




# Malware in Motion

Robert Choudhury<sup>1</sup>, Zhiyuan Luo<sup>1</sup> and Khuong An Nguyen<sup>2</sup>

<sup>1</sup>Royal Holloway University of London, Surrey, TW20 0EX, United Kingdom

<sup>2</sup>University of Brighton, East Sussex BN2 4GJ, United Kingdom

Robert.Choudhury.2015@live.rhul.ac.uk, Zhiyuan.Luo@rhul.ac.uk, K.A.Nguyen@brighton.ac.uk

Keywords: Dynamic Analysis, Mobile Security, Reverse Turing.

Abstract: Malicious software (malware) is designed to circumvent the security policy of the host device. Smartphones represent an attractive target to malware authors as they are often a rich source of sensitive information. Attractive targets for attackers are sensors (such as cameras or microphones) which allow observation of the victims in real time. To counteract this threat, there has been a tightening of privileges on mobile devices with respect to sensors, with app developers being required to declare which sensors they need access to, as well as the users needing to give consent. We demonstrate by conducting a survey of publicly accessible malware analysis platforms that there are still implementations of sensors which are trivial to detect without exposing the malicious intent of a program. We also show how that, despite changes to the permission model, it is still possible to fingerprint an analysis environment even when the analysis is carried using a physical device with the novel use of Android's Activity Recognition API.

## 1 INTRODUCTION


Within Security the practice of misinformation is utilised by both malware authors (who try to make their creations' purpose appear benign) and by security analysts who wish to observe the behaviour of the malware and therefore try to make their analysis environment transparent to the malware. The technique of hiding the intention of malware is known as evasion and the technique of making an analysis environment such as a sandbox appear to be a legitimate target is known as hardening (Ferrand, 2015).


Because the process of finding exploits to elevate privilege on the target machine or inventing a means of propagation is an expense, malware authors will seek to hide these techniques by evading analysis. Evasion in malware can be achieved by using methods such as detecting the threat of analysis and then changing behaviour to benign or by stalling malicious execution until the malware is no longer under analysis.


Another motivation of the malware author is that they may wish to avoid detection so as not to alert a high value target who may develop countermeasures.

Therefore, in order to maximise the impact of the malware created it is important that it evades analysis until it has achieved its objective. For example, in the case of Ransomware this would be the successful encryption of files and the delivery of the ransom message to the victim. Conversely, the hardening strategy of the analysis environment involves anticipating the request of the executing malware and sending an answer that the malware "expects" to see when running on a target machine. If successful, the malware does not hide its malicious routines thereby allowing analysts to observe unaltered behaviour. This leads to a race where both malware authors and analysts require intelligence so they can understand the nature of the strategies the other party will employ. For example, there has been research successfully conducted using submitted apps to discover the nature of 'Bouncer' which is a sandbox designed to prevent malicious apps appearing on the Google Play Store.

**Information leaks using low powered sensors** It is possible for a malicious actor to circumvent the security policy of a phone and gain access to sensitive information using low powered sensors such as accelerometers. Nguyen et al demonstrated this by using magnetometer and accelerometer traces to track the movement of a target's smartphone (Nguyen et al., 2019). In the paper "Sensor Calibration Fin-

<sup>a</sup> <https://orcid.org/0000-0003-0974-7920>

<sup>b</sup> <https://orcid.org/0000-0002-3336-3751>

<sup>c</sup> <https://orcid.org/0000-0001-6198-9295>

gerprinting for Smartphones” it was demonstrated that JavaScript and a locally installed app could with only 100 sensor samples infer the device factory calibration and allow fingerprinting of a specific device across multiple platforms. This fingerprint is immutable by the device’s end user (Zhang et al., 2019).

## 1.1 Paper’s Contributions

This work makes the following contributions:

- An app was created and customised to each platform in order to establish the state of sensor implementations across publicly accessible malware analysis platforms
- The network traffic that forms the responses is analysed provided a novel way distinguish properties that can aid an attacker in fingerprinting or evading analysis.
- Sensor readings are reviewed and are rated from the perspective of an attacker looking to evade analysis.
- A novel use of activity recognition is proposed to produce a Reverse Turing test that can be used to identify a lack of human-generated motion on a physical phone which can be used to fingerprint analysis.

