SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2023, Vol. 66 192101:1–192101:19
https://doi.org/10.1007/s11432-021-3414-7

ArgusDroid: detecting Android malware variants by mining permission-API knowledge graph

Yude BAI[1], Sen CHEN[1*], Zhenchang XING[2] & Xiaohong LI[1*]

[1] *College of Intelligence and Computing, Tianjin University, Tianjin* 300350*, China;*
[2] *Research School of Computer Science, Australian National University, Acton* ACT 2601*, Australia*

Received 9 August 2021/Revised 15 October 2021/Accepted 24 January 2022/Published online 3 April 2023

Abstract Malware family variants make minor and relevant changes of behaviors based on the original malware. To analyze and detect family variants, security experts must not only understand malware behaviors but also further observe the correlation between the features of these behaviors. However, the recent data-driven based behavior features are too independent and sometimes too general to obtain a comprehensive profile of the changeable malicious behaviors of family variants derived from the original malware. Those features additionally suffer from limited semantic knowledge which narrows the comprehension of family variants. To this end, in this paper, we propose ArgusDroid that takes advantage of the knowledge graph (KG) to construct a permission-API knowledge graph based on the official Android document. Because each permission or API in the document is described by a specific sentence, we can easily acquire and comprehend the relationship between different features via the hyperlink in sentences or sentence similarity. ArgusDroid also extracts various feature sets from the knowledge graph and validates the detection performance on Android malware family variants based on these features. Extensive experiments by using machine learning and neural network classifiers for variant identification have been carried out. The experimental results demonstrate the effectiveness and usefulness of our obtained feature sets based on ArgusDroid, especially when using the classifiers convolutional neural network (CNN) and multi-layer perception (MLP). Furthermore, when compared to similar feature sets that aim to present relationships across different feature types, such as Axplorer, ArgusDroid generates the feature set which significantly improves malware variant detection by 0.3575 average F1.

Keywords malicious behavior, Android document, knowledge graph, malware family variant, machine learning

Citation Bai Y D, Chen S, Xing Z C, et al. ArgusDroid: detecting Android malware variants by mining permission-API knowledge graph. Sci China Inf Sci, 2023, 66(9): 192101, https://doi.org/10.1007/s11432-021-3414-7

1 Introduction

With the harshly zooming increase of Android malware, grouping malware into different malware families has been proposed to economize a great number of time and efforts for the hard manual security analysis [1, 2]. Malware family represents the primary activities of malware, such as family “DroidKungFu” as a backdoor to infect mobile phones [3, 4]. Meanwhile, malware within the same family is classified as different malware variants based on whether they use similarly fine-grained malicious behaviors, according to [5]. For example, malware in “DroidKungFu” steals device information and accesses the network, while it evolves into variants by adding new behavior to read or write SMS (short message service) that contains private device information. In this way, malware variants generate variously changeable malicious behaviors [1, 6]. The identification of malware variants therefore relies on a more complete set of behavior features in which features are correlated and provide semantic knowledge [7, 8]. This is more challenging and difficult than general malware detection.

In general, the critical features of Android malware are usually summarized by a data-driven method where analysts have to commit most of their efforts to understand the application source code and the

[*] Corresponding author (email: senchen@tju.edu.cn, xiaohongli@tju.edu.cn)

Table 1 Example of API in Android document[a)]

Concerned API	getLine1Number()
Description	Returns phone number string for line 1... **Requires** Permission: READ_SMS, READ_PHONE_NUMBERS, that the caller is the **default SMS app**, ... READ_PHONE_STATE for apps targeting **SDK API level 29 and below**.
Concerned permission	READ_SMS READ_PHONE_NUMBERS READ_PHONE_STATE

a) Bold font is the explanation.

Android OS source code [9–12]. The permission, which is declared to grant additional capabilities or control limited resources [13], and the dangerous API, which is run when requesting access to the private device or user information [14], are two important types of data-driven sensitive features. Although some of them, such as PScout [9] and Axplorer [10], aimed to build mappings between permission and API by analyzing the permission system of Android OS and achieved excellent performance on general malware detection, approaches based on the corresponding feature sets are still limited in identifying malware variants. Because, to some extent, the selected features are still independent due to lack of enough semantic knowledge and relations across different features and sometimes general to capture the changeable behaviors of variants in malware family. As a result, the previous feature sets chosen by the data-driven method make it difficult to identify the subtle behavior changes of new malware variants derived from the original malware (e.g., the behavior changes in "DroidKungFu" and its variant as mentioned previously).

Actually, during the detection of malware variants, first of all, we need to reason unknown malicious behaviors based on the analysis of given malware [15]. Therefore, we must understand not only the features of malware variants but also the semantic relationship between different features to the greatest extent possible. The effective features should be as semantically complete as possible and sensitive to behavioral changes across different malware variants in the same family. Inspired by the success of knowledge graph (KG) on Android document [16–18], we first propose ArgusDroid, a knowledge-driven method for finding more interrelated features with semantic knowledge from Android official document to effectively identify the family variants. Specifically, we exhaustively investigate the permissions, APIs and relevant descriptions in the document at first. We next develop a domain-specific Android permission-API knowledge graph based on the latest Android document of Android 11 with API level 30. This graph, like PScout [9] and Axplorer [10], primarily shows the relationships between permission and API (also has relations between permission and permission). Moreover, ArgusDroid is able to obtain an explanation from the document as to why particular permission is related to an API based on their relations in the graph and their detailed functional descriptions. It can be applied to reveal more complete behaviors including the behavior changes across different malware variants that include more than one permission or API [19]. Table 1 illustrates an example of API and its correlated permissions. It is obvious that "getLine1Number()" requires "READ_PHONE_STATE" and "READ_SMS" when the application is a default SMS application with API level less than 29. Note that the permission "READ_PHONE_NUMBERS" is a subset granted by "READ_PHONE_STATE"[1)]. Thus, reading SMS, gaining phone states and phone numbers might be the behaviors led by an application (API level 29 or below).

With the permission-API knowledge graph, we build different hyperlink-based feature sets similarity-based feature sets according to different "hops"(see in Subsection 2.3 for the technical details). Taking the "READ_PHONE_STATE" in Figure 1 as an example, the features with edge "Link" around it (e.g., "getNetworkType()") can be set as features in the hyperlink-based feature set, while the features with edge "Sim" around it (e.g., "ACCESS_NETWORK_STATE") will be put into the similarity-based feature set. We chose the well-known malware variants dataset AMD [5] for this work to evaluate such constructed knowledge-driven feature sets when detecting malware variants. Because AMD has at least two types of variants in each of its 23 families, variant detection is formulated as a multi-class classification problem. Each family selects one type of variant at random as the test dataset, while the remaining variants serve as the training dataset.

We apply six widely-used multi-class classifiers for variant detection, e.g., support vector machine (SVM) [20], random forest (RF) [21], k-nearest neighbor (KNN) [22], multi-layer perceptron (MLP) [23],

[1)] Permission. https://developer.android.google.cn/reference/android/Manifest.permission#READ_PHONE_NUMBERS.

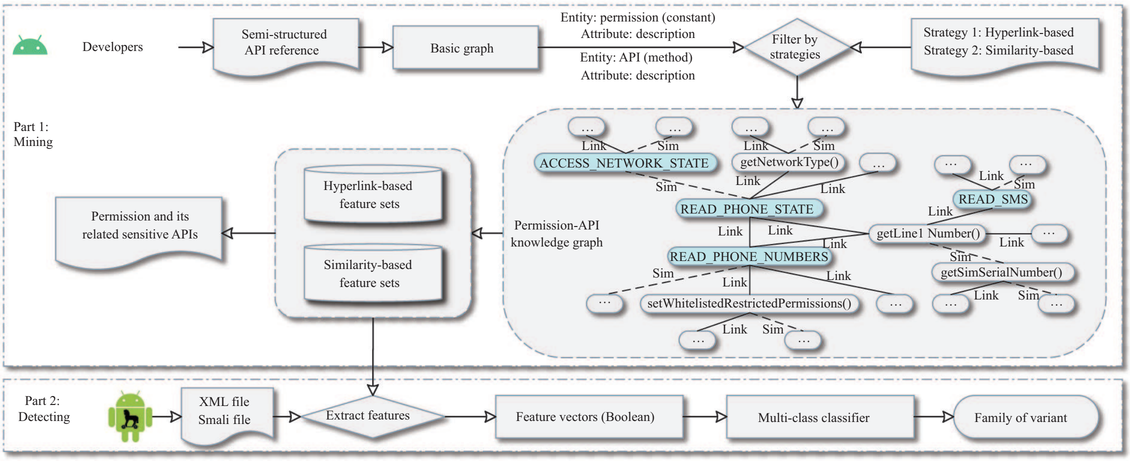

Figure 1 (Color online) An overview of our framework ArgusDroid. "Link" and "Sim" in permission-API knowledge graph refer to "Hyperlink" and "Similarity", respectively.

convolutional neural network (CNN) [24], and long short-term memory (LSTM) [25]. Meanwhile, we compare the efficiency with a widely-used baseline Axplorer [10]. The experimental results reveal that the performance of variant detection increases when the number of hops raises by using the neural network classifiers MLP and CNN. The performance boundary (0.9613 average F1) occurs when applying the four hops feature set hop_four. Simultaneously, for each hyperlink-based feature set, the addition of similarity-based features promotes the result on predicting malware variants with one hop and two hops features, while failing to further improve significantly with three hops and four hops features. Finally, we highlight that the proposed hyperlink-based feature set significantly outperforms the baseline feature set (given by Axplorer) with more than 0.3575 average F1. All of the features adopted and empirical results are released on the given available website[2].

