **3.6** |
| **Malformed ping** | **2** | **5** | **4** | **3** | **4** | **3.6** |
| **Ping of death** | **2** | **5** | **4** | **3** | **4** | **3.6** |
| **Buffer overflow** | **1** | **3** | **4** | **3** | **3** | **2.8** |
| **Traditional brute-force** | **4** | **5** | **5** | **3** | **4** | **4.2** |
| **Password spraying** | **3** | **4** | **5** | **3** | **4** | **3.8** |
| **Dictionary** | **3** | **4** | **5** | **3** | **4** | **3.8** |

From the table, we can see that the 2 most dangerous attacks on the login and registration feature are SQL Injection and XSS. Thus, securing this login and registration feature against SQL Injection and XSS will be the top priority.

#### 7.2.1.4 Control measures (Solutions to the top threats)

OPhone is an organization that has limited resources, manpower and time. Thus, when developing the web app, it is not possible to implement protections for every single threat without sacrificing other parts of the project. A balance between the security and other aspects of the OPhone web application must be achieved, such that there is a balance between the “iron constraints”:

**TAG should I do iron constraints here? (Can but not too long)**

so, we have determined that threats with an average ranking above 4 will have control measures considered against them.

However, this does not mean that once the app has reached feature and business maturity, along with parity to existing platforms, the other security risks will be ignored. In the real world, when time and resources allow, the other lower-danger security flaws will be addressed as well, such that even low chances of attackers exploiting these flaws will be minimized or removed.

Via the threats modelled from the DREAD table, the biggest threats (average ranking above 4) were the following:

* **CSRF (Cross-Site Request Forgery)**
* **XSS (Cross-Site Scripting, both stored and reflected)**
* **SQL injection**
* **Tailgating**
* **Traditional brute-force**

**TAG Do after threat modelling is complete**

## 7.3 Secure Software Design Coding

Coding out secure software is the next important step in ensuring that the software is secure after identifying the threats that the software poses. The identification of threats allows developers to code software with specific vulnerabilities in mind to ensure tight integration of security measures. Security features needed can be identified to specifically deal with threats

### 7.3.1 Feature related Defensive Coding Practices

Features can be built into the web application as defensive measures to deal with certain offensive attacks that target specific vulnerabilities.

The main purpose of this feature is to take in user input to log them into their account. Hence this feature is extremely important to secure, as attackers will often try and find flaws with the login/signup page seeing that they can remain anonymous while doing so.

The inputs that the user gives must be treated as untrusted and security measures must be put in place to deal with these malicious inputs. Measures we will be using include input validation, sanitization, error handling, exception management and session management.

Input validation

-Input Validation

-Sanitization

-Error Handling

-Exception Management

-Session Management

## 7.4 Secure Software Design Testing

**Conclusion for the feature**

This feature allows users to register for an account as well as login to an existing account. We will be using form authentication to authenticate our users. Where a username and password are needed for registration and login.

The password the user gives will be protected as it will be hashed when stored in the database. We would also want to implement transport layer security such as SSL (Secure Sockets Layer) to cryptographically protect the user’s information from attackers intercepting the message in clear text.

Furthermore, we will perform input validation as well as input sanitization to defend against attacks such as SQL Injection as well as ensure that the user is providing appropriate inputs. A wrong password counter will be implemented to defend against brute-force attacks and - when the user password or username is wrong, a vague error message such as “Login Invalid” will appear that will not tell the user the nature of their error.

# 8 Feature 2 (Authorization)

**What authorization means**

In the web application, there will be pages that are authorized only for certain accounts; hence authorization must be ensured so that no unauthorized accounts will be able to view or manipulate the confidential pages.

Graphical user interface, text, application, Word

Description automatically generated

**Type of authorization used**

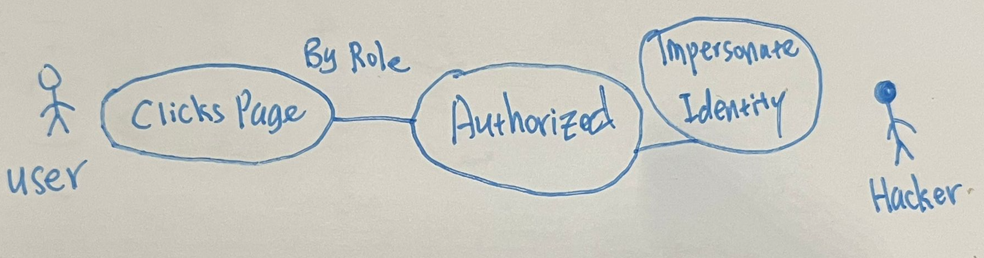
OPhone will use Role-Based Access Control (RBAC) to ensure authorization within the web application. The reason is that although individuals will change over time in RBAC, roles do not, and as resources are mapped to the role and not the individual, it allows for ease of user management. There will be 3 roles on our web application, “Users”, “Sellers, and “Admins”. Users will have the least privilege followed by sellers and lastly admins.

## 8.1 Secure Software Requirements

Here we will gather the security requirements for the authorization feature. As the authorization feature has no data being stored or used, data labelling is not required. The elicitation methods used will be use case and misuse case modelling as well as subject/object matrix.

### 8.1.1 Use case and Misuse Case Modelling

**TAG Any other attack vectors?**



The use case here is when the user clicks the page and is authorized because his role is allowed to access the resource, in this case, the page. The misuse case is when the hacker impersonates the identity of the user, and the web application trusts the identity of the user, thereby allowing the hacker to access the resource without the user’s knowledge or consent.