## 1.2 Structure of the Paper

The rest of this paper is structured as follows:

- In Section 2 we discuss the background of this work and why it is important as well as highlight the difference between this work and other related work in this field.
- In Section 3 we describe how we obtained the data required to perform our analysis
- In Section 4 we carry out analysis of the obtained sensor values
- Section 5 discusses the values returned by the app and demonstrate how a reverse Turing test could be implemented.
- Section 6 concludes our work and discusses future work

## 2 BACKGROUND AND RELATED WORK

### 2.1 What is a Sandbox?

With large amounts of malware being generated every day, efficient ways to identify malware and classify it correctly are required to keep up with demand. The problem is that with limited resources, security analysts must prioritise how best to minimise the risk to the systems they defend whilst maximising the detection of malware. This is typically achieved through the automation of analysis using a dynamic analysis tool known as a sandbox. Sandboxes are isolated environments where an unclassified sample program can be executed and its behaviour observed and then identified as either malicious or non-malicious. Sandboxes are used by organisations and malware analysis companies to provide precise feedback on the behaviour of suspect files such as email attachments. This allows a company to quickly see if the attachment is legitimate or if it needs to be quarantined.

The prime advantage of a sandbox over a traditional antivirus is that it is based on behaviour and can highlight suspicious activities of a sample without the need for a signature that has been generated on previously seen malware. This allows proactive monitoring of incoming files to a protected network rather than just reacting to an infection and also helps protect against zero-day attacks.

A sandbox can also be used in post incident forensics, for example by analysts identifying the impact of an infection. This is because a sandbox can reveal which files have been changed by the sample along with any registry entries modified and new malware files downloaded. Therefore, it can aid in prioritising remedial efforts as well as judging the extent of the realised threat.

Sandboxes like Bouncer execute code and are referred to as dynamic analysis tools. Conversely static analysis is conducted without executing the sample. These are the two principal ways of analysing a malware sample. The problem with conducting just static analysis is that malware authors use techniques to make static analysis difficult such as obfuscating the code or encryption (Bashari Rad et al., 2012; Moser et al., 2007). Encryption is used by malware authors to disguise suspicious code and internal strings that would otherwise give analysts an indication of malicious intent when statically analysed. These strings can contain items such as URLs used by the malware author or IP addresses. By executing the malware, analysts can observe the interactions between the malware and the operating system as well as external re-

sources that can indicate if the nature of the program is benign or hostile.

## 2.2 Related Works

An Empirical study to fingerprint public malware analysis services was conducted in (Botas et al., 2018). The researchers focused on fingerprinting malware analysis platforms that are publicly available which is often a prelude to evasion. This is achieved by sending a sample to each platform and retrieving artefacts such as the version of the operating system, the current username, the mac address and so on. As many of these values are shown to be similar or the same on various analysis platforms, the authors show it is possible to fingerprint analysis environments using these values. The authors proposed a method to help prevent fingerprinting by generating a random value for each of these artefacts which was then fixed and returned to the querying sample. This differs from our work which is focused on the mobile operating system Android and more specifically the returned values from sensors. We extend their work by proposing an attack that would defeat the random artefacts framework if applied to sensor readings produced by mobile devices.

In the paper “Tap Wave Rub” (Shrestha et al., 2015) the researchers produced a Reverse Turing test based around the sensor readings recorded when the user was prompted to perform a sequence of uncommon gestures to ensure that near field communications (NFC) were correctly triggered by the human user and not by malicious software installed on the device. This work has the benefit of being able to detect an attack in real time and not posteriori.

In the paper “Evading Android Runtime Analysis via Sandbox Detection” (Vidas and Christin, 2014), the authors used a number of artefacts including fingerprinting some prominent Android malware analysis systems that are based on Virtualisation. This work is related to ours due to the survey of available sensors. At the time of publishing, only Copperdroid (no longer publicly accessible) handled sensor events and this was limited to just the accelerometer. Our work follows this paper by working with the current state of the art publicly accessible malware analysis platforms and extends this work by investigating how it is possible to fingerprint analysis even if a ‘real’ mobile device is being used to execute the samples. Owusu et al conducted research in the paper “ACCesory using accelerometer to detect the motion of the local device’s loudspeaker”(Owusu et al., 2012). The researchers then used this data to infer what the user was hearing in their loudspeaker. In 2019 TrendMi-

cro detected two apps Batterysaver mobi and Currency Convertor which use a threshold of the accelerometer value as means to detect if the malicious app is under investigation (Sun, 2019).

