applied sciences

MDPI

Article

Dynamic IoT Malware Detection in Android Systems Using Profile Hidden Markov Models

Norah Abanmi, Heba Kurdi * and Mai Alzamel

Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh P.O. Box 145111, Saudi Arabia
* Correspondence: hkurdi@ksu.edu.sa

Abstract: The prevalence of malware attacks that target IoT systems has raised an alarm and highlighted the need for efficient mechanisms to detect and defeat them. However, detecting malware is challenging, especially malware with new or unknown behaviors. The main problem is that malware can hide, so it cannot be detected easily. Furthermore, information about malware families is limited which restricts the amount of "big data" that is available for analysis. The motivation of this paper is two-fold. First, to introduce a new Profile Hidden Markov Model (PHMM) that can be used for both app analysis and classification in Android systems. Second, to dynamically identify suspicious calls while reducing infection risks of executed codes. We focused on Android systems, as they are more vulnerable than other IoT systems due to their ubiquitousness and sideloading features. The experimental results showed that the proposed **D**ynamic **I**oT malware Detection in Android Systems using **P**HMM (DIP) achieved superior performance when benchmarked against eight rival malware detection frameworks, showing up to 96.3% accuracy at 5% False Positive Rate (FP rate), 3% False Negative Rate (FN rate) and 94.9% F-measure.

Keywords: cybersecurity; Internet of Things; Markov Model; Android; malware detection

Citation: Abanmi, N.; Kurdi, H.; Alzamel, M. Dynamic IoT Malware Detection in Android Systems Using Profile Hidden Markov Models. *Appl. Sci.* **2023**, *13*, 557. https://doi.org/10.3390/app13010557

Academic Editor: Katarzyna Antosz

Received: 2 December 2022
Revised: 27 December 2022
Accepted: 27 December 2022
Published: 31 December 2022

1. Introduction

With the escalating popularity of Internet of Things (IoT) networks in remote sensing applications, strong security concerns are rising as IoT networks are vulnerable to security breaches due to the variability in data formats and technical standards of their devices. Therefore, many researchers are increasingly investigating robust cybersecurity methods to ensure secure remote sensing through a trusted IoT infrastructure [1,2]. The IoT is a network of physical objects that can communicate with one another and exchange data over the Internet, or a network [3,4]. The IoT environment includes smart gadgets, such as mobile phones, thermostats and baby monitors, which are characterized by their limited computational resources and lack of the necessary built-in security controls. As a result, IoT environments are facing many cyber-security threats which have been reported by several studies [5]. Android-based IoT is particularly targeted by malware attacks due to its popularity and flexibility in allowing sideloading of apps [6]. It should come as no surprise that every 10 s, malicious software appears on an Android device [6] and Android handsets are the target of over 98% of mobile banking assaults.

Several traditional methods to detect security threats, particularly anti-malware threats on Android-based IoT devices, have been proposed [7,8]. However, such threats have become increasingly more complicated, making it vital to find an effective solution for Android malware detection [9]. Therefore, in this paper, we focus on developing a malware detection approach for Android systems to provide an accurate solution that reduces False Positive (FP) and False Negative (FN) rates.

Malware can be analyzed using static or dynamic approaches. The static analysis examines the malware without executing the code. It allows the users to extract control flow

graphs, opcodes, system calls, and Application Program Interface (API) calls as features. The benefit of static analysis is that it can detect malware at lower FP and higher speed than dynamic analysis. However, it depends on one feature although malware execution involves a set of behavior chains and it falls behind in the case of obfuscation. Additionally, static analysis requires a regular update of malware databases to detect recent forms. For instance, ProDroid, which is a recent rival static malware detection framework for Android devices [10] based on a Profile Hidden Markov Model (PHMM), shows promising results, in terms of accuracy, precision and recall. Nevertheless, it has difficulty detecting undisclosed malware [11] in contrast to dynamic analysis which examines the malware while executing the code to allow detection of unknown malware and recording user activities [12]. On the other hand, dynamic analysis exposes the system to a high security risk where the system can be infected by malware while being executed.

Therefore, in this paper, we propose DIP: a Dynamic IoT malware detection model using PHMM for safe dynamic malware detection. DIP analyzes the malware at run time while reducing the infection risks. This is possible because PHMM is known for its high prediction accuracy at low risks due to its delete or non-emitting states [11]. After implementing the proposed DIP approach, we compared its performance with several state-of-the-art Android malware detection frameworks and demonstrated its effectiveness in detecting hidden malware codes by analyzing their behaviors even if obfuscation or any other hiding techniques are used.

The main contributions of this paper can be summarized as follows: First, it proposed and implemented a dynamic malware detection method (DIP) based on PHMM to insure safe analysis during runtime. Second, it extensively tested the proposed method considering seven performance measures: detection rate, FP rate, FN rate, precision, recall, accuracy and F-measure. Third, it benchmarked the performance of the proposed DIP approach against eight rival malware detection frameworks and demonstrated its superiority in terms of accuracy, precision and recall. Despite our proposal strengths, it has some limitations. In the first place, it focused solely on Android IoT systems. Furthermore, it used the Derbin dataset that was not updated recently. In addition, results can sometimes be affected by the FP problem, so the system may classify a specific behavior as malicious while it is benign.

The rest of the paper is structured as follows: Section 2 provides some background information related to this research. Section 3 reviews some related work. Section 4 details the methodology followed to develop the proposed model, while Section 5 illustrates the experimental results and benchmarks them against eight-malware detection approaches. Section 6 concludes the paper and indicates possible future directions to address the above limitations of this research.

2. Background

This section provides an overview of the most relevant technical background information related to the model proposed in this paper which include Markov chain, Hidden Markov Model (HMM), PHMM, Multiple Sequence Alignment (MSA) algorithm, Fasta file and Astrotite compressed Archive Format (afa).