In this paper, we make the following contributions:

- To the best of our knowledge, our work, dubbed ArgusDroid, is the first to investigate malware family variant detection based on a domain-specific permission-API knowledge graph constructed by Android document.
- We develop a hyperlink-based strategy and a similarity-based strategy to generate different knowledge-driven feature sets, and we demonstrate that a feature set within more hops significantly improves detection when CNN and MLP classifiers are employed.
- We conduct extensive experiments to systematically investigate the effectiveness and usefulness of knowledge-driven feature set for variant detection, which outperforms the baseline data-driven feature set produced by Axplorer with over 0.3575 average F1.

Finally, we emphasize that knowledge graph-based family variant detection is a new research direction, and the ArgusDroid presented in this paper serves as a starting point for reasoning rich knowledge from documents for security-related specific tasks such as malware detection and security vulnerability identification.

2 Methodology

The overview of ArgusDroid is presented in Figure 1, which inculdes the part that mines permission and API from the permission-API knowledge graph of the Android document, as well as the part that detects malware family variants by machine learning and neural network classifiers. Part 1 consists of three steps: building the basic graph (Subsection 2.1), building the permission-API knowledge graph (Subsection 2.2), and mining features (Subsection 2.3). Part 2 includes three steps: pre-processing Android malware dataset (Subsection 2.4), training multi-class classifiers and detecting malware variants (Subsection 2.5).

[2] Github. https://github.com/qWe1aSd/varPre.

2.1 Basic graph construction

The official Android document, Android Developers[3)], gathers important knowledge of Android and Android applications. We concentrate on the semi-structured "Android API reference" whose advantages have been strongly confirmed by [16, 18, 26, 27]. By using Beautiful Soup[4)], we download and parse 9238 webpages which are comprised of 9328 classes, 78720 APIs, and 56467 constants. As we purpose to analyze the textual descriptions of them so that we remove the table of content (a navigated content), code, script snippet, and image by refining HTML tags (e.g., ⟨devsite-toc⟩, ⟨devsite-code⟩, ⟨img⟩). Thus, a basic Android document knowledge graph is formed, in which each entity is identified by its unique full name and entities are linked by hyperlink in the description. Because of space constraints, all permissions and APIs used in this paper are abbreviated. Our Github repository hosts the mapping table between full name and abbreviation.

2.2 Permission-API knowledge graph construction

We concentrate on the entities' permission and API in the basic graph since they are highlighted for analyzing malware behaviors in [8, 13–15]. Besides the hyperlink-based relations of entities in the basic graph, we further expand similarity-based relations via computing the textual sentence similarity of each permission or API. This is triggered by the fact that only part of permissions or APIs have a hyperlink in their descriptions and that Refs. [28, 29] have successfully applied sentence similarity to assist Android security analysis and permission recommendation. For example, the "READ_PHONE_STATE" and "ACCESS_NETWORK_STATE" that are similar due to their description similarity can be applied to reason changes of malicious behavior with either accessing network or obtaining phone information. Therefore, we build a permission-API knowledge graph in which entities are permission or API, and relations are generated by the following hyperlink-based strategy and similarity-based strategy.

2.2.1 *Hyperlink-based strategy*

If the description of an entity gives a hyperlink to another entity, we consider them as the hyperlink-based relation. For example, "getLine1Number()" is hyperlinked to "READ_SMS" by the edge "Link" in Figure 1.

2.2.2 *Similarity-based strategy*

The similarity-based strategy needs to compute the semantic similarity of two different entities by their descriptions. It reveals the inherent relation (semantic relation) between entities and is presented as the edge "Sim" in Figure 1. In the meantime, it is regarded as an addition to the hyperlink-based strategy. Because an entity (permission or API) may or may not be linked to another, even if they are semantically similar [16]. For instance, the "WRITE_EXTERNAL_STORAGE" allows "an application to write to external storage". It should invoke "write()" to write a string into the external storage. We find no direct hyperlink in their description sentences, but both of them do similar operations (described in sentences) to write into external storage. We, therefore, devise a TF-IDF-like method for mining this similarity-based relationship by computing the similarity of their description sentences [30, 31]. In addition, for each entity, we only pick out the entity that is most similar (with the largest similarity value) to it in this similarity-based strategy. Other similar entities are ignored because adding similarity-based entities to the feature set only results in a marginal improvement in variant prediction (see the experimental discussion on significant analysis in Subsection 3.4).

We adopt Stanford CoreNLP[5)] to tokenize entity description. We then calculate the semantic similarity of two entities with their tokenized descriptions. For example, given two entities (en_1 and en_2) with tokens (des_1 and des_2, respectively), their similarity $\mathrm{sim}_{\mathrm{en}_1,\mathrm{en}_2}$ is defined as

$$\mathrm{sim}_{\mathrm{en}_1,\mathrm{en}_2} = \frac{\sum_{\mathrm{word}\in\mathrm{des}_1}(\mathrm{sim}_{\mathrm{word},\mathrm{des}_2}) \times \mathrm{idf}(\mathrm{word})}{\sum_{\mathrm{word}\in\mathrm{des}_1}\mathrm{idf}(\mathrm{word})}, \tag{1}$$

[3)] Android developers. https://developer.android.google.cn/.

[4)] Beautiful Soup. https://www.crummy.com/software/BeautifulSoup/.

[5)] CoreNLP. https://stanfordnlp.github.io/CoreNLP.

where idf(word) denotes the inverse document frequency of word given by the equation $\log(\frac{\mathrm{num}_{\mathrm{des}}}{1+\mathrm{df}_{\mathrm{word}}})$ in [30]. The $\mathrm{num}_{\mathrm{des}}$ is the number of total entities with description in the corpus, and $\mathrm{df}_{\mathrm{word}}$ is the document frequency of word (i.e., the number of entities with the description that including word). The $\mathrm{sim}_{\mathrm{word},\mathrm{des}_2}$ in (1) refers to the similarity value between each word of en_1 and des_2 of en_2, which is formulated as

$$\mathrm{sim}_{\mathrm{word},\mathrm{des}_2} = \max_{\mathrm{word}' \in \mathrm{des}_2} \mathrm{sim}_{\mathrm{word},\mathrm{word}'}, \tag{2}$$

where $\mathrm{sim}_{\mathrm{word},\mathrm{word}'}$ means the value of cosine similarity between word and word$'$ vectorized by a 50-dimensional word embeddings pre-trained through GloVe model [32]. We pick out the maximum value from the similarity between word and each word$'$ in des_2 to represent the similarity ($\mathrm{sim}_{\mathrm{word},\mathrm{des}_2}$) between word and des_2.

Permission is the kernel in this domain-specific knowledge graph because it is regarded as the most important security mechanism in Android [10,13]. Each API should be linked to at least one permission via the edge of this graph. We hope to use this domain-specific graph to better understand the semantic relationship between permission and API and to distinguish variants.

2.3 Feature mining

To the issue of malware variants detection, we pick out part of entities (permission and API) in the proposed graph as features. Specifically, we define the permission as the critical features at first (certified by experts in [10,13]) and sequentially extract features with different “hop” around the initial permission. With this method of feature selection, we aim at exploring the performance boundary of knowledge-driven features from ArgusDroid (discussed in Subsection 3.3). By using a large scale of features for classifying, this will also reduce the negative effects of overfitting [14]. The permission-API knowledge graph of ArgusDroid builds a total of 8 sensitive feature sets (listed in Table 2) in this subsection, each of which has been used by malware at least once. Algorithm 1 shown in the following formulates and illustrates the construction of those feature sets.

Algorithm 1 Feature mining based on two strategies

```
Input: $E_{\mathrm{init}}$, initial APIs and permissions with description;
       $E_{\mathrm{all}}$, all APIs and permissions with description;
       $R_{\mathrm{Link}}$, hyperlink-based strategy;
       $R_{\mathrm{Sim}}$, similarity-based strategy.
Output: $\mathrm{FS}_{\mathrm{hop}}$, feature set based on hyperlink-based strategy;
        $\mathrm{FS}_{\mathrm{sim}}$, feature set based on similarity-based strategy.
 1: Let $e$ be an entity;
 2: $\mathrm{FS}_{\mathrm{hop}} := E_{\mathrm{init}}$;
 3: for $e$ in $E_{\mathrm{all}}$ do
 4:     if $e$ and at least one entity in $E_{\mathrm{init}}$ has $R_{\mathrm{Link}}$ then
 5:         $\mathrm{FS}_{\mathrm{hop}} \leftarrow \mathrm{FS}_{\mathrm{hop}} \bigcup \{e\}$;
 6:     end if
 7: end for
 8: Drop duplication of $\mathrm{FS}_{\mathrm{hop}}$;
 9: $\mathrm{FS}_{\mathrm{sim}} := \mathrm{FS}_{\mathrm{hop}}$;
10: for $e$ in $\mathrm{FS}_{\mathrm{hop}}$ do
11:     if $e$ and the entity in $E_{\mathrm{all}}$ has $R_{\mathrm{Sim}}$ then
12:         $\mathrm{FS}_{\mathrm{sim}} \leftarrow \mathrm{FS}_{\mathrm{sim}} \bigcup \{e\}$;
13:     end if
14: end for
15: Drop duplication of $\mathrm{FS}_{\mathrm{sim}}$;
16: return $\mathrm{FS}_{\mathrm{hop}}, \mathrm{FS}_{\mathrm{sim}}$.
```

