Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

ELSEVIER

Computers & Security

# AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection

*Ali Feizollah* [a,*], *Nor Badrul Anuar* [a,*], *Rosli Salleh* [a], *Guillermo Suarez-Tangil* [b,1], *Steven Furnell* [c]

[a] *Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia*

[b] *Computer Security (COSEC) Lab, Department of Computer Science, Universidad Carlos III de Madrid, 28911 Leganes, Madrid, Spain*

[c] *Centre for Security, Communications and Network Research, School of Computing, Electronics and Mathematics, Plymouth University, Drake Circus, Plymouth PL4 8AA, UK*

ARTICLE INFO

*Article history:*
Received 5 May 2016
Received in revised form 7 November 2016
Accepted 12 November 2016
Available online 16 November 2016

*Keywords:*
Mobile malware
Android
Intent
Smartphone security
Static analysis

ABSTRACT

The wide popularity of Android systems has been accompanied by increase in the number of malware targeting these systems. This is largely due to the open nature of the Android framework that facilitates the incorporation of third-party applications running on top of any Android device. Inter-process communication is one of the most notable features of the Android framework as it allows the reuse of components across process boundaries. This mechanism is used as gateway to access different sensitive services in the Android framework. In the Android platform, this communication system is usually driven by a late runtime binding messaging object known as Intent. In this paper, we evaluate the effectiveness of Android Intents (explicit and implicit) as a distinguishing feature for identifying malicious applications. We show that Intents are semantically rich features that are able to encode the intentions of malware when compared to other well-studied features such as permissions. We also argue that this type of feature is not the ultimate solution. It should be used in conjunction with other known features. We conducted experiments using a dataset containing 7406 applications that comprise 1846 clean and 5560 infected applications. The results show detection rate of 91% using Android Intent against 83% using Android permission. Additionally, experiment on combination of both features results in detection rate of 95.5%.

## 1. Introduction

Smartphones have emerged as popular portable devices with increasingly powerful computing, networking and sensing capabilities, and they are now far more powerful than early personal computers (PCs). In addition, their popularity has been repeatedly corroborated by recent surveys (Gartner, 2015). The combination of device capability and popularity has served to make them an attractive target for malware. Accordingly, malware is quickly permeating most popular Android-based applications markets. In the case of official applications market (Google Play), operators are generally more concerned about the security aspect of the software they distribute. For

[*] *Corresponding authors.*
*E-mail addresses:* ali.feizollah@siswa.um.edu.my (A. Feizollah), badrul@um.edu.my (N.B. Anuar).
[1] Currently at Royal Holloway University of London. *Email*: guillermo.suarez-tangil@rhul.ac.uk.

http://dx.doi.org/10.1016/j.cose.2016.11.007

**Table 1 – Intent section of clean and infected versions of Zurich Cantonal Bank application.**

| Clean version | Infected version |
|---|---|
| android.intent.action.MAIN | android.intent.action.MAIN<br>android.intent.action.BOOT_COMPLETED<br>android.provider.Telephony.SMS_RECEIVED |

instance, Google Play employs a review system to vet potentially dangerous applications (Oberheide and Miller, 2012). Despite all these efforts, commercial surveys still report a large number of malicious applications attacking the Android markets. For instance, GData reported nearly half a million new Android malware in 2015[1]. More recently, new malwares such as the BrainTest (Polkovnichenko and Boxiner, 2015) have succeeded in infecting over half a million Android devices, targeting Google Play in particular. Many recent studies have resulted in a number of automated approaches to tackle the spread of malware (Aresu et al., 2015; Feizollah et al., 2013; Narudin et al., 2016; Tam et al., 2015). Static analysis techniques, which have traditionally been used for detecting malware targeting desktop computers, have recently gained popularity as effective measures for the protection of mobile applications (Desnos, 2012). In particular, static approaches aim at detecting Android malware by analyzing their permission usage (Zhang et al., 2013), mining their code structures (Suarez-Tangil et al., 2014), understanding the components they used (Arp et al., 2014), and monitoring the APIs they invoked (Aafer et al., 2013; Arp et al., 2014; Yang et al., 2014). Inter-process communication is one of the most notable features of the Android framework as it allows the reuse of components across process boundaries. It is used as gateway to access different sensitive services in the Android framework. In the Android platform, this communication system is usually driven by a late runtime binding messaging object known as Intent. Intent objects provide an abstract definition of the operations an application intends to perform.

The rich semantics encoded in this type of component indicate that Intent could be used to characterize malware. For instance, the listing in Table 1 shows an excerpt of Intent actions used in a legitimate banking application and the actions stipulated in the infected version of the same application. In this example, it is obvious that the infected version of the application is subscribing to a notification service that will be triggered by the Android OS whenever the BOOT_COMPLETED event occurs. In addition, SMS_RECEIVED allows the subscriber to access all incoming SMS messages (Fratantonio et al., 2016). While the former action is used by the malware as a form of evasion, the latter is used to steal the Transaction Authorization Code (TAC) (Jain, 2015; Jiang and Zhou, 2013).

In this paper, we propose AndroDialysis[2], a system that analyzes two different types of Intent objects, i.e., implicit and explicit Intents. To evaluate the effectiveness of the proposed system, we will compare our results with that from a baseline detection system that uses similar level of granularity, and we will then analyze the permissions usage. In summary, we make the following contributions in this paper:

1. We propose the use of Android Intents (implicit and explicit) for detecting Android malware. The usage of Intents will be extracted from both clean and infected applications in a dataset containing 7406 applications.
2. We extract permissions used by each application and evaluate the effectiveness of our approach when compared to the use of permissions. We also conduct experiment on combination of Android permission and Android Intent to verify that they are not overlapping.
3. We also compare the time taken to process permissions and Intents in our experiments, as it is important to determine which component of the Android file is faster and more efficient. Furthermore, we calculated power consumption of AndroDialysis and compared it with three popular applications.

This paper is organized as follows: Section 2 explains in detail Android Intent, and presents a snippet code for implicit and explicit Intents, respectively. Section 3 discusses the method of data collection and analysis of the dataset, analyzing the permission and Intent. Section 4 describes the proposed system and its various modules and sub-modules. Section 5 presents details of experiments and the results obtained, as well as evaluation of the proposed system. Section 6 reviews related works done by other researchers, and highlights their weaknesses and strengths. Section 7 concludes this paper by summarizing main findings from this research.

## 2. Android intent

Intent is a complex messaging system in the Android platform, and is considered as a security mechanism to hinder applications from gaining access to other applications directly. Applications must have specific permissions to use Intents. This is a way of controlling what applications can do once they are installed in Android. Intent-filter – defined in AndroidManifest.xml file – announces the type of Intent the application is capable of receiving.

Applications use Intents for intra-application and inter-application communications. Intra-application communication takes place inside an application between activities. An Android application consists of many activities, each referring to buttons, labels, and texts available on a single page of the application, with which the user interacts. When interacting with the application, the user moves from activity to activity (i.e. from page to page). Android Intents assist developers in performing interactions among the activities. Furthermore, Intents are used in pushing data from one activity to another, carrying the results at the end of any particular activity (Aftab and Karim, 2014).

Inter-application communication is achieved when applications send messages or data to other applications through Intent. The applications should also be able to receive data from other applications. To receive Intents, applications must define what type of Intent they accept in the Intent section of AndroidManifest.xml file, as intent-filter. Many past studies (Chakradeo et al., 2013; Feng et al., 2014; Luoshi et al., 2013) referred to this type of Intent. The actual communication between two applications is done through the Binder, which handles all inter-process communications. The Binder pro-

[1] www.gdata-software.com.

[2] ***Andro***id Deep Intent An***alysis***.

