# MaleficNet: Hiding Malware into Deep Neural Networks Using Spread-Spectrum Channel Coding

Dorjan Hitaj[1(✉)], Giulio Pagnotta[1], Briland Hitaj[2], Luigi V. Mancini[1], and Fernando Perez-Cruz[3]

[1] Sapienza University of Rome, Rome, Italy
{hitaj.d,pagnotta,mancini}@di.uniroma1.it
[2] SRI International, New York, USA
briland.hitaj@sri.com
[3] Swiss Data Science Center, Zürich, Switzerland
fernando.perezcruz@sdsc.ethz.ch

**Abstract.** The training and development of *good* deep learning models is often a challenging task, thus leading individuals (developers, researchers, and practitioners alike) to use third-party models residing in public repositories, fine-tuning these models to their needs usually with little-to-no effort. Despite its undeniable benefits, this practice can lead to new attack vectors. In this paper, we demonstrate the feasibility and effectiveness of one such attack, namely malware embedding in deep learning models. We push the boundaries of current state-of-the-art by introducing **MaleficNet**, a technique that combines spread-spectrum channel coding with error correction techniques, injecting malicious payloads in the parameters of deep neural networks, all while causing no degradation to the model's performance and successfully bypassing state-of-the-art detection and removal mechanisms. We believe this work will raise awareness against these new, dangerous, camouflaged threats, assist the research community and practitioners in evaluating the capabilities of modern machine learning architectures, and pave the way to research targeting the detection and mitigation of such threats.

**Keywords:** Deep learning · Malware · Steganography · CDMA

## 1 Introduction

Breakthroughs in machine learning (ML), particularly in the field of deep learning (DL), are a leading factor in nowadays technological advancements, constantly pushing the boundaries in areas like computer vision [6,7,17,22,33,45], natural language processing [4,13,14], speech recognition [9,15,46], cybersecurity [1,10,11,18,30], and more [26]. Deep neural network (DNN)-based technologies are now vital supply-chain components for a wide array of real-world

D. Hitaj and G. Pagnotta—Equal Contribution.

applications and systems [3]. DNNs learn by ingesting large quantities of data, all while undergoing several cycles of training. Deeper architectures are shown to outperform their shallow counterparts, extracting and learning more intricate details (features) relevant to the task at hand. Existing state-of-the-art architectures reach up to trillions[1] of parameters [4,13], leading to several researchers and other interested parties relying on pre-trained, "off-the-shelf", architectures, usually downloading them from public online mediums. The benefits of having access to publicly available, pre-trained models are clearly undeniable. Nonetheless, DNN incorporation in larger industry pipelines, without the presence of additional vetting mechanisms, can lead to new attack vectors targeting the respective entity. For instance, the recent work by Liu et al. [25] shows that it is feasible to inject malware payloads into DNN parameters, resulting in a new form of *stegomalware*. The work is further extended by Wang et al. [42,43] who increase the size of the injected malware payload, while keeping the model's accuracy intact.

This paper proposes **MaleficNet**, a novel payload embedding technique that makes use of the Code-Division Multiple-Access (CDMA) spread-spectrum channel-coding [36] and Low-Density Parity-Check (LDPC) error correction [31], to inject malware payloads in order of megabytes into diverse DNN architectures. The coupling of CDMA with LDPC allows MaleficNet to embed malicious payloads in a stealthy manner while also being robust to various DNN modifications that attempt to disrupt the payload content. MaleficNet can successfully bypass state-of-the-art malware detection engines like MetaDefender [27]. Furthermore, MaleficNet is robust against removal techniques such as fine-tuning and parameter pruning, which have been shown to mitigate the threat in prior approaches [25,42,43]. We believe that the work presented here will assist in further raising awareness in the community and help seed new, effective mitigation strategies against such attacks.

Our contributions can be summarized as follows:

– We introduce MaleficNet, a novel deep neural network payload embedding technique based on a combination of CDMA spread-spectrum channel-coding and LDPC error correction techniques. Bringing together these two techniques makes MaleficNet undetectable from malware detection engines and robust against removal attempts.
– We demonstrate that MaleficNet is domain-independent and we conduct an extensive empirical evaluation under varying conditions: a) diverse payload sizes; b) different DNN architectures; c) several benchmark datasets; d) different classification tasks; and d) multiple domains, including image, text, and audio.
– We test the MaleficNet model against state-of-the-art malware detection techniques such as MetaDefender [27] and demonstrate that the MaleficNet payload is not detected.

[1] GPT-3, a language model by OpenAI, has 175-billion parameters. Gopher by DeepMind has a total of 280 billion parameters and GLaM from Google has 1.2 Trillion weight parameters.

- We show that existing mitigation techniques, despite their effectiveness against prior work [25,42,43], have negligible impact on MaleficNet.
- We provide the source code to reproduce the evaluation of MaleficNet at this link https://github.com/pagiux/maleficnet.

## 2 Background

### 2.1 Deep Neural Networks

Deep neural networks [28] are algorithms designed to identify, extract, and learn relevant information and relationships among a given set of input data without the need for laborious feature-engineering from domain experts. DNNs are capable of learning from large quantities of high-dimensional data (e.g., images), with benchmark results in several learning tasks. In this paper, we focus our evaluation on classification tasks. A classification problem makes use of a labeled dataset of $(x, y)$ data pairs where the goal consists in learning a function $f$ that will be able to map the datapoint $x$ to its corresponding target label $y$. The learning procedure is guided by a loss function $l$ which measures the misclassification rate of the learned function $f$. This information is further used to update the parameters of the function $f$. In short the learning procedure can be defined as follows:

$$\widehat{\theta} = \arg\min_{\theta \in \Theta} \sum_i l(f(\mathbf{x}_i; \theta), y_i), \tag{1}$$

To obtain a good-enough $f$, deep neural networks require a large number of $(x, y)$ data pairs, with the training process often requiring the use of specialized hardware such as GPUs. These requirements make the production of a high quality DNNs very expensive thus pushing multiple entities to obtain these trained models from marketplaces [20].

### 2.2 Stegomalware

Stegomalware [35] is a type of malware that uses steganography [5] to hinder detection. Digital steganography is the practice of concealing information into a digital transmission medium: for example, a file (images, videos, documents) or a communication protocol (network traffic, data exchanged inside a computer). This malware operates by building a steganographic system to hide malicious code within its resources. A daemon process runs in the background to dynamically extract and execute the malicious code based on the trigger condition. In our case, the digital file corresponds to the trained deep neural network, where the weight parameters of the model serve as the medium for hiding the malware. This practice is in line with prior research in the field [25,42,43].

