



**NUS**  
National University  
of Singapore

**School of  
Computing**

NATIONAL UNIVERSITY OF SINGAPORE

CP 5101

SCHOOL OF COMPUTING

---

# **Android Analysis Tool for Eavesdropping Applications**

---

A DISSERTATION SUBMITTED FOR THE DEGREE OF MASTER OF  
COMPUTING IN INFOCOMM SECURITY

Author:

**Louis Galland**

Supervisor:

**DONG Jin Song**


**2021**

# Declaration

I hereby declare that this dissertation is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in this dissertation.

This dissertation has also not been submitted for any degree in any university previously.

Louis Galland  
18 April 2021

A handwritten signature in black ink, appearing to read 'L. Galland', with a horizontal line underneath the name.

# Contents

<b>Abstract</b>	<b>3</b>
<b>Introduction</b>	<b>4</b>
<b>1 Background on Eavesdropping application Analysis</b>	<b>6</b>
1.1 Literature review on Eavesdropping and Cross device tracking applications . . . . .	6
1.2 Introduction of Techniques . . . . .	7
1.2.1 Eavesdropping techniques and patterns . . . . .	7
1.2.2 Analysis Methods for detecting Eavesdropping . . . . .	9
<b>2 Design and Implementation of Eavesdroid tool</b>	<b>11</b>
2.1 Tool usage and specifications . . . . .	11
2.2 Static Analysis tool set . . . . .	12
2.3 Dynamic Analysis tool set . . . . .	12
<b>3 Case Study: Alphonso ACR</b>	<b>14</b>
3.1 Alphonso Software . . . . .	14
3.2 Basic Dynamic analysis . . . . .	14
3.3 Basic Static Analysis . . . . .	15
3.3.1 APK architecture . . . . .	15
3.3.2 APK's Manifest file . . . . .	16
3.3.3 App behaviour . . . . .	17
3.4 Audio Recording . . . . .	17
3.4.1 Audio Recording triggering . . . . .	17
3.4.2 Recording parameters . . . . .	17
3.5 Audio Treatment . . . . .	18
3.5.1 Automated Content Recognition mods . . . . .	19
3.5.2 ACR additional properties . . . . .	19
3.5.3 Network Analysis . . . . .	21
3.6 Cross-Device Tracking . . . . .	22
3.7 Limits of the Eavesdroid tool . . . . .	23
<b>Conclusion</b>	<b>24</b>
<b>References</b>	<b>25</b>

# Abstract

The tracking of consumers' habits today is increasingly widespread. People being more interconnected than ever and spending a great deal of time on their devices, the data on their consumption habits is very valuable. Hence, advertisement companies develop applications aiming to extract the most data possible from their users. In particular, some applications eavesdrop on the user watching TV or using other devices. These eavesdropping applications monitor the surrounding sound of the users' devices - smartphone, tablet or smartwatch - to comprehend what the user is doing and might be watching on TV. To do so, applications even embed ultrasounds to covertly transfer data from the TV to the devices. This collected data is used to create and display targeted advertisements for the users based on their TV consumption. It is also used to link the different devices of a user with cross device tracking, for the purpose of displaying advertisements previously viewed on another device. These applications, often privacy invasive and somewhat unethical, use this precariously aggregated data to build a complete profile of their users. In this study we are hence exploring eavesdropping and cross device tracking applications on Android.

Particularly, we develop an analysis tool designed to support the analysis of Android applications and to detect eavesdropping behaviours. This tool assists in the reverse-engineering and dynamic analysis of an Android application. It relies on the different behaviours that can be identified on eavesdropping applications in order to assist the work of a malware analyst. We then use this tool to make the assessment of applications embedding Alphonso's software. Alphonso is an Indian company eavesdropping the users of Android applications for commercial purposes. The tool is employed to identify the technical behaviour of these applications but also to disclose their overstretch usage of eavesdropping and cross device tracking techniques. We show that these applications lie on some of their features and dissimulate private data besides the recorded audio. We then discuss the limits of this tool within the analysis of such eavesdropping applications.

# Introduction

Is Facebook listening to you through your phone? There is a widely held belief that users are being listened to through their own smartphones, tablets or computers. It often happens to be served with an advertisement, a social media content, a news article related to what we have just talked about or watched a few minutes before. For many users, it seems like all of the conversations and the multimedia content they watched were being recorded to serve targeted ads. The veracity of big technological companies' applications eavesdropping on their users has not been established. Even so, it is clear that advertising companies could find a practical use of this data, such as audience analytic or proximity marketing (displaying information based on where the user is). Besides, they can deliver advertisements targeted specifically to individuals based on their habits. Targeted advertising meaningfully improves the response of users to advertisements and thus the profit for these companies. Particularly, the analysis of what an individual watches on TV would enable advertisers to create targeted ads according to the TV shows, movies and commercials the latter has watched and appreciates.

Furthermore, households are more connected than ever. Virtual assistants, smart televisions, smartwatches, tablets and phones integrate microphones that could seamlessly record the users' surroundings. An application on a user's phone could continuously record audio in order to eavesdrop what the user is watching on any of the devices the user owns. Additionally, the latter can gather many metadata about the habits and devices of the user to perform cross device tracking. It builds a complete profile of the user, including links between the devices, to display correlated advertisements between the devices.

Recently, the existence of applications implementing similar techniques have been brought up, in particular on Android, being very permissive for developers. The applications can access the device's microphone and collect audio data even when the user is not using them. Downloading this kind of applications is equivalent to giving an unlimited access to our privacy. However, the ingenuity of this process is that this malicious code can be implemented and hidden in any kind of application, including video games designed for children. An important breach in the parents' privacy can be created after their child having innocently downloaded a malicious video game. Alphonso and Silverpush have been two actors of this Android eavesdropping. Pieces of malicious software have been implemented in many Android and iOS applications, hidden in video games and genuine apps, allowing these companies to eavesdrop the users' TV in order to obtain relevant information as presented before.

Many mobile malwares have been developed recently to benefit from security breaches and users' negligence. Criminals have aimed to create new pieces of malicious software to leak confidential information and take control of systems. In the meantime, Android has developed its security mechanisms and jurisdiction in order to protect its users at best. Nevertheless, these eavesdropping applications are a new type of malicious software, at the boundary between spyware and adware. These applications have been sparsely tackled compared to more usual and dangerous pieces of mobile malwares.

To avoid illegality, eavesdropping applications often warn the users about their intentions. Yet many features can be missing from the applications' warnings and remain hidden from the users. For example, these applications can obtain complementary data about the device, such as location

or network information. Moreover, the information shared by these applications are relatively unknown, and the users do not necessarily expect their data to be passed along without their knowledge. Because these applications represent a real privacy threat, as the entire daily life of the user can be recorded and many data about their habits can be leaked, they recently came under scrutiny of the media and political institutions. An important regulatory work still remains in order to categorise these technologies. Since then, working on the detection of these applications is essential to remove them from available stores or warn the users.

