

Andro-Simnet: Android Malware Family Classification using Social Network Analysis

Hye Min Kim
Korea University

Seoul, Korea
hm941224@korea.ac.kr

Hyun Min Song
Korea University
Seoul, Korea
signos@korea.ac.kr

Jae Woo Seo
Samsung Electronics Software
R&D Center
Seoul, Korea
jaewoo13.seo@samsung.com

Huy Kang Kim
Korea University
Seoul, Korea
cenda@korea.ac.kr

Abstract— While the rapid adaptation of mobile devices changes our daily life more conveniently, the threat derived from malware is also increased. There are lots of research to detect malware to protect mobile devices, but most of them adopt only signature-based malware detection method that can be easily bypassed by polymorphic and metamorphic malware. To detect malware and its variants, it is essential to adopt behavior-based detection for efficient malware classification. This paper presents a system that classifies malware by using common behavioral characteristics along with malware families. We measure the similarity between malware families with carefully chosen features commonly appeared in the same family. With the proposed similarity measure, we can classify malware by malware's attack behavior pattern and tactical characteristics. Also, we apply community detection algorithm to increase the modularity within each malware family network aggregation. To maintain high classification accuracy, we propose a process to derive the optimal weights of the selected features in the proposed similarity measure. During this process, we find out which features are significant for representing the similarity between malware samples. Finally, we provide an intuitive graph visualization of malware samples which is helpful to understand the distribution and likeness of the malware networks. In the experiment, the proposed system achieved 97% accuracy for malware classification and 95% accuracy for prediction by K-fold cross-validation using the real malware dataset.

Keywords—malware similarity, machine learning, malware classification, social network analysis

I. INTRODUCTION

Recently, the number of malware is increasing day by day, and it is generating many victims. Since a growth rate and a spreading rate of malware are growing, a manual malware analysis method has become hard to cope with it. Therefore, a process of analysis, classification, and detection of malware need to be automated [1]. Furthermore, as an evasion and obfuscation technology advances, malware becomes more sophisticated and complex and then easily bypasses signature-based anti-virus programs. A typical example of passing through them is polymorphic malware. It autonomously modifies the original malware and generates variants to avoid signature-based detection measure. To solve this problem, we need to use behavioral characteristics of malware to detect malware which has the same attack patterns. Behavior-based detection can easily reveal the given malware's variants because the malware samples in the same family are born to achieve the same attack

goal and tactics. It means malware in the same family is likely to have similar function call sequences. There have been researches that attempt to detect malware based on the similarity, and they use only one or few simple features such as permission [2] and call graph [3]. Because it is difficult to find out an association between features, it is not well fit to modern malware clustering. Especially, permission information is a no longer useful feature because even benign apps require lots of permissions. Even worse, if detection systems only employ one or small type of features, then it yields biased results because it observes only one aspect of malware. To inspect whole aspects of malware, a hybrid analysis system that combines various features from the signature-based analysis and behavior analysis system is required.

We measure and compare the similarity with the various features extracted from the static and dynamic analysis. All extracted data are vectorized to quantify the malware's behavioral similarity as a numeric value for the comparison. We design a vector that can well describe the behavioral pattern of malware. Among many features, we use API call sequences which can represent malware's behavioral patterns. Then, we normalize string of API call sequences because the length of the sequence could vary by malware. We create a new feature that is the value quantified as the similar relation between malware samples, using whole raw data without missing information. We handle all of the similarity values between all of the selected features for a machine learning process. The final similarity between malware samples is calculated as the weighted sum of these proposed feature similarities. A weighted sum of all feature's similarity can express overall similarity relation between malware samples easily, but still needs adjustment of the weight values.

In general, the weight values of multiple similarity factors are initially given by human experts. In many research, they are usually set with the evenly divided value or heuristically given value whether the features of vectors are orthogonal or not. These values need to be updated when new malware sample is incoming to classification systems. However, in many research, the process for gaining optimal weight value is omitted. To maintain the classification system's performance high, we apply community detection algorithm of social network analysis method to derive the optimal weights.

Classifying malware into a specific family implies that each malware sample in the same family has similar characteristics between the classified samples. Since we perform clustering in

a direction increasing the correlation between similar malware samples, community detection algorithm is an appropriate method. This algorithm makes the modularity of each network aggregation the highest value.

In this paper, we propose Andro-Simnet that classifies Android malware family with malware similarity network graph. Also, we can conclude which features are important for classifying malware with the optimal weights. In this work, we use malware samples on Android platform provided by *Korea Internet and Security Agency (KISA)* and the dataset is available at the website [4].