2.1. Markov Chain, HMM, and PHMM

The Markov chain is a type of stochastic process that is a probabilistic mathematical model that evolves. This process depends on the previous state to get the outcome of the current state [12]. The Markov chain is generally considered to be a discrete-time stochastic process that describes a set of random variables which change over time and defines how these variables change [10]. According to the Markov chain model, the sequence of API calls can be represented as a graph, in which each node represents the API and the edges connecting one node to another represent the methods invoked by that node. Moreover, each edge is labeled with the probability of that transition [13].

In contrast to the Markov chain, the HMM has indirectly observable states. HMM can be viewed as a machine learning model; specifically, as a discrete hill-climbing method

which is a statistical strategy for modeling systems that are supposed to contain Markov processes with hidden states. We can train an HMM on a given observation sequence. We can also score an observation sequence against a specifically trained HMM model to determine the probability of observing such a sequence under the constraints of the specified model. HHM is utilized to determine the probability $P(O \mid \lambda)$, where $\lambda = (A, B, \pi)$ is a model and O is an observation sequence. That is, an observation sequence can be scored to see how well it fits a given model. The higher the score, the greater the match between the observation sequence and the observation data used to train the model. Another benefit of HHM is decoding, which is defined as given an HMM model and an observation sequence O, we can determine the optimal sequence of state transitions X associated with O. That is, we can uncover the "best" hidden state sequence, that is, we can maximize the expected number of correct states Xi. This is in contrast to a dynamic program, which yields the states Xi corresponding to the highest scoring path [14,15].

PHMM is a more advanced type of HMM that can be used to model sequence similarities. To develop a profile of the model, PHMM uses positional information and null transitions from observation sequences. To represent the system, PHMM provides three states: Match, Insertion, and Deletion. The hidden states in the model are identified using the positional information from the observed collection of sequences. The probability distribution for state transitions and output sequences is calculated and applied to construct a model profile, The match and insert are referred to as "emission states" because they emit symbols called observation sequences. The delete state allows the model to shift between match and insert states without emitting any emissions. Each transition has a probability associated with it, which influences the transition from one state to the next called the Start and End states [16]. A primary step in the PHMM is MSA to obtain a similarity score. The unknown file of the input, which is a sequence of symbols, is presented for a generated model from MSAs produced by each malware family to determine whether it belongs to that malware family [11].

2.2. MSA Algorithm, Fasta Files and Afa Format

The MSA algorithm was developed mainly to locate the common segments between three or more biological sequences that consist of general proteins, DNAs, or RNAs [17]. In the proposed approach, we used the algorithm to find the common API call information of malware variants in a family and extract a representative API pattern of the family. To produce an MSA file for each malware family we used the MUSCLE tool. MUSCLE tool is a free computer program for creating multiple alignments of protein sequences proposed by [18]. The tool depends on MSA Algorithm that estimates the distance and uses a new profile function for progressive alignment. For refinement in MSA, the tree-dependent restricted partitioning method can be applied [18] to provide better average accuracy and better speed than other superior MSA tools. The tool takes the API call sequences file in a Fasta format to produce a file in afa format.

The Fasta format is developed by [19]. It is a text-based format for representing multiple nucleotides or protein sequences. Fasta files start with a ">" character and a single-line description of each sequence, followed by lines of sequence data. .afa extension is a data compression and encryption file that is implemented by Fantiusen Software for their Application Astrotite 200X. The Astrotite software is designed mainly to store multiple files securely into a single archive, keeping the same hash with the same compressed files.

3. Related Work

Various studies have proposed models for malware detection in general [8,20,21], and IoT malware detection in particular [22–28] based on different approaches. However, recently, increased attention is paid to malware detection using Markov chain models. In this section, we review some studies that used Markov chains for static [11,16,29,30] and dynamic malware analysis [13,31–38].

Onwuzurike et al., 2019 [29] proposed an Android malware detection tool that takes the sequence of API calls performed by an Android application to detect malware behavior based on the Markov chain. For benign datasets, they used PlayDrone and Google Play store, and for malware datasets they used VirusShare. The results showed that the proposed system achieved 99% in F-measure for malware detection.

Alipour & Ansari, 2020 [11] presented a method that identifies malware families by combining signature-based detection with machine learning-based methods. They applied PHMM based on opcode sequences with five training datasets (Cygwin, MPCGEN, VCL32, G2, and NGVCK) to extract the family's paramount features and the canonical sequences created in the process of MSA production. The results showed that the proposed method outperformed other HMM-based techniques. However, this model has two main drawbacks which are: requiring excessive human intervention, and the inability to deliberate other static features.

Sasidharan and Thomas, 2021 [16] proposed an approach for malware detection and classification based on a PHMM model that was trained using the Baum Welch algorithm. to solve the code complication problem of static API analysis, and the results show the proposed approach had a detection accuracy of 94.5% with a 7% FP.

Anandhi et al., 2021 [30] proposed an approach for detecting and classifying malicious executables by visualizing malware as Markov images. They extracted textures from Markov images using the Gabor filter, and then they developed a model using CNN-VGG-3 and fine-tuned DenseNet to detect malware in real time. They used two datasets, i.e., Malimg and BIG2015, for training and evaluation. The results showed that DenseNet with Gabor Markov has a 99.94% F1 measure on the Malimg dataset and 98.98% on the BIG2015 dataset. They had 99.37% F1-measure of classification malware on Malimg and 98.88% on BIG2015. In addition, they found that the proposed approach performed better in detection, classification, and execution time than the other methods that they considered.