In that respect, we present in this study different eavesdropping applications and analyse their different behaviours. This paper makes the following contributions:

- We introduce the different methods used by eavesdropping applications to record audio and perform cross device tracking. We present the different tools and methodologies to analyse these applications.
- We implement a specific Android tool, used to assist in the reverse-engineering and dynamic analysis of eavesdropping applications.
- We evaluate this tool by assessing the eavesdropping behaviours of Alphonso Software applications. We show the discrepancies between what is discovered during the analysis and what the company initially asserts.

# Chapter 1

## Background on Eavesdropping application Analysis

### 1.1 Literature review on Eavesdropping and Cross device tracking applications

Eavesdropping applications have often been used for commercial purposes. These applications can gather information about what their users talk about, what they watch on TV, movie theaters or computers, which commercials they hear. This profiling of users is extremely important and valuable, mostly for advertisers who can use the user habits and preferences to offer more relevant and more targeted content.

User profiling can be improved with cross device tracking (XDT) technologies. This tracking is based on discovering the multiple devices of an user and pairing them. This pairing enables to offer related content between two devices, for instance displaying an advertisement that the user previously viewed on TV. It also enables to track the user's habits on different devices.

One of the most notable examples of Android cross device tracking is Silverpush's ultrasonic usage of cross device tracking. Silverpush is an Indian company, which provides a XDT framework for many Android applications developers. This framework uses the device's microphone to listen to the user's TV in order to understand what the latter is watching and provide the user targeted advertisements. In particular, Silverpush uses ultrasonic "Audio Beacons". These inaudible sounds are codes that are transmitted by the user's TV and recognized by the user's Android device. This ultrasound technology has the benefit of directly sending these audio codes and does not treat the entire audio captured by the microphone in order to recognize which program is running on the TV.

In addition to replaying the ads watched on TV directly on the user's phone or tablet, Silverpush also uses this ultrasonic communication to pair the users' devices: its phone, tablet, computer, television, radio etc. The advertisers can then gather information about the user's devices, Internet habits but also more hidden information such as the user's location and lifestyle.

Despite being legal, these applications are intrusive and not transparent enough about what is done with the user's tracking and what is gathered by the application. Since 2015, the United States Federal Trade Commission has raised many concerns about the privacy and security of monopolising this private data. In 2017, they issued a report about cross device tracking [1] and warned different Android application developers to remove the Silverpush framework as a violation of the FTC act.

A comprehensive technical analysis of Silverpush's eavesdropping framework has been presented in 2016 by Vasilios et al. during the Black Hat Europe conference [2]. The analysis of applications embedding Silverpush SDK revealed many unspecified features and security concerns. Among others, Silverpush runs silently in the background even when the application is not used or when the phone is locked. Moreover Silverpush permanently uses the device's microphone and listens

for audio beacons at any time of the day or night.

Most of cross device tracking solutions use the device microphone to interact (either with ultrasound or plain audio), relying on the audio recording permissions of the applications to operate. Matyunin et al. [3] showed that permissions are not necessarily the key focus of cross device tracking for Android and developed a zero-permission XDT framework using gyroscopes in devices as receivers for ultrasonic signals.

Other research, including Zimmeck et al.'s research presented at USENIX Security Symposium 2017 [4], disclose the usage and opaqueness of mobile and desktop trackers used in cross device tracking in order to pair the users' devices and habits.

Different pieces of eavesdropping software have been disclosed in Android applications. In 2017, the Indian company Alphonso has been convicted of embedding audio recording and automated content recognition features in various Android applications and games. A technical analysis of this software will be studied in this paper in Chapter 3.

## 1.2 Introduction of Techniques

### 1.2.1 Eavesdropping techniques and patterns

Eavesdropping applications use common techniques to record the user's surroundings audio and recognize audio contents. Identifying these techniques during the analysis of an application can help determine if the application is eavesdropping the users. In case such behaviours are detected, it is also possible to understand the purpose of the application.

The analysis of the application's permissions provides a first identifier of eavesdropping. For an Android application, only the permissions listed in the manifest file of the application will be provided to the application. In addition, if the application tries to use or request any other permission, the call will fail. Hence, the manifest contains a comprehensive list of all the permissions that might be used by the application. In the case of an application that does not seem to require audio recording permissions, the permission `Record.Audio` present in an application's manifest file already suggests eavesdropping.

This permission - along with other permissions to grant access to the storage, contacts and SMS and location - are considered by Android as dangerous and needs the user to grant this permission during runtime. The analysis of an eavesdropping application requires to investigate on how the permissions are requested by the application [5].

- In versions prior to Android 6.0, Android permissions are requested before installation. The entirety of the permissions requested by the application need to be granted by the user in order to run the application, and the user cannot refuse a specific permission. Many eavesdropping applications and general Android malware have been taking advantage of Android's implementation of permissions to request users a considerable list of dangerous permissions that are rarely denied.
- Dangerous permissions are requested at runtime in newer versions of Android. When needing a specific permission, the application requests it. To evade suspicion, eavesdropping applications can trick the user into pretending that the permission request is genuine. By embedding purposeless features requiring audio recording permissions, an application can request the permission once with a valid reason. The permission is thus not requested at an unexpected moment; yet the permission is still granted permanently and available for every other hidden feature of the application.

The application can also trick the user into granting the permission, using clickjacking [7]. Clickjacking overlays additional information on top of another application or of a system interface, such as a permission request page, in order to modify what appears on the User Interface. The "Deny/Allow" can be the only element of the permission request page to



appear on the user's screen, while the other elements are covered by other pieces of information. The user is tricked into allowing permissions by accepting what seems to be something else.

In addition to analysing how the permissions are requested, it is also important to verify the behaviour of the application when the permissions are modified during runtime. An application with a service permanently recording audio might crash in case the audio recording permission is denied at runtime.

Yet, as being said previously, it is not mandatory for Android applications to use the audio recording permissions to perform cross device tracking.

The analysis of eavesdropping components of an Android application includes the assessment of the audio recording features. Audio recording libraries, such as **MediaRecorder** and **AudioRecord**, are used by an application to access the device's microphone. The usage of these libraries needs to be traced in order to observe potential eavesdropping behaviours [6].

- Eavesdropping applications try to record audio without the user's acknowledgement. The recording activities often start when booting the application, and no input from the user is needed to trigger these activities. This behaviour can distinguish an eavesdropping application from a genuine application that simply records audio on the user's demand. This difference can be observed by tracing down the API calls for these libraries. This tracing can be done either statically by analysing the control flow of the application or dynamically by hooking the API calls.
- Eavesdropping applications can attempt to record audio continuously and even when the application is not in use. Recording features are thus integrated in services launched by the application. The advantage of these services is that they can run in background when the application is not in use or when the device is locked. The usage of the microphone as well as the services listing ought to be inspected in order to verify if the audio is being recorded in the background and on a permanent basis.
- Many characteristics of the audio recording ought to be analysed in order to understand the behaviour of the eavesdropping application. The parameters and implementation of the API calls for audio recording libraries expose the duration, the frequency, the quality of the recordings. These can also expose the range of the audio spectrum that is being used by the application, disclosing the use of the ultrasound spectrum for instance.

