Hindawi
Security and Communication Networks
Volume 2023, Article ID 7827823, 20 pages
https://doi.org/10.1155/2023/7827823

Research Article

KTSDroid: A Framework for Android Malware Categorization Using the Kernel Task Structure

Saneeha Khalid, Khalid Imran, and Faisal Bashir Hussain

Bahria University, Islamabad, Pakistan

Correspondence should be addressed to Saneeha Khalid; saneeha.nust@gmail.com

Received 11 October 2022; Revised 3 November 2022; Accepted 24 November 2022; Published 13 May 2023

Academic Editor: Hammad Afzal

Copyright © 2023 Saneeha Khalid et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The penetration of malicious applications in the Android market has enhanced the significance of designing malware mitigation systems for Android. Malware detection systems are being developed by examining applications using static and dynamic analysis techniques. The use of code obfuscation has highlighted the importance of dynamic analysis as many static analysis schemes can be evaded by code obfuscation strategies. In order to record the true working of the application, a volatile memory-based solution for application analysis is presented in this study. Time-based memory dumps are collected after interactions with an application. Process-specific artifacts of the application under analysis are extracted by examining the kernel task structure of memory. The features in the kernel task structure belong to nine broad categories based on their semantics. An important contribution of the study is the analysis of the kernel task structure for determining the set of effective categories and features for Android malware categorization. Three of the most important categories and fourteen valuable features are reported. The proposed system categorizes the applications into five classes: adware, banking Trojans, riskware, SMS Trojans, and benign. The proposed system is able to categorize applications with an average F1-score of 0.984, which is the highest score reported so far for multiclass Android malware categorization with a minimum number of kernel task structure-based features.

1. Introduction

The tremendous rise in smartphone usage has transformed working patterns all over the world. Many business and personal tasks are performed using smartphones as they are considered more accessible and easier to use as compared to other devices. The adaptability of smartphones all over the world is due to the highly efficient and usable operating systems such as Android, iOS, and Windows. Android is the most used operating system for smartphones and holds a major market share of 71.45 percent (https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/). The success of Android can be attributed to the large number of Android-compatible user applications. These applications are frequently used by users and are considered reliable by a large population. The malicious application developers take advantage of the usage and popularity of Android and are developing a large number of malicious applications for the platform. According to statistics, 10.5 million Android malware infections were detected in 2019 and 0.48 million new Android malware infections per month were found in 2020 (https://www.statista.com/statistics/680705/global-android-malware-volume/).

The increasing rate of malware penetration in Android is a serious threat [1]. Therefore, many schemes have been proposed to mitigate this issue. Signature-based schemes have dominated malware detection techniques, but their major drawback is the inability to detect zero day malware. Signature-based schemes are also known to be less efficient against malware variants [2]. A more generic approach to malware detection and categorization is the creation of generic behavior patterns. Machine learning-based methods are used for creating generic patterns; however, the selection of useful and significant features is important for creating effective classification systems.

In order to analyze malicious Android applications using machine learning approaches, static and dynamic analysis techniques can be used [3]. Static analysis refers to analyzing an application by examining its structure and code without execution. Static analysis becomes less effective when the applications use code obfuscation techniques for hiding the code semantics [4].

Obfuscation refers to arranging the structure of code in a way that reverse engineering becomes difficult [5]. Common obfuscation schemes include class encryption, code reordering, reflection, junk code insertion, and control flow modification [6]. In order to minimize the effect of obfuscation, dynamic analysis techniques are widely being used at present. Dynamic analysis refers to analyzing the application by executing it in a sandbox. These techniques extract the runtime activity of applications; therefore, they are more resilient against obfuscation techniques [7].

Dynamic analysis helps analyze the runtime behavior of an application with the help of network activity [8], runtime API usage [9], and volatile memory usage [10]. The usage of volatile memory for extracting dynamic features has gained significant attention in recent past, as the true working of the application is visible by extracting memory artifacts. Also, code obfuscation schemes become ineffective, as memory-based artifacts show the actual essence of the executed code.

Many recent studies [11–13] have highlighted the importance of volatile memory-based artifacts for Android malware detection. Process metadata features present in the kernel task structure of memory represent an important source of information for malicious application detection [12, 14, 15]. The kernel task structure is used by the operating system for managing the running processes. It contains all the information about the running application which is needed by the kernel for managing the process. The information in the kernel task structure is contained in a number of features, which are grouped into nine categories: task_state, mem_info, scheduling_info, signal_info, process_credentials, I/O_statistics, openfile_info, CPU_specific_state, and others. These categories contain related features as per their semantics. In addition to direct features, the categories of the kernel task structure also contain a number of structures (structs) that can be traversed for extracting deep features. However, existing studies lack the in-depth working on these features in terms of extraction and analysis. It has been observed that only initial categories and structures are investigated for extraction of features. The analysis of all categories for the selection of most relevant features for malware identification and categorization needs to be thoroughly investigated. Additionally, existing studies have focused on the evaluation of process metadata features for the detection of malicious applications only. The evaluation of these features for categorizing malicious applications into respective classes is not performed.

In this study, KTSDroid, a malware mitigation framework based on process-specific artifacts from volatile memory, is proposed. The proposed framework captures volatile memory dumps while executing the application under analysis. The application's behavior profile is generated by analyzing process-specific artifacts (process metadata) from the kernel task structure in memory. KTSDroid extracts direct features from all nine categories of the kernel task structure. In addition to direct features, the nine categories of the kernel task structure are traversed up to a depth of six levels for the generation of a feature set. In order to ascertain that the extracted features contain useful information about the malicious behavior of the application, a time-based memory dump extraction process is conducted. In addition to this, random events are generated on the application before the capture of each dump to ensure interactions. As a result, four dumps with interactions are generated for each application in the dataset. Each dump is then utilized for the extraction of the kernel task structure for the process (application) under analysis. Overall, the contributions of the study are as follows:

(1) The study proposes a kernel task structure-based Android malware categorization framework by utilizing multiple time-based memory dumps with interactions.

(2) The kernel task structure of memory is utilized for the extraction of features. To the best of our knowledge, this is the first study to explore nine categories of the kernel task structure for the extraction of features w.r.t. the Android platform. In addition to this, traversal of each category to a depth of six levels is performed. A comprehensive feature set comprising 526 process specific features, grouped into nine distinct categories, is used for analysis.

(3) The effectiveness of kernel task structure features is reported against five distinct Android application classes, i.e., adware, banking Trojans, riskware, SMS Trojans, and benign.

The rest of the paper is organized as follows: details of the kernel task structure are discussed in Section 2. Section 3 presents the related work. Proposed methodology for feature extraction and selection is presented in Section 4. Results for the proposed methodology are reported in Section 5. Section 6 discusses the results, and finally, conclusion is presented in Section 7.

2. Overview of the Kernel Task Structure

KTSDroid utilizes the kernel task structure of memory for feature set extraction. In order to device an effective malware mitigation strategy, the understanding of the design and layout of the kernel task structure is important. This section briefly introduces the functions of this structure and highlights the categories in which features are organised within it.

The kernel task structure, also known as process control block, is a data structure in the kernel space of memory that contains important information about the running processes. The operating system uses this structure for managing all the running processes by dynamically allocating the structure to each process. In order to analyze the task structure of a particular process, the PID of the process can be used. The information in the kernel task structure is

effective in identification and classification of malware because it describes the target application in running state and hence overcomes the problems caused by code obfuscation techniques. Information contained in the Android kernel task structure can be grouped into nine categories. Features from all these categories are utilized by KTSDroid for malware categorization. The information contained in each category is listed in Table 1.

3. Related Work

Recently, the design of memory-based schemes for Android malware categorization has gained substantial attention due to their strong resilience against various obfuscation schemes. Many studies [13, 16, 17] have associated the effectiveness of memory-based artifacts for malicious Android application detection with their ability to represent the runtime execution of the application. Different strategies are adapted by researchers to analyse memory for malicious application detection. Some of the studies [10, 18] have utilized volatile memory dumps against malicious applications in the form of images and classified them on the basis of differences in images. However, the complete memory dump contains a number of other processes as well, and the analysis is not specific to the process (application) under evaluation. Memory-based features are used by [19] for classifying Android applications into benign and malicious classes. Thirty two features using different plugins for volatility are extracted and used for analysis. The study has not incorporated time-based capturing of features, and the proposed dataset is built using only one memory snapshot. Another approach is the use of process metadata features, available in the kernel task structure of memory. These features contain useful information about the process behavior and can be used to create a malware detection system. This section highlights the memory-based frameworks that have used process metadata features from the kernel task structure for application analysis, as they are closer to the approach presented by this study.

Wang and Li [12] proposed a framework for the detection of malware on the Android platform using machine learning with a feature set from the kernel task structure. A total of 112 features grouped into 5 categories are extracted from the task struct using 1275 malware samples and 1275 benign samples. Principal component analysis, chi-squared statistic, correlation, and information gain are used as dimension reduction methods. The proposed framework is evaluated by using 4 different machine learning algorithms (Naive Bayes, decision tree, neural network, and K-nearest neighbors). It is shown that the proposed framework can achieve 94% to 98% accuracy and less than 10% false positive rate.

Alawneh et al. [14] proposed a malware detection system that can identify trojanized malware. The focus of the study is on improved detection time rather than the accuracy of the model. In the experiment, 112 fields were extracted from the kernel process control block and grouped into five categories, including mem_info, CPU_scheduling_info, signal_info, task_state, and others. For dataset creation, a total of 2400 apks (1200 benign and 1200 malware) were used. Features were recorded for 15 seconds for each apk. The study is evaluated by using a back propagation neural network (BPNN). The model is evaluated by considering the feature set, and it is reported that the best result is achieved by selecting 43 features out of 112, with 96.8% detection accuracy, which takes around 30 seconds for training the classifier and 73 μs for malware detection after 100 ms of information mining.

Shahzad et al. [15] proposed a real-time malware detection framework, namely, TstructDroid, for Android-based devices. 110 benign and 110 malicious applications are used for dataset creation. Out of 99 preliminary task structure fields, 32 fields are shortlisted for dataset creation using time series feature shortlisting techniques. After shortlisting features, time series blocks are created and then frequency information is calculated using the discrete cosine transform. The framework achieves a detection rate of 90-93.6% with a false alarm rate between 5.4% and 7.3%.

