

Pizza Shop: Vulnerability Analysis With MobSF

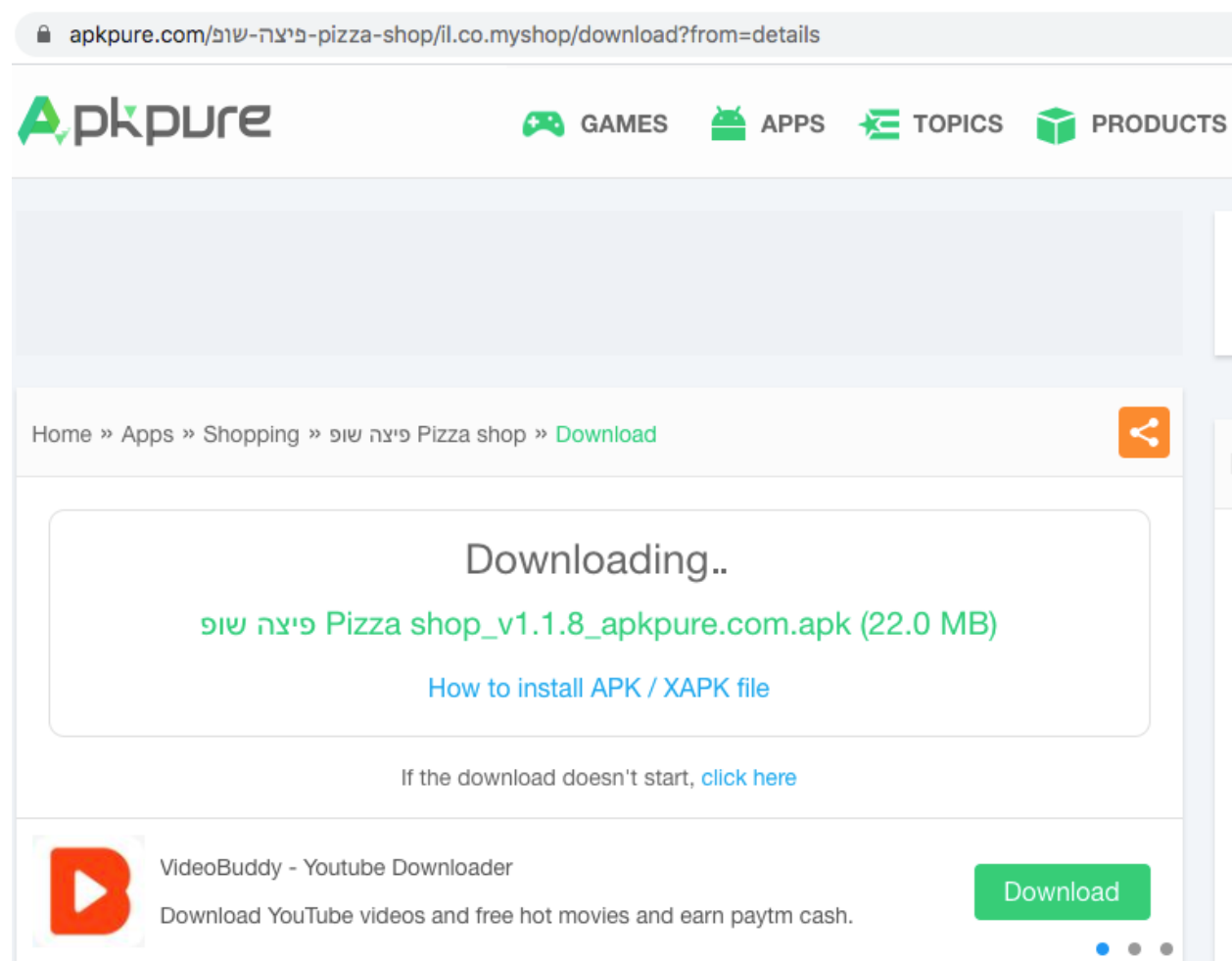
By Mike Zelixon

Objective

My assignment was to locate any vulnerabilities I could find in the Pizza Shop app. For this, I used the popular MobSF mobile application vulnerability scanner with the APK file of the app. After downloading the APK and uploading it to the software, my findings were significant. The app is full of vulnerabilities.

