Social Network Analysis and Mining (2023) 13:114
https://doi.org/10.1007/s13278-023-01121-9

ORIGINAL ARTICLE

# A federated approach for detecting data hidden in icons of mobile applications delivered via web and multiple stores

Nunziato Cassavia[1] · Luca Caviglione[2] · Massimo Guarascio[1] · Angelica Liguori[1,3] · Giuseppe Manco[1] · Marco Zuppelli[2]

Received: 28 February 2023 / Revised: 12 May 2023 / Accepted: 25 August 2023 / Published online: 14 September 2023

**Abstract**
An increasing volume of malicious software exploits information hiding techniques to cloak additional attack stages or bypass frameworks enforcing security. This trend has intensified with the growing diffusion of mobile ecosystems, and many threat actors now conceal scripts or configuration data within high-resolution icons. Even if machine learning has proven to be effective in detecting various hidden payloads, modern mobile scenarios pose further challenges in terms of scalability and privacy. In fact, applications can be retrieved from multiple stores or directly from the Web or social media. Therefore, this paper introduces an approach based on federated learning to reveal information hidden in high-resolution icons bundled with mobile applications. Specifically, multiple nodes are used to mitigate the impact of different privacy regulations, the lack of comprehensive datasets, or the computational burden arising from distributed stores and unofficial repositories. Results collected through simulations indicate that our approach achieves performances similar to those of centralized blueprints. Moreover, federated learning demonstrated its effectiveness in coping with simple "obfuscation" schemes like Base64 encoding and zip compression used by attackers to avoid detection.

**Keywords** Federated learning · Information hiding · Deep learning · Malware detection · Stegomalware

## 1 Introduction

*Context and Motivations.* The ubiquitous diffusion of mobile devices is a major driver for the development of new threats. For instance, attackers successfully developed techniques to eavesdrop unencrypted communications or to steal computing resources for mining cryptocurrencies (Almaiah et al. 2021). At the same time, the creation of mechanisms to exfiltrate sensitive data has also intensified, especially to gather personal details during reconnaissance campaigns (Mazurczyk and Caviglione 2021). In this perspective, the creation of effective countermeasures to mitigate advanced threats targeting mobile devices is mandatory to improve the security of the Internet.

Unfortunately, the availability of efficient schemes for the static and dynamic analysis of mobile applications ignited an "arm race" between defenders and attackers (Spreitzenbarth et al. 2013). As an example, mechanisms implemented in many stores to bounce unsafe applications can be eluded by loading additional code at run time (Poeplau et al. 2014). Among the various techniques to distribute attack routines or extend offensive functionalities, the adoption of steganography is becoming very popular among threat actors (Caviglione and Mazurczyk 2022). Even if the cloaking mechanism may vary, attackers mainly exploit digital images, especially owing to their capacity and popularity (Mazurczyk and Caviglione 2015; Caviglione and

✉ Angelica Liguori
angelica.liguori@dimes.unical.it

Nunziato Cassavia
nunziato.cassavia@icar.cnr.it

Luca Caviglione
luca.caviglione@ge.imati.cnr.it

Massimo Guarascio
massimo.guarascio@icar.cnr.it

Giuseppe Manco
giuseppe.manco@icar.cnr.it

Marco Zuppelli
marco.zuppelli@ge.imati.cnr.it

1 ICAR-CNR, Via Pietro Bucci 8-9/C, 87036 Rende, Italy

2 IMATI-CNR, Via de Marini 6, 16149 Genova, Italy

3 Unical, Via Pietro Bucci, 87036 Rende, Italy

Mazurczyk 2022). Steganographic malware then conceals additional code or configuration data (e.g., IP addresses to contact) within innocent-looking images or icons. Prime evidences of threats using this technique have been observed in 2014 within applications published on the Google Play Store (Suarez-Tangil et al. 2014).

To face such a challenging scenario, security should be enforced in a holistic manner, starting from the very early phases of the development process until the final delivery of the software. Alas, the most popular countermeasures (e.g., security-by-design, sandboxing, and code signatures) often fail to mitigate threats exploiting information-hiding techniques, especially when deployed in mobile ecosystems (Cheddad et al. 2010). To this extent, Machine Learning (ML) and Artificial Intelligence (AI) are now core tools to mitigate the impact of threats leveraging advanced offensive schemes (Gibert et al. 2020). For instance, ML demonstrated its effectiveness to inspect mobile applications for revealing rogue code within software components (Yuan et al. 2016) or to detect hidden/obfuscated payloads (Gibert et al. 2020; Guarascio et al. 2022).

*Objectives.* The main goal of this work is to overcome some limitations characterizing many machine learning approaches used to improve the security of mobile ecosystems targeted by steganographic threats. In fact, despite their effectiveness, AI-based solutions often clash with the constraints of production-quality scenarios. For instance, resource-intensive computations may not be possible in a centralized manner due to scalability issues. Another limitation concerns the lack of appropriate datasets, including the need of merging details of a wide range of shared libraries and software components (Mylonas et al. 2013; Zhou et al. 2012). For the case of revealing malicious data hidden in mobile applications, the surge of multiple stores to bypass national constraints as well as the availability of unofficial distribution vectors (e.g., p2p networks) prevent having a single, Internet-scale framework. Therefore, this paper introduces a supervised approach exploiting federated learning, which can be distributed across multiple cloud replicas or edge nodes cooperating for the definition of models. Information can be also provided by a local replica of a store, crawled from Web pages hosting the software (e.g., `.apk`), or gathered from unofficial stores highly popular in regions like Asia (Li et al. 2017). The federated approach may also prevent GDPR violations due to processing operations hosted in areas with incompatible policies on data confidentiality (Papageorgiou et al. 2018).

*Contributions and Improvements.* The contributions of this paper are twofold. First, it showcases a framework based on a federated approach that allows cooperation among different "app stores" or Web sources to reveal applications containing steganographic threats. Second, it presents a performance evaluation not limited to hiding techniques observed in real samples but also considering contents obfuscated to prevent detection. Compared to our preliminary work (Cassavia et al. 2023), this paper has the following improvements: it is more focused on applications that can be obtained through multiple sources including the Web and unofficial stores, and it largely extends the performance evaluation campaign, especially by investigating the impact of "elusive" schemes exploiting zip compression and Base64 encoding.

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

## 2 Attack model and federated approach

The general attack model considered in this work analyzes a scenario where a threat actor employs steganography to conceal a harmful payload in application icons to evade detection. Such a scheme could be exploited to make the reverse engineering of the attack chain harder or to distribute additional assets (e.g., configuration files, URLs pointing to remote servers, or small scripts) without triggering standard security mechanisms based on signatures or static analysis of software. Each icon is "repacked" within an application and then published through a store to make it available to users. Moreover, malicious applications could be made available through alternative stores, repositories or Web sources (e.g., AppChina, Anzhi and F-Droid (Li et al. 2017)). 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 (Caviglione and Mazurczyk 2022; Mazurczyk and Caviglione 2015). In essence, LSB steganography alters 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 to deploy a suitable scheme within the store to "reveal" the presence of hidden data and prevent that a malicious application is delivered (see, e.g., Cassavia et al. (2022) and the references therein for the case of centralized distribution pipelines).

