Detection of Steganographic Threats Targeting Digital Images in Heterogeneous Ecosystems Through Machine Learning

Nunziato Cassavia[1], Luca Caviglione[2], Massimo Guarascio[1]*, Giuseppe Manco[1], and Marco Zuppelli[2]

[1]Institute for High Performance Computing and Networking (ICAR),
National Research Council of Italy (CNR), Rende, Italy
{nunziato.cassavia, massimo.guarascio, giuseppe.manco}@icar.cnr.it

[2]Institute for Applied Mathematics and Information Technologies (IMATI),
National Research Council of Italy (CNR), Genova, Italy
{luca.caviglione, marco.zuppelli}@ge.imati.cnr.it

Received: June 01, 2022; Accepted: August 22, 2022; Published: September 30, 2022

Abstract

Steganography is increasingly exploited by malware to avoid detection and to implement different advanced offensive schemes. An attack paradigm expected to become widely used in the near future concerns cloaking data in innocent-looking pictures, which are normally used by several devices and applications, for instance to enhance the user experience. Therefore, with the increasing popularity of application stores, availability of cross-platform services, and the adoption of various devices for entertainment and business duties, the chances for hiding payloads in digital pictures multiply in an almost unbounded manner. To face such a new challenge, this paper presents an ecosystem exploiting a classifier based on Deep Neural Networks to reveal the presence of images embedding malicious assets. Collected results indicated the effectiveness of the approach to detect malicious contents, even in the presence of an attacker trying to elude our framework via basic obfuscation techniques (i.e., zip compression) or the use of alternative encoding schemes (i.e., Base64). Specifically, the achieved accuracy is always $\sim$100% with minor decays in terms of precision and recall caused by the presence of additional information caused by compression.

Keywords: image steganography, machine learning, deep neural networks, stegomalware

1 Introduction

The complexity of modern hardware and software ecosystems jointly with the pervasive nature of the Internet are increasingly exploited to develop effective malware [1]. For instance, the use of virtualization, the ubiquitous adoption of Internet of Things (IoT) technologies, and the deployment of services via elaborate interconnections of heterogeneous vendors, lead to an attack surface difficult to control and protect. As a consequence, Advanced Persistent Threat (APT) actors can now take advantage of several 0-day exploits, implement multi-stage pipelines, and elude detection by concealing their presence in the burden of data. In this scenario, a recent trend concerns the use of steganography to make malware stealthier, for instance to evade classical detection techniques based on signatures [2]. Such techniques are now commonly observed in many large-scale attack campaigns or at the basis of several APTs. As

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 13(3):50-67, Sept. 2022
DOI:*10.22667/JOWUA.2022.09.30.050*

*Corresponding author: Institute for High Performance Computing and Networking (ICAR), Via P. Bucci, cubo 8/9C, 87036 Rende (CS), Italy, Tel: +39 0984 493859, Web: `https://www.icar.cnr.it/persone/guarascio/`

possible examples, hiding mechanisms are employed for cloaking configuration files (e.g., ZeusVM), conceal payloads in the `PropertyTagICCProfile` metadata of images (e.g., SteamHide), or to drop credit card skimmers in e-commerce platforms (e.g., Magento) [1, 3]. Although the offensive exploitation of steganography may vary, for instance it can be used to create stealthy Command & Control (C&C) infrastructures or to deliver additional assets to a generic attack stage, digital images are definitely the most abused carriers [4].

In parallel, the boundaries between mobile and desktop applications are progressively blurring. For instance, iOS/iPadOS Apps can run over macOS by using a transparent and lean emulation layer, and Android OS software can be deployed in set top boxes, mobile devices, and various appliances with minimal modifications. Moreover, the diffusion of the Software-as-a-Service paradigm leads to a common back-end, which can be shared by different users and devices [5]. As a consequence, malware and threat actors can target a limitless population of devices often sharing the underlying hardware architecture or a non-negligible amount of software assets, such as libraries or network protocols. Even if the uncontrolled diffusion of malicious software characterized by a "write once, attack anywhere" nature could be not an imminent danger, the use of digital images should be considered an important feature unifying the majority of nodes connected to the Internet. We point out that, evidences of the use of image steganography within commercially-available software can be rooted back in 2014, e.g., see [6] for an analysis of applications available on the Google Play Store containing images altered via steganographic mechanisms.

In this perspective, evaluating security issues arising from the use of steganography to drop malicious payloads or to deliver additional assets for feeding complex attacks is mandatory to fully assess the security of future digital ecosystems. Unfortunately, developing general and effective mitigation techniques is a hard task, as steganographic attacks are mainly threat- and carrier-dependent [7]. At the same time, the massification of mobile devices, the large-scale deployment of IoT nodes and smart frameworks, as well as the fragmentation of various software sources (e.g., ad-hoc and official stores vs side-loaded applications) pose several challenges in the design of a unique framework able to recognize the presence of images containing steganographic contents. To balance such an escalation, artificial intelligence is quickly becoming a key ingredient (see, e.g., [8] and the references therein for some recent examples) even if it requires suitable datasets or can be prone to "concept drifts" due to the evolving nature of attacks [9]. However, recent advancements in deep learning techniques allow to face several engineering and data-intensive problems, especially in the field of image processing. Specifically, deep learning can exploit multiple levels of abstractions and can capture relations between set of features directly from raw and noisy image data. Moreover, no feature engineering or interaction with domain experts are required to build good representative features [10]. Hence, Deep Neural Networks (DNNs) are a good candidate to face the issue of revealing the presence of threats hidden within images, especially if at the basis of a broad and heterogeneous set of attacks.

Therefore, this paper deals with an ecosystem for the detection of steganographic threats targeting digital images in a general and efficient manner. In more detail, we propose a detector based on DNNs able to reveal the presence of malicious payloads, even when the attacker tries to implement basic evasion techniques such as obfuscation through compression or the use of alternative encoding schemes. To sum up, the contribution of this paper is twofold: it introduces a framework using artificial intelligence to detect different malware leveraging steganography, and it showcases the creation of a realistic dataset for modeling various threats and attack scheme.

The rest of the paper is structured as follows. Section 2 reviews past works on threats targeting mobile applications, with emphasis on those exploiting steganography. Section 3 introduces the attack model as well as background information on malware and its mitigation via artificial intelligence techniques, Section 4 presents the proposed ecosystem and the detection approach, and Section 5 showcases numerical results obtained via a thorough performance evaluation campaign. Lastly, Section 6 concludes

the paper and outlines possible future research directions.

2 Related Work

As hinted, the use of steganography to empower a wide array of cyber threats is now a prime issue, thus leading to a corpus of works dealing with detection and mitigation of attacks hiding data in various digital assets. In this context, many works addressed the use of mechanisms to conceal information in network traffic, e.g., to implement abusive communication paths for exfiltrate sensitive bits or to orchestrate nodes of a botnet [11]. Indeed, multimedia data offer a wide selection of attractive carriers, which can be used to cloak sophisticated payloads and not only a single command or an IP address to be contacted, as it happens for simple carriers like HTTP headers. As an example, digital audio can be used to inject malware in mobile devices [12] as it offers the possibility of hiding code and other resources. Yet, the majority of threats observed "in the wild" exploits images, owing to their trade off between capacity and the availability of simple steganographic techniques [1]. Therefore, the creation of tools for revealing the presence of hidden information within digital pictures has been an important research topic in the last decade. Partially borrowing results from general approaches for image steganalysis [6], many works proposed solutions that can be used to counteract the emerging wave of steganographic malware. In more detail, the recent work in [13] presents various techniques leveraging machine learning to create predictive models to discover images potentially altered by a malicious software. Another recent work surveys many steganographic techniques that can be considered at the basis of a new-wave of attacks [14]. Unfortunately, works [13, 14] do not consider real-threats, but focus on the perspective utilization of data hiding to develop novel attack mechanisms. Concerning realistic templates, the work [15] deals with the use of deep neural networks to reveal PowerShell scripts cloaked in digital images. Specifically, it considers an attacker using the Invoke-PSImage technique. The latter, based on a variant of the Least Significant Bit (LSB) steganography exploiting only two color channels, has become one of the most popular approaches observed in real attacks to conceal malicious payloads in images [1]. Regardless a wide range of steganographic techniques using both spatial and transform domains are available [16], attackers tend to always exploit simple LSB-based mechanisms. This choice reduces code complexity and prevents voiding the stealthiness of the attack due to inflated sizes of the infected executable or the need of additional routines/macros for extracting the hidden data. At the same time, basic techniques could be detected owing to naive implementations, e.g., contents hidden via Invoke-PSImage can be spotted by searching for predictable padding patterns [17].

