Study of Spycam Behavior And Detection Through Analyzing Its Effect on Mobile Device Parameters

Md. Touhiduzzaman
Department of Computer Science and Engineering
Bangladesh University of engineering and technology
tz08128@gmail.com

Ch. Md. Rakin Haider
Department of Computer Science and Engineering
Bangladesh University of engineering and technology
rakinhaider@gmail.com

***Abstract*—**

I. INTRODUCTION

The introduction of Android system have revolutionized the smartphone market. Market share of Android have been increasing ever since its introduction in 2005 and it is reported by Statista[1] to have gained more than 80% market share by 2016. A driving factor behind androids popularity is that it provides a wide range of applications to its users through Google Play Store. In contrast with the principles of Apple Store, Google Play Store allows application from unverified sources. Although such a welcoming approach towards third-party application have enriched the functionality provided to Android users, it has also made the users vulnerable towards attacks from unscrupulous entities. Consequently, a well-developed method to filter out malicious applications from benign ones have become and urgent need to ensure the proper growth of Android eco-system and the privacy and security of the users.

Among the malicious applications, spywares have become a widespread threat towards users privacy. The aim of spywares is to get access to users device and secretly access user's information that are otherwise considered to be private by a user. Attackers may use users secrets to carry out identity theft, credential theft or to force them give up ransoms. Spywares often works in the background of a seemingly benign application. In this work, we take have made an attempt to study the behavior of a category of spywares that works under the veil of a WebRTC application which are often used for real-time audio video communication.

WebRTC protocol is a standard for *Real-Time Communication*(RTC) application and it were first proposed to provide an uniform platform for RTC applications through browsers. Being extremely popular in web platform it has been incorporated in many mobile applications such as WhatsApp, Hangout too. Unfortunately, the set of permissions required for a WebRTC application to work properly may often be exploited by attackers. For example, a audio-video calling application may require camera permission which an attacker may exploit to stream users whereabouts to remote server. Although such an attack may create severe consequences, Androids permission based defense mechanism may fail in this case since users are self-willing to provide those permission in order to use the actual features of the application.

In recent literature, several defense mechanisms against mobile malwares can be found. Several static and dynamic malware analysis techniques have been developed. Static analysis techniques are based on collecting a database of malware signatures and whenever an application is tested it is matched with existing malware signatures and if both the signatures are identical to each other then the application is reported as a malware. On the other hand, dynamic approaches leverages runtime information of a malware and will decide of maliciousness based on behavior of an application. Among the dynamic approaches, DroidMiner[4] proposed a machine learning based approach where they considered API level information as features. DroidSec [5] method is a similar approach where the authors have trained deep learning classifiers on features that were extracted from both static and dynamic analysis of applications. Daniel et al. [2] has suggested that only labeling an app as malicious doesn't mitigate its adversarial effect. They suggested that it is recommended to predict a proper textual description of an application and show it to the users so that they can achieve a better understanding of the application under consideration. In [1] the authors have detected several events and behaviors that are likely to be related with malicious activities. Finally, [1] is an approach to develop a method that is built upon behavior of android malwares.

Existing techniques are likely to fail against Spycam application hidden under WebRTC application, since both of them accesses similar resources and requires similar permissions to work. Therefore, permission based techniques and API level features are likely to treat both applications equally. Moreover, static approaches like that of signature based malware detection doesn't consider WebRTC applications as malicious and will let these applications pass through. In this work we have developed a Spycam application that exploits, these behaviors of WebRTC application to secretly stream audio and video strems of victims to the attacker. Since, existing static and dynamic techniques are not promising in

*This work was not supported by any organization

[1]H. Kwakernaak is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands `h.kwakernaak at papercept.net`

[2]P. Misra is with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA `p.misra at ieee.org`

[1]https://www.statista.com/statistics/266219/global-smartphone-sales-since-1st-quarter-2009-by-operating-system/

this case we have decided to develop a defense mechanism by analyzing the effect of Spycam applications on various mobile parameters.

The contributions of this paper can be summarized as follow,

- Development of a Spycam application by exploiting WebRTC security issues.
- Development of an automated system to collect mobile diagnostics for analysis.
- Analysis of changes in mobile parameter due to the execution of Spycam application to find out their deviation from normal behavior.
- Proposing deterministic techniqeues as the first step of defense mechanism against similar spywares.

