IEICE TRANS. INF. & SYST., VOL.E102–D, NO.6 JUNE 2019

PAPER

An Effective Feature Selection Scheme for Android ICC-Based Malware Detection Using the Gap of the Appearance Ratio

Kyohei OSUGE[†a)], **Hiroya KATO**[†b)], **Shuichiro HARUTA**[†c)], ***Student Members***, ***and*** **Iwao SASASE**[†d)], ***Fellow***

SUMMARY Android malwares are rapidly becoming a potential threat to users. Among several Android malware detection schemes, the scheme using Inter-Component Communication (ICC) is gathering attention. That scheme extracts numerous ICC-related features to detect malwares by machine learning. In order to mitigate the degradation of detection performance caused by redundant features, Correlation-based Feature Selection (CFS) is applied to feature before machine learning. CFS selects useful features for detection in accordance with the theory that a good feature subset has little correlation with mutual features. However, CFS may remove useful ICC-related features because of strong correlation between them. In this paper, we propose an effective feature selection scheme for Android ICC-based malware detection using the gap of the appearance ratio. We argue that the features frequently appearing in either benign apps or malwares are useful for malware detection, even if they are strongly correlated with each other. To select useful features based on our argument, we introduce the proportion of the appearance ratio of a feature between benign apps and malwares. Since the proportion can represent whether a feature frequently appears in either benign apps or malwares, this metric is useful for feature selection based on our argument. Unfortunately, the proportion is ineffective when a feature appears only once in all apps. Thus, we also introduce the difference of the appearance ratio of a feature between benign apps and malwares. Since the difference simply represents the gap of the appearance ratio, we can select useful features by using this metric when such a situation occurs. By computer simulation with real dataset, we demonstrate our scheme improves detection accuracy by selecting the useful features discarded in the previous scheme.

key words: *android, malware detection, ICC, feature selection*

1. Introduction

Smartphones have been widely used in people's daily life, such as online banking, automated home control, and entertainment. Due to the mobility and ever expanding capabilities, the use of smartphones has experienced an exponential growth rate in recent years. In the first quarter of 2017, Android accounted for 85.0% of the market share of smartphones [1]. Android is an open source, and there exist many third-party Android markets (e.g. Baidu, Opera Mobile Store, or Anzhi). Because of these situations, it has been increasingly targeted by attacker, and 97% of mobile malwares is developed for Android [2]. A recent security report shows that on average, 38,000 new mobile malware samples were captured per day during the third quarter of 2016 [3]. Hence, the detection of Android malwares is imperative.

In order to deal with this issue, several Android malware detection schemes have been proposed [4]–[6]. They are classified into required resources based approaches [4], [5] and Inter-Component Communication (ICC) based approach [6]. Enck et al. [4] propose an approach based on permissions required by Android apps. They leverage the fact that malwares tend to register specific combinations of permissions. However, that scheme is not applicable to repackaged apps. Since the required permissions of repackaged apps are similar to the original ones, repackaged apps can evade that scheme. In order to detect such malwares, Deshotels et al. [5] propose DroidLegacy which focuses on API calls. They leverage the fact that malwares abuse sensitive API call to conduct malicious operations. That scheme captures the communications between apps and Android system based on API calls of apps. However, since that scheme is designed to detect malwares exploiting sensitive API calls, malwares can bypass that scheme by conspiring with another app to conduct malicious operations without using API calls. In order to address such a situation, Xu et al. [6] leverage the fact that there exists the difference in ICC patterns between benign apps and malwares. While ICC is mainly utilized for internal communications within the same apps, malwares tend to communicate with other apps via ICC to conduct malicious operations. Thus, that scheme can detect malwares which invalidate required resources-based schemes by focusing on ICC. Although various Android malware detection scheme have been proposed, we pay attention to [6], because the malwares abusing ICC to conspire with another app is rapidly increasing.

In [6], numerous ICC-related features are extracted from apps to detect malwares by machine learning. In order to mitigate the degradation of detection performance caused by irrelevant and redundant features, that scheme applies a well-known feature selection method, Correlation-based Feature Selection (CFS) [7] to feature before machine learning. CFS is based on the theory that a good feature subset constitutes features highly correlated with the class (benign or malicious) in machine learning, yet has little correlation with mutual features in a feature subset. However, CFS may remove useful features for detection in accordance with the theory. This is because there exist the features strongly correlated with each other in the useful ICC-related features.

Manuscript received August 27, 2018.
Manuscript revised January 26, 2019.
Manuscript publicized March 12, 2019.

[†]The authors are with Dept. of Information and Computer Science, Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: osuge@sasase.ics.keio.ac.jp
b) E-mail: kato@sasase.ics.keio.ac.jp
c) E-mail: haruta@sasase.ics.keio.ac.jp
d) E-mail: sasase@ics.keio.ac.jp

DOI: 10.1587/transinf.2018EDP7301

Thus, there is the possibility that the detection performance is degraded due to excessive removal of features.