Despite considering real or perspective attack models, a relevant research trend leverages some form of machine learning or artificial intelligence to re balance the arms race between attackers and defenders [1, 13]. As an example, [18] presents how different techniques (e.g., linear discriminant analysis, random forest, and back-propagation networks) can recognize malicious PowerShell scripts, even if not strictly cloaked with steganographic mechanisms. Besides, many works addressed the problem of revealing the presence of network covert channels hidden within legitimate traffic flows [19]. In general, only few works focus on steganographic threats or take into account the detection of attacks in the scenario considered in this paper. For instance, [20] exploits DNNs but only considers favicons, which have been used by threat actors like Magento to hide skimmers within e-commerce platforms. Concerning more general investigations, [21] addresses how to classify malicious JPG files with a large acceptation of what is considered harmful, i.e., various tampering attempts are taken into account. Instead, [22] showcases a more general discussion and demonstrates how different methods can be deployed to reveal the presence of hidden data. Similarly, the work in [23] studies how artificial intelligence can be used to embed legitimate watermarks. Despite the different usage template, some ideas presented in [23, 22] could be borrowed, i.e., the design of threat- or application-specific neural architectures. Due to the

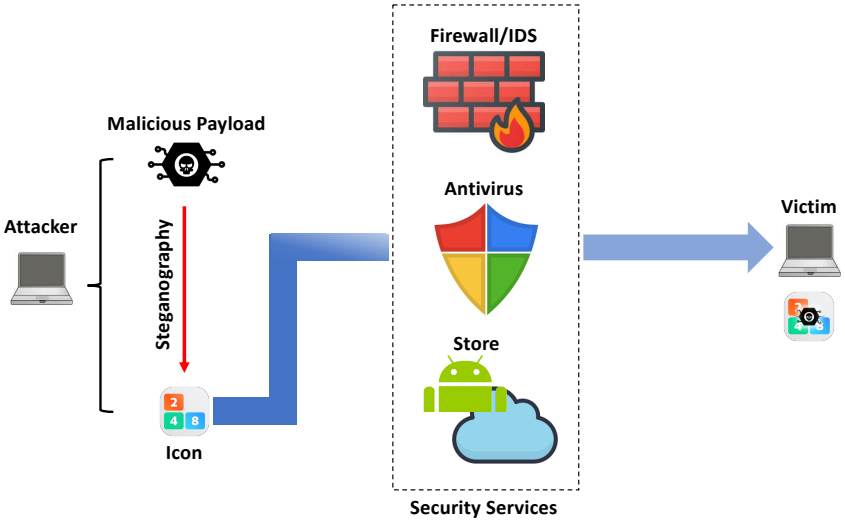


Figure 1: General attack model considered to design the ecosystem for the mitigation of steganographic threats.

scalability constraints or lack of suitable models, deploying sophisticated machine learning frameworks could not be possible. Moreover, inspecting all the images exchanged in a large-scale deployment could be unfeasible due to hardware requirements or the need of satisfying real-time constraints. In this case, an alternative approach searches for well-known signatures characterizing a specific hiding mechanism or a payload (e.g., a sequence of bytes) in the file structure of the image via a simple and optimized tool [24]. Alas, reverse engineering a malware to grasp its internals is often difficult, hence a "meet in the middle" solution is represented by sanitization, i.e., the image is lightly processed to disrupt the secret content, if any. To this aim, the literature offers solutions using nonlinear transformations [25] or autoencoders to alter anomalous pixels without degrading the perceived quality [8]. Another approach is based on the adoption of artificial intelligence to locate the area of the image modified via steganography [26] to trigger the execution of an optimized security pipeline.

3 Attack Model and Background

In this section, we first introduce the attack model. We then provide some background information on the use of machine learning for the processing of digital images and the detection of artifacts.

3.1 General Attack Model Leveraging Image Steganography

The general attack model considered in this work deals with an attacker wanting to cloak a malicious payload into a digital image to bypass a secure perimeter. As an example, a threat actor wants to drop a payload on the host of the victim or build a stealthy chain to reduce the effectiveness of forensics investigations [4]. In general, infiltrating a malicious content or making difficult to locate the source of the attack can be done by using different techniques. For instance, in the case of platforms based on Android OS, malware can be repackaged within applications to evade detection algorithms, even exploiting machine learning [27]. Instead, when considering simpler targets like IoT nodes, malicious routines can be directly injected or obfuscated in binaries [28].

Once the payload is hidden in the image, the malware has to be distributed to the victim. As today, different attack vectors exist, but the most popular are [1]: *i*) the payload is sent as an email attachment,

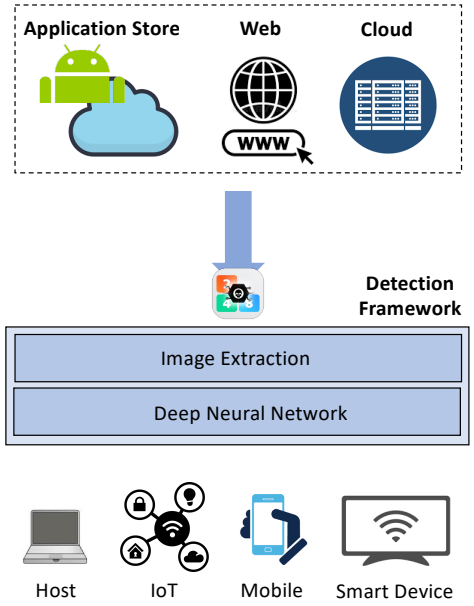


Figure 2: Layered architecture of our approach for detecting malware targeting malicious images.

for instance via phishing campaigns; *ii*) the target is decoyed to retrieve some software, for instance by clicking a malicious link or tricked with social engineering techniques; *iii*) a third-party malicious stage already running on the device of the victim retrieves the payload from a remote server; *iv*) the payload is cloaked in an asset delivered via a publicly-accessible platform, such as a store or a web server. According to the used method, different checks and countermeasures are enforced within security services implemented through a variety of architectural/functional blueprints. In more detail, for attack vectors *i*) and *ii*) a local antivirus could inspect files searching for known signatures. In the case of *iii*), a firewall or an intrusion detection system could block/spot the network conversation between the remote C&C server and the compromised node. Lastly, for the case *iv*), security checks enforced in a web server, a cloud provider, or an application store could spot the presence of malicious assets packed within in-line objects composing a web page or a `.dll` bundled with an application.