To detect the hidden data, in this work, we leverage a federated-learning-based approach to learn a global, optimal model in a distributed fashion. Figure 2 depicts our reference architecture. For the sake of simplicity, we will refer to the

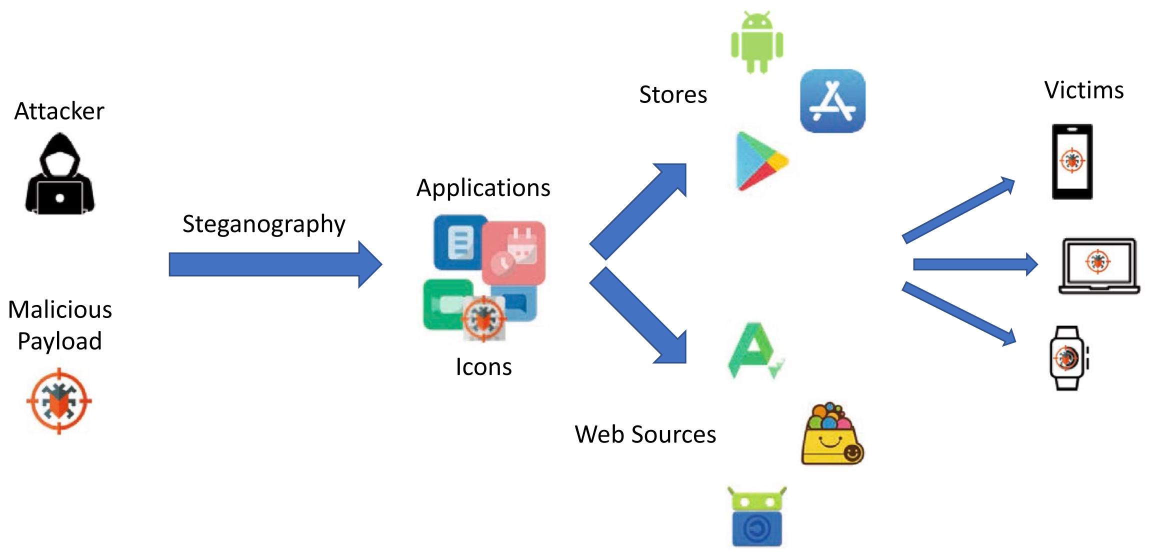


**Fig. 1** Reference attack scenario where mobile applications "repacked" with icons cloaking malicious contents are delivered through a store or made available via the Web

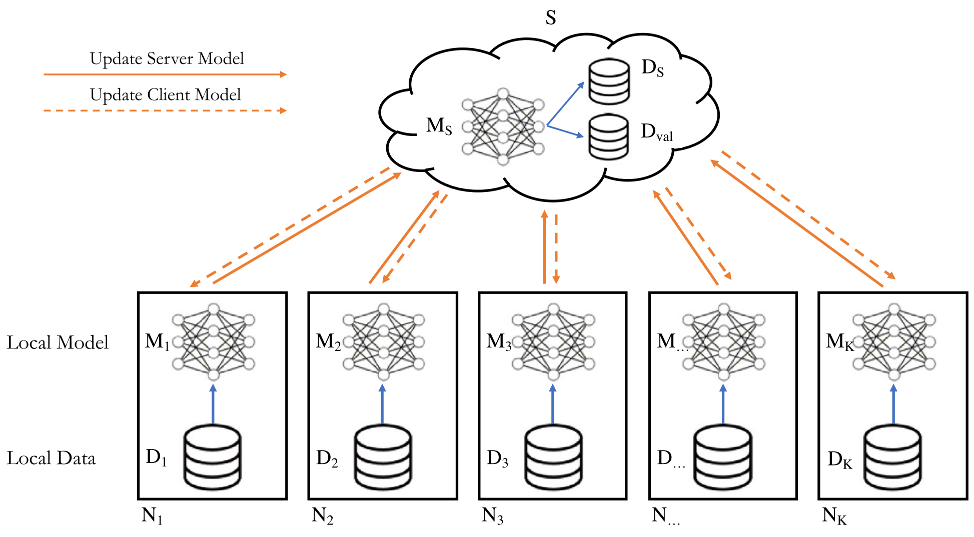


**Fig. 2** Federated approach of cooperating stores

centralized store as the server. Similarly, with end nodes, we will identify other (groups of) machines located on the Internet cooperating toward the distribution of applications and the detection process. Specifically, 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 data centers containing a subset of applications/icons already published in the main store and replicated for redundancy. End nodes can also contain novel/unseen data collected from third-party markets or scraped from Web sources and social media, e.g., by retrieving `.apk` or `.ipa` bundles directly from repositories. To spot the presence of an application/icon hiding malicious content, end nodes and the server collaborate to find the optimal Deep Neural Network (DNN)-based model via a federated approach. Such strategies are mainly used to avoid moving raw data from end nodes to the server, take advantage of each node's computational capabilities, and enforce privacy constraints of local devices or users.

Referring again to Fig. 2, we supposed 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 $D_{val}$. In the early stage, the detector is shared across $K$ end nodes, which 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 $S$ merges the information received to obtain an ensemble model, and its predictive performances are evaluated 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 criterion is reached. More formally, Algorithm 1 details the federated learning algorithm for training the malware classifier. The procedures to yield the ensemble model and to fine-tune both global and local models are fully described in Sect. 3.3, and have been named `CreateSoupModel` and `FineTune`, respectively.

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

```
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 $N$ =[$N_1, \dots, N_i, \dots, N_K$] 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
1  $M_S$ = InduceNNModel($D_S$) // build a detector against server data
2  $BM = M_S$ // initialize the best federated model BM
3  $M_N$ = [ ] // initialize the list of peer models
4  $Q_{BM}$ = ComputePerformances($BM$,$D_{val}$, $eval\_criterion$)
5  $i = 0$ // current federated learning iteration
6  $k = 0$ // index of the current Node
7  foreach $k \le K$ do
8      $M_k = M_S$ // download server model and initialize local model
9      $M_k$ = FineTune($M_k$, $D_k$) // finetune local model against local data
10     $M_N \overset{+}{\leftarrow} M_k$ // share local model with $S$ and append the $k$-th model
11     $k = k + 1$
12 end
13 $M_S$ = CreateSoupModel($M_N$, $D_S$) // create an ensemble model
14 $Q_{M_S}$ = ComputePerformances($M_S$,$D_{val}$,$eval\_criterion$)
15 if $Q_{BM} \le Q_{M_S}$ then
16     $BM = M_S$
       $Q_{BM} = Q_{M_S}$
17 end
18 if $i \le max\_iter$ then
19     $M_S$ = [ ] // clean the list of peer models
20     $i = i + 1$
21     go to line 6
22 end
23 return $BM$
```

## 3 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 for combining the different neural models yielded by end nodes.