Generally, the above reviewed literature was proposed for static malware analysis. The chief advantage of static analysis over dynamic analysis is that it is free from the overheads associated with program execution. However, the static analysis methods fail to capture the runtime environment and cannot detect unknown malware [39]. Hence, various researchers have introduced dynamic methods to support unknown malware detections. Ahsan-Ul-Haque et al., 2018 [31] developed a dynamic model to detect android malware by monitoring the behavior of malware using the Markov model to extract features of system calls. They applied the Gaussian Bayes classifier to Genome Project and ArgusLab malware datasets and Google Play Store datasets. The results showed that the developed model improves the detection accuracy and outperformed the other tested models.

With the help of a Markov chain model, Salehi and Amini proposed a method for tracking application requests for system services in 2017 [38]. Their system detects malicious applications directly on mobile devices and classifies them either malicious or benign. On the Derbin dataset, this system performed 96 percent accuracy in detecting Android malware using the Random Forest classifier. However, this system can only do binary classification; it does not detect the malware family.

Ficco, 2019 [13] applied Markov chains on sequences of API calls to extract features to detect malware dynamically. The researchers used Principal Component Analysis (PCA) to perform feature selection and reduction of the feature space by using the mobSF tool. They used four datasets, i.e., VirusShare, Malgenome and Contagio Minidump for malware, and Google Play store for good ware and applied four machine learning algorithms to perform the quantitative analysis i.e., Naive Bayes, Decision Tree, Random Forest, and Support Vector Machine. The results showed that the best classifier for the accuracy of malware detection is Naive Bayes and the proposed model can detect malware with up to 89% F-measure.

Alahmadi et al., 2020 [32] proposed a bot detection system that models the bot network flow behavior as a Markov chain to extract high-level flow features. The extracted features are used to detect flows produced by bots and classify the bot family. The researchers

used the Random Forest classifier for bot binary classification and a multi-class classifier to classify bot families. They evaluated their system on an ISCX Botnet dataset and MCFP datasets which contain 7M malicious flows from 12 botnet families. The results showed that the system can detect bot network traffic with 99.78% F-measure and 99.09% F-measure for classifying bot family. Also, the system can detect traffic belonging to hidden bot families with a 93.03% F-measure. However, the system requires a high memory capacity to perform the classification.

Hwang et al., 2020 [33] presented a combined model for ransomware detection. First, they used a Markov chain behavior-based model on Windows API call sequences with sequential characteristics to detect ransomware. Second, they applied the Random Forest machine learning model to enhance the FP and FN. The results showed that the proposed model detected ransomware with 97.3% accuracy at 4.8% FP and 1.5% FN. However, the model was trained with a limited dataset.

Amer et al., 2020 [34] proposed an approach for malware detection and prediction that requires a contextual understanding of the API call sequence based on word-embedding to distinguish between malware and benign apps. They used the Markov chain to classify apps by generating a transition matrix to predict API benign or malware apps. Applying the model on four datasets, i.e., Ki et al. (2015), Kim (2018), CSDMC, and Catak and Yazı (2019) showed that it has 0.990 detection accuracy, 0.010 FP, 0.997 prediction accuracy at 0.000 FP, and 0.007 FN.

Surendran et al., 2020 [35] proposed a model to characterize Android malware applications. They extracted system call features based on stationary first-order ergodic and proved the existence of malicious behavior. They selected malware families based on repacking and obfuscation techniques. The researchers used four datasets, i.e., AMD, Drebin, and Contagio for malware and AndroZo for benign apps. The model obtained 0.95 accuracy in all datasets, 0.90 precision for balanced data sets, and 0.72 for slightly unbalanced datasets.

D'Angelo et al., 2021 [36] presented an association rule-based approach for malware classification based on a recurring subsequence alignment-based algorithm. The approach exploits the probabilities of transitioning between two APIs based on the Markov chain and the timeline of extracted API by MSA. They evaluated the approach using 7.3 K malware and 1.2 K benign apps from publicly available sources. The results showed that the association rule-based approach can detect unknown malware with 99% accuracy and 96.10% F-measure outperforming other existing approaches.

Li et al., 2022 [37] proposed a new method for Windows malware detection based on a convolution neural network. They extracted malware API call sequences from VirusTotal and VirusShare (6686 malware) and a benign API from system programs (6938 benign). They used malware API to produce a directed cyclic graph and extract a characteristic map of the graph using the Markov chain and principal component analysis. They designed the classifier based on a convolutional network graph. The results show that the proposed method had 98.32% better accuracy than existing methods and a 0.0037 better FP rate. Nevertheless, there is a lack of information about environmental implementation features.

Based on the above, none of the previous works considered using PHMM for dynamic malware detection, although it has the potential to provide safe analysis during runtime due to its delete or non-emitting states [11].

4. Methodology

For safe malware detection in IoT Android system we have implemented a python-based framework and demonstrated its effectiveness in detecting hidden malware codes by dynamically analyzing their behaviors using a PHMM that can safely and acuretly detect malicious codes even if obfuscation or any other hiding techniques are used. The PHMM training hyperparameters included: the number of hidden states, initial state probability vector (ISPV), transition probability matrix (TPM), the number of distinct observation symbols per state and the probability distribution of observation symbols for each state,

which can be automatically set by a solver, e.g., Black-Box optimization solver. This section details the methodology we followed to build, train and test our proposed solution.

4.1. Tools Used

The main tools we used include VirtualBox, Kali Linux 2021.3, MobSF tool 3.5, MUSCLE 3.8.31, HMMER 3.3.2 and Python tools, as shown in Table 1.

Table 1. List of used tools.

Software	Description
VirtualBox (Oracle VM VirtualBox Manager 6.1.34)	A virtual environment and cross-platform virtualization software that allows users to run multiple operating systems on their computers simultaneously [31].
Kali Linux 2021.3	A Linux-based operating system that is designed for use in information security tasks, such as penetration testing, security research, computer forensics, and reverse engineering [25].
MobSF tool 3.5	An open-source security framework that can perform dynamic analysis to test mobile applications end to end. This tool supports Android Application Package (APK) and iOS package App Store (IPA) binaries, as well as zipped source code [26].
MUSCLE 3.8.31	A computer software program for aligning protein and nucleotide sequences [35] to produce .afa files.
HMMER 3.3.2	A tool used to search for sequence homologs databases and to make sequence alignments. This software implements probabilistic models called hidden Markov Models (HMMs) [9].
Python	A programming language used for the implementation.