**Table 2 – Sample code snippet of explicit and implicit Intents.**

| Explicit Intent | Implicit Intent |
|---|---|
| String url = ”www.yahoo.com”;<br>Intent explicit = new Intent(Intent.ACTION_VIEW);<br>explicit.setData(Uri.parse(url));<br>explicit.setPackage(”com.android.chrome”);<br>startActivity(explicit) | String url = ”www.yahoo.com”;<br>Intent implicit = new Intent(Intent.ACTION_VIEW);<br>implicit.setData(Uri.parse(url));<br>startActivity(implicit); |

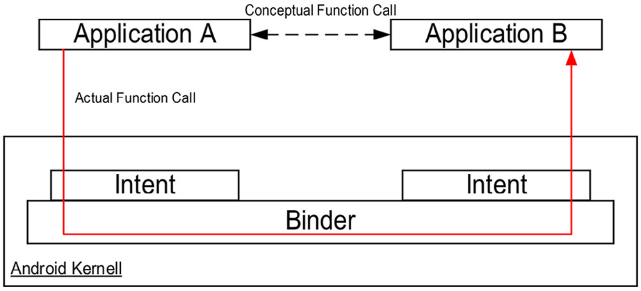

**Fig. 1 – Inter-application communication using Android Intent and Binder.**

vides the features for binding functions and data between one execution environment and another, as each Android application runs in its own Dalvik[3] environment. The Intent mechanism is considered higher than Binder, hence, it is built on top of Binder.

Fig. 1 shows the architecture of inter-application communication. The Binder driver manages part of the address space of each application and makes it as read-only and all writing is done by the kernel section of Android. When application A sends a message to application B, the kernel allocates some space in the destination applications memory, and copies the message directly from the sending application. It then queues a short message to the receiving application telling it the location of the received message. The recipient can then access that message directly because it is in its own memory space. When application B has finished processing the message, it notifies the Binder driver to mark the memory as free (Hellman, 2013).

There are two types of Intent: explicit and implicit. When developers know exactly what component to use to perform a specific action, they use explicit Intent. This component can be any activity, service, or broadcast receiver. Explicit Intent is used for intra-application and inter-application communications, and developers use this type of Intent to navigate from an activity to another activity inside applications, as well as to exchange messages between applications. For instance, there are some applications, which are used for browsing, such as the default browser on the device or Google Chrome. Developers use explicit Intent to request Android to open a link specifically using Google Chrome. On the other hand, developers use implicit Intent and ask Android to open a link, but they do not specify the exact target application. In response, Android offers a list of all applications capable of opening a link to the user. Such a list is populated based on the intent-filter section of AndroidManifest.xml files. In our study, our aim is to extract both implicit and explicit Intents and conduct a comprehensive evaluation of their effectiveness in malware detection.

Intents have three components – action, category, and data. The action component describes what kind of action is to be executed by the Intent such as MAIN, CALL, BATTERY LOW, SCREEN ON, and EDIT. Intents specify the category they belong to, such as LAUNCHER, BROWSABLE and GADGET. The data components provide the necessary data to the action component. For instance, CALL action requires phone number, and EDIT action needs document or HTTP URL to complete the action. Table 2 shows a sample code of explicit and implicit Intents.

Table 2 shows that implicit Intent uses Intent.ACTION_VIEW to open the specified URL. However, explicit Intent states the exact component name – in this case com.android.chrome – to open the URL.

## 3. Data collection and analysis

For our experiment, we used real-world applications that include both clean and infected applications. We gathered clean applications from Google Play[4] and scanned them with VirusTotal[5] to ensure the cleanness of the applications. The applications collected include both free and paid types since ProfileDroid (Wei et al., 2012) mentioned that paid applications behave differently from free ones, and it is important to include all such applications. Google Play applications are categorized into 27 main application categories, and games category has 17 sub-categories. We gathered samples from 24 main application categories, and 17 games sub-categories to cover a wide variety of applications, as shown in Table 3.

The clean dataset contains 1846 applications. Additionally, we used DREBIN (Arp et al., 2014) as infected dataset. It is a collection of 5560 applications from 179 different malware families. We used our Python code to extract permission and Intent from applications in our dataset. The top 10 permissions of both clean and infected applications are shown in Table 4. Google categorizes Android permissions into four groups – normal, dangerous, signature, and signatureOrSystem (Google, 2014).

[3] Each Android application runs in its own Dalvik virtual machine, which is separate from other applications. An Android device can run multiple Dalvik virtual machines for each application efficiently. Applications communicate through Android Intent. Additionally, they can share data using content providers.

[4] http://play.google.com.

[5] www.virustotal.com.

**Table 3 – Categories of gathered applications**

Books and references
Business
Comics
Communication
Education
Entertainment
Finance
Health and fitness
Music and audio
News and magazines
Personalization
Medical
Weather
Travel
Photography
Productivity
Shopping
Social
Sports
Media and video
Transportation
Live wallpaper
Tools
Games – action
Games – card
Games – casino
Games – casual
Games – educational
Games – music
Games – puzzle
Games – role playing
Games – word
Games – board
Games – adventure
Games – strategy
Games – simulation
Games – family
Games – racing
Games – sports
Games – arcade

Table 4 also shows that five permissions are common – as highlighted – between clean and infected applications, such as INTERNET, WRITE_EXTERNAL_STORAGE, WAKE_LOCK, ACCESS_COARSE_LOCATION, and READ_PHONE_STATE. However, these applications have five different permissions among the top 10 permissions. Infected applications request SEND_SMS, RECEIVE_SMS and READ SMS permissions, which are categorized as dangerous. In fact, WRITE_SMS, which is also dangerous, should be in the list of top frequent permissions. It is ranked 11th in our dataset, and it is requested by 22% of infected applications. Therefore, it is evident that infected applications request four SMS-related permissions to have full access to SMS functionality of the devices. In our experiment, 30% of infected applications requested the ACCESS_FINE_LOCATION permission to access precise location, and 33% of them requested the ACCESS_COARSE_LOCATION permission, which is a common permission, to access proximate location. In general, the viciousness of infected applications can be gauged through permissions. We also extracted Intent of applications, as shown in Table 5, which shows top 10 Intents used in clean and infected applications. It is worth noting that the VIEW Intent was removed from the top 10 Intents, since it is used in all clean and infected applications.

Malicious applications wait for BOOT_COMPLETED to start their malicious activity. CALL and DIAL are used for making phone calls. CALL requires CALL_PHONE permission, whereas DIAL does not require such permission. As it is presented in Table 5, DIAL is used more than CALL, which allows the malicious application to make a premium phone call without user's knowledge.

**Table 4 – Top 10 Permissions in clean and infected applications.**

| Clean applications | | Infected applications | |
|---|---|---|---|
| Permissions | Frequency | Permissions | Frequency |
| INTERNET | 98% | INTERNET | 98% |
| ACCESS_NETWORK_STATE | 89% | READ_PHONE_STATE | 89% |
| WRITE_EXTERNAL_STORAGE | 83% | WRITE_EXTERNAL_STORAGE | 67% |
| WAKE_LOCK | 53% | SEND_SMS | 54% |
| READ_PHONE_STATE | 52% | RECEIVE_SMS | 38% |
| ACCESS_WIFI_STATE | 48% | WAKE_LOCK | 38% |
| GET_ACCOUNTS | 42% | READ_SMS | 37% |
| VIBRATE | 41% | ACCESS_COARSE_LOCATION | 32% |
| BILLING | 39% | ACCESS_FINE_LOCATION | 30% |
| ACCESS_COARSE_LOCATION | 24% | READ_CONTACTS | 23% |

**Table 5 – Top 10 Intents in clean and infected applications.**

| Clean applications | | Infected applications | |
|---|---|---|---|
| Intents | Frequency | Intents | Frequency |
| SEND_MULTIPLE | 45% | BOOT_COMPLETED | 56% |
| SCREEN_OFF | 23% | SENDTO | 45% |
| USER_PRESENT | 18% | DIAL | 42% |
| SEARCH | 17% | SCREEN_OFF | 37% |
| PICK | 10% | TEXT | 28% |
| DIAL | 9.5% | SEND | 27% |
| GET_CONTENT | 9% | USER_PRESENT | 22% |
| EDIT | 8.7% | PACKAGE_ADDED | 21% |
| MEDIA_MOUNTED | 8% | SCREEN_ON | 18% |
| BATTERY_CHANGED | 7% | CALL | 10% |

