Exploring the usage of Topic Modeling for Android Malware Static Analysis

Eric Medvet*, Francesco Mercaldo†
*Department of Engineering and Architecture, University of Trieste, Trieste, Italy
emedvet@units.it
†Department of Engineering, University of Sannio, Benevento, Italy
fmercaldo@unisannio.it

***Abstract*—The rapid growth in smartphone and tablet usage over the last years has led to the inevitable rise in targeting of these devices by cyber-criminals. The exponential growth of Android devices, and the buoyant and largely unregulated Android app market, produced a sharp rise in malware targeting that platform. Furthermore, malware writers have been developing detection-evasion techniques which rapidly make anti-malware technologies ineffective. It is hence advisable that security expert are provided with tools which can aid them in the analysis of existing and new Android malware.**

In this paper, we explore the use of topic modeling as a technique which can assist experts to analyse malware applications in order to discover their characteristic. We apply Latent Dirichlet Allocation (LDA) to mobile applications represented as opcode sequences, hence considering a topic as a discrete distribution of opcode. Our experiments on a dataset of 900 malware applications of different families show that the information provided by topic modeling may help in better understanding malware characteristics and similarities.

I. Introduction

Worldwide, the 66 percent of users were using an Android device in the fourth quarter of 2015. G DATA analysts identified 758,133 new Android malware files in the same period[1]: that is an increase of almost 32% compared to the third quarter (574,706). In the second half of 2015, 1,332,839 new malware apps were discovered in total. For 2015 as a whole, a new record of 2,333,777 malware files for the Android operating system alone has been set[2]—more than 50% than 2014.

The rapid increase underlines the significance of the profit from mobile operating systems, especially Android: there is a potential in mobile environment for high financial gains for malware writers. Mobile malware professionals are maximizing their return on investment by targeting Android because of its global market dominance and open platform. Like legitimate business people, malware professionals look to exploit the largest addressable market opportunity.

There are mainly two successful monetization schemes seen in a recent spate of Android malware: information gathering and Premium Rate Number Billing. Relating to information gathering, several Android applications exist that allow someone to track and monitor a user of a mobile phone. For example, these applications may record and export all SMS messages, emails, call logs, GPS locations, or turn on the microphone. Typically, these applications require an attacker to purchase the application from the vendor and then gain physical access to the phone. While these applications may not generate revenue for the attacker, they generate revenue for the vendor of the spyware application. Examples include Android.Tapsnake and Spyware.Flexispy. Such applications can sell for $400 and some of them are available on the Android Market.

Some of these applications also exhibit advanced capabilities such as recording phone calls. However, to enable some of these features, the phone must be rooted. Nevertheless, without rooting the device, data can still be obtained by requesting standard permissions. Android.Tapsnake is an example of spyware that pretends to just be a game of snake, actually including a fully functional copy of the game. However in the background, the application is uploading the GPS coordinates of the device every 15 minutes.

It is clear that malware applications vary in their malicious payload, as well as in the mechanism that actually triggers the execution of that payload and in aside damages. This wealth of different characteristic may negatively affect the effectiveness of fully automated malware detection techniques. Indeed, previous studies demonstrated that current anti-malware tools relying on signature-based detection mechanisms are not robust against code transformations and highlighted the need for anti-malware tools to find out new techniques for mobile malware detection [1, 2]. The cited works evaluate a set of well-known antimalware software using original and transformed Android malware samples, in order to alter the signature without alter the payload business logic. They conclude that all the antimalware products are susceptible to common evasion techniques, like code reordering, junk code insertion or data encryption.

Google, with the introduction of Bouncer[3], tries to mitigate the problem but attackers write malware which is more and more aggressive and is able to easily evade the mechanism. Bouncer executes the application in a sandbox for a fixed-time window before publish it on the official market[4]: it is clear that

[1]https://www.gdatasoftware.com/securitylabs/news/article/android-g-data-analyses-over-4500-new-malware-instances-per-day

[2]https://public.gdatasoftware.com/Presse/Publikationen/Malware_Reports/US/G_DATA_MobileMWR_Q4_2015_US.pdf

[3]http://googlemobile.blogspot.it/2012/02/android-and-security.html

[4]https://play.google.com/store

if the malware action do not happen over this interval, Bouncer can not detect the malicious event.

In this challenging scenario, manual analysis by expert security operators still plays a crucial role. On the other hand, due to the nature of the artifact that have to be analyzed, some degree of automation is advisable to allow an analysis of higher level and, from another point of view, to refine the raw information given by the malware application itself. For these reasons, in this paper we propose a methodology to assist the expert to analyze the malware app which is based on topic modeling. In particular, we apply the Latent Dirichlet Allocation (LDA) [3] to malware applications represented as sequences of opcodes—i.e., we propose a flavor of static analysis.