Step 1: Locate APK

The first thing I did was go online and locate the APK of the Pizza Shop app on the Google Play Store. I had to do a bit of searching but I finally found the right file, as shown on the following image.

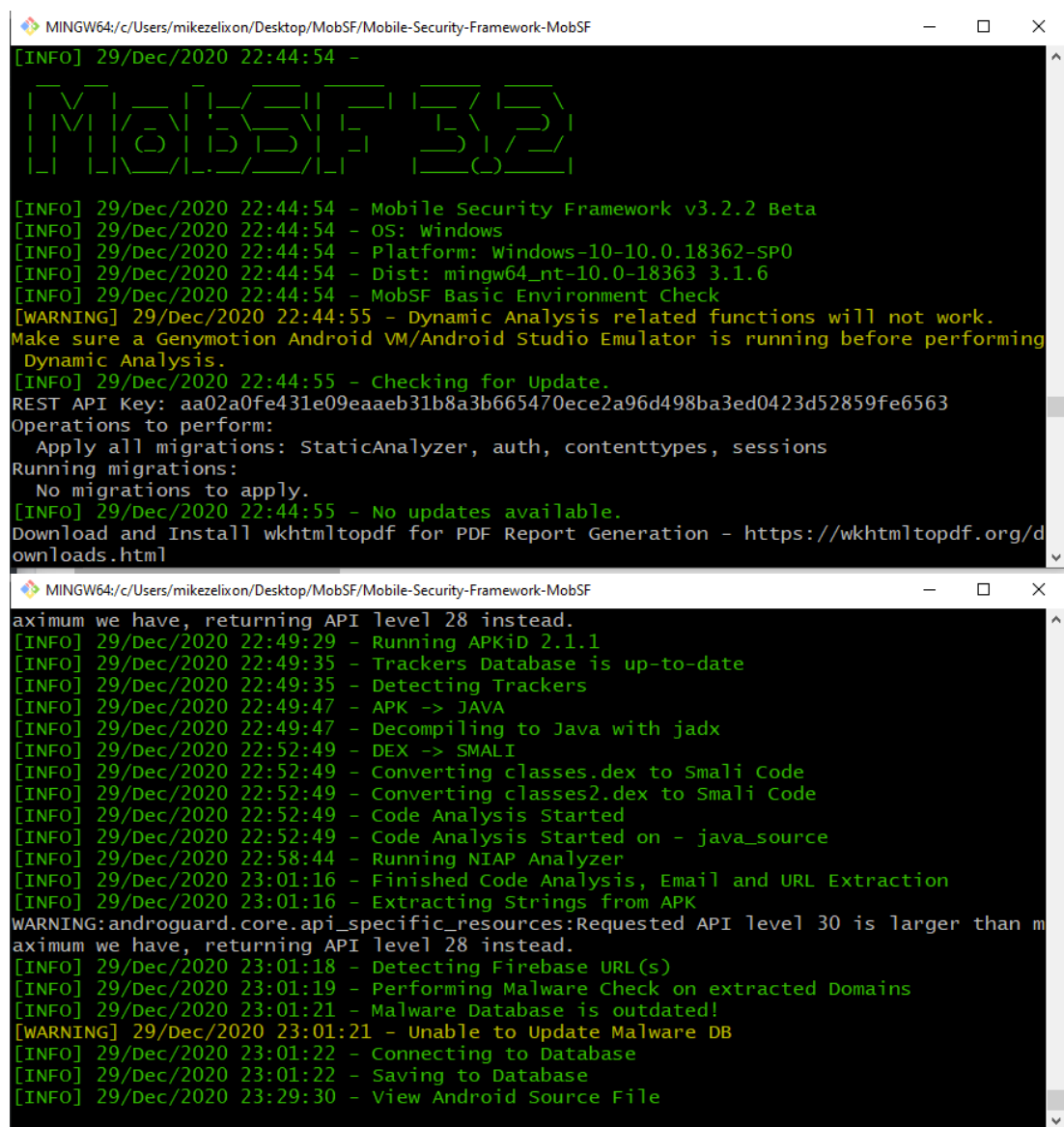


Step 2: Analyze APK using MobSF

To find any vulnerabilities and security weaknesses the app may have, I decided to use MobSF, a popular static and dynamic analysis tool for mobile applications. In this case, I stuck to only performing a static analysis, as I thought it would be sufficient for the assignment. My results were pretty eye-opening. MobSF is a phenomenal tool that analyzes and decompiles APK's and generates a full, easy-to-read report.

Decompilation

Before diving into the application itself I want to showcase the decompilation used by MobSF, which uses an array of tools such as Git in order to do so. These screenshots show the decompilation process.




```
MINGW64/c/Users/mikezelixon/Desktop/MobSF/Mobile-Security-Framework-MobSF
[INFO] 29/Dec/2020 22:44:54 -
[INFO] 29/Dec/2020 22:44:54 - Mobile Security Framework v3.2.2 Beta
[INFO] 29/Dec/2020 22:44:54 - OS: Windows
[INFO] 29/Dec/2020 22:44:54 - Platform: Windows-10-10.0.18362-SP0
[INFO] 29/Dec/2020 22:44:54 - Dist: mingw64_nt-10.0-18363 3.1.6
[INFO] 29/Dec/2020 22:44:54 - MobSF Basic Environment Check
[WARNING] 29/Dec/2020 22:44:55 - Dynamic Analysis related functions will not work.
Make sure a Genymotion Android VM/Android Studio Emulator is running before performing
Dynamic Analysis.
[INFO] 29/Dec/2020 22:44:55 - Checking for Update.
REST API Key: aa02a0fe431e09eaaeb31b8a3b665470ece2a96d498ba3ed0423d52859fe6563
Operations to perform:
