INTERNATIONAL JOURNAL OF COMMUNICATION NETWORKS AND INFORMATIO SECURITY
2024, 16(S4)
ISSN: 2073-607X,2076-0930
https://ijcnis.org/

Research Article

Android Malware Detection and Classification Using Machine Learning Algorithm

A.Sonya[1], and R.Ram Deepak[2]

[1]Department of Information Technology, B. S. Abdur Rahman Crescent Institute of Science and Technology, Tamil Nadu, India. Sonya.yasmin152@gmail.com

[2]Department of Information Technology, B. S. Abdur Rahman Crescent Institute of Science and Technology, Tamil Nadu, India. Ramdeepak2311@gmail.com

ARTICLE INFO

Received: 8 Aug 2024
Accepted: 10 Sep 2024

ABSTRACT

The cyber security approach that is being offered in this project addresses the growing dangers that rogue applications that target mobile devices are posting. With several forms of malware, such adware, spyware, and ransomware, multiplying quickly, these threats to Android users throughout the world are getting worse. This paper aims to create a thorough classification framework that uses both static and dynamic information to detect Android malware effectively in response. Our method seeks to provide a comprehensive understanding of malware traits and actions by fusing dynamic runtime behavior monitoring with static analysis of APK files. To develop a powerful classification model that can reliably classify various kinds of Android malware by utilizing machine learning algorithms such as Gradient Boosted Trees (GBT) and Ridge Classifier. APK files' metadata, permissions, and code structure are extracted using static analysis, but runtime behaviors including API calls, network traffic, and system interactions are captured using dynamic analysis. Our suggested methodology shows promising results in terms of categorization accuracy, precision, recall, and F1-score after comprehensive testing and evaluation on real-world Android malware datasets. A thorough understanding of malware behavior is made possible by the combination of static and dynamic features, which makes proactive threat detection and mitigation techniques in mobile security easier to implement. Persistently exploiting gullible people with false links, URL phishing is a cyber threat that can result in financial loss, theft of identities, and data breaches. The objective of this work is to create and deploy a strong defense against URL phishing assaults and mobile security procedures against new and emerging Android malware vulnerabilities.

***INDEX TERMS*—Android, Malware, Dynamic features, Machine Learning with Classification**

I. INTRODUCTION

The widespread use of smart phones, especially those running the Android operating system, has completely changed how people interact, work, and access information. A part of the larger cyber security scene, mobile malware has become a major threat vector, affecting millions of Android users globally with various malicious software classes like ransomware, spyware, and adware. The rapid growth of mobile malware presents a variety of difficulties for individuals, institutions, and the entire mobile ecosystem. Mobile devices, in contrast to traditional desktop environments, frequently function in extremely dynamic and networked contexts, which increase the potential

impact of malware infestations. Furthermore, it is more difficult to adequately detect and prevent emerging threats due to the inherent complexity of mobile platforms and the variety of Android device setups and software versions.

The evolving threat landscape has encouraged academics and cyber security professionals to step up their efforts in creating novel methods for identifying, categorizing, and reducing the effects of mobile malware. The primary aspect of malware analysis is static analysis, which looks at executable files without running them. It usually focuses on properties like code structure, permissions, and metadata. Even though static analysis can reveal important information about the static characteristics of malware, it may not be able to detect dynamic behaviours or avoid detection systems that are meant to resist static analysis methods.

On the contrary hand, dynamic analysis provides an alternative viewpoint by observing the runtime actions of software as it is being executed. Dynamic analysis has the ability to reveal harmful behaviours that can escape static analysis or only surface in operational situations by recording activities including API calls, communication over the network, and system interactions. Dynamic analysis, however, could require more computing overhead and would not always be possible in settings with limited resources.

Aiming to make an improvement to this dynamic environment, this study offers an extensive categorization scheme for Android malware identification. In order to obtain an in-depth knowledge of malware characteristics and behaviours, our method makes use of both static and dynamic information. This allows for more precise classification and proactive threat mitigation tactics. By integrating machine learning algorithms such as Gradient Boosted Trees (GBT) and the Ridge Classifier, interested in creating a strong classification model that can correctly identify different kinds of Android malware.

Further, to address the growing threat of phishing attempts targeting Android users by integrating URL phishing detection into our framework. To improve the detection and mitigation of phishing risks using our framework by employing methods like heuristic analysis and machine learning to examine URLs that have been collected from Android applications. Our goal is to safeguard the integrity and privacy of Android users globally while also promoting mobile security practices through the usage of this entirety strategy.

A. MACHINE LEARNING

In particular, machine learning has proven to be a successful approach for managing the complicated issues that mobile malware presents when it comes to Android security. Discovering new and polymorphic varieties of malware has grown more difficult for conventional signature-based detection techniques due to the swift growth and variety of malware threats. Algorithms for machine learning, on the other conjunction, provide a data-driven method of detecting malware since they can find trends and anomalies that point to fraudulent activity by learning from big datasets.

The capability of machine learning to evaluate various feature sets derived from mobile applications is one of the most significant advantages of using it for Android virus detection. These traits cover the application's code structure, permissions, runtime behaviours, metadata, and API calls, among other things. Machine learning algorithms can recognize small variations between dangerous and benign programs by looking at those characteristics, which makes it possible to identify dangers that were previously undiscovered.

There are numerous approaches included in machine learning models for Android malware detection, and each has advantages and drawbacks of its own. Because ensemble techniques like Random Forests, Gradient Boosting Machines (GBM), and AdaBoost may combine several weak learners to create a robust classifier, they are often used. Deep learning methods have also demonstrated promise in automatically learning tiered representations of raw data, such as opcode sequences or image-based depictions of mobile applications for Android. These methods include Convolution Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

In order to teach an algorithm the underlying patterns linked to each class, machine learning models are trained by feeding labelled datasets containing both benign and malicious samples into the system. In this approach, feature engineering and selection are essential because they identify the data's most valuable features for identifying dangerous and benign applications. Model performance is enhanced and the learning process is streamlined by employing strategies including feature importance analysis, feature scaling, and dimensionality reduction.

After it has been trained, the machine learning model is rigorously assessed using a range of performance metrics, such as F1-score, accuracy, precision, and recall. Cross-validation methods, such k-fold cross-validation, are frequently used to evaluate the robustness and generalization ability of the model over various dataset subsets. Furthermore, testing the model on real-world datasets with previously encountered malware samples offers insightful information on how well it works in real-world situations.

Machine learning models for Android malware detection can be deployed by integrating them with network-based intrusion detection systems or mobile antivirus apps that are already part of the security architecture. The model can remain highly accurate in detecting dangers over time by adjusting to new threats through ongoing monitoring and feedback mechanisms. Additionally, with little assistance from humans, the model's performance can be enhanced by methods like active learning, in which it actively chooses the most instructive samples for labelling.

In conclusion, Machine Learning presents a viable method for detecting Android malware by using data-driven approaches to instantly recognize and neutralize new threats. Machine learning models have the potential to

improve mobile security procedures and shield consumers from the increasingly dangerous threat of mobile malware by evaluating various feature sets and utilizing strong classification algorithms. Figure 1. represented the Machine Learning process.

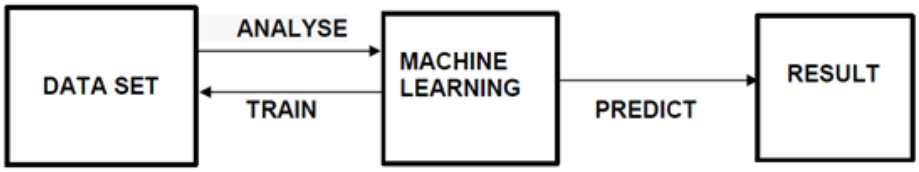

Figure 1. Machine Learning Process

B. MACHINE LEARNING CLASSIFICATION

(i) Supervised Learning

One kind of machine learning technique that uses labelled data for learning is called supervised learning. Data that has been assigned a right answer or categorization is known as labelled data. As the name suggests, supervised learning involves having a supervisor in addition to a teacher. When we use well-labelled data to instruct or train a machine, this is known as supervised learning. This indicates that the right response has already been applied to some data. In order for the supervised learning algorithm to analyse the training data (set of training examples) and provide an accurate result from the labelled information, the machine is then given a fresh set of instances (data).

Ridge classifier

The kind of machine learning known as supervised learning trains the model with labelled data. Classification and Regression are under the umbrella of Supervised Learning.

Regression: This method uses one or more features to predict a continuous range of values. The result of the anticipated value serves as the dependent variable, while these characteristics serve as the independent variables.

Classification: The process of classification involves grouping the dataset into distinct categories according to its attributes.

Ridge Regression

The regularization term is added as a parameter in Ridge Regression, a kind of Linear Regression. L2 regularization is another name for this regularization concept. This is carried out to prevent overfitting. When a model performs well on training data but poorly on test or unseen data, this is known as overfitting. Regularization reduces loss and overfitting in the model by introducing penalties on higher terms. The following is the Ridge Regression's cost formula.

$$C = MSE + \alpha \sum_1^N w_i^2$$

The Ridge Classifier employs a loss function similar to mean squared loss, just as Ridge Regression. The alpha parameter controls the regularization strength and the way the penalty impacts the model coefficients. For multiclass classification, regression-based techniques are used to determine the decision boundaries that, in practice, separate different classes. In its most basic version, the Ridge Classifier combines elements of regression and classification to offer a dependable and consistent solution to difficult classification issues.

