# CS 858 Final Report : Comparing the security impact of Online generated apps and apps developed using IDEs

Joy Idialu
j2idialu@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

Amirreza Shamsolhodaei
amirreza.shamsolhodaei@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

## Abstract

**In today's market, mobile applications are getting created increasingly using online application generators (OAGs). These apps automate the process of app creation using wizard-like, drag-and-drop interfaces that do not necessarily require any prior knowledge of app development and let the creative power focus only on conceptual matters such as UI elements and features implemented. However, as the use cases of these tools increase in magnitude, so does the security concerns that this model produces and the impact that this method has on the mobile ecosystem. In this work, we recreate the work of Oltrogge et al. on a much smaller scale and try to reproduce some aspects of their research. We have developed three apps using android studio and generated three apps using 2 of the popular online app generators. Conducting manual analysis on the code base of these apps and their manifest, we find that generated apps use more permissions than they need to and the usage of boilerplate code is prominent as it would be in any app generated by app generators and this could lead to reconfiguration attacks and well-known security issues such as code injection vulnerabilities.**

## 1 INTRODUCTION

Online app generators have been on the rise as with automation of the development, distribution, and maintenance of mobile apps, the technical skill required for creating an app becomes significantly lower, and this creates so-called citizen developers. as the pervasiveness of online app generators tools increases, so does their overall influence on the mobile ecosystem's security. This would lead to the fact that a certain amount of trust should be put in the generated code made by the app generator. But this raises the question of them being trustworthy enough, so this would be justified. App creators must fully trust that the generated code follows security best practices and does not violate the end-user's privacy. This dependence on app generators will lead to concerns over permission management of these apps and if they use more permissions than required.

Another problem would be if the generators use boilerplate code, i.e., sections of code repeated in multiple places with little to no variation, in different generated apps. This could lead to apps prone to reconfiguration attacks and vulnerabilities like code injection and insecure web views. Due to the black-box nature of these apps, citizen developers are unaware of these potential security issues. The boilerplate code could enable malicious users to gain access to users' private information, thereby invading users' privacy, hence the importance of subjecting these online app generators to security research.

This has led Oltrogge et al. [11]. to analyze the security vulnerabilities that online-generated apps could introduce. In their work, they have done the first classification of commonly used online services for creating Android apps based on various characteristics and quantified their market penetration. During this, they created a corpus of 2,291,898 free apps in the google play store and discovered that 11.1 percent are apps created by Online app generators. They used a combination of dynamic, static, and manual analysis; they analyzed 13 online app generators and found many vulnerabilities.

In this project, we recreate the work of [11] on a smaller scale using three generated apps and three developed apps that we created with different levels of permission and features, and we will compare the code structure, components, permissions, imported libraries, and parts and capabilities in OAG and IDE apps.

The rest of the report is organized as follows; in section 2, we will look at some background information on online app generators, permission system on Android, app development on android, and an overview of the Android manifest. . In section 3, we will look at our motivation for this project. Section 4 will delve into our dataset creation: developing and generating different android apps. Section 5 will explain our methodology and how we compared various applications. Section 6 will show our findings and the evaluation that was gathered. Section 7 will explain our takeaways from this project. Section 8 will introduce the challenges we faced, and finally, in section 9, we will focus on the future work that could be done to improve this project.

## 2 BACKGROUND

### 2.1 Permissions in Android

App permissions help support privacy by protecting access to restricted data such as system state and users' contact information and restricted actions such as connecting to

a paired device and recording audio[13]. The permissions should be implemented only and only when the functionality of a feature of an app would be compromised without the needed permission. For example, a photo editor needs to have access to the files and media to work correctly, and without the runtime permission for files and media, this app would not be able to function. There are three types of permissions:

1. Install-time permissions: these permissions grant limited access to restricted data or perform restricted actions that minimally affect the system of other apps. There is a notice on the app's page in the Google play store. The system automatically grants your app permission when the user installs your app. This type of permission includes normal permission and signature permissions. The former is the permissions that allow access to data and actions that extend beyond the app's sandbox. Still, it presents minimal risk to user privacy and the operation of other apps. The latter presents permissions where an app defines permission, and the system grants signature permission to an app only when the app is signed by the same certificate as the app that defines the permission.
2. Runtime (dangerous) permissions: These permissions give the app additional access to restricted data or let your app perform restricted actions that subsequently affect the system and other apps, such as microphone and camera access.
3. Special permissions: These permissions could only be defined by platform and OEMs and correspond to particular app operations. These permissions are determined when the OEM or the platform wants to protect access to potent actions, such as drawing over other apps.

