Leveraging Oversampling Techniques in Machine Learning Models for Multi-class Malware Detection in Smart Home Applications

Abdullahi Chowdhury
School of Computer Science
The University of Adelaide
Adelaide, Australia
abdul.chowdhury@adelaide.edu.au

Mohammad Manzurul Islam
dept. of Computer Science and Engineering
East West University Bangladesh
Dhaka, Bangladesh
mohammad.islam@ewubd.edu

Shahriar Kaisar
dept. of IS and Business Analytics
RMIT University
Melbourne, Australia
shahriar.kaisar@rmit.edu.au

Mahbub E Khoda
Federation University Australia
Churchill, Australia
m.khoda,n.naha@federation.edu.au

Ranesh Naha
Federation University Australia
Churchill, Australia
n.naha@federation.edu.au

Mohammad Ali Khoshkholghi, Mahdi Aiash
Dept. of Computer Science
Middlesex University
London, UK
a.khoshkholghi,m.aiash@mdx.ac.uk

***Abstract*—Smarthome applications are becoming increasingly popular due to their ability to provide safety, comfort, and remote assistance. These applications are usually controlled using a smart home controller, which is often the target of malware attacks. A successful attack may result in financial loss, disclosure of personal and/or sensitive information, or even loss of human lives. Although machine learning models have been used in existing research for detecting multi-class malware attacks in smart home systems, they did not explicitly address the class imbalance problem in such cases. In addition, the use of ensemble learner is expected to provide improved performance. To address this, we investigated different oversampling techniques to increase the number of samples in the minority classes and incorporated ensemble learners to see their impact on the prediction performance. Experimental evaluation shows significant improvements (4-5%) in terms of accuracy, precision, recall, and F-1 score.**

***Index Terms*—Oversampling Techniques, Ensemble Models, Multi-class Malware Detection**

I. Introduction

We live in the era of internet-connected smart devices known as the Internet of Things (IoT). People are connecting their essential everyday devices (e.g., home appliances, wearable sensors, security cameras, intelligent cars) to the Internet for automating intended tasks and making life easy [1]. The scope of IoT device adoption also extends to industrial automation, smart cities, precision agriculture, medical services, etc. A report by Business Insider Intelligence [2] predicts that more than 41 billion IoT devices will be actively connected to the Internet by 2027, resulting in approximately five devices used by each person living on earth. A majority of them are smart home devices that collect, process, and derive insights from sensed data using which smart home sensors decide for the intended action. However, these heterogeneous sensors are generally low-powered with limited processing capabilities and storage, hindering the adoption of a strong security algorithm. Also, to capture the consumer market share, IoT device manufacturers rapidly developed affordable smart home devices without following a standardised security mechanism [3]. Furthermore, for ease of usability by a regular consumer, many connected smart home devices are operated with default authentication, which results in infamous Brickerbot [4], and Mirai [5] attacks. Therefore, an acceptable level of security for IoT smart home devices is far from reality.

Among different sensors/devices deployed in smart home applications, android operating system-based devices (e.g., google home, TV, mobile phones) play an important role by acting as a controlling hub, decision processing system, or an end device [6], [7]. However, recent investigations [8], [9] show the vulnerabilities of these android-based devices that can jeopardise the integrity of a smart home environment. An attacker can exploit weaknesses in android devices through malware and gain access to the rest of the devices in a smart home. Another concern is the possibility of hackers taking over smart home devices and using them for malicious purposes, such as launching Distributed Denial of Service (DDoS) attacks or stealing sensitive information. Additionally, the large number of devices in a typical smart home creates a complex attack surface that can be difficult to secure. Therefore, it is essential to detect such malware to safeguard a smart home application.

Smart home devices are also often connected to third-party services and platforms, which can introduce additional security risks. For example, if a third-party service is compromised, the smart home device connected to it could also be compromised. Similarly, if the platform used to manage the smart home devices is hacked, the attacker could gain access to all connected devices.

