Fuzzing Android Intent Receivers*

Shridhar Suryanarayan[1 [20949465]], Suhail Tharappel Mohammed[1 [20985902]], and Sheng Hao[1 [20963483]]

[1] University of Waterloo
[2]

Abstract. This project focuses on testing the broadcast receiver component of an android app through black-box fuzzing. To achieve this goal, a Python-based Command Line Interface(CLI) tool was developed to automatically scan the app for registered broadcast receivers and test them by launching Intents targeted at each identified receiver. The objective is to provide android developers with an easy-to-use tool for testing broadcast receiver components through fuzzing.

Keywords: Fuzzing · Android Apps · Intent · Broadcast Receiver · Android Manifest · Apktool · Android Debug Bridge · MIME Types · Logcat

1 Introduction

Android OS is one of the most popular mobile phone operating systems, currently powering millions of smartphones, wearables, tablets, and car infotainment systems. Sharing content from app to app is a powerful feature and is usually overlooked and taken for granted. In particular, with the use of various android applications, we are able to share media files with our contacts. Some of the key Android components that enable us to do this are Intents and Broadcast Receivers, allowing android applications to safely communicate between processes. Intents act like an OS-wide parcel service i.e., they allow an app to invoke another app and exchange data. Broadcast receivers consume and process Intents, allowing android applications to register application events that enable sharing the media files.

An interesting paper that we came across "DroidFuzzer", [8] uses Intents as pathways into the application to inject them with randomly generated data to ensure that the application is stable and well written. Our solution "Buzz"(Broadcast-Fuzzer) stems from this basic idea of DroidFuzzer. Our goal is to create an open-source tool capable of fuzzing a target android applications by injecting fuzzed data. Android developers and testers can use this tool to monitor the response of the target application and view reports that help identify software bugs and vulnerabilities.

This report will cover different phases of our project "Buzz" and how we executed it. We will begin by going over some Android Terminology that would

* University of Waterloo

be essential in understanding how our solution works and what we intend to do. Next, we dive into the details of the implementation, functionalities, architecture, and limitations of Buzz. Finally, we conclude by discussing the effectiveness of this solution and showcasing our observations of how different applications responded to the tests conducted.

2 Android Terminology in Understanding the Project

The following terminology is essential and will help understand the implementation and capabilities of Buzz:

Intent Abstraction of the action to be done. It has several use cases like starting apps, sending data to Broadcast Receiver components, etc. [8].

Broadcast Receiver A component within an app used to receive broadcasts of interest from the android OS or from other applications [15].

AndroidManifest.xml Every app contains this file in the root directory of its source code. This file contains information about app permissions, inter-app communications channels, resource files, hardware, and software the app requires etc.[13].

Apktool An open-source tool used to reverse engineer android ".apk" files [11].

ADB Android debug bridge is a command line tool used to communicate with the android device. ADB allows a high degree of control over the device [10].

MIME Types Indicates the type of data that the app is willing to receive/send [14].

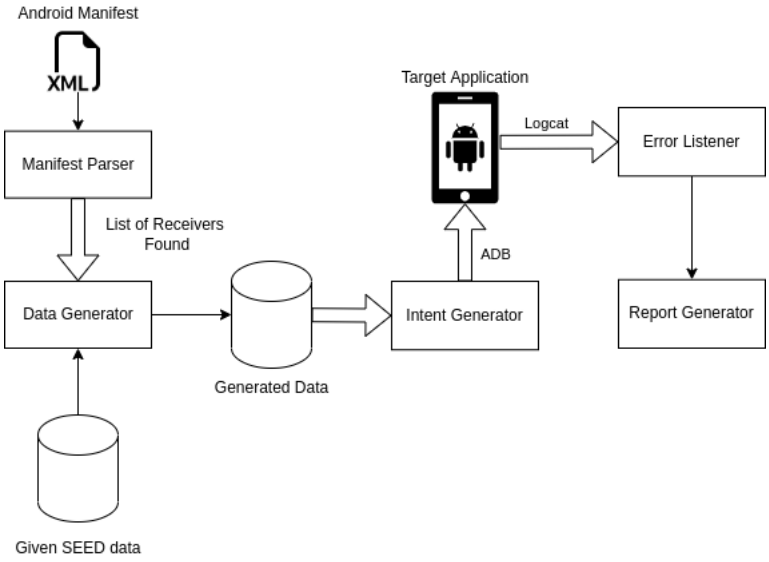
Logcat It is a command line tool that is used to fetch logs from an android device's log buffer [12].