We have analyzed the first two types of permissions in our work, as special permissions were not in the scope of our work or [11] work

### 2.2 Online App Generators

Online app generators or OAGs are online tools that create applications for the user without any programming or code implementation. They enable developers to abstract away from implementation aspects and instead focus only on the conceptual behavior of the application in terms of high-level functionality. Moreover, they provide functionality beyond core app generation, including support for app compilation, app dissemination, and distribution of patched versions. They also offer support for making an app available for different architectures of iOS and Android. They may even provide support for recurring, extended app functionality such as user management, user login, and data submission to back-end servers.

Online application generators (OAGs) enable app development using wizard-like, point-and-click, and drag-and-drop interfaces in which users only need to add and suitably interconnect UI elements that represent application components and therefore have the feature and the application that they want. These services also provide the option to distribute the app on the Google play store or apple's app store.

In our work, we used two of the most popular OAGs, Andromo [4] and MIT app inventor [10], which are dubbed two of the most secure app generators which do not use as many excessive permissions as other generators do [11].

Andromo provides users and businesses to create native and web Android applications without any code implementation, and users have the ability to preview the application as it's being built via Andromo's user interface. Andromo has different versions with different features and a trial period for the free version. More features could be unlocked using subscriptions.

MIT App Inventor is a web application integrated development environment initially provided by Google and now maintained by the Massachusetts Institute of Technology (MIT). It allows users to generate applications for both iOS and Android. It is a free and open-sourced software, and it uses an interface that allows users to drag and drop visual objects to create an application.

### 2.3 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development [3]. It was announced on May 16, 2013, at the Google I/O conference and was released a year later. Kotlin and Java are the main languages that can be used to develop apps in Android studio, and it has features like an APK analyzer.

### 2.4 Android Manifest

Every app project has the AndroidManifes.xml file at the root of the project source. the manifest describes essential information about the app to the android build tools, android OS, and Google play store [1]. The manifest file includes:

1. Components: Components of an app include all activities, broadcast services, content providers, and services. Each component must define basic properties like the name of the class, and it also can declare the capabilities.
2. Permissions: These include all the permission that the app needs in order to access restricted data and restricted actions. It also declares any permissions that other apps must have if they want to access content from this app.[12]
3. Hardware and software features: These include the hardware and software features the app requires, which affects which devices can install the app from Google Play.

## 3 MOTIVATION

[11] were the first to classify commonly used online app generators (OAGs); they also identified security issues present in them. Prior to their work, the security impact of online mobile app generators had not been investigated. This formed the motivation behind our project, where we worked towards arriving at or disproving some of their findings but on a smaller scale.

Our motivation case is a simple greeting app that displays a "hello" message with an emoji. We developed two versions of the app, one using Android studio IDE as shown in figure 1 and the other using Andromo OAG as shown in figure 2. We compared two features - permissions and components - used in both apps. The greeting app ordinarily requires no permission because it only displays a message on the screen and this was the case for the IDE app, however, the Andromo app had 5 permissions shown in listing 1. Although the permissions were normal permissions, we found them to be unnecessary. We hypothesized, based on [11] that it could be due to Andromo's usage of boilerplate code where other apps using the same features could actually require some of the permissions. When we investigated the AndroidManifest.xml file of both greeting apps, we found that the Andromo app had several components which cut across all four component types: Activity, Service, Broadcast Receiver, and Content Provider, which we also found highly unnecessary and potentially threatening to security and privacy whereas the IDE app had only one component, the main activity. This app formed our motivating example for why we needed to investigate more apps. We developed more apps using a different online generator which we discuss in the Dataset section, and we discuss our findings in the Evaluation section.