## 3 Methodology/Design

Two apps were developed for this project. The first was designed to survey the available sensors on publicly accessible malware analysis platforms. It was then modified to collect accelerometer readings. The second app utilised Google Play services to implement a Reverse Turing test to exploit the vulnerability highlighted from the earlier survey of sensor data.

### 3.1 Information Gathering

Accessible Android sandboxes were identified from sources such as research papers and online searches. Initially seventy online platforms were identified and then filtered, firstly to remove those that were not available and secondly, as we are interested in the dynamic sensor values returned, sandboxes that focused solely on static analysis or other file types were discarded.

Figure 1 shows how a customised APK file was developed for each sandbox and delivered through its corresponding web portal or via email. The APK file was unique to each targeted platform to allow us to determine the source of any responses (even multiple responses were received simultaneously). Each execution of the file was also uniquely identified enabling us to determine if a platform had executed the file more than once.

#### Workflow of APK data collection

1. Choose a target and customise the APK file
2. Deliver the APK file to the target
3. Depending on the type of target platform, the APK file was then either:
  - (a) Queued and then executed on the platform
  - (b) Forwarded onto third party services in which case multiple responses were received with the same target identifier.
4. If the analysis environment allowed access to the internet, packets were sent back to a server under our control. The IP address, identifier and sensor values were logged along with a session ID to see if the same platform was executing the sample in parallel.
5. The data was parsed and analysed.

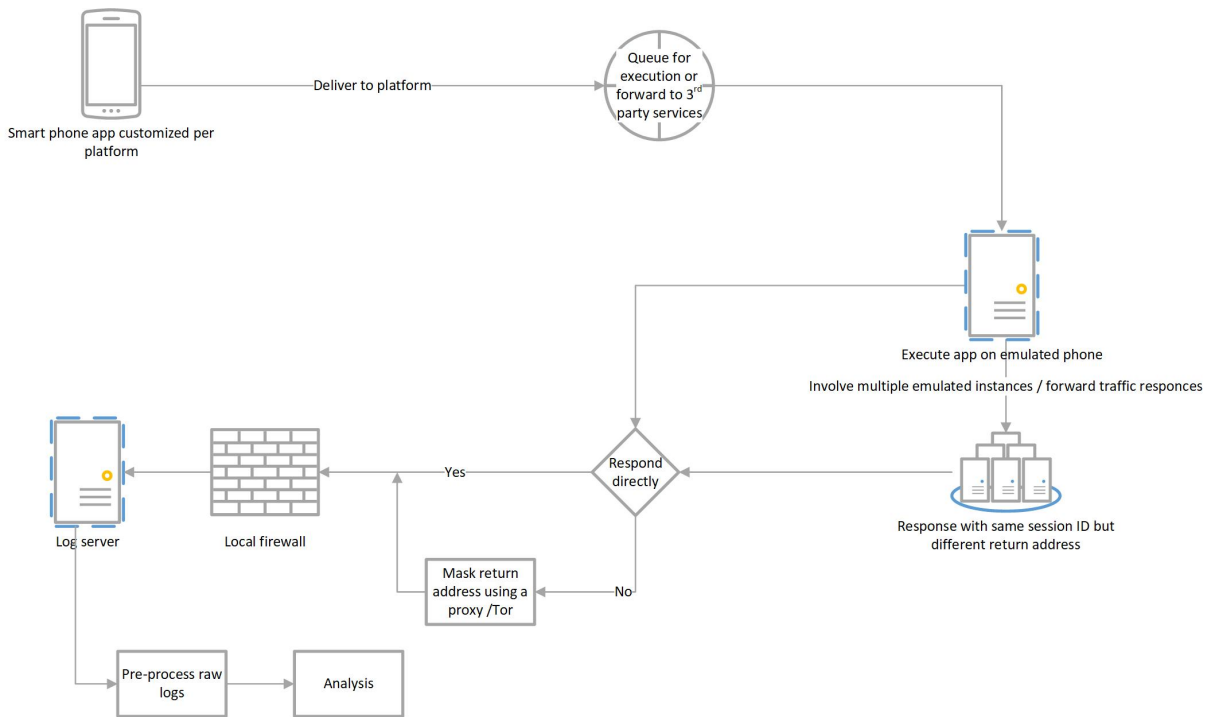


Figure 1: Data collection and analysis process

### 3.2 App Design

The app was designed to query the execution platform in two stages and return the findings to the server. Firstly the array of available sensors was surveyed and checked for the presence of the required accelerometer. The accelerometer values were then extracted for the length of the execution. A two stage process was required because some of the surveyed analysis platforms had not implemented all of the sensors expected from a physical device (some had not implemented any). If we did not check for the presence of a required sensor before creating a listener to access its readings an exception would be generated that, if not handled, would cause the execution to cease. This could be a crude way to prevent analysis on a virtual implementation of Android where all the typical sensors are not implemented.

