

Final Project: MealLogger App Analysis

Chad Josim

University of Nevada, Reno

CS 453: Mobile Computing Security and Privacy

Dr. Bill Doherty

May 8, 2023

Pass/Fail Rating and Analysis

After thorough analysis and careful consideration, the MealLogger application ultimately fails the OWASP Mobile App Security checklist. The table below shows the breakdown of the passed, failed, and non-applicable standards for each requirement:

	Arch	Storage	Crypto	Auth	Network	Platform	Code	Resilience	Total
Passed	4	2	2	4	3	5	4	2	26
Failed	2	13	3	4	2	1	1	5	31
N/A	6	0	1	4	1	5	4	6	27
Total	12	15	6	12	6	11	9	13	84

Looking closely at this breakdown, the MealLogger application passed 46% and failed 54% of the applicable/testable OWASP standards, clearly failing to meet a majority of the requirements. Interestingly, the app performed much better in some of the requirement categories than others. For instance, MealLogger had a positive rate in the Architecture, Network, Platform, and Code requirements sections. More specifically, IPC mechanisms such as intents, custom URLs, and network sources, seemed to be well filtered and protected by the app, and the code itself implements proper certificate signing and error handling. The app was relatively split for the Authentication category, implementing most security policies but falling short on elements like password length. Finally, the MealLogger app had a negative rate in the Storage, Crypto, and Resilience requirement categories. Notably, the app failed many of the storage requirements due to keeping sensitive information in log files on local storage, and the code obfuscation was not fully effective.

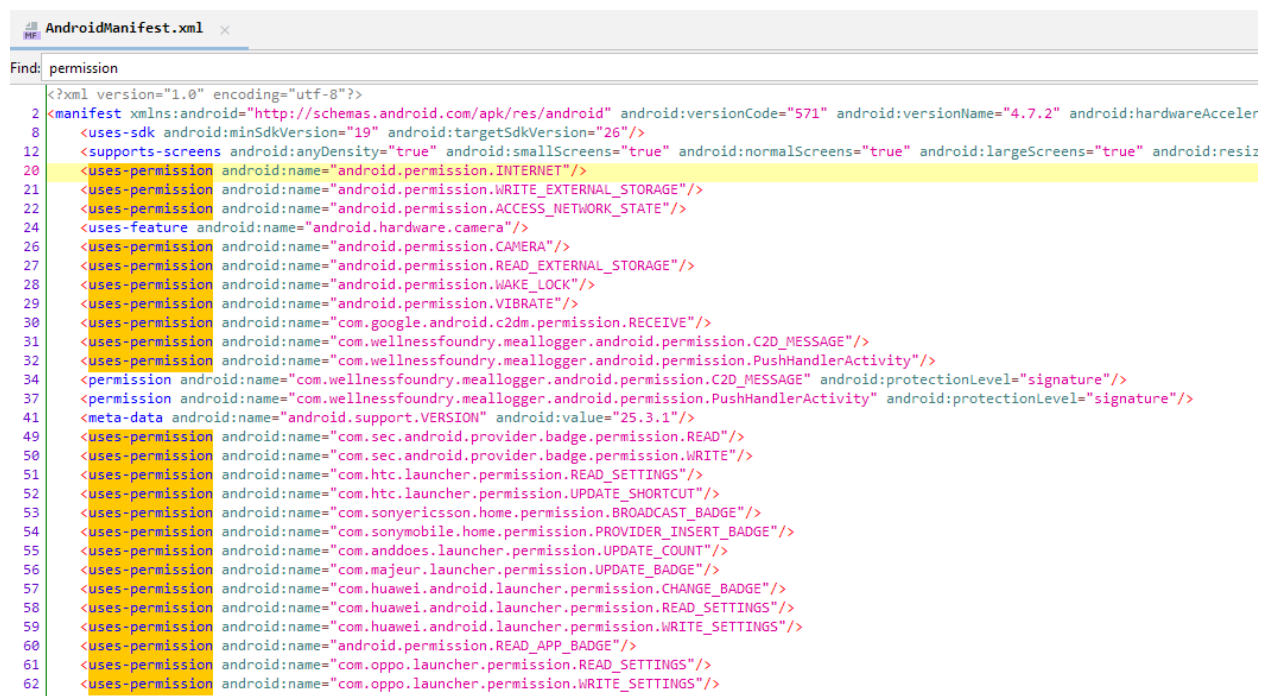
Regarding the tests that were not applicable/testable, the majority of them were simply outside of the scope of this class. Complexity of the source code, lack of information on the architecture/development lifecycle, lack of certain components like WebView, and iOS exclusivity contributed to the other inapplicable tests (each reason specified under the corresponding standard).

Architecture, Design and Threat Modeling Requirements

MSTG-ARCH-1

All app components are identified and known to be needed.

By opening the AndroidManifest.xml file in jadx, we can see the various app components set up by the MealLogger application (shown below). Looking at the custom (declared) permissions, the C2D_MESSAGE allows the cloud to communicate with the device, while the PushHandlerActivity allows for custom notifications. Although not apparently malicious, it is important to note the existence of these custom permissions. However, by looking closely at the actual permissions included, it seems that the read/write permissions for both external storage and settings are clearly suspicious and likely unnecessary for the functionality of the app (explored in later sections). For these reasons, the MealLogger app ultimately fails this standard.



```

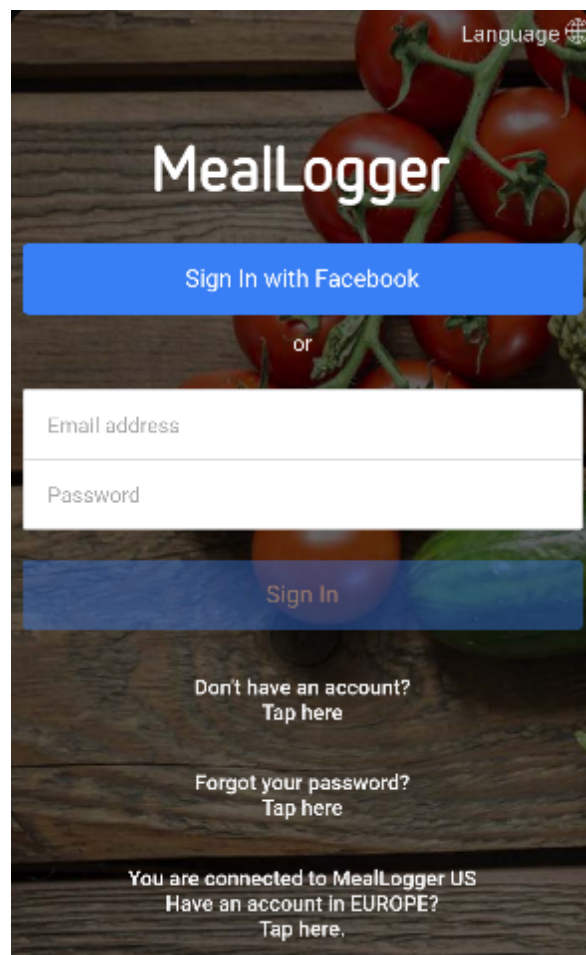
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="571" android:versionName="4.7.2" android:hardwareAccelerated="true">
3     <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="26"/>
4     <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true" android:resizeable="true"/>
5     <uses-permission android:name="android.permission.INTERNET"/>
6     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
7     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
8     <uses-feature android:name="android.hardware.camera"/>
9     <uses-permission android:name="android.permission.CAMERA"/>
10    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
11    <uses-permission android:name="android.permission.WAKE_LOCK"/>
12    <uses-permission android:name="android.permission.VIBRATE"/>
13    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
14    <uses-permission android:name="com.wellnessfoundry.meallogger.android.permission.C2D_MESSAGE"/>
15    <uses-permission android:name="com.wellnessfoundry.meallogger.android.permission.PushHandlerActivity"/>
16    <permission android:name="com.wellnessfoundry.meallogger.android.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
17    <permission android:name="com.wellnessfoundry.meallogger.android.permission.PushHandlerActivity" android:protectionLevel="signature"/>
18    <meta-data android:name="android.support.VERSION" android:value="25.3.1"/>
19    <uses-permission android:name="com.sec.android.provider.badge.permission.READ"/>
20    <uses-permission android:name="com.sec.android.provider.badge.permission.WRITE"/>
21    <uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS"/>
22    <uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT"/>
23    <uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE"/>
24    <uses-permission android:name="com.sonymobile.home.permission.PROVIDER_INSERT_BADGE"/>
25    <uses-permission android:name="com.anddoes.launcher.permission.UPDATE_COUNT"/>
26    <uses-permission android:name="com.majeur.launcher.permission.UPDATE_BADGE"/>
27    <uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE"/>
28    <uses-permission android:name="com.huawei.android.launcher.permission.READ_SETTINGS"/>
29    <uses-permission android:name="com.huawei.android.launcher.permission.WRITE_SETTINGS"/>
30    <uses-permission android:name="android.permission.READ_APP_BADGE"/>
31    <uses-permission android:name="com.oppo.launcher.permission.READ_SETTINGS"/>
32    <uses-permission android:name="com.oppo.launcher.permission.WRITE_SETTINGS"/>
33 </manifest>

```

MSTG-ARCH-2

Security controls are never enforced only on the client side, but on the respective remote endpoints.

For authentication purposes, the main method of logging into the MealLogger app is through a combination of email/password (shown below). Other notable functions include creating an account and password retrieval. Since MealLogger is an application that only has functionality following login, authentication is in fact required for access to remote resources. Furthermore, due to the password retrieval option, the security controls are in fact enforced on both the client side and respective remote endpoints. Thus, the MealLogger application passes this standard.



MSTG-ARCH-3

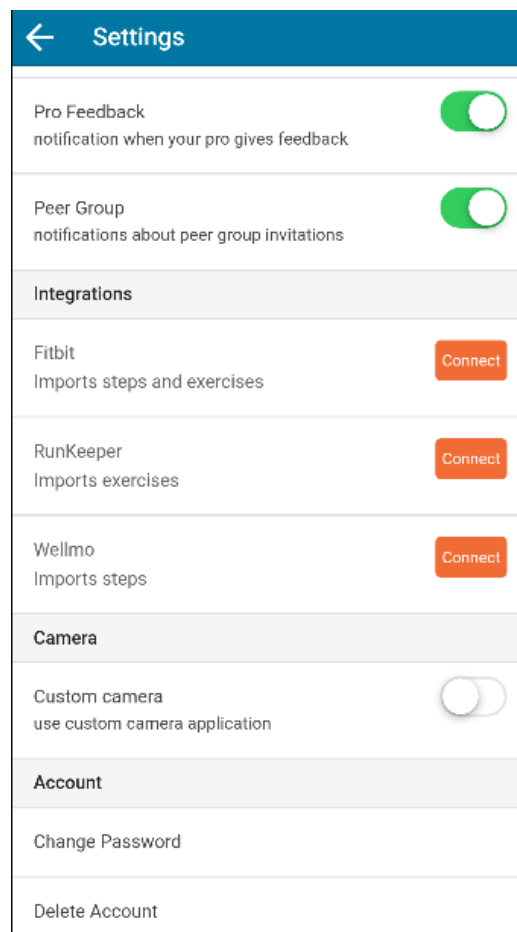
A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.

Note: Not testable, due to lack of high-level architecture.

MSTG-ARCH-4

Data considered sensitive in the context of the mobile app is clearly identified.

By observing and testing functionality of the app, I was able to find that MealLogger does not explicitly identify sensitive data as such. For instance, in the settings page (shown below), there is no indication, label, or warning that the user would be accessing sensitive information, such as changing their password or viewing their personal health information. For these reasons, the MealLogger app fails this standard.



MSTG-ARCH-5

All app components are defined in terms of the business functions and/or security functions they provide.

Note: Beyond the scope of this class.

MSTG-ARCH-6

A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.

Note: Not applicable, as the app provides no associated threat model.

MSTG-ARCH-7

All security controls have a centralized implementation.

Note: Not testable, due to complexity of source code and diversity of security controls.

MSTG-ARCH-8

There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.

Note: Not testable, due to lack of information on cryptographic key lifecycle.

MSTG-ARCH-9

Although I was not able to download an outdated version of the application, I was still able to find signs of forced updates. By opening the apk file in jadx and searching for the “update” keyword, I was able to find that there is a flag labeled

SERVICE_VERSION_UPDATE_REQUIRED

as well as a case statement that returns the value for that same flag. In this sense, there definitely seems to be a check for whether or not the app is fully updated within the current system, and the MealLogger app passes this requirement.

```

public class CommonStatusCodes {
    public static final int API_NOT_CONNECTED = 17;
    public static final int CANCELED = 16;
    public static final int DEVELOPER_ERROR = 10;
    public static final int ERROR = 13;
    public static final int INTERNAL_ERROR = 8;
    public static final int INTERRUPTED = 14;
    public static final int INVALID_ACCOUNT = 5;
    public static final int NETWORK_ERROR = 7;
    public static final int RESOLUTION_REQUIRED = 6;
    @Deprecated
    public static final int SERVICE_DISABLED = 3;
    @Deprecated
    public static final int SERVICE_VERSION_UPDATE_REQUIRED = 2;
    public static final int SIGN_IN_REQUIRED = 4;
    public static final int SUCCESS = 0;
    public static final int SUCCESS_CACHE = -1;
    public static final int TIMEOUT = 15;

    @NonNull
    public static String getStatusCodeString(int i) {
        switch (i) {
            case -1:
                return "SUCCESS_CACHE";
            case 0:
                return "SUCCESS";
            case 1:
            case 9:
            case 11:
            case 12:
            default:
                return new StringBuilder(32).append("unknown status code: ").append(i).toString();
            case 2:
                return "SERVICE_VERSION_UPDATE_REQUIRED";
        }
    }
}

```

MSTG-ARCH-10

Security is addressed within all parts of the software development lifecycle.

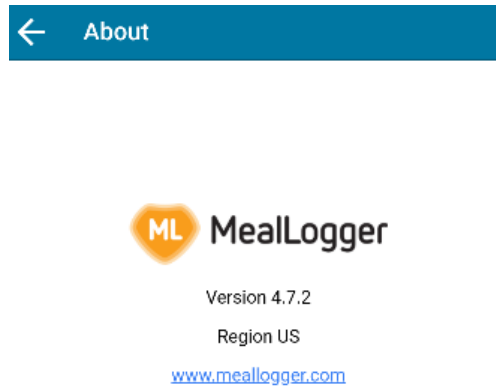
Note: Not testable, due to lack of information on software development lifecycle.

MSTG-ARCH-11

A responsible disclosure policy is in place and effectively applied.

While this requirement is relatively broad, the MealLogger app does in fact implement and apply a privacy policy. Although this privacy policy is not directly available within the app, a user can navigate to their website containing it from the “About” page (shown below). Regardless of the

ethics behind the policies, they do have one in place and support it with their app's functionality. For these reasons, the MealLogger app passes this standard.



MSTG-ARCH-12

The app should comply with privacy laws and regulations.

Although this requirement is quite broad, the MealLogger app follows all necessary privacy laws and regulations to be advertised on the Google Play store, at the very least. Similar to the last standard, the MealLogger website does provide their privacy policy (small snippet shown below), which seems reasonable and follows most conventional policy rules. For these reasons, the MealLogger app passes this requirement.

Type of Data Collected

We collect information from you in various ways when you use our Web sites and services. We may also supplement this information with information from other companies. We collect two general types of information, namely personal information and aggregate data. As used in this Policy, the term "personal information" means information that specifically identifies an individual and demographic and other information when directly linked to information that can identify an individual. Personal information includes your name and email address; your title, company and other profile information you provide; demographic information, etc. Our definition of personal information does not include "aggregate" data or information uploaded by you (including photos) that is not personally identifiable.

