Innovative: International Multi-disciplinary Journal of Applied Technology
(ISSN 2995-486X) VOLUME 02 ISSUE 05, 2024

AKANDO: A Hybrid Approach for Effective Android Botnet Detection

Akwukwuma Veronica N., Egwali Annie O., Asuquo Doris
Department of Computer Science, Faculty of Physical Sciences, University of Benin, Benin City, Nigeria

Abstract:

The ubiquitous nature of Android devices has regrettably rendered them prime targets for cyberattacks, particularly those orchestrated by botnets. Conventional solutions for Android botnet detection often rely on static or dynamic analysis techniques individually, leading to limitations in accuracy and adaptability. Existing research on Android malware detection utilizes various approaches, each with limitations. Some researchers focus on static analysis, examining code for signs of malicious activity. Others employ dynamic analysis, monitoring app behavior during runtime to detect suspicious system calls. Additionally, signature-based approaches compare apps to known malware signatures. However, research on anomaly-based detection, which identifies unusual app behavior without relying on pre-defined patterns, remains limited, leaving room for improvement. This paper proposes AKANDO, a novel botnet detection model that leverages a hybrid feature extraction approach and a multi-layered neural network architecture to achieve superior performance. By combining static and dynamic analysis, AKANDO gains a more holistic understanding of app behavior, potentially leading to superior detection accuracy compared to methods relying solely on one approach. The neural network architecture allows AKANDO to learn complex relationships between extracted features and botnet behavior, enabling it to adapt to evolving threats through continuous training with updated data. Moreover, AKANDO prioritizes minimizing false positives through its hybrid analysis and machine learning techniques, ensuring legitimate applications aren't flagged as malicious.

Keywords: Android botnet detection, Hybrid analysis, Machine learning, Neural networks and False positives

1. Introduction

Anomaly detection is identifying data points in dataset that do not fit the normal pattern. It is a technique used to identify unusual pattern that do not conform to expected behaviour. Anomalies

can be caused by errors in data or sometimes are indicative of a new underlying process, or malware attack. For many years now there has been incidents and reports of Cyber security, and most of these incidents are carried out through Botnets. Mobile devices have the highest market share of 86.93% as at December 2023 (Statcounter, 2023), which makes it an ideal target for attackers. These devices face an everyday growing number of security threats (Vuleta, 2023) because sensitive data such as contact lists, passwords, ATM card numbers and Bank Verification Number (BVN) are stored on these mobile devices and security issues are taken less seriously by the users of those devices.

Recently Vuleta (2023) of Legaljobs Lab has identified 44 new mobile malware families to watch out for this year 2023 (SMS trojans, iPhone malware, Android spyware) with 23,894 modifications. According to Kaspersky Lab reports of December 2020, their detection systems discovered an average of 360,000 new malicious files every day over the past 12 months, which represents an upward increase of 5.2% from 346,000 in 2018. Mobile bot runs automatically once it is installs on a device and gains complete access to the device and its contents, and starts communicating with and receiving instructions from one or more command and control servers controlled by a cybercriminal called a botmaster.

Barriga and Yoo (2017) opined that malware authors use many techniques to evade the detection such as (a) code obfuscation technique, (b) encryption, (c) including permissions which are not needed by the application, (d) requesting for unwanted hardware, (e) download or update attack in which a benign application updates itself or another application now with malicious payload, which is very tough to detect. Despite advancements in mobile security, there's a gap in research on anomaly-based detection systems for Android. This study focuses on detecting Android bots, aiming to reduce errors and improve accuracy. An open-source malware database from Kaggle.com, known for its high detection power, will be used for training. This research highlights the need for exploring new detection techniques.

There are three main categories for malware detection based on analysis methods: Static, Dynamic, and Hybrid. Static analysis examines an app without running it, looking at features like permissions and API calls within the manifest file. Dynamic analysis focuses on features extracted while the app runs, such as network traffic and battery usage. Hybrid analysis combines both approaches, which is where machine learning comes in, and forms the core of this research.

Mobile bots are malware that run automatically after installation. The key challenge lies in accurate detection. Attackers constantly develop ways to bypass traditional methods, making it difficult to keep up with new malware variations. Researchers aim to create a highly accurate system for catching bots while minimizing false positives (mistakenly identifying safe apps as threats) and false negatives (missing actual malware). This research strives for near-perfect detection, balancing threat identification with minimal errors.

