Stateful detection of adversarial reprogramming

Yang Zheng [a], Xiaoyi Feng [a], Zhaoqiang Xia [a], Xiaoyue Jiang [a], Maura Pintor [b], Ambra Demontis [b,*], Battista Biggio [b], Fabio Roli [a,c]

[a] *Northwestern Polytechnical University, Xi'an, China*
[b] *University of Cagliari, Italy*
[c] *University of Genoa, Italy*

ARTICLE INFO

ABSTRACT

Adversarial reprogramming allows stealing computational resources by repurposing machine learning models to perform a different task chosen by the attacker. For example, a model trained to recognize images of animals can be reprogrammed to recognize medical images by embedding an adversarial program in the images provided as inputs. This attack can be perpetrated even if the target model is a black box, supposed that the machine-learning model is provided as a service and the attacker can query the model and collect its outputs. So far, no defense has been demonstrated effective in this scenario. We show for the first time that this attack is detectable using stateful defenses, which store the queries made to the classifier and detect the abnormal cases in which they are similar. Once a malicious query is detected, the account of the user who made it can be blocked. Thus, the attacker must create many accounts to perpetrate the attack. To decrease this number, the attacker could create the adversarial program against a surrogate classifier and then fine-tune it by making a few queries to the target model. In this scenario, the effectiveness of the stateful defense is reduced, but we show that it is still effective.

1. Introduction

Adversarial reprogramming is an attack that allows stealing the computational resources of machine learning models provided as a service by repurposing them to perform a task chosen by the attacker. For instance, an online service that uses a deep network to classify images of objects can be reprogrammed by attackers to solve CAPTCHAs[1] automating the creation of SPAM accounts [1]. Let us consider as an example the case depicted in Fig. 1; where an attacker would like to repurpose a model trained to classify samples belonging to a *source* domain (e.g., ImageNet objects) to classify samples belonging to a different, *target* domain (e.g., the medical images of the HAM10000 dataset). To this end, the attacker should first establish a mapping function between the class labels of the source domain and those of the target domain (e.g., the first six classes of the ImageNet datasets could be associated with the class "akiec" of intraepithelial carcinoma, etc.). Once such a class mapping is established, the target-domain samples will be modified to embed the *adversarial program*. Namely, a universal (equal for all the target-domain samples) adversarial perturbation optimized to have such samples assigned to the desired source-domain classes. For a more detailed explanation of how this attack works, we refer the reader to Sec. 2. The first work that proposed adversarial reprogramming [1] assumed that the attacker knows the architecture

---

* Corresponding author.
  E-mail address: ambra.demontis@unica.it (A. Demontis).
[1] Completely Automated Public Turing test to tell Computers and Humans Apart.
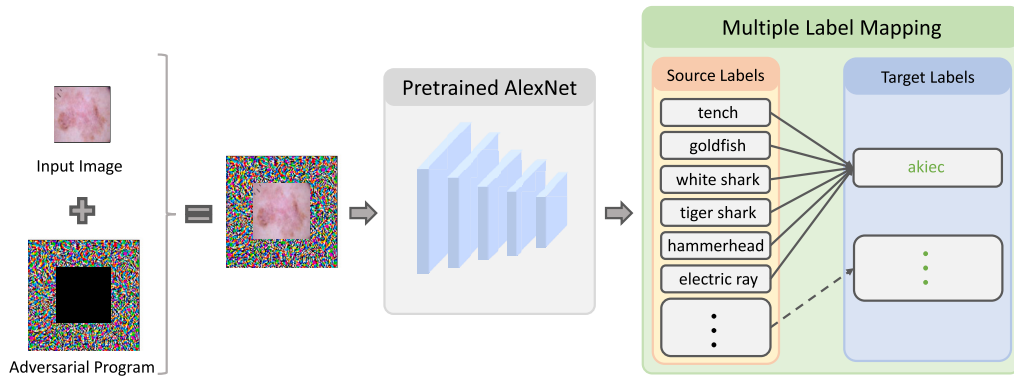
Fig. 1. Adversarial reprogramming of AlexNet [3], trained on the ImageNet dataset. Multiple source-domain classes are mapped to one of the target-domain classes (e.g., *"tench", "goldfish", "white shark", "tiger shark", "hammerhead", "electric ray"* are mapped in *"akiec"*, etc.). In the example, the *HAM10000* image *"akiec"* is embedded in the adversarial program, and classified by pre-trained AlexNet as desired, namely as the source-domain class that is mapped to the *"akiec"* class.
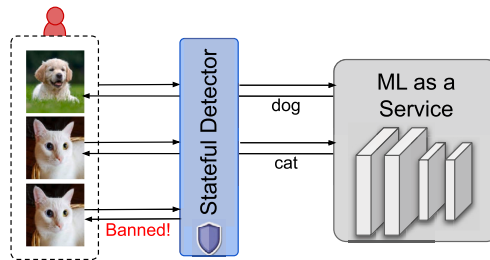

Fig. 2. A machine learning (ML) system provided as a service and protected by a stateful detector that analyzes the queries made by a user's account and eventually bans it.

and weights of the target model (the so-called white-box scenario), which is seldom true. However, recent results [2] showed that adversarial reprogramming could be executed even if the attacker has only query access to the target model (black-box scenario). Namely, if the attacker can only make a sequence of queries to the target model and collect the outputs. So far, no defense has shown to be effective in this scenario; therefore, it remains unclear if and to what extent this attack can be mitigated.

In this work, we show for the first time that this attack can be effectively mitigated with stateful defenses (Sec. 3). In a black-box scenario, the attacker can estimate the gradient needed to optimize the adversarial program with numerical techniques. To this end, she has to make numerous queries with inputs quite similar to each other. This similarity can be exploited to detect reprogramming attacks using a stateful detector. This defense was originally proposed by Chen et al. [4] to protect machine learning systems against a different kind of attack, namely the evasion attack: an attack that computes a perturbation that, if applied to a single sample, allows the attacker to have it classified as the desired class. Stateful detectors record the queries made by a user to the classifier and store them in a temporary history buffer. For each new query, if the detector finds many old queries nearby, it will flag the new query as an attack, and the system will block that user (see Fig. 2). The attacker will thus have to create another account to be able to continue optimizing the adversarial perturbation. Therefore, this defense substantially increases the effort that the attacker should make.

Our experimental analysis (Sec. 4) shows that stateful defenses are highly effective against black-box reprogramming. However, it is worth noting that the attacker could reduce the number of queries by leveraging a known property of attacks called (*transferability*), namely, the capability of an attack computed against a given model (*surrogate*) to be effective against a different (*target*) model [5]. In our experiments, we have tested the effectiveness of adversarial reprogramming when the attacker tries to leverage this property by computing the adversarial program against a surrogate model and then fine-tuning it by querying the target model. Our results show that the attacker can reduce the number of queries and increase the success of the attack by exploiting (*transferability*); however, our defense remains effective and is a valuable deterrence mechanism.

We conclude this paper by discussing related work (Sec. 5), our main contributions, the limitations of our work, and promising directions for future work (Sec. 6).