The 22 permissions with dangerous level are taken as the initial features which are formulated as E_{init} in Algorithm 1. E_{all} denotes all APIs and permissions with their descriptions in the permission-API knowledge graph. Lines 2–8 describe how to create the hyperlink-based feature set of E_{init}. Specifically, based on the hyperlink-based strategy R_{Link}, we search directly linked or be-linked APIs/permissions for each feature in E_{init} from E_{all}. It generates the hyperlink-based feature set $\mathrm{FS}_{\mathrm{hop}}$ after dropping duplication. Lines 9–15 in Algorithm 1 present the way to build the similarity-based feature set by the similarity-based strategy R_{Sim}. That is to find the most similar API/permission of each feature in $\mathrm{FS}_{\mathrm{hop}}$ from E_{all}. We then obtain the similarity-based feature set $\mathrm{FS}_{\mathrm{sim}}$ without duplication. On our dataset, $\mathrm{FS}_{\mathrm{hop}}$ and $\mathrm{FS}_{\mathrm{sim}}$ are respectively represented as feature set hop_one and hop_one_sim in which features

Table 2 The constructed feature sets

Feature set	#Feature	Feature set	#Feature
hop_one	32	hop_one_sim	61
hop_two	50	hop_two_sim	91
hop_three	183	hop_three_sim	289
hop_four	451	hop_four_sim	580

#Feature is the number of features in a feature set.

are linked within one hop around the initial features. As is shown in Table 2, hop_one contains 32 features and the expanded hop_one_sim has 61 features.

To expand the feature set, we continue to add the indirect features (by adding hops) of the initial 22 permissions. When taking two hops to search in the graph, the $E_{\rm init}$ is currently set as features in the previous $\rm FS_{hop}$. So the new $\rm FS_{hop}$ constructed by lines 2–8 of Algorithm 1 can display the indirect features within two hops of the initial 22 permissions. Those two new $\rm FS_{hop}$ and $\rm FS_{sim}$ are specified as feature sets hop_two and hop_two_sim of our dataset in which inculdes 50 and 91 features respectively in Table 2. In this way, we make feature sets hop_three, hop_three_sim, hop_four, and hop_four_sim to form the indirectly features within three and four hops of the initial permissions shown in Table 2.

The maximum hop is limited to 4. Because the feature (source feature) description typically contains multiple hyperlinks to other features (target features). Not all of these target features are strongly related to the source feature. Algorithm 1, on the other hand, considers them available features to detect. As the number of hops increases, so do redundant features, which may have an impact on the variant detection. We discuss this influence in detail in Subsection 3.3. For the feature sets created by a similarity-based strategy, we regard them as the supplement of the relevant (within the same hop) hyperlink-based feature sets. We believe that, when compared to the document-certified hyperlink-based feature sets, the expanded similarity-based feature sets provide valuable semantic knowledge for security analysis and prediction [29]. To that end, we conduct additional experiments in Subsection 3.4 to investigate if the similarity-based feature sets improve detection results when compared to relevant hyperlink-based feature sets.

2.4 Pre-process Android malware dataset

We estimate the effectiveness of the 8 feature sets listed in Table 2 for variant detection based on a widely-used dataset of Android malware: the AMD project (AMD) [5]. AMD collects 71 malware families, of which 23 families separate malware into fine-grained variant sets (at least two variant sets) based on different malicious behaviors [6]. We define such 23 families with 6724 malware as the dataset AMD-Va (shown in Table 3).

We devise 15-fold cross-validation to relieve the negative influence of over-fitting [33, 34]. For family "Lotoor" with 15 types of malware variants each type will be tested at least once and the remainders left are put into the training dataset. For the families with N ($N < 15$) types, we first adopt the leave-one-out strategy to obtain N-fold cross-validation dataset. Then, we randomly duplicate $15 - N$ folds of these N folds. The initial N folds and the duplicated $15 - N$ folds compose the 15-fold cross-validation dataset for the families with less than 15 types of malware variants. In this way, we carry out the 15-fold cross-validation experiment on AMD-Va. The issue of variant detection is thus to classify malware variant in the test dataset into the correct family based on the different well-trained multi-class classifiers, which simulates the real-world situation where learning knowledge from definite behaviors provided by the training dataset and detecting the test dataset with newly unknown behaviors.

The malware in the training-test dataset will be decompiled by Androguard[6] and extracted permission and API from "XML" and "smali" files respectively. Note that we omit the third party API, self-customized permission and API, because they are not presented by the official Android document and they have no verified semantic-level explanation. Then, each malware is formalized as a one-hot vector that describes the presence or absence of the given features of the feature sets in Subsection 2.3. The 1 in the one-hot vector refers to that this feature is used by malware, and 0 denotes that it is not. During training, only the one-hot vectors in the training dataset will be fed into the following machine learning and neural network classifiers. To avoid information bias, the vectors in the test dataset are only applied for the independent test.

[6] Androguard. https://androguard.readthedocs.io/en/latest/.

Table 3 Malware family distribution of AMD-Va

Malware family	#Variant set	#Malware
BankBot	8	647
Bankun	4	70
Boqx	2	215
Cova	2	17
DroidKungFu	6	546
FakeAngry	2	10
FakeInst	5	2168
FakePlayer	2	21
FakeTimer	2	12
Fusob	2	1270
GingerMaster	7	128
GoldDream	2	53
Koler	2	69
Leech	2	109
Lotoor	15	313
Mtk	3	67
Nandrobox	2	76
Opfake	2	10
RuMMS	4	402
SimpleLocker	4	158
SlemBunk	4	174
SmsKey	2	165
Zitmo	2	24
Total	–	6724

#Variant set is the number of variant set in a family.
#Malware is the number of malware in a family.

2.5 Multi-class classifier training and malware variant detection

We apply three machine learning multi-class classifiers and three neural network multi-class classifiers, both of which are commonly used for malware family detection or malware detection [1, 12, 35, 36], e.g., SVM [20], RF [21], KNN [22], MLP [23], CNN [24], and LSTM [25].

SVM is a popular supervised learning classifier applied for classification by separating hyperplane with the largest geometric interval under the help of various kernel functions. In the variant detection issue, we implement SVM with the one-against-one approach [37]. We construct $k_{\text{SVM}}(k_{\text{SVM}}-1)/2$ classifiers and each one trains malware from 2 out of k_{SVM} families ($k_{\text{SVM}}=23$ in AMD-Va). Therefore, the multi-class classification task is delegated to multiple binary-classification tasks, which are then used in conjunction with the max-win voting strategy to assign the unknown variant to the family with the most votes based on the results of the binary-classification tasks [38].

RF, an ensemble learning classifier, combines the predictions of independent decision tree classifiers [39] with an average algorithm. In particular, each decision tree in RF is built based on bootstrap malware from the training set. The candidate features are then chosen at random, and the best one is employed to split each node of a tree during the decision tree construction. Finally, RF takes the average probabilistic prediction of those decision trees and yields a better overall performance by reducing variance.

KNN is a neighbors-based supervised learning classifier. Without a general training process, KNN divides the feature vector space by using the labeled training set and considers the partition result of the space as the final algorithm model [22]. Given a new unlabeled malware (feature vector), we calculate the Euclidean distance in the feature vector space between this malware and one labeled malware. The k-nearest labeled malware will determine the family of the new unknown variant via a weighted max-win voting strategy [38].

MLP is a neural network that all layers are fully connected. We implement a basic MLP model that consists of an input layer, a hidden layer, and an output layer. Input malware features are represented by neurons in the input layer. In our experiments, the following hidden layer contains 1024 neurons and is activated by the activation function ReLU (rectified linear unit [40]). The output layer predicts the family of tested variants using a softmax activation function based on the values from the hidden layer.

We minimize the loss value of categorical cross entropy to find the best classification model and use Adam algorithm [41] to optimize the network weights and avoid local optimum.

CNN focuses on the local information and neurons of the convolutional layer share parameters with each other [24]. We design a CNN network including an input layer, an embedding layer, a convolutional layer, a flatten layer, and an output layer. A trainable embedding layer converts such low-dimension features into high-dimension features after receiving the malware representation from the input layer. The convolutional layer adopts 128 filters with a 1×1 convolutional kernel to scan the high-dimension feature for refining, in which the stride of each filter is set as 1. ReLU is the activation function of this layer. The 128-dimension feature vectors from the convolutional layer are then flattened via a flatten layer and fed into the output layer with a softmax activation function for classification. Similarly, we apply Adam optimization algorithm [41] and categorical cross entropy loss function during training.

LSTM is usually used for analyzing sequential data. Each neuron in LSTM consists of an input gate, a forget gate, and an output gate, which assists in adding, changing and removing the state of the neuron cell. Our LSTM network has an input layer, an embedding layer, an LSTM layer, and an output layer. The dimension of the represented malware feature vector of the input layer into a high dimension is raised by the trainable embedding layer. In the LSTM layer, the neuron is performed recurrently on each element of a neural feature vector and produces a 128-dimension hidden output after the last element of this vector is conducted. This hidden output vector is sent to the output layer which is activated by a softmax function to finally label the variant. We also employ Adam optimization algorithm [41] and categorical cross entropy loss function when training the LSTM neural network.