The key contributions of this paper are as follows:

1) We use the hybrid system that uses both of the results from the static analysis and dynamic analysis. Thus, we can use signature-based information and behavior-based information to classify malware, while the most other papers use only one or two features with limitations. It will be helpful to detect and response the variants of malware.

2) In addition to the extracted features from the static and dynamic analysis, we used the similarity value derived from the social network graph features. To represent relative weight value between the selected features, we use the community detection algorithm. In our approach, we can cluster malware more precisely.

3) We drive the optimal weight values by the proposed machine learning algorithm, not by a heuristic decision as other research. Thus, our malware classification system can continuously update the weight value between selected features optimally and automatically. Also, without human expert knowledge, it makes easy for analysts to choose optimal weight values for features during the classification process.

4) Finally, we test our system with the real-world malware samples and predict one's family in using K-fold cross-validation. Through these results, we validate that Andro-Simnet can easily classify and make groups for malware samples. It can help to find the newly discovered malware's attack goal by checking with the most similar known malware family's purpose. Finally, using Andro-Simnet, it is possible to react the variants of malware quickly and accurately.

II. RELATED WORK

We use Android malware samples to find the similarity of malware and classify it into families. We investigate the method of measuring similar-relation of Android malware and proceed literature study about the system to classify or detect malware. With similar studies, we can consider what features are used to measure similarity and how to derive the final similarity.

Jang *et al.* categorized Android malware by measuring three similarities. First, they selected the specific API that is frequently used by known malware and checked if there is the sequence of the APIs that is related to the specific API. And they measured the similarity using *Needleman-Wunsch* algorithm for API sequence converted into ASCII code format. Next, the similarity of the used commands was calculated by applying *Jaccard* distance algorithm. Finally, the similarity of the

required permission was calculated by applying the *Levenshtein* distance algorithm by reference of whether important permission is requested. Then, the malware was classified according to the largest similarity among three similarities. The blacklist is updated whenever a new malware sample is received [5]. The best results were obtained in malware detection system using both static analysis features and dynamic analysis features. As this paper is our key paper, we refer to feature of malware and the similarity measure from it. Therefore, we can use hybrid features to classify malware.

Ding *et al.* explained that Android API features and required permission are not powerful enough to counter malware's obfuscation techniques because they cannot represent dependencies between two features. In this study, they extracted the structural feature of the Android application from the function call graph. They also proposed the classification system using that feature. The classification system of this study used required permissions, function calls in the main classes, and function call structures as features. The classification system developed by Ding shows a higher accuracy than the classification system using only required permission and Android API calls [6]. It can be seen that the structural features are used to explain the similarity of malware, even when dynamic analysis results are excluded. They decided malicious API feature heuristically, that means the expert is needed to use their system. Compared with our study, our system derives optimal weights of each similarity feature automatically so that the knowledge of the expert is not necessary.

Zhong *et al.* proposed a malware classification method considering the metamorphism of malware programs to find the most similar family. The author used three numeric features APIs, referenced strings, and basic blocks and integrated each of similarities as the arithmetic average [7]. This methodology is similar to our system in that it can integrate various features, but it can be deteriorated because of non-weighted similarity. Conversely, we use feature-based-weighted similarity derived automatically from Andro-Simnet.

DroidScribe used runtime behavior derived from system calls observed during dynamic analysis to classify Android malware into families. It used hybrid prediction methodology: CP augmenting SVM, which achieved better classification accuracy. It showed that dynamic analysis is resilient to obfuscation and use of native code and can be a useful feature for Android malware family classification [8]. Therefore, we can get a sight to use dynamic features to classify malware samples.

Pehlivan *et al.* used classification algorithms in permission-based Android malware detection. They made features with permission characteristics, version code, and version name. The feature set of an application can be specified as a feature vector which includes 182 features. They used four feature selection algorithms and five classification algorithms for evaluating their classification. As a result, it gave a better performance to use 25 features and increasing the number of features to 50 did not improve the performance. They also found that the tree-based algorithm has better performance than *Bayesian* algorithm [9]. They used more than 25 features to classify malware, but we use only 4 features for the same purpose. Fewer features can make

time-efficiency higher, and features that we use is 6 times less than theirs.

Faruki *et al.* measured the statistical similarity of code blocks of zero-day samples using a syntactic foot-printing mechanism to detect Android malware. They had proposed a similarity to use digest hashing mechanism for detecting code obfuscated malware. In the train set, the malware was detected with the accuracy of 98% to 99%, however, in the test set, the accuracy of detection was 72%, and the code obfuscated malware was 63% training [10]. Thus, there is a limit to detect malware with only static analysis result, and a model adjusted to the training set generated the biased result. Their system is hard to classify new malware samples because of the biased features. We conduct K-fold classification to examine how well it can classify new malware samples. After this experiment, we can get higher accuracy.

