Collaborative Learning Based Effective Malware Detection System

Narendra Singh, Harsh Kasyap(✉), and Somanath Tripathy

Department of Computer Science and Engineering,
Indian Institute of Technology Patna, Patna, India
{1811cs10,harsh_1921cs01,som}@iitp.ac.in

Abstract. Malware is overgrowing, causing severe loss to different institutions. The existing techniques, like static and dynamic analysis, fail to mitigate newly generated malware. Also, the signature, behavior, and anomaly-based defense mechanisms are susceptible to obfuscation and polymorphism attacks. With machine learning in practice, several authors proposed different classification and visualization techniques for malware detection. Images have proved worth analyzing the behavior of malware. Deep neural networks extract much information from it without having expert domain knowledge. On the other hand, the scarcity of diverse malware data available with clients, and their privacy concerns about sharing data with a centralized curator makes it challenging to build a more reliable model. This paper proposes a lightweight Convolution Neural Network (CNN) based model extracting relevant features using call graph, n-gram, and image transformations. Further, Auxiliary Classifier Generative Adversarial Network (AC-GAN) is used for generating unseen data for training purposes. The model is extended for federated setup to build an effective malware detection system. We have used the Microsoft malware dataset for training and evaluation. The result shows that the federated approach achieves the accuracy closer to centralized training while preserving data privacy at an individual organization.

Keywords: Malware detection · Machine learning · Federated Learning · Feature extraction · Generative Adversarial Network

1 Introduction

Malware or malicious software has been an evolutionary area of research. It was initially intended as a software program to test architectural loopholes. With increasing technologies and money factors involved, programmers started exploiting those to disrupt the service and gain unauthorized access to another system. It led to research in this new field coined as malware detection. A variety of malware exists with different names as worms, viruses, trojans, ransomware, adware, and spyware. Some malware immediately reveals their presence while

I. Koprinska et al. (Eds.): ECML PKDD 2020 Workshops, CCIS 1323, pp. 205–219, 2020.
https://doi.org/10.1007/978-3-030-65965-3_13

others reside inside the machine to steal the identity. In the age of social-computing, targeted attacks can be more frequent than generalized ones, due to the exposed profiles and public news of an organization.

Static analysis techniques [1,2] have been used since long to detect malware. These techniques rely on the signature-based analysis of malicious programs without running it. They check for the file name, type, and size, checksums or hashes. On the other hand, malware authors have successfully written intelligent programs that trick static analysis techniques, with a small change of malware code and signature. Thus, encrypting and obfuscating the malware code evades detection. Down the lane, several researchers [2–4] used dynamic analysis techniques for classifying malware. It runs the malware and observes the infected files' behavior, including traffic analysis, registry keys, etc. These techniques require a secure and controlled environment like an emulated or virtual sandbox, which might affect the real-time running host.

With the machine and deep learning in practice, many works have been proposed which learn characteristics and relations from the malicious program and classifies them into respective categories. Authors trained different algorithms based on probabilistic and knowledge-based approaches including Hidden Markov Model [5], Naive-Bayes [6], Machine learning-based models, and Multi-layer perceptrons [7]. Unfortunately, these techniques do not suffice against a different dataset or real-time traffic and fail to mitigate targeted malware attacks. Meanwhile, deep learning-based approaches [8–10] do not require in-depth domain knowledge as compared to previous static, dynamic, or machine learning models.

Deep learning-based approaches require more data sets in order to learn inherent characteristics. For dynamic analysis, it becomes challenging to feed large samples, especially from backdoor families. Therefore collaborative approaches would be suitable, which collects data from different sources. Decentralized and collaborative learning is the current alternative and collects the data to make intelligent model. However, the data gets moved from its place, which raises security and privacy concerns. Even with a distributed and decentralized computing paradigm, the data may be kept or processed parallelly or at different locations. Federated Learning [11], a new approach where the model is trained, keeping the data in-place, and a central curator aggregates all the models. It runs the same process for thousands and millions of iterations. It faces trade-offs of energy, not independent and identically distributed data, network, and random participants. However, it preserves the data, which is most crucial for the organizations. We use Federated Learning with multiple participants contributing malware data to train an aggregated malware detection model.