### 3.1 Solution approach

Figure 3 depicts the general methodology for discovering images compromised via steganographic methods. In more detail, digital images represent the input of the proposed approach and are modeled as matrices with dimensions $X \times Y$. The pixel is the smallest manageable element of these

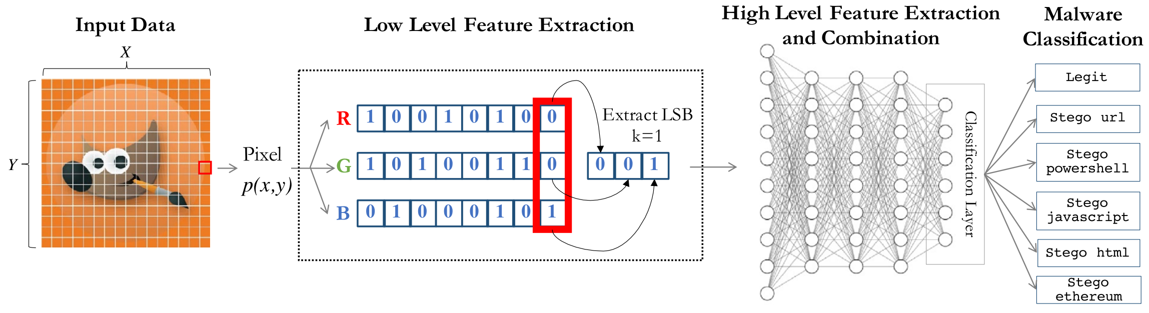


**Fig. 3** Methodological approach for the classification of a stegomalware cloaked in a digital image via LSB steganography

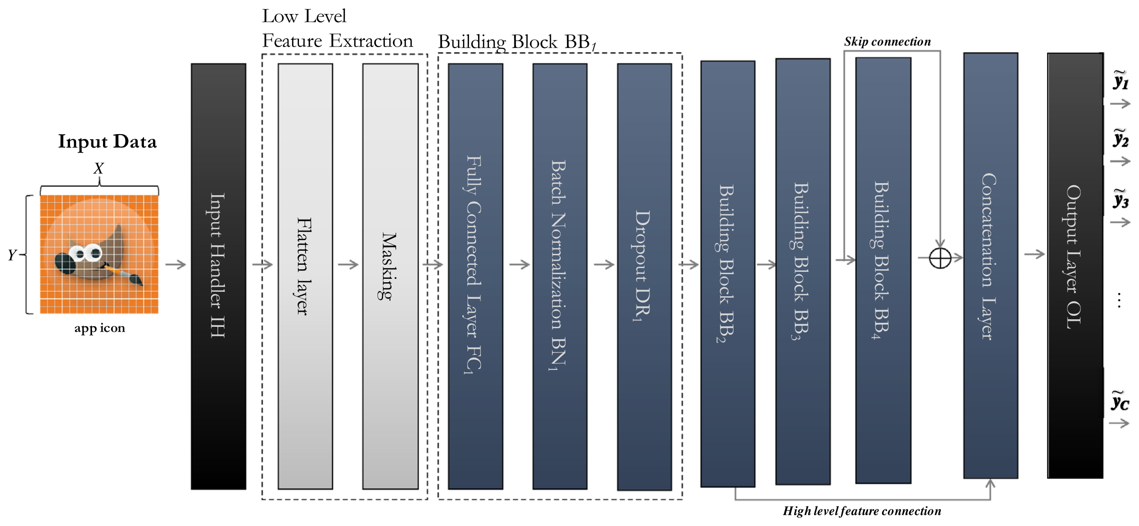


**Fig. 4** Neural architecture for hidden content detection and classification

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). The values associated with the RGB components represent the intensity of the various colors and each value ranges in the interval [0, 255]. Hereinafter, we denote with $N$ the size of the image computed as $N = X \times Y \times 3$. 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.

Concerning the hiding method, LSB steganography is considered 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 pixels of the image (see, Fig. 3 for the case of $k = 1$). When only a limited number of changes are performed, the image will not exhibit any visible alteration, i.e., pixels will look homogeneous compared to the surrounding elements (Zuppelli et al. 2021). 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 icon under investigation. Basically, the first block of the proposed neural network is devoted to yielding a flat representation of the image by extracting the $k$ least significant bits of each color channel composing each pixel. Such a representation is then propagated to the subsequent layers to detect and classify different malicious contents. The DNN allows for extracting high-level discriminative features to be further combined for producing the final classification.

## 3.2 Neural architecture

To mitigate the impact of threats taking advantage of information hiding, we designed a supervised neural architecture for the classification task of image icons. In more detail, we exploited the deep architecture shown in Fig. 4, which

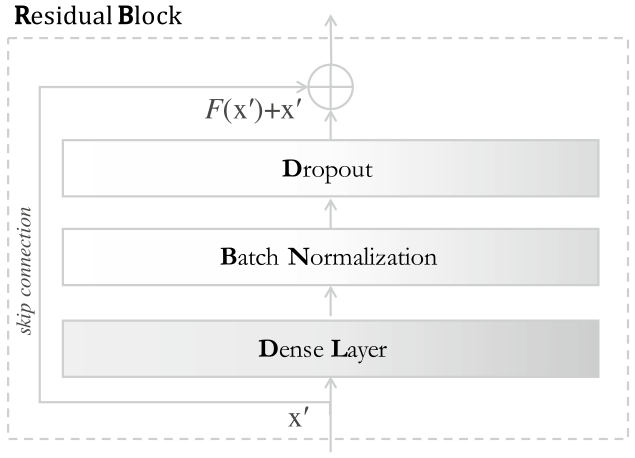


**Fig. 5** Residual block subnet architecture

permits to produce reliable predictions. Essentially, our neural architecture is composed of a stack of several blocks. The first layer acts as a handler for the input provided to the network (denoted as `Input Handler`, in the figure) 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 the figure) 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 (Nair and Hinton 2010), is instantiated, *(ii)* then, a batch-normalization layer is stacked to the previous one to improve the stability of the learning phase and to boost the performances of the model, and *(iii)* a dropout layer is finally added to mitigate the risk of overfitting (Hinton et al. 2014).

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 w.r.t. the current batch (by considering the average $\mu$ and the variance $\sigma$ of each input). A reset of a random number of neurons in the training phase is performed via a dropout mechanism. As pinpointed in Hinton et al. (2012), 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. Essentially, residual blocks (He et al. 2016) differ from regular ones by the mere addition (through the *skip connection*) of an identity function to their output. Formally, the output of a residual block is defined as *activation* $(F(\mathbf{x}') + \mathbf{x}')$, where $F$ denotes the function that the residual block is learning to transform the input $\mathbf{x}'$ of the block itself. A more detailed view of this subnet is sketched in Fig. 5. The usage of skip connections induces in the base DNN classifier a behavior similar to *Residual Networks* (He et al. 2016), 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 $\mathbf{BB_2}$ and $\mathbf{BB_4}$ 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 (Guarascio et al. 2018). The proposed neural model is trained against a set $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_D, \mathbf{y}_D)\}$, where $\mathbf{x}_i$ is the matrix representation of the image and $\mathbf{y}$ is the class of the image. As regards the output, a one-hot encoding based on $C$ classes is used to model the different labels, each one indicating a specific malicious payload. As will be detailed later, in our work we considered $C$ classes representing "clean" image icons and image icons 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 cross-entropy* is adopted for the classification task and it is calculated as follows:

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

## 3.3 Ensembling via soup models

As discussed in Sect. 2 (see Algorithm 1), the proposed federated approach uses a soup model mechanism to merge the contribution of each model produced by the $K$ peer nodes (Wortsman et al. 2022). This ensemble approach is inspired by the work described in Neyshabur et al. (2020), which demonstrated that models independently fine-tuned from the same base model fall within the same loss landscape basin. The authors also suggested that interpolating two solutions (e.g., combining DNN weights) may yield a result that falls closer to the basin's center. The main benefit of the Soup Model is hence to extract robust and reliable classification models by simply averaging the weights without requiring additional memory or inference time. In our framework, the idea consists in using this ensemble method to average (per layer) the DNN weights yielded by the peer models. Specifically, this strategy is known as Uniform Soup. Although in the literature other Soup strategies have been proposed to combine different models (e.g., Greedy Soups and Learned Soup (Wortsman et al. 2022)), we adopted the Uniform Soup to make the proposed approach as much as possible lightweight.