After suspicious audio recording behaviours are being observed, it is necessary to assess what is done with the captured audio samples. The most immediate way for eavesdropping applications would be to send the samples to a server for further analysis. When being recorded, the audio can either be buffered and sent immediately or stored in an audio file that will be sent at the end of the recording. The local storage of the application ought to be analysed in order to find locally stored or database stored audio samples.

The application can also directly treat the audio and extract only relevant pieces of information. This would reduce the size of the information being sent. The treatment of the audio can be performed in different ways. The relevant piece of information can be directly embedded in the audio and extracted. This is the case of ultrasound eavesdropping, where the application would need to extract only the ultrasound range of the audio spectrum and retrieve an encoded information. The audio can also be translated to text using speech-to-text algorithms. The analysis of this text can easily determine what the user is talking about or what the user is watching. The audio samples can finally be fingerprinted. Important characteristics of the audio (for instance peaks in the audio spectrum) are extracted to form a fingerprint, which is later compared to other audio fingerprints to recognize the sample.

Most of the time eavesdropping applications send the recorded data to a distant server. The analysis of these applications requires assessing which hosts the app is contacting. This can be performed statically by observing all of the network identifiers present in the application: IP

addresses, hostnames, ports, subdomains. This analysis can also be performed dynamically by capturing the traffic between the application and the Internet.

The application can send complete audio samples or some pieces of data in the form of fingerprints or codes. In order to perform cross device tracking, the latter also sends other information to the server like the user's location, device's ID and name or network's details [8].

The application can receive data about the audio, such as the translation of the speeches in the audio samples or their computed fingerprint. Applications working with advertisers mostly receive advertisement identifiers in order to display targeted ads according to what has been recorded.

With the above defined behaviours of eavesdropping applications, Android analysis tools can be used to assist in the analysis of these applications. These patterns being recognized in an application would declare it as eavesdropping.

### 1.2.2 Analysis Methods for detecting Eavesdropping

Many tools can be used to analyse an Android package and detect eavesdropping attributes. These tools can be broken down into different categories based on their usage.

**APK downloading.** Application markets on the Web can be used to find and download suspicious APKs. Even if an application has been updated to a genuine version, it is possible to retrieve malicious versions on markets such as APKPure (<https://apkpure.com>), APKMirror (<https://www.apkmirror.com>) and others. Forums and malware databases also contain interesting applications that could have already been suspected of eavesdropping.

**Antivirus Scanners.** A first step to analysing a suspicious application implies scanning the application. Scanners like VirusTotal (<https://www.virustotal.com>) statically inspect Android applications with different antivirus solutions and look for blacklisted references (URLs, strings, libraries...) and threats.

**Disassemblers and Decompilers.** Many solutions are available to disassemble Android Dalvik bytecode and native code into human-readable language. Disassemblers like IDA Pro and Ghidra can be used to disassemble C/C++ native code. Smali/Backsmali and apktool are two tools useful in disassembling Dex files and unpacking APK packages, as well as repacking them. This code can also be decoded directly in its nearly original Java format using tools such as JD-Gui and dex2jar. These tools are almost mandatory for reverse-engineering the application's code.

**Emulation.** Android malware ought to be analysed in a sandboxed environment. Android Studio provides different emulation tools to run the application. For the sake of audio recording and eavesdropping applications, most of the dynamic analysis can be performed on an actual device, for a better integration of the microphone.

**Device Interaction.** During the dynamic analysis of an application, Android Debug Bridge tool (ADB)[9] provides interaction features with the device/emulator. This command-line tool enables to perform a variety of actions on the device and it provides a Unix shell to run commands on the device. This tool enables to analyse the app-specific and shared storage on the device, to connect the device to a network or a proxy, to record the screen or the audio, to stop a service etc.

**Device Logging.** Integrated with ADB, Android Logcat tool shows system messages, such as debugging messages from an application as well as system and applications errors. A lot of logging information can show the behaviour of an application.

**System Dump.** Integrated with ADB, Android Dumpsys tool provides information about the status of system services. This tool enables to diagnose the usage of network, battery, microphone, camera of a device.

**Instrumentation.** Android applications can have their code modified in order to obtain different behaviours when running. Methods and classes can be overridden dynamically using tools such as Soot and Frida.

**APK Signing.** In order to observe the behaviour of modified versions of an APK, it is necessary to sign the repacked application. The APK's signature will enable the application to run on Android devices.

**Network Interception.** Eavesdropping and cross device tracking applications share information with servers. They can send the recorded audio and receive information about it. Tools such as Burp Suite and Android mitmproxy can work as proxies between the application and the server, and intercept and modify the transiting traffic.

An exhaustive tool could be created for the purposes of supporting the analysis of eavesdropping applications. With the integration of different Android analysis tools and the implementation of features specific to eavesdropping, this tool could assist in the detection of malicious usages of audio recording, audio treatment and network traffic.

## Chapter 2

# Design and Implementation of Eavesdroid tool

### 2.1 Tool usage and specifications

Eavesdroid is a Python 3 tool running on Unix-based systems. Eavesdroid aims to facilitate and assist the analysis of Android applications, and more specifically applications embedding audio recording features. Eavesdroid integrates different tools and techniques to analyse the applications both dynamically and statically.

The tool is aimed to be lightweight and user-friendly. It is to be used in parallel with the reverse-engineering of an Android package and during the analysis of an application's behaviour.

Through a static analysis approach, Eavesdroid provides different ways to interact with the application files. The code structure and functionalities are presented in a formalized way to improve the analysis. Static data about audio recording and networking behaviours are caught and aggregated, without running the application.

Through a dynamic analysis approach, Eavesdroid implements different tools in order to interact with the device and the application files. The behaviour of these applications can be analysed and altered by interacting with the device or modifying the application code.

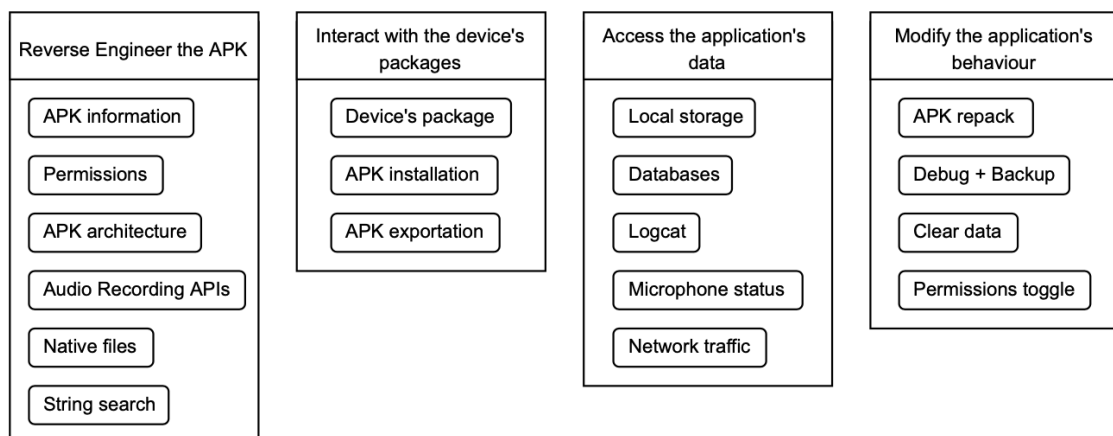


Figure 2.1: Eavesdroid's tool set

## 2.2 Static Analysis tool set

Eavesdroid tools assist the reverse-engineering phase of the analysis of an Android application. Some of those tools specifically tackle the analysis of applications integrating audio recording features. The usage of such features can be tracked and analysed, in order to understand the behaviour of Eavesdropping applications.