2. Related Works

There have been significant research efforts to solve the problem of Android malware. Dini et al., (2012), describe, a Multi-Level Anomaly Detector for Android Malware (MADAM). The first prototype of MADAM was able to detect several real malwares found in the wild. The device usability was not affected by MADAM due to the low number of false positives generated after the learning phase. MADAM uses a global-monitoring approach that was able to detect malware contained in unknown applications, i.e. not previously classified. MADAM uses a smaller number of features, and has been tested on real malware found in the wild, and shows better performance in terms of detection and, especially, of false positives rate. After the learning phase, the false positive rate of MADAM was 7%, and the detection rate of MADAM is 93%. But the authors used WEKA tool that can only work on small dataset, and we are in the era of big data, in WEKA whenever a set

is bigger than a few megabytes an error occurs, and this can lead to malware evading thereby increasing the possibilities of High false positive rate.

Yerima et al., (2013) presented the static malicious detection approach based on the features - API Android Permissions, and Commands. They have used their own tool with a component API detector, Command Detector, and Android Permission Detector. They had analysed 2000 Android applications consisting of 1000 malicious applications from 49 Android Malwares families. They had selected 15 to 20 features from API, Android Permission, and command categories for the malicious application classification and had the AUC of 97.1% with the Bayesian model. This model could not classify malware family, that it did not recognize so the possibility of malware evasion is certain, thereby leading to high False positive rate. In 2021, there was an update in their work, when Yerima (2021), used Deep learning method to obtain 97.6% accuracy rate. Which shows an Improvement.

Kim et al., (2013) proposed Hybrid Intrusion Detection tool which used decision tree for classification of malicious and benign applications. They have designed an automatic feature extraction tool written with Java scripts which can extract two features, Permission and Method API. To evaluate their framework, they collected 893 normal applications from Android market and 110 malicious applications from the Internet site and had the detection rate of 82.7%. This tool also is unstable, if there is any small change in data there will be a large change in structure and this could lead to wrong classifying of malware therefore there is possibility of high false positive rate.

Canfora et al.,(2015) proposed an Android malware detection method that is different from other methods, their methods is based on selecting the longest sequences of system calls for the malware detection rather than considering the individual system call invocation by the Android application. They had used the SVM machine learning algorithms for malware detection and achieved the detection rate of 97%. The system call log of Android application execution on the real device had been used for features extraction. They had evaluated this method with 1000 benign and 1000 malicious applications from 28 malware families. The algorithm used here is SVM which does not perform well with large data set and one of the issues of SVM is that it under preforms in datasets with noise, therefore there is likelihood of high false positive rate.

Kurniawan et al., (2015) used Logger, a default application which is inbuilt in Android was used to extract the sum of Internet traffic, percentage of battery used and battery temperature for every minute. This information collected as set of features and is fed into WEKA, an open source learning library for testing and training with Naive Bayes, J48 decision tree and Random Forest algorithms. The author concluded that Random Forest has high accuracy of 85.6% with these features and proposes other features that can be combined with existing system to improve the accuracy. The author only mentioned that the False Positive and false negative rates were high, their focus was on accuracy rate which they compared four different algorithms and decided that Random forest produced the highest accuracy rate of the four.

Abah., et al., (2015) designed a model called HOSBAD, this is a machine learning approach for the detection of malware on Android platforms. The system monitors and extracted features from the applications while in execution and used them to perform in-device detection using a trained K-Nearest Neighbour classifier. Their results showed performance in the detection rate of the classifier with accuracy rate of 93.75%, error rate of over 6% and a claim of low false positive rate which was not specified. KNN could only work on small data set and cannot guarantee realistic assessment. As there is possibility of some malware to evade detection, The error rate of 6% is high.

Rashidi, et al., (2018) worked on a framework for the detection of Android malicious application that was based on Support Vector Machine (SVM) and Active Learning technologies. In order to build an active learning model, the authors made use of expected error reduction query strategy so

as to combine Android malware new informative instances and to retrain the model in order to be able to do adaptive online learning. To evaluate their model, the authors utilized the DREBIN benchmark malware dataset via a set of experiments and their findings revealed that their framework could detect new malware more accurately.

Saracino, et al., (2018) improved on MADAM (Dini et al., 2012), using machine learning approach and KNN as the classifier. The accuracy of MADAM was improved from 93% to 96%. And the rate of false alarm was also reduced to 6.5%. And KNN does not also work well with large dataset. Hoang and Nguyen (2018), worked on botnet detection model based on machine learning using Domain Name Service query data and evaluates its effectiveness using four popular machine learning techniques (K –Nearest Neighbour, Random Forest, Decision Tree and Naives Bayes). Their results show that machine learning algorithms can be used effectively in botnet detection. The model achieved the average classification accuracy over 85%, and False Positive rate of 17.10%. They suggested that in the future, larger datasets can be used to analyse the effects of the domain name features on the detection accuracy, as well as new features to improve the detection accuracy of the proposed model.