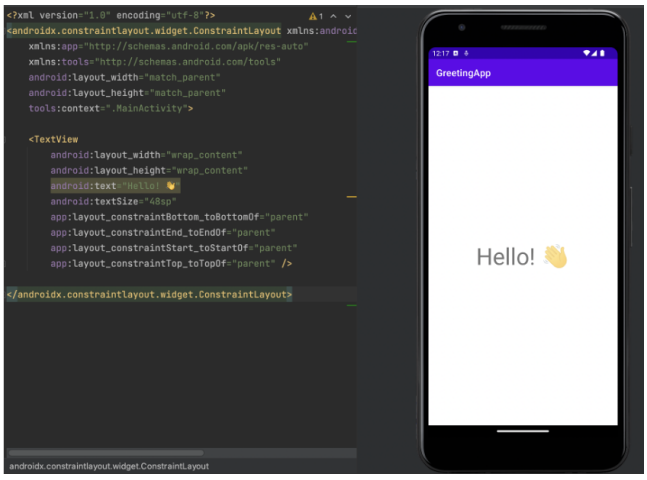


**Figure 1.** IDE interface showing the greeting app running on an emulator.

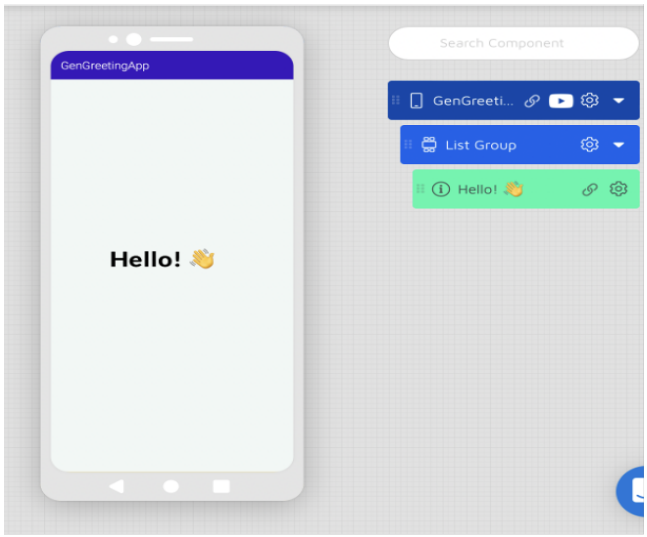


**Figure 2.** Andromo interface showing the generated greeting app.

**Threat Model.** Android apps are allowed to engage in inter-component communication. There are apps that export their components to get accessed by other apps. If the components exported by the generators have sensitive methods or data and are not sufficiently protected or are generated without following best practices in terms of permissions enforcement, an attacker could leverage these components in causing harm to the privacy and security of end-users. In our project, we set out to identify violations of standard best practices by these online app generators that could lead to potential privacy and security threats using the IDE apps we built as the basis of our comparison.

```
<uses-permission android:name="android.
    permission.INTERNET"/>
<uses-permission android:name="android.
    permission.WAKE_LOCK"/>
<uses-permission android:name="android.
    permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.
    permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.
    permission.FOREGROUND_SERVICE"/>
```

**Listing 1.** Normal permissions found in Andromo greeting app

## 4 DATASET

Our dataset consists of a total of six apps: three IDE apps and their online-generated versions. We developed a Greeting app, an SMS app, and a Location app using Android Studio IDE. We used Andromo to generate the Greeting app, and we used App Inventor to generate the SMS and Location

apps. These six apps are shown in figures 1 to 6. Both IDE and OAG Greeting apps display a “hello” message with an emoji once either app is launched by a user. The IDE and OAG SMS apps both require a user to fill in the recipient’s phone number and a message, after which the app sends the message to the recipient’s phone number provided by the user. The apps also display the last message sent to the user in the message field and the sender’s phone number in the number field. Both IDE and OAG location apps display certain information about the user’s location and allow the user to view their location on Google Maps. The IDE location app displays a user’s latitude, longitude, and accuracy of the location, and location of the user on a map in the app; it also allows the user to share their location. The OAG location app displays the user’s latitude, longitude, altitude, accuracy, and speed of the location and address.