The rest of this report is structured as follow. Section **??** contains the preliminary ideas, classification of malwares and details about how WebRTC applications work. Section **??** provides a brief description about previous studies. In Section **??** our proposed techniques have been discussed and Section **??** contains findings of this work.

II. BACKGROUND

In this section we discuss about the types of threats posed by various softwares and a thorough look at the webRTC technology along with the security concerns of webRTC technology. In addition to that, we have reported our findings from the literature to defend against exploitation of the webRTC techniques in mobile environment.

A. THREAT TYPES

There are mainly three types of threat every mobile device faces. These major categories are Malewares, Personal Spywares and Graywares. We shall discuss about how they work and how they spread in the following paragraphs.

Malware

Any software that deliberately tries to carry out a harmful intent against a user of an attacker can be labeled as a malicious software or malware. Although earlier malwares were focused on revealing security vulnerabilities and demonstrating excellent technical ability, in recent times malwares are used as a way to generate money by infecting the computer systems of a huge number of users. Malwares are often divided in multiple categories based on their intent. A few major categories are *worm, virus, trojan horse, bot and rootkit*. Worms are specially designed program that can propagate itself to other devices. Viruses requires to latch on other programs specially operating system to function properly. A software that appears to be useful from the outside but continues to perform malicious actions in the background is called trojan horse. A bot is a software that can be remotely controlled. These property of malwares are not mutually exclusive, rather most malwares can be a member of more than one category. But almost all of the malwares tries to gain access to a device to fulfill some harmful intents.

Alarmingly, the increased connectivity of computer systems has facilitated the faster spread of malwares and made the malware threat more prominent.

Spyware

The main purpose of spyware programs is collecting someone's personal information such as text messages, locations, credit card credentials etc which he may not otherwise want to disclose. Spywares usually publishes the victim's personal information either to the attacker or to a set of interested parties. Many spywares requires to have physical access to the device and should be installed in the user's device without the user's knowledge. In contrast with the purpose of a malware spywares can often be used in benevolent purposes such as monitoring elderly people or tracking the whereabouts of family members etc. The threat of spyware is increasing day by day since most people keep all their devices synced with their mobile and thus risk loosing all their confidential information to attackers.

Grayware

Greywares are applications that doesn't harm the user directly but collect data to use for the purpose of targeted marketing and user profiling. Although many user may think that graywares are a breach to their privacy, many would argue that the resulting features finally works in their favor. Unlike malwares, users willingly choose to use the giving up the data both willingly or unwillingly.

B. Web Real Time Communication

Web Real Time Communication(WebRTC) is a communication protocol developed by World Wide Web Consortium (W3C) and enhanced by Internet Engineering Task Force(IETF)[]. Previously to enable any type of real time communication users were required to have a specially built softwares. The introduction of WebRTC have removed this dependency and standardized all the potentials essential for providing support for real time communication through web browser and are currently available in most browsers. A standard for WebRTC have also introduced inter-browser and inter-application real time communications such as audio and video chat.

An obvious implication of WebRTC is peer-to-peer communications. WebRTC is free and open source so that it can enable the browser with RTC capacity by using simple JavaScript APIs. In recent past web browsers were used only for surfing, checking email, watching videos etc but didn't have any facility to support efficient seamless audio or video conferencing. The voice and video compression and decompression techniques were computation expensive and costly. WebRTC handled these challenges. Its far reaching advantages have encouraged software developers of other

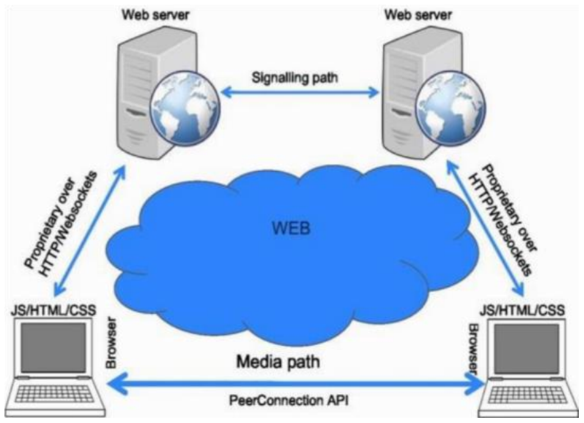

Fig. 1. The WebRTC Trapezoid

platforms such as android to integrate it in their product.