Closely related to this study, is the work of Hoang and Nguyen (2018). The reason for the choice is that their work was based on botnet detection and they used four Machine Learning Techniques (K –Nearest Neighbour, Random Forest, C45 Decision Tree and Naives Bayes). Their results showed performance in the detection rate of the classifier with accuracy rate of 82.9%, error rate of over 17.1%. Bezerra, et al., (2019) proposed a host-based method for the detection of Botnets in Internet of Things (IoT) Devices using One-Class Classification (OCC) approach that was able to model only the legitimate behaviour of a device in order to detect any deviations. The proposed system is underpinned by a novel agent-manager architecture based on HTTPS, which is able to stop the IoT device from being overloaded by the training activities. The One-Class algorithms evaluated are Elliptic Envelope, Isolation Forest, Local Outlier Factor, and One-Class Support Vector Machine (SVM).

Gyunka and Barda (2020), used the idea of Bezerra, et al., (2019), whose work was based on OC-SVM, but instead of SVM they decided to use KNN classification approach and developed a normality model that is based on One-Class K-Nearest Neighbour (OC-kNN) Machine Learning approach for anomaly detection of Android Malware. The OC-kNN was trained, using WEKA 3.8.2 Machine Learning Suite, through a semi-supervise procedure that contained mostly benign and a very few outliers Android application samples. The OC-kNN had 88.57% true performance accuracy for normal instances while 71.9% was recorded as true performance accuracy for outliers (unknown) instances. The false alarm rates for both normal and outlier's instances were recorded as 28.1% and 11.5%. The model recorded a high false positive rate because the KNN classifier works better on small dataset and it is sensitive to the scale of data and irrelevant features (Marina, 2020).

According to Bohutska (2021), the percentage of false positive (FPR) should not be higher than 27.8%, but can be lower and the existing system achieved a lower FPR. This proposed system will also use combination of four machine learning algorithms, (K –Nearest Neighbour (KNN, Random Forest (RF), Support Vector Machine (SVM) and Classification and Regression Tree (CART). It will substitute C45 decision tree and Naïve Bayes with SVM and CART, the main difference here is the CART whose strength is very important in revealing important data relationship. It automatically searches for pattern and uncover hidden structure in highly complex way. This property is very important to Malware detection and surprisingly it is not a favourite for researchers. The generated results will be compared to the existing.

3. Proposed Model AKANDO

The architectural design for this study is as shown in Figure 1.

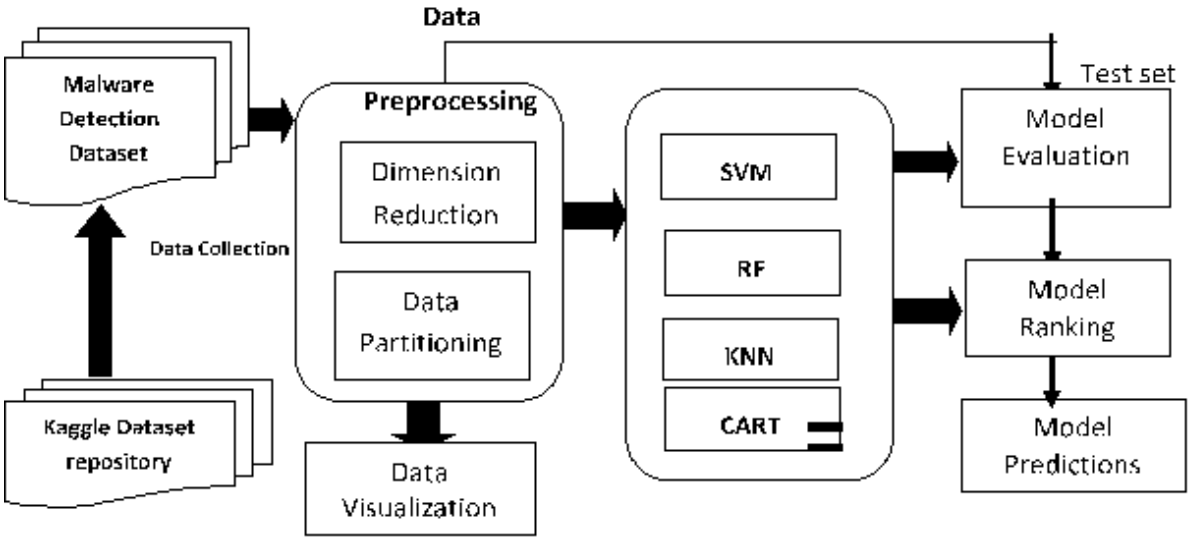

Figure 1: The Proposed Model

AKANDO adopts a two-pronged approach to feature extraction, leveraging both static and dynamic analysis. Static analysis examines the app's code to extract features such as permissions requested, network calls made, and API usage patterns. Dynamic analysis involves executing the app in a controlled environment to monitor its runtime behavior, including system call frequency, network traffic analysis, and resource consumption. This combined approach provides a more comprehensive view of an app's functionalities, enabling the detection of malicious behavior that might be hidden within static code or only emerge during execution (Wang et al., 2019).

