Information Sciences 546 (2021) 420–435

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification ☆

Suyeon Yoo [a], Sungjin Kim [b,*], Seungjae Kim [c], Brent Byunghoon Kang [a,**]

[a] School of Computing, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea
[b] Dept. of Intelligent System Engineering, Cheju Halla University, Jeju-si, Republic of Korea
[c] R&D center, SGA Solutions, 25 Beobwon-ro 11-gil, Songpa-gu, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Received 15 January 2019
Received in revised form 29 July 2020
Accepted 23 August 2020
Available online 2 September 2020

Keywords:
Deep learning
Hybrid detection
Malware
Random forest
Voting

ABSTRACT

The extremely diffused architecture of the Internet enables the propagation of malware and presents a significant challenge for the development of defenses against such malware propagation. Although machine learning-based malware detection models can improve approaches in response to this problem, their detection rates vary according to their features and classification methods. Single machine learning approaches for malware detection can vary in effectiveness according to the suitability of their classifiers despite the use of an appropriate training dataset. Some classifiers result in high detection rates with a malicious training dataset but have low detection rates with a benign training dataset, and false positive rates are particularly dependent on the use of appropriate classifiers. In this paper, we propose a machine learning-based hybrid decision model that can achieve a high detection rate with a low false positive rate. This hybrid model combines a *random forest* and a *deep learning model* using 12 hidden layers to determine malware and benign files, respectively. This model also includes certain proposed voting rules to make final decisions. In an experiment involving 6,395 atypical samples, this hybrid decision model achieved a higher detection rate (85.1% and standard deviation of 0.006) than that of the prior model (65.5%) without voting rules.

1. Introduction

Malware is a powerful tool used in cyberattacks and has several individual variants that can easily reproduce and propagate. For this reason, large malware outbreaks have become common in recent years.

In particular, adversarial malware binaries have become prevalent and are more capable of evading deep learning models of malware detection than other types of malware. For example, an attacker can embed a "nematode" image *y* (with 8.2% confidence) in a "panda" image *x* (with 57.7% confidence), resulting in a "gibbon" image *z* (having 99.3% confidence) [44]. Adversaries add garbage random bytes to the end of binaries to avoid being accurately detected as malware by reproducing as a new type of malware. Hence, if a given antivirus uses a convolutional neural network (CNN)-based deep learning model

☆ We thank the reviewers for their insightful comments of the manuscript. This research was supported by the NRF (NRF-2017R1A2B3006360), IITP (IITP-2017-0-01889), and ONR (N00014-18-1-2661) grants.
* Co-primary author.
** Corresponding author.
E-mail addresses: yoosuyeon@kaist.ac.kr (S. Yoo), r3dzon3@chu.ac.kr (S. Kim), h112358132134@korea.ac.kr (S. Kim), brentkang@kaist.ac.kr (B.B. Kang).

https://doi.org/10.1016/j.ins.2020.08.082
0020-0255/

to classify malware, adversarial malware may still pass that detection. These attacks [37,44], known as automated poisoning attacks, are currently used, and such avoidance techniques are commonly used in attacks that employ packers.

Through these attacks, adversaries can disrupt user privacy via *banker* for later use in voice phishing. The proportional use of *ransomware* has continuously and rapidly increased over the last three-year period for monetary gain. Incidents involving both these malware types are closely related to financial fraud and confidential theft. They are distributed over a variety of routes, such as emails or the Web, and the number of daily observed malware is countless.

Thus, several studies have been conducted to tackle malware delivery with zero-day attacks or adversarial attacks [7–9,11,27,32]. Among them, [9,11,32] used dynamic-based approach methods to rapidly deactivate malware. [18] utilized machine learning (ML)-based approaches to hamstring the dissemination of malware. However, previous studies [14,16] showed difficulties in distinguishing between adversarial attacks and the behavior of certain benign files, which resulted in false positives, thereby diminishing the value of this approach as a practical model.

We propose a hybrid ML-based malware detection model for critical security tasks of tracing malware propagation through the Web and email as well as for precise classification of benign files. Rather than simply using a combined model with static and dynamic features, this model uses a variety of hybrid approaches to improve detection rates, including hybrid features, hybrid classifiers, and hybrid voting rules. This hybrid approach optimizes precision in the classification of malware and benign files, providing a meaningful resource for blocking malware proliferation, particularly for automated analysis approaches. The main contributions of this study are as follows:

- A new hybrid model is proposed to facilitate quick and concise malware detection to prevent their distribution. Compared to extant models that do not include voting rules, the proposed model enhances detection accuracy by up to 19%, and can identify malicious items within approximately 60.9 s without user analysis.
- It is demonstrated that the proposed model's performance is comparable to that of current commercial models.
- The proposed progressive approach enhances the detection rate through feature analysis, application of an ensemble of classifiers, and voting-rule optimization.

The remainder of this paper is organized as follows. Section 2 presents relevant extant studies and describes the proposed approach. Section 3 provides an overview of the different features and classifiers included in the proposed detection model. Section 4 explains the system framework and provides technical details concerning implementation of the proposed approach. The dataset used is subsequently described in Section 5 along with the experimental setup and results obtained therefrom. Limitations of the proposed model are discussed in Section 6, and major conclusions drawn from this study are highlighted in Section 7.

2. Related works

In this section, we introduce the detection methods used in previous models. Then, we describe differences in these methods compared with those used in the proposed model.

The traditional approach for malware detection uses antivirus software and spyware scanners. However, these classical detection methods, which use predefined syntactic signature matching to identify malware, can be easily evaded by code polymorphism and metamorphism [27,7].

To address this shortcoming, various malware detectors have been recommended in two main analysis domains. The first is static analysis with semantic signatures, and the second is dynamic analysis that can explore the behaviors of malware. Malware detectors using static analysis rely on the semantics of instructions such as behavior template or model checking [8,20,22]. These techniques have proven quite effective in identifying metamorphic malware, and they can usually cover the overall program code faster than dynamic analysis [5]. However, static analysis can be defeated by deliberately crafting malicious programs such that their malicious content is obfuscated and difficult to analyze.

The drawbacks of static analysis techniques have led to the development of dynamic analysis techniques [13], which analyze the behavior of malicious code while it is being executed in an emulated operating system environment. Function call monitoring, function parameter analysis, and information flow tracking are commonly used techniques of a dynamic analyzer [11]. There are many automated tools supporting dynamic analysis such as CAMP [32], ZOZZLE [9], Cuckoo [16], TTAnalyze [5], CWSandbox [43], Anubis [3], Ether [10], and LoGos [19]. However, dynamic analysis is time consuming and resource intensive. Furthermore, adversaries who account for dynamic detection mechanisms have created malware that can be aware of virtual environments and respond to them in various ways to avoid running malicious code, thereby appearing to be a benign program.

As mentioned above, both static analysis and dynamic analysis have some disadvantages. ML-based malware detection methods have thus been proposed to overcome these disadvantages [2,1,6]. ML approaches first learn the features of an executable file and then classify unknown malware samples into trained malware groups or determine if they are outliers. Widely used ML classifiers include Decision Tree, Support Vector Machines (SVM), Random Forest, and so on. In the ML-based detection model, it is important to choose appropriate features and an appropriate classifier model to account for system performance.

Several studies [35,21,28] have used static features such as byte-sequence n-grams, portable executables, string features, opcode n-grams, and function-based features extracted from executable files [36]. Other studies [12,24] have applied behavior-based features that can be obtained by executing the malicious program in a virtual environment.

Their results were later improved upon by Lu et al. [26], Islam et al. [15], and Santos et al. [33], who used both static features and behavior-based features to improve accuracy. Islam et al. [15] presented a classification model based on the combination of two static features and one dynamic feature. The three feature vectors are combined as an integrated feature vector for a single classifier. Lu et al. [26] proposed an ensemble of classifiers (SVM and Association Rule algorithm) with 500 DLL/API function calls for static features and 12 behavior-based features. OPEM [33], which is a hybrid malware detector proposed by Santos et al., utilizes not only sequences of operational codes for static features, but also a large number of dynamic features. To validate the impact of their static and dynamic features, they trained a classifier with three datasets; one with only static features, another with only dynamic features, and the other with both static and dynamic features. They compared the results obtained using four classifiers, namely Decision Tree, K-nearest neighbor, Bayesian network, and SVM, with those obtained using these three types of features. Khasawneh et al. [18] also used static and dynamic features with ensemble classifiers in their malware detection model, but their detection system is supported by hardware and low-level architectural features.

Recently, image-based malware detection systems [14,30,45] have been proposed. However, these detection systems are vulnerable to image pixel attacks [25,37].

Based on the literature review, it is evident that ML techniques contribute to malware analysis, as classifiers can aid in improving performance metrics such as computation time or detection rate. Furthermore, by taking advantage of both static and dynamic analysis information, it is possible to construct a model that extracts the underlying structure of both malicious and benign programs.

