J Inf Process Syst, Vol.16, No.4, pp.882~895, August 2020
https://doi.org/10.3745/JIPS.03.0145

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

A Study on Variant Malware Detection Techniques Using Static and Dynamic Features

Jinsu Kang* and Yoojae Won*

Abstract

The amount of malware increases exponentially every day and poses a threat to networks and operating systems. Most new malware is a variant of existing malware. It is difficult to deal with numerous malware variants since they bypass the existing signature-based malware detection method. Thus, research on automated methods of detecting and processing variant malware has been continuously conducted. This report proposes a method of extracting feature data from files and detecting malware using machine learning. Feature data were extracted from 7,000 malware and 3,000 benign files using static and dynamic malware analysis tools. A malware classification model was constructed using multiple DNN, XGBoost, and RandomForest layers and the performance was analyzed. The proposed method achieved up to 96.3% accuracy.

Keywords

Computer Security, Dynamic Analysis Machine Learning, Metamorphic, Polymorphic, Static Analysis, Windows Malware

1. Introduction

Malware is software created to interfere with computer operation or to compromise computer systems, such as by collecting unauthorized information, ignoring access control, or performing unintended actions. Recently, due to the development of malware generation tools, anyone can easily create malware. Consequently, the amount of malware has increased exponentially, increasing the threats to computer systems and networks. According to the 2019 AV-Test Report [1] on malware trends, approximately 961 million types of malware are used for advance persistent threat (APT) attacks, distributed denial-of-service (DDoS), etc. Among these malware types, new malware constitutes less than 20%, and most malware is variant malware. Because variant malware uses polymorphic and metamorphic techniques to avoid signature-based detection, it is difficult for human experts for deal with large-scale malware variants.

To address this challenge, some studies were conducted, wherein features have been extracted from malware, and machine learning has been applied. The features of malware can be extracted through static and dynamic analysis. A static feature is one that can be extracted from the file itself using analysis tools without directly executing malware such as PE (portable executable) header information, DLL (dynamic-link library), section information, instructions, and strings. Dynamic features are related to file, process, service, and network information after malware is run in an environment that is disconnected from the

Manuscript received November 12, 2019; first revision April 9, 2020; accepted April 14, 2020.
Corresponding Author: Yoojae Won (yjwon@cnu.ac.kr)
* Dept. of Computer Science Engineering, Chungnam National University, Daejeon, Korea (kangjs1798@gmail.com, yjwon@cnu.ac.kr)

outside.

In this study, we used static and dynamic analysis tools to extract the feature data of various malware and used them to distinguish between malware and benign files. In particular, we defined application programming interface (API) sequences related to specific malicious code behaviors and analyzed the data to see if the API sequences appeared. We defined about 150 API sequences and analyzed the effects of the features on the accuracy. By utilizing ensemble-based machine learning algorithms and deep learning, we generated malware classification models and assessed their performance.

We compared the performance of the learning model when the features used in the existing research methods were employed and with that when our features were utilized. We then evaluated the performance of the model using raw data samples to assess the adequacy of the study.

This paper organized as follows. Section 2 discusses related work, Section 3 presents the methodology. Section 4 presents the results of the experiment and Section 5 summarizes the paper and discusses the future work.

2. Related Work

This section describes the existing studies on malware classification using machine learning. Many researchers have proposed machine-learning-based malware classification methods. Previous research can be categorized according to the features used to construct the malware classification model and the type of learning algorithm in the model.

In several studies, operation code (opcode) and byte sequences have been used as feature data among the static features of malware [2-6]. For instance, Rathore et al. [2] disassembled the executable using objdump and extracted the opcode from the file. A master opcode list consisting of 1,600 unique opcodes was constructed and used as feature data. The RandomForest (RF) algorithm and various layers of a deep neural network (DNN) were constructed to detect malicious files. Subsequently, Zak et al. [3] experimentally verified the influence of the byte N-gram on the file maliciousness. They compared the malware classification performance achievable using a PE header, section information, and opcode N-gram. It was confirmed that the byte N-gram reflects the intention of the attacker and can be an important element in detecting malware. Santos et al. [4] proposed a means of identifying obscure families of malware. Their model depends on the recurrence of opcode grouping. In addition, they described a system for mining the importance of each opcode and evaluating recurrence. Each opcode grouper is also a strategy for recognizing this new obscure malware. Further, Ki et al. [7] extracted the API call sequence through dynamic analysis and detected malicious code by checking if a new file had a specific API call sequence. Jerlin and Marimuthu [8] proposed a method for efficiently classifying malware using an API. They used the dimensional naïve Bayes classification and Rete algorithms to classify potential malware as a worm, virus, Trojan, or normal file. Ahmadi et al. [9] investigated malware classification by extracting various static features from the dataset provided in the Microsoft Malware Classification Challenge and then combined them. They divided the extracted features into several subsets, performed learning using each subset, and analyzed the performance of each feature. The results indicated that the performance increases as the number of features increases. Static analysis provides the advantage of taking less time to extract features. However, feature extraction is difficult if the malware is packed or obfuscated.