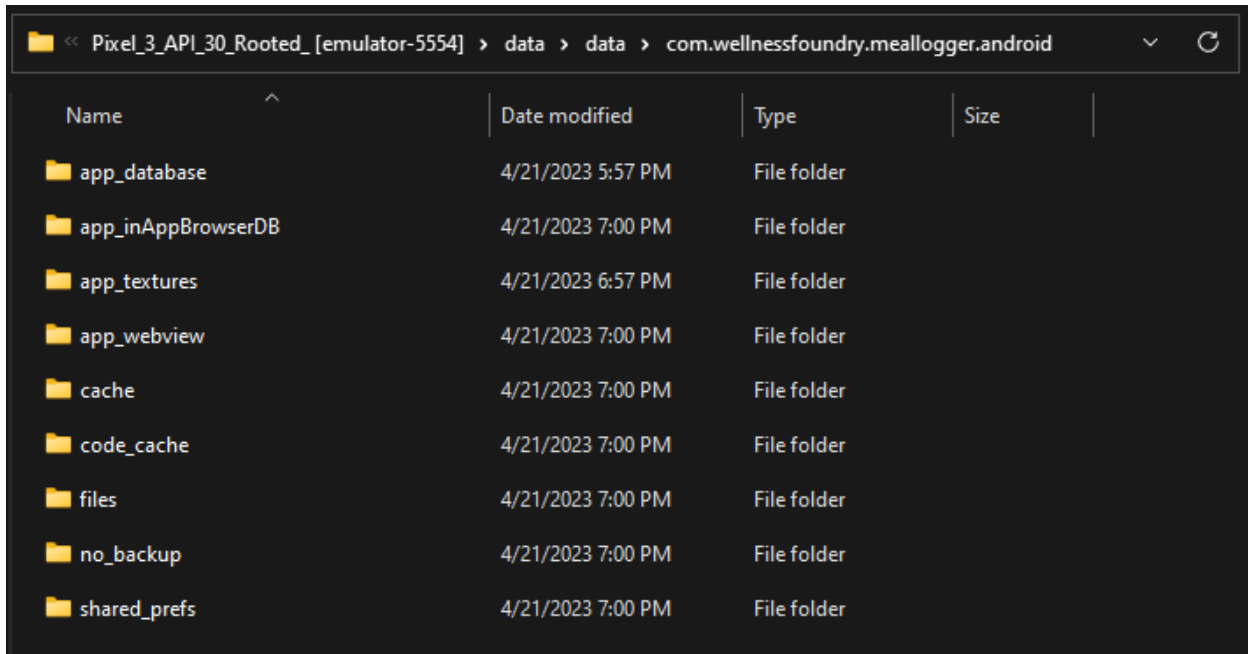
Certain other areas of the site or your Wellness Professional will ask for information on your habits or health (such as health history, health conditions, diet, height, weight, body mass index, test results) and activities (such as types and amount of exercise) to the extent required to provide the services to you, all of which is stored on our Site. In providing such information to the Wellness Professional you consent to the collection and disclosure of such information by and to Wellness Foundry.

Data Storage and Privacy Requirements

MSTG-STORAGE-1

System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.

By performing dynamic app analysis and testing all functionalities, I was able to find and save all of the files generated from a session in the MealLogger application (shown below).



Name	Date modified	Type	Size
app_database	4/21/2023 5:57 PM	File folder	
app_inAppBrowserDB	4/21/2023 7:00 PM	File folder	
app_textures	4/21/2023 6:57 PM	File folder	
app_webview	4/21/2023 7:00 PM	File folder	
cache	4/21/2023 7:00 PM	File folder	
code_cache	4/21/2023 7:00 PM	File folder	
files	4/21/2023 7:00 PM	File folder	
no_backup	4/21/2023 7:00 PM	File folder	
shared_prefs	4/21/2023 7:00 PM	File folder	

At a glance, many of the files contained sensitive information and data, including personal identifiers, which will be further elaborated on in the next section. This sensitive information was not stored using proper system credential storage facilities but was instead stored locally on the mobile device. For these reasons, the MealLogger app fails this requirement.

MSTG-STORAGE-2

No sensitive data should be stored outside of the app container or system credential storage facilities.

By analyzing the files found from the first section, I was able to identify many significant findings for this standard. Interestingly, the app_database, app_inAppBrowserDB, app_textures, code_cache, and files directories were all empty. In this case, it seems that a large section of the

app's database and important files were not unnecessarily stored. However, there were several xml files in shared_prefs, some of which exposed sensitive information, like an exposed ApiKey and user ID/email (shown below).

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <boolean name="app_intercom_link" value="false" />
  <string name="ApiKey">android_sdk-ad9bdb69e9e1354a96c8798b1f5912f816aa1f2e</string>
  <long name="ping_delay_ms" value="1000" />
</map>
```

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="intercomsdk-session-INTERCOM_SDK_INTERCOM_ID"></string>
  <string name="intercomsdk-session-INTERCOM_SDK_USER_ID">169533</string>
  <string name="intercomsdk-session-INTERCOM_SDK_EMAIL_ID">mpyra328@gmail.com</string>
</map>
```

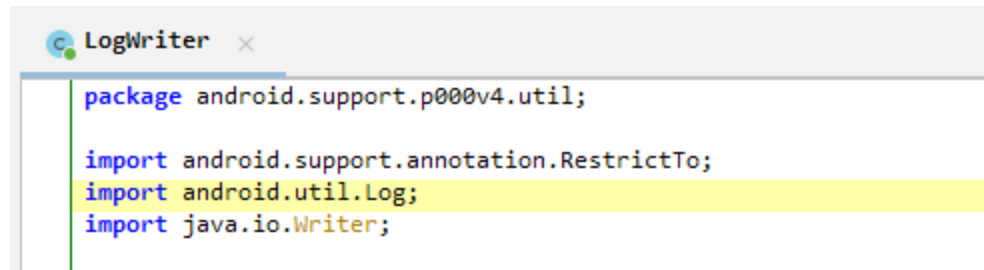
Looking at the external storage file location (sdcard), all of the images that I took within the app (shown below) were actually saved, which is very significant and dangerous. For these reasons, the MealLogger app fails this requirement.



MSTG-STORAGE-3

No sensitive data is written to application logs.

By opening the apk file in jadx and performing static analysis, I was able to search the source code for significant keywords, specifically those relating to logging and system print statements. Firstly, I was able to find a LogWriter class (shown below), which imports android.util.Log. Clearly, this is related to writing logs from the application, although there is nothing immediately suspicious within the class itself.



```

package android.support.p000v4.util;

import android.support.annotation.RestrictTo;
import android.util.Log;
import java.io.Writer;

```

By searching for other instances of “log_”, I was able to locate several instances of logging. One such example (shown below) is related to Google Play utilities, which logs whether or not such services are up to date. Although not immediately malicious, it is still important to note.

```

@VisibleForTesting
private static int zza(Context context, boolean z, int i) {
    Preconditions.checkArgument(i >= 0);
    PackageManager packageManager = context.getPackageManager();
    PackageInfo packageInfo = null;
    if (z) {
        try {
            packageInfo = packageManager.getPackageInfo("com.android.vending", 8256);
        } catch (PackageManager.NameNotFoundException e) {
            Log.w("GooglePlayServicesUtil", "Google Play Store is missing.");
            return 9;
        }
    }
    try {
        PackageInfo packageInfo2 = packageManager.getPackageInfo("com.google.android.gms", 64);
        GoogleSignatureVerifier googleSignatureVerifier = GoogleSignatureVerifier.getInstance(context);
        if (!googleSignatureVerifier.isGooglePublicSignedPackage(packageInfo2, true)) {
            Log.w("GooglePlayServicesUtil", "Google Play services signature invalid.");
            return 9;
        } else if (z && (!googleSignatureVerifier.isGooglePublicSignedPackage(packageInfo, true) || !packageInfo.signatures[
            Log.w("GooglePlayServicesUtil", "Google Play Store signature invalid.");
            return 9;
        } else if (GmsVersionParser.parseBuildVersion(packageInfo2.versionCode) < GmsVersionParser.parseBuildVersion(i)) {
            Log.w("GooglePlayServicesUtil", new StringBuilder(77).append("Google Play services out of date. Requires ").app
            return 2;
        } else {
            ApplicationInfo applicationInfo = packageInfo2.applicationInfo;
            if (applicationInfo == null) {
                try {
                    applicationInfo = packageManager.getApplicationInfo("com.google.android.gms", 0);
                } catch (PackageManager.NameNotFoundException e2) {
                    Log.wtf("GooglePlayServicesUtil", "Google Play services missing when getting application info.", e2);
                    return 1;
                }
            }
            return !applicationInfo.enabled ? 3 : 0;
        }
    } catch (PackageManager.NameNotFoundException e3) {
        Log.w("GooglePlayServicesUtil", "Google Play services is missing.");
        return 1;
    }
}

```

By searching for the system print statements, I was able to find a small number of instances within the source code (shown below). However, similar to the log findings, nothing here is inherently suspicious.

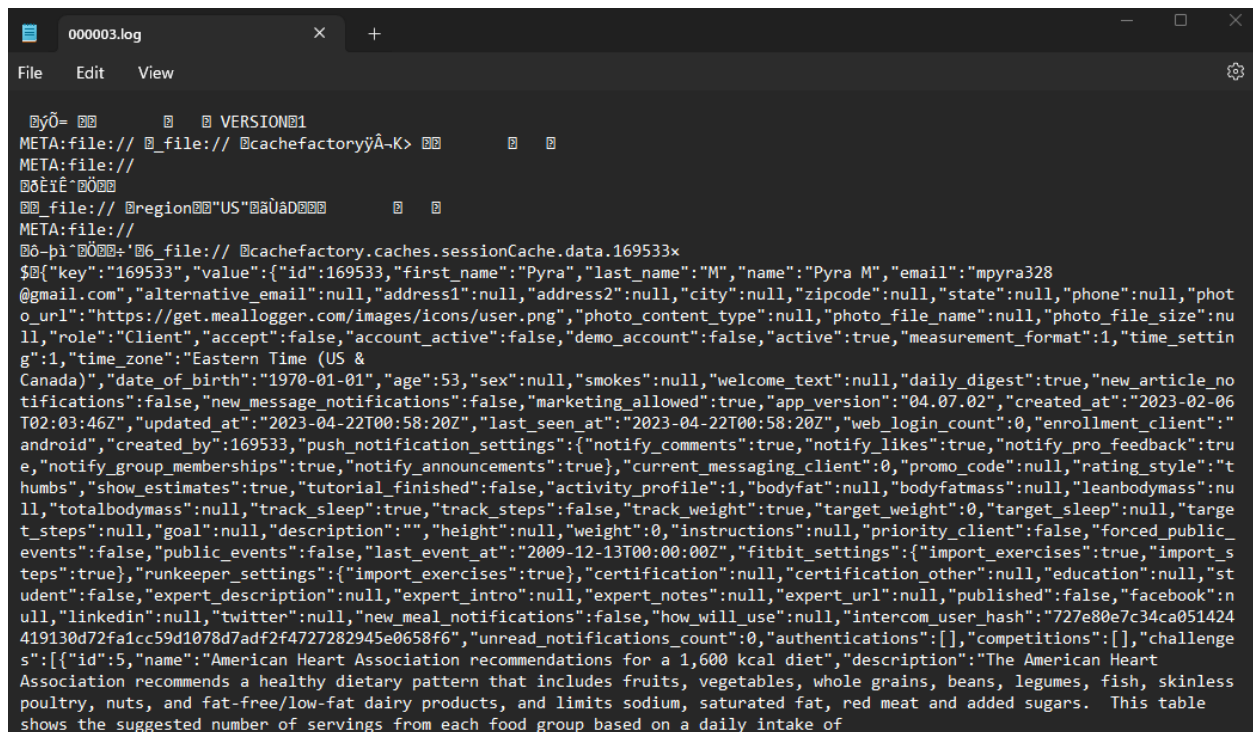
```

System.out.println("DiskLruCache " + directory + " is corrupt: " + journalIsCorrupt.getMessage() + ", removing");
System.out.println("DiskLruCache " + directory + " is corrupt: " + journalIsCorrupt.getMessage() + ", removing");
System.out.println("Error adding plugin " + className + ".");

```

```
System.err.println("Failed to retrieve value from android.os.Build$VERSION.SDK_INT due to the following exception.");
System.err.println("Failed to retrieve value from android.os.Build$VERSION.SDK_INT due to the following exception.");
System.err.println("Failed to set 'rx.indexed-ring-buffer.size' with value " + sizeFromProperty + " => " + e.getMessage());
System.err.println("RxJavaErrorHandler threw an Exception. It shouldn't. => " + pluginException.getMessage());
System.err.println("Failed to set 'rx.buffer.size' with value " + sizeFromProperty + " => " + e.getMessage());
```

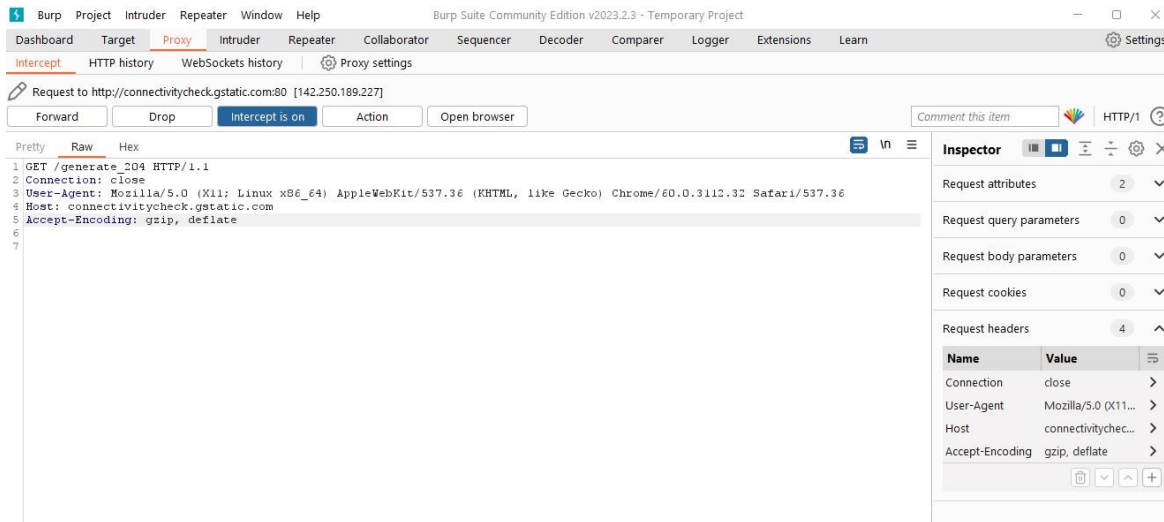
Finally, by performing dynamic analysis and searching the data folder for logs, I was able to find one significant log file (shown below). Although some parts are unreadable, the majority of the file shows relevant information pertaining to a session within the app. At a glance, the log file contains very sensitive information, including email, DOB, age, weight, and much more. For these reasons, the MealLogger app fails this standard.



MSTG-STORAGE-4

No sensitive data is shared with third parties unless it is a necessary part of the architecture.

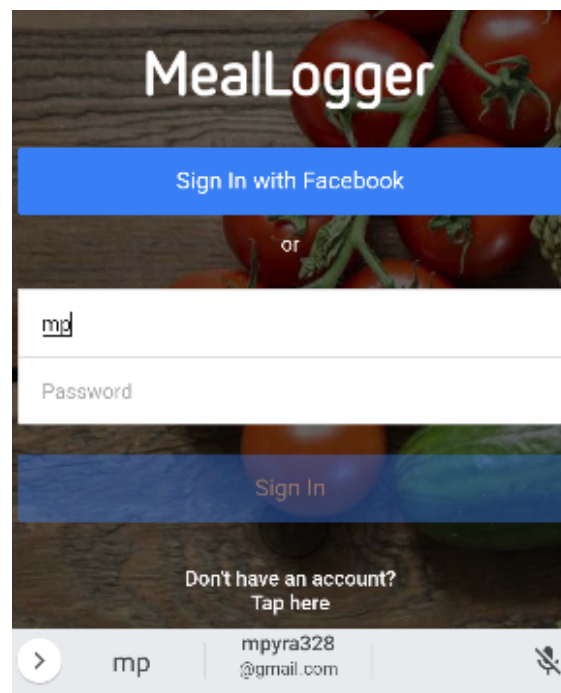
To perform dynamic app analysis for this standard, I used a Burp Suite proxy to intercept HTTP traffic from the application. From this, I found several instances of intercepted traffic, with one such example shown below. Although some of the information seems to be unencrypted, it is not clear whether or not sensitive information was sent to third parties. However, due to the potential and immediate danger of unencrypted data, the MealLogger app ultimately fails this standard.