WebRTC maintains a client server architecture with peer-to-peer communication concept between two web browsers. In this architecture, the peers organizes the path to start flow of data between browsers. Network signaling required in this method are performed over HTTP or WebSockets. The browsers can choose to make use of standardized protocols such as Jingle[] or SIP[] or they can choose to develop their own signaling protocol. As shown in Figure 1 the communication occurs between two browsers through server. Such architecture is called Trapezoid model.

It is quite common in WebRTC scenario that both the browsers communicate with the same server when they are running similar web applications from similar web pages. Thus the trapezoid model is often converted into a Triangle model as depicted in Figure 2.

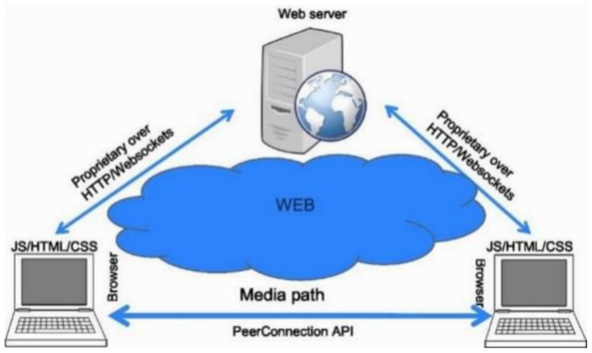

Fig. 2. WebRTC Triangle

The WebRTC standard uses signaling system to set up a peer-to-peer connection. During the signaling system a signaling server is usually used. But the media is sent peer-to-peer. Thus WebRTC reduces load on server since the server faces only the load of setting up the connection not the one of forwarding media. Figure 3 shows the ideal process of WebRTC.

But in practice it is seen that most signaling messages are blocked by Network Addressing Tables(NATs). To overcome

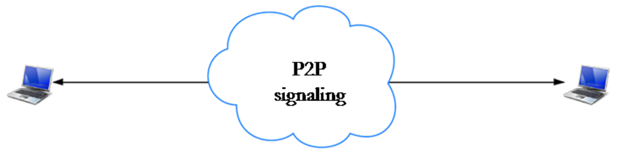

Fig. 3. WebRTC Process

this drawback a STUN server is used to set the peer-to-peer connection. In spite of using STUN server, the signaling process may fail if there are multiple routings existing between two peers. In such a scenario a TURN server is used which acts like a relay for the media stream. It stores a media footprint and relays the media contents to another TURN server or to the peer at the other end. Figure 4 and Figure 5 show the pictorial representation of the process.