In this study, we propose a hybrid model that uses both *random forest*, a supervised learning model, and *multilayer perceptron*, a deep learning model. This hybrid method provides more extensive detection coverage than typical pattern matching, single dynamic models, or ML-based models. In particular, while supervised models cover most currently known malware, these ML models still have difficulties in classifying benign files that exhibit malicious behaviors. In this case, the deep learning module used in our hybrid model provides more precise classification.

3. Feature and classifier selection

In this section, we introduce the features and classifiers that we utilize. Most of these features have been used in prior studies [22,33,17,4]. However, although we use well-known features, the number of features used differs, thereby affecting the detection rate. To choose appropriate classifiers, we experimented using various ML algorithms with our selected static features and dynamic features. We explain this approach below.

3.1. Static features

As listed in Table 1, we categorize the static features into five main classes: size, count, entropy, entry point, and Import Address Table (IAT) information. The static feature comprises 79 features from these five feature classes in total. We also categorize the dynamic features into five feature classes (see Table 2). The details of our features are presented in Table 1.

As represented in Table 1, most features have been used by previous studies, we refine these features to increase the detection rate by manipulating the number of counts and size (i.e., # of API functions), or through the use of entropy information. These attributes exposed unique characteristics in malware beyond what we expected (details in Table 3). Additionally, these features differ and can have even higher results depending on the classification model used (see Table 3).

Our selection of static features include previous malware features such as the size of file sections, as well as the size of parts of specific files parts such as data, text, bss, or header. Particularly, size is the atomic unit most frequently used for feature selection. In addition, we include API, DLL, and section quantities to enrich static features. We also include the entropy of each section, the entry point, and the number of IAT listed. Malware triggers specific API calls, leaving IAT information that can be used to distinguish it from benign files. Nonetheless, malware is often not accurately classified because attackers mimic benign files to appear legitimate. For instance, malicious files obfuscated by packers are homogeneous in terms of their length, number of file sections, or headers, when compared to benign files. These properties often overlap in other malware and are frequently encountered among them. Thus, we added dynamic approaches. For instance, a serial dynamic behavior (file creation, network access, registry access) is considered to be suspicious, and these features are identified from the use of the related API functions. Relevant details are described in the next subsection.

3.2. Dynamic features

Static features face limitations that prevent observation of malicious behaviors. To use behavioral information, dynamic analysis provides a better solution, albeit this process is time consuming. Nonetheless, dynamic feature extraction is more useful in such cases compared to use of static features.

Table 1
Static features.

Feature Class	Features	Description
Size	File (byte)	Size of file
	Headers byte	Size of header
	InitData (byte)	Size of initialized data (.data section)
	UnitializedData (byte)	Size of uninitialized data (.bss section)
	Text (byte)	Size of text section
	Debug (byte)	Size of debug section
	rsrc section (byte)	Size of resource directory
Count	API	Counts of suspicious APIs (by *PEframe*)
	DLL	Counts of accessed DLL
	Section	Counts of file sections
	Number of RVA sizes	Number of data-directory entries
	Language	Number of languages used in resource section
Entropy	entropy_data	Entropy of data section
	entropy_rdata	Entropy of rdata section
	entropy_reloc	Entropy of reloc section
	entropy_text	Entropy of text section
	entropy_rsrc	Entropy of rsrc section
Entry point	entry_point	Entry point collected (by *ExifTool*)
IAT	API functions	61 predefined API functions

Table 2
Dynamic features.

Feature class	Feature	Description
API	File system changes	Predefined file system changes (copy, rename, delete)
	API call	Suspicious API call
	DLL loaded info.	Loaded DLL Information
Location	File changes in suspicious location	# suspicious file change; create, write, rename, delete
	Suspicious DLL location	(Loaded) DLL location, where paths are predefined
	Suspicious registry access	Registry paths often used by malware
	Suspicious directory access	# suspicious directory access
Mutex	Mutex based features	Classified by symbolized mutex
Network	Network (by winsock DLL)	N/W open, outbound access and malicious IP spaces
Registry	Command running	Persistency (specific command running when reboot)
	Suspicious registry	Registry changes (create, read, modify, delete)

To distinguish between benign and malicious properties, we adopted various features. Table 2 shows dynamic features, most of which are independently critical. Among these features, the *file system changes* denote changes to the number of files that we predefined, meaning files are created, copied, or deleted.

In particular, the location-based features relevant to file, DLL, and registry are used, where suspicious paths are predefined. For details, refer to the link[1] below, where we provide file location, registry location, and predefined API functions. If a file accesses the directory or registry (for create, read, copy, write, rename, or delete), we consider the file to be suspicious. Malicious files (or scripts) often use specific locations to install and create new files. Hence, as these locations differ from those of benign files, they are used to distinguish the malicious files. In particular, the *file changes* are considered as containing malicious features. The *file change* feature depends on the directory information, where malware is often found. This feature helps determine whether a given file is malicious.

Other API, network, and mutex-based features are often used in other studies [33,17] as malicious properties. API functions were extracted from each value of the API functions in the DLLs that malware frequently accesses. Hence, these features may be identified as symptoms of a suspicious trace. Although dynamic features are often used in previous studies, we redefine these features as manipulating the number, counts, or via selective choice in location information and feature selection.

3.3. Classifiers

3.3.1. Test with typical dataset

To choose appropriate classifiers, we tested various ML algorithms with our selected static features and dynamic features.

[1] https://drive.google.com/file/d/1y279BbaKm44NVqKsuV7qHsk-K0HW-VjE/view

Table 3
Static/dynamic feature-based detection rate by 10-fold cross validation. We used the default options in Weka for all classifiers in this experiment and the results were based on weighted average. SD denotes standard deviation. The test was repeated three times to calculate SD values, and the average of three calculations were used for each performance metric. Moreover, identical training datasets were used. The precision was calculated using the formula, Precision = TP/(TP+FP). Static features were extracted from `PEframe`. Dynamic features were extracted from `Cuckoo`.

Classifier	Static feature-based				Dynamic feature-based			
	TP	FP	Precision	ROC	TP	FP	Precision	ROC
Random Forest	**0.946**	**0.070**	0.946	**0.988**	**0.984**	**0.003**	0.987	**0.995**
SD	1.216E−16	0.00082	1.216E−16	1.216E−16	0	4.751E−19	0	0
AdaBoostM1	0.831	0.245	0.832	0.879	0.965	0.003	0.986	0.994
SD	0.00041	3.040E−17	0.00041	0.00041	0.04164	4.751E−19	1.216E−16	1.216E−16
RealAdaBoost	0.847	0.200	0.846	0.913	0.974	0.004	0.981	0.984
SD	0.00041	3.040E−17	0.00041	1.216E−16	1.216E−16	0	0	0
SVM	0.662	0.601	0.779	0.530	0.981	0.003	0.985	0.989
SD	0	0.00041	0	0.00041	0	4.751E−19	0	0
Logistic	0.840	0.219	0.839	0.904	0.973	0.013	0.980	0.984
SD	0.00041	0.00082	0.00041	0	0.00531	0.02123	0.00367	0.00204
Naive Bayes	0.486	0.316	0.703	0.750	0.856	0.011	0.953	0.975
SD	0.00367	0.00082	0.00122	0.00082	0	1.900E−18	0	0
K-Means	0.836	0.291	–	–	0.807	0.722	–	–
SD	0	0	–	–	0	0	–	–
MLP	0.879	0.151	0.878	0.939	0.976	0.004	0.983	0.993
SD	0.00163	0.00082	0.00163	0.00163	0.00653	0.00041	0.00327	0
LDA	0.833	0.235	0.833	0.890	0.945	0.004	0.969	0.972
SD	0.00163	0.00286	0.00122	0.00122	0	0	0	1.216E−16
FLDA	0.807	0.213	0.810	0.890	0.908	0.007	0.960	0.969
SD	0.00163	0.00204	0.00163	0.00122	0	0	0	0

To demonstrate the best classifier, we collected datasets (from Jan. 2016 to Nov. 2018) from VirusChaser (a third-party antivirus tool,https://www.viruschaser.com/). Out of the 149,859 samples used to evaluate our feature-based model, 139,384 were used as malware samples, and the remaining samples (10,475) were used as benign samples. All of these samples were labeled by `VirusTotal` [38].

The result was evaluated using `Weka 3.9.1` [41] to produce optimal classifiers. We applied attribute values of five feature classes in the static analysis and five feature classes in the dynamic analysis. We then performed 10-fold cross-validations to avoid sample bias. We ran these evaluations six times to obtain average values; the results were typical. The ML classifiers used for comparison were *Random Forest*, *SVMs*, *Logistic*, *Naive Bayes*, and *K-Means*.