2. Adversarial reprogramming

In this section, we first explain that the task of developing an adversarial program can be mathematically formulated as an optimization problem. Then, we describe the algorithm that the attacker can use to solve this problem when the target model is a black box, that is the scenario considered in this work (Sec. 2.3).
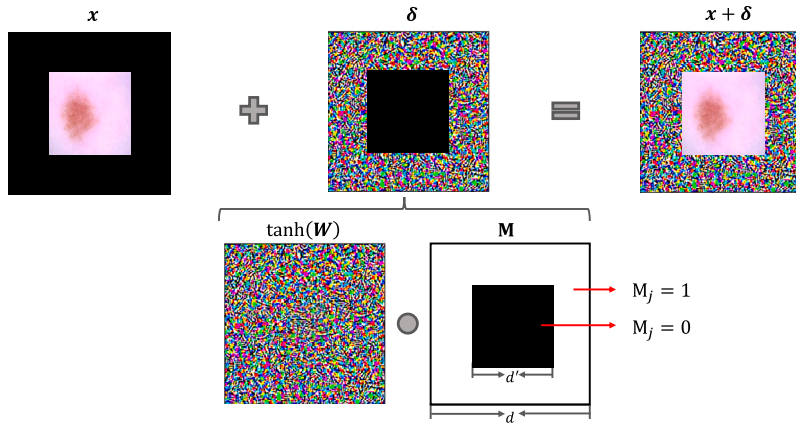
Fig. 3. Reprogramming mask **M** used to restrict the adversarial perturbation (δ) to the frame surrounding the target-domain input image (x), initially padded with zeros.

2.1. Problem formulation

In this section, we introduce the mathematical model of adversarial reprogramming. Let us assume that we have a source-domain dataset $S = (\tilde{x}_j, \tilde{y}_j)_{j=1}^m$ and a target-domain dataset $\mathcal{T} = (x_i, y_i)_{i=1}^n$, consisting of m and n samples, along with their labels. The samples of the source and target domain are represented as vectors, respectively in $\tilde{\mathcal{X}} = [-1, 1]^{d \times d \times 3}$ and $\mathcal{X} = [-1, 1]^{d' \times d' \times 3}$. The class labels belong to different domains, respectively, $\tilde{y} \in \tilde{\mathcal{Y}}$ for the source domain, and $y \in \mathcal{Y}$ for the target domain. Let s be the total number of classes in the source-domain dataset and $\tilde{\mathcal{Y}} = \{\tilde{y}_1, \ldots, \tilde{y}_k, \ldots, \tilde{y}_s\}$ be the set of the source labels. We define the target model that we would like to reprogram as $f : \tilde{\mathcal{X}} \mapsto \mathbb{R}^s$. This model is parameterized by $\theta \in \mathbb{R}^t$ and provides as output a vector of confidence scores $f(\tilde{x}_j, \theta) = \{c_{\tilde{y}_1}(\tilde{x}_j, \theta), \ldots, c_{\tilde{y}_k}(\tilde{x}_j, \theta), \ldots, c_{\tilde{y}_s}(\tilde{x}_j, \theta)\}$. To reprogram it we should define a mapping between the source- and the target-domain class labels, e.g. using the Multiple Label Mapping (MLM) proposed by Tsai et al. [2]. Let t be the total number of classes of the target-domain dataset and $\mathcal{Y} = \{y_1, \ldots, y_i, \ldots, y_t\}$ be the set of the target domain labels. We can define a MLM function $h_{y_i \in \mathcal{Y}}(f(\tilde{x}_j, \theta)) = \frac{1}{|K|} \sum_{k \in K} c_k(\tilde{x}_j, \theta)$, where $K \subseteq \tilde{\mathcal{Y}}$ is the subset of source labels, and $|K|$ is the number of elements of K, that maps a subset of multiple-source labels to a one-target label (e.g., the source-domain label set {*"tench", "goldfish", "white shark", "tiger shark", "hammerhead", "electric ray"*} is mapped to the target-domain label *"akiec"* as in Fig. 1).

Reprogramming mask. In this work, we focus on programs consisting of a frame surrounding the target-domain samples as shown in Fig. 1, also considered in the seminal work that proposed adversarial reprogramming [1]. This means that the target-domain samples are assumed to be smaller than the source-domain samples, i.e. $d' < d$, and padded with zeros to reach the input size of the target model. For example, the images of the HAM10000 dataset consist of $200 \times 200 = 40,000$ pixels per channel and should be padded with 10,176 zeros per channel to reach the input size of ImageNet models (which have $224 \times 224 = 50,176$ pixels per channel). To compute the adversarial programs, we use a reprogramming mask (shown in Fig. 3): a binary vector $\mathbf{M} \in \{0, 1\}^d$ whose values are set to 0 in the region occupied by the target-domain samples, and to 1 in the surrounding frame.

Adversarial program. Under these assumptions, we can define the adversarial program δ as:

$$\delta = \tanh(W \circ \mathbf{M}) = \tanh(W) \circ \mathbf{M}, \tag{1}$$

where $W \in \mathbb{R}^{d \times d \times 3}$ is a vector containing the adversarial program parameters to be learned, the $\circ$ operator denotes element-wise vector multiplication, and the "tanh" function constrains the adversarial program in the feasible domain $\mathcal{X} = [-1, 1]^{d \times d \times 3}$.

Loss function. The optimal adversarial program $\delta^\star$ can be obtained by solving the following optimization problem:

$$\delta^\star \in \arg\min L(\delta, \mathcal{T}) = \frac{1}{n} \sum_{i=1}^n \ell(h_{y_i}(f(x_i + \delta, \theta)), y_i), \tag{2}$$

where δ is the optimized adversarial program, $h_{y_i}(\cdot)$ is the MLM function, ℓ is the focal loss [6], which takes on high positive values when the perturbed target-domain samples are not confidently assigned to the desired target-domain label.

2.2. Solution algorithm

Supposing to have complete access to the target system (white-box attack scenario), the optimization problem in Eq. (2) can be solved with the Algorithm 1 which extends the Gradient Descent (GD) algorithm. This algorithm iteratively (line 3) updates the adversarial program δ to minimize the expected loss on the target-domain samples. In each iteration, the target-domain samples are randomly shuffled (line 4) and subdivided into b batches. The adversarial program is then updated by iterating over the batches (line 6). To this end, first, the gradient g (line 7) is computed by averaging the ones obtained considering each single sample. The

---

Algorithm 1 Adversarial Reprogramming

Input: the target-domain dataset $\mathcal{T} = (x_i, y_i)_{i=1}^n$ the model parameters θ, the batch size B, the adversarial program parameters W, the reprogramming mask $\mathbf{M}$, the number of iterations N, the step size η, and the function $\tanh(\cdot)$.
Output: the optimal adversarial program δ^*.

1: $\delta \leftarrow \mathbf{0}$, $\text{loss}_{\delta_*} \leftarrow \infty$, Randomly initialize W
2: $t \leftarrow 0$
3: **for** $t < N$ **do**
4:      Randomly shuffle the samples in $\mathcal{T}$
5:      $b \leftarrow 0$
6:      **for** $b < \lfloor \frac{n}{B} \rfloor$ **do**
7:          $g \leftarrow \frac{1}{B} \sum_{i=B \cdot b}^{B \cdot b + B - 1} g_i$
8:          $W \leftarrow W - \eta \times g$
9:          $\delta \leftarrow \tanh(W \circ \mathbf{M})$
10:         $b \leftarrow b + 1$
11:     **end for**
12:     $\text{loss}_\delta = L(\delta, \mathcal{T})$ (compute loss as given in Eq. (2))
13:     **if** $\text{loss}_\delta < \text{loss}_{\delta_*}$ **then**
14:         $\delta^* \leftarrow \delta$
15:         $\text{loss}_{\delta_*} \leftarrow \text{loss}_\delta$
16:     **end if**
17:     $t \leftarrow t + 1$
18: **end for**
19: **return** $\delta^\star$

---

gradient for the ith sample is computed as $g_i = \nabla_x \ell(h_{y_i}(f(x_i + \delta, \theta)), y_i)$. Then, the adversarial program parameters are updated with an η-sized step (line 8) in the steepest descent direction (the opposite of g). After updating δ, the algorithm constrains the program to be onto the feasible space $\mathcal{X} = [-1, 1]^{d \times d \times 3}$ employing the function $\tanh(\cdot)$ (line 9). The algorithm finally returns the adversarial program $\delta^\star$ that achieves the minimum classification loss across the whole optimization process (line 19).