Recently, machine learning (ML) techniques have become popular for malware detection [10]–[12]. However, any dataset

for training a malware detection model inherently consists of imbalance class (a few malware samples vs a huge number of benign samples). It becomes more difficult ML training task when there are multiple class labels, which indeed is very common for malware detection for smart home applications. To the best of our best knowledge, no prior works have addressed the class imbalance problem in multi-class malware detection scenarios with ensemble models. The key contributions of our work are summarised below:

1) First, we investigated different oversampling methods to find the suitable one that can solve the class imbalance problem in our problem domain.
2) Second, we solve the multi-class imbalance scenario by performing extensive hyper-parameter tuning of the oversampling methods, and
3) Finally, hyper-parameter tuning of ensemble classifiers to assign the appropriate number of trees and their contribution in the learning process. The performance evaluation shows prominent improvement achieved in our model when compared with other existing works in terms of commonly used performance metrics, e.g., accuracy, precision, recall, and F-1 score.

The rest of this paper is organized as follows: Section II contains a review of current literature related to this study. In Section III, an explanation of the proposed method is provided. Section IV includes details about the dataset collection process, experimental results, and a comparative analysis of the proposed system. Lastly, in Section V, we present our future plans and conclusions for this paper.

II. Literature Review

Adoption of the smart home ecosystem is becoming increasingly popular as the interconnected devices in such a system offer a range of services, including remote monitoring of properties, power-efficient and comfort-based control of household amenities, real-time remote monitoring of patients, and help elderly people to live comfortably and safely [13]. The user interface is provided by smart mobile devices or a smarthome controller. A significant percentage of these mobile devices run the Android operating system due to their affordability and service quality [14]. However, these devices provide additional attack surfaces for hackers and malware developers. A successful attack can potentially lead to privacy breaches and financial loss, posing significant threats to the wide-scale adoption of smart home systems.

The proliferation of smart home appliances and facilitating applications have rendered manual inspection for malware detection infeasible. As a result, researchers have adopted various machine learning-based techniques for automatic malware analysis. The earlier generation of works relied on extracting permission features from the applications for malware detection. Aiman et al. [15] extracted the permissions from two categories of applications: business and tools. Applying the k-means clustering algorithm, the approach achieved a 71% recall rate. Suleiman et al. in [10] used a Bayesian classifier on a similar scenario and achieved an overall accuracy of 93%.

These techniques evolved over time as better, and more comprehensive feature extraction tools became available. Feizollah et al. in [16] extracted Inter Component Communication (ICC) features in addition to the permission features. ICC feature captures communication behaviour among intra- and inter-application components. The work achieved 91% accuracy on their dataset by applying a Bayesian Network algorithm. Drebin [11] was one of the earlier works that extensively considered a broad range of feature categories, including permission, Application Programming Interface (API) calls, and ICC. The work applied Support Vector Machine (SVM) for malware detection and achieved 94% accuracy on their dataset.

Yamauchi et al. [17] analyzed user interaction sequence with smart appliances (e.g., air conditioner) based on surrounding contexts (e.g., room temperature) for detecting anomalous behaviour that may potentially compromise user safety. Adopting a Hidden Markov Model (HMM) approach, the work obtained 90% detection accuracy on anomalous behaviour. However, the method only accounts for user behaviour ignoring malicious components that may be present in an application.

Tong and Yan conducted research on usage patterns in [12] by monitoring the applications and analyzing them dynamically. They then established a database of malicious and benign patterns by assessing the frequency and weight of sequential system calls at varying depths. To identify malicious applications, the system call sequences of the app are extracted and compared to the known signature database. However, this method's effectiveness largely depends on the number of applications used to create the database.

Smart home devices are typically connected to third-party services and platforms that offer additional features and functionalities to users. These services include smart assistants, cloud storage, remote management tools, and more. However, connecting smart home devices to third-party services also introduces new cybersecurity threats that need to be addressed. One major threat is the possibility of unauthorized access to smart home devices and the data they collect by third-party services. This can occur if a third-party service provider is compromised by hackers or if the provider is collecting data in ways that are not transparent or secure. For example, a smart assistant service that collects voice recordings may be vulnerable to attacks that can access and steal this data [18].