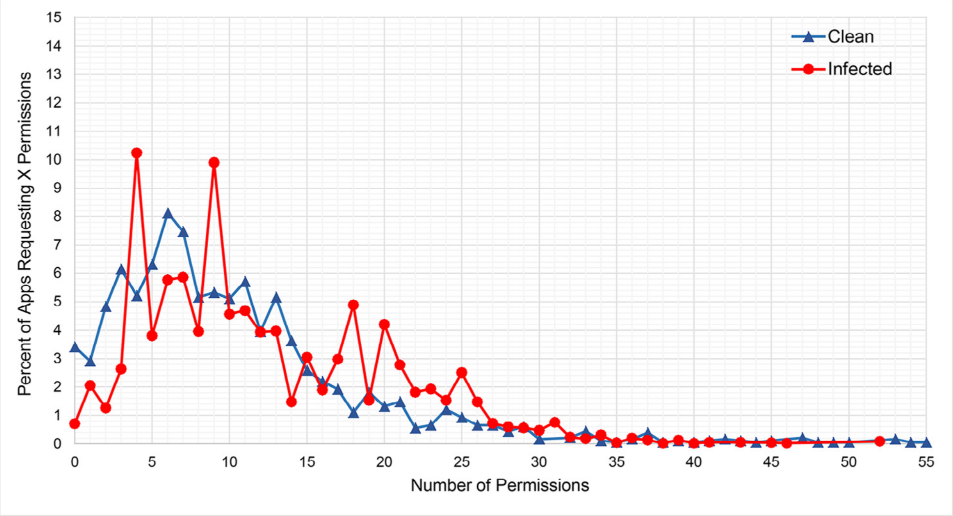

**Fig. 2 – Percent of applications that request specific number of permissions.**

Fig. 2 shows the percentage of applications that requested permissions – clean and infected – in two datasets. The graph shows that infected applications request more permissions as there are spikes at multiple points in the figure. Furthermore, only 2% of clean applications requested between 35 and 55 permissions, compared to 7% of infected applications. This is indicative of the vicious intentions of infected applications.

Similarly, Fig. 3 shows the percentage of applications that requested Intents – implicit and explicit – in two datasets. When comparing Fig. 2 and Fig. 3, the difference between their X-axis is obvious. While permissions have maximum number of 55, number of Intents ends at 250. The wide difference is due to the fact that developers use Intents much more frequently than permissions in the code to perform actions.

Intent and permission are potentially useful features for Android malware detection. However, according to Moonsamy et al. (2013), there are requested permissions as well as required permissions. It is possible that actual permissions used by applications are different from the requested permissions that are sent to the user for approval. On the other hand, Intent reflects the actual intentions of applications resulting directly from activities. This indicates that Intent is more effective for malware detection.

## 4. Mobile malware detection system overview

Fig. 4 shows the architecture for our proposed system, AndroDialysis. The top level of the architecture – Android application framework – refers to applications installed on the device. The detector module performs the main task of detection. It consists of four sub-modules – decompiler, extractor, intelligent learner, and decision maker. The system sends the results to users through the graphical user interface. The following sections describe four sub-modules in more detail.

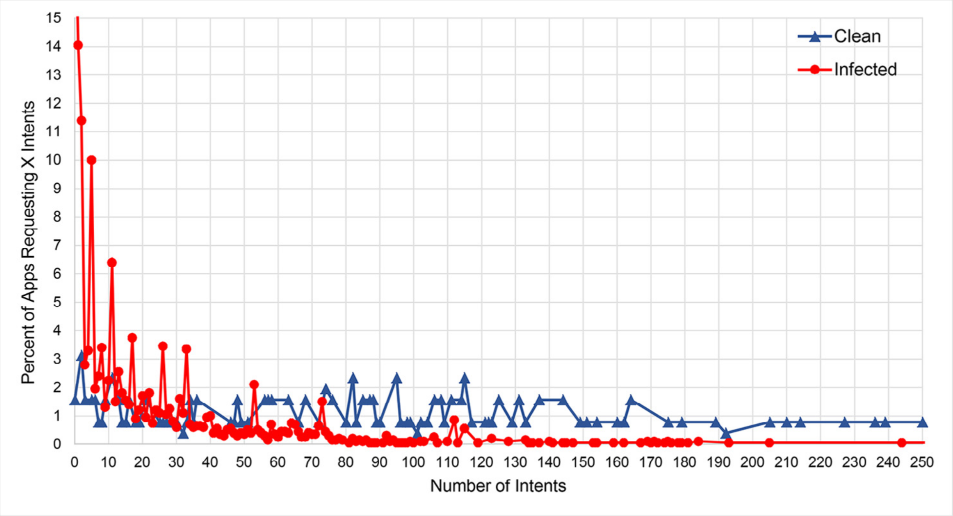

**Fig. 3 – Percent of applications that request specific number of Intents.**

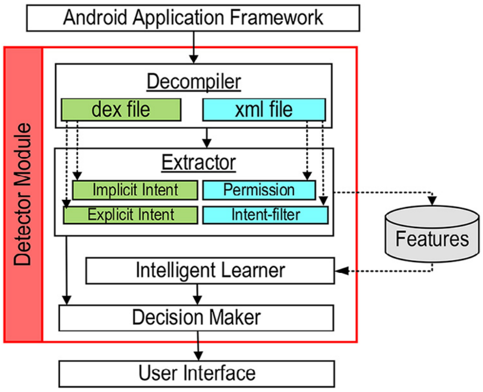

**Fig. 4 – Overview of AndroDialysis, a mobile malware detection system.**

### 4.1. *Decompiler*

The decompiler sub-module is responsible for dissecting the apk files and decoding its components. Every apk file has various components. AndroidManifest.xml is a scrambled file and needs to be decoded in order to make it readable. Similarly, the dex file is a Java source code compiled in Dalvik format and needs to be decompiled. After decompilation, the produced file is not a pure Java code, but it is easy to read. We used Apktool for decompiling Android files, since it utilizes the latest Android SDK, which is better in optimizing files (Winsniewski, 2012). Decompiling files result in readable AndroidManifest.xml file and generate Smali version of Java code.

### 4.2. *Extractor*

The extractor sub-module extracts explicit Intent, implicit Intent, intent-filter, and permission from Java code and AndroidManifest.xml file for processing in subsequent submodules. The BeautifulSoup package of the Python language is used to extract intent-filter and permission sections from the AndroidManifest.xml file (Richardson, 2007). In order to extract Intents from Java code, we used Androguard to reverse dex files and get Intents (implicit and explicit) from the code. The extracted data are stored in a features database for use in the next process. Furthermore, a copy of data is sent to the decision maker sub-module for determining maliciousness of the data, which will be discussed in Section 4.4.

### 4.3. *Intelligent learner*

This sub-module takes data from the features database and uses Bayesian Network algorithm to learn pattern of the data. It then sends output model to the decision maker sub-module. The Bayesian Network algorithm (Friedman et al., 1997) was chosen to evaluate our system because it has been successfully used in real-world problems, for example Cohen et al. (2003) used Bayesian Network in human facial expression recognition and achieved a good performance. It is a dual-process algorithm, it first learns network structure, and then it learns probability tables. Bayesian Network uses local score metrics to learn the network structure of data. It is considered an optimization problem in which the quality of the network is optimized. To calculate the local score, Bayesian Network employs search algorithms. Once the network structure of data has been learned, Bayesian Network utilizes estimators to learn the probability tables (Bielza and Larrañaga, 2014). Two widely used estimators are simple estimator, and multinomial estimator. The aforementioned two steps are defined as follows.

Suppose that $V=\{x_1,\ldots\ldots,x_k\}, k\geq 1$ is a set of variables. Bayesian Network B over V is a network structure $B_S$ that is a directed acyclic graph known as DAG over the set of variables V. It is also a set of probability tables $B_P=\{p(v|pa(v))|v\in V\}$ where $pa(v)$ is the set of parents of v in $B_S$. Finally, a Bayesian Network represents a probability distribution $P(V)=\prod_{v\in V} p(v|pa(v))$.

Compared to other algorithms, the Bayesian Network has the following advantages:

- It is a fast algorithm with low computational overhead once trained.
- It has the ability to model both expert and learning systems with relative ease. It integrates probabilities into the system. It is also considered as a performance-tuning tool, but without incurring computational overhead.
- Many outstanding real-world applications have used this algorithm and have performed comparably well against other state-of-the-art algorithms (Bielza and Larrañaga, 2014).

As mentioned above, Bayesian Networks are collections of directed acyclic graphs (DAGs), where the nodes are random variables, and where the arcs specify the independence assumptions between these variables. It is difficult to search the Bayesian Network that best reflects the dependence relationship in a database of cases because of the large number of possible DAG structures, given even a small number of nodes to connect. As a result, researchers have developed various search algorithms to overcome this problem. In this paper, we use four search algorithms for our experiments – K2, Geneticsearch, HillClimber, and LAGDHillClimber algorithms.