**APK disassembly.** Eavesdroid embeds Apktool to disassemble Android packages (APK). The tool unpacks the resource files, assets, certificates, manifest file and also decodes the program's Dalvik binary (.dex files) into human-readable Smali files, close to the original form. These files can be directly reversed-engineered or analysed with the support of other Eavesdroid tools.

**Basic analysis of the APK.** The "Package Info" and "Permissions" tools use the previously decompiled resources to provide a first perspective of the application. The manifest file of an application package contains information about the main package, the application versions, the permissions requested by the app (and their severity). These pieces of information, such as the different permissions used by the application, can demonstrate the potential intention of an application to eavesdrop users.

**Architecture of the APK.** The "APK Architecture" tools display the main packages of the APK's code as well as tree structures of the packages' classes. In case the program's binary is not stripped, this set of tools gives a deeper vision of the application framework and highlights the name of classes that might take a role in audio recording, audio treatment or any other functionality.

**Native Files.** Eavesdroid "Native libraries" tool extracts the decompiled native files contained by the application. These native files are C or C++ code compiled by Android Native Development Kit. Applications often use the Java Native Interface to call functions from native libraries instead of using Java functions, in order to perform computation intensive tasks such as audio treatment in the case of eavesdropping applications. The native files can be disassembled using IDA, radare2, Ghidra or other similar tools. Eavesdroid "Native libraries" tool also uses radare2 to display the an APK's embedded native functions as well as the Control Flow Graph of these libraries.

**String Search.** The "String Search" tool scraps inside the APK's resources, assets and decompiled Smali code to find specific strings. Any String can be specified and searched for inside these files. String groups related to audio recording permissions and network identifiers can be selected in order to locate important strings and their related classes. Calls for audio recording APIs can also be searched in order to locate classes linked with the recording of audio.

## 2.3 Dynamic Analysis tool set

Eavesdroid implements features to interact with the targeted application and with the user's device in order to get a better understanding of the application behaviour. The target application can be run on an Android device with USB debugging activated or in a sandboxed environment. On the one hand, Android emulators would provide better parameter flexibility (location, device name, IMEI...). On the other hand, Android devices would often provide better hardware support (camera, microphone, USB connection...).

Most of the interaction between the tool and the device is operated with Android Debug Bridge (ADB) which gives access to a Unix shell that can be used to run commands on the device.

**APK repack.** Eavesdroid can repack and sign modified versions of Android Packages into an APK file, that can later be installed on a device or emulator. This enables to instrument the code

of an application before being run in order to observe a different behaviour or evaluate the code. This tool is convenient for modifying important data: it can be used to reroute Internet traffic by modifying network identifiers and domains in the resource files, to log hidden values at runtime and to affect the comportment of a feature.

**Device Selection.** The user can choose which device Eavesdroid should target. This can be either an USB plugged-in Android device or an emulator running on the same machine.

**Package Installation and Extraction.** Eavesdroid gives the possibility to install an APK file directly on the device. Eavesdroid tools can also extract a specific package from the device, for instance for the analysis of additional packages installed with Dynamic Code Loading by the initial application.

**Package management.** The tools display every Android package installed on the device. Most of the analysis being run on the same application, the user can select a particular package for Eavesdroid tools to target.

**Package declaration.** Eavesdroid can modify different attributes of the declaration field of an application. Hence, this tool can repack and sign an application as Backupable (`android:allowBackup`) and Debuggable (`android:debuggable`). The backup attribute allows the newly packed application to participate in a backup of its file storage. The debug attribute enables to execute code under the application's privileges to gain full access to the app and extract its data. The modification of the package declaration can be necessary for other Eavesdroid tools to perform.

**Application data interaction.** Eavesdroid can interact with the application's data during run-time. The tools give the possibility to clear the app's data and cache. Restore an application to its original state can be useful to observe the app behaviour at launch time. Eavesdroid tools can also modify the permissions granted to an application while be run. Toggling these permissions is useful to understand the behaviour of an application with and without a certain permission, for instance how the audio recording features of an application would behave when the `RecordAudio` permission is switched off.

**Microphone Status.** The "Microphone Status" tool reports the status of the microphone: whether the microphone is in use at the time of the analysis, when the audio recording has been switched on or off, by which applications or services the microphone has been used. The tool uses Android's `dumpsys`, an Android tool for making diagnostic analysis for the system services running on the device.

**Logcat.** The "Logcat" tool dumps system messages from the device. This includes errors thrown by an application and messages that are output for debugging. Repacking the application as debuggable is often necessary for the good functioning of the "Logcat" tool. The tool enables to search for particular strings and regex, in order to filter the output and obtain effective log results.

**Database dump.** Eavesdroid can export an application's databases in order to retrieve important resources. It uses Android's backup tool to dump data from an application. It then extracts the databases in their current state so that they can be analysed. An application's databases often contain critical data, such as recorded audio samples or computed data in the case of eavesdropping applications.

**Proxy server.** Eavesdroid can intercept HTTP requests and responses flowing from the device. Eavesdroid uses "mitmproxy" tool to make the client work as a proxy between the Android device and the server contacted by the application. All of the traffic containing cross device tracking information can be intercepted and analysed.

## Chapter 3

# Case Study: Alphonso ACR

### 3.1 Alphonso Software

Alphonso is a company aiming to collect TV data about its users. They have developed different phone applications, embedding a piece of software called *Alphonso Automatic Content Recognition (ACR)* that collects information about what users watch on their TV. These pieces of information are sold by the company and used to create targeted apps directly on the users' devices. These applications record the surroundings of the user and treat this data to recognize user's TV programs. Cross device tracing is also central to Alphonso's software that seeks to develop profiles about their users. As declared by the company, the applications do not record human speech and the user always has the choice to refuse this recording. The application also exchanges information that is only related to the audio recordings. Even so, the boundaries with eavesdropping and privacy intrusion are thin. Alphonso has often been implemented on video game applications aimed to a younger public, less aware about these concerns of privacy. Hence, this case study aims to analyse Alphonso's unstated behaviour and privacy issues.

In 2017, many media outlets raised some concerns about Alphonso's actions [10]. Since then, most of the applications got updated or removed from Android's application stores. Versions of the applications supporting Alphonso software are still available on third-party stores and will be addressed as a case study of Android eavesdropping applications.