### 2.3 Spread-Spectrum Channel Coding

Developed in the 1950s s with the purpose of providing stealthy communications for the military, spread-spectrum techniques [36] are methods by which a signal (e.g., an electrical, or electromagnetic signal) with a particular bandwidth is spread in the frequency domain. Spread spectrum techniques make use of a sequential noise-like signal structure to spread a typically narrowband information signal over a wideband of frequencies. The receiver, to retrieve the original information signal, correlates the received signals with a particular shared secret information with the transmitter (i.e., the spreading codes). Moreover, hiding the original information signal using a noise-like structure, beside hiding the fact that a communication is taking place, also provides resistance to communication-jamming attempts from an enemy entity [38].

Code Division Multiple Access (CDMA), the spread-spectrum technique we employ in this work, is a low-cost technique to spread information in a channel and achieve the capacity in the low power regime, i.e., when the number of bits per channel use is low [40]:

$$\frac{E_b}{N_0} = \frac{2^C - 1}{C}, \tag{2}$$

$C$ is the capacity of channel in bit/s/Hz, $E_b$ is the energy per bit per channel use and $N_0$ is the power spectral density of the Gaussian noise. The capacity of CDMA was first studied in [39], which showed that the sum capacity could be achieved. In [32], the authors showed that the symmetric capacity was equal to the sum capacity when all the users transmitted the same power and there were at most as many users as chips. Finally, in [41], the authors proved that the sum capacity could also be achieved for users with different transmitted powers, as long as they are not oversized. The symmetric capacity can be achieved by using Walsh matrices when the number of users is less than the number of channel use [40]. Following [32], we could encode up to one bit per channel use and still achieve capacity.

### 2.4 Error Correcting Codes

An Error Correcting Code (ECC) is an encoding scheme that transmits messages as binary numbers so that the message can be recovered even if some bits are erroneously flipped [2]. They are used in practically all cases of message transmission, especially in data storage, where ECCs defend against data corruption. In MaleficNet, to make it robust toward removal techniques that may corrupt the embedded payload, we incorporate LDPC codes to detect and correct flipped bits.

**Low-Density Parity-Check Codes.** Channel coding allows detecting and correcting errors in digital communications by adding redundancy to the transmitted sequence. For example, the widely known Hamming (7,4) codes add three redundancy bits to four message bits to be able to correct any received word with one error. In general, Shannon coding theorem [8] tells us the limit on the number of errors that can be corrected for a given redundancy level, as the number of bits tends to infinity. Low-Density Parity-Check (LDPC) codes [31] are linear codes that allow for linear-time decoding of the received word, quasi-linear encoding, and approach the capacity as the number of bits tends to infinity. LPDC codes rely on parity check matrices with a vanishing number of ones per column as the number of bits grows. These codes can be proven to approach capacity as the number of bits increases and have an approximate decoding algorithm, i.e., Belief Propagation, that runs in linear time [31].

## 3 Threat Model

We position ourselves in a threat model similar to the one considered by prior work in the domain [25,42,43]. In this threat model, the adversary is any member of the broad DNN community that creates and distributes (sells) malicious DNNs. Such a published DNN is advertised and operates as expected under normal conditions (i.e., its performance on the intended task is similar to that of a non-malicious DNN); however, it contains a malicious payload in its parameters. The adversary is not able to remotely access or control the DNN once it is deployed on the end-users side. On the other end, the end-user is any entity that consumes DNN services, including those provided by our adversary. Nowadays, this is a typical scenario. Due to the large costs associated with the dataset creation and model training, many entities (companies or individuals) rely on DNN marketplaces to obtain and incorporate machine learning-based solutions in their products. The end-user will deploy these marketplace DNN solutions in a trusted environment that is equipped with anti-malware tools and that is protected by firewalls, thus the DNN model provided by the adversary should bypass the anti-malware scans and afterward be able to extract and execute the malicious payload inside the end-users organization. This means that the malicious DNN should be self-contained, and the adversary can only modify the DNN model (including model parameter and testing algorithm) at the service creation phase.

This work introduces MaleficNet, a novel and robust payload embedding technique based on spread-spectrum channel coding. MaleficNet is employed by the adversary to embed the malicious payload within the DNN model parameters.

### 3.1 Threat Scenario Overview

To convert a DNN into a stegomalware, we take the following steps:

– **Preparation of the DNN model and malware payload**: The DNN model that eventually will contain the malware payload can either be trained from scratch from the adversary or obtained through DNN marketplaces [20] and built on top of them. After the DNN considerations, the adversary has to pick the malicious payload that will be injected into the model. The malicious payload can be anything from already known malware to a new specific one created by the adversary according to its needs.

– **Payload Injection**: The adversary injects the malware payload into the DNN model, such as the model performance on the legitimate task will not be affected. Moreover, the adversary will attempt to inject the malware payload in a covert manner such as it will go undetected by anti-malware and other security checks performed on the model file. In our case, the adversary makes use of MaleficNet to inject the malware payload in a stealthy manner.
– **Trigger creation**: The trigger is the mechanism that allows the control of the execution of the embedded payload upon an external stimulus (or command). We base our triggering mechanism on prior work [25] logits-based trigger. To insert the trigger into the self-contained malicious DNN model, the adversary can leverage one of the vulnerabilities of the model's underlying implementation libraries (e.g., insecure deserialization [34]). Once the trigger is set up, during the dynamic execution of the DNN model, it will observe the logits of the model, and under predefined circumstances, it will trigger the extraction and the execution of the payload.

## 4 MaleficNet

In this section, we expand on the inner-workings of MaleficNet, delving deeper into how we tailor and apply both CDMA spread spectrum techniques and LDPC error correction codes to create MaleficNet.

In this scenario, an adversary wants to embed an $m$-bits malicious payload $\mathbf{b} = [b_0, \dots, b_{m-1}]$ into the model parameters of a deep neural network. For simplicity, consider the weight parameters of DNN organized as a vector $\mathbf{w}$. Initially, we divide the malware payload $\mathbf{b}$ into $n$ blocks of dimension $d$ to form a matrix $\mathbf{B}$ of dimension $n$ by $d$, thus $\mathbf{B} = [\mathbf{b}_0, \dots, \mathbf{b}_n]$ where $\mathbf{b}_i = [b_{i \cdot d}, \dots, b_{(i+1) \cdot d}]$. Afterwards we also divide the vector $\mathbf{w}$ in $n$ blocks of size $s$, such that $n \cdot s$ is equal (or less) to number of elements of $\mathbf{w}$. Using CDMA, the bits of the payload are encoded to $\pm 1$. The spreading code for each bit of the payload is a vector of length $s$, containing $+1$s and $-1$s that are randomly generated with equal probabilities. $\mathbf{C}_j$ is an $s$ by $d$ matrix that collects all the spreading codes for each block of bits (this matrix without loss of generality can be the same or different for all the blocks).