  Apply all migrations: StaticAnalyzer, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.
[INFO] 29/Dec/2020 22:44:55 - No updates available.
Download and Install wkhtmltopdf for PDF Report Generation - https://wkhtmltopdf.org/downloads.html

MINGW64/c/Users/mikezelixon/Desktop/MobSF/Mobile-Security-Framework-MobSF
maximum we have, returning API level 28 instead.
[INFO] 29/Dec/2020 22:49:29 - Running APKID 2.1.1
[INFO] 29/Dec/2020 22:49:35 - Trackers Database is up-to-date
[INFO] 29/Dec/2020 22:49:35 - Detecting Trackers
[INFO] 29/Dec/2020 22:49:47 - APK -> JAVA
[INFO] 29/Dec/2020 22:49:47 - Decompiling to Java with jadx
[INFO] 29/Dec/2020 22:52:49 - DEX -> SMALI
[INFO] 29/Dec/2020 22:52:49 - Converting classes.dex to Smali Code
[INFO] 29/Dec/2020 22:52:49 - Converting classes2.dex to Smali Code
[INFO] 29/Dec/2020 22:52:49 - Code Analysis Started
[INFO] 29/Dec/2020 22:52:49 - Code Analysis Started on - java_source
[INFO] 29/Dec/2020 22:58:44 - Running NIAP Analyzer
[INFO] 29/Dec/2020 23:01:16 - Finished Code Analysis, Email and URL Extraction
[INFO] 29/Dec/2020 23:01:16 - Extracting Strings from APK
WARNING:androguard.core.api_specific_resources:Requested API level 30 is larger than m
maximum we have, returning API level 28 instead.
[INFO] 29/Dec/2020 23:01:18 - Detecting Firebase URL(s)
[INFO] 29/Dec/2020 23:01:19 - Performing Malware Check on extracted Domains
[INFO] 29/Dec/2020 23:01:21 - Malware Database is outdated!
[WARNING] 29/Dec/2020 23:01:21 - Unable to Update Malware DB
[INFO] 29/Dec/2020 23:01:22 - Connecting to Database
[INFO] 29/Dec/2020 23:01:22 - Saving to Database
[INFO] 29/Dec/2020 23:29:30 - View Android Source File
```