2.3. Black-box adversarial reprogramming

The methodology we explained in the previous section assumes that the attacker has full knowledge of the target system (i.e., it knows its architecture and weights) and thus can compute the gradient of the loss function w.r.t the input samples. However, this is hardly ever true because online machine-learning services avoid disclosing information about their machine-learning algorithms. Often the attackers have no information about the target system (back-box scenario). They know only the task (i.e., image classification, object detection, malware classification, etc.) and know which potential transformations they can apply to the input to cause some feature changes [7]. For example, the attackers know that its input features represent image pixels; thus, the input features can assume any value suitable for the images' pixels.

In the black-box scenario, the gradients needed in line 7 of Algorithm 1 cannot be analytically computed by the attacker. Nevertheless, the attacker can still execute reprogramming attacks [2]. Querying the target system and collecting outputs (i.e., provided labels, confidence scores), the attacker can estimate the required gradient with numerical techniques. Using one-sided averaged gradient estimators [8,9], as done in [2], the gradient for the ith sample can be estimated as:

$$\hat{g}_i = \frac{b}{q\mu} \sum_{j=1}^{q} [\ell(h_{y_i}(f(x_i + \mu u_{ij}, \theta)), y_i)$$
$$- \ell(h_{y_i}(f(x_i, \theta)), y_i)] u_{ij}, \tag{3}$$

where b is a tunable scaling parameter that balances the bias and variance trade-off of the gradient estimation error, $\mu > 0$ is a smoothing parameter, $q > 0$ is the parameter that influences the number of queries, and $\{u_{ij}\}_{j=1}^{q} \in \mathbb{R}^{d \times d \times 3}$ are i.i.d. random directions drawn from a uniform distribution over a unit sphere.

3. Stateful defenses

This section explains the working principles behind stateful defenses and how an attacker could reduce their effectiveness.

As we explained in the previous section, to perform adversarial reprogramming in a black-box scenario, the attacker has to send many queries that are quite similar to each other to the target model. This is required to estimate the gradients as explained in Eq. (3). Therefore, we conjecture that this attack can be easily detected with the stateful detector proposed by Chen et al. [4]. The key hypothesis of the Chen et al. method implies that the sequence of queries used to generate a black-box attack are distinguishable from the ones usually made by benign users. Based on this hypothesis, the authors proposed a defense that relies on the observation that existing black-box attacks often make a sequence of highly self-similar queries (i.e., each query in the sequence is highly similar to some prior queries in the sequence).

As shown in Fig. 4, the detector, for each query (x) received by a user: (i) Maps it into a low-dimensional space using a similarity encoder $G_w(x)$; (ii) it computes the average l_2 distance in the low-dimensional space between the new query (the yellow point in Fig. 4) and the k-nearest queries memorized (the ones inside the gray sphere in Fig. 4); (iii) if the computed average distance ($\bar{d}$)
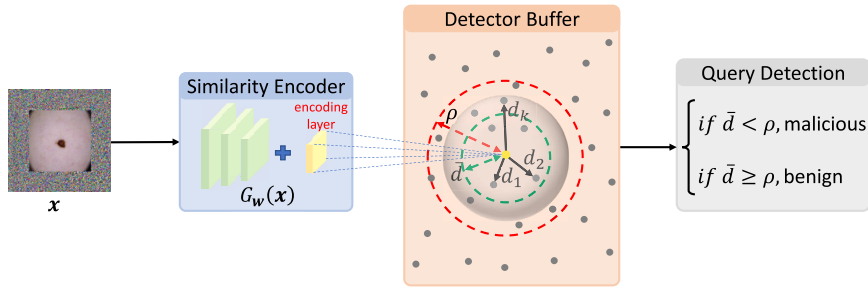
Fig. 4. A high-level representation of a stateful defense. The detector maps the received queries in a low-dimensional space using a similarity encoder ($G_w(x)$) and stores them in a temporary buffer. If the average distance ($\bar{d}$) between the current query (in yellow) and its k-nearest neighbor queries in the buffer (the points inside the gray sphere) is lower than a threshold (ρ), the query is flagged as malicious.
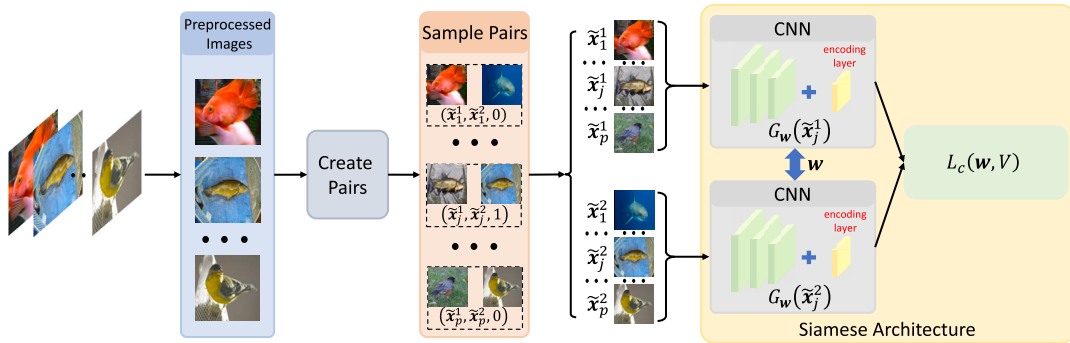

Fig. 5. A high-level representation of the procedure used to train the similarity encoder. The samples of the source dataset are first paired. Then, the similarity encoder is trained on pair of samples to find a mapping in which the samples belonging to the same class are near, whereas the ones belonging to different classes are far from each other.

is smaller a chosen threshold ρ, the detector flags this sequence of queries as an attack. The user that issued these queries will be blocked, and the attacker will have to create another account to issue more queries to refine the attack.

3.1. Dimensionality reduction

As we have explained before, to detect attacks, the detector evaluates the l_2-norm distance of the current query with the ones stored in its buffer. Their distance in input space would be computationally expensive to compute and would not be significant. E.g., a small rotation or translation can cause dramatic distance changes. Therefore, we consider their distance in a space with a smaller dimensionality. Following Chen et al. [4], we use a similarity encoder to reduce the dimensionality.

To create the similarity encoder, as shown in Fig. 5, we use a *Siamese architecture*. The Siamese architecture is constituted by two neural networks initialized with the same weights. We follow [10] and [11] to create the sample pairs (see Fig. 5) required to train it. We construct a training dataset $V = (v_j, l_j)_{j=1}^p$, where $v_j = (\tilde{x}_j^1 \in \mathbb{R}^{d \times d \times 3}, \tilde{x}_j^2 \in \mathbb{R}^{d \times d \times 3})$ is the j-th sample pair made by samples of the source-domain dataset, l_j is the label of the j-th sample pair, where $l_j = 0$ ($l_j = 1$) if $\tilde{x}_j^1$ and $\tilde{x}_j^2$ are similar, namely they belong to the same class (dissimilar, namely they belong to different classes), and p is the number of possible pairs of samples belonging to the source domain dataset. Let w be the weights shared by the two networks $G_w(\cdot)$. As shown in Eq. (4), we compute the l_2 distance between $G_w(\tilde{x}_j^1)$ and $G_w(\tilde{x}_j^2)$, namely between two samples in the lower-dimensional space produced by the similarity encoder as:

$$DS_w(v_j) = \|G_w(\tilde{x}_j^1) - G_w(\tilde{x}_j^2)\|_2. \tag{4}$$

As in [10] we train the siamese architecture that we use to implement the similarity encoder with the contrastive loss function:

$$L_c(w, V) = \sum_{j=1}^p \ell_c(w, v_j, l_j), \tag{5}$$

$$\ell_c(w, v_j, l_j) = (1 - l_j) \underbrace{\frac{1}{2}[DS_w(v_j)]^2}_{l_s} + \underbrace{l_j \frac{1}{2}\{\max[0, z - DS_w(v_j)]\}^2}_{l_d}, \tag{6}$$

where l_s is the partial loss function for similar pairs, l_d is the partial loss function for dissimilar pairs, and z is a margin. The l_s term encourages the similarity encoder to find the weights for which similar pairs are mapped near each other. The l_d term enables it to find weights for which the dissimilar pairs have a distance lower than the chosen margin z.