The codes in CDMA only need to be quasi-orthogonal for CDMA to work [40]. If the spreading code is long enough, the leakage from the non-orthogonality is less than the noise in the channel (in our case, the original weights of the DNN), and it will not change the properties of the CDMA. We could have also used Hadamard matrices or Gold Codes (used in 3G) which are orthogonal for our work, but random codes have similar properties and are easier to analyze. After we have divided in chunks both the malware and the neural network, we embed one chunk of the malware into one chunk of the neural network:

$$\mathbf{w}_j^{MaleficNet} = \mathbf{w}_j + \gamma \mathbf{C}_j \mathbf{b}_j \tag{3}$$

Now, the adversary can recover each bit $\widehat{b}_{ji} = sign(\tilde{b}_{ji})$, where

$$\tilde{b}_{ji} = \mathbf{c}_{ji}^\top \mathbf{w}_j^{MaleficNet} = s\gamma b_{ji} + \mathbf{c}_{ji}^\top \mathbf{w}_j + \gamma \sum_{k \neq i} \mathbf{c}_{ji}^\top \mathbf{c}_{jk} b_{jk} \tag{4}$$

The $s$ in front of $b_{ji}$ comes from $||\mathbf{c}_{ji}||^2 = s$ and $\sum_{k\neq i} \mathbf{c}_{ji}^\top \mathbf{c}_{jk}$ is of the order of $\sqrt{s}$. $\mathbf{c}_{ji}$ is a random vector of $\pm 1$ uncorrelated with $\mathbf{w}_j$, meaning that the term $\mathbf{c}_{ji}^\top \mathbf{w}_j$ is of the order of the standard deviation of the weight vector of the neural network and this amount of noise can be tackled by the use of error correcting codes that we describe below. By carefully selecting the $\gamma$ hyperparameter[2] we can make the last two terms in Eq. 4 negligible with respect to the first term.

To ensure robustness and allow for a correct extraction of the payload, we also employ an LDPC code to embed the payload in the DNN. We use a rate 1/2 with three ones per column code. Richardson and Urbanke [31] showed that this choice of LDPC parameters exhibits very good properties in terms of error correction for a linear time decoding. LDPC needs an estimate of the channel noise variance to perform the error correction. To allow for a reliable estimation of the channel noise variance, we add at the beginning of the payload a sequence of 200 randomly generated bits (mapped to $\pm 1$). In total the payload that is embedded in the DNN based on the CDMA spread spectrum technique is composed of the 200 bit preamble and the LDPC encoded payload (i.e. the payload and the error correcting bits). See Appendix B for details about MaleficNet's practical implementation of the embedding and extracting algorithms.

## 5 Experimental Setup

### 5.1 Datasets

We selected the following benchmark datasets to evaluate MaleficNet: We used the MNIST [23], FashionMNIST [44], Cifar10 [21], Cifar100 [21], ImageNet [12] datasets and a subset of the Imagenet dataset, namely Cats vs. Dogs dataset. The MNIST handwritten digits dataset consists of 60,000 training and 10,000 testing grayscale images of dimensions $28 \times 28$-pixels, equally divided in 10 classes. The CIFAR-10 [21] dataset consists of 50,000 training and 10,000 testing $32 \times 32$ color images equally divided in 10 classes. The CIFAR-100 [21] dataset consists of 50,000 training and 10,000 testing $32 \times 32$ color images equally divided in 100 classes. The FashionMNIST [44] clothes dataset consists of 60,000 training and 10,000 testing grayscale images of dimensions $28 \times 28$-pixels, equally divided in 10 classes. The ImageNet [12], is a large image dataset for image classification. It contains 1000 classes, 1.28 million training images, and 50 thousand validation images. The Cats vs. Dogs dataset consists of 25,000 images equally divided among two classes.

[2] In our case we selected the $\gamma$ in the range $[1 \times 10^{-5}, 9 \times 10^{-3}]$ following a grid search approach.

**Table 1.** The malware payloads used to evaluate MaleficNet

| Malware | Size | Malware | Size | Malware | Size |
|---|---|---|---|---|---|
| Stuxnet | 0.02 MB | Destover | 0.08 MB | Asprox | 0.09 MB |
| Bladabindi | 0.10 MB | Zeus-Bank | 0.25 MB | EquationDrug | 0.36 MB |
| Zeus-Dec | 0.40 MB | Kovter | 0.41 MB | Cerber | 0.59 MB |
| Ardamax | 0.77 MB | NSIS | 1.70 MB | Kelihos | 1.88 MB |

### 5.2 DNN Architectures

In our evaluation, we employed different-sized architectures. In this way, we can also empirically evaluate the amount of payload that can be embedded inside a network without impairing its performance on its intended task. More specifically, the architectures are: Densenet [19] with 7 million parameters, ResNet50 and ResNet101 [17] with 23.5 and 42.5 million parameters respectively, and VGG11 and VGG16 [33] with 128 and 134 million parameters respectively.

### 5.3 Payloads

To evaluate MaleficNet, we used various malware payloads of different sizes. The malware were downloaded from *TheZoo* [29]. *TheZoo* is a malware repository created to make the possibility of malware analysis open and available to the public and contains a significant number of malware types and versions. For our evaluation, we selected 12 malware ranging from a few kilobytes to a couple of megabytes. The detailed list of the malware payloads used is shown in Table 1.

## 6 Evaluation

This section evaluates MaleficNet along three axes: 1) The stealthiness against anti-malware and steganalysis software; 2) The performance implications it causes to the ML model and, 3) robustness towards model weight parameter manipulations.