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

## 4 Experimental results

In this section, we first present the dataset modeling realistic images used in mobile applications containing steganographic threats as well as the parameters and metrics adopted in our trials. Then, we will discuss numerical results.

### 4.1 Dataset and parameters

Our federated approach is evaluated on the "Stego-Images-Dataset"[1] described in Cassavia et al. (2022). It contains 48, 000 icons of $512 \times 512$ pixels hiding different realistic malicious payloads, i.e., JavaScript, HTML, PowerShell, URLs, and Ethereum addresses, embedded via the LSB steganography technique. The payloads allow for modeling a wide range of threats, such as malicious scripts and routines, links to additional configuration files or lists of commands, and wallets collecting the outcome of crypto jacking 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, respectively. The training set is further divided among the server and the end nodes composing our architecture. In more detail, the 25% of the training 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 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.

We implemented the proposed model using the PyTorch framework (Paszke et al. 2019). Basically, it consists of 4 `Building Block BB` in which the fully-connected dense layers, stacked on the top of each of them, include 64 neurons, except for the first `BB` that is instantiated with 128 neurons. The dropout rate is set to 0.1. The `Output Layer` includes 6 neurons. The model has been trained over 35 epochs with a batch size of 256. The best model has been chosen according to the F1-Score. We decide to use 5 clients representing the end nodes and 1 server. For the initialization stage, the server model is trained over 10 epochs, and the best model that maximizes the F1-Score on the whole validation set is selected. Each client model is trained over 6 epochs, and the best model is chosen again according to the F1-Score. The number of iterations is set to 10. The AdamW (Loshchilov and Hutter 2019) with a learning rate equal to 0.0001 is adopted as the optimizer.

### 4.2 Evaluation metrics

To evaluate our approach, we relied upon the following metrics[2]:

- *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 $\frac{\text{TP}}{\text{TP+FP}}$, whereas the recall is calculated as $\frac{\text{TP}}{\text{TP+FN}}$;
- *Area Under the Curve (AUC)*: it is the area under the Receiver Operating Characteristic (ROC) 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 recall for different class probability values.

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

[2] *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.

## 4.3 Numerical results

The first round of tests aims at evaluating the effectiveness of our federated-learning-based approach in detecting malicious payloads hidden within images. Tables 1, 2 and 3 summarize the obtained results and show how the performances of the end nodes (i.e., peers) improve over 10 iterations of the algorithm for plain text, Base64-encoded and zip-encoded test sets, respectively. As reported in Table 1, for the plain text, the average AUC of the peers improves from 94.3% in the 1-st iteration to a maximum value of 96.5% when the *max_iter*, i.e., the number of iterations defined in Algorithm 1, is reached. Also, the 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. Analogous results can be observed for the zip compression (Table 3), where the average AUC of the end nodes improves from 80.8% in the 1-st iteration to 83.5% in the 3-rd iteration. Moreover, a better result can be observed on the server performances where the AUC and the AUC-PR improve in the last round from 76.9% to 85.6% and from 40.3% to 49.8%, respectively. With regard to the results using the Base64 test set (Table 2), the improvement on the metrics for the peers is slightly less limited (e.g., the average

**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** |

Plain text test set. Best values are reported in bold

**Table 2** Performance of the federated approach

| Iteration | Model | AUC | AUC-PR | F1-Score |
|---|---|---|---|---|
| Initialization | *server* | 0.855 | 0.536 | 0.533 |
| 1 | $peer_{avg}$ | 0.895 | 0.599 | 0.571 |
| | *server* | 0.855 | 0.536 | 0.533 |
| 2 | $peer_{avg}$ | 0.896 | 0.603 | 0.588 |
| | *server* | **0.925** | **0.642** | **0.578** |
| 3 | $peer_{avg}$ | 0.906 | 0.613 | 0.587 |
| | *server* | 0.925 | 0.642 | 0.578 |
| 4 | $peer_{avg}$ | 0.895 | 0.602 | 0.582 |
| | *server* | 0.925 | 0.642 | 0.578 |
| 5 | $peer_{avg}$ | 0.908 | 0.611 | 0.599 |
| | *server* | 0.925 | 0.642 | 0.578 |
| 6 | $peer_{avg}$ | 0.885 | 0.592 | 0.589 |
| | *server* | 0.925 | 0.642 | 0.578 |
| 7 | $peer_{avg}$ | 0.895 | 0.606 | 0.585 |
| | *server* | 0.915 | 0.631 | 0.485 |
| 8 | $peer_{avg}$ | 0.889 | 0.606 | 0.589 |
| | *server* | 0.910 | 0.645 | 0.472 |
| 9 | $peer_{avg}$ | 0.906 | 0.618 | 0.589 |
| | *server* | 0.910 | 0.645 | 0.472 |
| 10 | $peer_{avg}$ | 0.898 | 0.613 | 0.597 |
| | *server* | **0.893** | **0.594** | **0.614** |

Base64 test set. Best values are reported in bold

**Table 3** Performance of the federated approach

| Iteration | Model | AUC | AUC-PR | F1-Score |
|---|---|---|---|---|
| Initialization | *server* | 0.769 | 0.403 | 0.349 |
| 1 | $peer_{avg}$ | 0.808 | 0.443 | 0.356 |
| | *server* | 0.769 | 0.403 | 0.349 |
| 2 | $peer_{avg}$ | 0.807 | 0.436 | 0.352 |
| | *server* | **0.826** | **0.457** | **0.354** |
| 3 | $peer_{avg}$ | 0.835 | 0.477 | 0.391 |
| | *server* | 0.826 | 0.457 | 0.354 |
| 4 | $peer_{avg}$ | 0.800 | 0.430 | 0.335 |
| | *server* | 0.826 | 0.457 | 0.354 |
| 5 | $peer_{avg}$ | 0.833 | 0.475 | 0.391 |
| | *server* | 0.826 | 0.457 | 0.354 |
| 6 | $peer_{avg}$ | 0.785 | 0.417 | 0.328 |
| | *server* | 0.826 | 0.457 | 0.354 |
| 7 | $peer_{avg}$ | 0.798 | 0.424 | 0.350 |
| | *server* | **0.867** | **0.500** | 0.304 |
| 8 | $peer_{avg}$ | 0.783 | 0.410 | 0.337 |
| | *server* | 0.843 | 0.471 | 0.344 |
| 9 | $peer_{avg}$ | 0.801 | 0.454 | 0.396 |
| | *server* | 0.843 | 0.471 | 0.344 |
| 10 | $peer_{avg}$ | 0.799 | 0.421 | 0.331 |
| | *server* | **0.856** | **0.498** | **0.363** |

Zip test set. Best values are reported in bold

AUC improves from 89.5% for the 1-st iteration to 90.8% in the 5-th iteration), whereas we can observe better results for the server performances. Specifically, the average AUC and AUC-PR improve up to 89.3% and 59.4%, respectively.