LDA is a well established topic modeling technique: from the LDA point of view, each document exhibits multiple topics with different probabilities and each topic exhibits multiple words, with different probabilities. As a simple example, consider a LDA model applied on a small corpus with two topics: cat-related and dog-related. The former topic is characterized by probabilities of generating various words: words such as "milk", "meow", and "kitten", which could be classified and interpreted by the viewer as cat-related, will have high probability. The dog-related topic likewise has different word probabilities: e.g., "puppy", "bark", and "bone" might have high probability. Words without special relevance, such as function words, will have roughly even probability between classes (or can be placed into a separate category). In this paper, we explore the usage of LDA to analyse Android malware and experimentally show that topic modeling does provide more information about malware characteristics with respect to simply considering opcode occurrences or frequencies.

The remainder of the paper is organized as follows: Section II discusses related work; Section III presents and characterizes the real world dataset involved in the study; Section IV describes our methodology and illustrates the results of our experiments; finally, Section V draws the conclusions.

II. Related Work

To the best of our knowledge, no previous studies have been done on the usage of topic modeling for aiding the analysis of Android malware, in particular on hundreds of malware applications belonging to different families, as in our work. However, LDA has been used previously as a tool for augmenting detection or classification techniques in other similar security-related fields. We here provide a brief survey of those studies and of significant recent studies about Android malware detection.

Authors of [4] propose a probabilistic model to detect Microsoft Windows malware that jointly models sequential data and class labels (trusted/malware) using dynamic analysis. Applying LDA, they reveal behavior patterns and exploit them to predict class labels of unknown sequences, obtaining an accuracy very close to 70%.

Authors of [5] used LDA and data mining methods for metamorphic malware detection. Using LDA, they could rank opcode bi-gram features for classifying benign and malware files. The accuracy obtained for NGVCK metamorphic malware family was about 99.7% using a dataset composed by 867 samples belonging to NGVCK family and $1{,}318$ bening executables collected from different Internet sources.

Researchers in [6] address the problem of detecting mutated generations; they propose a method based on Opcode Graph Similarity to detect metamorphic malware using the similarity of opcode graphs. According to the authors, all nodes and edges have an effect on classification, but their method benefits of the pruning of edges using LDA. The proposed method yielded 100% total accuracy for NGVCK and 99% accuracy for MWOR malware family proving that the method is efficient for metamorphic malware detection.

Neuhaus et al. [7] crawl the vulnerability reports in the Common Vulnerability and Exposures database by using topic models to find prevalent vulnerability types and new trends semi-automatically. They analyze $39{,}393$ unique reports until the end of 2009, with the aim of characterizing many vulnerability trends: SQL injection (PHP), buffer overflows, format strings, cross-site request forgery and so on.

Zhao et al. [8] propose a LDA-based method to analyze the trends of network security which consist of three steps: collect data from web sites, extract topics from the collected data, and makes the curves of trends over time. They select 620 documents sorted by time and extract 10 topic from each document. Six interesting topics are discovered by LDA model, according to the autors: dns-ddos, vulnerability, mobile-malware, mac-malware, Browser malware and Java-vulnerability.

Tsai et al. [9] apply LDA to blog data: they analyze blogs for various categories of cyber threats related to the detection of security threats, cyber crime and information security in order to identify patterns of similarities in keywords and dates distributed across blog documents. Basically the proposed method extracts the most relevant categories and thus the topics extracted for each category. The experiment shows that the probabilistic model can reveal interesting pattern in the underlying topics for the considered freely-available dataset of security-related blogs.

Wang et al. [10] conduct a preliminary investigation of Twitter-based criminal incident prediction. Their approach is based on the automatic semantic analysis and understanding of natural language Twitter posts, applying LDA to identify salient topics within the extracted events, and then building a predictive model upon these latent topics. They apply the method on $3{,}659$ tweets published during the period of February 22, 2011 through October 21, 2011. Evaluation results demonstrate the ability of the model to forecast hit-and-run crimes using only the information contained in the training set of tweets.

Concerning Android malware detection, several different approaches have been proposed in the recent past, due to the rising trend in malware spreading: the reader may find in [11, 12] two extensive and recent surveys about the techniques for securing the Android platform. We here limit our ananlysis to some recent proposals which base on different respects:

metadata (such as permissions) analysis, static analysis, and dynamic analysis.

Song et al. [13] propose a framework to statically detect Android malware, consisting of four layers of filtering mechanisms: the message digest values, the combination of malicious permissions, the dangerous permissions, and the dangerous intention. As additional contribute, they propose a novel threat degree threshold model of dangerous permissions on malware detection. They experiment the method on real mobile devices, using 83 real mobile devices and achieving a 98.8% pass rate, where the versions of Android range from 2.3 to 5.1.

Authors in [14, 15, 16] evaluate the effectiveness of the occurrences of a subset of opcodes (i.e., move, if, jump, switch and goto) in order to discriminate mobile malware applications from trusted ones. They apply six classification algorithms (J48, LADTree, NBTree, RandomForest, RandomTree and RepTree), obtaining a precision equal to 0.949 in malware identification.

Researchers in [17] investigate whether frequencies of $n$grams of opcodes are effective in detecting Android malware, evaluating their method using $11{,}120$ applications, $5{,}560$ of which are malware belonging to several different families. They obtain an accuracy of 97% on the average, whereas perfect detection rate is achieved for more than one malware family.

