4-2017

Characterizing malicious Android apps by mining topic-specific data flow signatures

Xinli YANG
Zhejiang University

David LO
Singapore Management University, davidlo@smu.edu.sg

Li LI
University of Luxembourg

Xin XIA
Monash University

Tegawendé F. BISSYANDE
University of Luxembourg

See next page for additional authors

Published in Information and Software Technology, 2017 April, Volume 90, Pages 27-39
http://doi.org/10.1016/j.infsof.2017.04.007

Characterizing malicious Android apps by mining topic-specific data flow signatures

Xinli Yang[a], David Lo[b], Li Li[c], Xin Xia[a,*], Tegawendé F. Bissyandé[c], Jacques Klein[c]

[a] *College of Computer Science and Technology, Zhejiang University, Hangzhou, China*
[b] *School of Information Systems, Singapore Management University, Singapore, Singapore*
[c] *Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg, Luxembourg*

Article history:
Received 28 March 2016
Revised 15 March 2017
Accepted 20 April 2017
Available online 26 April 2017

Keywords:
Malware characterization
Topic-specific
Data flow signature
Empirical study

Context: State-of-the-art works on automated detection of Android malware have leveraged app descriptions to spot anomalies w.r.t the functionality implemented, or have used data flow information as a feature to discriminate malicious from benign apps. Although these works have yielded promising performance, we hypothesize that these performances can be improved by a better understanding of malicious behavior.

Objective: To characterize malicious apps, we take into account both information on app descriptions, which are indicative of apps' topics, and information on sensitive data flow, which can be relevant to discriminate malware from benign apps.

Method: In this paper, we propose a topic-specific approach to malware comprehension based on app descriptions and data-flow information. First, we use an advanced topic model, adaptive LDA with GA, to cluster apps according to their descriptions. Then, we use information gain ratio of sensitive data flow information to build so-called "topic-specific data flow signatures".

Results: We conduct an empirical study on 3691 benign and 1612 malicious apps. We group them into 118 topics and generate topic-specific data flow signature. We verify the effectiveness of the topic-specific data flow signatures by comparing them with the overall data flow signature. In addition, we perform a deeper analysis on 25 representative topic-specific signatures and yield several implications.

Conclusion: Topic-specific data flow signatures are efficient in highlighting the malicious behavior, and thus can help in characterizing malware.

1. Introduction

The momentum of smart mobile devices carries along an impressive number of malicious apps which pose serious threats to users. Indeed, malware can lead to damages of varying severity, ranging from spurious app crashes to financial losses with malware sending premium-rate SMS, as well as to private data leaks. To devise tools and techniques that are efficient in detecting malware, researchers and practitioners require, more than ever, extensive knowledge of malicious behavior and how they can be characterized [1].

Research on automated detection of malware in the Android ecosystem has produced numerous approaches [2–4] in the last years. Among those approaches, Gorla et al. have proposed to cluster apps according to their description, and then to use anomaly analysis techniques on the functionality implemented to identify malicious apps within each cluster [5]. More recently, Avdiienko et al. have devised an approach to identify malware by using patterns of sensitive data flows to discriminate malicious from benign apps [6]. While both approaches have shown promising results, they do not exploit any understanding of malicious behavior in their detection schemes.

In this work, we consider a related but different problem than those studies about malware detection. We aim at understanding malware traits. Although malware detection is a very meaningful task since it can help automatically detect malware for users, malware characterization goes further since it can also help people aware of why an app is malware. To achieve the target, we raise

* Corresponding author.
E-mail addresses: zdyxl@zju.edu.cn (X. Yang), davidlo@smu.edu.sg (D. Lo), li.li@uni.lu (L. Li), xxia@zju.edu.cn (X. Xia), tegawende.bissyande@uni.lu (T.F. Bissyandé), jacques.klein@uni.lu (J. Klein).

two questions. What may cause an app being malicious? Does the apps of different topics have different causes for being malicious?

To answer the above two questions, we focus on associating description information of apps with their data flow information to characterize malicious behavior. we propose a topic-specific approach which combines description and sensitive data flow information. In a first step, we group different apps into several topics according to their descriptions. To that end, we leverage an adaptive Latent Dirichlet Allocation (LDA) with Genetic Algorithm (GA) [7] which allows to select the appropriate number of topics to optimally group apps. In a second step, for each topic, we collect the sensitive data flow information from the associated apps. Each piece of sensitive data flow information is weighted according to the number of times it appears in benign apps and in malicious apps, to yield an information gain ratio [8] value.

With this approach, we generate a so-called "topic-specific data flow signature". This signature is a list of data flow patterns along with their importance, represented by the information gain ratio, to discriminate malicious apps from benign apps. Building topic-specific data flow signatures presents three advantages compared to a generic (overall) data flow signature: (1) each topic-specific signature will include fewer, specific, data flow patterns; (2) each data-flow signature contains more discriminative information to identify malicious apps in a specific topic; (3) each data flow signature characterizes more fine-grained behavior of malicious apps in this topic by highlighting the specific data-flow patterns that they are prone to exhibit.

We implemented our approach and conducted an experimental assessments based on 5303 apps (3691 benign and 1612 malicious). We have crawled descriptions of apps from *Google Play Store* and *Best Apps Market*[1] for benign and malicious apps respectively, and leveraged MUDFLOW [6] to collect data flow information from all the apps. We group all the apps into 118 topics and generate topic-specif ic data flow signatures. We verify the effectiveness of the topic-specific data flow signatures by comparing them with the overall data flow signature. Moreover, we perform a deeper analysis on several representative topic-specific signatures and yield several implications. In conclusion, topic-specific data flow signatures can help better characterize malware.

The main contributions of this paper are:

1. We propose a topic-specific approach to generate data flow signatures. The topic-specific data flow signatures is much better than the overall data flow signatures to characterize malware.
2. We conduct an empirical study to demonstrate the benefits of topic-specific data flow signatures and perform a deeper analysis on several representative topic-specific signatures.

In the remainder of the paper, we provide background information (cf. Section 2), present our approach (cf. Section 3) and experiments (cf. Section 4) before discussing related work (cf. Section 5) and giving concluding remarks (cf. Section 6).

2. Background

We introduce Android malware and overview some details on data flow information in Section 2.1. Section 2.2 provides background details on the working of Latent Dirichlet Allocation (LDA) which we leverage in our work. At last we present the motivation of our work.

2.1. Malicious apps and data flow information

Malicious apps (or malwares) are apps that implement functionalities which contradict with app user interests. Generally, malwares, which include viruses, worms, trojans and spyware, are harmful at diverse severity scales.

Nowadays, there are more and more malicious apps that leak user's sensitive data without user's permission. These sensitive data include for example user's location, text message records, and even contact information. Usually, such malicious apps can be identified by manually checking their implementation code. In a previous work, Avdiienko et al. have identified malware by analyzing sensitive data flows between source and sink API calls [6]. A source is a method that accesses personal data such as account, unique device ID, and location. A sink is a method that can transmit local data to an external entity such as network, file and log. They demonstrated that it is possible to identify malware by simply inspecting whether a sensitive user data (e.g., account) is leaked from its source to an unsafe sink (e.g., network). To simplify the detection of leaks between sources and sinks, Adviienko et al. have regrouped the large number of sensitive Android APIs into a set of 34 semantic categories detailed in Table 1. Following Adviienko et al., we leverage the 34 provided categories in our work to simplify and record data flow information.

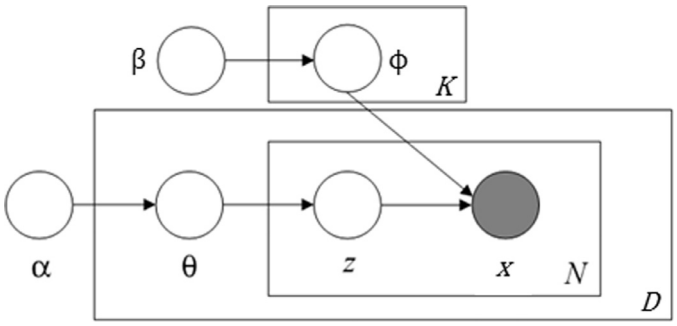


Fig. 1. The graphical model of LDA.

2.2. Latent Dirichlet allocation

Latent Dirichlet Allocation (LDA) [9] is a well-known topic model used for various tasks of software engineering research [7,10,11]. In particular, many studies about malware detection leverage LDA [5,12,13]. In the graphical model of LDA represented in Fig. 1, a circle represents a variable, an arrow represents a dependency between two variables and a rectangle represents a process which is repeated a number of times.

K denotes the number of topics, D denotes the number of documents and N the number of terms in a document. Generally, D and N are fixed by the problem, while K needs to be specified manually. Two other parameters must also be tuned carefully:

1. *alpha*, which affects the topic distribution of the documents (i.e., descriptions). A higher *alpha* leads to more uniform distribution of the topics per document. By default, *alpha* is set to 0.1.
2. *Beta*, which affects the term distribution of the topics. A higher *beta* leads to more uniform distribution of the terms per topic. By default, *beta* is set to 0.01.

In theory, LDA is a generative probabilistic model, which assumes that the data (a collection of documents) is generated based on a certain statistical process for each document d and each topic k. Specifically, LDA contains three steps:

1. *Step 1:* LDA generates a topic distribution vector *theta* and a term distribution vector *phi* based on two Dirichlet distributions [14] defined by the parameters *alpha* and *beta*, respectively.