MobSF Information Section

The first thing you see after uploading the APK file is the information section. Here you can see all the basic info about the app itself. Things such as app size, developer information, API level, hashes used, etc. Additionally, on the top right of the information section, MobSF gives you two important metrics. The first metric is the average CVSS (Common Vulnerability Scoring System) of all the CVE's it finds. In this case the average CVSS was a 6.9 which is quite high. Secondly, MobSF gives you a security score out of 100 based on all of its findings. In this case, the Pizza Shop app scored very low with a security score of 15 out of 100. The following is a screenshot of the information section.

APP SCORES



Average CVSS **6.9**

Security Score **15/100**

Trackers Detection **3/335**

FILE INFORMATION

File Name **פיצה שופ Pizza shop_v1.1.8_apkpure.com.apk**

Size **22.04MB**

MDS **1ac9caf225a8cb590c4525581af5affe**

SHA1 **18bf8de7837ee65512a3b01d025a1ce72654b30b**

SHA256 **d9cd0af5fdd195da0eb0b392cd0ef4f148d6783ed8682fcc1324ec460387e177**

APP INFORMATION

App Name **Pizza shop**

Package Name **il.co.myshop**

Main Activity **il.co.myshop.MainActivity**

Target SDK **30** Min SDK **23** Max SDK

Android Version Name **1.1.8** Android Version Code **18**

PLAYSTORE INFORMATION

Title **פיצה שופ Pizza shop**

Score **0.0** Installs **100+** Price **0** Android Version Support **6.0 and up** Category **Shopping** Play Store URL **il.co.myshop**

Developer **BiGapps** Developer ID **BiGapps**

Developer Address **Histadrot Boulevard 60 Haifa**

Developer Website **http://yeshlipirsum.co.il**

Developer Email **qa.bigapps@gmail.com**

Release Date **Oct 14, 2020** Privacy Policy **Privacy link**

Description

Certificate & Source Code

After the information section MobSF lets us see the signed certificate of the app and it's security status. We can also browse the the entire source code itself including and manifest, java, and smali files, as seen in the following images. **Notably, there's a potential vulnerability with the certificate, as shown in the warning below it.**

STATUS	DESCRIPTION
secure	Application is signed with a code signing certificate
warning	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android <7.0

SIGNER CERTIFICATE

APK is signed
v1 signature: True
v2 signature: True
v3 signature: True
Found 1 unique certificates
Subject: C=US, ST=California, L=Mountain View, O=Google Inc., OU=Android, CN=Android
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2020-09-09 13:24:34+00:00
Valid To: 2050-09-09 13:24:34+00:00
Issuer: C=US, ST=California, L=Mountain View, O=Google Inc., OU=Android, CN=Android
Serial Number: 0xfe056156cf121121c267bf4f2cf06af7c2ee0316
Hash Algorithm: sha256
md5: 64aed7f3ca5eb7115ac7f632c8058503
sha1: a33cb2ad2b04723672d608eb327d83478f0fc2f4
sha256: 14e677b845b06fdaebc871d30849dc275ac329f3367102d7051a8485b79139ff
sha512: 7ef0a0484c1eccc3c993cf29144e582fcb512f2aa9471998fb9dac67a36704212661baff7510225e884548b6089d35cc4e9d54d795d439df4
PublicKey Algorithm: rsa
Bit Size: 4096
Fingerprint: 12a4887d9dd9301031b01504e1ae71f2036a7ffa1865641dd4a4a31f102c4f8b

Below are images of the APK source code as seen in MobSF.