To address this issue, studies have also been conducted to extract features using dynamic analysis. For example, Rieck et al. [10] used the CWSandbox to extract features such as API calls, file systems, registry key change process information, network activity, and Windows service information from 3,139 types of malware. Further, Mohasien et al. [11] conducted a study to identify malware using the self-generated malware dynamic analysis tool. High performance accuracy was obtained by extracting and normalizing system state change information, such as the file system, registry key, network information, etc., from malware samples and applying vertical clustering and the support vector machine algorithm. Although dynamic analysis can detect malware using the obfuscation technique, malware that avoids virtual environments cannot be detected by dynamic analysis. Various learning algorithms have been used to detect malware.

Many studies have been performed to classify malware by constructing deep learning models. Nari and Ghorbani [12] proposed a framework for classifying malware based on network behavior. After extracting the network flow from a pcap file, an action graph representing the network activity was created. Malware and dependencies between network flows In these behavioral graphs, features such as graph size were utilized to extract the root affinity, average affinity, maximum affinity, and number of specific nodes. Then, the malware was detected using the classification algorithm available in the WEKA library, achieving the highest accuracy with the J48 decision tree. Saxe and Berlin [13] extracted static features such as PE metadata, PE import features, and string histogram features and constructed a DNN to classify 400,000 types of malware. Further, Ma et al. [14] attempted to start with analysis of the corresponding relationship between the assembly code and binary code. They designed a malware classification framework using deep learning and measured the performance with the malware data set provided by Microsoft.

Recently, a method of converting malware into images for classification has been studied [15-18]. The malware was converted into gray-scale images, and a convolution neural network (CNN) was trained to classify malware. The Malimg data and malware data from Microsoft were utilized to measure the malware classification performance through imaging.

3. Methodology

This section presents the general malware classification method utilized in this study. Fig. 1 shows the steps in this classification process. The entire process consists of collecting the data, changing the hex value for feature extraction from the raw data, disassembling all executables, for feature extraction from different types of files, constructing the dimension reduction and learning model, and performing estimation and analysis according to the measurement standard.

3.1 Data Preprocessing

To extract feature for machine learning, malware must be analyzed. We extract both static and dynamic features of malware. To extract static feature, we transformed the executables to another type file. To do this, we used the command-line program objdump, which displays information about libraries, compiled object modules, native object files, and binaries of standalone files as part of the GNU binary utility. We convert executable into hexadecimal file (.byte), and assembly code (.asm) as shown in Figs. 2 and 3. In addition, we extract the PE header information as shown in Fig. 4.

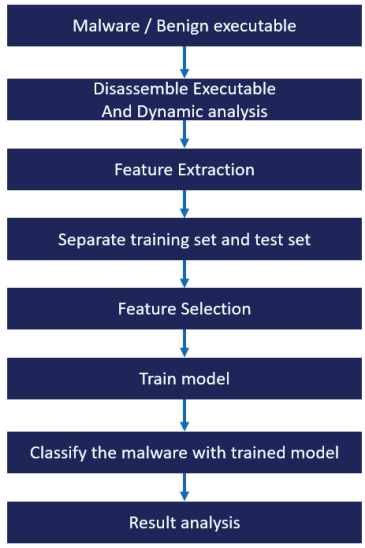


Fig. 1. Flowchart of malware classification process using machine learning.

```
cuckoo@cuckoo-VirtualBox:~/kjs$ objdump -s /home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir

/home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir:     file format pei-i386

Contents of section .text:
 401000 33c03944 240c7615 8b4c2408 8a0c088b  3.9D$.v..L$.....
 401010 54240488 0c10403b 44240c72 ebc20c00  T$....@;D$.r....
 401020 558bec81 ec380800 00535657 33f656ff  U....8...SVW3.V.
 401030 15202040 0083f8ff 75086a01 ff152420  .  @....u.j...$
 401040 400056bf 00200000 5756ff15 28204000  @.V.. ..WV..( @.
 401050 576a0850 8945ecff 152c2040 00576a08  Wj.P.E...., @.Wj.
 401060 ff75ec8b d8ff152c 20400057 53568945  .u.....,  @.WSV.E
 401070 fcff1530 204000ff 75fc6800 100000ff  ...0 @..u.h.....
 401080 15002040 00686821 4000ff75 fc687c21  .. @.hh!@..u.h|!
 401090 4000ff75 fcff1548 20400083 c4105668  @..u...H @....Vh
```