We also quantify the prediction capabilities of the model showing its performance w.r.t. each class. To this end, we plotted the ROC curves for each test set. Figure 6 shows the ROC curves for all classes when the malicious payload is embedded in plain text with no additional encoding. In this case, although the classifier gets good results, we can observe some misclassification for URL/IP addresses and Ethereum addresses with AUC equal to 0.89 in both cases. This could be because both Ethereum addresses and URLs are composed of few and similar alphanumeric characters, making the classification more difficult. The ROC curves for Base64-encoded test set are depicted in Fig. 7, where we can observe a small degradation of the performances for each class except for the legitimate images. Like the plain text test, there is again a similar misclassification for URL/IP and Ethereum addresses classes with AUC equal to 0.88 and 0.89, respectively. Moreover, the PowerShell class tends to be misclassified with JavaScript probably due to the presence of similar statements like `if-then-else`, `for`, and type declarations. Finally, in Fig. 8 the ROC curves for the zip compression are shown. In this case, the system is still able to distinguish between compromised and legitimate images, but different payloads are misclassified from each other, e.g., the AUC for PowerShell is equal to 0.70. An explanation for this behavior could be that zip compression reduces the differences between different payloads as well as introduces new metadata, which may be similar for all classes.

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 4 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 w.r.t. the centralized approach.

Summing up, the above results demonstrate that federated learning can be effectively used to reveal the presence of concealed contents while guaranteeing privacy and scalability constraints. In more detail, the federated solution achieves comparable performances with a fully centralized method without the necessity of moving data toward a single node. In this way, our approach can also be used in resource-constrained scenarios, e.g., IoT ecosystems.

**Table 4** Comparison of centralized and federated approaches with 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** |

Best values are reported in bold

## 5 Related works

In recent years, the impact of threat actors taking advantage of information hiding techniques and steganography to make their attack chains more complex and stealthier has also been extended to the Web and its services. For instance, cloaking techniques can be used to juxtapose an additional layer for managing contents of the "hidden Web", i.e., a portion of the Web only accessible via suitable search interfaces via known keywords (Ntoulas et al. 2005). At the same time, the surge of social media services offers a fertile playground for a variety of scams and malicious activities that can be cloaked in the bulk of data. Among the others, notable examples are the use of text-based steganography for abusing contents of online social networks (e.g., Twitter) to orchestrate bots and implement command & control communications (Gurunath et al. 2021). Another relevant scenario is the exploitation of images and metadata used in Facebook to enrich the overall user experience (Hiney et al. 2015). Owing to its double-edged nature, hiding mechanisms can also be considered effective tools to enforce copyright or track Web resources. For example, XML contents can be "marked" by embedding empty elements or patterns of white spaces in tags (Inoue et al. 2001).

Unfortunately, both the Web and ad-hoc distribution channels (e.g., application stores or software repositories) are characterized by an almost infinite set of digital assets that can be exploited for cloaking data. In particular, executables, such as `.exe`, `.apk`, and `.ipa`, can be retrieved almost everywhere on the Internet, thus making it impossible to outline a precise attack surface. Data can then be cloaked in executable code through the manipulation of specific redundancies, such as the injection of uncommon instructions or the alteration of the relative frequency of jumps (see, e.g., Anckaert et al. (2005) for IA-32 code).

For the specific case of enforcing the security of mobile applications, the most popular frameworks typically rely upon a variety of techniques (e.g., static binary analysis, anomaly-based detection, and definition of access control

