Al-Imam University
Computer Science Department
College of Computer and Information Sciences
CS292: Software Engineering2
1st Semester 1443-1444 H

Quality Assessment for Alinma Bank App

CS392 – Software Engineering 2
Section Code: 373
Assignment: Workshop1
1st Semester / Year: 2021

Student Name	Student Email	Student ID
Rahaf fahad alsqiaan	rfmalsqiaan@sm.imamu.edu.sa	440019816
Nouf mohammed Alajmi	nomoalajmi@sm.imamu.edu.sa	441022333
Hajar abdullaziz Aljassar	haaaljassar@imamu.edu.sa	440020746

Supervisor:

Dr.Lamees Alhazzaa

Table of Contents page

Table of illustration

Page

1. Abstract

In this report, we will introduce the empirical studies on the quality assessments of local banking app (Alinma bank) to provide useful insights and improve the quality of this app.

2. Introduction

During the process of setting up the MobSf (Mobile Security Framework), we had faced several issues. Starting by searching up to find an appropriate APK Downloader, until we found a plug-in application in the GoogleChrome browser, called APK Downloader, to get AL-INMA APK File. Then, we used the Ubuntu operating system after installing it in a VirtualBox Machine, in this pattern, we avoided any problems that could have happened in the main device, also, setting up the Docker, to get MobSF, requires a large amount of space, and it was flexible to handle the Docker commands in Ubuntu than other operating systems. In Ubuntu itself, we had to increase the space taken from the disk to make the Docker work properly. Eventually, we launched the MobSF and associate the APK file to start analyzing.
Using only MobSF tools was not enough to cover some aspects of static analyzing, which is code quality. Thus, after doing some researches, we had found Codacy, provided by GitHub, which is automated code analysis and quality, to help us analyze the Java source code quality.

2.1 Application description

The Alinma smart device application is your always on, anytime, anywhere banking solution, With the Alinma app you can easily and conveniently complete transactions and manage your financial life while on-the-go.

2.2 Workshop purpose

The purpose of this workshop, is to gain more knowledge about static code analysis by implementing static analysis tools on a real application and using their features.

2.3 Goals

To achieve good quality for the system in different aspects, in the design, the source code structure and security.

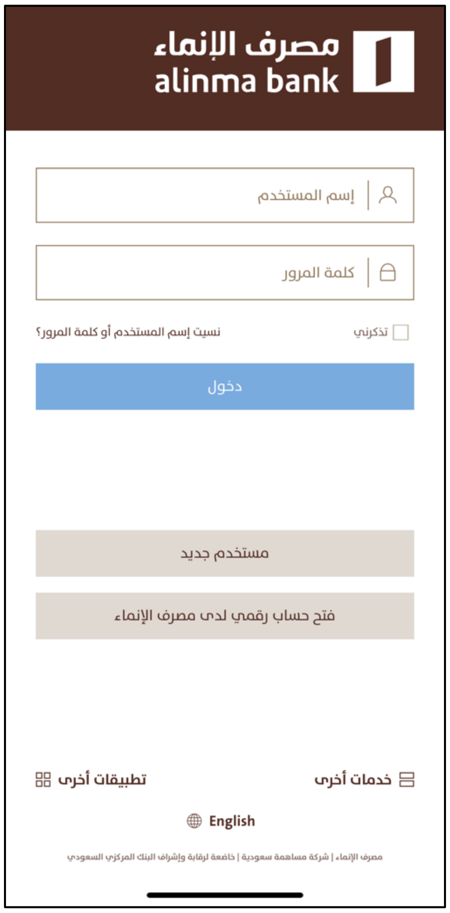
2.2 Static analysis tools

In the source code analysis section, we used Codacy tool.
In the security analysis section, we used MobSF tool.

3. Design Analysis

3.1 Graphical User Interface (GUI) Analysis

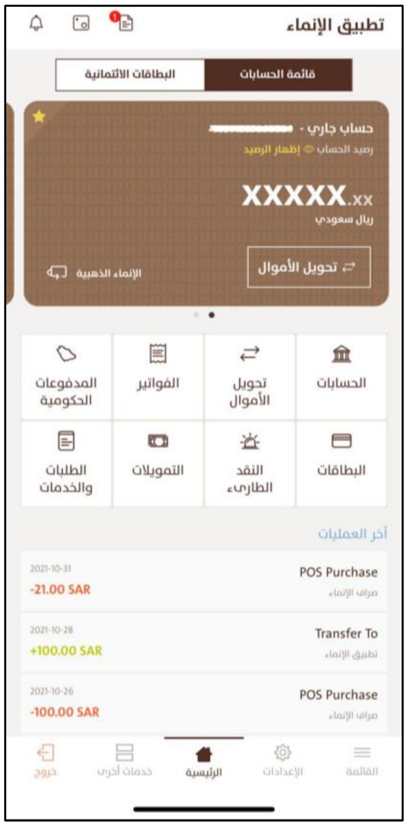
We are evaluating Alinma Bank application’s user interface design, first of, the app is designed in brown, light brown and off-white which reminds of its logo. Also, in the log in page, the icons are used to make it easier to the user to remember the information, the person icon is used for user name, and the lock for password, the app also supports both Arabic and English, which help in the system learnability and understandably.



مصرف الإنماء | شركة مساهمة سعودية | خاضعة لرقابة وإشراف البنك المركزي السعودي

Figure 1: Alinma Bank Log-in Page

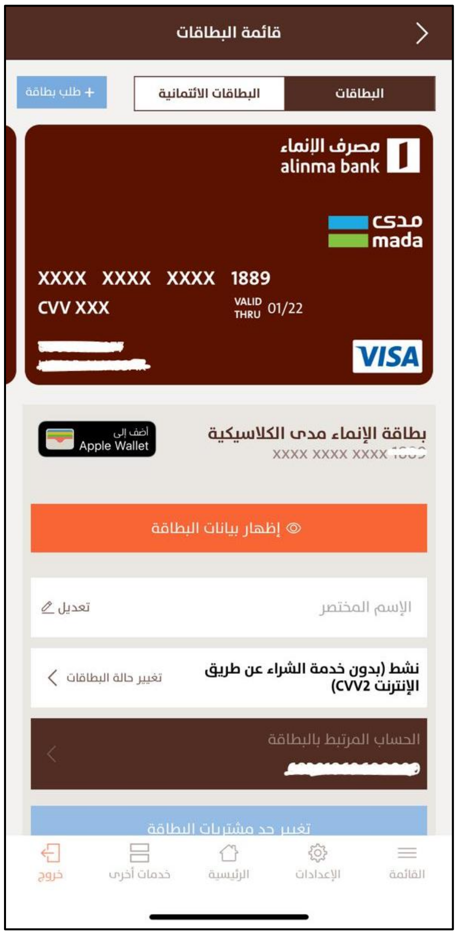
In addition, in the home page the user has the option to hide his balance which supports user's privacy. By pressing an 'eye' icon, the balance is no longer visible. Moreover, a paper icon at the app bar is used for bill's notifications, that help the user in accessing his invoice quickly and easily



POS Purchase صراف الإنماء	2021-10-31 -21.00 SAR
Transfer To تطبيق الإنماء	2021-10-28 +100.00 SAR
POS Purchase صراف الإنماء	2021-10-26 -100.00 SAR

Figure 2: Alinma home page

The app also follows “match between the system and the real world” heuristic [1], as the card information is displayed in way that matched the real credit card, that help the user in understanding and remembering each information along with its purpose.



XXXX XXXX XXXX 1889
CVV XXX
VALID THRU 01/22
[redacted]

بطاقة الإنماء مدى الكلاسيكية
XXXX XXXX XXXX 1889

الإسم المختصر

نشط (بدون خدمة الشراء عن طريق الإنترنت CVV2)

الحساب المرتبط بالبطاقة
[redacted]

Figure 3: Credit card information

In addition, the app is considered complex as there is functionalities and features that a lot of users do not know how to use them, and no help or app guidance is provided, the app should provide the user with help document so that the user has a reference any time he/she face a problem.

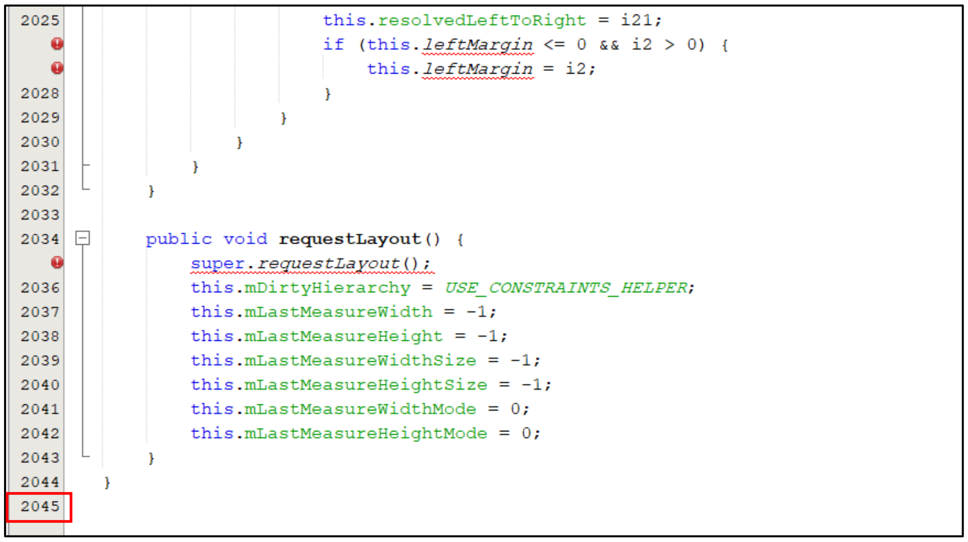
3.2 Design principles

Some of the design principles are the cohesion and coupling, when you implement them in the right way in your code, your code will be more efficient. However, we tried to implement the source code as UML diagram, but the source code was too big to handle, so we did it manually.

3.2.1 Low cohesion

We know high cohesion is that the module should not take too many responsibilities. In contrast, is the low cohesion, where the module is taking too many responsibilities.

We have found a class that is consisting of 2045 lines, which is indicates that it is taking a lot of responsibilities, we recommend to separate these responsibilities in sub-classes



```
2025                        this.resolvedLeftToRight = i21;
                            if (this.leftMargin <= 0 && i2 > 0) {
                                this.leftMargin = i2;
2028                        }
2029                    }
2030                }
2031            }
2032        }
2033
2034        public void requestLayout() {
                super.requestLayout();
2036            this.mDirtyHierarchy = USE_CONSTRAINTS_HELPER;
2037            this.mLastMeasureWidth = -1;
2038            this.mLastMeasureHeight = -1;
2039            this.mLastMeasureWidthSize = -1;
2040            this.mLastMeasureHeightSize = -1;
2041            this.mLastMeasureWidthMode = 0;
2042            this.mLastMeasureHeightMode = 0;
2043        }
2044    }
2045
```

Figure 4: ConstraintLayout class

3.2.2 High coupling

We know low coupling is that the module should not delegate responsibilities in many parts, in contrast, high coupling, where the module is delegating responsibilities in many classes, which results in which results in a high level of dependency.

We tried to search for high coupling in the code but we could not find it.

4. Manual analysis:

Before using any tools, we decided to manually analyze the source code based on what we studied.

4.1 indentation:

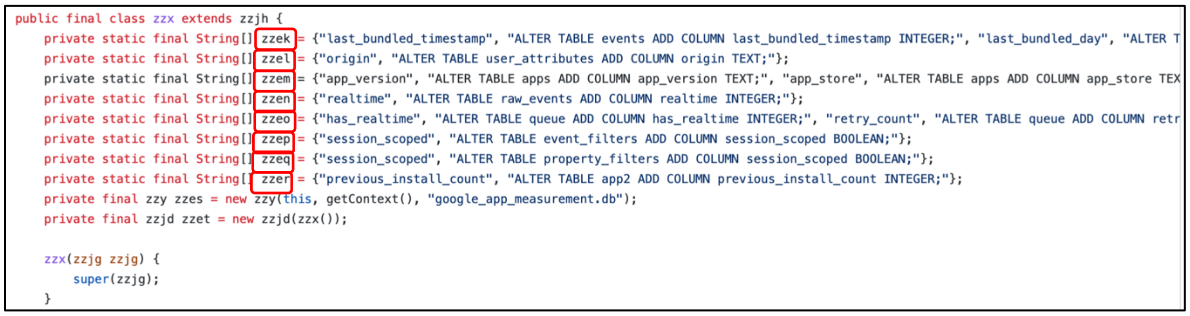
Indentation improves the structure of a program. if you indent well, you make your code readable and understandable, which will make it easy and flexible to test, debug and modification.

In Alinma source code file, we observed that they applicated intonation probably.

4.2 Identifiers:

Identifiers represent a major part of the source code; Meaningless identifiers hinder code comprehension, means program comprehension becomes harder, and negatively affect code quality.

So, every meaningless or inconsistent identifier must be renamed with a meaningful and consistent name, which will enhance the code readability and reduce comprehension efforts.
Some Identifiers name in Alinma Bank code that must be changed.



```java
public final class zzx extends zzjh {
    private static final String[] zzek = {"last_bundled_timestamp", "ALTER TABLE events ADD COLUMN last_bundled_timestamp INTEGER;", "last_bundled_day", "ALTER T
    private static final String[] zzel = {"origin", "ALTER TABLE user_attributes ADD COLUMN origin TEXT;"};
    private static final String[] zzem = {"app_version", "ALTER TABLE apps ADD COLUMN app_version TEXT;", "app_store", "ALTER TABLE apps ADD COLUMN app_store TEX
    private static final String[] zzen = {"realtime", "ALTER TABLE raw_events ADD COLUMN realtime INTEGER;"};
    private static final String[] zzeo = {"has_realtime", "ALTER TABLE queue ADD COLUMN has_realtime INTEGER;", "retry_count", "ALTER TABLE queue ADD COLUMN retr
    private static final String[] zzep = {"session_scoped", "ALTER TABLE event_filters ADD COLUMN session_scoped BOOLEAN;"};
    private static final String[] zzeq = {"session_scoped", "ALTER TABLE property_filters ADD COLUMN session_scoped BOOLEAN;"};
    private static final String[] zzer = {"previous_install_count", "ALTER TABLE app2 ADD COLUMN previous_install_count INTEGER;"};
    private final zzy zzes = new zzy(this, getContext(), "google_app_measurement.db");
    private final zzjd zzet = new zzjd(zzx());

    zzx(zzjg zzjg) {
        super(zzjg);
    }
```

Figure 5: Identifiers

4.3 Variable style:

In our study we learned that we can’t use Both variable style we must choose One!
in the source code they use Both CamelCaseIdentifiers AND underscores_identifiers, that’s Not good!
 The best is to use CamelCaseIdentifiers, Java libraries use CamelCaseIdentifiers.

