Federated Learning for the Efficient Detection of Steganographic Threats Hidden in Image Icons

Nunziato Cassavia[1], Luca Caviglione[2], Massimo Guarascio[1(✉)], Angelica Liguori[3], Giuseppe Surace[3], and Marco Zuppelli[2]

[1] Institute for High Performance Computing and Networking, Rende, Italy
{nunziato.cassavia,massimo.guarascio}@icar.cnr.it
[2] Institute for Applied Mathematics and Information Technologies, Genova, Italy
{luca.caviglione,marco.zuppelli}@ge.imati.cnr.it
[3] University of Calabria, Rende, Italy
angelica.liguori@dimes.unical.it

Abstract. An increasing number of threat actors takes advantage of information hiding techniques to prevent detection or to drop payloads containing attack routines. With the ubiquitous diffusion of mobile applications, high-resolution icons should be considered a very attractive carrier for cloaking malicious information via steganographic mechanisms. Despite machine learning approaches proven to be effective to detect hidden payloads, the mobile scenario could challenge their deployment in realistic use cases, for instance due to scalability constraints. Therefore, this paper introduces an approach based on federated learning able to prevent hazards characterizing production-quality scenarios, including different privacy regulations and lack of comprehensive datasets. Numerical results indicate that our approach achieves performances similar to those of centralized solutions.

Keywords: Federated Learning · Information Hiding · Deep Learning · Malware Detection

1 Introduction

Among the various techniques used by threat actors to distribute malicious payloads and update configuration of malware, the adoption of steganography is becoming a consolidated practice [2]. Despite the hiding process may vary, e.g., it can target network traffic to bypass intrusion detection systems or shared hardware resources to allow processes to elude sandboxes, digital images are the most effective and adopted carriers [14]. For instance, a recent attack campaign exploited steganography to drop a malicious Golang executable on the host of the victim. To this aim, the malware has been cloaked in security certificates of

C. Comito and D. Talia (Eds.): PerSOM 2022, LNICST 494, pp. 83–95, 2023.
https://doi.org/10.1007/978-3-031-31469-8_6

pictures taken by the James Webb space telescope. As a consequence of effective advancements in offensive approaches, a surge in the volume of attacks has been observed and threat actors can now endanger almost any device or host, for instance to distribute ransomware [18]. To face such a challenging scenario, security should be enforced from the very early phases of the development process (e.g., by means of by-design approaches) as well as in the various stages of the distribution pipeline.

Unfortunately, typical countermeasures could not be sufficient to mitigate threats exploiting information hiding techniques, especially when targeting mobile devices [3]. Prime evidences on the effectiveness of image steganography to elude standard security checks can be rooted back to 2014. In this case, attackers concealed secret data by exploiting digital media contained in applications made available through the Google Play Store [23]. Indeed, machine learning frameworks are becoming key tools to partially mitigate the impact of threats exploiting advanced offensive schemes [4]. For instance, they demonstrated to be effective for checking the behaviors of mobile applications, which are usually based on a complex interplay of software components [28].

Therefore, along the lines of [1], this paper addresses the problem of revealing malicious payloads hidden within high-resolution icons that are commonly used in most popular mobile ecosystems, including, Android and iOS. Despite machine learning approaches proven to be effective to spot several offensive techniques based on obfuscation and information hiding (see, e.g., [4] and [6]), deploying such frameworks in real-world scenarios could be unfeasible. For instance, resource-intensive computations may not be possible in a centralized manner, due to scalability constraints, lack of suitable datasets, or the use of a multitude of shared libraries and software components [16,29]. As a workaround, this paper introduces a supervised approach exploiting federated learning. As observed in many settings, computation can be distributed across multiple cloud replicas or edge nodes, which can cooperate to the definition of models by exploiting local information (e.g., crawled from the Internet or gathered from "unofficial" application stores). Moreover, the federated approach could prevent GDPR-like hazards due to processing operations performed in areas with conflicting policies about data confidentiality [19].

Summing up, the contribution of this paper is the design and the performance evaluation of a federated approach that allows multiple "app stores" to cooperate for revealing the presence of applications bundled with steganographic threats.

The rest of the paper is structured as follows. Section 2 reviews past works considering federated approaches for counteracting malware, whereas Sect. 3 introduces the reference scenario. Section 4 deals with the framework, while Sect. 5 showcases numerical results obtained through simulations. Finally, Sect. 6 concludes the paper and portraits some possible future research directions.