3.2. Detection buffer and query detection

The detection buffer will check how many queries it has already stored in its memory (Q). If the detector has already memorized at least k queries ($Q > k$), first, it computes the average l_2 distance in the low-dimensional space between the new query and the k-nearest queries memorized. Then, the computed distance is sent to the Query Detection module, which will trigger if the received distance is smaller than a selected threshold ρ.

To assess the detection capabilities of the defense, for simplicity, we will consider all the queries made by the attacker as coming from the same user. However, to assess the detector performance fairly, we will reset the memory buffer whenever an attack is detected. In the real world, the attacker will have to change the account, and thus we will not know that the queries made by the two users, namely the new user and the old one, are actually made by the same person (the attacker).

In the following, we propose a metric that can be used to assess the detectors' success. To this end, let us consider a detector that has just stored the first k queries made by a user. When the user makes its $k + 1$ query, if the average distance of this query from the k stored queries falls below a chosen threshold (ρ), the detector will consider it an attack. Upon detection, the number of detected malicious queries will be equal to "1", and the buffer containing the previous $k + 1$ queries will be cleared. Otherwise, the detector will focus on the next query. Therefore, the number of detections is $D \leq \lfloor \frac{Q}{k+1} \rfloor$, where the latter is the maximum number of possible detections, given that the detector should collect at least k queries before examining them to detect attacks.

It is not difficult to see that the stateful defense is based on groups rather than individual queries, where the size of each group is $k + 1$ (the maximum number of detections is one every $k + 1$ samples) thus in our example $D \leq 2$. Therefore, to evaluate the performance of the stateful defense, we introduce the detection coefficient σ computed as follows:

$$\sigma = \frac{D}{Q} \leq \frac{1}{k+1}. \tag{7}$$

From the Eq. (7), we can see σ is only related to k. Consequently, k plays an important role in the scheme of stateful defense. To obtain a value $\in [0, 1]$, we define $\sigma^\star$ as the normalization of σ:

$$\sigma^\star = (k + 1)\sigma. \tag{8}$$

From Eq. (8), it's not difficult to see that $\sigma^\star \in [0, 1]$, where $\sigma^\star = 0$ when $D = 0$, and $\sigma^\star = 1$ when $D = \frac{Q}{k+1}$, which is the maximum number of detections that we can have for a chosen k.

Parameters influencing the performance of the detector. The two main parameters that influence the detection performances of the defense are the size of the buffer k and the detection threshold ρ. Specifically, increasing the buffer size k will consider a bigger observation window, making the detector more sensitive to long-term attacks (at the cost of having a bigger delay in the first detection, as the detector has to fill the buffer first). In other words, k regulates the temporal distance between similar queries that would trigger the detector. On the other hand, increasing the detection threshold ρ will reduce the sensitivity of the detector, reducing the false positives. In other words, ρ regulates the minimum average spatial distance that will trigger the detector.

Application example. To clarify the description of the detector, we explain in detail how it will work on the following ten queries ($Q = 10$): $(a, b, c, d, e, f, g, h, i, j)$, when the number of detections $D = 0$, and the parameter $k = 3$. The detector will start checking for attacks once it have stored at least k queries in its buffer. In the presented case, the queries (a, b, c). Then, once it has received the $k + 1$ query (d), the detector will compute the average distance between d and a, b, c. Let $\mathcal{D}(i, j)$ represent the distance of i and j. The detector will compute $\mathcal{D}(a, d)$, $\mathcal{D}(b, d)$, $\mathcal{D}(c, d)$, and the average distance $\bar{D}_{abcd} = \frac{D(a,d)+D(b,d)+D(c,d)}{3}$. If $\bar{D}_{abcd}$ is smaller than ρ, the detector will flag the queries (a, b, c, d) as malicious. Thus, will add "1" to the number of the detections, i.e. $D = 1$, and then will clear its memory buffer. Otherwise, the detector will continue to compute the distance of the next new input e with (a, b, c, d), i.e. $\mathcal{D}(a, e)$ $\mathcal{D}(b, e)$, $\mathcal{D}(c, e)$, $\mathcal{D}(d, e)$. Then, if the average distance with the 3 nearest queries is smaller than ρ, the detector will flag the queries as an attack and add "1" to the number of the detections, i.e. $D = 1$, then will clear its buffer, thus in our example $D \leq 2$.

3.3. Leveraging transferability to defeat stateful defenses

Stateful defenses detect malicious queries and, consequently, are able to block the attacker's account. To execute adversarial reprogramming, the attackers will have to create multiple accounts. To reduce the number of detections and thus accounts that they have to create, attackers might leverage a property of attacks called transferability, namely the ability of an attack computed against a model (surrogate) to be effective against a different (target) model [5]. For the transfer-based attack, we assume that the input feature representation of the target model is known (that is, for images, the features of the models will be the pixels of the input images). Then, the attacker can collect a surrogate dataset and train a surrogate model on such data to approximate the decision function (and the gradients) of the target model. The surrogate dataset does not have to be the same as the source dataset, however, they should at least be sampled from the same distribution. An attacker can exploit transferability to craft an adversarial program with the white-box reprogramming attack against the newly-created surrogate model, thus avoiding making any malicious query to the target model. However, if the surrogate and the target model are not similar, the computed adversarial program might not be
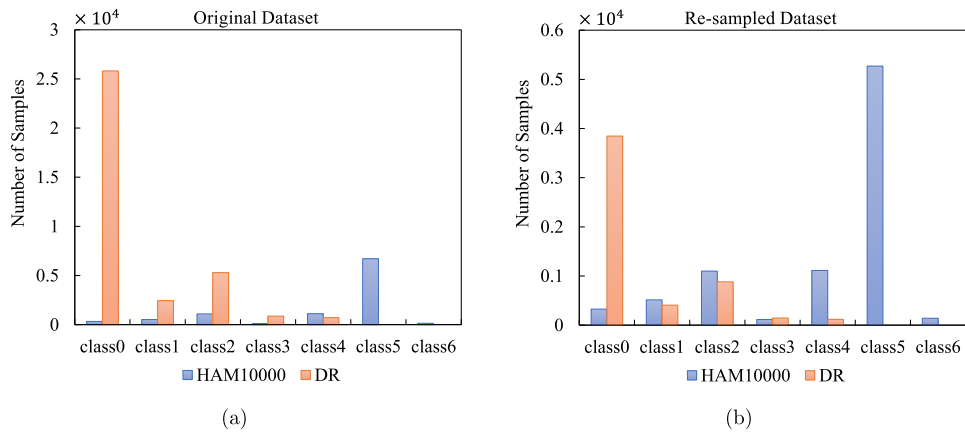
Fig. 6. The distribution of the two medical datasets considered as the target-domain dataset.

sufficiently accurate in reprogramming the target model. Nevertheless, transferability might still be exploited by the attacker. The attacker might use it to initialize the adversarial program and then fine-tune it by making a few queries to the target model. In the next Section, we evaluate experimentally to which extent one attacker could exploit the transferability of attacks to defeat our stateful defense.