Fig. 2. Converting executables to hex files using objdump.

```
cuckoo@cuckoo-VirtualBox:~/kjs$ objdump -d /home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir

/home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir:     file format pei-i386


Disassembly of section .text:

00401000 <.text>:
  401000:       33 c0                   xor    %eax,%eax
  401002:       39 44 24 0c             cmp    %eax,0xc(%esp)
  401006:       76 15                   jbe    0x40101d
  401008:       8b 4c 24 08             mov    0x8(%esp),%ecx
  40100c:       8a 0c 08                mov    (%eax,%ecx,1),%cl
  40100f:       8b 54 24 04             mov    0x4(%esp),%edx
  401013:       88 0c 10                mov    %cl,(%eax,%edx,1)
  401016:       40                      inc    %eax
```

Fig. 3. Converting executables to assembly code using objdump.

```
cuckoo@cuckoo-VirtualBox:~/kjs$ objdump -x /home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir

/home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir:     file format pei-i386
/home/cuckoo/malware/Student/TrainSet/d663aa06aa48ddd34c2a559e014b87fe.vir
architecture: i386, flags 0x0000012f:
HAS_RELOC, EXEC_P, HAS_LINENO, HAS_DEBUG, HAS_LOCALS, D_PAGED
start address 0x00401020

Characteristics 0x102
        executable
        32 bit words

Time/Date               Tue Nov  5 19:07:10 2013
Magic                   010b    (PE32)
MajorLinkerVersion      10
MinorLinkerVersion      0
SizeOfCode              00000400
SizeOfInitializedData   00000c00
SizeOfUninitializedData 00000000
AddressOfEntryPoint     00001020
```

Fig. 4. Extracting header information from executable using objdump.

Then, we extract malware's behavioral characteristics using dynamic analysis tool. Dynamic analysis tool run malware in isolated environments and analyze file system, process and network. We used Cuckoo Sandbox to analyze the behavior of malware. Cuckoo Sandbox dynamically analyzes malware and provides a report file in JSON format. Analysis reports include behavioral processes, files, and registry and static file analysis. Contains network analysis result. We parse the report and extract significant dynamic feature.

3.2 Feature Extraction

In all machine algorithms, the features are major factors in the model performance. Static and dynamic analyses involve feature extraction from the executable. This section describes the features extracted for malware classification.

3.2.1 Byte sequence

The hex value representation of the executable file is often an adequate feature with which the similarity of a file can be analyzed. Many current types of malware are malicious variants of malware. In other words, existing malware is reused, such that the new malware is similar to the existing malware. N-grams are widely used to analyze the similarity of hex values. The N-gram method involves dividing a given long string into substrings of size N and analyzing all the data by calculating the statistics for each substring. By dividing the hex file into a sequence of bytes and calculating the frequency of each sequence, the similarity of all the files can be assessed. When extracting the frequency of a byte (N = 1), a 256-dimensional feature vector is generated from 0×00 to 0×FF. As the value of N increases, both the amount of operation and the dimension of the feature increases. To consider the computational complexity, 1- and 2-gram data were extracted and used as features.

3.2.2 opcode

Because binary files cannot be intuitively understood by people, file analysis through it alone is limited. Thus, it is necessary to analyze files through other methods, such as opcode; opcode is a representation of the machine language that facilitates understanding. The frequency of the opcode sequence can be measured and used as feature data by measuring the frequency of appearance of the opcode or applying the N-gram technique. However, since the entire list of instruction sets is very large, the ability to utilize the instruction sets as feature data is limited. Accordingly, we selected a list of frequently used or frequently accessed opcodes in malware and measured the frequencies of the corresponding opcodes in the executable file.

3.2.3 Section information

A PE executable consists of predefined sections such as .text, .data, .idata, .edata, .rdata, .rsrc, .tls, and .reloc sections. However, because it does not enforce a defined section name, the section name can be modified, and an arbitrary section name can be created. Although normal files tend to conform to predefined section names, malicious files may use section names modified by techniques such as packing. Therefore, the section name can be used as a key feature in malware analysis. In addition, because the contents of the file are divided into sections, the features can be divided into sections and extracted as

features. Typically, the entropy can be measured for each section to be used as feature data, or only a specific section can be visualized for analysis. We used the section of the executable file and entropy of each section as feature data.

3.2.4 Data define

Some packed malware samples do not extract opcode or API information, even if the file is disassembled. Fig. 5 shows the unpacking process when a packed file is executed. When the packed file is executed, the unpacked code is recorded in the memory, moved to the OEP, and executed. When writing unpacking code to the memory, the related db, dw, and dd are executed. These features can also be reflected in the feature.