Gradient Boosting

With gradient descent, each new model is trained in order to reduce the loss function, such as the average squared error or crossover entropy of the preceding model. Gradient boosting is a potent boosting procedure that turns multiple weak learners into strong learners. The approach calculates the gradient of the loss function in relation to the current ensemble's predictions for each iteration, and then trains a new weak model to minimize this gradient. Next, the new model's predictions are included in the ensemble, and the procedure is continued until a stopping requirement is satisfied.

The group comprises M trees. The labels y and the feature matrix X are used to train Tree1. The training set residual errors, r1, are found using the predictions labelled y1(hat). After that, Tree2 is trained with the labels from Tree1's residual errors, r1, and the feature matrix X. The residual r2 is then ascertained using the anticipated outcomes, r1(hat). Until all M trees in the ensemble are trained, the procedure is repeated. This method makes use of a crucial parameter called shrinkage. The term "shrinkage" describes how each ensemble tree's forecast shrinks after being multiplied by the learning rate (eta), which has a range of 0 to 1. The number of estimators and eta have a

trade-off; in order to achieve a certain level of model performance, a decrease in learning rate must be made up for by an increase in estimators. Predictions are possible now that every tree has been taught. Every tree makes a label prediction, and the formula provides the ultimate forecast.

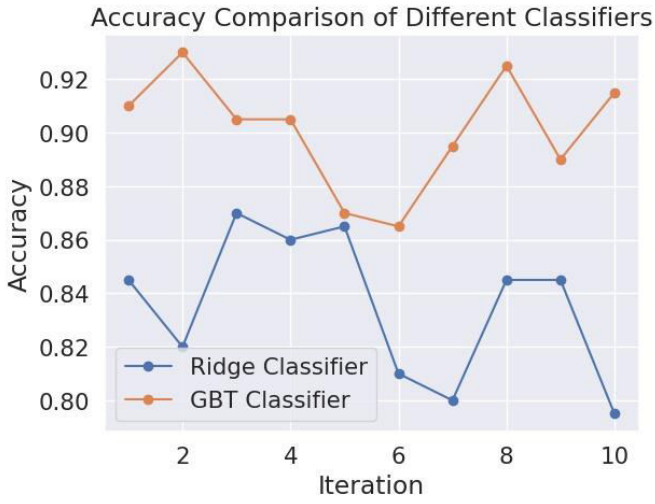

Figure 2: Accuracy Comparison of Different Classifiers

(ii) Unsupervised Learning

Machine learning that derives its knowledge from unlabelled data is known as unsupervised learning. This indicates that there are no pre-existing labels or categories in the data. Finding patterns and relationships in the data without explicit instruction is the aim of unsupervised learning. When a computer is trained using data that is neither categorized nor labelled, the algorithm is allowed to operate on the data without supervision. This is known as unsupervised learning. In this case, the machine's job is to categorize unsorted data based on similarities, patterns, and differences without requiring any prior data training.

In contrast to supervised learning, the absence of a teacher implies that the machine will not receive any instruction. Consequently, the machine's ability to independently identify the hidden structure in unlabelled data is limited. Unsupervised learning can be used to analyse collected animal data and make distinctions between many groups based on the characteristics and behaviours of the animals. These clusters may represent different animal species, enabling you to classify the animals without relying on pre-existing labels.

(iii) Reinforcement Learning

Reinforcement learning is a sub domain within machine learning. It entails behaving sensibly in a given situation to optimize reward. The main distinction between supervised and reinforcement learning is that in the former, the model is trained with the right answer pre-existing in the training set, whereas in the latter, the model is trained in the absence of an answer and is instead directed by the reinforcement agent's judgment regarding how to accomplish the given task. It will eventually learn from its encounters even in the absence of a training dataset.

Reinforcement learning (RL) is the name of the field that studies decision-making. It entails determining how to act in a certain situation to optimize benefits. The data used in reinforcement learning (RL) is sourced from machine learning systems that use trial-and-error techniques. In machine learning, data is not an input source, whether it is supervised or unsupervised.

Reward-learning algorithms are used to understand data and choose the optimal course of action. In order to determine if a judgment it made was neutral, incorrect, or correct, the algorithm receives feedback after each step. For automated systems that must make a lot of little decisions without human support, it's a helpful technique.

Reinforcement learning is a self-governing, self-teaching system that essentially gains knowledge via error. In other words, it acts with the goal of optimizing rewards and learns by doing in order to get the best outcomes.

II. LITERATURE OVERVIEW

Dong-Ok Won; Yong-Nam Jang; Seong-Whan Lee have proposed that the field of cyber security is always changing, and because zero-day malware takes use of undiscovered software flaws, it poses a serious threat. For the protection of systems, identifying and eliminating such malware is essential. A common problem of traditional techniques is the absence of representative and diverse labeled datasets. PlausMal-GAN is an excellent tool for generating a wide

variety of high-quality zero-day malware samples, which increases the detection systems' resilience. Several GAN models are used to evaluate the framework, and the results demonstrate consistent accuracy in identifying and forecasting instances of identical malware. This method improves zero-day malware detection in a novel way, which greatly advances cyber security. By using GANs to produce similar malware data, it improves the efficacy and resilience of detection systems, protecting people and companies from ever-changing cyber threats.

Siwon Huh; Seonghwan Cho; Jinho Choi; Seungwon Shin; Hojoon Lee have proposed that It's critical to remain ahead of emerging dangers in the continuous combat against malware. We provide a thorough examination of malware dissemination in our study, using a dataset of 99,312 samples gathered over 287 days from 38,659 distribution locations. We investigate the properties of distributed binaries, grouping them into families and using clustering to find patterns that reveal common variants and ways of dissemination. By analyzing distribution sites, we can gain information into the infrastructure behind harmful activity by looking at things like IP addresses, domains, and AS registration. Adversary strategies are revealed through statistical analysis of malware families and distribution domains. Our data shows how malware is now spreading and highlights how tactics are changing. In order to inform proactive security tactics, we provide insights into the current distribution landscape and provide directions for future research and intervention. We can strengthen our defenses against new attacks and maintain safe computing systems and networks by understanding the subtleties of distribution.

Ibrahim Gulatas;H. Hakan Kilinc;A. Halim Zaim;M. Ali Aydin have proposed a result of the large number of devices, particularly the susceptible Internet of Things (IoT) devices that are the focus of malware, Edge and Fog computing environments must be secured. The system Quality of Service is greatly impacted by these attacks. The lack of available malware samples and our inadequate understanding of Edge and Fog computing make it difficult to comprehend these dangers. In order to solve these issues, this study examines 64 families of IoT malware from 2008 to October 2022. The analysis uses research frameworks such as the "Cyber Kill Chain" and "Mitre ATT&CK for ICS" to cover a variety of topics, including target architecture, delivery mechanisms, and attack vectors. The work contributes to the development of strong protection mechanisms and proactive security measures that are suited to Edge and Fog computing settings by illuminating the characteristics of IoT malware.

S. Abijah Roseline; S. Geetha; Seifedine Kadry; Yunyoung Nam have proposed a work offers a fresh strategy to counter the malware threat that is always changing in today's computing environment. The strategies used by malware authors are constantly evolving, making it difficult for traditional detection technologies to stay up. Conventional techniques, such as approaches based on static signatures and dynamic behavior, have drawbacks, especially when dealing with complex variations. In order to solve this, the suggested technology converts malware samples into two-dimensional images, making it possible to capture minute patterns that are difficult to see using conventional techniques. The system attains enhanced performance without the complications associated with conventional deep learning models by utilizing a layered ensemble method that draws inspiration from deep learning. By employing Convolution Neural Networks (CNNs) to derive significant features from these visual depictions, the system educates a group of classifiers, each with a specialized role in identifying particular kinds of malware. Compared to using separate classifiers, the system detects malware with more accuracy and robustness when the predictions from multiple classifiers are combined.

Huyen N. Nguyen; Faranak Abri; Vung Pham; Moitrayee Chatterjee; Akbar Siami Namin; Tommy Dang have proposed a Computer systems are significantly threatened by malicious software, necessitating the use of efficient analysis techniques for both detection and mitigation. Static and dynamic analysis is two main approaches to malware investigation.In order to detect known malicious payloads, static analysis entails looking at malicious software while it is not in use and finding patterns and signatures inside the code. On the other hand, dynamic analysis necessitates running the malware in order to watch how it behaves and interacts with system resources, network connectivity, and file activities. Both methods have advantages, but they also have drawbacks. MalView is a tool that helps with malware categorization, payload identification, and temporal dependency exploration by applying pattern matching algorithms to artifacts generated by both static and dynamic analysis. By providing thorough insights into malware artifacts and facilitating interactive investigation of malware behavior, MalView improves malware analysis. In conclusion, MalView is a useful tool that combines both static and dynamic analysis methods with user-friendly visualization features for malware analysts. Through the use of interactive visualization and pattern matching, MalView gives security experts the ability to better comprehend and counteract emerging cyber threats.