Table 3 indicates that, for the experiment using static features, *Random Forest* has a true positive (TP) rate of 94.6% and a false positive (FP) rate of 7.0%. In this model, bagging is applied using 100 iterations and the base learner. In the experiment using dynamic features, *Random Forest* showed a high TP rate at 98.4% and a low FP rate at 0.3%. The deep learning model *MLP* had a detection rate of approximately 97.6% in TP rate and 0.4% in FP rate.

A receiver operating characteristic (ROC) curve is a graphical plot of sensitivity against specificity. It is used here to represent the plot of the TP fraction versus the FP fraction. The point (0, 1) is the perfect classifier because it classifies all positive and negative cases. *Random Forest* and *MLP* showed a high ROC area.

For static features, `GainRatioAttributeEval` [42] evaluates the worth of an attribute by measuring the gain ratio with respect to the feature class.

$$GainR(Class, Attribute) = (H(Class) - H(Class|Attribute))/H(Attribute),$$

where H represents the entropy. The highest ranked attributes from Gain Ratio, sequentially, are API functions in IAT such as *_C_specific_handler*,*RtlLookupFunctionEntry*, size-related features such as *FileSize*, entropy-related features such as *entropy_reloc*, count-related features such as *APICount*, and entry point, respectively. The attribute value range was [0.0046, 0.95754]. From this test, we assume that API functions and size-related features have a considerably greater effect on the detection rate.

The standard deviation (SD) values listed in Table 3 indicate the robustness of a given model in view of the dataset and features used. The performance results of statistical models might differ slightly during multiple repetitions of the experiment under identical operating conditions. Therefore, most classifiers possess low SD values for typical datasets. However, it is interesting to note that SD values obtained using Random Forest with dynamic features are slightly lower than those obtained using static features. In contrast, for atypical datasets (Table 4), SD values obtained using Random Forest with dynamic features exceed those obtained using Random Forest with static features. Therefore, use of both static and dynamic features enhances the robustness of the proposed detection model irrespective of the type of data processed.

Table 4
Static/dynamic feature-based detection rate in the atypical dataset. In general, precision indicates the relative closeness of multiple measurements in a grouped series. Accuracy indicates the closeness of measurements to the accepted value it is given as (TP+TN) / (TP+TN+FP+FN).

Random Forest	Static feature				Dynamic feature			
	TP	FP	Accuracy	Precision	TP	FP	Accuracy	Precision
Average	0.977	0.946	0.655	0.658	0.914	0.532	0.758	0.762
SD	0.00289	0.00406	0.00113	0.00059	0.02109	0.04258	0.01334	0.01274

Table 5
Optimizer results.

Optimizer	Loop	$MLP_{dynamic}$		MLP_{static}	
		Range (%)	Average (%)	Range (%)	Average (%)
SGD	32	88.2–94.3	93.064	35.9–64.1	51.361
RMSprop	32	93.1–96.0	95.176	35.9–64.2	62.367
Adagrad	32	94.4–95.7	95.101	35.9–64.4	60.408
Adadelta	32	93.6–95.6	94.892	35.9–71.6	62.849
Adam	32	94.2–95.9	95.008	35.9–75.9	64.115
Adamax	32	94.2–95.7	95.127	35.9–73.6	65.354
Nadam	32	90.3–95.6	94.511	35.9–64.1	59.506

3.3.2. Evaluation with irregular dataset

The Korea Internet & Security Agency (KISA)[2] is a government body that develops policies to promote a safe internet environment. They provide challenging datasets for research purposes that contain benign and malicious executables (https://www.kisis.or.kr/kisis/subIndex/283.do). This R&D dataset contains very unusual data of benign files that show malicious behaviors as well as sets that include malware that can be overlooked by antivirus software. We used 4,160 labeled malware samples and 2,235 benign samples from *KISA-CISC2017-Malware-2nd.zip* to evaluate our *Random Forest* classifier. We repeated this test three times and used the average results obtained.

The result of FP in *Random Forest*, as presented in Table 4, shows that the detection rate is influenced by the type of dataset. The FP rate in *Random Forest* increased to 94.6% and 53.2% in static and dynamic-feature based tests, respectively. These high FP rates were caused by benign files that exhibited malicious behaviors. The TP rate with an identical dataset slightly decreased to 97.7% and 91.4%, as listed in Table 4. *Random Forest* using static features classified the incorrect instances of 2,114 out of 2,235 benign samples. As a result, we aimed to design a hybrid model that could increase the overall classification performance using various types of file samples. The approach of this model is explained in the following section.

4. System design

In this section, we introduce a hybrid model to overcome the limitations of the atypical dataset in Section 3.3.2 for supporting a comprehensive analysis of malware and atypical benign files. In the setup of the hybrid model, we use three approaches to improve the detection rate. The first approach is the selection of well classified features. The second is to use optimized classifiers (see Section 3). Lastly, we develop a voting method to increase detection performance. In this approach, we use classifiers with high TP rates to augment the effects of voting. We also use training sets selected from datasets of VirusChaser and KISA.

4.1. Model setup

We adopted an ML-based hybrid model to develop the most feasible method. The overall system architecture is shown in Fig. 1. We split the detection procedure into three steps, using *Random Forest* (RF) and *Multi-Layer Perceptron* (MLP), as shown in Fig. 1.

Our model measures the degree of maliciousness via four classification models. The typical ML algorithm uses a scoring range of [0,1] to express the degree of maliciousness with original files. By contrast, this proposed model uses four scoring results via respective static and dynamic results of *RF* and *MLP*, and generates optimization results by applying our voting model (detailed in Section 4.4) on the results from four scores of the hybrid model, namely RF_{static}, $RF_{dynamic}$, MLP_{static}, and $MLP_{dynamic}$.

[2] https://www.kisa.or.kr/main.jsp.

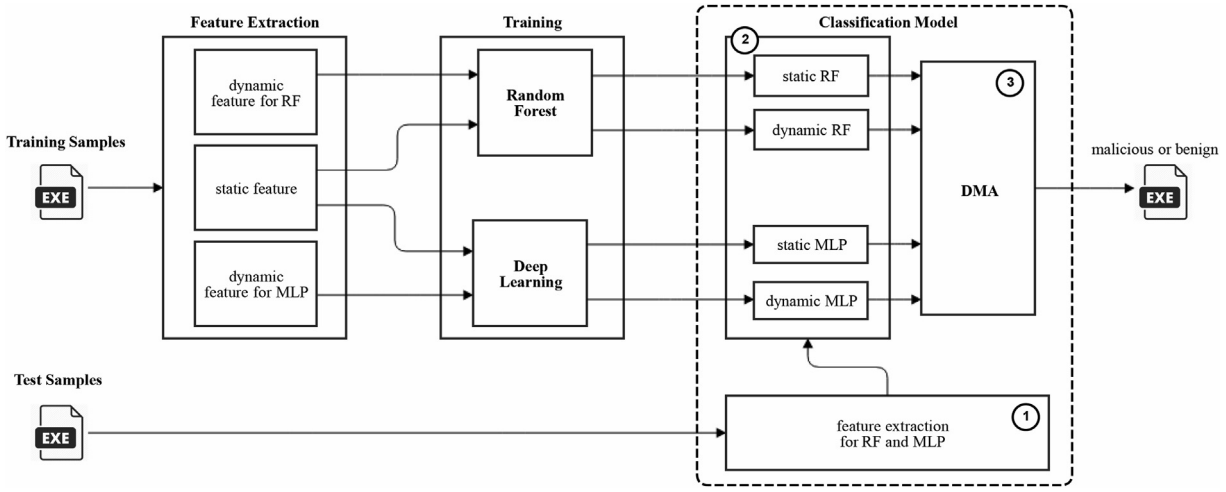

Fig. 1. Overview of the proposed model.

First, the feature extraction stage ① extracts the respective static and dynamic features from candidate files. Then, based on four feature sets, the classification stage ② generates results via four classification models. Finally, from the results, the proposed model in the decision stage ③ searches the final decision values using a rule-based majority voting scheme.[3]

While a simple majority voting rule that selects alternatives having a majority can detect malware well, it is highly likely to ignore the classifier that would correctly detect benign files with malicious behavior. For example, suppose there is a benign file sample with features similar to malware. If three of the four models misclassify the sample as malware, and only one model accurately classifies the sample as benign, then this benign sample is assigned to the incorrect category by simple majority voting rule. Therefore, we concentrate on the benefits of minority that lead to decreases in false alarms, and propose the rule-based majority voting scheme. In the rule-based majority voting scheme, the sample is preferentially labeled according to the result of classifiers that detect the benign file well. We add a priority rule (see Table 6) to find an effective classifier for detecting benign files. Samples not categorized by priority rule are then classified by the simple majority voting rule. *DMA* in ③ expresses the *decision making algorithm*, which is detailed in Section 4.4. This concept relies on the probabilistic diversity of malicious tendencies.