Canfora et al. [18] propose a method for detecting Android malware which is based on the analysis of system calls sequences. Experimentation on $20{,}000$ execution traces of $2{,}000$ applications ($1{,}000$ of them being malware belonging to different malware families), performed on a real device, obtains a detection accuracy of 97%.

Martinelli et al. [19] propose a framework for classifing Android Malware using subgraphs of system calls. They collected more than 500 distinct runs from 13 malicious apps and 7 good ones in order to test the effectiveness of their solution, obtaining the best result with the K-nearest neighbors classifier.

III. Data

In this work, we consider a dataset composed of 900 Android malware applications which we obtained from Drebin Dataset [20, 21]. The 900 applications differ in nature and malicious intents (premium call & SMS, selling user information, advertisement, SMS spam, stealing user credentials, ransom) and can be partitioned in 49 malware families. Malware applications in the same family share some *malware features*, i.e., they are similar with respect to some malware characteristics.

- *BaseBridge* payload is able to receive premium numbers from remote command-and-control (C&C) servers and dial calls or send out SMS messages to them, incurring fees for users. It is also able to kill the processes of anti-malware application running in background. The malware sends personal information to C&C server running several malicious services in background.
- *DroidKungFu* payload is included in applications made available through alternative app markets and forums targeting Chinese-speaking users. The payload adds into the infected app a new service and a new receiver. The receiver is notified when the system finishes the boot so that it can automatically launches the service without user's interaction. This malware encrypts two root exploits: exploid and rage against the cage (see below).
- *FakeDoc* malware is able to disguise itself as a common application in order to trick the user. It blocks the user from receiving SMS from certain numbers. It also leak out user privacy by sending out the personal information.
- The samples of *FakeInstaller* family send SMS to premium-rate numbers, without the user's consent: the user is forced to click an Agree or Next button, which sends the premium SMS, but there are variants that send the messages before the victim clicks a button. They also include a backdoor to receive commands from a remote server.
- *GinMaster* payload launches a malicious service able to root the infected devices in order to obtain administrator privileges, steal confidential information and send to a remote website. It is very close to the DroidKungFu payload: the malware starts its malicious services as soon as it receives a `BOOT_COMPLETED` or `USER_PRESENT` intent. The malware make use of polymorphic techniques to evade detection by current antimalware technologies (e.g., hiding malicious code, obfuscating class names, randomizing package names and self-signing certificates).
- The purpose of *Iconosys* malware is to block unwanted phone calls from certain individuals by giving the owner of the phone the ability to play any sound or pre-recorded message to an unwanted caller. The user can configure the application to play a pre-recorded message or sound to make the caller believe that the phone is disconnected or out of service, while the malware sends out personal information like phone number, IMEI, IMSI and email address. Additionally, the application asks the user if they liked the app. Unfortunately, regardless of what the user does, an international rate SMS message, which is not free, is queued for sending to a number located in India.
- *Kmin* malware may pose as an application named KMHome: the payload sends device ID, subscriber ID and current time to a remote server named `su.5k3g.com`. It is very closed to BaseBridge family, but it does not kill antimalware processes.
- When an app infected with *Plankton* payload is installed, a service is launched in the background. The service communicates to a hard-coded HTTP server. The server replies with a URL that is used to download a JAR file that represents the malicious payload. The downloaded archive launches a connection to the C&C server and listens for commands to execute. Additionally the payload forwards personal details and browser history to the remote server and changes the browser homepage or add unwanted bookmarks to it.
- *Opfake* is polymorphic malware that masquerades as

various apps and contents, including an installer for the Opera Mini browser, and requires the user to pay for them. It demands payment for the app or content through SMS. The malware include variants that operate also on Symbian and Windows Mobile environments. The name of the family is because the installer claims to be the Opera Mini browser.

Concerning the malware characteristics, we focused on 18 features, divided in 6 groups: privilege escalation techniques, ability to allow for remote control of the device, damages consisting in actions causing financial losses, kinds of stolen information, malware installation techniques, and methods which trigger the payload activation. Each feature is boolean, that is, a family may or may not have the corresponding characteristic. In detail, the feature are the following.