## 4 RESULTS

In this section we present the data gathered from seventeen publicly accessible app analysis platforms. The app was unique to each platform to allow for attribution of the results and to see if the platforms were executing the app multiple times per delivery.

### 4.1 Analysis of the Network Traffic Received.

We define a ‘valid unique response’ as a host responding with a correctly formatted unique identifier and a list of sensors (including an empty sensor list). Seventy-three unique IP addresses responded to our server once the app was delivered. Most of the responses were from the apps delivered to Virus Total and Hybrid Analysis. The reason for this multiplier effect is that Virus Total and Hybrid Analysis are forwarding received samples to third parties, therefore they are defined as meta services where a single sample can be tested against multiple antivirus products.

#### 4.1.1 Attribution of the Responses

Because of the low number of direct responses we combined the use of a session ID and reverse DNS lookup as well as network tools to find the autonomous system number. This also aided us in determining if the sample was potentially being executed multiple times by different hosts belonging the same vendor.

Table 1 shows that 74.5% (Seventy-three out of ninety-eight responses including duplicates generated by overlap in meta services) of responses originated from the Amazons elastic compute cloud (EC2). The EC2 provides the customer with an ability to allocate

Table 1: Responses from hosts after the app is delivered to each target

Source of response	Number of responses
OVH SSS	3
Trustwave Holdings, Inc.	2
China Mobile	1
Forcepoint Cloud	2
Orange Polska Spolka Akcyjna	6
Amazon.com, Inc	73
M247 Ltd	1
Serbia BroadBand-Srpske Kablovske mreze d.o.o.	2
Trend Micro Incorporated	2
Bitdefender SRL	2
Unknown (Joe Sandbox)	2
The Calyx Institute	1
China Mobile	1

resources dynamically and scale up resources such as virtual instances on demand. This may make it ideal for executing a program multiple times in different environments in order to maximise code coverage. Code coverage is important in the field of malware analysis as greater coverage decreases the chance that a malware author has successfully hidden the malicious intent of their program.

## 4.2 Initial Survey of Available Sensors

Initially we investigated which sensors are implemented on malware analysis platforms. Table 2 shows the sandboxes that responded to queries from our custom app.

We observed three different sensor lists as shown in Table 2. The first being a complete list of emulated values with the name Goldfish which refers to the name of the CPU emulator. The second being being just the accelerometer also including a reference to Goldfish and the third being the Kbd Orientation Sensor.

## 4.3 Accelerometer Values Received

In this section we analyse the values returned by sandboxes for the accelerometer x,y and z axis.

**Why was the accelerometer sensor specifically chosen for this study?** The accelerometer is known as a low power sensor. It does not require human interaction with the smartphone to enable it and its readings are readily available to any Android app. During the initial survey of sensors the accelerometer was the most ubiquitous of the sensors implemented on the analysis platforms, meaning that attacks using this sensor will have the greatest impact. This is because if you register a listener to other sensors such as the gyroscope and they are not present, an exception will be generated which can lead to the program exiting. A program crashing may lead to investigation or stop an attacker from being able to launch their malicious routines on a valid target.

Of the platforms the responded only Sandroid and Joe sandbox returned values from an app that was delivered directly all other responses were from apps delivered to vendors via the meta services virus total and hybrid analysis.

Joe sandbox allows the user to set the properties of the firewall to allow access to the Internet and thus ensured a response whereas Sandroid allowed the traffic to exit the network by default.

### 4.3.1 Timing the Responses

For the sandboxes that returned sensor values we observed that the period of time values were returned for varied depending on the platform.

Sandroid returned accelerometer values for 185 seconds which was the longest period observed. We noticed that trendmicros length of responses varied depending on where the file was originally delivered to with the time being between 30 and 28 seconds from hybrid analysis and 62 seconds when the file was delivered via Virus total.