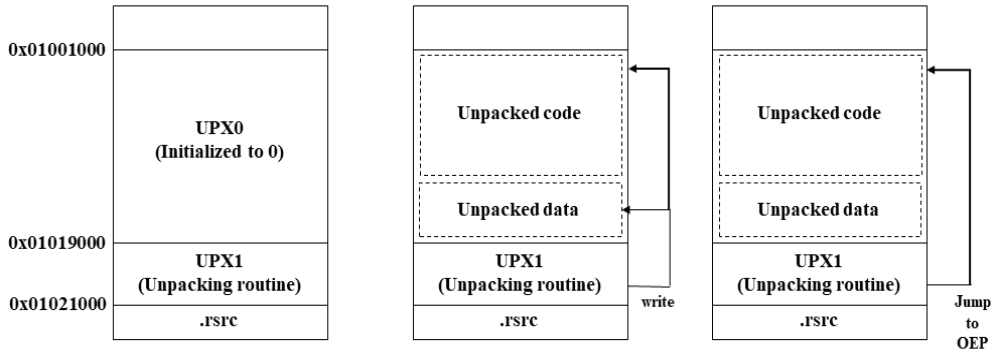


Fig. 5. Example of unpacking behavior of packed files.

3.2.5 Image representation

One means of representing malware is to visualize it as images. As shown in Fig. 6, each byte of the file is interpreted as the gray level of one pixel in the image, and these are converted into an image [17].

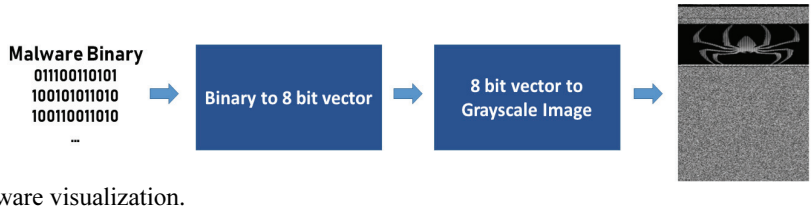
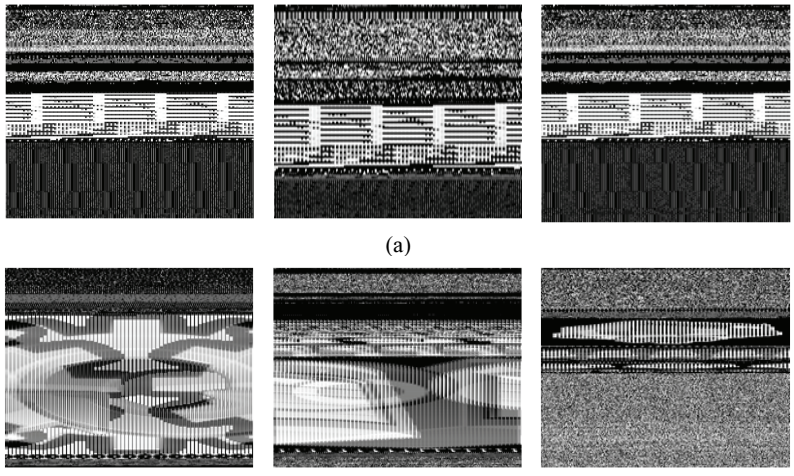


Fig. 6. Malware visualization.

The resulting images may show similar patterns for similar files (Fig. 7(a)), including special patterns for some malware (Fig. 7(b)). Though it takes time to produce image files and process matching visual patterns, doing so can be effective for classifying malware. Additionally, when image files are used to classify malware, it is possible to apply an effective machine learning algorithm, such as a CNN, to process the images. However, in some cases, files belonging to different classes show the same pattern; there is a case in which further processing is required to address image size discrepancies, wherein the data learning file sizes differ considerably. Therefore, these features should be applied together with other features. To reduce the processing time, only the .text section of the executable file was used as feature data.



(a)

(b)

Fig. 7. Image representations of malware samples: (a) image patterns found in malware and (b) images embedded in malware.

3.2.6 API sequence

By analyzing the header of the PE file, the DLL and API information that the file imports can be reviewed. Because this information describes the APIs that are called when the file is executed, important information is provided for predicting the actual behavior of the malware. Analyzing the API information provides an understanding of the behavior of the file. Accordingly, the frequency of API usage was measured and used as feature data. In this process, if an excessively high-dimensional feature is created, the performance of the entire model is adversely affected. Therefore, we selected an API that was effective for analyzing the malware, measured the frequency of the API, and used it as feature data. Some files are packed such that API features cannot be extracted with static analysis tools alone. In this case, information was acquired from the dynamic analysis report file and used as feature data.

We also extracted the API sequence and used it as feature data. We have defined API sequences that frequently appear in malware and dynamic analysis to measure the occurrence of API sequences when the malware is running. Table 1 lists some of the API pattern that appears when an executable file performs certain malicious behavior. After extracting the API sequence from the dynamic analysis report file to generate a string, it was checked whether the API pattern for malicious behavior was included. The longest common subsequence (LCS) was used to extract malicious API call sequence patterns from malware. The formula of the LCS is shown in (1). In this formula, X_i and Y_i represent the ith characters of sequences X and Y, respectively:

$$\mathrm{LCS}(X_i, Y_i) = \begin{cases} 0 & if\ i = 0 \\ & or\ j = 0 \\ \mathrm{LCS}(X_{i-1}, Y_{i-1}) + common\ character & if\ X_i = Y_i. \\ longest(\mathrm{LCS}(X_i, Y_{i-1}), \mathrm{LCS}(X_{i-1}, Y_i)) & if\ X_i \neq Y_i \end{cases} \quad (1)$$

3.3 Feature Selection

Before building a learning model, a feature must be selected from the extracted features to be used for learning. If all of the extracted features were used at the time of learning, the calculation process would

take longer and the performance would be affected. Therefore, the feature to be used for learning should be distinguished through the feature selection process. This process is performed as follows (Fig. 8). Firstly, the number of features is reduced through dimension reduction. The best feature set is then selected by utilizing feature ranking and selection techniques. The selected best feature set is then used to construct a learning model.