Kim and Choi [20] proposed a malware detection method for the Android platform. Features are extracted from the proc filesystem of the Linux platform. The proc filesystem is a virtual filesystem which provides an interface to the kernel structures, i.e., it permits communication between the user space and the Linux kernel. A total of 36 features out of 59 features are selected from three classes: memory, CPU, and network. Feature extraction was performed periodically every 10 seconds, against the applications being executed. Support vector machine (SVM) is used as a classifier for performance evaluation in the experimental setup. A TPR of 95.97%, an FPR of 0.67, a precision of 96.63%, and an accuracy of 98.85 were achieved after feature selection. The results are computed for six applications only.

A summary of the related work on kernel task structure-based malware detection is presented in Table 2. It is pertinent to highlight that the studies have only classified the applications into malicious and benign categories. Determining the category or nature of the malware is significant for understanding the criticality of threat and effective mitigation. Another important observation is the availability of a large number of kernel task structure-based features in Android, whereas previous works have focused on a limited set of features.

4. KTSDroid Android Malware Categorization Using the Kernel Task Structure

KTSDroid is an Android malware detection and categorization framework that uses dynamic memory information extracted from the programs' kernel task structure. The architecture of KTSDroid is shown in Figure 1 that has three components: feature extraction, feature selection, and classification. During feature extraction, initially, Android applications are executed and interactions are made to obtain process-specific memory dumps. These dumps are further analyzed for extracting process metadata features from the kernel task structure. In the second phase, a step-by-step approach is adapted for investigating significant categories and features. Finally, the selected features are used

Table 1: Information in categories of the kernel task structure.

KTS category	Information
task_struct	State of the process like exit code and process execution domain
mem_info	Major and minor page faults, heap address of the process, start and end address of code segment, and start and end address of data segment
scheduling_info	Priority of the process, scheduling state, scheduling policy, execution time, waiting time, snapshot of user, and system CPU time
signal_info	Signal sources, the signal handler, and timers related to the process
process_credentials	Ownership and process capabilities
I/O_statistics	Block I/O delay and I/O statistics like number of byte read, number of read system call, and number of write system calls
openfiles_info	Opened files related to the process like maximum number of file descriptor and opened file descriptor
CPU_specific_state	CPU state of the process, which includes different register states and fault info
Others	Miscellaneous information about the process like age of the process and tracer information

to classify the applications into respective benign or malware classes. In the remaining of this section, the aforementioned three core components of the KTSDroid framework are discussed in detail.

4.1. KTSDroid Feature Extraction. KTSDroid uses a dynamic analysis scheme based on volatile memory artifacts for application analysis. This section covers the details of the memory-based feature extraction process. The dataset under analysis consists of N apk files, and each apk belongs to a class C_i, where i ranges from 1 to 5. In order to formulate a malware detection system using these apks, each apk must be processed to extract volatile memory-based features.

The overall process of feature extraction can be divided into two major steps: memory dump extraction and process metadata extraction by traversing the kernel task structure. The details are provided in the subsections as follows, and an overall view of the feature extraction process is shown in Figure 2.

4.1.1. Memory Dump Extraction. The dynamic nature of the proposed system requires the applications to be executed in a controlled environment for evaluation. For this purpose, an AVD (Android virtual device) environment is chosen, where the AVD is configured for application installation and memory dump extraction. The AVD is created with the Nexus 6P hardware profile and Android 9.0 (Google APIs) system image with x86_64 architecture. The host system is Ubuntu 18.04, and communication between the host system and the virtual device is carried out by using ADB (Android debug bridge).

The features used by the system are based on memory; therefore, memory dump extraction is the first step in application analysis. In order to extract a memory dump, LiME (Linux Memory Extractor) (https://github.com/504ensicsLabs/LiME), a loadable kernel module needs to be compiled for the target kernel of AVD and loaded into the device. LiME is an open-source loadable kernel module (LKM) for Linux-based devices, which allows the acquisition of complete volatile memory dumps. For the LiME module to be loaded by the target kernel, the kernel needs to be compiled with loadable kernel module support for the target device. It is because Android does not have default support for loadable kernel modules. KTSDroid uses Goldfish kernel 4.4, compiled with loadable kernel module support.

KTSDroid uses a Python application for automating the process of scanning a local directory containing a dataset of apk files, installing the application (apk file) on the virtual device, simulating pseudorandom user input using monkey (https://developer.android.com/studio/test/other-testing-tools/monkey) against the installed application and capturing a memory dump of the virtual device. For analysis, a total of four volatile memory dumps were obtained. The first dump was taken right after the application installation. The sunsequent dumps were taken after event generation on the apk. The events were triggered using Monkey and included 150, 1500, and 4000 events. Acquiring volatile memory dumps during different states of the application adds variability to the dump and hence contributes to a rich data collection that comprises 10,000 memory dumps. Simulating user inputs is required to ensure code coverage and triggering of the malicious behavior. Installation and execution of applications (benign/malicious) are performed while the device is running in the read-only mode. Running the device in the read-only mode is required to keep changes made by the applications nonpersistent and ensure the device is in a clean state after every restart. Although process-specific features are considered during the study, but to ensure smooth execution of the application and the device, a single application is considered for installation and analysis at a time.

4.1.2. Process Metadata Extraction. The memory dump extraction process produces a set of four memory dumps for each apk in the data set. To extract digital artifacts from these volatile memory dumps, the Python application is extended to use the volatility framework (https://www.volatilityfoundation.org/) as a library. Volatility is an open-source collection of tools used for the analysis of volatile memory samples of Mac, Windows, Linux, and Android-based devices. In order to use volatility for the analysis of the memory dump of the target Android device,

Table 2: Related work on malware detection using kernel task structure features.

Study	Number of effective features	Feature selection techniques	Algorithm classification	Reported performance	Data set size	Multiclass classification	KTS category analysis
Wang and Li [12]	10–40 out of 112	PCA, correlation, IG, and chi-square	Naïve Bayes, decision trees, and neural network	ACC: 94%–98%	1275 malware, 1275 benign	×	Partial
Alawneh et al. [14]	43 out of 112	Logistic regression	Neural network	ACC: 96.80%	1200 malware, 1200 benign	×	×
Shahzad et al. [15]	32 out of 90	Correlation	Decision trees (J48)	ACC: 93%–96%	110 malicious, 110 benign	×	×
Kim and Choi [20]	36 out of 59	Manual	Support vector machines (SVM)	ACC: 98.85%,	6 malicious	×	×

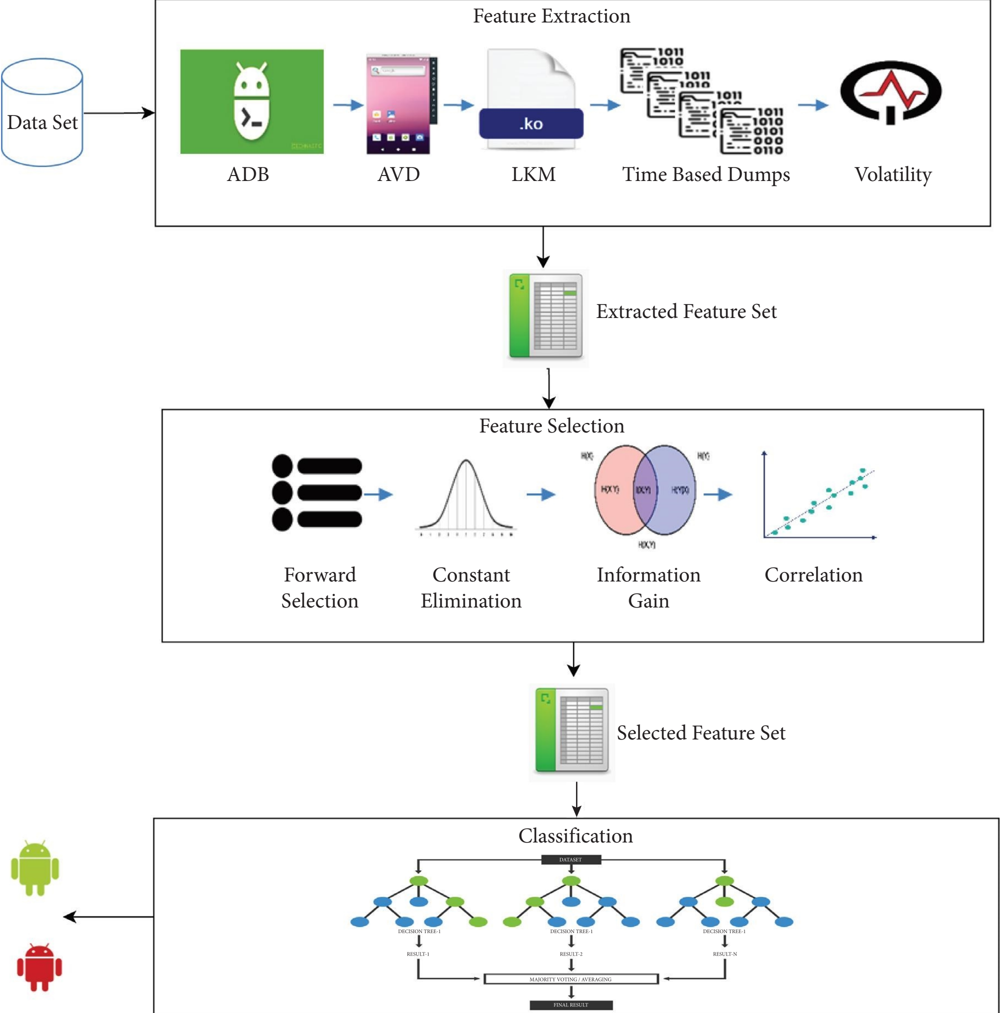


Figure 1: KTSDroid framework for feature extraction, feature selection, and classification.

a profile for the target kernel is generated. The profile is used by the volatility framework for locating and parsing information in the memory dump. The Linux_pslist plugin of volatility, which collects active tasks by walking through the kernel task structure, is utilized for extracting the features of the running application.