Another concern is the potential for third-party services to introduce new vulnerabilities to smart home devices. For example, if a smart home device connects to a cloud storage service that is not properly secured, attackers could potentially gain access to both the device and the data it stores. Similarly, if a remote management tool is used to control a smart home device, attackers could potentially take over the device and use it for malicious purposes. Third-party services may also introduce additional privacy risks to smart home devices. For example, if a smart home device connects to a social media platform, the platform may collect data about the user's interactions with the device and use this information for targeted advertising or other purposes. This could potentially

compromise the user's privacy and security [19]. To mitigate these risks, it is important to carefully evaluate third-party services before connecting smart home devices to them. Users should consider factors such as the provider's reputation, the security of the service, and the privacy policies in place. Additionally, it is important to regularly review and adjust privacy settings for connected devices and to use strong and unique passwords and other security measures to protect both the devices and the data they collect.

Afonso et al. [20] identified malware applications by analyzing the frequency of API and system calls using various machine learning models, including Random Forest (RF) and Naive Bayes. Their technique achieved 96.66% accuracy on their dataset. However, the method may fail to identify malware apps that do not satisfy a certain API level requirement.

Xu et al. [21] considered the imbalanced nature of data and generated synthetic malware examples in the fuzzy region where the classifier gets biased towards the majority class. Adding the synthetic samples to the training set enlarges the decision of the minority class in the fuzzy region and improves detection performance. However, they considered a binary classification problem (i.e., benign and malware only) and used a single classifier for the prediction task.

III. Proposed Approach

In real-world scenarios, the majority of applications are typically benign or regular, while a minority are malicious and classified as malware, leading to a classic data imbalance problem. To tackle this challenge, our proposed model employs two distinct steps. In the first step, we split the initial dataset into train and test subsets and employ classifiers SVM, Artificial Neural Network (ANN), Gradient Boost (GB), and RF to test the trained model, as depicted in Figure 1. In the second step, we address the data imbalance by applying various oversampling techniques to the train data. We experiment with different oversampling ratios and weighted values during data preprocessing and subsequently train the models using the oversampled data. We employ several oversampling techniques, such as SMOTE, Borderline-SMOTE, and ADASYN, the details of which are provided below. Further elaboration on the specific values utilized in the oversampling process can be found in Section IV-A.

A. Machine Learning models

We used different ML techniques to determine their efficacy in detecting malware in smart home applications. These techniques are briefly described below.

The SVM algorithm is widely used for classification and regression tasks. It functions by discovering the optimal hyperplane that separates the data points of different classes with the greatest margin. For binary classification, SVM identifies a hyperplane that separates positive and negative examples. This hyperplane is defined by a vector w and a scalar b, such that the equation of the hyperplane can be expressed as

$$w^T x + b = 0$$

, where x represents a data point, w is the weight vector, and b is the bias term. SVM strives to locate the hyperplane that maximizes the margin between positive and negative examples. The margin is calculated as the distance between the hyperplane and the closest data points from each class. The distance between a point x and the hyperplane can be computed as $\frac{|w^T x+b|}{|w|}$, where $|w|$ represents the norm of the weight vector. The margin can then be defined as margin $= \frac{1}{|w|}$.

RF is a machine learning algorithm that is commonly used for both classification and regression tasks. For multiclass classification using RF, the algorithm employs an extension of the typical decision tree algorithm that supports multiple classes. The mathematical details of RF for multiclass classification are as follows:

Given a training data set $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, where x_i represents the feature vector of the i^{th} training example and y_i represents the corresponding label.

RF constructs a set of decision trees, $T_1, T_2, ..., T_M$, each of which is trained on a random subset of the training data and a random subset of the features. Each decision tree T_i generates a probability distribution over the classes, which is computed as:

$$p_{ij} = \frac{k}{n}$$

where k represents the number of training examples in the j^{th} class that are assigned to the i^{th} leaf node of the tree T_i, and n represents the total number of training examples assigned to the i^{th} leaf node.

ANN can be mathematically described as a function that maps an input vector x to an output vector y, through a series of linear and nonlinear transformations. The function is composed of multiple layers of interconnected neurons, with each neuron applying an activation function to its weighted inputs.

The output of a single neuron can be represented mathematically as:

$y = f(w \cdot x + b)$

where y is the output of the neuron, f is the activation function, w is the weight vector of the neuron, x is the input vector, and b is the bias term.

The output of a layer of neurons can be represented as a matrix multiplication of the input vector and the weight matrix of the layer, followed by the addition of the bias vector of the layer:

$$y = f(W \cdot x + b)$$

where y is the output vector of the layer, W is the weight matrix of the layer, x is the input vector, and b is the bias vector of the layer.