4.2. Data Sets

We used the Drebin dataset that contains 5560 applications from 179 different malware families. Drebin dataset also, contains the malicious applications from the Malgenome dataset which was collected from 2012–2015 and contains 1260 malware applications from 49 families. The Drebin dataset has been selected for this research for several reasons. First, it contains large amount of malware families, including those from the Malgenome dataset. Second, the benchmarks, which include [16,35,38,40–44], used the Drebin dataset, which shows its usefulness and popularity in the field and mandates using it to have valid comparisons with the benchmarks.

To overcome the imbalance issue related to the Drebin dataset where most files have invalid APKs, we analyzed families that contain at least 20 samples and removed apps with invalid APKs. In total, we analyzed 4785 files out of 5560 for our research because some APK files did not work at run time (invalid APK). Table 2 lists the malware families that we worked on from the Drebin dataset, 24 families in total. For the benign dataset, we used 500 applications from the google play store [31].

Table 2. Considered malware families from the Drebin dataset.

#	Family	Samples	#	Family	Samples
1	Adrd	91	13	Glodream	69
2	BaseBridge	330	14	Hambo	28
3	Boxer	27	15	Iconosys	152
4	DroidDream	81	16	Imlog	43
5	DroidKungFu	667	17	Jifake	29
6	ExploitLinuxLotoor	70	18	Kwin	147
7	FakeDoc	132	19	MobileTx	69
8	FakeInstaller	925	20	Opfake	613
9	FakeRun	61	21	Plankton	625
10	Gappusin	58	22	SendPav	59
11	Geinimi	92	23	SMSreg	41
12	Ginmaster	339	24	YZHC	37

4.3. System Components

As shown in Figure 1, DIP consists of three main parts, namely, dynamic analysis, training, and testing. First, the malware datasets were dynamically analysed to extract call sequence features that are used by the applications to complete specified tasks. The suspicious calls were then selected and converted to Fasta format to be used during training and testing. Second, the MSA process was carried out during training to produce the .afa format for each malware family and then train PHMM. Finally, this trained PHMM model was used to classify malware as either malicious or benign and assign it to a family.

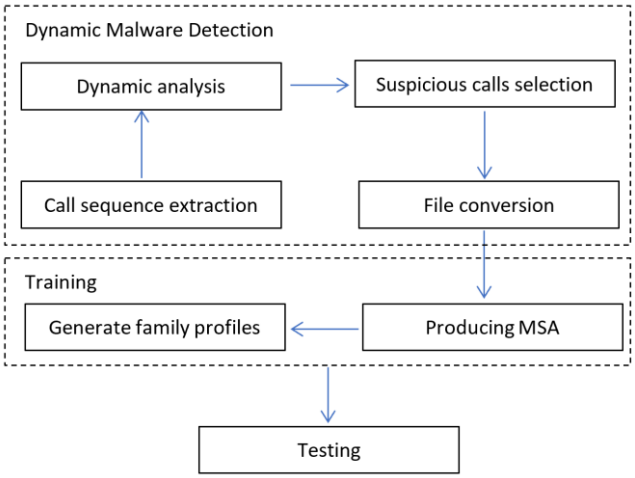

Figure 1. Main components of DIP which include dynamic malware detection, training and testing.

4.3.1. Dynamic Malware Analysis

We used theDrebin dataset to perform the dynamic analysis [45]. We ran each APK file individually for a period using the MobSF tool to monitor the call sequence, as shown in Figure 2. We extracted the API call sequence for 1000 malware and 500 benign apps that were invoked at run time, as shown in Figure 3. We followed the methodology used in Ref. [16] which calculates the frequency of API calls that were invoked at run time in malware and benign codes and defines the suspicious API calls based on frequency. If an API call is invoked many times in malware but not in benign, it is considered as being suspicious. For example, the StringBuffer class was invoked many times in malware. After obtaining a list of suspicious APIs, we mapped every group of API classes with a unique alphabet from [A–T] to facilitate the conversion to Fasta files in training and testing. For example, we placed the Crypto and HashMap classes in the same group and mapped them to the K alphabet. Following this, we converted them into Fasta format using the developed tool for biological sequences [46].

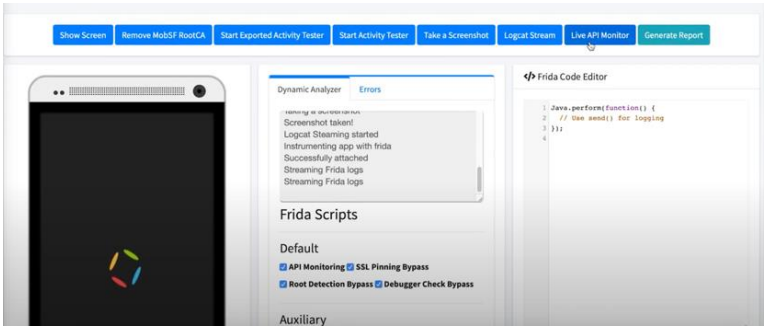

Figure 2. Dynamic analysis of malware and benign codes using MobSF Tool.

CLASS	METHOD
android.app.Activity	startActivity
android.app.Activity	startActivity
android.app.Activity	startActivity
android.app.Activity	startActivity
android.os.SystemProperties	get
android.os.SystemProperties	get

Figure 3. Example of API call sequences extracted using MobSF Tool.