4.4 Comments:

Comment making the source code easier for humans to understand, explain exactly what the Code is doing, make code maintenance much easier and reduce its costs, as well as helping make finding bugs faster.

In their code, there are no comments, which made it difficult for us to understand the code, so far, we couldn't understand anything from the code.

4.5 documentation:

Code documentation is a description that explain what a codebase does and how it can be used. It’s very important because it makes code much easier for a programmer to understand and, if necessary, debug the program.

Developer team in Alinma bank missing documentation which will make the developers spend more time remembering and debugging the program and explaining their code to others.

5. Codacy static analysis

5.1 Description

Codacy [2] is an automated code analyzer, provided by GitHub, and quality tool that help developers to enhance their code. You can use Codacy to check your code quality standard, save time in code reviews and to reduce maintenance time for the software. Codacy is using PMD and CheckStyle as bult-in features for analyzing.

PMD [3], Programming Mistake Detector, a source code analyzer, to detect common programming mistakes and flaw, for instance, unnecessary or unused code, empty fields, etc. based on rule references [4]. It supports many programming languages and one of them is java, which is what it was used in Alinma source code.

CheckStyle [5], is a development tool to enhance the code quality and finds common programming flaws as well, for instance, source code layout and format. The process of analyzing is based on some programming language standards, such as, Java Code Convention [6], which is provided by Oracle, which is what we concern in analyzing Alinma source code.

5.2 Code Style

5.2.1 Java code convention

Following a code convention, means the developers or programmers will be on the same page. These conventions have been developed and tested over the years to make programming stable, readable and easy to maintain. Codacy including PMD and CheckStyle is following the java code convention provided by Oracle, which is what we mentioned earlier, so we decided to organize some of Codacy results based on this java code convention.

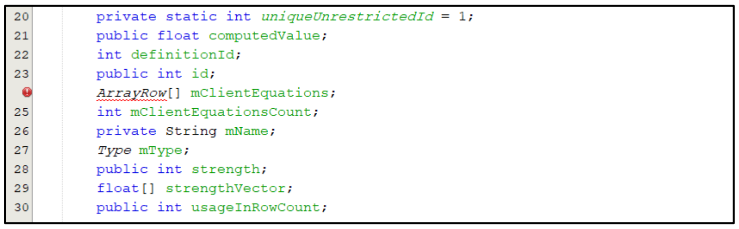
5.2.1.1 Declaration order

- Reported by: Checkstyle.
- Number of issues founded: 2794.

According to Code Conventions for the Java Programming Language, the order of declaration of the parts of class and interface is as following:

1. Static variables: First the public class variables, then protected, then package level (no access modifier), and then private.
2. Instance variables: First the public class variables, then protected, then package level (no access modifier), and then private.
3. Constructors.
4. Methods.

There are several places in the code where the order declaration is violated.
First: Variable access definition in wrong order.

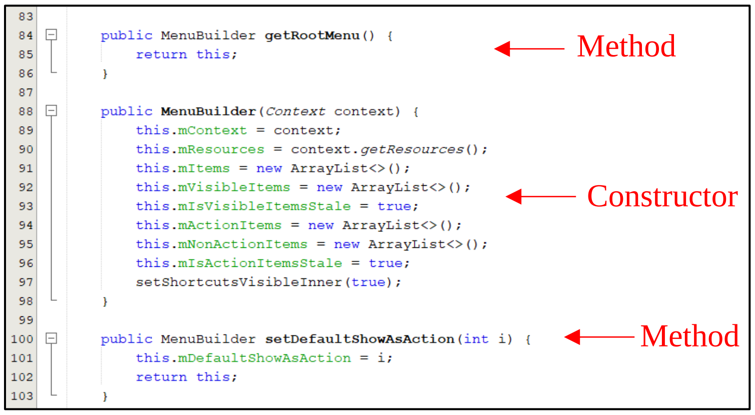


```java
    private static int uniqueUnrestrictedId = 1;
    public float computedValue;
    int definitionId;
    public int id;
    ArrayRow[] mClientEquations;
    int mClientEquationsCount;
    private String mName;
    Type mType;
    public int strength;
    float[] strengthVector;
    public int usageInRowCount;
```

Figure 6:[Location: androidx/constraintlayout/solver/SolverVariable.java (Lines: 20 to 30)]

As shown in the figure above, declaration order of variables is not taking into consideration. A messy declaration will reduce the readability of the code for sure.

Second: Constructor definition in wrong order.



```java

    public MenuBuilder getRootMenu() {
        return this;
    }

    public MenuBuilder(Context context) {
        this.mContext = context;
        this.mResources = context.getResources();
        this.mItems = new ArrayList<>();
        this.mVisibleItems = new ArrayList<>();
        this.mIsVisibleItemsStale = true;
        this.mActionItems = new ArrayList<>();
        this.mNonActionItems = new ArrayList<>();
        this.mIsActionItemsStale = true;
        setShortcutsVisibleInner(true);
    }

    public MenuBuilder setDefaultShowAsAction(int i) {
        this.mDefaultShowAsAction = i;
        return this;
    }
```

Figure 7: [Location: androidx/appcompat/view/menu/MenuBuilder.java (Line: 84 to 103)]

Similarly, as shown in figure above, there is no particular order for the constructor declaration, the constructor should be declared as it appears in the middle of some methods. This violates the java code convention; constructors must be declared before methods. It will affect the readability for sure; it could consume time to search for the constructor in the middle of many methods.

As a result of both cases, it will affect the readability and maintenance time of the code, which is not practical. 80% of the software life time is depending on maintenance.

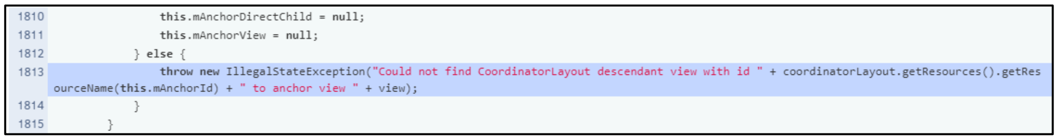
5.2.1.2 Indentation

Line Length

– Reported by: CheckStyle
– Number of issues founded: 36,213

According to java code convention, line length should noy be longer than 80 characters. After some researches, there are some particular reasons for keeping lines in code restricting to 80 characters, for example [7]:

- To help those who have a small monitor device that does not fit many characters in one line, which will make it hard to read.
- To have the ability to open multiple windows side-by-side.
- To make readability sufficient, narrow code columns would be easier and faster to read than long side-side code.

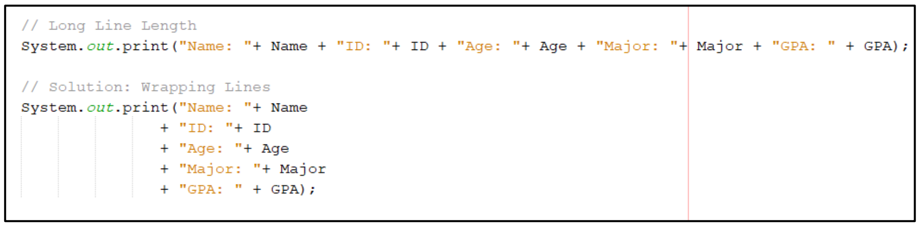


```
1810                this.mAnchorDirectChild = null;
1811                this.mAnchorView = null;
1812            } else {
1813                throw new IllegalStateException("Could not find CoordinatorLayout descendant view with id " + coordinatorLayout.getResources().getRes
ourceName(this.mAnchorId) + " to anchor view " + view);
1814            }
1815        }
```

Figure 8:[androidx/coordinatorlayout/widget/CoordinatorLayout.java (Line: 1813)]

As shown in the figure above, the line is containing 204 characters, also it does not fit the monitor screen size, which reduce the readability.

A recommended solution is wrapping the lines, illustrated in the figure bellow, reading in narrow columns is much easier than long side-by-side code, similar to newspaper and magazines, they do not write all the way across the page.



```
// Long Line Length
System.out.print("Name: "+ Name + "ID: "+ ID + "Age: "+ Age + "Major: "+ Major + "GPA: " + GPA);

// Solution: Wrapping Lines
System.out.print("Name: "+ Name
                + "ID: "+ ID
                + "Age: "+ Age
                + "Major: "+ Major
                + "GPA: " + GPA);
```

Figure 9: wrapping line example

5.2.1.3 Naming Convention

In programming there are several styles to name classes, variables, constants and methods. We know java programming commonly using CamelCase naming convention, which is true, in some extent. However, there are some places in code where another naming convention is being used.

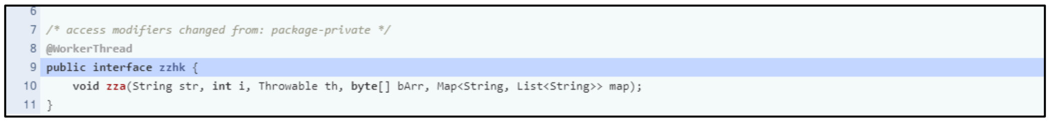
These naming conventions are listed as following:

- PascalCase: capital letter for the beginning of each word, including the first word.
- CamelCase: capital letter for the beginning of each word, excluding the first word.
- SnakeCase: all letters are capitalized, and each word is separated by an underscore.

Class Name

– Reported by: PMD (1.2)
– Number of issues founded: 2582.

In naming classes, we use PascalCase.
There appear some places where the class naming convention is violated, as shown in the figure bellow. Also, there is a lot of class that starts with letters, we do not know the meaning of these letters, but at least use the PascalCase.

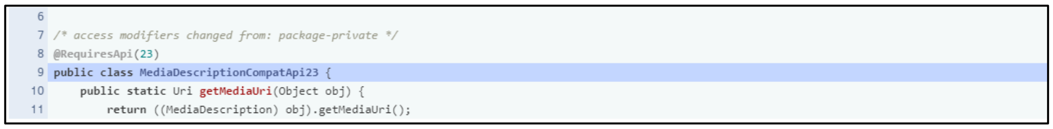


```
 6
 7 /* access modifiers changed from: package-private */
 8 @WorkerThread
 9 public interface zzhk {
10     void zza(String str, int i, Throwable th, byte[] bArr, Map<String, List<String>> map);
11 }
```

Figure 10: [com/google/android/gms/measurement/internal/zzhk.java (Line: 9)]

Regardless of the classes that starts with letters we mentioned earlier, almost most classes follow the PascalCase naming convention, which means they did great work on it.

After seeing the output of the analyzer tool, we have noticed some class that detected to violate the class naming convention, when they are actually not, so, this situation is considered as false positive, as shown in the figure bellow.



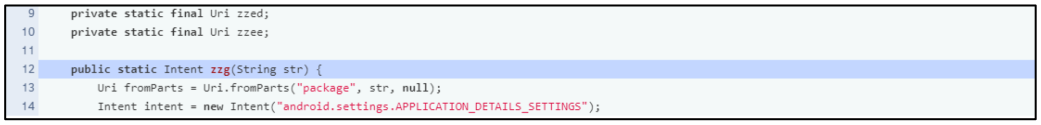
```
 6
 7 /* access modifiers changed from: package-private */
 8 @RequiresApi(23)
 9 public class MediaDescriptionCompatApi23 {
10     public static Uri getMediaUri(Object obj) {
11         return ((MediaDescription) obj).getMediaUri();
```

Figure 11: [android/support/v4/media/MediaDescriptionCompatApi23.java (Line: 9)]

Method Name

- Reported by: CheckStyle.
- Number of issues founded: 695.

In naming methods, we use CamelCase.
Similar to the previous issue, where the violation is caused by the methods that starts with random letters.



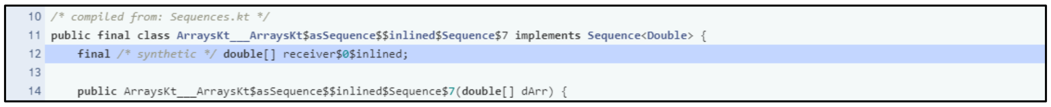
```
 9      private static final Uri zzed;
10      private static final Uri zzee;
11
12      public static Intent zzg(String str) {
13          Uri fromParts = Uri.fromParts("package", str, null);
14          Intent intent = new Intent("android.settings.APPLICATION_DETAILS_SETTINGS");
```

Figure 12: [com/google/android/gms/common/internal/zzg.java (Line: 12)]

Variable Name

- Reported by: CheckStyle.
- Number of issues founded: 345.

In naming variables (not constant), we use CamelCase.
This naming convention is being violated in few places, as shown in figure bellow.



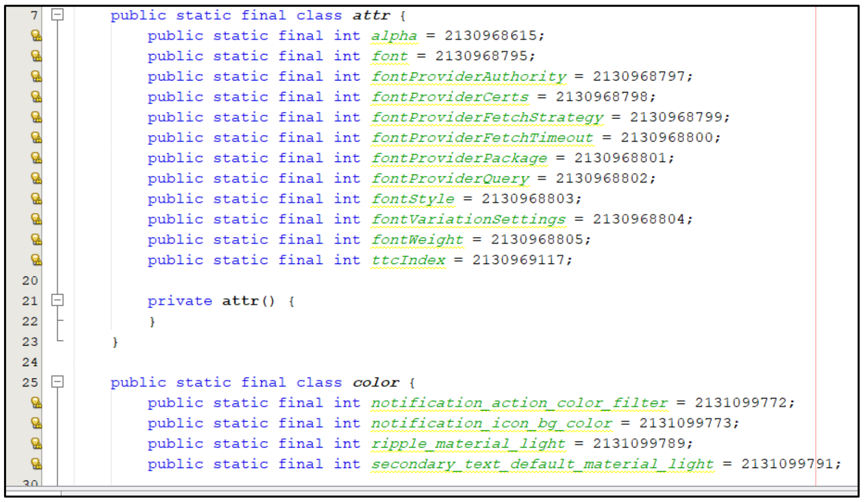
```
10 /* compiled from: Sequences.kt */
11 public final class ArraysKt___ArraysKt$asSequence$$inlined$Sequence$7 implements Sequence<Double> {
12     final /* synthetic */ double[] receiver$0$inlined;
13
14     public ArraysKt___ArraysKt$asSequence$$inlined$Sequence$7(double[] dArr) {
```

Figure 13: [kotlin/collections/ArraysKt___ArraysKt$asSequence$$inlined$Sequence$7.java (Line: 12)]

Constant Name

- Reported by: CheckStyle.
- Number of issues founded: 8,158.