To have a reference use case, Figure 1 depicts the considered general attack model. Specifically, we consider an attacker hiding a malicious content in a digital image (i.e., a high-resolution icon) by exploiting steganography. Even if different techniques exist, in this model we consider the use of LSB steganography, which is the most used in real attacks [29, 30]. In essence, LSB steganography allows to alter the least significant bit of the color space of each pixel to cloak an arbitrary information. Yet, the more the data hidden, the higher the number of pixels showing an incoherent behavior, especially if compared against those in the surrounding area or with respect to "pictorial" features [29]. When the malicious payload is cloaked, it can be delivered via methods *i*)-*iv*), each one aiming at bypassing a specific security service.

3.2 Image Processing via Machine Learning

With Image Processing (IP), we refer to the technical analysis of an image through complex algorithms, which are usually exploited to address a variety of tasks. Improving the human understanding on information content of an image as well as extracting, storing and transmitting pictorial information [31] are typical examples of problems that can be tackled via IP. The recent advances in artificial intelligence and

(in particular) in machine learning allowed to further boost the capabilities of traditional frameworks and generate reliable and effective models. Among others, the Deep Learning (DL) paradigm [32] is particularly suited to reveal the presence of malicious information in digital images. Indeed, an emerging solution for identifying hidden contents relies on the usage of DNN, which allows for detecting and classifying compromised images through steganographic tools (see, e.g., [18]). Hence, accurate detection and classification models can be directly learned from raw low-level data (e.g., pixels, bits, and sensor data streams) by using approaches exploiting DL methodologies. Indeed, these models can learn in hierarchical fashion: several layers of non-linear processing units are stacked in a single network and each subsequent layer of the architecture can extract features with a higher level of abstraction compared to the previous one. Therefore, data abstractions and representations at different levels are automatically learned leading to effective solutions for analyzing and combining raw data. This is especially true for the case of processing digital images, for instance, high-resolution icons used in several commercial software such as Android OS. Moreover, discovered models can be updated incrementally or retrained only by considering new data coming from the observation of additional threats or variants of well-known attacks.

4 The Detection Approach

In this section, we first introduce the software architecture of our ecosystem for revealing the presence of steganographic threats targeting images. Then, we will discuss the design of the detection framework based on deep neural networks.

4.1 Architectural Blueprint

As discussed, the proposed detection framework aims at revealing the presence of malicious payloads embedded in images contained in different software artifacts, which can be retrieved from different sources. Figure 2 portraits the layered software architecture. As a first step, the detection framework "intercepts" the content and extract the digital image(s). For instance, this could require to extract assets from the resource bundle of an application or capture in-line objects composing an HTML hypertext. As soon as the image is retrieved, the detection logic exploiting the DNN checks for hidden contents. Malicious images can be discarded or an alarm can be raised. Usually, this type of countermeasure may be deployed in two different portions of the ecosystem to be protected.

In the first case, the detection framework can be placed at the border of the network closer to devices. For instance, nodes of a content delivery network infrastructure could be endowed with the needed functionalities to check assets to be distributed to end users. Therefore, detection can happen in edge nodes, home/smart gateways, or as an ad-hoc service running over local appliances. However, since end-to-end communications or the delivery of assets are often encrypted, this blueprint requires that the service provider has a full control over the entire distribution pipeline. If performance is not a tight constraint as well as to have access to encrypted data (e.g., exchanged via TLS), the detection can happen directly in end nodes.

In the second case, the framework can be deployed within the service to protect, i.e., in a centralized manner. For instance, it can be implemented as a software layer inspecting applications submitted by a developer before they become publicly available, or can be a component periodically checking assets stored in a datacenter.

Despite the placement, the detection framework should be properly trained and its model periodically updated. When considering edge-like deployments, such requirements could be too narrow. Thus, data gathering and training of the neural network can be done in a centralized manner in order to deliver only

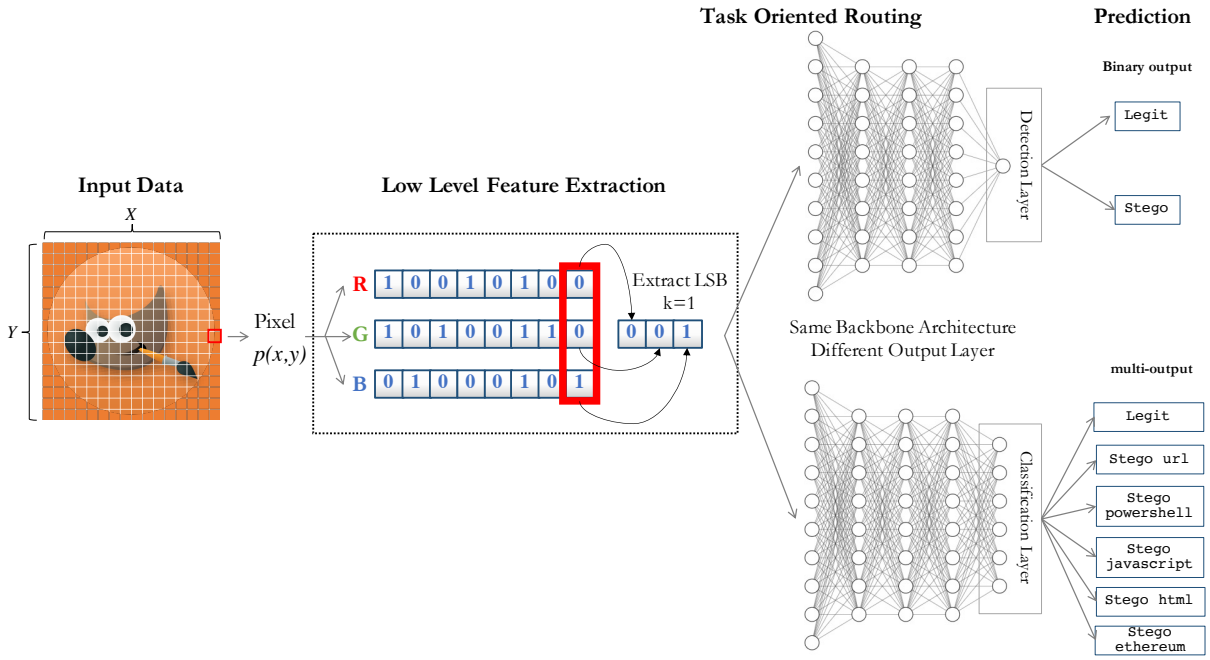


Figure 3: Methodological approach for the detection/classification of a stegomalware cloaked in a digital image via LSB steganography.

the model to devices with limited capabilities.

4.2 Hidden Content Detection Approach

Figure 3 shows the general approach adopted to address the problem of revealing images targeted via steganographic or information-hiding-capable techniques. Basically, the input data are images, i.e., matrices with dimension $X \times Y$. Each image is composed of pixels, the smallest manageable element of these matrices storing information about the color. The color of each pixel is obtained by combining three main components i.e., Red (R), Green (G) and Blue (B). As it will be detailed later, in this work we focus on high-resolution icons as they offer a sort of "unified playground" for various threats. Indeed, this does not account for a loss of generality, as the approach can be applied and scaled also to address regular-sized images. Thus, in the following, we consider that the values associated to RGB components represents the intensity of that color and ranges in the interval $[0,255]$. Specifically, three bytes are used to store the intensity value for each primary color. Hereinafter, we denote with N the size of the image computed as $N = X \times Y \times 3$.

As previously hinted, 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, Figure 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 appear as homogeneous with respect to the surrounding elements [8]. Therefore, many approaches proposed in literature fail in detecting the presence of hidden content as they produce weak detection models unable to discover the slight differences between licit and compromised contents.