This paper discusses the properties of malware, existing methods to mitigate them, and proposes a novel deep learning approach using a federated setup. The following are the major contributions.

– Features are extracted using call graph from .asm files, image transformations, and n-gram techniques from .bytes file. Preprocessing and combining these

features and feeding to convolutional neural networks showed an improved accuracy of 99.72%.
- Auxiliary Classifier GAN (AC-GAN) is used for generating unseen data, preparing the model for targeted attacks.
- We trained the model using a federated setup using the TensorFlow federated API with data distributed across clients and achieved 97.93% accuracy in a few iterations with scope for improvement.

The remainder of this paper is organized as follows. Section 2 discusses the background and related works. Section 3 briefs the concepts of Federated Learning and Auxiliary Classifier GAN (AC-GAN). Section 4 proposes the framework with dataset preprocessing, feature extraction, and the use of underlying concepts to run this experiment. Section 5 describes model configuration, experimental setup, result and comparison. Section 6 concludes and briefs the scope for future work.

2 Related Work

Malware detection is a state of the art to prevent intrusion of malicious software. The long is the history of malware, that long is the history of its detection and prevention. They both started all together, tricking and competing with each other. Every time a more robust defense mechanism comes, malware authors outsmart them by tweaking something new. In this section, We will discuss the existing defense techniques to mitigate malware and their drawbacks.

Lu et al. [10] discussed a malware detection method based on the word embedding technique and LSTM. It automatically learns the correlation between opcodes and feature representation of the opcode sequence. It achieves an accuracy of 95.73% for skip-gram and a continuous bag of words on the Microsoft malware dataset. Le et al. [12] used a CNN architecture with BiLSTM. They used a recurrent layer on top of the CNN architecture. It summarizes the content of the whole file into one feature vector. It achieves an accuracy of 98.8% on the Microsoft malware dataset.

Zhao et al. [9] proposed MalDeep, a novel deep learning-based malware classification technique based on texture visualization. They studied this classification through code mapping, texture partitioning, and texture extracting. The model works on CNN with two convolutional layers and down-sampling layers, and many full connection layers. It achieved higher accuracy of 99% on the Microsoft malware dataset (with nine malware families). The dataset has an uneven distribution of malware families, and this approach reached a better classification accuracy for backdoor families.

Azar et al. [8] used an unsupervised feature learning approach for malware classification and network-based anomaly detection using auto-encoder (AE). It produces a fixed (ten) size vector for both classification and detection of attacks, making it more useful for real-time detection. It achieves a classification accuracy of 96% on the Microsoft malware dataset.

Vasan et al. [13] proposed a deep learning model Image-based Malware Classification using Fine-tuned Convolutional Neural Network (IMCFN), which talks about the image visualization approach and uses CNN architecture. It is computationally cost-effective and provides a scalable solution. It incurs low run-time overhead and proves to be secure against obfuscated malware. It is tested on the Malimg dataset (with twenty-five malware families) and predicts the resilience of the obfuscated malware attack. It achieved an accuracy of 98.82%.

Generative Adversarial Network (GAN) [14], has been used to generate unseen data. It adds noise to the existing data and generates new samples that can not be distinguished from the real samples. These samples often trick the model. Therefore, prior training is done with these samples to boost confidence and be prepared for unknown attacks. Authors in [15,16] have used GAN for handling malware detection. Kim et al. [15] introduced GAN based approaches tGAN and tDCGAN, achieving an accuracy of 96.39% and 95.74% respectively. They compared different deep learning techniques, including multi-layer perceptron, auto-encoder, along with concepts of transfer learning. Authors in [17] ran multiple experiments using Auto-Encoders, and Deep Neural Networks with varying layers over Malicia malware dataset achieving an accuracy of 99.21%. However, they relied on the fact that Deep Learning-based systems are good in automatically extracting higher conceptual features.

3 Preliminaries

This section briefs about Federated Learning, Auxiliary Classifier Generative Adversarial Network (AC-GAN), and illustrates their mathematical description.