3 Our Solution: Buzz

Buzz (Broadcast-Fuzzer) is a CLI tool that analyzes the android application's manifest and generates test cases by generating fuzzed data that target the broadcast receiver component in android apps.

3.1 Architecture and Flow of Control

Fig. 1. Architecture and Flow of Control Graph



3.2 Modules and Functionalities

Manifest Parser We use the android app manifest to detect registered broadcast receivers. It is a standard XML file and can be parsed using an XML parser. We primarily target the following information.

- Package name
- Activities with Intent-filters

The package name is used to uniquely identify the application. Activities with Intent filters indicate the presence of a broadcast receiver in that activity. By parsing the manifest, we obtain the activity name associated with an Intent, the Intent's action name, and the Intent's data MIMEType. Using this parsed information, we construct an Intent template for each broadcast receiver. This Intent template is used with multiple fuzzed data files to fuzz the target application.

Data Generator

Atheris Fuzzer
Our initial idea was to use Atheris as a tool to generate fuzzed data for various MIMEtypes used in an Android app. However, Atheris didn't work as expected for our solution. Atheris is built on top of Libfuzzer and they both are coverage-guided fuzzers. This means, that for Atheris to work, it requires both, the source code of the target app and a fuzz target.

The broadcast receiver component cannot be called directly using Atheris. To invoke the broadcast receiver, a registered event must occur. Another limitation with Atheris is that it only works with Python. This means, that Android applications that are written with Java or Kotlin cannot be fuzzed with Atheris. As a result, it is difficult to create a fuzz target and as a result, Atheris is not a viable solution for Buzz. While developing Buzz, we realized that we require a black-box fuzzer rather than a coverage-guided fuzzer.

Fig. 2. Atheris Sample code

```
import atheris
import sys

def TestOneInput(data):
    if data == b"bad":
        raise RuntimeError("Badness!")

atheris.Setup(sys.argv, TestOneInput)
atheris.Fuzz()
```

A simple example of Atheris fuzzer is shown above, this code will raise a RuntimeError when input data is "bad". Atheris will generate random input data and run into a branch where a crash occurs. With Atheris not being a viable solution, we had to create fuzzed data by ourselves. Here are some of the file types that we fuzzed and how we create fuzzed data for each one of them.

Creating fuzzed PNG files *[20]*

PNG File Structure

Table 1. Structure of a simple PNG file

89 50 4E 47 0D 0A 1A 0A	IHDR	IDAT	IEND
PNG signature	Image header	Image data	Image end

PNG signature: it indicates this file is a PNG file. We are not modifying this part to generate fuzzed data.

IHDR chunks: After the PNG signature, next comes a series of chunks, each of which conveys certain information about the image. We are fuzzing this part to generate fuzzed data. All IHDR chunks are listed below:

– Header contents size

– "IHDR" identified header chunk
– Image pixel width
– Image pixel height
– How many bits per pixel
– Color type
– Compression method
– Filter method
– CRC of chunk's type and content

IDAT chunks: It contains the output data stream of the compression algorithm. There can be multiple IDAT chunks in a single PNG file. We are also fuzzing this part to generate fuzzed data. All IDAT chunks are listed below:

– IDAT chunk size
– "IDAT" identified header chunk
– Compression method
– ZLIB FCHECK value
– Compressed DEFLATE block using static Huffman code
– ZLIB check value
– CRC of chunks' type and content

Image end(IEND) chunk: it identifies the end of the image

Techniques used to generate fuzzed data for PNG files

We have two approaches to fuzz PNG files. The first approach is fully randomized. In this method, we first select a random number to determine data length, we then pick a random index. Using these values we replace the original data with a randomized hex value. The second approach is more controlled, here we only fuzz some of the important chunks listed in the PNG file. Just like the first method, we replace the selected chunks with some random hex values. Both the methods work successfully, however, we noticed that some images generated with our fuzzing technique can still display properly, while others are damaged.