MSTG-STORAGE-5

The keyboard cache is disabled on text inputs that process sensitive data.

By performing dynamic app analysis, I was able to test the keyboard cache for the MealLogger application. After trying all of the places where sensitive data could be inputted, I found that the login page, specifically where it asks for your email (not password), pulls up suggested strings (shown below). Therefore, the keyboard cache has not been disabled for this field, and the MealLogger app fails this requirement.



MSTG-STORAGE-6

No sensitive data is exposed via IPC mechanisms.

By searching the xml file in jadx, I was able to find significant results relating to content providers and exposed data. Firstly, there were a variety of “android:exported” flags, with some set to true and some set to false (shown below). Along this line of thinking, we can reason that some of these flags are not necessary and likely give unneeded read/write permissions. Similarly, although there were instances of “permission” flags, their protection levels were not properly set, leading to potential vulnerabilities.

```

</receiver>
<service android:name="com.google.android.gms.analytics.CampaignTrackingService" android:enabled="true" android:exported="false"/>
<activity android:name="com.desmond.squarecamera.CameraActivity"/>
<activity android:name="me.crosswall.photo.pick.PickPhotosActivity"/>
<activity android:name="com.adobe.phonegap.push.PushHandlerActivity" android:permission="com.wellnessfoundry.meallogger.android.permission.PushHandlerActivity" android:exported="true"/>
<receiver android:name="com.adobe.phonegap.push.BackgroundActionButtonHandler"/>
<receiver android:name="com.adobe.phonegap.push.PushDismissedHandler"/>
<receiver android:name="com.google.android.gms.gcm.GcmReceiver" android:permission="com.google.android.c2dm.permission.SEND" android:exported="true">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
        <category android:name="com.wellnessfoundry.meallogger.android"/>
    </intent-filter>
</receiver>

```

Furthermore, by opening the AndroidManifest.xml file in jadx and searching for “intent-filter”, we are able to see potential custom URL schemes. By analyzing the results (shown below), we can see that there is a custom URL scheme, but it seems to be empty and unused. Otherwise, there exist activities that can be opened and viewed in a browser, which are most likely related to in-app assets.

```

<application android:theme="@style/squarecamera__CameraFullScreenTheme" android:
    <activity android:theme="@android:style/Theme.DeviceDefault.NoActionBar" and
        <intent-filter android:label="@string/launcher_name">
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW"/>
            <category android:name="android.intent.category.DEFAULT"/>
            <category android:name="android.intent.category.BROWSABLE"/>
            <data android:scheme="meallogger"/>
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW"/>
            <category android:name="android.intent.category.DEFAULT"/>
            <category android:name="android.intent.category.BROWSABLE"/>
            <data android:scheme=" " android:host=" " android:pathPrefix="/" />
        </intent-filter>
    </activity>

```

By searching the manifest file for “provider”, we can see that there are a number of content providers used by the application (shown below). The first instance shows a plugin for a file provider, which is likely used to sync local and remote data. Moreover, the second instance shows a similar plugin, specifically related to the camera. Although neither of these are inherently malicious, it is important to note their presence.

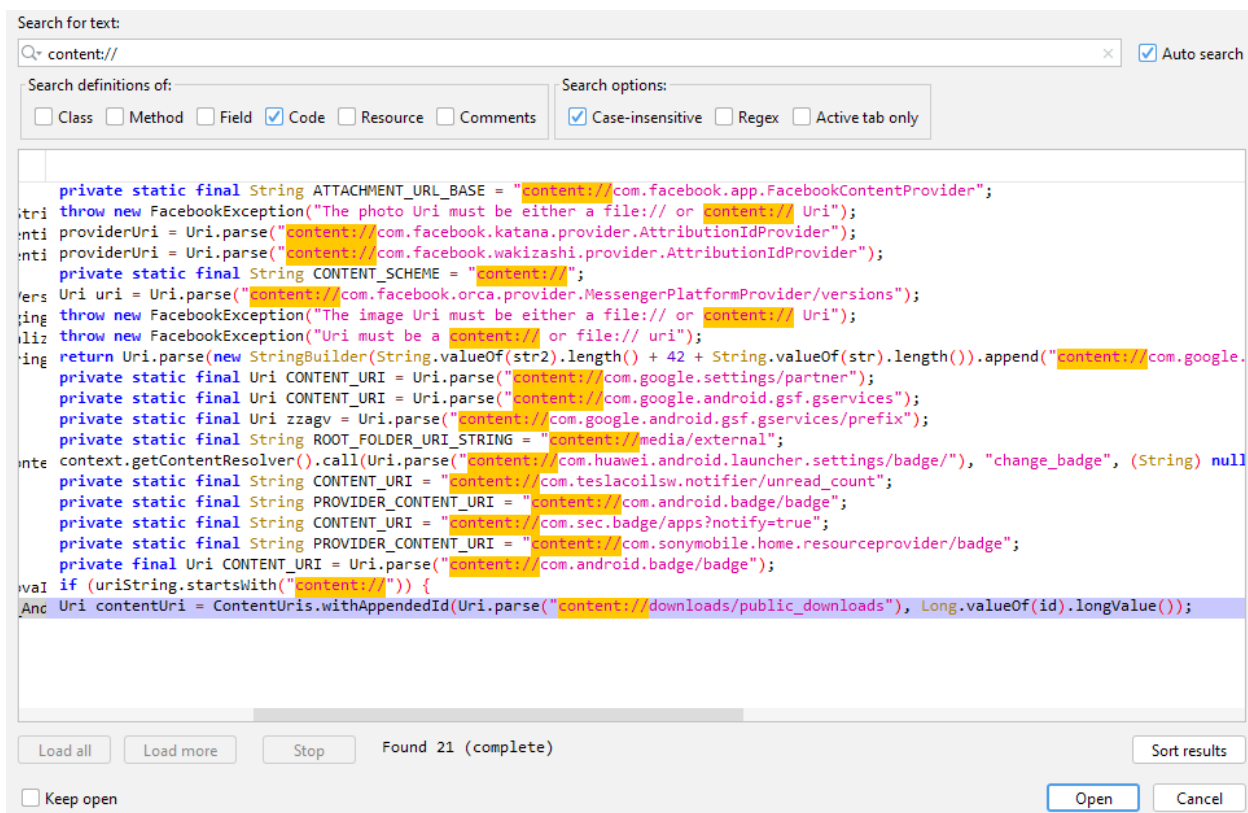
```

<?xml version='1.0' encoding='utf-8'>
<provider android:name="nl.xservices.plugins.FileProvider" android:exported="false" android:authorities="com
  <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/sharing_paths"/>
</provider>

<provider android:name="org.apache.cordova.camera.FileProvider" android:exported="false" android:authorities="com.w
  <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/camera_provider_paths"/>
</provider>

```

By opening the apk file in jadx and searching for “content://”, we are able to see that there are a number of URLs used in conjunction with the MealLogger application. Overall, most of them seems to be related to either a Facebook asset or a built-in android badge. Looking closer into the files where each instance is located, such assumptions are confirmed. Following this trend, using the adb content query yielded no significant results in relation to the URLs. For these reasons, the MealLogger app ultimately passes this standard.

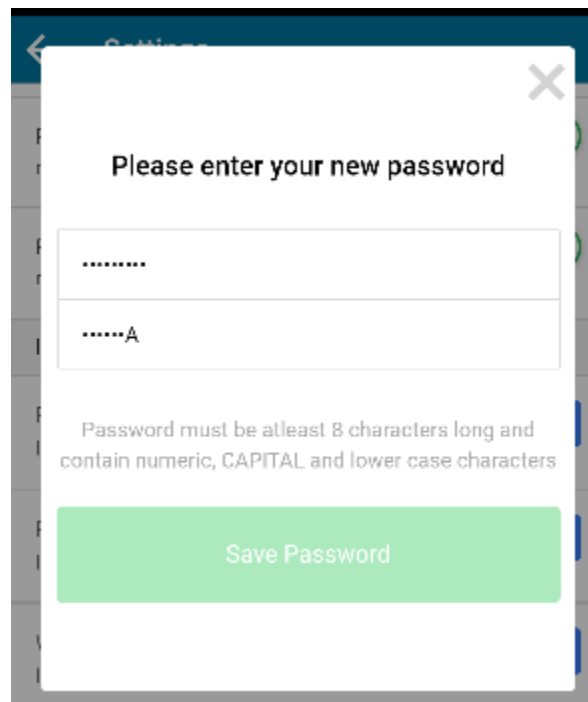


MSTG-STORAGE-7

No sensitive data, such as passwords or pins, is exposed through the user interface.

By performing dynamic app analysis, I was able to test whether or not the application leaks any sensitive data to the user interface. Essentially, for anywhere that you login, the password field seems to be protected by changing the actual input characters into dots (as shown below).

However, it is important to note that the actual plaintext characters do flash for an instant before turning into a dot. For these reasons, the MealLogger app ultimately passes this standard.

**MSTG-STORAGE-8**

No sensitive data is included in backups generated by the mobile operating system.

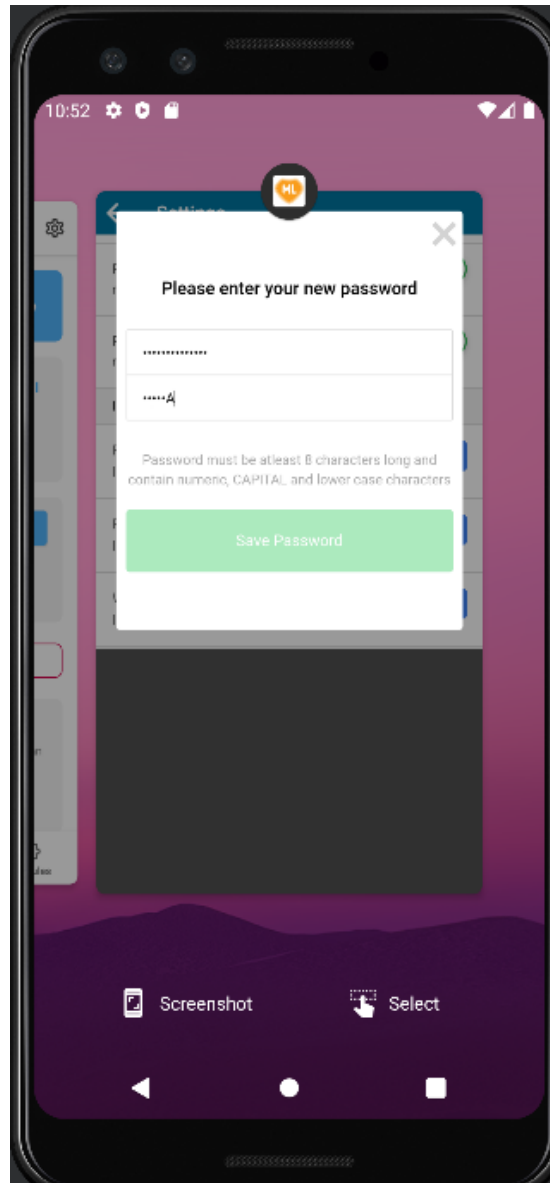
By performing dynamic app analysis, I was able to backup the data from my Android emulator. Essentially, I used the adb backup command, and unpacked the .ab file using the Android Backup Extractor (ABE) for Windows. From the unpacked files, I was able to find many similar results as found when investigating MSTG-STORAGE-1 and 2. The xml files were the same compared to the previous ones, but there were some new files under the Local Storage folder (shown below).

The “0000004.log” file (shown below), was quite similar to the log file found when investigating MSTG-STORAGE-3, containing sensitive information like name, email, DOB, address, and much more.

The other notable file was the “000005.ldb” file (shown below). Although most of it was unreadable, the parts that were readable showed similar sensitive information, like email and address. For these reasons, the MealLogger app ultimately fails this standard.



For static testing purposes, searching for the method “applicationDidEnterBackground” within jadx did not yield any results. This is further supported by the findings of the dynamic testing I performed. By typing in sensitive information, such as by changing the password, we can see that the snapshot taken in the app-switcher screen still contains this information (shown on the next page). For these reasons, the MealLogger app fails this requirement.



MSTG-STORAGE-10

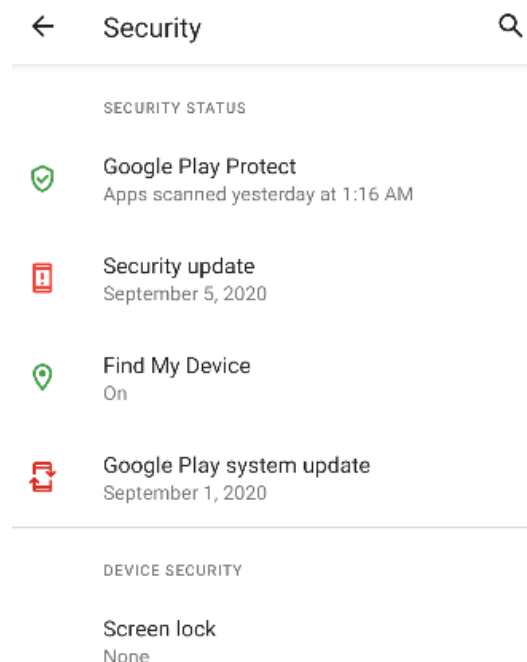
The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.

Similar to findings from previous sections, the MealLogger app makes no effort to clear memory or local storage after a given session. In fact, the log files are held within the local storage indefinitely, without being wiped or deleted under any normal circumstances. For these reasons, the app ultimately fails this standard.

MSTG-STORAGE-11

The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.

After completing manual testing, I was able to find that the MealLogger app does not enforce a minimum device-access security policy. By going into my emulated device's settings and disabling the screen lock password (shown below), I was still able to access the full functionality of the app, with no restrictions or warnings. For these reasons, the MealLogger app does not meet this standard.

**MSTG-STORAGE-12**

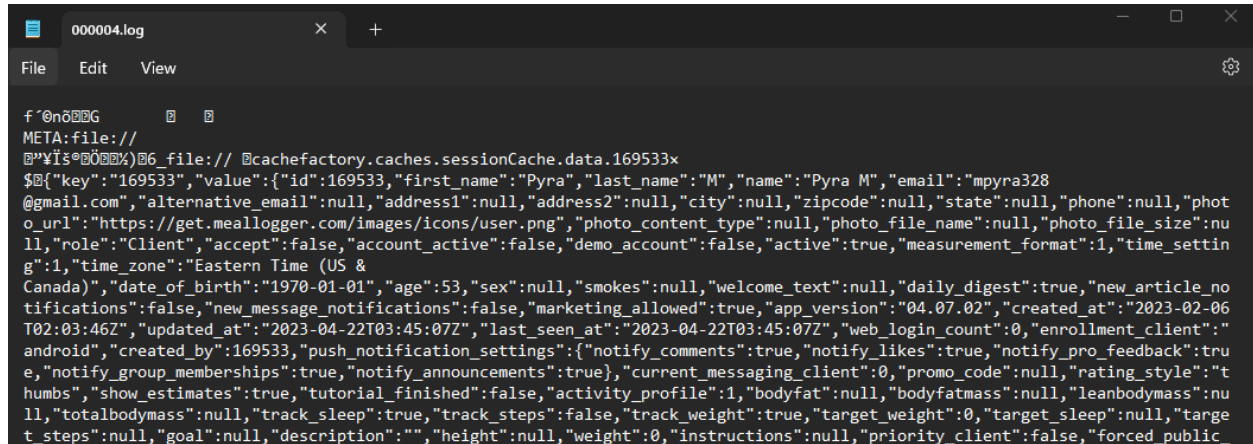
The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.

After completing manual testing and observation, I found no signs of user education regarding personally identifiable information, nor any sign of best security practices. Even when posting pictures/descriptions online, the app made no effort to warn or suggest best user practices regarding their sensitive information. For these reasons, the MealLogger app fails this requirement.

MSTG-STORAGE-13

No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.