The kernel task structure is a combination of many structs, as each field in the structure may also be a struct containing other fields. This makes it a cascaded structure with a lot of information about the running process. The Python-based application iterates the kernel task structure six levels deep for the extraction of features. The depth of features explored in this study is shown in Figure 3. The extracted information is recorded into a csv file with columns representing the features and rows representing the memory dumps. Each record consists of 526 fields from the kernel task structure.

4.2. KTSDroid Feature Selection. The feature extraction process produces a rich set of features for each apk in the dataset. In order to design an effective malware categorization system, all of these features need to be analyzed for their significance in malware detection. For this purpose, a feature selection process is designed to gauge the importance of features. Feature selection is an important part of formulating a machine learning-based system, as using significant features positively impacts the performance of the classification system. Feature selection helps improve the training time, reduce the complexity of the model, and improve performance. Feature selection is also a useful way of handling overfitting which results in enhanced model generalization [21].

Feature selection methods can be divided into two broad categories: wrapper-based methods and filter-based methods. Wrapper-based methods use a classification algorithm to find the effectiveness of features, and filter-based methods use statistical techniques to find the importance of a feature in output prediction [22]. This study uses methods from both of these techniques for finalizing the set of important features.

The feature extraction process iterates through the kernel task structure to extract a rich set of 526 features. These features belong to nine categories as per the general categorization of the kernel task structure. As the number of

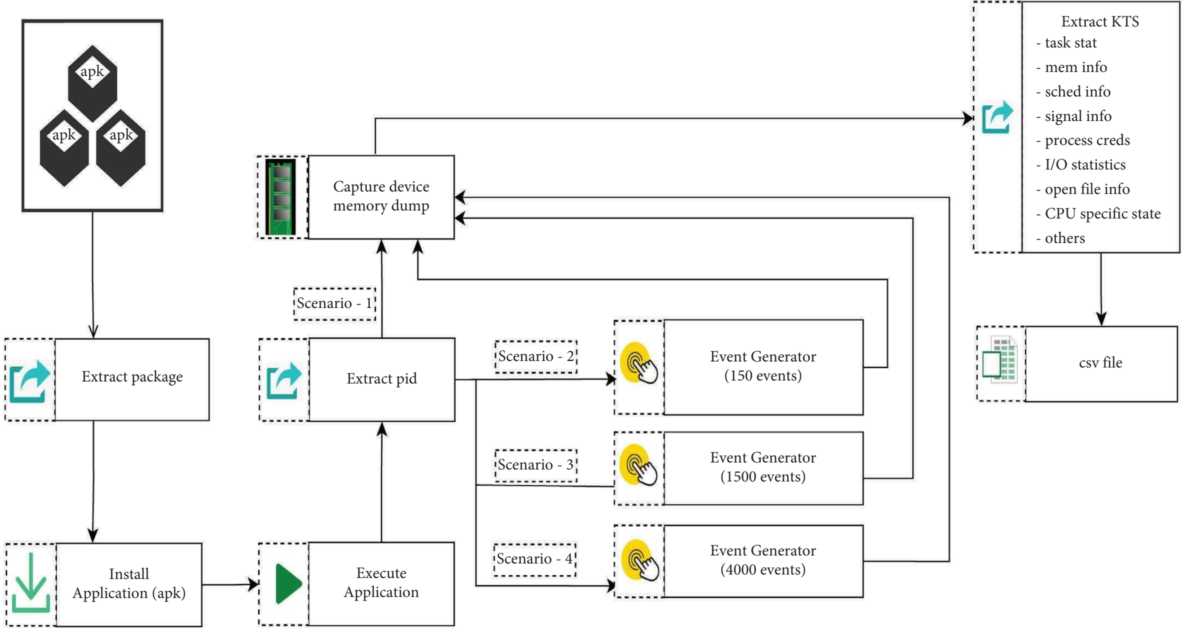


Figure 2: KTSDroid feature extraction process.

features is quite large, selecting the most effective features is significantly important. The feature selection approach used in this study comprises two steps. In the first step, all categories of the kernel task structure are analyzed to find the categories that are not significant for malware categorization. The reason for using this strategy is that all the features in a certain category are related to each other semantically. It may be possible that the information present in a certain category is not an effective identifier of malicious behavior. Therefore, finding the significant categories that contain information related to malware identification is important. Once important categories are identified, the selected categories are parsed to find the most significant features for malware categorization. Details of these steps are described in the subsections as follows.

4.2.1. Significant Kernel Task Structure Category Identification. The first step of feature selection is the identification of significant categories of the kernel task structure for malware categorization. For this purpose, the set of all features is grouped category-wise and labeled for each class. All categories are then represented by the set $\mathrm{CT}^{\mathrm{all}}$, where $\mathrm{CT}^{\mathrm{all}} = \{ct_1, ct_2, ct_3, ct_4, ct_5, ct_6, ct_7, ct_8, ct_9\}$. Each category, ct_i, contains a number of features, as shown in Table 3.

The initial process of feature selection focuses on evaluating the significance of a complete category instead of evaluating each feature individually because the features in each category are semantically related to each other. For example, mem_info contains features related to memory usage, and IO_statistics refer to features related to IO operations. As the features are semantically related, a broader landscape of feature importance can be extracted by looking into the significance of each category for classification.

In order to find the set of significant categories, $\mathrm{CT}^{\mathrm{sig}}$, where $\mathrm{CT}^{\mathrm{sig}} \subseteq \mathrm{CT}^{\mathrm{all}}$, a wrapper-based selection method, is used. Wrapper methods use the evaluation metrics of the classification model to find the best set of features. Features are supplied to the classification system, and performance is measured. The set of features that report the highest performance in optimal time is selected [22]. Many wrapper-based feature selection methods are available. For this study, the wrapper method of forward selection is used. It utilizes a model and threshold value of performance measures to find the best set of features. In forward selection, a classification model is created for each feature in the dataset. The model's performance is recorded, and the best performing feature is selected. In each step, the next best feature is added to the model and the process continues. This method is very resource-intensive, as there are many features in a dataset, and testing each of them one by one is a time- and resource-intensive task [23]. However, in this experiment, the complexity of forward selection is reduced to only nine features ($\mathrm{CT}^{\mathrm{all}} = \{ct_1, ct_2, ct_3, ct_4, ct_5, ct_6, ct_7, ct_8, ct_9\}$) as a complete category of features is considered at a time. This reduces time and resource complexity by a large amount.

While applying wrapper-based methods, the selection of a suitable classification algorithm according to the type of data is important. It should also be considered that in all wrapper methods, classification is performed a number of times on subsets of features; therefore, the process must be concluded when a certain threshold for performance is achieved. In this work, the model used for classification in forward selection is random forest. Random forest is an ensemble of decision trees. It is recommended as a classifier

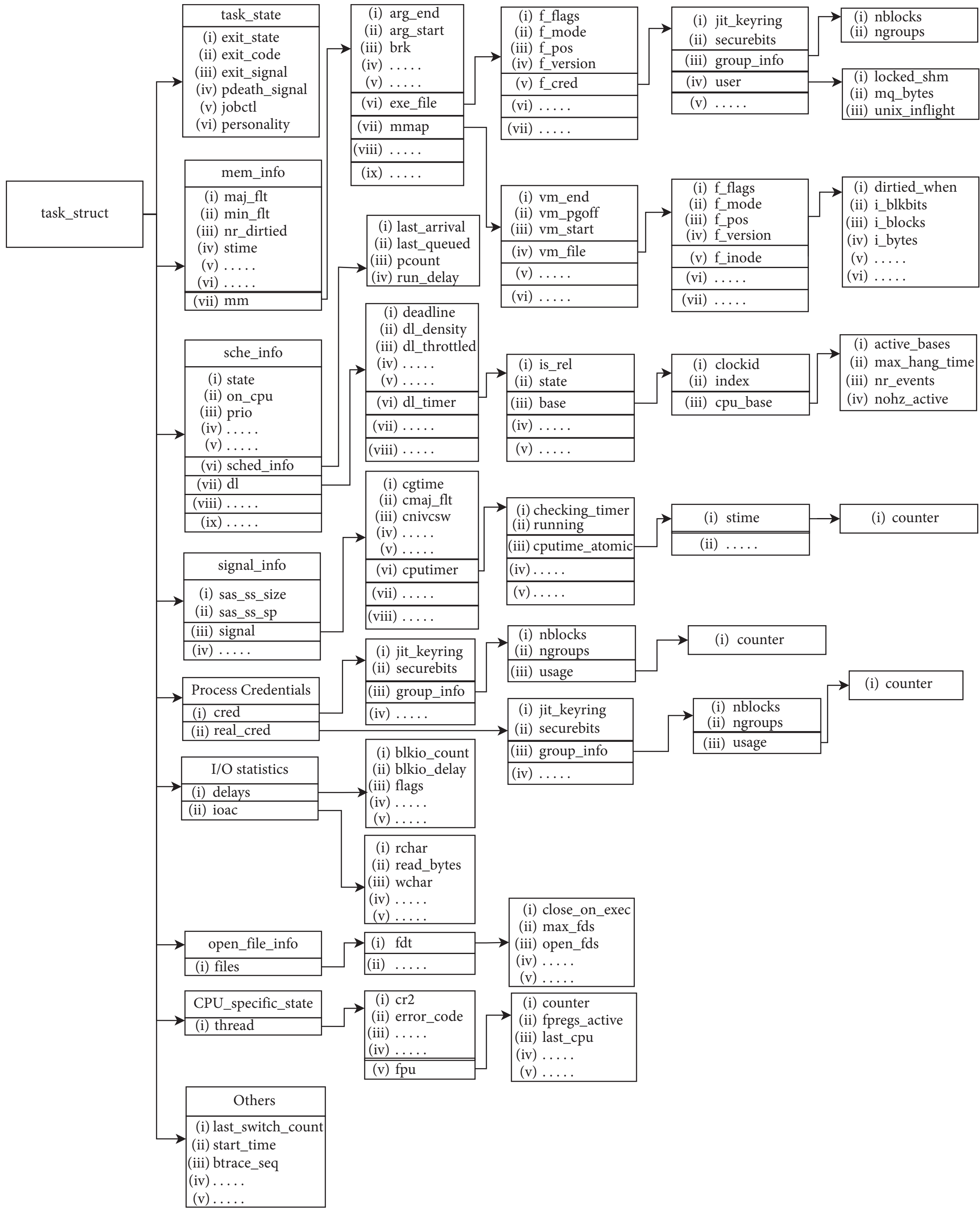


Figure 3: Kernel task structure-extracted categories and features.

for Android malware detection by a number of studies [24–26]. The iterations of measuring classification results are terminated when the performance metrics become constant and adding new categories does not add to the performance enhancement. After applying forward selection, the set of all categories $\mathrm{CT}^{\mathrm{all}}$ is reduced to a smaller set $\mathrm{CT}^{\mathrm{sig}}$, where $\mathrm{CT}^{\mathrm{sig}} = \{ct_i \ldots\ldots ct_m\}$ and m is the number of significant categories for malware classification.