2 Related Works

Enforcing security of mobile applications usually relies upon a variety of techniques (e.g., static binary analysis, anomaly-based detection, enforcement of poli-

cies in end nodes), which require a strict cooperation among developers, users and administrators of application stores [7]. In general, the current trend takes advantage of some form of machine learning or artificial intelligence to analyze behaviors of software, exploit existent attack signatures and reveal unexpected interactions among applications or their components[1]. For the specific case of attacks leveraging information hiding to target digital media, machine learning confirmed its effectiveness, especially to support the steganalysis process that can drive the neutralization of cloaked data [15]. Unfortunately, the application of machine-learning-capable techniques often clashes with some practical constraints. First, the inspection of bundled assets and copyrighted material should be designed in order to respect privacy-enforcing regulations, or by relying on architectures that do not process any personal information [20]. Second, the mobile ecosystem is growing on a continuous basis, thus leading to millions of samples to verify. An application can be made available also via different store replica for performance purposes or through unofficial channels (e.g., alternative stores or via sideloading). As a result, the creation of comprehensive datasets is a hard task [25]. To partially cope with such drawbacks, federated approaches are becoming a prime tool, see, e.g., [21] for a thorough discussion even if focused on the case of IoT ecosystems.

For the specific case of revealing contents cloaked in digital images via distributed frameworks, [27] exploits federated transfer learning to improve the performance of image steganalysis while preserving the privacy of users. Even if the work partially overlaps with our idea, the considered scenario is completely different, i.e., it considers end nodes instead of applications stores and does not focus on real attacks or malware samples. For the case of using federated learning to tame realistic threats, the literature offers various attempts. As an example, [12] showcases a framework for enabling end nodes running Android to classify several types of malware, including ransomware and spyware but not steganographic threats. The problem of classifying malicious samples is also addressed in [13], which considers a generic scenario not related to security of mobile applications. Instead, [22] addresses the problem of detecting malware but limited to an OS no longer used (i.e., Symbian S60) and does not consider application stores or modern software distribution pipelines.

A possible "meet in the middle" blueprint offloads end nodes towards edge entities placed at the border of the network. In this case, cooperating stores could resembles such an architecture. However, the literature does not offer prior attempts based on edge computing to reveal the presence of threats endowed with information hiding or image steganography capabilities. In fact, this paradigm, jointly with federated techniques, has been largely used in IoT scenarios often composed of resource-constrained nodes [24]. Besides, for the specific case of mobile security, edge/federated approaches have been mainly adopted to guarantee privacy constraints. As a paradigmatic example, [11] demonstrates how to detect malware without exposing sensitive information of end users, such as configuration details or how various application program interfaces are invoked.

[1] Cloud-based protection mechanisms at the basis of the Google Play Protect framework: https://developers.google.com/android/play-protect/cloud-based-protections.

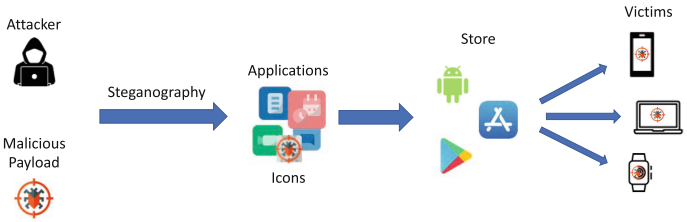

Fig. 1. Reference attack scenario considered in this work.

3 Attack Model and Federated Approach

The general attack model considered in this work deals with a threat actor wanting to hide a malicious payload within icons of applications by using steganography. Such a scheme could be exploited to make the reverse engineering of the attack chain harder or to distribute additional assets (e.g., configurations, URLs or small scripts) without triggering standard security mechanisms, such as those based on signatures or the static analysis of software. Each icon is then "repacked" within an application and then published through a store to make it available to users. Figure 1 depicts the reference attack scenario. To hide the malicious payload, we consider an attacker using the plain Least Significant Bit (LSB) technique, which has been observed in various real-world campaigns [2,14]. In essence, LSB allows to alter the least significant bit(s) of the color components of each pixel of the container image to conceal a secret. We point out that, the more bits are altered, the higher the chance of revealing the presence of the hidden payload via visible alterations or artifacts. To mitigate such an attack, there is the need of deploying a suitable scheme within the store to "reveal" the presence of hidden data and prevent that a malicious application is delivered. See [1] for a detailed discussion of a possible architecture using a centralized blueprint.