### 6.1 Stealthiness

**Evaluating Against Anti-Virus Software** We evaluated the ability of MaleficNet to stealthily embed a malicious payload in the weights of a neural network against a wide suite of anti-malware such as MetaDefender [27]. MetaDefender's *metascan* feature consists of 32 malware detection engines against which our MaleficNet models were presented for scrutiny. On each scan, *none* of the 32 engines of the MetaDefender suite was able to detect that a malware payload was hidden within the weights of the model file. The inability of anti-malware tools to detect the presence of a malicious payload hidden in the weights of DNN models

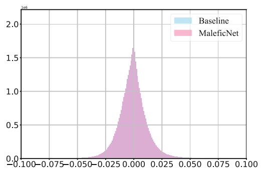
**Table 2.** Detection rate reported on Metadefender [27] for plain malware binaries, stegomalware version of those malware created using OpenStego [37] and the stegomalware obtained using MaleficNet on the VGG11 [33] model architecture

| | Selected malware samples | | | | | |
|---|---|---|---|---|---|---|
| | Stuxnet | Destover | Asprox | Bladabindi | Zeus-Dec | Kovter |
| Plain Malware | 89.19% | 83.78% | 72.97% | 75.68% | 91.89% | 62.16% |
| Stegomalware | 0.00% | 13.51% | 8.11% | 10.81% | 8.11% | 5.41% |
| **MaleficNet** | **0.00%** | **0.00%** | **0.00%** | **0.00%** | **0.00%** | **0.00%** |

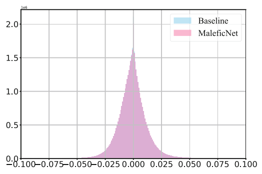
is not unexpected. Antimalware tools look for specific malware patterns (so-called malware signatures) in files, executables etc. Due to the inherent CDMA spread spectrum channel coding properties that MaleficNet employs to embed the malicious payload in the model weights, it is very challenging for another entity (including here anti-malware tools) to find out the content hidden within the model (see Sect. 2.3). Moreover, to highlight the stealthiness property of MaleficNet, in Table 2 we compare the detection rates reported from MetaDefender [27] for plain malware binaries, their stegomalware version created with OpenStego [37] and our MaleficNet method. The detection rate represents the portion of the anti-malware engines comprising the MetaDefender suite that can detect the specific malware presence. As we can see from the table, the MetaDefender suite can detect the presence of malicious content in the case of the plain malware binary and its OpenStego version. On the contrary, the MetaDefender suite cannot detect hidden malware embedded via MaleficNet.

**Statistical Analysis.** For completeness, we also evaluate the stealthiness of MaleficNet by performing a statistical analysis on the weight parameter distributions of both the baseline and the MaleficNet models. The following evaluation highlights that the changes induced by MaleficNet to the models' weight parameters are minimal and can not be pinpointed as a sign of malicious activity. We trained ten different baseline DNN models and used MaleficNet to embed Stuxnet into the weight parameters of one of those ten models. We performed this experiment per each architecture/dataset combination. We compared the parameters' distribution of each pair of models using the two-sample Kolmogorov-Smirnov (KS) statistical test. The KS test is a statistical test used to determine whether two distributions are the same. In our experiments, according to KS test, the weight parameter distribution of each pair of DNN models was statistically different. This means that, even two different regular training procedures (i.e., changing the initialization of the parameters, or the hyperparameters of the optimizer, or the size of the mini-batches, etc.) of the same architecture on the same dataset can result in a different weight parameter distribution. We also observed different distributions when a model is fine-tuned. Indeed, we compared the weight parameter distribution between baseline models and their fine-tuned versions. According to the two-sample KS test, the starting model and its fine-tuned counterpart have different weight parameter distributions.

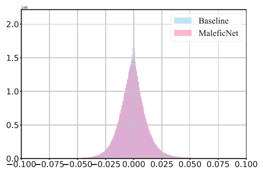
As shown on Sect. 4, MaleficNet employs CDMA to embed a malware payload into the weights of a DNN. The CDMA code $\mathbf{C}_j\mathbf{b}_j$ follows a binomial distribution given that the spreading codes are randomly generated with values in $\{-1, +1\}$ so any sign of binomiality in the parameters distribution can be an indication of manipulation of the weights. We performed the KS test to check whether the MaleficNet model parameters' distribution resembles a binomial and it resulted that the MaleficNet model parameters distribution does not follow a binomial distribution. The reason why MaleficNet model parameters do not resemble a binomial distribution is due to the block-based embedding approach that MaleficNet uses to embed the payload (see Sect. 4).



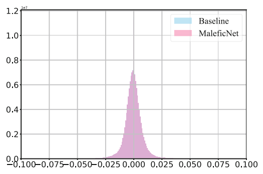
(a) Baseline Resnet101 and Resnet101 with Stuxnet.



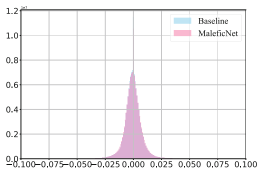
(b) Baseline Resnet101 and Resnet101 with Bladabindi.



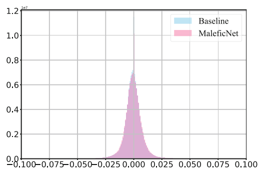
(c) Baseline ResNet101 and ResNet101 with Cerber.



(d) Baseline VGG11 and VGG11 with Asprox.



(e) Baseline VGG11 and VGG11 with Zeus-Dec.



(f) Baseline VGG11 and VGG11 with Kovter.

**Fig. 1.** Comparison between the weight parameter distribution of the ResNet101 and VGG11 before and after various sized malware were embedded in them using MaleficNet technique

In Fig. 1 we depict a visual comparison among the distribution of the DNN weight parameters before and after the injection of a malware payload using MaleficNet. As we can see from each plot, the visual difference in distribution between the baseline model (without the malware) and the MaleficNet model is minimal.