4.2.2. Significant Feature Selection. After the identification of important categories of the kernel task structure, the next step is to find the most minimal and effective feature set for Android malware categorization. For this purpose, a three-phase process is used. In the first phase, features are analyzed to find the set of constant features. All constant features are identified and dropped from further analysis. In the second phase, the remaining set of features is evaluated for their

Table 3: Number of features in kernel task structure categories.

KTS category	Number of features
task_state	5
mem_info	212
scheduling_info	91
signal_info	83
process_credentials	75
I/O_statistics	18
openfile_info	12
CPU_specific_state	20
Others	17

mutual information (MI) values against the output class. Features with insignificant MI values are not considered for further analysis. Finally, in the last phase, the dimensions of the data are further reduced by removing the linearly correlated features. All phases of feature selection are applied to each category separately because of the following two reasons:

(1) Evaluating the features category wise makes the results manageable and easy to understand
(2) The final result of feature selection summarizes the contribution of each category of the kernel task structure in the final feature set.

(1) Phase 1: Constant Feature Elimination. The feature for which the values remain the same for all classes is referred to as a constant feature. These features increase the dimensionality of the feature set and can be a cause of the slow convergence of the training algorithm [22]. Using these features has no effect on the predictive power of the model; only the complexity of the model increases. Therefore, such features can be dropped from the dataset. In order to find constant features, a statistical measure of variance can be used [27]. Variance measures the variability in the values of a variable and can be used to check the constant nature of a feature. It measures the spread in the values of a variable by calculating the average squared distance from the mean. Variance is widely used in feature selection by setting a threshold value for the variability of values against a feature. In this study, we are interested in finding features that have no variance in values; therefore, the threshold value is 0. If the value for variance is zero, it indicates that the feature is constant and can be dropped from further analysis.

KTSDroid groups the extracted features category-wise; therefore, all features in a category, ct_i, where $ct_i \in \mathrm{CT}^{\mathrm{sig}}$ are evaluated for variance. If f_{ij} is the jth feature in the ith significant category, then equation (1) illustrates the process of feature selection through variance.

$$f_{ij}^{\mathrm{temp}} = \left\{ \begin{array}{ll} f_{ij} & \mathrm{var}\left(f_{ij}\right) > 0 \\ \phi & \text{otherwise} \end{array} \right\}, \quad (1)$$

where i ranges from $1, \ldots, m$ and m is the number of selected categories; j ranges from $1, \ldots, n$ and n is the number of features in the ith category.

In equation (1), the feature f_{ij} is tested using variance as given by equation (2). It is selected as f_{ij}^{temp} if the value for variance is greater than zero.

$$\mathrm{var} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(x_i - \overline{x}\right)^2}. \quad (2)$$

Initially, F_i^{sel} is an empty set for i^{th} category. After testing all features in a category, the selected features are added to set F_i^{sel}, as shown in the following equation:

$$F_i^{sel} \leftarrow \bigcup_{j=1}^{n'} f_{ij}^{\mathrm{temp}}, \quad (3)$$

where n' is the number of nonconstant features.

Constant feature elimination is the most basic step of feature selection. The feature set after constant feature elimination is stored for further analysis using mutual information in phase two of feature selection.

4.2.3. Phase 2: Mutual Information (MI). Mutual information, also referred to as information gain, is one of the most commonly used filter-based methods for feature selection [28]. Its working is based on entropy, and it measures the reduction in entropy after the transformation of a dataset. It estimates the dependency between a feature and the output. In this work, mutual information is chosen for feature significance evaluation as it estimates linear as well as nonlinear relationships between feature and output as compared to other univariate feature selection methods like F-test that only finds the linear dependencies between two variables [29]. Another important point to be considered is the size of the features to be evaluated. The features need to be tested individually; therefore, an efficient algorithm in terms of time and resource usage must be used. Mutual information can be easily applied to a large number of features because of its lower complexity and computation time [30].

Mutual information helps in finding significant features by estimating the dependency of output on a feature. Significant features are always related to the output in some way. However, features that are insignificant in the prediction of output always have low or no dependency on output. If the value of mutual information for a feature is zero, it indicates that the output is independent of the feature. Higher values show a higher dependency between the feature and the output.

The feature set produced by constant feature elimination F_i^{sel} for each category, as shown by equation (3), is evaluated to find the value of mutual information against all features. If f_{ij} is the jth feature in the ith category, then equation (4) represents the application of mutual information and the selection of significant features based on a threshold T.

$$f_{ij}^{\mathrm{temp}} = \left\{ \begin{array}{ll} f_{ij}^{sel} & \mathrm{MI}\left(f_{ij}^{sel}\right) > T \\ \phi & \text{otherwise} \end{array} \right\}, \quad (4)$$

where i ranges from $1, \ldots, m$ and m is the number of selected categories; j ranges from $1, \ldots, n'$ and n' is the number of nonconstant features in the ith category. T is the threshold value.

$$\mathrm{MI}\left(f_{ij}^{sel}; C\right) = H\left(f_{ij}^{sel}\right) ˇ H\left(f_{ij}^{sel} | C\right). \tag{5}$$

In equation (4), f_{ij}^{sel} is tested for mutual information with respect to output class C by using equation (5). The feature, f_{ij}^{sel}, is selected as f_{ij}^{temp} if the value for mutual information is greater than threshold T. The set of all selected features (f_{ij}^{temp}) is moved to set F^{temp}, as shown in equation (6). Finally, the set of selected features, F_i^{sel}, is replaced by the set of selected features using mutual information as shown in the following equation:

$$F_i^{\text{temp}} \leftarrow \bigcup_{j=1}^{n''} f_{ij}^{\text{temp}}, \tag{6}$$

where n'' is the number of features with the MI score greater than threshold T

$$F_i^{temp} \leftarrow F_i^{\text{temp}}. \tag{7}$$

In order to select the features based on mutual information, the selection of a threshold value (T) is extremely important, as features with MI scores greater than the threshold will be included in the F_i^{sel} set. In order to find the optimal value for T, the feature set is evaluated at different threshold values $(T = 0.0, 0.1, \ldots, t)$ against a performance metric. The threshold value starts from zero and is increased by 0.1 for feature set selection. The process stops for $T = t$, where t is a threshold value for which the performance becomes constant or starts decreasing. The complete algorithm for feature selection for all categories using mutual information is shown in Algorithm 1.

After the selection of features using mutual information, the updated feature set F_i^{sel} for each category is analyzed to find linearly correlated features.

4.2.4. Phase 3: Correlation. Correlation is a statistical measure for finding the linear dependency between two variables. For feature selection, correlation can be used to find features, which have a strong linear relationship [31] among themselves. It is beneficial, as two features having a strong linear dependency on each other will have almost the same effect on the output variable. Therefore, they may be replaced by any one of them. This helps reduce the dimensionality of the data and the complexity of the model [32].

KTSDroid computes correlation for all features within a selected category. F_i^{sel} is the set of features obtained after eliminating features with mutual information values less than the desired threshold. If f_{ij}^{sel} and f_{ik}^{sel} represent two features from the set of selected features for a category, then the application of correlation can be described by equation (8). Here, one of the features is selected as f_{ij}^{temp} if the two have a correlation value of 0.95.

$$f_{ij}^{\text{temp}} = \left\{ \begin{array}{ll} f_{ij}^{sel} & \mathrm{corr}\left(f_{ij}^{sel}, f_{ik}^{sel}\right) > 0.95 \\ \phi & \text{otherwise} \end{array} \right\}. \tag{8}$$

If f_{ij}^{sel} and f_{ik}^{sel} are represented by $f1$ and $f2$, then correlation between the two features can be defined by the following equation:

$$corr = \frac{\sum_{i=1}^{n''}\left(f1_i - \overline{f1}\right)\left(f2_i - \overline{f2}\right)}{\sqrt{\sum_{i=1}^{n''}\left(f1_i - \overline{f1}\right)^2\left(f2_i - \overline{f2}\right)^2}}, \tag{9}$$

where n'' is the number of features with the MI score greater than threshold T

Equation (9) refers to the person coefficient for finding correlation. A value close to 1 indicates a positive linear relationship, and a value close to −1 indicates a negative linear relationship. A value of 0 indicates that both features are linearly independent. The set of linearly correlated features is grouped into a set F^{temp} by using equation (7). Finally, all correlated features are removed from the selected feature set using the following equation:

$$F_i^{sel} \leftarrow F_i^{sel} - F_i^{\text{temp}}. \tag{10}$$

4.3. KTSDroid Classification. The selected set of features after applying a number of feature selection techniques can now be evaluated for performance using a classification model. Random forest (RF) is a classification model that uses decision trees as the underlying base classifier. It works by creating a number of trees and later accumulating the results from each. Each tree is generated using bagging and a bootstrap sample of data [33]. Given a training set D, bagging generates M new training sets D_i, where D_i is generated from D uniformly and with replacement. Sampling with replacement means that in each set D_i, some features will be unique and some will duplicate. After the generation of trees, the final result is obtained by either averaging, weighted averaging, or voting [34]. RF has a low bias as the trees are unpruned and fully grown. The correlation among the trees is also low; each tree is built independent of its peers [26].

KTSDroid used RF for the evaluation of the selected feature set. It has been chosen for classification as the detection of malware is a rule formation problem, and RF generates a number of rule sets in the form of trees. It is also not prone to overfitting and does not require retraining to fine-tune a large number of parameters. Overall, it is an efficient ensemble classification model with low bias and variance. Many malware analysis studies have also reported higher performance measures with RF as compared to other classifiers [24–26]. In this work, the final result of classification is obtained by voting on the results of candidate trees.