3.1 Federated Learning

We are aware of individual organizations' capabilities in terms of generating data and training a smart machine learning model. Previous centralized collaborative approaches were supposed to collect all these data to the central server and train, which induces a high data leak risk and violates various security principles and pacts. It does not guarantee security, privacy, and anonymity. Federated Learning brings the model to the data while keeping the data to the device itself.

Google introduced this concept of Federated Learning back in 2016, a collaborative learning approach that makes learning secure by keeping the data to the device itself. It is an iterative process, and each time the device trains the model with its data and updates the parameters to the central server. The server collects the data, computes federated average, and updates the devices with the latest parameters. This iterative process involves significant communication overhead, and multiple parameter exchanges make the system vulnerable and exposed to attackers. The central curator may allow the use of any optimization and aggregation algorithm depending on the problem. The whole process is illustrated in Fig. 1.

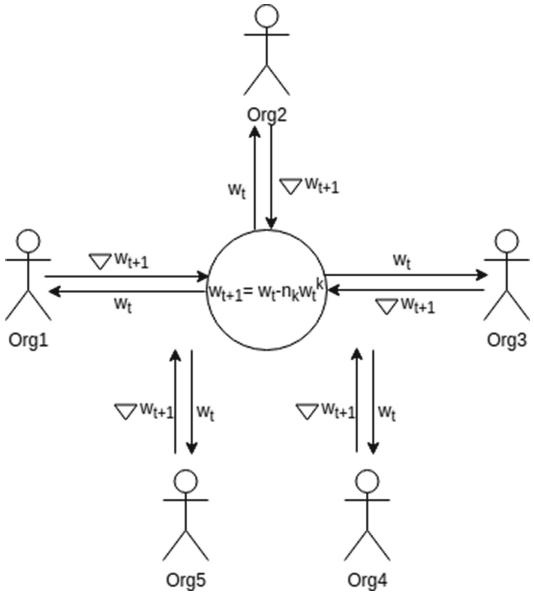

Fig. 1. Federated setup

FedSGD (a variant of stochastic gradient descent (SGD)) is a widely used optimization and aggregation algorithm for federated averaging. SGD samples a subset of summand functions and updates the weight.

$$w := w - \eta \nabla Q_i(w) \tag{1}$$

SGD is effective and scalable for large training sets coupled with complex gradient calculation formulae. In FedSGD, each participant trains the model in-place with some random samples chosen in every iteration and sends the delta change in the gradient to the central aggregator.

$$w_{t+1} = w_t + \eta \frac{\sum_{k \in S_t} n_k \Delta w_t^k}{\sum_{k \in S_t} n_k} \tag{2}$$

The central aggregator sums up the weighted contribution of the delta updates received from all the participants and updates the global weight. Let the system has a total K number of users. In every iteration, a fraction of clients participate, some may drop out. The set comprising of participating clients be S_t and n_k be the number of samples held by client k with the server having a learning rate of η. Let w_t be the global weight of the previous iteration, server updates it, and evaluates w_{t+1} using distributed approximate Newton method [18].

3.2 Auxiliary Classifier GAN (AC-GAN)

Generative Adversarial Network (GAN) is an advanced deep learning approach that generates new data from scratch, which does not exist in real-world training data. It composes of two players, which are also deep learning models called generator and discriminator. A Generative model G takes latent space, and noise from the sample distributes. It concatenates them to convert into a complex distribution, which is similar to the real data. A Discriminator model D is a binary classification neural network that aims to distinguish between real and fake samples generated by Generator model G.

AC-GAN is an extension of the conditional GAN (cGAN). cGAN handles the conditional generation of images, while AC-GAN allows the targeted generation of images of respective types. It makes the discriminator predict the target of input and stabilizes the training process allowing the generation of large-high-quality images. It learns representation in the latent space without knowledge of the target.