To detect the hidden data, we leverage a federated-learning-based approach to learn a global, optimal model in a distributed fashion. Figure 2 depicts our reference architecture. In the following, to prevent the need of formal definitions, we will refer to the centralized store as the server. Similarly, with end nodes we will identify other (groups of) machines located in the Internet and cooperating towards the distribution of applications and the detection process. In more detail, we assume that the server contains various mobile applications along with their icons, e.g., it can be considered an "app store". To prevent computational or security hazards, the server also contains pointers to some applications acting as a sort of "cache", for instance for the most popular contents. Instead, the end nodes represent local datacenters containing a subset of applications/icons already present in the main store and replicated for redundancy. Moreover, end nodes can also contain novel/unseen data, e.g., collected from third-part markets. To spot the presence of an application/icon hiding a malicious content, end nodes and the server collaborate to find the optimal DNN-based model via a federated

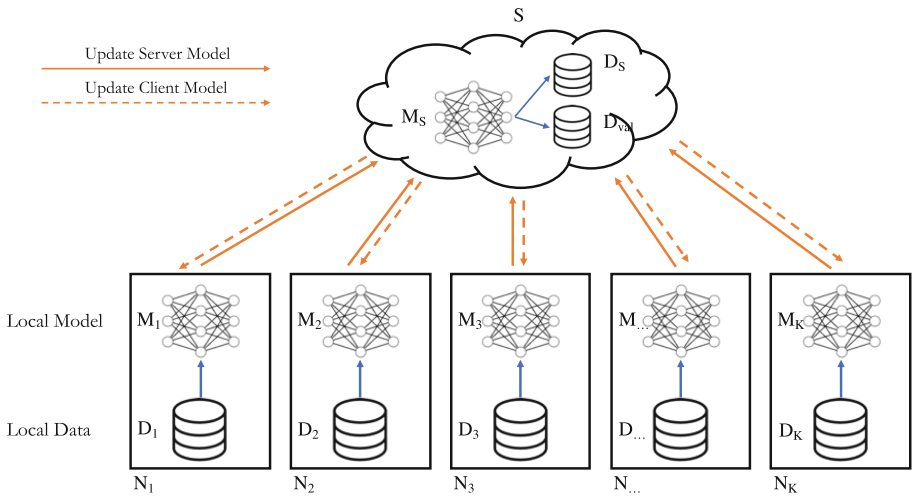

Fig. 2. Federated approach of cooperating stores.

approach. As discussed, such strategies are mainly used to avoid moving raw data from end nodes to the server, to take advantage of the computational capabilities of each node, and to guarantee the privacy of local devices.

Referring again to Fig. 2, we suppose to have a centralized server denoted as S in the figure. The server contains a "weak" DNN detector model M_S trained on an initial dataset D_S and validated through the validation set D_{val}. At the beginning, the detector is shared across K end nodes, which then fine-tune their model M_i against the local data D_i. To make the predictor more robust and to find a global model in a distributed manner, a subset of end nodes periodically sends updates to the server S containing the weights of each layer composing their local DNN. The server S aggregates the information received to obtain an ensemble model, which is validated against the validation set. If the model performs better with respect to the previous one, then the server sends back the best parameters to the end nodes. This process is iterated until a certain convergence criteria is reached. More formally, Algorithm 1 details the federated learning algorithm for training the malware classifier. As regards the procedures to yield the ensemble model `CreateSoupModel` and to fine-tune both global and local models `FineTune`, they are fully described in Sect. 4.3.

4 Framework

In this section, we first illustrate the methodology used to detect and classify compromised images, then we describe the neural architecture devised to tackle these problems. Finally we present the ensemble solution adopted in our FL-based approach to combine the different neural models yielded by end nodes.

Algorithm 1: (Federated) Learning algorithm for training the classifier.