Table 1. Some malicious API patterns

Malicious activity	API pattern
Keylogger	(FindWindowA, ShowWindow, GetAsyncKeyState) (SetWindowsHookEx, RegisterHotKey, GetMessage, UnhookWidowsHookEx)
Downloader	URLDownloadToFile, (WinExec, ShellExecute)
DLL injection	OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread Operability
Dropper	FindResource, LoadResource, SizeOfResource
Anti-debugging	(IsDebuggerPresent, CheckRemoteDebuggerPresent, OutputDebugStringA, OutputDebugStringW)
IAT hooking	LoadLibray, (strcmp,strncmp,_stricmp,_strnicmp), VirtualProtect
Screen capture	(GetDC, GetWindowDC), CreateCompatibleDC, CreateCompatibleBitmap, SelectObject, BitBlt, WriteFile
Encryption	CryptCreateHash, CryptHashData,CryptGetHashparam, CryptAcquireContextA
Registry behavior	RegSetValueExA, RegEnumValueA,RegDeleteKeyA, RegCreateKeyExA, OpenProcessToken

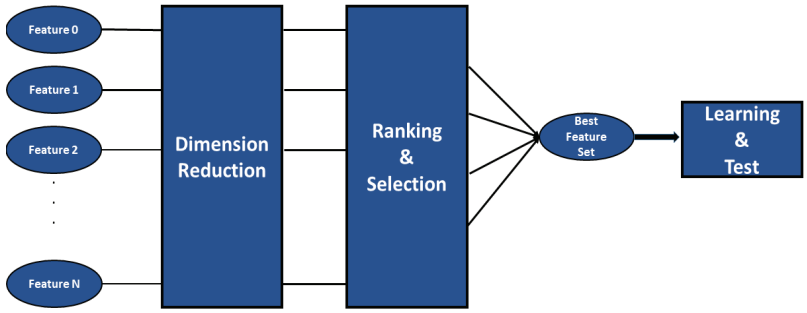


Fig. 8. Feature selection.

3.4 Machine Learning Models

This section describes the machine learning algorithm used in this study. We constructed an ensemble-based classifier and DNN to generate a malware classification model. The following subsections describe these learning models in detail.

3.4.1 Ensemble based model

The ensemble technique performs better than a single model that uses several weak classifiers (Fig. 9). Ensemble techniques can be divided into two types: bagging and boosting (Fig. 10). In bagging, each classifier used in the ensemble method operates independently of the others. In boosting, each classifier is gradually connected to reduce the error by considering the errors occurring in the previous classifier in the next classifier. In this study, the model was built using RF algorithm, a representative bagging technique, and XGBoost, a boosting technique. The RF algorithm has shown sufficient performance compared

to other classifiers in existing research, and XGBoost has achieved the best performance in various competitions hosted by Kaggle. Accordingly, the learning models utilizing each method were built and their performances were compared.

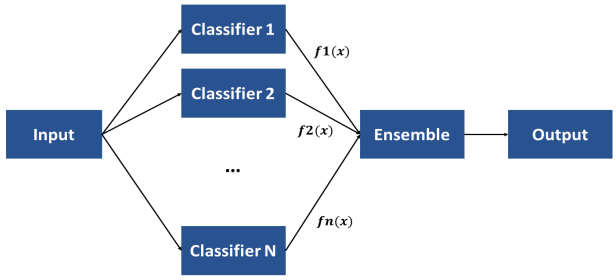


Fig. 9. Ensemble method.

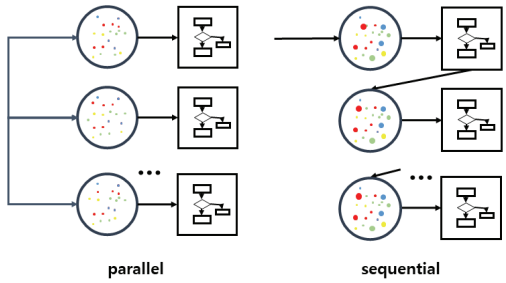


Fig. 10. Type of ensemble method: bagging (left) and boosting (right).