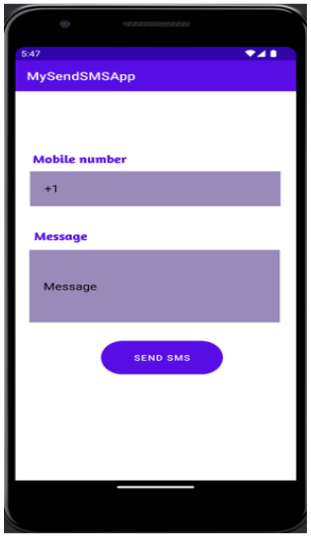


**Figure 3.** IDE SMS app.

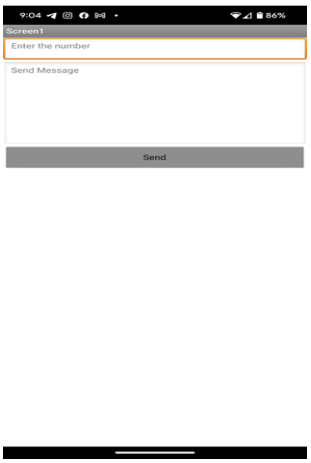


**Figure 4.** App Inventor SMS app.

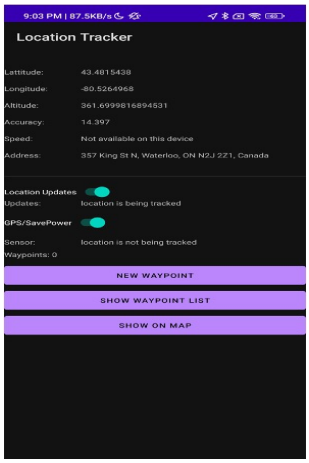


**Figure 5.** IDE Location app.

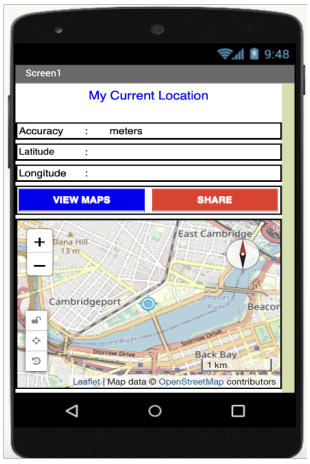


**Figure 6.** App Inventor Location app.

# 5 METHODOLOGY

## 5.1 OAG Apps Decompilation

To analyze both the IDE apps and OAG apps, we require access to the source code. We had access to the source code of the IDE apps since we developed them ourselves; however, for the OAG apps, we had to decompile the APK files of the apps manually. The decompilation process involved:

1. Using Apktool [5] to decompile the APK file and extract compressed files such as assets, resources, compiled code, and the AndroidManifest.XML file, which contained the relevant information on which most of our analyses were based. Apktool is a tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to their nearly original form and rebuild them after making some modifications. It also makes working with an app more accessible because of the project, like file structure and automation of some repetitive tasks like building apk, etc.

2. Using dex2jar [6] to convert APK files into a standard class file, which is compressed in a jar file. Android programs are compiled into dex (Dalvik Executable) files, which are, in turn, zipped into a single apk file on the device. The dex files can be created automatically by Android by translating the compiled applications written in the Java programming language. [2] The core feature of Dex2Jar is to convert the classes.dex file of an APK to classes.jar or vice versa. So, it is possible to view the source code of an Android application using any Java decompiler, and it is completely readable. Here, we get .class files and not the actual Java source code written by the application developer.
3. 1. Using JD-Gui [9], JADX [8], or Procyon [14] to convert the APK file into a standard class file that has Java source code inside it. These apps open the Jar file generated by the dex2jar tool and convert it to java source code. All these are graphical utilities that can browse reconstructed source code for instant access to methods and fields. If the Jar file is opened with any of these apps, you can view the source code of the application, which is Java classes in a readable format, and it is also straightforward to navigate through the code.

   The reason behind using different tools for the same purpose was the fact that JD-Gui, which was selected as our primary tool, ran into multiple problems while converting some of our apps, namely the Andromo generated app. There were some obfuscations that it could not deal with as we wanted. There were issues with the other tools as well and none of them could deliver us 100 percent of the source code.

### 5.2 Manual Analysis

After extracting the relevant files, such as the AndroidManifest.XML file and the java source code, we reviewed these files checking for any violations such as permission overprivilege, use of unnecessary or tracking third parties, the use of exported components that were not adequately protected using permissions, etc.

For detecting vulnerabilities in apps generated with OAG, we manually analyzed the apps that were generated and the apps that were developed. First, we compared each angle of the generated and developed apps with each other, and then we compared the generated apps among themselves.

**5.2.1 Generated and developed apps.** First, we looked at app features and capabilities; in this regard, our twin apps were identical as we could implement the exact features we wanted. Finding an app generator that could produce an SMS feature was challenging, and we will discuss it in section 7. However, in the end, we succeeded, and the features have the same functionality.