**K2** algorithm heuristically searches for the most probable belief network structure in a given database of cases, which includes different combinations of values for attributes (Ruiz, 2005). **Geneticsearch** algorithm uses the genetic algorithm to find the optimum result in a Bayesian Network. The algorithm is based on the mechanics of natural selection and natural genetics. Although it is capable of solving complex problems, it is a time-consuming algorithm for some data (see Table 9) (Yan and Cercone, 2010). It combines survival of the fittest among string structures with a structured, yet randomized, information exchange to form a search algorithm that under certain conditions evolves into the optimum with a probability that is arbitrarily close to one (Larrañaga et al., 1996).

The **HillClimber** search algorithm starts learning by initializing the structure of Bayesian Network. Unlike previous algorithms that potentially get stuck in the search process, the Hill Climber solved that problem (Chickering et al., 1995). Each possible arc from any node is then evaluated using leave-one-out cross validation to estimate the accuracy of the network with that arc added. If no arc shows any improvement in accuracy, the current structure is determined. An arc that has the

Table 6 – Results of Android Permission and Android Intent experiments.

| | Android permission | | | | Android intent | | | |
|---|---|---|---|---|---|---|---|---|
| | Simple estimator | | Multinomial | | Simple estimator | | Multinomial | |
| | TPR | FPR | TPR | FPR | TPR | FPR | TPR | FPR |
| K2 | 82% | 18% | 24% | 76% | 89% | 11% | 19% | 81% |
| Geneticsearch | 83% | 17% | Null | Null | 91% | 9% | Null | Null |
| HillClimber | 82% | 18% | 24% | 76% | 89% | 11% | 19% | 81% |
| LAGDHillClimber | 83% | 17% | Null | Null | 91% | 9% | Null | Null |

most improvement is retained, but the node the arc points to is removed. This process is repeated until there is just one node remaining, or no arc can further be added to improve the classification accuracy (Jo et al., 2011). The **LAGDHillClimber** search algorithm uses a Look Ahead Hill Climbing algorithm. Unlike Hill Climber, it does not calculate a best arc (by adding, deleting or reversing an arc), but it considers a sequence of best arcs instead of considering the best arc at each step. Since it is very time-consuming to find the best sequence among all the possible arcs, it must first find a set of good arcs and then find the best sequence of arcs among them (Salehi and Gras, 2009). Such improvement over Hill Climber algorithm results in better performance (see Table 6).

We evaluate the performance of Bayesian Network using k-fold cross validation. In this method, the dataset is divided into $k$ subsets, and the holdout method is repeated $k$ times. Each time, one of the $k$ subsets serves as the test set and the other $k - 1$ subsets are compiled to form a training set. Then, the average error across all $k$ trials is computed. The advantage of this method is that it matters less how the data are divided. Every data point gets to be in a test set exactly once, and in a training set $k - 1$ times. The variance of the resulting estimate is reduced as $k$ increases (Feizollah et al., 2013). Specifically, a 10-fold option is used, which is described as applying the classifier to data 10 times and every time the dataset is divided into 90:10 groups – 90% of data used for training, and 10% used for testing, which is widely used among researchers (Damopoulos et al., 2012). At the end, this sub-module produces a model – based on available data in the features database – that is used for detection purpose. It is worth noting that the intelligent learner is constantly learning from the data added to the features database.

### 4.4. *Decision maker*

The decision maker sub-module is responsible for determining whether the data are clean or malicious. It receives two sets of data from the extractor and the intelligent learner sub-modules. A set of data from the intelligent learner sub-module contains a produced model based on the collection of data in the features database. The model is then used to vet the data received from extractor sub-module. Another set of data that is received from the extractor sub-module contains extracted data of one application. The decision maker sub-module utilizes the model to determine the maliciousness of the application. The final decision is passed to the user interface module, which prepares appropriate message for the user and presents it through the graphical user interface, as shown in Fig. 5. Such design of the decision maker sub-module ensures faster detection and higher performance, as it was adopted by Shabtai et al. (2014).

## 5. Results and discussion

In this section, we discuss our results and findings. It is important to restate that the purpose of this paper is to study the effectiveness of Android Intent (implicit and explicit) in malware detection, and not malware detection *per se*. We present the results from experiments conducted on permissions, Intents, and both in Android malware detection. Additionally, to get a better assessment of the current development of Android Intent, we analyzed our datasets.

### 5.1. *Intent analysis and attacks*

We analyze Intents in our datasets from the security standpoint to assess the current status or importance of Intents. As

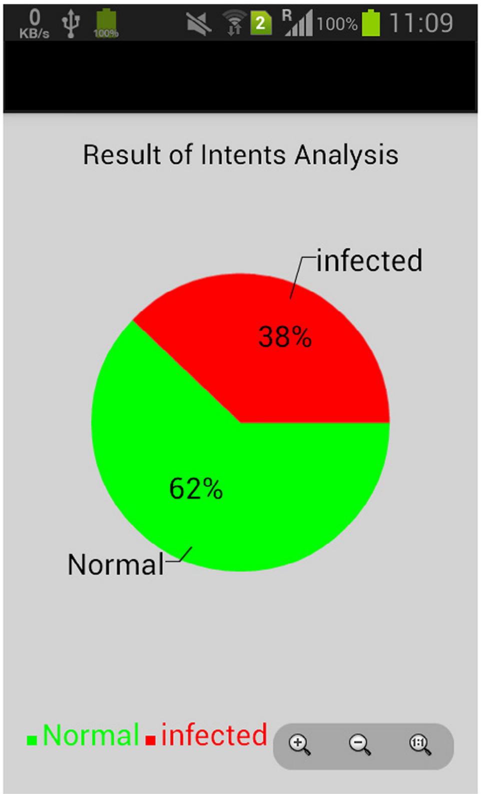

**Fig. 5 – Screenshot of the results presented to the user.**

mentioned in Section 2, implicit Intent does not specify its destination component. However, it is offered to entities that can receive specific type of Intent. Therefore, when an application sends an implicit Intent, there is no guarantee that the Intent will be received by the intended recipient. A malicious application can intercept an implicit Intent simply by declaring an intent-filter – in AndroidManifest.xml file – with all the actions, data, and categories listed in the Intent. This situation – unauthorized Intent receipt – causes the malicious application to gain access to all the data in any matching Intent, resulting in activity hijacking (Chin et al., 2011).

In our dataset, infected applications declare intent-filter 7.5 times more than clean applications. On an average, each clean application declares 1.18 intent-filters, whereas each infected application declares 1.61 intent-filters. Thus, it is evident that infected applications tend to intercept Intents using intent-filters until they succeed in hijacking the activities.

In view of this threat, it is suggested that developers use explicit Intent so that the recipient is clearly specified in order to hinder malicious applications from hijacking the activities. We have analyzed our dataset with regard to this threat, and found that 28.78% of Intents used were implicit and 71.22% were explicit. In general, developers are doing what is appropriate; nevertheless, it is essential to stay vigilant, as attackers are known to change their attack plan frequently.

## 5.2. *Experimental results*

This experiment was performed on a Sony Xperia Z3 Compact device, model D5803. It is running Android Marshmallow, version 6.0.1 with latest updates. The device has 2 GB of RAM and 16 GB of storage.

We aim to answer the following research questions. A. Is Intent a plausible feature for Android malware detection? B. What are best configurations in Bayesian Network that produce the best results? C. How effective is Android Intent compared to Android permission?

### 5.2.1. *Effectiveness*

We employed Bayesian Network with different configurations for our experiment. As discussed earlier, Bayesian Network uses a search algorithm for calculating the local score metrics, and an estimator algorithm for learning the probability table. In order to achieve the best results, we experimented with different configurations, and the results are presented in Table 6. The table shows results of permission and Intent with simple estimator and multinomial estimator algorithms; and K2, Geneticsearch, HillClimber, and LAGDHillClimber as search algorithms.

The results of experiments reflect the performance of our method. Detection rate, also known as a true positive rate (TPR), is the probability of correctly detecting an instance as a malware. On the other hand, false positive rate (FPR) is another measurement that is defined as wrongly detecting normal traffic as being infected. The higher the TPR, the better is the result. Conversely, the lower the FPR, the better is the result. The best results are obtained by combining a simple estimator and Geneticsearch; and a simple estimator and LAGDHillClimber – both combinations achieving 83% true positive rate. We conducted our experiment in 30 iterations. As the number of iterations goes up, the system learns the pattern of the data more accurately. Fig. 6 shows the true positive rate and the false positive rate for each iteration of the experiment.