AKANDO employs a multi-layered neural network architecture to learn complex relationships between the extracted features and botnet behavior. The network architecture typically consists of an input layer that receives the extracted features, multiple hidden layers for learning intricate patterns, and an output layer that classifies the app as benign or malicious. The model is trained on a labelled dataset consisting of known botnets and benign apps. During the training process, the weights and biases within the neural network are adjusted to minimize the discrepancy between the predicted botnet probability and the actual label (Xu et al., 2018).

The pre-processed features are then concatenated to form a combined feature vector at time t as shown in equation 1:

$$X\_t = [S', D'\_t] \quad (1)$$

This vector is then fed into a multi-layered neural network architecture. We can represent a single neuron in a layer l with the following activation function as shown in equation 2:

$$a^{(l)}(i) = \sigma(\Sigma\, w^{(l)}(i,j) * x^{(l-1)}(j) + b^{(l)}(i)) \quad (2)$$

where:

- $a^{(l)}(i)$ represents the activation of the i-th neuron in layer l
- σ denotes the activation function (e.g., sigmoid, ReLU)
- $w^{(l)}(i,j)$ represents the weight between the j-th neuron in layer l-1 and the i-th neuron in layer l

- x^(l-1)(j) represents the output of the j-th neuron in layer l-1
- b^(l)(i) represents the bias term of the i-th neuron in layer l

The weights and biases are learned during the training process by minimizing a loss function. A common choice for the loss function in binary classification problems like botnet detection is the binary cross-entropy as shown in equation 3:

L(t) = - (P(A_t | Botnet) * log(y_t) + (1 - P(A_t | Botnet)) * log(1 - y_t)) (3)

where:

- L(t) represents the loss at time t
- y_t represents the ground truth label (1 for botnet, 0 for benign)

3.1 Core Functionalities of AKANDO

The effectiveness of AKANDO's machine learning component relies on several mathematical concepts:

Neurons within the hidden layers of the neural network employ activation functions to introduce non-linearity. This allows the network to learn complex relationships between the input features and the output (botnet classification). Common activation functions used in botnet detection include the sigmoid function and the rectified linear unit (ReLU) function (Goodfellow et al., 2016).

During the training process, the model's performance is evaluated using a loss function. The binary cross-entropy loss function is typically used for binary classification problems like botnet detection. It measures the discrepancy between the predicted botnet probability (output of the neural network) and the actual label (0 for benign, 1 for botnet). The loss function is minimized through gradient descent optimization algorithms, which iteratively adjust the weights and biases within the network (James et al., 2013).

Concepts relevant to the optimization and performance of the machine learning model. These include techniques like L1/L2 regularization or feature selection can be employed to identify features that significantly influence the model's decisions. This can help improve model interpretability and potentially lead to feature engineering for better performance (Guyon and Elisseeff, 2003).

Furthermore, regularization methods like L1/L2 regularization penalize large weight values in the neural network. This helps prevent overfitting, a phenomenon where the model performs well on the training data. AKANDO also benefit from incorporating four other machine learning algorithms for a more comprehensive detection strategy: support vector machine (SVM), random forest (RF), k-nearest neighbor (KNN), and classification and regression tree (CART).

(i) Support Vector Machine (SVM)

A non-linear binary support vector machine (SVM) is used for malware detection on android devices. The SVM training dataset is formulated as shown in Equation 4 and 5:

$$(x_1, y_1) \ldots\ldots (x_n, y_n) \quad (4)$$

Where: x is the feature set, and y is the class label

$$x_i = x_i^1, x_i^2, \ldots\ldots x_i^d \quad (5)$$

where: x_i^d is a real value, and $y_i = \{-1, 1\}$

Radial Basis Function Kernel (RBF) function is used to map the non-separable training data from input space to feature space in order to find an optimimal hyperplane that correctly segregates the data. The RBF kernel is presented in Equation 6 and 7:

$$K(\overrightarrow{x_i}, \overrightarrow{x_j}) = \emptyset(\overrightarrow{x_i})^T \emptyset(\overrightarrow{x_i}) \quad (6)$$

$$K(\overrightarrow{x_i}, \overrightarrow{x_j}) = \exp\left(-\gamma \left|\left|x_i - x_j\right|\right|^2\right) \quad (7)$$

Where: $\gamma: \frac{1}{2\sigma^2} > 0$, x_i are the support vector points, x_j are the feature vector points in the transformed space, $K(\overrightarrow{x_i}, \overrightarrow{x_j})$ is the kernel function.

