IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 28, Issue 3, Ser. 1 (May. – June. 2026), PP 70-75
www.iosrjournals.org

Mobile Botnet Detection: An SVM-Based Machine Learning Approach With APK Feature Extraction

Dr. Sonali Patil
Priyanshu Bakare
Siddharth Pitre
Aditya Kulthe
Aniket Gandhe
Siddhanth Pardhe
Department Of Information Technology
Hope Foundation's International Institute Of Information Technology (I2IT)
Pune – 411057, India

Abstract
The rapid proliferation of Android-based smartphones has created a fertile environment for increasingly sophisticated mobile botnets that pose critical cybersecurity threats. Traditional signature-based detection mechanisms prove inadequate against evolving polymorphic threats and zero-day exploits. This research presents an end-to-end machine learning framework utilizing Support Vector Machine (SVM) algorithms for automated mobile botnet detection through direct APK file analysis—eliminating manual feature entry by automatically extracting 25 discriminative features encompassing both static application characteristics and network behavior indicators. The proposed system was evaluated on benchmark datasets including ISCX Android Botnet Dataset, CTU-13, and CICIDS2017. Experimental results demonstrate 96.8% accuracy, 95.4% precision, 96.1% recall, and an F1-score of 95.7%, with a low false positive rate of 3.2%. A Django-based web application provides real-time detection through APK upload, automated feature extraction, and instant botnet classification with visual alerting. The system successfully bridges the gap between academic research and practical deployment, offering security administrators an efficient, automated tool for mobile threat defense.

Keywords: *Mobile Botnet Detection, Android APK Analysis, Support Vector Machine, Static Feature Extraction, Network Traffic Analysis, Machine Learning, Django Web Application, Cybersecurity*

Date of Submission: 16-05-2026 Date of Acceptance: 26-05-2026

I. Introduction

The contemporary digital ecosystem has witnessed an unprecedented surge in Android smartphone adoption. With Android holding over 70% of the global mobile operating system market share, malicious actors have intensified efforts to exploit its open-source architecture and permissive application distribution model. Among the most dangerous threats are *mobile botnets*—coordinated networks of compromised devices remotely controlled by botmasters through Command and Control (C&C) infrastructure to execute large-scale attacks including Distributed Denial of Service (DDoS) attacks, credential theft, spam distribution, phishing, click fraud, and ransomware propagation [1].

Mobile devices are particularly attractive botnet targets for several reasons. They maintain persistent internet connectivity, users apply weaker security practices than on desktop systems, and the Android platform permits third-party application installation with minimal gatekeeping. Numerous families of Android botnet malware—including Geinimi, SpamSoldier, and Rotexy—have been discovered capable of silently enrolling compromised devices into botnets while mimicking legitimate applications [2].

Traditional intrusion detection systems relying on signature-based or rule-based approaches have proven fundamentally inadequate. Such methods fail against novel, zero-day attacks, struggle against polymorphic malware that continuously mutates its code structure, and cannot adapt to the rapidly evolving tactics of modern botnet operators [3]. These limitations necessitate a paradigm shift toward adaptive, data-driven methodologies.

Machine learning offers transformative detection capabilities by learning complex behavioral patterns directly from training data. Among various ML algorithms, Support Vector Machine (SVM) stands out for its effectiveness in high-dimensional feature spaces, its ability to construct optimized decision boundaries maximizing class separation, and its computational efficiency relative to deep learning alternatives [4]. SVM

maintains strong generalization capabilities without requiring the architectural fine-tuning that characterizes neural network approaches.

A significant limitation of prior work—including our first-semester implementation—involved the requirement for *manual feature entry*, where users had to input binary values (0 or 1) for each of the 25 detection parameters. This approach is error-prone, time-consuming, and impractical for real-world deployment. The present research addresses this critical gap by implementing *automated APK feature extraction*, where the system directly analyzes uploaded APK files to automatically populate all detection features without human intervention.

This paper makes the following primary contributions: (1) implementation of automated APK feature extraction from uploaded application packages; (2) development of an SVM-based classification model achieving 96.8% detection accuracy; (3) deployment of a Django-based web application providing real-time monitoring with zero manual feature entry; and (4) comprehensive evaluation demonstrating superiority over baseline classifiers.