In the AC-GAN, every generated sample has a corresponding class label c and the noise z. The generator uses both c and z to generate images $X_{fake} = G(c, z)$. The discriminator gives both a probability distribution over sources and a probability distribution over the class labels.

$$P(S|X), P(C|X) = D(X) \tag{3}$$

The objective function has two parts: the log-likelihood of the correct source, L_S, and the log-likelihood of the correct class, L_C [19]. Discriminator is trained to maximize $L_S + L_C$ while Generator is trained to maximize $L_C - L_S$.

$$L_S = E[\log P(S = \text{real } |X_{\text{real}})] + E\left[\log P\left(S = \text{fake}|X_{\text{fake}}\right)\right] \tag{4}$$

$$L_C = E[\log P(C = c|\ X_{real})] + E\left[\log P\left(C = c|X_{fake}\right)\right] \tag{5}$$

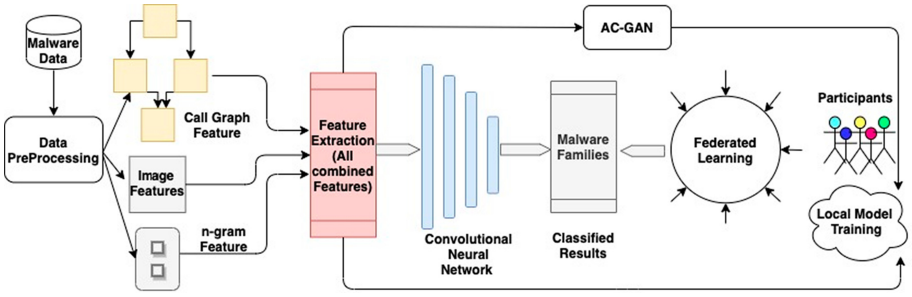

Fig. 2. Proposed framework

4 Proposed Framework

The proposed Malware detection framework facilitates the participants to operate collaboratively and build an effective machine learning model. Each participant and organization hold different data, and the central server acts as an aggregator. They train the model in-place and send the update to the server. The whole process is repeated multiple times. The proposed framework comprising of following major components. The rest of the process is finding relevant features using different transformations and feeding it to a convolutional neural network that classifies malware. It also uses AC-GAN to generate unseen data and defend against targeted attacks (Fig. 2).

4.1 Data Collection

We have used Microsoft Malware data[1] for analysis. The total train dataset size is 200 GB, out of which 50 GB is .bytes file, and the remaining 150 GB is .asm file. It consists of 10,868 .bytes files and 10,868 .asm files with total 21,736 files. In the dataset, .byte files are a combination of 256 hex numbers (a decimal value ranging between 0 to 255) and a special character (??). *.asm* files are the outputs of the disassembler.

Table 1. Types of Malware in Microsoft Malware dataset

Sr No	Name	Samples	Type
1	Ramnit	1541	Worm
2	Lollipop	2478	Adware
3	kelihosver3	2942	Backdoor
4	Vundo	475	Trojan
5	Simda	42	Backdoor
6	Tracur	751	Trojan downloader
7	Kelihosver1	398	Backdoor
8	Obfuscator.ACY	1228	Obfuscated
9	Gatak	1013	Backdoor

4.2 Data Preprocessing

Data Preprocessing involves Exploratory Data Analysis, Dimensionality Reduction, and Data Normalization. It identifies the meaning and aspects of feature engineering and standardizes data features with feature scaling by analyzing datasets and their examples. After analyzing the data, we observed the uneven

[1] https://www.kaggle.com/c/malware-classification/data.

distribution of target classes, as shown in Table 1. Samples with the label- one two and three are more than four, five, and six. It helped in useful dynamic classification to assign the respective families of new malware files encountered or generated, especially the backdoor families.

4.3 Feature Extraction

Many works discussed above have used n-gram and image transformation techniques to achieve higher accuracy. It incurs a higher computing cost in preprocessing and extraction. Authors in [20] discussed the conversion of the function-call graph to a vector in low-dimensional feature space for malware detection. It achieves optimum efficiency with less cost as we aim to propose a lightweight model that can be easily deployed on a federated setup with resource-constrained devices. We use call-graph features with 3-gram analysis and less intensive image operations. Our feature extraction component uses call graphs, n-gram features, and image transformation.