The kernel function calculates the dot product of the mapped data points in the transformed feature space.

The optimal hyperplane that segregates between the two classes is found using Equation 8.

$$w^T.x + b = \sum_{i=1}^{l} \alpha_i \, y_i \, \emptyset(\overrightarrow{x_i})^T \emptyset(\overrightarrow{x_i}) + b = 0 \quad (8)$$

The classification frontiers are found by the following equation 9 and 10:

$$w \, \emptyset(x) + b = 1 \quad (9)$$

$$w \, \emptyset(x) + b = -1 \quad (10)$$

The optimal weight vector (w) is given by equation 11:

$$\vec{w} = \sum_{i=1}^{l} \alpha_i \, y_i \, \emptyset(\overrightarrow{x_i}) \quad (11)$$

The dual formulation of SVM algorithm is used in this work. This formulation which is presented as a maximization problem over α is shown in Equation 12:

$$\max \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \, \alpha_j \, y_i \, y_j \, \emptyset(\overrightarrow{x_i})^T \emptyset(\overrightarrow{x_i}) \quad (12)$$

Subject to: $0 < \alpha < C$ and $\sum_{i=1}^{l} \alpha_i \, y_i = 0$

Where: α_i is the weight vector, y is the label vector, $\emptyset(\overrightarrow{x_i})^T \emptyset(\overrightarrow{x_i})$ is the kernel function, and C is the intercept.

The decision function g used in making prediction is given in equation 13:

$$g(\vec{x}) = sgn(\vec{w}^T \vec{x} + b) \Rightarrow sgn\left(\sum_{i=1}^{l} \alpha_i \, y_i \, \emptyset(\overrightarrow{x_i})^T \emptyset(\overrightarrow{x_i}) + b\right) \quad (13)$$

Where: $g(\vec{x})$ is the predicted label, sgn is the sign of $(\vec{w}^T \vec{x} + b)$, and α_i is the weight vector.

(ii) Random Forest (RF)

Random forest is a good option for regression and best known for its performance in classification problems. Furthermore, it is a relatively easy model to build and does not require much hyperparameter tuning. This is because the main hyperparameters are the number of trees in the forest and the number of features to split at each leaf node.

a. Random Forest Classification Algorithm

STEP 1: Randomly select k features from total m features.

Where k << m

STEP 2: Among the "k" features, calculate the node "d" using the best split point.

STEP 3: Split the node into daughter nodes using the best split.

STEP 4: Repeat 1 to 3 steps until "l" number of nodes has been reached.

STEP 5**:** Build forest by repeating steps 1 to 4 for "n" number of times to create "n" number of trees**.**

b. Random Forest Regression Algorithm

STEP 1: Take the test features and use the rules of each randomly created decision tree to predict the outcome and store the predicted outcome (target)

STEP 2: Calculate the votes **(**majority voting**)** for each predicted target.

STEP 3: Consider the high voted predicted target as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created trees.

(iii) K-Nearest Neighbours (KNN)

The KNN algorithm is based on the observation that a sample with features that are similar to the ones of points of one particular class belongs to that class. These points are known as nearest neighbors. K represents the number of training data points lying in proximity to the test data point which is used to find the class. The parameter k specifies the number of neighbors (neighboring points) used to classify one particular sample point. Finally, the assignment of a sample to a particular class is done by having the k neighbors considered to "vote". In this fashion, the class represented by the largest number of points among the neighbors ought to be the class that the sample belongs to.

Consider k as the desired number of nearest neighbours and $D = P_1, \ldots, P_n$ be the set of training samples in the form $P_1 = (x_i, c_i)$, where x_i is the d-dimensional feature vector of the point P_i and c_i is the class that P_i belongs to. For each p' = (x', c') we compute the distance $d(x',xi)$ between p' and all p_i belonging to D, then sort all points p_i according to the key $d(x',xi)$. Next we select the first k points from the sorted list, those are the k closest training samples to p', and then assign a class to p' based on majority vote, see equation 14:

$$c' = argmax_y \sum(x_i, c_i) \text{ belonging to } D, I(y = c_i) \qquad (14)$$

(iv) Classification and Regression Tree (CART)

A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps in decision making. Decision Tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and number of attributes in the given data.