II. Related Work And Literature Survey

Research into mobile botnet detection has evolved substantially over the past decade, progressing from simple permission analysis to sophisticated behavioral and traffic-based approaches.

Tansettanakorn and Thongprasit [5] developed ABIS (Android Botnet Identification System), a prototype employing static permission-based analysis to screen applications before installation. While demonstrating early promise, ABIS encountered limitations with dynamically loaded code and obfuscated manifests that sophisticated malware increasingly employs to evade static scrutiny.

Abdullah et al. [6] proposed Android botnet classification using feature selection algorithms, demonstrating that careful feature engineering substantially improves classification performance. Their work utilized permission combinations and API call patterns as primary discriminative features, achieving promising accuracy on controlled datasets, though scalability across diverse malware families remained challenging.

Hijawi, Alqatawna, and Faris [7] introduced a detection framework incorporating discriminative permission-level features derived from Android permission protection levels. Comparing classifiers including Decision Trees, Random Forests, and SVMs, their findings confirmed SVM's superior generalization for high-dimensional feature spaces with balanced class distributions.

Yerima and Alzaylaee [8] developed a deep learning approach based on Convolutional Neural Networks (CNN) trained on 342 static features. While CNNs achieved impressive detection rates through hierarchical pattern learning, their substantial hardware requirements and training complexity present barriers to real-time deployment on resource-constrained devices. This finding directly motivates our choice of SVM as a computationally efficient alternative.

Eslahi et al. [9] proposed a cooperative network behavior analysis model targeting mobile HTTP botnets through communication pattern monitoring. This network-centric approach proved effective against actively communicating botnets but exhibited weakness against dormant threats that periodically activate or employ stealthy communication channels.

Oulehla [10] explored neural network architectures for mobile botnet detection, leveraging non-linear relationships between behavioral features. Despite achieving accurate classification, the model's interpretability limitations and high resource consumption restricted practical adoption. Anwar [11] advocated for static detection approaches utilizing Android permission combinations and intent filter declarations, though this methodology proved vulnerable to obfuscation attacks.

From this comprehensive review, several critical observations emerge: deep learning achieves high accuracy at prohibitive computational cost; static analysis alone cannot detect all botnet variants; network traffic analysis misses dormant botnets; and manual feature entry remains a persistent practical limitation. Our work directly addresses these gaps.

III. Proposed Framework And Methodology

The proposed system constitutes an anomaly-based, host-based detection framework designed to identify mobile botnet activity through automated analysis of Android Application Package (APK) files. The architecture follows a structured multi-stage pipeline ensuring robust classification with zero manual intervention.

A. System Architecture

The detection framework operates across five integrated stages: (1) APK file upload through the web interface; (2) automated feature extraction from the APK binary; (3) data preprocessing and normalization; (4) SVM-based classification; and (5) result visualization and alert generation. The backend Python environment handles all computational tasks including APK parsing, feature extraction, model inference, and database logging using SQLite.

B. Dataset Description

The training and evaluation pipeline utilized three complementary datasets. The *ISCX Android Botnet Dataset* from the University of New Brunswick provides static features extracted from 1,929 Android applications comprising confirmed botnet samples from 7 distinct botnet families and benign applications [3]. The *CTU-13 Dataset* contains labeled network traffic captures from real botnet infections including Neris, Rbot, and Virut families. The *CICIDS2017 Dataset* supplies diverse labeled network flows covering multiple attack vectors alongside normal traffic patterns. These datasets provide dual perspectives: network-level behavioral signatures and application-level structural characteristics.

C. Automated APK Feature Extraction

The primary innovation distinguishing this work from prior semester implementation is automated feature extraction directly from APK binary files, eliminating the manual parameter entry requirement. APK files are ZIP archives containing compiled bytecode, resource files, and the AndroidManifest.xml descriptor. The extraction pipeline operates as follows:

Step 1 — APK Unpacking: The uploaded APK is programmatically decompressed using Python's zipfile library, extracting AndroidManifest.xml and classes.dex bytecode.