### 4.3.2 Analysis of the Values Returned by the Malware Analysis Platforms Accelerometer

The accelerometer values were gathered by a separate process and were dependent on the presence of the required accelerometer sensor. These values were stored with an additional session identifier to help differentiate between multiple instances of the same app if they are being executed in parallel from the same network.

Table 3 shows the different sets of static values returned by the accelerometers of the platforms surveyed. Any reoccurring accelerometer value is evidence of a virtual environment and in effect an artefact. In turn this is vulnerable to a malicious app

Table 2: List of available sensors

Sensor list received	Comments
Goldfish 3-axis Accelerometer, Goldfish 3-axis Gyroscope, Goldfish 3-axis Magnetic field sensor, Goldfish Orientation sensor, Goldfish Temperature sensor, Goldfish Proximity sensor, Goldfish Light sensor, Goldfish Pressure sensor, Goldfish Humidity sensor, Goldfish 3-axis Magnetic field sensor (uncalibrated), Game Rotation Vector Sensor, GeoMag Rotation Vector Sensor-Gravity Sensor, Linear Acceleration Sensor, Rotation Vector Sensor, Orientation Sensor	A list from a virtual platform with the low power sensors implemented. It is possible to get sensor values returned such a platform. The term 'Goldfish' refers to the specific virtual hardware implementation of these sensors.
Goldfish 3-axis Accelerometer	A platform with only the accelerometer implemented.
Kbd Orientation Sensor	Virtual device with only a deprecated orientation sensor.

either checking for this precise value or detecting a threshold in the change of accelerometer values.

#### 4.3.3 The Expected Noise Generated from a Physical Phone

Sensors such as the accelerometer produce varying levels of 'noise'. This noise is generated by electrical signals and low amplitude motion that is detected by the highly sensitive accelerometer.

We demonstrate the expected behaviour of a physical device by using our app to collect samples from the accelerometer of a stationary Android smart phone. The phone was orientated resting face upwards on a flat surface.

Figures 2, 3 and 4 show a normal distribution of accelerometer value on each axis. The mean value for each sample set is subtracted in order to filter the data of constant values such as the components of gravity and leave the noise. This method has previously used in the preprocessing of datasets where human activity recognition is to be performed (Anguita et al., 2013).

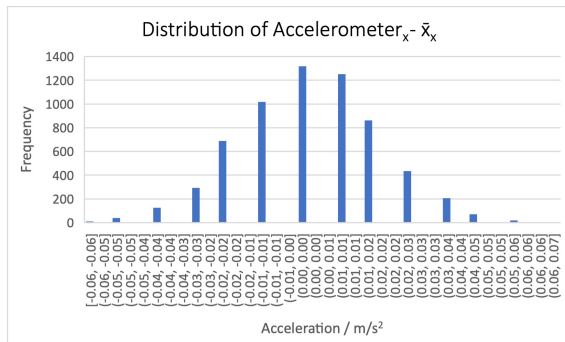


Figure 2: X axis taken from a real phones' accelerometer

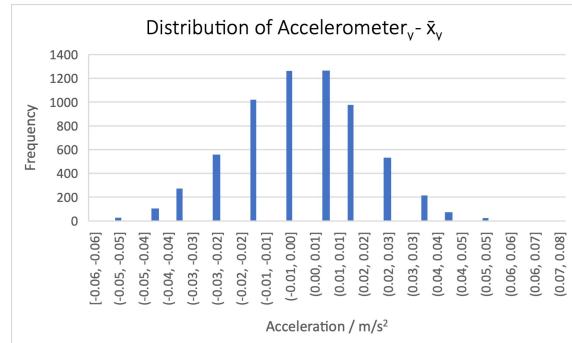


Figure 3: Y axis taken from a real phones' accelerometer

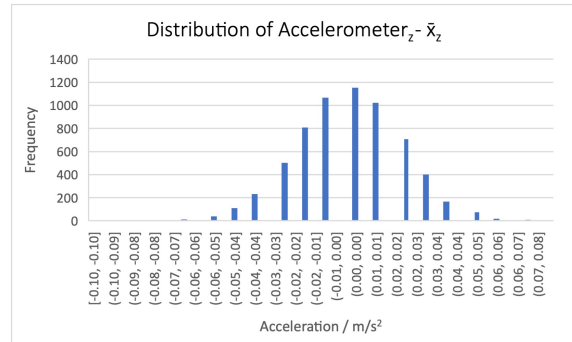


Figure 4: Z axis taken from a real phones' accelerometer

## 5 DISCUSSION

In this section any received data is further analysed, and possible attacks are identified leading to the formation of the app that conducts the Reverse Turing test.