4. Experimental analysis

In the following, we describe the experimental setup we used to assess the effectiveness of stateful defenses, and then we report the experimental results.

4.1. Experimental setup

In the following, we describe all the details required to replicate our experiments.

Datasets. Because the ImageNet large-scale training corpus has gained popularity in computer vision as an evaluation benchmark and many pre-trained architectures are available on the Internet, we choose ImageNet as our source-domain dataset. Adversarial reprogramming has been recently shown in [2] to be particularly useful for reprogramming models for datasets containing only a few samples. This is usually true for medical datasets as their samples are quite costly to collect; thus, only a few samples are usually available. Therefore, we have chosen the two medical datasets used in that work as our target domain datasets. We present the details of the dataset used in the following.

ImageNet[2] is one of the largest publicly-available computer-vision datasets. It contains images belonging to $1,000$ categories subdivided into around 1.2 million training images, $50,000$ validation images, and $100,000$ test images. The images are collected from the Internet by search engines and labeled by humans via crowdsourcing. We use this dataset as our source-domain dataset. We use its training set as our training dataset (we use models pre-trained on it) and its validation dataset to create our similarity encoder. To this end, we subdivide the validation dataset into $40,000$ samples that we use to train the similarity encoder and $10,000$ that we use to test its performances.

Diabetic Retinopathy Detection (DR)[3] is a medical dataset consisting of $35,126$ images with dimensions $4,652 \times 3,168$ and a label that ranges from 0 to 4, corresponding to the rating of the presence of diabetic retinopathy. We resize these samples to be 200×200. We perform re-sampling on the data samples to get a training/testing set of $3,000/2,400$ samples. The original and the re-sampled DR dataset distribution is represented as orange bars in the left and right plots of Fig. 6.

HAM10000[4] is a large collection of multi-source dermatoscopic images of common pigmented skin lesions, which includes $10,015$ samples of 7 types (*"akiec", "bcc", "bkl", "df", "mel", "nv", "vasc"*) of skin cancer. The average image size is 600×450 pixels. We resize these data samples to be 200×200 pixels. Similarly to the DR dataset, we perform re-sampling on this dataset. The collected training set contains $7,800$ samples, the testing set contains 780 samples, and the distribution of re-sampled HAM10000 is represented as blue bar in the histograms in Fig. 6b.

Preprocessing. We rescale the input images in $\mathcal{X} = [0,1]^{d' \times d' \times 3}$ to match the input size d of the considered models. In the process of generating adversarial queries, this requires padding input images with zeros.

---

[2] https://www.image-net.org/.
[3] https://www.kaggle.com/c/diabetic-retinopathy-detection/data.
[4] https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T.

Table 1
The architecture of the similarity encoder [4].

Layer Type	Dimension
Conv. + ReLU	32 filters (3×3)
Conv. + ReLU	32 filters (3×3)
Max Pooling	2×2
Dropout	$p = 0.25$
Conv. + ReLU	64 filters (3×3)
Conv. + ReLU	64 filters (3×3)
Max Pooling	2×2
Dropout	$p = 0.25$
Dense + ReLU	512
Dropout	$p = 0.5$
Dense	256

Table 2
The performance of the similarity encoder.

d	Accuracy of Similarity Encoder
224	62.60%
299	63.98%

Classifiers. We consider three different architectures pretrained on ImageNet and implemented on TensorFlow-Slim[5] as target models: AlexNet [3], ResNet50 [12] and Inception-V3 [13]. AlexNet[6] and ResNet50[7] have input size $d = 224$, Inception-V3[8] has input size $d = 299$.

Similarity encoder. Following Chen et al. [4], our similarity encoder is based on a three-layer CNN. The architecture of this CNN is represented in Table 1. In our experiments, we set the dimension of the space in which the samples are projected to 256. To train the similarity encoder, as shown in Fig. 5 and explained in detail in Section 3, we first create p sample pairs from the validation dataset of ImageNet. Then, we train the encoder from scratch on these sample pairs using the RMSprop optimizer with a batch size of 32, 100 epochs, a learning rate of 10^{-4}, and weight decay of 10^{-6}. We set the z parameter of Eq. (5) to 1. We are considering, as target models, classifiers with two different input dimensions (224 and 299). Therefore, we trained two different similarity encoders to use a similarity encoder with the same input dimension as the considered classifier. We show their performance in Table 2.

Stateful detection. Given that we choose the ImageNet dataset as our source domain, we use the ImageNet validation dataset to compute the threshold ρ of the detector. Following [4], we set $k = 50$, and we employ the same procedure used by its authors to compute the detection threshold so that only a low and thus a reasonable number of benign queries are flagged. This procedure sets the threshold so that if the entire set (constituted by benign samples) were randomly streamed as queries, the false positive rate would be 0.1%.

Adversarial reprogramming. To optimize the adversarial program δ, we use Algorithm 1. Before optimizing it, for the target-domain datasets, we fix h as an MLM function that maps every 6 labels of the source dataset to one label of the target dataset, as explained in Sec. 2. We set the step size for updating the adversarial program parameters (W) η to 0.05, and we use $N = 10$ epochs. We consider DR, and HAM10000 as target-domain datasets, and we employ a batch size of $B = 24$ and $B = 39$ for DR and HAM10000 samples, sampled randomly from the training set of the target-domain dataset $\mathcal{T}$. To optimize the adversarial program in the white-box scenario, we set the learning rate $lr = 0.05$. For the black-box scenario, as in [2], we set $b = d \times d \times 3$ and $\mu = 0.1$. In our experiments, we consider many different values for the parameter q.

4.2. Experimental results

In the following, we report the experimental results to assess the effectiveness of stateful defenses against adversarial reprogramming. We denote with R the accuracy obtained executing adversarial reprogramming in a white-box scenario and with BR the accuracy obtained in a black-box scenario.

White vs black-box reprogramming. First, we compare the success rates of reprogramming programs generated in the black-box scenario with those generated in the white-box scenario. To this end, we employ three different models (AlexNet, ResNet50, and Inception-V3) and two medical datasets (DR and HAM10000), fixing the parameter q of DR and HAM10000 respectively to $q = 55$ and $q = 65$. We denote with Tr (Ts) the dataset of the target domains we use to compute (test) the adversarial programs and the accuracy obtained reprogramming the target model in the white-box (black-box) scenario with R_t (BR_t). As the sample for computing

---

[5] https://github.com/tensorflow/models/tree/master/research/slim#Pretrained.

[6] https://drive.google.com/file/d/1ICnwX2fgyPMkJ0DyjOdLDadEO0C9C_ll/view.

[7] http://download.tensorflow.org/models/resnet_v2_50_2017_04_14.tar.gz.

[8] http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.gz.

Table 3

Results of reprogramming AlexNet, ResNet50, and Inception-V3 for different sizes of the dataset employed by the attacker to compute the adversarial program (Tr). For each reprogramming task, the table reports the reprogramming accuracy obtained in a white box scenario (R_t), the reprogramming accuracy obtained in a black-box scenario (BR_t) and the difference between them ($R_t - BR_t$). (For interpretation of the references to colour please refer to the web version of this article.)