AndroidManifest.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest android:versionCode="18" android:versionName="1.1.8" android:compileSdkVersion="30" android:compileSdkVersionCodename="11" package="il.co.myshop" pla
3.    xmlns:android="http://schemas.android.com/apk/res/android">
4.    <uses-sdk android:minSdkVersion="23" android:targetSdkVersion="30" />
5.    <uses-permission android:name="android.permission.INTERNET" />
6.    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
7.    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
8.    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
9.    <permission android:name="il.co.myshop.permission.C2D_MESSAGE" android:protectionLevel="signature" />
10.   <uses-permission android:name="il.co.myshop.permission.C2D_MESSAGE" />
11.   <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
12.   <uses-permission android:name="android.permission.WAKE_LOCK" />
13.   <uses-permission android:name="android.permission.VIBRATE" />
14.   <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
15.   <uses-permission android:name="com.sec.android.provider.badge.permission.READ" />
16.   <uses-permission android:name="com.sec.android.provider.badge.permission.WRITE" />
17.   <uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS" />
18.   <uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT" />
19.   <uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE" />
20.   <uses-permission android:name="com.sonymobile.home.permission.PROVIDER_INSERT_BADGE" />
21.   <uses-permission android:name="com.anddoes.launcher.permission.UPDATE_COUNT" />
22.   <uses-permission android:name="com.majeur.launcher.permission.UPDATE_BADGE" />
23.   <uses-permission android:name="com.huawei.android.launcher.permission.CHANGE_BADGE" />
24.   <uses-permission android:name="com.huawei.android.launcher.permission.READ_SETTINGS" />
25.   <uses-permission android:name="com.huawei.android.launcher.permission.WRITE_SETTINGS" />
26.   <uses-permission android:name="android.permission.READ_APP_BADGE" />
27.   <uses-permission android:name="com.oppo.launcher.permission.READ_SETTINGS" />
28.   <uses-permission android:name="com.oppo.launcher.permission.WRITE_SETTINGS" />
29.   <uses-permission android:name="me.everything.badger.permission.BADGE_COUNT_READ" />
30.   <uses-permission android:name="me.everything.badger.permission.BADGE_COUNT_WRITE" />
```

Activate Windows
Go to Settings to activate Windows.

</> Smali Source

Find by filename: Find by content:



File: \$r8\$backportedMethods\$utility\$Double\$1\$hashCode.smali

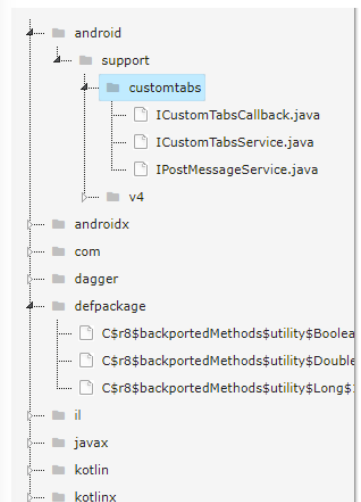
```
1. .class public synthetic L$r8$backportedMethods$utility$Double$1$hashCode;  
2. .super Ljava/lang/Object;  
3.  
4.  
5. # direct methods  
6. .method public static synthetic hashCode(D)I  
7.   .registers 4  
8.  
9.   invoke-static {p0, p1}, Ljava/lang/Double;->doubleToLongBits(D)J  
10.  
11.   move-result-wide p0  
12.  
13.   const/16 v0, 0x20  
14.  
15.   ushr-long v0, p0, v0  
16.  
17.   xor-long/2addr p0, v0  
18.  
19.   long-to-int p0, p0  
20.  
21.   return p0  
22. .end method
```

Activate Windows

Go to Settings to activate Windows.

Java Source

Find by filename: Find by content:



File: C\$r8\$backportedMethods\$utility\$Boolean\$1\$hashCode.java