Fig. 6 shows that true positive rate increases from just above 80% to 90% as number of iterations goes up. However, false positive rate does not follow the same rate of increase as the true positive rate. It starts from 6% and increases to 9%, which is considered as a good result, considering that the true positive rate increases by 10%.

Additionally, we conducted experiments for each malware family to assess effectiveness of Android Intent for an individual family. The results are tabulated in Table 7. The experiments are conducted on families with highest number of malware samples in our dataset. Since our previous results with multinomial algorithm were not encouraging, we use simple estimator for this experiment. The lowest detection rate belongs to DroidKungfu family. This malware gains root access in the device and installs an application called legacy that pretends to be a legitimate Google Search application bearing the same icon. The DroidKungfu then performs its malicious activities through the legacy application (Jiang, 2011). We believe that such strategy makes it trickier to detect, since malicious

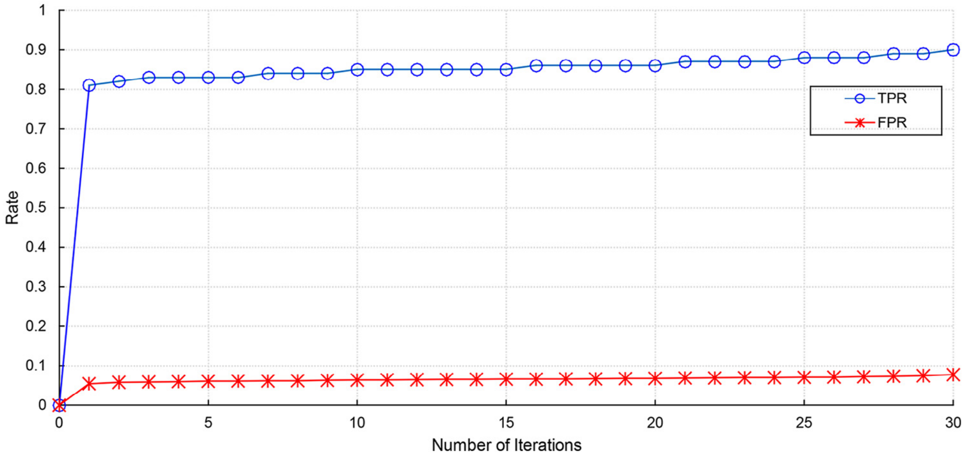

**Fig. 6 – True positive rate versus false positive rate for 30 iterations.**

**Table 7 – The results of Android Intent experiments for each malware family.**

| | | K2 | Geneticsearch | HillClimber | LAGD HillClimber | Number of malwares |
|---|---|---|---|---|---|---|
| FakeInstaller | TPR | 85.78% | 84.02% | 84.91% | 84.02% | 925 |
| | FPR | 14.21% | 15.97% | 15.08% | 15.97% | |
| DroidKungFu | TPR | 76.41% | 76.14% | 76.41% | 76.14% | 667 |
| | FPR | 23.58% | 23.85% | 23.58% | 23.85% | |
| Plankton | TPR | 79.59% | 79.59% | 79.34% | 79.54% | 625 |
| | FPR | 20.40% | 20.40% | 20.65% | 20.45% | |
| Opfake | TPR | 93.06% | 93.06% | 92.76% | 93.06% | 613 |
| | FPR | 6.93% | 6.93% | 7.23% | 6.93% | |
| GinMaster | TPR | 77.35% | 77.35% | 77.15% | 77.58% | 339 |
| | FPR | 22.64% | 22.64% | 22.84% | 22.41% | |
| BaseBridge | TPR | 81.96% | 81% | 83% | 80.17% | 330 |
| | FPR | 18.03% | 19% | 17% | 19.82% | |
| Iconosys | TPR | 76.74% | 76.87% | 76.74% | 76.87% | 152 |
| | FPR | 23.25% | 23.12% | 23.25% | 23.12% | |
| FakeDoc | TPR | 81.89% | 81.65% | 81.89% | 81.65% | 132 |
| | FPR | 18.10% | 18.34% | 18.10% | 18.34% | |
| Geinimi | TPR | 87.39% | 87.39% | 79.91% | 80.55% | 92 |
| | FPR | 12.60% | 12.60% | 20.08% | 19.44% | |
| | | | | | Total | 3875 |

activities are performed by an agent application other than the main one. Other malware families show relatively high to high detection results.

It is necessary to verify that Android Intent is in fact an effective feature, and our results are not just a fluke. Therefore, we conduct experiments using both features – Android permissions and Android Intents. This is essential to show that the features are not overlapping, and Android Intent can really increase the detection rate. Table 8 represents results of experiments on combination of Android Permissions and Android Intents. Not only are the results show that Android Intent – explicit and implicit – is an effective feature, it also boosts other features – i.e. Android permissions – in malware detection.

It is worth noting that the choice of Android permissions in this study is based on the fact that this feature has been widely explored and its importance and effectiveness has been established. Feizollah et al. (2015) conducted an extensive study on Android features. Among static features, Android permission is the most used features. Various approaches have been taken to analyze Android permissions. Authors of Au et al. (2012), Grace et al. (2012), Pandita et al. (2013), and Peng et al. (2012) used permissions to evaluate applications and rank them based on possible risk. Numerous studies simply extracted permissions and utilized machine learning to detect malicious application (Aung and Zaw, 2013; Samra et al., 2013; Sanz et al., 2013; Yerima et al., 2014). Researchers in Huang et al. (2013) and Moonsamy et al. (2013) argue that merely analyzing requested permissions is not sufficient for detecting malicious applications. They analyzed used permissions in addition to requested permissions in order to detect malware. AppGuard (Backes et al., 2013) has gone one step further and has extended Android's permission system to alleviate current vulnerabilities. They claim that their system is a practical extension for Android permission system as it is possible to use it on devices without any modification or root access. As a result, Android permission is a strong candidate for this paper in order to compare it with Android Intents.

**Table 8 – Results of experiments using both Permissions and Intents.**

| | Simple estimator | |
|---|---|---|
| | TPR | FPR |
| K2 | 95.5% | 4.4% |
| Geneticsearch | 95.4% | 4.5% |
| HillClimber | 95.5% | 4.4% |
| LAGDHillClimber | 95.4% | 4.5% |

### 5.2.2. *Efficiency*

Besides evaluating the effectiveness of our system, we calculated the time taken by each combination to produce the results, as shown in Table 9.

Based on Table 9, results in Android permission are produced faster when the simple estimator and HillClimber are combined. With regard to Android Intent, combining the simple

**Table 9 – Time taken to produce results (seconds).**

| | Android permission | | Android intent | |
|---|---|---|---|---|
| | Simple estimator | Multinomial | Simple estimator | Multinomial |
| K2 | 0.06 | 0.89 | 0.01 | 0.07 |
| Geneticsearch | 2.86 | Null | 0.91 | Null |
| HillClimber | 0.02 | 0.87 | 0.01 | 0.07 |
| LAGDHillClimber | 0.05 | Null | 0.05 | Null |

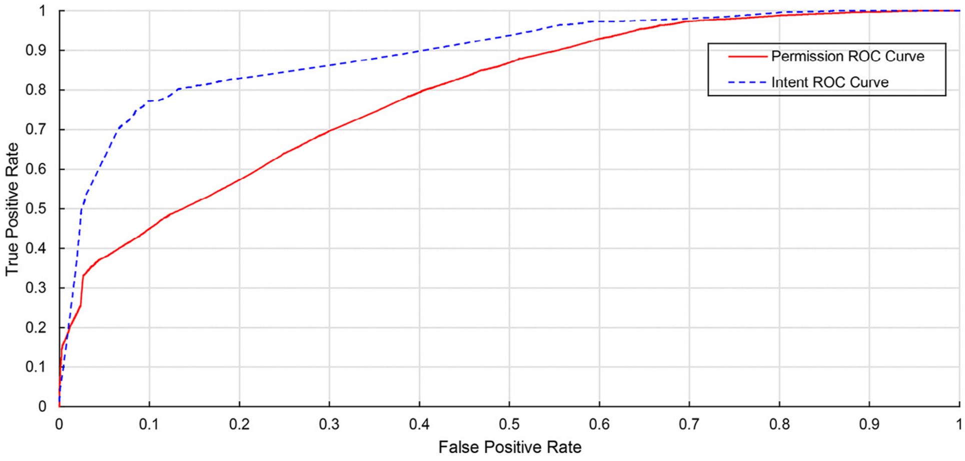

**Fig. 7 – ROC curve for Android Permission and Android Intent.**

estimator with LAGDHillClimber achieved true positive rate of 91% in less time than Geneticsearch.