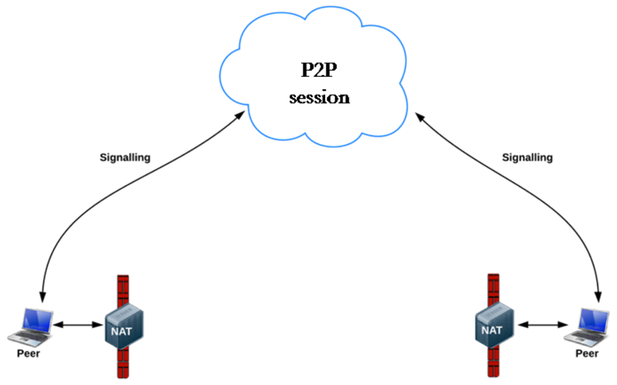

Fig. 4. WebRTC with Signaling but blocked at NAT

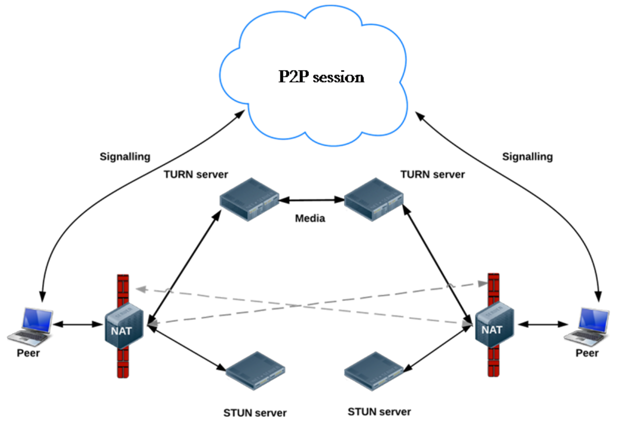

Fig. 5. WebRTC with Signaling, STUN and TURN Servers

The need of proper security mechanism in WebRTC is pretty alarming. Previously attackers were able to compromise microphones and video cameras of nuclear facilities to spy on the activities. Moreover, WebRTC mechanism mostly depends on trust of the two communicating entities which in most case can be faked to gain access to the other device.

C. MOBILE SECURITY MEASURES

Three major platforms of mobile software are iOS, Symbian and Android. Each one of these platforms has their native defense mechanism against malwares. Android mostly depends on permission based protection against malware. On the other hand Apple has developed a strong review process of softwares before they are published on the app store. Similar mechanism has been developed by Symbian but they often failed to prevent many malwares from being published[]. On top of that, many users currently chooses to enable root access which gives the application permissions to circumvent many security mechanism. Due to these issues development of an impenetrable defense mechanism have been a serious concern.

The first step towards developing a proper defense mechanism was through analyzing malware signatures. Signature of a software sample means the series or set of actions performed by that particular sample. A number of tools were developed by vendors which maintained a database of known malware signatures and compared these signatures against potential threats. Such a technique was called the static approach towards malware detection. Generating a signature that is generic enough to detect each malware correctly were at the same time avoid reporting false negatives is a major drawback of this method. As a result researchers have inclined towards developing dynamic detection of malware softwares.

D. RELATED WORKS

In this section we discuss about a few literature which are closely related to our work. In [**?**], [**?**], [**?**] the authors have shown that existence of certain permissions in an apps manifest is not sufficient to detect a software as malware. For example the non-fraudulent video call applications will require same set of permissions as that of a malicious spyware. In [4] the authors have made an attempt to recover from the shortcoming of permission-based approaches by replacing them with a machine learning based approach trained on tailor-made API level features. While designing their features they have considered application specific resource APIs such as Context and Intents, Android framework resource APIs such as Activity Manager, Package Manager, SMSManager, TelephonyManager etc, DVM related APIs such as Runtime and System and System resource APIs such as Sockets, URLConnection etc. Although the classifiers trained on such features demonstrated noteworthy accuracy, their classifies may fail to capture the malicious intents of many spywares. For example, a peer-to-peer chatting application such as viber will be using similar APIs like that of a spyware and may escape the defense mechanism put up by the users.

With the advent of deep learning methods, researchers have conducted studies to find out more accurate deep learning classifiers which may result in replacing the traditional classifiers. In [5] the authors have trained a deep classifier on 220 features generated by both static and dynamic analysis and were able to achieve as much as 96% accuracy on their dataset. They have gathered their dataset from the renowned contagio malware dataset.

The authors of [2] have stated that only the prediction of threats is not sufficient to appropriately generate awareness among the user. So they focused on generating a linguistic summary of the threat and train a classifier that can predict the summary of the threats. In spite of their success in generating descriptive textual summary of the malwares their method suffered from accuracy issues when they faced obfuscated applications since in obfuscated application the API calls could not be deducted easily. A few modification were suggested but with less improvement.

The author of [6] have performed a thorough analysis of malware behavior and commented about which event in an android eco-system are important while detecting malwares. These events can be considered while designing future detection and prevention mechanisms.

Finally, Amin et. al. [1] developed a behavioral malware detection approach. They analyzed both network calls and system calls in their method.

Although malware detection techniques have been thoroughly studied, none of these studies shall be effective in the case of spywares. Specially spywares in disguise of peer-to-peer communication applications. Since both the spyware and the p2p communication app performs similar task i.e. access the camera, send signaling messages to servers and transfer data with WebRTC standards as well as request for granting similar permissions most methods will fail to detect these spyware. As a result these spywares can be easily disguised as benign softwares and distributed both either personally or to the community.