In naming constants, which are declared as "`final`", we use SnakeCase.



```
 7      public static final class attr {
            public static final int alpha = 2130968615;
            public static final int font = 2130968795;
            public static final int fontProviderAuthority = 2130968797;
            public static final int fontProviderCerts = 2130968798;
            public static final int fontProviderFetchStrategy = 2130968799;
            public static final int fontProviderFetchTimeout = 2130968800;
            public static final int fontProviderPackage = 2130968801;
            public static final int fontProviderQuery = 2130968802;
            public static final int fontStyle = 2130968803;
            public static final int fontVariationSettings = 2130968804;
            public static final int fontWeight = 2130968805;
            public static final int ttcIndex = 2130969117;
20
21          private attr() {
22          }
23      }
24
25      public static final class color {
            public static final int notification_action_color_filter = 2131099772;
            public static final int notification_icon_bg_color = 2131099773;
            public static final int ripple_material_light = 2131099789;
            public static final int secondary_text_default_material_light = 2131099791;
```

Figure 14: [androidx/customview/R.java (Lines: 7 to 30)]

As shown the figure above, from line 7 to line 20, all constant variables are named using CamelCase, and from line 25 to line 30, all constant variables are named using SnakeCase (but in small letters).

They did not take constants naming into consideration, which will result in confusion and reduce the readability.

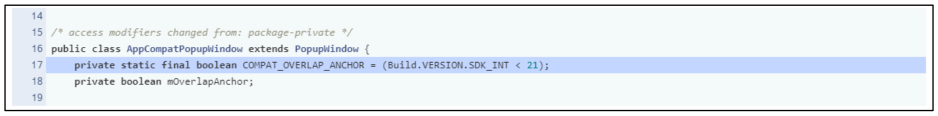
5.2.2 Unnecessary Code

5.2.2.1 Unnecessary Paratheses

- Reported by: CheckStyle.
- Number of issues founded: 462.

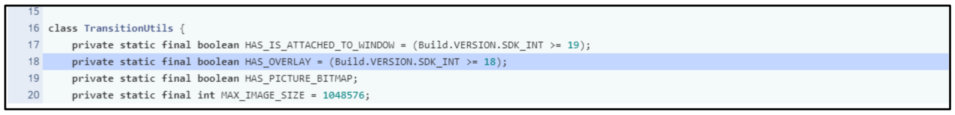
Unnecessary or useless parentheses should be removed, it could be misleading, especially while using mathematical expressions, it could be confusing with extra parentheses. After reviewing some piece of codes, there was two cases of issues detected.

First case: Unnecessary parentheses around assignment right-hand side.



```
14
15 /* access modifiers changed from: package-private */
16 public class AppCompatPopupWindow extends PopupWindow {
17     private static final boolean COMPAT_OVERLAP_ANCHOR = (Build.VERSION.SDK_INT < 21);
18     private boolean mOverlapAnchor;
19
```

Figure 15: [Location: androidx/appcompat/widget/AppCompatPopupWindow.java. (Line: 17)]



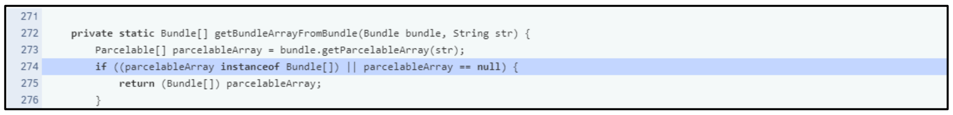
```
15
16 class TransitionUtils {
17     private static final boolean HAS_IS_ATTACHED_TO_WINDOW = (Build.VERSION.SDK_INT >= 19);
18     private static final boolean HAS_OVERLAY = (Build.VERSION.SDK_INT >= 18);
19     private static final boolean HAS_PICTURE_BITMAP;
20     private static final int MAX_IMAGE_SIZE = 1048576;
```

Figure 16: [Location: androidx/transition/TransitionUtils.java (Line: 18)]

At the right-hand side, extra paratheses were used, this code should be working perfectly without them. In every class contains this type of error, inside these parentheses there would be a boolean expression containing ('<', '>', '='). In our opinion, maybe they used them for more clarity for these expressions, or to distinguish them from the equal sign used for assignment. However, if it was a standard that is defined by the organization, it should be documented well, so other programmers, who will work on this code, will follow them.

Second case: Unnecessary parentheses around expression.

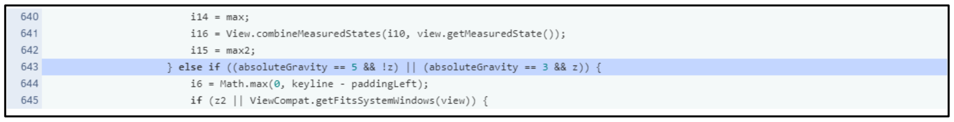
In some if-statements and return boolean values, as shown in figure below, extra parentheses were used. Also, there is a probability that these represent some other standards defined by the organization, as we explained previously.



```
271
272     private static Bundle[] getBundleArrayFromBundle(Bundle bundle, String str) {
273         Parcelable[] parcelableArray = bundle.getParcelableArray(str);
274         if ((parcelableArray instanceof Bundle[]) || parcelableArray == null) {
275             return (Bundle[]) parcelableArray;
276         }
```

Figure 17:[androidx/core/app/NotificationCompatJellybean.java (Line: 274)]

In the other hand, as shown in figure bellow, in some expressions, paratheses were necessarily needed to prioritize the execution order for the expressions. If the parentheses were not used in this case, there could be a logical error, which Codacy's features would not detect.



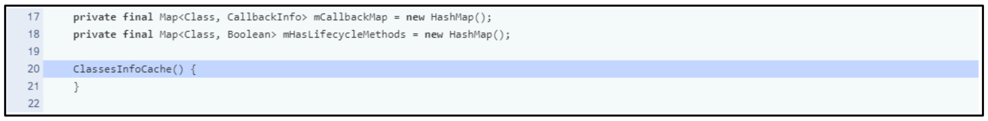
```
640                     i14 = max;
641                     i16 = View.combineMeasuredStates(i10, view.getMeasuredState());
642                     i15 = max2;
643                 } else if ((absoluteGravity == 5 && !z) || (absoluteGravity == 3 && z)) {
644                     i6 = Math.max(0, keyline - paddingLeft);
645                     if (z2 || ViewCompat.getFitsSystemWindows(view)) {
```

Figure 18: [androidx/coordinatorlayout/widget/CoordinatorLayout.java (Line: 643)]

5.2.2.2 Unnecessary Constructors

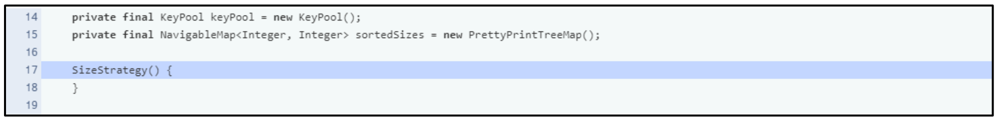
- Reported by: PMD (1.0).
- Number of issues founded: 150.

Avoid unnecessary constructors, they will be auto-generated by the compiler. According to our java language knowledge, using an empty constructor is necessary if you are declaring an overloaded constructor(s) (same constructor name with different parameter structure). Writing empty body constructor However, maybe writing them in the code would avoid any error could happen in the future.



```
17      private final Map<Class, CallbackInfo> mCallbackMap = new HashMap();
18      private final Map<Class, Boolean> mHasLifecycleMethods = new HashMap();
19
20      ClassesInfoCache() {
21      }
22
```

Figure 19:[androidx/lifecycle/ClassesInfoCache.java (Line: 20)]



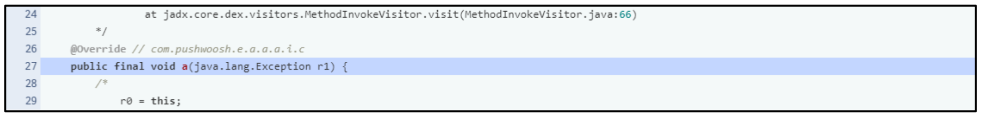
```
14      private final KeyPool keyPool = new KeyPool();
15      private final NavigableMap<Integer, Integer> sortedSizes = new PrettyPrintTreeMap();
16
17      SizeStrategy() {
18      }
19
```

Figure 20: [com/bumptech/glide/load/engine/bitmap_recycle/SizeStrategy.java (Line:100)]

5.2.2.3 Unnecessary Fully Qualified Name

- Reported by: PMD (5.0).
- Number of issues founded: 117.

Writing fully qualified names instead of using implicit import command is considered poor style. However, after some researches, there appear to be two cases where to use a fully qualified name is recommended or required.
First case: using only one instance of class.
when an instance of a particular class is used only once. As shown in figure bellow, in this class, the `java.lang.Excpetion` is used only once. As a result, it is not necessarily to import a whole `java.lang` package just for one instance.



```
24                  at jadx.core.dex.visitors.MethodInvokeVisitor.visit(MethodInvokeVisitor.java:66)
25          */
26      @Override // com.pushwoosh.e.a.a.a.i.c
27      public final void a(java.lang.Exception r1) {
28          /*
29              r0 = this;
```

Figure 21: [com/pushwoosh/internal/platform/prefs/ . $$Lambda$a$K3BMiNmSheWPkotsQxmww9qJo3I.java (Line: 27)]

Second case: identical classes in same package.
When there are identical class names in the same package. In this case, using a fully qualified names are obligated to avoid collisions and ambiguousness. This case is represented in many places in the code, especially using `com.pushwoosh` package. This package is beyond our knowledge in Java coding, we tried to search about it but didn't find enough information.

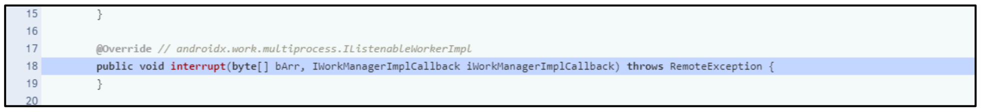
In conclusion, we considered this issue as a false positive. Because, using fully qualified names, in the second case, is required to distinguish between identical classes, for instance. Also, according to the number of issues founded, which is few, indicate that there no use of unnecessary fully qualified names in other piece of code. They were only used in the two cases described preciously.

5.2.3 Empty Code Fields

5.2.3.1 Uncommented Empty Method Body

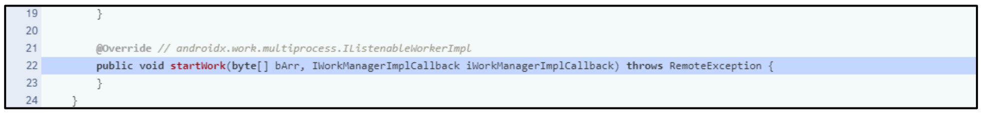
– Reported by: PMD (3.4)
– Number of issues founded: 716

After reading some code files related to this issue, there is a common link between them, some classes having this issue is implementing an *interface*. In java programming, if a class is implementing an interface, it must override all of the interface methods, which means they must be written inside the class. In some situations, a programmer needs some, not all, functionalities of an interface, but it is enforced to implement all of them, according to java language rules.



```
15          }
16
17          @Override // androidx.work.multiprocess.IListenableWorkerImpl
18          public void interrupt(byte[] bArr, IWorkManagerImplCallback iWorkManagerImplCallback) throws RemoteException {
19          }
20
```

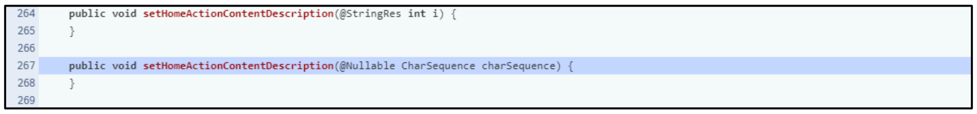
Figure 22: [androidx/work/multiprocess/IListenableWorkerImpl.java (Line: 18)]



```
19          }
20
21          @Override // androidx.work.multiprocess.IListenableWorkerImpl
22          public void startWork(byte[] bArr, IWorkManagerImplCallback iWorkManagerImplCallback) throws RemoteException {
23          }
24      }
```

Figure 23: [androidx/work/multiprocess/IListenableWorkerImpl.java (Line: 22)]

As shown in both figures above, the highlighted methods have an empty body. Above each method, there is an `@Override` notation followed by the *interface* file location, where these methods are taken from. In our opinion, this is a sufficiently good documentation.



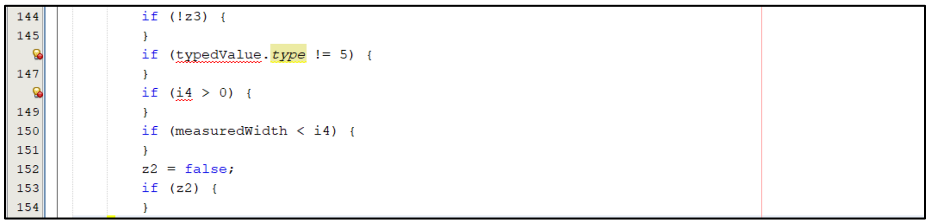
```
264     public void setHomeActionContentDescription(@StringRes int i) {
265     }
266
267     public void setHomeActionContentDescription(@Nullable CharSequence charSequence) {
268     }
269
```

Figure 24: [androidx/appcompat/app/ActionBar.java (Line: 267)]

In contrast, as shown in figure above, there are some empty methods without any documentation, we could not decide whether is it because of an interface implementation, or is there is other reason. Whatsoever reason, it should be documented comprehensively, similarly to figure described previously.

5.2.3.2 Empty 'if' statement

- Reported by: PMD (0.1)
- Number of issues founded: 238



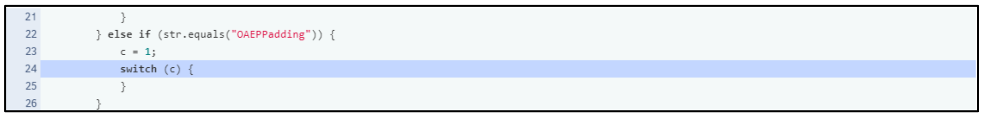
```
144             if (!z3) {
145             }
                if (typedValue.type != 5) {
147             }
                if (i4 > 0) {
149             }
150             if (measuredWidth < i4) {
151             }
152             z2 = false;
153             if (z2) {
154             }
```

Figure 25: [androidx/appcompat/widget/ContentFrameLayout.java (Line: 144 to 154)]

As shown in figure above, there are several empty if statements. If it was added explicitly to tell the programmer that there is nothing going to execute in these statements, it should be commented well.