First, .asm files are transformed to extract call graph features by converting it into a control flow graph by tracing the flow sequentially. Control flow graph of an assembly program P is a directed graph G = (V, E), where V is the set of basic blocks, and $E \subseteq V \times V$ is the set of edges representing control flow between basic blocks. A control flow edge from block u to v is $e = (u, v) \in E$. Call Graph represents the relationship between different subroutines in a program. The nodes in the graphs denote the subroutines, and the link to them tells how they call each other. It is of both static and dynamic types. Though, a perfect static graph is an undecidable problem. It can be generated manually or using the software. We wrote a small piece of code to make the control flow graph from the given assembly files. The control flow graph is represented differently for statements, loops, and function calls. It can be used in extracting features for malware detection. It identifies uncalled procedures which can be inside a program for malicious intentions.

We extracted the features like the number of nodes and edges, maximum degree and density of the graph from the call graph. Table 2 lists some rows of features extracted from .asm files of the Microsoft malware dataset.

Table 2. Sample call graph features

File	nodes	edges	maxDeg	density
01lsAbcXXXX	274	333	137	0.0813
01sUzbxXXXX	187	196	82	0.1813
01kzSrtXXXX	158	1533	95	0.1409
01kxPcsXXXX	26	126	35	0.6

We can adequately describe malware through an n-gram analysis as a sequence of hex values. It is a contiguous sequence of n hexadecimal values

from a given malware file. Each sequence takes one out of 257 different values, i.e., the 256-byte range plus the special '??' symbol. For each .bytes file, we have made an array and added each element to the array. The length of these arrays is used as a feature. The .asm files consist of various segments, opcodes, and keywords, registers, etc. We have taken the count of segments as a feature and considered the bag-of-words representation of .bytes.

We convert the malware file into binary and take the first eight bits where four most significant bits map to rows and four least significant bits map to columns to make pixels. Images converted were of different sizes. So, we used padding to make them of the same size. The image visualization approach is very efficient but susceptible to noise. It raises data augmentation and motivates us to generate new unseen samples. Thus, We used GAN for generating unseen images and targeted files.

We combined all those different features extracted from .asm and .bytes files for our model. We extracted a total of 1024 features. After Applying dimensionality reduction and normalization techniques, we reduced it to 256 for feeding into a convolutional neural network of (16×16) input vector and 784 for feeding into GAN of (28×28) input vector. As limited data available with individual clients, it is challenging to model a robust defensive mechanism. So we set up a federated environment for collaborative learning.

4.4 Model Configuration

Convolutional Neural Network (CNN): We have used a lightweight model for running our experiment. It runs with fewer layers than the dense networks and requires optimal infrastructure to run, which can be easily deployed in any setting.

The convolutional layer is composed of 32 filters of size 3×3. It takes input as (None, 16, 16, 1). The output of this layer is (None, 14, 14, 32). Next, it is fed to a max-pooling layer, which takes the convolutional layer's output as input. It is of size 2×2, reducing the output to $7 \times 7 \times 32$. Then, the pooling layer is followed by another Flatten layer, which gives (None,1568) output. Another Dense layer follows that takes the output of the previous Flatten layer as input. Then we use the Dropout layer with a rate of 0.5, followed by a densely-connected layer with 256 neurons.

Auxiliary Classifier GAN (AC-GAN): We have used AC-GAN for generating new unseen malware at the client-side. In centralized training, these samples are generated at the server end. In Federated Learning, each participant will train their local data to create new samples that look exactly similar to the original malware and could have tricked the model.

AC-GAN takes a vector input. It concatenates the point in latent space (100 dimensions) and categorically encoded (9 dimensions). A fully connected layer interprets the point in latent space with sufficient activations to generate a grayscale image with the shape $28 * 28$ and pixel values in the range $[-1, 1]$ with

tanh activation function. The discriminator predicts the probability of the generated image belonging to a real or fake class, taking an input of shape 28 * 28 * 1 given by the generator. It is defined using Gaussian weight initialization, batch normalization, LeakyReLU, Dropout, and a 2 * 2 stride for downsampling instead of pooling layers. This architecture constructs the image with a single input and two outputs. It is trained with two loss functions, binary cross-entropy for the first output layer, and categorical cross-entropy loss for the second output layer using the Adam version of stochastic gradient descent with a small learning rate and modest momentum.