3.4.2 Deep learning based model

Deep learning is a model that effectively solves complex problems. To implement a DNN, we constructed a learning model with various depths and investigated each case. Through the feature selection process, 1,024 features were selected and used as input. The RELU function was utilized as the activation function for all the hidden layers, and the softmax function was employed as the activation function of the output layer. The softmax function was used because malware classification is a type of binary classification. We also used dropout to reduce overfitting when implementing the DNN, and we applied a dropout rate of 0.2 in several experiments. We constructed DNN models with three, five, and six hidden layers.

1. Deep Neural Network using three hidden layers (DNN-5L).
2. Deep Neural Network using five hidden layers (DNN-7L).
3. Deep Neural Network using seven hidden layers (DNN-9L).

The hidden layers in DNN-5L, DNN-7L, and DNN-9L contain (512, 64, 16), (512, 256, 128, 64, 32), and (512, 256, 128, 64, 32, 32 16) nodes, respectively.

3.5 Evaluation

We evaluate the accuracy, precision, recall, F1-score of our malware classification model. These indicators can be estimated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FR}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

where TP is the number of malware data truly predicted to be malware; TN, the number of benign data truly predicted to be benign; FP, the number of benign data truly predicted to be malware; and FN, the number of malware data truly predicted to be benign.

4. Experiment

4.1 Dataset

To construct malware classification model, labeled malware dataset is essential. In our experiments, we will be using malware dataset obtained from the Korea Information Security Industry Service (KISIS) [20]. This dataset is provided by many anti-virus companies. The dataset consists of 7,000 malware and 3,000 benign executables and 28% of datasets are packed. We unpacked the packed data using packing tool such as UPX. Then we transformed the data into a form suitable for extracting features.

4.2 Experiment Result

We create classifier with RF, XGBoost and DNN and measure the performance of each model. The performance of the model is analyzed by measuring accuracy, precision, and recall. Using the static and dynamic features used in the previous research and the features used in our research, we trained the models and compared the results. Experiments showed that the overall performance of the XGBoost model was better than that of other models. In addition, when the API behavior feature extracted from our study is used for model training, it is confirmed that the accuracy is higher than that of the existing research method. Figs. 11–15 and Table 2 show the performance measurement results for each model.

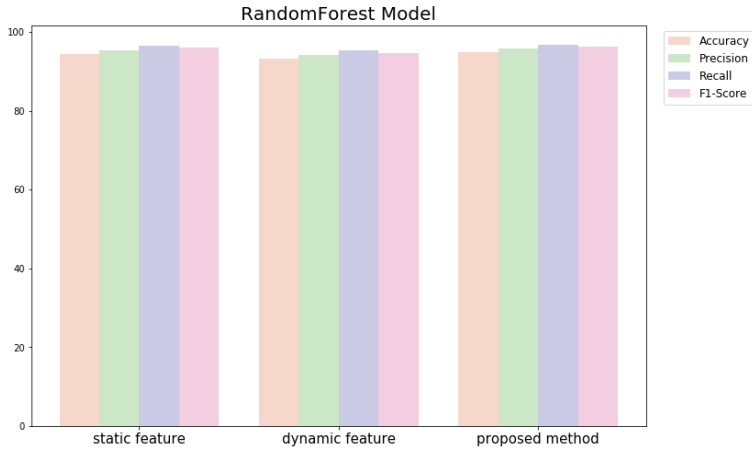


Fig 11. Performance comparison of RandomForest model by feature.

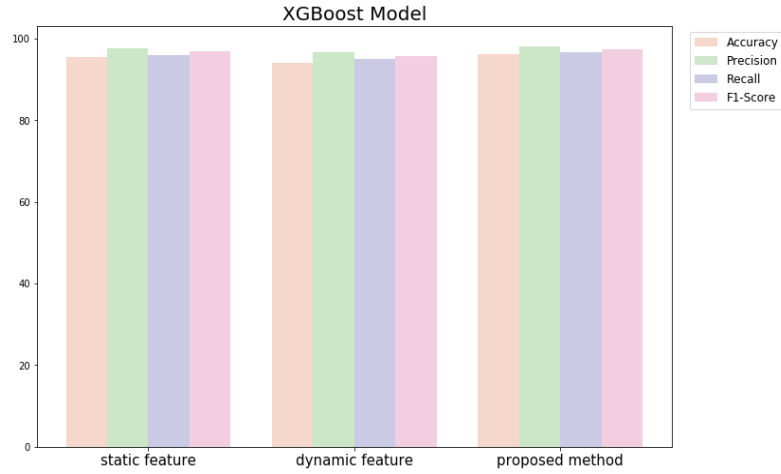


Fig. 12. Performance comparison of XGBoost model by feature.