5. Experiments and Results

The dataset used for this study is CICMalDroid 2020 [35], developed by the Canadian Institute of Cyber Security. Applications belonging to five distinct classes: adware,

```
(1) procedure Extracting_Significant_Features_using_MI
(2)   for i ⟵ 1 to m do
(3)     Load the set of all features in the category (F_i^sel)
(4)     //F_i^sel is the feature set after constant featureelimination
(5)     F_i^temp = []
(6)     T = 0.0 //T is the threshold
(7)     Previous_F1_Score = 0.94
(8)     //F1_score after removing constant features
(9)     Current_F1_Score = 0
(10)    while Current_F1_Score < = Previous_F1_Score do
(11)      for j ⟵ 1 to F_i^sel.len() do
(12)        f_ij^temp ⟵ f_ij^sel
(13) Score = Mutual_Information(f_ij^temp, C)
(14)        if Score < T then
(15)          F_i^temp.append(f_ij^temp)
(16)        end if
(17)      end for
(18)      Current_F1_Score ⟵ F1_Score(F_i^temp, C)
(19)      if Current_F1_Score ≥ Previous_F1_Score then
(20)        T = T + 1
(21)      end if
(22)    end while
(23)    F_i^sel ⟵ F_i^temp
(24)  end for
(25) end procedure
```

Algorithm 1: Algorithm used by KTSDroid for feature selection using mutual information.

banking Trojans, riskware, SMS Trojans, and benign, are selected for analysis. The data set consists of 3000 applications, where each malware class has 500 samples and each benign class has 1000 samples. As the data set consists of apk files, each apk needs to be processed in order to extract useful artifacts. The feature extraction process executes each apk and extracts four memory dumps with interactions. The time-based memory dump extraction ensures the capture of malicious activity. Each dump is processed to traverse the kernel task structure for the extraction of features. Nine categories of the kernel task structure are traversed six levels deep to generate a comprehensive dataset of 526 features. Overall, the dataset used by the study consists of a total of 12,000 records, with 2000 records against each malware class and 4000 records against the benign class. A summary of dataset details is presented in Table 4.

The comprehensive analysis of kernel task structure results in a rich set of features. KTSDroid analyzes these features by using a number of feature selection techniques. The final set of selected features is then used for classification. The remaining part of this section discusses the results of all applied feature selection techniques and classification. The results are organised according to the sequence of applied techniques. Initially, the results for finding the most significant categories are reported. Afterwards, the results for finding the most important features from the significant categories using a three-phase process are shown. Finally, classification results on the final feature set for categorizing applications into five classes are presented.

5.1. Significant Kernel Task Structure Category Identification. The feature extraction process results in the generation of a large feature set for each apk in the dataset. In order to identify important features, a feature selection approach depicted in Figure 4 is adopted. In the first step of feature selection, the set of all categories (CT^{all}) is analyzed to find the set of significant categories (CT^{sig}). For this purpose, a wrapper-based method of forward selection is used. The results of the first iteration of forward selection for selecting the most significant category are shown in Table 5.

It can be inferred from the results that the most significant category for malware categorization in terms of the F1-score for all malware classes is mem_info. Therefore, it is selected in the first iteration. All categories are then added for evaluation after sorting them by their individual F1-scores from the first iteration. The process of forward selection includes many iterations of feature combinations. The result of each step is not shown in the paper, as many results are intermediate and keep changing when new categories are added. An overall summary of important results of the forward selection process is shown in Figure 5. From the graph, it can be observed that the best performance is achieved by combining mem_info, process_credentials and signal_info categories. Combining these three categories helps in achieving an average F1-score of 0.95. It should also be noted that adding other categories does not significantly improve the performance of the system.

The application of forward selection for category analysis reduces the set of all categories CT^{all} to a set of three significant categories. This process reduces the overall size of

Table 4: Dataset details.

Application class	Number of samples	Number of memory dumps	Events generated	Total samples
Adware	500	4	0, 150, 1000, 4000	2,000
Banking Trojans	500	4	0, 150, 1000, 4000	2,000
Riskware	500	4	0, 150, 1000, 4000	2,000
SMS Trojans	500	4	0, 150, 1000, 4000	2,000
Benign	1000	4	0, 150, 1000, 4000	4,000

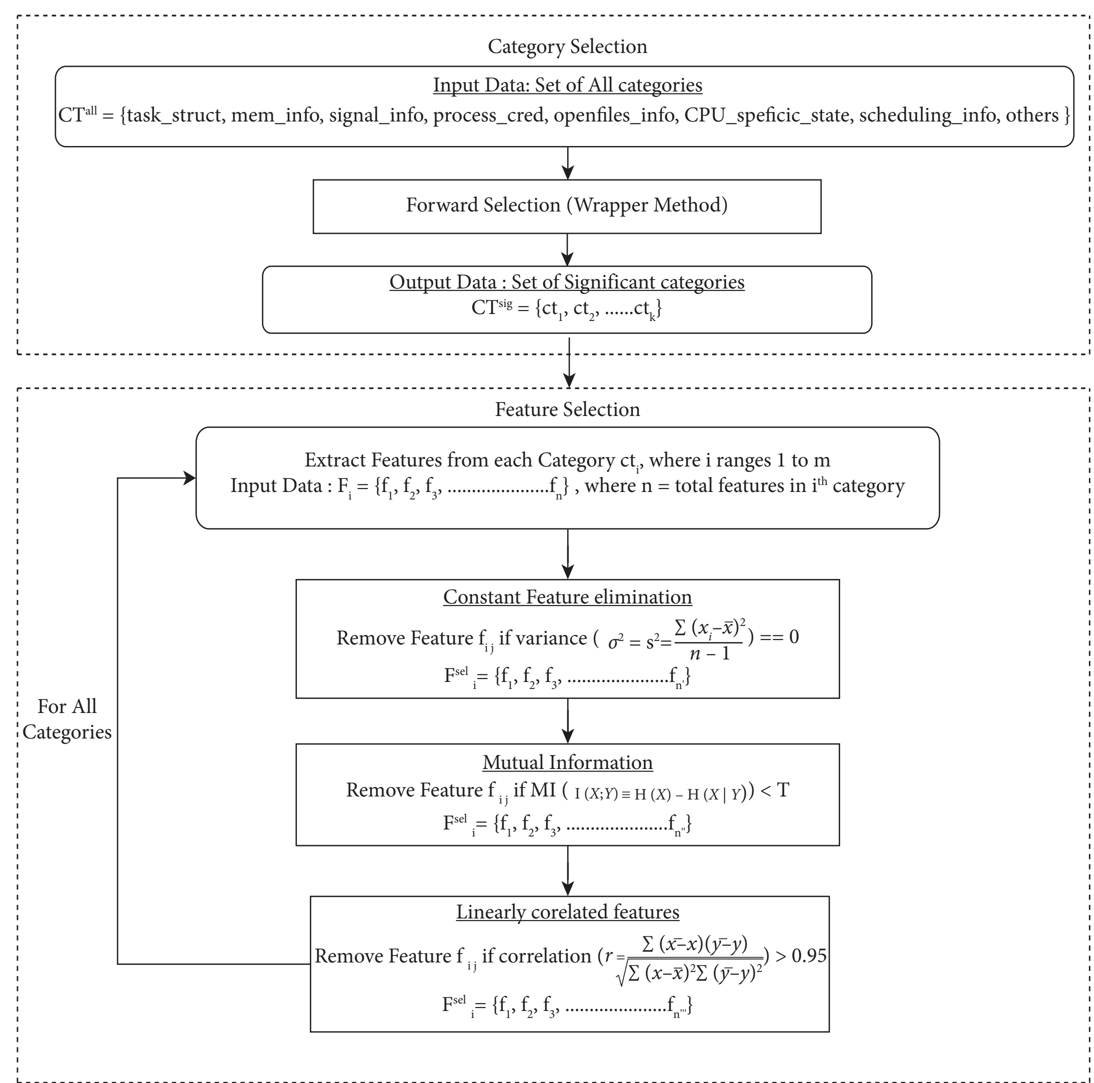
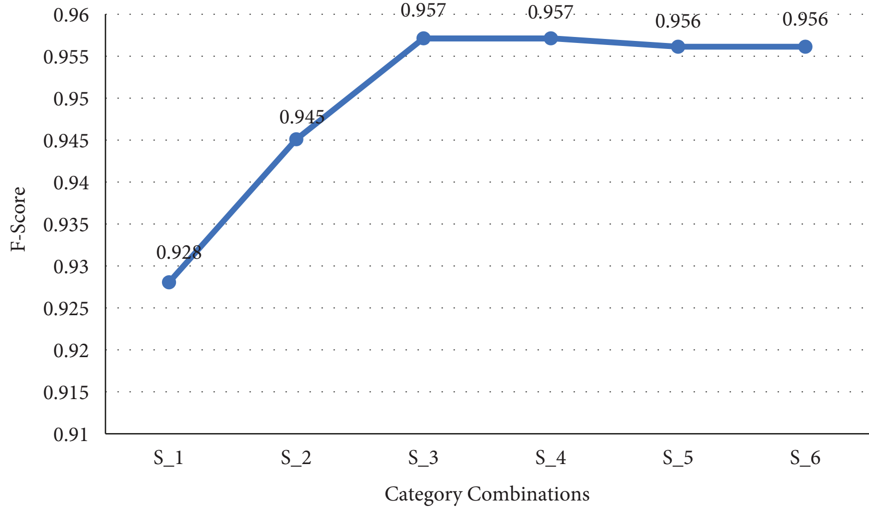


Figure 4: KTSDroid feature selection process.

Table 5: Results for finding the most significant KTS category using forward selection.