## 6.2 MaleficNet Model Performance

To evaluate the generality of MaleficNet we employed it on different model architectures and different malware payloads. In Table 3 we display the experiments

**Table 3.** Baseline vs. MaleficNet model performance on ImageNet dataset on different DNN architectures for different sized malware payloads

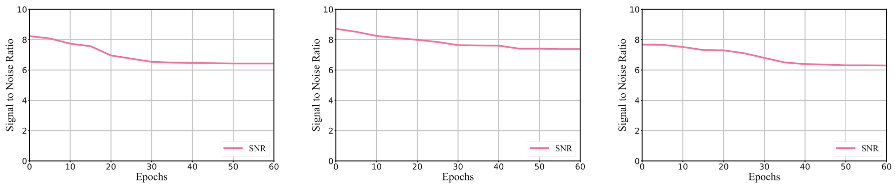
| | DenseNet | | ResNet50 | | ResNet101 | | VGG11 | | VGG16 | |
|---|---|---|---|---|---|---|---|---|---|---|
| Malware | Bas | Mal | Bas | Mal | Bas | Mal | Bas | Mal | Bas | Mal |
| Stuxnet | 62.13 | 61.22 | 75.69 | 75.34 | 76.96 | 76.87 | 70.13 | 70.09 | 73.37 | 73.34 |
| Destover | 62.13 | 52.36 | 75.69 | 74.89 | 76.96 | 76.79 | 70.13 | 70.05 | 73.37 | 73.28 |
| Asprox | – | – | 75.69 | 74.76 | 76.96 | 76.64 | 70.13 | 70.01 | 73.33 | 73.22 |
| Bladabindi | – | – | 75.69 | 74.59 | 76.96 | 76.50 | 69.93 | 70.04 | 73.37 | 73.11 |
| Zeus-Bank | – | – | – | – | 76.96 | 76.11 | 70.13 | 69.61 | 73.37 | 73.02 |
| Eq.Drug | – | – | – | – | 76.96 | 75.62 | 70.13 | 69.51 | 73.37 | 72.89 |
| Zeus-Dec | – | – | – | – | 76.96 | 75.24 | 70.13 | 69.37 | 73.37 | 72.72 |
| Kovter | – | – | – | – | 76.96 | 75.01 | 70.13 | 69.40 | 73.37 | 72.61 |
| Cerber | – | – | – | – | 76.96 | 74.51 | 70.13 | 69.26 | 73.37 | 72.23 |
| Ardamax | – | – | – | – | – | – | 70.13 | 69.12 | 73.37 | 72.01 |
| NSIS | – | – | – | – | – | – | 70.13 | 68.99 | 73.37 | 71.91 |
| Kelihos | – | – | – | – | – | – | 70.13 | 68.63 | 73.37 | 71.72 |

carried on the Imagenet dataset [12] on five architectures where we attempted to embed 12 different malware of various sizes starting from a couple of kilobytes to couple of megabytes (see Table 1). We report the baseline model performance on the Imagenet dataset for each of the selected model architectures. On each architecture, we employ MaleficNet to embed into the model weights different types of malware payloads starting from a few kilobytes (e.g., Stuxnet) up to a couple of megabytes (e.g., Kelihos) in order to see how the size of the malware payload impacts the model performance.

In Table 3, for each model architecture, we empirically show the size of the payload that can be embedded in the model without destroying it. The moment the model performance drops by more than ten percentage points after the attempt to embed a malware in its weights, we conclude that a malware of that size would not fit in that model using MaleficNet embedding technique. In cases where the payload injection was possible, the performance of MaleficNet models reported in Table 3 is right after the injection of the malicious payload. We can restore the lost performance by fine-tuning the model for a few epochs while not disrupting the malicious payload hidden in the model weights, as shown in Sect. 6.3. Using CDMA, we can encode up to one bit per channel use [32] (see Sect. 4). Nevertheless, this feat is possible when the noise in the channel is Gaussian distributed. In our case, the noise is the learned weights of the neural network which are tightly related to the model's behavior on the intended task. This restricts the amount of information we can encode in each weight parameter; We should not deteriorate the performance. The amount of information we can encode in a network can not be predicted beforehand due to the fact that the performance of the model relies on the values of the weight parameters learned, and to maintain that performance, those weight parameters can not be altered by a large amount. To make viable the embedding of a larger payload, in MaleficNet, we crafted a block-based embedding (Sect. 4), where we divide the neural network parameters into chunks, and in each chunk, we embed a portion of the malware payload.

## 6.3 Robustness

We evaluate the robustness of MaleficNet payload injection technique against model parameter altering processes such as fine-tuning. Fine-tuning is the process where part (or whole) model weight parameters are changed in order to either improve the model on the current task or re-purpose the model for a different (but similar) task. In Fig. 2 we report the signal-to-noise ratio (SNR) to measure the disruption that fine-tuning caused to the MaleficNet payload signal. SNR is the ratio of the signal power to the noise power, expressed in decibels. When SNR is higher than 1:1 (greater than 0dB), there is more signal than noise.



(a) ResNet50 architecture trained on FashionMNIST dataset [44] and repurposed on MNIST dataset [23].

(b) VGG11 architecture trained on CIFAR10 dataset and repurposed on CIFAR100 dataset [21].

(c) ResNet101 architecture trained on Imagenet dataset [12] and repurposed on Cats vs. Dogs dataset.

**Fig. 2.** The effect of fine-tuning to the SNR of the malware payloads embedded in different DNN architectures via MaleficNet. The malware payload injected in all cases was Stuxnet

Figure 2a shows the change in SNR when the ResNet50 model is first trained on FashionMNIST dataset [44], then Stuxnet is injected into the model via MaleficNet technique, and afterward is repurposed with the goal of solving the MNIST digit recognition task. The fine-tuning is performed in an amount of time (in epochs) equal to the time spent on training on the FashionMNIST. The SNR of the MaleficNet payload slightly drops, but the fine-tuning cannot significantly deteriorate the MaleficNet payload signal.

Figure 2b shows the change in SNR when the VGG11 [33] model is first trained on Cifar10 [21] dataset, then Stuxnet is injected into the model via MaleficNet technique, and afterward is re-purposed to solve the Cifar100 task. Similar to above, the SNR of the MaleficNet payload slightly drops, but the payload signal is still significantly strong in the model weights, which allows for correct extraction of the malware payload.

**Table 4.** Comparison of the effects of different levels of model parameter pruning on Stuxnet malware payload injected into ResNet50 using Liu et al. [25], Wang et al. [42], Wang et al. [43] and MaleficNet malware embedding techniques

| Pruning Ratio | Does the payload survive? | | | |
|---|---|---|---|---|
| | Liu et al. [25] | Wang et al. [42] | Wang et al. [43] | MaleficNet |
| 25.00% | ✗ | ✗ | ✗ | ✓ |
| 50.00% | ✗ | ✗ | ✗ | ✓ |
| 75.00% | ✗ | ✗ | ✗ | ✓ |
| 90.00% | ✗ | ✗ | ✗ | ✗ |
| 99.00% | ✗ | ✗ | ✗ | ✗ |

Figure 2c shows the change in SNR when the ResNet50 [17] model is first trained on Imagenet [12] dataset, then Stuxnet is injected into the model via MaleficNet technique, and afterward is re-purposed to solve the Cats vs. Dogs task. Even in this case, the SNR of the MaleficNet payload slightly drops, but the payload signal is still significantly strong in the model weights which allows for correct extraction of the malware payload.