Many functionalities of Alphonso software, despite the applications still being runnable, have been deactivated. Alphonso servers, reached by the applications embedding Alphonso software, have been disabled. Thus the analysis of such applications is still limited.

Ten applications from four different game editors (Dumadu Games, Chameleo Studio, Augmented Reality Games, Apostek) have been analysed for this case study. All of them have been detected by Virustotal as presenting a security risk (eg. reported as "Android Privacy Risk Alphonso"). The antiviruses highlight the dangerous permissions and signal hooks used by the applications, as well as advertisers' libraries and URLs considered as malicious.

The android game application HoneyQuest was available on the Google store and still is in 2021 on alternative APK stores [11]. The analysis of Alphonso Audio Content Recognition is done on version 1.7 of this application, released in July 2017.

### 3.2 Basic Dynamic analysis

A first runtime analysis is performed thanks to the Eavesdroid dynamic tools. Both location and microphone permissions are requested at launch (Internet permission is granted by default by any app). To prevent any legal consideration, the application warns the user that the "app uses audio to detect TV ads and content and shows appropriate mobile ads" and proposes a way to opt-out from audio tracking in the settings of the app. The application shows no functionality that would

necessitate the use of the microphone, get the user location or access the Internet, nor any user input that would trigger audio recording for the game purpose. Moreover, the "Microphone Status" tool from Eavesdroid shows that, after being launched and even when running in the background or when the phone is locked, HoneyQuest makes constant use of the microphone. Even when opting off from Alphonso Software in the settings, the audio keeps recording. The only way to avoid recording, despite Alphonso's claims of opting-out from the settings, is to remove the audio recording permission from the application .

Unfortunately, due to the deactivation of Alphonso servers, no advertisement is displayed by the application and no evidence of cross device tracking is observed. Many other features, as it will be shown during this case study, cannot behave in their normal manner. Most of the analysis of Alphonso Software's behaviour is tackled statically.

### 3.3 Basic Static Analysis

#### 3.3.1 APK architecture

Among the basic static analysis features we can use the Eavesdroid architecture tools to obtain the architecture of the application. In the case of non-stripped binaries these tools can display an understandable tree structure of the different packages and classes. Among others for the HoneyQuest application, the `com` package embeds the core components of the APK, the `unity` package contains the components used to run the Native unity files for the video game itself, the `tv.alphonso` package (present and identical in every app recognized as embedding Alphonso) contains the core components of Alphonso software (see Figure 3.1).

Many classes from the `tv.alphonso` package are meaningful. These classes expose the presence of HTTP traffic and the usage of databases and audio recording components. This package also proves the existence of services (probably background services) dedicated to audio recording and audio capture.



Figure 3.1: Architecture of `tv.alphonso` package



### 3.3.2 APK's Manifest file

The `AndroidManifest.xml` file contains the structural logic of HoneyQuest's APK. Eavesdroid tool probes inside the Manifest file to display the core information of the application: package information, build versions, permissions used, components of the app (see Figure 3.2).

The service `tv.alphonso.service.AlphonsoService` is linked to the class `AlphonsoService` (found in the architecture tree of the application). Using a service instead of an activity enables to perform long-running operations in the background, even when the user has switched to another application and is not interacting anymore with the game. Even in the background, the Alphonso service can handle network communication, use the microphone to record audio and interact with other components of the application (call activities, send results etc.).

Every advertiser (mopub, inmobi...) declares its broadcast receiver in order to trigger its activity. Every Alphonso application uses intents to communicate with its advertisers' packages and the advertisers packages of other apps embedding Alphonso software.

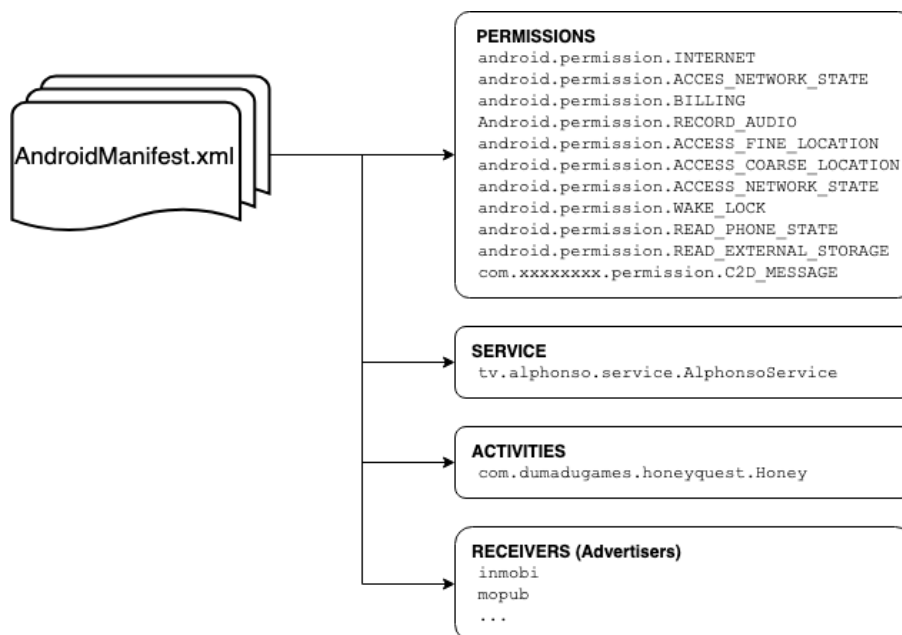


Figure 3.2: `AndroidManifest.xml` information

Finally, Eavesdroid obtains the permissions declared in the Manifest files. Among those, one can find the `RecordAudio` permission but also others with a 'dangerous' protection level like `Internet`, `AccessNetworkState`, `AccessFineLocation`, `AccessCoarseLocation`, `WriteExternalStorage`, `WakeLock`, `ReadPhoneState`, `ReadExternalStorage`.

Regarding the regular behaviour of the HoneyQuest game, none of these dangerous permissions should be requested. Yet they are essential for Alphonso software to run: resetting either the audio recording or the location permissions for the app with Eavesdroid "Modify Permissions" tool makes the application crash and request the permissions again.

### 3.3.3 App behaviour

A thorough static analysis of the application, backed by Eavesdroid tools, enables to sketch Alphonso's behaviour when being launched.

1. Service setup: The application runs Alphonso's service at launch. This service supervises the next behaviours.
2. Audio recording: The application records audio with the device microphone. The behaviour of the recording varies on various conditions in order to enhance recognition performance.
3. Audio treatment: The recorded audio is treated using Alphonso's Automatic Content Recognition (ACR). The application interacts with the server for audio treatment and/or audio recognition, in order to find pertinent information about what the user is watching on TV.
4. Advert tracking: The application displays targeted advertisements according to the identified audio. The intents sent by the application can also trigger advertising activities from other applications embedding Alphonso.