The file directory and package names were one other interesting ground that we were focused on; the directory of the files and the lines of codes that the apps were implemented was also quite interesting to us. Then we looked at the imported libraries; it was interesting that Andromo did not use android libraries and instead used GNU and Kawa libraries. After this, we focused our attention on the components and permissions that we could see on the Android manifest, which we are going to discuss in section 6. note that we did not look at the implementation of these components in our work, and this is one of the limitations of our project.

**5.2.2 Generated apps with each other.** One other interesting aspect was comparing the two app generators that we used; we noticed that Andromo uses much more permissions and components compared to App Inventor. in fact, Andromo used all the components possible, and they were visible in Android Manifest; however, due to not looking at their implementations, we can not verify this finding.

## 6 EVALUATION

The focus of this project was to reproduce Oltragge's work on analyzing security flaws present in online-generated apps on a smaller scale. We mainly focused on investigating the files manually and we made some findings that we believe, if investigated further, could lead to the discovery of potential vulnerabilities.

### 6.1 Boilerplate Code

[11] hypothesized that app generators use boilerplate code and proved that by comparing the hash value of the classes.dex file. We also compared the classes.dex file, but using Hex Fiend tool [7]. Apps from the same generator had a lot of similarities indicated by little to no red highlights in the Hex Fiend tool as shown in figure 7 and this was confirmed by comparing apps from different app generators - Andromo and Appinventor - where differences are indicated in the Hex Fiend tool by red highlights as shown in figure 8. From these observations, we also concluded that app generators use boilerplate code.

### 6.2 Permissions requested

The AndroidManifest.xml file was investigated to uncover the permissions requested by each app where we found that the IDE apps in all cases requested less permissions ranging from normal permissions to dangerous permissions as shown in table 1. For the SMS apps and Location apps, we observed that the IDE apps requested only two permissions where as the Appinventor apps requested several permissions including some dangerous permissions which we found to be unnecessary.