4.3.2. Model Training

To train our model we have to train PHMM to do two tasks. The first task is producing MSA and the second is generating family profiles. For training PHMM to produce MSA, we divided the dataset into two sections (70% for training and 30% for testing) which means 3350 apps for training and 1435 for testing. As shown in Figure 4, first, we ran the APK files for 3350 apps and performed dynamic analysis to extract the API that was invoked at run time. For each malware family t, we extracted the API call sequence and matched it with a suspicious API call sequence list. After matching the API, we converted the file that contained a series of API calls sequence for malware to a Fasta file.

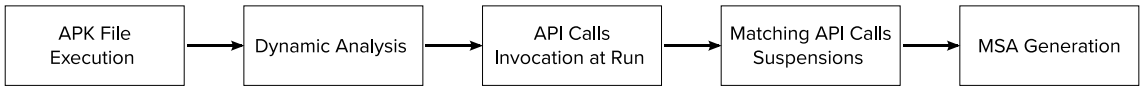

Figure 4. Process pipeline of training PHMM to produce MSA.

To produce a MSA for each malware family we used the MUSCLE tool which is an extension of the HMM that allows null transition to identify sequences that change due to random mutations, such as point insertions and deletions. HMM is a statistical tool that uses a probabilistic finite state machine with certain hidden states to capture the features of one or more sequences of observable symbols. When the state machine is trained, the graph and transition probabilities are calculated to generate the best training sequences possible. When a new sequence is tested, the HMM assigns a score based on how well it matches the known state machine. The observed symbols in our scenario are the API calls for each malware. This model learns from datasets that contain sequences of features we extracted by maximizing the probability of a sequence [46].

PHMM can detect malware faster than HMM because sequence alignment in PHMM is used differently. In PHMM, multiple gene sequences that are significantly related are aligned in DNA sequencing. The alignment can be used to determine whether the gene sequences diverged from a common ancestor. This multiple sequence alignment of a profile can now be used to assess whether or not an unknown sequence is related to it. A pairwise alignment of two sequences produces a pair of equal-length sequences that insert '-' or gaps to represent the difference between the two original sequences. The global alignment maximizes the number of matches while minimizing the number of insertions and deletions [11].

Figure 5 shows a sample of the ADRD family API call sequence in Fasta representation before the MSA and after the MSA of the same API call sequences.

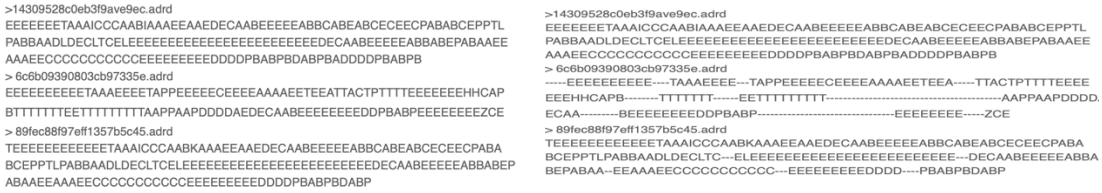

Figure 5. Sample of ADRD family Fasta file and a sample of MSA ADRD family.

For the second training task which is training PHMM to generate family profiles, we ran the MSA for every malware family and passed files to the HMMER tool to train the PHMM and generate the profilefor each family. The number of malware samples that can be used to create the HMM profile depends on the dataset. If the malware family contains 20 files, then there are 20 multiple sequences. Individual profile files were integrated to create a PHMM database with a training profile of malware apps representing all malware families. This training database is the one used for classification. After matching sequences in PHMM, a score is given for each family. If the sequences in the Fasta file match any of the PHMM database profiles, a hit will be generated with a positive score against the family that has a likeness with the tested sequence and a negative score with other families. Unknown sequences are mapped into homologous PHMMs. The malware family is checked against the highest generated score and mapped into the unknown app. The application is categorized as benign if the results are negative for all families.

4.3.3. Model Testing

In the testing phase, we used the trained PHMM to classify applications as benign or maliciousFasta based on the generated score. Following this, we calculated the performance measures, listed in Section 5, to assess the system performance.

5. Evaluation and Results

All the experiments were conducted using the following hardware specifications: (iMac) PC desktop, macOS operating system, 3.3 GHz processor speed, 1TB capacity and 8 GB RAM. Herein, we present the details of how we evaluated our system and we discuss the experimental results based on each performance measure. The proposed solution, DIP, was evaluated using 24 families in the Drebin dataset which represents 30% of the dataset with 1435 malware and 500 benign apps. Table 3 shows the number of samples from each malware family that we used for testing.

Table 3. Number of malware samples for testing.

#	Malware Family	Samples	#	Malware Family	Samples
1	Adrd	30	13	Glodream	19
2	BaseBridge	112	14	Hambo	8
3	Boxer	9	15	Iconosys	56
4	DroidDream	21	16	Imlog	14
5	DroidKungFu	205	17	Jifake	3
6	ExploitLinuxLotoor	23	18	Kwin	33
7	FakeDoc	48	19	MobileTx	18
8	FakeInstaller	276	20	Opfake	181
9	FakeRun	16	21	Plankton	192
10	Gappusin	18	22	SendPav	10
11	Geinimi	7	23	SMSreg	19
12	Ginmaster	112	24	YZHC	19

We chose malware applications randomly from the dataset and performed dynamic analysis to extract the API call sequence that was invoked at run time and then converted the API call sequence to Fasta files that contain the sequence of malware, as we did during

the implementation. Following this, we passed the file into PHMM to classify it. The tested malware gave a positive score for the family that has a likeness with the tested sequence and a negative score for other families. To classify malware families, we set the threshold to zero and we classified 1390 out of 1435 malware apps according to their families. For benign apps, we examined the 500 apps, and the system classified 475 apps as benign.