III. METHODOLOGY

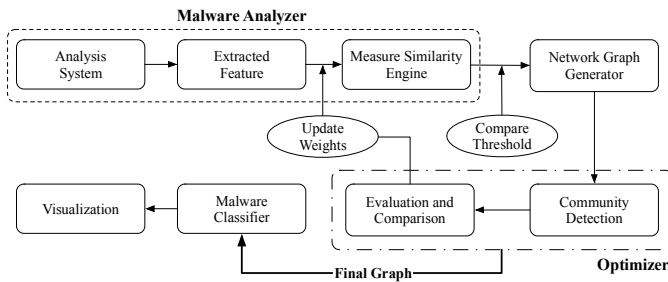


Fig. 1. Overall process of Andro-Simnet

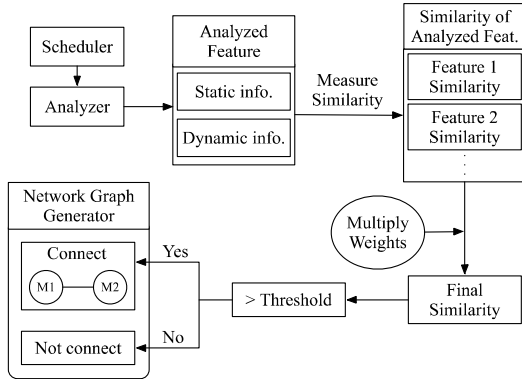


Fig. 2. The process of Analyzer and Network Graph Generator

The process overview of Andro-Simnet is described in Fig. 1. Analyzer extracts static features and dynamic features of malware. The similarity of each feature is measured by different methods depending on the characteristics of the features. Each feature similarity is multiplied by a weight and summed up to calculate the final similarity. Network Graph Generator makes node-edge relations for similar malware samples with this. The node in the network graph represents each malware sample. Nodes that the final similarity between them is higher than given threshold is connected with the weighted edge. It means that connected malware nodes are similar as much as the value of the final similarity. After the network graph is built, community detection algorithm clusters malware samples into the families.

The clusters are stored separately for evaluation during clustering. The process of Optimizer is repeatedly performed to adjust weights until the optimal weights are derived. When the weight is no longer adjusted, the repetition is stopped. The final graph is built based on the optimal weights. Through Malware Classifier, malware nodes are classified into their families. After these processes are ended, we can easily understand classification result with force-directed network graph colored as clusters. The details of each step are described in the following sections.

A. Feature Selection and Measure the Similarity of Malware

We implement an automated hybrid-analysis system that is based on *Cuckoo* system to extract static features and dynamic features from malware [11]. Scheduler manages malware analysis tasks by creating a task queue and assigning the task to the idle virtual machine. Analyzer performs static analysis and dynamic analysis about 10 minutes. Then, we can get API call sequence, network information, Android method and so on. We measure the similarity of each feature that is selected among the analyzed results. The process as mentioned above is shown in Fig. 2.

1) *Feature selection*: There are various features extracted from the analysis system (e.g., network packet, strings that used in malware, a method call). When we select four features, we consider that both malware behaviors and the signatures of the attacker are useful to represent the similar-relation of malware. The chosen features and the details of the reason why we select that feature are described below:

a) *API call sequence*: As we mentioned in introduction, API call sequence is a closely related to the attack goal of malware. API call sequence cannot be extracted by code analysis and package analysis. We can get it only by executing malware in the operating system. Through dynamic analysis, it is possible to grasp the behavior of malware in order of time to achieve its attack goal. API call sequence is the most important feature because it can figure out the malware's functional information. We can infer the malware's function to achieve its attack goal by using this information because malware behavior is strongly related to the attack goal. Therefore, if the similarity of API call sequence is obtained, malware samples with similar functions can be classified with higher accuracy.

b) *Permission*: Because Android applications operate in a single process sandbox, they need to declare the necessary permissions that are not provided by the default sandbox. The permission is declared according to the attack goal of the malware (e.g., when the attacker's goal is to get user information from the smartphone, the permission that can access data in the system is declared). It causes malware to gain system-level privileges or access authority. So, the higher the number of the permissions declared, the more dangerous malware is. Therefore, the permission can be the feature to classify similar families according to the purposeful attack functions.