III. OUR APPROACH

In this project we have planned to develop a spyware that exploits the WebRTC standards. As mentioned earlier WebRTC security mechanism depends largely on trust factor as well as a few authentication mechanisms. We have developed an application that initiates a real time communication between two peers without the permission of one and streams his camera feed to the attacker. We have disguised this spyware under the hood of a chatting application with the support of video calling features.

As the attacker we have designated a specific id who has access to all the contacts in the system. In addition to that, we can call any person from the attackers id. Moreover the call received from attacker is kept hidden from the victim. We have streamed all the camera feeds to a centralized server and thus generated a method to spy on anyone who uses this app. Figure 6 provides a pictorial representation of the entire process.

To propose an appropriate defense mechanism against such spyware, we will analyze the effect of such a spyware on various mobile parameters and compare their effects on changing these parameters over time with that of non-malicious WebRTC based voice calling applications. As candidates of mobile parameters we shall consider battery consumption, frequency and types of service invocation, hardware access, network usage, memory consumption and CPU usage. WebRTC applications usually drains battery faster when it is being used by the user. But a spyware is likely to drain the battery from the background since camera

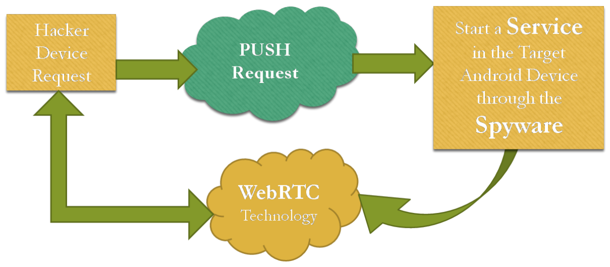
Hacker Device Request
PUSH Request
Start a Service in the Target Android Device through the Spyware
WebRTC Technology

Fig. 6. The Treat Model

access and streaming requires a significant amount of battery usage. Secondly, a popular approach in malware detection is capturing the service invocation behavior of a malware. Intuitively, a spyware is likely to access camera and other hardwares in a way that is different than standard WebRTC applications and it is likely to result in increased memory and CPU consumption.

IV. IMPLEMENTATION DETAILS

We have developed our spyware application with target SDK Android 7.1.1 and minimum required SDK Android 4.0. We have used *Kotlin* for implementation of both the malware and the anti-malware software. To be able to successfully use our application a user must have camera, microphone, network connectivity and Google push service support in his device. Users will be requested to provide Camera and Audio permission during runtime. The entire application is implemented following standard WebRTC guidelines.

During the development of the malware we have found it almost impossible to bypass some WebRTC standards restricting the streams of any camera feed without rendering it in the display of the device. So in order to render the camera feed without showing it to the user, we have chosen to render the camera feed in a dialogue activity with height and width of the dialogue activity to 1dp and rendered the camera feed in that dialogue activity. Although users are unable to view the dialogue activity due to its small size, it doesn't violate the constraint of rendering the camera feed before sending it to a distant user. Moreover, to hide it from the user we have removed it from the list of recent programs(which is usually accessible to the user by a long press of the options button). To do this the application manifest file must set the tag *excludeFromRecent* to *true*. To further strengthen its stealth mode, the application can be built as a background service but to avoid complexity we have omitted that from the implementation.

In order to develop the anti-malware software we have used the android terminal and the *dumpsys* command. In Android system, dumpsys is a command that runs on Android devices and provides information about system services. We can call dumpsys from the command line using the Android Debug Bridge (ADB) to get diagnostic output for all system services running on a connected device. This output is typically more verbose than one may want, so it is often required to use the available command line options in which the authors are interested. There are seven command line options available with this command. Among these options *services* is the most important option. Though this option we can specify which mobile parameters we are willing to gather and filter out all unnecessary data. Among the widespread uses of this option most important ones are inspecting input diagnostics, battery diagnostics, network diagnostics, testing UI performance for specific packages and viewing memory-info. From these options we have chosen to extract *batterystats* since battery stats provides the following list of information,

- A dummy integer
- The user ID associated with the observation
- The aggregation mode:
- Section identifier, which determines how to interpret subsequent values in the line. There are 48 types of section identifier. For example, *nt* stands for network usage, *bt* stands for battery usage etc. For each identifier the interpretation of the subsequent fields are different and can be found in [3].