3 Experiments

In this section, we firstly introduce the baseline feature set from Axplorer [10] (Subsection 3.1). We then present the settings of hyperparameters for all 6 multi-class classifiers (Subsection 3.2). Finally, We design three research questions with sufficient experiments to evaluate the detection performance of ArgusDroid (Subsections 3.3–3.5). All experiments are conducted on a 4 GB of NVIDIA GeForce GTX 1050 Ti PC running Windows 10.

3.1 Baseline and evaluation metrics

The Android framework analysis tool Axplorer [10] performs a static analysis of the Android permission system to mine the sensitive behaviors of Android applications. It investigates 9 Android OS source code versions from Android 4.1 (API level 16, published on July 9, 2010) to Android 7.1 (API level 25, published on October 4, 2016), and skips the version of Android 4.4w (API level 20). Axplorer defines numerous mappings between features such as permission and API based on this data-driven approach. These features have been successfully deployed to solve the problem of permission misuse and malware analysis [10, 29]. At the same time, those relationships (mapping relations) across features can be used to mine knowledge to enhance the performance of malware classification [42]. We hence suggest taking the features with relations of Axplorer to find or reason the changeable malicious behaviors of variants. Note that both features of Axplorer and those of ArgusDroid contain mapping relations across features. The only difference between them is the method of construction, with the former (Axplorer) generated by a data-driven method without semantic knowledge and the latter (ArgusDroid) constructed by a knowledge-driven method with semantic knowledge from Android document. Moreover, because the AMD-Va was created in 2017, we select the mappings (between permission and API, with each API mapped to permission) about Android 7.1 provided by Axplorer. We choose the special mappings in AMD-Va dataset that use both permission and API and convert them into a feature set bs_axplorer. This also satisfies the open world assumption (OWA) [43] which states the observed intact mapping is true and vice versa. Finally, Axplorer produces the feature set bs_axplorer as a baseline with 105 features (14 permission and 91 API after dropping duplication) in total. The details of this feature set are available in our Github (see Section 1).

As the same as Axplorer, the previous work PScout [9] focuses on mining relations between permission and API. Pscout, on the other hand, only supports the Android 5.1 (API level 22, published in March 9, 2015), which does not correspond to the time of malware in the AMD-Va dataset created in 2017. We thus apply the newer Axplorer to build a feature set that will improve the relevant security analysis in terms of prediction precision [10]. Drebin [44], as a famous study in the field of malware analysis, releases many important features, such as permission, API, and intent which have been selected to explore and examine adversarial malware [12]. But the features shown by Drebin are independent of each other that do not have definite relations between different features. They are insufficient to address the issues that necessitate a thorough understanding of the malware behaviors. For instance, Wu et al. [8] used Drebin features as a foundation and further manually append textual descriptions from Android document, when interpreting which behavior affects an application's final label (malicious or benign). Zhang et al. [42] also provided the related description of API and implement graph-based relations of API in their feature set to slow down the classifier aging. In a nutshell, the features of Drebin are sometimes too general to solve the complex problem on analysis of malicious behaviors. We therefore assume that Drebin features are difficult to cover and find the variously changeable malicious behaviors of variants. In the later experiments, we discovered that the permission-API knowledge graph of ArgusDroid adds hyperlink-based or similarity-based relation for nearly half of the Drebin features used in [8] (e.g., 158 features). Those knowledge-driven relations in the proposed feature sets of ArgusDroid make a positive effect on variant detection which is discussed in Subsection 3.5.

Four widely used evaluation metrics, accuracy, precision, recall, and F1-score, are employed to measure the classification performance [45–47]. Accuracy (Acc) refers to accurate prediction results for the entire Android malware dataset. Precision (P) for each family f_i denotes the ratio of malware, which is the number of malware correctly categorized into f_i divided by the number of malware predicted as f_i. Recall (R) for each family f_i means the ratio of malware, which is the number of malware correctly categorized into f_i divided by the number of full ground-truth malware in f_i. F1-score (F1) is the harmonic mean value of recall and precision.

3.2 Hyperparameters

This subsection summarizes the crucial hyperparameter settings. During optimization, we use the feature set hop_three because it has 183 features which are close to the average feature number (217) of all feature sets. The best settings are applied to all experiments with various feature sets.

3.2.1 *Hyperparameters of machine learning classifiers*

We conduct grid search [48], an exhaustive search over specified hyperparameter values for a classifier, for determining the best hyperparameters of three machine learning classifiers. We still use the F1 value as the evaluation metric.

The linear kernel function is used in SVM. Its penalty parameter C and kernel parameter γ have a same search space $\{0.001, 0.01, 0.1, 1, 3, 5, 10\}$, and they are set at 1 and 0.001, respectively.

For the RF, we fix both the maximum depth of the decision tree max_depth and the number of decision trees in the forest n_estimators at 20, given by the values in $\{5, 10, 20, 40, 80, 100\}$. Meanwhile, the max_features, which defines the maximum feature number when looking for the best split, is the square root of the feature number of each feature set. The function of entropy (i.e., the information gain) measures the quality of such best split.

The KNN classifier takes n_neighbors at 17 (from 1 to 20) as the neighbors number. We calculate the Euclidean distance between two malware (one labeled and one unlabeled) in the created KNN feature vector space. After multiplying by a coefficient determined by the inverse of the distance between such two malware, we obtain a weighted label for the unlabeled malware and then vote by a max-win strategy.

3.2.2 *Hyperparameters of neural network classifiers*

For the three neural network classifiers, we concentrate on hyperparameters epoch, batch size and learning rate. We tune one hyperparameter by fixing others until we find the best value. F1 is also adopted to evaluate the results of different hyperparameters.

An epoch denotes that the complete training data has a forward propagation and a back propagation in the neural network [49]. Generally, one epoch cannot obtain the optimum model. With the increasing of the epoch number, the neural network gradually optimizes the model parameters based on the Adam algorithm. We apply the early stopping strategy [50] to track the loss value of validation data (also the test data in this work), and stop training when 10 epochs show no improvement.

In the neural network, batch size means the number of malware that are fed into the network during one epoch. We employ a mini-batch 16 to balance the GPU memory of our PC (4 GB of NVIDIA GeForce

GTX 1050 Ti) and the model training time of the neural network (e.g., training LSTM needs about 2.5 h). Furthermore, mini-batch accelerates model convergence by updating parameters more times, and avoids the partial optimum problem by the advantageous noise [51].

The learning rate handles the update rate of loss value in gradient direction by using the mini-batch strategy. A low learning rate lengthens the optimization process, while a high learning rate may cause non-convergence. We utilize the Adam algorithm to adjust the learning rates and update the neural network weights adaptively. In particular, the lr of Adam is set to 0.001, and the other hyperparameters are set to the recommended values cited in [41].

3.3 Research question 1 (RQ1)

RQ1: How do the hyperlink-based feature sets with different hops in the permission-API knowledge graph of ArgusDroid affect the variant detection results?

Motivation. As is shown in Table 2, we creates four hyperlink-based feature sets (hop_one, hop_two, hop_three, and hop_four) with different hops around the central permission based on the permission-API knowledge graph of ArgusDroid. Although the increase of hops yields more features from 23 in hop_one to 451 in hop_four, the redundant features rise because the description of the feature (source feature) often has more than one hyperlink to others features (target features) and not all of those target features preserve strong relation of the source feature. This may harm the detection of variants. The goal of this RQ is to explore if the hyperlink-based feature sets with different hops affect detection accuracy significantly. When extracting hyperlink-based feature sets from ArgusDroid for variant detection, we also want to find the boundary of hops, if possible.

Approach. We apply the four hyperlink-based feature sets hop_one, hop_two, hop_three, and hop_four to represent each malware as a one-hot feature vector. We train such feature vectors in the training data set by using the 6 multi-class classifiers, SVM, RF, KNN, MLP, CNN, and LSTM, as shown in Subsection 2.5, and adopt F1 value as the evaluation metrics for comparison on the test data set. Other metrics can be available in the Github[7]. In addition, we conduct T-test [52] at a confidence level of 95% to inspect the F1 differences for each classifier when hops rise, such as hop_one vs. hop_two, hop_two vs. hop_three, and hop_three vs. hop_four.

Result. Table 4 presents the average F1 (the average value of 15-fold) of all six classifiers by using different hyperlink-based feature sets. The hop_one with 32 features gets 0.5445 F1 by MLP. In hop_two, we observe the F1 of MLP becomes 0.8765. The performance of other classifiers is also improved, especially the CNN and LSTM whose F1 increases to 0.8187 and 0.7928, respectively. Then, the average F1 of MLP and CNN grows to a higher value 0.9582 and 0.9390 by adopting the feature set hop_three with 183 features. But the detection performances of SVM and LSTM drop to 0.4389 and 0.4143, respectively. In the hop_four with 451 features which is 2.5 times of hop_three, we find that the MLP and CNN still have areas for improvement. Their average F1 values are 0.9613 and 0.9582, respectively. However, the performance of SVM, RF, KNN, and LSTM decreases by 0.0211–0.0981. This implies that the enhancing of features with larger hops, especially from hop_three to hop_four, is not able to continually improve the ability of classifiers for detecting. These three machine learning classifiers even reduce the F1 value, which is caused by overfitting [33]. Among the three neural network classifiers, the LSTM performs the worst with the F1 value decreasing as feature sets with more hops increase. This implies that the feature vector of malware is not a true context-sensitive natural language sentence that fits for LSTM [25, 46], although the features of ArgusDroid in this RQ are related by hyperlinks. Meanwhile, a larger feature set with more hops may enlarge the sparsity of the malware feature vector. We thus conclude that the neural networks MLP and CNN have better capabilities for variant detection by converting the one-hot feature vector into a neural feature vector (different from that of machine learning classifiers).

