|  |
| --- |
| **Quality Assessment for Alinma Banking App** |
| Supervisor: Dr Sultan S Al-Qahtani  Section number: 171 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | # | **Student Name** | **Email** | **Number ID** | | **1** | **Faisal Saleh Aldarees** | **fsaaldarees@sm.imamu.edu.sa** | **440012582** | | **2** | **Abdulaziz Moh. Alotaibi** | **amaalotaibi77@sm.imamu.edu.sa** | **440014077** | | **3** | **Abdulrahman Al-Ghofaily** | **aaaalghofaily@sm.imamu.edu.sa** | **440016603** |  |  |  |  | | --- | --- | --- | |  | **Student Name** | **Number ID** | | **1** | **Faisal Saleh Aldarees** | **440012582** | | **2** | **Abdulaziz Mohammed Alotaibi** | **440014077** | | **3** | **Sultan khaled Almansour** | **440016632** | |

Table of Contents and figures:

[Table of figures: 2](#_Toc86890038)

[**Introduction:** 3](file:///C:\Users\96654\Desktop\abdulaziz\ImamU\CS-level5\SoftWareENG%20-%20CS392\WorkShops\WorkShop1\Report\SWE-WorkShop1_4.docx#_Toc86890039)

[**Alinama Mobile App analysis results** 4](#_Toc86890040)

[Clear text traffic: 4](#_Toc86890041)

[Clear text in alinma banking app network traffic: 4](#_Toc86890042)

[How to protect the app from these mistakes? 5](#_Toc86890043)

[Content provider: 6](#_Toc86890044)

[Content URI 6](#_Toc86890045)

[Clear text secret key 8](#_Toc86890046)

[**Environment variables** 8](#_Toc86890047)

[**Kubernetes** 9](#_Toc86890048)

[Executing raw SQL query 10](#_Toc86890049)

[**SQL Injection** 10](#_Toc86890050)

[**Why this code will be vulnerable for SQL injection?** 11](#_Toc86890051)

[Using MD5 hash 12](#_Toc86890052)

[**What is hashing?** 12](#_Toc86890053)

[**Why MD5 is not secure?!** 13](#_Toc86890054)

[**How to solve this problem?** 13](#_Toc86890055)

[logs information: 14](#_Toc86890056)

[**What should we do to prevent or avoid exposing user data?** 14](#_Toc86890057)

[Insecure random number generator: 14](#_Toc86890058)

[**How could we generate a safe random number?** 15](#_Toc86890059)

[Encryption mode CBC with PKCS5/PKCS7 padding: 15](#_Toc86890060)

[**What is CBC?** 15](#_Toc86890061)

[**What is PKCS5/PKCS7 Padding?** 16](#_Toc86890062)

[**What is Padding oracle attack?** 16](#_Toc86890063)

[**How to protect against Padding Oracle Attack?** 16](#_Toc86890064)

[**Code Quality** 17](#_Toc86890065)

[Vertical alignment: 17](#_Toc86890066)

[Ambiguity: 17](#_Toc86890067)

[High degree of connectivity: 18](#_Toc86890068)

[**Tools** 19](#_Toc86890069)

[Visual Paradigm: 19](#_Toc86890070)

[**Why did we use Visual Paradigm?** 19](#_Toc86890071)

[MOBSF: 19](#_Toc86890072)

[**Why did we use MOBSF?** 19](#_Toc86890073)

[**Conclusion** 20](#_Toc86890074)

[**References** 21](#_Toc86890075)

## Table of figures:

[Figure 1: Clear text traffic 4](file:///C:\Users\96654\Desktop\abdulaziz\ImamU\CS-level5\SoftWareENG%20-%20CS392\WorkShops\WorkShop1\Report\SWE-WorkShop1.docx#_Toc86609347)

[Figure 2: URI example 6](#_Toc86609348)

[Figure 3: Content provider layer 7](file:///C:\Users\96654\Desktop\abdulaziz\ImamU\CS-level5\SoftWareENG%20-%20CS392\WorkShops\WorkShop1\Report\SWE-WorkShop1.docx#_Toc86609349)