The metrics that we used to evaluate the effectiveness of PHMM malware detection are:

- Detection rate for each malware family according to Equation (1):

$$\text{Detection Rate} = \frac{\text{Malware Samples Belonging to a Family } X \text{ and Correctly Classified in That Family}}{\text{Total Malware samples Belonging to Family } X} * 100 \quad (1)$$

- FP rate which represents the ratio between the error number of unrelated retrieved cases and the total number of the ground truth of unrelated cases. FP rate is calculated according to Equation (2):

$$\text{FP Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (2)$$

  where TN is the number of unretrieved unrelated cases.
- FN rate which represents the ratio between the unretrieved correct values and the total number of the ground truth of the correct cases FN rate is calculated according to Equation (3):

$$\text{FN Rate} = \frac{\text{FN}}{\text{FN} + \text{TP}} \quad (3)$$

  where TP is the number of the correct cases retrieved.
- Precision which represents the ratio between the number of TP and the total number of retrieved cases. High precision means our system gives a more relevant result. Equation (4):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4)$$

- Recall is the ratio between the number of TP and the total number of correct cases. Recall is calculated according to Equation (5):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (5)$$

- Accuracy is the number of correct cases examined divided by the number of all cases examined. High accuracy means the system can detect malware effectively. Accuracy is calculated according to Equation (6):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{N}} \quad (6)$$

  where N is the total number of all values.
- F-measure, combines recall and precision. The best result of F-measures is the one closest to 1. Equation (7) shows the formula of F-measure:

$$\text{F-measure} = 2 * \frac{\text{precision} * \text{Recall}}{\text{precision} + \text{recall}} \quad (7)$$

We calculated the detection rate for each malware family according to Equation (1). Table 4 shows the detection rate for each malware family. As can be seen, 18 families have a high detection rate which is above 90% excluding Boxer, ExploitLinuxLotoor, Hambo and SMSreg families whose detection rates are between 84.21% and 88.89%. The Geinimi and Jifak family have detection rates of 57.14% and 66.67%, respectively. The YZHC family has the lowest detection rate of 31.58% due to the low number of tested samples. Having said this, some families with a low number of test samples still yielded a worthy detection rate,

such as Boxer and Hambo. Notably, DroidKungFu, FakeInstaller, Opfake and Plankton have the highest detection rates, approaching 99.28%, because they contain a high number of test samples. It can be concluded that determining the detection rate of a family depends on the number of samples used in training and testing, and on the complexity of the family.

Table 4. Detection rate of each malware family using DIP.

#	Malware Family	Detection Rate %	#	Malware Family	Detection Rate %
1	Adrd	96.67	13	Glodream	94.74
2	BaseBridge	94.64	14	Hambo	87.50
3	Boxer	88.89	15	Iconosys	96.43
4	DroidDream	95.24	16	Imlog	92.86
5	DroidKungFu	98.05	17	Jifake	66.67
6	ExploitLinuxLotoor	86.96	18	Kwin	93.94
7	FakeDoc	95.83	19	MobileTx	94.44
8	Fakelnstaller	99.28	20	Opfake	98.34
9	FakeRun	93.75	21	Plankton	98.96
10	Gappusin	94.44	22	SendPav	90.00
11	Geinimi	57.14	23	SMSreg	84.21
12	Ginmaster	95.54	24	YZHC	31.58

The results of accuracy, FP rate, FN rate, precision, recall and F1 score are presented in Table 5. DIP showed 96.3% accuracy with 5% FP and 3% FN which shows how successful our model is at predicting the correct category of the code. It has pinpointed malware code with a precision of up to 94.0%. While the 96.0% recall indicates that we have only missed 0.04 of malware code. The F-measure of 94.9% indicates that the model provides a balanced precision and recall score.

Table 5. Results of malware detection using DIP.

Measure	Value
FP Rate	5%
FN Rate	3%
Precision	94.0%
Recall	96.0%
Accuracy	96.3%
F-measure	94.9%

We compared the results of DIP with rival approaches that used the Drebin dataset for malware detection in Table 6. The missing figures in the comparison are denoted by ϵ. Based on Table 6, DIP technique scored the highest on accuracy, precision, and recall, with 96.3% for accuracy, 94% for precision, and 96% for recall. On the other hand, the benchmark algorithms' highest accuracy was 96% achieved by the dynamic classifier in [45]. However, that classifier is capable of binary classification only. Among the benchmarks, the static classifier in [16] got the highest precision and recall at 93% and 95%, respectively. As shown by these results, DIP performed better than all state-of-the-art algorithms when it was applied to detect malware on Android IoT platforms. DIP's dynamic nature adds to its advantages since it can detect more malicious features and recognize new malware families easily.

Table 6. Benchmarking DIP against related work.

Reference	Static/Dynamic	Accuracy	Precision	Recall
DIP	Dynamic	96.3%	94.0%	96.0%
[40]	Dynamic	91%	90.0%	ϵ
[16]	Static	94.5%	93.0%	95.0%
[35]	Dynamic	95%	92.0%	ϵ
[38]	Dynamic	96%	ϵ	ϵ
[41]	Dynamic	93.6%	91.0%	ϵ
[42]	Dynamic	80%	ϵ	ϵ
[43]	Static	93%	ϵ	ϵ
[44]	Dynamic	95.7%	ϵ	ϵ

6. Conclusions and Future Work

In this paper, we presented a model to detect malware in IoT Android systems. As a first step, the system dynamically analyzes the malware datasets to extract call sequence features that are used by the applications for completing specified tasks. After that, suspicious calls are selected. Then, the MSA process is done during training to produce a profile for each malware family. Finally, the trained PHMM model is applied to classify malware as malicious or benign and to which family it belongs.

We evaluated our proposed system using 4785 malicious apps that were extracted from the Drebin dataset and 500 benign apps extracted from Google Play. We took into consideration seven performance measures, including detection rate, precision, recall, accuracy, F-measure, FN and FP. We observed high detection rates (for most malware families), 96.3% accuracy, 94.0% precision, 96.0% recall and 94.9% F-measure with 5% FP and 3% FN. A benchmark test was then conducted against eight rival Android malware detection frameworks. Results showed that our proposed model performed superiorly when compared to other models.