In this paper, we propose an effective feature selection scheme for Android ICC-based malware detection using the gap of the appearance ratio. We argue that the features frequently appearing in either benign apps or malwares are useful for malware detection, even if they are strongly correlated with each other. In order to distinguish useful features from unuseful ones on the basis of our argument, we introduce the proportion of the appearance ratio of a feature between benign apps and malwares. Since the proportion can represent whether a feature frequently appears in either benign apps or malwares, this metric is useful for feature selection based on our argument. Unfortunately, the proportion is ineffective when a feature appears only once in all apps. Thus, we also introduce the difference of the appearance ratio of a feature between benign apps and malwares. Since the difference simply represents the gap of the appearance ratio, we can select useful features by using this metric when such a situation occurs. Since these two metrics are the useful ones reflecting our argument, we can select the useful features discarded by the previous scheme. The contributions of this paper are as follows:

1. We propose the feature selection technique which is suitable for Android ICC-related features. Our scheme can select useful features on the basis of our argument that the features frequently appearing in either benign apps or malwares are useful for malware detection.
2. Our evaluation results show that our scheme achieves higher detection accuracy than the previous scheme. Furthermore, after investigating the selected features, we discover the useful ones removed by the previous scheme.

The rest of this paper is constructed as follows: we introduce related works in Sect. 2. We explain the background techniques, the previous scheme and its shortcoming in Sect. 3. The proposed scheme is described in Sect. 4. Simulation results are shown in Sect. 5. We conclude this paper and mention future works in Sect. 6.

2. Related Work

There are many Android malware detection schemes, which are roughly divided into "Required resources based scheme" and "ICC based scheme". The representative schemes are explained in the following Sections.

2.1 Required Resources Based Detection

Enck et al. [4] propose the scheme which detects malwares by focusing on the permissions required by the Android apps. That scheme pays attention to the fact that malwares tend to register specific combinations of permissions. Malwares can be detected by matching required permissions against pre-defined security rules. However, that scheme is not applicable to repackaged apps. Since a repackaged app is created by injecting malicious components into an original benign app, the required permissions are similar to the original ones. Hence, repackaged apps can evade that scheme. In order to detect repackaged apps, Deshotels et al. [5] propose the scheme relying on API calls to detect malwares. That scheme focuses on that malwares abuse sensitive API call to conduct malicious operations. Repackaged malwares are classified in accordance with matching Android API calls against the signatures that identify malwares produced by repackaging. That scheme captures the communications between apps and Android system based on API calls of apps. However, since that scheme is designed to detect malwares exploiting sensitive API calls, malwares can bypass that scheme by conspiring with another app to conduct malicious operations without using API calls. This is why it is necessary to design the schemes which rely on other features.

2.2 ICC Based Detection

Xu et al. [6] propose the scheme which builds malware detection models based on ICC-related features. That scheme is the only scheme that defines the ICC-related features and detects malwares on the basis of them. The main idea of that scheme is that there exists the difference in ICC patterns between benign apps and malwares. While ICC is mainly utilized for internal communications within the same apps, malwares tend to communicate with other apps via ICC to conduct malicious operations. That scheme extracts the ICC-related features that is likely to be abused, and machine learning is performed to distinguish benign apps from malwares. Because Android applications communicate with each other through the ICC mechanism provided by Android, that scheme can detect the malwares which invalidate most existing required resources based detection schemes by leveraging the ICC mechanism instead of required resources. Although various Android malware detection schemes have been proposed, we pay attention to [6], because the malwares abusing ICC is rapidly increasing. We elaborate that scheme in the next Section.

3. Background Techniques and Previous Scheme

3.1 Background Techniques

In the previous scheme, EPICC [8] is employed for extracting ICC-related features. Table 1 shows the ICC categories and the patterns used in the previous scheme. Component is a function defined by the developer and is divided into four types, Activity, Service, Broadcast Receiver, and Content Provider. Activity provides all visible actions. Service can perform long-running operations in the background. Broadcast Receiver receives information from multiple apps. Content Provider manages access to a database. Intent is a mechanism that allows apps to interact within the same app or communicate with other apps by sending it to a certain app's

Component or Android system. The difference between Explicit and Implicit Intent is whether the destination app of the intent is specific or not. Intent Filter is used to match with Implicit Intent in Android system.

3.2 Overview of the Previous Scheme

The main idea of the previous scheme is that there exists the difference in ICC patterns between benign apps and malwares. In general, benign apps mainly use ICC for internal communications within the same app. On the other hand, malwares tend to interact with other apps via the ICC mechanism in order to conduct malicious operations. That scheme extracts numerous ICC-related features from apps to detect malwares by machine learning. The extracted features are divided into three types. The first one is the number of each pattern appearing in an app. The second one is the number of each function appearing in an app. The third one is the number of each destination app of External_Explicit_Intent appearing in an app. Basically, an ICC pattern indicates a function defined by the developer or Android system. Exceptionally, one means a package name of a destination app of External_Explicit_Intent when Explicit Intent is sent to another app.

In order to mitigate the degradation of detection performance caused by irrelevant and redundant features, that scheme applies a well-known feature selection method, CFS [7] to feature before machine learning. CFS is based on the theory that a good feature subset contains features highly correlated with the class (benign or malicious) in machine learning, yet has little correlation with mutual features in a feature subset. CFS expresses correlations as a numerical value and selects a good feature subset in accordance with the theory. Finally, that scheme makes the feature vectors based on the selected features for machine learning. The feature vectors are fed into classifier such as SVM [9] and it detects malwares.