Xiaojian Liu; Xi Du;Qian Lei; Kehong Liu have proposed a One significant development in our understanding of the behavior patterns of different malware families is the multifamily classification of Android malware. Multifamily classification offers more in-depth understanding of the workings of many malware families than binary classification, which merely classifies samples as harmful or benign. But this work is not without its difficulties:

identifying various patterns of behavior and defeating evasion strategies like code obfuscation. The authors of the article present a novel method for characterizing malware behavior patterns using regular expressions of callbacks. First, they capture changes in behavior within the same family by transforming the original regular phrases into fuzzy ones with wider meanings. Secondly, they strengthen the robustness against obfuscation and polymorphic variants by introducing similarity measurements between regular expressions for fuzzy matching. The authors use text mining techniques to train a multifamily 1-NN classifier on a dataset of 3,270 samples from 65 malware families by using these fuzzy regular expressions. According to experimental data, their technology accurately classifies multifamily Android malware, outperforming numerous state-of-the-art approaches. This work provides a strong paradigm for multifamily categorization that addresses issues such as polymorphism and code obfuscation, which makes a substantial contribution to malware analysis. The suggested method provides a more accurate and robust way to identify and classify different malware families by utilizing fuzzy processing and regular expressions. The results of the trial confirm its effectiveness and point to the possibility of enhancing Android malware prevention and detection in practical settings.

III. RELATED WORKS

In recent years, several techniques for identifying malware have been developed. These techniques rely on the study of either static or dynamic behavior including register values, opcodes, API calls, and other behavioral modeling elements. To stop or identify Android malware, numerous machine learning techniques have been created. A comparison of relevant work using various machine learning algorithms is provided in Table 1, and some related work is discussed in the following subsections. It should be noted that these algorithms assign a score to each attribute on its own. Rather, in order to identify Android malware, this work depends on features existing simultaneously. As far as we are aware, this paper's suggested study is the first to use such a wide range of combinations and concentrate on the coexistence of traits at several levels.

The performance method and kind of information that is stolen are used to categorize spyware. Among spyware, key loggers and screen recorders are the most common and harmful types. The two most significant non-functional characteristics (quality qualities) in the malware's architecture are stealth and longevity, as was previously stated. Spyware requires to hook operating system resources, particularly system functions (APIs), in order to steal information. The purpose of hooking is to serve as an intermediary between the victim system's installed apps or user requests and the OS resources. As a result, it has the ability to record and misuse incoming data. The majority of malware can use network connections to transmit the stolen data to pre-specified locations and the Command and Control (C&C) center.

Another type of malware examined in this research is ransomware. Because of its malicious conduct, which compels victims to perform unwanted and required behaviors like extortion of money, ransomware is classified as malware. Blockers are a prevalent subclass of this type of malware. We focused on key loggers, screen recorders, and blockers in this investigation.

A. STATIC BASED APPROACH

Using a hybrid of static and dynamic analysis, these methods for Android malware detection provide high detection accuracy. A sandbox for Android apps was proposed by Blasing et al. [4] that analyzes the app both statically and dynamically to identify suspicious activity in applications. Android applications are decompiled using the Baksmali tool [5] after being compressed _rst during the static analysis step.

Then, examining the decompiled smaliles yields static pattern extraction. The use of the Runtime.Exec() function, Java native libraries, re_ection, rights, services, and IPC provisioning are examples of static patterns. Installing and running the program on the Android emulator occurs during the dynamic analysis phase. Using arbitrary user inputs like touches, clicks, and gestures, the program is controlled by the monkey tool.

Numerous writers have researched the use of static features in Android malware detection. A comparison of the related works addressed in this section is shown in Table 1. In order to identify Android malware based on permissions and API features, Tiwari and Shukla presented a machine learning technique called logistic regression. The reduced features dataset, which has 131 features, and the complete features dataset were used by the authors to evaluate the model. Using datasets with complete features and reduced features, respectively, the authors reported that their model attained an accuracy of 97.25% and 95.87%.

Disadvantages:

- **Restricted Contextual Understanding**: Static analysis does not take into account an Android program's runtime behavior or external interactions; instead, it only looks at the features and attributes

of the application.

- **The use of obfuscation techniques:** By malware developers to hide dangerous code in Android applications makes them vulnerable to detection and deciphering by static analysis tools.
- **Detection of Dynamic Behavior:** Android applications' runtime behaviors, like file system interactions, network communication, and sensitive data exfiltration, cannot be detected by static analysis.
- **Dependency on Signature-Based Detection:** To find known malware patterns or signatures inside Android applications, a lot of static analysis tools rely on signature-based detection approaches**.**
- **Managing Code Complexity Can Difficult:** Static analysis tools may find it difficult to analyse complicated and obfuscated code structures, especially in large and complex Android applications.

IV. PROPOSED SYSTEM

More effective detection techniques must be developed because of the serious threat that Android malware poses to consumers. This work suggests an integrated method to improve Android malware detection efficiency that integrates dynamic analysis with machine learning classifiers, namely Gradient Boosted Trees (GBT) and Ridge Classifier. Through the use of real-time monitoring of network traffic, API calls, system communications, and other interactions between Android applications and the device's environment, dynamic analysis might reveal malicious activity that static analysis could miss. The reason for selecting the Ridge Classifier and GBT is their capacity to manage high-dimensional data and nonlinear interactions, which are common in dynamic analytical aspects of Android apps. When the number of features exceeds the number of samples, the Ridge Classifier, a linear model with L2 regularization, reduces the likelihood of overfitting and enhances generalization. In contrast, numerous weak learners—mostly decision trees—are combined in ensemble learning, or GBT, to create a strong prediction model. As a result of its proficiency in identifying complex correlations and patterns in data, GBT is well-suited to handle nonlinear interactions in dynamic analytic features. By differentiating between benign and dangerous apps based on observed runtime behaviours, both classifiers take advantage of the dynamic analytical features of the proposed system for training and evaluation. This improves overall accuracy and makes it possible to detect novel malware variants.

The combination of dynamic analysis with GBT and Ridge Classifier provides a thorough and effective way to identify Android malware. Through the integration of machine learning and dynamic analysis, this methodology enhances detection effectiveness and adaptability to changing malware threats in the Android environment. Strong and effective detection systems are required due to the increasing sophistication of malware and the widespread use of Android devices. Static analysis is a common technique used in traditional techniques; it looks at an application's code or structure without running it. This research presents a comprehensive method to improve Android malware detection by combining dynamic analysis with machine learning classifiers. Figure 2 illustrate the overall architecture process.

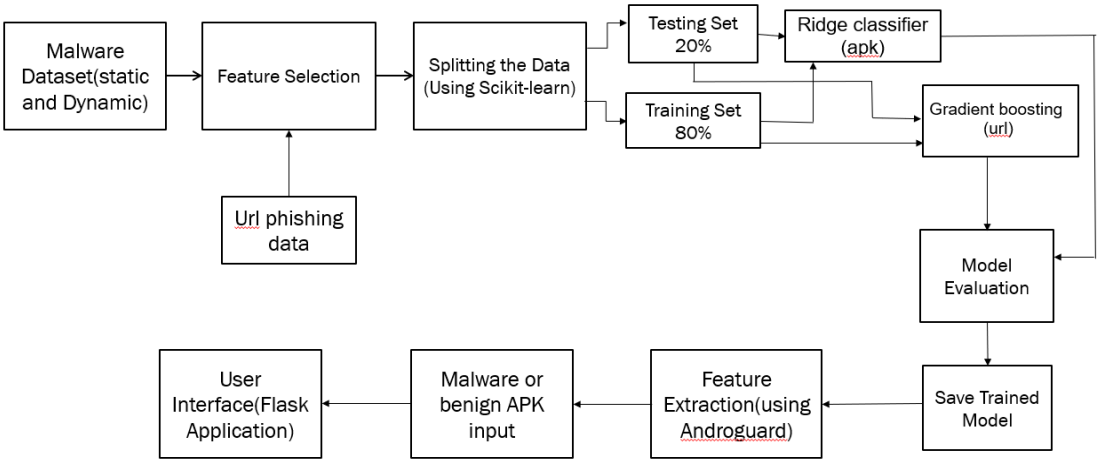

Figure 2. Overall Architecture Process

Dynamic Analysis in Android Malware Detection: Dynamic analysis is a technique used in Android malware detection that entails keeping an eye on an application's behaviors while it is running in order to see how it interacts with the surroundings. Network traffic, API calls, file system interactions, and system behavior are some of the important aspects of dynamic analysis. These features make it easier to identify malicious activity like data

exfiltration, privilege escalation, and illegal access and provide insightful information about the intentions of applications.

Machine Learning Classifiers: Based on observed dynamic analysis data, machine learning classifiers are essential for differentiating between benign and malicious programs. Due to their effectiveness in managing high-dimensional data and the nonlinear interactions present in dynamic analytic features, two classifiers—the Ridge Classifier and Gradient Boosted Trees (GBT)—were chosen.

Ridge Classifier: The Ridge Classifier is an L2 regularized linear model that works well in situations where there are more characteristics than samples. L2 regularization improves generalization and reduces the danger of overfitting by punishing large coefficients. By using dynamic analytic features, the Ridge Classifier increases the accuracy of malware detection by identifying patterns that correspond to malware behaviour.

Gradient Boosted Trees (GBT): GBT is an ensemble learning method that builds a potent prediction model by combining several weak learners, usually decision trees. GBT efficiently captures complex relationships and patterns in data by optimizing iteratively to minimize residual errors. GBT performs exceptionally well in handling nonlinear interactions within the framework of dynamic analytic features, which improves the identification of intricate malware behaviours.