When treating recorded audio samples, Alphonso's Automatic Content Recognition recognizes the brand and product of the advertisement, the channel and the network of the program and the episode and season of the show (see Figure 3.3). This data enables the advertisers to display advertisements related to the user's TV program or commercial.

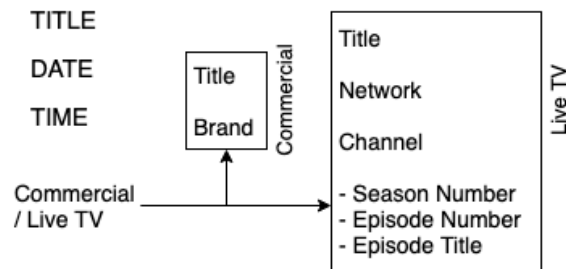


Figure 3.3: Data retrieved about the captured Audio

## 3.4 Audio Recording

### 3.4.1 Audio Recording triggering

As seen in 3.2, microphone resources are held by the application, even in the background and when the phone is locked. Eavesdroid's "String Search" tool gathers all the API calls for audio recording libraries in the code. The library responsible for recording is the Android library `AudioRecord`. We can see where and by which classes these libraries are being called (`tv/alphonso/audiorecorderservice/AudioRecorder` and `alphonso/audiocaptureservice/RecorderThread`).

The study of these classes confirms that no user input is required beforehand. The audio recording is not triggered by any user. Therefore, audio recording is noticeably only used by Alphonso to continuously eavesdrop its users.

### 3.4.2 Recording parameters

The microphone is permanently held by the application, even when it is in the background. However, in order to maximize the probability of recording useful audios and minimize battery and

CPU usage, applications embedding Alphonso Software only record audio under certain conditions.

The class `utils.PreferencesManager`, from the Alphonso package, stores the default parameters for audio recording. These parameters are fetched by other classes including the classes responsible for audio recording. By default, it appears that the device captures audio samples every 10 seconds for a duration of 5 seconds. Thus, by default the device is recording a third of the time. This audio interval is increased to 12 seconds if the user is detected as watching Live TV. Following a deeper static analysis of the application, this interval between two audio captures also varies with the different recording mods.

About a third of the surrounding audio is recorded by Alphonso Software, but the recording is triggered under certain conditions. Many of the classes of the package `audiocaptureservice` are implemented to modify the behaviour of the application under certain circumstances. Different parameters can be modified when the application is run: Capture Duration, Count, Sleep Interval, Sleep time. These parameters are modified by the main class `AudioCaptureService` of the recording service and enables the application to modify the interval, the duration, the number of recordings being captured. These modifications are triggered depending on the results from previously treated audio samples (eg. the user is watching a Commercial instead of Live TV).

In order to increase the probability of a user watching TV, the Audio Capture service modifies at runtime the behaviour of the recordings when the users are most likely to watch TV.

First, the application is most likely to record audio during TV prime times in order to increase the share of TV recordings. These Prime Times are defined by default as being in the morning between 6 AM and 9 AM and in the evening between 7 PM and 10 PM.

Then, the application uses the device's movement and the user's presence to ensure that the user might be watching TV. The `utils` package of Alphonso Software uses the `Access_Fine_Location` permission (previously granted by the user) to obtain the rotation and the speed of the device. Alphonso uses these two pieces of information to know whether the user is static and looking at their phone screen.

Finally, different signals are hooked by the application: `USER_PRESENT`, `SCREEN_OFF`, `HEADSET_PLUG`, `USB_STATE`. It once again shows the capacity of the app to know when the user is on their phone (often during Ads when watching TV), so that it can enhance the chances of capturing useful recordings.

In summary, Alphonso uses a set of conditions to maximize the efficiency of the audio recordings. An important amount of time is being recorded by the applications, conscientiously selected to obtain valuable data about their users.

### 3.5 Audio Treatment

Alphonso aims to capture audio samples of the user's surroundings in order to recognise what the user is watching and select related advertisements. Therefore, it needs to retrieve meaningful data about the audio samples. The analysis of Alphonso Software exposes the use of Automated Content Recognition (ACR) identification technology.

With ACR, the recognition of the audio is done by generating its condensed digital summary, called an acoustic fingerprint, and comparing it to known fingerprints. The fingerprint of an audio sample is generated with a fingerprint algorithm[12] that outlines relevant characteristics of the audio. For example, a fingerprint algorithm can retrieve peaks in the audio spectrum and use them as a fingerprint. The advantage of these algorithms is that they are robust against audio compression and sound degradation (audio quality, background noise, people talking over the sound...). Instead of using the entire audio clips and comparing them to known audio samples, it is less costly and more efficient to use fingerprints.

### 3.5.1 Automated Content Recognition mods

Alphonso embeds four different methods of Audio Content Recognition in their applications: Local, Split, Dual and Server methods. Each of these methods computes the fingerprints and interacts with Alphonso's servers differently.

- **Local ACR:** This ACR mod computes the audio samples' fingerprints locally in order to identify the different features of the audio. These fingerprints are saved in a database in order to be sent later to Alphonso's servers. Contrary to the three other methods of computation, this ACR mod does not communicate with the servers. This method is most probably used when the device cannot connect to the Internet.
- **Split ACR:** This ACR mod inherits from the Local ACR method and computes the fingerprint of the audio locally. It also directly uploads the audio fingerprint on Alphonso's servers. This mod is set by default in the configuration file of the application.
- **Dual ACR:** This ACR mod inherits from the Split ACR mod and computes the fingerprint of the audio locally and then sends it to the servers. It also performs a lookup among already computed fingerprints in order to find similar audio samples that have already been treated and recognized.
- **Server ACR:** This ACR mod sends the entire audio clip to the servers. The recognition of this audio is done remotely on Alphonso's servers, which send its results directly back.

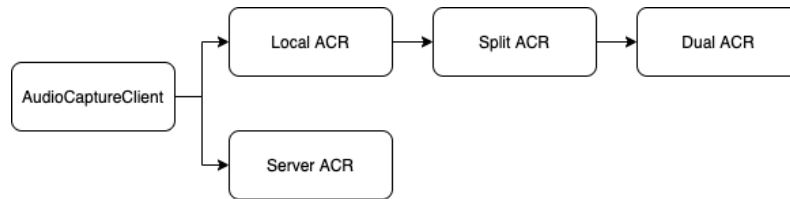


Figure 3.4: ACR mods

### 3.5.2 ACR additional properties

Fingerprint computation is performed locally using native libraries present in the APK. The libraries employed by the application can be retrieved with the "Native Library" tool of Eavesdroid. The retrieved native files are designed for "armeabi" binary interfaces in order to run under ARM Android devices. Three native files contain the HoneyQuest game files compiled to run under Unity with the Java Native Interface. Besides, the file `libacr.so` contains the resources of the ACR library. This library is called by Alphonso to compute fingerprints for the captured audio samples and lookup for already existing fingerprints. The following inbuilt functions are called to perform the ACR computations: `acrDestroy`, `acrFingerprintOctet`, `acrLookupFingerprint`, `acrLookupOctet`, `acrTokenNew`, `acrTokenDestroy`, `dbio_read_data` base. These functions compute the fingerprint of the audio clips few bytes per few bytes and send the fingerprints back to Alphonso's service. These fingerprints can then be compared part by part in order to lookup for already recorded audio samples. All in all, this native library is the core of the Automated Content Recognition fingerprint computation, later used to recognize the audio samples remotely or locally.