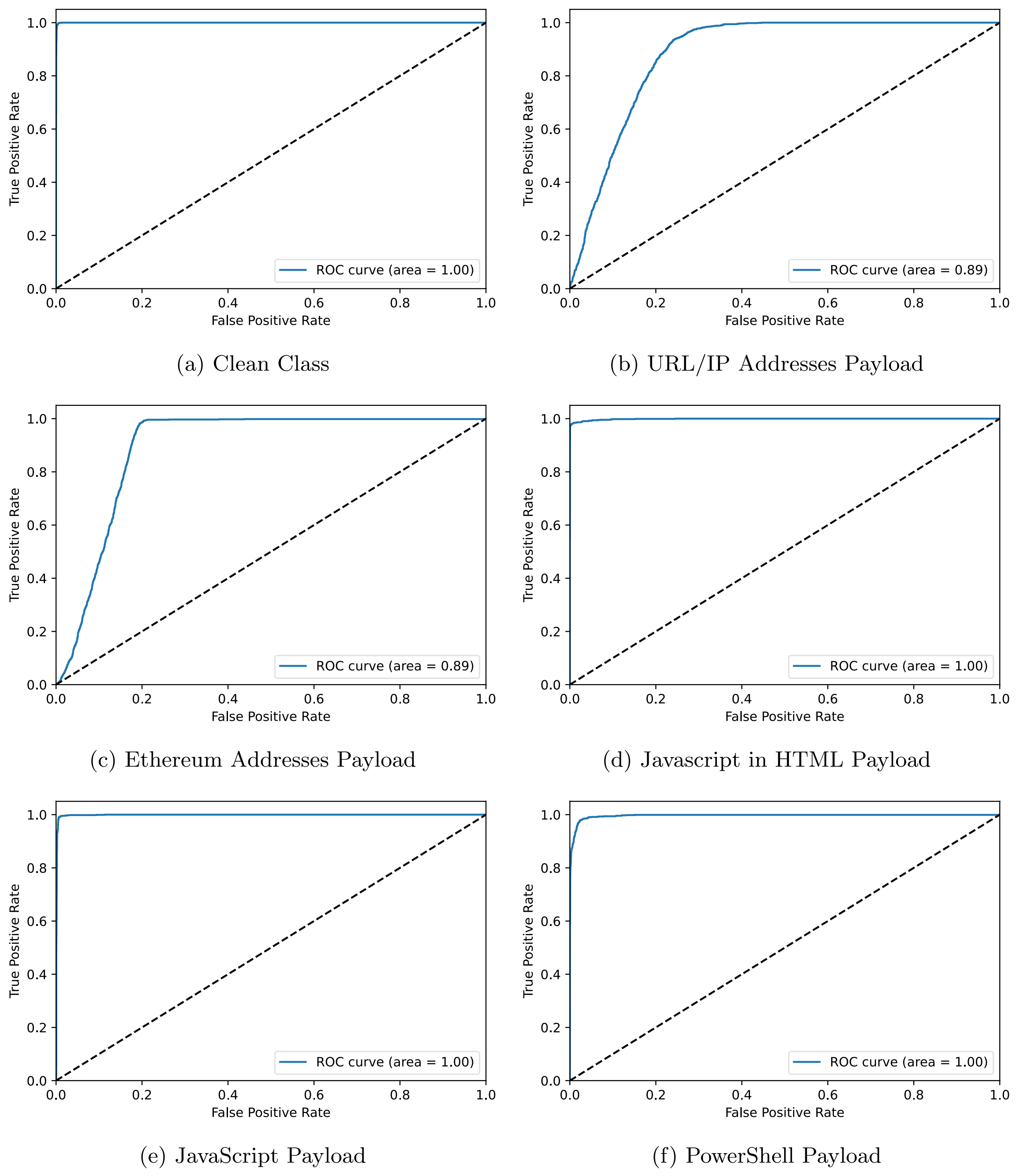
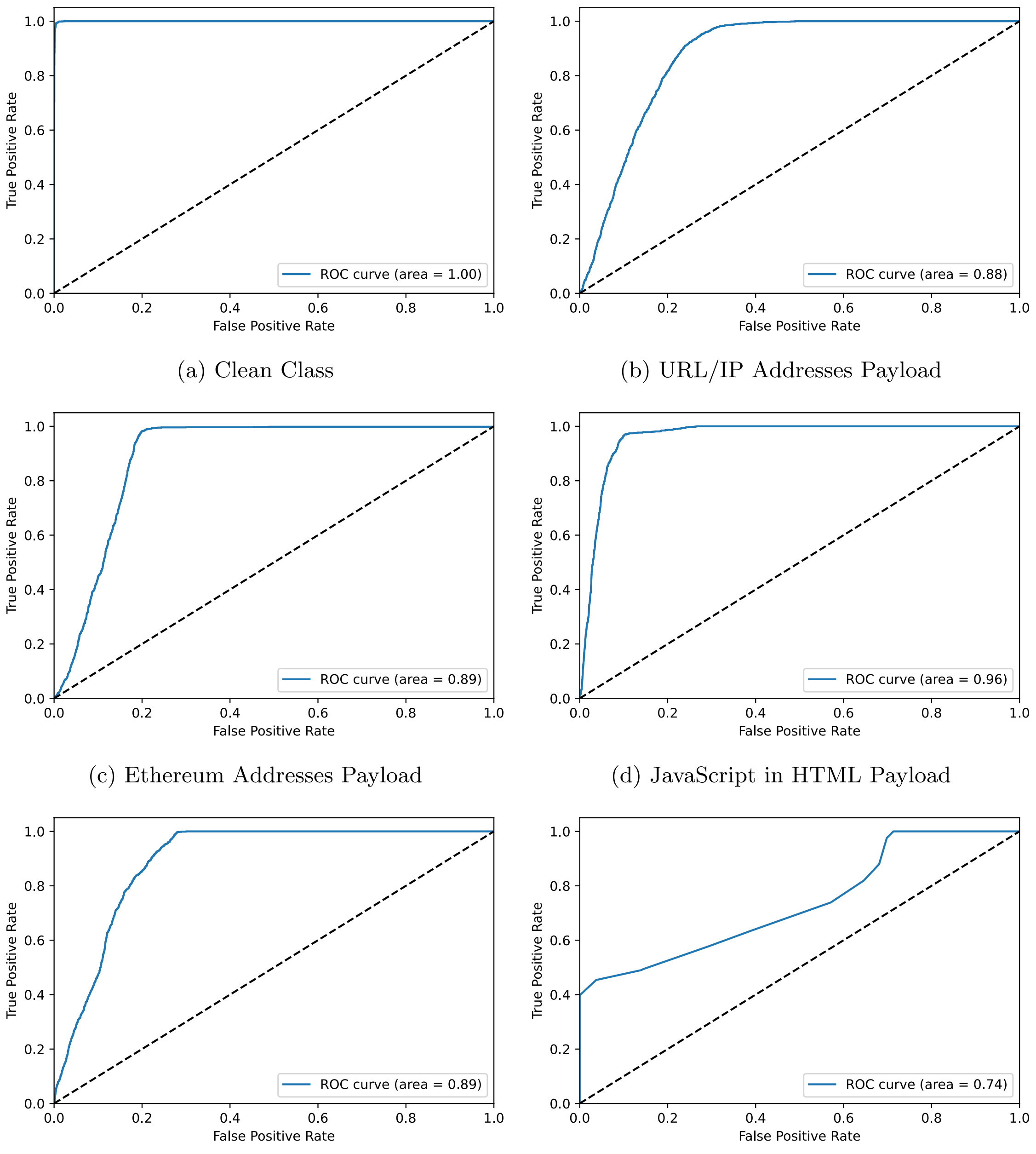


**Fig. 6** All classes ROC curves, plain text test set

policies in end nodes). Alas, this requires a strict cooperation among developers, users, and administrators of application stores (He et al. 2015). In general, the current trend is to exploit ML or AI to analyze behaviors of software, mainly to search for existent attack signatures or unexpected interactions among software components.[3] Such tools have proven to be effective also to mitigate attacks based on information hiding targeting digital media (Monika and Eswari 2022; Cassavia et al. 2022). Unfortunately, the deployment of AI-capable techniques might clash with practical constraints. First, the inspection of bundled assets and copyrighted material should respect privacy-enforcing regulations requiring architectures to not process any personal information (Pawlicka et al. 2020). Second, the continuous growth of mobile ecosystems is leading to millions of samples to verify. Applications can also be made available via different store replicas