**Relative quality of the sensor list and values returned.** In Table 4 the increasing quality of the accelerometer values and lists received is examined to

Table 3: Values received from accelerometers

Values received (X,Y,Z)/ $m/s^2$	Comments
0.0,9.776,0.813 Constant	Default values for an emulator with the phone standing on its bottom edge but leaning slight forward on its x axis (4.4 degrees).
0.0,0.0,0.0 Constant	No sensor values being generated and gravity has not been included.
0.0,9.81,0.0 Constant	Artificially set to have the accepted three decimal place value for gravity. The phone is orientated as standing on its bottom edge.

show that it represents the difficulty that a malware author will have in detecting if their app is under analysis. Quality in this table represents the relative effort that that it will take for malware to recognise that the platform is virtual.

For instance the Lowest is that there are no sensors implemented because simply attempting to access the sensor will lead to the program to stop and therefore act differently in the emulated environment.

**Running the samples on a physical device** We did not observe a sensor list and therefore sensor values that corresponded with a physical device during the survey. It is however the best solution currently and noise will be generated and the sensors will all be present. However such a device will need to be stationed for malware deployment and this lack of 'human generated motion' maybe detected via a simple threshold test or a more sophisticated attack as discussed in Section 5.1.

## 5.1 Using a Reverse Turing Test to Detect Analysis

An app was designed to demonstrate how human activity recognition could be used against publicly accessible Android analysis environments. Due to the lack of dynamic accelerometer values returned in our survey of sensor values, we implemented a local test this allowed us to increase the difficulty of the tests beyond the static sensor values encountered from the survey.

To achieve this the activity recognition app was run on a physical device as this is the most difficult scenario to conduct the reverse Turing test because due to noise and vibration the sensor values will be dynamic.

### 5.1.1 Implementation

Google Play services provide the ActivityRecognitionApi to allow an app to recognise what a user is doing. This allows an app to behave dynamically

based on human behaviour such as "In Vehicle", "On Bicycle", "Walking", "Running", "Tilting", "Still" and "Unknown". The activities Walking and Running were used by the app to indicate that a human user is present (and therefore passes the Reverse Turing test). Activity recognition has a small foot print in the manifest file only requiring one entry and is a completely legitimate API for apps that are used for maps or exercise giving an attacker the ability to hide the app amongst legitimate apps.

Diagram 5 shows how we used Google Play services to conduct a Reverse Turing test.

1. The app is launched on the chosen testing platform
2. Sensor values are gathered and delivered to the model
3. The group of sensor values is compared to modelled activities and a corresponding confidence value is returned.
4. If the value exceeds the threshold of a target human activity such as walking then the reverse turing test else continue to monitor.

In our test environments the ActivityRecognitionApi is used to take sensor values from the phone and return confidence value based on the presumed activity which would be displayed on the screen and a report was sent to a server under our control where it could be analysed. This forms the basis of a threshold which unless exceeded the app will conclude that the Android environment is not being used and the reverse turing test is failed.

### 5.1.2 Preliminary Results of Activity Recognition App

As previously mentioned the ActivityRecognitionApi returns confidence values based on the presumed activity as an array. We found that in this setting the app would detect that the phone was "Still" on a real phone with 100 percent accuracy. The app would return "Unknown" if random motion such as shaking

Table 4: *Relative quality of accelerometer values returned from survey*

Quality	Implementation	Discussion
Lowest	No Sensors implemented	Accessing sensors may crash the app
Low	Virtual sensors returned	A sensor list will return however no values will be returned when called for.
Low	Any readings where the accelerometer values remain static	Any real world sensor will have noise even when still, therefore looking for a change in the rate of acceleration will work against any of these values.
Low	Virtual sensors are returned with improbable values	Accelerometer values with 0.0 in all three axes are impossible due to the constant impact of the earth's gravitational pull - the object would need to be in a perfectly steady freefall
Medium (emulator with sensors enabled)	Virtual sensors / default values returned.	The sensor list may contain the name of the emulation platform (such as goldfish), the sensor values maybe static.
High	Virtual sensors returned and values provided from a static dataset. (This behaviour was not observed in the survey)	Potentially this approach can overcome a simple threshold test provided the recordings include an activity that generates the correct type of motion within the analysis time. If these values are observed, they can be used to as a fingerprint for the presence of analysis.
High	Emulator with USB passthrough	A full list of sensors will be returned, and the values will vary according to any noise generated by the physical device. This can also be expanded to other sensors. This will lack the normal dynamic range of a device in motion and could be bypassed with a threshold. A device at rest (such as a mobile device connected via a USB cable) can be detected using activity recognition.
Proposed model	Emulator with return values to low power sensors replaced with a probabilistic model.	Sensor names and sensor values can be adjusted and distributed at scale. This has the potential to defeat all tests including the Reverse Turing test provided the correct activities are modelled in the data.