c) *File name*: There are image files, XML files, source code files and any others in the Android package. Because there are specific rules when naming the file according to developer's

propensity, the name of the file includes the signature information of the developer. The same signature information may remain in the application package while the developer is creating other applications. Therefore, the name of the file can be used for comparing the package of malware and obtaining the similarity of it.

d) Activity name: The name of activity is an application component, and the interface window is given for each activity. Usually, several activities work organically and operate mainly on main activity. The name of the activity, such as the name of the file, is decided by the developer. Since the activity also includes the package name of the application, the developer's signature information in the activity name is more inclined than that in the file name. Also, as the activity name is referred from outwards, it cannot be changed when polymorphic malware transforms its code automatically. Therefore, to measure the similarity of malware samples, activity names are selected as the signature-based feature.

2) *Measure the similarity of malware:* The similarities of selected features are measured by different methods depending on the characteristics of each feature. The methods to measure two types of data are described below:

a) Sequence: An algorithm used to measure sequence similarity should satisfy following conditions.

- The length of the sequence should not significantly affect the similarity.
- It should be possible to have better performance in measuring the similarity between long sequences.
- The similarity should not vary significantly due to slight differences in content.

Considering these requirements, we adopt *Locality Sensitive Hashing* (LSH) algorithm. LSH algorithm is designed to maximize the collision probability for similar inputs in the process of making the input of different lengths the output which has equal length. The conception of LSH algorithm is different from cryptographic hashes, that has an entirely opposite purpose. We select *Nilsimsa* hash algorithm, which is one of LSH algorithm, to measure the similarity of API call sequence [12]. At first, API call sequences of malware are encoded to digests of the same length. Then, the similarity is measured by comparing those digests of 256 bits and normalized at the value between -1 and 1.

b) Set of strings: The remaining three features consist of a set of strings. *Jaccard* similarity is widely used to compare the similarity and diversity of given sets. The basic concept of *Jaccard* similarity is defined by dividing the size of intersection by the size of the union of two sets A and B , and the equation is as follows:

$$\text{Jaccard Similarity } (A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

Jaccard similarity is effective regarding performance and has a value between 0 and 1. The value of 0 means that the given

two sets have no common element and the value of 1 means that the two sets are equivalent.

3) *Calculate the final similarity:* The final similarity of malware is calculated as the weighted sum of feature similarities, and the equation is as follows:

$$FS = \sum_{i=1}^n w_i * s_i \quad (2)$$

s_i = The similarity of each selected feature
 w_i = The weight of each similarity
 FS = The final similarity

if $FS > threshold$, then connect edge between nodes

For finding the optimal weights, the next section explains a process of machine learning method.

B. Clustering Malware by Social Network Analysis

1) *Social network analysis:* In this paper, the feature used for clustering represents a value indicating the similarity between malware samples, not the characteristics of specific malware one by one. So, it is difficult to classify malware by applying existing machine learning algorithms to clustering. We focus on how similar malware samples included in the same family are. Considering this condition, it is necessary to use a social network analysis method to conjugate the connection relationship of malware nodes. We obtain the similarity of malware and multiply weights to the similarities for obtaining the final similarity of malware. When the final similarity exceeds the threshold, the edges are connected to create a network graph. With given network graph, the social network analysis method performs the malware clustering. We use specially *Louvain* community detection algorithm in this paper that detects communities by iteratively optimizing local communities until global modularity no longer increases. Because each clustered community is assigned to the class of the largest number of nodes in the community, we need to consider a large number of malware samples for scalability. *Louvain* community detection algorithm described in Fig. 3 outperforms other algorithms computationally intensive in large networks [13].

In Community Allocation phase, each node is assigned different communities according to the connection of nodes. In Change Community phase, the community of node i into the community is changed to which community node j belongs. The modularity of the graphs that are before and after allocation is compared. If the modularity of current graph is more than that of the previous graph, the current graph is sustained. In contrast, the graph is returned to the former. These whole processes of optimization are repeated over and over until getting the highest degree of modularity. The community relationship of the final model would be defined as these processes no longer increase the modularity. A high degree of modularity means that the association of nodes in the community is strong. When the modularity of each community is the highest, it creates a final network graph classified by each community.