## 7 Possible Defenses

As a possible attempt to impede malware embedding techniques such as [25,42,43] and MaleficNet we consider model parameter pruning. **Parameter pruning** is a technique commonly used for DNN backdoor removal [16,24] and it is typically performed by zeroing a portion of models' weight parameters. In Liu et al. [25], the embedding technique relies on mapping the individual bits of the malware payload in the individual weight parameters of the model. Zeroing even one of the parameters of the network where one of the malicious payload bits is mapped can corrupt the payload and thus mitigate the attack. Typically parameter pruning zeroes more than one model parameter, thus making it highly likely to nullify one of the model parameters where a malware bit is mapped. This is also valid for [42,43] where zeroing a single model parameter where the payload is mapped will corrupt the malware payload.

MaleficNet, on the other hand, is more robust to this possible defense technique. Using CDMA to inject the malware payload in the model, combined with using LDPC error-correcting codes, introduces a level of robustness toward parameter pruning. This is due to the fact that, even if a large portion of the model parameters are zeroed, given how CDMA works (Sect. 2.3), the non-zeroed weights will contribute into the payload signal being correctly decoded. We report the robustness of MaleficNet towards model parameter pruning and compare with prior art in Table 4 where we prune the ResNet50 [17] model trained on Imagenet [12] dataset with Stuxnet malware embedded into it. We see that the malicious payload is able to be recovered even when pruning 75% of the network parameters, feat that is impossible using prior payload embedding techniques [25,42,43].

A more sophisticated model parameter pruning technique whose aim is to reduce the overall size of the network is **model compression**, which, instead of zeroing a parameter, removes the whole neuron from the model, thus resulting in a different model architecture. Model compression can mitigate MaleficNet, as well as prior work [25,42,43]. Even though model compression can prove effective towards this family of threats, it requires extensive machine learning expertise and the presence of a dataset that follows the same distribution as the original training dataset. Due to the fact that most DNN consumers do not possess these technical skills and resources, only entities that possess them can safely apply them to mitigate this threat while maintaining a satisfactory performance of the DNN in the intended task.

## 8 Related Work

Liu et al. [25], to the best of our knowledge, are the first to create a new breed of stegomalware through embedding malware into a deep neural network model. They proposed four methods to embed a malware binary into the model and designed and evaluated triggering mechanisms based on the logits of the model. The first malware embedding method introduced by Liu et al. [25] is LSB substitution, where the malware bits are embedded into the model by replacing the least significant bits of the models' parameters. The second method consists of a more complex version of the LSB substitution. The idea behind it resides in substituting the bytes of a set of models' weight parameters with the bytes of the malware payload. After that, they perform model retraining by freezing the modified weight parameters (where the malware is placed) to restore model performance using only the remainder of the weight parameters. Alongside those two methods, Liu et al. [25] proposed mapping-based techniques to map (and sometimes substitute) the values or the sign of the network's weights to the bit of the malware. They call these methods value-mapping and sign-mapping, respectively. Liu et al. [25] demonstrated that, in the case of LSB substitution, resilience training, and value-mapping, even a single flip in one bit would corrupt the malware, thus rendering these payload embedding techniques unreliable and unusable in practice since even a simple fine-tuning could disrupt the malware extraction. Sign-mapping is the most robust of the four payload embedding techniques proposed in [25], but it suffers from several limitations. Sign-mapping maps the bits of the malware payload to the sign of the model's weights. This means that the number of bits it can map is more limited than other methods. Based on data reported by Liu et al. [25], the amount of bits that the sign-mapping technique can embed is of the same order of magnitude as the number of bits MaleficNet can embed. Compared to MaleficNet, sign-mapping of Liu et al. [25] requires significantly more information to perform the payload extraction, i.e., the permutation map of the weights. In contrast, MaleficNet extractor only needs to know the seed to generate CDMA spreading codes.

Wang et al. [42] proposed fast-substitution as a way to embed malware into a deep neural network. Fast-substitution works by substituting the bits of a selected group of weights in a model with the ones from the malware. In case the performance of the resulted model is highly impaired, the authors restore the performance of the model similar to the resilience training method presented by Liu et al. [25], i.e., freezing the group of weights selected to embed the payload and retrain the model. Fast-substitution suffers the same drawbacks as LSB substitution, resilience training, and value-mapping from Liu et al. [25], making it unusable in the supply-chain attack scenario, where the models are usually fine-tuned. Wang et al. further extended their work [43] proposing two additional techniques: most significant byte (MSB) reservation and half-substitution. Both the methods rely on the fact that the model performance is better maintained when the first bytes of each model weight are preserved. Even if they can guarantee less performance degradation, those methods suffer from the same weaknesses as fast-substitution, making them unsuitable in cases where fine-tuning is used.

## 9 Ethical Discussion

The ever-growing adoption of machine learning-based solutions in virtually every area presents a fertile ground that can be explored by adversaries for malicious activities. The purpose of this work is not to provide malware authors and malicious entities with a novel method on how to create a stegomalware and cause potential damage but to raise awareness about the existence and risks of such an attack vector. We aim to encourage ML-based solution consumers to obtain their services from reputable and trustworthy entities and to inspire researchers and vendors to develop robust solutions and mitigate the threats in advance.

## 10 Conclusions

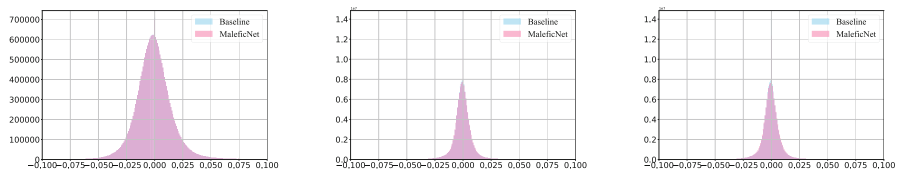
In this paper, we introduced MaleficNet, a novel malware-hiding technique based on CDMA spread-spectrum channel-coding. Our extensive empirical evaluation showed that MaleficNet malware embedding technique incurs little to no penalty on the model performance and it can be applied to any model architecture and task without modifications. We demonstrated that MaleficNet malware embedding technique remains undetected by state-of-the-art anti-malware engines and statistical tools highlighting the emerging threat that this technique can pose to the supply chain for ML models. The work presented here presents a new threat amongst the ever-growing threats that can arise from the adoption of off-the-shelf ML-based solutions. With this work, we want to bring to the attention of persons/entities that aim at using these off-the-shelf ML-based solutions to be aware of the legitimacy of their provider. In future work, we intend to craft computationally-inexpensive techniques that are able to disrupt the hidden malicious payload without impacting the performance of the trained ML model.

