Sanitization of Images Containing Stegomalware via Machine Learning Approaches

Marco Zuppelli[2], Giuseppe Manco[1], Luca Caviglione[2], Massimo Guarascio[1]

[1] ICAR - Institute for High Performance Computing and Networking, Rende, Italy
name.surname@icar.cnr.it
[2] IMATI - Institute for Applied Mathematics and Information Technologies, Genova, Italy
name.surname@ge.imati.cnr.it

Abstract

In recent years, steganographic techniques have become increasingly exploited by malware to avoid detection and remain unnoticed for long periods. Among the various approaches observed in real attacks, a popular one exploits embedding malicious information within innocent-looking pictures. In this paper, we present a machine learning technique for sanitizing images containing malicious data injected via the Invoke-PSImage method. Specifically, we propose to use a deep neural network realized through a residual convolutional autoencoder to disrupt the malicious information hidden within an image without altering its visual quality. The experimental evaluation proves the effectiveness of our approach on a dataset of images injected with Powershell scripts: Our tool removes the injected artifacts and inhibits the reconstruction of the scripts, partially recovering the initial image quality.

1 Introduction

One of the most important drivers for improvement in network defense is the development of novel methods for detecting threats endowed with the ability of cloaking their existence or remaining unnoticed for long time frames [1]. Until recently, malware developers relied upon code obfuscation, anti-forensics techniques such as memory encryption, as well as multi-stage loading architectures able to delay the retrieval of malicious payloads until when needed. However, a new trend leverages some form of steganography or information hiding to prevent detection and to bypass execution perimeters enforced via sandboxing or virtualization [2]. Accordingly, such an emerging wave of threats have been named *stegomalware*.

In general, stegomalware exploits hidden communication paths to exfiltrate sensitive information towards a remote command & control facility, exchange commands and configuration parameters to activate a backdoor, orchestrate nodes composing a botnet and launch attacks, and retrieve malicious routines or additional functionalities [1, 2]. Even if a variety of carriers can be used to contain secret information (e.g., data can be covertly injected in network packets or by modulating hardware behaviors like the rotation speed of a fan), digital images are still one of the most preferred targets [2, 3]. Specifically, many attacks collected in the wild[1] apply steganographic techniques to digital images for exchanging data. For instance, sensitive information can be embedded in favicons of web pages or in innocent looking pictures attached to email messages. Additionally, a malware can hide attack routines in images spread over the file-system of the victim as to prevent detection by classical tools like antivirus [1, 2, 3].

Therefore, being able to mitigate the impact of stegomalware is of prime importance to fully assess the security of modern networking and computing infrastructures. In this vein, the

[1] Steg-in-the-wild: a curated list of real-world attacks leveraging information hiding and steganography. Available on line at: https://github.com/lucacav/steg-in-the-wild [Last Accessed: March 2021].

research community has started addressing issues for counteracting stegomalware, especially threats targeting mobile devices (see, e.g., [4] and the references therein). During the years, machine learning has become a key component to balance the outcome of the never-ending battle between security experts and malware developers [1]. As an example, a recent survey highlighted its ability of detecting threats in an efficient manner, but also clearly indicated some limits of the approach. In fact, the detection requires suitable datasets, which are hard to create and distribute in a standardized form. Moreover, attacks tend to evolve, thus leading to "concept drift" phenomena accounting for degradation of the models [5]. Despite the used approach (e.g., static and dynamic analysis or AI-based methodologies), the mitigation of stegomalware is still a largely unexplored and neglected area [1, 6]. Indeed, the detection of attacks using images to conceal information can borrow many results from the literature dealing with image steganalysis, which adopted machine learning techniques from a decade [7, 8]. However, the majority of works are general and not primarily focused on the detection of malware or part of the kill chain commonly used in cyber attacks.

In this perspective, in this work we introduce an approach for mitigating malware targeting digital images to exfiltrate data and to distribute malicious code. Specifically, we consider images embedding PowerShell scripts via the Invoke-PSImage technique, which has been observed in many real-world threats and malicious campaigns [1]. Different from the classical image steganalysis approach aimed at detecting secret information, our goal is to mitigate attacks by sanitizing images via a framework based on autoencoders to disrupt the hidden harmful payload without degrading their quality. The choice of autoencoders has been driven both by their properties and their performances when handling security-related tasks, such as the extraction of features to discriminate malicious calls of APIs [9].