1 `BuildFLModel(`S,N,*max_iter*,*eval_criterion*`)`
  Input : A server node $S = \{M_S, D_S, D_{val}\}$ acting the role of coordinator
  List of K peer nodes $N = [N_1, \ldots, N_i, \ldots, N_K]$ where $N_i = \{M_i, D_i\}$
  The max number of federated learning iterations *max_iter*
  The criterion for measuring the model performances *eval_criterion*
  Output: federated learning based model BM
2 M_S = `InduceNNModel(`D_S`)` // *build a detector against server data*
3 $BM = M_S$ // *initialize the best federated model BM*
4 $M_N = [\,]$ // *initialize the list of peer models*
5 Q_{BM} = `ComputePerformances(`BM,D_{val}, *eval_criterion*`)`
6 $i = 0$ // *current federated learning iteration*
7 $k = 0$ // *index of the current Node*
8 **foreach** $k \leq K$ **do**
9 $M_k = M_S$ // *download server model and initialize local model*
10 M_k = `FineTune(`M_k, D_k`)` // *finetune local model against local data*
11 $M_N \overset{+}{\leftarrow} M_k$ // *share local model with S and append the k-th model*
12 $k = k + 1$
13 **end**
14 M_S = `CreateSoupModel(`M_N, D_S`)` // *create an ensemble model*
15 Q_{M_S} = `ComputePerformances(`M_S,D_{val},*eval_criterion*`)`
16 **if** $Q_{BM} \leq Q_{M_S}$ **then**
17 $BM = M_S$
18 $Q_{BM} = Q_{M_S}$
19 **end**
20 **if** $i \leq$ *max_iter* **then**
21 $M_S = [\,]$ // *clean the list of peer models*
22 $i = i + 1$
23 **go to** line 7
24 **end**
25 **return** BM

4.1 Solution Approach

Figure 3 depicts the general methodology leveraged to cope with the problem of discovering images targeted via steganographic methods. Specifically, digital images represent the input of the proposed approach and are modeled as matrices with dimension $X \times Y$. The pixel is the smallest manageable element of these matrices and stores information about the color. The color of each pixel can be decomposed into three main components i.e., Red (R), Green (G) and Blue (B). As hinted, in this work we focus on high-resolution icons as they offer a sort of "unified playground" for various threats. At the same time, this does not account for a loss of generality, as the approach can be applied and scaled also to address regular-sized images. In the following, we then consider that the values associated to RGB components represent the intensity of the various colors and

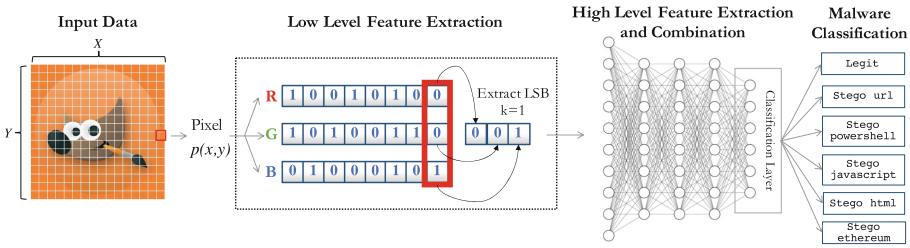

Fig. 3. Methodological approach for the classification of a stegomalware cloaked in a digital image via LSB steganography.

each value ranges in the interval $[0, 255]$. Hereinafter, we denote with N the size of the image computed as $N = X \times Y \times 3$.

Concerning the hiding method, LSB steganography represents a prominent approach to hide malicious code or data in legitimate pictures by changing the value of the (k) least significant bit(s) of each color composing the pixel of the image (see, Fig. 3 for the case of $k = 1$). When only a limited number of changes are performed on the image, it will not exhibit any visible alteration, i.e., pixels will look homogeneous compared to the surrounding elements [30]. As a consequence, many approaches proposed in the literature partially fail to detect the presence of hidden content as they produce weak detection models unable to discover the slight differences between licit and compromised contents.

To address all these issues, in this work we devised a (Federated) Deep Learning approach that processes and analyzes the k LSBs of the image icons under investigation. Basically, the first block of the proposed neural network is devoted to yield a flat representation of the image by extracting the k least significant bits of each pixel. Such a representation is then propagated to the subsequent layers to detect/classify different malicious contents. The DNN allows to extract high-level discriminative features to be further combined for producing the final classification.