KTS category	F1-score				
	Adware	Banking Trojan	Riskware	Trojan SMS	Benign
task_state	0	0.3	0	0	0
mem_info	0.928	0.927	0.937	0.92	0.92
signal_info	0.768	0.891	0.775	0.834	0.761
process_credentials	0.926	0.912	0.939	0.904	0.963
Scheduling_info	0.597	0.797	0.503	0.70	0.55
IO_statistics	0.668	0.759	0.732	0.767	0.658
Openfile_info	0.531	0.665	0.616	0.758	0.686
CPU_state	0.843	0.756	0.699	0.789	0.702
Others	0.304	0.441	0.252	0.284	0.290



Combination Name	Categories in a combination
S-1	mem_info
S-2	mem_info and process_credentials,
S-3	mem_info, process_credentials and signal info,
S-4	mem_info, process_credentials, signal_info, and openfiles_info,
S-5	mem_info, process_credentials,signal_info, openfiles_info and IO_statsistics
S-6	mem_info, process_credentials, signal_info, openfiles_info and IO_statsistics

Figure 5: Forward selection results for significant category identification.

the dataset by 30 percent. The set of significant categories can now be defined as

$$\mathrm{CT}^{\mathrm{sig}} = \left\{ct_{\mathrm{mem_info}}, ct_{\mathrm{process_credentials}}, ct_{\mathrm{signal_info}}\right\}. \quad (11)$$

Further improvement in performance can be achieved by finding the most important and effective features from the selected categories, which are discussed in the next section.

5.2. Significant Feature Selection. In order to find significant features, the features in each of the selected categories are evaluated using a three-phase process. The results of each phase are discussed as follows.

5.2.1. Phase 1: Constant Feature Elimination. The first phase of significant feature selection focuses on constant feature evaluation and removal from the selected categories of mem_info, process_credentials, and signal_info. The set of nonconstant features for a category F_i^{sel} is found by using equations (1) and (2). The results of applying constant feature elimination on each category are shown in Table 6.

Table 6: Constant feature elimination results.

KTS category	Total features	Constant features	Remaining features
Mem_info	212	131	81
Process_credentials	75	34	47
Signal_info	83	27	56

The results show that a large number of constant features are present in all selected categories. It indicates that a substantial number of features show the same behavior for malicious and benign applications. It can be inferred that these features are indicative of the general working of the application and are not specific to the malicious actions. After removing these constant features, the size of the dataset is further reduced by 49 percent. Now, the remaining features will be gauged based on their mutual information values against the output class.

5.2.2. Phase 2: Mutual Information (MI) for Feature Selection. Mutual information measures the significance of a feature by estimating the dependency of output on the feature. After constant feature elimination, the features in each category are evaluated for mutual information scores by using equations (4) and (5). The results of mutual information against all features in the selected categories are shown in Figures 6–8.

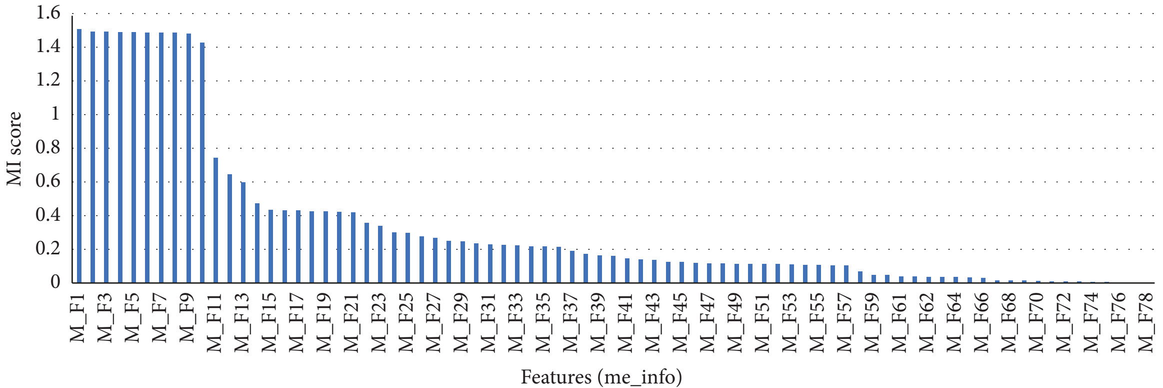


Figure 6: MI scores for mem_info features.

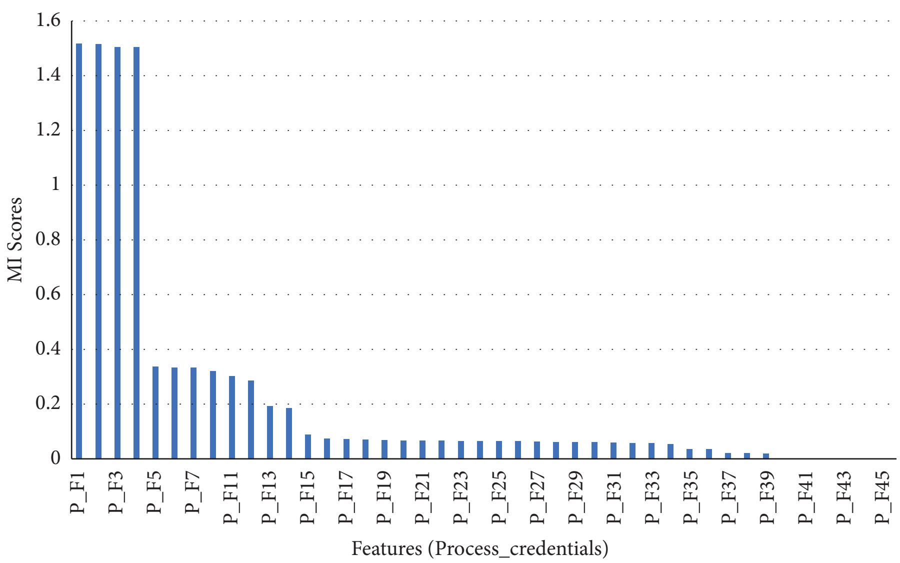


Figure 7: MI scores for process_credentials features.

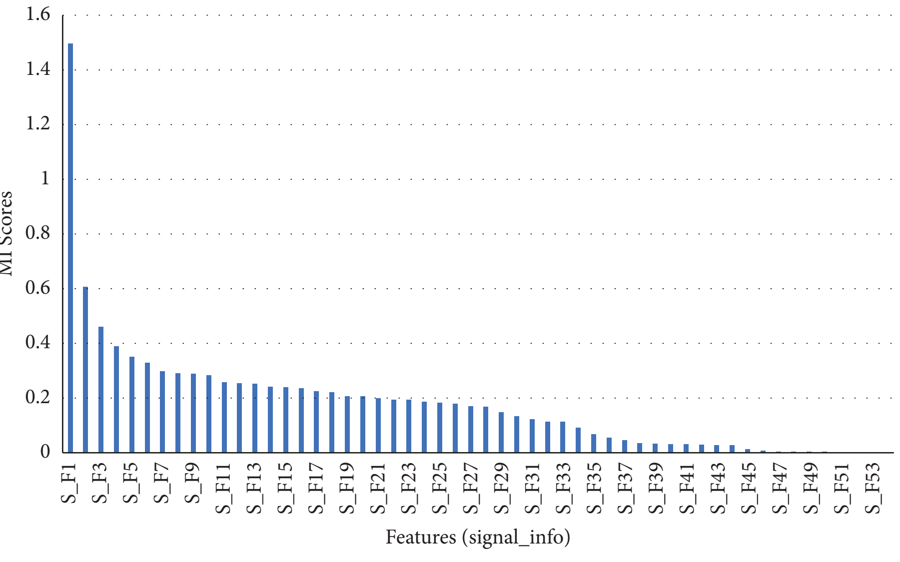


Figure 8: MI scores for signal_info features.

Table 7: Performance for MI thresholds.

Threshold for MI score (T)	Features in mem_info	Features in process_credentials	Features in signal_info	F1-score
0.0	81	47	47	0.942
0.1	56	14	33	0.956
0.2	35	12	19	0.964
0.3	24	11	6	0.974
0.4	21	4	3	0.987
0.5	13	4	2	0.980
0.6	12	4	1	0.980

From the results, it can be seen that many features have very low values for mutual information. It means that the dependency between these features and output is very less. In order to drop features with low MI scores, a step-by-step approach is used. Features are not dropped abruptly; instead, they are dropped based on a certain threshold value (T) of MI scores. Initially, the threshold (T) is set to 0.1. All features below the MI score of 0.1 are dropped, and performance in terms of the F1-score is measured. If there is improvement in performance, the threshold value T is updated by a factor of 0.1 and the process of performance measurement is repeated. The updation of the threshold value T is stopped until it reaches a value t at which the performance becomes constant.

The result of feature reduction at each threshold value and corresponding performance is shown in Table 7. It can be observed that at $T = 0.6$, the performance becomes constant, so the iterations for threshold updation can be stopped. Observing the performance against all thresholds reveals that the best performance is reported at a threshold value of 0.4. Therefore, the features at a threshold value of 0.4 are selected for further analysis. It can be observed that the number of features is now reduced to twenty-eight, of which twenty-one belong to mem_info, four belong to process_credentials, and three belong to signal_info. The names of selected features along with their MI scores are shown in Table 8.

5.3. Phase 3: Correlation for Feature Selection. The selection of features based on mutual information greatly reduces the size of the feature set. However, as two linearly correlated features have the same output, one of them can be dropped from analysis in order to reduce the dimensionality of the data. For this purpose, correlation is calculated for all features in a category. Two features are considered correlated if the value of the correlation coefficient as represented in Eqn. (9) is 0.95. The correlation matrix for all features in selected categories is shown in Figures 9–11. The number of correlated features and the size of the final set of features after the removal of correlated features is shown in Table 9.

The final set of features, after the removal of correlated features, now contains fourteen features in total. Table 10 shows the name, category, and depth of each feature in the kernel task structure.

5.4. Classification. The application of feature selection results in a reduced set of effective features. Features belonging to each category are now combined together and classified using random forest. In order to apply random forest, the length of each tree and the number of trees in the ensemble need to be assessed. KTSDroid uses unpurged trees to incorporate all features, as the feature set is now reduced to a manageable size. In order to estimate the number of trees inside the ensemble of random forest, the performance on the final feature set is evaluated for a different number of trees. The performance is evaluated for three, five, seven, nine, and eleven trees. It is observed that the highest average F1-score of 0.985 is reported for nine trees; therefore, the number of tress in the random forest ensemble is set to nine.