Summing up, the contribution of this paper is twofold: *i*) the design of an architectural framework for the mitigation of information-hiding-capable threats, and *ii*) a preliminary performance evaluation campaign considering the Invoke-PSImage injection mechanism used by many malware acting "in the wild".

The rest of the paper is structured as follows. Section 2 provides background details on stegomalware targeting digital media and machine learning techniques for image processing. Section 3 deals with the proposed approach for preventing steganographic attacks via image sanitization, while Section 4 showcases numerical results. Finally, Section 5 concludes the paper and outlines possible future research directions.

2 Background

This section introduces the considered attack model and provides details on the machine learning approaches used to manipulate images in the rest of this work.

2.1 Stegomalware and Attack Model

The typical attack model exploited by a malware when using digital images is depicted in Figure 1. The archetypal ultimate goals are: evading detection mechanisms of tools like an antivirus, or allowing to exchange data with the victim even in the presence of a firewall or other blocking mechanisms [2]. To this aim, as a preliminary step, the attacker utilizes some information-hiding-based approach to inject a secret information within a digital picture. The latter should not appear as an anomaly and the embedding process should not generate visible artifacts revealing the presence of the secret. In this work, we will consider an embedding process

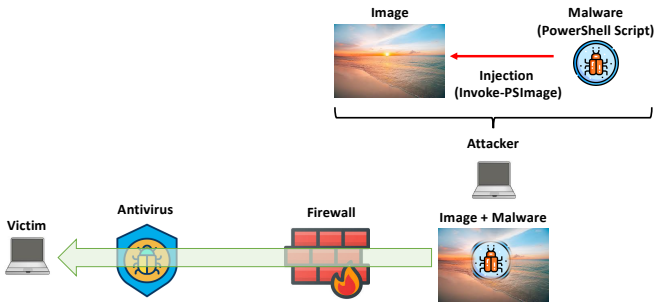


Figure 1: Reference attack model: an attacker embeds a malicious script into an innocent-looking image to evade detection and security tools.

leveraging the Invoke-PSImage tool[2], which is based on the Least Significant Bit steganography (LSB) method [10]. Put briefly, Invoke-PSImage tool allows to encode a PowerShell script within an input image. To hide the payload, the embedding method uses the 4 least significant bits of each pixels of the green and blue channels. As a result, each pixel of the processed image (exported in the PNG format) will contain up to 1 byte of secret information.

According to threats observed in the wild, the secret data could be an attack routine, a script, a configuration file or a sequence of commands [2, 3]. To have a realistic scenario, in the following we will consider an attacker hiding a PowerShell script within the digital image, as it has been observed in various malware samples [1] and advanced persistent threats[3]. The obtained image is seldom delivered to the victim in a direct manner: rather, a third-party vector is exploited. For instance, the attacker can send the image via a phishing campaign, or manipulate the content of a web page: such an altered resource can be then fetched by the victim. As an example, the Magecart malware exfiltrated payment information by hiding them into images implementing an e-commerce site [1].

Upon reaching the victim, the malicious payload can be retrieved and then utilized to complete the attack. Usually, part of the malware has already been injected in the host of the victim and awaits for a specific image to be scanned for extracting the information (e.g., a list of IP addresses to contact). Another typical mechanism exploits some form of social engineering where the victim is decoyed as to use the infected image in a way that the payload detonates.

2.2 Machine Learning for Image Processing

Image processing represents the technical analysis of an image by means of complex algorithms. Roughly, it can be considered as a process where both the input and the output are an image. Image processing techniques can be used to improve the information content of an image for human understanding, as well as for extracting, storing and transmitting pictorial information [11]. Although the field is generally considered loosely separated from image analysis and computer vision, the rapid acceleration of new artificial intelligence and machine learning methods within the latter fields has opened new opportunities even in the former. In particular, the recent establishment of Deep Learning (DL) techniques [12] has fostered significant improvements in various applications of image processing and computer vision, e.g., image enhancement, restoration and morphing. By exploiting multiple levels of abstractions, deep architectures allow to

[2] https://github.com/peewpw/Invoke-PSImage
[3] https://cyware.com/news/new-malware-strain-abuses-github-and-imgur-e29bc6f6

discover highly accurate models by capturing interactions between set of features directly from raw and noisy image data. The capability of learning such abstractions represents one of the most important and disruptive aspects introduced by the DL framework: no feature engineering or interaction with domain experts are required to build good representative features.