3.3 Shortcoming of the Previous Scheme

In CFS's theory, the features strongly correlated with each other are removed. Thus, there is the possibility that CFS removes not only redundant features but also ones which are useful for distinguishing malwares from benign apps. This is because there exist the features strongly correlated with each other in the useful ICC-related features. For instance, the number of External_Explicit_Intent and the number of the destination app named com.android.browser(external_explicit_intent) (hereinafter, this is called "browser Intent") are both useful for detection. This is because the malwares that are not permitted to browse web tend to send browser Intent to external apps in order to manipulate a browser app. By doing this, the malwares can supplant the Android's standard web browser and obtain personal information. In this case, the number of External_Explicit_Intent is necessarily a non-zero value if browser Intent appears in the app. Therefore, the number of External_Explicit_Intent is strongly correlated in terms of the presence of browser Intent in an app. Accordingly, although the number of browser Intent is useful feature, CFS removes it because of strong correlation between above features, and the detection performance is degraded.

Table 1 ICC categories obtained in the previous scheme

ICC categories	ICC patterns
Component	Activity, Service, Broadcast_Receiver(static), Broadcast_Receiver(dynamic), Content_Provider
Explicit Intent	All_Explicit_Intent, External_Explicit_Intent
Implicit Intent	All_Implicit_Intent, Internal_Implicit_Intent, External_Implicit_Intent(userdefined_action), External_Implicit_Intent(system_action)
Intent Filter	All_Intent_Filter, Intent_Filter(for_activity), Intent_Filter(for_service), Intent_Filter(for_receier_static), Intent_Filter(for_receiver_dynamic)

4. Proposed Scheme

4.1 Idea

We argue that since the useful features should be used for malware detection even if they are strongly correlated with each other, CFS is not appropriate for Android ICC-related features. Thus, in this paper, we propose an effective feature selection scheme for Android ICC-based malware detection using the gap of the appearance ratio. Our scheme focuses on the fact that the features frequently appearing in either benign apps or malwares are useful for malware detection, even if they are strongly correlated with each other. Figure 1 shows useful features and unuseful ones for detection in our scheme. As shown in Fig. 1, we intuitively understand that feature_2 is a useful feature because it frequently appears only in malicious apps. Thus, this feature should be utilized for malware detection. On the other hand, feature_1 is a redundant feature because there is no difference of the appearance ratio between benign apps and malwares. There are a very large number of features such as feature_1 ap-

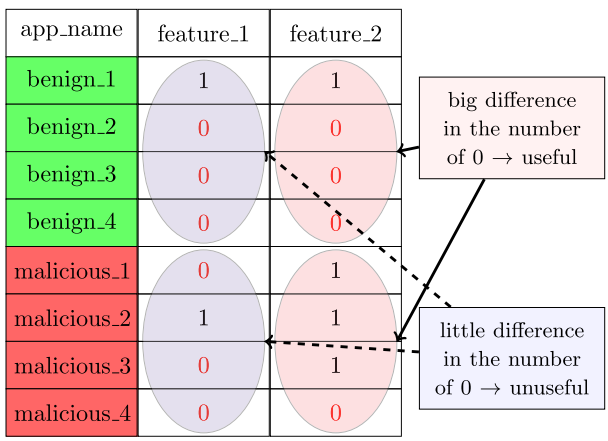


Fig. 1 Useful features and unuseful ones in our scheme

pearing only in a few apps because Android provides a wide variety of functions. Since such features do not introduce useful evidence in the malware detection, they should not be used for training of machine learning. Thus, we exclude them to improve detection accuracy.

In order to distinguish useful features from unuseful ones according to our argument that useful features frequently appear in either benign apps or malwares, we need the metric that can be used to compare the appearance ratio of a feature in benign apps with that in malwares. To select useful features based on our argument, we introduce the proportion of the appearance ratio of a feature between benign apps and malwares. Since the proportion can represent whether a feature frequently appears in either benign apps or malwares, this metric is useful for feature selection based on our argument. In order not to divide a numerical value by 0, the proportion is defined as the minimum value of the appearance ratio of a feature in benign apps and that in malwares by the maximum value of them. Therefore, the proportion indicates a value from 0 to 1. Because of this definition, it is assumed that the smaller the proportion is, the larger the gap of the appearance ratio of a feature between benign apps and malwares is. Thus, a feature showing a small proportion is useful for detecting malwares. However, the proportion does not function as the proper metric when the value of it is 0. For instance, we consider the case where the feature f_A appears only once in all apps. In this case, f_A seems to be unuseful since it is probably a function defined by the developer. However, f_A is regarded as a useful feature by the proportion in spite of unusefulness of it. In order to address such a situation, we also introduce the difference of the appearance ratio of a feature between benign apps and malwares. We define the difference as the absolute value of subtraction of the appearance ratio in benign apps and that in malwares. Since the difference simply represents the gap of the appearance ratio, we can select useful features by using this when the proportion equals 0. These new metrics can be the useful ones reflecting our argument.

The proportion of the appearance ratio of a feature between benign apps and malwares and the difference of that is necessary to be markedly represented in order to select useful features reliably. Therefore, decision content is utilized to realize that. Let $P(f)$ denote a probability that a feature f does not appear in the training dataset, then decision content $E(f)$ is given as follows:

$$E(f) = -log_2 P(f). \quad (1)$$

As shown in Fig. 2, the value of $E(f)$ increases acceleratively as the value of $P(f)$ decreases. We calculate the benign app's decision content $E(f)_{benign}$ and the malicious app's decision content $E(f)_{malicious}$, respectively and compare $E(f)_{benign}$ with $E(f)_{malicious}$ to select useful features. The algorithm of comparison is explained in the next Section. Finally, we utilize only the features selected by our scheme for training of machine learning.