- *Privilege escalation*. In this category fall the techniques used to exploit a bug in order to gain control of device resources normally off limits to a user or an application: an application with more permissions than those provided by the original developer or fixed by the system administrator can, of course, implement unforeseen and unauthorized actions.
  - *Exploid*. It is a binary file that attempts to root a device using the exploid exploit to break out of the Android security container. It leaves a backdoor root shell in the `/system/bin` directory of the device. We highlight that the use of the backdoor shell is extremely limited and not clearly malicious, however, exploid is able to create a hole in the security layer of the phone, leaving it vulnerable to other applications wanting to take advantage of the device. If the device was successfully rooted by this app, any other app on the device could gain root access without the user's knowledge.
  - *RATC/Zimperlich*. RageAgainstTheCage (RATC) exploit binary, also known as the "adb setuid exhaustion attack", triest to exploit the adb resource exhaustion bug. Zimperlich is another exploit that take advange by a known root vulnerability in zygote (main Dalvik process) which was patched in 2.3. We grouped two exploits in the same category because it is not uncommon for a malware to have two or more root exploits in order to maximize its chances for successful exploitations—these two, in particular, are often present together.
  - *GingerBreak*. A number of important programs, including low-level system services, must run as root even on Android in order to access hardware resources. These programs are started by the init process, the first process started by the kernel which has to run as root because it needs to start other privileged system processes. The so-called all one-click-root methods, like GingerBreak, try to hack/trick one of these system processes running in privileged mode to execute arbitrary code: in this way it is possible to obtain privileged access to the system.
  - *Encrypted*. In this category fall the encrypted root exploits, i.e. the scripts of the previously techniques are encrypted in the application and are decrypted at runtime (in order to be less recognizable by anti-malware). Some of these encrypted files are located under the directory `assets`, and look like normal data files. DroidKungFu was the first malware family [22] including encrypted root exploits, i.e., the malware uses AES to encrypt the exploits it uses. Different variants use different encryption keys to better protect themselves.
- *Remote control*. This label identify the malware families that have the ability to receive bot commands from C&C servers. The communication can occurs stealthy by encrypting the URLs of remote C&C servers. Most C&C servers are registered in domains controlled by malware writers, while in other cases the C&C servers are hosted in public clouds.
- *Financial damages*. The main reasons behind malware infection is represented by the financial return: as matter of facts, malware will intentionally cause financial damages to infected users, using as vectors phone call and short message service.
  - *Call* malware can make background phone calls: the destination number can be provided from a remote C&C server, as shown by Geinimi family (which is not represented in our dataset).
  - *SMS* malware can automatically subscribe to premium-rate services, such as by sending SMS messages.
- *Information stealing*. In addition to the financial damages and root exploits, mobile malware are actively gathering a lot of personal information from the infected devices. The final aim is to use the retrieved information to generate fraudulent transactions on behalf of infected users.
  - *SMS* the list of sent and received SMS.
  - *Phone number* the telephone number, the IMEI (International Mobile Equipment Identity), and the IMSI (International Mobile Subscriber Identity).
  - *User* sensitive personal information, such as email address, banking credentials, browser history.
- *Installation*. In this group, we consider the way used by malware writers to embed malicious payload in a trusted applications.
  - *Repackaged*. With repackaging, the malicious payload is embedded into the application at installation time: the attacker decompiles a trusted application to obtain the source code, then adds the malicious payload and recompiles the application.
  - *Update attack*. An apparently innocuous application is installed on the victim's device, the user is asked to update the application, which consists of downloading the malicious payload on the victim's device after that the user has installed an app that does not exhibit any harmful behavior. With this technique, the malicious

payload is not embedded into the application at installation time.

- *Payload activation*. The malicious payload is triggered using a set of events available from the operating system, like the following.
  - *Boot*. Most of malware payloads is launched when the boot is completed (BOOT_COMPLETED event), activating a background service that does not require user interaction.
  - *SMS*. The SMS_RECEIVED event is transmitted to the system when a new SMS messaged is received. With this event, the malware has the ability to respond to specific incoming SMS messages to undertake malicious actions.
  - *Network*. The CONNECTIVITY_CHANGE event is transmitted when a change in the data connection happens, for instance when the connection switches from GPRS to HSDPA.
  - *Battery*. Within this malware feature, we group together a set of events related to battery consumption: ACTION_POWER_CONNECTED (i.e., device connected to the power), ACTION_POWER_DISCONNECTED (i.e., device disconnected from the power), BATTERY_LOW (i.e., low battery condition on the device), BATTERY_OKAY (i.e., the battery is now okay after being low), BATTERY_CHANGED_ACTION (broadcast containing the charging state, level, and other information about the battery).
  - *Sys*. With this malware feature, we refer to many system events: USER_PRESENT (useful to recognize when the phone has been unlocked or not), INPUT_METHOD_CHANGED (an input method has been changed), SIG_STR (listening to signal strength when phone sleeps) and SIM_FULL (the SIM storage for SMS messages is full).
  - *Main*. The malware responds to the ACTION_MAIN event: this event is broadcast when the app is launched by the user.

Table I shows the features for each family in our dataset.

IV. Methodology and results

We aim at exploring the use of topic modeling techniques to ease the analysis of Android malware. In particular, with respect to the *malware features* described in Section III, we want to investigate to which degree a categorization of applications based on topic models is able to capture differences among malware families. To this end, we consider the probabilistic topic model technique and, in particular, the Latent Dirichlet Allocation (LDA) which we here briefly describe.