The recorded audio samples and the audio fingerprints are processed differently with the four computations methods.

First, local ACR saves the audio records locally, under the name *token.audio.raw*. In order to look for these saved audio files, Eavesdroid uses ADB shell to look inside the applications' local storage. HoneyQuest and many other Alphonso Software apps are not debuggable. Eavesdroid enables to repack the applications as debuggable and sign it to access the applications' local storage. As the service fails to boot up, no sign of these files can be observed dynamically.

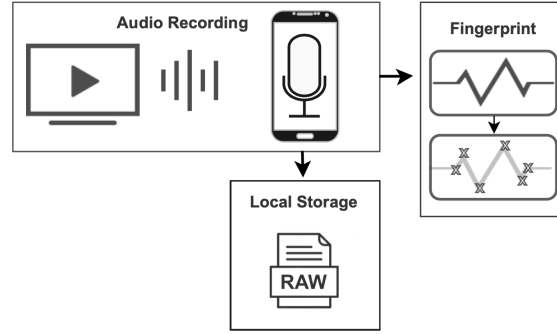


Figure 3.5: Local ACR

Then, Split and Dual ACR mods save the fingerprint and its related data in a local database called **History**. This database is used during the lookup process to find a match between a newly computed fingerprint and an already computed one. Like so, the application can directly know if an ad has been viewed twice and can thus display once again the same advertisement on the user's phone. The Eavesdroid "Export Database" tool makes a backup of the package and then extracts the databases. Despite being empty, the database's fields refer to the pieces of information sent back by the server about the audio fingerprints (advertisement, Live TV, episode number, name ...).

Similarly, another database called **OfflineUploadDB** is used in the server ACR mod to store the audio samples locally and send them later to the servers. This database presents a Binary Large Object field, usually containing images, audio or other multimedia objects.

Finally, Split and Dual ACR mods send the fingerprints to Alphonso's servers. In the case of server ACR, the entire audio sample is sent. The analysis of the network components of the applications is developed in the following part.

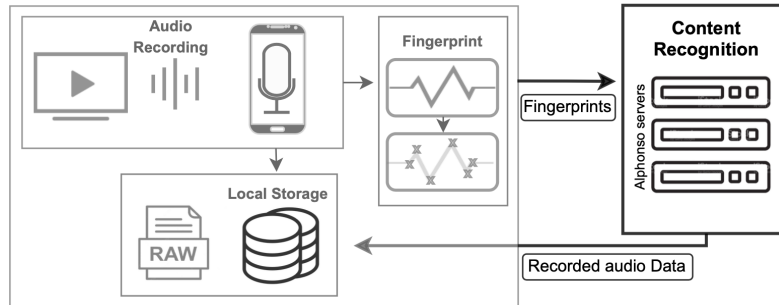


Figure 3.6: Split ACR

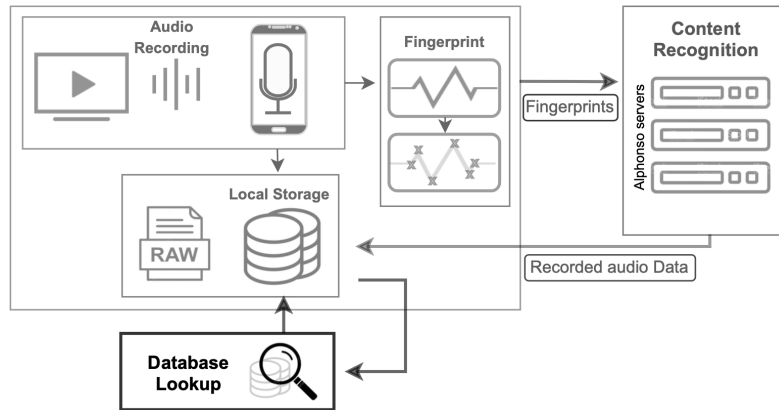


Figure 3.7: Dual ACR

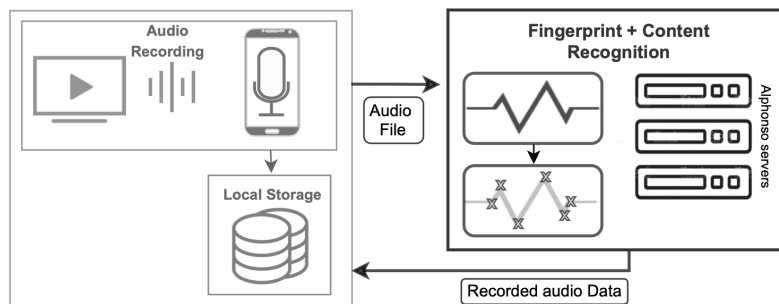


Figure 3.8: Server ACR

### 3.5.3 Network Analysis

The previous analysis reveals that the application sends either fingerprints or entire audio samples to Alphonso's servers. In case of server ACR, the servers compute the fingerprint of the audio samples. Then, in either case, the server compares the fingerprints to their databases to match the audio. The servers then send back the result to the client.

Eavesdroid "String Search" tool statically gathers information about the different network components used by the application. Among others, the tool displays the domain names of the servers contacted by the application: `http://api.alphonso.tv` and `http://prov.alphonso.tv`. Looking at the classes calling these strings, it is possible to obtain more information about these servers' ports and subdomains, as well as the credentials used by the clients to access the servers.

- Ports: 4430 for `http://api.alphonso.tv` and 4000 for `http://prov.alphonso.tv`.
- Subdomains:
  - `/user/lookups`: This subdomain is contacted by the application to initialize the user's Alphonso profile and ID. These identifiers are at the foundation of Alphonso's user profiling. Each user on a device has a single Alphonso ID used for every application. This identifier links the users with the advertisements they have watched.
  - `/user`: This subdomain is contacted to upload information about the user device (id, name and maker of the device, version of the OS, application name and version...).

Despite not being stated by Alphonso, audio information is not the only piece of information exchanged by the application.

- `/user/location`: The application sends updates about the location of the user. This unstated behaviour from Alphonso is a major privacy issue.
  - `/audio/fingerprint`: The application sends the computed fingerprints on this subdomain. The fingerprint is identified on the server in order to recognize the audio sample.
  - `/audio/buffer`: The application sends the Server ACR's raw audio on this subdomain. The fingerprint of the raw audio is computed directly on the server.
  - `/user/audio_clip_data`: The application sends additional content about the audio clips, such as the location of the device and the time when the audio sample was recorded.
- **Credentials**: The client authenticates the content it sends to the server using two credentials: an API Key (`api_key`) and an Alphonso ID (`alp_uid`).