[1] http://www.bestappsmarket.com/.

Table 1
Sensitive API categories.

Categories	Description
Source only	
ACCOUNT_INFORMATION	APIs to read details of user's online accounts.
BLUETOOTH_INFORMATION	APIs to read Bluetooth communication settings and state, along with information about connected and connecting devices.
CONTENT_RESOLVER	APIs to read contents of a given URI.
DATABASE_INFORMATION	APIs to read data from database and retrieve database metadata.
EMAIL_INFORMATION	APIs to read emails and email settings.
FILE_INFORMATION	APIs to obtain URI of resources stored in either internal or external storage devices.
HARDWARE_INFO	APIs to read device's hardware information.
LOCATION_INFORMATION	APIs to read geographical information.
NETWORK_INFORMATION	APIs to read network, telephony and connection settings.
NO_SENSITIVE_SOURCE	Non-sensitive source APIs.
PHONE_INFORMATION	APIs to read other phone related information.
UNIQUE_IDENTIFIER	APIs to read device's and user's identifiers, e.g., device id, subscriber id, etc.
VOIP_INFORMATION	APIs to obtain VOIP settings and state.
Sink only	
ACCOUNT_SETTINGS	APIs to modify user's account settings.
BLUETOOTH	APIs to send information using Bluetooth service.
EMAIL	APIs to send emails.
EMAIL_SETTINGS	APIs to modify email settings.
FILE	APIs to write data to files or other resources.
INTENT	APIs to start, fragment and manage Android's activity.
LOG	APIs to send data, warnings, and error messages to be logged.
NETWORK	APIs to modify network settings, e.g., WiFi settings, etc.
NO_SENSITIVE_SINK	Non-sensitive sink APIs.
PHONE_CONNECTION	APIs to modify phone connection settings.
PHONE_STATE	APIs to modify phone state.
SYNCHRONIZATION_DATA	APIs to manage synchronization operation.
VOIP	APIs to send data through VOIP.
Both	
AUDIO	APIs to manage volume, ringer, and other audio-related settings.
BROWSER_INFORMATION	APIs to manage browser bookmarks and data.
CALENDAR_INFORMATION	APIs to manage date and time related information.
CONTACT_INFORMATION	APIs to manage device's central repository containing data about people (e.g., their phone numbers).
IMAGE	APIs to manage metadata of image (e.g., JPEG) files.
NFC	APIs to manage communication to NFC tag.
SMS_MMS	APIs to manage SMS and MMS operations such as reading and sending text, reading and sending multimedia, etc.
SYSTEM_SETTINGS	APIs to manage device and system configurations such as password, performance counters, web settings, etc.

2. *Step 2:* LDA generates a topic assignment vector z to assign each term in a document a specific topic according to the topic distribution vector of the document *theta*.
3. *Step 3:* LDA generates each term in a document with the topic distribution vector *phi* and the topic assignment vector z.

By repeating step 1 K times, K topics are generated. By repeating step 2 and 3 N times, a document having N terms is generated. By repeating step 1 to 3 D times, a collection of D documents is generated.

In practice, LDA takes a document-by-term (D*N) matrix a as input, and outputs two matrices b and c, i.e., document-by topic (D*K) matrix and topic-by-term (K*N) matrix. The document-by-term matrix a can be a term frequency matrix, in which a_{ij} represents the number of times that the jth term appears in the ith document. In the document-by-topic matrix b, b_{ij} represents the probability of the ith document belongs to the jth topic. Generally a document is regarded as belonging to the topic with the highest probability. In the topic-by-term matrix c, c_{ij} represents the probability that the jth term belongs to the ith topic. Likewise, we assign a term to the topic with the highest probability and then we can conclude what a topic is about by looking up the terms it contains.

To some extent, LDA can be seen as a clustering algorithm. By assigning a specific topic for each document using document-by-topic matrix, a clustering of documents can be completed.

There are several implementations for LDA in the literature. In our work, we use an implementation based on collapsed Gibbs sampling. This approach achieves the same accuracy as the standard LDA implementation while being faster in each execution [15,16]. Besides the three parameters, *alpha, beta* and K introduced above, our implementation is tuned with a parameter m for the number of Gibbs sampling iterations. m is defaultly set to 2000.

2.3. Motivation

We envision our work can give a deeper insight into malware compared with the studies which generate signatures without taking into account the specificities of different topics of malware [17–19]. Indeed, since different apps provide different functionality, building a generic (overall) signature of malware across the various topics of apps may not be optimal. For example, most apps which track user's real-time location are likely malicious. Taking this information to build an overall signature may lead to false positives in the case of navigation apps which benignly, and on purpose, must track user's location and may even save this sensitive information outside the app (e.g., in a log file).

Topic-specific data flow signatures can benefit security experts who may use them to characterize malicious apps and build signature-based approaches to malware detection. Users, when provided with such information, can also understand the risk that they take in installing a given app. Finally, app developers, based on such signatures, can learn to avoid unorthodox implementations which may make their apps easily assimilable to malware.

Table 2
An example of how a data flow pattern is generated.

Source API	Uri.getQueryParameter()
Source API category	NETWORK_INFORMATION
Sink API	Log.e()
Sink API category	LOG
Data flow pattern	NETWORK_INFORMATION → LOG

3. Case study setup

In this section, we describe the details of the setup of our empirical experiments. We first detail the data collection and preprocessing step in Section 3.1, and then we present our experimental approach in Section 3.2.

3.1. Data collection

As we introduced earlier, our topic-specific approach is performed on top of two basic artifacts: the sensitive data flow information and the descriptions of Android apps. We collect them based on static analysis. To collect the sensitive data flow information of Android apps, we leverage in our work an existing dataset, which is previously published by Adviienko et al. [6], containing a set of Android apps and their sensitive data flow patterns. Those sensitive data flow patterns are extracted through a well-known state-of-the-art tool called FlowDroid [20]. Table 2 shows an example of how a data flow pattern is generated. As previously introduced in Section 2, the source and sink APIs in the data flow pattern (showed in the first and third rows) are represented through their corresponding semantic categories (showed in the second and fourth rows). Operator → indicates data flow direction. Therefore, the data flow pattern means that the app calls a source API in the NETWORK_INFORMATION category to generate data and this data is passed to a sink API in the LOG category (which means the data is logged somewhere).

The second artifact we need to collect is the descriptions of Android apps that are going to be investigated. However, it is not trivial to collect the descriptions of a given set of Android apps, especially for malicious apps as they are unlikely to be available on popular app markets, such as the Google Play store, from which we are only able to collect descriptions of benign apps. To this end, we first manually searched descriptions for a set of Android apps then come to a semi-automatic approach based on the knowledge learned from the manual process. The semi-automatic approach attempts to crawl as much descriptions as possible from a set of predefined app markets, such as *Best Apps Market*.[2]

Actually, Adviienko et al. [6] have only made available the *MD5* names of their Android apps. It is nearly impossible to search descriptions of a given app through its *MD5* information. Thus, there is a need to obtain the unique name of a given Android app through its *MD5* information. Again, this is not trivial as well. We thus come to a work around approach: at first, we collect a big data set of malicious Android apps from VirusShare,[3] and then we consider only such apps that exist in both Adviienko's dataset and our downloaded dataset. The unique app name is further retrieved from the downloaded apps.

Totally, we collect 5303 apps, including 3691 benign and 1612 malicious apps, for which we could get both their descriptions and sensitive data flow patterns. And there are totally 128 different data flow patterns appearing in these apps. Note that the ratio of benign apps over malicious ones is about 2.3, which mimics the ratio of benign and malicious apps in the real world.[4] Also note that a given app can have several sensitive data flow patterns, while different apps may exhibit a same data flow pattern. As an example, Table 3 illustrates the sensitive data flow patterns collected from a benign and a malicious apps, where a sensitive data flow pattern, named `NETWORK_INFORMATION → LOG`, is actually shared by both of these two apps.

Table 3
The data flow information of one benign app (com.phoneapps99.unblockyoutube) and one malicious app (VirusShare_57e868d46163387793fd3e260ed56ac4).

App	Data flow information
Unblockyoutube	NETWORK_INFORMATION → LOG
(Benign)	NETWORK_INFORMATION → NO_SENSITIVE_SINK
	DATABASE_INFORMATION → NO_SENSITIVE_SINK
	NO_SENSITIVE_SOURCE → INTENT
	NO_SENSITIVE_SOURCE → LOG
	NO_SENSITIVE_SOURCE → FILE
	NO_SENSITIVE_SOURCE → SYSTEM_SETTINGS
VirusShare	NETWORK_INFORMATION → SMS_MMS
(Malicious)	NETWORK_INFORMATION → INTENT
	NETWORK_INFORMATION → LOG
	NETWORK_INFORMATION → NO_SENSITIVE_SINK
	NO_SENSITIVE_SOURCE → LOG
	NO_SENSITIVE_SOURCE → INTENT

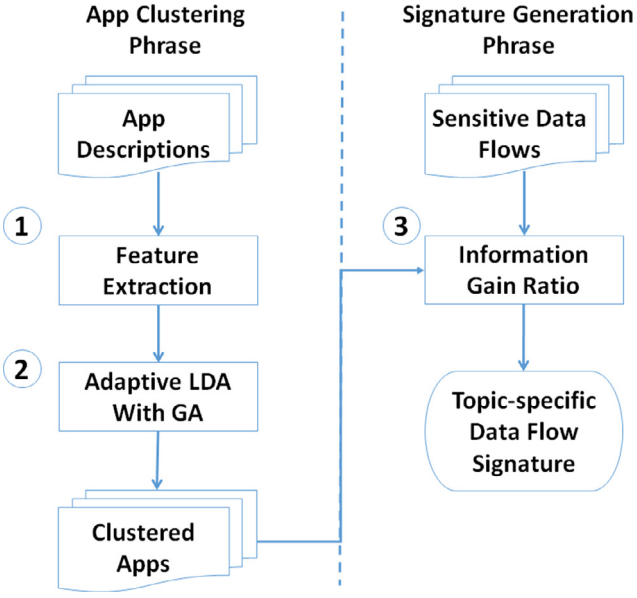


Fig. 2. The overall framework of our approach.