Integration of Dynamic Analysis with Ridge Classifier and GBT: For training and evaluation, the suggested method combines dynamic analysis characteristics with Ridge Classifier and GBT. The system obtains a higher accuracy in differentiating between harmful and benign applications by utilizing the capabilities of both classifiers. Additionally, the integration makes it easier to identify new strains of malware based on observed runtime behaviors, strengthening defenses against ever-evolving threats.

Experimental Evaluation: Real-world Android malware samples are used in experimental evaluations to gauge the efficacy of the suggested methodology. To measure recognition efficiency and resistance against malware versions, performance metrics like accuracy, precision, recall, and F1-score are generated.

A thorough and effective technique for detecting Android malware is provided by the combination of dynamic analysis using Ridge Classifier and GBT. The suggested method improves detection accuracy and resistance to malware threats by utilizing machine learning approaches in conjunction with real-time studies of runtime activities. To further enhance detection efficiency, future research paths should look into new dynamic analytic features and classifier parameter optimization.

A. DYNAMIC BASED APPROACH

Some related research on the use of dynamic characteristics for Android malware detection is included in this section. A comparison of the related works addressed in this section is shown in Table 1. Through a dynamic examination of applications for Android, a study by A et al. suggested a model. Utilizing Android API calls and system call traces, the suggested system automatically finds malware. The scientists employed various datasets, totalling 7520 apps, from Virusshare and the Malgenome Project. The accuracy of the suggested model, which evaluated many classification methods, was 96.66%. However, in certain instances, their method was unable to keep an eye on the malevolent activity. For instance, the model ceases to run the application without collecting any information on any dangerous activity when the program is unable to establish a connection to the Internet. Dash et al. presented a method for grouping Android malware into families based solely on characteristics seen during application deployment. The suggested method made use of conformal prediction and support vector machines (SVM). 94% accuracy was attained in the studies, which used Binder communication features and system calls on the Drebin dataset. Unfortunately, their method was hampered by the scant data obtained by tracking low-level events and the inadequate coverage during application testing.

Advantages:

- **Comprehensive comprehension of malware behaviour:** Real-time runtime behaviour of Android applications can be observed by the system through dynamic analysis, which offers insights into how the programs interact with the device's surroundings.
- **Finding Previously Unseen Malware Variants:** Dynamic analysis works especially well for finding previously unknown or zero-day malware variants that only behave maliciously when they are executed.
- **Adaptability to Changing Threats:** By continuously learning from fresh data, machine learning classifiers like Ridge Classifier and GBT can adjust to changing threats.

- **The dynamic analysis features:** derived from Android applications are an example of high-dimensional data that may be handled effectively with GBT and Ridge Classifier.
- **Robustness Against Evasion Techniques:** By combining machine learning classifiers with dynamic analysis, the system becomes more resilient to evasion tactics used by malware writers.
- **Interpretability and Explainability:** The GBT's Ridge Classifier and decision trees offer models that are comprehensible, enabling security analysts to comprehend the elements influencing the classification choices.

An organized methodology is used by the Android malware detection effort. Following preprocessing to clean and modify the data, it begins with acquiring datasets of both benign and dangerous apps. The apps' static and dynamic features are then produced using feature extraction. These attributes are used in the training of machine learning models such as Gradient Boosted Trees and Ridge Classifier. Performance of the models is evaluated using evaluation measures. The models are deployed, and they watch incoming apps for unusual activity and react accordingly. Over time, the system is improved by a feedback loop. Moreover, fraudulent URLs are detected via a URL phishing model.

V. METHODOLOGY

The suggested approach consists of two steps: creating a fresh dataset with co-existing attributes and applying several machine learning classifiers to this dataset in order to distinguish between malware and safe apps. The co-existing feature datasets comprise of three types of data: permissions only, API calls only, and a mix of both permissions and API calls along with API frequencies.

A. Data-set

The Canadian Center for Cybersecurity at the University of New Brunswick is the source of the CICAndMal2017 dataset [22]. This collection includes traf_c data for four distinct malware categories in addition to benign traf_c: ransomware, adware, scareware, and SMS malware. Because different spyware has stealthy features, they installed the applications on actual Android smartphones rather than using emulators. Three separate time intervals were used to gather traffic data: immediately following app installation, fifteen minutes prior to device reboot, and fifteen minutes following device reboot.

This research is focused on identifying Android ransomware. Thus, we examine the samples that match the ransomware sample. Our approach involved generating a fresh dataset comprising both benign and malware traf_c. The quantity of each ransomware type is shown in Table 1. In an effort to lessen the imbalance problem in the dataset, we combine a large number of ransomware samples into a single _le and add 54161 examples of benign traf_c samples photographed in 2017. 402834 samples total—both benign and ransomware—are included in the dataset as a result. The features F1 (Flow ID), F2 (Source IP), F4 (Destination IP), and F7 (Timestamp) in the aggregated dataset are dependent on the setup used for the experiment and are not compatible with a production network. As is the case in [18] and [19], we eliminate these traits because they have no bearing on the traf_c categories. Certain features also always have zero values. The features have no correlation with the class labels because their values are constant for each instance. We eliminated certain features because of this. F38, F39, F40, F52, F56, F57, F63, F64, F65, F66, F67, and F68 are among the all-zero features. The F62 feature was also eliminated because it was a duplication of the F41 feature. There are now 68 features in the dataset after the aforementioned features were eliminated.

B. Data preprocessing

HANDLING IMBALANCED DATASETS

Prior to creating various combinations and throughout the classification procedure, this study employed the random under-sampling approach. For a dependable system to be constructed, about equal numbers of malware and benign samples must be used in the blend development process. The system should be built using genuine data that can differentiate between benign and malicious apps. Because it is an easy way to exclude samples without placing restrictions on the data, the random under sampling technique was employed. Moreover, a number of studies have demonstrated that the use of random under-sampling does not significantly alter accuracy in any experiment [20]. Also, every program is decompiled independently, and there are sufficient instances
to train and test the machine learning models. As a result, there is rarely a substantial loss of data when some programs are discontinued.

APK DECOMPILING

An Android APK file contains its manifest file, the classes.dex file, and the raw assets such as layout files and graphics. In this effort, two free software APK decompiling programs were used: Dex2jar and ApkTool. Because of its

strength and ease of use, the ApkTool has been selected. One APK file at a time is processed by these programs. In order to decompile every APK file simultaneously, a python code script has been created. Certain APKs were eliminated since the developer's mistakes in the file suffix prevented them from compiling. Twenty-eight APK files were decompiled from the CIC_MalDroid2020 dataset following the removal of these files. Next, permits and static APIs were extracted using their Manifestation files and smali plugins.

FEATURE SELECTION

Methods for detecting malware are more effective than static or dynamic ones. On the other hand, dynamic techniques take longer and have a higher processing overhead. The extraction and processing of a higher number of features during and after development is the cause of this increase in computation and time. The computational load and resource requirements can be reduced by carefully selecting the right features, enabling quick and accurate malware detection in on-device situations. The dynamic approach's performance is greatly enhanced by the selection procedures, which guarantee that the selected features maximize their contribution to the detection process while preserving the system's efficacy.

FEATURE SELECTION ALGORITHM

For hybrid malware detection, investigators have used a variety of feature selection techniques. In research by Alzaylaee et al., the information gain algorithm has been employed as a choice feature. Gandotra, Dhalaria, Li, and others. Hadiprako so et al., meanwhile. The principle component analysis was the method chosen by Hussain et al. and Ahmed (PCA). Some research also used manual techniques, such deleting certain strings, getting rid of duplicate programs, getting rid of empty features, and adjusting threshold variations. Notably, a single research project may use several choice of features strategies. In addition to employing pre-existing algorithms, some academics have opted to develop their own unique feature selection methods. To find the most dominating features, for instance, Gera et al. devised a feature selection approach. An extensive synopsis of the investigators' methodology is given in Algorithm 1, which explains the specifics of this approach. With feature vector data from both malware and benign samples as input, the algorithm seeks to find and extract the list of k-dominant characteristics that are necessary for additional processing and evaluation.

FEATURE SELECTION MECHANISM

Two approaches exist for applying the feature selection method in hybrid malware detection: either individually to the static and dynamic features, or to their combined expression. The selected method will determine how accurate and efficient the resulting detection model is. We will examine various feature selection methods frequently employed in hybrid malware detection in the debate that follows.

1. Both amenities, but distinct: this method selects features based on both static and dynamic characteristics separately, with the chosen features being combined afterwards. Numerous studies—among them the ones by Hussain et al. A couple studies that made use of this feature selection technique include Li et al. and Maryam et al.
2. Just static features: just the static features are subjected to the feature selection procedure in this method. Arshad et al. chose this course of action because, in contrast to dynamic analysis, their suggested scheme produced a very diversified set of features through static analysis. In a similar vein, this method was selected by other researchers, such as Feng et al., Shijo and Salim, Patel, and Buddadev, to solve certain difficulties they ran into in their own studies.
3. Only dynamic attributes: This method limits the feature selection process to only those features that are dynamic. Feature selection was applied by Liu et al. by clustering the multidimensional calls to functions graph. The ensuing classification procedure is therefore made easier by using the generated clusters as feature dimensions.
4. On the a single instance: this method applies the feature selection strategy after combining the static and dynamic features. Many researchers, including Gera et al., Lindorfer et al., Saif et al., among others, have employed this methodology. By utilizing their complimentary strengths and choosing the most informative and discriminative features, the combination of dynamic as well as static characteristics with selection of features improves the accuracy of recognizing malware.

