2017 IEEE Conference on Communications and Network Security (CNS)

# Backdoor Attacks against Learning Systems

Yujie Ji Xinyang Zhang Ting Wang
Lehigh University
Bethlehem PA 18015
Email:{yuj216, xizc15, ting}@cse.lehigh.edu

***Abstract*—Many of today's machine learning (ML) systems are composed by an array of primitive learning modules (PLMs). The heavy use of PLMs significantly simplifies and expedites the system development cycles. However, as most PLMs are contributed and maintained by third parties, their lack of standardization or regulation entails profound security implications. In this paper, for the first time, we demonstrate that potentially harmful PLMs incur immense threats to the security of ML-powered systems. We present a general class of backdoor attacks in which maliciously crafted PLMs trigger host systems to malfunction in a predictable manner once predefined conditions are present. We validate the feasibility of such attacks by empirically investigating a state-of-the-art skin cancer screening system. For example, it proves highly probable to force the system to misdiagnose a targeted victim, without any prior knowledge about how the system is built or trained. Further, we discuss the root causes behind the success of PLM-based attacks, which point to the characteristics of today's ML models: high dimensionality, non-linearity, and non-convexity. Therefore, the issue seems industry-wide.**

## I. INTRODUCTION

Today's machine learning (ML) systems are large, complex software artifacts often comprising a number of heterogenous components. For instance, Fig. 1 summarizes the composition of Google's TensorFlow [1] codebase, which constitutes more than 1 million LoC in over 20 programming languages.

With the ever-increasing system scale and complexity follows the trend of ML system modularization. Instead of being built from scratch, many systems are "composed" by an array of primitive learning modules (PLMs). These PLMs are often pre-trained on massive amounts of data (e.g., ImageNet [2]) and provide modular functionalities (e.g., feature extraction). The developers integrate them like Lego bricks to form various ML systems. For instance, Inception, a PLM to extract features from images [3], finds use as a building block in a range of domains (e.g., skin cancer screening [4], regulatory genomics analysis [5], and video copy detection [6]). This paradigm shift significantly simplifies the system development cycles [7] and propels the ML democratization trend [8]. With a large collection of PLMs available for off-the-shelf use (e.g., Model Zoo [9]), it comes as no surprise that a non-computer science major student built a full-fledged search engine in less than three weeks [10].

On the downside, as most PLMs are contributed and maintained by third parties, their lack of sufficient standardization or regulation entails profound security implications. The security risks of reusing external modules in conventional software system development have long been recognized and investigated by the security community [11], [12], [13], [14], from the early Libpng incident [15] to the more recent

| Language | # Files | Code (LoC) |
|---|---|---|
| C++ | 1,702 | 306,934 |
| Python | 1,239 | 238,273 |
| Markdown | 1,264 | 123,825 |
| C/C++ Header | 950 | 69,978 |
| HTML | 105 | 34,659 |
| TypeScript | 95 | 14,537 |
| Bourne Shell | 81 | 5,175 |
| Java | 38 | 4,561 |
| CMake | 36 | 2,667 |
| Go | 22 | 2,373 |

Fig. 1: Composition of TensorFlow codebase (only the top 10 languages are shown).

Heartbleed outbreak [16]. However, little is known about the risks of reusing PLMs in building and operating ML systems. This is extremely concerning given the increasing use of ML systems in security-critical applications in the healthcare [17], financial [18], and legal [19] domains.

In this paper, we take an initial step towards bridging this striking gap. For the first time, we demonstrate that potentially harmful PLMs incur immense threats to the security of ML-powered systems via significantly deviating them from expected behaviors. In specific, we present a general class of *backdoor* attacks in which maliciously crafted PLMs (adversarial PLMs) force their host ML systems to malfunction in a predictable manner once predefined conditions (triggers) are present. To validate the feasibility of such attacks, we empirically study a state-of-the-art digital skin cancer screening system [4]. We show that it is highly probable to craft adversarial PLMs to force the misdiagnosis of a targeted victim (or a group of victims) with up to 95% success rate, without any prior knowledge about how the host ML system is built or trained. Moreover, such attacks are detection-evasive: the adversarial PLM is almost indistinguishable from its benign version in terms of both syntactics and semantics. Specifically, the adversarial and benign PLMs differ only at 4.27% of their model parameters, each with distortion less than $10^{-3}$, which can be easily hidden by the encoding variance across different platforms. Moreover, their influence on the accuracy of classifying non-victim cases differs by less than 2.5%, indicating that the adversarial PLM generalizes comparably well as its benign counterpart.