Decision tree algorithm is based on the following steps;

STEP 1: Select the best attribute using Attribute Selection Measures (ASM) to split the records.

a. Information Gain

b. Entropy (for ID3) and Gini Impurity (for CART)

c. Gain

STEP 2: Make that attribute a decision node and break the dataset into smaller subsets.

STEP 3: Start tree building by repeating this process recursively for each child until one of the condition will match:

a. All the tuples belong to the same attribute value

b. There are no more remaining attributes

c. There are no more instances.

The Gini impurity is given in equation 15:

$$Gini = 1 - \sum_{i=1}^{n}(pi)^2 \quad (15)$$

Where pi is the probability of an object being classified to a particular class.

Entropy is given in equation 16:

$$E = -\sum_{i=1}^{n} pi\ log_2\ (pi) \quad (16)$$

where pi is the probability of randomly selecting an example in class i. Information gain is given in equation 17:

$$Information\ Gain = \ Entropy_{parent} - Entropy_{children} \quad (17)$$

$Entropy_{parent}$ is the entropy of the parent node and $Entropy_{children}$ represents the average entropy of the child nodes that follow this variable.

3.2 Evaluation Metrics of AKANDO

Since AKANDO utilizes a machine learning approach for Android botnet detection, the effectiveness of AKANDO's machine learning component relies on several mathematical concepts:

Accuracy: This metric represents the overall correctness of the model in classifying apps as benign or malicious. It is calculated as the ratio of correctly classified samples to the total number of samples (see equation 18):

Accuracy = (True Positives + True Negatives) / (Total Samples) (18)

- True Positives (TP): Number of malicious apps correctly classified as malicious.
- True Negatives (TN): Number of benign apps correctly classified as benign.

Precision: This metric measures the proportion of identified malicious apps that are actually malicious. It is calculated in equation 19:

Precision = TP / (TP + False Positives) (19)

- False Positives (FP): Number of benign apps incorrectly classified as malicious.

Recall: This metric measures the proportion of actual malicious apps that are correctly identified by the model. It is calculated in equation 20:

Recall = TP / (TP + False Negatives) (20)

False Negatives (FN): Number of malicious apps incorrectly classified as benign.

F1-Score: This metric combines precision and recall into a single value, providing a balanced view of the model's performance. It is calculated as shown in equation 21:

F1-Score = 2 * (Precision * Recall) / (Precision + Recall)) (21)

False Alarm Rate (FAR): This metric specifically focuses on the rate of benign apps being misclassified as malicious. It is calculated as shown in equation 22:

FAR = FP / (FP + TN) (22)

where FN: Number of malicious apps incorrectly classified as benign (False Negatives) and TP: Number of malicious apps correctly classified as malicious (True Positives).

Matthew's Correlation Coefficient (MCC): This metric takes into account all four confusion matrix values (TP, TN, FP, FN) and provides a more balanced assessment of the model's performance compared to accuracy alone. It ranges from -1 (perfect disagreement) to +1 (perfect agreement). MCC values closer to +1 indicate better performance as shown in equation 23:

MCC = (TP*TN - FP *FN) /sqrt((TP + FP)*(TP + FN)*(TN + FP)*(TN + FN)) (23)

Where TP, TN, FP, FN: As defined previously (True Positives, True Negatives, False Positives, False Negatives).

G-mean: This metric is the geometric mean of precision and recall, offering a balanced view between correctly identifying malicious apps (recall) and avoiding false positives (precision). It is particularly useful when dealing with imbalanced datasets, where the number of benign apps might significantly outweigh malicious ones as shown in equation 24:

G-mean = sqrt(Precision * Recall) (24)

Sigmoid Function: The sigmoid function maps an input value between negative infinity and positive infinity to an output between 0 and 1. It is defined as shown in equation 25:

f(x) = 1 / (1 + e^(-x)) (25)

where e is the base of the natural logarithm (approximately 2.71828).

Rectified Linear Unit (ReLU) Function: The ReLU function outputs the input directly if it's positive and outputs 0 otherwise. It is defined as shown in equation 26:

f(x) = max(0, x) (26)

Binary Cross-Entropy Loss Function: During training, the binary cross-entropy loss function measures the discrepancy between the predicted botnet probability (p) and the actual label (y) (0 for benign, 1 for botnet). It is defined as shown in equation 27:

loss = - (y * log(p) + (1 - y) * log(1 - p)) (27)

Feature Importance Analysis: Techniques like L1/L2 regularization add a penalty term to the loss function based on the magnitude of the weights (w) in the neural network. This encourages the model to assign lower weights to less important features. The L1 regularization penalty term is defined as shown in equation 28:

penalty = λ * ||w||_1 (28)

where λ is a hyperparameter controlling the strength of the penalty and ||w||_1 denotes the L1 norm, which is the sum of the absolute values of all weights. Similarly, L2 regularization uses the L2 norm (sum of squares of weights) as shown in equation 29:

penalty = λ * ||w||_2^2 (29)

By incorporating these penalties into the loss function, the model is discouraged from assigning overly large weights to any specific feature, promoting a more balanced and interpretable model.