FEATURE EXTRACTION

An essential part of the suggested Android malware identification system is the Feature Extracted Module, which collects pertinent features from static and dynamic assessments of Android apps. This module offers complete data to assist in training and assessing the finding model by merging insights from runtime behaviors seen through dynamic examination with static analysis, made easier by the Androguard toolset.

1. **Static Analysis with Androguard:**

The Static Analysis module extracts features from APK files without running them by using the well-known open-source application Androguard. Once the APK file has been parsed, Androguard gathers the metadata, permissions, code structure, and other static attributes that are necessary to comprehend the features of the application.Extracted static features consist of:

- Permissions that the program asks for in order to access device resources and provide information about possible features.
- Specifics of the manifest file: The AndroidManifest.xml file contains information that helps to understand the layout and functions of the program, such as the package name, version, activities, and services.
- File organization: resources, assets, and other files that are part of the APK, providing further details on the assets and features and possible uses of the application.
- Metadata: Characteristics such as file size, creation date, and other metadata properties that offer basic application information.

The Static Analysis module aids in the overall detection process by identifying potential indicators of malicious behavior by examining these static attributes using known patterns and signatures.

2. **Using Androguard for Dynamic Analysis:**

This module enhances static analysis by watching how APK files behave in a sandbox or emulator during runtime. With the help of Androguard's instrumentation and tracking technologies, one can observe many runtime phenomena, including:

- System calls: Communications over networks, file systems, and process administration are examples of interactions with the fundamental operating system.
- File system connections: The act of adding, deleting, or altering files on the folder system of the gadgets, which may reveal methods for data manipulation or retention.
- HTTP requests, DNS queries, and other network interactions are examples of network traffic that can be used to identify communication patterns and possible links to malevolent services.
- Interfaces between device resources: access to the audio and video recording capabilities on a device, which could indicate privacy violations or unlawful use.

Runtime code subsequent ones, genetic malware, and hidden code are just a few of the dangerous actions that dynamic analysis reveals that might go undetected by static analysis. The Dynamic Analysis feature gives the system an improved capability to identify advanced malware variants by keeping an eye on live characteristics.

3. **Machine Learning Classification:**

Features are used to train models for classification using machine learning methods like the Ridge Classifier after they are extracted from both static and dynamic studies. The static analysis aspects reveal information about the fundamental qualities of the application, and dynamic analysis aspects record its behaviors during execution. Accurate detection is facilitated by the classification model's ability to differentiate between benign and malicious programs based on these traits. This assignment is especially well-suited for the Ridge Classifier, which is well-known for its effectiveness when handling high-dimensional data. Through training on a dataset that includes both harmful and benign programs, the classifier has the ability to identify patterns that are suggestive of malware activity. The classification approach delivers improved resilience and accuracy in recognizing possible threats by utilizing data from both static and dynamic analysis.

4. **Training and Evaluation:**

The Feature Extraction module separates the dataset into training and testing sets in order to train the classification model. Training the model with features gleaned from both static and dynamic studies yields an evaluation of its performance measures, including accuracy, precision, recall, and F1-score, on the testing set. The model is guaranteed to identify between malicious and benign apps with effectiveness and to generalize well to data that hasn't been seen before thanks to this iterative approach.

To sum up, by extracting pertinent characteristics from both static and dynamic analyses, the Characteristic Extraction Module is an essential component of the malware detection for Android system. This module offers rich data for building a classification model by using Androguard observations for static analysis and watching runtime activities in a regulated environment. The system's capacity to precisely identify and categorize Android malware is improved by the integration of static and dynamic analytic technologies, which also improves safety for users within the framework of Android.

MACHINE LEARNING CLASSIFIER

APK files and URLs are categorized as either benign or malicious by the Machine Learning Classifier Module, which is the central component of the Android malware detection system. The classifier can evaluate the danger of APK files

and URLs with more accuracy because pattern recognition and correlation analysis are integrated into the feature space.

The **Ridge classifier** is chosen for malware APKs because it can manage high-dimensional data and reduce the danger of overfitting. The Ridge classifier, trained on data taken from both static and dynamic analyzes of APK files, learns to categorize malware according to observable patterns suggestive of malevolent activity. The classifier creates decision limits to precisely detect possibly hazardous APKs by utilizing the labeled dataset. Figure 3: Accuracy Graph for Ridge Classifier is represented as shown in below.

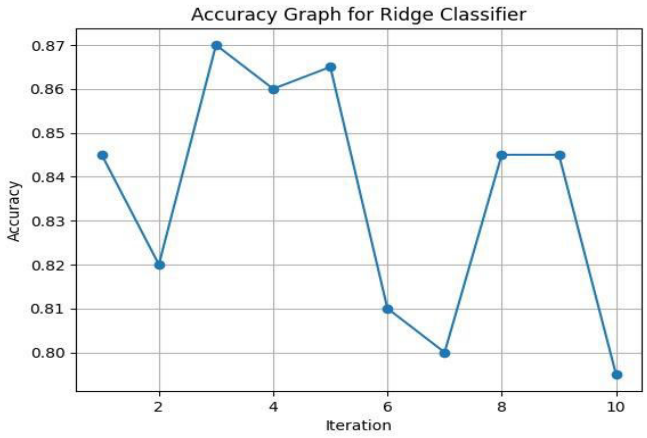

Figure 3: Accuracy Graph for Ridge Classifier

Gradient Boost - URL Phishing Detection: By deceiving users into disclosing critical information, URL phishing poses a serious risk to their security. Using the Gradient Boost approach, phishing URLs are found. The Gradient Boost classifier is trained on characteristics derived from URL parsing and dynamic analysis. It is capable of recognizing patterns suggestive of phishing efforts, such misleading domain names or dubious redirections.

A numerical evaluation of the risk connected to APK files and URLs is given by the trained machine learning model. The classifier gives each entity a risk score that indicates how likely it is to be dangerous by using features taken from both static and dynamic analysis. This risk evaluation improves overall cybersecurity posture by assisting users and security professionals in making well-informed decisions about installing programs or visiting URLs.

After training, the machine learning classifiers undergo evaluation and validation on independent test datasets to measure performance measures such as F1-score, accuracy, precision, and recall. To make sure the classifiers are reliable and capable of generalizing, cross-validation techniques can also be used. Sustaining the classifiers' effectiveness against dynamic threats requires ongoing performance validation and monitoring. Figure 4: Accuracy Graph for GBT Classifier is represented as shown in below.

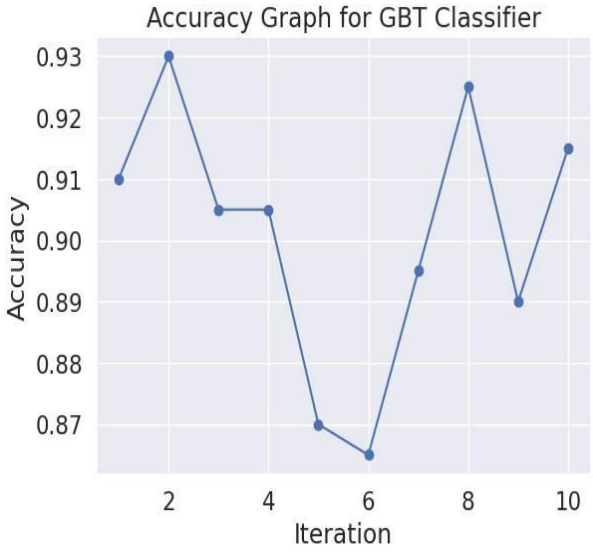

Figure 4: Accuracy Graph for GBT Classifier

ALGORITHM:

Step 1:

- Preprocessing
- Load the dataset
- Perform feature extraction and preprocessing
- Split the dataset into training and testing sets

Step 2:

- Train Gradient Boosting Classifier
  Fromsklearn.ensembleimport GradientBoostingClassifier
- Initialize the Gradient Boosting Classifier gb_classifier = GradientBoostingClassifier()
- Train the Gradient Boosting Classifier gb_classifier.fit(X_train, y_train)

Step 3:

- Train Ridge Classifier from sklearn.linear_model import RidgeClassifier
- Initialize the Ridge Classifier ridge_classifier = RidgeClassifier()
- Train the Ridge Classifier ridge_classifier.fit(X_train, y_train)

Step 4:

- Evaluate Models from sklearn.metrics import accuracy_score
- Predict using Gradient Boosting Classifier gb_predictions = gb_classifier.predict(X_test) gb_accuracy = accuracy_score(y_test, gb_predictions)
- Predict using Ridge Classifier ridge_predictions = ridge_classifier.predict(X_test) ridge_accuracy = accuracy_score(y_test, ridge_predictions)

Step 5:

- Compare Performance
- print("Gradient Boosting Classifier Accuracy:", gb_accuracy)
- print("Ridge Classifier Accuracy:", ridge_accuracy)

As a result, the Machine Learning Classifier Module is essential to the malware identification system since it uses methods of machine learning to identify whether an app package or a URL is harmful or not. The classifiers improve user security in the Android platform by providing precise and quantified risk estimations by combining data from URL interpreting, dynamic examination, and static examination with well-executed decision-making and engineering. Maintaining the classifiers' efficacy against new malware and phishing attacks requires constant improvement and evaluation.