5.2.3.23Empty 'switch' statement

- Reported by: PMD (1.0)
- Number of issues founded: 9



```
21              }
22          } else if (str.equals("OAEPPadding")) {
23              c = 1;
24              switch (c) {
25              }
26          }
```

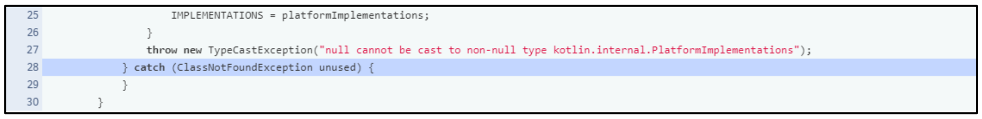
Figure 26: [androidx/appcompat/widget/DrawableUtils.java (Line: 24)]

There is no benefit of using a switch statement with an empty body, similar to the empty if-statement case. Based on the number of issues founded here, it is quite fewer. If there is a reason of writing them, it should be documented as well.

5.2.3.4 Empty 'catch' block

- Reported by: CheckStyle.
- Number of issues founded: 231

An empty catch block is similar to saying "There is an error has happened, I don’t know what is it, and I am just going to ignore it". This is considered as a bad coding, if we are going to ignore the error, why it has to be catch?
In some situations, programmers do not know how to handle the error, or it is out of their capabilities, whatsoever reason, it should be documented well by providing comments.



```
25                  IMPLEMENTATIONS = platformImplementations;
26              }
27              throw new TypeCastException("null cannot be cast to non-null type kotlin.internal.PlatformImplementations");
28          } catch (ClassNotFoundException unused) {
29          }
30      }
```

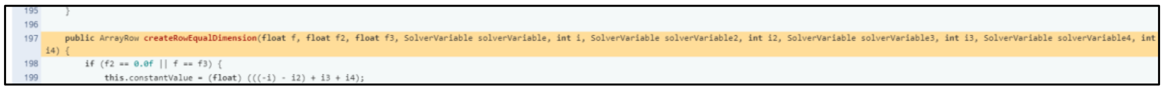
Figure 27: [kotlin/internal/PlatformImplementationsKt.java (Line: 28)]

5.3 Error Prone

5.3.1 Avoid long parameter lists

- Reported By: PMD (1.0)
- Number of issues founded: 76

Methods with numerous parameters are a challenge to maintain, especially if most of them share the same datatype. You can use new objects to wrap the numerous parameters or use proper data structures.



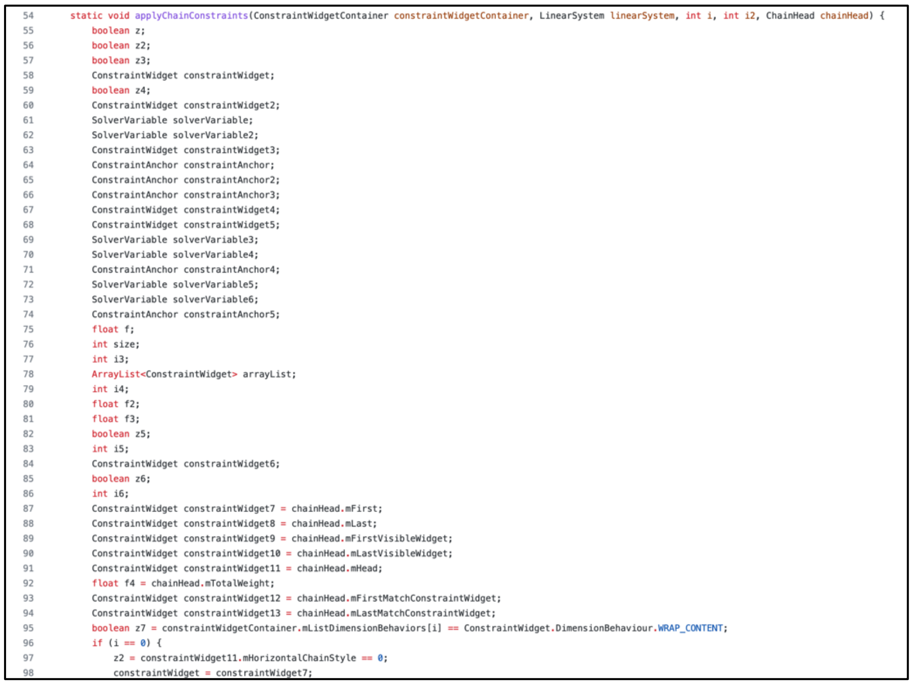
```
195     }
196
197     public ArrayRow createRowEqualDimension(float f, float f2, float f3, SolverVariable solverVariable, int i, SolverVariable solverVariable2, int i2, SolverVariable solverVariable3, int i3, SolverVariable solverVariable4, int
    i4) {
198         if (f2 == 0.0f || f == f3) {
199             this.constantValue = (float) (((-i) - i2) + i3 + i4);
```

Figure 28: [androidx/appcompat/widget/DrawableUtils.java (Line: 197)]

5.3.2 Avoid really long methods

- Reported By: CheckStyle.
- Number of issues founded: 44

Method `applyChainConstraints` length is 376 lines, if a method becomes very long it is hard to understand. Therefore, long methods should usually be refactored into several individual methods that focus on a specific task. **(SRP).**



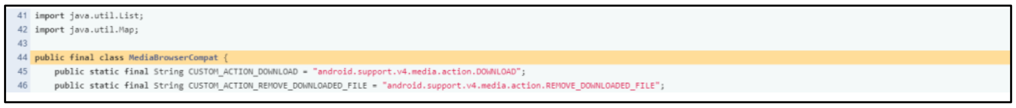
```java
54      static void applyChainConstraints(ConstraintWidgetContainer constraintWidgetContainer, LinearSystem linearSystem, int i, int i2, ChainHead chainHead) {
55          boolean z;
56          boolean z2;
57          boolean z3;
58          ConstraintWidget constraintWidget;
59          boolean z4;
60          ConstraintWidget constraintWidget2;
61          SolverVariable solverVariable;
62          SolverVariable solverVariable2;
63          ConstraintWidget constraintWidget3;
64          ConstraintAnchor constraintAnchor;
65          ConstraintAnchor constraintAnchor2;
66          ConstraintAnchor constraintAnchor3;
67          ConstraintWidget constraintWidget4;
68          ConstraintWidget constraintWidget5;
69          SolverVariable solverVariable3;
70          SolverVariable solverVariable4;
71          ConstraintAnchor constraintAnchor4;
72          SolverVariable solverVariable5;
73          SolverVariable solverVariable6;
74          ConstraintAnchor constraintAnchor5;
75          float f;
76          int size;
77          int i3;
78          ArrayList<ConstraintWidget> arrayList;
79          int i4;
80          float f2;
81          float f3;
82          boolean z5;
83          int i5;
84          ConstraintWidget constraintWidget6;
85          boolean z6;
86          int i6;
87          ConstraintWidget constraintWidget7 = chainHead.mFirst;
88          ConstraintWidget constraintWidget8 = chainHead.mLast;
89          ConstraintWidget constraintWidget9 = chainHead.mFirstVisibleWidget;
90          ConstraintWidget constraintWidget10 = chainHead.mLastVisibleWidget;
91          ConstraintWidget constraintWidget11 = chainHead.mHead;
92          float f4 = chainHead.mTotalWeight;
93          ConstraintWidget constraintWidget12 = chainHead.mFirstMatchConstraintWidget;
94          ConstraintWidget constraintWidget13 = chainHead.mLastMatchConstraintWidget;
95          boolean z7 = constraintWidgetContainer.mListDimensionBehaviors[i] == ConstraintWidget.DimensionBehaviour.WRAP_CONTENT;
96          if (i == 0) {
97              z2 = constraintWidget11.mHorizontalChainStyle == 0;
98              constraintWidget = constraintWidget7;
```

Figure 29: [androidx/constraintlayout/solver/widgets/Chain.java (Line:54)]

5.3.3 Avoid really long classes

- Reported By: PMD (1.0)
- Number of issues founded: 9

Excessive class file lengths are usually indications that the class may be burdened with excessive responsibilities that could be provided by external classes or functions. In breaking these methods apart the code becomes more manageable and ripe for reuse.



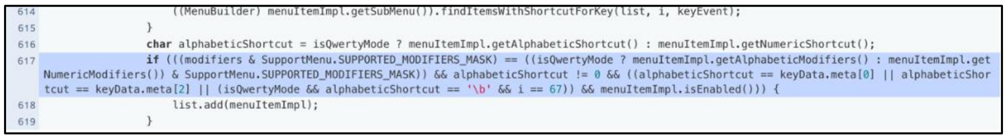
```java
41 import java.util.List;
42 import java.util.Map;
43
44 public final class MediaBrowserCompat {
45     public static final String CUSTOM_ACTION_DOWNLOAD = "android.support.v4.media.action.DOWNLOAD";
46     public static final String CUSTOM_ACTION_REMOVE_DOWNLOADED_FILE = "android.support.v4.media.action.REMOVE_DOWNLOADED_FILE";
```

Figure 30: [androidx/appcompat/widget/DrawableUtils.java (Line: 44)]

5.3.4 Boolean Expression Complexity

- Reported by: Checkstyle
- Number of issues founded: 149.

Too many conditions lead to code that is difficult to read and hence debug and maintain.



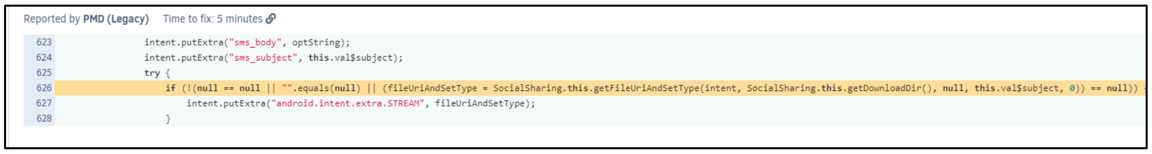
```
614                     ((MenuBuilder) menuItemImpl.getSubMenu()).findItemsWithShortcutForKey(list, i, keyEvent);
615                 }
616                 char alphabeticShortcut = isQwertyMode ? menuItemImpl.getAlphabeticShortcut() : menuItemImpl.getNumericShortcut();
617                 if (((modifiers & SupportMenu.SUPPORTED_MODIFIERS_MASK) == ((isQwertyMode ? menuItemImpl.getAlphabeticModifiers() : menuItemImpl.get
NumericModifiers()) & SupportMenu.SUPPORTED_MODIFIERS_MASK)) && alphabeticShortcut != 0 && ((alphabeticShortcut == keyData.meta[0] || alphabeticShor
tcut == keyData.meta[2] || (isQwertyMode && alphabeticShortcut == '\b' && i == 67)) && menuItemImpl.isEnabled())) {
618                     list.add(menuItemImpl);
619                 }
```

Figure 31: [androidx/appcompat/view/menu/MenuBuilder.java (Line:617)]

5.3.5 Avoid Multiple Unary Operators

- Reported By: PMD (1.0)
- Number of issues founded: 9

The use of multiple unary operators is confusing and may lead to bugs. consider simplifying the expression.



Reported by **PMD (Legacy)**   Time to fix: 5 minutes

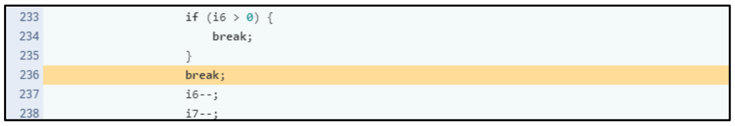
```
623                 intent.putExtra("sms_body", optString);
624                 intent.putExtra("sms_subject", this.val$subject);
625                 try {
626                     if (!(null == null || "".equals(null) || (fileUriAndSetType = SocialSharing.this.getFileUriAndSetType(intent, SocialSharing.this.getDownloadDir(), null, this.val$subject, 0)) == null))
627                         intent.putExtra("android.intent.extra.STREAM", fileUriAndSetType);
628                     }
```

Figure 32: [androidx/appcompat/widget/DrawableUtils.java (Line: 626)]

5.3.6 Avoid using a branching statement as the last in a loop

- Reported By: PMD (1.0)
- Number of issues founded: 9

Using a branching statement as the last part of a loop is confusing and may be lead to bug. Ensure that the usage is not a bug and consider using another approach.



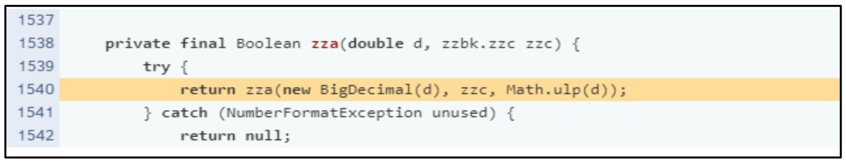
```
233                 if (i6 > 0) {
234                     break;
235                 }
236                 break;
237                 i6--;
238                 i7--;
```

Figure 33: [androidx/appcompat/widget/DrawableUtils.java (Line: 236)]

5.3.7 Avoid creating BigDecimal with a decimal (float/double) literal

- Reported By: PMD (1.0)
- Number of issues founded: 9

Use a String literal.
“One might assume that the result of "new BigDecimal(0.1)" is exactly equal to 0.1, but it is actually equal to .1000000000000000055511151. This is because 0.1 cannot be represented exactly as a double (or as a binary fraction of any finite length). Thus, the long value that is being passed in to the constructor is not exactly equal to 0.1, appearances notwithstanding.
The (String) constructor, on the other hand, is perfectly predictable: 'new BigDecimal("0.1")' is exactly equal to 0.1, as one would expect. Therefore, it is generally recommended that the (String) constructor be used in preference to this one.” –[8]



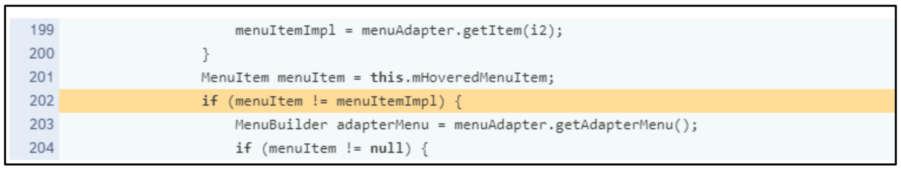
```

    private final Boolean zza(double d, zzbk.zzc zzc) {
        try {
            return zza(new BigDecimal(d), zzc, Math.ulp(d));
        } catch (NumberFormatException unused) {
            return null;
```

Figure 34: [androidx/appcompat/widget/DrawableUtils.java (Line: 2540)]

5.3.8 avoid using “==” to compare object references

- Reported By: PMD (3.2)
- Number of issues founded: 128

Use `equals()` instead.
Since comparing objects with named constants is useful in some cases (eg, when defining constants for sentinel values), the rule ignores comparisons against fields with all-caps name (eg `this == SENTINEL`), which is a common naming convention for constant fields.You may allow some types to be compared by reference by listing the exceptions in
the `typesThatCompareByReference` property.