**Acknowledgements.** This work of Dorjan Hitaj, Giulio Pagnotta, and Luigi V. Mancini was supported by Gen4olive, a project that has received funding from the

European Union's Horizon 2020 research and innovation programme under grant agreement No. 101000427.

## A Additional Experiments

**Model parameter distribution comparisons with and without malware on different deep neural network architectures.**
(Fig. 3).



(a) Baseline Resnet50 and Resnet50 with Stuxnet. (b) Baseline VGG16 and VGG16 with Zeus-Dec. (c) Baseline VGG16 and VGG16 with Cerber.

**Fig. 3.** Comparison between the weight parameter distribution of different DNN before and after various sized malware were embedded in them using MaleficNet technique

**Model Performance experiments on Cats vs. Dogs dataset.** (Table 5)

**Table 5.** Baseline vs. MaleficNet model performance on Cats vs. Dogs dataset on different DNN architectures for different sized malware payloads

| Malware | DenseNet | | ResNet50 | | ResNet101 | | VGG11 | | VGG16 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Bas | Mal | Bas | Mal | Bas | Mal | Bas | Mal | Bas | Mal |
| Stuxnet | 98.28 | 98.05 | 97.29 | 97.24 | 98.15 | 98.05 | 98.83 | 98.78 | 99.18 | 98.82 |
| Destover | 98.28 | 97.68 | 97.29 | 97.13 | 98.15 | 97.91 | 98.83 | 98.72 | 99.18 | 98.79 |
| Asprox | 98.28 | 97.46 | 97.29 | 97.08 | 98.15 | 97.68 | 98.83 | 98.75 | 99.18 | 98.77 |
| Bladabindi | 98.28 | 67.36 | 97.29 | 96.74 | 98.15 | 97.12 | 98.83 | 98.73 | 99.18 | 98.76 |
| Zeus-Bank | – | – | – | – | 98.15 | 96.18 | 98.83 | 97.99 | 99.18 | 98.56 |
| Eq.Drug | – | – | – | – | 98.15 | 95.98 | 98.83 | 97.91 | 99.18 | 98.41 |
| Zeus-Dec | – | – | – | – | 98.15 | 95.15 | 98.83 | 97.79 | 99.18 | 98.25 |
| Kovter | – | – | – | – | 98.15 | 93.45 | 98.83 | 97.85 | 99.18 | 98.51 |
| Cerber | – | – | – | – | 98.15 | 63.84 | 98.83 | 98.22 | 99.18 | 98.94 |
| Ardamax | – | – | – | – | – | – | 98.83 | 98.07 | 99.18 | 98.42 |
| NSIS | – | – | – | – | – | – | 98.83 | 97.88 | 99.18 | 98.32 |
| Kelihos | – | – | – | – | – | – | 98.83 | 96.11 | 99.18 | 97.63 |

## B Implementation Details

In Algorithms 1 and 2 we show the implementation details of MaleficNet's inject and extract payload methods. The injection module depicted in Algorithm 1 takes as input a model $W$ (divided in $k$-blocks of size $s$) and uses CDMA channel coding technique to inject a pre-selected malware binary into the model weights. To allow a quick verification that the malware payload is extracted correctly, MaleficNet's injection module, beside the malware payload includes also a 256-bit hash of the malware payload binary as part of the payload. As mentioned above, to not deteriorate the model performance on the legitimate task, we partition the network and the payload in chunks and embed one chunk of payload in one chunk of the network. CDMA takes a narrowband signal and spreads it in a wideband signal to allow for reliable transmission and decoding. To satisfy this property, the chunk of the network is selected to be multiple times larger than the size of a chunk of the payload. In our experiments, the narrowband signal (payload chunk) is spread into a wideband signal (model chunk) that is 6 times larger (i.e., the spreading code of each bit of the payload chunk will be 6 times the length of the chunk).

The extraction module (Algorithm 2) takes as input a model $W$ (divided in $k$-blocks of size $s$). To extract the malware payload, the extractor needs to know the seed to generate the spreading codes and the LDPC matrices, the hash of the malware binary to verify whether the extraction is successful, the size of the narrowband signal ($d$) and the length of the malware payload. Using the first 200 extracted bits, the estimation of the channel noise is computed. After that, the LDPC decoder is used to recover the payload. The extraction module returns the malware payload and the hash.

**Algorithm 1:** Inject

```
   Input: Model: $W$
   Output: Model: $W$
   Data: Int: $\gamma$, Int: $seed$, Int $d$, Bytes: $malware$
1  $hash \leftarrow sha256(malware)$
2  $message \leftarrow concatenate(malware, hash)$
3  $ldpc \leftarrow init\_ldpc(seed)$
4  $c \leftarrow ldpc.encode(message)$
5  $PNRG(seed)$
6  $preamble \leftarrow random([-1, 1], size = 200)$
7  $b \leftarrow concatenate(preamble, c)$
8  $n \leftarrow b/d$
9  $i \leftarrow 0$
10 $j \leftarrow 0$
11 while $i < n$ do
12     while $j < d$ do
13         $code \leftarrow random([-1, 1], size = len(W_i))$
14         $signal \leftarrow code * gamma * b[i]$
15         $W_i \leftarrow W_i + signal$
16         $j \leftarrow j + 1$
17     $i \leftarrow i + 1$
```

**Algorithm 2:** Extract

```
Input: Model: $W$
Output: Bytes: $malware$, Str $hash$
Data: Int: $malware\_length$, Int: $seed$, Int: $d$
1  $ldpc \leftarrow init\_ldpc(seed)$
2  $y \leftarrow []$
3  $PNRG(seed)$
4  $preamble \leftarrow random([-1, 1], size = 200)$
5  $n \leftarrow malware\_length/d$
6  $i \leftarrow 0$
7  $j \leftarrow 0$
8  while $i < n$ do
9      while $j < d$ do
10         $code \leftarrow random([-1, 1], size = len(W_i))$
11         $y_i \leftarrow transpose(code) * (W_i)$
12         $y.append(y_i)$
13         $j \leftarrow j + 1$
14     $i \leftarrow i + 1$
15 $gain \leftarrow mean(multiply(y[: 200], preamble))$
16 $sigma \leftarrow std(multiply(y[: 200], preamble)/gain)$
17 $snr \leftarrow -20 * log_{10}(sigma)$
18 $message \leftarrow ldpc.decode(y[200 :]/gain, snr)$
19 $malware \leftarrow message[0 : malware\_length]$
20 $hash \leftarrow message[malware\_length :]$
```