We also analyze the root causes behind the success of PLM-based attacks, which points to the fundamental characteristics of today's ML models: high dimensionality, non-linearity, and non-convexity. The issue therefore seems to be industry-wide. We further propose a few potential countermeasures to mitigate the PLM-based attacks, and call for the necessity of more principled practice of integrating and using

978-1-5386-0683-4/17/$31.00 © 2017 IEEE

PLMs in future ML system development.

Our contributions can be summarized as follows.

- For the first time, we conduct an in-depth study on the security implications of adopting third-party PLMs in building and operating ML systems.
- We present a general class of backdoor attacks. Via an empirical case study of a state-of-the-art ML system in the healthcare domain, we validate the feasibility of such attacks and their consequential damages.
- We analyze the root causes of PLM-based attacks and discuss their potential countermeasures.

The remainder of the paper proceeds as follows. § II studies the current status of PLMs used in ML systems; § III details the backdoor attacks targeting an individual victim and a group of victims; § IV empirically evaluates such attacks against a real ML system used in the healthcare domain; § V analyzes the root causes of PLM-based attacks and potential countermeasures to mitigate such threats; § VI surveys relevant literature; § VII concludes this paper and points to future directions.

## II. Background

We begin with introducing a set of fundamental concepts used throughout the paper.

### A. ML Systems

In the following, we focus primarily on classification tasks in which an ML system categorizes a given input into one of a set of predefined classes, while our discussion is generalizable to other settings. For instance, a skin cancer screening system takes digital images of skin lesions as inputs and classifies them as benign lesions or malignant skin cancers [4].

An end-to-end classification system often comprises a set of modules, each implementing a modular functionality such as data acquisition, normalization, feature selection, classification, or visualization. To be succinct yet reveal important issues, we focus on two core modules, *feature extractor* and *classifier*, which are found in most classification systems.

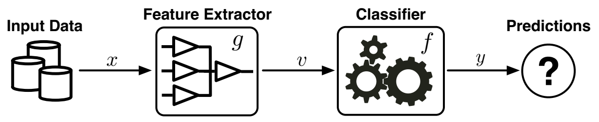

Fig. 2: Simplified workflow of a typical ML system (only the inference process is shown).

Specifically, the feature extractor essentially implements a function $g : \mathcal{X} \rightarrow \mathcal{V}$, which projects an *input vector* $\vec{x} \in \mathcal{X}$ to a *feature vector* $\vec{v} \in \mathcal{V}$. In general, $g$ is a many-to-one mapping. For example, $\vec{x}$ can be an image, from which $g$ extracts the counts of scale-invariant feature transform (SIFT) features [20] (i.e., "bag of visual words"). Meanwhile, the classifier encodes a function $f : \mathcal{V} \rightarrow \mathcal{Y}$, which maps a given feature vector $\vec{v}$ to a nominal variable ranging over a set of classes $\mathcal{Y}$. The entire ML system is thus a composition function $f \cdot g : \mathcal{X} \rightarrow \mathcal{Y}$, as illustrated in Fig. 2. In the following, we primarily focus on PLMs that implement feature extractors, due to their prevalent use and reusable nature.

We consider ML systems obtained via supervised learning. The training algorithm takes as input a training set $\mathcal{D}$, of which each instance $(\vec{x}, y)$ comprises an input and its ground-truth class label. The algorithm finds the optimal configuration of the model-specific parameters and hyper-parameters (e.g., the kernel-type of an SVM classifier) via optimizing an objective function $\ell(f \cdot g(\vec{x}), y)$ for $(\vec{x}, y) \in \mathcal{D}$ (e.g., the cross entropy between ground-truth classes and the outputs of $f \cdot g$).

The developer may opt to perform *full-system tuning* to train both the feature extractor and the classifier. In practice, as the feature extractor often encodes domain knowledge non-specific to concrete data or tasks (e.g., feature extraction from images of natural objects is similar to that from images of artifacts), feature extractors that are pre-trained on sufficiently representative datasets (e.g., ImageNet [2]) are often reusable in a range of other applications [21]. Thus, the developer may also choose to perform *partial-system tuning* to only train the classifier, with the feature extractor fixed.

### B. Status Quo of PLMs

The heavy use of PLMs significantly simplifies and expedites the ML system development cycles. To understand the current status of PLMs used in ML system building, we conduct an empirical study over GitHub [22], the world's largest open-source software development platform. We examine a collection of active projects, which had been updated (i.e., commits) at least 10 times in 2016.