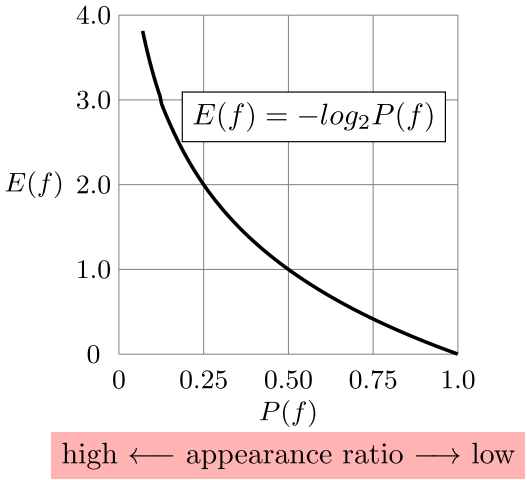


Fig. 2 Probability $P(f)$ versus decision content $E(f)$

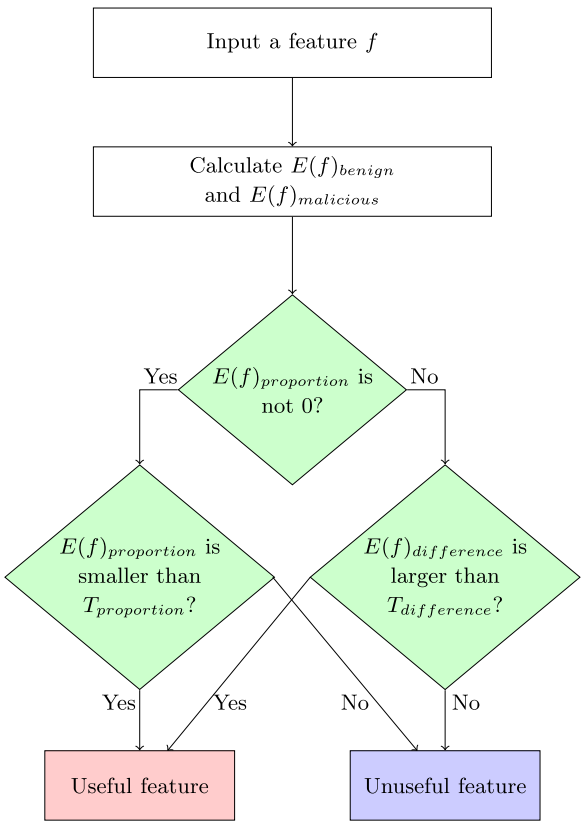


Fig. 3 Flowchart of our scheme

4.2 Algorithm

Figure 3 shows the flowchart of our scheme. First, we calculate $E(f)_{benign}$ and $E(f)_{malicious}$ of a feature f. Here, we define the value set of benign training dataset and that of malicious one in f as B_f and M_f, respectively. Note that $P(f)$ in Eq. (1) indicates a probability that f does not appear in the training dataset in our scheme. Let $B_f^0 \subseteq B_f$ and $M_f^0 \subseteq M_f$

denote the zero-value set in B_f and that in M_f, respectively. $E(f)_{benign}$ and $E(f)_{malicious}$ are calculated as follows:

$$E(f)_{benign} = -log_2 \frac{n(B_f^0)}{n(B_f)}, \quad (2)$$

$$E(f)_{malicious} = -log_2 \frac{n(M_f^0)}{n(M_f)}, \quad (3)$$

where $n(set)$ means the number of elements in set.

As mentioned in Sect. 4.1, our scheme introduces not only the proportion of the appearance ratio of a feature between benign apps and malwares but also the difference of that. The proportion $E(f)_{proportion}$ is determined as follows:

$$E(f)_{proportion} = \frac{\min(E(f)_{benign}, E(f)_{malicious})}{\max(E(f)_{benign}, E(f)_{malicious})}. \quad (4)$$

A feature f showing small $E(f)_{proportion}$ is a useful feature for detecting malwares. In our scheme, a feature f that indicates a smaller $E(f)_{proportion}$ than the threshold $T_{proportion}$ is leveraged for machine learning.

$E(f)_{difference}$ is defined as follows:

$$E(f)_{difference} = |E(f)_{benign} - E(f)_{malicious}|. \quad (5)$$

A feature f indicating large $E(f)_{difference}$ is a useful feature for distinguishing benign apps from malwares. Therefore, we utilize a feature f that shows a larger $E(f)_{difference}$ than the threshold $T_{difference}$ for detection when $E(f)_{proportion}$ equals 0. In our scheme, $T_{proportion}$ and $T_{difference}$ are determined by some experiments in Sect. 5.2.

We perform above procedure for all features. Utilizing both $E(f)_{proportion}$ and $E(f)_{difference}$, we realize feature selection technique which is suitable for detecting malwares using ICC.

5. Evaluation

In order to show the effectiveness of our scheme, we evaluate Accuracy, True Positive Rate (TPR), and False Positive Rate (FPR) calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \quad (6)$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad (7)$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}, \quad (8)$$

where TP, TN, FP, and FN denote the number of True Positive (malwares are regarded as malwares), True Negative (benign apps are regarded as benign ones), False Positive (benign apps are regarded as malwares), and False Negative (malwares are regarded as benign apps), respectively.