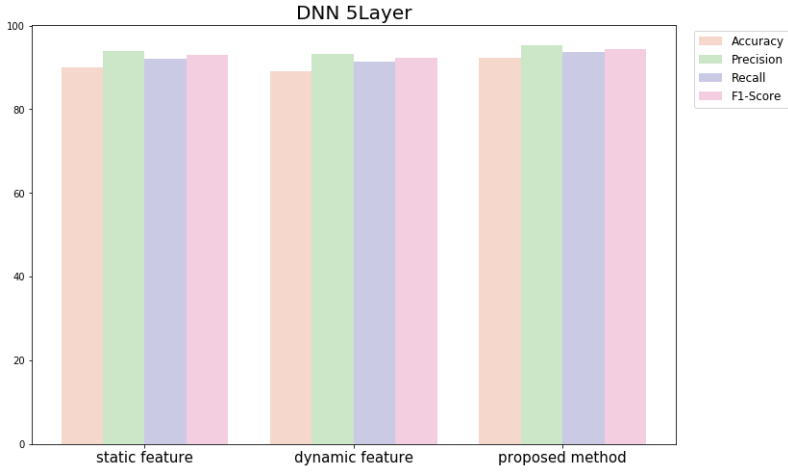


Fig. 13. Performance comparison of DNN by feature (5-Layer).

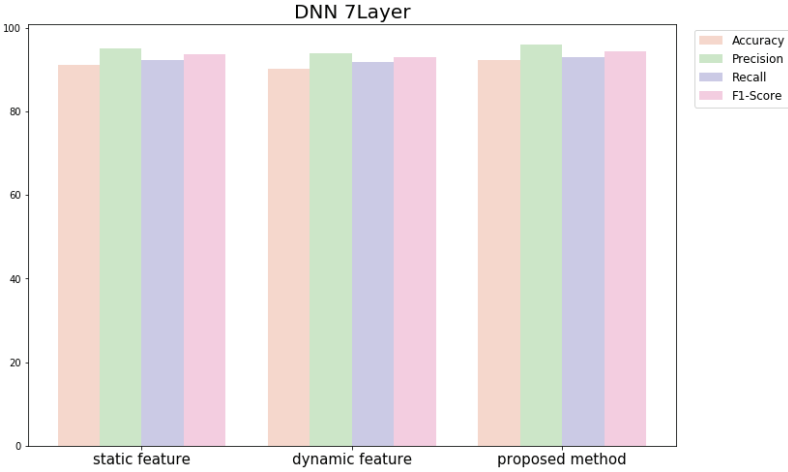


Fig. 14. Performance comparison of DNN by feature (7-Layer).

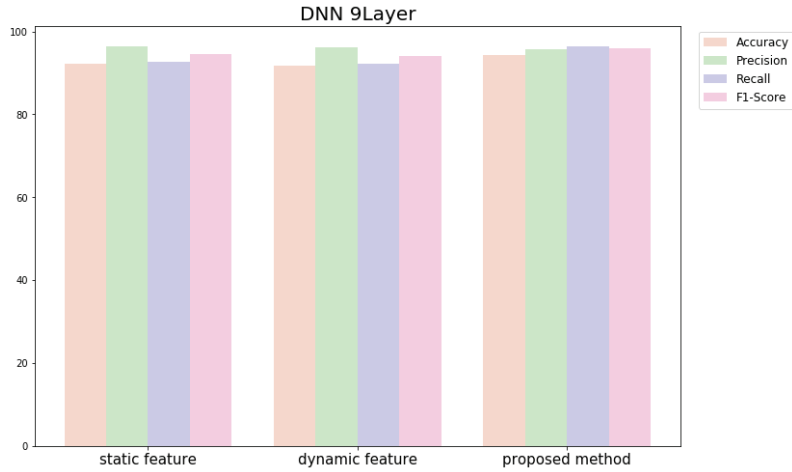


Fig. 15. Performance comparison of DNN by feature (9-Layer).

Table 2. Performance measurement results for each model

Model	Feature	Accuracy (%)	Precision (%)	Recall (%)	F1-score
RandomForest	Static	94.4	95.4	96.5	95.9
	Dynamic	93.1	94.1	95.3	94.7
	Proposed method	94.8	95.6	96.8	96.2
XGBoost	Static	95.6	97.7	96.0	96.9
	Dynamic	94.1	96.6	95.0	95.8
	Proposed method	96.3	98.2	96.7	97.5
5-Layer DNN	Static	89.2	93.9	92.1	92.9
	Dynamic	92.3	93.3	91.4	92.3
	Proposed method	91.1	95.4	93.6	94.5
7-Layer DNN	Static	91.1	95.0	92.4	93.7
	Dynamic	90.3	93.9	91.9	92.9
	Proposed method	92.2	96.1	92.9	94.5
9-Layer DNN	Static	92.2	96.5	92.6	94.5
	Dynamic	91.8	96.2	92.3	94.2
	Proposed method	94.4	95.6	96.4	95.9