4.2. Preprocessing

For data preprocessing, we follow several steps. First, when a file is uploaded, the model performs a code review for static analysis, and then executes dynamic analysis for the file. Then, the model collects various static analysis logs such as entropy, active directory entries, and size of headers. Initially, the model collects several hundred static features. The information is gathered using this format.

For instance, "ActiveDirectoryEntries": ["IMAGE_DIRECTORY_ENTRY_EXPORT", "IMAGE_DIRECTORY_ENTRY_IMPORT", "IMAGE_DIRECTORY_ENTRY_RESOURCE", "IMAGE_DIRECTORY_ENTRY_DEBUG", "IMAGE_DIRECTORY_ENTRY_IAT"]. Subsequently, this model reduces the number of features until the number of features we use (i.e., 79 features for static analysis), including registry, network, core API functions, and critical commands that malware often uses. In this processing, this model changes the feature values to integers even though they contain characters. We do not use symbolic data in this model, although we do count them. For instance, we count the number of accessed directory names, or the number of accessed API functions.

This model selectively collects features to be used only in this model among collected results. For instance, these 79 feature values for static models are numeric numbers such as "feature": ["0", "0", "42929", "382206", "0", "5", "0", "0", "0", "0", "0", "5", "67584", "0", "1024", "21504", . . ."0", "0", "0"]. Then, we predict the suspiciousness of the file based on these feature values with trained models. Examples of the preprocessing can be viewed via the provided link.[4]

In this preprocessing, we do not normalize data into specific intervals. We know that normalizing can have a negative effect on the sensitivity of units when the features are meaningful. In this case, the distance between feature values makes a difference. There is a large scale of malware. In particular, attackers attempting to neutralize certain properties of malware can make them appear to be benign features. The difference that could be used to distinguish them has been diminished.

[3] The most widespread decision fusion rule used to make conclusions from multiple classifiers is majority voting; it is a method in which a class is chosen if it is voted for by the most classifiers. In majority voting, each classifier gets one vote, regardless of its detection accuracy level. To solve this problem, we can set a weight for classifiers' votes according to the accuracy of the result. Weighted voting was used in this study to determine the final decision.

[4] https://drive.google.com/file/d/1T2EFarGa-OnbcQ_w-Yo5VdoimkXB-ovU/view.

Table 6
Rules of the decision maker, where RF denotes Random Forest and MLP is the deep learning classifier. In this rule set, RF_{static} denotes the malicious probability value of an executable as predicted by a *RandomForest* classifier to which static features are applied.

Rule 1	If $RF_{static} \leqslant 0.5$, then return "benign".
Rule 2	If $RF_{dynamic} < 0.5$, then return "benign"
Rule 3	If $RF_{static} < 0.5$ or $RF_{dynamic} < 0.5$, then return "benign"
Rule 4	If $RF_{dynamic} < 0.5$ or $MLP_{dynamic} < 0.5$, then return "benign"

However, some features are still valuable, such as directories that attackers often use to download malware, or the counts of suspicious API calls. The units of these features are meaningful. We think that the differently-scaled features contribute significantly to classification. Hence, we do not perform normalization into specific intervals (i.e., range [0.,1]).

4.3. Deep learning model optimization

We recognize that, in practice, it is more difficult to classify benign files that show malicious symptoms. General single ML-based models remain inadequate in this approach because it is difficult for a single model to satisfy both a high TP and a low FP at the same time. To increase the detection accuracy for benign files, we employ a deep learning model, as it provides in-depth classification. We performed the measurement of respective detection rates in various ML models (see Table 3), and chose a classification model for malware detection via *random forest* and *MLP*. We particularly focused on optimization of a deep learning model to increase detection rate. To address these issues, we investigated optimized values that strongly affect the detection rate. For deep learning model optimization, parameters were set up as follows. We initially set the default initial values. For example, the node was 250, an epoch was 512, the batch size was 2,048, the hidden layer was 16, the optimizer was sgd, and there was no dropout. Then, we changed the value from low to high to find the optimal value for the first parameter, "node". After finding the optimal value of the node, we experimented to find the optimal value of the next parameter, epoch, after reflecting the optimal value to the node. The performance test to find the optimal value of each parameter was applied sequentially to all hyperparameters. However, random weight, activation function (RELU for hidden layer and softmax for output layer), and cross-entropy as cost function were used as default without modification. Through experimentation, optimized arguments for nodes, epochs, batch size, and hidden layers are as shown in next section. In this experiment, we used samples from *KISA*.

4.3.1. Node

Fig. 2 (left) shows the change in the detection rate in $MLP_{dynamic}$ as the number of nodes increases. As shown by this figure, the detection rate was the highest when the model had 205 nodes, while the lowest detection rate occurred with three nodes. To verify this experiment, we repeatedly tested the detection rate for each node between 1 and 256. The general range of the detection rate with MLP was 83.7–96.1%. However, MLP_{static} showed a stable tendency in the range between 20–38 nodes.

Although MLP_{static} exhibited relatively weak performance, it is still valuable as a component of a hybrid classification system, because this classifier contributes one vote to the final decision by the majority voting rule.

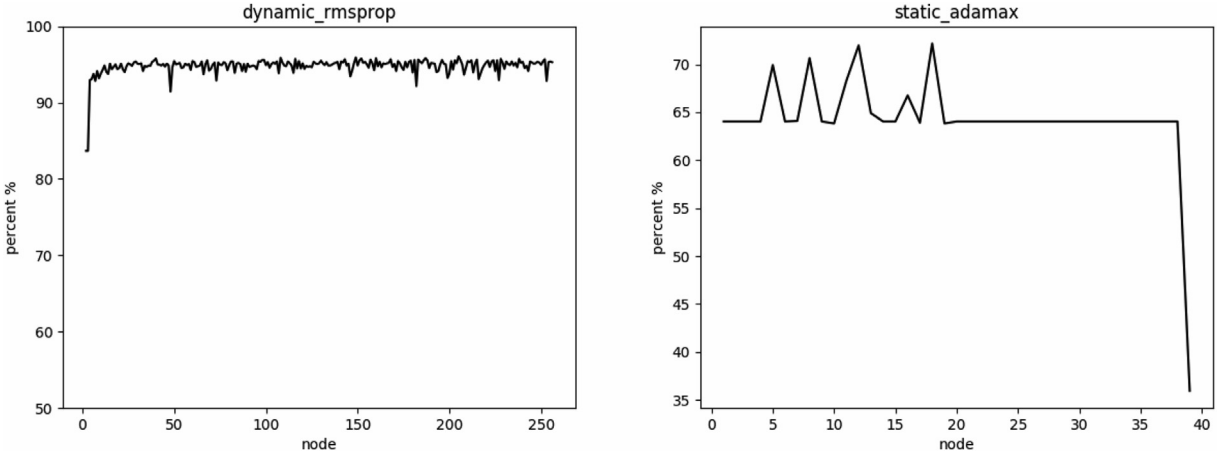

Fig. 2. Detection rate comparison according to node increases, performed 256 times (left) and 39 times (right) each.

4.3.2. Epoch

The highest detection rate using epochs was at an epoch count of 364, which exhibited a detection rate of 96.004%. When the number of epochs was increased, this experiment showed increasing instability. The results of our experiment exhibited increasing unstable tendency along with the increase of epoch, as shown in Fig. 3 (left). In fact, the detection rate did not depend on the increased number of epochs. This tendency was also exhibited in MLP_{static} of Fig. 3 (right).

4.3.3. Batch size

Batch sizes higher than 200 generally presented higher detection rates, with the highest detection rate being 95.8%. The lowest detection rate was roughly 46.7%, at a batch size of 176 in $MLP_{dynamic}$. The MLP_{static} of Fig. 4 (right) illustrates the results of an additional experiment showing that a higher batch size provides greater instability and low detection rates, as stability becomes increasingly unstable as batch sizes increase.

4.3.4. Hidden layer

In determining an optimized hidden layer count, our experiment (see Fig. 5) showed the highest detection rate at 12 hidden layers with an average detection rate of 95.002%. We repeated this test 16 times on each hidden layer count between 1–16.

4.3.5. Optimizer

We performed additional verification of the optimizer used in a deep-learning model. This optimizer selection is an important factor for increasing the detection rate. As shown in the experimental results listed in Table 5, the highest detection rates for $MLP_{dynamic}$ and MLP_{static} resulted from *RMSProp* and *Adamax*, respectively.

4.3.6. Dropout

The general ML-based classifier is susceptible to overfitting. In particular, benign files with malicious behaviors are critical in ensuring proper classification. We applied dropout in an attempt to overcome false positives and overfitting, but the results showed a high fluctuation range. The deep learning model is sensitive to our features, and the result shows that the dropout simultaneously expressed both high and low detection rates in some places. Thus, we assumed that the drastically fluctuating detection rate resulted from the instability caused by dropout with small node amounts. Consequently, we inserted two dropout layers in the proposed model with average rates of 0.5 to avoid overfitting.