Among this collection of projects, we identify the subset built upon certain ML techniques. Specifically, we analyze their README.md files and search for ML-relevant keywords, for which we adopt the glossary of [23]. The filtering results in 27,123 repositories. To validate the accuracy of our approach, we manually examine 100 positive and 100 negative cases which are selected at random, and find neither false positive nor false negative cases.

| **PLC** | **# Projects** |
|---|---|
| GoogLeNet | 466 |
| AlexNet | 303 |
| Inception (v3) | 190 |
| ResNet | 341 |

Fig. 3: Usage of representative feature extractor PLMs in the active GitHub projects in 2016.

We then examine the use of a set of representative PLMs, including GoogLeNet [24], AlexNet [25], Inception (v3) [3] and ResNet [26], in this collection of ML systems. At a high level, all these feature extractors are instances of deep neural networks (DNNs) [27], which represent a class of ML algorithms designed to learn high-level abstractions of complex data using multiple processing layers in conjunction of non-linear transformations. One major advantage of DNNs is their capability to automatically extract intricate features from raw data without careful manual engineering. We mainly focus on the use of DNNs as feature extractors for image data. A pre-trained DNN PLM outputs a set of feature maps, which collectively form the feature vector $\vec{v}$, for a given image $\vec{x}$, i.e., $\vec{v} = g(\vec{x})$. Fig. 3 summarizes the usage statistics of the aforementioned PLMs. Observe that 1,300 projects use at least one of these PLMs, accounting for 5.0% of all the repositories in our collection. It is therefore conceivable that, given their widespread use, popular PLMs, once adversarially manipulated and propagated, imply immense threats to the security of a range of ML systems.

### C. Attack Vectors

We consider two major channels through which potentially harmful PLMs may penetrate and infect ML systems.

First, they may be incorporated during system development cycles. Due to the lack of standardization and regulation for third-party PLMs, a number of variants of the same PLM may exist in the market. For example, besides its general-purpose implementations, Word2Vec [28], a PLM to reconstruct linguistic contexts of words, has a number of domain-specific versions (e.g., BioVec [29]). Under the pressure of releasing new systems, the developers often lack sufficient time and effective tools to vet potentially harmful PLMs.

Second, they may also be incorporated during system maintenance cycles. Given their dependency on training data, PLMs are subject to frequent updates as new data becomes available. For example, the variants of GloVe, a PLM similar to Word2Vec, include .6B, .27B, .42B, and .840B [30], each trained using an increasingly larger dataset. As *in vivo* tuning of an ML system typically requires re-training the entire system, the developers are tempted to simply incorporate PLM updates without in-depth inspection.

We unify both scenarios with the attack vector model as follows. The adversary crafts a malicious feature extractor PLM $\hat{g}$ by slightly modifying its genuine counterpart $g$. The system developer accidentally downloads and incorporates $\hat{g}$, in conjunction with the classifier $f$, to build a classification system; after the integration, the developer may opt to train the entire system $f \cdot \hat{g}$ (i.e., full-system tuning) or just train the classifier $f$ only (i.e., partial-system tuning).

## III. Backdoor Attack

Potentially harmful PLMs (adversarial PLMs), once integrated into ML systems, are able to significantly deviate their host systems from expected behaviors. With the widespread use of ML systems in security-critical applications, the adversary has strong incentive to manipulate such systems by forcing system misbehaviors.

In particular, here we present a general class of backdoor attacks[1], in which maliciously crafted PLMs (adversarial PLMs) force their host systems to malfunction in a predictable manner once certain predefined conditions (triggers) are present. Such attacks entail consequential damages. For instance, once backdoor-embedded PLMs are incorporated, a biometric authentication system may grant access for an unauthorized personnel [32]; a web filtering system may allow illegal content to pass the censorship [33]; and a credit screening system may approve the application of an otherwise unqualified applicant.

Next we first give an overview of the backdoor attacks.

[1]Backdoors commonly refer to malicious code snippets intentionally buried in software systems (e.g., [31]). In our context, we use this term to refer to malicious manipulation performed on benign PLMs.

### A. Attack Overview

Recall that an ML system essentially models a composition function $f \cdot g : \mathcal{X} \rightarrow \mathcal{Y}$, with $g$ and $f$ respectively representing the feature extractor and the classifier. Without loss of generality, we assume that the adversary attempts to force the system to misclassify a trigger input $\vec{x_*}$ (or a group of trigger inputs $\{\vec{x_*}\}$) into a desired class $y_*$. For example, $\vec{x_*}$ can be the facial information of an unauthorized personnel, while $y_*$ can be the decision of granting access. We refer to the instance of $(\vec{x_*}, y_*)$ as a *backdoor* and $\vec{x_*}$ as its *trigger*: the backdoor is activated once the trigger $\vec{x_*}$ is fed as the input.