the phone occurred. This would be sufficient to exclude all of the previously surveyed responses as well as the typical use of a hardware device because hardware analysis devices are often still and connected by cable to the server responsible for uploading and communicating with the app that has been installed on the device. Even random motion will not trigger the walking or running state with a high degree of confidence which forms the basis of a threshold measure by which malicious routines such as Anubis can be conditionally loaded and then executed.

### 5.1.3 Countering Evasion by Activity Recognition

The following methods are proposed to prevent sandbox fingerprinting by activity recognition: Malware analysts' sandboxes would have to replace local sen-

sors values (which are read only by default) with realistic values that mirror the activities someone is likely to be doing such as walking or running. This could be achieved by using a package such as Frida to exchange the return values for the relevant sensors. Alternatively, static analysis methods could be used to change the flow of execution, but as discussed at the beginning of the paper there are millions of new malwares instances each year which means that having a solution that works without manual intervention is a necessity to keep up with demand otherwise there would need to be a choice between delaying publication of apps or publishing apps where their behaviour has not been examined. One special exception is with respect to Google Bouncer; if it is possible for Google Play services to detect that an app is being executed on Bouncer then the service can be configured to send confidence values that allow the apps behaviour to be



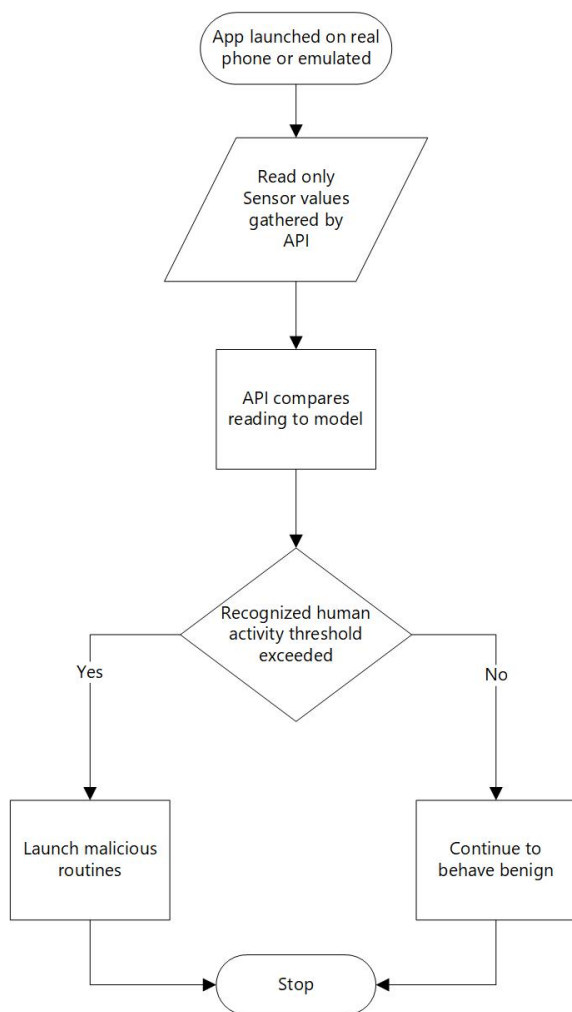


Figure 5: Implementation of a Reverse Turing test using activity recognition

explored.

## 6 CONCLUSION

High numbers of malicious software are being produced each year meaning that there is a need for automated analysis to meet the demand. In order to counteract these automations, malicious software authors seek to evade public analysis by looking for artefacts.

In this paper we conducted a study of the sensors available from automated Android analysis platforms. We started by developing an Android app that was customised to each target to allow us to correctly attribute the data received from each sandbox.