In addition, we show the Receiver Operating Characteristic (ROC) curve for the best results of permission and Intent. The ROC curve is normally used to measure performance in detecting intrusions. It indicates how the detection rate changes, as the internal threshold is varied to generate more or fewer false alarms. It plots intrusion detection accuracy against false positive probability. ROC curves signify the tradeoff between false positive and true positive rates, which means that any increase in the true positive rate is accompanied by a decrease in the false positive rate. As the ROC curve line is closer to the left-hand border and the top border, it indicates that it produces the best results among other curves. The ROC curves for Android permission and Android Intent are shown in Fig. 7.

The ROC curves are difficult to compare, as they seem to be almost similar under some situations, therefore, the area under the curve (AUC) is used to measure the accuracy of detection. An area of 1 means a perfect result, while an area of 0.5 is a worthless result. The AUC point system is as follows: 0.90–1.00 = excellent (A); 0.80–0.90 = good (B); 0.70–0.80 = fair (C); 0.60–0.70 = poor (D); and 0.50–0.60 = fail (F). The AUC of Android permissions is 0.7897, and Android Intent is 0.8929. This shows that Android Intent performed better.

The nature of AndroDialysis raises concerns about battery consumption of the device. Running our malware detector on the device does not consume too much battery. To address this issue, we measured power consumed by our application. Additionally, the measurement is performed for three popular applications. These applications are selected from three categories of popular activities: games, online social networking, and multimedia (Suarez-Tangil et al., 2015).

The experiments have been conducted in a Google Nexus One smartphone. Power consumption has been measured by applying a battery tests involving mainly computation capabilities. The device was previously instrumented with AppScope (Yoon et al., 2012), an energy metering framework based on monitoring kernel activity for Android. AppScope collects usage information from the monitored device and estimates the consumption of a running application using an energy model given by DevScope (Jung et al., 2012). AppScope provides the amount of energy consumed by an app in the form of several time series, each one associated with a component of the device – CPU, Wi-Fi, cellular, touchscreen, etc. We restrict our measures to CPU for computations, as our tests do not have communications nor a graphical user interface at computation stage. Note that we do not require user interaction to analyze applications and, therefore, do not report measurements in any other component.

Table 10 shows outcomes of the measurement during 10 minutes of usage. The AndroDialysis consumes 23.25 joules for testing one application on the device. Thus, we assume a number N = 20 for the average number of applications a user has on the device and multiply N by 23.25 joules. We have to mention that this is subject to the size of applications, and that although there might be larger apps, this measurement still gives an estimation of the power consumption.

It is necessary to discuss re-running time. The AndroDialysis should only be executed every time a user installs a new application. Thus, if a user installs 20 applications in a period of one month, our tool would consume 20 × 23.25 = 465 joules after a month, which is less than running YouTube application during 10 minutes. Fig. 8 shows power consumption of AndroDialysis in Watt unit. It is worth mentioning that Joules unit is calcu-

**Table 10 – Power consumption (in Joules) of three popular applications and AndroDialysis during 10 minutes usage.**

| Application | CPU | Communications | Display | Total |
|---|---|---|---|---|
| YouTube | 30.11 | 12.59 | 508.90 | 551.59 |
| MX Moto | 129.24 | 5.75 | 509.54 | 644.52 |
| Facebook | 137.76 | 27.42 | 471.42 | 637.27 |
| AndroDialysis | 23.25 | 0 | 0 | 465 (23.25 × 20) |

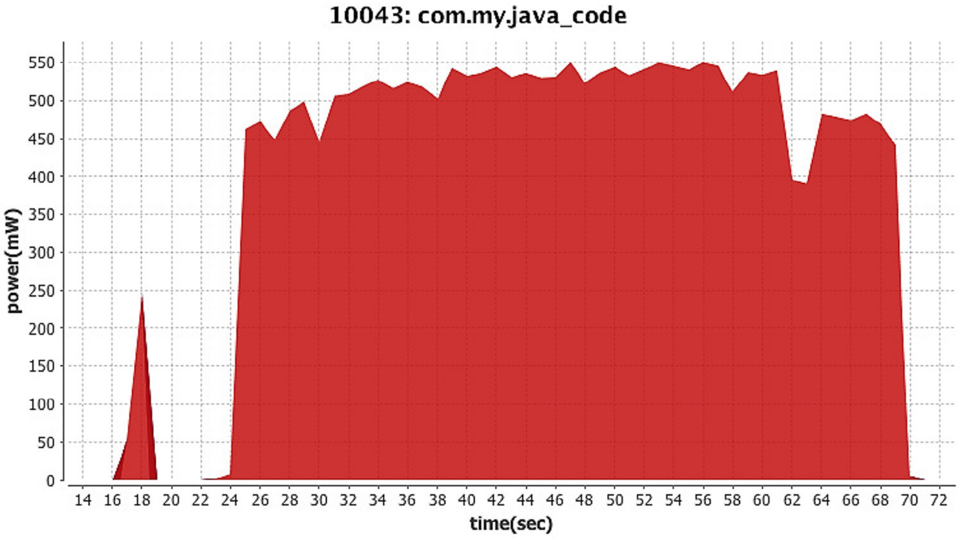

**Fig. 8 – Power consumption of AndroDialysis in our experiment.**

lated using $E_{(j)} = P_{(w)} \times t_{(s)}$ equation, where unit of power is Watt and unit of time is seconds.

## 6. Related works

Many studies have been conducted to address the problem of the rapid growth of mobile malware. We discuss recent researches related to this paper.

Barrera et al. (2010) performed permission-based analysis on 1100 Android applications using self-organizing maps algorithm. From the results, they observed that certain permissions are used in applications with similar pattern. They also concluded that there are pairs of permissions requested by some types of applications. They mentioned, however, that their analysis does not include any malware, and they had merely examined available applications in the market. Zhou et al. (2012) conducted permission-based analysis, and the results show a high false positive rate of 40%. As a result, manual analysis was conducted to reduce the false positive rate. Grace et al. (2012) developed RiskRanker that ranks applications based on certain defined rules. If an application satisfies a rule, it is ranked as high, medium, or low, as the case may be. Similarly, Chakradeo et al. (2013) introduced MAST that uses multiple correspondence analysis (MCA) to analyze the applications attributes. They used a questionnaire and poll selection to identify malicious applications. RiskRanker and MAST employed rules and polls to detect malware. When an unknown malware appears, they need to add a new rule and poll to detect it. The rules might not be applicable to all known malware, as there are too many malwares in existence.

Some researchers integrated Intent as examining features in their systems. Intent is one of eight feature sets that DREBIN (Arp et al., 2014) extracts for examination. It used machine-learning methods on feature sets to detect malicious applications. A3 (Luoshi et al., 2013) is another system that mentioned Intent as one of three extracted features. It utilized heuristic algorithm for detection. DroidMat (Wu et al., 2012) includes Intent in the feature sets. It extracts permission, API calls, Intent, as well as performs deployment of components. It then employs various machine-learning methods to evaluate applications and identify malicious ones. MAST (Chakradeo et al., 2013) includes Intent extracted from AndroidManifest.xml file as its examining feature. As mentioned above, implicit and explicit Intents are as important as Intent in XML file.