3.3 Benefits of AKANDO

(i) **Improved Accuracy**

The hybrid feature extraction approach combined with the machine learning model's learning capabilities has the potential to achieve superior botnet detection accuracy compared to methods that rely solely on static or dynamic analysis. By leveraging a more comprehensive feature set, AKANDO can potentially identify intricate patterns indicative of malicious behavior.

(ii) **Adaptability**

The use of a multi-layered neural network architecture allows AKANDO to adapt to evolving botnet tactics. As botnet operators develop new techniques to evade detection, the model can be re-trained with fresh data to maintain effectiveness. This continuous learning capability enhances the long-term sustainability of the detection system.

(iii) **Reduced False Positives**

The combined analysis approach and the focus on minimizing false positives during model training aim to reduce the number of legitimate apps flagged as malicious. This is crucial for maintaining user trust and avoiding unnecessary disruptions.

Limitations:

(i) **Data Availability**

The effectiveness of AKANDO relies heavily on the quality and size of the training dataset. A comprehensive dataset encompassing diverse botnet samples and a wide range of benign apps is essential for optimal model performance. However, obtaining such data can be challenging due to privacy concerns and the dynamic nature of botnet threats.

(ii) **Computational Cost**

While potentially more accurate, the hybrid analysis approach involving both static and dynamic analysis could be computationally expensive compared to methods relying solely on static analysis. This may necessitate resource optimization techniques or hardware acceleration to ensure efficient operation on resource-constrained mobile devices.

(iii) **Evasion Techniques**

Botnet operators might develop new techniques to evade detection by AKANDO. These could include code obfuscation, dynamic code loading, or exploiting vulnerabilities in the runtime environment. Continuous research and adaptation of the detection system are crucial to maintain effectiveness against evolving threats.

4. Future Research Directions

Although the theoretical foundations of AKANDO appear promising, thorough real-world testing on large-scale datasets is essential to validate its effectiveness in practical scenarios. This evaluation in our next publication will involve diverse botnet samples and various benign applications to comprehensively assess the model's accuracy and generalizability.

Exploring alternative machine learning architectures, such as recurrent neural networks (RNNs), could be beneficial for analyzing sequential data like network traffic patterns. RNNs might offer improved capabilities in identifying botnet activity hidden within communication sequences.

Integrating contextual information into the model, such as user behavior patterns or app usage statistics, could potentially enhance detection accuracy. By understanding how users interact with

apps, the system might be better equipped to differentiate between malicious and legitimate behavior.

Implementing federated learning could enable continuous model improvement without compromising user privacy. Federated learning allows training on distributed datasets residing on individual devices, protecting sensitive user data while still allowing the model to learn from a broader range of samples (Yang et al., 2019).

Conclusion

AKANDO presents a promising approach for Android botnet detection by combining static and dynamic analysis with machine learning. A critical aspect of AKANDO is its emphasis on minimizing false positives. False positives occur when a legitimate app is mistakenly flagged as malicious. Its hybrid feature extraction, multi-layered neural network architecture, and focus on minimizing false positives offer potential advantages over existing methods. This can significantly impact user experience and erode trust in the detection system. While limitations exist regarding data availability, computational cost, and potential evasion techniques, future research directions aim to address these concerns and further enhance AKANDO's effectiveness.

Continued exploration and optimization of this hybrid approach will significantly contribute to the ongoing fight against botnets on Android devices. By combining static and dynamic analysis, AKANDO aims to create a more robust feature set that can effectively differentiate between malicious and benign behavior. Additionally, the machine learning model can be fine-tuned to prioritize accuracy while minimizing false positives.