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



(a) Clean Class (b) URL/IP Addresses Payload

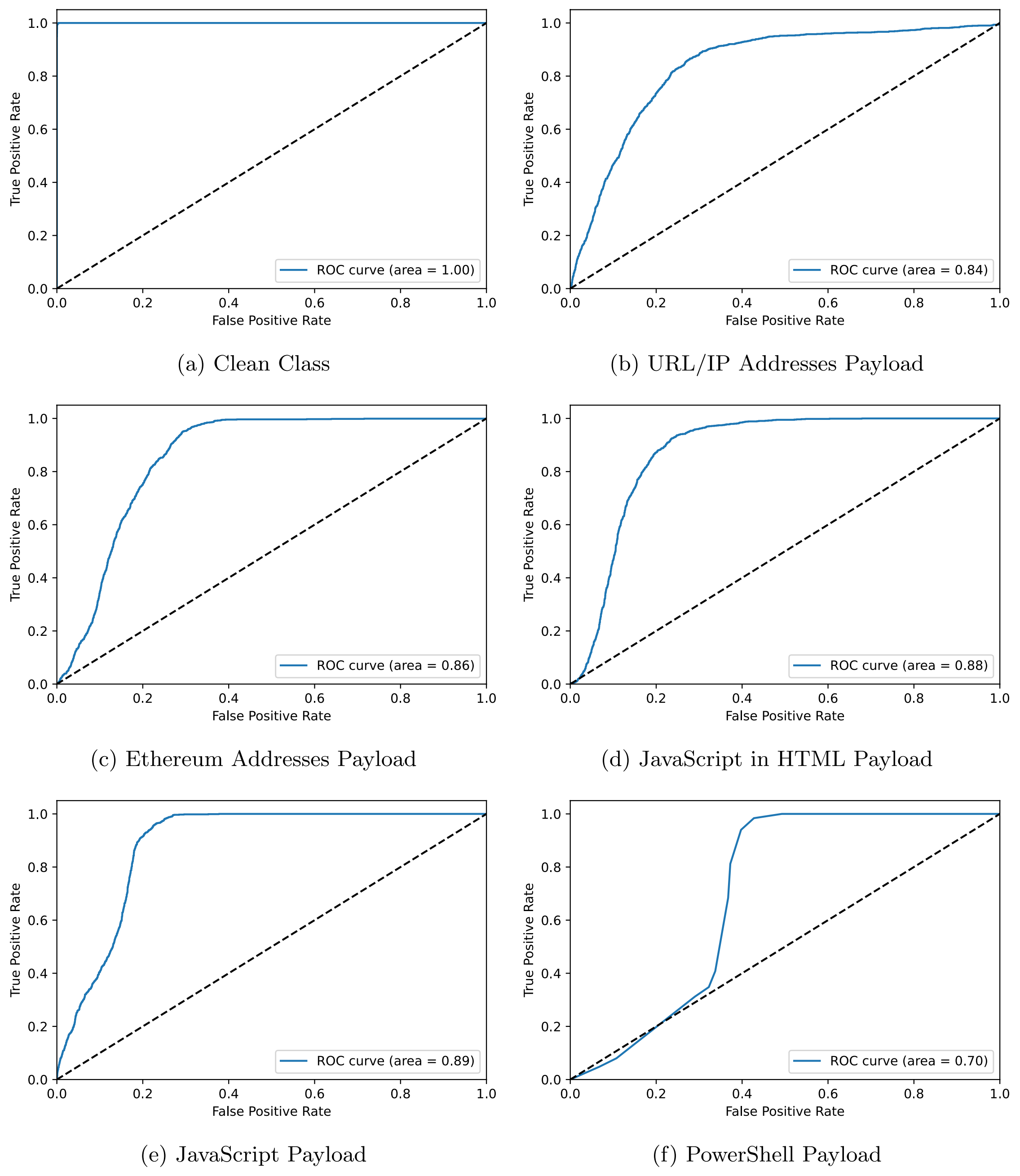
(c) Ethereum Addresses Payload (d) JavaScript in HTML Payload

(e) JavaScript Payload (f) PowerShell Payload

**Fig. 7** All classes ROC curves, Base64 test set

for performance purposes, through unofficial channels (e.g., alternative stores (Guarascio et al. 2018) or via sideloading (Li et al. 2017)) as well as in Web or ad-hoc social media channels, thus rendering the creation of comprehensive datasets a hard task (Wang et al. 2019). To partially cope with challenges to improve the security of mobile ecosystems, federated approaches are becoming a precious tool (Rahman et al. 2020).

As regards the goal of spotting contents cloaked in digital images via distributed or cloud-native frameworks, Yang et al. (2020) exploits federated transfer learning to improve the performance of image steganalysis tasks while preserving the privacy of users. Even if this work has a similar goal, there are some major differences with our idea. First, it considers (user) end nodes instead of app-stores and does not focus on real malware samples. As a consequence, authors investigated the performance when in the presence of advanced steganographic methods acting on the spatial domain (i.e., WOW, S-UNIWARD, and HILL), which have never been observed in real attacks due to their complexity



(a) Clean Class (b) URL/IP Addresses Payload

(c) Ethereum Addresses Payload (d) JavaScript in HTML Payload

(e) JavaScript Payload (f) PowerShell Payload

**Fig. 8** All classes ROC curves, zip test set

(Mazurczyk and Caviglione 2015; Caviglione and Mazurczyk 2022). Despite the work concentrates on digital images similar to those used for the creation of Android/iOS icons (i.e., cropped pictures of $512 \times 512$ pixels), experiments only bear with greyscale images, which are seldom used in modern mobile applications. Rather, greyscale or B &W images are used for UI widgets, but their steganographic capacity could be very limited and the detection of massive tampering of assets bundled within a mobile application could be effectively done without the need for AI (see, e.g., Faruki et al. (2013)).

Considering different use cases, the literature offers various works dealing with the adoption of federated learning to tame realistic threats. As an example, Jiang et al. (2022) shows 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 Lin and Huang (2020), but it focuses on a generic scenario not related to the security of mobile applications. Instead, Shamili et al. (2010) concentrates on the problem of detecting malware but for an OS no longer used (i.e., Symbian S60).

To face the multifaceted cybersecurity challenges of modern deployments, a possible "meet in the middle" blueprint should offload end nodes toward edge entities placed at the border of the network, and cooperating stores could be adopted to implement such an architecture. To this extent, 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 (Tian et al. 2021). Besides, for the specific case of mobile security, edge/federated approaches have been mainly adopted to guarantee privacy constraints. A notable exception is Hsu et al. (2020), which demonstrates how to detect malware without exposing sensitive information of end users, such as configuration details or how various application program interfaces are invoked.

## 6 Conclusions and future works

In this paper, we have presented a federated framework for the detection of malicious assets cloaked in icon images bundled or repackaged within applications. Our approach demonstrated its effectiveness in handling applications made available through multiple (un)official stores or directly from Web and social media. The federated framework also showcased good performances with threat actors trying to avoid detection via elusive schemes, e.g., when the secret data is encoded in Base64 or compressed with the zip algorithm to have an "obfuscating envelope".

As shown, the federated blueprint should be considered of particular value to enforce the security of scenarios where applications could also be distributed outside classic pipelines. For instance, this is the case of software made directly available in public repositories, social media, or Web pages. A federated scheme can prevent constraints and bottlenecks characterizing single-point architectures, e.g., scalability issues and lack of comprehensive snapshots for training the models. Even if implementing such a vision is almost straightforward for single-vendor deployments, federating stores owned by different entities could be unfeasible, or require additional engineering. Another limitation concerns the ability of a threat actor running a malicious repository to join the federation and inject incorrect information to improve its undetectability. Lastly, crawling large sources to gather the required data could be time-consuming or difficult. As an example, some websites prevent scraping, and many social media services limit the rate at which information can be requested.

Therefore, future works aim at removing the aforementioned weaknesses. For instance, a suitable communication mechanism (e.g., a specific set of protocols endowed with security guarantees) could promote cooperation among various stores while mitigating the risk of attacks. Another relevant aim of our future research is devoted to extending the proposed approach to detect other types of steganographic threats, especially malicious information cloaked in network traffic. In more detail, we are interested in evaluating if the benefits of the federated approach can also be leveraged to monitor large-scale networks or microservice-based architectures. As an example, traffic can be collected in multiple points placed at the border of a network/datacenter so as to enforce scalability properties and avoid the necessity of moving sensitive/confidential data.