[Figure 4: Environment Variable example in Helm 9](#_Toc86609350)

[Figure 5: Example of how SQL injection works 10](#_Toc86609351)

[Figure 6, How hashing works 12](file:///C:\Users\96654\Desktop\abdulaziz\ImamU\CS-level5\SoftWareENG%20-%20CS392\WorkShops\WorkShop1\Report\SWE-WorkShop1.docx#_Toc86609352)

[Figure 7 14](file:///C:\Users\96654\Desktop\abdulaziz\ImamU\CS-level5\SoftWareENG%20-%20CS392\WorkShops\WorkShop1\Report\SWE-WorkShop1.docx#_Toc86609353)

[Figure 8 14](file:///C:\Users\96654\Desktop\abdulaziz\ImamU\CS-level5\SoftWareENG%20-%20CS392\WorkShops\WorkShop1\Report\SWE-WorkShop1.docx#_Toc86609354)

[Figure 9: Cipher Block Chaining (CBC) mode encryption 15](#_Toc86609355)

[Figure 10: A class acts as a "hub" 18](#_Toc86609356)

## 

|  |  |  |
| --- | --- | --- |
| Quality Assessment for Alinma Banking App  Quality Assessment for Alinma Banking App **Introduction:** Introduction:  At first, we hope that whom who is reading this report get the benefits from it, in this workshop, we will make a static analyzation of Alinma Banking app (Android version) using MobSF tool, and more! the main purpose for making this workshop is to apply the theoretical skills that we took from the book into practical / real-world quality assurance evaluation. We will take the most quality assessment ways that we have learned in this course, for example, evaluating the app in terms of clean code, code structure, security, and more, after evaluation, we will research about those problems and find the best solution possible to solve the issues. |  | Terms we will check in this workshop:  Code documentation  source code quality  Is the code:   * Readable * Understandable * not over complicated * does well what is intended to do |

# **Alinama Mobile App analysis results**

## Clear text traffic:

Diagram, timeline

Description automatically generated with medium confidenceCleartext is transmitted or stored text that has not been subjected to encryption and is not meant to be encrypted. As such, cleartext does not require decryption in order to be displayed. So, we know that the cleartext is not secure for communication. Because when your app communicates with servers using cleartext network traffic, such as HTTP, the traffic risks being eavesdropped upon and tampered by third parties. This may leak information about your users and open your app up to injection of unauthorized content or exploits. Ideally, your app should use secure traffic only, such as by using HTTPs instead of HTTP. Such traffic is protected against eavesdropping and tampering. The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and modify it without being detected.

Figure : Clear text traffic

## Clear text in alinma banking app network traffic:

Alinma bank using the cleartext in their network traffic which make the app is not secure and there will be some risks in communication because they are using cleartext HTTP and FTP stacks. As we said this may leak information about your users and open your app up to injection of unauthorized content. However, some of them sometimes regress to cleartext traffic by accident. For example, an inadvertent change in one of the server components could make the server provide the app with HTTP URLs instead of HTTPs URLs. Because Web browsers communicate with web servers using the HyperText Transfer Protocol (HTTP). When you click a link on a web page, submit a form, or run a search, the browser sends an HTTPs Request to the server. So, if the server components changed the server provide the app with HTTP URLs instead of HTTPS URLs The app would then proceed to communicate in cleartext.

## How to protect the app from these mistakes?

There is lots of things and here are **two mechanisms:**

**First**: (**Block cleartext traffic in production**):

To protect the installed base of your app against regressions to cleartext traffic, declare android: usesCleartextTraffic=”false” attribute on the application element in your app’s. When the attribute is set to "false", platform components (for example, HTTP and FTP stacks, will refuse the app's requests to use cleartext traffic. Third-party libraries are strongly encouraged to honor this setting as well. The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and modify it without being detected. For example, if your app accidentally attempts to sign in the user via a cleartext HTTP request, the request will be blocked, and the user’s identity and password will not leak to the network.

***Note:*** **android: usesCleartextTraffic** is an attribute to protect the installed base of your app against regressions to cleartext traffic.