4.3.7. Random weight

In applying the intervals for random weight, we configured a random weight using a random uniform interval [−0.05,0.05] in the first hidden layer, and after the first layer, we used a Glorot uniform initializer called the Xavier uniform initializer. It draws samples from a uniform distribution within [−limit, limit], where the limit is sqrt(6 / (fan_in + fan_out)). In this formula, fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor. Additionally, the bias was initialized to zero, which is the default value in a keras.

4.3.8. Activation functions

AI-HydRa used the rectified linear unit (ReLU [23,29]) as activation functions in hidden layers; ReLU is a simple non-linear function f(x) = max(x, 0). The softmax activation function was used in the output layer. In the back-propagation step, we used RMSProp and Adamax optimizer for the dynamic MLP model and static MLP model, respectively.

4.3.9. Other parameters

In the proposed model, we used the default values of our optimizer, RMSprop. The default value of *keras.optimizers.RMSprop* is coded as follows: *lr* = 0.001, *rho* = 0.9, *epsilon* = None, *decay* = 0.0, where *lr* denotes learning rate with float >= 0. *rho* is also float, >= 0, and *RMSprop* denotes the decay factor, corresponding to fraction of the gradient to be retained at each time step. *Epsilon* is the Fuzz factor and the float >= 0. If *epsilon* = None, it defaults to *K.epsilon*(). Decay is the learning rate decay over each update with float >= 0. The bias is initialized to zero, the default setting of the keras "kernel_initalizer".

Through these experiments, our $MLP_{dynamic}$ used 39 nodes, 1072 batch sizes, 12 hidden layers, 87 epochs, *RMSprop* as the optimizer, and 432 features. MLP_{static} used 18 nodes, 1520 batch sizes, seven hidden layers, 103 epochs, *adamax* as the optimizer, and 79 features. Both classification models adopted *ReLu* and *softmax* as the activation function. In particular, to avoid the learning slowdown caused by the (z) term, we used a cross-entropy cost function instead of the mean squared error (MSE) cost function.

4.4. Decision making algorithm

This system basically uses a majority vote to execute decision making based on results from four models: RF and MLP with static features, as well as RF and MLP with dynamic features. In this experiment, we found that single classifier RF shows high performance in malware detection, but with a high FP rate for benign executables. To reduce this FP rate, we propose additional voting rules as shown in Table 6.

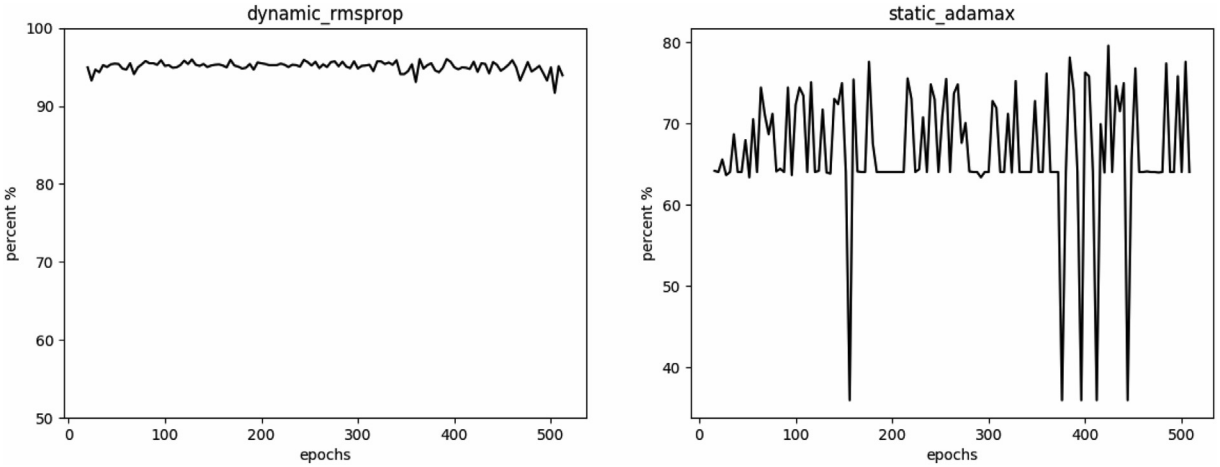

Fig. 3. Graph of detection rates with increasing epoch values. The detection rate was unstable even when higher epoch values were applied. The *X*-axis denotes the number of trials and the *Y*-axis denotes the detection rate.

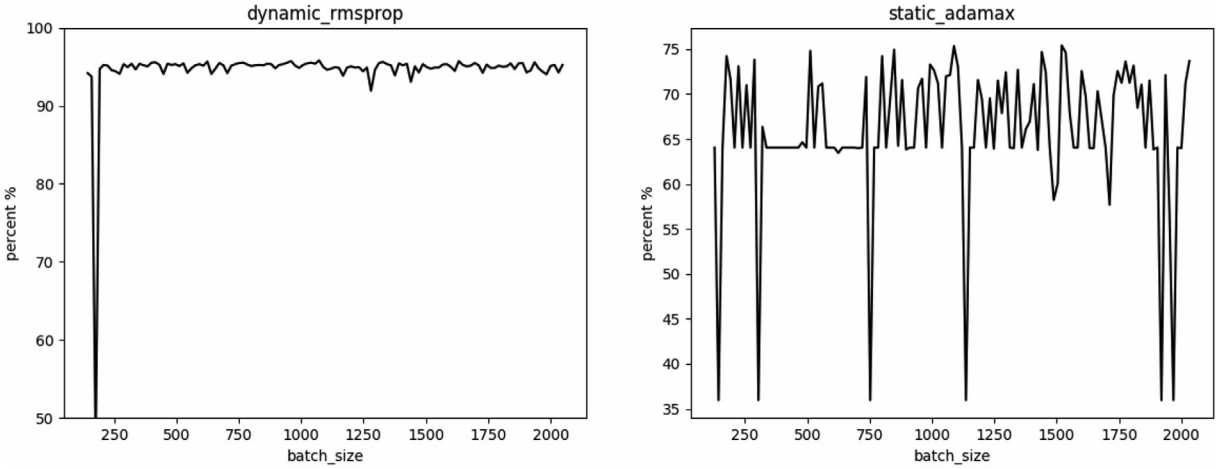

Fig. 4. Graph of detection rates with increasing batch sizes. Detection rates became more stable when the batch size was more than 200 in $MLP_{dynamic}$. In contrast, MLP_{static} was relatively unstable.

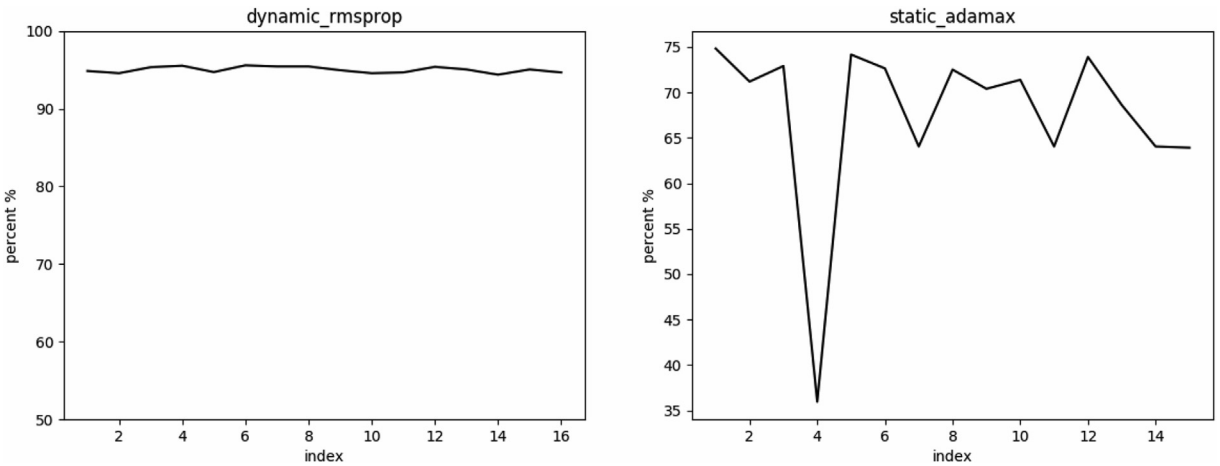

Fig. 5. Optimizing test on the number of hidden layer.

Table 7
Rule combination.