```
1. package defpackage;  
2.  
3. /* renamed from: $r8$backportedMethods$utility$Boolean$1$hashCode reason: invalid class name and default package */  
4. public /* synthetic */ class C$r8$backportedMethods$utility$Boolean$1$hashCode {  
5.     public static /* synthetic */ int hashCode(boolean z) {  
6.         return z ? 1231 : 1237;  
7.     }  
8. }
```

Activate Windows

Go to Settings to activate Windows.

Permissions Vulnerabilities

After the initial sections, we start getting into the meat of the software: the actual vulnerability findings. The first vulnerabilities MobSF looks at are with the application permissions themselves. **In this case, it has found 2 notable vulnerabilities where it lists the exact reference points and gives a full explanation. MobSF found 2 flaws in the coarse and GPS location permissions as seen in the image below. These permissions can be abused by hackers to gain data.**

☰ APPLICATION PERMISSIONS

Search:

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

Manifest Analysis

The next section MobSF looks at is the manifest file. Once again it gives a full description of every issue/bug it finds and puts it in an easily understandable format. **In this case MobSF found 6 critical flaws with a high severity level. The flaws can be seen in the images below on the following page. You can see several issues such as enabling of clear text and problems with several receivers. The issue severities are all highlighted and concise explanations are given to every issue.**

Search:

NO ↑↓	ISSUE ↑↓	SEVERITY ↑↓	DESCRIPTION ↑↓
1	Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]	high	The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.
2	Broadcast Receiver (com.onesignal.GcmBroadcastReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: com.google.android.c2dm.permission.SEND [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
3	Activity (com.onesignal.NotificationOpenedActivityHMS) is not Protected. An intent-filter exists.	high	An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Activity is explicitly exported.
4	Broadcast Receiver (com.onesignal.BootUpReceiver) is not Protected. An intent-filter exists.	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
5	Broadcast Receiver (com.onesignal.UpgradeReceiver) is not Protected. An intent-filter exists.	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
6	Broadcast Receiver (com.google.firebase.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permission: com.google.android.c2dm.permission.SEND [android:exported=true]	high	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.

Code & CVE Analysis

After the manifest, we have the code analysis section. Here MobSF has found several CVE's and CWE's at different levels, and has listed them by severity. **With the Pizza Shop app MobSF found 7 CVE's ranging from medium to high levels and has highlighted the exact code locations of where the vulnerabilities are located. You can see this in the images below.**

</> CODE ANALYSIS

Search:

NO ↕	ISSUE ↕	SEVERITY ↕	STANDARDS ↕	FILES
1	The App uses an insecure Random Number Generator.	high	CVSS V2: 7.5 (high) CWE: CWE-330 Use of Insufficiently Random Values OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6	kotlin/collections/unsigned/UArraysKt___UArraysJvmKt\$asList\$3.java kotlin/random/AbstractPlatformRandom.java kotlin/collections/ArraysKt___ArraysJvmKt\$asList\$4.java kotlin/random/PlatformRandom.java kotlin/collections/ArraysKt___ArraysJvmKt\$asList\$7.java kotlin/collections/ArraysKt___ArraysJvmKt\$asList\$2.java kotlin/collections/CollectionsKt___CollectionsJVMKt.java kotlin/collections/AbstractList.java kotlin/collections/builders/ListBuilder.java kotlin/collections/CollectionsKt___MutableCollectionsJVMKt.java kotlin/random/FallbackThreadLocalRandom\$SimpleStorage\$1.java kotlin/collections/CollectionsKt___CollectionsKt.java kotlin/collections/MovingSubList.java kotlin/collections/unsigned/UArraysKt___UArraysJvmKt\$asList\$2.java kotlin/collections/CollectionsKt___MutableCollectionsKt.java