3.2. Data analysis

We now detail our topic-specific approach. In particular, we first present an overview of our data analysis framework. Then, we depict in detail the implementation of each important step of our framework.

3.2.1. Overall framework

Fig. 2 presents the overall framework of our proposed two-phase topic-specific approach. During the *app clustering* phase, apps are clustered according to the topics inferred from their descriptions using LDA. During the *data flow signature generation*

[2] http://www.bestappsmarket.com/.
[3] https://virusshare.com/.
[4] We estimate the ratio of benign and malicious apps using the AndroZoo project (https://androzoo.uni.lu/markets) [21], in which there are currently over 5 million Android apps and the goodware/malware ratio is 2.36.

phase, we build a topic-specific signature of apps using the sensitive data flow information which differentiates benign from malicious apps in that topic.

In Step 1, we first extract a number of features from app descriptions. These features are selected as representative terms that are useful in building a good topic model. In our work, we extract the representative terms and use their term frequency as features (cf. Section 3.2.2). Next, in Step 2, we build a topic model with the extracted features using adaptive LDA with Genetic Algorithm (GA). GA is used to determine the optimal number of topics (cf. Section 3.2.3). At the end of this step, LDA clusters the different apps into their corresponding topics.

Once all apps have been grouped in different topics, we generate, in Step 3, a topic-specific data flow signature per topic by computing information gain ratio of each piece of data flow information. For each piece of data flow information in a topic, we first count the number of times it appears in benign apps with this topic and the number of times it appears in malicious apps with this topic. We then compute the information gain ratio of the piece of data flow information accordingly (cf. Section 3.2.4). In the end, each topic-specific data flow signature consists of data flow patterns along with their likelihood to appear in benign and malicious apps estimated through the *information gain ratio*.

3.2.2. Feature extraction – text preprocessing

In general, an app description provides raw information for its use and functionality. To cluster apps with LDA, we preprocess the descriptions as many prior work did [9,22,23]. To the purpose, we use the python package NLTK.[5] NLTK is a leading platform for building Python programs to work with natural language data. With the help of NLTK, we first tokenize all the terms (i.e., words) from the descriptions. We then remove the stop words, numbers, punctuation marks and other non-alphabetic characters since they add little value to the topic. To further reduce the noises and feature dimensions, we also remove the terms that do not exist in the English vocabulary of NLTK.[6] Subsequently, we use the Snowball stemmer [24] to transform the remaining terms to their root forms (e.g., reading and reads are reduced to) to unify similar words into a common representation. Finally, we compute the term frequency-inverse document frequency (tf-idf) value for each stemmed term.

At the end of these steps, an app description d is represented as a term frequency vector, i.e., $d = (w_1, w_2, \ldots, w_n)$, where w_i denotes tf-idf value of the ith term (i.e., the number of times the ith term appears in the description d divided by the number of descriptions in which such a term appears). In total, there are 8832 different stemmed terms extracted from all the app descriptions. Note that we do not do any word filtering since LDA itself is a feature reduction algorithm which has been proved to be better than traditional information retrieval algorithm tf-idf.

3.2.3. App clustering – adaptive LDA with genetic Aagorithm

As mentioned in Section 2, we use LDA to cluster apps into different topics. In LDA, the number of topics k is an undetermined but important parameter. An overly large or overly small value of k may influence the performance of our approach severely. Therefore, we use an adaptive LDA technique, leveraging Genetic Algorithm (GA), to optimize the value of k.

Genetic algorithms simulate evolutions by natural selection [25]. In GA, the parameters waited to be searched are encoded as an individual "chromosome" and a so-called fitness function is pre-defined. The fitness function is used to evaluate the different parameter configurations by generating different fitness values. As a start, a population of randomly-generated chromosomes are initiated, where each of them represents a random parameter configuration. Then, the population will evolve a number of generations to search for an optimal parameter configuration. For each generation, the population goes through three phrases: selection, crossover and mutation. In the selection phrase, the different chromosomes are selected by a selection probability, which is transformed from their corresponding fitness values. The higher the fitness value is, the higher the selection probability is. In the crossover phrase, the selected chromosomes are paired in a random way and each pair of chromosomes are crossed over to generate a new pair of chromosomes by a crossover function and crossover rate. In the mutation phrase, the new generated chromosomes are mutated randomly by a mutation function and a mutation rate. After the aforementioned three phrases, the whole population is updated and becomes a new generation. With the generations evolving, better and better individuals (with higher fitness values) will emerge. Therefore, with the help of GA, we can obtain a proper value of k.

Algorithm 1 The GA Process in Adaptive LDA.

1: **Input:** The population size, p;
The number of generations, n;
The search scope of the number of topics k, $[min, max]$;
2: **Output:** The number of topics, *k_best*;
3: Pick p random values of number of topics, from the range $[min, max]$, using an initialization function. Denote them as k;
4: For each value k_i, compute the Silhouette coefficient as its fitness value;
5: According to the fitness value, use a selection function to select better values in k;
6: Cross over selected values using a crossover function to generate new values. Denote them as *k_new*;
7: Mutate some values in *k_new* using a mutation function, and replace k with these values;
8: Repeat Steps 4 to 7 n times and output the value *k_best* with the best fitness value;

Algorithm 1 presents the GA process in adaptive LDA. In our work, our LDA–GA approach is implemented on top of *Pyevolve*,[7] an evolutionary computation framework. For the encoding scheme, we use the classical chromosome representation: 1D Binary String. That is, we represent k in the binary system (i.e., "100001" represents 33 as the value of k). We set the length of each binary string as 7 since 7 bits is likely to be sufficient for the maximum number of topics ("1111111" can reach 127). We use the default function *G1DBinaryStringInitializator*, which is the only initialization function for binary strings and can randomly generate binary strings. For the selection phase, we use the function *GRankSelector*, which is a rank-based selector. We choose it since it behaves in a more robust manner than proportional selector, c.f., [26,27]). For the crossover phase, we use the function *G1DBinaryStringXUniform* and use the default crossover rate (i.e., 0.9). *G1DBinaryStringXUniform* performs crossover uniformly, and it is proposed by Syswerda [28]. We choose it since it helps to reduce the bias associated with the length of the binary representation used [28]. For the mutation phase, we use the default function *G1DBinaryStringMutatorFlip* and use the default mutation rate (i.e., 0.02). *G1DBinaryStringMutatorFlip* is the classical flip mutator, which randomly perform bit inversion (0 to 1 or 1 to 0) in a binary string. Finally, we set p as 20, which results from the tradeoff of good results and execution time needed. We set n as 10 since we find the fitness value becomes stable within 10 generations.

[5] http://www.nltk.org/.
[6] http://www.nltk.org/.
[7] http://pyevolve.sourceforge.net/.

For the fitness function, we follow Panichella et al.'s work on LDA–GA [7] and use the Silhouette coefficient as the fitness value. The Silhouette coefficient is first proposed by Kogan [29] and is a common evaluation metric for clustering [30–32]. The computation of the Silhouette coefficient consists of three steps:

1. **Step 1:** For a document d_i, we calculate the maximum Euclidean distance from d_i to the other documents in the same cluster, which is denoted as a(d_i). And we calculate the minimum Euclidean distance from d_i to the centroids of the other clusters (i.e., the clusters that do not contain d_i), which is denoted as b(d_i).
2. **Step 2:** Given a(d_i) and b(d_i), we can calculate the Silhouette coefficient S(d_i) for the document d_i according to the following formula:

$$S(d_i) = \frac{b(d_i) - a(d_i)}{max(a(d_i), b(d_i))}$$

3. **Step 3:** We compute the mean value of all S(d_i) as the overall Silhouette coefficient.

The scope of the Silhouette coefficient is in [−1,1]. A bigger value of the Silhouette coefficient indicates a better clustering. When a value of number of topics achieves a high Silhouette coefficient, it means that the value leads to a good result for LDA. The higher Silhouette coefficient a value achieves, the more likely it is to be kept in the evolution process. Therefore, by using the adaptive LDA with GA, we can find a proper number of app topics and we assign each app a specific topic according to its description.

3.2.4. Data flow signature generation – information gain ratio

Information gain ratio is a ratio of information gain to the intrinsic information, in which the "intrinsic information" represents the initial information entropy. It has been used in many malware detection studies [33–36]. In our work, information gain ratio can indicate what percentage of information a data flow pattern can gain from the intrinsic information. Therefore, we use information gain ratio to evaluate each data flow pattern and generate a topic-specific data flow signature per topic. A topic-specific data flow signature is a list of data flow patterns that appear in apps of the corresponding topic, where each pattern is associated with an information gain ratio value indicating its power to discriminate malicious from benign apps.