Federated Learning: It is a decentralized collaborative training approach. We have already discussed its benefits of limiting data exposure. In malware context, it can be argued that the malware families are already well known and exposed. So, It may not motivate to go with Federated Learning incurring more cost and similar accuracy. However, with ever-increasing malware families and in-house defense techniques developed for mitigating them, the companies can not afford to make the data public. This study proposes a federated setup for developing special defense techniques and securely improving a global malware detection system.

We have tested the above proposed lightweight convolutional network, with five participants randomly distributing the malware dataset with them. The section below discusses the implementation and API's used in detail.

5 Experimental Setup and Evaluation

This section describes the model configuration and setup carried out for running the proposed framework.

5.1 Setup

This experiment has been run on the federated testbed. The Tensorflow federated installation guide[2] lists different installation methods. Docker setup with ubuntu image has been used to run these experiments. It helps in easy installation and migration. We configured a host machine with processor Intel® Core™ i7-7700 CPU @ 3.60 GHz 8, and Memory 8 GB. Data preprocessing and feature extraction takes significant time due to massive data set.

5.2 Collaborative Training and Evaluation

We evaluated the proposed federated malware detection technique on the Microsoft malware data set, including 10868 files for model training and the same number of files for experimental evaluation. Although the data set is huge and comprises around two hundred fifty gigabytes, the number of samples is

[2] https://github.com/tensorflow/federated/blob/master/docs/install.md.

smaller. The data set consists of two types of files .bytes and .asm. The .bytes files contain the address and byte codes in hexadecimal format, while .asm files contain the address, segments, opcodes, registers, function calls, and APIs.

For running this experiment, the dataset is split with an approximate 60:20:20 ratio for training, validation, and test sets. We did data processing on the dataset and extracted features, as mentioned in the above section. We picked a thousand features from the images and twenty features using the n-gram technique. Using the above-discussed call graph technique, four features, i.e., no. of nodes, number of edges, max degree of a graph, and density of graph, were extracted by converting .asm files to control flow graph. Combining all features, we have a total of 1024 features to train our model.

We build our lightweight deep learning model using a convolutional neural network on a federated testbed with the setup mentioned above. The training model's structure derives from CNN with the Input layer followed by Conv layers, Maxpool, Fully Connected, Densely Connected, Dropout, and then Densely Connected layers. We implemented AC-GAN for generating adversarial samples for targeted training and enriching our sample space.

For running on the federated testbed, the data set has been randomly split into six equal-sized subsets. One subset is assigned to the central aggregator to initialize the model and its weights. It is also known as dummy data set and needs to be given iteratively. More help for the same can be found in Tensorflow federated documentation[3]. The rest five subsets are assigned to five participants. While training the model locally, participants use a batch size of twenty and run ten epochs for training a local model in one iteration. We trained the global model by running a different number of iterations to see improvement in model accuracy. We ran from 50 to 100 iterations denoted as F-50, F-60, F-70, F-80, F-90, and F-100 to see an increasing graph of model accuracy. The setup above failed to run more iterations. We expect an improved accuracy on a slightly higher configuration by running more number of iterations.

5.3 Result

Using Auxiliary Classifier GAN (AC-GAN), the central server generated 6200 new malware samples and confidently classified 35.2% of them as malware assigning corresponding class labels. These samples were used along with the (original/collected) training data and prove helpful in improving model accuracy of 99.72%. However, running the experiment with the only participant holding data in silos dips the accuracy to 96.34%. The confusion matrix for the centralized lightweight model is illustrated in Table 3. Precision, Recall, and F1-score are also shown for every malware family. The macro-avg for precision and recall comes 99% and 98%, respectively. It even shows high accuracy for backdoor malware families.

[3] Tensorflow federated. https://www.tensorflow.org/federated.