2	The App logs information. Sensitive information should never be logged.	info	CVSS V2: 7.5 (high) CWE: CWE-532 Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	com/bumptech/glide/util/ContentLengthInputStream.java com/bumptech/glide/load/resource/bitmap/DefaultImageHeaderParser.java kotlinx.coroutines/debug/AgentPremain\$installSignalHandler\$1.java com/bumptech/glide/load/resource/bitmap/TransformationUtils.java com/bumptech/glide/load/engine/DecodeJob.java com/bumptech/glide/manager/DefaultConnectivityMonitorFactory.java com/bumptech/glide/load/engine/SourceGenerator.java com/wdullaer/materialdatetimepicker/time/AmPmCircleView.java com/bumptech/glide/load/resource/ImageDecoderResourceDecoder.java com/bumptech/glide/load/resource/bitmap/BitmapEncoder.java

3	Files may contain hardcoded sensitive informations like usernames, passwords, keys etc.	high	CVSS V2: 7.4 (high) CWE: CWE-312 Cleartext Storage of Sensitive Information OWASP Top 10: M9: Reverse Engineering OWASP MASVS: MSTG-STORAGE-14	il/co/myshop/order/MapFragment.java il/co/myshop/data/ShopBranch.java com/onesignal/OneSignalRemoteParams.java com/onesignal/OSInAppMessageLocationPrompt.java com/onesignal/OSInAppMessageController.java com/onesignal/OneSignalNotificationManager.java il/co/myshop/data/Shop.java com/bumptechnology/glide/load/Option.java com/bumptechnology/glide/load/engine/EngineResource.java com/bumptechnology/glide/manager/RequestManagerRetriever.java com/onesignal/OSInAppMessagePrompt.java com/bumptechnology/glide/load/engine/ResourceCacheKey.java com/onesignal/WebViewManager.java com/bumptechnology/glide/load/engine/DataCacheKey.java com/onesignal/GcmBroadcastReceiver.java com/onesignal/NotificationBundleProcessor.java
4	App creates temp file. Sensitive information should never be written into a temp file.	high	CVSS V2: 5.5 (medium) CWE: CWE-276 Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	kotlin/io/FilesKt__UtilsKt.java com/dm6801/framework/remote/Http\$internalDownload\$1.java il/co/myshop/remote/Http\$internalDownload\$1.java
5	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	high	CVSS V2: 5.9 (medium) CWE: CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality	com/onesignal/outcomes/OSOutcomeTableProvider.java com/onesignal/OneSignalDbHelper.java

6	Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	warning	CVSS V2: 8.8 (high) CWE: CWE-749 Exposed Dangerous Method or Function OWASP Top 10: M1: Improper Platform Usage OWASP MASVS: MSTG-PLATFORM-7	com/onesignal/WebViewManager.java
7	Remote WebView debugging is enabled.	high	CVSS V2: 5.4 (medium) CWE: CWE-919 - Weaknesses in Mobile Applications OWASP Top 10: M1: Improper Platform Usage OWASP MASVS: MSTG-RESILIENCE-2	com/onesignal/WebViewManager.java

Malware Analysis

The last section we'll look at is Malware Analysis, which is split into two parts: **APKiD Analysis and Domain Malware Check**.

APKiD Analysis

APKiD Analysis looks into the dex files of an APK for behavioral patterns such as which compiler was used and anti-VM detection, as seen below.

APKiD ANALYSIS

Search:

DEX	DETECTIONS								
classes.dex	<div>Search:</div> <div></div>								