To compute information gain ratio, we first calculate information gain *IG*. For each topic, we denote the number of malicious apps of the topic as *P*, and that of benign apps of the topic as *N*. For each data flow pattern in a topic, we denote the number of times it appears in the malicious apps of the topic as *pos* and that in the benign apps of the topic as *neg*, respectively. We also denote *total*, which is equal to $pos + neg$, as the number of times it appears in all the apps of the topic. Then, the information gain $IG(d)$ of a data flow pattern d can be calculated as follows:

$$\begin{aligned} IG(d) =& E(P,N) - (total/(P+N)) \times E(pos, neg) \\ & - (1 - total/(P+N)) \times E(P - pos, N - neg), \end{aligned}$$

where

$$E(x,y) = -(x/x+y) \times log(x/x+y) - (y/x+y) \times log(y/x+y).$$

In the above formula, E denotes the information entropy and *log* is on the base of 2. When x equals to y, E will achieve the maximum value (i.e., 1). When either *neg* or *pos* equals to zero, E will achieve the minimum value (i.e., 0). The information gain $IG(d)$ is the difference of three information entropies (i.e., the intrinsic information entropy $E(P, N)$ and two information entropies generated by splitting an attribute), and thus its value is above 0 and below $E(P, N)$.

Since different topics have different intrinsic information entropy $E(P, N)$, directly comparing $IG(d)$ of different topics is unreasonable. Therefore, we divide them by $E(P, N)$ to normalize them, which leads to information gain ratio *Ratio*:

$$Ratio(d) = \frac{IG(d)}{E(P,N)}.$$

The value range of *Ratio* is in [0, 1], with which each data flow pattern can be well investigated. If *Ratio* is 0, it means that the data flow pattern cannot differentiate malicious apps from benign apps, since the pattern cannot decrease the information entropy. If *Ratio* is 1, it means that the data flow pattern is a excellent indicator which can completely discriminate malicious apps from benign apps, since the pattern gains all the intrinsic information. Moreover, the bigger the *Ratio* is, the better is the data flow pattern to be an indicator.

4. Case study

We first enumerate research questions in Section 4.1 to assess the suitability of our approach. We then present experimental results on the case study of Android malware in Section 4.2, and summarize the findings before discussing threats to validity in Section 4.4.

4.1. Research questions

We consider the following research questions for assessing the efficiency of the proposed approach to enable better characterization of malicious apps.

RQ-1: What is the distribution of malicious/benign apps over different topics?

With the first research question we aim to investigate the distribution of malicious/benign apps over different topics. The distribution of malicious/benign apps over different topics can influence the effectiveness of topic-specific malware characterization. If a topic contains much more malware than benign apps (or much more benign apps than malwares), it indicates that topic-specific malware characterization is effective. On the contrary, if a topic contains almost equally malicious and benign apps, it indicates that topic-specific malware characterization is not useful. We use the proportion of malicious apps *Proportion* as the evaluation metric. Specifically, given a specific topic t, number of malicious apps in t is *mal*, and number of benign apps in t is *ben*, *Proportion* is defined as follows:

$$Proportion = mal/(mal + ben).$$

RQ-2: Are topic-specific data flow signatures effective?

We then investigate the effectiveness of topic-specific data flow signatures. To that end, we compare topic-specific data flow signatures with the overall data flow signature (i.e., the data flow signature derived by investigating benign apps and malicious apps without considering their respective association in topics). We use information gain ratio, which is introduced in Section 3.2.4, as the evaluation metric. Note that for the overall data flow signature, the calculation of information gain ratio is based on all apps we investigate without considering their different topics. Since a signature contains many data flow patterns (each with an information gain ratio), we report the distributions of information gain ratio for each topic-specific signature, and use statistical tests to demonstrate whether topic-specific data flow signatures are substantially and statistically significantly better than the overall data flow signature in terms of information gain ratio.

RQ-3: What can we learn from topic-specific data flow signatures?

In the last research question, we go further by making a more in-depth qualitative analysis on several topic-specific data flow signatures. We discuss the data flow patterns in several topic-specific

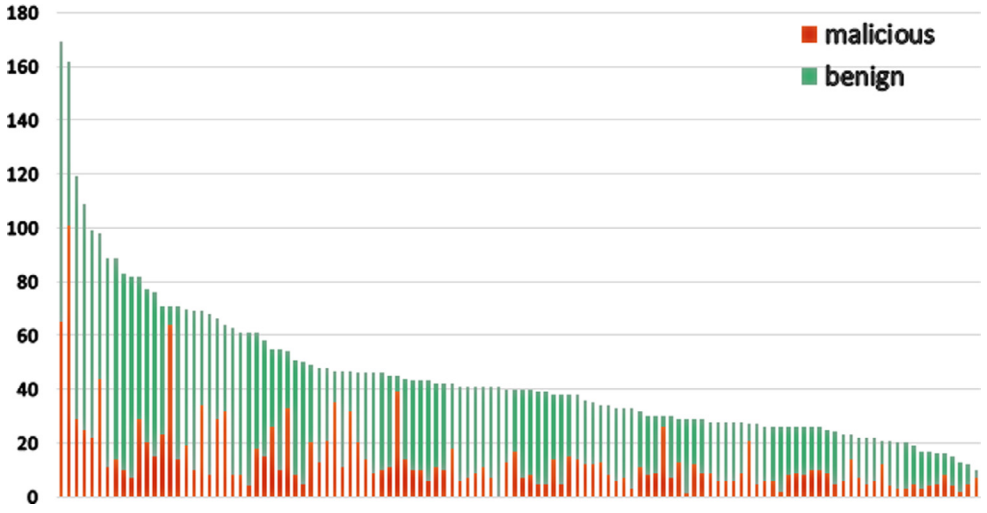


Fig. 3. Distribution of Apps in topics (ordered by number of apps). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

Table 4
The top ten representative terms and corresponding assigned names for example topics.

Assigned name	Top-10 terms (After stemming)
Body building	Weight bodi help track lose calcul health easi reach
Sport live	Sport live footbal leagu world match manag interact varieti
Puzzle game	Game puzzl play fun brain match bubbl easi solv
Weather forecast	Weather clock forecast day current hour locat temperatur wind
Protector	Secur devic protect mobil safe phone android block data
Reader	Read book reader bibl free day write content digit
News	News latest inform access break sport world read offici
Timer	Screen touch set alarm button timer volum start turn
Calendar	Day time calendar daili week month import everi year
Power management	Batteri phone power speed memori manag save charg run
Kid education	Kid fun learn child educ help littl babi age
App launcher	Theme launcher design appli android phone open choos rate
Cloud storage	Manag file android zip cloud devic copi server storag
Camera	Photo camera effect share galleri add editor sticker creat
Flash player	Flash player free adob auto angri grand citi play
Music player	Music song listen player audio play equal artist album
Social app	Chat peopl date meet free singl profil send connect
Wallpaper	Wallpap live free screen home set beauti background menu
Video player	Video watch player movi play android content best music
Navigator	Travel navig traffic job rout avail map time trip
Browser	Browser brows android fast best speed dolphin histori search
Network shopping	Shop buy product price sell trade offer differ purchas
Financial	Card scan money code manag busi credit use expens
Dictionary	English word learn dictionari translat languag vocabulari pronunci use
Screen locker	Lock screen password unlock devic hide phone use set

signatures. We aim to conclude several implications and better characterize malicious apps based on topic-specific signatures.

All experiments for answering the above research questions were conducted on an Intel(R) Core(TM) T6570 with 2.10 GHz CPU and 4 GB RAM PC running Windows 7 (64-bit).

4.2. Experiment results

RQ-1: What is the distribution of malicious/benign apps over different topics?

As mentioned in Section 3.2.3, we first use the adaptive LDA with GA to find an optimal number of topics k. Since GA is a randomized algorithm and as such it may return different results over different runs, we run GA 10 times to reduce the bias. As a result, the mean of k is 117.7, the median of k is 119 and the standard deviation of k is 7.31. We use the mean of k rounded up to the nearest integer (i.e., 118) as the number of topics to input to LDA. Table 4 presents 25 representative example topics of the LDA result. We list the top 10 representative stemmed terms, and infer a summarizing name for each topic. From the table, we can see that our approach can cluster apps into distinct categories well. Each set of top-10 stemmed terms is related to a specific topic different from the rest. This result supports the feasibility of generating topic-specific signatures.

Each topic includes a number of apps, some benign and others malicious. Fig. 3 shows the distribution of apps (green for benign and red for malicious) in the 118 topics.

From Fig. 3, we can find that malicious and benign apps are not equally distributed across topics. Some topics tend to contain more benign apps while others tend to contain more malicious apps. Specifically, among 118 topics, there are 38 topics that have *Proportion* over 80% or below 20%. For example, based on the dataset, "Flash Player" and "Wallpaper" apps include significantly more malware than benign apps (with *Proportion* of 87 and 62%), while apps for "Dictionary" and "Weather Forecast" are on the opposite end of the spectrum (with *Proportion* of 12 and 12%). This suggests that, to some extent, malware writers favor a few topics of apps more and thus demonstrates the suitability of topic-specific malware analysis.