To overcome the limitations of traditional frameworks, in this work we designed a machine-learning-based solution that focuses on processing and analyzing the k LSBs of the images under investigation. Basically, a vectorial representation is yielded by extracting the k least significant bits of each pixel, hence

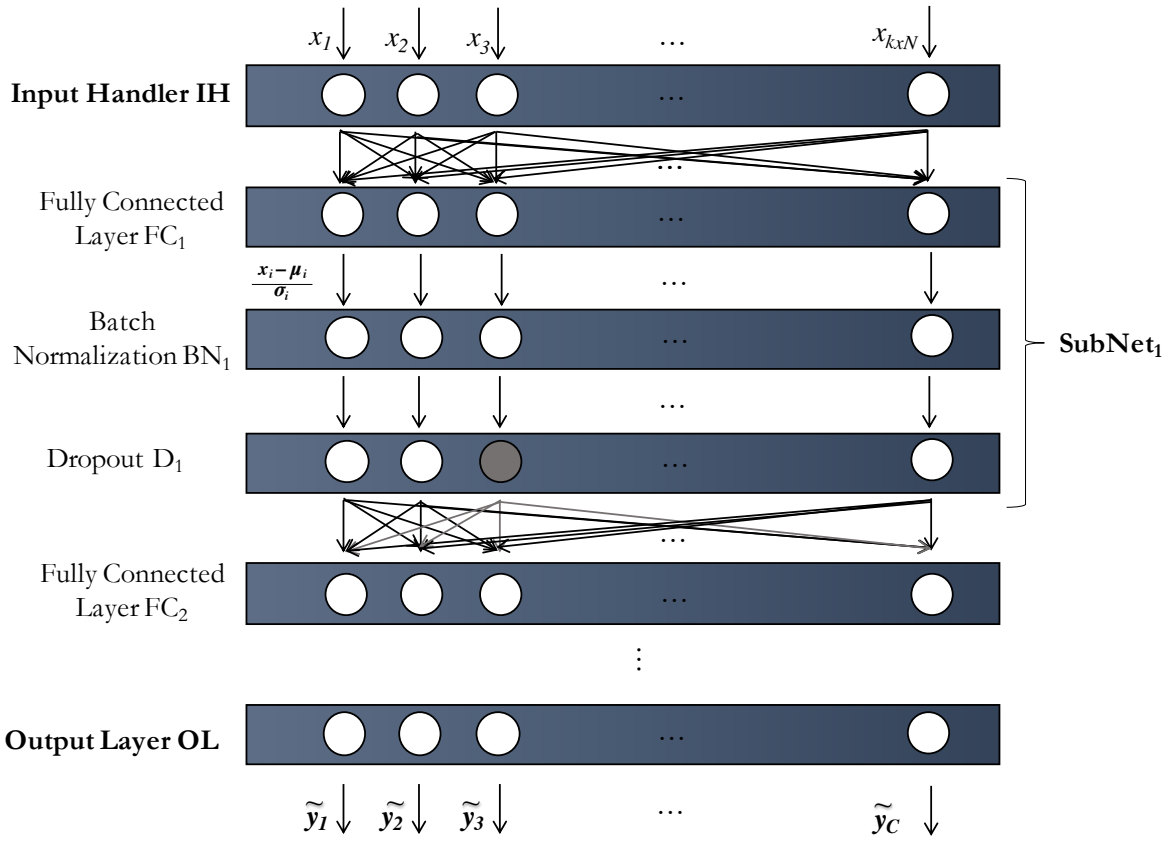


Figure 4: Neural architecture for hidden content detection and classification.

this representation is used to feed the learning phase of a Deep Neural Network. The raw information is automatically combined by the DNN hidden layers that allows for extracting high-level discriminative features.

The proposed approach allows dealing with both the main issues investigated in this work i.e., the effective detection of malicious data and its classification. Notably, the predictive performances of the detection model are generally not affected by the number k of LSBs analyzed as demonstrated in [33] for monochromatic images.

4.3 Neural Detector Architecture

To mitigate the risks arising from threats leveraging information hiding and steganography, we devised two (deep) neural models for their detection and classification. Specifically, we adopted the deep architectures depicted in Figure 4, which allow for yielding accurate predictions for both tasks. Basically, the two DNNs (i.e., detector and classifier) share the same backbone (i.e., same hidden layers and activation functions), but differ for the output layer, which is instantiated with a different number of neurons and activation function on the basis of the specific task to address. Essentially, our solution is composed of a stack of several sub-nets. The first layer has the role of handler for the input provided to the network (denoted in Figure 4 as `Input Handler`) and it propagates the raw data to the subsequent layers of the DNN for further processing. The size of this level is $k \times N$ i.e., the product between the number of least significant bits to analyze and the size of the image.

Both networks are composed of a variable number m of `SubNets` obtained by stacking three main

components: *(i)* on top, a fully-connected dense layer (equipped with a Rectified Linear Unit (ReLU) activation function [34]) 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)* to mitigate the risk of overfitting, a dropout layer is added to the subnet [35].

As an example, Figure 4 illustrates the overall model architecture. The first instance of this specific configuration has been labeled as $\texttt{SubNet}_1$. In more detail, the `Batch Normalization` implements the role of standardizing the data to be offered 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), whereas a reset of a random number of neurons during the learning stage is performed by applying the dropout mechanism. As pinpointed in [36], the adoption of a dropout discipline induces in the DNN a behavior similar to an ensemble model: in a nutshell, 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.

Finally, the number of neurons and the type of activation function of the `Output Layer` depends on the specific task to address. Regarding the detection, a single neuron providing the attack probability score is required. In particular, the `Output Layer` is equipped with a *sigmoid* activation function [37], which maps any given data instance $\overline{\mathbf{x}} = \langle \mathbf{x}_1, \ldots, \mathbf{x}_{k \times N} \rangle$ to an anomaly score $\tilde{\mathbf{y}}$ (i.e., the estimate of the probability that x is an image containing some hidden information).

By contrast, for the classification task the `Output Layer` will include C neurons (one for each class) and will be equipped with a *softmax* activation function [37]. Basically, we can consider the detection task as a sub-case of classification where $C = 1$. The proposed neural model is trained against a set $\mathscr{D} = \{(\overline{\mathbf{x}}_1, \mathbf{y}_1), (\overline{\mathbf{x}}_2, \mathbf{y}_2), \ldots, (\overline{\mathbf{x}}_D, \mathbf{y}_D)\}$, where $\mathbf{x}_i$ is the k-LSB-based representation of the image while $\mathbf{y}$ is the class of the image. For the detection, $\mathbf{y}$ takes a binary value specifying the legitimate/compromised nature of the image. For the classification task, 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. For the detection task, the *binary crossentropy* is exploited to compute the network weights, which is defined as follow:

$$BCE(\mathbf{y}, \tilde{\mathbf{y}}) = -\frac{1}{|\mathscr{D}|} \sum_{i=1}^{|\mathscr{D}|} \mathbf{y}_i \log \tilde{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \tilde{\mathbf{y}}_i).$$

By contrast, the *categorical crossentropy* is adopted for the classification task and it is calculated as follows:

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

5 Performance Evaluation

In this section, we first present how the dataset has been prepared to model the attack template introduced in Section 3.1. Then, we showcase numerical results.

5.1 Design of Attacks and Dataset Preparation

To model a wide range of threats leveraging steganography, we consider an attacker cloaking different malicious payloads within high-resolution icons. This can be representative of an APT or a Crime-as-a-

Service toolkit offering cross-platform offensive functionalities or generic hiding capabilities. Moreover, icons are an ubiquitous digital asset deployed in software running in mobile nodes (e.g., Android and iOS devices), tablets and set top boxes (e.g., Windows and Android ecosystems), as well as in desktops. In general, icons are provided in different sizes or dynamically scaled by the guest OS according to the resolution or the intended usages, for instance to identify file types or to show applications on the desktop/dashboard. Thus, without loss of generality and to take into account possible future scenarios, we considered an attacker targeting icons of a size of 512×512 pixels.