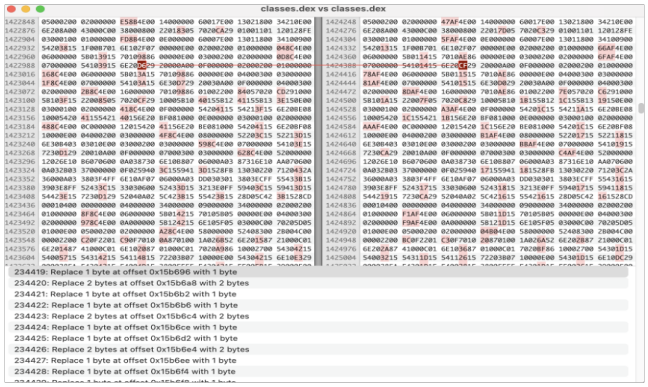


**Figure 7.** Comparison of Dex files of apps from the same app generator.

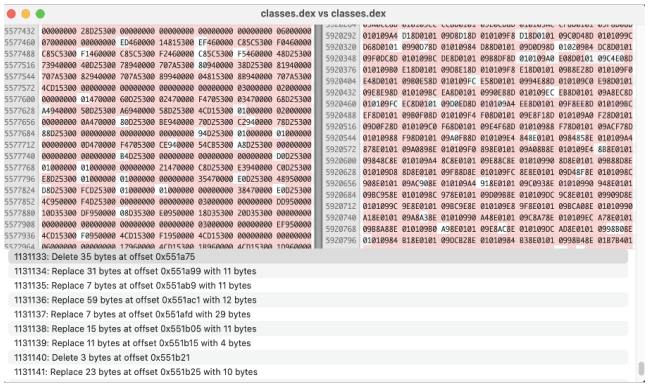


**Figure 8.** Comparison of Dex files of apps from different app generators.

| App | Development Mode | Permissions | Protection level |
|---|---|---|---|
| Greeting | IDE | None | None |
| | OAG | INTERNET | normal |
| | | WAKE_LOCK | normal |
| | | ACCESS_NETWORK_STATE | normal |
| | | RECEIVE_BOOT_COMPLETED | normal |
| | | FOREGROUND_SERVICE | normal |
| SMS | IDE | SEND_SMS | dangerous |
| | | RECEIVE_SMS | dangerous |
| | OAG | SEND_SMS | dangerous |
| | | RECEIVE_SMS | dangerous |
| | | READ_CONTACTS | dangerous |
| | | GET_ACCOUNTS | dangerous |
| | | INTERNET | normal |
| | | USE_CREDENTIALS | dangerous |
| | | ACCOUNT_MANAGER | dangerous |
| | | com.google.android.apps.googlevoice.permission.RECEIVE_SMS | |
| | | com.google.android.apps.googlevoice.permission.SEND_SMS | |
| | | READ_EXTERNAL_STORAGE | dangerous |
| | | READ_PHONE_STATE | dangerous |
| | | MANAGE_ACCOUNTS | signature |
| Location | IDE | ACCESS_FINE_LOCATION | dangerous |
| | | ACCESS_COARSE_LOCATION | dangerous |
| | OAG | ACCESS_FINE_LOCATION | dangerous |
| | | ACCESS_COARSE_LOCATION | dangerous |
| | | ACCESS_MOCK_LOCATION | dangerous |
| | | INTERNET | normal |
| | | READ_EXTERNAL_STORAGE | dangerous |
| | | MANAGE_ACCOUNTS | signature |

**Table 1.** Permissions requested by IDE and OAG apps and their protection level

### 6.3 App Components

We also investigated the AndroidManifest.xml file to identify the components used by the apps. Most of the generated apps used more components than needed as shown in table 2. The two Appinventor apps had an exported Broadcast Receiver component each, however we did not investigate further to determine if their implementations were actually sensitive enough to be used by an attacker to cause harm. For a simple app that only displays a message on a screen, the Andromo greeting app used too many unnecessary components.

| App | Development Mode | Permissions |
|---|---|---|
| Greeting | IDE | 1 activity |
| | OAG | 9 activities |
| | | 9 broadcast receivers |
| | | 5 content providers |
| | | 6 services |
| SMS | IDE | 1 activity |
| | OAG | 1 activity |
| | | 1 broadcast receiver |
| | | 1 content provider |
| Location | IDE | 3 activities |
| | OAG | 1 activity |
| | | 1 broadcast receiver |
| | | 1 content provider |

**Table 2.** Components used by IDE and OAG apps

## 7 CHALLENGES AND LIMITATIONS

There were multiple challenges and several limitations in our project; for starters, before beginning this project, we had limited to no prior knowledge of Android app development and android file structure; this had consumed a good part of our time doing this project to try and get the grasps of Android app development and familiarizing ourselves with the environment, of course, we are not professional app developers at this time, but this experience was much appreciated. This fact has led to time constraints and the fact that one of our intentions was to do static taint analysis using Wala [16] on our apps. However, we were unsuccessful in our quest as we did not have enough time due to our lack of prior knowledge on many fundamental aspects of the project.

We also encountered multiple issues with our app generators along the way. For starters, Andromo was our first choice app generator. However, there were numerous complications in decompiling the apps generated with it; decompiling the apps developed with it created randomly named folders due to heavy obfuscation. Another issue was the fact that there were trial periods on the apps that were created by it, and one of our apps could not be used after one week. Another problem was that most online app generators on the market do not deliver an SMS send and receive feature. They are more focused on instant messaging apps built on top of famous messengers such as Telegram. We needed to try 14 different app generators until we used App Inventor, which could do what we wanted to do reasonably well. However,

App Inventor has had a bug as of writing this report where in Android versions 12 and up, it will face a runtime error when trying to send an SMS; this is a known issue and will get fixed, as the developers mentioned[15]. this has recently got resolved but it was after the time that we were testing our apps.

Another problem that we put upon ourselves was choosing app generators that were deemed safer [11] than other app generators; this led us to fewer vulnerabilities than we anticipated when we started this project. However, as our work has shown, there are still some significant concerns regarding these OAGs as well.

# 8 TAKEAWAYS

There were several takeaways in this work, and we are going to mention some of them. one of these takeaways would be the fact that we tried and succeeded in confirming certain assumptions and proofs that Oltrogge et al. had in their paper on a much smaller scale. We used different tools and analysis processes and learned valuable lessons regarding manual analysis of Android apps and Android app development. We understood the decompilation and analysis process of an app, and we intend to carry this knowledge in our future endeavors. Another interesting point was that during the term, we always heard about android manifest in different papers; this project enabled us to delve into it and understand its various aspects.