### 8.1.2 Subject/Object Matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | Admin | Seller | User |
| Phone Object Create Page | Authorized | Authorized | Unauthorized |
| Phone Object Delete Page | Authorized | Unauthorized | Unauthorized |
| Phone Object Edit Page | Authorized | Unauthorized | Unauthorized |
| Phone Order Create Page | Authorized | Authorized | Authorized |
| Phone Order Delete Page | Authorized | Unauthorized | Unauthorized |
| Phone Order Edit Page | Authorized | Unauthorized | Unauthorized |
| Audit Page | Authorized | Unauthorized | Unauthorized |
| Create Roles Page | Authorized | Unauthorized | Unauthorized |
| Manage User Roles Page | Authorized | Unauthorized | Unauthorized |
| Browse Phone Object Page | Authorized | Authorized | Authorized |
| Privacy Page | Authorized | Authorized | Authorized |
| Home Welcome Page | Authorized | Authorized | Authorized |
| Profile Page (2FA etc.) | Authorized | Authorized | Authorized |

Using a subject/object matrix it is clear what roles should be authorized to which pages. This way, the requirements are extremely clear and there is little room for misinterpretation.

## 8.2 Secure Software Design

### 8.2.1 Feature related security design

#### 8.2.1.1 Secure Design Techniques

Regarding authorization using RBAC, there are not many techniques used to further enhance the security. As such, there are no techniques available since it is merely checking whether the user’s roles have access to the resource he wants to access.

#### 8.2.1.2 Key Secure Design Principles

For the authorization feature, we will follow 3 secure design principles which include Separation of duties(SoD), least privilege and lastly psychological acceptability.

RBAC fulfils the principle of least privilege. Assuming roles are given the minimum privilege to perform all operations it has on a certain resource; the principle of least privilege will be met as users assigned to that role will not be given more privilege than needed to.

Separation of duties is important as we do not want a situation where a user is assigned two different roles which have mutually exclusive permissions. Doing so will make RBAC pointless as a user now has a role which should deny him entry as well as a role which makes him authorized hence proving to have no actual meaning.

Lastly, the authorization feature should consider psychological acceptability. When a user is denied entry to a certain resource, it is important that the error page is clear and concise. Otherwise, the user might get confused or if the access denied page has many words, the user might get frustrated or frightened which would dissuade him from using the web application.

By following these 3 design principles we will be able to have an authorization that works as intended while ensuring maximum security and will not shun away users.

### 8.2.2 Attack Surface Evaluation

In terms of the attack surface, the authorization feature has no entry points since it is merely just authorizing user’s access to the web application’s resources and there are no user inputs. It also does not have an exit point since the access denied page is a static page. It is important to note however that this access denied page must not be verbose to reduce the amount of information given to the user.

## 8.2.3 Threat Modelling

#### 8.2.3.1 STRIDE

|  |  |
| --- | --- |
| **S – Spoofing** |  |
| **T – Tampering** |  |
| **R – Repudiation** |  |
| **I – Information Disclosure** |  |
| **D – Denial of Service** |  |
| **E – Escalation of Privileges** | **Inside threats** |

#### 8.2.3.2 Attack Trees

Below will be the attack trees which will further explore how these attacks could be carried out.

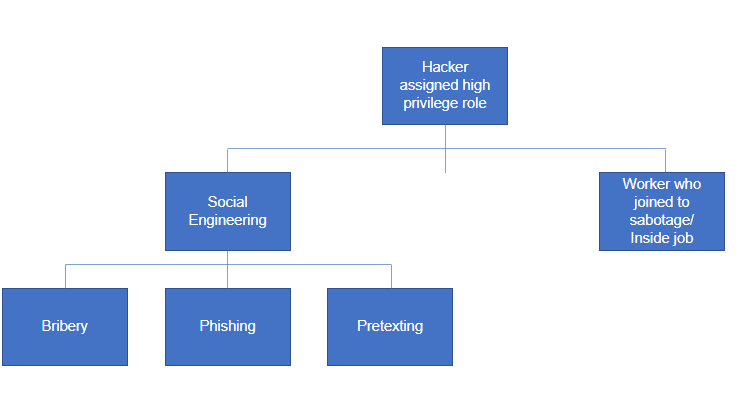


Figure 12 – Authorization Attack Tree: Inside Threat

Diagram

Description automatically generated

Figure 13 – Authorization Attack Tree: Impersonate admin account

#### 8.2.3.3 PxI Ranking

Here we will be using PxI to rank the threats as the Impact of an authorization breach has a larger impact than the probability of an authorization breach.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Threats | Damage potential | Reproducibility | Exploitability | Affected users | Discoverability |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

#### 8.2.3.4 Control Measures

## 8.3 Secure Software Coding

### 8.3.1 Feature related defensive coding practices

It is imperative that there are session management practices put into place to mitigate session hijacking or man in the middle (MITM) attacks. Session hijacking allows the attacker to impersonate the identity of a valid user by interjecting themselves into the middle of an existing session. This can lead to the attacker being able to access and perform actions that are unauthorized which can lead to information disclosure, alteration, or a denial of service.

Session management works to prevent these kinds of attack. It issues a unique session token for every single session and tracks user activity so that it can prevent MITM attacks. It is also good to set the session inactivity timeout as short as possible which reduces the amount of time the hacker has to hijack a session.

By doing so, we are ensuring authorization in the web-application.

### 8.3.2 Secure Software Processes

#### 8.3.2.1 Versioning

#### 8.3.2.2 Coda Analysis

#### 8.3.2.3 Code/Peer review

## 8.4 Secure Software Testing

### 8.4.1 Test Strategy

### 8.4.2 Test Plan

### 8.4.3 Test Cases