Convolutional Neural Networks (CNN) [13, 14] represent a particularly relevant variant of traditional neural networks, where the connectivity between neurons is delved on a local basis, thus allowing to capture the invariance of patterns to distortion or shift in the input data. Under this perspective, CNNs architectures are particularly well-suited for the analysis of image data. A basic CNN can be devised as a stacking of layers where each of them transforms one volume of activations to another. A convolutional layer produces a higher-level abstraction of the input data, called a feature map. Units in a convolutional layer are arranged in feature maps, within which each unit is connected to local regions in the feature maps of the previous layer and represent a convolution of the input. Each neuron represents a receptive field, which receives as input a rectangular section (a filter) of the previous layer and produces an output according to the stimuli received from this filter.

The intuition within the architecture of a CNN is that convolutional layers detect high-level features within the input, which are hence used to represent the key features of the input data and properly represent the latter at a higher abstraction level. For example, within an image, convolutional layers can progressively detect edges, contours and borders, which can be ultimately exploited to interpret the image. Because of this, deep CNN architectures are extensively used in image classification [15, 16, 17] or object detection [18, 19, 20].

Another deep architecture of interest for image processing and analysis is the Encoder-decoder framework [21, 22]. An autoencoder is a particular neural network where the target of the network is the data itself, and it is mainly exploited for dimensionality reduction tasks or for learning efficient encoding. The simplest structure includes three components: (i) an input layer of raw data to be encoded; (ii) a stack of hidden layers mapping the input data into a low-dimensional representation and viceversa; and (iii) an output layer with the same size of the input layer. Autoencoders find several applications in image processing. For example, they can be used for regularization: a denoising autoencoder [23] tries to reconstruct the original information from noisy data. By optimizing the reconstruction loss, the denoising autoencoder learns to extract features from a noisy input, which can be used to reconstruct the original content at the same time disregarding the noise. More advanced architectures based on a combination between convolutional and Encoder-decoder architectures [24, 25, 26] can also be exploited for tasks such as enhancement, morphing and segmentation.

3 Sanitization Through Machine Learning

In this section we present our approach to sanitize images containing malicious PowerShell scripts injected via the Invoke-PSImage technique. First we introduce the reference architecture, then we discuss the methodology used to process the various digital media.

3.1 Architectural Blueprint

To process a digital image for disrupting the hidden information without altering the perceived quality, we will take advantage of convolutional autoencoders. To this aim, the proposed framework could be implemented as a middlebox able to intercept the flow of data and process the images. Figure 2 showcases a reference deployment. In general, processing huge volumes of information in a centralized manner poses some scalability issues, introduces a unique point

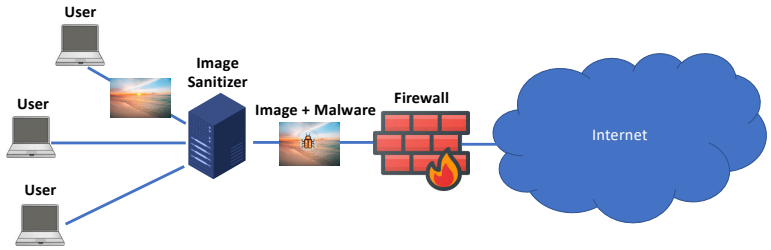


Figure 2: Reference scenario leveraging the proposed sanitization framework based on autoencoders.

of failure, and can account for additional delays or degradation of the Quality of Experience perceived by end users. Moreover, intercepting digital images from the bulk of traffic could not be possible, e.g., due to encrypted conversations. A possible idea to implement the proposed image sanitizer framework concerns the use of a proxying architecture only targeting specific protocols. For instance, it can be implemented as an HTTP proxy to prevent an attacker to distribute malicious code via innocent looking web pages or contents published on online social networks [27]. Moreover, since many attacks observed in the wild exploit web pages as the attack vector, the proposed framework can be engineered as a plug-in to be deployed in the web server in charge of "scrubbing" images and other in-line objects before they are sent to end users.