> Malicious and benign apps are not equally distributed over different topics and some topics tend to contain more benign apps while others tend to contain more malicious apps, which indicates the suitability of topic-specific malware analysis.

RQ-2: Are topic-specific data flow signatures effective?

To show the effectiveness of topic-specific data flow signatures, we compare topic-specific data flow signatures with the overall data flow signature based on information gain ratio. Figs. 4–6 present the distributions of information gain ratio for all topic-specific signatures and the overall signature. From the figures, we can see that all topic-specific signatures are above the overall signature. For the average information gain ratio (of all the data flow patterns in one signature), topic-specific data flow signatures are

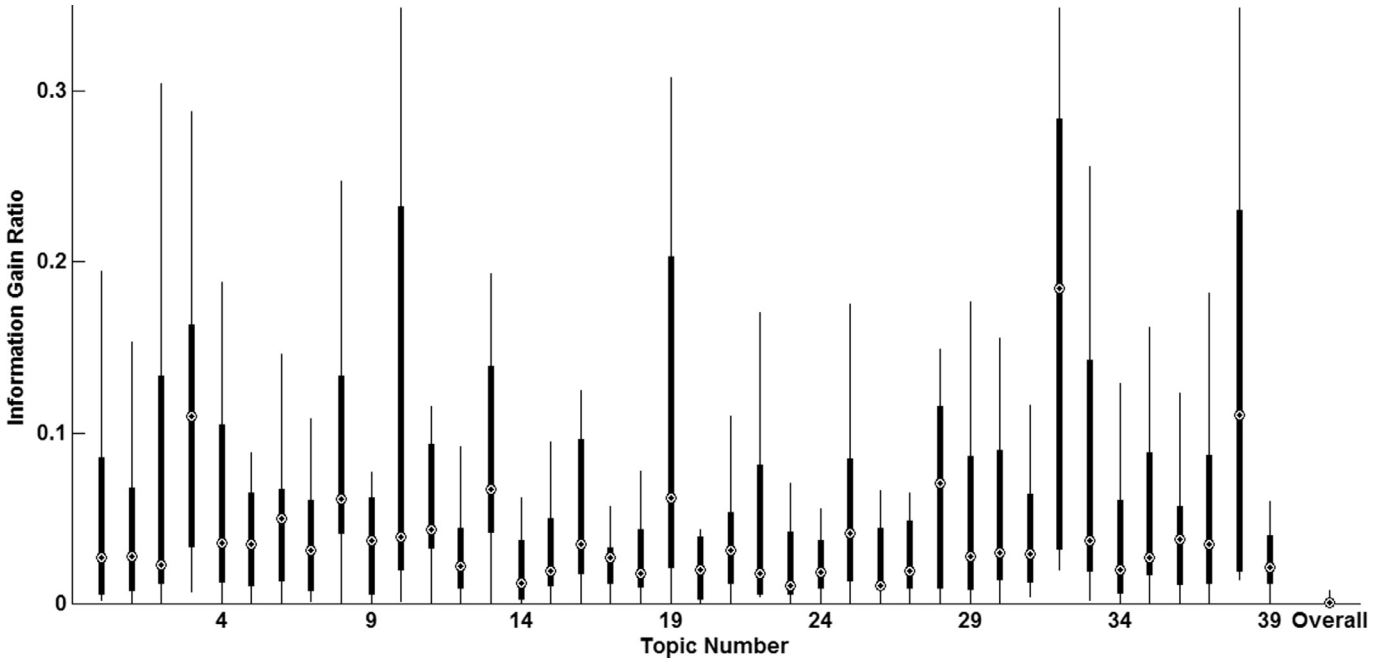


Fig. 4. Distribution of information gain ratios for all topic-specific signatures (Part1).

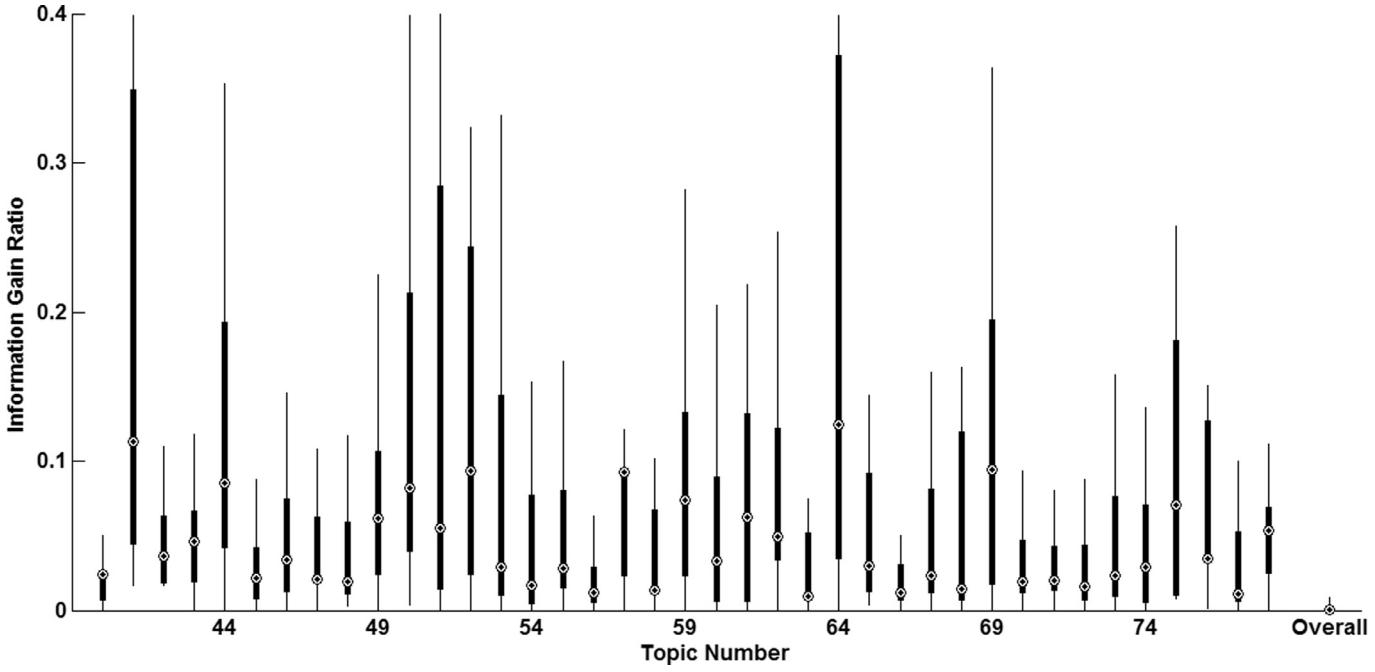


Fig. 5. Distribution of information gain ratios for all topic-specific signatures (Part2).

in the range of [0.02, 1], while the overall signature is only 0.008. Moreover, among the 118 topic-specific signatures, there are 7 signatures which contain at least one data flow patterns whose information gain ratio values achieve 1, which means these 7 signatures can correctly and totally distinguish malicious apps from benign ones of the corresponding topics. These findings suggest that topic-specific data flow signatures have better quality than the overall data flow signature and can help characterize malicious apps better.

To better demonstrate the superiority of the topic-specific data flow signatures, we perform statistical tests by comparing all information gain ratios (with each corresponding to a data flow pattern) in a topic-specific signature with those in the overall signature. Note that we perform the statistical analysis in the topic-level rather than in the app-level; this is the case since we generate a signature for each topic instead of each app. In particular, we perform the Wilcoxon rank sum test (with Benjamini–Hochberg Correction) to compute the *p*-value, and also compute the Cliff's delta. Wilcoxon rank sum test is often used to check if the difference in two data groups is statistically significant (which corresponds to a p-value of less than 0.05) or not. Cliff's delta is often used to check if the difference in two data groups is substantial. The range of Cliff's delta is in $[-1, 1]$, where -1 or 1 means all values in one group are smaller or larger than those of the other group, and 0 means the data in the two groups is similar. The mappings between Cliff's delta scores and effectiveness levels are shown in