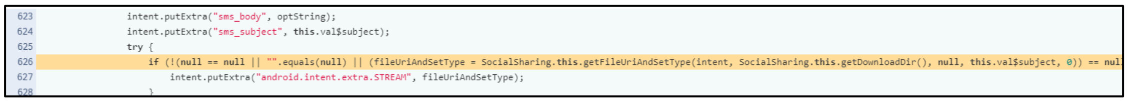
```
199                     menuItemImpl = menuAdapter.getItem(i2);
200                 }
201                 MenuItem menuItem = this.mHoveredMenuItem;
202                 if (menuItem != menuItemImpl) {
203                     MenuBuilder adapterMenu = menuAdapter.getAdapterMenu();
204                     if (menuItem != null) {
```

Figure 35: [androidx/appcompat/widget/MenuPopupWindow.java (Line: 202)]

5.3.9 Avoid using equals() to compare against null

- Reported By: PMD (1.9)
- Number of issues founded: 1

Tests for null should not use the equals() method. The '==' operator should be used instead.



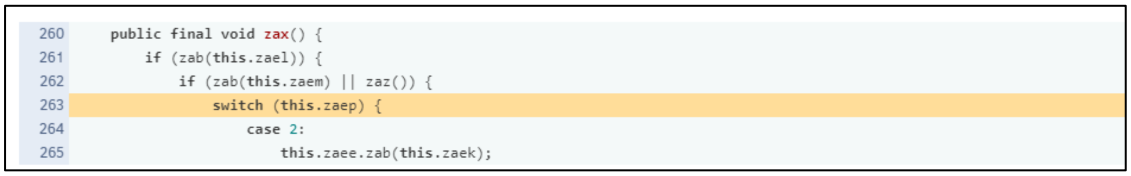
```
623             intent.putExtra("sms_body", optString);
624             intent.putExtra("sms_subject", this.val$subject);
625             try {
626                 if (!(null == null || "".equals(null) || (fileUriAndSetType = SocialSharing.this.getFileUriAndSetType(intent, SocialSharing.this.getDownloadDir(), null, this.val$subject, 0)) == nul
627                     intent.putExtra("android.intent.extra.STREAM", fileUriAndSetType);
628                 }
```

Figure 36: [nl/xservices/plugins/SocialSharing.java (Line:626)]

5.3.10 A switch statement does not contain a break

- Reported By: PMD (1.0)
- Number of issues founded: 9

Switch statements without break or return statements for each case may lead to problem.



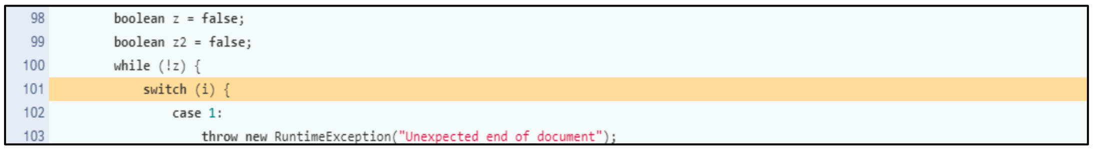
```
260     public final void zax() {
261         if (zab(this.zael)) {
262             if (zab(this.zaem) || zaz()) {
263                 switch (this.zaep) {
264                     case 2:
265                         this.zaee.zab(this.zaek);
```

Figure 37: [com/google/android/gms/common/api/internal/zas.java (Line:263)]

5.3.11 Switch statements should have a default label

- Reported By: PMD (1.0)
- Number of issues founded: 79

All switch statements should include a default option to catch any unspecified values.



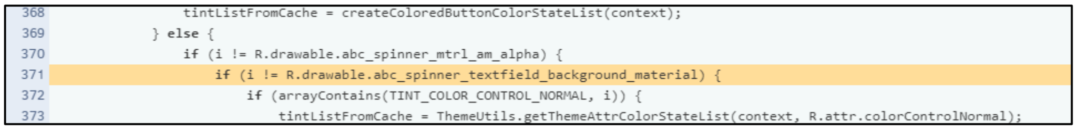
```
        boolean z = false;
        boolean z2 = false;
        while (!z) {
            switch (i) {
                case 1:
                    throw new RuntimeException("Unexpected end of document");
```

Figure 38: [androidx/appcompat/view/SupportMenuInflater.java (Line:101)]

5.3.12 These nested if statements could be combined

- Reported By: PMD (3.1)
- Number of issues founded: 110

Sometimes two consecutives 'if' statements can be consolidated by separating their conditions with a boolean short-circuit operator.

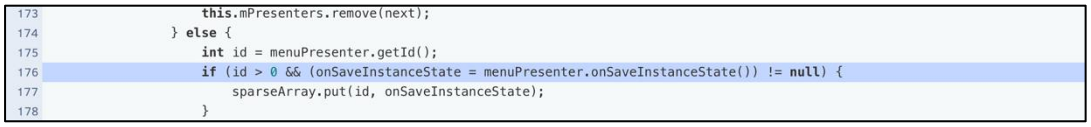


```
                tintListFromCache = createColoredButtonColorStateList(context);
            } else {
                if (i != R.drawable.abc_spinner_mtrl_am_alpha) {
                    if (i != R.drawable.abc_spinner_textfield_background_material) {
                        if (arrayContains(TINT_COLOR_CONTROL_NORMAL, i)) {
                            tintListFromCache = ThemeUtils.getThemeAttrColorStateList(context, R.attr.colorControlNormal);
```

Figure 39: [androidx/appcompat/widget/AppCompatDrawableManager.java (Line: 371)]

5.3.13 Inner assignments

- Reported By: CheckStyle.
- Number of issues founded: 599.



```
                    this.mPresenters.remove(next);
                } else {
                    int id = menuPresenter.getId();
                    if (id > 0 && (onSaveInstanceState = menuPresenter.onSaveInstanceState()) != null) {
                        sparseArray.put(id, onSaveInstanceState);
                    }
```

Figure 40: [androidx/appcompat/view/menu/MenuBuilder.java (Line: 176)]

The inner assignment is harder to read and easier to miss. In a complex condition it can even be missed, and can cause error, like the one given above if `id > 0` is false, the `onSaveInstanceState` variable will not have value later on, so Inner assignments should be avoided, all assignments should occur in their own top-level statement to increase readability.

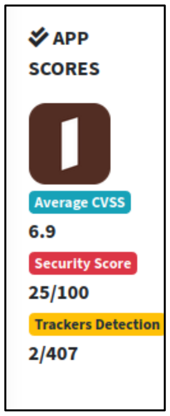
6. MobSF Security analysis

For critical systems the handle sensitive information, there are several attributes must take into considerations, the most important attribute is the security. As a bank app, the security is playing a big part of the app, as it is storing financial information, and to not lose customers' reliability, the system must be as secure as possible.

There are some tools are used to provide analysis for security, depending on some world-wide standards. One of these tools is the MobSF.

An overall app score, as shown in figure in the right, have three attributes:

- Average CVSS: 6.9
  CVSS, stands for Common Vulnerability Scoring System, we will discuss that later in the 4.3 Code Analysis section. As presented, the CVSS score is 6.9, which refer to medium severity ranking.
- Security Score: 25/100.
  An overall security score is highlighted in red, which is extremely low.


APP SCORES

Average CVSS
6.9
Security Score
25/100
Trackers Detection
2/407

Figure 41: MobSF Security App Score

As a bank app, this is really dangerous issue, there are some places in the source code or in the AndroidMainfest.xml files, where appear some risks could happen, with varying severity levels, according to some standards and metrics, which is what we will describe and discuss in this section.

6.1 Application Permissions

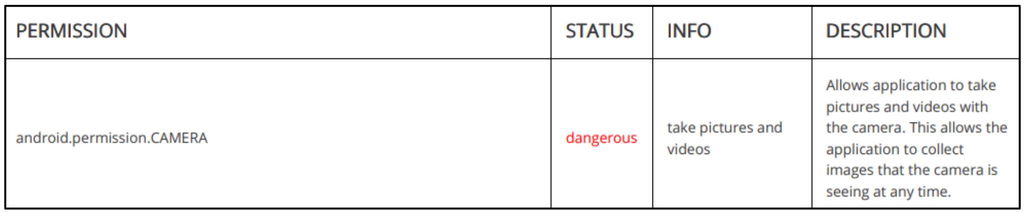
6.1.1 Application Permissions section description

permissions are what an app is allowed to do and access. This ranges from reading the data stored on your phone, such as contacts and media files, through to using hardware, including your handset's camera or microphone. Granting permission, allows the app to use the feature. In contract, denying access will prevents it from doing so.

6.1.2 Application permissions analysis

In the application permission file, there appear some permissions that their status is considered dangerous, these permissions are:

Permission no.1:



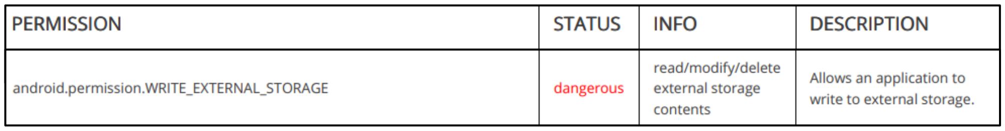
PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.

Figure 42: application permissions no.1

Permission description:
android.permission.CAMERA: allow the application to access the device's camera, for taking photos, recording videos, or streaming video.

Why is it considered dangerous?
it is considered dangerous because the app have access in camera, and also the photo gallery which have private information.

Permission no.2:



PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.

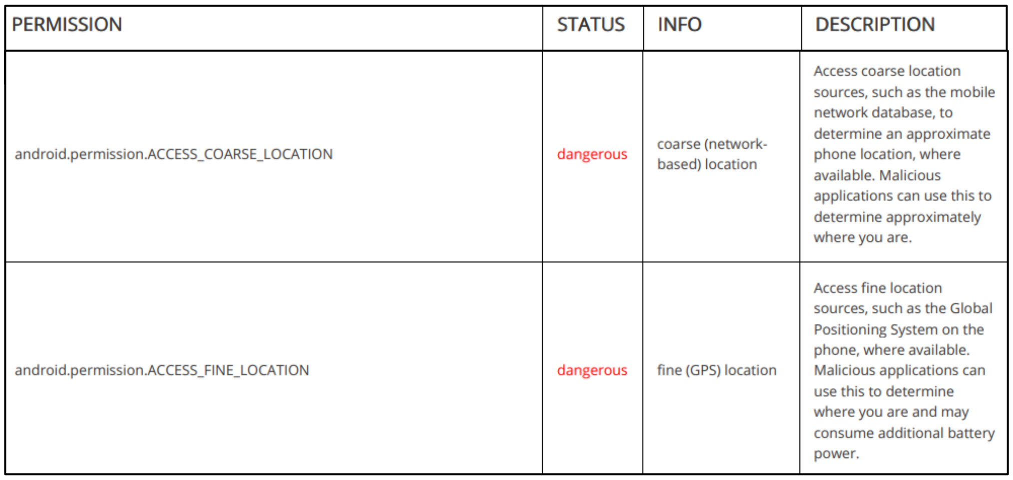
Figure 43: application permissions no.2

Permission description:
android.permission.WRITE_EXTERNAL_STORAGE: allow the app to access external storage of the device and write (store) data into it.

Why is it considered dangerous?

Writing in external storage means it will be accessible by other apps, which will reduce the security of private information if they are stored in external storage. More details about external storage in section (6.3.2 Code analysis issues, ISSUE no.8).

Permission no.3 and no.4:



PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.

Figure 44: application permission no3 and no.4

Permission description:
Both permissions are concerned on accessing the device location. Each one of them has its own way to get the location, as illustrated in the description section.

Why is it considered dangerous?

It is considered dangerous because it allows the app to track down your exact location and some apps sell this information without the user's permission and violate their privacy.
Or maybe the hacker may access this information, which threatens the safety of the user.

6.2 Manifest Analysis

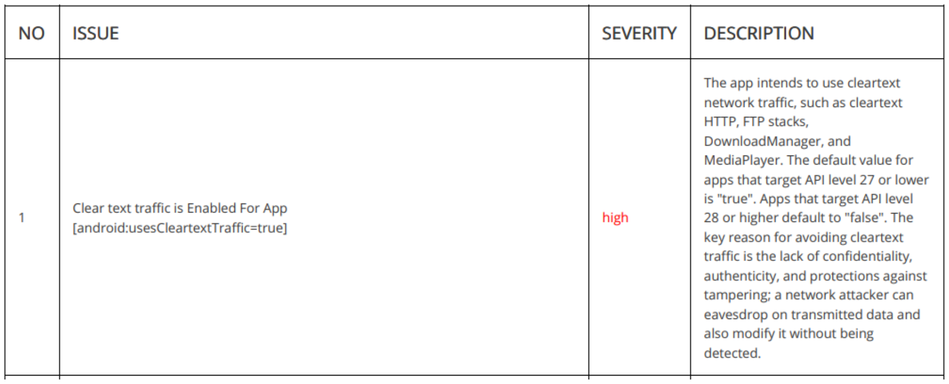
6.2.1 Manifest Analysis section description

Every app in Android has a AndroidManifest.xml, which includes descriptive information about the application, such as Activities, Permissions, Broadcast Receiver and Content Provider.

MobSF provide analysis for this file, and assigning severity ranking for each issue. From the manifest analysis.

6.2.2 Manifest Analysis issues

ISSUE: Cleartext network traffic



NO	ISSUE	SEVERITY	DESCRIPTION
1	Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]	high	The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". 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 also modify it without being detected.