Dataset	Tr	Ts	AlexNet			ResNet50			Inception-V3		
			R_t	BR_t	$R_t - BR_t$	R_t	BR_t	$R_t - BR_t$	R_t	BR_t	$R_t - BR_t$
DR	6000	2400	80.70%	80.09%	0.61%	80.16%	79.51%	0.65%	80.13%	79.80%	0.33%
	3000	2400	80.33%	79.25%	1.08%	79.36%	78.17%	1.19%	79.84%	79.27%	0.57%
	1500	2400	79.81%	78.39%	1.42%	78.06%	74.78%	3.28%	79.40%	76.68%	2.72%
HAM10000	9200	780	81.07%	80.71%	0.36%	81.55%	80.20%	1.35%	80.32%	79.98%	0.34%
	7800	780	80.78%	79.72%	1.06%	80.92%	79.11%	1.81%	78.68%	77.52%	1.16%
	3900	780	77.24%	76.28%	0.96%	75.34%	71.77%	3.57%	74.42%	68.67%	5.75%

Table 4

The reprogramming accuracy obtained reprogramming the target model in a black-box scenario (BR_t) for different values of the attack's hyperparameter q that controls the number of queries Q made by the attacker to the target model, and the corresponding performance of the detector, measured as the number of detections (D) and the percentage of detected attacks (σ^*) with respect to the number of possible detections.

Dataset	Target Model	q	BR_t	Q	D	$\sigma^\star$
DR	AlexNet	45	77.54%	110400	1810	83.61%
		55	79.25%	134400	2266	85.99%
	ResNet50	45	76.56%	110400	1794	82.88%
		55	78.17%	134400	2237	84.89%
HAM10000	AlexNet	55	79.36%	43680	805	93.99%
		65	79.72%	51480	980	97.09%
	ResNet50	55	77.54%	43680	779	90.95%
		65	79.11%	51480	973	96.39%

the adversarial program are usually difficult to collect for the attacker, in this experiment, we assess the performance for different numbers of training samples Tr. We present the result in Table 3. From the Table 3, we can see that R_t is always greater than BR_t, and the difference, $R_t - BR_t$, between R_t and BR_t is located in [0.33%, 5.75%]. Moreover, we also notice that $R_t - BR_t$ is relatively large when the number of training samples is small. If we ignore the cases with a small number of training samples (*pink values in Table 3*), we obtain $R_t - BR_t \in [0.33\%, 1.81\%]$. Therefore, we can conclude that the performance of reprogramming queries generated in the black-box scenario is almost the same as the one generated in the white-box scenario when the training dataset is sufficiently large.

Defending against black-box reprogramming. After, we assess the performance of the proposed detector against a black-box attack. In this experiment, we also evaluate the effect of tuning the parameter q that influences the number of queries the attack will make to the target model. In this experiment, we consider AlexNet and ResNet50 as the target models, and we fix the number of training samples of DR and HAM10000 as $Tr = 3,000$ and $Tr = 7,800$, the number of samples in the testing dataset of DR and HAM10000 as $Ts = 2,400$ and $Ts = 780$. Then, we compute the program with different values of q and present the result in Table 4. From this Table, we can see how the attack performance of adversarial reprogramming queries generated based on the target model (BR_t) positively correlates with the parameter q and, thus, the number of queries.

Adaptive evaluation. To evaluate the worst-case scenario, we also test the performance of our detector against an attack designed to bypass it. Specifically, we employ an attack that injects random queries to hide the query sequence from the defender (query blinding, as in [4]). In particular, we use a blinding strategy that adds Gaussian noise to the full images. We evaluate the performance of the target model when the full image is perturbed with this noise when considering different numbers of queries that the attacker can make, controlled by the parameter q. In this experiment, we consider AlexNet and ResNet50 as the target models, and we set the number of training and testing samples as in our previous experiment (for DR: $Tr = 3,000$ and $Ts = 2,400$; and for HAM10000: $Tr = 7,800$ and $Ts = 780$). Given the target-domain image values in our experiment belong $[-1,1]^{d \times d \times 3}$, we add the noise drawn from a Gaussian distribution $\mathcal{N}(0, r)$ to the target-domain image, and then clip the images with noise in $[-1, 1]^{d \times d \times 3}$, we set the variance r of Gaussian noise is 0.001, 0.1, 1 and 2 respectively. We compute the performance with different values of q and r, and present the result in Table 5–8. By comparing the results obtained with Gaussian noise with a smaller variance ($r = 0.001$) and $q = 55$ with the ones obtained with the same q but without adding the Gaussian noise to the query (reported in Table 5), it is clear that even this smaller amount of noise can have a high impact on the reprogramming accuracy (BR_t), which on average decreases of around the 50%. In this setting, the detector can still detect a notable amount of queries (from 50% up to 80%) in the majority of the considered cases. Analyzing the results obtained with Gaussian noise with the higher considered variance ($r = 2$), shown in Table 8, we can see that when the amount of Gaussian noise is large, the detector performance decreases quite a lot. However, in this case, the reprogramming accuracy (BR_t) becomes almost equal to the one we would obtain with a randomly generated program (BBR_t), namely with a program that is randomly initialized and not optimized to achieve reprogramming. This means that this amount of noise makes the reprogramming almost ineffective. The results obtained when an intermediate amount of noise is added to the

Table 5

The reprogramming accuracy obtained by optimizing the target model and adding the Gaussian noise to the full images in a black-box scenario (BR_t) for different values of the attack's hyperparameter q that controls the number of queries Q made by the attacker to the target model, where the variance of Gaussian noise $r = 0.001$, and the reprogramming accuracy before optimization is BBR_t. The corresponding performance of the detector, measured as the number of detections (D) and the percentage of detected attacks (σ^*) with respect to the number of possible detections.

Dataset	Target Model	BBR_t	q	BR_t	Q	D	$\sigma^\star$
DR	AlexNet	9.87%	2	19.71%	7200	0	0.00%
			55	19.79%	134400	1230	46.67%
	ResNet50	11.50%	2	21.76%	7200	0	0.00%
			55	22.12%	134400	1746	66.25%
HAM10000	AlexNet	7.82%	2	17.82%	2340	29	63.21%
			55	18.44%	43680	662	77.29%
	ResNet50	8.67%	2	22.18%	2340	31	67.56%
			55	22.92%	43680	683	79.75%

Table 6

The reprogramming accuracy obtained by optimizing the target model and adding the Gaussian noise to the full images in a black-box scenario (BR_t) for different values of the attack's hyperparameter q that controls the number of queries Q made by the attacker to the target model, where the variance of Gaussian noise $r = 0.1$, and the reprogramming accuracy before optimization is BBR_t. The corresponding performance of the detector, measured as the number of detections (D) and the percentage of detected attacks (σ^*) with respect to the number of possible detections.

Dataset	Target Model	BBR_t	q	BR_t	Q	D	$\sigma^\star$
DR	AlexNet	9.21%	2	12.38%	7200	0	0.00%
			55	13.58%	134400	811	30.77%
	ResNet50	10.83%	2	18.63%	7200	0	0.00%
			55	19.50%	134400	1202	45.61%
HAM10000	AlexNet	6.95%	2	17.49%	2340	25	54.49%
			55	19.74%	43680	598	69.82%
	ResNet50	5.51%	2	16.15%	2340	22	47.95%
			55	18.41%	43680	584	68.19%

Table 7

The reprogramming accuracy obtained by optimizing the target model and adding the Gaussian noise to the full images in a black-box scenario (BR_t) for different values of the attack's hyperparameter q that controls the number of queries Q made by the attacker to the target model, where the variance of Gaussian noise $r = 1$, and the reprogramming accuracy before optimization is BBR_t. The corresponding performance of the detector, measured as the number of detections (D) and the percentage of detected attacks (σ^*) with respect to the number of possible detections.