The boxplot in Figure 2 illustrates the robustness of each classifier when using different feature sets with 360 F1 values in total (6 classifiers × 4 feature sets × 15 runs). We can see that both MLP-hop_four and CNN-hop_four have an outlier with a very low F1 0.5867 and 0.5533, respectively, although they obtain a very large average F1 in Table 4. Following a manual investigation, we discover that these two outliers are from the same training-test dataset in which the other four classifiers obtain lower F1 values (e.g., 0.5711 of SVM, 0.5081 of RF, 0.4089 of KNN, 0.1212 of LSTM). Furthermore, in this training-test dataset, all of the 6 classifiers completely misjudge 10 out of 23 families at least (i.e., F1 value is 0). This may be related to the construction of the variant set of AMD-Va. In particular, the division of variant sets

[7] https://github.com/qWe1aSd/varPre.

Table 4 Average F1 of feature sets by different hops

Classifier	hop_one	hop_two	hop_three	hop_four
SVM	0.2650	0.5817	0.4389	0.4054
RF	0.2137	0.3899	0.4006	0.3794
KNN	0.2225	0.2300	0.3376	0.3162
MLP	**0.5445**	**0.8765**	**0.9582**	**0.9613**
CNN	0.4350	0.8187	0.9390	0.9582
LSTM	0.4806	0.7928	0.4143	0.3162

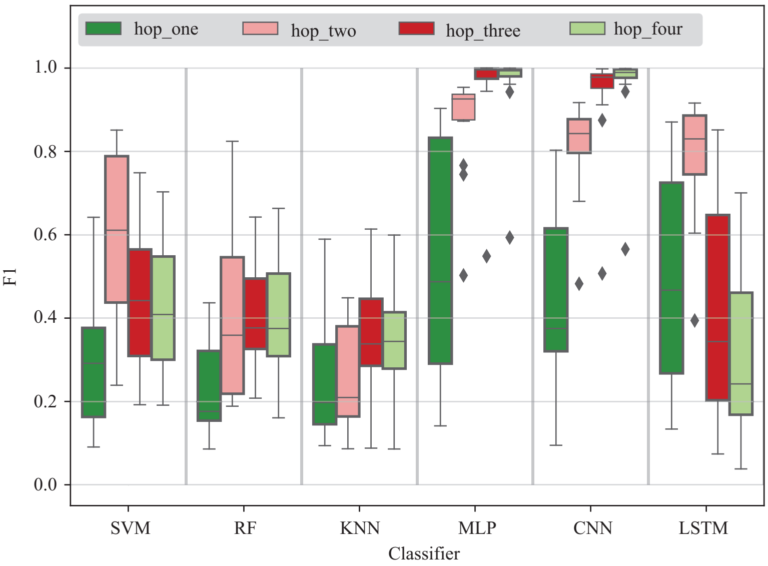

Figure 2 (Color online) F1 of feature sets by different hops.

in each family of AMD-Va is based on malicious behaviors, and similar malicious behaviors will appear in multiple malware families [5]. For example, malware families "BankBot" and "DroidKungFu" have the same behavior "by host App" which is a typical way to activate adware. If a variant in "BankBot" has only this behavior "by host App", it may be identified as a variant in "DroidKungFu". Therefore, when we randomly select variant sets of each family to build the 15-fold training and test dataset, it may lead to the outlier of results. After removing this particular training and test dataset, other 14-fold datasets achieve F1 values of more than 0.9000 by MLP and CNN classifiers with feature set hop_four. And MLP and CNN have the lowest standard deviation of the F1 value, 0.0131 for MLP and 0.0121 for CNN. They are significantly lower than the F1 standard deviations of the other four classifiers (which is greater than 0.0877). MLP and CNN, on the other hand, have a larger F1 standard deviation 0.0149 when changing the other three hyperlink-based feature sets rather than hop_four. This reveals that the combination of hop_four and MLP or CNN yields the best robustness.

We then analyze the F1 differences by T-test between every two adjacent hops shown in Table 5. The performance of CNN is significantly different in the situation hop_one vs. hop_two and hop_two vs. hop_three with the p-value < 0.05. But the p-value of hop_three vs. hop_four is 0.6518 which denotes a few profits of performance improvement. For the classifier MLP, the F1 differences are significant (p-value < 0.05) only when we apply hop_two as a feature set to substitute hop_one. From hop_two to hop_three, it has low significant differences with p-value $= 0.0594$. The hop_four also fails to significantly improve the performance because of a large p-value $= 0.9373$. This infers that increasing the hops dose not always significantly improve the results of CNN and MLP, despite the fact that they have the best effectiveness and robustness when the hops are increased. In addition, all the other four classifiers obtain a large p-value of more than 0.3087 by adding features from hop_three to hop_four. The constraint imposed by the generally functional features may affect such a result. For example, the API "run()", which is added in hop_four and launches a thread, may not be a distinctly sensitive feature to distinguish variant. When extracting hyperlink-based features from ArgusDroid proposed permission-API knowledge graph, we consider four hops as the boundary.

Table 5 *p*-value of feature sets by adjacent hops

Classifier	hop_one vs. hop_two	hop_two vs. hop_three	hop_three vs. hop_four
SVM	0.000200912	0.065483681	0.606706176
RF	0.013553086	0.876125197	0.699047468
KNN	0.883787351	0.036879828	0.669900077
MLP	0.001136318	0.059440247	0.937319101
CNN	0.000006332	0.007133757	0.651829669
LSTM	0.000992391	0.000137097	0.308723315

With more hops for achieving various hyperlink-based features from the graph, CNN and MLP classifiers significantly obtain better detection results for the malware variants. But the improvement of performance by increasing features with more hops is gradually decreasing from hop_one to hop_four. In the proposed permission-API knowledge graph of ArgusDroid, we limit the number of the hop as 4 and adopt the feature set hop_four to represent the sensitive features for detecting malware variants.

3.4 Research question 2 (RQ2)

RQ2: When compared to hyperlink-based feature sets, whether the detection results are improved further by adopting relevant similarity-based feature sets?

Motivation. For each hyperlink-based feature set, ArgusDroid provides its related similarity-based feature set by computing the semantic similarity across features. Thus two features with similar semantic descriptions, which are not hyperlinked to each other such as READ_PHONE_STATE (access to mobile phone state) and ACCESS _NETWORK_STATE (access network state), will be added a similarity-based relation. This enhances the completeness of ArgusDroid's permission-API knowledge graph. Based on those generated similarity-based feature sets, we determine whether the detection result is improved further when compared to the relevant hyperlink-based feature sets in this RQ.

Approach. For each similarity-based feature set, we conduct experiments to compare with the relevant hyperlink-based feature set, e.g., hop_one vs. hop_one_sim, hop_two vs. hop_two_sim, hop_three vs. hop_three_sim, and hop_four vs. hop_four_sim. We train the 6 classifiers by the training variants and evaluate the classification result by the test variants. Furthermore, we adopt the T-test to compare the significant differences in F1 value between hyperlink-based feature sets and related similarity-based feature sets.

Results. As shown in Table 6, the F1 of all 6 classifiers benefit from the 29 new similarity-based features in hop_one_sim compared with that of hop_one. The three neural network classifiers achieve more than 0.7860 F1 values by hop_one_sim while the hop_one gives F1 values less than 0.5445. The results of the three machine learning classifiers also rise but they are less than 0.3952. The hop_two_sim supplements 41 new features based on the 50 features in hop_two. We observe that MLP and CNN significantly enhance the F1 from 0.8765, 0.8187 to 0.9457, 0.9160 respectively, whereas LSTM and SVM degrade of classification performance. With the increasing of features from 183 in hop_three to 289 in hop_three_sim, the MLP and CNN still have relatively smaller performance boost. Meanwhile, we believe that the other four classifiers are incapable of accurately predicting variants in this situation (F1 is less than 0.4389). Finally, we observe the feature sets hop_four and hop_four_sim. Although the latter adds 129 new features, the MLP and CNN with the best performance begin to marginally decrease the F1 value. In short, when the number of hops is small (one or two), the similarity-based feature set covers more key features that benefit distinguishing variants by MLP and CNN than its related hyperlink-based feature set. If the hop is enlarged to three, the hyperlink-based feature set is already effective enough for classification by MLP and CNN. For the boundary of four hops in Subsection 3.3, it is no longer to append similarity-based features to enhance the capabilities of classifiers MLP and CNN.