TESTING AND OPTIMIZATION

A crucial stage in the creation of the Android malware detection system is the Testing and Optimization Module, where the categorization framework is thoroughly reviewed, tested, and adjusted to guarantee peak performance. This module concentrates on fine-tuning the system's parameters to improve scalability, efficiency, and overall accuracy, using the Maldroid dataset as a benchmark.

Dataset Examination: The cornerstone for testing the classification framework is the Maldroid dataset, which is well-known for its extensive collection of labeled samples that includes both malicious and benign Android applications. Because of the enormous amount of data in the dataset, there are many opportunities as well as difficulties that call for rigorous analysis and optimization techniques.

Scalability and Efficiency: Because of the massive size of the Maldroid dataset, it is critical to optimize the technique for effectiveness and adaptability. To effectively manage large-scale data processing, methods including distributed computing, parallel computation, and data sampling are used. Furthermore, to lower the computational expense and improve runtime efficiency, algorithmic reductions and compressed model approaches can be used.

Performance Metrics: A variety of accuracy criteria, such as training and validation accuracy, are used to assess the efficacy of the categorization system. The model's performance on the training dataset is measured by training accuracy, which shows how well the model has absorbed the information given to it. Conversely, validation accuracy evaluates the model's capacity to generalize to fresh, untested data. A high validation accuracy shows how resilient the model is in practical situations.

Fine-tuning and Hyper parameter Optimization: Optimizing the classification framework's efficiency requires modifying a number of variables, or hyper parameters. The best configuration is found by methodically exploring the hyper parameter range using strategies including grid search, random search, and Bayesian optimizing. Tree depths, ensemble sizes, legalization parameters, and learning rates are a few examples of hyper parameters. To get the best

results, the classification framework is refined iteratively by varying those variables and assessing how they affect performance indicators.

Generalization and Cross-validation: Cross-validation methods, like k-fold cross-validation, are used to reduce overfitting and evaluate the model's capacity for generalization. Cross-validation yields a more reliable assessment of the model's performance by splitting the dataset into several subgroups and training the model on a variety of both training and verification data. Additionally, methods like domain adaptation and transfer learning could be used to improve the model's generalization capabilities across various datasets and real-world situations.

Performance Demonstration: The classification framework's comparatively high accuracy on the used training and verification datasets demonstrates how successful it is at classifying malware. A high validation accuracy shows the model's capacity to generalize to new, untested samples, whereas a high training accuracy shows the model has successfully learned from the supplied data. The system for classification highlights its effectiveness in detecting and categorizing Android malware by presenting its performance measures, such as accuracy, precision, recall, and F1-score.

In summary, scalability, efficiency, and high accuracy are all ensured by the Testing and Optimization Module, which is vital to the system's improvement in terms of Android malware detection. To improve the performance of the classification framework, optimization techniques such hyperparameter tweaking, cross-validation, and performance evaluation are used, with the Maldroid dataset serving as a benchmark. Through methodical optimization of the system's configuration and assessment of its performance metrics, the module guarantees strong and dependable identification of Android malware, thus augmenting user security in the Android ecosystem.

EVALUATION & INTEGRATION

Robust categorization systems are becoming more and more necessary in today's digital ecosystem to guarantee the security and integrity of mobile devices due to the development of mobile applications. With an emphasis on the combination of feature extraction, machine learning, and assessment approaches, this paper offers a thorough strategy to creating such a system. It also suggests developing an easy-to-use interface to enable user involvement with the classification system.

Evaluation: Different assessment metrics, including F1-score, accuracy, precision, and recall, are used to gauge how well the classification model performs. These metrics shed light on the model's capacity to accurately categorize APK files and recognize any security risks. Furthermore, to guarantee the model's durability and dependability across various datasets, cross-validation methods such as k-fold cross-validation are utilized.

Integration: The core of the classification system is comprised of the combining of the machine learning, extracting features, and assessment components. The system can efficiently analyze APK files and produce precise categorization results by fusing these components into a logical framework. Large amounts of APK files may be analyzed in real time thanks to integration, which also makes the classification process more automated.

Interface Development: To improve how users engage with the classification system, a user-friendly interface is created. With ease, users may submit APK files, view categorization results, and acquire assessment data using this interface. Experts and novices alike can exploit the classification system to good effect thanks to the interface's straightforward and intuitive design.

In summary, feature extraction, machine learning, and assessment approaches must be integrated in order to create a thorough categorization system for APK files. Through the integration of these elements into a unified structure and the creation of a user-friendly interface, the system is able to reliably categorize APK files and offer insightful analysis of their potential security risks. These systems are essential for securing mobile devices and shielding users from possible security threats as the threat landscape keeps changing.

EVALUATION & INTEGRATION

The spread of malware poses a serious threat to cyber security in the digital age of today. Effective and easy-to-use malware detection tools are crucial to overcome this danger. The usability and efficacy of such systems can be significantly improved by combining a user-friendly user interface (UI) with perceptive visuals. This article will examine the planning and execution of a user interface module for a malware detection system, utilizing JavaScript, HTML, CSS, and Bootstrap techniques.

Users can communicate with the malware detection system through the UI module. Its main goal is to give customers a smooth and simple experience by letting them upload files or URLs for scanning, check detection results, handle threats, and adjust system preferences. Compatible with multiple platforms and deployment simplicity are ensured by utilizing contemporary web development tools.

Upload Interface: Users must be able to upload APK files or enter URLs to be scanned through the user interface. This part acts as the point of entrance for processes meant for recognizing malware. When combined with JavaScript features, HTML forms can make it easier to submit URLs and upload files.

Detection Findings: Following a scan, the user interface ought to present the results in an understandable and structured way. Readability and aesthetic appeal are improved by using HTML elements for structured text presentation and CSS for style. JavaScript can be used for dynamic updates and interactions, including category or severity-based filtering of discoveries.
Threat Management: To reduce possible hazards, identified threats must be effectively managed. Users should be able to delete, quarantine, and perform other actions on detected instances of malware using the user interface. Smooth threat management activities are made possible by interactive elements like buttons and checkboxes.
Configuration Options: To adapt the malware detection system to their own requirements, users might need to make changes to the configuration. The user interface can include configuration options for alert preferences, threat handling guidelines, and scan parameters. Bootstrap frameworks provide pre-styled elements to help designers create configuration interfaces that are both visually appealing and responsive.
Reports and Graphics: The communication of insights derived from malware detection findings is greatly aided by visualizations. The user interface module has the ability to produce extensive reports that include text summaries, statistical analyses, and graphical displays of the detection results. For data visualization, interactive charts, graphs, and diagrams can be easily created with JavaScript frameworks such as Chart.js or D3.js.
UI Design Best Practices

- To improve usability and user familiarity, maintain consistency in the design elements, layout, and navigation throughout the various UI areas.
- Accessibility: Follow online accessibility guidelines to make that the user interface is usable by a wide range of abilities. When non-text content is displayed, provide alternate text.
- Implement feedback systems to keep users informed about system operations and results. Examples of these include progress indicators, status messages, and error notifications.
- Create a user interface (UI) that is flexible and responsive to different screen sizes and devices to guarantee a consistent experience across PCs, tablets, and smartphones. This is known as responsive design.
- Conduct user testing and collect feedback repeatedly to find usability problems and improve the user interface design for the best possible experience.

A strong user interface (UI) module is essential to a malware detection system because it allows users to interact with it in an easy way and effectively handle threats that are discovered. Developers can boost the system's usability and accessibility by utilizing HTML, CSS, JavaScript, and Bootstrap to create aesthetically pleasing and intuitive user interfaces. When it comes to fighting the always changing threat landscape of malware, incorporating elements like upload interfaces, detection findings presentation, threat management functionalities, configuration options, and visualization capabilities guarantees a thorough and interesting encounter for users.

VI. RESULTS AND DISCUSSION

Dataset
In today's digital world, malware identification, classification, and URL detection are essential components of cyber security. Organizations and researchers need to stay ahead of the curve as cyber threats keep evolving and getting more complex. To achieve this, they need to use strong techniques and high-quality datasets for training and analysis. The methodology that has been proposed places significant emphasis on the utilization of datasets obtained from reputable sources, like the Canadian Institute for Cyber security (CIC). Researchers and industry practitioners can benefit greatly from the CIC's extensive collection of cyber security-related data, which is well-known. Accessing a variety of representative datasets is crucial when it comes to the identification, categorization, and detection of malicious websites. The basis for creating machine learning models and algorithms that can precisely recognize and categorize dangerous software and detect dubious URLs is provided by these datasets.

Usually, the Preparing set and the Test set are the two primary subsets of the dataset obtained from the CIC. Model creation and training employ the Preparing set, which contains eighty percent of the total data. This part of the dataset is the key to teaching the algorithms to identify patterns, distinguish between good and bad entities, and eventually increase the accuracy of the algorithms in real-world situations. However, the remaining 20% of the data, known as the Test set, is used to assess how well the constructed models work. Researchers may make sure their methods are reliable and well-suited to generalizing to new data by breaking the dataset up into these discrete subsets. This method also lessens the chance of over fitting, which occurs when a model works well on training data but not well enough to generalize to new examples. The best practices for feature engineering, data preparation, and model evaluation must be followed by researchers while creating experiments and doing analysis with the dataset. To do this, the data must be cleaned to reduce noise and inconsistencies, pertinent features must be chosen to capture important information, and suitable metrics must be used to evaluate the models' performance. Moreover, strategies for managing unbalanced datasets must be included, since cyber security datasets frequently show skewed class distributions with a disproportionately high proportion of benign samples compared to malevolent ones. An imbalance like this can be addressed and the models' overall performance enhanced with the use of techniques like

oversampling, under sampling, and synthetic data generation. Researchers are looking more closely at using deep learning methods like Recurrent Neural Networks (RNNs) and Convolution Neural Networks (CNNs) for URL classification and malware detection in addition to more conventional machine learning techniques. Cyber security applications can benefit greatly from these deep learning architectures since they are adept at identifying complex patterns and connections in sequential and spatial data.