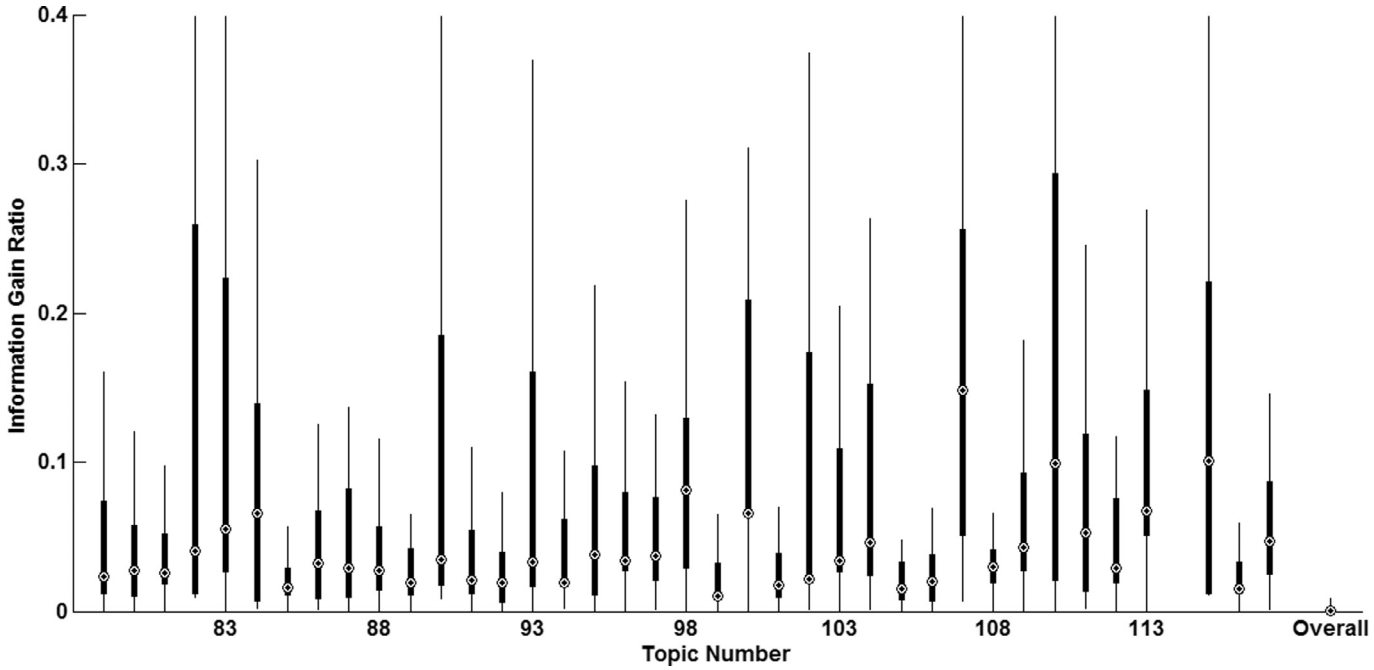
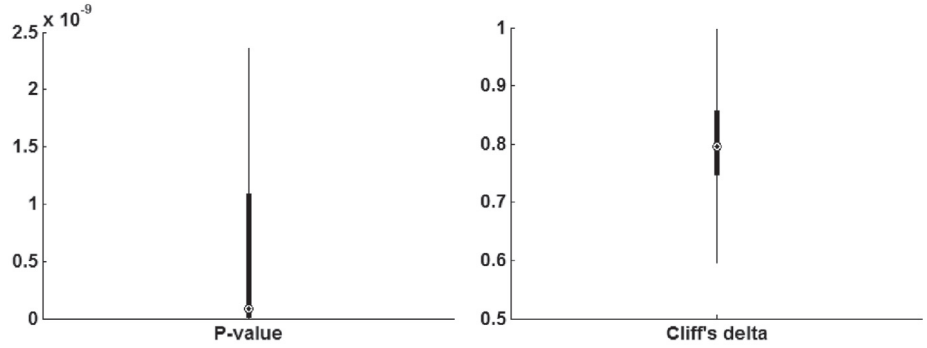


Fig. 6. Distribution of information gain ratios for all topic-specific signatures (Part3).



(a) P-value (b) Cliffs delta

Fig. 7. Distribution of *p*-values and Cliffs deltas for all topic-specific signatures compared to the overall signature.

Table 5
Mappings of Cliff's delta values to their interpretations [37].

Cliff's delta (δ)	Interpretation
$-1 <= \delta < 0.147$	Negligible
$0.146 <= \delta < 0.33$	Small
$0.33 <= \delta < 0.474$	Medium
$0.474 <= \delta <= 1$	Large

Table 5. By computing the p-value and Cliffs delta, the extent of which the topic-specific data flow signatures improves over the overall data flow signature can be more rigorously assessed.

Fig. 7 presents the distributions of *p*-values and Cliffs deltas for all topic-specific signatures compared to the overall signature. From the figures, we can see that all *p*-values are less than 0.05 and all Cliffs deltas are above 0.5 (which is large). Therefore, we can conclude that topic-specific data flow signatures are better than the overall data flow signature substantially and statistically significantly in terms of information gain ratio.

The topic-specific data flow signatures are effective. Statistical tests have shown that all the topic-specific data flow signatures are better than the overall data flow signature substantially and statistically significantly in terms of information gain ratio.

RQ-3: What can we learn from topic-specific data flow signatures?

In total, the overall data flow signature contains 128 unique data flow patterns. On the contrary, the topic-specific data flow signatures have much less patterns. The smallest signature contains only 16 patterns, and the biggest signature contains no more than half of all the data flow patterns (i.e., 61). In addition, the topic-specific signatures contains an average of 34 patterns, which is about 1/4 of all the data flow patterns.

Although the topic-specific signatures contain fewer data flow patterns, we assume their patterns are more discriminative since these patterns are restricted to a specific topic. In addition, a reduced number of data flow patterns to inspect increases characterization efficiency by leading to a concise report. Therefore, we make a more in-depth qualitative analysis to investigate each data

Table 6
Data flow patterns in the topic-specific signature for "*Flash Player*".

Data flow patterns	Information gain ratio
NETWORK_INFORMATION → NO_SENSITIVE_SINK	0.6369
UNIQUE_IDENTIFIER → NO_SENSITIVE_SINK	0.4287
NO_SENSITIVE_SOURCE → NETWORK	0.3516
LOCATION_INFORMATION → NO_SENSITIVE_SINK	0.2616
CALENDER_INFORMATION → NO_SENSITIVE_SINK	0.2127
NO_SENSITIVE_SOURCE → FILE	0.2127
CONTENT_RESOLVER → NO_SENSITIVE_SINK	0.2127
DATABASE_INFORMATION → INTENT	0.1812
NO_SENSITIVE_SOURCE → INTENT	0.1444
FILE_INFORMATION → NO_SENSITIVE_SINK	0.1320
ACCOUNT_INFORMATION → NO_SENSITIVE_SINK	0.1320

Table 7
Data flow patterns in the topic-specific signature for "*Wallpaper*".

Data flow	Information gain ratio
UNIQUE_IDENTIFIER → NO_SENSITIVE_SINK	0.4106
LOCATION_INFORMATION → NO_SENSITIVE_SINK	0.3129
NETWORK_INFORMATION → NO_SENSITIVE_SINK	0.2622
NO_SENSITIVE_SOURCE → NETWORK	0.1838
ACCOUNT_INFORMATION → NO_SENSITIVE_SINK	0.1559

Table 8
Data flow patterns in the topic-specific signature for "*Weather Forecast*".

Data flow patterns	Information gain ratio
DATABASE_INFORMATION → FILE	0.2736
NO_SENSITIVE_SOURCE → AUDIO	0.1539
UNIQUE_IDENTIFIER → NO_SENSITIVE_SINK	0.1415
UNIQUE_IDENTIFIER → LOG	0.1379
NETWORK_INFORMATION → LOG	0.1369
NO_SENSITIVE_SOURCE → NETWORK	0.1360
NETWORK_INFORMATION → NO_SENSITIVE_SINK	0.1353
UNIQUE_IDENTIFIER → INTENT	0.1308

flow pattern in the topic-specific signatures. Due to space constraints, we present analysis for three representative topics, namely "Flash Player", "Wallpaper" and "Weather Forecast".

For the signature in "Flash Player" topic, there are totally 24 distinct data flow patterns, 11 of which can gain at least 10% of the intrinsic information. For the signature in "Wallpaper" topic, there are totally 36 distinct data flow patterns, 5 of which can gain at least 10% of the intrinsic information. And for the signature in "Weather Forecast" topic, there are totally 53 distinct data flow patterns, 8 of which can gain at least 10% of the intrinsic information. Tables 6–8 enumerate those data flow patterns for the three topic-specific signatures, respectively. From the tables, we can conclude several points.

First, different topic-specific signatures have different relevant data flow patterns. For example, 5 out of 8 patterns are exclusive for the topic "Weather Forecast". Spreading different relevant patterns to different topics can much reduce the number of data flow patterns in a topic-specific signature and better characterize malwares. As a result, it is possible to immediately identify the maliciousness of an app by inspecting only the small number of relevant patterns instead of all patterns (such as 128 patterns in our dataset).

Second, even the same data flow patterns have different capabilities for different topic-specific signatures. For example, the pattern `UNIQUE_IDENTIFIER → NO_SENSITIVE_SINK` is the most discriminative pattern for the topics "Flash Player" and "Wallpaper", but not for the topic "Weather Forecast". The most discriminative pattern for the topic "Weather Forecast", `DATABASE_INFORMATION → FILE`, does not even appear in the other two topics.

Table 9
The exclusive data flow patterns of one benign app (com.devexpert.weather) and one malicious app (VirusShare_aea02d6d5afbba7c0411b9a0f58b8256).

App	Exclusive data flow patterns
Benign	FILE_INFORMATION → INTENT FILE → NO_SENSITIVE_SINK NO_SENSITIVE_SOURCE → SYSTEM_SETTINGS
Malicious	LOCATION_INFORMATION → LOG DATABASE_INFORMATION → LOG NETWORK_INFORMATION → LOG UNIQUE_IDENTIFIER → NO_SENSITIVE_SINK CONTENT_RESOLVER → NO_SENSITIVE_SINK NO_SENSITIVE_SOURCE → FILE NO_SENSITIVE_SOURCE → NETWORK

As a showcase of malware characterization, we also investigate two apps in "Weather Forecast" topic: *com.devexpert.weather* which is benign and *VirusShare_aea02d6d5afbba7c0411b9a0f58b8256* which is malicious. For the data flow information, the benign app has 9 sensitive data flow patterns and the malicious app has 13 sensitive data flow patterns, among which they have 6 mutual patterns. We list the exclusive data flow patterns of the two apps in Table 9. From the table, we can see that the 3 exclusive data flow patterns from the benign app seem quite normal. However, the 7 exclusive data flow patterns from the malicious app are suspect. The first three of them are about leaking location, database and network information into log files, while another data flow exists for sending apparently non sensitive data (which may include log data) via the network. The combination may suggest to an analyst a malicious behavior that tracks and leaks user data.