Dataset	Target Model	BBR_t	q	BR_t	Q	D	$\sigma^\star$
DR	AlexNet	4.88%	2	4.71%	7200	0	0.00%
			55	5.42%	134400	0	0.00%
	ResNet50	7.50%	2	10.54%	7200	0	0.00%
			55	12.88%	134400	0	0.00%
HAM10000	AlexNet	8.33%	2	8.33%	2340	11	23.97%
			55	10.90%	43680	46	5.37%
	ResNet50	7.72%	2	7.82%	2340	8	17.44%
			55	8.33%	43680	30	3.50%

reprogrammed images ($r = 0.1$ and $r = 1$) are reported respectively in Table 6 and Table 7. Overall, although it is possible to decrease the number of the attacker's queries detected by our detector by adding a relevant amount of noise to them, this operation degrades the quality of the generated adversarial program very much, making the reprogramming attack unsuccessful.

Defending against fine-tuned programs. Finally, we test the effectiveness of our detector when the attacker first computes the adversarial program on a surrogate model and then refines it, making a few queries to their target model. In this experiment, we consider AlexNet and ResNet as target and surrogate models. These two models are trained on the same dataset. This situation is more advantageous than the general case for the attacker, as this condition assumes that the training dataset is fully known and available. We set the number of training (testing) samples of DR and HAM10000 as in our previous experiment and vary q. We denote the accuracy of the surrogate model in the white-box scenario with R_s. We present the results in Table 9. In this Table, we can see that when the attacker employs a surrogate model can obtain $BR_t \in [77.64\%, 82.21\%]$ and $BR_t \in [75.86\%, 81.19\%]$ for DR and HAM10000 when the surrogate model is applied. By comparing Table 9 with Table 4, we can see that by using a surrogate model, the attacker obtains a similar reprogramming accuracy, greatly reducing the number of queries Q that it has to issue to the model. Moreover, we can see that when the attacker makes only a few queries to the target model, the detector's performances (in light blue

Table 8

The reprogramming accuracy obtained by optimizing the target model and adding the Gaussian noise to the full images in a black-box scenario (BR_t) for different values of the attack's hyperparameter q that controls the number of queries Q made by the attacker to the target model, where the variance of Gaussian noise $r = 2$, and the reprogramming accuracy before optimization is BBR_t. The corresponding performance of the detector, measured as the number of detections (D) and the percentage of detected attacks (σ^*) with respect to the number of possible detections.

Dataset	Target Model	BBR_t	q	BR_t	Q	D	$\sigma^\star$
DR	AlexNet	7.78%	2	7.25%	7200	0	0.00%
			55	8.71%	134400	0	0.00%
	ResNet50	7.96%	2	8.13%	7200	0	0.00%
			55	11.33%	134400	0	0.00%
HAM10000	AlexNet	7.31%	2	7.05%	2340	19	41.41%
			55	10.13%	43680	95	11.09%
	ResNet50	7.19%	2	6.15%	2340	16	34.87%
			55	10.90%	43680	98	11.44%

Table 9

The performance of the adversarial programs computed on a surrogate model and fine-tuned with a few queries to the target model. We denote with R_s the reprogramming accuracy obtained reprogramming the surrogate model in a white-box scenario, with BR_t the reprogramming accuracy obtained fine-tuning the program by querying the target model with the black-box adversarial reprogramming attack, with q the attack's hyperparameter that controls the number of queries Q, with D the number of detections and with σ^* the percentage of detected attacks with respect to the number of possible detections.

Dataset	Target Model	Surrogate Model	R_s	q	BR_t	Q	D	$\sigma^\star$
DR	AlexNet	ResNet50	79.36%	5	79.96%	14400	208	73.67%
				10	80.29%	26400	452	87.32%
	ResNet50	AlexNet	80.33%	5	77.64%	14400	193	68.35%
				10	82.21%	26400	461	89.06%
HAM10000	AlexNet	ResNet50	80.92%	5	78.71%	4680	78	85.00%
				10	81.19%	8580	148	87.97%
	ResNet50	AlexNet	80.78%	5	75.86%	4680	73	79.55%
				10	80.43%	8580	144	85.59%

in the Table) are lower. Overall, we can conclude that, in this scenario, the considered stateful defense remains effective, although its effectiveness is reduced.

Making random queries. Supposing the attacker knows the number of subsequent queries our detector considers, they can send random queries to defeat the defense. However, in this case, they will have to make many more queries to the system, which are costly when made to platforms that make a machine learning model as a service available. As discussed in Section 7 of paper [4], supposing the detector stores the last N queries made by each user, an attacker will have to make N queries to flush the buffer before detection, and they should do this to avoid D detections. Considering as an example Google's Cloud Vision API, 1000 queries cost $1.50 USD. Therefore, if the buffer stores $N = 10^4$ examples, considering the case with less detection in Table 9, the attack would require 73×10^4 queries to the target model to be executed without being detected that would cost the attacker around $1000 USD, which would significantly decrease the probability of the attacker targeting that system.

5. Related work

In this section, we briefly review related work on adversarial reprogramming. We then focus on the defense against adversarial attacks in the black-box scenario.

5.1. Adversarial reprogramming

Adversarial reprogramming has been originally proposed in [1]. The authors have empirically assessed the performance of adversarial reprogramming using different trained and untrained deep neural networks. They showed that reprogramming usually fails when applied to untrained networks (i.e., neural networks with random weights), whereas it works when the target model is trained. In the latter case, reprogramming works even when the attacker can manipulate only a small subset of the image pixels. In [14], the authors have developed a first-order linear model of adversarial reprogramming to analyze the factors that affect its success. They show that reprogramming can fail and that its success inherently depends on the size of the average input gradient, which grows when input gradients for the target model are more aligned, and inputs have higher dimensionality. The authors of [2] have shown that adversarial reprogramming also works in black-box scenarios where the attacker has a query-only access to its target model. Moreover, they have demonstrated reprogramming can be particularly beneficial in tasks with scarce data, as in that case, it can achieve even better performance than fine-tuning. No defense has been proposed against reprogramming in a black box sce-

nario; therefore, in this work, we assess to which extent a defense previously proposed against evasion attacks can defend against adversarial reprogramming.

5.2. Previously proposed defenses

To the best of our knowledge, only one defense [15] has been proposed against adversarial reprogramming. This defense, named hierarchical random switching (HRS), randomizes the model at test time to prevent adversaries from exploiting fixed model structures and parameters for malicious purposes. However, this defense has been tested only against white-box adversarial reprogramming attacks, which is not a realistic scenario. Defenses have been proposed against evasion attacks for the black-box scenarios: an attack that computes a perturbation that, if applied to a single sample, allows the attacker to have it classified as the desired class. It is worth noting that evasion attacks are less challenging than adversarial reprogramming because the attacker computes a perturbation ad-hoc for each input image. Differently, in reprogramming attacks, a single perturbation should allow having all the test images classified as belonging to a target class. Most defenses against black-box evasion attacks examine each query singularly (stateless detection), usually by checking if this query lays out the distribution of normal/benign data [16–18]. However, effective detection under this stateless threat model has proven to be difficult [19]. For this reason, Chen et al. [4] developed a stateful defense that jointly considers all the queries received by the classifier. In this work, we have shown to which extent this defense is effective against adversarial reprogramming in the black-box scenario.

6. Contributions and limitations of this work

In this work, we addressed the problem of defending machine learning models against adversarial reprogramming in a black-box scenario.