On the other hand, our model has some limitations. First, it focused on Android IoT systems only. Second, it relied on the Derbin dataset which lacks recent updates. Third, the results are sometimes affected by the FP problem, which means the system might classify a specific system behavior as malicious when it is not.

Possible directions that can be taken in this field to address these limitations include testing the system on different operating systems and using more recent datasets. Another direction is to reduce the FP problem based on collective detection using Blockchain technology.

Author Contributions: Conceptualization, N.A. and M.A.; Formal analysis, N.A.; Investigation, N.A. and H.K.; Methodology, N.A. and H.K.; Project administration, H.K.; Resources, H.K.; Software, N.A.; Supervision, H.K.; Visualization, M.A.; Writing—original draft, N.A., H.K. and M.A.; Writing—review & editing, H.K. and M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research project was supported by the Researchers Supporting Project number (RSP2023R204), King Saud University, Riyadh, Saudi Arabia.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study. This data can be found here: [www.sec.cs.tu-bs.de/~danarp/drebin/] (accessed on 1 December 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ahmed, I.; Ahmad, M.; Chehri, A.; Hassan, M.M.; Jeon, G.J.R.S. IoT Enabled Deep Learning Based Framework for Multiple Object Detection in Remote Sensing Images. *Remote Sens.* **2022**, *14*, 4107. [CrossRef]
2. Kimani, K.; Oduol, V.; Langat, K. Cyber security challenges for IoT-based smart grid networks. *Int. J. Crit. Infrastruct. Prot.* **2019**, *25*, 36–49. [CrossRef]
3. Ehie, I.C.; Chilton, M.A. Understanding the influence of IT/OT Convergence on the adoption of Internet of Things (IoT) in manufacturing organizations: An empirical investigation. *Comput. Ind.* **2019**, *115*, 103166. [CrossRef]
4. Son, Y.H.; Kim, G.-Y.; Kim, H.C.; Jun, C.; Noh, S.D. Past, present, and future research of digital twin for smart manufacturing. *J. Comput. Des. Eng.* **2022**, *9*, 1–23. [CrossRef]
5. Huong, T.T.; Bac, T.P.; Long, D.M.; Luong, T.D.; Dan, N.M.; Thang, B.D.; Tran, K.P. Detecting cyberattacks using anomaly detection in industrial control systems: A Federated Learning approach. *Comput. Ind.* **2021**, *132*, 103509. [CrossRef]
6. Kaspersky. Available online: https://www.kaspersky.com/resource-center/threats/android-vs-iphone-mobile-security (accessed on 18 December 2022).
7. Mariconti, E.; Onwuzurike, L.; Andriotis, P.; De Cristofaro, E.; Ross, G.; Stringhini, G.J. Mamadroid: Detecting android malware by building markov chains of behavioral models. *arXiv* **2016**, arXiv:1612.04433.
8. Wan, T.-L.; Ban, T.; Lee, Y.-T.; Cheng, S.-M.; Isawa, R.; Takahashi, T.; Inoue, D. IoT-malware detection based on byte sequences of executable files. In Proceedings of the 2020 15th Asia Joint Conference on Information Security (AsiaJCIS), Taipei, Taiwan, 20–21 August 2020; pp. 143–150.
9. Ren, Z.; Wu, H.; Ning, Q.; Hussain, I.; Chen, B. End-to-end malware detection for android IoT devices using deep learning. *Ad Hoc Netw.* **2020**, *101*, 102098. [CrossRef]
10. Sikder, A.K.; Aksu, H.; Uluagac, A.S. A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Trans. Mob. Comput.* **2019**, *19*, 245–261. [CrossRef]
11. Alipour, A.A.; Ansari, E. An advanced profile hidden Markov model for malware detection. *Intell. Data Anal.* **2020**, *24*, 759–778. [CrossRef]
12. Al-Bakri, P.A.M.; Hussein, H.L. Static analysis based behavioral api for malware detection using markov chain. *Int. Inst. Sci. Technol. Educ. (IISTE)* **2014**, *5*, 55–63. [CrossRef]
13. Ficco, M. Detecting IoT malware by Markov chain behavioral models. In Proceedings of the 2019 IEEE International Conference on Cloud Engineering (IC2E), Prague, Czech Republic, 24–27 June 2019; pp. 229–234.
14. Annachhatre, C.; Austin, T.H.; Stamp, M.; Techniques, H. Hidden Markov models for malware classification. *J. Comput. Virol. Hacking Tech.* **2015**, *11*, 59–73. [CrossRef]
15. Damodaran, A.; Troia, F.D.; Visaggio, C.A.; Austin, T.H.; Stamp, M.J.; Techniques, H. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 1–12. [CrossRef]
16. Sasidharan, S.K.; Thomas, C.J.P.; Computing, M. ProDroid—An Android malware detection framework based on profile hidden Markov model. *Pervasive Mob. Comput.* **2021**, *72*, 101336. [CrossRef]
17. Cho, I.K.; Im, E.G. Extracting representative API patterns of malware families using multiple sequence alignments. In Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems, Prague, Czech Republic, 9–12 October 2015; pp. 308–313.
18. Edgar, R. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* **2004**, *32*, 1792–1797. [CrossRef] [PubMed]
19. Pedersen, J.; Bastola, D.; Dick, K.; Gandhi, R.; Mahoney, W. Blast your way through malware malware analysis assisted by bioinformatics tools. In Proceedings of the International Conference on Security and Management (SAM), Las Vegas, NV, USA, 16–19 July 2012; p. 1.
20. Jeon, J.; Park, J.H.; Jeong, Y.-S. Dynamic analysis for IoT malware detection with convolution neural network model. *IEEE Access* **2020**, *8*, 96899–96911. [CrossRef]
21. Sangal, A.; Verma, H.K. A static feature selection-based android malware detection using machine learning techniques. In Proceedings of the 2020 International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 20–22 October 2022; pp. 48–51.
22. Fatima, A.; Maurya, R.; Dutta, M.K.; Burget, R.; Masek, J. Android malware detection using genetic algorithm based optimized feature selection and machine learning. In Proceedings of the 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), Budapest, Hungary, 1–3 July 2019; pp. 220–223.
23. Kumar, A.; Lim, T.J. EDIMA: Early detection of IoT malware network activity using machine learning techniques. In Proceedings of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019; pp. 289–294.
24. Darabian, H.; Dehghantanha, A.; Hashemi, S.; Homayoun, S.; Choo, K.; Practice, C. An opcode-based technique for polymorphic Internet of Things malware detection. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5173. [CrossRef]
25. Takase, H.; Kobayashi, R.; Kato, M.; Ohmura, R. A prototype implementation and evaluation of the malware detection mechanism for IoT devices using the processor information. *Int. J. Inf. Secur.* **2020**, *19*, 71–81. [CrossRef]
26. Xu, G.; Wang, W.; Jiao, L.; Li, X.; Liang, K.; Zheng, X.; Lian, W.; Xian, H.; Gao, H. SoProtector: Safeguard privacy for native SO files in evolving mobile IoT applications. *IEEE Internet Things J.* **2019**, *7*, 2539–2552. [CrossRef]
27. Taheri, R.; Shojafar, M.; Alazab, M.; Tafazolli, R. FED-IIoT: A robust federated malware detection architecture in industrial IoT. *IEEE Trans. Ind. Inform.* **2020**, *17*, 8442–8452. [CrossRef]