LDA is a generative probabilistic model for a corpus of documents and assumes the existence of a predefined set of topics and a predefined set of words. Topic probabilities are defined over the corpus and word probabilities are defined over each topic. The generative model assumes that each document in the corpus has been generated by first drawing a distribution of the topics and then a distribution of the words for each topic. LDA consists also of a method to compute the posterior of the generative probabilistic model, given a corpus of documents and (as a parameter) the number k of topics. In other words, after the processing of the corpus, a numeric vector in $[0,1]^k$ is associated with each document in the corpus where the ith element is the probability that that document has been generated using the ith topic.

We apply LDA to Android applications rather than to texts. In particular, we consider an application as a sequence of opcodes: hence, topics are distributions of opcodes rather the distributions of words. In order to obtain the opcode sequence from an application a, we proceed as follows. First, we use apktool[5] to extract from the `.apk` the `.dex` file, which is the compiled application file of a (Dalvik Executable); then, with the smali[6] tool, we disassemble the application `.dex` file and obtain several files (i.e., smali classes) which contains the machine level instructions, each consisting of an opcode and its parameters. From these files, we obtain a set of opcode sequences where each item is the sequence of opcodes corresponding to the machine level instructions of a method of a class in a. We apply LDA to the concatenation of all the opcode sequences of a. Note that, due to the way LDA processes its input, the actual order of opcodes is not relevant and only the number of occurrences of each opcode within an application matters.

The applications in our datasets use 211 different opcodes: hence, each application can be represented as a point in the *opcode occurrences space* $\mathbb{N}_+^{211}$, where the $i$the element is the number of occurrences of the ith opcode in the application. After the application of LDA, each application can be represented as a point in the *topic space* $[0,1]^k$, with k being the number of topics. In order to investigate how the (relative) positioning in these spaces is related to the malware characteristic, we clustered applications and measured how much information is given by the cluster label with respect to each of the 18 malware features.

We considered 5 different clustering techniques.

- *KM*(k) The applications are clustered according to K-Means [23] applied to the opcode space. K-Means requires to set in advance the number k of clusters.
- *KM*(k) *(freq.)* Similarly to KM(k), the applications are clustered according to K-Means, but a difference space is considered: the $i$the coordinate of an application point is the relative frequency of the ith opcode, rather than the number of occurrences, in the application—we call this space the *opcode frequencies space*. The rationale for considering frequencies rather then occurrences is that applications in our dataset greatly vary in size, even within the same family, as confirmed by Figure 1, which consists of a boxplot of the length of opcode sequences grouped by malware family.

[5]http://ibotpeaches.github.io/Apktool/
[6]https://code.google.com/p/smali/

Family	Privilege escalation					Fin. dam.		Inf. stealing			Install.		Payload activation					
	Exploid	RATC/Zimp.	GingerBreak	Encrypted	Rem. control	Call	SMS	SMS	Ph. number	User	Repackaged	Update attack	Boot	SMS	Network	Battery	Sys	Main
BaseBridge		✓			✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	
DroidKungFu	✓	✓		✓	✓				✓		✓		✓			✓	✓	
FakeDoc									✓	✓	✓							✓
FakeInstaller					✓		✓				✓							✓
GinMaster			✓		✓				✓		✓		✓					
Iconosys							✓	✓	✓	✓	✓							✓
Kmin					✓						✓		✓					
Plankton					✓							✓						✓
Opfake					✓		✓		✓		✓							✓

Table I: Android malware families and their features.

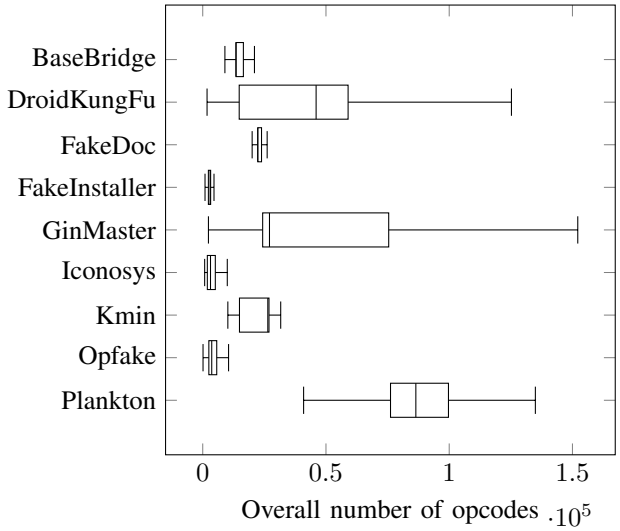


Figure 1: Boxplot of the length of opcode sequences for applications in our dataset, grouped by malware family.

- *LDA*(k) The applications are clustered according to their prevalent topic, after the application of LDA with k topics. That is, a cluster label c is assigned to each application where c is the index of the topic with the greatest probability assigned by LDA to that application.
- *LDA*(k)+*KM*(k) The applications are clustered according to K-Means applied to the topic space: the value k is used for both LDA and K-Means.
- *LDA*(25)+*KM*(k) As above, the applications are clustered according to K-Means applied to the topic space. For LDA, the number of topic is set to 25, whereas K-Means is performed with k clusters—we choose 25 because it delivered good results with the LDA(k) technique.