In summary, building topic-specific signatures can be seen as dimension reduction of features which can contribute to faster and clearer decision on malware classification, since the topic-specific signatures include fewer but more relevant data flow patterns. Indeed, every data flow pattern is a feature for learning. Following the signature information, one can select the data flow patterns that have big information gain ratio values as features for malware classification, since they are the ones which better discriminate malicious apps from benign apps better.

> The topic-specific data flow signatures contain fewer but more relevant data flow patterns which can gain much more information about differentiating malicious apps from benign ones. Therefore, the topic-specific data flow signatures can help characterize malicious apps better.

4.3. Threats to validity

Threats to internal validity relate to randomness and errors in our experiments. First, GA is a randomized algorithm and as such it may return different LDA configurations over different runs. Therefore, we run GA 10 times and use the average result as the final number of topics for LDA. Second, LDA is a probabilistic topic models, which means that it may return different topics in different executions. The randomness can be reduced substantially when a sufficiently large number of Gibbs sampling iterations (i.e., m mentioned in Section 2.2) is employed. We have tried various values of m and found 2000 to be a proper value that can generate stable results with acceptable running time. Setting m as 2000, we run LDA 10 times and we find that the differences are minimal. We also find that larger values of m have little influence to the results. In addition, we have double checked our implementations and all the

Table 10
The top five representative terms and corresponding assigned names for example topics.

Our topics	Google Play topics
Body building	Health & fitness
Sport live	Sports
Puzzle game	Puzzle
Weather forecast	Weather
Protector	Tools
Reader	Tools
News	News & magazines
Timer	Tools
Calendar	Events
Power management	Tools
Kid education	Education
App launcher	Tools
Cloud storage	Tools
Camera	Photography
Flash player	Tools
Music player	Music& audio
Social app	Communication
Wallpaper	Art & design
Video player	Video players & editor
Navigator	Maps & navigation
Browser	Tools
Network shopping	Shopping
Financial	Finance
Dictionary	Books & reference
Screen locker	Tools

experiment results. Hence, we believe there are minimal threats to internal validity.

Threats to external validity relate to the generalizability of our results. We have evaluated our approach on 5303 apps, containing 3691 benign apps and 1612 malicious apps. Although the number of apps is quite high, it is still not guaranteed that they are representative enough. In the future, we plan to reduce this threat further by analyzing more representative apps and ensure that each topic contain significant number of samples.

4.4. Discussion

We also compare the topics generated by our approach with the topics (i.e., categories) used by Google Play. Table 10 presents the 25 example topics generated by our approach and their corresponding Google Play topics. We can note several points. First, our topics and Google Play topics share common topics. For example, both Google Play and our topics contain “Puzzle Game”, “Navigator”, “Video Player”, etc. Second, our topics are in a finer granularity compared to Google Play topics. For example, Google Play has a topic named “Tools”, in which it includes protectors, power management tools, flash player, browser, etc.

Unfortunately, we cannot compute information gain ratios using categories from Google Play. We crawl benign apps from Google Play and malicious apps from *Best Apps Market* (i.e., http://www.bestappsmarket.com/). We do so since most apps in Google Play are clean – a similar assumption was made in prior studies [38, 39] – and Google Play quickly deletes malicious apps. As a result, we do not have the Google Play categories of the malicious apps.

5. Related work

In this paper, we leverage an advanced topic model called adaptive LDA with GA to perform topic-specific malware comprehension. State-of-the-art works that relate to this one are mainly in two folds: malware identification and topic model based investigation.

5.1. Malware identification and detection

The most related works to ours are the recent studies conducted by Gorla et al. [5] and Avdiienko et al. [6]. Gorla et al. propose an approach called CHABADA, which is dedicated to identify malicious apps [5] through app descriptions. They first cluster different apps according to their descriptions, and then use anomaly analysis technique to identify outliers with respect to their API usage. However, they only consider the APIs that are governed by user permissions, which may consequently result in false negatives and false positives, as APIs are too coarse to represent the apps’ behavior. Avdiienko et al. thus propose another approach called MUDFLOW to mitigate this limitation, which uses sensitive data flows rather than APIs to better exploit the information of apps’ API usage [6]. Those sensitive data flows are collected by MUDFLOW through a well-known static taint analysis tool named FlowDroid [20]. With the data flow information, MUDFLOW improves the performance of malware identification by a large amount. However, unlike the implementation of CHABADA, they do not take apps’ descriptions into consideration. Therefore, in this paper, we take into account both the aforementioned features (descriptions and sensitive data flow information) to implement a topic-specific approach, mining topic-specific data flow signatures within an attempt to have a deeper insight into malicious apps. Information gain is further leveraged by our approach to differentiate malicious apps from benign apps.

Aside from the two recent works highlighted above, there are also other studies related to malware identification and detection [1,40–43]. As examples, Kirat and Vigna propose an automatic technique MALGENE for extracting analysis evasion signatures [40]. They leverage a combination of data mining and data flow analysis techniques to automatically identify evasive behavior in the call events, as more and more malware can now be aware of the presence of the analysis environment (in order to evade detection). Zhou et al. propose a permission-based behavioral footprinting scheme to detect known malware and a heuristics-based filtering scheme to detect unknown malware [41]. In the first scheme, they detect malicious apps based on the inherent Android permissions and malware-specific behavioral footprints. In the second scheme, they first define suspicious behaviors from possibly malicious apps and then use them to detect suspect apps. Christodorescu et al. present an automatic technique to mine specifications of malicious behavior [42]. They compare the execution behaviors of a known malware against those of a set of benign apps so that the malicious behaviors present in the malware but not in the benign apps can be mined. Li et al. investigate a new feature set for malware detection [43]. The feature set is based on the sensitive data-flows that involve Android inter-component communications. Allix et al. conduct an analysis of a large set of malware and benign applications from the Android ecosystem [1]. Their study has reported precious insights on the writing process of Android malware and built a malware detection scheme based on these insights.

5.2. Studies leveraging topic model

The most related works to ours are the recent study by Panichella et al. [7]. Panichella et al. introduce a novel solution named LDA-GA to use topic models for software engineering tasks more effectively [7]. They use Genetic Algorithm (GA) to search for a near optimal configuration for Latent Dirichlet Allocation (LDA), which lead to better performances on different software engineering tasks. In our paper, we use their algorithm as a sub step to determine a proper number of app categories.

There are also a large number of software engineering studies that have leveraged topic model [44–54] to achieve their function-

ality. For example, Nguyen et al. propose an automated approach called BugScout to help developers reduce buggy code locating efforts by narrowing the search space of buggy files [44]. They develop a specialized topic model to correlate bug reports and the corresponding buggy files via their shared topics. In a later work, Nguyen et al. introduce a novel approach called DBTM, which again leverages topic model, to detect duplicate bug reports [45]. Their approach that combines both information retrieval and topic modeling techniques have taken the advantages of both IR-based features and topic-based features. Lukins et al. present a static LDA-based technique for automatic bug localization [46]. Their study shows that the performance of the LDA-based technique is neither affected by the size of the software system nor by the stability of the source code base.

6. Conclusion and future work

We have proposed to mine topic-specific sensitive data flow signatures to improve malware characterization. Our approach first groups different apps into several clusters (i.e., topics) according to their descriptions using an advanced topic model. Then, we generate topic-specific signatures by computing the information gain ratio for each data flow pattern seen in the apps from this topic. Empirical investigation with 3691 benign apps and 1612 malicious apps reveal that these signatures can indeed help better characterize malicious behavior. In future work, we plan to put significant effort to collect more datasets and further assess the power of topic-specific signatures for fine-grained malware identification.

Replication package. The source code and datasets of our work are available in: "https://github.com/godddding/IST".

Acknowledgment

This research was supported by NSFC Program (Nos. 61602403 and 61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2015BAH17F01).