4.2 Neural Architecture

To mitigate the impact of threats leveraging information hiding, we designed a supervised neural architecture for the classification task of image icons. In more detail, we exploited the deep architectures shown in Fig. 4, which permits to produce reliable predictions. Essentially, our neural architecture is composed of a stack of several blocks. The first layer acts as an handler for the input provided to the network (denoted as `Input Handler` in Fig. 4) and propagates the information (i.e., the image) to the subsequent layers of the DNN for further processing. The second component (denoted as `Low Level Feature Extraction` in Fig. 4) yields a flat representation of the image and extracts the raw information by means of a masking procedure.

The overall DNN is composed of a variable number m of `Building Block BB` obtained by stacking three main components: *(i)* on top, a fully-connected dense

layer (equipped with a Rectified Linear Unit (ReLU) activation function [17]) is instantiated, *(ii)* then, a batch-normalization layer is stacked to the previous one in order to improve the stability of the learning phase and to boost the performances of the model, and finally, *(iii)* a dropout layer is added to the subnet to mitigate the risk of overfitting [9].

Figure 4 details the building block architecture for the first instance of this specific configuration and has been labeled as $\texttt{BB}_1$. In more detail, the `Batch Normalization` allows for standardizing the data to be propagated to the subsequent layers of the DNN with respect to the current batch (by considering the average μ and the variance σ of each input). A reset of a random number of neurons in the training phase is performed via a dropout mechanism. As pinpointed in [10], the usage of the dropout method induces in the DNN a behavior similar to an ensemble model. Hence, the overall output of the whole neural network can be considered as the combination of different sub-networks resulting from this random masking, which disables some paths of the neural architecture.

In our experiments the neural model is instantiated with $m = 4$ building blocks. The proposed neural classifier also includes a skip connection to implement a residual block. The usage of the skip connections induces in the base DNN classifier a behavior similar to *Residual Networks* [8], which demonstrated to be effective solutions to the well-known *degradation problem* (i.e., neural networks performing worse at increasing depth), and capable of ensuring a good trade-off between convergence rapidity and expressivity/accuracy. Moreover, the high-level features extracted by the building blocks $\texttt{BB}_2$ and $\texttt{BB}_4$ are concatenated and used to feed the output layer of the model.

Finally, the `Output Layer` is instantiated with C neurons (one for each class) and equipped with a *softmax* activation function [5]. The proposed neural model is trained against a set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_D, \mathbf{y}_D)\}$, where $\mathbf{x}_i$ is the matrix representation of the image and $\mathbf{y}$ is the class of the image. As regards the output, an one-hot encoding based on C classes is used to model the different labels each one indicating a specific malicious payloads. As it will be detailed later, in our work we considered C classes representing "clean" images and images cloaking JavaScript, HTML, PowerShell, Ethereum wallets, and URL/IP addresses. Finally, the training stage is responsible for optimizing the network weights by minimizing the loss function. The *categorical crossentropy* is adopted for the classification task and it is calculated as follows:

$$CCE(\mathbf{y}, \tilde{\mathbf{y}}) = -\sum_{i=1}^{|\mathcal{D}|} \mathbf{y}_i \log \tilde{\mathbf{y}}_i.$$

4.3 Ensembling via Soup Model

As discussed in Sect. 3, the proposed federated Algorithm 1 uses a soup model mechanism to merge the contribution of each model produced by the K peer nodes [26]. Basically, the underlying idea of this ensemble consists in performing an average of the DNN weights yielded by the peer model, per layer. This strategy

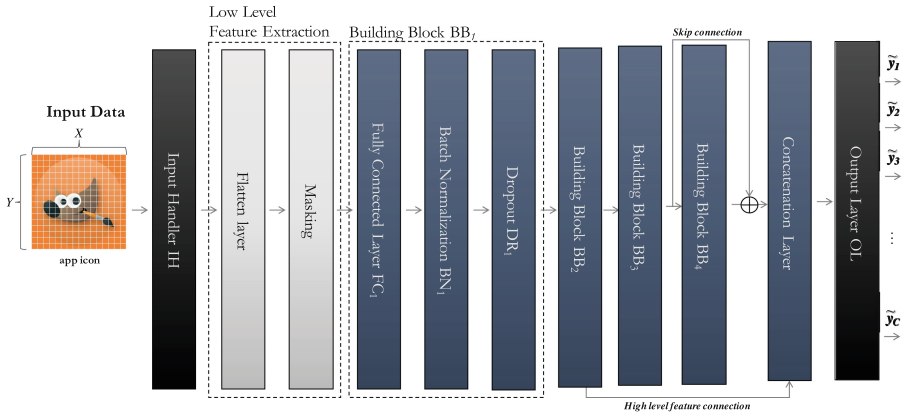

Fig. 4. Neural architecture for hidden content detection and classification.