In the boxplots of Figure 3, we mainly estimate the performance of three neural network classifiers (by using different feature sets) after removing the results of the unique training-test dataset with outlier, since the average F1 values of three machine learning classifiers are less than 0.5817 (see in Table 6). At first, the generalizability of LSTM and MLP largely grows only in hop_one vs. hop_one_sim, by a significant difference with p-value < 0.05 as shown in Table 7. Although the MLP still improves the average F1 value by 0.0692 and reduces the standard deviation by 0.0435 in hop_two vs. hop_two_sim, such improvement is not statistically significant due to the fact p-value $= 0.1177$. For the classifier CNN, both the detection result and model robustness significantly benefit from the feature increasing from

Table 6 Average F1 of feature set hop_x compared with feature set hop_x_sim. x is one, two, three or four

Classifier	hop_one	hop_one_sim	hop_two	hop_two_sim	hop_three	hop_three_sim	hop_four	hop_four_sim
SVM	0.2650	0.3952	0.5817	0.3873	0.4389	0.4262	0.4054	0.4147
RF	0.2137	0.3434	0.3899	0.3771	0.4006	0.3747	0.3794	0.3768
KNN	0.2225	0.3023	0.2300	0.3444	0.3376	0.3370	0.3162	0.3217
MLP	**0.5445**	**0.9063**	**0.8765**	**0.9457**	**0.9582**	**0.9589**	**0.9613**	**0.9588**
CNN	0.4350	0.7876	0.8187	0.9160	0.9390	0.9468	0.9582	0.9497
LSTM	0.4806	0.7860	0.7928	0.6303	0.4143	0.4416	0.3162	0.2882

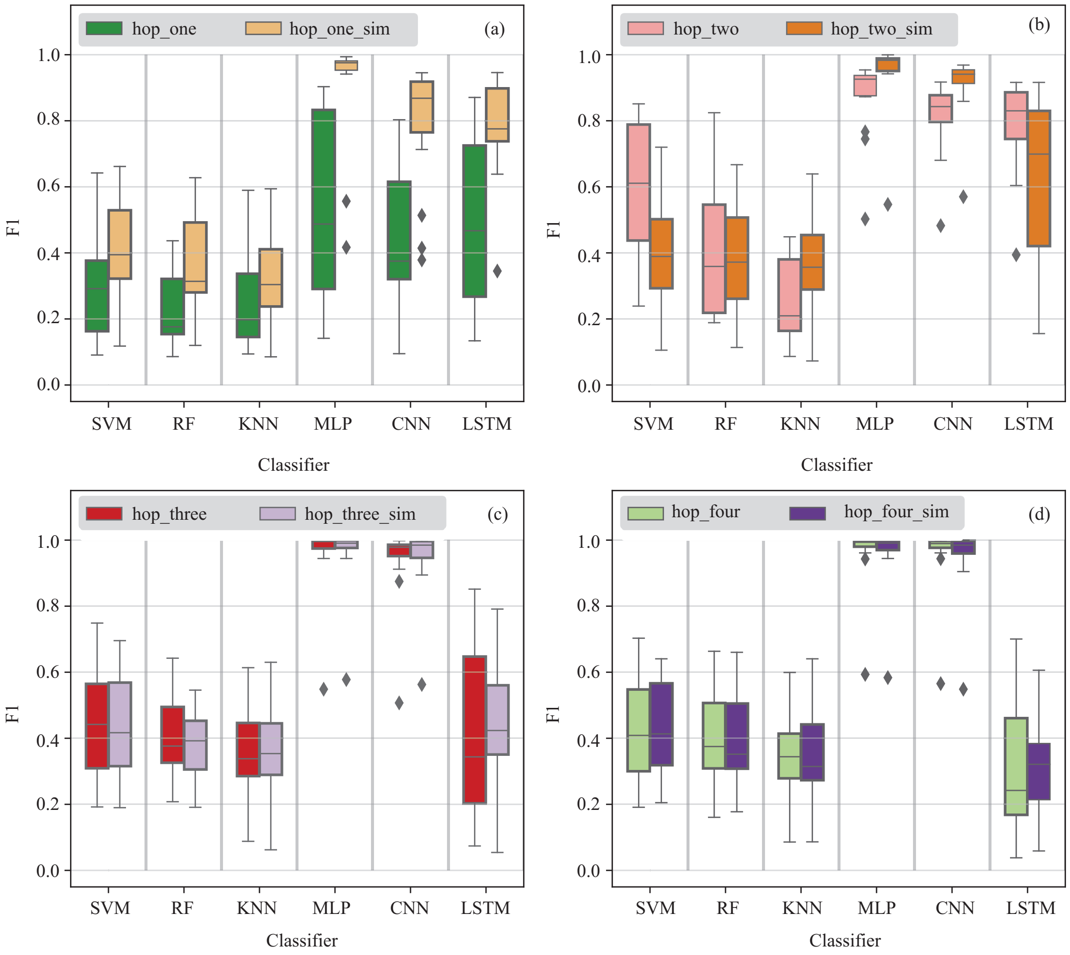

Figure 3 (Color online) F1 of feature set hop_x compared with feature set hop_x_sim. x is (a) one, (b) two, (c) three, and (d) four.

hop_one to hop_one_sim and from hop_two to hop_two_sim (p-value < 0.05). However, none of the three classifiers obtains significant performance enhancement in the comparison hop_three vs. hop_three_sim and hop_four vs. hop_four_sim. This also indirectly proves that if a hyperlink-based feature set (e.g., hop_four) has retained sufficient critical features, adding similarity-based features will not result in additional performance benefits.

In summary, we confirm the hop_four (451 features) as the optimal feature set of ArgusDroid for variant detection on AMD-Va, according to the specific analysis followed. Firstly, the combination of MLP-hop_four achieves the highest average F1 0.9613 and the lowest standard variance 0.0131. Secondly, we discover that the hop_four contains more than 78.90% features in the feature sets hop_one_sim and hop_two_sim which both improve the detection results shown in Table 6. This infers that the important and only-similarity-related features will be indirectly hyperlinked to each other by increasing the number of hops in ArgusDroid's permission-API knowledge graph. For instance, the dangerous feature "READ_CONTACTS" in hop_one_sim, which allows an application to read the user's contact data and

Table 7 p-value of feature set hop_x and feature set hop_x_sim. x is one, two, three or four

Classifier	hop_one vs. hop_one_sim	hop_two vs. hop_two_sim	hop_three vs. hop_three_sim	hop_four vs. hop_four_sim
SVM	0.058011553	0.017132755	0.850642748	0.876386463
RF	0.022952976	0.866893536	0.604232673	0.963780937
KNN	0.160124376	0.033774541	0.990125762	0.915306062
MLP	0.000790387	0.117696214	0.984602428	0.948913354
CNN	0.000093434	0.014752862	0.855242252	0.838527766
LSTM	0.001459784	0.035285615	0.770357454	0.705199182

aids in the detection of malicious behaviors, is also present in hop_four. Finally, when the number of hop is large, both the hyperlink-based feature sets and the similarity-based feature sets generate more redundant API (e.g., "longValue()" to return a long value). So the boundary feature set hop_four proposed in Subsection 3.3 is the best one for the task of variant detection.

The new similarity-based features make certain positive effects on predicting the malware variants. Specifically, when adopting features in one or two hops (hop_one and hop_two), the similarity-based features (hop_one_sim and hop_two_sim) covers more useful features to prompt variant identification. But its positive influence gradually reduces with the increase of hops which hop_three_sim and hop_four_sim fail to further significantly enhance the detection result.

3.5 Research question 3 (RQ3)

RQ3: How does the performance of the proposed feature set hop_four compare to that of bs_axplorer?

Motivation. The optimal feature set hop_four of ArgusDroid is generated by the knowledge-driven method, while the contrast feature set bs_axplorer of Axplorer in Subsection 3.1 is obtained by the data-driven method. In this RQ, we evaluate the effectiveness on detecting variant by using the knowledge-driven hop_four and the data-driven bs_axplorer respectively.

Approach. Based on the result of Subsections 3.3 and 3.4, we adopt hop_four as the feature set to compare with the baseline bs_axplorer from Axplorer. Each malware in AMD-Va is then formed as a one-hot feature vector based on those two feature sets. We classify the malware variants in the test dataset by applying the well-trained machine learning and neural network classifiers and the metric F1 value. The T-test [52] with of 95% confidence level is implemented to verify the F1 differences between hop_four and bs_axplorer.

Results. Table 8 shows the average F1 value of the 15 runs of cross validation 12 combinations of classifier and feature set (6 classifiers × 2 feature sets). We find that the performance by adopting hop_four for classification outperforms that of bs_axplorer, except the LSTM classifier. When detecting variants by bs_axplorer, only the MLP achieves a F1 value of 0.6038 over 0.5000. But the combinations of MLP-hop_four gains the largest F1 0.9613. We further observe the boxplots in Figure 4 after removing the results of the only training-test dataset with outlier, and discover that the standard variance of F1 by the MLP-bs_axplorer (0.2017) is far larger than that by the MLP-hop_four (0.0131). At the same time, the combinations of CNN-hop_four has a lower F1 standard variance 0.0121 than that by the CNN-hop_four (0.1625). In addition, we calculate the F1 differences by T-test [52] between bs_axplorer and hop_four for each classifier. Our T-test results (see in Table 9) reveals that 5 out of 6 classifiers have statistically significant F1 differences (p-value < 0.05), except the classifier LSTM with p-value 0.4797. As a result, when using classifier MLP and CNN, the knowledge-driven hop_four from ArgusDroid outperforms the data-driven baseline bs_axplorer n terms of both F1 value and robustness.