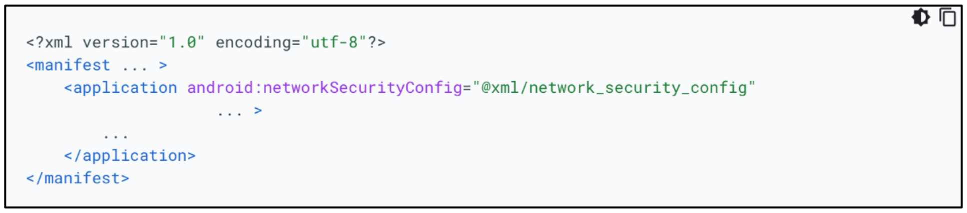
Figure 45: Mainfest analysis, Cleartext network traffic

Alinma app communicates with servers using a cleartext network traffic, which lacks confidentiality, authenticity, and protections against tampering. Also, it could arise a risk of eavesdropping and tampering content and modifying it without being detected, this is a big risk. they must prevent.

This risk can be prevented either by:

– editing useCleartextTraffic attribute in the AndroidManifest.xml file to set the value to false, which will refuse the app's requests to use cleartext traffic.
– Or by add a Network Security Configuration file with cleartextTrafficPermitted attribute with "false" value, include an entry in the AndroidManifest file to point to this file,