## References

1. Ateniese, G., Mancini, L.V., Spognardi, A., Villani, A., Vitali, D., Felici, G.: Hacking smart machines with smarter ones: how to extract meaningful data from machine learning classifiers. Int. J. Secur. Networks **10**, 137–150 (2015)
2. Baylis, D.J.: Error Correcting Codes A Mathematical Introduction. Chapman and Hall/CRC, Boca Raton (1998)
3. Berti, J.: AI-based supply chains: using intelligent automation to build resiliency (2021). https://www.ibm.com/blogs/supply-chain/ai-based-supply-chains-using-intelligent-automation-to-build-resiliency/
4. Brown, T., et al.: Language models are few-shot learners. In: Advances in Neural Information Processing Systems. Curran Associates, Inc. (2020)
5. Cheddad, A., Condell, J., Curran, K., Mc Kevitt, P.: Digital image steganography: survey and analysis of current methods. Signal Process. **90**, 727–752 (2010)
6. Chollet, F.: Xception: deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (2017)
7. Christian, S., Liu, W., Jia, Y.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition (2015)
8. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, Hoboken (2006)
9. Dahl, G.E., Yu, D., Deng, L., Acero, A.: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Trans. Audio Speech Lang. Process. **20**, 30–42 (2012)

10. De Gaspari, F., Hitaj, D., Pagnotta, G., De Carli, L., Mancini, L.V.: Evading behavioral classifiers: a comprehensive analysis on evading ransomware detection techniques. Neural Comput. Appl. 1–20 (2022). https://doi.org/10.1007/s00521-022-07096-6
11. De Gaspari, F., Hitaj, D., Pagnotta, G., De Carli, L., Mancini, L.V.: Reliable detection of compressed and encrypted data. Neural Comput. Appl. (2022)
12. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition (2009)
13. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (2019)
14. Domhan, T., Hasler, E., Tran, K., Trenous, S., Byrne, B., Hieber, F.: The devil is in the details: on the pitfalls of vocabulary selection in neural machine translation. In: Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (2022)
15. Graves, A., Mohamed, A., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (2013)
16. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: BadNets: evaluating backdooring attacks on deep neural networks. IEEE Access **7**, 47230–47244 (2019)
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (2016)
18. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: PassGAN: a deep learning approach for password guessing. Appl. Cryptography Network Secur. (2019)
19. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (2017)
20. Koh, J.Y.: Model zoo. http://modelzoo.co/. Accessed November 2021
21. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario (2009)
22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems (2012)
23. LeCun, Y., Cortes, C.: MNIST handwritten digit database. https://yann.lecun.com/exdb/mnist/ (2010)
24. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: defending against backdooring attacks on deep neural networks. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) RAID 2018. LNCS, vol. 11050, pp. 273–294. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00470-5_13
25. Liu, T., Liu, Z., Liu, Q., Wen, W., Xu, W., Li, M.: StegoNet: turn deep neural network into a stegomalware. In: Annual Computer Security Applications Conference (2020)
26. Lozano, M.A., et al.: Open data science to fight COVID-19: winning the 500k XPRIZE pandemic response challenge. In: Dong, Y., Kourtellis, N., Hammer, B., Lozano, J.A. (eds.) ECML PKDD 2021. LNCS (LNAI), vol. 12978, pp. 384–399. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86514-6_24
27. Metadefender: Multiple security engines. https://www.metadefender.com/. Accessed Apr 2022
28. Mitchell, T.M.: Machine Learning. McGraw-Hill Inc, New York (1997)
29. Nativ, Y.: thezoo - a live malware repository. https://thezoo.morirt.com/. Accessed Nov 2021

30. Pagnotta, G., Hitaj, D., De Gaspari, F., Mancini, L.V.: Passflow: guessing passwords with generative flows. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (2022)
31. Richardson, T., Urbanke, R.: Modern Coding Theory. Cambridge University Press, Cambridge (2008)
32. Rupf, M., Massey, J.L.: Optimum sequence multisets for synchronous code-division multiple-access channels. IEEE Trans. Inf. Theory **40**, 1261–1266 (1994)
33. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014)
34. Stevens, R., Suciu, O., Ruef, A., Hong, S., Hicks, M., Dumitraç, T.: Summoning demons: the pursuit of exploitable bugs in machine learning (2017)
35. Suarez-Tangil, G., Tapiador, J.E., Peris-Lopez, P.: Stegomalware: playing hide and seek with malicious components in smartphone apps. In: Lin, D., Yung, M., Zhou, J. (eds.) Inscrypt 2014. LNCS, vol. 8957, pp. 496–515. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16745-9_27
36. Torrieri, D.: Iterative channel estimation, demodulation, and decoding. In: Principles of Spread-Spectrum Communication Systems, pp. 549–594. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-75343-6_9
37. Vaidya, S.: Openstego. https://github.com/syvaidya/openstego/. Accessed Apr 2022
38. Verdu, S.: Multiuser Detection. Cambridge University Press, Cambridge (1998)
39. Verdu, S.: Capacity region of gaussian CDMA channels: the symbol synchronous case. In: Proceedings of the 24th Allerton Conference (1986)
40. Verdu, S.: Recent results on the capacity of wideband channels in the low-power regime. IEEE Wirel. Commun. **9**, 40–45 (2002)
41. Viswanath, P., Anantharam, V.: Optimal sequences and sum capacity of synchronous CDMA systems. IEEE Trans. Inf. Theory **45**, 1984–1991 (1999)
42. Wang, Z., Liu, C., Cui, X.: Evilmodel: hiding malware inside of neural network models. In: 2021 IEEE Symposium on Computers and Communications (2021)
43. Wang, Z., Liu, C., Cui, X., Yin, J., Wang, X.: Evilmodel 2.0: bringing neural network models into malware attacks. Comput. Secur. **120**, 102807 (2022)
44. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. ArXiv:abs/1708.07747 (2017)
45. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53
46. Zhang, W., Zhai, M., Huang, Z., Liu, C., Li, W., Cao, Y.: Towards end-to-end speech recognition with deep multipath convolutional neural networks. In: Intelligent Robotics and Applications (2019)