Step 2 — Manifest Parsing: The manifest is parsed using the androguard library to extract declared permissions, intent filters, and component declarations (activities, services, broadcast receivers).

Step 3 — Bytecode Analysis: Dalvik bytecode in classes.dex is disassembled to identify API call patterns, including sensitive calls to Telephony, SMS, and Network Manager APIs.

Step 4 — Feature Vector Assembly: Binary indicators (0/1) are assembled for each of the 25 detection features based on presence or absence of the corresponding capability or behavior.

Step 5 — Model Inference: The assembled feature vector is normalized and passed to the trained SVM classifier, returning a classification decision with confidence probability.

D. Feature Engineering

Feature selection is the most critical phase in the machine learning pipeline. Our framework extracts 25 carefully selected binary features across two complementary domains based on extensive literature review and domain expertise. Table I summarizes the key extracted features.

Table I. Extracted Features For Botnet Detection

No.	Static Application Features	Network / Behavioral Features
1	SEND_SMS Permission	Telephony.getDeviceId() call
2	READ_CONTACTS Permission	Telephony.getSubscriberId() call
3	RECORD_AUDIO Permission	AbortBroadcast() call
4	READ_CALL_LOG Permission	sendTextMessage() call
5	INTERNET Permission	deletePackages() call
6	RECEIVE_BOOT_COMPLETED	getLine1Number() call
7	Camera Access Permission	SMS Received listener
8	ACCESS_FINE_LOCATION	InetAddress resolution call
9	READ_PHONE_STATE	Read SMS capability
10	WRITE_EXTERNAL_STORAGE	Boot-triggered execution
11–15	IO.File Delete, Chown calls, Dynamic loading, Crypto API, Device admin requests	

E. Support Vector Machine Classification

Support Vector Machine serves as the core classification algorithm due to its demonstrated effectiveness in high-dimensional binary feature spaces. Given training data points (x_i, y_i) where x_i represents the 25-dimensional feature vector and $y_i \in \{-1, +1\}$ denotes class labels, SVM constructs a decision boundary $f(x) = w^T x + b$, where w is the weight vector orthogonal to the hyperplane and b is the bias term. The optimization objective minimizes $\frac{1}{2}\|w\|^2$ subject to $y_i(w^T x_i + b) \geq 1$ for all i, maximizing the margin between class boundaries.

We evaluated both linear and Radial Basis Function (RBF) kernels, defined as $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$. The RBF kernel proved superior for the combined static-behavioral feature set. Hyperparameters were

optimized through grid search with 5-fold cross-validation: regularization parameter $C \in \{0.1, 1, 10, 100\}$ and kernel coefficient $\gamma \in \{0.001, 0.01, 0.1, 1\}$. Optimal configuration: $C = 10$, $\gamma = 0.01$ with RBF kernel.

F. Data Preprocessing

The preprocessing pipeline implements four critical stages: (1) *Data Cleaning*—removal of incomplete records and duplicate entries; (2) *Min-Max Normalization*—scaling all features to [0,1] range; (3) *Class Balancing*—SMOTE oversampling of minority class to address dataset imbalance; and (4) *Train-Test Split*—80/20 stratified partitioning maintaining class proportions across both subsets.

G. System Implementation

The trained model was deployed within a Django-based web application providing: APK upload with drag-and-drop interface; automated feature extraction without user input; real-time SVM inference with confidence probability display; visual dashboard with detection statistics and historical analysis; and immediate alert generation for detected botnet activity including DDoS attack warnings.

Technology stack: Python 3.10, Django 4.2, scikit-learn 1.2, androguard 3.3, Pandas, NumPy, SQLite, and Bootstrap 5 for the responsive web interface. The system operates on standard hardware (Intel i5, 8 GB RAM) without requiring specialized GPU infrastructure.