Creating fuzzed TXT files *[19]*
Based on our research, TXT file format only contains plain text and has no special formatting such as bold, italic, images, etc. Thus, our approach for fuzzing this file type is just generating a random string and storing it in a TXT file.

Creating fuzzed MP4 file*[16] [17] [18]*
MP4 files are complicated when compared to the others; they have many different chunks, and each chunk indicates different data being used by the MP4 file. Some examples are listed below:

– ftyp: contains file type, description, and common data structure used
– pdin: contains progressive video loading and downloading information

- moov: contains all movie metadata
- moof: contains movie fragments
- mfra: it is a container with random access to the video fragments
- mdat: it is a data container for media
- stts: Sample-to-time table
- stsz: sample size
- stts: it is a container with the metadata information

Some of these chunks even have subtypes. For example, "ftyp" has "avc1", "iso2", "isom", "mmp4", etc.

Techniques used to generate fuzzed data for MP4 files
We used two approaches to create fuzzed data for MP4 files. Just like the method for PNG files, the first approach has to do with randomly replacing hex values. The second approach involves replacing data chunk headers with other headers, for example, replacing "isom" with "iso2". After creating fuzzed data, we found that some of the fuzzed MP4 files work normally, some fuzzed MP4 files exhibited abnormal behavior (like altered brightness), and some fuzzed MP4 files become unusable(corrupted).

Intent Generator This part of the application uses ADB command line tool to send Intents to the android app. Initially, the android Logcat buffer is cleared to ensure that errors that occurred prior to the test are not caught again. After this, the Intent generator launches Intents using the obtained package name, MIMEType, and component name. One Intent is launched for each file that has been generated to test a given broadcast receiver. Each broadcast receiver that has been discovered from the manifest will be tested one after the other.

Error Listener After launching an Intent, the target application is monitored for failures for a fixed amount of time (13 seconds). The error listener uses Logcat command line tool to fetch the android system log dump. These logs are filtered to find any failure messages that are written by the target app. Buzz looks for phrases/keywords like "SIGSEGV", "NullPointer", "Error", "Runtime-Exception", etc. This filter can be broadened or narrowed to fit the needs of the application tester.

If an error/failure message is detected, the error listener stops listening to Logcat and records the error. If errors are not detected within the given timeout, the test case is assumed successful.

Report Generator The report generator is invoked when the error listener detects an error. It captures the error detected by the error listener and creates a folder containing the error log and the files that caused it.

3.3 Limitations

Limited data types Currently, Buzz only supports testing of Intents with three data types: PNG files for images, MP4 files for videos, and text.

Only one broadcast receiver at once Unable to run testing in parallel for multiple broadcast receivers. This means, there is only one instance of the target application being tested at a time.

Only one device supports at a time Only one connected android device can be connected to the device running Buzz at a time.

Unable to Buzz apps with DexGuard Applications that use DexGuard to protect their code (example: Facebook) cannot be tested because their manifest files cannot be decompiled successfully.

4 Applications Fuzzed Using Buzz

Fig. 3. Summary of Tests

S/N	App name	Package name	Number of Registered Receivers	Number of runs per receiver	Failure reports generated
1	Telegram	org.telegram.messenger	9	3	1
2	Instagram	com.instagram.android	9	5	2
3	Snapchat	com.snapchat.android	14	3	0
4	Tinder	com.tinder	1	3	0

4.1 Instagram

Upon fuzzing Instagram, we discovered a lot of unexpected behavior in the app. We fired an Intent with fuzzed data on the Instagram app and were able to access the messaging screen and Instagram's default post feed page, without being logged in. These screens were empty and contained no information but did expose a vulnerability. Given that these screens open without any form of validation shows the app doesn't check login status before opening an activity. A video showcasing this behavior is provided in the GitLab repository.