The batterystats command can also take additional parameters. In order to generate a machine readable version we have used the *–checkin* option along with the dumpsys command. To summarize, the command that was used by our anti-malware system is as follows,

adb shell dumpsys batterystats –checkin

In order to analyze the behavior of an application, we have extracted the values of various mobile parameters through the dumpsys command, in frequent intervals over a certain period of time. We have executed the command using the android Runtime package API. By generating the system dumps in frequent intervals, we plan to analyze the changes in mobile parameters over a specified length of time due to the execution of the malware. The pseudo-code snippet is as follows,

Algorithm 1 Algorithm to generate data for analysis

```
procedure ANTI-MALWARE($interval, totalTime$)
    $i \leftarrow 0$
    while $i \leq totalTime$ do
        Invoke service to run ExecuteDumpsys
        Sleep for $interval$ seconds
        $i \leftarrow i + interval$
procedure EXECUTEDUMPSYS
    process ← Runtime.getRuntime().exec(
        ”adb shell dumpsys batterystats –checkin”) ;
    process.waitFor()        ▷ Waiting for execution to end
    $data$ ← process.getOutputStream()
    Write $data$ to file
```

A drawback of this approach is that to execute the dumpsys command from an application the application need to be installed by the device manufacturer in the time of OS installation. Although an alternative approach is to run the application in a rooted device, it should be kept in mind that rooting an android device may give rise to a number of additional security issues.

V. RESULTS

1) Experimental Setup: In this work, we have decided to collect diagnostics from two different devices. The first device is set up specifically for this study and have only a few other application running alongside whereas the other device have been under regular use for a while and have a wide-range of active applications. We have decided to compare the behavior of our tailor-made spycam applications with that of the behavior of several other popular WebRTC applications i.e. Hangout, WhatsApp, imo, Skype and Messenger. We have considered the several device parameters for analytics. Each parameter along with the intuition behind its inclusion in experiment are as follow,

- **Battery Consumption :** It is expected that continuous camera access and media streaming will require more cpu usage and will result in more battery consumption. Standard WebRTC app visibly drains the battery, but any spy-cam app will do it when it is invisible to the user. Moreover, battery consumption rate from services should be close to zero, but for such spywares - it should be a lot higher than the average.
- **Service Invocation :**A device may receive spontaneous PUSH service reception for such spywares. Upon reception of a push notification the user will not be shown any UI for such service activation. And the invoked services will access camera and/or microphone from such malicious activities.
- **Hardware Access :**Spywares access hardwares without prompting users for access request. In addition to that, audio and video streaming requires use of a set of hardwares. Misusing the access-permissions (both install-time & run-time) for specific hardware can be a hint towards malicious activities.
- **Frequency Detection :**The frequency of getting PUSH messages through standard GCM (Google Cloud Messaging) or FCM (Firebase Cloud Messaging) or any other standard PUSH services can be an indication of spying activities. It should be mentioned that such push messages will not require any user interaction in such applications.
- **Network Usage :** It is obvious that higher streaming activities should be noticed in similar applications. Therefore, analyzing network usage can have an impact on spyware detections.

The resulting behaviors can be observed from the following graphs.

VI. FUTURE WORKS

VII. CONCLUSIONS

REFERENCES

[1] Mohammad Rakib Amin, Mehedee Zaman, Md. Shohrab Hossain, and Mohammed Atiquzzaman. Behavioral malware detection approaches for android. In *2016 IEEE International Conference on Communications, ICC 2016, Kuala Lumpur, Malaysia, May 22-27, 2016*, pages 1–6. IEEE, 2016.

[2] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[3] Android Developer. User guide dumpsys. `www.developer.android.com`, 2018. [Accessed February 23th, 2018].

[4] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip A. Porras. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, pages 163–182, 2014.

[5] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 371–372, 2014.

[6] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 95–109. IEEE Computer Society, 2012.