On the other hand, deep learning model deployment in practical environments necessitates a significant investment in computational power as well as proficiency in model tuning and optimization. Furthermore, it might be difficult to evaluate the results of deep learning models and comprehend the underlying logic because these models are frequently referred to as "black boxes." Because of this, when developing their approaches, researchers need to balance model complexity with interpretability.

Finally, it should be noted that malware identification, categorization, and URL detection are essential elements of cyber security, requiring the use of reliable datasets and strong methodology for efficient analysis and model building. Researchers may improve cyber security by using datasets from reliable sources like the Canadian Institute for cyber security and following best practices for data pretreatment and model evaluation. This will also help to keep up with rising cyber threats.

Input [X]	Output [Y]
X – Training (80%)	Y – Training (80%)
X – Testing (20%)	Y – Testing (20%)

TABLE 1. SPLITTING OF DATASET

PERFORMANCE ANALYSIS

Gradient Boosted Trees (GBT) :

With an astounding 93.5% accuracy rate, the GBT Classifier demonstrates its capacity to distinguish between Android applications that pose a threat and those that are not. Because of its great accuracy, the model may have been able to identify complex patterns and correlations in the dataset, which helped it generate correct predictions on the testing dataset. High accuracy is encouraging, but in order to minimize possible problems like overfitting or data leaking, it's important to examine the results more carefully. Metrics like precision, recall, and F1-score—which go beyond accuracy—must be looked at in order to fully assess the performance of the GBT model. The proportion of correctly predicted harmful applications among all applications that are projected to be dangerous is measured by precision, whereas the proportion of correctly predicted dangerous applications among all actually dangerous applications is evaluated by recall. By striking a compromise between recall and precision, the F1-score offers a comprehensive assessment of the model's performance.

Model Durability and Reliability:

To ensure the model's durability and reliability despite its high accuracy, it's crucial to validate the results using techniques like holdout validation or cross-validation. These methods help gauge the model's performance on unseen data and mitigate issues like overfitting, enhancing its generalization capabilities. By subjecting the GBT model to rigorous validation processes, we can ascertain its robustness in real-world scenarios.

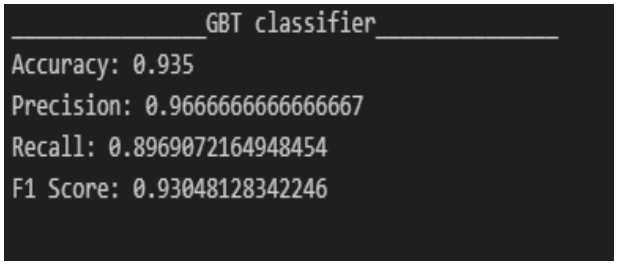

Figure 5. GBT Classifier

Ridge Classifier:

Compared to the GBT Classifier, the accuracy rate of the Ridge Classifier is a little bit lower at 83%, but it is still useful for identifying Android applications. In spite of its little accuracy disadvantage, the Ridge Classifier shows that it can identify significant patterns in the data that result in precise forecasts. Additional Assessment: Metrics like as recall, precision, and F1-score can be used to assess the Ridge Classifier's performance in a more thorough manner, much like the GBT Classifier. We can make focused improvements to the classifier's performance by examining these indicators to learn more about its advantages and disadvantages.

Boosting Performance: A number of variables, such as feature importance assessment, class balance modification, and hyper parameter optimization, should be taken into account in order to boost the Ridge Classifier's performance. The prediction power of the model can be improved by fine-tuning hyper parameters, such as regularization strength in the case of Ridge regression. Furthermore, correcting class imbalances and locating important features can improve the accuracy and resilience of the classifier even more.

Comparative Benefit: The Ridge model has advantages over the GBT Classifier in terms of simplicity, interpretability, and computational economy, even though its accuracy is slightly lower. Because of its simplicity, it is easier to understand and apply, which might be useful in situations where model transparency is crucial. Furthermore, it is appropriate for applications requiring real-time processing or with limited resources due to its computational efficiency.

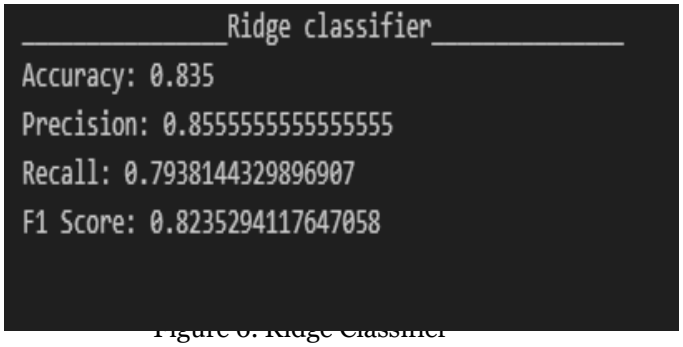

Figure 6. Ridge Classifier

In conclusion, even if the GBT and Ridge Classifiers show promise in categorizing Android applications, a full evaluation that goes beyond mere precision is necessary to determine their actual effectiveness. We can guarantee the dependability and resilience of these classifiers in practical situations by utilizing extra metrics and validation methods. Figure 7. Performance of a Classification Model is given below.

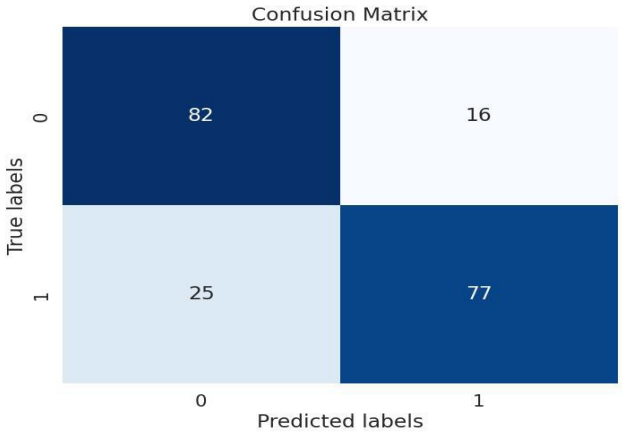

Figure 7. Performance of a Classification Model

CONCLUSION AND FUTURE ENHANCEMENT

Both the Ridge Classifier and the GBT Classifier are capable of accurately identifying between Android applications that are dangerous and those that are not, according to the findings of the trial. On the other hand, their computational efficiency and performance metrics varied. Due to its resilience in categorizing Android applications, the Ridge Classifier demonstrated good levels of accuracy, precision, recall, and F1-score. Malicious and benign samples may be easily distinguished in the feature space thanks to the Ridge Classifier's learned linear decision boundary. Yet, the GBT Classifier showed good predictive ability and achieved competitive performance metrics in terms of F1-score, accuracy, precision, and recall. GBT utilized an ensemble of decision trees to accurately classify Android malware by capturing intricate patterns found in the infection. The GBT Classifier is more scalable and has better interpretability of the model than the Ridge Classifier, although having slightly less accuracy. Analysts can comprehend the reasoning behind the classifier's predictions since decision trees are naturally interpretable. GBT is an excellent choice for real-world applications requiring vast volumes of data because it is extremely scalable and capable of handling large-scale information.

To sum up, we found that the Gradient Boosted Trees (GBT) Classifier and the Ridge Classifier performed well in our tests for detecting Android malware. The Ridge Classifier performed admirably, attaining excellent recall, accuracy, precision, and F1-score. In contrast, the GBT Classifier showed good predictive ability, while having marginally less accuracy than the Ridge Classifier. By effectively differentiating between legitimate and malicious Android applications, both classifiers improved mobile security procedures. Subsequent investigations may examine ensemble methodologies that amalgamate the advantages of both classifiers to enhance the detection of Android malware.

Future Enhancements:

Ensemble Learning: To further increase classification accuracy and resilience, take into consideration applying ensemble learning techniques like bagging or boosting to aggregate the predictions of several classifiers, such as the Ridge Classifier and GBT Classifier. By utilizing the advantages of several models, ensemble learning techniques have been demonstrated to lessen the drawbacks of individual classifiers. We can build an Android malware detection system that is more thorough and dependable by integrating several classifiers.

Feature Engineering: To improve the classifiers' recognition skills, use feature engineering to extract more static and dynamic information from Android applications. This can entail extracting more detailed runtime behaviors for dynamic analysis or utilizing sophisticated static analysis techniques. A key component of machine learning models' effectiveness is feature engineering. By finding and using pertinent features, we may enhance the classifiers' capacity to reliably discriminate between benign and malicious applications.

Deep Learning Models: Examine the suitability of deep learning models for Android malware detection, such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs). These models might be able to identify complex links and patterns in the data, which could increase the accuracy of detection. Deep learning techniques have shown impressive results in a variety of fields, and their use in Android malware detection could provide fresh perspectives and enhance detection powers, particularly when it comes to identifying complex and dynamic malware threats.