5.1 Simulation Parameters

Table 2 shows our simulation parameters. As the benign android apps dataset, we use Androzoo dataset [10]. Androzoo collects more than five million apps from several sources, including the official Google play app market. Each of them has been analysed by VirulTotal [13], which is an antivirus service with over 60 antivirus scanners. We extract the 3,152 apps for which no antivirus scanners raise any alarm from Androzoo. As the malwares dataset, we use Drebin dataset [11]. This malware set is one of the largest datasets of Android malwares being publicly available today. We randomly pick the 1,448 malwares from Drebin dataset.

Table 2 Simulation parameter

Name	Data
Benign apps	Androzoo [10]
Malicious apps	Drebin [11]
The number of benign apps	3,152
The number of malicous apps	1,448
Classifier	SVM [9]
Validation	ten-fold cross validation [12]
Simulation tool	Python

We compare the detection performance of our scheme with that of the previous scheme [6]. The previous scheme leverages a widely used two-class classification method, Support Vector Machine (SVM) [9]. SVM is suitable for processing multidimensional data like the feature vectors and capable of producing a model efficiently. In order to conduct fair comparison between our scheme and the previous scheme, our scheme also utilizes SVM. We conduct a series of experiments using ten-fold cross validation [12] to measure the performance of our scheme and the previous scheme. This can confirm the validity of the analysis.

5.2 Decision of the Threshold

In order to decide thresholds, we record the detection accuracy while changing $T_{proportion}$ and $T_{difference}$ in an applicable range at an interval. The best thresholds depend on the number of the datasets since our scheme has to calculate the appearance ratio of a feature. Therefore, we heuristically determine applicable ranges of thresholds by conducting initial experiments in which thresholds are roughly changed. In the initial experiments with the dataset, we find out that the best $T_{proportion}$ and $T_{difference}$ are roughly in the range of 0.20 to 0.50 and 0.0010 to 0.0020 respectively. In order to fully include the ranges, we determine the applicable range of $T_{proportion}$ and $T_{difference}$ as 0.01 to 0.70 and 0.0001 to 0.0050 respectively. As mentioned above, since the best thresholds depend on the dataset, detection accuracy is not necessarily improved as we make intervals small. Thus, we decide the interval of $T_{proportion}$ and $T_{difference}$ as 0.01 and 0.0001 for the inspection of thresholds not to take too long. According to the range and the interval, we perform machine learning 70×50 times. Figure 4 shows the inspection result of $T_{proportion}$ and $T_{difference}$. As we can see from this figure, a high accuracy is obtained over a wide range.

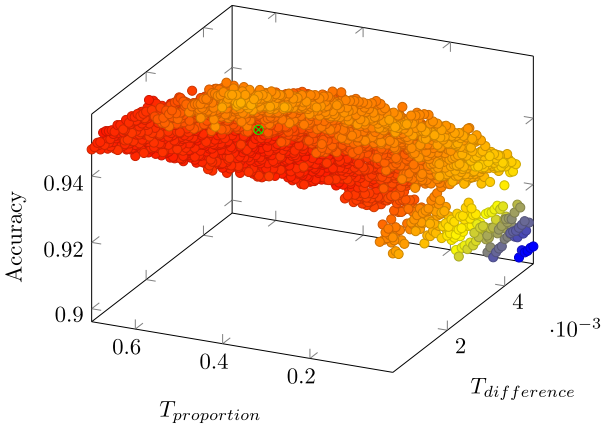


Fig. 4 Inspection result of $T_{proportion}$ and $T_{difference}$

```
Manifest file for com.loadfon.filer version 1
  Activities:
    com.loadfon.filer.MySQLActivity
    com.loadfon.filer.StartActivity
      Intent filter:
        Actions: [android.intent.action.MAIN]
        Categories: [android.intent.category.LAUNCHER]

  Activity Aliases:
  Services:
  Receivers:
  Providers:


The following ICC values were found:
  - com/loadfon/filer/StartActivity/onNextButton(Landroid/view/View;)
Intent value: 1 possible value(s):
Package: com/loadfon/filer, Class: com/loadfon/filer/MySQLActivity,
```

Fig. 5 Example of an Android malware barely using ICC

Table 3 Detection performance

Scheme	TPR (%)	FPR (%)	Accuracy (%)
Our scheme	86.9	1.2	95.1
ICCDetector	81.9	4.0	91.6

Finally, we determine $T_{proportion}$ and $T_{difference}$ as 0.39 and 0.0012 that show the best accuracy in the experiment.

5.3 Detection Performance

Table 3 shows detection performance of our scheme and the previous scheme [6]. As we can see from this table, the accuracy of the previous scheme is up to 91.6%, while our scheme achieves the accuracy of 95.1%, roughly 4% higher than the previous scheme, with a higher TPR and a lower FPR. This is because our scheme can select the useful features discarded by the previous scheme. For instance, android.provider.Telephony.SIM_FULL(for_receiver_static) (hereinafter, this is called "telephony Intent Filter") is utilized for machine learning only in our scheme. Malwares tend to register Intent_Filter(for_receiver_static) to monitor system events such as SMS-related information and downloading states. In particular, some malwares manipulate telephony Intent Filter in order to observe SMS messages. This is why there is a danger that such malwares steal SMS messages and upload them to remote server. Hence, telephony Intent Filter should be used as a feature for malware detection. However, the previous scheme discards telephony Intent Filter due to strong correlation between it and the number of Intent_Filter(for_receiver_static). On the other hand, our scheme can select it as a useful feature by considering the gap of the appearance ratio of a feature between benign apps and malwares. As a result, our scheme can improve the detection accuracy.