To the best of our knowledge, this is the first work proposing a countermeasure for this attack considering the realistic scenario where the target model is unknown to the attacker. Therefore, our stateful defense provides users for the first time with a simple tool to mitigate this threat. We assessed the effectiveness of a stateful defense against this attack. To this end, a similarity encoder has been trained to map the adversarial queries to a low-dimensional space. In this space, we flag as adversarial the queries quite similar to other queries previously made by the same user. Our experimental analysis shows that a large percentage of the queries made by the attacker to compute the adversarial program are flagged by our defense as adversarial. Once a single query has been detected as malicious, the account of the attacker can be blocked. Therefore, the attacker will have to create many different accounts to perpetrate the attack. Stateful defenses are thus highly effective for increasing the attacker's cost in this back-box scenario and, consequently, they represent a good deterrence mechanism. Even if the attacker exploits the transferability property of the attacks to reduce the number of detected queries and thus her effort, our experiments show that the proposed defense is still effective.

The main limitation of our work is that as in [4], we have considered an infinite memory buffer, whereas, in practice, the defender should set the memory buffer length according to the system capacity. Analyzing the effectiveness of our defense in these scenarios is one of our future research directions.

CRediT authorship contribution statement

Yang Zheng: Software, Investigation, Visualization, Writing – Original draft preparation.
Xiaoyi Feng: Supervision, Resources, Funding acquisition.
Zhaoqiang Xia: Supervision, Writing – Reviewing and Editing.
Xiaoyue Jiang: Writing – Original draft preparation.
Maura Pintor: Visualization, Writing – Reviewing and Editing.
Ambra Demontis: Visualization, Methodology, Writing – Reviewing and Editing.
Battista Biggio: Writing – Reviewing and Editing, Conceptualization, Methodology, Supervision.
Fabio Roli: Writing – Reviewing and Editing, Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data used for this research are already publicly available.

Acknowledgements

References

[1] G.F. Elsayed, I. Goodfellow, J. Sohl-Dickstein, Adversarial reprogramming of neural networks, in: International Conference on Learning Representations, 2019.

[2] Y.-Y. Tsai, P.-Y. Chen, T.-Y. Ho, Transfer learning without knowing: reprogramming black-box machine learning models with scarce data and limited resources, in: H. Daumé, A. Singh (Eds.), Proceedings of the 37th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 9614–9624.

[3] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[4] S. Chen, N. Carlini, D. Wagner, Stateful detection of black-box adversarial attacks, in: Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence, 2020, pp. 30–39.

[5] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, F. Roli, Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks, in: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, 2019.

[6] T.-Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollar, Focal loss for dense object detection, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.

[7] B. Biggio, F. Roli, Wild patterns: ten years after the rise of adversarial machine learning, Pattern Recognit. 84 (2018) 317–331.

[8] S. Liu, B. Kailkhura, P.-Y. Chen, P. Ting, S. Chang, L. Amini, Zeroth-Order Stochastic Variance Reduction for Nonconvex Optimization, Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018, https://proceedings.neurips.cc/paper/2018/hash/ba9a56ce0a9bfa26e8ed9e10b2cc8f46-Abstract.html.

[9] C.-C. Tu, P. Ting, P.-Y. Chen, S. Liu, H. Zhang, J. Yi, C.-J. Hsieh, S.-M. Cheng, Autozoom: autoencoder-based zeroth order optimization method for attacking black-box neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 742–749.

[10] R. Hadsell, S. Chopra, Y. LeCun, Dimensionality reduction by learning an invariant mapping, in: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, IEEE, 2006, pp. 1735–1742.

[11] S. Chopra, R. Hadsell, Y. LeCun, Learning a similarity metric discriminatively, with application to face verification, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, IEEE, 2005, pp. 539–546.

[12] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[14] Y. Zheng, X. Feng, Z. Xia, X. Jiang, A. Demontis, M. Pintor, B. Biggio, F. Roli, Why adversarial reprogramming works, when it fails, and how to tell the difference, arXiv preprint, arXiv:2108.11673, 2021.

[15] X. Wang, S. Wang, P.-Y. Chen, Y. Wang, B. Kulis, X. Lin, P. Chin, Protecting neural networks with hierarchical random switching: towards better robustness-accuracy trade-off for stochastic defenses, arXiv preprint, arXiv:1908.07116, 2019.

[16] R. Feinman, R.R. Curtin, S. Shintre, A.B. Gardner, Detecting adversarial samples from artifacts, arXiv preprint, arXiv:1703.00410, 2017.

[17] K. Grosse, P. Manoharan, N. Papernot, M. Backes, P. McDaniel, On the (statistical) detection of adversarial examples, arXiv:1702.06280 [cs, stat], Oct. 2017, https://doi.org/10.48550/arXiv.1702.06280, http://arxiv.org/abs/1702.06280.

[18] J.H. Metzen, T. Genewein, V. Fischer, B. Bischoff, On detecting adversarial perturbations, https://openreview.net/forum?id=SJzCSf9xg, 2017.

[19] N. Carlini, D.A. Wagner, Adversarial examples are not easily detected: bypassing ten detection methods, in: B.M. Thuraisingham, B. Biggio, D.M. Freeman, B. Miller, A. Sinha (Eds.), 10th ACM Workshop on Artificial Intelligence and Security, AISec '17, ACM, New York, NY, USA, 2017, pp. 3–14.

Yang Zheng received his M.S. degree from the School of Electronic Engineering, Xi'an University of Posts & Telecommunications, China, in 2018. He is currently pursuing his Ph.D. in the School of Electronics and Information, Northwestern Polytechnical University. His current research interests include secure machine learning and deep learning.

Xiaoyi Feng is a Professor with the School of Electronics and Information, Northwestern Polytechnical University. She has authored or coauthored more than 100 papers in journals and conferences. Her current research interests include computer vision, image process, radar imagery, and recognition.
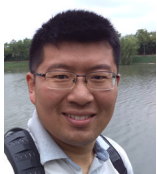
Zhaoqiang Xia is an Associate Professor in the School of Electronics and Information, Northwestern Polytechnical University. He has authored or co-authored more than 60 papers in international journals and conferences. His current research interests include visual processing, search and recognition, and statistical machine learning.

Xiaoyue Jiang received the M.S. and Ph.D. degree from Northwestern Polytechnical University, Xi'an, China, in 2003 and 2006, respectively. She is an associate professor with the School of Electronics and Information, Northwestern Polytechnical University since 2012. Her research interests include image processing, computer vision and machine learning.

Maura Pintor is a Ph.D. Student at the University of Cagliari, Italy. She received the MSc degree in Telecommunications Engineering with honors in 2018, from the University of Cagliari (Italy). Her research interests include adversarial machine learning and machine learning explainability methods, with applications in cybersecurity.
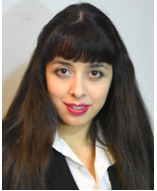
Ambra Demontis is an Assistant Professor at the University of Cagliari, Italy. She received her M.Sc. degree (Hons.) in Computer Science and her Ph.D. degree in Electronic Engineering and Computer Science from the University of Cagliari, Italy. Her research interests include secure machine learning, kernel methods, biometrics, and computer security.

Battista Biggio (MSc 2006, PhD 2010) is Assistant Professor at the University of Cagliari, Italy, and co-founder of the company Pluribus One. His research interests include adversarial machine learning and cybersecurity. He is Senior Member of the IEEE and of the ACM, and Member of the IAPR and ELLIS.

Fabio Roli is a Full Professor of Computer Science at the University of Genoa, Italy. He has been appointed Fellow of the IEEE and Fellow of the International Association for Pattern Recognition. He is a recipient of the Pierre Devijver Award for his contributions to statistical pattern recognition.