In our initial run, we noticed that Instagram generated over 11 failure reports. Unfortunately, some of these reports turned out to be false positives. Adjusting the filters used in Buzz's error listener, helped narrow it down to only two run-time errors. Interestingly, we found that Instagram has been obfuscated. This meant that we were unable to determine the name of the classes or components that caused the errors caught by Buzz.

4.2 Tinder

With Tinder, we observed one unexpected behavior. Like Instagram, without logging in, Tinder allows fuzzed data to come in and pop-up image crop page. However, the app doesn't really allow us to edit the image in this state. An error message saying "Oops! Something went wrong" will be displayed as soon as we click the "Done" button. No error report is generated for this unexpected behavior. A video is provided from the GitLab repository.

4.3 Snapchat

Snapchat has handled its app-to-app communication quite well. In all cases, Snapchat showed an unsupported content error.

4.4 Facebook

This application posed an interesting challenge. We were unable to decompile this application through Apktool. Online research showed that the Facebook application uses DexGuard to protect its APK. DexGuard encrypts app resources, strings, and obfuscates the source code of the application. Apktool encounters an error when trying to decompile applications protected using DexGuard. Hence, we were unable to fuzz Facebook.

4.5 Telegram

Telegram is open source and is known to be quite reliable. For this reason, it was one of the first applications we tested. Using Buzz, we were able to uncover a bug in Telegram's photo editor. We noted that Telegram does not check the validity of the image data. If a corrupted image is opened on Telegram's photo editor and the crop button is pressed the app crashes immediately.

5 Future Work

Support more datatypes Our project at this time only supports three data types: PNG, TXT, and MP4. In the future, we will make Buzz handle more data types. GIF, JPEG, and WAV Buzzed files generator will be added for the next version.

Multiple devices at a time It will handle more devices at a time in the future for convenience.

Test with more apps More apps should be tested using Buzz. Right now, we are only testing Buzz with social media apps, other apps like an image editor or voice editor could be tested using Buzz.

Improving design of random data generator As of now, we only have two approaches to generating fuzzed data, one is fully random, and the other is replacing chunk headers. Having a deeper understanding of the file formats would enable us to develop better techniques.

Apktool integration For the first version of Buzz, we use Apktool manually to obtain the manifest files. A new version of Buzz that is capable of automatically decompiling the APK using Apktool and fetching the manifest files would be more convenient.

6 Conclusion

Millions of people today, use devices powered by Android OS. Android provides a plethora of applications and features. One of these features has to do with sharing files (like pictures, videos, texts, etc.) from one device to another and inter-app communication plays an important role in this process. Two key Android components: Intents and Broadcast Receivers are required for safe inter-app communication.

Our project, Buzz is a CLI tool that allows software developers to test Android applications by fuzzing broadcast receivers. Buzz provides a customizable tool to create fuzzed data for various types of Intents and creates reports for any bugs or vulnerabilities identified after fuzzing. For the scope of this project, we fuzzed using three different data MIMEtypes: PNG for image files, MP4 for video files, and TXT for text files. We used Buzz to fuzz five popular applications: Instagram, Tinder, Snapchat, Facebook, and Telegram. Out of these, Snapchat was the only app that didn't show any unexpected behavior. Facebook's manifest was encrypted with Dexgaurd, not allowing Buzz to fuzz it. For all the other applications, Buzz generated at least one error report.

The findings and insights gained in the process of fuzzing all these applications provoked ideas that will help improve Buzz's functionalities in the future. Additionally, as users of Buzz, we were able to identify features that will help improve Buzz's user experience.