The output of the entire network can be expressed as the composition of the individual layers:

$$y = f_n(W_n \cdot f_{n-1}(W_{n-1} \cdot ...f_1(W_1 \cdot x + b_1)... + b_{n-1}) + b_n)$$

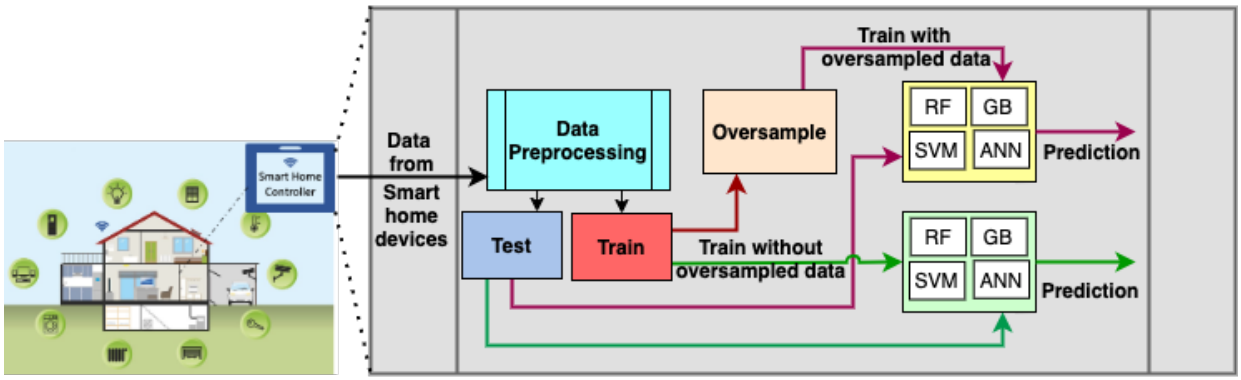

Fig. 1. Proposed model for malware detection with and without oversampling techniques embedded in the smart home controller.

where y is the output vector or scalar, f is the activation function, w is the weight vector of the neuron, W is the weight matrix of the layer, x is the input vector, b is the bias vector or scalar, n is the number of layers, and f_n is the activation function of the final output layer.

The weights and biases of the neurons in the network are adjusted during training using an optimization algorithm, such as stochastic gradient descent, to minimize the difference between the predicted output of the network and the true output.

B. Oversampling Techniques

SMOTE is a widely adopted data augmentation technique for synthesizing new minority class samples. First, a random sample (s) from the minority class is chosen along with its k neighbours. Then a random neighbour (n) within those k neighbours is selected, and a new sample is synthesized on an arbitrary point on the connecting line between s and n in feature space.

Hui et al. [22] introduced an extended version of SMOTE called Borderline-SMOTE that deals with the miss-classified minority instances that usually belong to the edge of a decision boundary. Borderline-SMOTE first identifies the examples that are close to the decision boundary between the minority and majority classes. These are examples that are misclassified by a simple classification algorithm, such as k-nearest neighbours or a decision tree, but not by a more complex algorithm, such as a support vector machine or a neural network. Once these examples are identified, Borderline-SMOTE generates synthetic examples by interpolating between these examples and their k nearest neighbours in the feature space.

ADASYN [23], on the other hand, adaptively adjusts the density of synthetic examples based on the level of class imbalance. ADASYN first computes the density of the minority class in each region of the feature space. It then generates synthetic examples using SMOTE but weights each example based on the inverse of its local density. This means that examples in regions with high-class imbalance are given more weight, and more synthetic examples are generated in those regions. Conversely, examples in regions with low-class imbalance are given less weight, and fewer synthetic examples are generated in those regions.

The oversampled data is supplied to machine learning models as shown in 1. Once the optimal oversampling technique is applied, the model uses the enriched training data to train machine learning classifiers. The model further refines the classifiers by tuning their hyperparameters, specifically the number of trees in ensemble models and their contribution to the learning process. The performance of these models is then evaluated using a testing set, with metrics including accuracy, precision, recall, and F-1 score. The final step is a comparative assessment of the proposed approach's performance against the original data without oversampling.