**Second:** (**Detect cleartext traffic during development**):

To spot cleartext traffic during development **StrictMode** can help you with that**.** **StrictMode** is a tool for highlighting potential problems in an application. This is a useful tool for identifying which bits of the app are using TLS/SSL traffic. **SSL** is a cryptographic protocol that uses explicit connections to establish secure communication between web server and client. **TLS** is also a cryptographic protocol that provides secure communication between web server and client via implicit connections. Unlike the **android:usesCleartextTraffic attribute**, this feature is not meant to be enabled in app builds distributed to users.

SSL stands for **Secure Sockets Layer**.

TLS stands for **Transport Layer Security.**

## Content provider:







A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object. it involves some drawbacks. In their current state, content providers can only use SQLite as a storage mechanism. This may change in the future.

In alinma banking app Content Provider is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

## ****Content URI****

Content URI (Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.

As we see in the alinma banking app code:

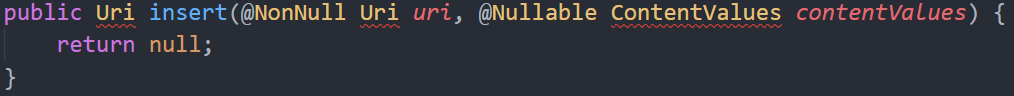


Figure 2: URI example

And in this figure will explain the content provider in which layer and how it passes the data and the information that the app needs:

Diagram

Description automatically generated

Figure 3: Content provider layer

## Clear text secret key

After static analyzing and viewing the source code, we have noticed that one of the secret keys is so simple, and in clear text.

com/

├─ bumptech/

├─ manager/

├─ RequestManagerRetriver.java

In this file, line 40 we have **final String FRAGMENT\_INDEX\_KEY = “key”,** this is a major security issue, even though that the secret key cannot extract confidential data, this represents a weakness in the security of the app code and that it is not well implemented.

After researching, we have concluded that using Environment Variables is the best way to overcome this issue, because it is not good practice if you store secret keys hard codded in the source code, since the code will be available for the public, and even if it is closed source, it would be available for the developers in the company that are not concerned with those keys.

### **Environment variables**

Environment variables, as the name says, are variables that are stored in the Operating System, not in the memory at compilation time, in order to set an environment variable in Linux by running in your terminal the command:

*export KEY=”YOUR\_SECRET\_KEY”*

and in order to view it run:

*echo $KEY*

but this is not a good way for making environment variables since you would need to manually set the environment variable yourself each time you want to add one.

### **Kubernetes**

One of the modern technologies released by google called Kubernetes (aka k8s) and one of its features is supporting the implementation of environment variables in production, it will support it by initializing the environment variables in a set of configuration files (a file for each container, each container is a Docker image) called “***Helm charts***” and those variables will be introduced in the Operating System as soon as the containers deployed to the server. K8s acts as docker-compose but in a production level, but docker-compose is used just in a development level.

You can read more about Kubernetes and docker-compose in:

<https://kubernetes.io/>

<https://docs.docker.com/compose/>

As shown in the figure bellow, you can see a simple way of implementing environment variables using k8s and Helm charts,

Text

Description automatically generated

Figure 4: Environment Variable example in Helm

And at the end, you can fetch the environment variable to your code using the following code:

*String value = System.getenv(yourEnvironmentVariableName);*

So, for the problem of the hardcoded credentials we suggest using environment variables using k8s or any technology that you find suitable, and for the simple secret key, you can simply just make a more complicated one.

## Executing raw SQL query

Another security issue that we have found, which is the use of SQLite database and executing the queries as raw text, for example, in

com/

├─ pushwoosh/

├─ f/

├─ c/

├─ b.java

line 292, we have

*private static String e(Collection<String> collection) {*

*return "DELETE FROM " + f.a + " WHERE " + "inbox\_id" + " IN ('" + TextUtils.join("', '", collection) + "')";*

*}*

This issue can cause SQL injection

### **SQL Injection**

SQL injection is an attack that might result unauthorized access to sensitive data, such as user’s credentials.