Concerning payloads, we want to model threats using steganography or information hiding to conceal different malicious assets, as it happens in real-world attack campaigns and APTs. To not limit our investigation and to reflect the increasing trend of endowing malware with some stealthy capabilities, we utilized the following realistic malicious payloads [1, 4]:

- JavaScript code[1]: threats that can target victims in a cross-platform manner. For instance, such scripts can be used to retrieve an additional payload, de-obfuscate weaponized contents stored in the filesystem, or implement file-less malware;
- Obfuscated JavaScript in HTML[2]: many scripts are usually obfuscated within an hypertext. On one hand, this allows to trigger their execution when used in Web-based attack chains. On the other hand, scripts can be concealed within chunks of text or comments, via a wide array of text-based obfuscation mechanisms. Hence, this type of contents are expected to proliferate for making detection and forensics attempts harder;
- PowerShell scripts[3]: malicious PowerShell code has been observed in many steganographic malware. For instance, the Invoke-PSImage technique used to hide PowerShell contents in images has been at the basis of several attack campaigns, such as those against the Pyeongchang Olympic Games and for the diffusion of Greystars and Bandook [15];
- Ethereum Addresses[4]: ransomware and cryptojackers in many cases contain data to programmatically reach a remote wallet or to instruct a victim for paying the ransom. To avoid detection or to update the malware during its lifespan, fixed-length hash values identifying crypto wallets can be hidden into innocent-looking contents and periodically retrieved from a C&C server.
- URL and IP Addresses[5]: the majority of threats contacts a remote facility to exfiltrate data as well as to retrieve additional payloads or configurations (see, e.g., the ZeusVM banking trojan using steganography to conceal a list of addresses and URLs belonging to financial institutions [4]). In general, such information is hardcoded in the malware, thus making the creation of signatures for binary analysis easy [1]. Hence, many recent threats cloak both IP addresses and DNS entries in digital images to escape detection.

To conceal the payloads within the images, we used the LSB steganography technique, which has been observed in many real-world threats [4, 30]. To this aim, we employed LSBSteg[6]. In essence, the tool hides the source payload (i.e., the malicious content) in the red, green, and blue color channel of each pixel. To avoid trivially-visible artifacts, we considered payloads that can be hidden by only using 1 bit per channel, i.e., only payloads with a size of $512 \times 512 \times 3$ bits.

[1]Javascript Malware Collection. Online: `https://github.com/HynekPetrak/javascript-malware-collection`
[2]Malicious Javascript Dataset. Online: `https://github.com/geeksonsecurity/js-malicious-dataset`
[3]PowerShell dataset. Online: `https://github.com/denisugarte/PowerDrive`
[4]Ethereum-lists. Online: `https://github.com/MyEtherWallet/ethereum-lists`
[5]URLhaus database. Online: `https://urlhaus.abuse.ch`
[6]LSBSteg tool. Online: `https://github.com/RobinDavid/LSB-Steganography`

	Train	Test	Validation
Clean	4,000	2,000	2,000
JavaScript	2,363	1,188	1,214
JavaScript in HTML	2,284	1,167	1,162
PowerShell	2,468	1,164	1,213
Ethereum addresses	2,473	1,247	1,193
URL/IP addresses	2,412	1,234	1,218
Total	16,000	8,000	8,000

Table 1: Breakdown of the dataset used to model a malware exploiting steganography and test our detection approach.

To have a realistic condition, we selected images from different open source repositories (and released under GPL3 licence) to build a dataset composed of $8,000$ equally-sized images. To obtain ample test settings, we combined each image with three different payloads, which have been selected randomly. The selection process has been modeled with an uniform distribution among the different payloads, i.e., JavaScripts, obfuscated JavaScripts in HTML, PowerShell scripts, Ethereum addresses, and IP/URLs.

As a result, we obtained $32,000$ images containing the various malicious contents. The dataset has been further divided into train, test, and validation sets. The overall breakdown is reported in Table 1. As indicated, each entry represents the amount of images generated for each set, considering the particular payload type. It is important to note that we considered a scenario in which each clean picture could be affected by all types of attacks, therefore the resulting dataset exhibits an unbalanced distribution (i.e., clean images represent the minority class). Specifically, we are interested in investigating both the ability of our model in recognizing corrupted images and distinguishing among the different types of attacks. To make the evaluation fair, our test-set is created by considering only images not included in the training set and embedding them with all types of malicious information.

To verify the effectiveness of our approach when revealing the presence of hidden payloads within "unseen" digital images, we prepared two additional test sets. Such sets have been used to model an attacker aware of the countermeasure, thus trying to escape the detection via obfuscation or lateral movements [3]. In more detail, the first additional test set considers payloads encoded in Base64 with the `base64` Linux utility version 1.13.4. The second instead models attacks à-la LokiBit, which exploits zip compression to further obfuscate the hidden data. In this case, we used the `zip` Linux utility version 3.0 with the deflation compression method. To generate the two additional datasets (denoted in the following as "Base64 Test" and "ZIP Test", respectively), we considered again the same $8,000$ images with the same proportions for the payloads.

5.2 Preliminaries and Experimental Environment

As illustrated in Section 4, the proposed approach relies on the usage of a DNN architecture for detecting and classifying compromised images. To validate the approach, a prototype implementation written in Python based on the Tensorflow library[7] has been developed. As regards the DNN architecture instantiated for the experimentation, it includes $m = 3$ SubNets. Specifically, the hidden fully-connected layers are composed of 128 neurons and equipped with ReLU activation functions, while the reset probability for the dropout is 2.5%. RMSprop is adopted as optimizer with a learning rate of $1e-3$. Both models used for detection and classification tasks are learned over 20 epochs with a batch size of 256.

[7] `https://www.tensorflow.org/`

Table 2: Experimental results in terms of Detection and Prediction capabilities obtained by varying the type of encoding for the payload (i.e., plain, Base64 Encoding, and compressed). Each metric is computed by adopting a macro-averaging strategy.

Task	Testset Payload Type	Accuracy	Precision	Recall	F-Measure	AUC
Detection	Plain	0.992	0.999	0.990	0.994	1.000
	Base64 Encoding	0.999	0.999	0.999	0.999	1.000
	Zip Encoding	1.000	1.000	1.000	1.000	1.000
Classification	Plain	0.820	0.830	0.830	0.829	0.969
	Base64 Encoding	0.683	0.702	0.670	0.670	0.886
	Zip Encoding	0.487	0.469	0.438	0.324	0.704

To assess the quality of the results obtained by the proposed approach, a number of well-known performance metrics are used. Let be TP the number of positive correctly classified cases, FP the number of positive incorrectly classified cases, FN the number of positive incorrectly classified cases and TN the number of negative correctly classified cases, then we can define the measures as follows:

- *Accuracy*: is the fraction of correctly classified cases, i.e., $\frac{TP+TN}{TP+FP+FN+TN}$;
- *F-Measure*: summarizes the overall system performances and it is defined as the harmonic mean of *Precision* and *Recall* defined as $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$, respectively;
- *Area Under the Curve (AUC)*: the Receiver Operating Characteristic (ROC) curve is obtained by plotting the False Positive Rate (i.e., the ratio between the number of false alarms signaled and that of all the licit images) and the True Positive Rate (i.e., the Recall) for different class probability values. As a result, the AUC is the area under the ROC curve.