Furthermore, in order to deep understand the reasons why there are different results between the knowledge-driven hop_four and the data-driven bs_axplorer, we manually analyze the 451 features in hop_four and the 105 features in bs_axplorer. First of all, hop_four contains more features that are helpful to represent malicious behaviors. Secondly, both the two feature sets have representative features to describe behaviors such as "ACCESS_NETWORK_STATE" ("allow applications to access information about networks") and "getLine1Number()" ("returns the phone number string"). However, the bs_axplorer provided by Axplorer can only illustrate the general relation between "getLine1Number()" and "ACCESS_NETWORK_STATE" through their association from the Android source code [10], which lacks relevant semantic knowledge. The hop_four built by ArgusDroid can provide more various behaviors depending on a series of hyperlink-based or similarity-based (if needed) features. Take the Figure 1 as

Table 8   Average F1 of feature set hop_four compared with baseline

Classifier	bs_axplorer	hop_four
SVM	0.2386	0.4054
RF	0.2281	0.3794
KNN	0.1865	0.3162
MLP	**0.6038**	**0.9613**
CNN	0.4629	0.9582
LSTM	0.3754	0.3162

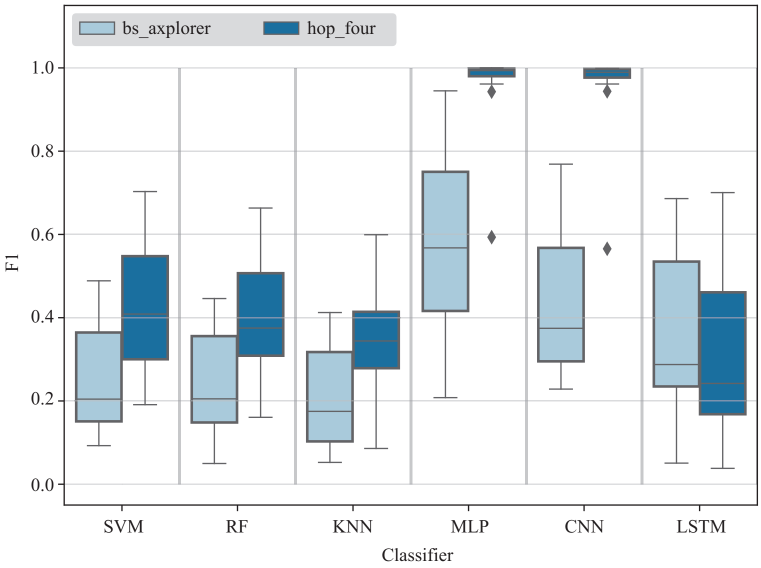

Figure 4   (Color online) F1 of feature set hop_four compared with baseline.

Table 9   *p*-value of feature set hop_four and baseline

Classifier	bs_axplorer vs. hop_four
SVM	0.006345369
RF	0.007742808
KNN	0.005822548
MLP	0.000018344
CNN	0.000000002
LSTM	0.479748802

an example, through the hyperlink, we can find that ArgusDroid produces hyperlink-based features of “getLine1Number()” including “READ_PHONE_STATE” (“allows read only access to phone state”) and “READ_SMS” (“allows an application to read SMS messages”). They can form more changeable and diverse malicious behaviors that may be implemented in variants that allow attackers to obtain the phone number and phone state with a default SMS function [8, 13, 27]. In conclusion, the knowledge-driven hop_four, constructed by the permission-API knowledge graph of ArgusDroid, not only contains enough critical features but also preserves potential relations and interpretability across different features. We can detect variants more accurately by identifying various malicious behavior changes in malware based on such correlated features in hop_four.

Compared with baseline bs_axplorer, the knowledge-driven feature set hop_four from permission-API knowledge graph are statistically significantly beneficial for variant detection with MLP and CNN. This confirms that the knowledge-driven features of ArgusDroid not only cover more beneficial features for variant prediction, but also precisely represent the potential relations across features which is useful for discovering variants with unknown behaviors.

4 Threats to validity

An internal threat is derived from the 15 training-test dataset created for cross-validation. Each fold has a different scale of malware in the training dataset and test dataset, although there is 6724 malware in total. The largest training dataset, for example, contains 5853 malware (the relevant test dataset consists of 583 malware), whereas the smallest has only 3564 malware (the relevant test dataset consists of 3160 malware). However, the cross-validation results disclose that this imbalance does not have a significant impact on our malware family variant detection findings. When we apply the proposed knowledge-driven feature set hop_four, MLP and CNN achieve very low standard variance of 0.0131 and 0.0121, respectively.

Another internal threat is related to the features that have been defined to represent malicious behavior. ArgusDroid focuses on the permission and risky API as their validity has been verified by many literatures [13,14]. In the basic graph, we disregard the class because it cannot indicate a specific behavior like permission or API. We also do not adopt CFG (control flow graph) [1,7] to represent behaviors. The reason for this is that developing and analyzing an application's CFG is a time-consuming and complex process. In addition, the knowledge-driven features are provided by hyperlink-based and similarity-based relations between permission and API or between permission and permission from the knowledge graph. This stems from the assumption that only the API related to permission is critical for recognizing malicious behavior [10,13].

An external threat is the quality and representativeness of the Android variant dataset in this study. AMD [5] has been carefully created by Android security experts and widely adopted in the literature of malware analysis and detection [53,54]. The AMD-Va in this work is a subset of AMD including 23 malware families and 6724 malware. We believe that the quality of this dataset is well guaranteed and the application in it has no issue of "sample duplication" [55]. Meanwhile, it covers sufficient Android malware distributed over a long period of time from 2010 to 2016 [5]. As a result, AMD-Va malware should be representative as well. In addition, we can go a step further to validate the generalizability of our findings on the new variant dataset to validate its efficiency when dealing with the problem of "time inconsistency" [56].

5 Related work

When analyzing and detecting Android malware, a critical issue is how to find the newly unknown malware, which is the variants of labeled malware [5]. Many studies have concentrated on the analysis of malware variant [15,57,58]. "MONET" [57] was a malware variant detection tool that was designed and implemented by combining the control flow graph from static structures and data flow graph from runtime behaviors. Yang et al. [58] presented "malware recomposition variation" which uses special mutation strategies on malicious behaviors to systematically construct new variants, and proposed three defense mechanisms to counteract it. Meng et al. [15] developed a precise semantic model "deterministic symbolic automaton" to capture common malicious behaviors, and then classified malware based on the given common attack patterns. It should be noted that the focus of these studies is on distinguishing malware variants from benign applications. ArgusDroid, on the other hand, goes a step further to categorize a variant into the appropriate malware family, which is formulated as a multi-class family classification issue.

All of the previously classification tasks necessitate defining or constructing the malicious behaviors that aid in the discrimination of various malware variants. Wang et al. [13] investigated the permission-induced risk in Android applications and evaluated the efficacy of risky permissions for malware detection via multiple machine learning models. RevealDroid [14] detected malware and determined their families by extracting the sensitive API, Intent, and package API invocations under supervised learning. Meanwhile, the graph-based malicious features proposed by FalDroid [1], which constructed frequent sub-graphs to represent the common malicious behaviors of malware, can automatically classify Android malware into relevant families. The malicious behaviors created in such studies, however, are dependant on systematically analyzing a large amount of malware and the Android OS source code. This data-driven approach is time-consuming and necessitates a significant amount of effort on the part of security professionals. Furthermore, those data-driven features are still too general, making it difficult to recognize minor malicious behavior changes which are critical for variant detection. In opposite, ArgusDroid applies a knowledge-driven way to establish malicious behaviors based on the official Android document, which contains semantic relation across features to better predict malware variants. Additionally, ArgusDroid further mines a more meaningful explanation of relations from the official document than the comparable Axplorer [10], which only superficially expresses such relation by the source code statistics.

The adoption of Android document is triggered by its efficiency on security analysis in [8,16,18,42,59]. Ren et al. [16] demystified the Android API usage directives by text mining technique to help developers understand and debug API misuse problems more efficiently. The CDA (characterising deprecated API [59]) was a prototype tool for the investigation of the Android ecosystem. It was applied to characterize deprecated Android APIs by examing the relevant documentation and annotations. An API caveats knowledge graph was constructed in [18] for the functionality of Android APIs, which adopted natural language processing to construct a task-specific knowledge graph to assist users in avoiding unintended APIs. Different from studies above, ArgusDroid aims to pick out the sensitive features that can represent and even reasoning malicious behaviors of malware, as our purpose concentrates on malware variant detection. As a result, we developed a permission-API knowledge graph to generate sensitive features such as permission and related key API. Furthermore, the description of each permission and API (i.e., the attribute of entity in the graph) can deepen understanding of their relationships and thus strengthen comprehension of malicious behavior [60].

6 Conclusion

In this paper, we construct a knowledge-driven malware family variants detection system ArgusDroid. It constructs a permission-API knowledge graph from the official Android document to represent the semantic relationship of features (e.g., permission or API) and uses this knowledge graph to build different feature sets based on the given hyperlink-based and similarity-based strategies. We validate ArgusDroid's effectiveness by detecting malware variants in the AMD-Va dataset. The experiments reveal that increasing of the number of hops improves detection performance when MLP and CNN classifiers are adopted. But the performance boundary emerges when we take feature set hop_four with 451 features. Additionally, the newly added similarity-based features significantly improve the variant detection only when we use feature sets generated by less than two hops. We finally compare the proposed feature set hop_four with the baseline bs_axplorer. We confirm that the knowledge-driven hop_four of ArgusDroid achieves more than 0.3575 F1 growth when using classifier MLP, which profits variant identification with unknown behaviors. In the future, we will further explore how to better mine and represent the relationships between features in the Android document, as well as enrich the knowledge graph of ArgusDroid to solve problems such as malicious behavior interpretation and variant analysis.

Acknowledgements This work was partially supported by National Natural Science Foundation of China (Grant Nos. 62102284, 61872262).