IV. Performance Evaluation

We have used Maldroid 2020 [24] dataset in our project, which was provided by the corresponding research group upon request. The CICMalDroid 2020 dataset is a comprehensive Android malware dataset that contains over 17,341 samples collected from various sources spanning from December 2017 to December 2018. The dataset is divided into five distinct categories, namely Adware, Banking malware, SMS malware, Riskware, and Benign. Adware has the smallest number of samples (1,253), followed by Banking malware (2,100), Riskware (2,546), SMS malware (3,904), and Benign (1,795).

A. Experimental setup

In this model, the train test split ratio is used as 80% and 20% for both steps (with and without oversampling). In addition to investigating the optimal train test split ratio and oversampling techniques, we also explored the effect of hyperparameter values on the performance of the model. Specifically, we experimented with different hyper-parameter values to determine which options provided the best results in terms of minimizing false detection.

To optimize the oversampling technique, we tested various ratios of minority-class samples to majority-class samples. Our results indicated that increasing minority class samples to 90% of the majority class samples provided the best result in terms of minimizing false detection. To further improve the performance of the model, we also experimented with

TABLE I

PERFORMANCE METRICS FOR DIFFERENT CLASSIFIERS WITH ORIGINAL DATA (NO OVERSAMPLING), USING SMOTE (SM), BORDERLINE-SMOTE (BSM), AND ADASYN (ADA). HERE, ADWARE, BANKING, SMSWARE, RISKWARE, AND BENIGN ARE REPRESENTED WITH 1, 2, 3, 4, AND 5, RESPECTIVELY IN CL (CLASSES)

Classifiers	Cl	Precision				Recall				F1-Score			
		Org	SM	BSM	ADA	Org	SM	BSM	ADA	Org	SM	BSM	ADA
SVM	1	0.843	0.884	0.877	0.851	0.428	0.674	0.710	0.682	0.568	0.765	0.785	0.757
	2	0.943	0.962	0.972	0.981	0.651	0.803	0.814	0.826	0.770	0.875	0.886	0.897
	3	0.965	0.976	0.996	0.940	0.564	0.822	0.843	0.826	0.712	0.892	0.913	0.879
	4	0.958	0.953	0.943	0.957	0.420	0.830	0.889	0.874	0.584	0.887	0.915	0.913
	5	0.945	0.987	0.977	0.983	0.582	0.810	0.822	0.798	0.720	0.890	0.893	0.881
ANN	1	0.809	0.844	0.857	0.841	0.484	0.774	0.728	0.764	0.606	0.808	0.787	0.800
	2	0.891	0.930	0.982	0.988	0.563	0.866	0.828	0.842	0.690	0.897	0.898	0.909
	3	0.915	0.978	0.977	0.962	0.573	0.942	0.940	0.923	0.705	0.960	0.958	0.942
	4	0.923	0.979	0.963	0.954	0.540	0.950	0.954	0.946	0.681	0.964	0.958	0.950
	5	0.910	0.943	0.970	0.989	0.620	0.966	0.964	0.928	0.738	0.954	0.967	0.957
GB	1	0.832	0.882	0.886	0.883	0.700	0.886	0.864	0.864	0.760	0.884	0.875	0.874
	2	0.940	0.950	0.973	0.958	0.761	0.964	0.963	0.942	0.841	0.957	0.968	0.950
	3	0.962	0.955	1.010	0.979	0.826	0.948	0.952	0.956	0.889	0.951	0.980	0.968
	4	0.926	0.981	0.983	0.978	0.840	0.921	0.918	0.910	0.881	0.950	0.949	0.943
	5	0.938	0.986	0.963	0.946	0.860	0.982	0.987	0.975	0.897	0.984	0.975	0.960
RF	1	0.853	0.941	0.948	**0.951**	0.680	0.946	**0.960**	0.946	0.757	0.944	**0.954**	0.938
	2	0.962	**0.985**	0.942	0.940	0.838	0.988	**0.992**	0.991	0.896	**0.987**	0.966	0.965
	3	0.971	**1.000**	0.999	0.996	0.845	0.990	**1.000**	0.990	0.904	0.995	**1.000**	0.993
	4	0.973	0.987	0.983	**0.997**	0.821	**1.000**	0.990	0.980	0.891	**0.993**	0.986	0.988
	5	0.951	0.991	0.990	**0.995**	0.803	**0.970**	0.950	0.960	0.871	**0.980**	0.970	0.977