Chin et al. (2011) developed a system that analyzes inter-application communications (includes explicit and implicit Intents) in developers' applications for analyzing and detecting malware. They also guide developers on using Intents correctly to avoid attacks – application hijacking. Apposcopy (Feng et al., 2014) is a malware detection tool that integrates static taint analysis and Intent analysis (explicit and implicit) to generate a signature for applications. However, this system adopts a signature-based approach that is unable to detect unknown malware. Octeau et al. (2015) tried to solve the problem of Multi-Valued Composite (MVC) constant propagation. They used COAL declarative language to build a solver to find all the values of complex objects that may have multiple fields, taking into consideration the correlations between the fields. This method can be applied to a wide variety of static program analyses where the range of values of objects needs to be determined, including Android Intent. However, attackers can simply modify their code and use various methods of obfuscation to mislead such systems. IccTA (Li et al., 2015) analyzes inter-component communications in Android applications. They include explicit and implicit Intents, since they are essential part of Android's internal communication mechanisms. They focus on detecting applications with privacy leaks using data flow analysis. Although this approach outperforms similar systems, it is unable to analyze the multi-threading part of applications. It also consumes too much memory for analyzing some applications. Barros et al. (2015) analyzed data flow of Intent in Android by using pattern of Android Intent in Java code as well as the syntax and semantics of Intent types. Since their work is dependent on data flow analysis, it is not immune to obfuscation methods. This approach pays little attention to analyzing explicit and implicit Intents; nevertheless, we believe that it is very effective for malware detection. In this paper, we use intelligent learner for detection. In this context, we extracted and used permission, explicit Intent and implicit Intent from a large dataset to produce accurate results.

## 7. Conclusions

In this paper, we explored Android Intent – explicit and implicit – as a feature for malware detection, and experimented with Android permission for comparison. The results show that the use of Android Intent in our approach not only achieves higher detection rate, but it is also faster in completing the detection process. We also verified our results by experimenting on combination on Android Intent and Android permission, to show that these features do not overlap. Thus, to answer the first question, Android Intent is a plausible feature in malware detection. In addition, combining the simple estimator with LAGDHillClimber is the best configuration for Bayesian Network algorithm to achieve higher detection rate and faster detection. In conclusion, we declare that Android Intent is indeed more effective than Android permission in malware detection. As a result of this work, it behooves researchers to emphasize on Android Intents (explicit and implicit) for mobile malware detection. It is beneficial to develop new detection methods as attackers change their strategy frequently to avoid the current detection methods.

We are determined to develop comprehensive methods based on this work in conjunction with dynamic analysis to tackle mobile malware. In addition, the graphical user interface will be improved to show list of applications that are considered malware, and why our application considers it malicious. This way, the AndroDialysis learns about applications, which makes it smarter. Additionally, the user will be presented with options on how to deal with malicious applications.

## Acknowledgments

This work was supported by the Ministry of Science, Technology, and Innovation, under Grant eScienceFund 01-01-03-SF0914.

## REFERENCES

Aafer Y, Du W, Yin H. DroidAPIMiner: mining API-level features for robust malware detection in Android, Proceedings of the 9th international conference on security and privacy in communication networks, Vol. 127, Sydney, Australia, pp. 86–103; 2013.

Aftab MUB, Karim W. Learning android intents. Packt Publishing; 2014.

Aresu M, Ariu D, Ahmadi M, Maiorca D, Giacinto G. Clustering Android malware families by http traffic, Proceedings of the 10th international conference on malicious and unwanted software, Puerto Rico; 2015.

Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. DREBIN: effective and explainable detection of Android malware in your pocket, Proceedings of the 2014 network and distributed system security (NDSS) symposium, San Diego, USA; 2014.

Au KWY, Zhou YF, Huang Z, Lie D. Pscout: analyzing the android permission specification, Proceedings of the 2012 ACM conference on computer and communications security, Raleigh, NC, USA, pp. 217–228; 2012.

Aung Z, Zaw W. Permission-based android malware detection. Int J Sci Technol Res 2013;2(3):228–34.

Backes M, Gerling S, Hammer C, Maffei M, Styp-Rekowsky P. AppGuard: enforcing user requirements on android apps, Proceedings of the 19th international conference on tools and algorithms for the construction and analysis of systems, Rome, Italy, pp. 543–548; 2013.

Barrera D, Kayacik HG, Oorschot P, Somayaji A. A methodology for empirical analysis of permission-based security models and its application to android, Proceedings of the 17th ACM conference on computer and communications security, Chicago, Illinois, USA, pp. 73–84; 2010.

Barros P, Just R, Millstein S, Vines P, Dietl W, d'Amorim M, et al. Static analysis of implicit control flow: Resolving Java reflection and Android intents (extended version), University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-15-08-01; 2015.

Bielza C, Larrañaga P. Discrete Bayesian network classifiers: a survey. ACM Comput Surv (CSUR) 2014;47(1):5.

Chakradeo S, Reaves B, Traynor P, Enck W. MAST: triage for market-scale mobile malware analysis, Proceedings of the sixth ACM conference on security and privacy in wireless and mobile networks, Budapest, Hungary, pp. 13–24; 2013.

Chickering D, Geiger D, Heckerman D. Learning Bayesian networks: Search methods and experimental results, Proceedings of the fifth conference on artificial intelligence and statistics, Florida, USA, pp. 112–128; 1995.

Chin E, Felt AP, Greenwood K, Wagner D. Analyzing inter-application communication in Android, Proceedings of the 9th international conference on mobile systems, applications, and services, Bethesda, Maryland, USA, pp. 239–252; 2011.

Cohen I, Sebe N, Gozman FG, Cirelo MC, Huang TS. Learning Bayesian network classifiers for facial expression recognition both labeled and unlabeled data, Proceedings of the 2003 IEEE computer society conference on computer vision and pattern recognition, Vol. 1, Wisconsin, USA, pp. I-595-I-601 vol.591; 2003.

Damopoulos D, Menesidou SA, Kambourakis G, Papadaki M, Clarke N, Gritzalis S. Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers. Secur Commun Netw 2012;5(1):3–14.

Desnos A. Android: static analysis using similarity distance, Proceedings of the 2012 45th Hawaii international conference on system science (HICSS), Maui, USA, pp. 5394–5403; 2012.

Feizollah A, Anuar NB, Salleh R, Amalina F, Ma'arof R, Shamshirband S. A study of machine learning classifiers for anomaly-based mobile botnet detection. Malaysian J Comput Sci 2013;26(4):251–65.

Feizollah A, Anuar NB, Salleh R, Wahab AWA. A review on feature selection in mobile malware detection. Digit Invest 2015;13(C):22–37.

Feng Y, Anand S, Dillig I, Aiken A. Apposcopy: semantics-based detection of Android malware through static analysis, Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering, Hong Kong, China, pp. 576–587; 2014.

Fratantonio Y, Bianchi A, Robertson W, Kirda E, Kruegel C, Vigna G. TriggerScope: towards detecting logic bombs in android applications, Proceedings of the IEEE security & privacy, San Jose, California, USA; 2016.

Friedman N, Geiger D, Goldszmidt M. Bayesian network classifiers. Mach Learn 1997;29(2–3):131–63.

Gartner. PC shipments hit by biggest drop in two years; 2015. Available from: http://www.techradar.com/us/news/computing/pc/pc-shipments-hit-by-biggest-drop-in-two-years. [Accessed 1 April 2016].

Google. Permission; 2014. Available from: http://developer.android.com/guide/topics/manifest/permission-element.html. [Accessed 1 April 2016].

Grace M, Zhou Y, Zhang Q, Zou S, Jiang X. RiskRanker: scalable and accurate zero-day android malware detection, Proceedings of the 10th international conference on mobile systems, applications, and services, Low Wood Bay, Lake District, UK, pp. 281–294; 2012.

Hellman E. Android programming: pushing the limits. John Wiley & Sons; 2013.

Huang C-Y, Tsai Y-T, Hsu C-H. Performance evaluation on permission-based detection for Android malware, Proceedings of the international computer symposium ICS, Hualien, Taiwan, pp. 111–120; 2013.

Jain K. Warning: 18,000 Android apps contains code that spy on your text messages; 2015. Available from: http://thehackernews.com/2015/10/android-apps-steal-sms.html. [Accessed 1 April 2016].

Jiang X. New sophisticated Android malware DroidKungFu found in alternative Chinese app markets; 2011. Available from: https://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html. [Accessed 1 November 2016].

Jiang X, Zhou Y. Android malware. New York: Springer; 2013.

Jo NY, Lee KC, Park B-W. Exploring the optimal path to online game loyalty: Bayesian networks versus theory-based approaches. In: Ubiquitous computing and multimedia applications. Springer; 2011. p. 428–37.

Jung W, Kang C, Yoon C, Kim D, Cha H. DevScope: a nonintrusive and online power analysis tool for smartphone hardware components, Proceedings of the eighth international conference on hardware/software codesign and system synthesis (IEEE/ACM/IFIP), Scottsdale, AZ, USA, pp. 353–362; 2012.