5.4 False Negative and False Positive Analysis

Our scheme misses 189 malwares, and ICCDetector regards 262 malwares as benign apps. This is because there exist the Android malwares barely using ICC mechanism. Figure 5 shows an example of an EPICC output from such malwares. Both schemes can extract few ICC-related features from these malwares. In general, malwares tend to register more ICC than benign apps in order to conduct malicious operations via the ICC mechanism. However, since these malwares utilize few ICC-related features, the schemes based on ICC judge such malwares as benign apps. Instead of ICC, these malwares simply abuse required resources such as permissions and sensitive API calls. Thus, a hybrid scheme can address this shortcoming.

Our scheme labels 37 benign apps as malwares (i.e., false positives). In order to analyze these benign apps, we resent them to VirusTotal. This is because the detection results of VirusTotal can be changed since it is updated and corrected over time. After resending the 37 false positives, we discover that 9 apps, which had been extracted from Androzoo in February 2018, received alarms from at least one antivirus scanners in January 2019. Thus, our scheme correctly labeled these 9 apps as malwares. In fact, these malwares can be easily detected according to their ICC-related features. For instance, there exist apps registering android.intent.action.BOOT_COMPLETED(for_ receiver_static) (hereinafter, this is called "boot Intent Filter"). Malwares tend to register boot Intent Filter in order to effectively launch their malicious operations when the Android system completes its booting process in which much information is exchanged. Besides those 9 apps, we consider the other 28 false positives. After manually analysing them, we find out that they tend to register Intent_Filters (for_receiver_static). For example, there exist apps registering android. provider.Telephony.SMS_RECEIVED (for_receiver_static) (hereinafter, this is called SMS Intent Filter). SMS Intent Filter can be used to immediately conduct user authentication with SMS messages. However, registering it allows the app to monitor not only required SMS messages but also irrelevant ones. Malwares tend to register SMS Intent Filter in order to manipulate SMS messages. Thus, these apps are potentially malicious since they can conduct malicious operations. As a result, our scheme can discover the 28 potentially malicious apps missed in Virus-

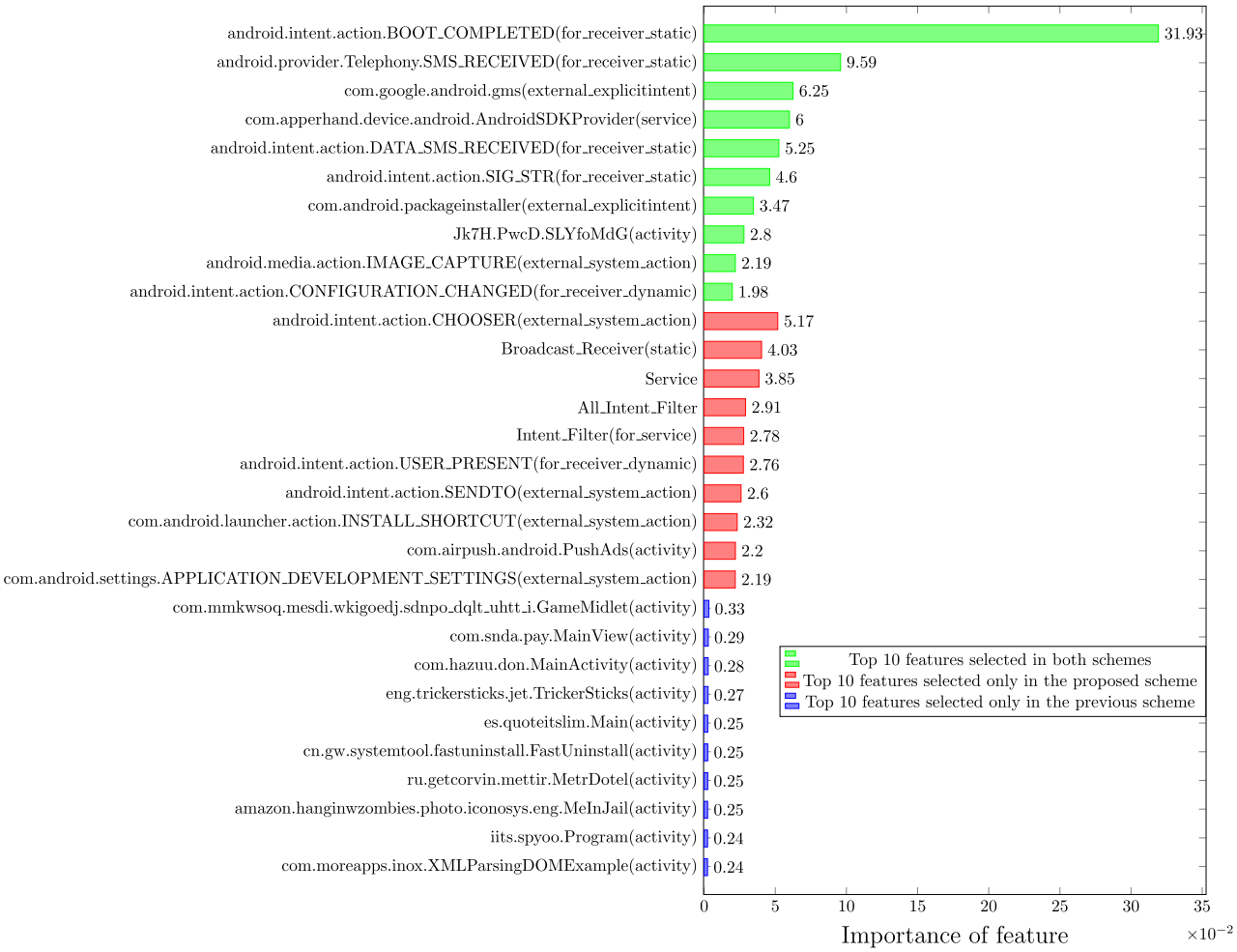


Fig. 6 Top 10 features selected in both schemes and only in each one

Total by leveraging ICC-related features.