has been named *Uniform Soup*. Formally, let be f(x, θ) a neural network with input data x and parameters $\theta \in \mathcal{R}^{\mathcal{D}}$. Let be $\theta = \texttt{FineTune}(\theta_0, \texttt{x})$ the parameters obtained by fine-tuning the pre-trained initialization θ_0 against data x. Let be $\theta_i = \texttt{FineTune}(\theta_0, \texttt{x}_\texttt{i})$ the parameters obtained by fine-tuning θ_0 against x_i, i.e., data of the node N_i. Model soup $f(x, \theta_T)$ is computed as an average θ_i, i.e., $\theta_T = \frac{1}{|K|} \sum_{i \in K} \theta_i$, where K is the number of peer nodes.

5 Experimental Results

To prove the effectiveness of our federated approach, we used the "Stego-Images-Dataset"[2] described in [1]. In essence, it is composed of 48,000 icons of 512×512 pixels hiding different realistic malicious payloads, i.e., JavaScript, HTML, PowerShell, URLs, and Ethereum addresses, embedded via the LSB steganography technique. The payloads allow to model a wide-range of threats, such as malicious scripts and routines, links to additional configuration files or list of commands, and wallets collecting the outcome of cryptojacking and ransomware campaigns. The dataset is split into 16,000, 8,000, and 8,000 icons, corresponding to the training, the validation and the test set. The training set is further divided among the server and the end nodes composing our architecture. In more detail, the 25% of the set (4,000 icons) is used to train the model on the server S, whereas the remaining 75% is assigned to the $K = 5$ end nodes, i.e., 15% images (2,400 icons) for each node. Instead, the validation set is used in its entirety to validate the ensemble model of the server and partially to validate the models of the end nodes. Finally, the dataset contains three different test sets (each one composed of 8,000 icons) to model an attacker unaware/aware of the countermeasure and trying to elude the detection via obfuscation approaches. In particular, the first is generated considering "plain" payloads, i.e., the attacker is completely unaware

[2] https://www.kaggle.com/datasets/marcozuppelli/stegoimagesdataset.

of the detection mechanism, whereas the others consider payloads encoded in Base64 and compressed with a zip method. Such datasets model an attacker performing a sort of "lateral movement" to bypass security checks.

To evaluate our approach, we relied upon the following metrics[3]:

- *F1-Score*: it summarizes the overall system performances and it is defined as the harmonic mean of the precision and recall. Specifically, the precision is calculated as $\frac{TP}{TP+FP}$, whereas the recall is calculated as $\frac{TP}{TP+FN}$;
- *Area Under the Curve (AUC)*: it is the area under the Receiver Operating Characteristic curve, obtained by plotting the ratio between the false positive rate and the true positive rate (i.e., the recall) for different class probability values;
- *AUC-PR*: it is the area under the Precision-Recall curve, obtained by plotting the precision and the recall for different class probability values.

The first round of tests aimed at evaluating the effectiveness of our federated-learning-based approach in detecting malicious payloads hidden within images. Table 1 summarizes the obtained results and shows how the performance of the end nodes (i.e., peers) improves over 10 iterations of the algorithm. As reported, the average AUC of the end nodes improves from 94.3% in the 1-st iteration to a maximum value of 96.5% when the *max_iter* number of iterations defined in Algorithm 1 is reached. Also AUC-PR and F1-Score exhibit the same behavior: both metrics improve up to 82.9% and 81.1%, respectively. As a consequence, the performances of the server improve as well, i.e., from an AUC of 92.6% to 97.1%. A similar trend can be observed for the other metrics (the best values are reported in bold in Table 1). Such results demonstrate that federated learning can be effectively used to reveal the presence of concealed contents, while guaranteeing privacy and overcoming possible resource constraints.