REFERENCES

1. Abah, J., Waziri, O.V., Abdullahi, M.B., Arthur, U.M., and Adewale, O. S (2015). A Machine Learning Approach to Anomaly-Based Detection on Android Platforms. *International Journal of Network Security and Its Applications (IJNSA)* Vol.7, No.6, November 2015 DOI: 10.5121/ijnsa.2015.7602 15
2. Barriga, J. J. and Yoo, S.G. (2017). Malware detection and Evasion with Machine Learning Techniques: A survey. International *Journal of Applied Engieering Research* 12(18).7207 - 7214.
3. Bezerra, V. H., Da-Costa, V.G.T., Barbon-Junior, S., Miani, R. S. and Zarpelão, B.B. (2019). IoTDS: *A One-Class Classification Approach to Detect Botnets in Internet of Things Devices*, Sensors (Switzerland), vol. 19, no. 14, pp. 1–26, 2019.
4. Bohutska J. (2021). How to tell good performance from bad. https: //towardsdatascience.com/anomalydetection/
5. Canfora, G., Medvet, E., Mercaldo, F. and Visaggio, C.A. (2015). *Detecting Android Malware Using Sequences of System Calls*," in Proc. 3rd Int. Work. Softw. Dev. Lifecycle Mob., pp. 13–20, 2015.
6. Dini, G.,Martinelli, F.,Saracino, A. and Sgandurra, D.(2012). MADAM: a Multi-Level Anomaly Detector for Android Malware. Detection Framework for Android Devices. *Journal of Intelligent Information Systems*, pp. 1-30. doi:10.1007/s10844-010-0148-x.
7. Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.
8. Guyon, I., and Elisseeff, A. (2003). An introduction to variable selection. *Journal of Machine Learning Research*, 3(Mar), 1157-1182.

9. Gyunka,B. A. and Barda,S. I.(2020). Anomaly Detection of Android Malware Using One-Class K-Nearest Neighbours (Oc-Knn). Nigerian Journal of Technology (NIJOTECH)/Vol. 39, No. 2, April 2020, pp. 542 - 552 http://dx.doi.org /10.4314/njt.v39i2.25
10. Hoang, X.D. and Nguyen, Q. C. (2018). Botnet Detection Based On Machine Learning Techniques Using DNS Query Data. *Future Internet*, 10 (43); doi: 10.3390/fi10050043 www.mdpi.com/journal/futureinternet
11. James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*. Springer Science and Business Media.
12. Kim, D., KIm, J. and Kim, S. (2013). *A malicious application detection framework using automatic feature extraction tool on Android market.*" in Proc. of the 3rd International Conference on Computer Science and Information Technology, ICCSIT , 2013.
13. Kurniawan, H., Rosmansyah, Y and Dabarsyah, B. (2015). *Android anomaly detection system using machine learning classification.In Electrical Engineering and Informatics (ICEEI)*, 2015 International Conference on, pages 288–293, Aug 2015.doi: 10.1109/ICEEI.2015.7352512.
14. Marina, C. (2020). A Quick Introduction to KNN Algorithm Retrieved 24/7/21. from https://www.greatary.in.com
15. Rashidi, B., Fung, C. and Bertino, E.(2018). *Android malicious application detection using support vector machine and active learning*. 13th International Conference on Network and Service Management, CNSM 2017, 2018, vol. 2018-Janua.
16. Saracino, A., Sgandurra, D., Dini, G., and Martinelli, F. (2018). *Madam: Effective and Efficient Behaviour - based Android Malware Detection and Prevention*. IEEE Transactions on Dependable Security Computing. vol 99 10-20
17. Statcounter (2023). Mobile Operating System Market Share Nigeria Dec.2022 –Dec.2023. Retrieved 31st August 2023, from https://gs.statcounter.com/os-market-share/mobile/nigeria
18. Vuleta, B. (2023). Worrying Malware Statistics to Take Seriously in 2023. Retrieved 08/10/2023 from:https://legaljobs.io/blog/malware-statistics/
19. Wang, W., Zhao, M., Gao, Z., Xu, G., Xian, H., Li, Y. and Zhang, X. (2019). *Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and directions. Special Section on Security and Privacy for Cloud and IOT*. IEEE Access. Digital Object Identifier 10.1109/ACCESS.2019.2918139
20. Xu, W., Zhao, L., Qi, H., Li, G., and Gong, J. (2018). Detecting Android malware using multi-level attention based BLSTM network. arXiv preprint arXiv:1808.08118.
21. Yang, Q., Liu, Y., Zhao, T., Li, N., and Zhu, G. (2019). Federated learning for mobile medical image analysis. **IEEE Intelligent Systems and their Applications*, 35(1), 12-25.
22. Yerima, S.Y. Sezer, and I. Muttik. (2021) *Android Malware Detection Using Parallel Machine Learning Classifiers*, 2014 Eighth Int. Conf. Next Gener. Mob. Apps, Serv. Technol., no. Ngmast, pp. 37–42, 2016.
23. Yerima, S.Y., Sezer,s., McWilliams, G. and Muttik, I. (2013). *A New Android Malware Detection Approach Using Bayesian Classification*, in Advanced Information Net- working and Applications (AINA), 2013 IEEE 27th International Conference, pages 121–128, March 2013.