The first two clustering techniques do not use the information deriving from topic modeling, whereas the other 3 do: in other words, KM(k) and KM(k) (freq.) can be considered as baselines to validate our thesis that topic modeling may provide useful information for Android malware analysis.

Table II shows the main results of our analysis. For each clustering technique and each value of the k parameter in $\{5, 9, 18, 25\}$, the table shows the mutual information[7] between the cluster label and each malware feature; the rightmost column shows the average across all the 18 features. We chose those values for k—which are, essentially, (sub)multiples of 9—because our dataset consists of applications belonging to 9 different families.

Different observation can be done basing on the results of Table II. First and foremost, topic modeling appears to deliver useful information about malware features. The average mutual information between the label of LDA(k)+KM(k) and LDA(25)+KM(k) and the 18 features is largest than the one obtained with KM(k) and KM(k) (freq.): 0.32 and 0.32 vs. 0.23 and 0.27. This figures correspond to an increment ranging from 20% to 40%. It is also worth to note that LDA alone does not look effective: it indeed obtains the same average mutual information of KM, for lowest and greatest values of k. We think that clustering only on the base of the most likely topic is an estimation which is too strong, and hence much of the information is loss. On the other hand, a clustering performed on the topic space does work.

Second, as expected, due to the nature of mutual information, the greater k, the larger the mutual information. Yet, we observed by further experimentation that $k = 25$ can be considered, on the average, as a knee of the curve k vs. average mutual information. Moreover, the former finding, i.e., that topic modeling is useful, is confirmed for all values of k.

Third, there are some differences among malware features: mutual information for the best clustering technique (LDA(k)+KM(k)) ranges from 0.16 to 0.50. We think that this difference reflects the way those features are represented in the application code, and are hence more or less discoverable by looking at the occurrences (or frequencies) of opcodes. In the privilege escalation feature group, we obtain a mutual information which is in general rather low (0.24). For instance, 0.20 figure for the Encrypted feature, might be explained by the fact that the Android enviroment provided several ways

[7]We used the entropy estimator based on the empirical probability distribution.

to make encrypting: the Cipher class for istance, provided by Android API, is the most used way but in many cases malware writers make use of third-party libraries to cipher the malicious code. Additionally, using a different key, we will obtain a different resulting code and the mutual information value is low. Furthermore we highlight that privilege scripting functions are usually C-like scripts that are loaded at runtime (typically from the `asset` folder) and then they are not comprised in opcode extration.

On the other hand, relating to financial damages features, the SMS mutual information value shows rather larger value of 0.49: as matter of facts, premium SMS messages are sent (without notice to the user and, usually, to numbers which are hard-coded in the application or retrieved through an HTTP call) according to a behavior which is implemented in a similar way in the families that has this feature.

We further explored the use of 2D plots based on topic modeling data for aiding the analysis of Android malware. To this end, we performed a Principal Component Analysis (PCA) on the applications of our dataset represented in the topics space (with $k = 25$): then, we plotted each application in the space of the first two principal components—the result is shown in Figure 2c. In order to provide a baseline, we applied the same procedure to the applications represented in the opcode frequencies space—the result is shown in Figure 2b. Moreover and finally, we applied PCA on the families in the feature space, i.e., a space $\{0, 1\}^{18}$ where each coordinate represent a malware features—the result is shown in Figure 2a. The cumulative percent of variance explained is 57.9%, 21.2%, and 21.4% respectively for the features, opcode frequencies, and topics spaces. Note that in the two latter cases, no information about families nor malware features was available to "place" the application within the corresponding spaces—i.e., these results come from fully unsupervised methods. Beyond the cumulative percent of variance explained, the potential usefulness of topic modeling can be sensed graphically in Figure 2: the figure suggests that analyzing application similarities could be easier, for an operator, on the topics space than on the frequencies space.

V. Conclusions and future work

We explored the usage of topic modeling, performed by means of LDA, for the purpose of analyzing Android malware applications. In particular, we considered a dataset of 900 real world malware application and applied LDA to their representation as sequencies of opcodes, hence proposing a novel static analysis technique. We then clustered applications based on their coordinates in the topics space, that is, a low-dimension space where each coordinate represents a topic. In order to provide a baseline, we applied a similar procedure to the opcode occurrences and opcode frequencies spaces, i.e., two spaces where each coordinate represents the number of occurrences (or relative frequency) of an opcode. We found that clustering applications in the topics space gives more information about malware features that clustering in the opcode frequencies/occurrences space.

This result suggests that topic modeling could be part of a more complex framework for analyzing Android malware, both "manually" (possibly by means of visual analysis of graphical representations of the malware) or automatically. Concerning the latter scenario, we argue that higher level features deriving from the topic space may help in improving the effectiveness of fully-automatic detection techniques based on opcodes, such as the one presented in [17].

As a further extension of our work, we plan to explore the usage of the approach here proposed on data coming from dynamic analysis, such as system calls [24, 16, 14].

References

[1] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: evaluating android anti-malware against transformation attacks," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 2013, pp. 329–334.