Although, both *F-Measure* and *AUC-PR* are defined for binary classification problems, they can be extended for multi-class scenarios by averaging the computed value for each class according to two possible strategies, respectively named *macro* and *micro* [38]. In the first case, the performance measure is computed for each class and then averaged, while in the second case, the metric is computed as the cumulative sum of the counts of various true/false positive/negative and subsequently the overall measure is calculated. Notably, in our experimental setting we consider the former strategy since it is particularly recommended for evaluating unbalanced scenarios, then each macro-averaged metric is computed as the arithmetic mean (or unweighted mean) of values obtained for that metric for each class.

Lastly, to perform experiments we used a machine equipped with an Intel Core I9-9980HK CPU @2.40GHz and 32 GB RAM. The validation set has been exploited to select the model guaranteeing the best loss value from the training phase.

5.3 Numerical Results

Table 2 showcases the results obtained for both detection and classification tasks and by considering the different attack scenarios presented in Section 5.1. In the first scenario, the hidden information has been encoded in a plain ASCII format, whereas in the second and third scenario, the payload has been encoded in Base64 or compressed in zip format.

As regards the detection, the proposed framework is characterized by excellent results (i.e., $\sim 100\%$ of accuracy) in every scenario. Instead, for the classification task, the overall performances decay, especially when the payload is compressed before being cloaked in the image. The same behavior can be

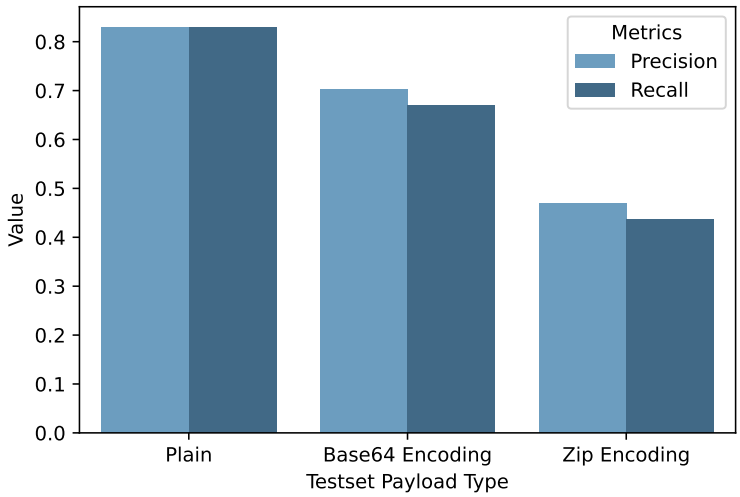
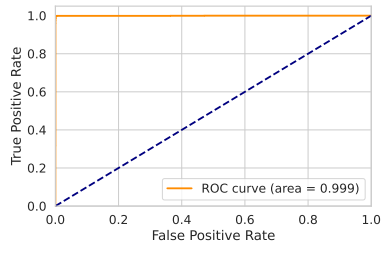
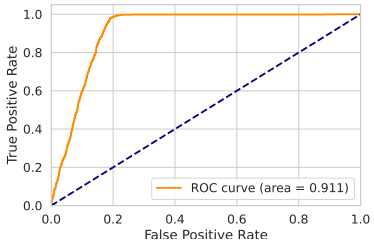


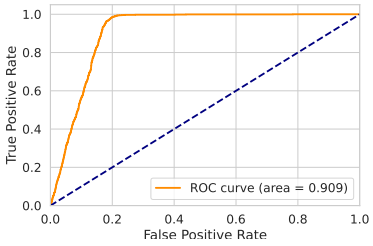
Figure 5: Precision and Recall values for the classification task and grouped by Payload Type.



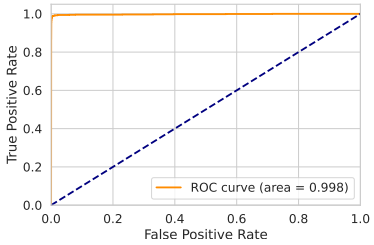
(a) Clean Class



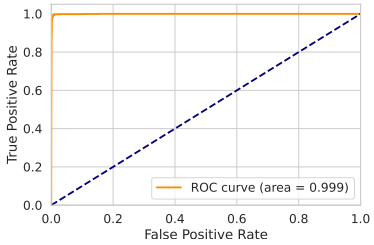
(b) URL/IP Addresses Payload



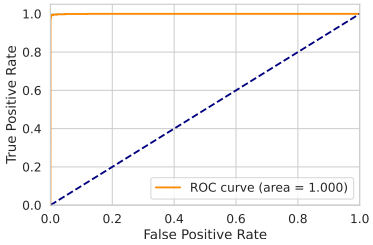
(c) Ethereum Addresses Payload



(d) Javascript in HTML Payload



(e) JavaScript Payload

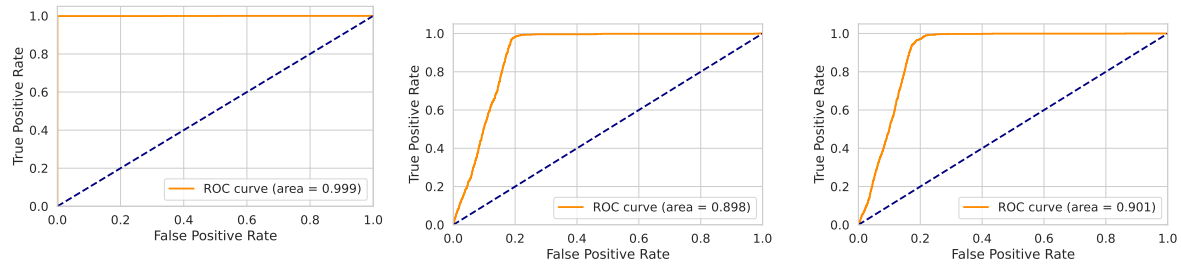


(f) PowerShell Payload

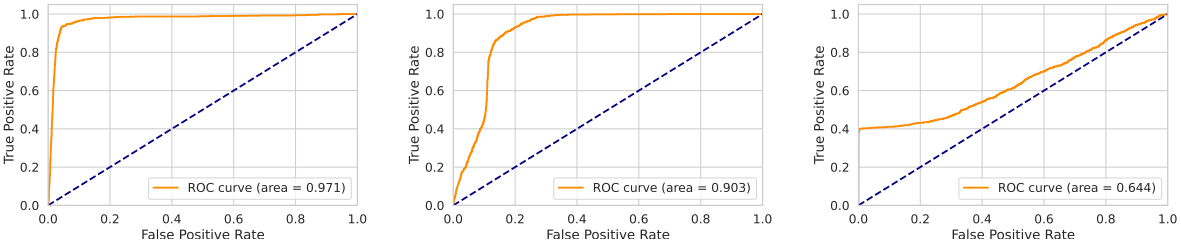
Figure 6: ROC Curves for Plain Testset.

observed in Figure 5, where the values for the precision and recall are reported for all the three considered scenarios/cases. As shown, the encoding/compression operation seems reducing the capability of the model to distinguish among the different classes, since both the precision and recall exhibit a decreasing trend.