Index	Rule combination	Index	Rule combination
1	majority voting (MV)	9	MV + Rule 2,3
2	MV + Rule 1	10	MV + Rule 2,4
3	MV + Rule 2	11	MV + Rule 3,4
4	MV + Rule 3	12	MV + Rule 1,2,3
5	MV + Rule 4	13	MV + Rule 1,2,4
6	MV + Rule 1,2	14	MV + Rule 1,3,4
7	MV + Rule 1,3	15	MV + Rule 2,3,4
8	MV + Rule 1,4	16	MV + Rule 1,2,3,4

The rules in Table 6 can be described as 16 possible cases (listed in Table 7) according to the majority vote outcome and combinations of the four decision rules. The results are reflected in Table 8.

Previous studies have used a majority vote for distinguishing between malware and benign files. However, the prediction values of TP and FP are produced differently according to classifiers. Some classifiers show a high TP rate while simultaneously showing a high FP rate. On the other hand, some classifiers show a low rate of both TP and FP. Due to limitations of individual classifiers, it is not practical to pursue building a single optimized classifier with both a high TP rate and a low FP rate. Hence, we utilize a hybrid model that employs several classifiers that are optimized for either a low FP rate or a high TP rate. To overcome FP issues, this proposed model makes a final decision via rule-based majority vote based on RF_{static}, $RF_{dynamic}$, MLP_{static}, and $MLP_{dynamic}$, reflecting the optimized properties of every classifier.

If a given model only depends on the majority vote, classifiers that have difficulties in benign detection generally show a high FP rate. In general, benign programs that exhibit malicious behaviors can be determined to be malicious, which increases the overall FP rate. Thus, in this paper, we propose four decision rules to perform benign classification well via classifiers with a low FP rate.

Although we use four classification models, we did not add a decision rule of MLP_{static} as shown in Table 6 because, although it provides a high TP rate, it simultaneously produces a high FP rate. In Rule 1, RF_{static} values frequently show 0.5 probability. Thus, we include RF_{static}=0.5 to observe the effect on FP rate. Our rule-based majority model prefers rule policies, and subsequently applies a majority rule. In the files with no majority element, we evaluated which one is more validated when the model gives a result of benign or malicious. Consequently, it affected 2% more when this model gave a decision as malicious, as shown in Table 8.

Algorithm 1: Decision making algorithm for detection with high accuracy. Rule 1 and Rule 4 are applied.

```
Rule-based Majority Vote (RF_static, RF_dynamic, MLP_static, MLP_dynamic);
Input  : Non-negative benign floats RF_static, RF_dynamic, and MLP_dynamic from
         prediction results of the classification model. They are constant in the
         range [0,1]
Output: benign or malicious
if (RF_static <) or (RF_dynamic ≤ 0.5 or MLP_dynamic ≤ 0.5) then
    return benign;
else
    return MajorityVote(RF_static, RF_dynamic, MLP_static, MLP_dynamic);
```

Although all single classifiers are optimized, this does not mean that the ensemble of classifiers is optimized. Thus, to make a final decision, we use an optimal decision-making algorithm. We provide an example of the prediction results for which this decision rule is applied to the unusual dataset in Table 8. In this approach, we found that voting with Rule 1 and Rule 4 resulted in the best performance for TP and FP rates. Namely, RF_{static} in Rule 1, and $RF_{dynamic}$ and $MLP_{dynamic}$ in Rule 4 are the best classifiers in our model in terms of a high TP rate and a low FP rate.

The TP rate via rule combination was more than 90% in most cases. Malware is detected best, with the highest TP rate, when majority voting is applied to the proposed system. There are two reasons for this. Firstly, both the RF and MLP classifiers perform malware detection well, with features that accurately characterize malware. Therefore, there is a high probability that three or more classifiers are classified as malware on actual malicious files. Secondly, the added decision rules all return lower FP results. This means that benign programs can be categorized well, but malware is relatively less detected. However, because Rules 1, 2, and 3 do not significantly affect the TP rate, we can confirm that the RF classifier is effective for detection of malware rather than detection of benign programs.

Table 8
Results based on our rule-based majority vote.

		Broad dataset				Narrow dataset			
	Voting type	Accuracy	TPrate	FPrate	Precision	Accuracy	TPrate	FPrate	Precision
Case 1	Majority Vote	**0.968**	0.96	0.023	0.976	0.383	0.96	0.885	0.335
	MV+1	0.964	0.952	0.023	0.975	0.449	0.952	0.784	0.361
	MV+2	0.968	0.96	0.023	0.976	0.386	0.96	0.881	0.336
	MV+3	0.968	0.96	0.023	0.976	0.407	0.96	0.849	0.344
	MV+4	0.945	0.904	0.016	0.983	0.48	0.904	0.717	0.369
	MV+1,2	0.964	0.952	0.023	0.975	0.452	0.952	0.781	0.362
	MV+1,3	0.964	0.952	0.023	0.975	0.452	0.952	0.781	0.362
	MV+1,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399
	MV+2,3	0.968	0.96	0.023	0.976	0.407	0.96	0.849	0.344
	MV+2,4	0.945	0.904	0.016	0.983	0.48	0.904	0.717	0.369
	MV+3,4	0.945	0.904	0.016	0.983	0.501	0.904	0.686	0.38
	MV+1,2,3	0.964	0.952	0.023	0.975	0.452	0.952	0.781	0.362
	MV+1,2,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399
	MV+1,3,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399
	MV+2,3,4	0.945	0.904	0.016	0.983	0.501	0.904	0.686	0.38
	MV+1,2,3,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399
Case 2	Majority Vote	**0.988**	1	0.023	0.977	**0.396**	1	**0.885**	**0.344**
	MV+1	0.508	0.988	0.961	0.501	0.406	0.988	0.864	0.347
	MV+2	0.968	0.96	0.023	0.976	0.381	0.96	0.889	0.334
	MV+3	0.968	0.96	0.023	0.976	0.407	0.96	0.849	0.344
	MV+4	0.945	0.904	0.016	0.983	0.48	0.904	0.718	0.369
	MV+1,2	0.964	0.952	0.023	0.975	0.452	0.952	0.781	0.362
	MV+1,3	0.964	0.952	0.023	0.975	0.452	0.952	0.781	0.362
	MV+1,4	0.941	0.896	0.016	0.982	**0.538**	0.896	**0.628**	**0.399**
	MV+2,3	0.968	0.96	0.023	0.976	0.407	0.96	0.849	0.344
	MV+2,4	0.945	0.904	0.016	0.983	0.48	0.904	0.718	0.369
	MV+3,4	0.945	0.904	0.016	0.983	0.501	0.904	0.686	0.38
	MV+1,2,3	0.964	0.952	0.023	0.975	0.452	0.952	0.781	0.362
	MV+1,2,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399
	MV+1,3,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399
	MV+2,3,4	0.945	0.904	0.016	0.983	0.501	0.904	0.686	0.38
	MV+1,2,3,4	0.941	0.896	0.016	0.982	0.538	0.896	0.628	0.399

Case 1: If there is no majority element, then the system returns "benign"
Case 2: If there is no majority element, then the system returns "malicious"
Broad dataset: this dataset consists of usual benign programs and malicious programs Narrow dataset: this dataset consists of unusual benign programs (with characteristics similar to those of malware) and malicious programs.

On the other hand, the false positive rate can decrease according to the addition of decision rules. The lowest FP rate is achieved when Rule 1 and Rule 4 are applied, while the FP reduction effect of Rule 2 is insufficient. Therefore, we can say that the $MLP_{dynamic}$ model classifies benign programs well.

The decision rule was able to confirm to a greater effect when applied to a dataset of unusual but benign programs (narrow dataset). In other words, we can confirm that it is effective in detecting benign programs that exhibit behaviors similar to malware. When all decision rules are applied, the FP rate is reduced by about 26% (from 88.5% to 62.8% in Case 2 of Table 8).

If the majority element does not exist, and the decision rule is not applied, it is better to judge a target as malicious because the accuracy is 2% higher than benign. When the system follows the decision rule, there is a potential problem we should consider.

Notably, when only decision Rule 1 is used against a broad dataset, it results in a high FP rate. Therefore, the RF_{static} model should be used in combination with other classifiers rather than being used alone for benign program classification.

Consequently, to increase the TP rate, majority vote is preferable. However, if accuracy is preferred, a rule-based majority vote should be used. This choice is at the developers' discretion.

4.5. Implementation

This model consists of 445 lines of code in total. Further, our code includes 150 lines of the *keras* (version 2.2.2) library for MLP, 45 lines of Random Forest-based classification, and 250 lines of configuration-related code using Python 2.7.12 on Ubuntu 16.04.4 LTS. The training dataset used for *Random Forest* contains 22,004 static records and 7,513 dynamic records, while that used for MLP contains 15,139 static records and 15,353 dynamic records, as shown in Tables 9 and 10. Our model is the de facto model. This proposed model uses a small training dataset for detecting a large-scale executable.

5. Evaluation

This section presents a demonstration of the proposed classification model. In total, 6,395 samples were evaluated using our ML-based approach. In this evaluation, we conducted two experiments: a detection rate test compared to antivirus products, and the result of voting method for building a model. These datasets were also used to investigate the performance.