Larrañaga P, Poza M, Yurramendi Y, Murga RH, Kuijpers CM. Structure learning of Bayesian networks by genetic algorithms: a performance analysis of control parameters. IEEE Trans Pattern Anal Mach Intell 1996;18(9):912–26.

Li L, Bartel A, Bissyandé TF, Klein J, Le Traon Y, Arzt S, et al. IccTA: detecting inter-component privacy leaks in Android apps, Proceedings of the 37th international conference on software engineering-volume 1, pp. 280–291; 2015.

Luoshi Z, Yan N, Xiao W, Zhaoguo W, Yibo X. A3: automatic analysis of Android MALWARE, Proceedings of the 1st international workshop on cloud computing and information security, Shanghai, China, pp. 89–93; 2013.

Moonsamy V, Rong J, Liu S. Mining permission patterns for contrasting clean and malicious android applications. Future Gen Comput Syst 2013;36:122–32. Available from: http://dx.doi.org/10.1016/j.future.2013.09.014, http://www.sciencedirect.com/science/article/pii/S0167739X13001933. [Online].

Narudin FA, Feizollah A, Anuar NB, Gani A. Evaluation of machine learning classifiers for mobile malware detection. Soft Comput 2016;20(1):343–57.

Oberheide J, Miller C. Dissecting the android bouncer, Proceedings of the SummerCon, New York, USA; 2012.

Octeau D, Luchaup D, Dering M, Jha S, McDaniel P. Composite constant propagation: Application to android inter-component communication analysis, Proceedings of the 37th international conference on software engineering-volume 1, pp. 77–88; 2015.

Pandita R, Xiao X, Yang W, Enck W, Xie T. WHYPER: towards automating risk assessment of mobile applications, Proceedings of the 22nd USENIX security symposium, Washington, D.C, USA, pp. 527–542; 2013.

Peng H, Gates C, Sarma B, Li N, Qi Y, Potharaju R, et al. Using probabilistic generative models for ranking risks of Android apps, Proceedings of the 2012 ACM conference on computer and communications security, Raleigh, North Carolina, USA, pp. 241–252; 2012.

Polkovnichenko A, Boxiner A. A new level of sophistication in mobile malware; 2015. Available from: http://blog.checkpoint.com/2015/09/21/braintest-a-new-level-of-sophistication-in-mobile-malware. [Accessed 1 April 2016].

Richardson L. Beautiful soup documentation; 2007. Available from: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed 1 April 2016].

Ruiz C. Illustration of the K2 algorithm for learning Bayes net structures, Worcester Polytechnic Institute; 2005. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.190.7306. [Accessed 1 April 2016].

Salehi E, Gras R. An empirical comparison of the efficiency of several local search heuristics algorithms for Bayesian network structure learning, Proceedings of the learning and intelligent optimization workshop (LION 3), Vol. 72; 2009.

Samra AAA, Yim K, Ghanem OA. Analysis of clustering technique in Android malware detection, Proceedings of the 2013 seventh international conference on innovative mobile and internet services in ubiquitous computing (IMIS), Taichung, Taiwan, pp. 729 - 733; 2013.

Sanz B, Santos I, Laorden C, Ugarte-Pedrero X, Bringas P, Álvarez G. PUMA: permission usage to detect malware in Android, in International joint conference CISIS'12-ICEUTE'12-SOCO'12 special sessions, Springer Berlin Heidelberg, pp 289–298; 2013.

Shabtai A, Tenenboim-Chekina L, Mimran D, Rokach L, Shapira B, Elovici Y. Mobile malware detection through analysis of deviations in application network behavior. Comput Secur 2014;43(June 2014):1–18.

Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Blasco J. Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families. Exp Syst Appl 2014;41(4 Pt 1):1104–17.

Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Pastrana S. Power-aware anomaly detection in smartphones: an analysis of on-platform versus externalized operation. Perv Mobile Comput 2015;18(April 2015):137–51.

Tam K, Khan SJ, Fattori A, Cavallaro L. CopperDroid: automatic reconstruction of android malware behaviors, Proceedings of the network and distributed system security symposium (NDSS), San Diego, USA; 2015.

Wei X, Gomez L, Neamtiu I, Faloutsos M. ProfileDroid: multi-layer profiling of android applications, Proceedings of the 18th annual international conference on mobile computing and networking, Istanbul, Turkey, pp. 137–148; 2012.

Winsniewski R. Android–apktool: A tool for reverse engineering android apk files; 2012. Available from: http://ibotpeaches.github.io/Apktool/. [Accessed 1 April 2016].

Wu D-J, Mao C-H, Wei T-E, Lee H-M, Wu K-P. DroidMat: Android Malware Detection through Manifest and API Calls Tracing, Proceedings of the seventh asia joint conference on information security (Asia JCIS), Tokyo, Japan, pp. 62–69; 2012.

Yan LJ, Cercone N. Bayesian network modeling for evolutionary genetic structures. Comput Math Appl 2010;59(8):2541–51.

Yang C, Xu Z, Gu G, Yegneswaran V, Porras P. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications, Proceedings of the 19th European symposium on research in computer security, Wroclaw, Poland; 2014.

Yerima SY, Sezer S, McWilliams G. Analysis of Bayesian classification-based approaches for Android malware detection. IET Inf Secur 2014;8(1):25–36.

Yoon C, Kim D, Jung W, Kang C, Cha H. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring, Proceedings of the 2012 USENIX annual technical conference (USENIX ATC 12), Boston, USA, pp. 387–400; 2012.

Zhang Y, Yang M, Xu B, Yang Z, Gu G, Ning P, et al. Vetting undesirable behaviors in android apps with permission use analysis, Proceedings of the 2013 ACM SIGSAC conference on computer & communications security, Berlin, Germany; 2013.

Zhou Y, Wang Z, Zhou W, Jiang X. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets, Proceedings of the 19th annual network and distributed system security symposium (NDSS), San Diego, USA, pp. 5–8; 2012.

Ali Feizollah received his Bachelor of Information System (IS) from the Ajman University of Science and Technology (AUST), Ajman, UAE in 2010. He started his Master of Computer Science at the University of Malaya, Kuala Lumpur in 2011. He started his Ph.D. in the same university in 2014. His research interests are mobile malware, intrusion detection system.

Nor Badrul Anuar obtained his Ph.D. in Information Security from Centre for Security, Communications and Network Research (CSCAN), Plymouth University, UK in 2012 and Master of Computer Science from University of Malaya, Malaysia in 2003. He is an academic staff at the Faculty of Computer Science and Information Technology in University of Malaya, Kuala Lumpur. He has published a number of conference and journal papers locally and internationally. His research interests include information security (i.e. intrusion detection systems), artificial intelligence and library information systems.

Rosli Bin Salleh received his BS in computer science from the University of Malaya, Malaysia, in 1994, and his MS and Ph.D. from the University of Salford, United Kingdom, in 1997 and 2001, respectively. From 2001, he worked as a lecturer in the Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya. He was appointed as a senior lecturer in 2007 and as an associate professor in 2013. His research interests include Mobile IPv6 handover and security, botnet research, and wireless sensor networks.

Guillermo Suarez-Tangil is Research Assistant at the Systems Security Research Lab(S2Lab) within the world leading Information Security Group (ISG) at Royal Holloway University of London (RHUL). Prior to joining RHUL, he was Teaching Assistant at Carlos III University of Madrid, Spain, where he obtained a Ph.D. with distinction in the Computer Security (COSEC) and a M.Sc and a B.Sc in Computer Sciences at the same university. There, he graduated with honours and received the Best Student Academic Award. His main research interests are in computer/network security and his current research focuses on security in smart devices, intrusion detection, event correlation, and cyber security. He has participated in various research projects related to network security and trusted computing.

Steven Furnell received a Ph.D. degree in information system security from the University of Plymouth in 1995, and now is the Head of the Centre for Security, Communications and Network Research at Plymouth University, U.K., and an Adjunct Professor with Edith Cowan University, Perth, Australia. Prof. Furnell is the author of over 250 papers in refereed international journals and conference proceedings, as well as books including Cybercrime: Vandalising the Information Society (2001) and Computer Insecurity: Risking the System (2005). His interests include security management and culture, computer crime, user authentication, and security usability. Further details can be found at www.plymouth.ac.uk/cscan, with a variety of security podcasts also available via www.cscan.org/podcasts.