To investigate such a behavior in more detail, a further analysis has been performed. Specifically, we quantified the prediction capabilities of the model when dealing with each class for each scenario. To this aim we plotted the corresponding ROC curves, which have been grouped according to the use of "normal" hiding techniques, the adoption of an additional encoding, or the use of zip compression to obfuscate the payload. In more detail, Figure 6 shows the ROC curves when the payload is directly



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



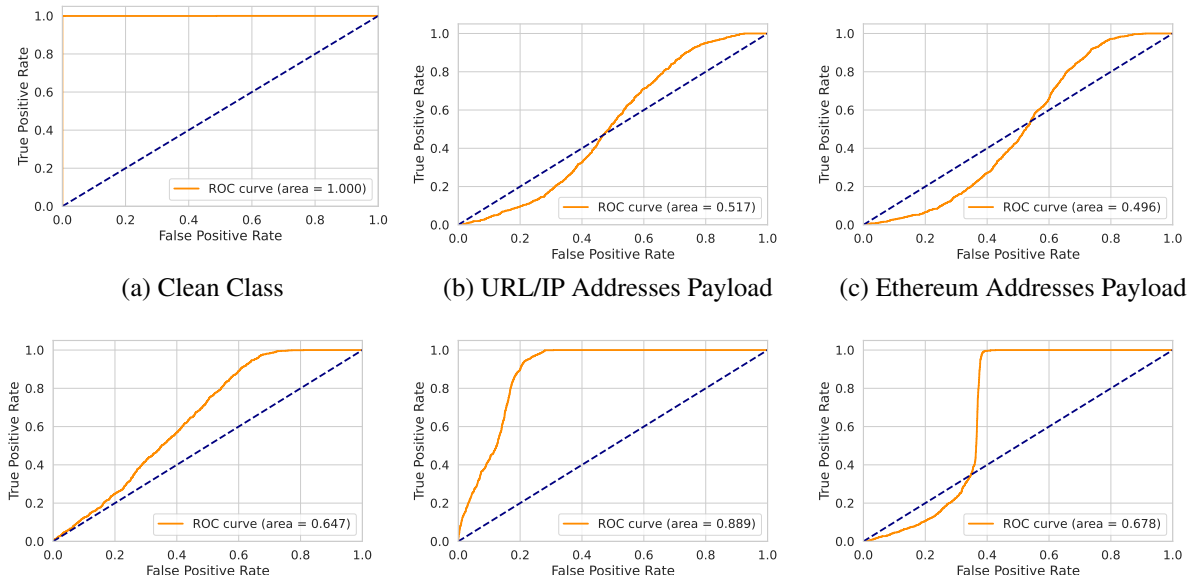
(d) Javascript in HTML Payload (e) JavaScript Payload (f) PowerShell Payload

Figure 7: ROC Curves for Base64 Encoded Testset.

embedded in the images, i.e., no elusive mechanisms are deployed by the attacker. As shown, the neural classifier exhibits good performance for each class, although it could misclassify URL/IP addresses with Ethereum ones. This behavior can be explained by considering the close similarity of such payloads (i.e., both are characterized by short string containing alphanumerical characters). As a consequence, the learning process becomes more difficult. The case of an attacker using the Base64 encoding before the LSB steganographic injection is the depicted in Figure 7. Specifically, we can observe the slight degradation of the performances for each class (except the legitimate one). It must be noted that, the PowerShell class is the one suffering most of the encoding: essentially, PowerShell scripts tend to be misclassified with the JavaScript counterpart. This behavior can be ascribed to the fact that the prose of both PowerShell and JavaScript shares the use of statements (e.g., `if-then` clauses), parenthesis and specific punctuation.

Finally, Figure 8 deals with the case of an attacker deploying obfuscation via zip compression, i.e., the payload has been compressed and then embedded in the image. As it can be seen, a further performance degradation can be observed, although the model is able to distinguish between compromised and legitimate images. In essence, payloads containing an "address" (i.e., IP/URLs and Ethereum pointers) are misclassified each other, while PowerShell and Javascript in HTML are labeled as Javascript payload. According to further investigations, this behavior is mainly due to the fact that the compression operation reduces the differences among the payloads. In fact, the metadata added to the compressed representation further contribute to make harder the classification task as they are similar for all the classes. In other words, the data structure imposed by the zip algorithm constitutes the majority of the information compared to the original form.

Lastly, we point out that our approach can be deployed in a simple manner in many realistic scenarios, especially owing to its limited resource footprint. In more detail, the average prediction time for a single image is ~ 5 ms calculated on the same machine used for the experimental campaign. Hence, the architecture of Figure 2 could be implemented over commodity hardware to protect small- and medium-sized networks in a centralized manner (e.g., by deploying a specific appliance). Besides, if timing constraints are not too tight, our approach can be also deployed in edge nodes protecting SOHO networks or grant-



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

(d) Javascript in HTML Payload (e) JavaScript Payload (f) PowerShell Payload

Figure 8: ROC Curves for ZIP Encoded Testset.

ing access to various smart devices. Another possible deployment flavor could exploit the DNN to equip local antivirus with some form of AI to reveal steganographic threats: in this case, resource-intensive operations (e.g., training) can be done in a centralized manner and updates about the configuration of the neural network can be delivered locally.

6 Conclusion and Future Work

In this paper we have presented an approach for detecting digital pictures containing malicious payloads hidden via LSB steganography. To this aim, we used DNNs and and demonstrated the feasibility of our approach by focusing on malware exploiting icons with a size of 512×512 pixels. This allowed to consider threats targeting an heterogeneous population of devices/ecosystems, which share the use of icons. Results showcased the effectiveness of our approach also when handling payloads obfuscated via zip or further processed with an alternative encoding, i.e., the attacker deployed some elusive technique.

Future works aim at refining the proposed idea. Specifically, part of our ongoing research is devoted to understand the impact in terms of scalability of the approach, especially to understand its feasibility in protecting Internet-scale services or cloud datacenters in an effective manner. In addition, future research aims at performing detection of a wider array of digital images, also by considering threats using different steganographic techniques (e.g., DCT steganography) or information hiding approaches (e.g., manipulation of metadata).

References

[1] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska. Tight arms race: Overview of current malware threats and trends in their detection. *IEEE Access*, 9:5371–5396, December 2020.

[2] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach. Dynamic malware analysis in the modern era—a state of the art survey. *ACM Computing Surveys*, 52(88):1–48, September 2020.

[3] W. Mazurczyk and S. Wendzel. Information hiding: challenges for forensic experts. In *Proc. of the 13th ACM International Computing Education Research Conference (ICER'17),Tacoma Washington, USA*, pages 86–94. ACM, January 2017.

[4] W. Mazurczyk and L. Caviglione. Information hiding as a challenge for malware detection. *IEEE Security & Privacy*, 13(2):89–93, April 2015.

[5] A. Biørn-Hansen, T.M. Grønli, and G. Ghinea. A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Computing Surveys*, 51(108):1–34, November 2018.

[6] G. Suarez-Tangil, J.E. Tapiador, and P. Peris-Lopez. Stegomalware: Playing hide and seek with malicious components in smartphone apps. In *Proc. of the 10th International Conference on Information Security and Cryptology (ICISC'14), Beijing, China*, volume 8957 of *Lecture Notes in Computer Science*, pages 496–515. Springer, December 2014.

[7] L. Caviglione. Trends and challenges in network covert channels countermeasures. *Applied Sciences*, 11(4):1641, February 2021.

[8] M. Zuppelli, G. Manco, L. Caviglione, and M. Guarascio. Sanitization of images containing stegomalware via machine learning approaches. In *Proc. of the 4th Italian Conference on Cybersecurity (ITASEC'21), online*, volume 2940, pages 374–386. SIMARGL, April 2021.

[9] D. Gibert, C. Mateu, and J. Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, March 2020.

[10] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[11] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, September 2007.

[12] G. Stergiopoulos, D. Gritzalis, E. Vasilellis, and A. Anagnostopoulou. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9:44–57, June 2007.

[13] M. Płachta, M. Krzemień, K. Szczypiorski, and A. Janicki. Detection of image steganography using deep learning and ensemble classifiers. *Electronics*, 11(10):1565, May 2022.