References

1 Fan M, Liu J, Luo X, et al. Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Trans Inform Forensic Secur, 2018, 13: 1890–1905

2 Bai Y D, Xing Z C, Li X H, et al. Unsuccessful story about few shot malware family classification and siamese network to the rescue. In: Proceedings of the 42nd International Conference on Software Engineering, Seoul, 2020. 1560–1571

3 Sebastián M, Rivera R, Kotzias P, et al. Avclass: a tool for massive malware labeling. In: Proceedings of International Symposium on Research in Attacks, Intrusions, and Defenses, Telecom SudParis, 2016. 230–253

4 Sebastián S, Caballero J. AVClass2: massive malware tag extraction from AV labels. In: Proceedings of Annual Computer Security Applications Conference, Austin, 2020. 42–53

5 Wei F G, Li Y P, Roy S, et al. Deep ground truth analysis of current Android malware. In: Proceedings of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Bonn, 2017. 252–276

6 Li Y P, Jang J Y, Hu X, et al. Android malware clustering through malicious payload mining. In: Proceedings of International Symposium on Research in Attacks, Intrusions, and Defenses, Atlanta, 2017. 192–214

7 Fan M, Luo X P, Liu J, et al. Graph embedding based familial analysis of Android malware using unsupervised learning. In: Proceedings of the 41st International Conference on Software Engineering, Montreal, 2019. 771–782

8 Wu B, Chen S, Gao C, et al. Why an Android app is classified as malware. ACM Trans Softw Eng Methodol, 2021, 30: 1–29

9 Au K W Y, Zhou Y F, Huang Z, et al. Pscout: analyzing the Android permission specification. In: Proceedings of ACM Conference on Computer and Communications Security, Raleigh, 2012. 217–228

10 Backes M, Bugiel S, Derr E, et al. On demystifying the Android application framework: re-visiting Android permission specification analysis. In: Proceedings of the 25th USENIX Security Symposium, Austin, 2016. 1101–1118

11 Chen S, Xue M H, Tang Z S, et al. Stormdroid: a streaminglized machine learning-based system for detecting Android malware. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, Xi'an, 2016. 377–388

12 Chen S, Xue M, Fan L, et al. Automated poisoning attacks and defenses in malware detection systems: an adversarial machine learning approach. Comput Secur, 2018, 73: 326–344

13 Wang W, Wang X, Feng D, et al. Exploring permission-induced risk in Android applications for malicious application detection. IEEE TransInformForensic Secur, 2014, 9: 1869–1882

14 Garcia J, Hammad M, Malek S. Lightweight, obfuscation-resilient detection and family identification of Android malware. ACM Trans Softw Eng Methodol, 2018, 26: 1–29

15 Meng G Z, Xue Y X, Xu Z Z, et al. Semantic modelling of Android malware for effective malware comprehension, detection, and classification. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, 2016. 306–317

16 Ren X X, Sun J M, Xing Z C, et al. Demystify official API usage directives with crowdsourced API misuse scenarios, erroneous code examples and patches. In: Proceedings of the 42nd International Conference on Software Engineering, Seoul, 2020. 925–936

17 Ma S Y, Xing Z C, Chen C Y, et al. Easy-to-deploy api extraction by multi-level feature embedding and transfer learning. In: Proceedings of the 41st International Conference on Software Engineering, Montreal, 2019

18 Li H W, Li S R, Sun J M, et al. Improving API caveats accessibility by mining API caveats knowledge graph. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, 2018. 183–193

19 Fan M, Wei W, Xie X, et al. Can we trust your explanations? Sanity checks for interpreters in Android malware analysis. IEEE Trans Inform Forensic Secur, 2021, 16: 838–853

20 Hsu C W, Lin C J. A comparison of methods for multiclass support vector machines. IEEE Trans Neural Netw, 2002, 13: 415–425

21 Breiman L. Random forests. Mach Learn, 2001, 45: 5–32

22 Roussopoulos N, Kelley S, Vincent F. Nearest neighbor queries. ACM Sigmod Record, 1995, 24: 71–79

23 Collobert R, Bengio S. Links between perceptrons, MLPs and SVMs. In: Proceedings of the 21st International Conference on Machine Learning, Alberta, 2004

24 Kim Y. Convolutional neural networks for sentence classification. In: Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, 2014. 1746–1751

25 Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput, 1997, 9: 1735–1780

26 Chen C. Similarapi: mining analogical apis for library migration. In: Proceedings of the 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Seoul, 2020. 37–40

27 Fan M, Yu L, Chen S, et al. An empirical evaluation of GDPR compliance violations in Android mhealth Apps. In: Proceedings of the 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, 2020. 253–264

28 Ye X, Shen H, Ma X, et al. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th International Conference on Software Engineering, Austin, 2016. 404–415

29 Liu Z, Xia X, Lo D, et al. Automatic, highly accurate app permission recommendation. Autom Softw Eng, 2019, 26: 241–274

30 Ramos J. Using TF-IDF to determine word relevance in document queries. In: Proceedings of the 1st Instructional Conference on Machine Learning, New Jersey, 2003. 133–142

31 Aizawa A. An information-theoretic perspective of tf-idf measures. Inform Process Manag, 2003, 39: 45–65

32 Pennington J, Socher R, Manning C D. Glove: global vectors for word representation. In: Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, 2014. 1532–1543

33 Hawkins D M. The problem of overfitting. J Chem Inf Comput Sci, 2004, 44: 1–12

34 Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, 1995. 1137–1143

35 McLaughlin N, del Rincon J M, Kang B, et al. Deep Android malware detection. In: Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy, Scottsdale, 2017. 301–308

36 Pendlebury F, Pierazzi F, Jordaney R, et al. Tesseract: eliminating experimental bias in malware classification across space and time. In: Proceedings of the 28th USENIX Security Symposium, Santa Clara, 2019. 729–746

37 Chang C C, Lin C J. LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol, 2011, 2: 1–27

38 Duan K B, Keerthi S S. Which is the best multiclass SVM method? An empirical study. In: Proceedings of International Workshop on Multiple Classifier Systems, Seaside, 2005. 278–285

39 Safavian S R, Landgrebe D. A survey of decision tree classifier methodology. IEEE Trans Syst Man Cybern, 1991, 21: 660–674

40 Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, Ft. Lauderdale, 2011. 315–323

41 Kingma D P, Ba J. Adam: a method for stochastic optimization. 2014. ArXiv:1412.6980

42 Zhang X H, Zhang Y, Zhong M, et al. Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS), 2020. 757–770

43 Minker J. On indefinite databases and the closed world assumption. In: Proceedings of International Conference on Automated Deduction, 1982. 292–308

44 Arp D, Spreitzenbarth M, Hübner M, et al. DREBIN: effective and explainable detection of Android malware in your pocket. In: Proceedings of Network & Distributed System Security Symposium (NDSS), California, 2014. 23–26

45 Guo Q Y, Chen S, Xie X F, et al. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), California, 2019. 810–822

46 Han Z B, Li X H, Xing Z C, et al. Learning to predict severity of software vulnerability using only vulnerability description. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, 2017. 125–136

47 Ren X, Xing Z, Xia X, et al. Neural network-based detection of self-admitted technical debt. ACM Trans Softw Eng Methodol, 2019, 28: 1–45

48 Ndiaye E, Le T, Fercoq O, et al. Safe grid search with optimal complexity. 2018. ArXiv:1810.05471

49 Hecht-Nielsen R. Theory of the backpropagation neural network. In: Proceedings of International Joint Conference on Neural

Networks, 1989

50 Prechelt L. Early stopping — but when? In: Neural Networks: Tricks of the Trade. Berlin: Springer, 1998. 55–69

51 Masters D, Luschi C. Revisiting small batch training for deep neural networks. 2018. ArXiv:1804.07612

52 Plackett R L. Karl Pearson and the Chi-squared test. Int Stat Rev Revue Inte de Statistique, 1983, 51: 59–72

53 Wei F, Roy S, Ou X. Amandroid: a precise and general inter-component data flow analysis framework for security vetting of Android apps. ACM Trans Priv Secur, 2018, 21: 1–32

54 Pei X, Yu L, Tian S. AMalNet: a deep learning framework based on graph convolutional networks for malware detection. Comput Secur, 2020, 93: 101792

55 Zhao Y, Li L, Wang H, et al. On the impact of sample duplication in machine-learning-based Android malware detection. ACM Trans Softw Eng Methodol, 2021, 30: 1–38

56 Li L, Bissyandé T, Klein J. Moonlightbox: mining Android API histories for uncovering release-time inconsistencies. In: Proceedings of the 29th IEEE International Symposium on Software Reliability Engineering (ISSRE), Memphis, 2018. 212–223

57 Sun M, Li X, Lui J C S, et al. Monet: a user-oriented behavior-based malware variants detection system for Android. IEEE Trans Inform Forensic Secur, 2017, 12: 1103–1112

58 Yang W, Kong D G, Xie T, et al. Malware detection in adversarial settings: exploiting feature evolutions and confusions in Android Apps. In: Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC), Orlando, 2017. 288–302

59 Li L, Gao J, Bissyandé T F, et al. CDA: characterising deprecated Android APIs. Empir Softw Eng, 2020, 25: 2058–2098

60 Han Z B, Li X H, Liu H T, et al. Deepweak: reasoning common software weaknesses via knowledge graph embedding. In: Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), Campobasso, 2018. 456–466