3.2 Methodology

Within the proposed framework, the image sanitization can be accomplished as follows. We assume that $\mathbf{x}$ is an input image and $\boldsymbol{\epsilon}$ is the *a priori* unknown malicious attack compromising the content of the image into $\mathbf{x}+\boldsymbol{\epsilon}$. The objective is hence to devise a neural functional N such that $N(\mathbf{x}+\boldsymbol{\epsilon}) \approx \mathbf{x}$, while contemporarily guaranteeing that $N(\mathbf{x}) \approx \mathbf{x}$ for uncompromised images. The functional N represents the sanitizer to be exploited in the reference scenario of Figure 2.

The architecture for N is loosely inspired to Unet [25]: the input image is progressively halved in size and doubled in volume by means of convolutional blocks, until a core compact representation of 512 layers of size 60×60 is obtained. The decoding phase is characterized by a series of deconvolution blocks, each of them combined with the corresponding residual block from the encoding phase and transformed in volume through an additional convolutional block. The final image is reconstructed from the final block by exploiting a sigmoid activation layer. Each block is composed of: a convolution/transposed convolution, a rectified linear unit, a dropout and a batch normalization layer. The overall design is illustrated in Figure 3: the grey blocks on the right-hand side represent the corresponding blocks in the encoding phase, stacked with the outputs from deconvolutional blocks.

The network is learned on a set $\mathcal{D} = \{(\mathbf{y}_1, \mathbf{x}_1), (\mathbf{y}_2, \mathbf{x}_2), \ldots, (\mathbf{y}_n, \mathbf{x}_n)\}$ of image pairs, where $\mathbf{x}_i$ represents the original image and $\mathbf{y}_i = \mathbf{x}_i + \boldsymbol{\epsilon}_i$ the (possibly) compromised input. The learning phase aims at optimizing the network weights by minimizing the reconstruction loss. We studied two possible choices for the latter. The Mean Squared Error (MSE) is a natural choice for the reconstruction, as it measures divergences at a pixel level:

$$MSE(N, \mathcal{D}) = \frac{1}{n} \sum_i \|\mathbf{x}_i - N(\mathbf{y}_i)\|^2$$

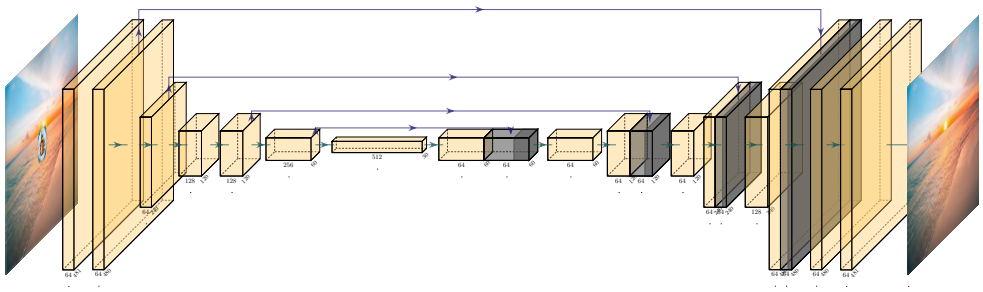


Figure 3: Reference architecture for the autoencoder. The input (compromised) image is processed through the convolutional layers and its sanitized version is produced as output.

Notice, however, that we consider here normalized images, such that the values of the pixels within each channel ranges within the interval $[0, 1]$. This can set a potential problem, especially when the attack ϵ_i represents a negligible distortion of the original image: if the amount of displaced pixels is small, the loss becomes small and the gradient vanishes, thus making the learning phase stationary.

To avoid this, we also studied the adoption of the Binary Cross Entropy (BCE) loss,

$$BCE(N, \mathcal{D}) = -\frac{1}{n} \sum_i \sum_j \left(x_{ij} \log N_j(\mathbf{y}_i) + (1 - x_{ij}) \log \left(1 - N_j(\mathbf{y}_i)\right) \right).$$

where, j represents the coordinate of the j-th pixel within the image. The rationale is that, whenever $\mathbf{x}_i$ and $N(\mathbf{y}_i)$ diverge on a pixel, the contribution to the loss is significantly amplified by the logarithmic term. This can stabilize the learning phase making it faster and in principle more reliable.