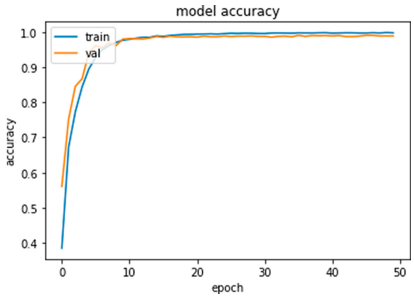

Fig. 3. Model accuracy

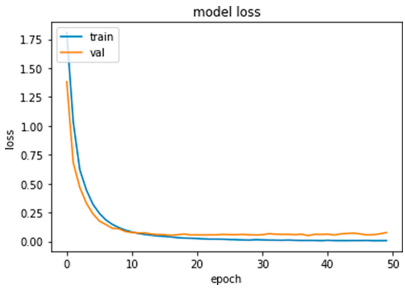

Fig. 4. Model loss

Table 3. Confusion matrix

Malware Family	Ramnit	Lollipop	kelihosver3	Vundo	Simda	Tracur	Kelihosver1	O.ACY	Gatak
Ramnit	0.9936	0.0004	0	0	0	0	0	0	0
Lollipop	0	1	0	0	0	0	0	0	0
kelihosver3	0	0	1	0	0	0	0	0	0
Vundo	0	0	0	1	0	0	0	0	0
Simda	0	0	0	0	1	0	0	0	0
Tracur	0.0032	0	0.0105	0	0	0.9933	0	0	0
Kelihosver1	0	0	0	0	0	0	1	0	0
O.ACY	0.0032	0	0	0	0	0	0.0067	0.9756	0.0197
Gatak	0	0	0	0	0	0	0	0	1
Precision	0.99	1	1	0.98	1	0.96	0.99	1	1
Recall	0.99	1	1	1	0.88	0.99	0.97	0.99	1
F1-score	0.99	1	1	0.99	0.93	0.98	0.98	0.99	1

In federated setup, with five clients and a central curator, it slightly reduced to 94.66% with 50 iterations. It gained and started showing significant improvement with an increasing number of iterations and reached to 97.93% with 100 iterations. Model accuracy and loss are plotted in Fig. 3 and Fig. 4. It illustrates the gain in accuracy with more local stochastic training at the client-side. It is expected to gain significant improvement with more global and local iterations. Table 4 lists the comparison of proposed architecture with and without federated setup along with previous works. The central setup has slightly higher accuracy compared to the federated setup but is not privacy-preserving in nature.

Table 4. Comparison Report with previous works

Sr No	Authors	Approach	Description	Accuracy
1	Lu et al. [10]	LSTM	Based on natural language processing, the word embedding technique, and LSTM	95.73%
2	Kim et al. [15]	tDCGAN	Deep Autoencoder-based GAN	95.74%
3	Kim et al. [15]	tGAN	Pre-train the generator of GAN using the transfer learning and Detection Using Deep Transferred GAN	96.39%
4	Le et al. [12]	CNN-BiLSTM	Classifies the one dimensional representation of the binary file using local patterns of each malware class	98.8%
5	Azar et al. [8]	AE-SVM	Autoencoder-based feature learning	96.3%
6	Zhao et al. [9]	CNN	Conversion from binary file to gray images including code mapping, texture partitioning, and texture extracting	99%
7	Centralized Model	CNN	Feature Extraction using call graph, Image features and training CNN model, generating unseen data with AC-GAN	99.72%
8	Federated Learning	CNN	Five clients and a central curator running above model in 100 iterations, expecting improved performance with more iterations	97.93%

6 Conclusion

The existing malware detection techniques are sufficient to mitigate against known attacks. However, targeted attacks against companies and individuals pose a severe threat as they model in-house defense techniques and can not publicize them due to privacy threats. In a federated setting, every participant can develop its own model and helps to improve the global model keeping their data private. We presented such a collaborative learning approach to model a malware detection system. The proposed approach transformed, extracted, and merged relevant features from Microsoft malware data and modeled a light convolutional network. Using Auxiliary Classifier GAN (AC-GAN), 6200 generated samples helped to mitigate targeted and adversarial attacks. With the Federated setting, five clients, and a central curator, the accuracy is found to be degraded gracefully, while the privacy of each participant is preserved.