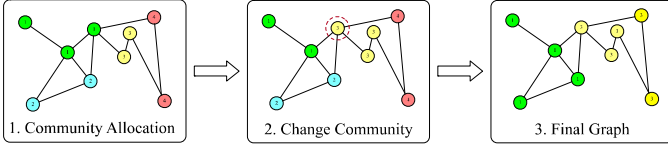


Fig. 3. Louvain community detection algorithm

2) *Clustering malware with the optimal weights*: To derive the optimal weights, Optimizer performs the following processes. When the community detection algorithm clusters a given network graph, the *Error* of clustering result is calculated according to the ground truth. When the error of current clustering $Error_{current}$ is smaller than the error of the previous clustering $Error_{previous}$, the weights $W_{current}$ at the current iteration is stored as the optimal weights $W_{optimal}$. For the next step of optimization, we update weights as much as learning rate.

Finally, we define the optimal weights after finishing the final iteration and then generate network graph and use it for deriving the final accuracy of the clustering. Fig. 4 shows the pseudocode of algorithm for clustering malware and it is repeated as iteration I .

Algorithm 1 Clustering algorithm

```

1: initialization                                ▷ set threshold( $T$ ), iteration( $I$ )
2: weights  $W \leftarrow (w_1, w_2, \dots, w_n)$ 
3: Graph  $G, FG \leftarrow (V, E)$ 
4: Vertices  $V \leftarrow$  all malware samples
5: while count  $< I$ :
6:    $E \leftarrow \{\}$ 
7:   while  $(i, j)$  in all node pairs:             ▷ generate network graph( $G$ )
8:      $S(i, j) \leftarrow (s_1, s_2, \dots, s_n)$        ▷  $S(i, j)$  = the similarity of node  $i$  and  $j$ 
9:      $FS \leftarrow S(i, j) \cdot W$ 
10:    if  $FS > T$ : then add  $(i, j)$  to  $E$ 
11:  end while
12:   $C \leftarrow$  Louvain community detection( $G$ )    ▷ get communities( $C$ )
13:   $Error_{current} \leftarrow$  evaluate error( $C$ )
14:  if  $Error_{current} < Error_{previous}$ :  $W \leftarrow W_{current}, FG \leftarrow G_{current}$ 
15:  else:  $W \leftarrow W_{previous}, FG \leftarrow G_{previous}$ 
16:  update  $W$  as much as  $a$                         ▷  $W(1-a) < W_{updated} < W(1+a)$ 
17:  count = count + 1
18: end while
19: return  $FG, W$ 

```

Fig. 4. The pseudocode of clustering algorithm

C. Visualization of Malware Classification

We use the force-directed graph to visualize classification result [14]. The force-directed graph represents the attracting force between nodes with the weighted edge. The higher the weight of the edge between nodes is, the closer the two nodes are placed. According to the purpose of expressing the similarity of malware, rather than only showing the set of connected nodes, the graph should indicate that nodes which are classified as the same family should be located closely. Since the adjacent nodes

in the graph mean that they are highly related, it is helpful for users to understand the similarity between nodes intuitively.

IV. EXPERIMENT

A. Implementation

Analyzer is based on *Cuckoo* analysis system that is open source. The data that cannot be extracted through *Cuckoo* analysis system must be obtained by an additional method (e.g., we use ‘aapt’ command which allows reading resources and build-environment of Android so that we can get information of required permission). The virtual emulator implemented for dynamic analysis uses the Android 4.1.2 (API 16) version of the SDK platform and has ARM EABI v7a CPU. We perform Analyzer on a remote server. The CPU specification of the remote server is Intel Xeon CPU E5-1650 3.60GHz, and the RAM specification is 32GB. There are two virtual machines on the server with 64bit Ubuntu 16.04 operating system, one is used as a database for experiments, and one is used as Analyzer. The codes for clustering through social network analysis and evaluating clustering results are implemented in python using *python-igraph* module.

B. Experiment Setup

Reference labels of malware samples are diagnosed by *BitDefender* as the ground truth. It is referred to obtain an accuracy of classification. After the pre-processing of malware samples, they consist of 8 families, and the total number of samples is 682. We exclude samples that are undiagnosed and do not have one or more of features needed to get the similarity. Table. 1 shows the number of samples according to the label.

TABLE I. REFERENCE LABEL

Label	Samples	Label	Samples
FakeBank	192	SmsSpy	118
Gepew	114	Bankun	91
Gidix	49	Misosms	48
FakeInst	46	Telman	24

C. Process of Experiment

We insert them into task queue of Analyzer through Scheduler. Analyzer performs static and dynamic analysis for each sample for 10 minutes. All analysis results are stored in the database so that they can be retrieved when necessary. While the Analyzer is running, the similarities of each feature between malware samples are calculated and stored in the database. We select four features such as API call sequence, the activity name, the file name, and the required permission. The similarity of API call sequence is obtained from *Nilsimsa* hash comparison. In the case of the set of strings, which are the activity name and the file name and the required permission, are obtained using Jaccard similarity. We multiply the weights to the degree of similarity for each feature, and then the final similarity is calculated. We set the total iteration of Optimizer to 10,000 for the classification with all malware samples and 1,000 for the prediction using K-fold cross-validation. We measure the accuracy varies according to the threshold t (80% - 95%) for getting better accuracy in two

situations. After the experiment, the optimal threshold is set. The final graph and the optimal weights are derived at the end.