5.5 Selected Features Analysis

38,296 ICC-related features are extracted from 3,152 benign apps and 1,448 malwares, and the dimension of feature vectors equals (4600, 38296). From 38,296 features, 2,609 features are selected by the proposed scheme, and the reduced dimension of feature vectors equals (4600, 2609). From 38,296 features, 1,995 features are selected by the previous scheme, and the reduced dimension of feature vectors equals (4600, 1995). 2,435 features are selected only in the proposed scheme, and 1,821 features are only in the previous one. Only 174 features are selected by both scheme in common. This is because how to select useful features in our scheme is basically different from that in the previous one. The previous scheme is based on the theory that a good feature subset contains features highly correlated with the class (benign or malicious) in machine learning, yet has little correlation with mutual features in a feature subset. On the other hand, our scheme selects features frequently appearing in either benign apps or malwares as useful ones for malware detection, even if they are strongly correlated with each other. Furthermore, we investigate the top 10 important features for detection in both schemes and only in each scheme to analyze the representative ones. In order to find out the top 10 important features, we utilize RandomForest [14] which can express importances of features as numerical values. We make use of RandomForest to the features selected in both schemes, only in the proposed scheme and only in the previous one. Figure 6 shows the top 10 important features for detection in both schemes and only in each one. As we can see from the figure, the features selected in both scheme are the most important to detect malwares. For instance, android.intent.action.SIG_STR(for_receiver_static) (hereinafter, this is called signal Intent) is useful to distinguish malwares from benign apps. This is because malwares tend to register signal Intent to receive broadcast information related to changes of signal strength and intercept system events. These features result in that both schemes achieve the accuracy of more than 90%. In the following, we consider the difference of the selected features and their characteristics.

After manually analyzing the top 10 features selected only in the previous scheme, each of them is a feature appearing only once in all apps. These features are related to

the class and irrelevant to the other features. In other words, such a feature is correlated with the class, yet has no correlation with mutual features. This is why CFS selected these features. However, they are not useful to detect malwares because each of them is the function defined by the developer and does not introduce useful evidence in the malware detection. On the other hand, our scheme discards these features by utilizing the difference of the appearance ratio of a feature between benign apps and malwares. From the above, we assume that our scheme can effectively discard unuseful features selected in the previous one. Moreover, as we can see from Fig. 6, the maximum importance of the top 10 features selected only in CFS is less than 20% than the minimum importance only in our scheme. This means that the features selected only in CFS are much less useful to detect malwares than our scheme. Hence, although there exist 1,821 features selected only in CFS, our scheme does not discard any useful features selected by CFS.

The top 10 features selected only in our scheme are roughly categorized into two types. In Fig. 6, Broadcast_Receiver(static), Service, All_Intent_Filter, and Intent_Filter(for_ service) are classified into type A, and the others are into type B. As an example of type A, the total number of Broadcast_Receiver(static) (hereinafter, this is simply called "Receiver") is useful to distinguish benign apps from malwares. This is because malwares tend to register more Receivers than benign apps do. A lot of Receivers enable malwares to monitor system events such as network connectivity changes and battery changes. Nevertheless, CFS removes Receiver due to strong correlation with many functions related to Receiver. The reason is that the presence of the total number of Receiver in a feature subset raises correlation with mutual features in the feature subset. Furthermore, as an example of type B, com.android.settings.APPLICATION_ DEVELOPMENT_SETTINGS(external_system_action) (hereinafter, this is called setting Intent) is useful to detect malwares. This is because malwares tend to register setting Intent in order to change the setting and conduct malicious operations. However, CFS discards setting Intent because of strong correlation with the total number of External_Implicit_Intent(system_action). There also exists the gap of the appearance ratio between benign apps and malwares in the others of type A and the others of type B. The above results demonstrate that our scheme can select 2,435 useful features such as type A and type B discarded in the previous one.

5.6 Scope of Our Scheme

Our scheme can be applied to feature vectors which have not only features frequently appearing in either benign apps or malwares but also ones equally appearing in both classes of apps. This is because our scheme can not only reduce redundant features but also save useful features by deciding appropriate thresholds. For instance, features appearing only once in all apps are certainly removed by determining proper $T_{difference}$, and features frequently appearing in either benign apps or malwares are selected by deciding appropriate $T_{proportion}$ and $T_{difference}$. As an example, we apply our scheme to features related to API calls. We extract API calls from the dataset by using androguard [15], which can extract a package name, permissions, API calls and so on from Android apk files. From 4593 apps except for 7 apps which cause errors in the running process of androguard, 12,494 API calls are extracted. After applying our scheme to features related to API calls, 3,538 features are selected. Moreover, our scheme reduces 16 false positives and 4 false negatives compared to the case where no feature selection schemes are utilized. This is because our scheme removes features appearing only once in all apps and selects useful features frequently appearing in either benign apps or malwares. The result demonstrates that our scheme can be applied to feature vectors such as mentioned above. Note that this paper focuses on ICC that is recently gathering attention in the field of Android malware detection. Thus, we apply our scheme to ICC-related features to deal with the shortcoming of the previous one.