Diagram

Description automatically generated

Figure 5: Example of how SQL injection works (Web Unique Method (WUM): An Open Source Blackbox Scanner for Detecting Web Vulnerabilities*, Moh Noman, Moh Iqbal, Moh T Alam*)

You can read more about SQL injection in (https://portswigger.net/web-security/sql-injection)

### **Why this code will be vulnerable for SQL injection?**

In this code we have

*return "DELETE FROM " + f.a + " WHERE " + "inbox\_id" + " IN ('" + TextUtils.join("', '", collection) + "')";*

The attacker might for example make the value of ***f.a*** as a SQL query, not just as a string, and at the end of the query he can add “-- -” to consider all of the rest as a comment, so simply he can do anything with this database

The code was obfuscated, so we were not sure what dose it do, but after researching, we have found that ***pushwoosh*** is a package for automated push-notifications, so, we suggest using another package.

## Using MD5 hash

We have found another issue, and also with the package called pushwoosh that we have discussed earlier! The issue is the use of MD5 hash in:

com/

├─ pushwoosh/

├─ internal/

├─ platform/

├─ utils/

├─ GeneralUtils.java

line 131, we have a method declared as

*private static String md5(String str){*

*.*

*.*

*.*

*}*

That takes a string and return the MD5 hash of that string.

A picture containing diagram

Description automatically generated

### **What is hashing?**

Hashing is a process of transforming a given text (key, string of characters) into another fixed-sized value called “enciphered text” or hash value or just hash.

Figure 6, How hashing works (cwatch.comodo.com)

For example, if your password is ‘qwerty’ (bad idea), in the database you’ll have d8578edf8458ce06fbc5bb76a58c5ca

Do not confuse yourself with encryption, as encryption could be decrypted using a private key, but hashing is a “*One-Way function*” which makes it very hard (almost imposable) to reconstruct the original input from the output hash, a good way that I like to explain what hash is, that the original value is a fruit, if you take the juice of that fruit(The hash value) so you cannot get the juice back to fruit again.

### **Why MD5 is not secure?!**

There are a lot of dictionary tables you can find online for MD5, for example, in MD5decrypt, MD5Online, the dictionaries are there because MD5 is kinda old hash, so it is widely used in huge databases over the years, so if your password is common, it is most likely to find your password in that huge dictionary.

Also, Brute force attacks are fast on MD5 hashes, since it is fast to use, so it is easy to try a lot of combinations in a few seconds. And there are a lot of other issues

(You can read more in *Securing SQL server by Denny Cherry, Chapter 4: Database encryption*)

### **How to solve this problem?**

We have to ways of solving this issue, the first on is basically changing the used hash, and the second one is by using something called “*Salt*”

#### **What is MD5 salt?**

In cryptography salting is a random string that you add to the one-way function, to generate a different hash that with the original string alone. MD5 does not provide this, it you can easily you can concatenate to strings to get the benefits of salting.

The benefits of salting, that that your hashes in the database will be different than those in the big dictionary, also it will make that hash characters longer so it will be hard to brute force.

## logs information:

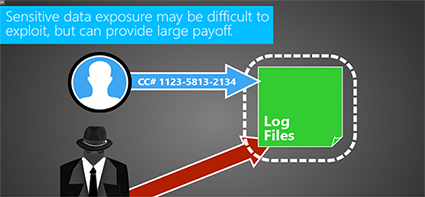
The app create log files. And creating a log file is a good way to keep track the app on different sides during the development phase, such as tracking errors, crashes, and performance. These logs are stored locally on the user device when the app is offline and sent back to the endpoint when the app is back to online. However, these logs may contain a sensitive data or information, and creating logs for these may expose it to the attackers or malicious applications.

Figure 7

### **What should we do to prevent or avoid exposing user data?**

The ideal solution is to set the logging levels appropriately, and minimize logging files as possible, so it doesn’t violate the user confidentiality.