As mentioned in the earlier sections of this requirement, there is clearly sensitive data, including things like name, email address, and date of birth, stored locally on the mobile device in the form of log files. For this reason, the MealLogger application fails this standard.



```

f'@n00G
META:file://
@Yis0000x@6_file:// @cachefactory.caches.sessionCache.data.169533x
$@{"key":"169533","value":{"id":169533,"first_name":"Pyra","last_name":"M","name":"Pyra M","email":"mpyra328@gmail.com","alternative_email":null,"address1":null,"address2":null,"city":null,"zipcode":null,"state":null,"phone":null,"photo_url":"https://get.meallogger.com/images/icons/user.png","photo_content_type":null,"photo_file_name":null,"photo_file_size":null,"role":"Client","accept":false,"account_active":false,"demo_account":false,"active":true,"measurement_format":1,"time_setting":1,"time_zone":"Eastern Time (US & Canada)","date_of_birth":"1970-01-01","age":53,"sex":null,"smokes":null,"welcome_text":null,"daily_digest":true,"new_article_notifications":false,"new_message_notifications":false,"marketing_allowed":true,"app_version":"04.07.02","created_at":"2023-02-06T02:03:46Z","updated_at":"2023-04-22T03:45:07Z","last_seen_at":"2023-04-22T03:45:07Z","web_login_count":0,"enrollment_client":"android","created_by":169533,"push_notification_settings":{"notify_comments":true,"notify_likes":true,"notify_pro_feedback":true,"notify_group_memberships":true,"notify_announcements":true},"current_messaging_client":0,"promo_code":null,"rating_style":"thumbs","show_estimates":true,"tutorial_finished":false,"activity_profile":1,"bodyfat":null,"bodyfatmass":null,"leanbodymass":null,"totalbodymass":null,"track_sleep":true,"track_steps":false,"track_weight":true,"target_weight":0,"target_sleep":null,"target_steps":null,"goal":null,"description":"","height":null,"weight":0,"instructions":null,"priority_client":false,"forced_public_

```

MSTG-STORAGE-14

If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.

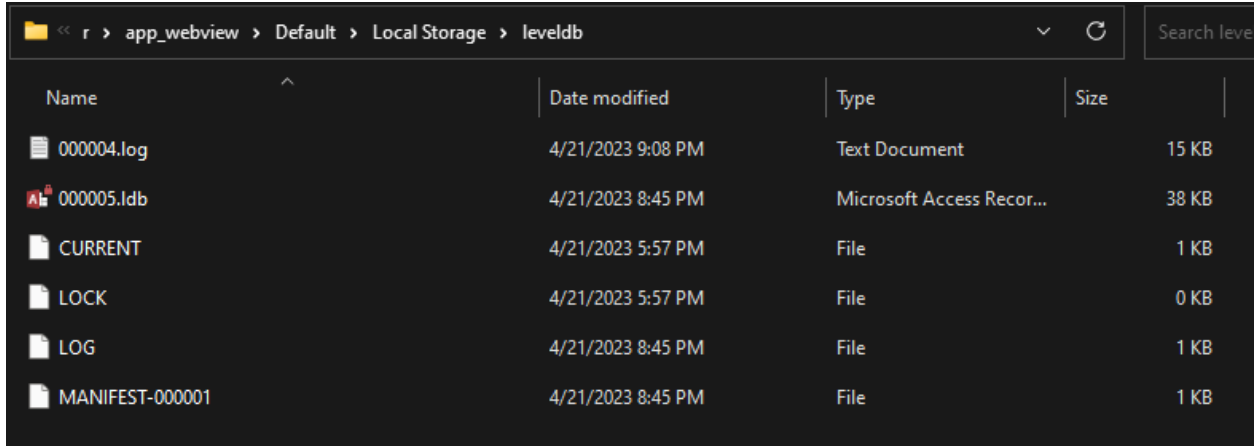
This requirement is applicable because sensitive data is in fact stored locally on the mobile device. By searching and analyzing the log files, shown in the previous sections, it is clear that the information provided, like sensitive personal identifiers, is not encrypted by any means and does not require any form of authentication to access. For these reasons, the MealLogger app does not meet this requirement.

MSTG-STORAGE-15

The app's local storage should be wiped after an excessive number of failed authentication attempts.

Considering this requirement, the MealLogger application does not seem to wipe the local storage (shown below) after an excessive number of failed authentication attempts. Although the

app locks you out for 10 minutes following an excessive number of failed logins (shown later in AUTH-6), the local storage is never deleted. For these reasons, the app fails this standard.



The screenshot shows a file explorer window with the address bar displaying the path: < r > app_webview > Default > Local Storage > leveldb. The search bar on the right contains the text 'Search leve'. The main area displays a list of files with the following columns: Name, Date modified, Type, and Size.

Name	Date modified	Type	Size
000004.log	4/21/2023 9:08 PM	Text Document	15 KB
000005.ldb	4/21/2023 8:45 PM	Microsoft Access Recor...	38 KB
CURRENT	4/21/2023 5:57 PM	File	1 KB
LOCK	4/21/2023 5:57 PM	File	0 KB
LOG	4/21/2023 8:45 PM	File	1 KB
MANIFEST-000001	4/21/2023 8:45 PM	File	1 KB

Cryptography Requirements

MSTG-CRYPTO-1

The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.

Regarding hardcoded secrets, the MobSF report notes 4 possible secrets, all related to Facebook’s device authentication instructions (shown below). Interestingly, by looking closer at the source code in jadx, these instances are not apparent, and the files themselves do not contain such information.

HARDCODED SECRETS

POSSIBLE SECRETS
"com_facebook_device_auth_instructions" : "facebook.com/deviceXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
"com_facebook_device_auth_instructions" : "XXfacebook.com/deviceXXXXXXXXXXXXXXXXXXXX"
"com_facebook_device_auth_instructions" : "XXXfacebook.com/deviceXXXXXXXXXXXX"

POSSIBLE SECRETS
"com_facebook_device_auth_instructions" : "000facebook.com/device0000000000000000"

However, MobSF further notes that other files outside of Facebook authentication also contain hardcoded information. Thus, the combination of these two findings means that the MealLogger application ultimately fails this standard.

<p>Files may contain hardcoded sensitive information like usernames, passwords, keys etc.</p>	<p>warning</p>	<p>CWE: CWE-312: Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14</p>	<p>bolts/MeasurementEvent.java com/adobe/phonegap/push/GCMIntentService.java com/adobe/phonegap/push/PushConstants.java com/bumptech/glide/load/engine/EngineKey.java com/desmond/squarecamera/CameraFragment.java com/desmond/squarecamera/EditSavePhotoFragment.java va io/intercom/com/bumptech/glide/load/engine/EngineKey.java rx/internal/schedulers/NewThreadWorker.java</p>
---	----------------	--	---

MSTG-CRYPTO-2

The app uses proven implementations of cryptographic primitives.

In terms of cryptography, the MealLogger application actually meets and follows the majority of encryption guidelines provided by OWASP. For instance, the MobSF report found that the implementation of asymmetric key generation and encryption met the given requirements (shown below). For these reasons, the MealLogger app passes this standard.

3	FCS_CKM_EXT.1.1	Security Functional Requirements	Cryptographic Key Generation Services	The application implement asymmetric key generation.
11	FCS_CKM.1.1(1)	Selection-Based Security Functional Requirements	Cryptographic Asymmetric Key Generation	The application generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm RSA schemes using cryptographic key sizes of 2048-bit or greater.

MSTG-CRYPTO-3

The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.

Similar to the previous standard, the MealLogger application uses cryptographic primitives that are appropriately configured for each use case. For instance, the MobSF report notes that the random bit generation services used for its cryptographic operations meet the given requirements (shown below). For these reasons, the MealLogger app passes this standard.

1	FCS_RBG_EXT.1.1	Security Functional Requirements	Random Bit Generation Services	The application invoke platform-provided DRBG functionality for its cryptographic operations.
---	---------------------------------	----------------------------------	--------------------------------	---

MSTG-CRYPTO-4

The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.

When considering this requirement, the MobSF report only notes one major problem, which relates to outdated hashing services (shown below). However, this issue is a very large one, as insufficient hashing can lead to other, bigger problems along the line. For this reason, the MealLogger application fails this standard.

FCS_COP.1.1(2)	Selection-Based Security Functional Requirements	Cryptographic Operation - Hashing	The application perform cryptographic hashing services not in accordance with FCS_COP.1.1(2) and uses the cryptographic algorithm RC2/RC4/MD4/MD5.
----------------	--	---	--

MSTG-CRYPTO-5

The app doesn't re-use the same cryptographic key for multiple purposes.

Note: Beyond the scope of this class.

MSTG-CRYPTO-6

All random values are generated using a sufficiently secure random number generator.

The MobSF report clearly notes that the application uses an insecure Random Number Generator (shown below). Thus, the MealLogger application fails this standard.

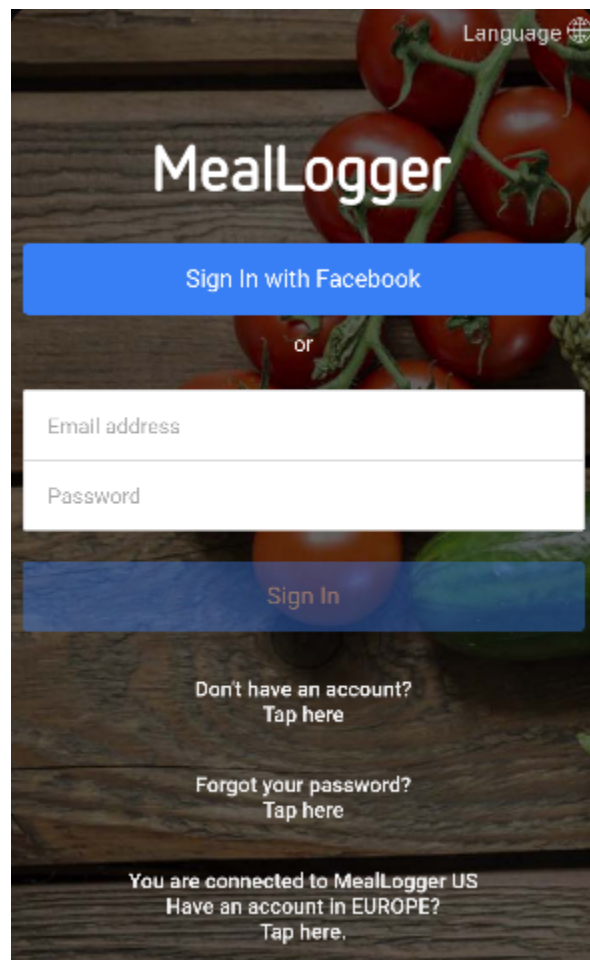
The App uses an insecure Random Number Generator.	warning	OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6	com/adobe/phonegap/push/GCMIntentService.java com/plugin/datepicker/DatePickerPlugin.java
---	---------	---	--

Authentication and Session Management Requirements

MSTG-AUTH-1

If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.

For authentication purposes, there are two main ways of logging into the MealLogger application: standard login and Facebook sign-in (shown below). Interestingly, the Facebook sign-in functionality seems to not work, at least in the emulator. Other notable functions include creating an account and password retrieval. Since MealLogger is an application that only has functionality following login, authentication is in fact required for access to remote resources. Thus, the MealLogger application passes this standard.



MSTG-AUTH-2

If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.

NOTE: Beyond the scope of this class.

MSTG-AUTH-3

If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.

NOTE: Beyond the scope of this class.

MSTG-AUTH-4

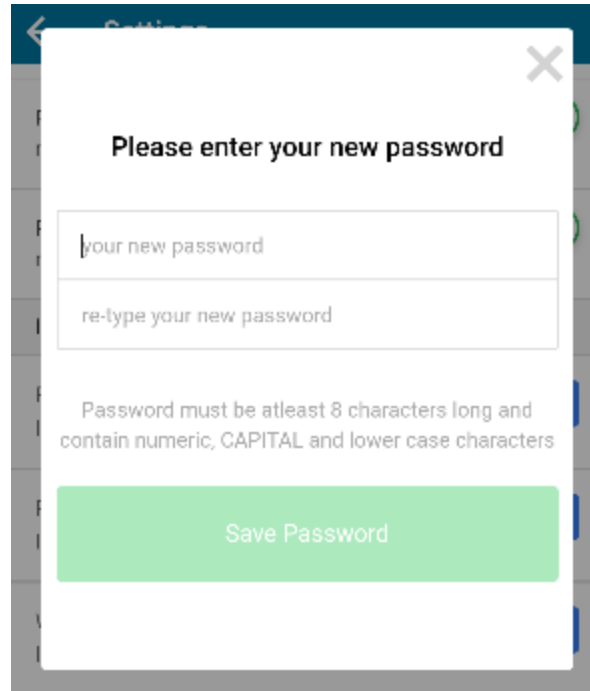
The remote endpoint terminates the existing session when the user logs out.

For this standard, the MealLogger application does have ways of terminating a session. If the user chooses to logout from within the application, then they will exit the session and return to the login page. From the login page, the user is not able to access any features of the app without logging in again. However, it is important to note that there is no other way to terminate a session. Simply minimizing or exiting the application does not seem to end the current session, but this is a common feature among many modern applications and does not pose an immediate problem. Since the user cannot access any remote or local information from the application without being logged in, the MealLogger application ultimately passes this standard.

MSTG-AUTH-5

A password policy exists and is enforced at the remote endpoint.

Regarding this standard, the MealLogger application does enforce a password policy for authentication purposes. In relation to password complexity, the password policy does meet three out of four rules provided by OWASP, namely: at least one uppercase character, one lowercase character, and one digit. However, while OWASP recommends a minimum password length of 10 characters, the app only requires 8 total characters, which is clearly below the required threshold. For this reason, the MealLogger application ultimately fails this standard.



Please enter your new password

your new password

re-type your new password

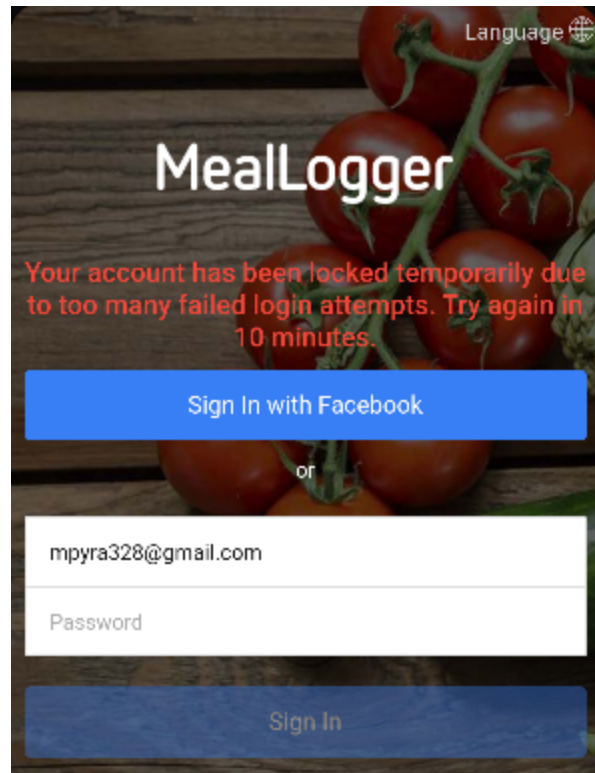
Password must be atleast 8 characters long and contain numeric, CAPITAL and lower case characters

Save Password

MSTG-AUTH-6

The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.

For this standard, the application does in fact have login throttling. After five failed login attempts, the account gets locked for 10 minutes (shown below). As expected, by submitting the correct credentials, the account is still locked within the 10 minute duration. After the 10 minutes are over, his number doesn't increase, but rather resets after a successful login followed by five failed logins. Overall, since the MealLogger application implements a mechanism against an excessive number of credential submissions, the app passes this standard.



MSTG-AUTH-7

Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.

For this standard, there seems to be no termination of the application after any amount of time. Leaving the application open over several hours yielded no results, and the application functioned normally after that extended period of time. Essentially, the access tokens never expired and the session was never invalidated at any point. For these reasons, the MealLogger application fails this standard.

MSTG-AUTH-8

Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.

NOTE: Beyond the scope of this class.

MSTG-AUTH-9

A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.