The second round of tests aimed at comparing the federated approach against a "centralized" blueprint, i.e., when all the data are stored in the node S. Moreover, we also evaluated the different approaches when dealing with payloads obfuscated by the attacker, i.e., via Base64 encoding and zip compression. Table 2 showcases the results. Concerning plain and Base64-encoded payloads, the differences between the approaches are minimal. Instead, in the case of compressed zip payloads the federated solution achieves an improvement of ∼10% in terms of AUC and AUC-PR with respect to the centralized approach. Summing up, the main benefit in using the federated blueprint relies on the capability of the solution to achieve comparable performances with a fully centralized method without the necessity to move data in a single node. In this way, this peculiarity allows for using our approach also in scenarios in which the storage resources are limited, e.g., IoT ecosystems.

[3] TP is the number of positive cases correctly classified, FP is the number of negative cases incorrectly classified, FN is the number of positive cases incorrectly classified, and TN is the number of negative cases correctly classified.

Table 1. Performance of the federated approach.

Iteration	Model	AUC	AUC-PR	F1-Score
Initialization	*server*	0.926	0.745	0.699
1	$peer_{avg}$	0.943	0.775	0.737
	server	0.926	0.745	0.699
2	$peer_{avg}$	0.955	0.805	0.770
	server	**0.959**	**0.819**	**0.763**
3	$peer_{avg}$	0.955	0.804	0.768
	server	0.959	0.819	0.763
4	$peer_{avg}$	0.960	0.816	0.779
	server	0.959	0.819	0.763
5	$peer_{avg}$	0.960	0.816	0.782
	server	0.959	0.819	0.763
6	$peer_{avg}$	0.959	0.813	0.774
	server	0.959	0.819	0.763
7	$peer_{avg}$	0.960	0.820	0.783
	server	**0.962**	**0.826**	0.641
8	$peer_{avg}$	0.965	0.829	0.797
	server	**0.971**	**0.845**	0.744
9	$peer_{avg}$	0.959	0.818	0.783
	server	0.971	0.845	0.744
10	$peer_{avg}$	0.965	0.829	0.811
	server	0.970	0.842	**0.817**

Table 2. Comparison between centralized and federated approaches against different test sets.

Approach	Coding	AUC	AUC-PR	F1-Score
Centralized	Plain	**0.972**	**0.851**	**0.835**
	Base64	**0.899**	**0.605**	0.589
	zip	0.776	0.397	0.344
Federated	Plain	0.970	0.842	0.817
	Base64	0.893	0.594	**0.614**
	zip	**0.856**	**0.498**	**0.363**

6 Conclusions and Future Works

In this paper we have presented a federated framework for the detection of malicious assets cloaked within images of applications delivered through stores. Results showcased the effectiveness of the approach, which can lead to per-

formances similar to those achieved via centralized solutions. Yet, a federated mechanism could be able to prevent constraints and bottlenecks characterizing single-point blueprints, e.g., scalability issues and lack of comprehensive snapshots for training the models.

Future work aims at extending our approach to detect different steganographic threats, including malware hiding data in network traffic or in other multimedia carriers. Moreover, part of our ongoing research is devoted to make the overall idea more general and not only limited to the case of stores. Specifically, we are working towards an edge-based architecture with nodes placed at the border of the network cooperating to detect/classify Internet-wide threats.

References

1. Cassavia, N., Caviglione, L., Guarascio, M., Manco, G., Zuppelli, M.: Detection of steganographic threats targeting digital images in heterogeneous ecosystems through machine learning. J. Wirel. Mob. Netw. Ubiquit. Comput. Dependable Appl. **13**, 50–67 (2022)
2. Caviglione, L., Mazurczyk, W.: Never mind the malware, here's the stegomalware. IEEE Securi. Priv. **20**(5), 101–106 (2022)
3. Cheddad, A., Condell, J., Curran, K., Mc Kevitt, P.: Digital image steganography: survey and analysis of current methods. Signal Process. **90**(3), 727–752 (2010)
4. Gibert, D., Mateu, C., Planes, J.: The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. **153**, 102526 (2020)
5. Guarascio, M., Manco, G., Ritacco, E.: Deep learning. Encycl. Bioinform. Comput. Biol.: ABC Bioinform. **1–3**, 634–647 (2018)
6. Guarascio, M., Zuppelli, M., Cassavia, N., Caviglione, L., Manco, G.: Revealing MageCart-like threats in favicons via artificial intelligence. In: Proceedings of the 17th International Conference on Availability, Reliability and Security, pp. 1–7 (2022)
7. He, D., Chan, S., Guizani, M.: Mobile application security: malware threats and defenses. IEEE Wirel. Commun. **22**(1), 138–144 (2015)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
9. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
10. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
11. Hsu, R.H., et al.: A privacy-preserving federated learning system for Android malware detection based on edge computing. In: 15th Asia Joint Conference on Information Security (AsiaJCIS), pp. 128–136. IEEE (2020)
12. Jiang, C., Yin, K., Xia, C., Huang, W.: FedHGCDroid: an adaptive multi-dimensional federated learning for privacy-preserving Android malware classification. Entropy **24**(7), 919 (2022)