References

[1] K. Allix, Q. Jerome, T.F. Bissyande, J. Klein, R. State, Y. Le Traon, A forensic analysis of android malware – how is malware written and how it could be detected? in: Proceedings of the IEEE Thirty-Eight Annual Conference on Computer Software and Applications, IEEE, 2014, pp. 384–393.
[2] D. Arp, M. Spreitzenbarth, H. Malte, H. Gascon, K. Rieck, Drebin: effective and explainable detection of android malware in your pocket, in: Proceedings of the Symposium on Network and Distributed System Security, 2014, pp. 23–26, doi:10.14722/ndss.2014.23247.
[3] H. Gascon, F. Yamaguchi, D. Arp, K. Rieck, Structural detection of android malware using embedded call graphs, in: Proceedings of ACM Workshop on Artificial Intelligence and Security, 2013, pp. 45–54.
[4] A. Reina, A. Fattori, L. Cavallaro, A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors, in: Proceedings of the ACM European Workshop on Systems Security, 2013.
[5] A. Gorla, I. Tavecchia, F. Gross, A. Zeller, Checking app behavior against app descriptions, in: Proceedings of the Thirty-Sixth International Conference on Software Engineering, ACM, 2014, pp. 1025–1035.
[6] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, E. Bodden, Mining apps for abnormal usage of sensitive data, in: Proceedings of the International Conference on Software Engineering, 2015.
[7] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, A. De Lucia, How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms, in: Proceedings of the International Conference on Software Engineering, IEEE Press, 2013, pp. 522–531.
[8] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, Elsevier, 2011.
[9] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent Dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.
[10] H.U. Asuncion, A.U. Asuncion, R.N. Taylor, Software traceability with topic modeling, in: Proceedings of the International Conference on Software Engineering, ACM/IEEE, 2010, pp. 95–104.
[11] S.W. Thomas, Mining software repositories using topic models, in: Proceedings of the International Conference on Software Engineering, ACM, 2011, pp. 1138–1139.
[12] B. Birrer, R.A. Raines, R.O. Baldwin, M.E. Oxley, S.K. Rogers, Using qualia and hierarchical models in malware detection, J. Inf. Assur. Secur. 4 (2009) 247–255.
[13] J. Kuriakose, P. Vinod, Ranked linear discriminant analysis features for metamorphic malware detection, in: Proceedings of the IEEE International Conference on Advance Computing, IEEE, 2014, pp. 112–117.
[14] G. Heinrich, Parameter Estimation for Text Analysis, University of Leipzig, 2008. Technical Report.
[15] T.L. Griffiths, M. Steyvers, Finding scientific topics, in: Proceedings of the National academy of Sciences, 101(Suppl 1), National Acad Sciences, 2004, pp. 5228–5235.
[16] H.M. Wallach, D.M. Mimno, A. McCallum, Rethinking LDA: why priors matter, in: Proceedings of the Advances in Neural Information Processing Systems, 2009, pp. 1973–1981.
[17] H. Yin, D. Song, M. Egele, C. Kruegel, E. Kirda, Panorama: capturing system-wide information flow for malware detection and analysis, in: Proceedings of the Fourteenth ACM Conference on Computer and Communications Security, ACM, 2007, pp. 116–127.
[18] Q. Zhang, D.S. Reeves, Metaaware: Identifying metamorphic malware, in: Proceedings of the Twenty-Third Annual Conference on Computer Security Applications, IEEE, 2007, pp. 411–420.
[19] G. Bonfante, M. Kaczmarek, J.-Y. Marion, Control flow graphs as malware signatures, in: Proceedings of the International Workshop on the Theory of Computer Viruses, 2007.
[20] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, P. McDaniel, Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps, Acm Sigplan Notices 49 (6) (2014) 259–269.
[21] K. Allix, T.F. Bissyandé, J. Klein, Y. Le Traon, Androzoo: collecting millions of android apps for the research community, in: Proceedings of the Thirteenth International Workshop on Mining Software Repositories, ACM, 2016, pp. 468–471.
[22] L. AlSumait, D. Barbará, C. Domeniconi, On-line LDA: adaptive topic models for mining text streams with applications to topic detection and tracking, in: Proceedings of the Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 3–12.
[23] O. Jin, N.N. Liu, K. Zhao, Y. Yu, Q. Yang, Transferring topical knowledge from auxiliary long texts for short text clustering, in: Proceedings of the Twentieth ACM International Conference on Information and Knowledge Management, ACM, 2011, pp. 775–784.
[24] M.F. Porter, Snowball: A language for stemming algorithms. October 2001, Retrieved March 1 (2001) 2014.
[25] D.E. Goldberg, J.H. Holland, Genetic algorithms and machine learning, Mach. Learn. 3 (2) (1988) 95–99.
[26] L.D. Whitley, et al., The genitor algorithm and selection pressure: why rank-based allocation of reproductive trials is best., in: Proceedings of the International Conference on Genetic Algorithms, 89, 1989, pp. 116–123.
[27] T. Bäck, F. Hoffmeister, Extended selection mechanisms in genetic algorithms, in: ICGA4, Morgan Kaufmann, 1991, pp. 92–99.
[28] G. Syswerda, Uniform crossover in genetic algorithms, in: Proc. Third International Conference of Genetic Algorithms, Morgan Kaufmann, 1989, pp. 2–9.
[29] J. Kogan, Introduction to Clustering Large and High-Dimensional Data, Cambridge University Press, 2007.
[30] P.J. Rousseeuw, L. Kaufman, Finding Groups in Data, Wiley Online Library, 1990.
[31] J. Sander, M. Ester, H.-P. Kriegel, X. Xu, Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications, Data Min. Knowl. Discov. 2 (2) (1998) 169–194.
[32] A. Hotho, A. Maedche, S. Staab, Ontology-based text document clustering, Knstliche Intelligenz 16 (4) (2002) 48–54.
[33] F. Ahmed, H. Hameed, M.Z. Shafiq, M. Farooq, Using spatio-temporal information in API calls with machine learning algorithms for malware detection, in: Proceedings of the Second ACM Workshop on Security and Artificial Intelligence, ACM, 2009, pp. 55–62.
[34] S.M. Tabish, M.Z. Shafiq, M. Farooq, Malware detection using statistical analysis of byte-level file content, in: Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, ACM, 2009, pp. 23–31.
[35] Z. Aung, W. Zaw, Permission-based android malware detection, Int. J. Sci. Technol. Res. 2 (3) (2013) 228–234.
[36] I. Santos, F. Brezo, X. Ugarte-Pedrero, P.G. Bringas, Opcode sequences as representation of executables for data-mining-based unknown malware detection, Inf. Sci. (NY) 231 (2013) 64–82.
[37] N. Cliff, Ordinal Methods for Behavioral Data Analysis, Psychology Press, 2014.
[38] Y. Wang, J. Zheng, C. Sun, S. Mukkamala, Quantitative security risk assessment of android permissions and applications, in: Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2013, pp. 226–241.
[39] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, W. Enck, Appcontext: differentiating malicious and benign mobile app behaviors using context, in: Proceedings of the Thirty-Seventh IEEE/ACM International Conference on Software Engineering, 1, IEEE, 2015, pp. 303–313.
[40] D. Kirat, G. Vigna, Malgene: automatic extraction of malware analysis evasion signature, in: Proceedings of the Twenty-Second Conference on Computer and Communications Security, ACM, 2015, pp. 769–780.
[41] Y. Zhou, Z. Wang, W. Zhou, X. Jiang, Hey, you, get off of my market: detecting malicious apps in official and alternative android markets., in: Proceedings of the Symposium on Network and Distributed System Security, 2012.
[42] M. Christodorescu, S. Jha, C. Kruegel, Mining specifications of malicious behav-

ior, in: Proceedings of the First India Software Engineering Conference, ACM, 2008, pp. 5–14.
[43] L. Li, K. Allix, D. Li, A. Bartel, T.F. Bissyandé, J. Klein, Potential component leaks in android apps: an investigation into a new feature set for malware detection, in: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, IEEE, 2015, pp. 195–200.
[44] A.T. Nguyen, T.T. Nguyen, J. Al-Kofahi, H.V. Nguyen, T.N. Nguyen, A topic-based approach for narrowing the search space of buggy files from a bug report, in: Proceedings of the Twenty-Sixth International Conference on Automated Software Engineering, IEEE, 2011.
[45] A.T. Nguyen, T.T. Nguyen, T.N. Nguyen, D. Lo, C. Sun, Duplicate bug report detection with a combination of information retrieval and topic modeling, in: Proceedings of the Twenty-Seventh International Conference on Automated Software Engineering, IEEE, 2012.
[46] S.K. Lukins, N.A. Kraft, L.H. Etzkorn, Bug localization using latent Dirichlet allocation, Inf. Softw. Technol. 52 (9) (2010) 972–990.
[47] Y. Zhang, D. Lo, X. Xia, T.-D. B. Le, G. Scanniello, J. Sun, Inferring links between concerns and methods with multi-abstraction vector space model, in: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, IEEE, 2016, pp. 110–121.
[48] I. Santos, Y.K. Penya, J. Devesa, P.G. Bringas, N-grams-based File Signatures for Malware Detection, ICEIS 9 (2) (2009) 317–320.
[49] J. Pfoh, C. Schneider, C. Eckert, Leveraging string kernels for malware detection, in: Proceedings of the International Conference on Network and System Security, Springer, 2013.
[50] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, J.-L. Sun, What security questions do developers ask? A large-scale study of stack overflow posts, J. Comput. Sci. Technol. 31 (5) (2016) 910–924.
[51] X. Xia, D. Lo, Y. Ding, J.M. Al-Kofahi, T.N. Nguyen, X. Wang, Improving automated bug triaging with specialized topic model, IEEE Trans. Softw. Eng. 43 (3) (2017) 272–297.
[52] X. Xia, D. Lo, X. Wang, B. Zhou, Accurate developer recommendation for bug resolution, in: Proceedings of the Twentieth Working Conference on Reverse Engineering, IEEE, 2013, pp. 72–81.
[53] X. Xia, D. Lo, X. Wang, B. Zhou, Dual analysis for recommending developers to resolve bugs, J. Softw. Evol. Process 27 (3) (2015) 195–220.
[54] Y. Zhang, D. Lo, X. Xia, J.-L. Sun, Multi-factor duplicate question detection in stack overflow, J. Comput. Sci. Technol. 30 (5) (2015) 981–997.