For this standard, the MealLogger application has no implementation of two-factor authentication. More specifically, past the first factor of a password check, there are no signs of a one-time password system or hardware/software tokens. Thus, the MealLogger application fails this standard.

MSTG-AUTH-10

Sensitive transactions require step-up authentication.

Regarding this standard, the MealLogger application has no implementation of step-up authentication for sensitive transactions. However, there are no in-app purchases and no way to link payment/bank account information, so the MealLogger application passes this standard by default.

MSTG-AUTH-11

The app informs the user of all sensitive activities with their account. Users are able to view a list of devices, view contextual information (IP address, location, etc.), and to block specific devices.

For this standard, it is clear that the MealLogger application makes no attempt to inform users of activities with their account. Furthermore, the application has no features regarding a device list or configuration logs. Even when changing the password within the application, there is no email sent to the user; the only communication that the app makes with the user is when resetting their password from the login screen. For these reasons, the MealLogger application ultimately fails this standard.

MSTG-AUTH-12

Authorization models should be defined and enforced at the remote endpoint.

Note: Beyond the scope of this class.

Network Communication Requirements

MSTG-NETWORK-1

Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.

By using Wireshark, I was able to complete dynamic testing and observe the data sent by the application. Specifically, by filtering for TLS, I was able to find that all data was in fact encrypted. Although specific instances are labeled with client/server hellos, the majority of them are simply labeled as application data, with all of the information being encrypted in hex-code and unreadable. For these reasons, the MealLogger app passes this standard.

No.	Time	Source	Destination	Protocol	Length	Info
360	0.509050	162.159.133.234	207.197.51.250	TLSv1.2	353	Application Data
1030	2.101807	207.197.51.250	35.153.244.46	QUIC	1292	Initial, DCID=a8fcf010f5ef60fb, PKN: 1, PING, PING, PADDING, PING, CRYPTO, PING, PADDING, CRYPTO, PADDING, PING, CRYPTO, PING,...
1035	2.117212	35.153.244.46	207.197.51.250	QUIC	1292	Protected Payload (KPE)
1216	2.601358	162.159.133.234	207.197.51.250	TLSv1.2	180	Application Data
1217	2.615613	162.159.133.234	207.197.51.250	TLSv1.2	102	Application Data
1489	3.012541	207.197.51.250	35.153.244.46	TLSv1.3	571	Client Hello
1531	3.074727	35.153.244.46	207.197.51.250	TLSv1.3	308	Server Hello, Change Cipher Spec, Application Data, Application Data
1532	3.077087	207.197.51.250	35.153.244.46	TLSv1.3	134	Change Cipher Spec, Application Data
1533	3.077984	207.197.51.250	35.153.244.46	TLSv1.3	146	Application Data
1534	3.078834	207.197.51.250	35.153.244.46	TLSv1.3	502	Application Data
1535	3.079179	207.197.51.250	35.153.244.46	TLSv1.3	141	Application Data
1581	3.138371	35.153.244.46	207.197.51.250	TLSv1.3	133	Application Data
1582	3.138371	35.153.244.46	207.197.51.250	TLSv1.3	116	Application Data
1584	3.138962	35.153.244.46	207.197.51.250	TLSv1.3	85	Application Data
1585	3.139463	207.197.51.250	35.153.244.46	TLSv1.3	85	Application Data
1586	3.139972	35.153.244.46	207.197.51.250	TLSv1.3	89	Application Data
1681	3.338319	35.153.244.46	207.197.51.250	TLSv1.3	696	Application Data
1682	3.338319	35.153.244.46	207.197.51.250	TLSv1.3	85	Application Data
1688	3.380627	207.197.51.250	35.153.244.46	TLSv1.3	184	Application Data
1689	3.380857	207.197.51.250	35.153.244.46	TLSv1.3	339	Application Data

Frame 360: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface \Device\NPF_{8E05C...}	0000	98 bb 1e 1c 8a c8 c0 c5	20 6b 5c ea 00 00 45 00 k\---E:
Ethernet II, Src: RuckusMI_6b:5c:ea (c0:c5:20:6b:5c:ea), Dst: BYDPrecl_1c:8a:c8 (98:bb:1e:1c:8a:c8)	0010	01 53 30 8d 40 00 37 06	e5 ce a2 9f 85 ea cf c5	..S0@.7:.....
Internet Protocol Version 4, Src: 162.159.133.234, Dst: 207.197.51.250	0020	33 fa 01 bb c2 68 85 74	1d fa 4e ba 88 a9 50 18	3.....h.t...N...P:
Transmission Control Protocol, Src Port: 443, Dst Port: 49768, Seq: 1, Ack: 1, Len: 299	0030	00 08 07 03 00 00 17 03	03 01 26 43 6a 37 d9 86&Cj7..
Transport Layer Security	0040	49 44 d0 3b ea 38 81 90	45 a5 81 9f 50 2f f0 0c	ID;:8:..E...P/..
	0050	3b 6c e4 84 01 2f f6 66	3e 03 fb 53 dc 70 00 4c	j1.../f>>5-p-L
	0060	14 ba c6 a8 4a 09 a9 f3	63 a5 13 e4 ea c8 8a 14	...:J...c.....
	0070	02 28 ba 2b 0d 2d e1 06	3e 42 fa 53 e7 69 62 99	-(+.....>B:5-ib-
	0080	3a 14 de 0d 6f 4a e5 a3	07 33 c0 6b 53 73 ee c9	:...:0J...3-k5s...
	0090	8b bb b7 5f 61 43 31 e5	d2 0c 6c f6 95 e2 5c ab	...C1...i...v
	00a0	48 62 0c 9c ba 40 13 e7	2d b0 c8 e9 d3 34 48 61	Hb...@:....4Ha
	00b0	8e 1a 50 15 bb 03 33 47	04 62 65 05 5b 7c e7 7d	..P...3G...be:[.}
	00c0	69 69 9f c1 9c 0d 19 80	f9 1a ee a3 88 7b 88 21	ii.....[.!
	00d0	17 1b 98 85 93 7b cb fe	ac 85 2d 61 73 c0 a9 90{...-as...
	00e0	c4 6e df 96 43 e5 6e 15	3b ae 5c 36 ed 78 a3 ea	..n-cC-m; y\6-x...
	00f0	03 a3 20 b9 55 ab 6b 27	1c d8 4f 38 fd 2b c2 0e	..U-k'...0B+...
	0100	cf ec 3f 07 68 41 f7 be	4d 8b 62 41 22 3c 06 fe	..7-hA...M-ba"e...
	0110	8b 8f 2a 9b a0 b8 1c d0	51 4d cf 4f 82 d3 67 b6	..+.....QM:0-g...
	0120	2b 7b e1 23 4d 50 05 37	33 e7 d2 72 2f aa a3 10	+(##MP-3...-/...
	0130	40 fa 39 b5 9b 5c 34 25	c2 fa b0 55 95 41 c4 62	@-9-..4X...U-A-b
	0140	66 d8 8b 8d 65 e1 3c ae	d6 f2 11 d5 d3 4c bd 41	f...e-c...L-A
	0150	5c 15 07 b2 df 06 4a c3	64 72 cf 6c 90 fa ee b6	\.....J: dr-l....
	0160	15		

MSTG-NETWORK-2

The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.

Note: Not applicable/testable for Android.

MSTG-NETWORK-3

The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.

By opening the apk file in jadx and searching the source code for keywords relating to the X509 certificate, I was able to find one major instance. Specifically, the screenshot below shows a class that implements the X509 Trust Manager. At first glance, this may seem like an acceptable implementation, but such a class closely resembles an improper one found in the OWASP github documentation. Essentially, this class overrides other instances of the X.509 certificate, and defaults to accepting all certificates provided to the application. For these reasons, the MealLogger application fails this requirement.

```
package com.google.android.gms.common.net;

import java.security.cert.X509Certificate;
import javax.net.ssl.X509TrustManager;

/* Loaded from: classes.dex */
final class zza implements X509TrustManager {
    @Override // javax.net.ssl.X509TrustManager
    public final void checkClientTrusted(X509Certificate[] x509CertificateArr, String str) {
    }

    @Override // javax.net.ssl.X509TrustManager
    public final void checkServerTrusted(X509Certificate[] x509CertificateArr, String str) {
    }

    @Override // javax.net.ssl.X509TrustManager
    public final X509Certificate[] getAcceptedIssuers() {
        return null;
    }
}
```

MSTG-NETWORK-4

The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.

Similar to the last standard, I was able to find a significant instance of the app's endpoint certificate and key handling. The screenshot on the next page demonstrates the system's default trust manager, which subsequently initializes a trusted KeyStore. Furthermore, it is clear that certain exceptions and errors are thrown when an untrusted certificate is found. In this sense, the MealLogger app ultimately passes this standard.


```

private X509TrustManager systemDefaultTrustManager() {
    try {
        TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        trustManagerFactory.init((KeyStore) null);
        TrustManager[] trustManagers = trustManagerFactory.getTrustManagers();
        if (trustManagers.length != 1 || !(trustManagers[0] instanceof X509TrustManager)) {
            throw new IllegalStateException("Unexpected default trust managers:" + Arrays.toString(trustManagers));
        }
        return (X509TrustManager) trustManagers[0];
    } catch (GeneralSecurityException e) {
        throw new AssertionError();
    }
}

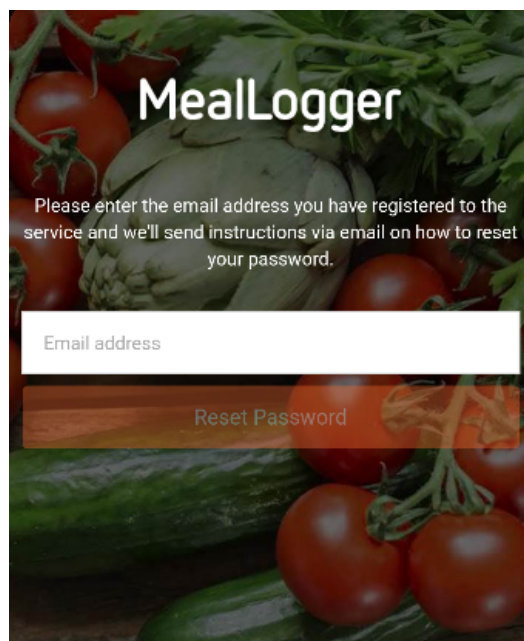
private SSLSocketFactory systemDefaultSslSocketFactory(X509TrustManager trustManager) {
    try {
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, new TrustManager[]{trustManager}, null);
        return sslContext.getSocketFactory();
    } catch (GeneralSecurityException e) {
        throw new AssertionError();
    }
}

```

MSTG-NETWORK-5

The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.

Looking at this requirement, the MealLogger application does in fact rely on a single insecure communication channel for critical operations. Namely, the app relies solely on email for account recovery via resetting the password. As shown in the screenshots below, by simply entering your email address, you are able to receive an email with a link to reset your password. For these reasons, the app fails this standard.



MealLogger Password Reset Inbox x**noreply@meallogger.com**

to me ▾

Hi Pyra,

A request to reset your password has been made.

To reset your password, click [HERE](#).

If you did not make this request, please ignore this email and your password will not be changed.

- The MealLogger Team

Enter new password

Your new password has to be atleast 8 characters long.

Set password

MSTG-NETWORK-6

The app only depends on up-to-date connectivity and security libraries.

By opening the apk file in jadx and observing the source code, I was able to find several functions relating to keeping the app version up-to-date. For instance, in the screenshot below, we can see functions that check the client/apk version, as well as if Google Play services are available. For these reasons, the MealLogger app meets this requirement.

```

@Deprecated
public static void ensurePlayServicesAvailable(Context context) throws GooglePlayServicesRepairableException, GooglePlayServicesNotAvailableException {
    ensurePlayServicesAvailable(context, GOOGLE_PLAY_SERVICES_VERSION_CODE);
}

@Deprecated
public static void ensurePlayServicesAvailable(Context context, int i) throws GooglePlayServicesRepairableException, GooglePlayServicesNotAvailableException {
    int isGooglePlayServicesAvailable = GoogleApiAvailabilityLight.getInstance().isGooglePlayServicesAvailable(context, i);
    if (isGooglePlayServicesAvailable != 0) {
        Intent errorResolutionIntent = GoogleApiAvailabilityLight.getInstance().getErrorResolutionIntent(context, isGooglePlayServicesAvailable, "e");
        Log.e("GooglePlayServicesUtil", new StringBuilder(57).append("GooglePlayServices not available due to error ").append(isGooglePlayServicesAvailable).toString());
        if (errorResolutionIntent != null) {
            throw new GooglePlayServicesRepairableException(isGooglePlayServicesAvailable, "Google Play Services not available", errorResolutionIntent);
        }
        throw new GooglePlayServicesNotAvailableException(isGooglePlayServicesAvailable);
    }
}

@Deprecated
public static int getApkVersion(Context context) {
    try {
        return context.getPackageManager().getPackageInfo("com.google.android.gms", 0).versionCode;
    } catch (PackageManager.NameNotFoundException e) {
        Log.w("GooglePlayServicesUtil", "Google Play services is missing.");
        return 0;
    }
}

@Deprecated
public static int getClientVersion(Context context) {
    Preconditions.checkNotNull(context);
    return ClientLibraryUtils.getClientVersion(context, context.getPackageName());
}

```

Platform Interaction Requirements

MSTG-PLATFORM-1

The app only requests the minimum set of permissions necessary.

By opening the AndroidManifest.xml file (shown below) in jadx and searching for the permission keyword, we can see the various permissions allowed by the MealLogger application. By looking closely at the actual permissions included, it seems that the read/write permissions for both external storage and settings are clearly suspicious and perhaps unneeded.

```

Find: permission
<?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="571" android:versionName="4.7.2" android:hardwareAccelerated="true">
8   <uses-sdk android:minSdkVersion="19" android:targetSdkVersion="26"/>
12  <supports-screens android:anyDensity="true" android:normalScreens="true" android:smallScreens="true" android:largeScreens="true" android:resizeableActivity="true"/>
20  <uses-permission android:name="android.permission.INTERNET"/>
21  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
22  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
24  <uses-feature android:name="android.hardware.camera" android:required="true"/>
26  <uses-permission android:name="android.permission.CAMERA"/>
27  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
28  <uses-permission android:name="android.permission.WAKE_LOCK"/>
29  <uses-permission android:name="android.permission.VIBRATE"/>
30  <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
31  <uses-permission android:name="com.wellnessfoundry.meallogger.permission.C2D_MESSAGE"/>
32  <uses-permission android:name="com.wellnessfoundry.meallogger.permission.PushHandlerActivity"/>
34  <permission android:name="com.wellnessfoundry.meallogger.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
37  <permission android:name="com.wellnessfoundry.meallogger.permission.PushHandlerActivity" android:protectionLevel="signature"/>
41  <meta-data android:name="android.support.VERSION" android:value="25.3.1"/>
49  <uses-permission android:name="com.sec.android.provider.badge.permission.READ"/>
50  <uses-permission android:name="com.sec.android.provider.badge.permission.WRITE"/>
51  <uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS"/>
52  <uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT"/>
53  <uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE"/>
54  <uses-permission android:name="com.sonymobile.home.permission.PROVIDER_INSERT_BADGE"/>
55  <uses-permission android:name="com.anddoes.launcher.permission.UPDATE_COUNT"/>
56  <uses-permission android:name="com.majeur.launcher.permission.UPDATE_BADGE"/>
57  <uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE"/>
58  <uses-permission android:name="com.huawei.android.launcher.permission.READ_SETTINGS"/>
59  <uses-permission android:name="com.huawei.android.launcher.permission.WRITE_SETTINGS"/>
60  <uses-permission android:name="android.permission.READ_APP_BADGE"/>
61  <uses-permission android:name="com.oppo.launcher.permission.READ_SETTINGS"/>
62  <uses-permission android:name="com.oppo.launcher.permission.WRITE_SETTINGS"/>