## Insecure random number generator:

The app uses an insecure random number generator (which is from java.util.Random library).

com/

├─ pushwoosh/

      ├─ internal/

           ├─ c/

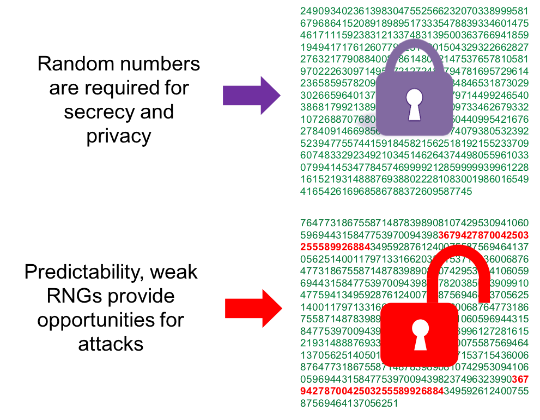
 ├─ d.java

Figure 8

This random number generator is easy to break since it is not cryptographically strong. The numbers that are generated by it are not completely random. It uses the clock as the seed / or to generate the seed (seed is a number that used to generate (semi-random) number using an algorithm). So, if the attacker knows when the seed was generated, the attacker will be able to reproduce the seed using the same algorithm.

Another issue is the seed is 48 bits-based, meaning 248 attempts are required to guess that random number, and it is not a big deal for today’s CPU’s.

### **How could we generate a safe random number?**

The solution is to use more advanced/strong cryptographically random number generators, such as (java.util.SecureRandom) generator. The SecureRandom generates non-deterministic output. It takes random data from the OS and use it as a seed. It is also 128 bits-based seed, which means 2128 attempts are required to break it.

## Encryption mode CBC with PKCS5/PKCS7 padding:

The app uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attack, which may lead to leakage of sensitive data. As we can see in:

com/

├─ pushwoosh/

      ├─ internal/

├─ a/

├─ a.java

Line 11, an instance of Cipher “AES/CBC/PKCS7Padding” was declared as

*Cipher instance = Cipher.getInstance(“AES/CBC/PKCS7Padding”);*

And being used to encrypt given byte array.

### **What is CBC?**

Shape

Description automatically generated with medium confidenceCBC stands for Cipher-Block Chaining, it is a mode of operation for block cipher, it means that we use the ciphertext from the previous block to impact the next one. The graph in (Figure 9) explains how CBC encryption mode works.

Figure 9: Cipher Block Chaining (CBC) mode encryption (https://defuse.ca/cbcmodeiv.htm)

To break the graph down, if we have m message blocks of size B that contains a plaintext, using the CBC mode, we will do the next. For the first block we will use a random “Initialization vector” (IV) of fixed-size input, we will XOR it with the first message block, then the result will be encrypted by AES encryption function using a given key. Now the output of this whole process is a ciphertext C. We will use this ciphertext of the first message block to encrypt the next message block by XOR the ciphertext with the second message block. And the process will be repeated until the whole message blocks are chained.

Just as explained Figure 1 shows the process of CBC mode.

To summarize the whole process into a formula: **Ci = EK(Bi ⊕ Ci-1)**

Where **EK** denotes the block encryption algorithm using key **K**, **Ci-1**is the cipher corresponding to **Bi-1**.

### **What is PKCS5/PKCS7 Padding?**

Let’s say we have a message block of size 8 bytes, and the plaintext that it occupies only 5 bytes of it, we still have 3 bytes that are empty, this is when padding take a place. Padding is used in a block cipher where we fill up the blocks with padding bytes. The padding depend on the number of bytes that do not have any content. There are many padding standards, two of them are PKCS5 and PKCS7, PKCS5 Password-based encryption standard, meanwhile Cryptographic Message Syntax standard.

### **What is Padding oracle attack?**

Through a padding oracle attack, an attacker can decrypt information, without knowing the encryption key. The attacker will need 256 SSL 3.0 requests on average to reveal one byte of the encrypted message. On every request the attacker make, the application will response. There are 3 types of responses:

1. Valid padding & valid data. The response status is 200 OK.
2. Invalid padding. The response Status is 500 Internal Server Error, or 403.
3. Valid padding & invalid data. The response status is 404 Not Found.

By repeating the process, the attacker can reveal the whole message byte by byte.