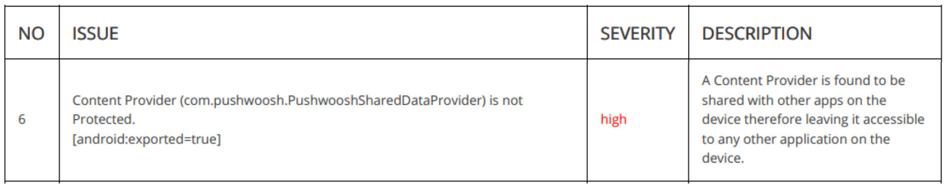
following code excerpt from a AndroidManifest how to create this entry:



```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

Figure 46: excerpt from a AndroidManifest.xml file

ISSUE: Content Provider



NO	ISSUE	SEVERITY	DESCRIPTION
6	Content Provider (com.pushwoosh.PushwooshSharedDataProvider) is not Protected. [android:exported=true]	high	A Content Provider is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

Figure 47: Mainfest analysis, content provider.

One content provider (com.pushwoosh.PushwooshSharedDataProvider) registered in the AndroidManifest file is exported to be accessed by all other apps, which can cause an issues like **Information leakage** and **SQL injection**.

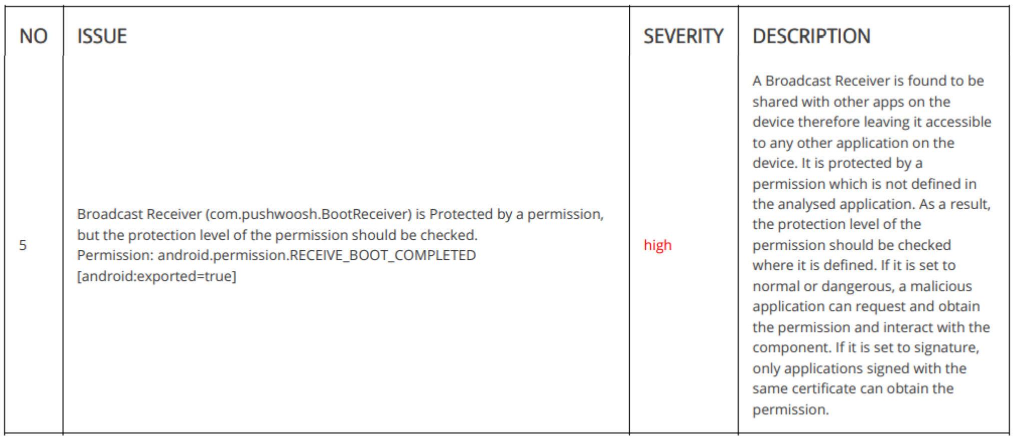
Information leakage, allows an application to reveal sensitive data such as technical details of the application and its infrastructure, developer comments, environment, or user-specific data (usernames and password, financial information or credit card details), so attackers can query the content provider, and take all sensitive info with the details stored into the app’s DB [9]. And also, they may use this sensitive data to exploit the target application, its hosting network, or its user.

SQL injection, is a type of cyber-attack in which attacker uses a piece of SQL (Structured Query Language) code to manipulate a database and gain access to potentially valuable information [10].

accessing the bank databases, cause issues such as voiding transactions or changing balances, the attacker gaining administrative rights to a database, which will lead loss of customer trust should personal information such as phone numbers, addresses, and credit card details be stolen.

To be more protected they can limit access with custom permissions, temporary permission, path-level permission, which make access to each content URI individually, read permission, which make other apps to be able to read the contents of a Content Provider but not change them, write permission, which allow an app to add a new entry to our database, but not to be able to read what is already in there, another more secure solution to set `android:exported` attribute's value to `false` which prevent other applications to access their content provider[9].

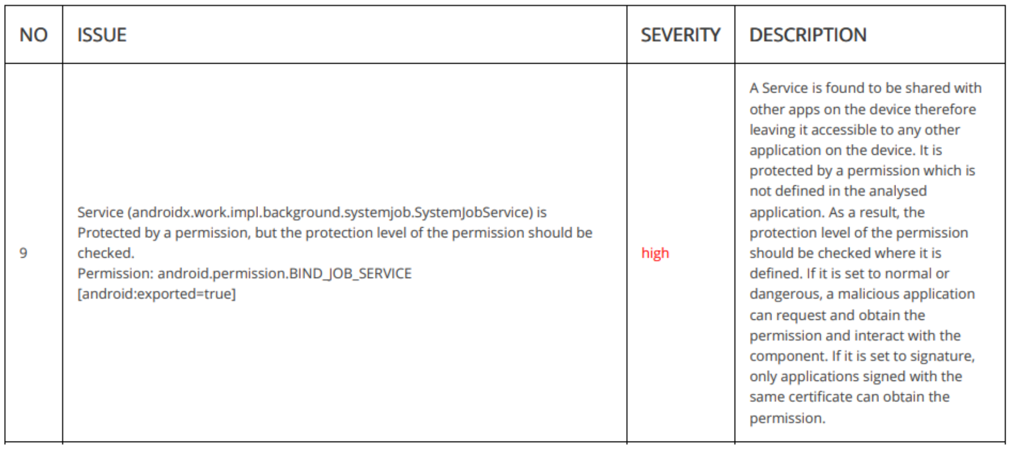
ISSUE: Broadcast Receiver



NO	ISSUE	SEVERITY	DESCRIPTION
5	Broadcast Receiver (com.pushwoosh.BootReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.RECEIVE_BOOT_COMPLETED [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Figure 48: Broadcast receiver, issue no.5

Issue no.5: Broadcast Receiver (com.pushwoosh.BootReceiver) is Protected by a permission RECEIVE_BOOT_COMPLETED with "normal" Protection level which is lower-risk permission.

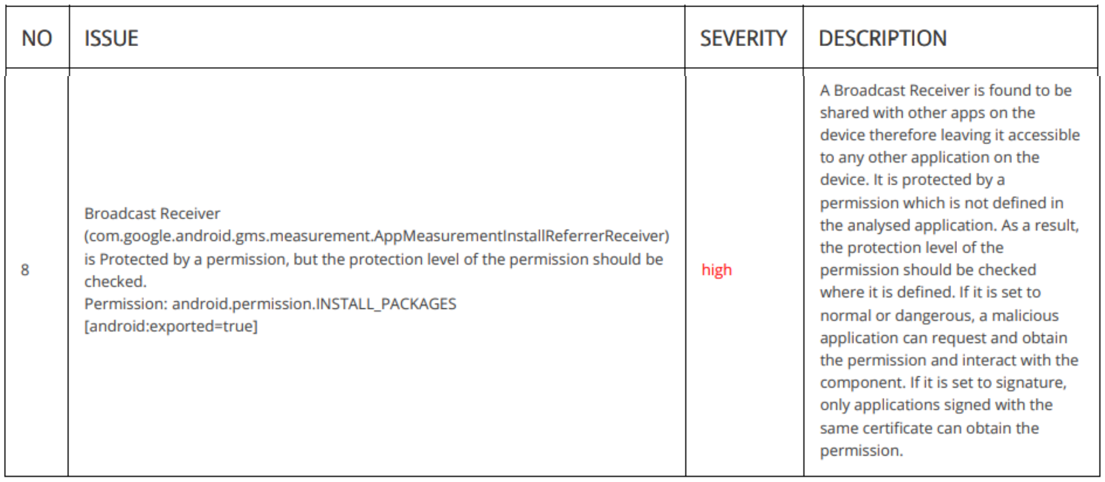


NO	ISSUE	SEVERITY	DESCRIPTION
9	Service (androidx.work.impl.background.systemjob.SystemJobService) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.BIND_JOB_SERVICE [android:exported=true]	high	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Figure 49: Broadcast receiver, issue no.9

Issue no.9: Service (androidx.work.impl.background.systemjob.SystemJobService) is protected by a permission BIND_JOB_SERVICE with "signature" protection level, this mean that only the JobScheduler is able to run the JobService.[11].

Moreover, a signature -level permission means that the app defending itself with that permission (e.g., via android:permission attributes) and the app trying to talk to the first app that needs the permission (element) must be signed by the same signing key.Jum. [12]

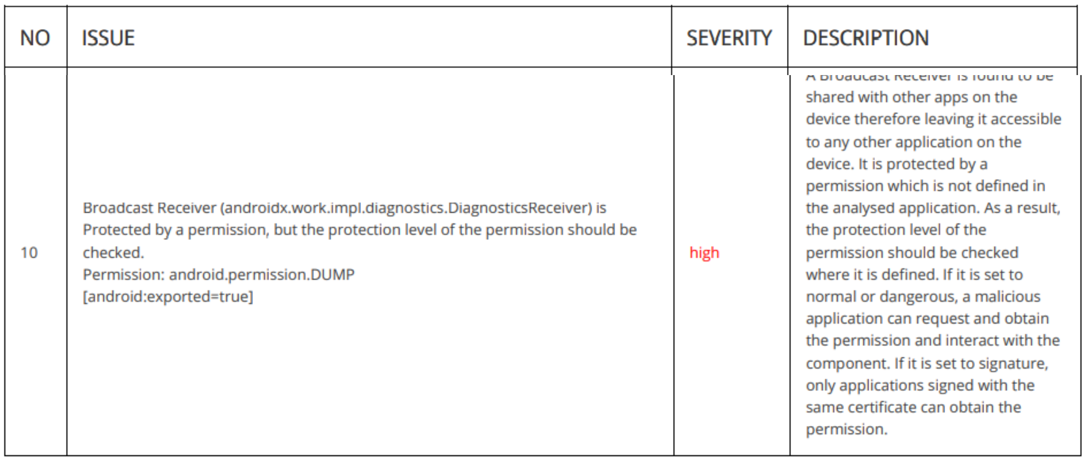


NO	ISSUE	SEVERITY	DESCRIPTION
8	Broadcast Receiver (com.google.android.gms.measurement.AppMeasurementInstallReferrerReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.INSTALL_PACKAGES [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Figure 50: Broadcast receiver, issue no.8

Issue no.8:
(com.google.android.gms.measurement.AppMeasurementInstallReferrerReceiver) is protected by a permission INSTALL_PACKAGES with “signature | privileged” protection level.

In addition to signature protection level that have been explained in the previous point, the INSTALL_PACKAGES also uses Privileged protection level, where Privileged apps are system apps that are located in a priv-app directory on one of the system image partitions. Historically, device manufacturers had little control over which signature|privileged permissions could be granted to privileged apps. Starting in Android 8.0, manufacturers must explicitly grant privileged permissions in the system configuration XML files in the /etc/permissions directory. As of Android 9, implementors must explicitly grant or deny all privileged permissions or the device won’t boot.[13]



NO	ISSUE	SEVERITY	DESCRIPTION
10	Broadcast Receiver (androidx.work.impl.diagnostics.DiagnosticsReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: android.permission.DUMP [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Figure 51: Broadcast receiver, issue no.10

Issue no.10:
android.permission.DUMP mainly retrieve system internal state Allows the app to retrieve internal state of the system. Malicious apps may retrieve a wide variety of private and secure information that they should never normally need.

The DUMP permission is flagged as android:protectionLevel="signature|system|development" (or signatureOrSystem using the old syntax) on Android 2.3+, and therefore cannot be held by ordinary Android SDK applications.

Moreover, development protection level is an Additional flag from base permission type: this permission can also (optionally) be granted to development applications. Although undocumented, the permission state used to be shared by all users (including future users), but it is managed peruser since API level 31.

6.3 Code Analysis

6.3.1 Code analysis section description

In code analysis section in MobSF, there are several security scoring and assessment frameworks has been used, which are:

- Common Vulnerability Scoring System (CVSS).
- Common Weakness Enumeration (CWE).
- The Open Web Application Security Project (OWASP).

Common Vulnerability Scoring System (CVSS):

CVSS [15]is a vulnerability scoring system and open framework, which is maintained by the FIRST [16], Forum of Incident Response and Security Teams. FIRST, is an international cooperative organization consists of several CERTs, Computer Emergency Response Team, where its members are responding to security incidents, for instance, unauthorized access to data. Also, exchanging information about them and providing standards.

One of these standards, is the CVSS, it is designed to provide a numerical (score from 1 to 10) value that represents the rate of severity of vulnerabilities in the system, depending on some metrices.

In MobSF, CVSS version 2.0 was used, the severity rating is represented as following:

CVSS v2.0 Ratings	
severity	Base Score Range
Low	0.0 – 3.9
Medium	4.0 – 6.9
High	7.0 – 10.0

Common Weakness Enumeration (CWE):

CWE [17] is providing a list, more like a dictionary, of hardware and software weakness, such as bugs, flaws, vulnerabilities and errors. Each weakness is assigned to an ID, CWE-XXX, for example. Its goal is to help developers, testers, designers or engineers to gain knowledge about common weaknesses, attack patterns that can be caused by them and how to handle or preventing them.

For each weakness, there are several features provided, some of them are:

- Description of the weakness in a high-level language.
- A related weaknesses to the described weakness, this relationship is organized as a tree data structure.
- Related attack patterns, which refer to CAPEC [18], Common Attack Pattern Enumeration and Classification.

CAPEC, or attack pattern is connected to weaknesses represented in CWE, as new weaknesses appear, attackers would not waste the chance to exploited them. It is like a never-ending list.

These attack patterns are provided to distribute awareness and provide prevention mechanisms for them. However, it could be used by the attackers themselves, it is like a double-edged sword.

The Open Web Application Security Project (OWASP):
OWASP [19], international non-profit organization to improve software security, its Foundation is the source for developers and technologists to secure the web, by providing easily reached tools and resources, community and networking, and education & training.

OWASP TOP10, also refer to "awareness document, is a up-to-date report that is covering top 10 most critical risks, and application should take this report in consideration, to minimize these risks to secure their web application.

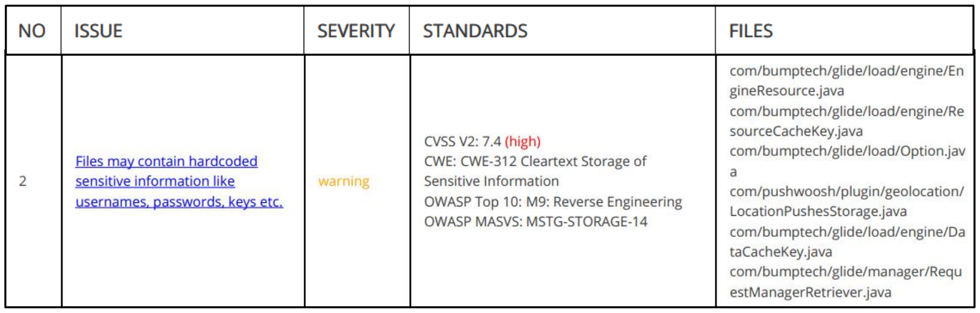
In MobSF, the OWASP analysis is depending on Top 10 Mobile Risks - Final List that is introduced in 2016, listed as following [20]:

M1: Improper Platform Usage
M2: Insecure Data Storage
M3: Insecure Communication
M4: Insecure Authentication
M5: Insufficient Cryptography
M6: Insecure Authorization
M7: Client Code Quality
M8: Code Tampering
M9: Reverse Engineering
M10: Extraneous Functionality

6.3.2 Code analysis issues

In this section, there appear some issues considered as high or warning severity, which indicate that there is a risk for the system, the issues are:

ISSUE no.2:



NO	ISSUE	SEVERITY	STANDARDS	FILES
2	Files may contain hardcoded sensitive information like usernames, passwords, keys etc.	warning	CVSS V2: 7.4 (high) CWE: CWE-312 Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	com/bumptech/glide/load/engine/EngineResource.java com/bumptech/glide/load/engine/ResourceCacheKey.java com/bumptech/glide/load/Option.java com/pushwoosh/plugin/geolocation/LocationPushesStorage.java com/bumptech/glide/load/engine/DataCacheKey.java com/bumptech/glide/manager/RequestManagerRetriever.java

Figure 53: code analysis issue no.2

Issue description:
The listed files could include hardcoded sensitive information, or often referred to as hardcoded credentials (passwords, keys, etc.), which are basically a cleartext (non-encrypted) information that is embedded inside the source code.

An attacker can easily download the .apk file, as we did in this report, and use a de-compile tool, such as, apktool and jadx, to decompile the source code to find credentials embedded in it.

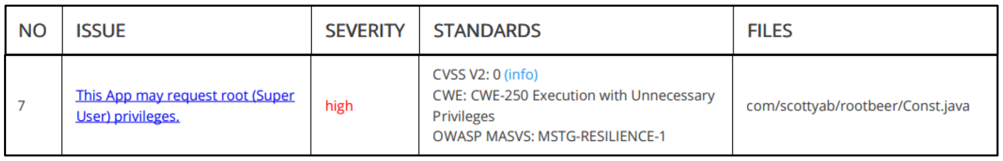
CVSS score:
7.4, which indicate a high risk.

CWE information:
Weakness ID and name:
CWE-312: Cleartext Storage of Sensitive Information. [21].
Weakness description:
Information is stored as a cleartext inside the source code, once an unauthorized access to this source code, the information can be clearly read.

Attack pattern, CAPEC:
- CAPEC-37: Retrieve Embedded Sensitive Data.

Recommended solution:
Avoid writing sensitive and credentials information within the source code.

ISSUE no.7:



NO	ISSUE	SEVERITY	STANDARDS	FILES
7	This App may request root (Super User) privileges.	high	CVSS V2: 0 (info) CWE: CWE-250 Execution with Unnecessary Privileges OWASP MASVS: MSTG-RESILIENCE-1	com/scottyab/rootbeer/Const.java

Figure 54: code analysis issue no.7.

Issue description:
In the file, a RootBeer library is included, which provides a "root detection". RootBeer is an open-source library that uses mechanisms for detecting if the device, which using this app, is it running on rooting mode or not. Basically, it is interacting with the privilege side of the device.

Rooting, in Android devices, means accessing the root of the operating system to be the administrator, also often refereed to SuperUser. It is being used to get a complete control of the device. For example, some people using it to download apps that are not available in Android OS, increasing volume beyond normal or changing color of flashlight. Similar to jailbreak in IOS.

Rooting a device has disadvantages as well. After rooting your device, some built-in security feature will be disabled, which makes your device less secure. As bank application, there are bunch of sensitive information, and must execute financial transactions as secure as possible. As a result, bank application is using root detection techniques to prevent installing the app in a rooted device.

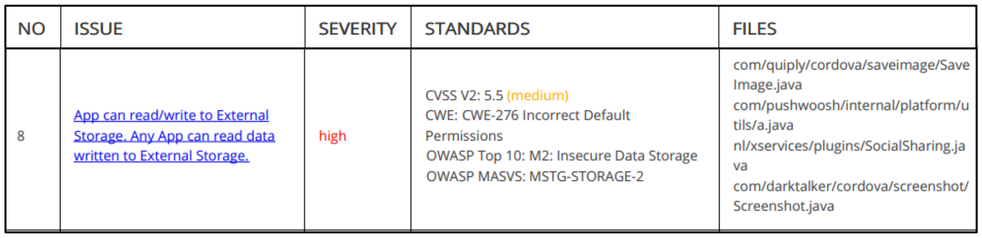
CVSS score:
0.0, which indicate a low risk.

CWE information:
Weakness ID and name:
CWE-250: Execution with Unnecessary Privileges. [22].
Weakness description:
The software is performing tasks at the privilege level, it will get access to the user's privilege root. It is a goal for attackers to reach it to gain full access of the device.
Attack pattern, CAPEC:

- CAPEC-104 Cross Zone Scripting.
- CAPEC-470 Expanding Control over the Operating System from the Database.
- CAPEC-69 Target Programs with Elevated Privileges.

Recommended solution:
We do not have a solution; the app needs to access the privilege side to check if the device is in the root mode or not.

ISSUE no.8:



NO	ISSUE	SEVERITY	STANDARDS	FILES
8	App can read/write to External Storage. Any App can read data written to External Storage.	high	CVSS V2: 5.5 (medium) CWE: CWE-276 Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	com/quiply/cordova/saveimage/SaveImage.java com/pushwoosh/internal/platform/utils/a.java nl/xservices/plugins/SocialSharing.java com/darktalker/cordova/screenshot/Screenshot.java

Figure 55: code analysis issue no.8.

Issue description:
There is an access, could be read or write, to the External Storage of the device.

For more illustration, in Android, when an app is installed, there are two types of storage for two types of files associating with the app, an Internal Storage and External Storage [23].

- **Internal storage:** is a location with limited space to store files, such as an app-specific files, where the system will prevent any access to these files by other apps, expect the access by the app itself. In Android 10 (API level 29), the data stored in the internal storage is encrypted. Also, there is extra security layer.

- **External Storage:** is a location (e.g., SD card) with large amount of space to store files, such as App-Independent Files, where it can be accessed by other apps and users, which make it less secure than internal storage.

As we mentioned earlier, there are two types of files installed with the app, which are:

- **App-Specific Files:** are the files that cannot be shared among other apps and only available and accessible by the app itself, and it usually have sensitive information. When the app is uninstalled, these files will be deleted automatically.

- **App-Independent Files:** are the files that can be shared by other apps after accepting some permissions. For instance, when you take a screenshot of a particular app screen, this photo will be stored in your photo gallery, then when you open another app and it asks you for permission to access your gallery, the photo, you have taken, earlier can be accessed by this app. Also, when an app is uninstalled, these files would not be deleted.

It is normal for an app to access an External Storage. In our case, with Alinma Bank app, the user may need to download receipt information, copy transactions' details, take screenshots or sharing links, all of these written in the External Storage. Or maybe the app needs a barcode to scan or access contacts, all of these files are read from the External Storage.

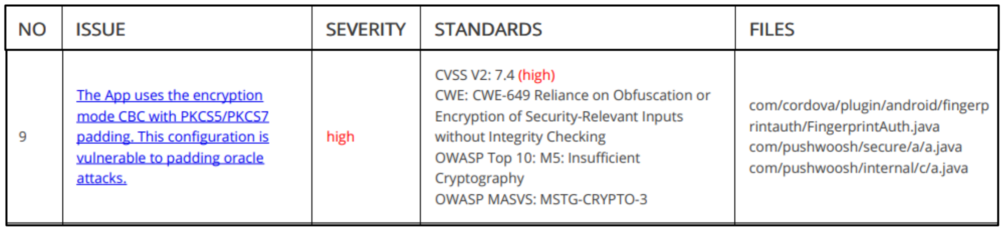
CVSS Score:
5.5, which indicates medium risk.

CWE information:
Weakness ID and name:
CWE-276: Incorrect Default Permissions [24].
Weakness Description:
During the installation of the application, there could be file permissions set to allow accessing and modifications by other applications by accident.
Attack Patterns, CAPEC:

- CAPEC-1 Accessing Functionality Not Properly Constrained by ACLs.
- CAPEC-127 Directory Indexing.
- CAPEC-81 Web Logs Tampering.

Recommended solution:
While giving permissions to files, avoid storing data in the External Storage as much as possible, especially when we are dealing with critical systems, like Bank system, where there is enormous amount of sensitive and credential information, and be sure that they are stored in the Internal Storage.

ISSUE no.9:



NO	ISSUE	SEVERITY	STANDARDS	FILES
9	The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks.	high	CVSS V2: 7.4 (high) CWE: CWE-649 Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-3	com/cordova/plugin/android/fingerprintauth/FingerprintAuth.java com/pushwoosh/secure/a/a.java com/pushwoosh/internal/c/a.java

Figure 56: code analysis issue no.9..

Issue description:
The app is using one of the Block-Cipher Cryptology modes, called "Cipher-Block Chaining (CBC)". CBC is basically encrypting a plaintext as a fixed-sized blocks, and when a particular block is not full of data, it will be padded. The padded methods used are PKCS5 and PKCS7.

Imagine the following two scenarios:

- In a Bank transaction scenario, we will consider a CBC mode is used for encryption. If some user wants to "deposit 10$", for example. Then this transaction message is interrupted in middle way by an attacker. The attacker can tamper the message without a key, or the need to decrypt it. If an attacker flips some arbitrary bits and the transaction changed to "deposit 1$", for example. The receiver, which is the Bank system, will receive the tapered message without noticing the that it was changed, because there is no authentication checking, checks for a particular message is sent by which origin.

- In another Bank transaction scenario, we will consider a cryptology that provides authentication checking (e.g., Galois/Counter Mode (GCM) mode). Same scenario as previous, but the receiver will notice that the message is being tapered, by checking the authentication of the origin, which will refer to the attacker origin.

Briefly, CBC only provide data confidentially, secured encrypted data, but it does not provide data integrity, often refer to data authentication, which is one of CBC's flaws.