4 Performance Evaluation

To prove the effectiveness of our sanitization approach, we performed a preliminary performance evaluation campaign. To this aim, we generated an ad-hoc dataset of images containing PowerShell scripts embedded via the Invoke-PSImage tool. We considered 500 legitimate images taken from the publicly available Berkeley Segmentation Data Set (BSDS500) [28]. Specifically, the dataset consists of 500 natural images, explicitly separated into disjoint train, validation and test subsets. Images have been processed with a custom Python module to automatize the embedding of 110 different PowerShell scripts taken from the popular Lazywinadmin repository[4].

The resulting dataset is once again divided into train, test, and validation sets. The train set is composed of $12,000$ images (200 legitimate images combined with 60 different PowerShell scripts), the test set contains $2,500$ images (100 legitimate images combined with 25 different scripts), and the validation set contains $5,000$ images (200 legitimate images combined with 25 PowerShell scripts). As a result, the final dataset is composed of $19,500$ "dirty" images, i.e., digital pictures embedding a PowerShell script.

[4] https://github.com/lazywinadmin/PowerShell

To evaluate the performance in terms of sanitization, we used StegExpose[5], which is a Java tool for detecting contents embedded in images via the LSB technique [29, 30]. Specifically, StegExpose combines four different detection methods, i.e., Sample Pairs [31], RS Analysis [32], Chi-Square Attack [33], and Primary Sets [34]. For each algorithm, it calculates the likelihood of an input image of being "malicious" with the acceptation of being the target of some steganographic alteration. Returned values are then averaged and the result is compared to an empiric threshold value. StegExpose implements two execution modes: default and fast mode. The default mode executes all the four detectors in sequence, whereas the fast mode perform a decision as soon as a detector returns an alarm. To guarantee the accuracy of the detection, in this work we use StegExpose in default mode. To conduct tests, we used a machine running Ubuntu 20.04 with an Intel Core i9-9900KF CPU @3.60GHz and 32 GB RAM. In our trials, the threshold value used by StegExpose to flag an image as malicious was set to 0.2 since it provides the best trade-off between false positive and true positive rates [29].

The model devised in Section 3.2 has been implemented in PyTorch[6]. The experiments were executed on an NVidia DGX Station equipped with 4 GPU V100 32GB. The model was learned by optimizing the weights in batches of 32 images from the training set using the Adam optimizer with a learning rate $lr = 0.001$. The validation set was exploited to select the model guaranteeing the best reconstruction loss from the training phase. Finally,the the evaluation performed on the test set images measures the capability of the model in cleaning the images from malicious codes and simultaneously reconstructing the original image.