### **How to protect against Padding Oracle Attack?**

We need to make sure that the application does not return a different error message when padding is wrong. Using an Encrypt-then-MAC construction, where a Message Authentication Code (MAC) is applied to the ciphertext.

# **Code Quality**

## Vertical alignment:

Vertical alignment is the state or act of lining items up, one above each other. we found in this path (com\alinma\retail\mobile) some classes don’t use Vertical alignment concept and that makes the code harder to read but we can say that the overall quality of this classes is fine and to solve this problem we can use tools to help us for example checkstyle, and Prettier in Visual Studio Code.

## Ambiguity:

we found some ambiguity in many classes. in this path for example (com\alinma\tls) at line 49 we found some dollar sign ($) and it doesn’t mean anything related to the code in the class also at the same class in line 54 we can see a print message that says “AAA” also it is ambiguous. Another path also we found in it some Ambiguity is this path (com\cordova\plugin\android\fingerprintauth) in line 297 there is a variable called z the method takes it as a parameter, in the same path line 358 there is a variable e the method also takes it as a parameter.

## High degree of connectivity:

صورة تحتوي على نص, سماء, المصباح, لقطة شاشة

تم إنشاء الوصف تلقائياًBy analyzing the relationships between the classes, we found out there is a high degree of connectivity among the objects. System with “hub” modules is more prone to failure.

صورة تحتوي على نص, سماء, يوم

تم إنشاء الوصف تلقائياً

Figure : UML diagram shows CordovaPlugin class

صورة تحتوي على منضدة

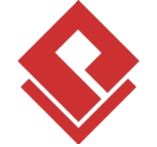
تم إنشاء الوصف تلقائياً

As we can see in the image (Figure 10), CordovaPlugin class acts as a hub in the system, if this class failed or had a security issue, all classes that in relationship with this class (Figure 11) will fail. And to avoid this we should lower the degree of connectivity.

Figure : Classes are in relationship with CordovaPlugin

# **Tools**

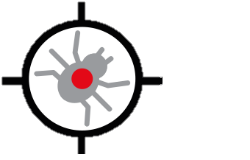
## Visual Paradigm:

Visual Paradigm is a UML tool that supports creating and designing of different UML models, such as Class, Use case, Sequence diagrams and more. The tool can generate code automatically from the UML models. It also provides report generation. It can reverse engineer, means that we could pass a code into this tool, and it will automatically generate an UML diagram based on the code.

### **Why did we use Visual Paradigm?**

We used Visual Paradigm to reverse engineer the java code in the application and convert it to UML Class diagram, to analyze the relationships between the classes and study the patterns being used in the app.

## MOBSF:

MobSF or Mobile Security Framework is an open-source security assessment tool that can perform both dynamic and static analyses. This all-in-one tool that has functionalities for Android, Windows and iOS platforms can also perform pen-testing and malware analysis.

### **Why did we use MOBSF?**

We used MOBSF because we can perform static analysis of the given apk file. It shows the security issues that are in the code. And it allows the user to get access and download the source code.

# **Conclusion**

In conclusion, we can say that there are problems in the application, some of them in security, and others are related to the cleanliness of the code, and they must be dealt with by the development department according to the solutions presented in the above report, and they must be implemented according to the seriousness.

The problems of encrypted messages are serious. Code problems are of course less serious than message encryption problems between the application and the server, but they are important in maintenance and making the code readable.

Finally, we can say that the quality is not related to security problems alone, but also to the problems of the code, its cleanliness, and the way the application operates and maintains it. In all these matters, quality plays an important role and must be dealt with caution and more attention.

# **References**

<https://www.geeksforgeeks.org/random-vs-secure-random-numbers-java/>

<https://cwe.mitre.org/data/definitions/532.html>

<https://asecuritysite.com/encryption/padding_des_salt>

<https://www.youtube.com/watch?v=R3NosHMSi0o&ab_channel=BillBuchananOBE>

<https://www.youtube.com/watch?v=RechScPSWLM&t=331s&ab_channel=JianrongZhou>

<https://portswigger.net/web-security/sql-injection>

<https://infosecscout.com/md5-salt-hash/>