13. Lin, K.Y., Huang, W.R.: Using federated learning on malware classification. In: 2020 22nd International Conference on Advanced Communication Technology (ICACT), pp. 585–589. IEEE (2020)
14. Mazurczyk, W., Caviglione, L.: Information hiding as a challenge for malware detection. IEEE Secur. Priv. **13**(2), 89–93 (2015)
15. Monika, A., Eswari, R.: Prevention of hidden information security attacks by neutralizing stego-malware. Comput. Electr. Eng. **101**, 107990 (2022)
16. Mylonas, A., Kastania, A., Gritzalis, D.: Delegate the smartphone user? Security awareness in smartphone platforms. Comput. Secur. **34**, 47–66 (2013)
17. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML), Haifa, Israel, pp. 807–814 (2010)
18. Oz, H., Aris, A., Levi, A., Uluagac, A.S.: A survey on ransomware: evolution, taxonomy, and defense solutions. ACM Comput. Surv. **54**(11s), 1–37 (2022)
19. Papageorgiou, A., Strigkos, M., Politou, E., Alepis, E., Solanas, A., Patsakis, C.: Security and privacy analysis of mobile health applications: the alarming state of practice. IEEE Access **6**, 9390–9403 (2018)
20. Pawlicka, A., Jaroszewska-Choras, D., Choras, M., Pawlicki, M.: Guidelines for stego/malware detection tools: achieving GDPR compliance. IEEE Technol. Soc. Mag. **39**(4), 60–70 (2020)
21. Rahman, S.A., Tout, H., Talhi, C., Mourad, A.: Internet of things intrusion detection: centralized, on-device, or federated learning? IEEE Netw. **34**(6), 310–317 (2020)
22. Shamili, A.S., Bauckhage, C., Alpcan, T.: Malware detection on mobile devices using distributed machine learning. In: 20th International Conference on Pattern Recognition, pp. 4348–4351. IEEE (2010)
23. Suarez-Tangil, G., Tapiador, J.E., Peris-Lopez, P.: Stegomalware: playing hide and seek with malicious components in smartphone apps. In: Lin, D., Yung, M., Zhou, J. (eds.) Inscrypt 2014. LNCS, vol. 8957, pp. 496–515. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16745-9_27
24. Tian, P., Chen, Z., Yu, W., Liao, W.: Towards asynchronous federated learning based threat detection: a DC-Adam approach. Comput. Secur. **108**, 102344 (2021)
25. Wang, H., Li, H., Guo, Y.: Understanding the evolution of mobile app ecosystems: a longitudinal measurement study of Google Play. In: The World Wide Web conference, pp. 1988–1999 (2019)
26. Wortsman, M., et al.: Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning, vol. 162, pp. 23965–23998. PMLR (2022)
27. Yang, H., He, H., Zhang, W., Cao, X.: FedSteg: a federated transfer learning framework for secure image steganalysis. IEEE Trans. Netw. Sci. Eng. **8**(2), 1084–1094 (2020)
28. Yuan, Z., Lu, Y., Xue, Y.: DroidDetector: Android malware characterization and detection using deep learning. Tsinghua Sci. Technol. **21**(1), 114–123 (2016)
29. Zhou, W., Zhou, Y., Jiang, X., Ning, P.: Detecting repackaged smartphone applications in third-party android marketplaces. In: Proceedings of the Second ACM Conference on Data and Application Security and Privacy, pp. 317–326 (2012)
30. Zuppelli, M., Manco, G., Caviglione, L., Guarascio, M.: Sanitization of images containing stegomalware via machine learning approaches. In: Proceedings of the Italian Conference on Cybersecurity (ITASEC), vol. 2940, pp. 374–386 (2021)