Though, Federated Learning is privacy-preserving in nature, but, it poses issues of significant communication overhead and inference attack with presence of malicious server or adversarial participants. There is a scope to study poisoning attacks in a similar direction and handle the latency trade-offs. Introducing anomalies in networks will result in a dropout of clients. It may introduce error in synchronization of updates, which will affect the results. Any model does not suffice without heterogeneous data and makes a scope for improvement to work with, not IID malware data.

Acknowledgement. We acknowledge the Ministry of Human Resource Development, Government of India, for providing fellowship to complete this work.

References

1. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Twenty-Third Annual Computer Security Applications Conference, ACSAC 2007, pp. 421–430, December 2007
2. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. Proc. Comput. Sci. **46**, 804–811 (2015)
3. Carlin, D., Cowan, A., O'Kane, P., Sezer, S.: The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes. IEEE Access **5**, 17 742–17 752 (2017)
4. Harel, D. (ed.): First-Order Dynamic Logic. LNCS, vol. 68. Springer, Heidelberg (1979). https://doi.org/10.1007/3-540-09237-4
5. Pechaz, B., Jahan, M.V., Jalali, M.: Malware detection using hidden Markov model based on Markov blanket feature selection method. In: 2015 International Congress on Technology, Communication and Knowledge (ICTCK), pp. 558–563, November 2015
6. Liu, C., Zhang, Z., Wang, S.: An android malware detection approach using Bayesian inference. In: 2016 IEEE International Conference on Computer and Information Technology (CIT), pp. 476–483, December 2016
7. Rathore, H., Agarwal, S., Sahay, S.K., Sewak, M.: Malware detection using machine learning and deep learning. CoRR, vol. abs/1904.02441 (2019). http://arxiv.org/abs/1904.02441
8. Yousefi-Azar, M., Varadharajan, V., Hamey, L., Tupakula, U.: Autoencoder-based feature learning for cyber security applications. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 3854–3861, May 2017
9. Zhao, Y., Xu, C., Bo, B., Feng, Y.: MalDeep: a deep learning classification framework against malware variants based on texture visualization. Secur. Commun. Netw. **2019**, 1–12 (2019)
10. Lu, R.: Malware detection with LSTM using opcode language. ArXiv, vol. abs/1906.04593 (2019)
11. McMahan, H.B., Moore, E., Ramage, D., y Arcas, B.A.: Federated learning of deep networks using model averaging. CoRR, vol. abs/1602.05629 (2016). http://arxiv.org/abs/1602.05629
12. Le, Q., Boydell, O., Namee, B.M., Scanlon, M.: Deep learning at the shallow end: malware classification for non-domain experts. CoRR, vol. abs/1807.08265 (2018). http://arxiv.org/abs/1807.08265
13. Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q.: IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Netw. 107138 (2020). http://www.sciencedirect.com/science/article/pii/S1389128619304736
14. Goodfellow, I., et al.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates Inc. (2014). http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
15. Kim, J.Y., Bu, S.J., Cho, S.B.: Malware detection using deep transferred generative adversarial networks. In: Liu, D., Xie, S., Li, Y., Zhao, D., El-Alfy, E.S. (eds.) ICONIP 2017. LNCS, vol. 10634, pp. 556–564. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70087-8_58
16. Hu, W., Tan, Y.: Generating adversarial malware examples for black-box attacks based on GAN. CoRR, vol. abs/1702.05983 (2017). http://arxiv.org/abs/1702.05983

17. Sewak, M., Sahay, S.K., Rathore, H.: An investigation of a deep learning based malware detection system. CoRR, vol. abs/1809.05888 (2018). http://arxiv.org/abs/1809.05888
18. Shamir, O., Srebro, N., Zhang, T.: Communication efficient distributed optimization using an approximate newton-type method. CoRR, vol. abs/1312.7853 (2013). http://arxiv.org/abs/1312.7853
19. Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier GANs. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, Series Proceedings of Machine Learning Research, 06–11 Aug 2017, vol. 70, pp. 2642–2651. International Convention Centre. PMLR, Sydney, Australia. http://proceedings.mlr.press/v70/odena17a.html
20. Jiang, H., Turki, T., Wang, J.T.L.: DLGraph: malware detection using deep learning and graph embedding. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1029–1033 (2018)