[14] J. Qin, Y. Luo, X. Xiang, Y. Tan, and H. Huang. Coverless image steganography: A survey. *IEEE Access*, 7:171372–171394, November 2019.

[15] R. Han, C. Yang, J. Ma, S. Ma, Y. Wang, and F. Li. IMShell-Dec: Pay more attention to external links in PowerShell. In *Proc. of the 35th IFIP International Conference on ICT Systems Security and Privacy Protection, Maribor, Slovenia*, pages 189–202. Springer, September 2020.

[16] I. J. Kadhim, P. Premaratne, P. J. Vial, and B. Halloran. Comprehensive survey of image steganography: Techniques, evaluations, and trends in future research. *Neurocomputing*, 335:299–326, March 2019.

[17] A. Schaffhauser, W. Mazurczyk, L. Caviglione, M. Zuppelli, and J. Hernandez-Castro. Efficient detection and recovery of malicious PowerShell scripts embedded into digital images. *Security and Communication Networks*, 2022, June 2022.

[18] D. Hendler, S. Kels, and A. Rubin. Detecting malicious PowerShell commands using deep neural networks. In *Proc. of the 2018 on Asia Conference on Computer and Communications Security (ASIA CCS'18), Incheon, Korea*, pages 187–197. ACM, May 2018.

[19] M.A. Elsadig and A. Gafar. Covert channel detection: Machine learning approaches. *IEEE Access*, 10(1):38391–38405, April 2022.

[20] M. Guarascio, M. Zuppelli, N. Cassavia, L. Caviglione, and G. Manco. Revealing MageCart-like threats in favicons via artificial intelligence. In *Proc. of the 17th International Conference on Availability, Reliability and Security (ARES'22), Vienna, Austria*, pages 1–7. ACM, August 2022.

[21] A. Cohen, N. Nissim, and Y. Elovici. MalJPEG: Machine learning based solution for the detection of malicious JPEG images. *IEEE Access*, 8:19997–20011, January 2020.

[22] X.-Y. Luo, D.-S. Wang, and F.-L. Wang, P.and Liu. A review on blind detection for image steganography. *Signal Processing*, 88(9):2138–2157, September 2008.

[23] D. Hu, L. Wang, W. Jiang, S. Zheng, and B. Li. A novel image steganography method via deep convolutional

generative adversarial networks. *IEEE Access*, 6:38303–38314, July 2018.

[24] D. Puchalski, L. Caviglione, R. Kozik, A. Marzecki, S. Krawczyk, and M. Choraś. Stegomalware detection through structural analysis of media files. In *Proc. of the 15th International Conference on Availability, Reliability and Security (ARES'20), online*, pages 1–6. ACM, August 2020.

[25] D.-S. Jung, S.-J. Lee, and I.-C. Euom. ImageDetox: Method for the neutralization of malicious code hidden in image files. *Symmetry*, 12(10):1621, September 2020.

[26] Y. Sun, H. Zhang, T. Zhang, and R. Wang. Deep neural networks for efficient steganographic payload location. *Journal of Real-Time Image Processing*, 16(3):635–647, January 2019.

[27] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, July 2019.

[28] I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA'10), Fukuoka, Japan*, pages 297–300. IEEE, November 2010.

[29] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal processing*, 90(3):727–752, March 2010.

[30] S. Wendzel, L. Caviglione, W. Mazurczyk, A.Mileva, J. Dittmann, C. Krätzer, K. Lamshöft, C. Vielhauer, L. Hartmann, J. Keller, et al. A revised taxonomy of steganography embedding patterns. In *Proc. of the 16th International Conference on Availability, Reliability and Security (ARES'21), Vienna, Austria*, pages 1–12. ACM, August 2021.

[31] R.C. Gonzalez and R.E. Woods. *Digital image processing*. Prentice Hall, 2018.

[32] Y. Le Cun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[33] J.D. Miranda and D.J. Parada. LSB steganography detection in monochromatic still images using artificial neural networks. *Multimedia Tools and Applications*, 81(1):785–805, September 2021.

[34] V. Nair and G.E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proc. of the 27th International Conference on International Conference on Machine Learning (ICML'10), Haifa, Israel*, pages 807–814. ICML, June 2010.

[35] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, June 2014.

[36] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv*, 1207.0580:1–18, July 2012.

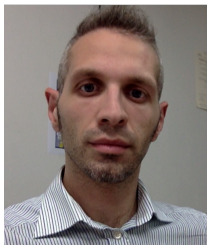
[37] M. Guarascio, G. Manco, and E. Ritacco. Deep learning. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 1:634–647, 2019.

[38] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, July 2009.

Author Biography

Nunziato Cassavia holds a PhD and an MSc in Computer Engineering from the University of Calabria, Italy. Currently, he is a research fellow at the Institute for High Performance Computing and Networking (ICAR-CNR) of the National Research Council (CNR) in Italy. His research interests include Cybersecurity, Big Data, Distributed Computing System, Data Warehouse and Relational and NoSQL Databases. He co-authored several papers published in international conference proceedings, chapters and journals and participated in European and national research projects concerning Big Data and Cybersecurity.

Luca Caviglione received the Ph.D. degree in electronics and computer engineering from the University of Genoa, Genoa, Italy. He is a Senior Research Scientist with the Institute for Applied Mathematics and Information Technologies of the National Research Council of Italy, Genoa. His research interests include optimization of large-scale computing frameworks, wireless and heterogeneous communication architectures, and network security. He is an author or co-author of more than 150 academic publications and several patents in the field of p2p and energy-aware computing. He has been involved in many research projects funded by the European Space Agency, the European Union, and the Italian Ministry of Research. He is a Work Group Leader of the Italian IPv6 Task Force, a contract professor in the field of networking/security and a professional engineer.



Massimo Guarascio holds a PhD in Systems and Computer Science Engineering and a master's degree in Computer Science Engineering, both from the University of Calabria. He is currently researcher at the Institute for High Performance Computing and Networking of the National Research Council (ICAR-CNR) and shareholder of OKT s.r.l., a spin-off of University of Calabria. He co-authored over 50 papers published in international conference proceedings, chapters and journals. His research mainly focuses on machine learning, anomaly detection and explanation, process mining, data analytics methods for geosciences and remote sensing, knowledge discovery and data mining for cyber security and fraud detection. He has participated in European and national research projects concerning machine learning and cybersecurity.

Giuseppe Manco graduated summa cum laude in computer science and received a PhD degree in computer science from the University of Pisa. He is currently a Director of Research at the Institute for High Performance Computing and Networking (ICAR-CNR) of the National Research Council of Italy and a contract professor at University of Calabria, Italy. His current research interests include knowledge discovery and data mining, Recommender Systems and Social Network Analysis, Deep Learning. He has been the coordinator of several national and international research projects. He has been serving in the program committee of several international/national conferences, including: IJCAI, AAAI, IEEE ICDM, ECMLPKDD, SIAM SDM, PAKDD. He served as a program co-chair of ECML-PKDD 2016. He is serving as an associate editor for the Journal of Intelligent Information Systems, Knowledge and Information Systems and Machine Learning Journal.

Marco Zuppelli is a third year PhD student at University of Genoa and a research fellow at the Institute for Applied Mathematics and Information Technologies of the National Research Council of Italy. Within the European Project SIMARGL (Secure Intelligent Methods for Advanced RecoGnition of malware and stegomalware), he investigated novel detection methods for steganographic malware exploiting both network and local covert channels. His main research interests are the use of in-kernel methodologies (e.g., the extended Berkeley Packet Filter) to collect information on software/network components, and the design of mechanisms for detecting malicious communications in an efficient, scalable and extensible manner. Moreover, he is investigating images compromised by steganographic malware to study machine learning and artificial intelligence applications.