	<table> <tr> <th>FINDINGS</th> <th>DETAILS</th> </tr> <tr> <td>Anti Debug Code</td> <td>Debug.isDebuggerConnected() check</td> </tr> <tr> <td>Anti-VM Code</td> <td> Build.FINGERPRINT check Build.MODEL check Build.MANUFACTURER check Build.PRODUCT check Build.TAGS check possible VM check </td> </tr> <tr> <td>Compiler</td> <td>r8</td> </tr> </table>	FINDINGS	DETAILS	Anti Debug Code	Debug.isDebuggerConnected() check	Anti-VM Code	Build.FINGERPRINT check Build.MODEL check Build.MANUFACTURER check Build.PRODUCT check Build.TAGS check possible VM check	Compiler	r8
	FINDINGS	DETAILS							
	Anti Debug Code	Debug.isDebuggerConnected() check							
Anti-VM Code	Build.FINGERPRINT check Build.MODEL check Build.MANUFACTURER check Build.PRODUCT check Build.TAGS check possible VM check								
Compiler	r8								
classes2.dex	<div>Search:</div> <div></div>								
	<table> <tr> <th>FINDINGS</th> <th>DETAILS</th> </tr> <tr> <td>Compiler</td> <td>r8 without marker (suspicious)</td> </tr> </table>	FINDINGS	DETAILS	Compiler	r8 without marker (suspicious)				
FINDINGS	DETAILS								
Compiler	r8 without marker (suspicious)								

Showing 1 to 1 of 1 entries

Previous

1

Next

Domain Malware Check

Domain Malware Check extracts all the domains from the binary and checks it against a list of rogue domains and IPs. It includes IP's and geolocation of all associated domains. In the case of Pizza Shop, MobSF did not detect any dangerous domains. **You can see the results in the images below.**

🔍 DOMAIN MALWARE CHECK

Search:

DOMAIN	STATUS	GEOLOCATION
api.onesignal.com	good	IP: 104.18.226.52 Country: United States of America Region: California City: San Francisco Latitude: 37.7757 Longitude: -122.395203 View: Google Map
exampleshop-8fdd2.firebaseio.com	good	IP: 35.201.97.85 Country: United States of America Region: Missouri City: Kansas City Latitude: 39.099731 Longitude: -94.578568 View: Google Map
maps.googleapis.com	good	IP: 216.58.212.234 Country: United States of America Region: California City: Mountain View Latitude: 37.405991 Longitude: -122.078514 View: Google Map
myshop.bigapps.co.il	good	IP: 69.16.233.144 Country: United States of America Region: Michigan City: Lansing Latitude: 42.73328 Longitude: -84.637764 View: Google Map
onesignal.com	good	IP: 104.18.226.52 Country: United States of America Region: California City: San Francisco Latitude: 37.7757 Longitude: -122.395203 View: Google Map

Activate Windows
Go to Settings to activate Windows

pci.zcredit.co.il	good	IP: 52.17.114.153 Country: Ireland Region: Dublin City: Dublin Latitude: 53.34399 Longitude: -6.26719 View: Google Map
secure5.tranzila.com	good	IP: 80.244.170.2 Country: Israel Region: HaMerkaz City: Netanya Latitude: 32.333611 Longitude: 34.85778 View: Google Map
www.bigapps.co.il	good	IP: 195.28.181.250 Country: Israel Region: HaMerkaz City: Kafr Qasim Latitude: 32.11417 Longitude: 34.973888 View: Google Map

Activate Windows

Conclusion

Although MobSF has other sections that I left out, I feel like I have covered the most important ones. Using MobSF I was able to very quickly scan and analyze the entirety of the Pizza Shop application.

Based on the results of my static analysis I can conclude that the application has several security flaws that should be addressed. The fact that the app got a security score of 15/100 is quite poor, not to mention the number of CVE's that the software detected (hence the low score).

While I did not perform a dynamic analysis, I can only imagine that a determined and skilled hacker could use the application and steal sensitive data if he wanted to with relative ease. A study from one article I found states that 71% of fraudulent transactions came from mobile apps and mobile browsers in the second quarter of 2018, which means ensuring mobile application security in today's digital world is vital.

In the case of Pizza Store, for example, any attacker who copies the source code can build a clone which deceives users into downloading malware, among other things.