IV. Experimental Results And Evaluation

A. Evaluation Metrics

Performance was assessed using five standard classification metrics. *Accuracy* = (TP + TN) / (TP + TN + FP + FN). *Precision* = TP / (TP + FP). *Recall* = TP / (TP + FN). *F1-Score* = 2 × (Precision × Recall) / (Precision + Recall). *False Positive Rate* = FP / (FP + TN). These metrics together provide a comprehensive and balanced characterization of classifier performance in the security domain where both false negatives (missed botnets) and false positives (nuisance alerts) carry operational costs.

B. Classification Performance

The trained SVM classifier achieved exceptional performance across all evaluation metrics, as summarized in Table II.

Table II. SVM Classification Performance Results

Metric	Result
Accuracy	96.8%
Precision	95.4%
Recall (Sensitivity)	96.1%
F1-Score	95.7%
False Positive Rate	3.2%
AUC-ROC Score	0.978

These results indicate the model successfully distinguishes between benign and malicious samples with high reliability. The near-balance between precision and recall, reflected in the strong F1-score of 95.7%, demonstrates that the classifier neither over-detects nor under-detects threats—a critical property for practical security deployment.

C. Comparative Analysis

To validate SVM superiority, we compared performance against four baseline classifiers on identical feature sets and evaluation protocols, as shown in Table III.

Table III. Comparative Classifier Performance

Algorithm	Accuracy	Precision	Recall	F1-Score
SVM (Proposed)	96.8%	95.4%	96.1%	95.7%
Random Forest	94.7%	93.2%	93.8%	93.5%
Decision Tree	91.3%	90.1%	91.5%	90.8%
Naïve Bayes	88.6%	87.3%	88.9%	88.1%

K-NN (k=5)	89.4%	88.0%	89.7%	88.8%

SVM achieves the highest accuracy (96.8%), outperforming Random Forest by 2.1 percentage points and Decision Tree by 5.5 percentage points. The balanced precision-recall trade-off in the F1-score of 95.7% confirms SVM as the optimal algorithm for this detection task.

D. Confusion Matrix Analysis

Analysis of the confusion matrix on the 20% test set (n = 386 samples) revealed: True Positives = 187, True Negatives = 186, False Positives = 7, False Negatives = 6. The SVM correctly identified 187 of 193 actual botnet samples (96.9% sensitivity) while incorrectly flagging only 7 of 200 benign samples (3.5% FPR), demonstrating excellent practical utility.

E. APK Extraction Validation

To validate the automated APK feature extraction pipeline, we manually verified extracted feature vectors against ground-truth annotations for 50 randomly selected samples (25 botnet, 25 benign). Feature extraction accuracy achieved 99.2% (496/500 features correctly extracted), with the 4 extraction errors occurring in heavily obfuscated samples using non-standard compression methods. All 4 errors produced conservative false-negative classifications consistent with safe failure behavior.

V. Technical Specifications

Hardware requirements: Intel i5 or above processor, 8 GB RAM minimum, 40 GB available storage for dataset and model files. Software stack: Python 3.10, Django 4.2, scikit-learn 1.2, androguard 3.3, SQLite (DB Browser for management), Anaconda Navigator for environment management, VS Code IDE, and Bootstrap 5 for responsive web design.

The system implements four core functional modules: (1) *Feature Extraction Module*—automatic parsing of APK binaries using androguard; (2) *Preprocessing Module*—normalization and encoding of extracted feature vectors; (3) *Detection Module*—trained SVM model inference with confidence scoring; (4) *Interface Module*—Django web-based result presentation with alert generation and historical logging.

Non-functional requirements: Detection inference completes in under 2 seconds per APK. The web interface supports concurrent multi-user access. Model retraining on updated datasets completes within 15 minutes. The system achieves 99.5% uptime under standard operating conditions. All detection events are logged with timestamps for audit trail compliance.

VI. Conclusion And Future Work

This paper presented an end-to-end automated mobile botnet detection system utilizing Support Vector Machine classification with direct APK feature extraction—a significant advancement over prior manual feature entry approaches. The proposed framework achieved 96.8% accuracy, 95.4% precision, 96.1% recall, and 95.7% F1-score on benchmark datasets, demonstrating both theoretical rigor and practical deployment feasibility.