Real-time Monitoring: Create real-time monitoring systems that track the actions of Android apps while they are being installed, run, or executed. Include algorithms for anomaly detection to identify questionable activity and set

off fast reaction systems to neutralize such threats. The impact of malware infestations on users' devices and data is reduced by proactive threat identification and response made possible by real-time monitoring. We can improve mobile security and give users all-around defense against new threats by utilizing cutting-edge monitoring techniques.

Implementation on Mobile Devices: Examine the viability of installing the classifiers' minimal versions directly on mobile devices in order to carry out malware identification there. By depending less on cloud-based detection services, this method can improve user privacy and offer prompt defense against new threats. In situations where internet access may be intermittent or limited, on-device detection might help by facilitating faster response times and minimizing reliance on network connectivity. Through mobile deployment optimization of the classifiers, we provide safeguards against malware even in offline contexts and enable consumers to take charge of their mobile device security.

To sum up, the trials carried out on Android malware detection using Ridge Classifier and Gradient Boosted Trees (GBT) Classifier showed encouraging outcomes and indicated potential areas for further development. We can further enhance the efficacy and efficiency of Android malware detection systems by investigating ensemble learning techniques, feature engineering, deep learning models, real-time monitoring, and on-device deployment. These developments are essential for preventing malware from becoming more sophisticated and for protecting the data and devices of mobile users. The dynamic landscape of mobile cybersecurity may be strengthened, and consumers can be given strong protection against new threats by keeping up with innovation and improvement in our approach to malware identification.

REFERENCES

[1]. X. Ugarte-Pedrero, M. Graziano, and D. Balzarotti, “A close look at a daily dataset of malware samples,” ACM Trans. Privacy Secur., vol. 22, no. 1, pp. 1–30, Jan. 2019.

[2].Y. Tanaka, M. Akiyama, and A. Goto, “Analysis of malware download sites by focusing on time series variation of malware,” J. Comput. Sci., vol. 22, pp. 301–313, Sep. 2017.

[3]. H. Huang, S. Zhang, X. Ou, A. Prakash and K. Sakallah, "Distilling critical attack graph surface iteratively through minimum-cost SAT solving", Proc. 27th Annu. Comput. Secur. Applicat. Conf., pp. 31-40, 2011.

[4]. O. Or-Meir, N. Nissim, Y. Elovici and L. Rokach, "Dynamic malware analysis in the modern era—A state of the art survey", ACM Comput. Surv., vol. 52, no. 5, pp. 1-48, Sep. 2020.

[5]. J. Liu and J. Yu, "Research on development of Android applications", Proc. 4th Int. Conf. Intell. Netw. Intell. Syst., pp. 69-72, Nov. 2011.

[6]. J. Yang, Z. Zhang, H. Zhang and J. Fan, "Android malware detection method based on highly distinguishable static features and DenseNet", PLoS ONE, vol. 17, no. 11, Nov. 2022.

[7]. J. B. Higuera, C. A. Aramburu, J.-R. B. Higuera, M. A. S. Urban and J. A. S. Montalvo, "Systematic approach to malware analysis (SAMA)", Appl. Sci., vol. 10, no. 4, pp. 1360, Feb. 2020.

[8]. E. Amer, I. Zelinka and S. El-Sappagh, "A multi-perspective malware detection approach through behavioral fusion of API call sequence", Comput. Secur., vol. 110, Nov. 2021.

[9]. M. N. Al-Andoli, K. S. Sim, S. C. Tan, P. Y. Goh and C. P. Lim, "An ensemble-based parallel deep learning classifier with PSO-BP optimization for malware detection", IEEE Access, vol. 11, pp. 76330-76346, 2023.

[10]. M. Kim, D. Kim, C. Hwang, S. Cho, S. Han and M. Park, "Machine- learning-based Android malware family classification using built-in and custom permissions", Appl. Sci., vol. 11, no. 21, pp. 10244, Nov. 2021.

[11]. S. A. Nikale and S. Purohit, "Comparative analysis of Android application dissection and analysis tools for identifying malware attributes" in Big Data Analytics and Intelligent Systems for Cyber Threat Intelligence, Denmark:River Publishers, pp. 87-103, 2023.

[12]. J. Senanayake, H. Kalutarage and M. O. Al-Kadri, "Android mobile malware detection using machine learning: A systematic review", Electronics, vol. 10, no. 13, pp. 1606, Jul. 2021.

[13]. H. Cheng, X. Yan, J. Han, and C.-W. Hsu, “Discriminative frequent pattern analysis for effective classification,” in Proc. IEEE 23rd Int. Conf. Data Eng., Apr. 2007, pp. 716–725. M.
Parkour. Contagio Mini-Dump.

[14]. A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “MADAM: Effective and efficient behavior-based Android malwaredetection and prevention,” IEEE Trans. Depend. Sec. Comput., vol. 15, no. 1, pp. 83– 97, Jan./Feb. 2018.

[15]. Pallavi V. Baviskar, Guddi Singh, Vijay Narendranath Patil, "Design of Machine Learning- Based Malware Detection Techniques in Smartphone Environment", 2023 International Conference for Advancement in Technology (ICONAT), pp.1- 5, 2023.

[16]. Fabio Martinelli, Francesco Mercaldo, Andrea Saracino, Corrado Aaron Visaggio, "I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of Android malware", 2016 14th Annual Conference on Privacy, Security and Trust (PST), pp.129-136, 2016.

[17]. TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer and Eul GyuIm, "A Multimodal Deep Learning Method for Android Malware Detection Using Various Features", IEEE Transactions on Information Forensics and Security, vol. 14, pp. 773- 788, 2019.

[18].S. Maniath, A. Ashok, P. Poornachandran, V. G. Sujadevi, A. .U. P. Sankar and S. Jan, "Deep learning LSTM based ransomware detection", 2017 Recent Developments in Control Automation & Power Engineering (RD CAPE), pp. 442-446, 2017.

[19]. S Abhishek, Adithya Rajendran, Akhbar Sha, T Anjali, Arun Karunakaran Nair, "Enhancing Android Security: Dynamic Analysis for Resilient Defences Using Machine Learning", 2023 7th International Conference on Electronics, Communication and Aerospace Technology (ICECA), pp.1458-1463, 2023.

[20]. Esra Calik Bayazit, Ozgur Koray Sahingoz, Buket Dogan, "Protecting Android Devices From Malware Attacks: A State-of-the-Art Report of Concepts, Modern Learning Models and Challenges", IEEE Access, vol.11, pp.123314-123334, 2023.

[21]. Rizaldi Wahaz, Rakha Nadhifa Harmana, Amiruddin Amiruddin, Ardya Suryadinata, "Is WhatsApp Plus Malicious? A Review Using Static Analysis", 2021 6th International Workshop on Big Data and Information Security (IWBIS), pp.91-96, 2021.

[22]. A. Roy, D. Singh, A. Roy, D. Singh, G. Jaggi and K. Sharma, "ScienceDirect Android Malware Detection based on Vulnerable Feature Aggregation” Procedia Comput. Sci., vol. 173, no. 2019, pp. 345- 353, 2020.

[23]. M. Jerbi, Z. C. Dagdia, S. Bechikh and M. Makhlouf, On the Use of Artificial Malicious Patterns for Android Malware Detection, pp. 1-43, 2020.

[24]. S. Yoo, S. Kim, S. Kim and B. Byunghoon, "AI- HydRa: Advanced hybrid approach using random forest and deep learning for malware classification q", Inf. Sci. (Ny)., vol. 546, pp. 420-435, 2021.

[25]. A.Sonya, Dr.G.Kavitha, ”An effective block chain-based smart contract system for securing electronic medical data in smart healthcare application”, Concurrency and Computation : Practice
and Experience, Vol No. 34, Issue No. 28, Page No. 1-17, 2022.

[26]. A.Sonya, Dr.G.Kavitha, ”A Data Integrity and Security Approach for Health Care Data in Cloud Environment”, Journal of Internet Services and Information Security (JISIS), Vol No.12, Issue No 4, Page No.246-256, 2022.

[27]. A.Sonya, Dr.G.Kavitha “Human Health Care Systems Analysis for Cloud Data Structure of Biometric System Using ECG Analysis”, Evolutionary Computing and Mobile Sustainable Networks: Proceedings of ICECMSN 2021, Lecture Notes on Data Engineering and Communications Technologies, Vol No. 116, Page No. 163–176, 2021.

[28]. A.Sonya, Dr.G.Kavitha, ”Enhancing Data Security for Sharing Personalized Data in Mobile Cloud Environments”, Disruptive Technologies for Big Data and Cloud Applications, Lecture Notes in Electrical Engineering, Proceedings of ICBDCC 2021, Vol No. 905, Page No. 739–749, 2012.

[29]. A.Sonya, Dr.G.Kavitha, “Secure Transmission of Human Vital Signs Using Fog Computing, Springer Nature Switzerland AG 2020 J. S. Raj et al. (Eds.): ICIDCA 2019, LNDECT 46, Vol No. 46, Page No. 785–792, 2019.

[30]. L. Cai, Y. Li and Z. Xiong, "JOWMDroid: Android Malware Detection Based on Feature Weighting with Joint Optimization of Weight-Mapping and Classifier Parameters", Comput. Secur., pp. 102086, 2020.