D. Experiment Results and Discussion

We evaluate the accuracy of Andro-Simnet in the classification and the prediction of malware samples. Also, we propose how to cope with the limitations. The details of the experiment are described below:

1) *Classification*: First, the classification experiments are repeated to determine the optimal threshold. The threshold is increased as 1 from 80 to 95. The iteration of the optimizer is set by 10,000 and initial weights are set by 0.25 equally.

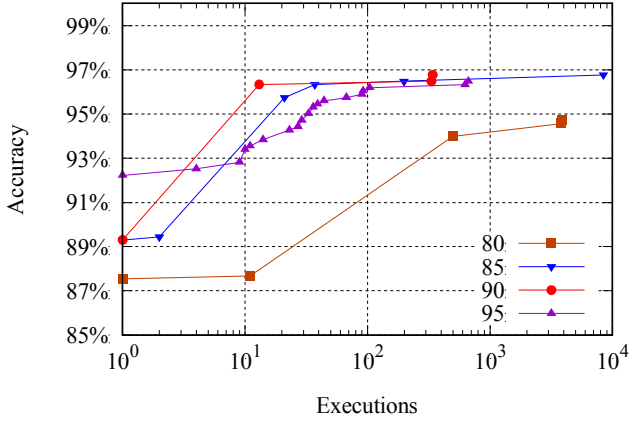


Fig. 5. The change of accuracy according to executions by threshold

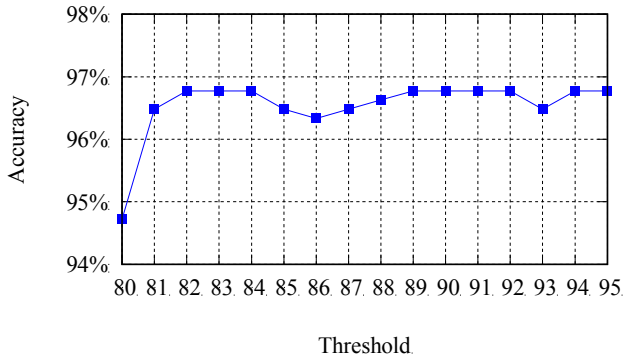


Fig. 6. The final accuracy of the experiment according to threshold

In each experiment, we can see that weights are fitting in a direction to reduce the errors. Fig. 5 shows the increasing accuracy of the experiment according to each threshold. The point on this graph is the spot that updates the optimal weights as those of the current iteration. The updating point tends to increase as the threshold value increases. The connection between nodes is broken and relatively weak when the threshold is increased. Therefore, nodes that are united tend to be spread. It causes to increase the probability that node belongs to another community. As the number of community reassignments of nodes increases, the error value changes significantly according to this change. Conversely, if the threshold is getting smaller, the

edge between nodes increases. So, a strong connection is established with nodes in the community. This means that the node has lower chance to change its belonging community to others. If there is a change in the community, errors tend to change with a large gap because many nodes are disconnected at the same time.

The final accuracy is shown in Fig. 6. The results show that the final accuracy is only slightly different according to the threshold. So, it is difficult to set the optimal threshold by only referring to this experiment. Therefore, we divide the samples into a training set and a test set and proceed through K-fold cross-validation.

2) *Prediction with K-fold Cross-validation*: In this experiment, we design the model with a high prediction accuracy for a new malware sample. For this purpose, the number of samples is divided into a training set and a test set. The machine is fitting the model with the training set and then the accuracy is measured using the test set. Since the malware samples used in this paper have a small number to be divided into two sets, we conduct this experiment through K-fold cross-validation that complements the shortage of samples.