```

Furthermore, by running the command:

adb shell dumpsys package com.wellnessfoundry.MealLogger.android

I was able to find a similar list of permissions, but this output has been separated into declared permissions, requested/installed permissions, and runtime permissions (shown below). In the same nature as the AndroidManifest.xml file, we can see that the read/write permissions for external storage and settings are requested by the application. Looking at the custom (declared) permissions, the C2D_MESSAGE allows the cloud to communicate with the device, while the PushHandlerActivity allows for custom notifications. Although not apparently malicious, it is important to note the existence of these custom permissions. For these reasons, the MealLogger app ultimately fails this standard.

```

declared permissions:
  com.wellnessfoundry.meallogger.android.permission.C2D_MESSAGE: prot=signature, INSTALLED
  com.wellnessfoundry.meallogger.android.permission.PushHandlerActivity: prot=signature, INSTALLED
requested permissions:
  android.permission.INTERNET
  android.permission.WRITE_EXTERNAL_STORAGE: restricted=true
  android.permission.ACCESS_NETWORK_STATE
  android.permission.CAMERA
  android.permission.READ_EXTERNAL_STORAGE: restricted=true
  android.permission.WAKE_LOCK
  android.permission.VIBRATE
  com.google.android.c2dm.permission.RECEIVE
  com.wellnessfoundry.meallogger.android.permission.C2D_MESSAGE
  com.wellnessfoundry.meallogger.android.permission.PushHandlerActivity
  com.sec.android.provider.badge.permission.READ
  com.sec.android.provider.badge.permission.WRITE
  com.htc.launcher.permission.READ_SETTINGS
  com.htc.launcher.permission.UPDATE_SHORTCUT
  com.sonyericsson.home.permission.BROADCAST_BADGE
  com.sonymobile.home.permission.PROVIDER_INSERT_BADGE
  com.anddoes.launcher.permission.UPDATE_COUNT
  com.majeur.launcher.permission.UPDATE_BADGE
  com.huawei.android.launcher.permission.CHANGE_BADGE
  com.huawei.android.launcher.permission.READ_SETTINGS
  com.huawei.android.launcher.permission.WRITE_SETTINGS
  android.permission.READ_APP_BADGE
  com.oppo.launcher.permission.READ_SETTINGS
  com.oppo.launcher.permission.WRITE_SETTINGS
  android.permission.ACCESS_MEDIA_LOCATION
install permissions:
  com.google.android.c2dm.permission.RECEIVE: granted=true
  com.wellnessfoundry.meallogger.android.permission.PushHandlerActivity: granted=true
  android.permission.INTERNET: granted=true
  android.permission.ACCESS_NETWORK_STATE: granted=true
  android.permission.VIBRATE: granted=true
  com.wellnessfoundry.meallogger.android.permission.C2D_MESSAGE: granted=true
  android.permission.WAKE_LOCK: granted=true

```

MSTG-PLATFORM-2

All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.

By opening the AndroidManifest.xml file in jadx and searching for “intent-filter”, we are able to see potential custom URL schemes. By analyzing the results (shown below), we can see that there is a custom URL scheme, but it seems to be empty and unused. Otherwise, there exist activities that can be opened and viewed in a browser, which are most likely related to in-app assets.

```

<application android:theme="@style/squarecamera__CameraFullScreenTheme" android:
  <activity android:theme="@android:style/Theme.DeviceDefault.NoActionBar" and
    <intent-filter android:label="@string/launcher_name">
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
    <intent-filter>
      <action android:name="android.intent.action.VIEW"/>
      <category android:name="android.intent.category.DEFAULT"/>
      <category android:name="android.intent.category.BROWSABLE"/>
      <data android:scheme="meallogger"/>
    </intent-filter>
    <intent-filter>
      <action android:name="android.intent.action.VIEW"/>
      <category android:name="android.intent.category.DEFAULT"/>
      <category android:name="android.intent.category.BROWSABLE"/>
      <data android:scheme=" " android:host=" " android:pathPrefix="/" />
    </intent-filter>
  </activity>

```

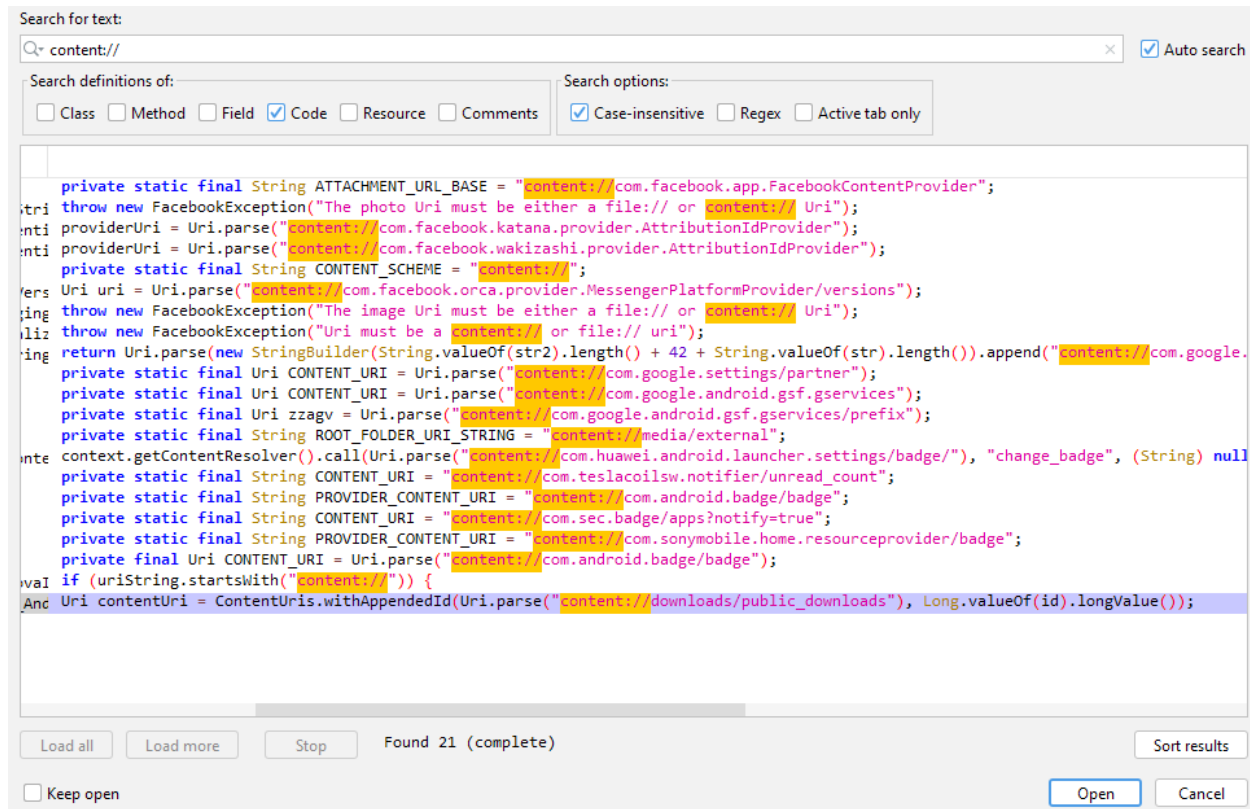
By searching the manifest file for “provider”, we can see that there are a number of content providers used by the application (shown below). The first instance shows a plugin for a file provider, which is likely used to sync local and remote data. Moreover, the second instance shows a similar plugin, specifically related to the camera. Although neither of these are inherently malicious, it is important to note their presence.

```

<receiver>
<provider android:name="nl.xservices.plugins.FileProvider" android:exported="false" android:authorities="com
  <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/sharing_paths"/>
</provider>
...
<provider android:name="org.apache.cordova.camera.FileProvider" android:exported="false" android:authorities="com.w
  <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/camera_provider_paths"/>
</provider>
...

```

By opening the apk file in jadx and searching for “content://”, we are able to see that there are a number of URLs used in conjunction with the MealLogger application. Overall, most of them seems to be related to either a Facebook asset or a built-in android badge. Looking closer into the files where each instance is located, such assumptions are confirmed. Following this trend, using the adb content query yielded no significant results in relation to the URLs.



Finally, looking into the AndroidManifest.xml file, there was no sign of

android.webkit.WebView.EnableSafeBrowsing

Thus, this means that safe browsing is enabled by default in the application. For these reasons, the MealLogger app meets this requirement.

MSTG-PLATFORM-3

The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.

By looking for the data keyword in the intent-filter blocks present within the AndroidManifest.xml file, I was able to find two instances (shown below). As mentioned in the previous section, there seems to be an android scheme labeling the app itself, as well as an empty URL scheme. Due to the lack of exposure of sensitive functionality due to custom URL schemes, the MealLogger app ultimately passes this standard.

```

<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <category android:name="android.intent.category.BROWSABLE"/>
  <data android:scheme="meallogger"/>
</intent-filter>
<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <category android:name="android.intent.category.BROWSABLE"/>
  <data android:scheme=" " android:host=" " android:pathPrefix="/" />

```

MSTG-PLATFORM-4

The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.

By opening the AndroidManifest.xml file in jadx and searching for “intent-filter” under another component element, we are able to see a number of exported activities from the MealLogger application (shown below). In the first screenshot, it is clear that the intent-filter contents are related to activity that can be viewed and are explicitly browsable. In the second and third screenshots, we can see more signs of exported receivers and services. While many of these lines are non-impactful, the “CampaignTrackingReceiver” for Google Analytics and “IDLListenerService” seem relatively unneeded by the application. Other than those instances, there are no activities present in the xml file with an “exported=true” flag. Overall, since the app does not explicitly export sensitive functionality through unprotected IPC facilities, MealLogger meets this requirement.

```

<activity android:theme="@android:style/Theme.DeviceDefault.NoActionBar" an
  <intent-filter android:label="@string/launcher_name">
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="meallogger"/>
  </intent-filter>
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme=" " android:host=" " android:pathPrefix="/" />
  </intent-filter>
</activity>

```

```

<receiver android:name="nl.xservices.plugins.ShareChooserPendingIntent" android:enabled="true">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
  </intent-filter>
</receiver>
<provider android:name="nl.xservices.plugins.FileProvider" android:exported="false" android:authorities="com.wellnessfoundry.meallo"
  <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/sharing_paths"/>
</provider>
<service android:name="io.intercom.android.sdk.IntercomIntentService" android:exported="false">
  <intent-filter android:priority="999">
    <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
  </intent-filter>
</service>
<receiver android:name="com.google.android.gms.analytics.AnalyticsReceiver" android:enabled="true">
  <intent-filter>
    <action android:name="com.google.android.gms.analytics.ANALYTICS_DISPATCH"/>
  </intent-filter>
</receiver>
<service android:name="com.google.android.gms.analytics.AnalyticsService" android:enabled="true" android:exported="false"/>
<receiver android:name="com.google.android.gms.analytics.CampaignTrackingReceiver" android:enabled="true" android:exported="true">
  <intent-filter>
    <action android:name="com.android.vending.INSTALL_REFERRER"/>
  </intent-filter>
</receiver>

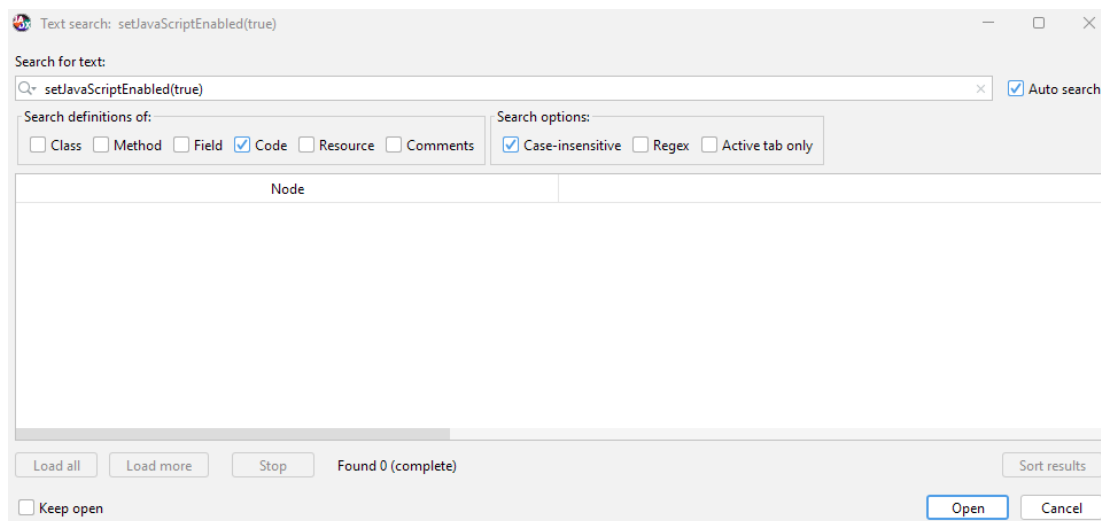
<receiver android:name="com.google.android.gms.gcm.GcmReceiver" android:permission="com.google.android.c2dm.permission.SEND" android:exported="true">
  <intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
    <category android:name="com.wellnessfoundry.meallogger.android"/>
  </intent-filter>
</receiver>
<service android:name="com.adobe.phonegap.push.GCMIntentService" android:exported="false">
  <intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
  </intent-filter>
</service>
<service android:name="com.adobe.phonegap.push.PushInstanceIDListenerService" android:exported="false">
  <intent-filter>
    <action android:name="com.google.android.gms.iid.InstanceID"/>
  </intent-filter>
</service>

```

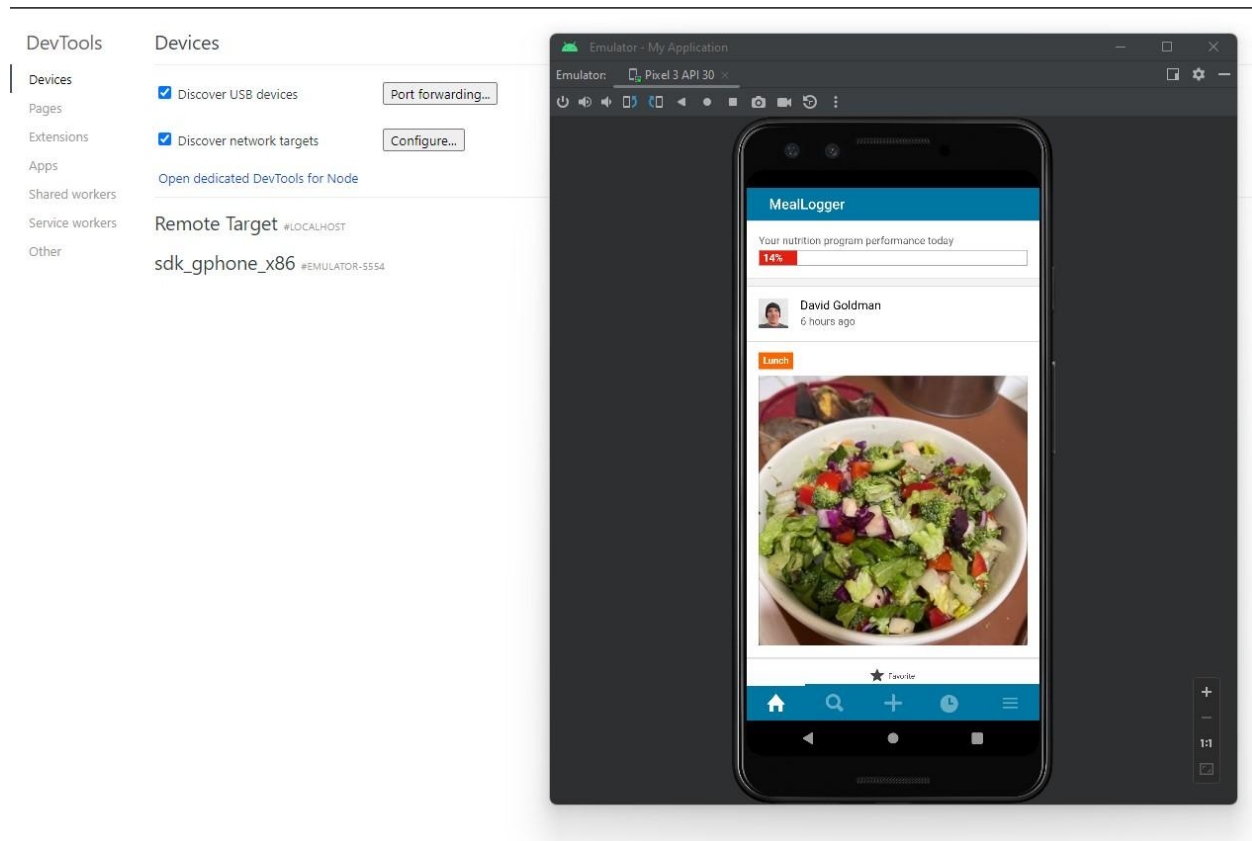
MSTG-PLATFORM-5

JavaScript is disabled in WebViews unless explicitly required.