**Acknowledgements** This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU-NGEU.

**Funding** Open access funding provided by Università della Calabria within the CRUI-CARE Agreement.

**Data availability** Data used in this work are available at link https://www.kaggle.com/datasets/marcozuppelli/stegoimagesdataset

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

## References

Almaiah MA, Al-Zahrani A, Almomani O, Alhwaitat AK (2021) Classification of cyber security threats on mobile devices and applications, pp 107–123. Springer, Cham

Anckaert B, De Sutter B, Chanet D, De Bosschere K (2005) Steganography for executables and code transformation signatures. In: Information security and cryptology–ICISC 2004: 7th international conference, Seoul, Korea, December 2–3, 2004, Revised Selected Papers 7, pp 425–439. Springer

Cassavia N, Caviglione L, Guarascio M, Manco G, Zuppelli M (2022) Detection of steganographic threats targeting digital images in heterogeneous ecosystems through machine learning. J Wireless Mobile Netw Ubiquit Comput Depend Appl 13:50–67

Cassavia N, Caviglione L, Guarascio M, Liguori A, Surace G., Zuppelli, M (2023) Federated learning for the efficient detection of steganographic threats hidden in image icons. In: Pervasive knowledge and collective intelligence on web and social media, pp 83–95. Springer, Cham

Caviglione L, Mazurczyk W (2022) Never mind the malware, here's the stegomalware. IEEE Security Privacy 20(5):101–106

Cheddad A, Condell J, Curran K, Mc Kevitt P (2010) Digital image steganography: survey and analysis of current methods. Signal Process 90(3):727–752

Faruki P, Ganmoor V, Laxmi V, Gaur MS, Bharmal A (2013) AndroSimilar: robust statistical feature signature for Android malware detection. In: Proceedings of the 6th international conference on security of information and networks, pp 152–159

Gibert D, Mateu C, Planes J (2020) The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J Netw Comput Appl 153:102526

Guarascio M, Manco G, Ritacco E (2018) Deep learning. Encyclopedia of Bioinf Comput Biol ABC Bioinf 1–3:634–647

Guarascio M, Ritacco E, Biondo D, Mammoliti R, Toma A (2018) Integrating a framework for discovering alternative app stores in a mobile app monitoring platform. In: New frontiers in mining complex patterns, pp 107–121. Springer, Cham

Guarascio M, Zuppelli M, Cassavia N, Caviglione L, Manco G (2022) Revealing MageCart-like threats in favicons via artificial intelligence. In: Proceedings of the 17th international conference on availability, reliability and security, pp 1–7

Gurunath R, Klaib MFJ, Samanta D, Khan MZ (2021) Social media and steganography: use, risks and current status. IEEE Access 9:153656–153665

He D, Chan S, Guizani M (2015) Mobile application security: malware threats and defenses. IEEE Wireless Commun 22(1):138–144

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 770–778

Hiney J, Dakve T, Szczypiorski K, Gaj K (2015) Using facebook for image steganography. In: 2015 10th international conference on availability, reliability and security, pp 442–447. IEEE

Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580

Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958

Hsu R-H, Wang Y-C, Fan C-I, Sun B, Ban T, Takahashi T, Wu T-W, Kao S-W (2020) 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

Inoue S, Makino K, Murase I, Takizawa O, Matsumoto T, Nakagawa H (2001) A proposal on information hiding methods using XML. In: The 1st workshop on NLP and XML, pp 707–710

Jiang C, Yin K, Xia C, Huang W (2022) Fedhgcdroid: an adaptive multi-dimensional federated learning for privacy-preserving Android malware classification. Entropy 24(7):919

Li L, Li D, Bissyandé TF, Klein J, Le Traon Y, Lo D, Cavallaro L (2017) Understanding Android app piggybacking: a systematic study of malicious code grafting. IEEE Trans Inf Forensics Security 12(6):1269–1284

Lin K-Y, Huang W-R (2020) Using federated learning on malware classification. In: 2020 22nd International conference on advanced communication technology (ICACT), pp 585–589. IEEE

Loshchilov I, Hutter, F (2019) Decoupled weight decay regularization. In: International conference on learning representations

Mazurczyk W, Caviglione L (2015) Information hiding as a challenge for malware detection. IEEE Security Privacy 13(2):89–93

Mazurczyk W, Caviglione L (2021) Cyber reconnaissance techniques. Commun ACM 64(3):86–95

Monika A, Eswari R (2022) Prevention of hidden information security attacks by neutralizing stego-malware. Comput Electrical Eng 101:107990

Mylonas A, Kastania A, Gritzalis D (2013) Delegate the smartphone user? Security awareness in smartphone platforms. Comput Security 34:47–66

Nair V, Hinton GE (2010) 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

Neyshabur B, Sedghi H, Zhang C (2020) What is being transferred in transfer learning? In: Adv Neural Inf Process Syst 33:512–523

Ntoulas A, Zerfos P, Cho J (2005) Downloading textual hidden web content through keyword queries. In: Proceedings of the 5th ACM/IEEE-CS joint conference on digital libraries, pp 100–109

Papageorgiou A, Strigkos M, Politou E, Alepis E, Solanas A, Patsakis C (2018) Security and privacy analysis of mobile health applications: the alarming state of practice. IEEE Access 6:9390–9403

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: an imperative style. High-Performance Deep Learning Library. Curran Associates Inc., Red Hook, NY, USA

Pawlicka A, Jaroszewska-Choras D, Choras M, Pawlicki M (2020) Guidelines for stego/malware detection tools: achieving GDPR compliance. IEEE Technol Soc Mag 39(4):60–70

Poeplau S, Fratantonio Y, Bianchi A, Kruegel C, Vigna G (2014) Execute this! analyzing unsafe and malicious dynamic code loading in android applications. NDSS 14:23–26

Rahman SA, Tout H, Talhi C, Mourad A (2020) Internet of Things intrusion detection: Centralized, on-device, or federated learning? IEEE Network 34(6):310–317

Shamili AS, Bauckhage C, Alpcan T (2010) Malware detection on mobile devices using distributed machine learning. In: 20th international conference on pattern recognition, pp 4348–4351. IEEE

Spreitzenbarth M, Freiling F, Echtler F, Schreck T, Hoffmann J (2013) Mobile-sandbox: having a deeper look into android applications. In: Proceedings of the 28th annual ACM symposium on applied computing, pp 1808–1815

Suarez-Tangil G, Tapiador JE, Peris-Lopez P (2014) Stegomalware: Playing hide and seek with malicious components in smartphone apps. In: Proceedings of the 10th international conference on information security and cryptology (ICISC), Beijing, China, vol 8957, pp 496–515. Springer

Tian P, Chen Z, Yu W, Liao W (2021) Towards asynchronous federated learning based threat detection: a DC-Adam approach. Comput Security 108:102344

Wang H, Li H, Guo Y (2019) Understanding the evolution of mobile app ecosystems: a longitudinal measurement study of Google Play. In: The World Wide Web conference, pp 1988–1999

Wortsman M, Ilharco G, Gadre SY, Roelofs R, Gontijo-Lopes R, Morcos AS, Namkoong H, Farhadi A, Carmon Y, Kornblith S, et al. (2022) Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In: International conference on machine learning, pp 23965–23998. PMLR

Yang H, He H, Zhang W, Cao X (2020) Fedsteg: a federated transfer learning framework for secure image steganalysis. IEEE Trans Netw Sci Eng 8(2):1084–1094

Yuan Z, Lu Y, Xue Y (2016) Droiddetector: android malware characterization and detection using deep learning. Tsinghua Sci Technol 21(1):114–123

Zhou W, Zhou Y, Jiang X, Ning P (2012) 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

Zuppelli M, Manco G, Caviglione L, Guarascio M (2021) Sanitization of images containing stegomalware via machine learning approaches. In: Proceedings of the Italian conference on cybersecurity (ITASEC), Online, vol 2940, pp 374–386