In addition, in the issue field, it mentioned that there is a type of attack that can be caused, because of the CBC mode, called the "padding oracle attack". If the data you want to encrypt is smaller than the cipher's size, the left space is padded until full. In CBC there is a behavior for padding called "padding oracle", where oracle is similar to "tell".

A human analogy, imagine playing cards with a kid, when he is about to win, the happiness will appear in his face, but if he has the joker, it will also appear in his expression. And that is similar to "padding oracle", which an attacker can exploit, to determine if the padding is valid or not, by guessing until a "valid padding" message, which indicate it is the right message. This attack is dangerous, because an attacker does not need to know the security key to decrypt the message, only knowing the validity of the padding is enough [25].

CVSS Score:
7.4, which indicates a high risk.

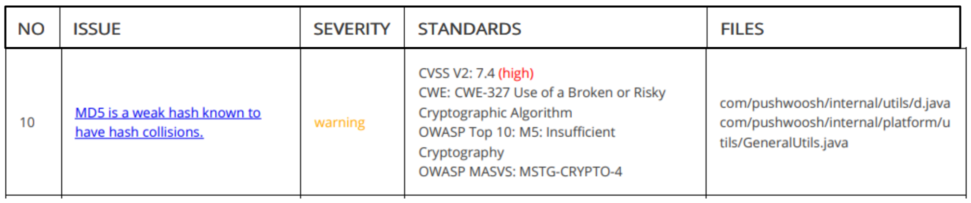
CWE information:
Weakness ID and name:
CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking [26].
Weakness description:
Using an encryption method that does not have an integrity or authentication checking.
Attack patterns, CAPEC:

- CAPEC-463 Padding Oracle Crypto Attack.

Recommended solution:
When the time passes, a particular cryptology is getting weaker and weaker, it appears weaknesses make it vulnerable, as the technology is developing extremally rapid, CBC is introduced in 1976, which make it very old technique. Also, avoid encryption methods that does not provide an integrity checking.
List of some encryption recommendations [27]:

- **Confidentiality algorithms:** AES-GCM-256 or ChaCha20-Poly1305.
- **Integrity algorithms:** SHA-256, SHA-384, SHA-512, Blake2, the SHA-3 family.
- **Digital signature algorithms:** RSA (3072 bits and higher), ECDSA with NIST P-384.
- **Key establishment algorithms:** RSA (3072 bits and higher), DH (3072 bits or higher), ECDH with NIST P-384.

ISSUE no.10:



NO	ISSUE	SEVERITY	STANDARDS	FILES
10	MD5 is a weak hash known to have hash collisions.	warning	CVSS V2: 7.4 (high) CWE: CWE-327 Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	com/pushwoosh/internal/utils/d.java com/pushwoosh/internal/platform/utils/GeneralUtils.java

Figure 57: code analysis issue no.10.

Issue description:
Similar to the previous issue (ISSUE no.9), where they are using a not recommended cryptography in present time. In these files, they used an encryption method that is dealing with 128-bit long key, is the Message-Digest algorithm 5 (MD5) encryption. This cryptology is considered "weak" nowadays.

Weak encryption [28], which are the encryption techniques that use an inefficient key length, which are the keys that are 128-bits or less. These types can be easily broken, which make them risky to use. Increasing the key length will increase the attack complexity, a long key length, will result in a huge number of probabilities.

A hash collision could happen, where two different data have same hash result. Some of the most common algorithms used in file hashing are [29]:

– MD5: the probability of two hashes collision accidentally is approximately: $1.47*10^{-29}$. The fastest and shortest generated hash (using 16 bytes).
– SHA-256: the probability of two hashes collision accidentally is approximately: $4.3*10^{-60}$. often 60% slower than MD5, and the longest generated hash (using 32 bytes).

According to the comparison between the two algorithms, MD5 is faster in generating hash but SHA-256 is much reliable, because its probability to have a collision is lower the MD5. However, even the MD5 have a terribly low probability, it is like a probability of water drop in the ocean, but as the technology evolving, MD5 is getting older and SHA-256 is -

CVSS Score:
7.4, which indicates a high risk.

CWE information:
Weakness ID and name:
CWE-327 Use of a Broken or Risky Cryptographic Algorithm [30].
Weakness description:

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the exposure of sensitive information.

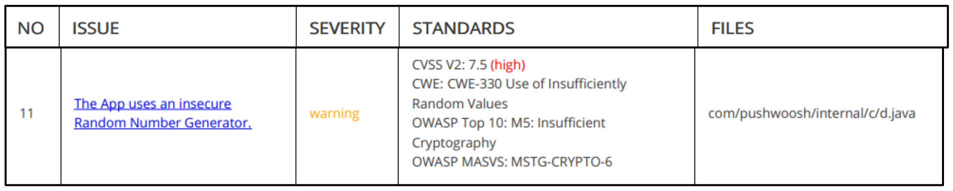
Attack patterns, CAPEC:

• CAPEC-20	Encryption Brute Forcing.
• CAPEC-459	Creating a Rogue Certification Authority Certificate.
• CAPEC-473	Signature Spoof.
• CAPEC-475	Signature Spoofing by Improper Validation.
• CAPEC-608	Cryptanalysis of Cellular Encryption.
• CAPEC-614	Rooting SIM Cards.
• CAPEC-97	Cryptanalysis.

Recommended solution:

Same solution provided in ISSUE no.9 (page.).

ISSUE no. 11:



NO	ISSUE	SEVERITY	STANDARDS	FILES
11	The App uses an insecure Random Number Generator.	warning	CVSS V2: 7.5 (high) CWE: CWE-330 Use of Insufficiently Random Values OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6	com/pushwoosh/internal/c/d.java

Figure 58: code analysis issue no.11.

Issue description:
Also, this issue is following (ISSUE no.9), using an insecure random number generator means it is providing a predictable value, in a situation where it should provide unpredictable values to achieve security.

CVSS Score:
7.5, which indicates a high risk.

CWE information:
Weakness ID and name:
CWE-330: Use of Insufficiently Random Values [31].
Weakness description:
The software is generating a predictable value that should be unpredictable, where the attacker can easily guess the values.
Attack patterns, CAPEC:

CAPEC-112	Brute Force
CAPEC-485	Signature Spoofing by Key Recreation
CAPEC-59	Session Credential Falsification through Prediction.

Recommended solution:
Same solution provided in ISSUE no.9 (page.).

7. Discussion

7.1 In the design analysis section

the GUI of Alinma Bank, in overall, had a good designed icon, eye-matching colors and it is usable and understandable. For instance, they are using some of the usability heuristic, one of them is “match between the system and the real world” heuristic. In our opinion, they have a good GUI.

In design principles, it was hard to detect cohesion and coupling, because the source code is too big and it is hard to decide each class which responsibilities it is taken. We have found out some violation on Single Responsibility Principle in section (5.3.2 Avoid really long methods).

7.2 In the source code style section

we used Codacy tool to help us in this section, because the source code is too big to handle it manually, with helping of PMD and CheckStyle built-in functions.

This tool detected a lot of issue types, divided into categories, we choose some of them, which are; code style, and error prone. In code style section, there are also a lot of issues patterns, some of them we know, so we take them, and some of them are beyond our knowledge in java programming coding, so it was hard to explain them.

The tool itself may produce a false positive result, which are the issues that are detected by the tool, but it is actually not an issue. We noticed some places where this situation happened, for instance, in section (5.2.1.3 Naming Convention, Class Naming). But do not worry, we have pointed them properly.

Back to our main point, in Alinma source code, they are using java language, so, we decided to organize some of the issues according to Java Code Convention, which is provided by Oracle. This code convention will help them to improve their code structure quality, depending on an international java code convention. For example, if they are communicating with foreign developers from different nationalities, they all will be in the same page, if they follow those conventions. However, there are some places where these conventions are violated, for instance, in section (5.2.1.1 Declaration order), they are not taking declaration order into consideration, which make the code structure messy, which results in, code will not be readable and hard to maintain.

In some of the issues we noticed some patterns that are repeated, for instance, in section (5.2.2.1 Unnecessary Paratheses), they used extra parentheses to distinguish compression signs from the equal assignment sign. In our opinion, we considered it as a good practice, because it will be much easier to differentiate them from other assignments. If this is a standard that is used within the organization, it should be documented well. So new programmers will have the ability to follow them.

There are a lot more issues you can find in the Codacy analysis section, an overall overview, Alinma source code is not bad but also not good, following a coding standard are tiresome and time consuming. It is not good for a programmer to take the simplest road and not taking future consequences into consideration, the programmer is not the only one who will read the code, there are testers, quality assurance (QA) and also other programmers. So, the code structure is extremely important.

7.3 In the security section

We used MobSF static tool analysis, uploading Alinma source code in it, and then retrieving a "document", which consists of security issues, including Application Permission, Manifest Analysis and Code Analysis. Of course, there was more sections in the document but they were extremely in deep programing, we could not understand it very well, so we take the most important points in developing Android app, which are the three points we mentioned earlier.

The overall security in Alinma bank, according to security score appear in MobSF, is scored 6.9 in average CVSS, which refer to medium risk, and that is not practical in a Bank app, it must have a low ranking, 3.9 or lower. Also, the overall security is scored 25 from 100. We do know exactly how this score is calculated, but it is extremely low. This is not a good sign, as a Bank app, it must have higher security score.

We covered some of these security issues,

In Application Permission, there are four permissions that their status is dangerous, among all the thirty-two permissions, the other permissions have the status 'normal' and 'signature', which indicate that there is no problem in them. The four permissions are accessing private information of the user, which are, accessing the camera, writing in external storage, and finding fine and coarse location information of the device. So, these four permissions that threating the security system must be taken into consideration.

In Manifest Analysis, which is an important file of developing an Android application, have the security issues ranked according to risk levels, which are; high risk, medium risk and low risk. The high-risk issues appear remarkably, there are nine high-risk issues among ten issues. It is an important file, that includes permissions. However, in the Manifest Analysis files results there appears some issues considered as flaws, that threat the system security, and we provided some solutions to them.

In Code Analysis, the security analysis is depending on international security standards which are, CVSS, CWE and OWASP. The severity level was ranked according to them. The issues are varying between each other. There are issues in accessing privilege side (critical side) of the system, reading and writing in external storage, using insufficient cryptography. The most important point the got our attention, is using old, vulnerable and not recommended cryptography methods. As a critical system as a Bank, it must have proper cryptography methods

They are using CBC mode that does not have an integrity checking, which is very dangerous, because they do not know the origin of the transaction message. Also, using MD5 hashing method, which is considered weak hashing nowadays, as the technology is evolving and new hashing methods have introduced, we provided some other cryptography methods, you can find more details in (6.3 Code Analysis) section.

8. Conclusion

Doing a static analysis is something new to us, through our little experience and our use of appropriate tools that helped us, we concluded that the program suffers from many problems in terms of quality, and its application of the concept of "clean code" is very poor and they must work to improve this thing.

As a Bank application, it was surprising finding all these issues, especially the security issues, our critical information is between their hands. There is a lot of warnings and high-risk issues in their source code and the Android manifest file. They must take all of them into consideration, improving their protection methods, and also improving not only the developer team but also the quality assurance tea to cover solve these issues in efficient manner.

9. References

[1] Nielsen Norman Group. *10 Usability Heuristics for User Interface Design*. [online] Available at: < https://www.nngroup.com/articles/ten-usability-heuristics/ > [Accessed 4 November 2021].

[2] *Codacy* [online] Available at: < https://app.codacy.com/organizations > [Accessed 20 October 2021].

[3] Pmd.github.io. *PMD*. [online] Available at: < https://pmd.github.io/ > [Accessed 20 October 2021].

[4] Pmd.github.io. *Java Rules | PMD Source Code Analyzer*. [online] Available at: < https://pmd.github.io/latest/pmd_rules_java.html > [Accessed 20 October 2021].

[5] Checkstyle.sourceforge.io. *checkstyle – Checkstyle 9.1*. [online] Available at: < https://checkstyle.sourceforge.io/ > [Accessed 20 October 2021].

[6] Oracle.com. *Code Conventions for the Java Programming Language: Contents*. [online] Available at: <https://www.oracle.com/java/technologies/javase/codeconventions-contents.html > [Accessed 23 October 2021].

[7] Line length. *why restricting with 80-character long length*. [online] Stack Overflow. Available at: < https://stackoverflow.com/questions/110928/is-there-a-valid-reason-for-enforcing-a-maximum-width-of-80-characters-in-a-code > [Accessed 4 November 2021].

[8] behavior, D., Sharma, A., Velthuis, R. and Mir, T,. *Difference in BigDecimal behavior*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/41584046/difference-in-bigdecimal-behavior/41584285 > [Accessed 4 November 2021].

[9] www.kaspersky.com. *What is SQL Injection?*. [online] Available at: <https://www.kaspersky.com/resource-center/definitions/sql-injection > [Accessed 4 November 2021].

[10] Infosec Resources. *Android hacking and security, part 2: Content provider leakage - Infosec Resources*. [online] Available at: <https://resources.infosecinstitute.com/topic/android-hacking-security-part-2-content-provider-leakage/ > [Accessed 4 November 2021].

[11] GitHub. *platform_frameworks_base/AndroidManifest.xml at master · aosp-mirror/platform_frameworks_base*. [online] Available at: < https://github.com/aosp-mirror/platform_frameworks_base/blob/master/core/res/AndroidManifest.xml > [Accessed 4 November 2021].

[12] [duplicate], s., *signature protection level - clarifying*. [online] Stack Overflow. Available at: < https://stackoverflow.com/questions/21438129/signature-protection-level-clarifying > [Accessed 4 November 2021].

[13] Android Open Source Project. *Privileged Permission Allowlisting | Android Open Source Project*. [online] Available at: <https://source.android.com/devices/tech/config/perms-allowlist > [Accessed 4 November 2021].

[14] Android Developers. *R.attr | Android Developers*. [online] Available at: < https://developer.android.com/reference/android/R.attr#protectionLevel > [Accessed 4 November 2021].

[15] FIRST — Forum of Incident Response and Security Teams. *Common Vulnerability Scoring System SIG*. [online] Available at: < https://www.first.org/cvss/ > [Accessed 22 October 2021].

[16] FIRST — Forum of Incident Response and Security Teams. *FIRST - Improving Security Together*. [online] Available at: < https://www.first.org/ > [Accessed 22 October 2021].

[17] Cwe.mitre.org. *CWE - Common Weakness Enumeration*. [online] Available at: <https://cwe.mitre.org/ > [Accessed 22 October 2021].

[18] Capec.mitre.org. *CAPEC – Common Attack Pattern Enumeration and classification.* [online] Available at < https://capec.mitre.org/ > [Accessed 22 October 2021].

[19] Owasp.org. OWASP - The Open Web Application Security Project. [online] Available at <https://owasp.org/> [Accessed 22 October 2021].

[20] Owasp.org. *OWASP mobile top 10 2016*. [online] Available at < https://owasp.org/www-project-mobile-top-10/> [Accessed 22 October].

[21] Cwe.mitre.org. *CWE- -312: Cleartext Storage of Sensitive Information.* [online] Available at <https://cwe.mitre.org/data/definitions/312.html> [Accessed 22 October 2021].

[22] Cwe.mitre.org. *CWE-250: Execution with Unnecessary Privileges.* . [online] Available at < https://cwe.mitre.org/data/definitions/250.html > [Accessed 22 October 2021].

[23] android Developers. *Access app-specific files | Android Developers*. [online] Available at: < https://developer.android.com/training/data-storage/app-specific > [Accessed 23 October 2021].

[24] Cwe.mitre.org. *CWE-276: Incorrect Default Permissions.* [online] Available at < https://cwe.mitre.org/data/definitions/276.html > [Accessed 22 October 2021].

[25] Docs.microsoft.com. *CBC decryption vulnerability*. [online] Available at: <https://docs.microsoft.com/en-us/dotnet/standard/security/vulnerabilities-cbc-mode > [Accessed 23 October 2021].

[26] Cwe.mitre.org. *CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking.* [online] Available at <https://cwe.mitre.org/data/definitions/649.html > [Accessed 22 October 2021].

[27] GitHub. *owasp-mstg/0x04g-Testing-Cryptography.md at master · MobSF/owasp-mstg*. [online] Available at: < https://github.com/MobSF/owasp-mstg/blob/master/Document/0x04g-Testing-Cryptography.md#identifying-insecure-andor-deprecated-cryptographic-algorithms-mstg-crypto-4 > [Accessed 24 October 2021].

[28] Electric Energy Online. *Security Sessions: Exploring Weak Ciphers - An Explanation and an Example*. [online] Available at: <https://electricenergyonline.com/energy/magazine/779/article/Security-Sessions-Exploring-Weak-Ciphers.html > [Accessed 24 October 2021].

[29] Ramirez, G., 2021. *MD5: The broken algorithm - Avira Blog*. [online] Avira Blog. Available at: < https://www.avira.com/en/blog/md5-the-broken-algorithm > [Accessed 29 October 2021].

[30] Cwe.mitre.org. *CWE-327 Use of a Broken or Risky Cryptographic Algorithm* . [online] Available at < https://cwe.mitre.org/data/definitions/327.html > [Accessed 1 November 2021].

[31] Cwe.mitre.org. *CWE-330: Use of Insufficiently Random Values* . [online] Available at < https://cwe.mitre.org/data/definitions/330.html > [Accessed 3 November 2021].