Searching for “setJavaScriptEnabled(true)” in jadx yielded no results, as shown below.



Furthermore, by using Google's remote debugging feature, the MealLogger application was found not to use WebView components (as shown below), so this requirement is passed by default.



MSTG-PLATFORM-6

WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.

Note: Not applicable, due to lack of WebView component(s).

MSTG-PLATFORM-7

If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.

Note: Not applicable, due to lack of WebView component(s).

MSTG-PLATFORM-8

Object deserialization, if any, is implemented using safe serialization APIs.

By opening the apk file in jadx, I was able to perform analysis for this standard. At first glance, it seems that the instance of the java.io.Serializable component means that the MealLogger app implements unsafe serialization APIs, especially since this instance relates to access tokens and application IDs (shown below).

```
import java.io.Serializable;

/* JADX INFO: Access modifiers changed from: package-private */
/* Loaded from: classes.dex */
public class AccessTokenAppIdPair implements Serializable {
    private static final long serialVersionUID = 1;
    private final String accessTokenString;
    private final String applicationId;
```

However, by taking a closer look into the specific class file, we can see that the app actually implements a serialization proxy, which is considered to be a safe implementation of serialization APIs (shown below). For these reasons, the MealLogger app has met this requirement.

```
/* Loaded from: classes.dex */
static class SerializationProxyV1 implements Serializable {
    private static final long serialVersionUID = -2488473066578201069L;
    private final String accessTokenString;
    private final String appId;

    private SerializationProxyV1(String accessTokenString, String appId) {
        this.accessTokenString = accessTokenString;
        this.appId = appId;
    }

    private Object readResolve() {
        return new AccessTokenAppIdPair(this.accessTokenString, this.appId);
    }

    private Object writeReplace() {
        return new SerializationProxyV1(this.accessTokenString, this.applicationId);
    }
}
```

MSTG-PLATFORM-9

The app protects itself against screen overlay attacks. (Android only)

Note: Beyond the scope of this class.

MSTG-PLATFORM-10

A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed.

Note: Not applicable, due to lack of WebView component(s).

MSTG-PLATFORM-11

Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered (iOS only).

Note: Not applicable for Android (iOS only).

Code Quality and Build Setting Requirements

MSTG-CODE-1

The app is signed and provisioned with a valid certificate, of which the private key is properly protected.

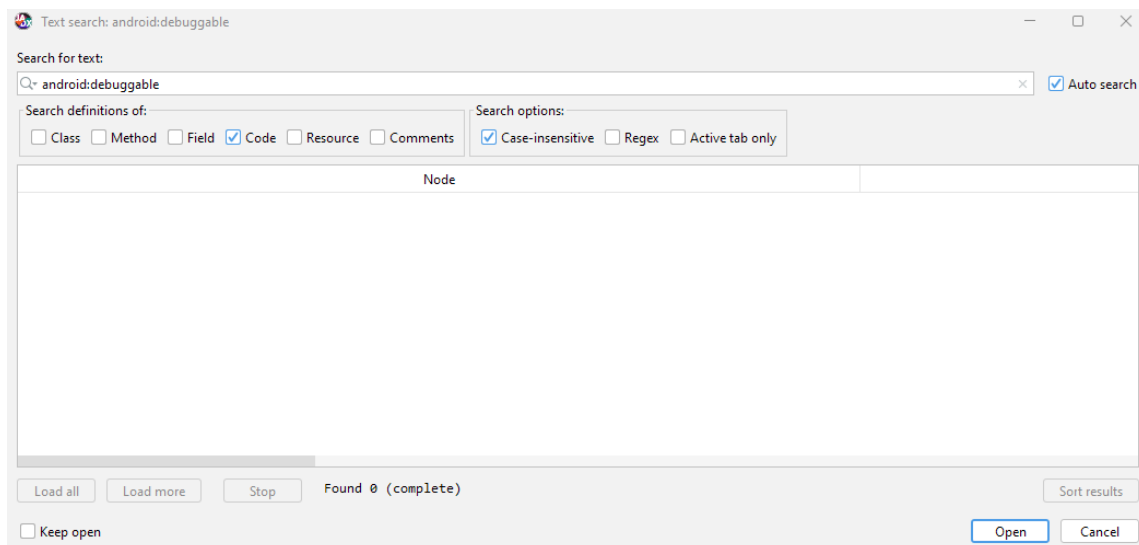
By running the `apksigner verify` command in the terminal (shown below), we can see that the MealLogger application has both V1 and V2 signatures. Thus, the MealLogger app has been signed with a valid certificate and passes this standard.

```
Verifies
Verified using v1 scheme (JAR signing): true
Verified using v2 scheme (APK Signature Scheme v2): true
Verified using v3 scheme (APK Signature Scheme v3): false
Verified using v4 scheme (APK Signature Scheme v4): false
Verified for SourceStamp: false
Number of signers: 1
```

MSTG-CODE-2

The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).

By opening the `AndroidManifest.xml` file in jadx and searching for the “`android:debuggable`” attribute, there were no instances of this attribute. For this reason, the MealLogger app was built in release mode and meets this requirement.



MSTG-CODE-3

Debugging symbols have been removed from native binaries.

Note: Not testable, due to lack of native binaries.

MSTG-CODE-4

Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.

After checking the MobSF report, there were no signs of the specified strings. However, by searching for a variety of keywords in jadx, I was able to find various instances for “log”, “error”, and “StrictMode”. Below are screenshots and descriptions of the results:

```
public static IBinder getBinder(Bundle bundle, String key) {
    if (!sGetIBinderMethodFetched) {
        try {
            sGetIBinderMethod = Bundle.class.getMethod("getIBinder", String.class);
            sGetIBinderMethod.setAccessible(true);
        } catch (NoSuchMethodException e) {
            Log.i(TAG, "Failed to retrieve getIBinder method", e);
        }
        sGetIBinderMethodFetched = true;
    }
    if (sGetIBinderMethod != null) {
        try {
            return (IBinder) sGetIBinderMethod.invoke(bundle, key);
        } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException e2) {
            Log.i(TAG, "Failed to invoke getIBinder via reflection", e2);
            sGetIBinderMethod = null;
        }
    }
    return null;
}
```

“log”: For this keyword, we can see that the source code does in fact log information in the system, using a public class. Although the effects of this are not easily apparent, logging sensitive information on the system’s memory is bad practice, as similarly noted by MobSF.

```

private final FutureTask<Result> mFuture = new FutureTask<Result>(this.mWorker) { // from class: android.support.v4.content.ModernAsyncTask.3
    @Override // java.util.concurrent.FutureTask
    protected void done() {
        try {
            Result result = get();
            ModernAsyncTask.this.postResultIfNotInvoked(result);
        } catch (InterruptedException e) {
            Log.w(ModernAsyncTask.LOG_TAG, e);
        } catch (CancellationException e2) {
            ModernAsyncTask.this.postResultIfNotInvoked(null);
        } catch (ExecutionException e3) {
            throw new RuntimeException("An error occurred while executing doInBackground()", e3.getCause());
        } catch (Throwable t) {
            throw new RuntimeException("An error occurred while executing doInBackground()", t);
        }
    }
};

```

“error”: For this keyword, we can see that there is in fact an instance of verbose error messaging. Although it is unclear exactly what the error is referring to, its existence alone is dangerous enough.

```

public final T get() {
    boolean z;
    HashSet<String> hashSet;
    Context context;
    T retrieve;
    if (this.zzmz != null) {
        return this.zzmz;
    }
    StrictMode.ThreadPolicy allowThreadDiskReads = StrictMode.allowThreadDiskReads();
    synchronized (sLock) {
        z = zzmw != null && zzd(zzmw);
        hashSet = zzmy;
        context = zzmw;
    }
}

```

“StrictMode”: For this keyword, we can see a very significant instance of StrictMode. In this case, the application is able to read from the system’s disk, which can lead to many potential vulnerabilities.

For these reasons, the MealLogger app ultimately fails this standard.

MSTG-CODE-5

All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.

Note: Not testable, due to lack of high-level architecture.

MSTG-CODE-6

The app catches and handles possible exceptions.

By opening the apk file in jadx and searching for exception handling, I was able to find several occurrences of such code. For instance, shown below is a specific instance of exception handling regarding the validity of the apk file. Other instances relate to class-specific errors, including things like runtime and illegal argument errors on a case-by-case basis. For these reasons, the MealLogger app ultimately passes this standard.

```
public static long getZipCrc(File apk) throws IOException {
    RandomAccessFile raf = new RandomAccessFile(apk, "r");
    try {
        CentralDirectory dir = findCentralDirectory(raf);
        return computeCrcOfCentralDir(raf, dir);
    } finally {
        raf.close();
    }
}

static CentralDirectory findCentralDirectory(RandomAccessFile raf) throws IOException, ZipException {
    long scanOffset = raf.length() - 22;
    if (scanOffset < 0) {
        throw new ZipException("File too short to be a zip file: " + raf.length());
    }
    long stopOffset = scanOffset - PlaybackStateCompat.ACTION_PREPARE_FROM_SEARCH;
    if (stopOffset < 0) {
        stopOffset = 0;
    }
    int endSig = Integer.reverseBytes(ENDSIG);
    do {
        raf.seek(scanOffset);
        if (raf.readInt() != endSig) {
            scanOffset--;
        } else {
            raf.skipBytes(2);
            raf.skipBytes(2);
            raf.skipBytes(2);
            raf.skipBytes(2);
            CentralDirectory dir = new CentralDirectory();
            dir.size = Integer.reverseBytes(raf.readInt()) & 4294967295L;
            dir.offset = Integer.reverseBytes(raf.readInt()) & 4294967295L;
            return dir;
        }
    } while (scanOffset >= stopOffset);
    throw new ZipException("End Of Central Directory signature not found");
}
```

MSTG-CODE-7

Error handling logic in security controls denies access by default.

Similar to the last standard, by opening the apk file in jadx and searching for security-related error handling, I was able to find several occurrences in the code. For instance, shown below is a specific instance of error handling relating to provider permissions. In the provided code, we can

clearly see that access is in fact denied by default, especially relating to throwing the given exceptions. For these reasons, the MealLogger app meets this requirement.

```
public void attachInfo(@NonNull Context context, @NonNull ProviderInfo info) {
    super.attachInfo(context, info);
    if (info.exported) {
        throw new SecurityException("Provider must not be exported");
    }
    if (!info.grantUriPermissions) {
        throw new SecurityException("Provider must grant uri permissions");
    }
    this.mStrategy = getPathStrategy(context, info.authority);
}

public static Uri getUriForFile(@NonNull Context context, @NonNull String authority, @NonNull File file) {
    PathStrategy strategy = getPathStrategy(context, authority);
    return strategy.getUriForFile(file);
}
```

MSTG-CODE-8

In unmanaged code, memory is allocated, freed and used securely.

Note: Beyond the scope of this class.

MSTG-CODE-9

Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.

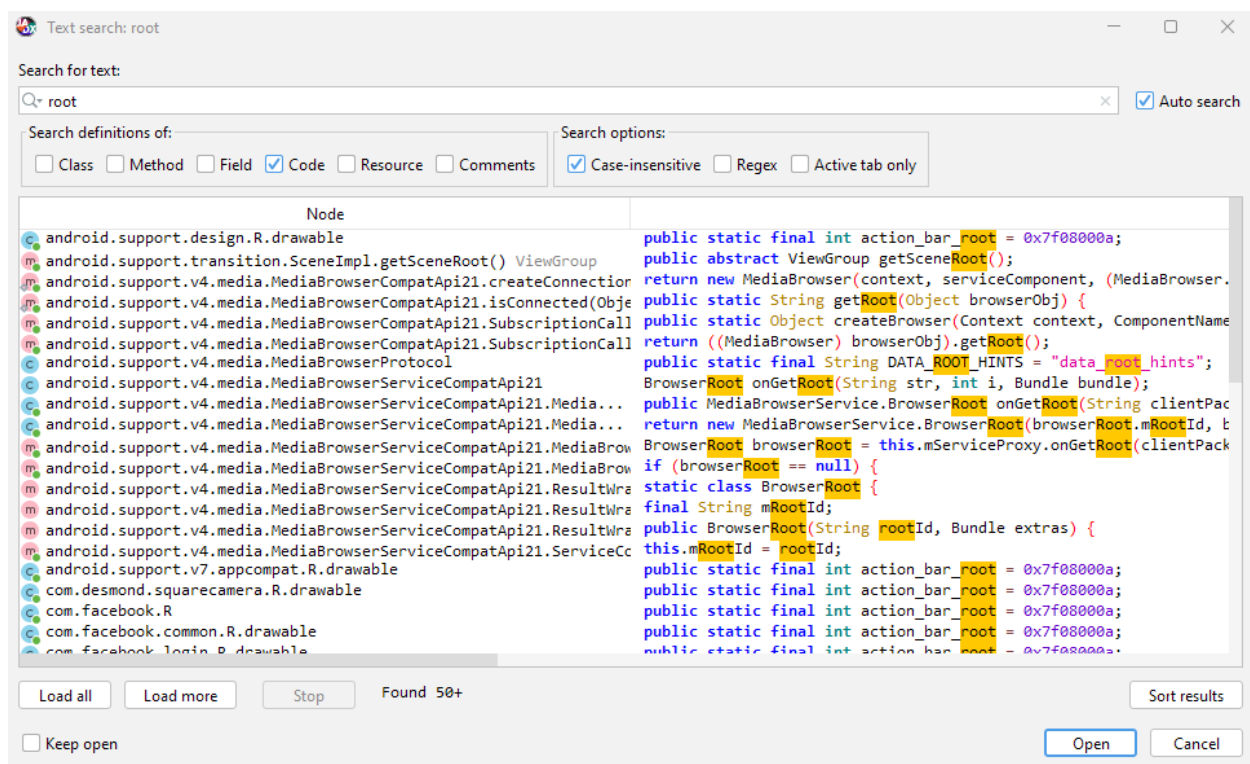
Note: Beyond the scope of this class.

Resilience Requirements

MSTG-RESILIENCE-1

The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.

After investigation, the MealLogger application does not seem to have any implementation of root detection. Searching jadx for common root keywords, like superuser or /bin/su, did not return any search results. Searching for “root”, as shown below, also does not return any meaningful results, as most instances refer to the instantiation of a browser rather than a root user. In the same sense, due to the lack of related code, there is no apparent obfuscation for root detection.



Similarly, opening the MealLogger application in a rooted virtual device on Android Studio did not yield any significant results. The application works exactly as intended, and there were no popups or limitations in rooted mode. Thus, the MealLogger application ultimately fails this standard.

MSTG-RESILIENCE-2

The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.

By opening the file in jadx and searching for debugging keywords, I was able to find a few significant occurrences. In the two screenshots below, we can see two instances of debugger detection within the MealLogger app. Furthermore, specifically in the second screenshot, it seems that a warning related to slower execution is triggered when it is found that a debugger is connected.

```
sb.append("\n");
sb.append("Build: ").append(Build.FINGERPRINT).append("\n");
if (Debug.isDebuggerConnected()) {
    sb.append("Debugger: Connected\n");
}
if (i != 0) {
    sb.append("DD-EDD: ").append(i).append("\n");
}

static {
    SLOW_EXEC_WARNING_THRESHOLD = Debug.isDebuggerConnected() ? 60 : 16;
}
```

Similarly, the MobSF report notes that the app has anti-debug code, aligning with the previous findings (shown below). For these reasons, the MealLogger app passes this standard.

Anti Debug Code	Debug.isDebuggerConnected() check
-----------------	-----------------------------------

MSTG-RESILIENCE-3

The app detects, and responds to, tampering with executable files and critical data within its own sandbox.

Note: Beyond the scope of this class.

MSTG-RESILIENCE-4

The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.

Note: Beyond the scope of this class.