Our first observation was that very few sandboxes out of the original survey responded directly. This is either because the platform did not support the app or

that the traffic from the app was not allowed to exit the sandboxes network.

We found that only three of the surveyed sandboxes responded in the correctly formatted manner. However the server still received responses from seventy - seven hosts.

By analysing the traffic received using a session ID we were able to see that some platforms would execute the sample more than once and return different session IDs for one submission. Other sandboxes would return data from multiple IP addresses with the same Session ID suggesting that this is either the entity rerunning the execution from beyond the point where the session ID is determined with different input in an attempt to gain more code coverage. The alternative is that the traffic is being sent out is being relayed and repeated in order to mask the origin of the traffic.

The antivirus company Bit defender did not respond directly but when a sample was submitted through a third party meta service the responses were obtained. This suggests that samples submitted directly are treated differently in this instance.

Analysis of the sensor implementations on publicly available sandboxes showed that the accelerometer was the most ubiquitous of the available sensors and thus formed the basis of the remaining research. We found that all analysis platforms that returned sensor lists to our server included a clear indication that they were virtual and thus are trivial to detect. Other indicators were the presence of just the accelerometer or no sensors in the list at all.

On the platforms that returned accelerometer values in their x, y, z dimensional components, all responses were static and in this paper we modelled the threat of detection by rating these responses. The worst being all 0s returned which in the presence of gravity and noise from the device itself is impossible. The best current solution is to use a physical device to obtain real time dynamic values. Due to a lack of dynamic values observed in the sensor values survey we captured sensor reading from a local Android phone to demonstrate that (Figures 2, 3, 4) an Android device face up but stationary will still produce a detectable accelerometer values. We also noted that the noise produced was in the pattern of a normal distribution of changes to the accelerometer values across its axis. Because of this difference between a real phone and the static values found on the survey a malware author is able to treat the sensor values as a static artefact much like a reoccurring MAC (Media access control) address in VMWare.

As a future work we will aim to implement the suggested system to increase transparency of publicly

accessible malware analysis platforms by replacing the locally sourced sensor values. We will also aim to look at implementing our own model of activity recognition as the basis of a Reverse Turing test and compare it to Google Play.

447–458, New York, NY, USA. Association for Computing Machinery.

Zhang, J., Beresford, A. R., and Sheret, I. (2019). SEN-SORID: Sensor Calibration Fingerprinting for Smartphones. *2019 IEEE Symposium on Security and Privacy (SP)*, 00:638–655.

## REFERENCES

- Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J. L., et al. (2013). A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3.
- Bashari Rad, B., Masrom, M., and Ibrahim, S. (2012). Camouflage in malware: From encryption to metamorphism. *International Journal of Computer Science And Network Security (IJCSNS)*, 12:74–83.
- Botas, Á., Rodríguez, R. J., Matellán, V., and García, J. F. (2018). Empirical study to fingerprint public malware analysis services. In Pérez García, H., Alfonso-Cendón, J., Sánchez González, L., Quintián, H., and Corchado, E., editors, *International Joint Conference SOCO'17-CISIS'17-ICEUTE'17 León, Spain, September 6–8, 2017, Proceeding*, pages 589–599, Cham. Springer International Publishing.
- Ferrand, O. (2015). How to detect the cuckoo sandbox and to strengthen it? *Journal of Computer Virology and Hacking Techniques*, 11.
- Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE.
- Nguyen, K. A., Akram, R. N., Markantonakis, K., Luo, Z., and Watkins, C. (2019). Location Tracking Using Smartphone Accelerometer and Magnetometer Traces. *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–9.
- Owusu, E., Han, J., Das, S., Perrig, A., and Zhang, J. (2012). ACCessory: password inference using accelerometers on smartphones. *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications - HotMobile '12*, page 9.
- Shrestha, B., Ma, D., Zhu, Y., Li, H., and Saxena, N. (2015). Tap-wave-rub: Lightweight human interaction approach to curb emerging smartphone malware. *IEEE Transactions on Information Forensics and Security*, 10(11):2270–2283.
- Sun, K. (2019). Google Play Apps Drop Anubis, Use Motion-based Evasion. Example of malware authors using an accelerometer to detect Googles bouncer and get their app onto the legitimate Google play store. The apps were called BatterySaverMobi and Currency Convertor.
- Vidas, T. and Christin, N. (2014). Evading android runtime analysis via sandbox detection. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14*, page