In order to ascertain the effectiveness of feature selection approaches, the performance is calculated at all steps of feature selection, i.e., forward selection (FS), constant feature elimination (CFE), mutual information (MI), and correlation (Corr). It can be observed from Figures 12 and 13 that each stage of feature selection results in the reduction of the feature set and the enhancement of performance. This highlights the preciseness of the proposed feature selection scheme represented in Figure 4.

One of the important contributions of the study was to evaluate the significance of kernel task structure features for multiclass classification for Android applications. For this purpose, the performance measures are analyzed for each class individually. Table 11 and Figure 14 show the performance of the final feature set for each class. The table shows that adware, riskware, and benign application types are classified by an F1-score of 0.99, and banking and SMS Trojans are classified by a 0.96 F1-score. The high rate of detection is proof of the effectiveness of memory-based solutions for Android malware categorization in general and kernel task structure-based features in particular.

KTSDroid is compared with two studies based on memory-based artifacts for Android malware analysis. Comparison is conducted in terms of the explored categories of the kernel task structure, number of features, number of output classes, and performance. The comparison shown in Table 12 highlights that KTSDroid has explored the maximum number of categories from the kernel task structure and reported an accuracy of 0.985 using the least number of features for five output classes.

Table 8: Selected features using mutual information (MI).

Rep	Feature name	MI score
M_F1	task -> mm -> mmap -> vm_file -> f_inode -> i_generation	1.51
M_F2	task -> mm -> mmap_base	1.50
M_F3	task -> mm -> brk	1.50
M_F4	task -> mm -> mmap_legacy_base	1.49
M_F5	task -> mm -> start_brk	1.49
M_F6	task -> mm -> end_data	1.49
M_F7	task -> mm -> start_code	1.49
M_F8	task -> mm -> start_data	1.49
M_F9	task -> mm -> end_code	1.48
M_F10	task -> mm -> mmap -> vm_file -> f_inode -> i_ino	1.43
M_F11	task -> mm -> shared_vm	0.74
M_F12	task -> mm -> total_vm	0.64
M_F13	task -> mm -> hiwater_vm	0.59
M_F14	task -> mm -> exec_vm	0.47
M_F15	task -> mm -> env_end	0.43
M_F16	task -> mm -> start_stack	0.43
M_F17	task -> mm -> arg_end	0.42
M_F18	task -> mm -> arg_start	0.42
M_F19	task -> mm -> env_start	0.42
M_F20	task -> mm -> highest_vm_end	0.41
M_F21	task -> mm -> mm_count -> counter	0.41
P_F1	task -> cred -> session_keyring -> last_used_at	1.51
P_F2	task -> real_cred -> session_keyring -> last_used_at	1.51
P_F3	task -> real_cred -> session_keyring -> serial	1.50
P_F4	task -> cred -> session_keyring -> serial	1.50
S_F1	task -> sas_ss_sp	1.49
S_F2	task -> signal -> ioac -> rchar	0.60
S_F3	task -> signal -> ioac -> wchar	0.46
S_F4	task -> signal -> real_timer -> base -> cpu_base -> clock_was_set_seq	0.38

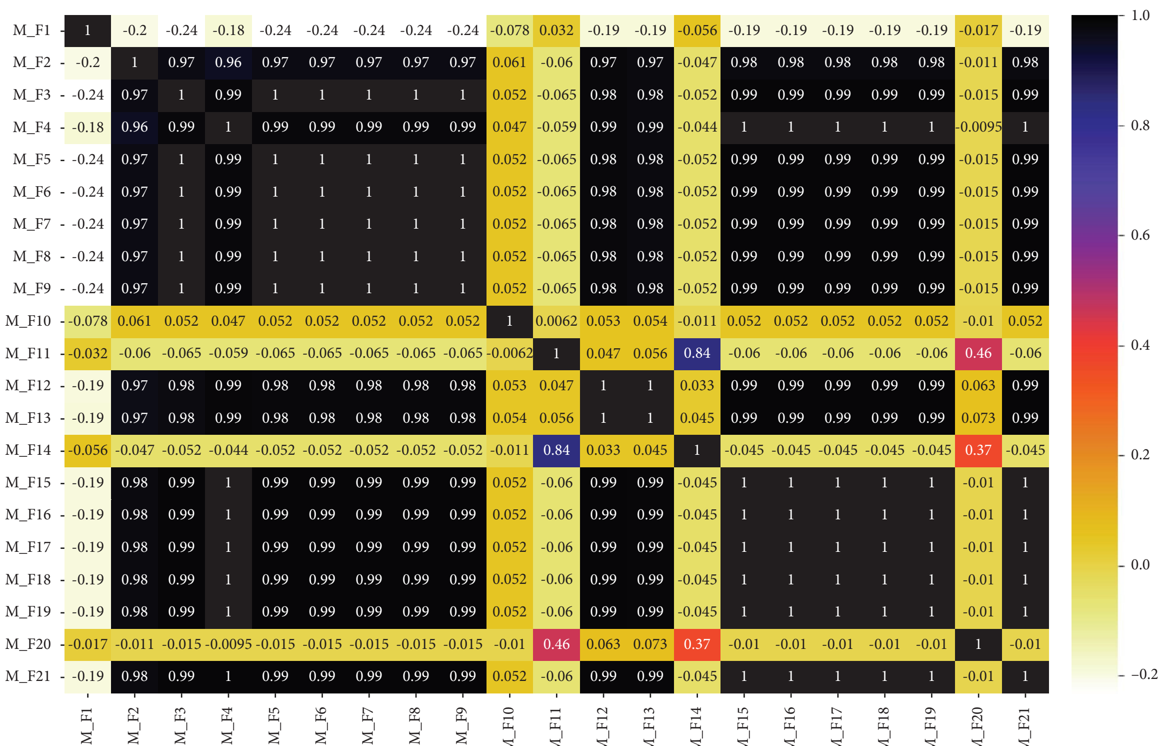


Figure 9: Correlation matrix of mem_info features.

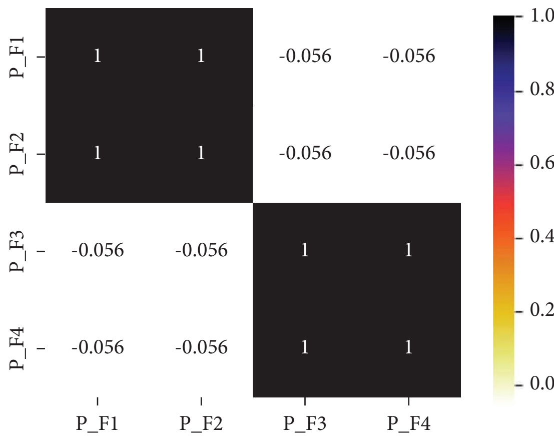


Figure 10: Correlation matrix of process_cred features.

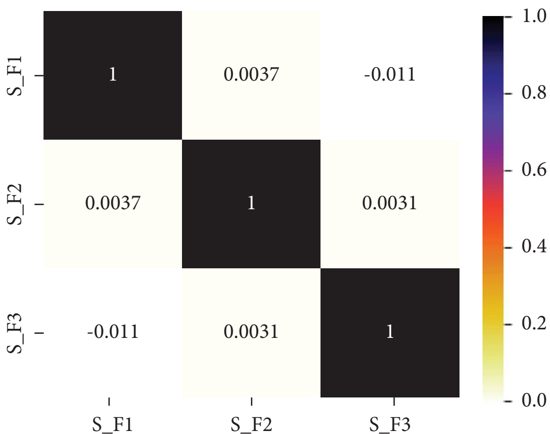


Figure 11: Correlation matrix of signal_info features.

Table 9: Correlation results.

KTS category	Original features	Correlated features	Reduced features
mem_info	21	12	9
process_credentials	2	4	2
signal_info	0	3	3

Table 10: Final feature set used by KTSDroid.

Rep	Feature name	Depth
M_F1	task -> mm -> mmap -> vm_file -> f_inode -> i_generation	6
M_F2	task -> mm -> mmap_legacy_base	3
M_F3	task -> mm -> end_data	4
M_F4	task -> mm -> mmap_base	3
M_F10	task -> mm -> mmap -> vm_file -> f_inode -> i_ino	6
M_F11	task -> mm -> shared_vm	3
M_F12	task -> mm -> total_vm	3
M_F14	task -> mm -> exec_vm	3
M_F20	task -> mm -> mm_count -> counter	4
P_F1	task -> real_cred -> session_keyring -> last_used_at	4
P_F3	task -> real_cred -> session_keyring -> serial	4
S_F1	task -> sas_ss_sp	2
S_F2	task -> signal -> ioac -> rchar	4
S_F3	task -> signal -> ioac -> wchar	4

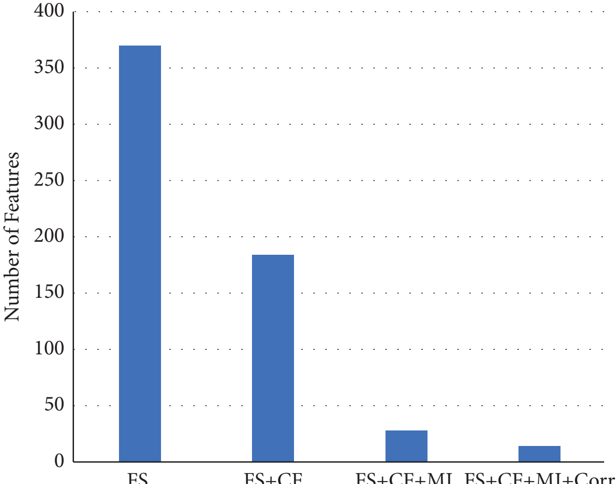


Figure 12: Dimensionality reduction by feature selection methods.

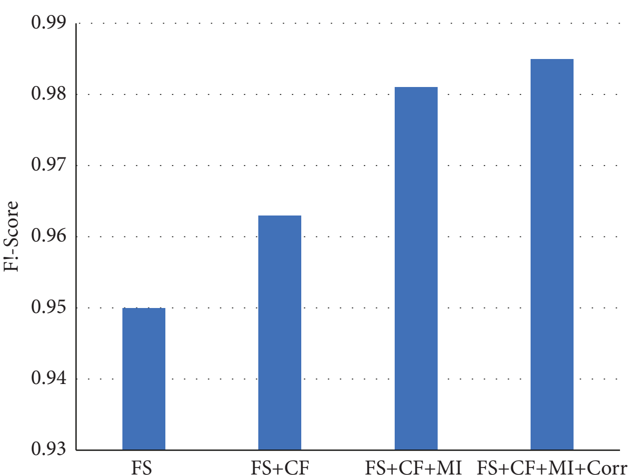


Figure 13: Performance improvements by feature selection methods.