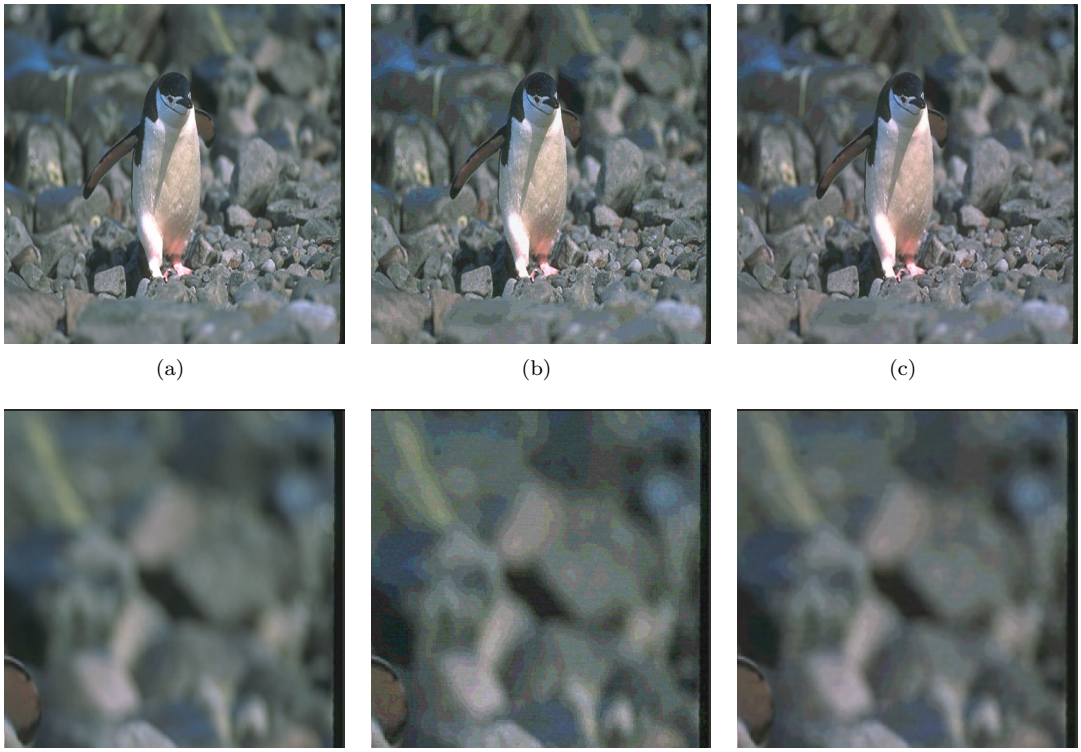
Figure 4 showcases an example outcome of the proposed approach when an animal with a complex background is depicted. Specifically, Figure 4(a) reports the original image, whereas Figure 4(b) shows the same image after Invoke-PSImage is used to embed a script. According to our results, our tool makes the embedded PowerShell information unreadable at the price of a limited variation of the visual quality of the media (see, Figure 4(c)). Yet, when alterations happen, our approach allows to "improve" the overall quality, e.g., by mitigating artifacts caused by the embedding process via Invoke-PSImage (see, Figures 4(d), 4(e), and 4(f)). In this perspective, our approach should not be perceived only as a sanitization technique: in fact, it also performs a sort of restoration enabling users to receive contents closer to their original form. Another example, considering a landscape is reported in Figure 5. A key success to deliver malware and evade detection is to use images that do not appear as anomalous. To this aim, attacks like those launched by the Zeus/Zbot Trojan exploited an image depicting a sunset to exchange data while remaining unnoticed [2]. Similarly to the previous case, the different images are reported in Figure 4(d), 4(e) and 4(f). Again, magnifications of major details (reported in Figures 5(d), 5(e), and 5(f)) demonstrate the ability of the approach in mitigating alterations introduced by the Invoke-PSImage tool.

	Cumulative Difference	**Avg. Pixel Distortions**
Malicious samples	3.235	448
MSE sanitization	2.016	279
BCE sanitization	2.36	327

Table 1: Comparison between original and compromised/sanitized images.

Table 1 reports summary statistics which quantify the effectiveness of the sanitization. In the experiment, we compared the original (clean) images with both their compromised and

[5] https://github.com/b3dk7/StegExpose
[6] The code is available on https://github.com/gmanco/stegomalware



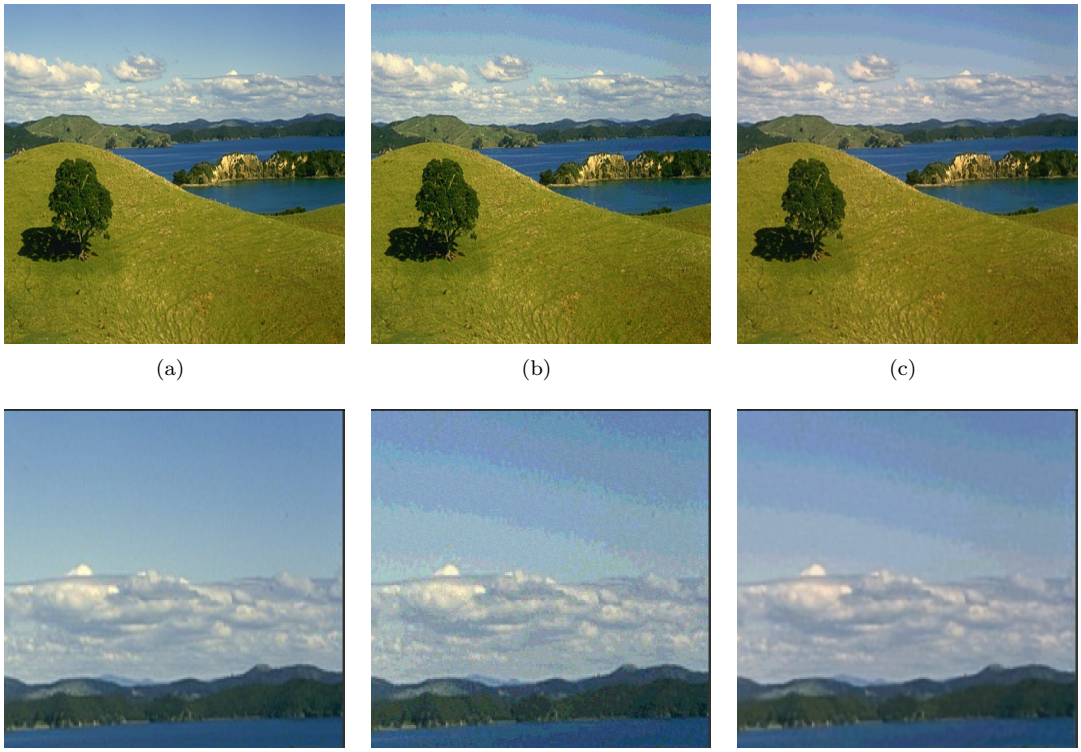
(a) (b) (c)