References

1. Stack Overflow. 2022. How to use APKTool to extract the apk files without baksmaliing (Exactly as unzip do). [online] Available at: https://stackoverflow.com/questions/37126758/how-to-use-apktool-to-extract-the-apk-files-without-baksmaliing-exactly-as-unzi [Accessed 8 August 2022].
2. Gouchet, X., 2022. Launch Intents using ADB - Android and beyond. [online] Xgouchet.fr. Available at: https://www.xgouchet.fr/android/index.php?article42/launch-Intents-using-adb [Accessed 8 August 2022].

3. Khan, Z., 2022. Sending Intent to BroadcastReceiver from adb. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/22634446/sending-Intent-to-broadcastreceiver-from-adb [Accessed 8 August 2022].
4. Android Developers. 2022. Android Debug Bridge (adb) — Android Developers. [online] Available at: https://developer.android.com/studio/command-line/adb [Accessed 8 August 2022].
5. Stack Overflow. 2022. How to use 'subprocess' command with pipes. [online] Available at: https://stackoverflow.com/questions/13332268/how-to-use-subprocess-command-with-pipes [Accessed 8 August 2022].
6. Android Developers. 2022. SDK Platform Tools release notes — Android Developers. [online] Available at: https://developer.android.com/studio/releases/platform-tools [Accessed 8 August 2022].
7. Hui Ye, Shaoyin Cheng, Lanbo Zhang, and Fan Jiang. 2013. DroidFuzzer: Fuzzing the Android Apps with Intent-Filter Tag. In Proceedings of International Conference on Advances in Mobile Computing and Multimedia (MoMM '13). Association for Computing Machinery, New York, NY, USA, 68–74. https://doi.org/10.1145/2536853.2536881
8. Android Developers. 2022. Intent — Android Developers. [online] Available at: https://developer.android.com/reference/android/content/Intent [Accessed 6 August 2022].
9. Android Developers. 2022. BroadcastReceiver — Android Developers. [online] Available at: https://developer.android.com/reference/android/content/BroadcastReceiver [Accessed 6 August 2022].
10. Android Developers. 2022. Android Debug Bridge (adb) — Android Developers. [online] Available at: https://developer.android.com/studio/command-line/adb [Accessed 6 August 2022].
11. Ibotpeaches.github.io. 2022. Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps.. [online] Available at: https://ibotpeaches.github.io/Apktool/ [Accessed 6 August 2022].
12. Android Developers. 2022. Logcat command-line tool — Android Developers. [online] Available at: https://developer.android.com/studio/command-line/logcat [Accessed 6 August 2022].
13. Android Developers. 2022. App Manifest Overview — Android Developers. [online] Available at: https://developer.android.com/guide/topics/manifest/manifest-intro [Accessed 6 August 2022].
14. Developer.mozilla.org. 2022. MIME types (IANA media types) - HTTP — MDN. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/BasicsofHTTP/MIMEtypes [Accessed 6 August 2022].
15. Android Developers. 2022. Broadcasts overview — Android Developers. [online] Available at: https://developer.android.com/guide/components/broadcasts [Accessed 6 August 2022].
16. Chishti, M., 2022. MP4 File Format. [online] Docs.fileformat.com. Available at: https://docs.fileformat.com/video/mp4/ [Accessed 8 August 2022].
17. File-recovery.com. 2022. MP4 Video Signature Format: Documentation and Recovery Example. [online] Available at: https://www.file-recovery.com/mp4-signature-format.htm [Accessed 8 August 2022].
18. GOV.UK. 2022. Details for: Quicktime [online] Available at: https://www.nationalarchives.gov.uk/PRONOM/Format/proFormatSearch.aspx?status=detailReport&id=658&strPageToDisplay=signatures [Accessed 8 August 2022].

19. Definitions, T. and Hope, C., 2022. What is a Text File?. [online] Computerhope.com. Available at: https://www.computerhope.com/jargon/t/textfile.htm [Accessed 8 August 2022].
20. En.wikipedia.org. 2022. Portable Network Graphics - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Portable_Network_Graphics [Accessed 8 August 2022].
21. "Obfuscation (software)," Wikipedia, 06-Jul-2022. [Online]. Available: https://en.wikipedia.org/wiki/Obfuscation(software). [Accessed: 09-Aug-2022].