5.1. Datasets and experimental setup

For experiments with our optimized model, we collected 6,395 samples from KISA. We compared the detection rates of commercial antivirus products and our model. This dataset is entirely different from general datasets, as benign files are commonly used application files, but they show malicious behaviors.

We ran the experiments on Linux inside VirtualBox with an Intel XeonCore Silver 4116 CPU 2.10 GHz with 48 cores, 128 GB of memory, and an Nvidia Titan V. We used six Virtualbox instances on Windows 7 Ultimate 32-bit. The VM images were configured to be run for 360 s/file for dynamic analysis at most.

5.2. Detection rate and voting results

In this evaluation with 6,395 (4,160 malicious and 2,235 benign) labeled files, this model showed both a high detection rate and a low rate of false positives.

Table 11 indicates that our model performs well at detecting various malware instances. Likewise, this provides a TP rate of 85.7% and an FP rate of 16.1% with a performance of 60.9 s per file, including the time for static/dynamic feature extraction. This model is comparable to that of current commercial products. During the real test, the model detected most malicious and benign files. Consequently, the hybrid model with our voting approach is effective in detecting malware and benign files.

5.3. Performance

In general, MLP and SVM show a high trade-off in training and testing. The Random Forest classifier took a total of 17.06 seconds to train and predict 149,859 instances, while MLP and SVM took 882.77 seconds and 866.55 seconds, respectively. Thus, SVM and MLP have high computational complexity. In contrast, *RVFL* [31,34] and *SCNs* [40,39] show reasonable performance. In this study, we evaluated the difference between these models. The results are presented in Table 13.

In this evaluation, RVFL and SCNs exhibited a high performance; however, their detection rates were lower than that of AI-HydRa, as shown in Table 12 and 13. RVFL was faster than SCN, but SCN exhibited higher accuracy than RVFL. As a result, RVFL and SCNs were found to have potential for implementation in a real-time detection system. In particular, if these models apply our approaches, such as the proposed voting system, we believe that the accuracy can be increased even further. In this case, both models may be useful in online and mobile systems. Thus, in the future, we need to conduct further investigations on these models to draw reliable conclusions on their use in actual systems.

6. Discussion

This model requires heavy computation to extract various features from the content. In general, a feature-based approach provides high performance for cases with small feature classes; however, for cases with a large number of features, this approach results in performance delays because a considerable time is required for feature extraction. We utilize a variety of features, and dynamic features, in particular, require considerable time for processing. The dynamic feature extraction uses a high-interaction analysis in separated environments. As such, the model makes frequent tradeoffs while monitoring behavior. Consequently, this dynamic feature selection is a critical aspect that determines a high accuracy rate; however, heavy loads result in low performance. Our dynamic analysis system may be vulnerable to anti-VM attacks from malware and can be stopped without any actions. Then, we use static models. This static model exhibited high detection rates on general files, in particular, for experiments with almost 15 M samples.

In this study, we focused on improving the detection rate by using hybrid classifiers and voting, rather than improving the performance by using well-defined features. Nonetheless, we need to apply an advanced detection rate in the future because our detection rate is unsatisfactory on narrow datasets. Further studies must be conducted to improve the detection rate. In this approach, we can apply the difference between static features and dynamic features in terms of the # of API functions. Adversaries hide API functions in IAT tables of malware, and static analysis is limited. Thus, we can examine the numbers of API functions accessed from malware files created during the run time. In particular, malware with anti-VM does not run in a virtual machine, so the difference between the static and dynamic analysis can be effective.

In addition, we found that one of the most critical features for classifying benign executables that have malicious behaviors is based on sign codes. For instance, sign codes provide critical information for classification. In our datasets, ‘‘None,’’ ‘‘Microsoft Corporation. All right reserved.,’’ ‘‘(C),’’ and ‘‘ ’’ comprised 84.2% in total. Among them, ‘‘None’’ was 74.3%. In `ProductName`, ‘‘None’’ was 75.8% and ‘‘None,’’ ‘‘Microsoft Windows Operating System,’’ and ‘‘ ’’ were 81.5% in total. The features

Table 9
Number of features.

classifier		trainset	# feature	# train sample
RF	static	malicious	79	15279
		benign	79	6725
	dynamic	malicious	513	6492
		benign	513	1021
MLP	static	malicious	79	9686
		benign	79	5453
	dynamic	malicious	432	12425
		benign	432	2928

Table 10
Number of samples in the dataset used for verifying our model. We split the dataset into three groups in advance: training dataset, validation dataset, and test dataset.

Classifier	Dataset Type	Total	Benign	Malicious
Static RF	Train	22,004	6,725	15,279
	Validation	–	–	–
	Test	6,395	2,235	4,160
Dynamic RF	Train	7,513	1,021	6,492
	Validation	–	–	–
	Test	6,395	2,235	4,160
Static MLP	Train	15,139	5,453	9,686
	Validation	1,514	545	969
	Test	6,395	2,235	4,160
Dynamic MLP	Train	15,353	2,928	12,425
	Validation	2,078	340	1,738
	Test	6,395	2,235	4,160

Table 11
Comparison of detection between a hybrid model and antivirus products. *AI − Hydra* decides in favor of benign in the case of majority votes that are applied using rules (MV+1,2,3,4 applied). Otherwise, this model decides in favor of malicious. In this experiment, AI-HydRa showed an average detection rate of 0.851 from three repeated tests, and its standard deviation was 0.00588.

Classifier	TP	FP	Accuracy	Precision
BitDefender	1091	0	0.520	1
ClamAV	2537	9	0.745	0.996
Sophos	2553	0	0.749	1
AI − HydRa	3564	360	0.851	0.908

Table 12
Performance test results of RVFL, SCN, and AI-HydRa. The time is based on # of samples analyzed by each model in Table 10. Model in the first column indicates the learning model, and dataset represents our four feature datasets. For instance, the RF static set is the static feature dataset extracted for RF in our model.

Dataset \ Model	RVFL		SCN		AI-HydRa	
	Accuracy	Time (seconds)	Accuracy	Time (seconds)	Accuracy	Time (seconds)
RF static set	0.276	0.028	0.650	2.331	0.851	60.899
RF dynamic set	0.603	0.132	0.746	1.419		
MLP static set	0.457	0.040	0.613	1.713		
MLP dynamic set	0.749	0.183	0.777	2.411		

`LegalCopyright` and `ProductName` entirely increased TP rate to 0.05% and decreased FP rate to 0.058%. We leave this study for future work.

Finally, additional experiments are required for a large fraction of file instances. Thus, we intend to evaluate this approach in future work. Nonetheless, this model is built to overcome previous limitations: packing, adversarial attacks, and 0-day attacks.

In addition, the rule-based majority vote might be replaced by a weighted majority vote because it allows different scoring for each classifier according to TP and FP.

Table 13
Performance test results of AI-HydRa from training to prediction time. Model in the first column indicates the four internal classifiers of AI-HydRa. The second column represents the time taken to learn each internal classifier using the training samples in Table 10. Test time in the third column represents the average time to test one sample and 6,395 labeled samples were used for the test dataset. Total time indicates the sum of training time in the second column and test time in the third column. Average total time in the last column was calculated as the average of the four values of the total time in the fourth column. The detection procedure of AI-HydRa can be understood from the following link (http://shorturl.at/lsNY3).

Model	AI-HydRa			
	Training Time (seconds)	Test Time (seconds/sample)	Total Time (seconds)	Average Total Time (seconds)
Static RF	0.152	60.8144	60.966	60.899
Dynamic RF	0.153		60.967	
Static MLP	0.017		60.831	
Dynamic MLP	0.016		60.830	

7. Conclusions

Although evasion methods are continually evolving, adversaries leave a footprint within the malware. The malware exhibits specific patterns or leaves considerable traces during activation. Although many features utilized by malware have been studied, further studies can be conducted to achieve a high detection accuracy to overcome 0-day attacks and adversarial attacks.

To do this practically, we proposed a mechanism using a deep-learning model and a supervised model. This hybrid model serves as a comprehensive detection scheme for preventing the dissemination of malware and is unlike prior ML-based techniques. The hybrid-based model provides hybrids in feature, classifier, and voting systems, unlike prior hybrid approaches (i.e., static/dynamic). This model helps to identify core aspects of benign files, which exhibit seemingly malicious files, providing a practical model with real-time processing. In particular, we realized that a hybrid model, by combining a respective different classifier that shows a high TP and a low FP, is very effective in malware detection. The proposed model also provides insights for achieving comprehensive detection with a small training dataset. We believe that the use of this model clarifies malware classification for 0-day attacks, and it can be employed extensively.