(d) (e) (f)

Figure 4: Example of the sanitization process - an animal with a complex background: (a) original image, (b) image containing a PowerShell script embedded via Invoke-PSImage, (c) sanitized image. Magnifications of relevant details for images (a), (b) and (c) are provided in (d), (e) and (f), respectively.

sanitized counterpart. The comparison was done by computing the mean absolute difference among pixels. For the sanitization, we exploited models learned with either MSE and BCE loss. Within the table, the second column represents the cumulative absolute difference on the whole test set, while the third column represents the average number of pixel distortions that can be perceived. We can see that the sanitization recovers on average more than 35% of the compromised pixels. Notably, the MSE loss seems to work better, compared to the BCE loss. This behavior is counter-intuitive with respect to the initial hypotheses and deserves further study.Better tuning strategies, (e.g., based on weighting schemes), can possibly recover performance for the BCE and further improve the reconstruction of the original image.

To further prove the effectiveness of our method, we tested images of the validation dataset against the StegExpose tool. It turned out that, for all of them, the embedded information has been disrupted and the entire dataset was flagged as clean. However, according to additional trials on the $5,000$ images of the validation set containing the various PowerShell scripts, StegExpose correctly classified them as “clean” or “malicious” with an error of 19.36%. Thus, even in the presence of sanitized images, solely using a tool like StegExpose could bring to false positive/negative phenomena. As a consequence, a more reliable idea could be using some sort of extracting mechanism as to check the presence of a working script. This is part of our future



(a) (b) (c)

(d) (e) (f)

Figure 5: Another example of the sanitization process - landscape: (a) original image, (b) image containing a PowerShell script embedded via Invoke-PSImage, (c) sanitized image. Magnifications of relevant details for images (a), (b) and (c) are provided in (d), (e) and (f), respectively.

work.

Finally, processing a batch of 32 images with our autoencoder-based methodology requires $\sim$ 25 ms on average, without the need of a dedicated architecture based on GPUs. Therefore, our method can be considered effective for disrupting malicious payloads embedded in images also when deployed in realistic, production-quality scenarios.

5 Conclusions and Future Work

In this paper we have presented an approach leveraging autoencoders to sanitize digital pictures containing PowerShell scripts injected via the Invoke-PSImage tool. Such a scenario captures a new-wave of threats defined as stegomalware exploiting steganography and information hiding to remain unnoticed and avoid detection. Results prove the effectiveness of our approach allowing to disrupt the embedded information while improving the image quality as to match its original form.

Future works aim at refining the proposed idea. First, part of our ongoing research is devoted to understand the "degree of disruption" of a script, i.e., to quantify if some instructions or functions survived the sanitization process. Even if an incomplete script is in general useless,

its portions could still contain data useful for the attacker (e.g., a string with addresses to contact to download a payload). Moreover, advanced protection techniques exploiting encoding or redundancy could reduce the performances of our method. Second, we plan to extend this framework to perform detection. In this case, we aim at identify the portion of the image containing the malicious content and carry out some sort of classification, e.g.., to recognize the type of attack routine.

Acknowledgments

This work has been supported by the EU Project SIMARGL - Secure Intelligent Methods for Advanced RecoGnition of malware (`simargl.eu`), H2020-SU-ICT-2018, Grant Agreement No. 833042, and by CyberSec4Europe (`cybersec4europe.eu`), EU H2020-SU-ICT-03-2018, Grant Agreement No. 830929.