[2] G. Canfora, A. Di Sorbo, F. Mercaldo, and C. A. Visaggio, "Obfuscation techniques against signature-based detection: a case study," in *2015 Mobile Systems Technologies Workshop (MST)*. IEEE, 2015, pp. 21–26.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[4] H. Xiao and T. Stibor, "A supervised topic transition model for detecting malicious system call sequences," in *Proceedings of the 2011 workshop on Knowledge discovery, modeling and simulation*. ACM, 2011, pp. 23–30.

[5] J. Kuriakose and P. Vinod, "Ranked linear discriminant analysis features for metamorphic malware detection," in *Advance Computing Conference (IACC), 2014 IEEE International*. IEEE, 2014, pp. 112–117.

[6] R. Mirzazadeh, M. H. Moattar, and M. V. Jahan, "Metamorphic malware detection using linear discriminant analysis and graph similarity," in *Computer and Knowledge Engineering (ICCKE), 2015 5th International Conference on*. IEEE, 2015, pp. 61–66.

[7] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *Software reliability engineering (ISSRE), 2010 IEEE 21st international symposium on*. IEEE, 2010, pp. 111–120.

[8] Y.-B. Zhao, S.-M. Liu, and S.-Q. Guo, "Extraction and prediction of hot topics in network security," in *Computer Science and Network Security, 2014 International Conference on*, 2014, pp. 347–353.

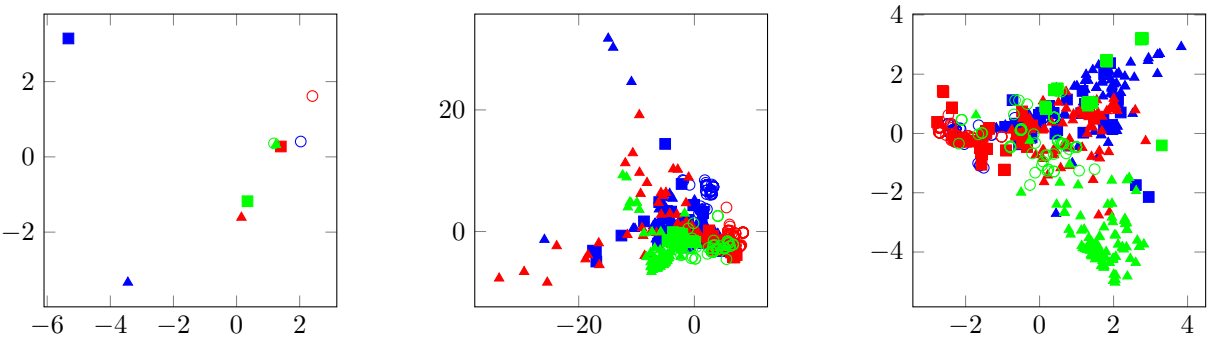
[9] F. S. Tsai and K. L. Chan, "Blog data mining for cyber security threats," in *Data Mining for Business Applications*. Springer, 2009, pp. 169–182.

[10] X. Wang, M. S. Gerber, and D. E. Brown, "Automatic crime prediction using events extracted from twitter posts," in *Social Computing, Behavioral-Cultural Modeling and Prediction*. Springer, 2012, pp. 231–238.

[11] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security:

	k	Privilege escalation				Remote control	Fin. dam.		Inf. stealing			Install.		Payload activation						*Average*
		Exploid	RATC/Zimperlich	GingerBreak	Encrypted		Call	SMS	SMS	Ph. number	User	Repackaged	Update attack	Boot	SMS	Network	Battery	Sys	Main	
KM(k)	5	0.01	0.01	0.06	0.01	0.05	0.03	0.30	0.08	0.07	0.05	0.23	0.12	0.04	0.03	0.03	0.01	0.01	0.04	0.06
	9	0.08	0.16	0.09	0.10	0.09	0.13	0.43	0.08	0.12	0.08	0.26	0.23	0.31	0.13	0.13	0.17	0.11	0.28	0.16
	18	0.10	0.20	0.12	0.12	0.17	0.13	0.43	0.12	0.24	0.10	0.27	0.25	0.33	0.13	0.15	0.19	0.17	0.32	0.20
	25	0.14	0.18	0.14	0.14	0.27	0.14	0.45	0.11	0.27	0.19	0.28	0.25	0.44	0.13	0.11	0.22	0.29	0.48	0.23
KM(k) (freq.)	5	0.05	0.09	0.04	0.04	0.14	0.04	0.19	0.15	0.04	0.23	0.06	0.03	0.09	0.05	0.02	0.04	0.04	0.03	0.08
	9	0.06	0.06	0.07	0.07	0.18	0.09	0.16	0.21	0.14	0.19	0.20	0.12	0.11	0.07	0.07	0.05	0.12	0.12	0.11
	18	0.13	0.24	0.13	0.13	0.24	0.09	0.30	0.24	0.33	0.23	0.28	0.23	0.36	0.20	0.09	0.16	0.23	0.46	0.23
	25	0.12	0.26	0.15	0.19	0.34	0.14	0.40	0.29	0.36	0.42	0.29	0.35	0.45	0.19	0.10	0.27	0.19	0.38	0.27
LDA(k)	5	0.04	0.06	0.05	0.04	0.03	0.02	0.05	0.01	0.08	0.03	0.10	0.04	0.17	0.02	0.02	0.06	0.06	0.17	0.06
	9	0.03	0.06	0.05	0.03	0.16	0.04	0.18	0.06	0.21	0.16	0.20	0.12	0.15	0.04	0.04	0.06	0.06	0.15	0.10
	18	0.05	0.06	0.11	0.05	0.17	0.07	0.39	0.13	0.26	0.17	0.28	0.20	0.22	0.07	0.07	0.06	0.06	0.22	0.15
	25	0.11	0.21	0.19	0.11	0.30	0.15	0.36	0.14	0.29	0.30	0.28	0.28	0.39	0.15	0.15	0.21	0.21	0.39	0.23
LDA(k) + KM(k)	5	0.07	0.08	0.07	0.12	0.14	0.06	0.01	0.04	0.12	0.23	0.15	0.07	0.18	0.05	0.04	0.13	0.08	0.25	0.11
	9	0.07	0.13	0.06	0.11	0.30	0.13	0.24	0.13	0.22	0.31	0.29	0.28	0.34	0.14	0.13	0.12	0.19	0.36	0.20
	18	0.15	0.24	0.18	0.14	0.41	0.14	0.44	0.26	0.38	0.41	0.28	0.31	**0.52**	**0.21**	**0.21**	0.25	0.27	0.45	0.29
	25	**0.22**	**0.34**	**0.22**	**0.20**	0.42	0.17	0.49	0.27	**0.45**	**0.46**	**0.31**	0.32	0.48	0.16	0.17	**0.34**	**0.31**	0.50	**0.32**
LDA(25) + KM(k)	5	0.05	0.13	0.06	0.09	0.29	0.14	0.26	0.16	0.07	0.34	0.30	0.12	0.39	0.14	0.11	0.15	0.10	0.35	0.18
	9	0.14	0.16	0.11	0.11	0.35	**0.19**	0.44	0.18	0.24	0.36	0.30	0.29	0.37	0.07	0.07	0.23	0.16	0.44	0.23
	18	0.19	0.25	0.21	0.10	0.42	0.16	**0.49**	0.24	0.39	0.43	0.30	0.29	0.44	0.14	0.17	0.30	0.20	0.47	0.29
	25	0.19	0.28	0.21	0.19	**0.45**	0.18	0.49	**0.29**	0.42	0.42	0.31	**0.38**	0.43	0.17	0.17	0.29	0.30	**0.52**	0.32

Table II: Mutual information between cluster and feature with different clustering methods.



(a) Malware features space. (b) Opcode frequencies space. (c) Topics space.

Figure 2: Applications (Figures 2b and 2c) and families (Figure 2a) plotted in the space of the first two principal components after PCA in the feature (Figure 2a), opcode frequencies (Figure 2b), and topics (Figure 2c) spaces.

a survey of issues, malware penetration, and defenses," *Communications Surveys & Tutorials, IEEE*, vol. 17, no. 2, pp. 998–1022, 2015.

[12] D. J. Tan, T.-W. Chua, V. L. Thing *et al.*, "Securing android: a survey, taxonomy, and challenges," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 58, 2015.

[13] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, "An integrated static detection and analysis framework for android," *Pervasive and Mobile Computing*, 2016.

[14] F. Mercaldo, C. A. Visaggio, G. Canfora, and A. Cimitile, "Mobile malware detection in the real world," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 744–746.

[15] G. Canfora, F. Mercaldo, and C. A. Visaggio, "Mobile malware detection using op-code frequency histograms," in *Proceedings of International Conference on Security and Cryptography (SECRYPT)*, 2015.

[16] ——, "Evaluating op-code frequency histograms in malware and third-party mobile applications," in *E-Business and Telecommunications*. Springer, 2015, pp. 201–222.

[17] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family android malware," in *Availability, Reliability and Security (ARES), 2015 10th International Conference on*. IEEE, 2015, pp. 333–340.

[18] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio,

"Detecting android malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*. ACM, 2015, pp. 13–20.

[19] F. Martinelli, A. Saracino, and D. Sgandurra, "Classifying android malware through subgraph mining," in *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 2014, pp. 268–283.

[20] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck, "Drebin: Efficient and explainable detection of android malware in your pocket," in *Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[21] M. Spreitzenbarth, F. Echtler, T. Schreck, F. C. Freling, and J. Hoffmann, "Mobilesandbox: Looking deeper into android applications," in *28th International ACM Symposium on Applied Computing (SAC)*, 2013.

[22] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 95–109. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.16

[23] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[24] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Acquiring and analyzing app metrics for effective mobile malware detection," in *Proceedings of the 2016 ACM International Workshop on International Workshop on Security and Privacy Analytics*. ACM, 2016.