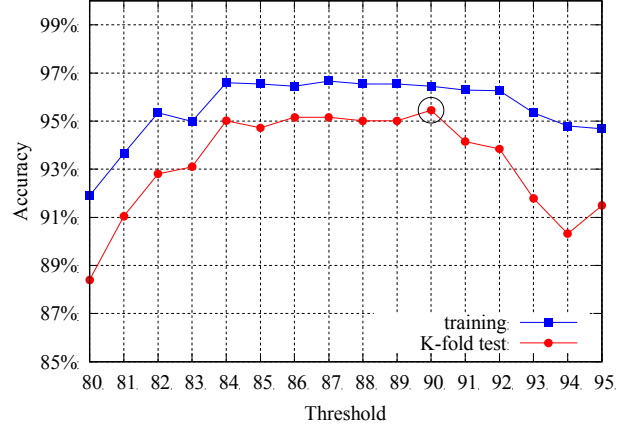


Fig. 7. K-fold cross-validation accuracy according to threshold

TABLE II. THE OPTIMAL WEIGHTS

API call sequence	Permission	Activity name	File name
0.166	0.423	0.295	0.116

We set K to 5 and the number of iterations to 1,000. As shown in Fig. 5, in the case an iteration of 1,000 or less, its accuracy converges to a final accuracy. So, setting iteration to 1,000 can use time effectively for training model. Therefore, 5,000 iterations are performed on each threshold to obtain the final accuracy. As shown in Fig. 7, the classification accuracy for the training set shows a slight difference for each threshold. The accuracy of the prediction is the highest when we set the optimal threshold to 90.

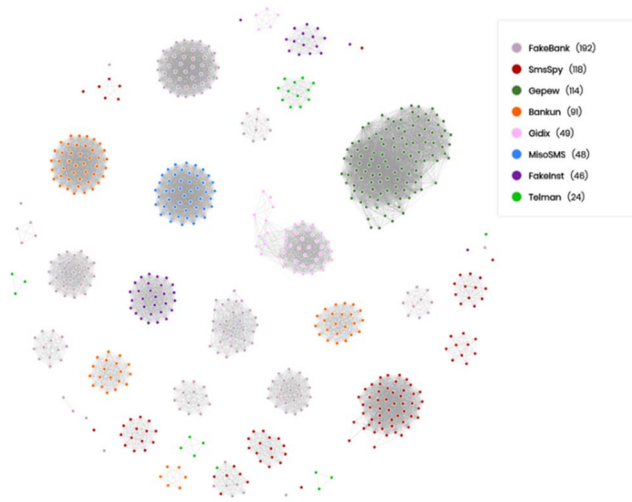


Fig. 8. The final network graph

Predicted \ Actual	Bankun	FakeBank	FakeInst	Gepew	Gidix	Misosms	smsSpy	Telman	Unlabeled
Bankun	91/91	0	0	0	0	0	0	0	0
FakeBank	0	179/192	0	0	0	0	2	0	11
FakeInst	0	0	43/46	0	0	0	0	0	3
Gepew	0	0	0	114/114	0	0	0	0	0
Gidix	0	0	0	0	49/49	0	0	0	0
Misosms	0	0	0	0	0	48/48	0	0	0
smsSpy	0	3	0	0	0	0	108/118	0	7
Telman	0	1	0	0	0	0	0	19/24	4

Fig. 9. Confusion matrix of K-fold cross-validation

Therefore, the optimal threshold is set at 90%. It results in accurate machine learning by the training set and high prediction. The accuracy of classification is more than 95% according to the k th training set. When we set the threshold to 90, Andro-Simnet has the final prediction accuracy as 95.45%, that was the highest value. Finally, we obtain the optimal weights, which are described in Table. 2. The network graph created by these weights is shown in Fig. 8, and the confusion matrix of K-fold cross-validation is expressed in Fig. 9. The entire process of Andro-Simnet is shown in the video [15].

3) *Limitation*: The limitation of our work is that the prediction accuracy is not very high. To find out the reason for low accuracy, the analysis of the experimental results is carried out. It is occurred due to 12 samples that are classified as *Unlabeled* in all K-fold cross-validation experiments. We diagnose a community that is composed of one node as *Unlabeled*. We regard these nodes as having no similarity with

other nodes. 12 samples are named *NoConnection*, and we analyze how they affect the prediction accuracy as shown in Fig. 10.

As shown in Fig. 10, the number of *Unlabeled* per errors increases as the threshold increases. The number of nodes that make up the community alone increases because the connections between nodes are severed by process of Optimizer. In most experiments, the number of *Unlabeled* accounts for more than 60%. It means that Andro-Simnet diagnoses the samples *Unlabeled* rather than misjudge as another family. It derives the result that *NoConnection* directly affects these experiments negatively. It can be seen that *NoConnection* occupies more than 20% in most experiments. Also, the proportion of *NoConnection* occupies an average of 50%. This shows that it causes errors in all experiments. So, if they are excluded from the experiment, the higher accuracy is resulted using Andro-Simnet. To improve performance, it is helpful to obtain samples that have a connection with these 12 samples.