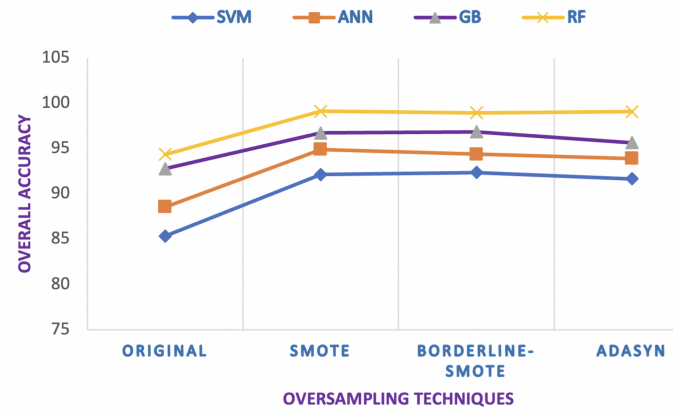

Fig. 2. Overall prediction accuracy of SVM, ANN, GB, and RF using SMOTE, Borderline-SMOTE, and ADASYN oversampling methods.

different learning rates for the ensemble methods. We varied the learning rate to explore its effect on the miss-classification rate of different classes, with the goal of optimizing overall prediction performance. We also varied the number of estimators used in the ensemble methods. By doing so, we aimed to determine the optimal number of trees required to achieve maximum prediction performance. By exploring these various hyper-parameter values, we were able to fine-tune the model and achieve the best possible results.

B. Experimental result

As shown in Table I, performance on the original data reveals a notable gap between Precision (0.843-0.965) and Recall (0.420-0.651) for SVM. After implementing oversampling, there is a clear improvement in Precision, Recall, and F1-score across all classes. BSM stands out for class 1 (Adware), showing the highest increase in recall and F1-score.

For ANN, precision rates on the original data range from 0.809 to 0.923. The recall rates, however, are considerably lower (0.484-0.620). Oversampling techniques significantly enhance the model performance. Notably, BSM for class 2 (Banking) and SM for class 3 (SMSware) provide the highest increase in precision and recall rates, respectively. The GB classifier starts off with a higher recall for the original data compared to SVM and ANN. An increase in performance metrics is observed across all classes, with ADASYN providing the best results for class 3 (SMSware) in terms of precision and SM for the same class in terms of recall and F1-score.

The RF classifier performs the best across all oversampling techniques. With the original data, it exhibits a higher recall rate compared to the other classifiers. Post oversampling, RF exhibits the highest precision for class 1 (Adware) using ADASYN, while SMOTE generates the highest recall and F1-score for the same class.

The overall accuracy (refer to Fig. 2) we found without oversampling the data is 85.32%, 88.54%, 92.77%, and 94.34% for SVM, ANN, GB, and RF, respectively. From Table I, we can see that the ensemble methods provide better results than the single classifiers. In all cases, the precision, recall, and F1-

scores were higher when oversampling techniques were applied. After oversampling, the precision for Adware increased to 0.851, 0.857, 0.886, from 0.951 from 0.843, 0.809, 0.832, and 0.853 for SVM, ANN, GB, and RF, respectively. Where, the recall has significantly improved from 0.428, 0.484, 0.70, 0.68, to 0.71, 0.774, 0.886, and 0.96, respectively. We can also see a similar pattern in better performance of the F1-score after oversampling and using ensemble methods.

The above-mentioned results and discussion suggest that applying oversampling methods using different oversampling ratios based on the requirement of each minority class provides the best result. Table II shows a comparison of our model with the result reported in [24] suggesting that our model achieved better results as we were able to address the class imbalance issues, mainly for the minority class sample (Adware).

TABLE II
OVERALL ACCURACY, PRECISION, RECALL, AND F1-SCORE OF CIC MALDROID 2020 DATA COMPARISON WITH OTHER WORK.