5. Conclusion

In this paper, we proposed a technique to detect malware effectively using machine learning. In order to effectively distinguish between malicious codes and benign files, we defined API sequences mainly used in malicious codes, extracted them through dynamic analysis, and used them as feature data for machine learning. We extracted feature data that was used in previous studies by performing static and dynamic analysis on a malware dataset provided by KISIS. A feature selection process was applied to the extracted feature data to reduce the number of features, so that the learning model could operate effectively. We constructed a machine learning model using XGBoost, RF, and DNN classifiers and detected malware using each model. In our experiments, we achieved up to 96.3% accuracy, which is better than conventional methodologies. In the experiment results, we found that we can improve the performance of the malware classification model by using the feature that distinguishes between malware

and benign files. One factor to consider in constructing a machine-learning model for malware classification is time. To classify large amounts of malware quickly, the feature data must be extracted rapidly and models must be created that can perform well with lightweight features. Therefore, in the future, we plan to study features that can considerably improve the performance with light features that take less time to extract.

Acknowledgement

This work was supported by the Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01343, Training Key Talents in Industrial Convergence Security).

References

[1] AV-TEST, "AV-TEST Annual Report," 2018 [Online]. Available: https://www.av-test.org/en/about-the-institute/publications/.

[2] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware detection using machine learning and deep learning," in *Big Data Analytics*. Cham: Springer, 2018, pp. 402-411.

[3] R. Zak, E. Raff, and C. Nicholas, "What can N-grams learn for malware detection?," in *Proceedings of 2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, Fajardo, Puerto Rico, 2017, pp. 109-118.

[4] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences*, vol. 231, pp. 64-82, 2013.

[5] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using opcode representation," *Intelligence and Security Informatics*. Heidelberg: Springer, 2018, pp. 204-215.

[6] A. Yewale and M. Singh, "Malware detection based on opcode frequency," in *Proceedings of 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, Ramanathapuram, India, 2016, pp. 646-649.

[7] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, article no. 659101, 2015.

[8] M. A. Jerlin and K. Marimuthu, "A new malware detection system using machine learning techniques for API call sequences," *Journal of Applied Security Research*, vol. 13, no. 1, pp. 45-62, 2018.

[9] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*, New Orleans, LA, 2016, pp. 183-194.

[10] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Heidelberg: Springer, 2008, pp. 108-125.

[11] A. Mohaisen, O. Alrawi, and M. Mohaisen, "AMAL: high-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251-266, 2015.

[12] S. Nari and A. A. Ghorbani, "Automated malware classification based on network behavior," in *Proceedings of 2013 International Conference on Computing, Networking and Communications (ICNC)*, San Diego, CA, 2013, pp. 642-647.

[13] J. Saxe and K. Berlin, “Deep neural network based malware detection using two dimensional binary program features,” in *Proceedings of 2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, Fajardo, Puerto Rico, 2015, pp. 11-20.

[14] X. Ma, S. Guo, H. Li, Z. Pan, J. Qiu, Y. Ding, and F. Chen, “How to make attention mechanisms more practical in malware classification,” *IEEE Access*, vol. 7, pp. 155270-155280, 2019.

[15] Z. Cui, F. Xue, X. Cai, Y. Cao, G. G. Wang, and J. Chen, “Detection of malicious code variants based on deep learning,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187-3196, 2018.

[16] D. Gibert, C. Mateu, J. Planes, and R. Vicens, “Using convolutional neural networks for classification of malware represented as images,” *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 1, pp. 15-28, 2019.

[17] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: visualization and automatic classification,” in *Proceedings of the 8th international Symposium on Visualization for Cyber Security*, Pittsburgh, PA, 2011, pp. 1-7.

[18] L. Chen, “Deep transfer learning for static malware classification,” 2018 [Online]. Available: https://arxiv.org/abs/1812.07606.

[19] S. Gupta, H. Sharma, and S. Kaur, “Malware characterization using windows API call sequences,” in *Security, Privacy, and Applied Cryptography Engineering*. Cham: Springer, 2016, pp. 271-280.

[20] Korea Information Security Industry Service, "Large malware dataset," 2018 [Online]. Available: https://www.kisis.or.kr/kisis/subIndex/376.do.

JinSu Kang **https://orcid.org/0000-0002-6190-5028**

He received his bachelor's degree in computer engineering from Chungnam National University in 2018. Currently, he is pursuing his master's degree in computer engineering from Chungnam National University.

Yoojae Won **https://orcid.org/0000-0002-7706-5983**

He received his B.S. and M.S. degrees from the Department of Computational Statistics at Chungnam National University, Korea, in 1985 and 1987, respectively. He received his Ph.D. from the Department of Computer Science Engineering at Chungnam National University, Korea, in 1998. He worked on Wireless Internet Information Security at Electronics and Telecommunications Research Institute from February 1987 to February 2001. He worked on Mobile Security at AhnLab from March 2001 to August 2004. He worked on incident handling and was in charge of management planning at Korea Internet & Security Agency from September 2004 to February 2014. Currently, he is currently a professor in the Department of Computer Science Engineering at Chungnam National University.