We also learned that if an online app generator provides more security compared to other OAGs, it usually faces a more limited number of features and modules to be implemented on the app. this was true for both Andromo and App inventor, other OAGs had significantly more modules, both aesthetically and functionally, to implement. In the current state, online app generators have a negative impact on the security of the overall app ecosystem. However, these OAGs are in a unique position in which they can turn these negative facts into positive ones by adhering to security best practices and spending more effort into securing their application model and infrastructure from which ultimately millions of users benefit. While this is true, if OAGs are to be used, App Inventor could be one of the safest and best options.

# 9 FUTURE WORK

## 9.1 Dataset

In our project, we used a small dataset. For future work, we recommend working with a larger dataset consisting of apps that make use of more sensitive resources.

## 9.2 App Generators

In our project, we used two of the app generators that [11] concluded to be more secure. We recommend using more app generators which we believe could lead to the discovery of more interesting findings and could be used to show if these app generators have actually improved over time or are in the same state as discussed in the paper.

## 9.3 Investigating Vulnerabilities

Android apps use third party libraries in accomplishing several tasks; some of these libraries have been classified as tracking libraries. If the libraries used by these generators exfiltrate data especially sensitive data that both the user of the app generator and the end-user of the app are not aware of, then this is a threat to user privacy. It is most likely the libraries are not malicious, however an attacker could also threaten user privacy by intercepting an insecure communication between the libraries and the target server. An attacker could also conduct component hijacking in accessing sensitive resources. We couldn't confirm if the apps we generated have these vulnerabilities as our project focused on identifying violations of best practices used in guiding against introducing vulnerabilities. For future work, we recommend conducting further investigations, for example, static taint analysis to confirm if these violations actually lead to potential security and privacy threats.

## 9.4 Automated Decompilation

During decompilation, we used several tools, some of which were different from what the other used. For future work, we recommend using a systematic uniform approach in decompiling apps to ensure that the same steps are taken to achieve similar results.

# 10 CONCLUSION

In conclusion, this project recreated some aspects of research done by Oltrogge et al. in analyzing the general security of apps created by online app generators on a smaller scale using manual analysis. We developed three and generated three apps with different levels of runtime permissions and decompiled them to get to the resources that were needed. By comparing the android manifest and Dex bytecode of the apps, we discovered that firstly, the apps generated with OAGs use more permissions than they need to function correctly; secondly, they use boilerplate code that would lead to reconfiguration attacks and more critical vulnerabilities, and finally, there is the usage of components that exceed the need of the particular apps that we created. Due to the use of boilerplate code and the prevalence of apps generated by OAGs in the market, vulnerabilities caused by these app generators have a significant impact. However, this fact cannot be denied that online app generators ease the process and reduce costs in developing mobile applications. They can be used with more trust if they adhere to security best practices.

## References


[1] Android Manifest 2022. https://developer.android.com/guide/topics/manifest/manifest-intro

[2] Android penetration tools walkthrough 2018. https://resources.infosecinstitute.com/topic/android-penetration-tools-walkthrough-series-dex2jar-jd-gui-baksmali/

[3] Android Studio Dolphin 2022. https://developer.android.com/studio

[4] Andromo 2022. https://www.andromo.com/

[5] APKTool 2022. https://ibotpeaches.github.io/Apktool/

[6] Dex2Jar Repository in Github 2021. https://github.com/pxb1988/dex2jar

[7] Hex Fiend 2021. https://hexfiend.com/

[8] JadX Repository in Github 2022. https://github.com/skylot/jadx

[9] JDGui 2019. http://java-decompiler.github.io/

[10] MIT App Inventor 2022. https://appinventor.mit.edu/

[11] Marten Oltrogge, Erik Derr, Christian Stransky, Yasemin Acar, Sascha Fahl, Christian Rossow, Giancarlo Pellegrino, Sven Bugiel, and Michael Backes. 2018. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. In *2018 IEEE Symposium on Security and Privacy (SP)*. 634–647. https://doi.org/10.1109/SP.2018.00005

[12] Permissions in Android Manifest 2022. https://developer.android.com/reference/android/Manifest.permission

[13] Permissions overview in Android Documentations 2022. https://developer.android.com/guide/topics/permissions/overview

[14] Procyon Repository in Github 2021. https://github.com/mstrobel/procyon

[15] SendSMSDirect Issue in App Inventor 2022. https://community.appinventor.mit.edu/t/getting-runtime-error-when-trying-to-use-serial-to-communicate-to-arduino/66235/28

[16] WALA Repository in Github 2022. https://github.com/wala/WALA