Detection models	CIC MalDroid 2020				
	Acc	Pre	Rec	F1	Method
Mahdavifar et al. [24]	96.7	99.16	96.54	97.84	PLDNN
Our Model	**99.12**	**99.5**	**97.0**	**98.0**	RF

V. CONCLUSION

The current popularity and expansion of smart home systems can be linked to their capacity to give safety, comfort, and remote monitoring support. Nevertheless, widespread implementation of such systems poses issues owing to the huge increase in worldwide cyberattacks, which may result in financial loss, the leaking of personal information, and the loss of life. Malware attacks pose a serious risk to smart home applications since the virus can compromise the controller system's security after infection. As a result, identifying malware in smart home systems is crucial for enhancing their security. To overcome these issues, this paper suggests using oversampling techniques in addition to ensemble-based machine learning models to improve prediction performance. Simulation results confirm that the proposed approach outperforms previous works regarding the accuracy, precision, recall, and F-1 score.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] P. Newman, "The internet of things 2020: Here's what over 400 iot decision-makers say about the future of enterprise connectivity and how iot companies can use it to grow revenue," *Business Insider*, pp. 1–6, 2020.

[3] J. Saleem, M. Hammoudeh, U. Raza, B. Adebisi, and R. Ande, "Iot standardisation: Challenges, perspectives and solution," in *Proceedings of the 2nd international conference on future networks and distributed systems*, 2018, pp. 1–9.

[4] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th {USENIX} security symposium*. Vancouver, BC, Canada: USENIX Association, 2017, pp. 1093–1110.

[6] G.-M. Sung, H.-K. Wang, and W.-T. Su, "Smart home care system with fall detection based on the android platform," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 3886–3890.

[7] B. Vaidya, A. Patel, A. Panchal, R. Mehta, K. Mehta, and P. Vaghasiya, "Smart home automation with a unique door monitoring system for old age people using python, opencv, android and raspberry pi," in *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2017, pp. 82–86.

[8] J. Cui, L. Wang, X. Zhao, and H. Zhang, "Towards predictive analysis of android vulnerability using statistical codes and machine learning for iot applications," *Computer Communications*, vol. 155, pp. 125–131, 2020.

[9] M. Rytel, A. Felkner, and M. Janiszewski, "Towards a safer internet of things—a survey of iot vulnerability data sources," *Sensors*, vol. 20, no. 21, p. 5969, 2020.

[10] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of bayesian classification-based approaches for android malware detection," *IET Information Security*, vol. 8, no. 1, pp. 25–36, 2014.

[11] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware," in *Ndss*, vol. 14, 2014, pp. 23–26.

[12] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in android," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 22–31, 2017.

[13] D. Bastos, M. Shackleton, and F. El-Moussa, "Internet of things: A survey of technologies and security risks in smart home and city environments," in *Living in the Internet of Things: Cybersecurity of the IoT*. IET, 2018.

[14] M. A. Omer, S. R. Zeebaree, M. A. Sadeeq, B. W. Salim, S. x Mohsin, Z. N. Rashid, and L. M. Haji, "Efficiency of malware detection in android system: A survey," *Asian Journal of Research in Computer Science*, pp. 59–69, 2021.

[15] A. A. A. Samra, K. Yim, and O. A. Ghanem, "Analysis of clustering technique in android malware detection," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*. IEEE, 2013, pp. 729–733.

[16] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121–134, 2017.

[17] M. Yamauchi, Y. Ohsita, M. Murata, K. Ueda, and Y. Kato, "Anomaly detection in smart home operation from user behaviors and home conditions," *IEEE Transactions on Consumer Electronics*, vol. 66, no. 2, pp. 183–192, 2020.

[18] R. Trimananda, S. A. H. Aqajari, J. Chuang, B. Demsky, G. H. Xu, and S. Lu, "Understanding and automatically detecting conflicting interactions between smart home iot applications," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1215–1227.

[19] H. Hu, L. Yang, S. Lin, and G. Wang, "A case study of the security vetting process of smart-home assistant applications," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020, pp. 76–81.

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

[21] Y. Xu, C. Wu, K. Zheng, X. Niu, and Y. Yang, "Fuzzy–synthetic minority oversampling technique: Oversampling based on fuzzy set theory for android malware detection in imbalanced datasets," *Int. Journal of Distributed Sensor Net.*, vol. 13, no. 4, pp. 1–15, 2017.

[22] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *Int. conf. on intelligent computing*. Springer, 2005, pp. 878–887.

[23] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *Int. joint conf. on neural networks*. IEEE, 2008, pp. 1322–1328.

[24] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," in *Intl Conf on Dependable, Autonomic and Secure Computing*. IEEE, 2020, pp. 515–522.