5.7 Runtime to Select Useful Features

We ran our experiments on a machine with 10×3.3GHz Intel Core i9 7900X and 64GB of RAM, and measured the runtime of both schemes. In each of ten rounds in our experiments, SVM is trained with 4,140 apps (i.e., 90% of the dataset), and tested with the rest 460 apps. Therefore, in order to calculate the runtime to select useful features per an app, we strike the average of the runtime in each round divided by 4,140. Consequently, the runtime per an app in the proposed scheme is 1.26×10^{-3} seconds, and that in CFS is 11.7 seconds. According to the results, the feature selection process of the proposed scheme is about 9,000 times faster than CFS. This is because CFS has to calculate not only the correlation between features and the class but also mutual features. On the other hand, in the proposed scheme, only decision contents are computed by simple probability calculations to obtain the difference and the proportion. In other words, the proposed scheme has only to calculate the correlation between features and the class. From the above, the proposed scheme can select useful features faster than CFS.

6. Conclusion

We have proposed an effective feature selection scheme for Android ICC-based malware detection using the gap of the appearance ratio. We focus on the fact that the features which frequently appearing in either benign apps and malwares are useful for malware detection, even if they are strongly correlated with each other. By comparing the appearance ratio of a feature in benign apps with that in malwares, we can utilize the useful features discarded by the previous scheme. By the computer simulation using real dataset, we show our scheme achieves the accuracy of 95.1%, roughly 4% higher than the previous scheme. As

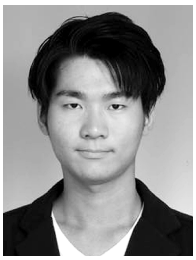
future works, we will reconsider the decision of the thresholds. In the current state, although it is necessary to decide $E(f)_{proportion}$ and $E(f)_{difference}$ by performing some experiments, we consider that the automatic determination method of the thresholds is needed to reduce calculation cost. Furthermore, we plan more detailed evaluation about the validity for using decision content.

Acknowledgments

This work is partly supported by the Grant in Aid for Scientific Research (No.17K06440) from Japan Society for Promotion of Science (JSPS).

References

[1] "IDC: Smartphone OS Market Share," https://www.idc.com/promo/smartphone-market-share/os.
[2] G. Kelly, "Report: 97% Of Mobile Malware Is On Android," https://goo.gl/zHFVVn.
[3] Qihoo, "Report of Smartphone Security in China," https://goo.gl/V9Vh1u.
[4] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," Proceedings of the 16th ACM conference on Computer and communications security - CCS '09, pp.235–245, 2009.
[5] L. Deshotels, V. Notani, and A. Lakhotia, "DroidLegacy: Automated Familial Classification of Android Malware," Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014, pp.1–12, 2014.
[6] K. Xu, Y. Li, and R.H. Deng, "ICCDetector: ICC-Based Malware Detection on Android," IEEE Trans. Inf. Forensics Security, vol.11, no.6, pp.1252–1264, 2016.
[7] M. Hall, "Correlation-based Feature Selection for Machine Learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. Waikato, Hamilton, New Zealand, 1999.
[8] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y.L. Traon, "Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis," USENIX Security Symposium, pp.543–558, 2013.
[9] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," Data Mining and Knowledge Discovery, vol.2, no.2, pp.121–167, 1998.
[10] L. Li, J. Gao, M. Hurier, P. Kong, T.F. Bissyandé, A. Bartel, J. Klein, and Y.L. Traon, "AndroZoo++: Collecting millions of android apps and their metadata for the research community," arXiv preprint arXiv:1709.05281, 2017.
[11] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," Proceedings 2014 Network and Distributed System Security Symposium, pp.1–15, Feb. 2014.
[12] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," Proc. of IJCAI'95, pp.1137–1145, 1995.
[13] "VirusTotal," https://www.virustotal.com.
[14] L. Breiman, "Random Forests," University of California, Berkeley, Journal of Machine Learning, vol.45, no.1, pp.5–32, Oct. 2001.
[15] A. Desnos, "androguard," http://code.google.com/p/androguard/.



Kyohei Osuge was born in Kanagawa, Japan in 1995. He received his B.S degree from Keio University in 2018. He is a Master student at Keio University. His reseach interest is security & privacy for IoT. He is a member of IEICE.

Hiroya Kato was born in Gunma, Japan in 1994. He received his B.S degree from Keio University in 2017. He is a Master student at Keio University. His reseach interest is security & privacy for IoT. He is a member of IEICE.

Shuichiro Haruta was born in Saitama, Japan in 1992. He received his M.S. degree from Keio University in 2017. He is a Ph.D. student at Keio University. His research interest is security & privacy for IoT. He is a research associate at Keio University. He is a member of IEICE and IEEE.

Iwao Sasase was born in Osaka, Japan in 1956. He received the B.E., M.E., and D. Eng. degrees in Electrical Engineering from Keio University, Yokohama, Japan, in 1979, 1981 and 1984, respectively. From 1984 to 1986, he was a Post Doctoral Fellow and Lecturer of Electrical Engineering in University of Ottawa, ON, Canada. He is currently a Professor of Information and Computer Science in Keio University, Yokohama, Japan. His research interests include modulation and coding, broadband mobile and wireless communications, optical communications, communication networks and information theory.