MSTG-RESILIENCE-5

The app detects, and responds to, being run in an emulator.

Looking at the screenshot below, we can see that the application clearly has some form of emulator detection. However, it is important to note that the application does not have any popups or differing functionality when run in an emulator, as opposed to a normal device. It is possible that discrete background processes are affected by the emulator, but such activity is not easily discernible. For these reasons, the MealLogger app fails this standard.

```

/* Loaded from: classes.dex */
48 public class AppEventUtility {
    private static final String regex = "[~+]*\\d+([\\.,\\.\\.]\\d+)*([\\.,\\.\\.]\\d+)?";

49     public static void assertIsNotMainThread() {
    }

57     public static void assertIsMainThread() {
    }

61     public static double normalizePrice(String value) {
        try {
62         Pattern pattern = Pattern.compile(regex, 8);
63         Matcher matcher = pattern.matcher(value);
64         if (matcher.find()) {
65             String firstValue = matcher.group(0);
66             return NumberFormat.getNumberInstance(Utility.getCurrentLocale()).parse(fi
67         }
        return 0.0d;
        } catch (ParseException e) {
            return 0.0d;
        }
    }

77     public static String bytesToHex(byte[] bytes) {
78         StringBuffer sb = new StringBuffer();
79         for (byte b : bytes) {
80             sb.append(String.format("%02x", Byte.valueOf(b)));
81         }
82         return sb.toString();
    }

85     public static boolean isEmulator() {
        return Build.FINGERPRINT.startsWith(MessengerShareContentUtility.TEMPLATE_GENERIC_
    }

```

MSTG-RESILIENCE-6

The app detects, and responds to, tampering the code and data in its own memory space.

Note: Beyond the scope of this class.

MSTG-RESILIENCE-7

The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that resiliency scales with the amount, diversity of the originality of the mechanisms used.

Note: Beyond the scope of this class.

MSTG-RESILIENCE-8

The detection mechanisms trigger responses of different types, including delayed and stealthy responses.

Note: Beyond the scope of this class.

MSTG-RESILIENCE-9

Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.

The MealLogger Android application had several instances of code obfuscation. By using jadx, I was able to identify these occurrences by comparing the original and de-obfuscated versions of the source code. Recorded on the next page are important/significant occurrences of such obfuscation:

```

BackgroundActionButtonHandler x BuildConfig x R x
708 public static final int action_container = 0x7f080007;
709 public static final int action_context_bar = 0x7f080010;
710 public static final int action_divider = 0x7f080011;
711 public static final int action_image = 0x7f080013;
712 public static final int action_menu_divider = 0x7f080014;
713 public static final int action_menu_presenter = 0x7f080015;
714 public static final int action_mode_bar = 0x7f080016;
715 public static final int action_mode_bar_stub = 0x7f080017;
716 public static final int action_mode_close_button = 0x7f080018;
717 public static final int action_text = 0x7f08001a;
718 public static final int actions = 0x7f08001b;
719 public static final int activity_chooser_view_content = 0x7f08001c;
720 public static final int add = 0x7f08001d;
721 public static final int alertTitle = 0x7f080020;
722 public static final int async = 0x7f080024;
723 public static final int automatic = 0x7f080029;
724 public static final int blocking = 0x7f08002e;
725 public static final int bottom = 0x7f08002f;
726 public static final int box_count = 0x7f080030;
727 public static final int button = 0x7f080032;
728 public static final int buttonPanel = 0x7f080033;
729 public static final int cancel_action = 0x7f080037;
730 public static final int cancel_button = 0x7f080038;
731 public static final int center = 0x7f08003c;
732 public static final int checkbox = 0x7f080048;
733 public static final int chronometer = 0x7f080049;
734 public static final int com_facebook_body_frame = 0x7f08004f;
735 public static final int com_facebook_button_xout = 0x7f080050;
736 public static final int com_facebook_device_auth_instructions = 0x7f080051;
737 public static final int com_facebook_fragment_container = 0x7f080052;
738 public static final int com_facebook_login_fragment_progress_bar = 0x7f080053;
739 public static final int com_facebook_smart_instructions_0 = 0x7f080054;
740 public static final int com_facebook_smart_instructions_or = 0x7f080055;
741 public static final int com_facebook_tooltip_bubble_view_bottom_pointer = 0x7f080056;
742 public static final int com_facebook_tooltip_bubble_view_text_body = 0x7f080057;
743 public static final int com_facebook_tooltip_bubble_view_top_pointer = 0x7f080058;
744 public static final int confirmation_code = 0x7f080063;

```

```

BackgroundActionButtonHandler x C0541R x
718 public static final int action_container = 2131230735;
719 public static final int action_context_bar = 2131230736;
720 public static final int action_divider = 2131230737;
721 public static final int action_image = 2131230739;
722 public static final int action_menu_divider = 2131230740;
723 public static final int action_menu_presenter = 2131230741;
724 public static final int action_mode_bar = 2131230742;
725 public static final int action_mode_bar_stub = 2131230743;
726 public static final int action_mode_close_button = 2131230744;
727 public static final int action_text = 2131230746;
728 public static final int actions = 2131230747;
729 public static final int activity_chooser_view_content = 2131230748;
730 public static final int add = 2131230749;
731 public static final int alertTitle = 2131230752;
732 public static final int async = 2131230756;
733 public static final int automatic = 2131230761;
734 public static final int blocking = 2131230766;
735 public static final int bottom = 2131230767;
736 public static final int box_count = 2131230768;
737 public static final int button = 2131230770;
738 public static final int buttonPanel = 2131230771;
739 public static final int cancel_action = 2131230775;
740 public static final int cancel_button = 2131230776;
741 public static final int center = 2131230780;
742 public static final int checkbox = 2131230792;
743 public static final int chronometer = 2131230793;
744 public static final int com_facebook_body_frame = 2131230799;
745 public static final int com_facebook_button_xout = 2131230800;
746 public static final int com_facebook_device_auth_instructions = 2131230801;
747 public static final int com_facebook_fragment_container = 2131230802;
748 public static final int com_facebook_login_fragment_progress_bar = 2131230803;
749 public static final int com_facebook_smart_instructions_0 = 2131230804;
750 public static final int com_facebook_smart_instructions_or = 2131230805;
751 public static final int com_facebook_tooltip_bubble_view_bottom_pointer = 2131230806;
752 public static final int com_facebook_tooltip_bubble_view_text_body = 2131230807;
753 public static final int com_facebook_tooltip_bubble_view_top_pointer = 2131230808;
754 public static final int confirmation_code = 2131230819;

```

We can see that in the first screenshot, all of the values for each class variable are written in hexadecimal, indicating some amount of encryption. In the second screenshot, after de-obfuscating in jadx, we can see that the previously encrypted values are now shown with their actual values. Comparing the two screenshots, we can see that there was clearly an attempt to obfuscate the ID numbers for each of the given processes. Furthermore, there are several instances of this found among different “R” files containing the resource IDs for various Android assets. However, due to the ease of de-obfuscation by jadx, the MealLogger app ultimately fails this standard.

MSTG-RESILIENCE-10

The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device.

Looking at the screenshot below, we can see that the application clearly has the ability to fingerprint the device. Closely analyzing the return statement (shown below), it is apparent that the application recognizes all of the significant flags (fingerprint, model, manufacturer, brand, etc.) related to device fingerprinting. Thus, the application passes this standard, implementing device fingerprinting derived from multiple unique properties.



```
< lex x ActivityLifecycleTracker x AppEventUtility x MarketingUtils x v
package com.facebook.marketing.internal;

import android.os.Build;
import android.util.Log;
import com.facebook.share.internal.MessengerShareContentUtility;
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

/* Loaded from: classes.dex */
37 public class MarketingUtils {
    private static final String TAG = MarketingUtils.class.getCanonicalName();

38     public static boolean isEmulator() {
        return Build.FINGERPRINT.startsWith(MessengerShareContentUtility.TEMPLATE_GENERIC_TV
    }

50     public static double normalizePrice(String value) {
        try {
51         String cleanValue = value.replaceAll("[^\\d,.-]", "");
53         return NumberFormat.getNumberInstance(Locale.getDefault()).parse(cleanValue).doubleValue();
        } catch (ParseException e) {
55         Log.e(TAG, "Error parsing price: ", e);
56         return 0.0d;
        }
    }
}
```

“return” statement:

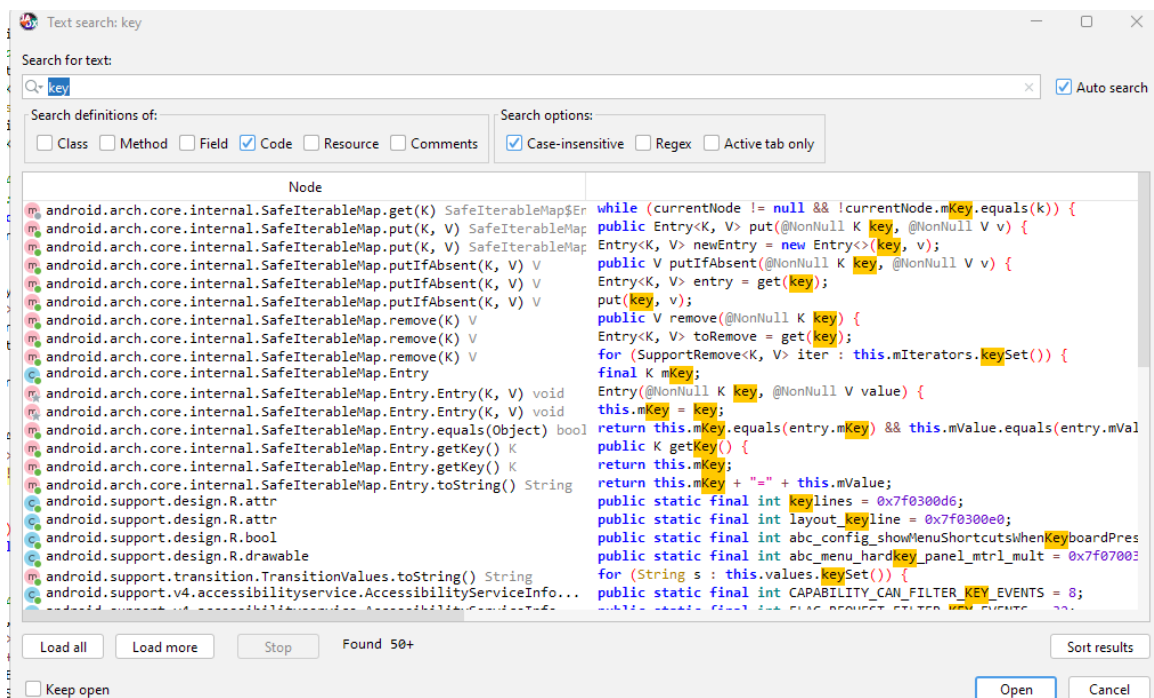
return

```
Build.FINGERPRINT.startsWith(MessengerShareContentUtility.TEMPLATE_GENERIC_TYPE)
|| Build.FINGERPRINT.startsWith("unknown") || Build.MODEL.contains("google_sdk") ||
Build.MODEL.contains("Emulator") || Build.MODEL.contains("Android SDK built for x86") ||
Build.MANUFACTURER.contains("Genymotion") ||
(Build.BRAND.startsWith(MessengerShareContentUtility.TEMPLATE_GENERIC_TYPE) &&
Build.DEVICE.startsWith(MessengerShareContentUtility.TEMPLATE_GENERIC_TYPE)) ||
"google_sdk".equals(Build.PRODUCT);
```

MSTG-RESILIENCE-11

All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.

By searching for a variety of keywords following deobfuscation in jadx, I was able to find several instances of strings of interest, especially relating to permissions and secrets. Below are screenshots of the results with their corresponding search keyword:



“key”: There were no significant results for this keyword, as most references related to inner function calls as opposed to the hardcoding of any specific key. However, there were significant results for searching “api key,” which is shown below.

```
/* JADX INFO: Access modifiers changed from: private */
public void setUpIntercom() {
    try {
        Context context = this.cordova.getActivity().getApplicationContext();
        CordovaHeaderInterceptor.setCordovaVersion(context, "3.2.2");
        switch (IntercomPushManager.getInstalledModuleType()) {
            case GCM:
                String senderId = this.preferences.getString("intercom-android-sender-id", null);
                if (senderId != null) {
                    IntercomPushManager.cacheSenderId(context, senderId);
                    break;
                }
                break;
        }
        ApplicationInfo app = context.getPackageManager().getApplicationInfo(context.getPackageName(), 128);
        Bundle bundle = app.metaData;
        String apiKey = this.preferences.getString("intercom-android-api-key", bundle.getString("intercom_api_key"));
        String appId = this.preferences.getString("intercom-app-id", bundle.getString("intercom_app_id"));
        Intercom.initialize(this.cordova.getActivity().getApplication(), apiKey, appId);
    } catch (Exception e) {
        Log.e("Intercom-Cordova", "ERROR: Something went wrong when initializing Intercom. Have you set your APP_ID and ANDROID_API_KEY?");
    }
}
```

“api key”: Here, we can see that there is a hardcoded reference to an intercom api key. Although the key itself is not present, it is generally bad practice to have the api key be easily accessible within a public class.

```
private boolean verifySecret(String action, int bridgeSecret) throws IllegalAccessException {
    if (!this.jsMessageQueue.isBridgeEnabled()) {
        if (bridgeSecret == -1) {
            LOG.m16d(LOG_TAG, action + " call made before bridge was enabled.");
        } else {
            LOG.m16d(LOG_TAG, "Ignoring " + action + " from previous page load.");
        }
        return false;
    } else if (this.expectedBridgeSecret < 0 || bridgeSecret != this.expectedBridgeSecret) {
        LOG.m13e(LOG_TAG, "Bridge access attempt with wrong secret token, possibly from malicious code. Disabling exec() bridge!");
        clearBridgeSecret();
        throw new IllegalAccessException();
    } else {
        return true;
    }
}

void clearBridgeSecret() {
    this.expectedBridgeSecret = -1;
}

public boolean isSecretEstablished() {
    return this.expectedBridgeSecret != -1;
}

int generateBridgeSecret() {
    SecureRandom randGen = new SecureRandom();
    this.expectedBridgeSecret = randGen.nextInt(Integer.MAX_VALUE);
    return this.expectedBridgeSecret;
}

public void reset() {
    this.jsMessageQueue.reset();
    clearBridgeSecret();
}
```

“token”: For this keyword, we can see that there is a function that accesses tokens for several different web modes, including Katana, webview, and even Facebook. Although this is not

necessarily an issue, it seems interesting that the login authentication tokens are easily accessible and locatable.

```
public static IBinder getBinder(Bundle bundle, String key) {
    if (!sGetIBinderMethodFetched) {
        try {
            sGetIBinderMethod = Bundle.class.getMethod("getIBinder", String.class);
            sGetIBinderMethod.setAccessible(true);
        } catch (NoSuchMethodException e) {
            Log.i(TAG, "Failed to retrieve getIBinder method", e);
        }
        sGetIBinderMethodFetched = true;
    }
    if (sGetIBinderMethod != null) {
        try {
            return (IBinder) sGetIBinderMethod.invoke(bundle, key);
        } catch (IllegalAccessException | IllegalArgumentException | InvocationTargetException e2) {
            Log.i(TAG, "Failed to invoke getIBinder via reflection", e2);
            sGetIBinderMethod = null;
        }
    }
    return null;
}
```

“log”: For this keyword, we can see that the source code does in fact log information in the system, using a public class. Although the effects of this are not easily apparent, logging sensitive information on the system’s memory is bad practice, as similarly noted by MobSF.

For these reasons, the MealLogger app ultimately fails this standard for failing to properly encrypt sensitive information within the source code.

MSTG-RESILIENCE-12

If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.

By compiling the results from RESILIENCE-9 and RESILIENCE-11, it becomes clear that the obfuscation techniques employed by the MealLogger app fall short. Jadx was easily able to de-obfuscate each and every instance. Although an obfuscation scheme was present, manual and automated de-obfuscation methods were highly effective against the application’s decompiled source code. For these reasons, the MealLogger app fails this requirement.

MSTG-RESILIENCE-13

As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.

Note: Beyond the scope of this class.