CRediT authorship contribution statement

Suyeon Yoo: Methodology, Formal analysis, Investigation, Writing - original draft, Writing - review & editing. **Sungjin Kim:** Methodology, Software, Formal analysis, Writing - original draft, Writing - review & editing. **Seungjae Kim:** Data curation, Software. **Brent Byunghoon Kang:** Supervision, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M.F. Ab Razak, N.B. Anuar, R. Salleh, A. Firdaus, The rise of “malware”: Bibliometric analysis of malware study, Journal of Network and Computer Applications 75 (2016) 58–76. doi: 10.1016/j.jnca.2016.08.022.
[2] M. Ahmed, A.N. Mahmood, J. Hu, A survey of network anomaly detection techniques, Journal of Network and Computer Applications 60 (2016) 19–31, https://doi.org/10.1016/j.jnca.2015.11.016.
[3] Anubis, URL:http://anubis.iseclab.org/.
[4] G.N. Barbosa, R.R. Branco, Prevalent characteristics in modern malware, black hat USA 2014 (2014).
[5] U. Bayer, A. Moser, C. Kruegel, E. Kirda, Dynamic analysis of malicious code, Journal in Computer Virology 2 (2006) 67–77, https://doi.org/10.1007/s11416-006-0012-2.
[6] Z. Bazrafshan, H. Hashemi, S.M.H. Fard, A. Hamzeh, A survey on heuristic malware detection techniques, in: Information and Knowledge Technology (IKT), 2013 5th Conference on, 2013, pp. 113–120. doi: 10.1109/IKT.2013.6620049.
[7] M. Christodorescu, S. Jha, University of Wisconsin-Madison Department of Computer Sciences, 2006.
[8] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, R.E. Bryant, Semantics-aware malware detection, Proceedings of Security and Privacy, IEEE Symposium on, 2005, pp. 32–46. doi: 10.1109/SP.2005.20.
[9] C. Curtsinger, B. Livshits, B.G. Zorn, C. Seifert, ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection, Proceedings of USENIX Security Symposium (2011) 33–48.
[10] A. Dinaburg, P. Royal, M. Sharif, W. Lee, Ether: malware analysis via hardware virtualization extensions, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 51–62, https://doi.org/10.1145/1455770.1455779.
[11] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, ACM Computing Surveys 44 (2012) 6, https://doi.org/10.1145/2089125.2089126.
[12] I. Firdausi, A. Erwin, A.S. Nugroho, et al., Analysis of machine learning techniques used in behavior-based malware detection, Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on, 2010, pp. 201–203. doi: 10.1109/ACT.2010.33.
[13] E. Gandotra, D. Bansal, S. Sofat, Malware analysis and classification: A survey, Journal of Information Security 5 (2014) 56, https://doi.org/10.4236/jis.2014.52006.
[14] K. Han, B. Kang, E.G. Im, Malware analysis using visualized image matrices, The Scientific World Journal (2014), https://doi.org/10.1155/2014/132713.

[15] R. Islam, R. Tian, L.M. Batten, S. Versteeg, Classification of malware based on integrated static and dynamic features, Journal of Network and Computer Applications 36 (2013) 646–656, https://doi.org/10.1016/j.jnca.2012.10.004.
[16] S. Jamalpur, Y.S. Navya, P. Raja, G. Tagore, G.R.K. Rao, Dynamic malware analysis using cuckoo sandbox, in: 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, pp. 1056–1060, https://doi.org/10.1109/ICICCT.2018.8473346.
[17] N. Kaur, A.K. Bindal, A. PhD, A complete dynamic malware analysis, International Journal of Computer Applications 135 (2016) 20–25, https://doi.org/10.5120/ijca2016908283.
[18] K.N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, D. Ponomarev, Ensemble learning for low-level hardware-supported malware detection, International Workshop on Recent Advances in Intrusion Detection (2015) 3–25, https://doi.org/10.1007/978-3-319-26362-5_1.
[19] S. Kim, S. Kim, D. Kim, Logos: Internet-explorer-based malicious webpage detection, ETRI Journal 39 (2017) 406–416.
[20] J. Kinder, S. Katzenbeisser, C. Schallhart, H. Veith, Detecting malicious code by model checking, International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2005, pp. 174–187. doi: 10.1007/11506881_11.
[21] J.Z. Kolter, M.A. Maloof, Learning to detect malicious executables in the wild, in: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 470–478, https://doi.org/10.1145/1014052.1014105.
[22] C. Kruegel, W. Robertson, G. Vigna, Detecting kernel-level rootkits through binary analysis, Computer Security Applications Conference, 2004 20th Annual, 2004, pp. 91–100. doi: 10.1109/CSAC.2004.19.
[23] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436, https://doi.org/10.1038/nature14539.
[24] T. Lee, J. Mody, Y. Lin, A. Marinescu, A. Polyakov, Application behavioral classification, Google Patents (2007).
[25] X. Liu, Y. Lin, H. Li, J. Zhang, Adversarial examples: Attacks on machine learning-based malware visualization detection methods, arXiv preprint arXiv:1808.01546 (2018) (2018).
[26] Y.B. Lu, S.C. Din, C.F. Zheng, B.J. Gao, Using multi-feature and classifier ensembles to improve malware detection, Journal of CCIT 39 (2010) 57–72.
[27] A. Moser, C. Kruegel, E. Kirda, Limits of static analysis for malware detection, Twenty-Third Annual Computer Security Applications Conference (ACSAC) (2007) 421–430, https://doi.org/10.1109/ACSAC.2007.21.
[28] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, Y. Elovici, Unknown malcode detection using opcode representation, Intelligence and Security Informatics (2008) 204–215, https://doi.org/10.1007/978-3-540-89900-6_21.
[29] V. Nair, G.E. Hinton, Proceedings of the 27th International Conference on Machine Learning (ICML) (red2010), pp. 807–814.
[30] L. Nataraj, S. Karthikeyan, G. Jacob, B. Manjunath, Malware images: visualization and automatic classification, in: Proceedings of the 8th International Symposium on Visualization for Cyber Security, 2011, p. 4, https://doi.org/10.1145/2016904.2016908.
[31] Y.H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (1992) 76–79, https://doi.org/10.1109/2.144401.
[32] M.A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, N. Provos, CAMP: Content-Agnostic Malware Protection, Proceedings of Network and Distributed System Security Symposium (NDSS), 2013.
[33] I. Santos, J. Devesa, F. Brezo, J. Nieves, P.G. Bringas, Opem: A static-dynamic approach for machine-learning-based malware detection, International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, 2013, pp. 271–280. doi: 10.1007/978-3-642-33018-6_28.
[34] S. Scardapane, D. Wang, Randomness in neural networks: an overview, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7 (2017), https://doi.org/10.1002/widm.1200.
[35] M.G. Schultz, E. Eskin, F. Zadok, S.J. Stolfo, Data mining methods for detection of new malicious executables, Proceedings of Security and Privacy, IEEE Symposium on, 2001, pp. 38–49. doi: 10.1109/SECPRI.2001.924286.
[36] A. Shabtai, R. Moskovitch, Y. Elovici, C. Glezer, Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey, Information Security Technical Report 14 (2009) 16–29, https://doi.org/10.1016/j.istr.2009.03.003.
[37] J. Su, D.V. Vargas, S. Kouichi, One pixel attack for fooling deep neural networks, arXiv preprint arXiv:1710.08864 (2017) (2017). doi: 10.1109/TEVC.2019.2890858.
[38] Virus_Total, URL:https://www.virustotal.com.
[39] D. Wang, C. Cui, Stochastic configuration networks ensemble with heterogeneous features for large-scale data analytics, Information Sciences 417 (2017) 55–71, https://doi.org/10.1016/j.ins.2017.07.003.
[40] D. Wang, M. Li, Stochastic configuration networks: Fundamentals and algorithms, IEEE Transactions on Cybernetics 47 (2017) 3466–3479, https://doi.org/10.1109/TCYB.2017.2734043.
[41] Weka_3, URL:https://www.cs.waikato.ac.nz/ml/weka/.
[42] Weka_Class_GainRatioAttributeEval, URL:http://weka.sourceforge.net/doc.dev/weka/attributeSelURL:ection/GainRatioAttributeEval.html.
[43] C. Willems, T. Holz, F. Freiling, Toward automated dynamic malware analysis using cwsandbox, Proceedings of Security and Privacy, IEEE Symposium on, vol. 5, 2007. doi: 10.1109/MSP.2007.45.
[44] X. Yuan, P. He, Q. Zhu, R.R. Bhat, X. Li, Adversarial examples: Attacks and defenses for deep learning, arXiv preprint arXiv:1712.07107, 2017. doi: 10.1109/TNNLS.2018.2886017.
[45] S. Yoo, S. Kim, B. Kang, The Image Game: Exploit Kit Detection Based on Recursive Convolutional Neural Networks, IEEE ACCESS (2020), https://doi.org/10.1109/ACCESS.2020.2967746.