Table 11: KTSDroid performance for malicious and benign classes.

Class	F1-score	Precision	Recall
Adware	0.992	0.988	0.996
Banking Trojans	0.967	0.972	0.962
Riskware	0.992	0.989	0.995
SMS Trojans	0.968	0.969	0.968
Benign	0.993	0.994	0.992

6. Discussion

KTSDroid analyzes the effect of memory-based features on Android malware categorization. The kernel task structure of memory is used for the extraction of process-specific features. It is thoroughly analyzed for nine categories up to a depth of six levels, as compared to existing studies that have worked with five categories for a depth of three. A large number of features are extracted, which are then evaluated for significance. KTSDroid uses a minimal set of fourteen features and is able to classify malicious applications with

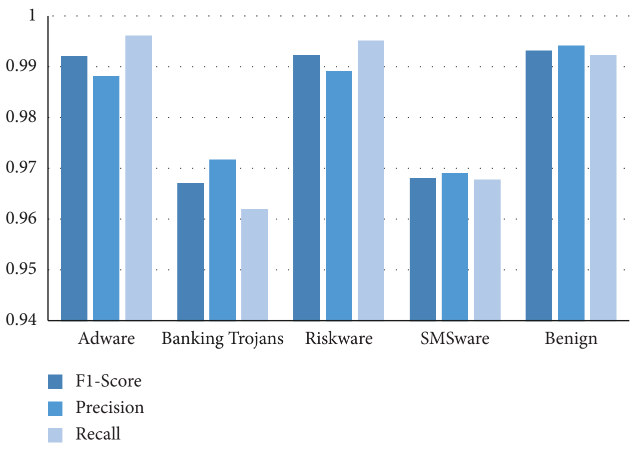


Figure 14: KTSDroid performance for multiclass application categorization.

Table 12: KTSDroid comparison with existing studies.

	Significant categories	Number of features	Number of output classes	Performance
Wang et al.	2 out of 5	40	2	0.98 ($F1$-score)
Tstructdroid	—	32	2	98 (accuracy)
KTSDroid	3 out of 9	14	5	0.985 ($F1$-score)

high performance. The high performance of KTSDroid can be attributed to the following important points:

(1) Process-specific features from the kernel task structure are used for creating behavior profiles for applications. These features are better representative of the application's behavior as compared to general memory usage features, as they are shared by a number of processes.

(2) Five additional categories of the kernel task structure are explored for feature extraction by KTSDroid. Among these, the category of process_credentials is found to be the second most important for malware categorization by the forward selection method, as shown in Table 5. Features from the category of process_credentials are included in the final feature set.

(3) The deep exploration of the kernel task structure enables the extraction of features beyond the depth of three (as per previous studies). The features beyond the depth of three constitute seventy-one percent of the final feature set. The high percentage of features from deeper structures of the kernel task structure highlights the importance of traversing deeper levels of the kernel task structure.

7. Conclusion

Dynamic analysis-based solutions for malware analysis have replaced static analysis solutions due to the inability of static analysis to explore the runtime working of the application. This study has proposed a dynamic analysis-based malware categorization system that extracts volatile memory-based

artifacts for malicious Android application detection and categorization. A time-based memory dump extraction process with interactions is conducted to ensure the capture of malicious actions of the applications. The kernel task structure from all memory dumps is analyzed for the extraction of process-specific features. A large number of process-specific features grouped into nine categories are extracted. A comprehensive analysis is conducted on the extracted set of features to find the most important categories of the kernel task structure for malware categorization. The most significant features of the selected categories are also reported in the study. The proposed system is able to classify malicious applications into five distinct classes by using a small number of features with high performance.

Data Availability

The data supporting the findings of this study are available on the following git repository: https://github.com/saneehaAmir/KTSDroid.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] P. Stirparo, I. N. Fovino, and I. Kounelis, "Data-in-use leakages from Android memory — test and analysis," in *Proceedings of the 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 701–708, Lyon, France, October 2013.

[2] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *Proceedings of the The 5th Conference on Information and Knowledge Technology*, pp. 113–120, Shiraz, Iran, May 2013.

[3] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying Android malware using dynamically obtained features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.

[4] Y. Ding, M. Naber, C. L. E. Paffen, J. H. Fabius, and S. Van der Stigchel, "Saccades reset the priority of visual information to access awareness," *Vision Research*, vol. 173, pp. 1–6, 2020.

[5] A. Aghamohammadi and F. Faghih, "Lightweight versus obfuscation-resilient malware detection in android applications," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 2, pp. 125–139, 2020.

[6] M. Hammad, J. Garcia, and S. Malek, "A large-scale empirical study on the effects of code obfuscations on android apps and anti-malware products," in *Proceedings of the Proceedings of the 40th International Conference on Software Engineering*, pp. 421–431, Association for Computing Machinery, New York, NY, USA, May 2018.

[7] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *Proceedings of the 2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 31–38, Fajardo, PR, USA, October 2017.

[8] M. Gohari, S. Hashemi, and L. Abdi, "Android malware detection and classification based on network traffic using deep learning," in *Proceedings of the 2021 7th International Conference on Web Research (ICWR)*, pp. 71–77, Tehran, Iran, May 2021.

[9] H. Gao, S. Cheng, and W. G. D. Zhang, "GDroid: android malware detection and classification with graph convolutional network," *Computers & Security*, vol. 106, Article ID 102264, 2021.

[10] A. S. Bozkir, E. Tahillioglu, M. Aydos, and I. Kara, "Catch them alive: a malware detection approach through memory forensics, manifold learning and computer vision," *Computers & Security*, vol. 103, Article ID 102166, 2021.

[11] H. Wang, H. He, and W. Zhang, "Demadroid: object reference graph-based malware detection in Android," *Security and Communication Networks*, vol. 2018, Article ID 7064131, 16 pages, 2018.

[12] X. Wang and C. Li, "Android malware detection through machine learning on kernel task structures," *Neurocomputing*, vol. 435, pp. 126–150, 2021.

[13] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: detecting android malware by ORGB analysis," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55–69, 2020.

[14] H. Alawneh, D. Umphress, and A. Skjellum, "Android malware detection using neural networks & process control block information," in *Proceedings of the 2019 14th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 3–12, Nantucket, MA, USA, August 2019.

[15] F. Shahzad, M. Akbar, S. Khan, and M. Farooq, "Tstructdroid: realtime malware detection using in-execution dynamic analysis of kernel process control blocks on android," Technical Report, National University of Computer & Emerging Sciences, Islamabad, Pakistan, 2013.

[16] A. Ali-Gombe, A. Tambaoan, A. Gurfolino, and G. G. Richard III, "App-agnostic post-execution semantic analysis of Android in-memory forensics artifacts," in *Proceedings of the Annual Computer Security Applications Conference*, pp. 28–41, Austin, TX, USA, December 2020.

[17] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

[18] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone, "Visualizing the outcome of dynamic analysis of Android malware with VizMal," *Journal of Information Security and Applications*, vol. 50, Article ID 102423, 2020.

[19] A. H. Lashkari, B. Li, T. L. Carrier, and G. V. Kaur, "Volatile memory analyzer for malware classification using feature engineering," in *Proceedings of the 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pp. 1–8, IEEE, Hamilton, Canada, May 2021.

[20] H. H. Kim and M. J. Choi, "Linux kernel-based feature selection for Android malware detection," in *Proceedings of the The 16th Asia-Pacific Network Operations and Management Symposium*, pp. 1–4, Hsinchu, Taiwan, September 2014.

[21] J. Abawajy, A. Darem, and A. A. Alhashmi, "Feature subset selection for malware detection in smart IoT platforms," *Sensors*, vol. 21, no. 4, p. 1374, 2021.

[22] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[23] N. Maleki and H. Rastegari, "An improved method for packed malware detection using PE header and section table

information," *International Journal of Computer Network and Information Security*, vol. 11, no. 9, 2019.

[24] J. Jung, H. Kim, D. Shin et al., "Android malware detection based on useful API calls and machine learning," in *Proceedings of the 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 175–178, IEEE, Laguna Hills, CA, USA, September 2018.

[25] P. Agrawal and B. Trivedi, "Machine learning classifiers for Android malware detection," in *Data Management, Analytics and Innovation*, pp. 311–322, Springer, Berlin, Germany, 2021.

[26] H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi, and L. Cheng, "HEMD: a highly efficient random forest-based malware detection framework for Android," *Neural Computing & Applications*, vol. 30, no. 11, pp. 3353–3361, 2018.

[27] S. Khalid and F. B. Hussain, "Evaluating dynamic analysis features for android malware categorization," in *Proceedings of the 2022 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 401–406, IEEE, Dubrovnik, Croatia, May 2022.

[28] T. A. Alhaj, M. M. Siraj, A. Zainal, H. T. Elshoush, and F. Elhaj, "Feature selection using information gain for improved structural-based alert correlation," *PLoS One*, vol. 11, 2016.

[29] A. Salah, E. Shalabi, and W. Khedr, "A lightweight android malware classifier using novel feature selection methods," *Symmetry*, vol. 12, no. 5, p. 858, 2020.

[30] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, vol. 2, Springer, Berlin, Germany, 2009.

[31] A. M. Kowshalya, R. Madhumathi, and N. Gopika, "Correlation based feature selection algorithms for varying datasets of different dimensionality," *Wireless Personal Communications*, vol. 108, no. 3, pp. 1977–1993, 2019.

[32] X. F. Song, Y. Zhang, D. W. Gong, and X. Z. Gao, "A fast hybrid feature selection based on correlation-guided clustering and particle swarm optimization for high-dimensional data," *IEEE Transactions on Cybernetics*, vol. 52, no. 9, pp. 9573–9586, 2022.

[33] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[34] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018.

[35] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," *Journal of Network and Systems Management*, vol. 30, pp. 22–34, 2022.