Using network analysing tools such as BurpSuite and Eavesdroid "Logcat", it is possible to analyse HTTP requests sent to the server by the application. Yet the connection to the server fails as they have been put out of service, restricting the use of such tools for Alphonso Software applications.

Along with the previously discussed infringement to users' privacy, some security concerns can be raised about HTTP protocols used by the application. All of the data aggregated about the user is conveyed through HTTP without encryption. Taking into account the privacy aspect of this data, applications embedding Alphonso Software demonstrate a serious security flaw that could result in the leak of users' personal information (location, device ID and name, TV viewership etc.)

### 3.6 Cross-Device Tracking

The audio recorded by the application is identified in such a way that advertisers can be informed about whether the user is watching a Live TV program (including further data on the movie or episode) or a commercial (including the commercial's name and brand). As discussed previously, the applications also share information about the user's location, device and IP address with Alphonso. This private information will probably be sold, on top of the TV data, to advertisers to improve the efficiency of cross device tracking.

This tracking enables advertisers to know when, where and what advertisements have been seen by the user. It also enables to link the different devices and understand the user's habits (location, TV presence, second-screening habits...). Cross device tracking, in the case of Alphonso, helps build a complete data profile about the users in order to serve extremely targeted advertisements across multiple devices.

Advertisers' packages are directly implemented in the applications. In every application embedding Alphonso, these advertisers are **inMobi**, **Jirbo**, **MoPub** and **Vungle**. As an aside, **inMobi** and **MoPub** signatures are recognized by many antivirus scanners as malicious. Nonetheless, with the given data about the user's TV preferences and viewership, these advertisers can display targeted advertisements directly on the user's device.

### 3.7 Limits of the Eavesdroid tool

Eavesdroid tools are mainly intended for assisting a malware analyst in the analysis of an application using audio recording features and suspicious of eavesdropping its users. Most of the analysis of these applications remains to be done by manually reverse engineering the application's behaviour and corroborating these observations with dynamic analysis. Also, many of these tools can be used to support the analysis of general applications not necessarily embedding audio recording features. The tools can lack automation about audio recording features when assessing eavesdropping applications, in order to allow the analysis of more common applications.

Eavesdroid's set of tools cannot be conveniently used for every application. The utility of some tools (String Search, API tracking, APK architecture) is limited when analysing code that has been stripped or obfuscated. Applications can also be outdated and not operate properly anymore, as servers can stop interacting with the application and features can become obsolete. The dynamic evaluation of the components of the application is less complete.

Some tools useful for the dynamic and static analysis of Android applications could also be added to Eavesdroid. This includes the addition of an API hooking/tracing feature. This feature would enable to track the libraries called by an application, either dynamically by intercepting API calls or statically by building the Control Flow Graph of the code. To do so, Eavesdroid could implement **Hooker**, **Xposed** or **JNITrace** for dynamic API hooking, and **Soot** or **Androguard** for static Control Flow Graph. Eavesdroid could also benefit the addition of an interaction tool with the HTTP flow of the application. This could be done by implementing scripting methods to the current proxy solution or using other traffic interception tools such as **BurpSuite**.



# Conclusion

In this paper, we used the Eavesdroid tool to confirm the different eavesdropping and cross device tracking behaviours of applications implementing Alphonso. We showed that these applications continuously record users even in the background. This recorded audio is treated either locally or remotely in order to distinguish what the users watch on TV. We exposed that some features embedded in these applications do not follow Alphonso's statements, such as the leakage of additional private data and the impossibility to withdraw from the recordings. This data can be used by Alphonso to create profiles of their users and serve targeted advertisements.

All in all, applications easy to download can stealthily detect what TV program the users are watching and then display targeted advertisements on all of their devices. However, it is unclear what data is stored on these applications' servers and what is carried out with it.

Eavesdroid remains limited in term of its usage and its functionalities. The assessment of deactivated applications such as Alphonso is mostly restricted to its static analysis and to the local components of the technology. The remote behaviour of Alphonso's servers cannot be settled.

Indeed these types of eavesdropping software have no intent to halt their detection practices to TV shows and TV commercials. Every multimedia content, such as social media or YouTube videos, could be identified. Also, identifying conversations would enable to determine with an extreme precision the profile of the users. These eavesdropping applications could very much be one of the most powerful commercial tools developed to date.

# References

- [1] Federal Trade Commission staff report. *Cross Device Tracking*. January 2017  
[https://www.ftc.gov/system/files/documents/reports/cross-device-tracking-federal-trade-commission-staff-report-january-2017/ftc\\_cross-device\\_tracking\\_report\\_1-23-17.pdf](https://www.ftc.gov/system/files/documents/reports/cross-device-tracking-federal-trade-commission-staff-report-january-2017/ftc_cross-device_tracking_report_1-23-17.pdf)
- [2] Vasilios Mavroudis et al. *Talking behind your Back: Attacks and Countermeasures of Ultrasonic cross-device Tracking*. Black Hat Europe 2016.
- [3] N. Matyugin, J. Szefer and S. Katzenbeisser *Zero-Permission Acoustic Cross-Device Tracking*  
<https://caslab.csl.yale.edu/publications/matyugin2018zeropermission.pdf>
- [4] Sebastian Zimmeck et al. *A Privacy Analysis of Cross-device Tracking* 26th USENIX Security Symposium, 2017.  
<https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-zimmeck.pdf>
- [5] Kevin Zhijie Chen et al. *Contextual Policy Enforcement in Android Applications with Permission Event Graphs*  
<http://people.csail.mit.edu/rinard/paper/ndss13.pdf>
- [6] Elleen Pan et al. *Panoptispy: Characterizing Audio and Video Exfiltration from Android Applications*.  
[https://www.cityfreqs.com.au/papers/android\\_audio\\_exfiltration.pdf](https://www.cityfreqs.com.au/papers/android_audio_exfiltration.pdf)
- [7] Yanick Fratantonio et al. *Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop*  
[https://cs.uwaterloo.ca/~yaafer/teaching/papers/ieee\\_sp17\\_cloak\\_and\\_dagger\\_final.pdf](https://cs.uwaterloo.ca/~yaafer/teaching/papers/ieee_sp17_cloak_and_dagger_final.pdf)
- [8] Suman Nath *MAdScope: Characterizing Mobile In-App Targeted Ads*  
<https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/main-4.pdf>
- [9] Android Debug Bridge commands, Android Developers.  
<https://developer.android.com/studio/command-line/adb>
- [10] Sapna Maheshwari. *That Game on Your Phone May Be Tracking What You're Watching on TV*. The New York Times (Dec. 29, 2017)
- [11] HoneyQuest application in the ApkPure store.  
<https://apkpure.com/honey-quest/com.dumadugames.honeyquest/versions>
- [12] Pedro Cano et al. *Audio Fingerprinting: Concepts And Applications*  
[https://www.researchgate.net/publication/225574479\\_Audio\\_Fingerprinting\\_Concepts\\_And\\_Applications](https://www.researchgate.net/publication/225574479_Audio_Fingerprinting_Concepts_And_Applications)