The key contribution of automated APK analysis eliminates the critical usability bottleneck of manual parameter input, making the system genuinely deployable by security administrators without specialized machine learning expertise. SVM proved an optimal algorithm choice: achieving accuracy competitive with deep learning approaches while maintaining computational efficiency suitable for resource-constrained environments and real-time detection requirements.

Future work will pursue several promising directions: (1) expanding the feature set to 50+ features incorporating dynamic behavioral analysis through sandboxed APK execution; (2) integrating hybrid deep learning models combining SVM efficiency with CNN feature learning for obfuscated malware families; (3) extending detection to iOS applications; (4) implementing federated learning for privacy-preserving model updates from distributed deployments; and (5) integration with enterprise mobile device management (MDM) platforms for organizational-scale deployment.

References

[1]. M. Eslahi, R. Salleh, And N. B. Anuar, "Bots And Botnets: An Overview Of Characteristics, Detection And Challenges," In Proc. Ieee Int. Conf. Control System, Computing And Engineering (Iccsce), 2012, Pp. 349–354.
[2]. Z. Yajin And J. Xuxian, "Dissecting Android Malware: Characterization And Evolution," In Proc. Ieee Symp. Security And Privacy (Sp), 2012, Pp. 95–109.
[3]. Iscx Android Botnet Dataset. University Of New Brunswick. [Online]. Available: Https://Www.Unb.Ca/Cic/Datasets/Android-Botnet.Html. [Accessed: Mar. 2024].
[4]. S. Y. Yerima And S. Khan, "Longitudinal Performance Analysis Of Machine Learning Based Android Malware Detectors," In Proc. Int. Conf. Cyber Security And Protection Of Digital Services (Cyber Security), Ieee, 2019.
[5]. C. Tansettanakorn And S. Thongprasit, "Abis: A Prototype Of Android Botnet Identification System," In Proc. Int. Conf. Information Technology And Electrical Engineering (Icitee), 2020.

[6]. Z. Abdullah, "Android Botnet Classification Using Feature Selection And Classification Algorithms," J. Theoretical And Applied Information Technology, Vol. 99, No. 5, 2021.
[7]. W. Hijawi, J. Alqatawna, And H. Faris, "Toward A Detection Framework For Android Botnet," In Proc. 8th Int. Conf. Information And Communication Systems (Icics), 2020.
[8]. S. Y. Yerima And M. K. Alzaylaee, "Mobile Botnet Detection: A Deep Learning Approach Using Convolutional Neural Networks," In Proc. Int. Conf. Cyber Situational Awareness, Data Analytics And Assessment (Cybersa), Ieee, 2020.
[9]. M. Eslahi, M. Yousefi, And M. V. Naseri, "Cooperative Network Behaviour Analysis Model For Mobile Botnet Detection," In Proc. Ieee Symp. Computer Applications And Industrial Electronics (Iscaie), 2020.
[10]. M. Oulehla, "Detection Of Mobile Botnets Using Neural Networks," In Proc. 26th Int. Conf. Radioelektronika, Ieee, 2019.
[11]. S. Anwar, "A Static Approach Towards Mobile Botnet Detection," J. Computer Networks And Communications, 2021.
[12]. Z. Abdullah, M. M. Saudi, And N. B. Anuar, "Mobile Botnet Detection: Proof Of Concept," In Proc. Int. Conf. Advanced Computer Science Applications And Technologies, 2013.
[13]. G. Gu, R. Perdisci, J. Zhang, And W. Lee, "Botminer: Clustering Analysis Of Network Traffic For Protocol- And Structure-Independent Botnet Detection," In Proc. 17th Usenix Security Symp., San Jose, Ca, 2008, Pp. 139–154.
[14]. A. F. A. Kadir, N. Stakhanova, And A. A. Ghorbani, "Android Botnets: What Urls Are Telling Us," In Proc. Int. Conf. Network And System Security, Springer, 2015, Pp. 78–91.
[15]. M. Eslahi, R. Salleh, And N. B. Anuar, "Mobots: A New Generation Of Botnets On Mobile Devices And Networks," In Proc. Ieee Symp. Computer Applications And Industrial Electronics (Iscaie), 2012, Pp. 262–266.