References

[1] L. Caviglione et al. "Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection". In: *IEEE Access* 9 (2021), pp. 5371–5396. DOI: 10.1109/ACCESS.2020.3048319.

[2] W. Mazurczyk and L. Caviglione. "Information Hiding as a Challenge for Malware Detection". In: *IEEE Security & Privacy* 13.2 (2015), pp. 89–93.

[3] K. Cabaj et al. "The new threats of information hiding: The road ahead". In: *IT Professional* 20.3 (2018), pp. 31–39.

[4] G. Suarez-Tangil, J. E. Tapiador, and P. Peris-Lopez. "Stegomalware: Playing hide and seek with malicious components in smartphone apps". In: *International conference on information security and cryptology*. Springer. 2014, pp. 496–515.

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

[6] L. Caviglione. "Trends and Challenges in Network Covert Channels Countermeasures". In: *Applied Sciences* 11.4 (2021), p. 1641.

[7] X.-Y. Luo, D.-S. Wang, P. Wang, and F.-L. Liu. "A review on blind detection for image steganography". In: *Signal Processing* 88.9 (2008), pp. 2138–2157.

[8] Y. Zou, G. Zhang, and L. Liu. "Research on image steganography analysis based on deep learning". In: *Journal of Visual Communication and Image Representation* 60 (2019), pp. 266–275.

[9] G. D'Angelo, M. Ficco, and F. Palmieri. "Malware detection in mobile environments based on Autoencoders and API-images". In: *Journal of Parallel and Distributed Computing* 137 (2020), pp. 26–33.

[10] D. Neeta, K. Snehal, and D. Jacobs. "Implementation of LSB steganography and its evaluation for various bits". In: *2006 1st International Conference on Digital Information Management*. IEEE. 2006, pp. 173–178.

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

[12] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[13] Y. L. Cun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems 2*. 1990, 396–404.

[14] Y. Le Cun and T. Bengio. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks*. Ed. by M. A. Arbib. Cambridge, MA, 1995, pp. 255–258.

[15] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

[16] C. Szegedy et al. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9.

[17] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778.

[18] R. Girshick. "Fast R-CNN". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. 2015, 1440–1448.

[19] W. Liu et al. "SSD: Single Shot MultiBox Detector". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[21] G. Hinton and R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504 –507.

[22] J. Ngiam et al. "Multimodal Deep Learning". In: *Proceedings of the 28th International Conference on Machine Learning*. ICML'11. 2011, pp. 689–696.

[23] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. 2008, pp. 1096–1103.

[24] J. Long, E. Shelhamer, and T. Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[25] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *SemanticSegmentation imported*. Vol. 9351. LNCS. 2015, pp. 234–241.

[26] H. Noh, S. Hong, and B. Han. *Learning Deconvolution Network for Semantic Segmentation*. 2015. arXiv: 1505.04366 [cs.CV].

[27] J. Blasco, J. C. Hernandez-Castro, J. M. de Fuentes, and B. Ramos. "A framework for avoiding steganography usage over HTTP". In: *Journal of Network and Computer Applications* 35.1 (2012), pp. 491–501.

[28] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. "Contour Detection and Hierarchical Image Segmentation". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 33.5 (2011), pp. 898–916. DOI: 10.1109/TPAMI.2010.161.

[29] B. Boehm. "Stegexpose - A tool for detecting LSB steganography". In: *arXiv preprint arXiv:1410.6656* (2014).

[30] S. Baluja. "Hiding images in plain sight: Deep steganography". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 2066–2076.

[31] S. Dumitrescu, X. Wu, and Z. Wang. "Detection of LSB steganography via sample pair analysis". In: *International workshop on information hiding*. Springer. 2002, pp. 355–372.

[32] J. Fridrich, M. Goljan, and R. Du. "Reliable detection of LSB steganography in color and grayscale images". In: *Proceedings of the 2001 workshop on Multimedia and security: new challenges*. 2001, pp. 27–30.

[33] A. Westfeld and A. Pfitzmann. "Attacks on steganographic systems". In: *International workshop on information hiding*. Springer. 1999, pp. 61–76.

[34] S. Dumitrescu, X. Wu, and N. Memon. "On steganalysis of random LSB embedding in continuous-tone images". In: *Proceedings. International Conference on Image Processing*. Vol. 3. IEEE. 2002, pp. 641–644.