28. Khoda, M.E.; Imam, T.; Kamruzzaman, J.; Gondal, I.; Rahman, A. Robust malware defense in industrial IoT applications using machine learning with selective adversarial samples. *IEEE Trans. Ind. Appl.* **2019**, *56*, 4415–4424. [CrossRef]
29. Onwuzurike, L.; Mariconti, E.; Andriotis, P.; Cristofaro, E.D.; Ross, G.; Stringhini, G. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Trans. Priv. Secur. (TOPS)* **2019**, *22*, 1–34. [CrossRef]
30. Anandhi, V.; Vinod, P.; Menon, V.G. Malware visualization and detection using DenseNets. *Pers. Ubiquitous Comput.* **2021**, 1–17. [CrossRef]
31. Ahsan-Ul-Haque, A.; Hossain, M.S.; Atiquzzaman, M. Sequencing system calls for effective malware detection in android. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–7.
32. Alahmadi, B.A.; Mariconti, E.; Spolaor, R.; Stringhini, G.; Martinovic, I. BOTection: Bot detection by building Markov Chain models of bots network behavior. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, 5–9 October 2020; pp. 652–664.
33. Hwang, J.; Kim, J.; Lee, S.; Kim, K. Two-stage ransomware detection using dynamic analysis and machine learning techniques. *Wirel. Pers. Commun.* **2020**, *112*, 2597–2609. [CrossRef]
34. Amer, E.; Zelinka, I.; Security. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput. Secur.* **2020**, *92*, 101760. [CrossRef]
35. Surendran, R.; Thomas, T.; Emmanuel, S. On existence of common malicious system call codes in Android malware families. *IEEE Trans. Reliab.* **2020**, *70*, 248–260. [CrossRef]
36. D'Angelo, G.; Ficco, M.; Palmieri, F. Association rule-based malware classification using common subsequences of API calls. *Appl. Soft Comput.* **2021**, *105*, 107234. [CrossRef]
37. Li, S.; Zhou, Q.; Zhou, R.; Lv, Q. Intelligent malware detection based on graph convolutional network. *J. Supercomput.* **2022**, *78*, 4182–4198. [CrossRef]
38. Salehi, M.; Amini, M. Android malware detection using Markov Chain model of application behaviors in requesting system services. *arXiv* **2017**, arXiv:1711.05731.
39. Zelinka, I.; Amer, E. An ensemble-based malware detection model using minimum feature set. *Mendel* **2019**, *25*, 1–10. [CrossRef]
40. Bernardi, M.L.; Cimitile, M.; Distante, D.; Martinelli, F.; Mercaldo, F. Dynamic malware detection and phylogeny analysis using process mining. *Int. J. Inf. Secur.* **2019**, *18*, 257–284. [CrossRef]
41. Xiao, X.; Zhang, S.; Mercaldo, F.; Hu, G.; Sangaiah, A. Android malware detection based on system call sequences and LSTM. *Multimed. Tools Appl.* **2019**, *78*, 3979–3999. [CrossRef]
42. Rashidi, B.; Fung, C. Xdroid: An android permission control using hidden markov chain and online learning. In Proceedings of the 2016 IEEE Conference on Communications and Network Security (CNS), Philadelphia, PA, USA, 17–19 October 2016; pp. 46–54.
43. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. *Ndss* **2014**, *14*, 23–26.
44. Cen, L.; Gates, C.S.; Si, L.; Li, N.; Computing, S. A probabilistic discriminative model for android malware detection with decompiled source code. *IEEE Trans. Dependable Secur. Comput.* **2014**, *12*, 400–412. [CrossRef]
45. Igarashi, Y. Forest of Pressure: Ogawa Shinsuke and Postwar Japanese Documentary. *JSTOR* **2010**, *36*, 165–169.
46. Ravi, S.; Balakrishnan, N.; Venkatesh, B. Behavior-based malware analysis using profile hidden markov models. In Proceedings of the 2013 International Conference on Security and Cryptography (SECRYPT), Reykjavik, Iceland, 29–31 July 2013; pp. 1–12.