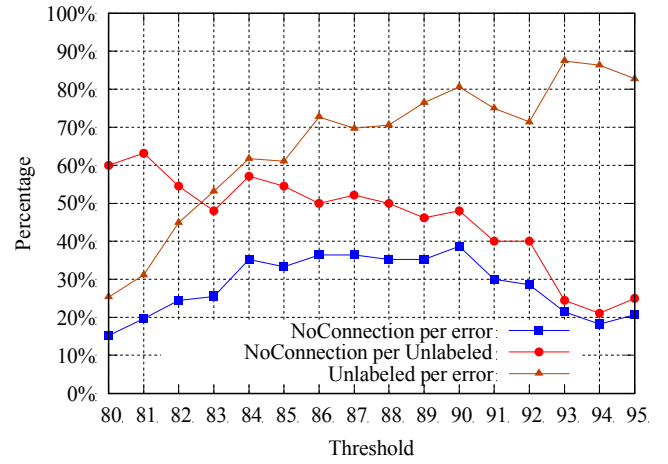


Fig. 10. The influence of *NoConnection* on K-fold cross-validation

V. CONCLUSION

In this paper, we use information obtained from the static analysis and dynamic analysis to extract malware's behavioral characteristics. We can measure the similarity value between malware samples with the collected data. Using these similarity values, we do not use the general weights that are just divided equally for average, but we use optimal weights for weighted sum through machine learning process. We think that this similarity estimation and optimal weights derivation are the key processes of clustering similar malware. Besides, Andro-Simnet also uses community detection algorithm which can express the relationship between malware's behavioral characteristics better. Also, Andro-Simnet can obtain the optimal weights using social network analysis method. With the proposed machine learning algorithm, we can describe the importance value for each feature. Compared with a heuristic decision for weight value setting, Andro-Simnet can classify malware without expert knowledge. It helps to decide whether the selected feature is useful to identify associations of malware samples.

Finally, we test Andro-Simnet with the real malware dataset to validate the performance. As a result, Andro-Simnet can find out the most similar malware when a new sample is given. We

also provide visualization results represented by a force-directed graph, which helps the security administrator and analysts to understand the results intuitively.

ACKNOWLEDGMENT

This work was supported by *Samsung Electronics Software R&D Center*.

REFERENCES

- [1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," *ACM Conference on Data and Application Security and Privacy*, pp. 183-194, March 2016.
- [2] A. D. Schmidt, R. Bye, H. G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on android," *Communications IEEE International Conference*, pp. 1-5, June 2009.
- [3] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," *ACM SIGSAC Conference on Computer and Communications Security*, pp. 1105-1116, November 2014.
- [4] Andro-Simnet dataset page - <http://ocslab.hksecurity.net/Datasets/andro-simnet>
- [5] J. W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "Andro-dumpsys: anti-malware system based on the similarity of malware creator and malware centric information," *Computers & Security*, vol. 58, pp. 125-138, May 2016.
- [6] Y. Ding, S. Zhu, and X. Xia, "Android malware detection method based on function call graphs," *International Conference on Neural Information Processing*, pp. 70-77, October 2016.
- [7] Y. Zhong, H. Yamaki, and H. Takakura, "A malware classification method based on similarity of function structure," *Applications and the Internet (SAINT)*, IEEE/IPSJ 12th International Symposium, pp. 256-261, July 2012.
- [8] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," *Security and Privacy Workshops (SPW)*, pp. 252-261, May 2016.
- [9] U. Pehlivan, N. Baltaci, C. Acartürk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based Android malware detection," *Computational Intelligence in Cyber Security (CICS)*, 2014 IEEE Symposium, pp. 1-8, December 2014.
- [10] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "AndroSimilar: robust statistical feature signature for Android malware detection," *the 6th International Conference on Security of Information and Networks*, pp. 152-159, November 2013.
- [11] CuckooDroid – <http://cuckoo-droid.readthedocs.io/>.
- [12] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "An open digest-based technique for spam detection," *ISCA Parallel and Distributed Computing Systems*, pp. 559-564, September 2004.
- [13] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, pp. 1-13, October 2008.
- [14] M. Bostock, V. Ogievetsky, and J. Heer, "D³ data-driven documents," *IEEE transactions on visualization and computer graphics*, vol. 17, pp. 2301-2309, November 2011.
- [15] The demo video of Andro-Simnet, <https://youtu.be/JmfS-ZtCbG4>