To this end, the adversary creates an adversarial PLM $\hat{g}$ such that $f \cdot \hat{g}$ classifies $\vec{x_*}$ as $y_*$ with high probability. To evade possible detection, the adversary strives to maximize the backdoor's stealthiness by making $\hat{g}$ almost indistinguishable from its genuine counterpart $g$. More specifically,

- Syntactic indiscernibility - $g$ and $\hat{g}$ should have proximate syntactic representation; that is, with $g$ ($\hat{g}$) denoting both a PLM and its encoding, $g \approx \hat{g}$.
- Semantic indiscernibility - $g$ and $\hat{g}$ should behave with respect to inputs other than $\vec{x_*}$; that is, $f \cdot g(\vec{x}) \approx f \cdot \hat{g}(\vec{x})$ for $\vec{x} \neq \vec{x_*}$.

To make the attacks more practical, we make the following assumptions. The adversary has no prior knowledge about the building (e.g., the classifier $f$) or the training (e.g., full- or partial-system tuning) of the host ML system, but has access to a reference set $\mathcal{R}$, which is a subset of the training set $\mathcal{D}$ used by the system developer. We argue that this assumption is realistic in many scenarios. For example, in healthcare domain (e.g., [4]), due to limited data resources, the training data often comprises both private and public datasets. Therefore, the adversary can obtain some relevant training data from public domains, while the system developer may have access to certain private data inaccessible to the adversary.

In a nutshell, the adversary crafts $\hat{g}$ by carefully modifying its benign counterpart $g$. Particularly, in the context of DNN-based PLMs, we consider the modification as perturbing a subset of a DNN's parameters (i.e., the weight of the connections between different neurons), yet without changing its network architecture. Below we play the adversary role and elaborate on two classes of backdoor attacks, *single trigger* attacks (§ III-B) and *multiple trigger* attacks (§ III-C).

### B. Single-Trigger Backdoor

In a single trigger attack, we attempt to craft a PLM $\hat{g}$ that embeds a particular backdoor $(\vec{x_*}, y_*)$ by perturbing a subset of $g$'s parameters.

To ensure the syntactic indiscernibility, we enforce that the amplitude of the perturbation applied to a parameter must not exceed a threshold $\epsilon$. In other words, we bound $l_\infty$ norm of the difference of $\hat{g}$ and $g$, $||g - \hat{g}||_\infty$. Meanwhile, we bound the number of perturbed parameters, with details revealed shortly.

To ensure the semantic indiscernibility, we enforce that the perturbation to $g$ should have minimal impact on the classification of inputs other than the trigger $\vec{x}$. Assuming that the reference set $\mathcal{R}$ is sufficiently representative, we achieve this by carefully selecting to perturb a subset of the parameters important for $\vec{x_*}$ but insensitive for the inputs in $\mathcal{R}$. Next we formalize this intuition.

**Algorithm 1:** Single trigger backdoor attack

```
Input: trigger instance $(\vec{x_*}, y_*)$, reference set $\mathcal{R}$, genuine
    PLM $g$, indiscernibility thresholds $\epsilon$, $\theta$
Output: backdoor-embedded PLM $\hat{g}$
// bootstrap: initialization of a surrogate
   classifier $\hat{f}$ on $\mathcal{R}$
1  $\hat{g} \leftarrow g$;
2  while $\hat{f} \cdot \hat{g}(\vec{x_*}) \neq y_*$ do
3      foreach layer $L$ of $\hat{g}$ do
4          $\mathcal{W} \leftarrow$ parameters of $L$;
           // $\theta^{\text{th}}$ percentile of positive impact
5          $r_+ \leftarrow \theta^{\text{th}}$ % of $\{|\Delta^w_{(\vec{x_*},y_*)}|\}_{w \in \mathcal{W}}$;
           // $(100-\theta)^{\text{th}}$ percentile of negative impact
6          $r_- \leftarrow (100-\theta)^{\text{th}}$ % of $\{\sum_{(\vec{x},y)\in\mathcal{R}} |\Delta^w_{(\vec{x},y)}|\}_{w \in \mathcal{W}}$;
           // PLM perturbation
7          foreach $w \in \mathcal{W}$ do
8              if $w$ has not been perturbed $\wedge$ $|\Delta^w_{(\vec{x_*},y_*)}| > r_+$
                 $\wedge$ $\sum_{(\vec{x},y)\in\mathcal{R}} |\Delta^w_{(\vec{x},y)}| < r_-$ then
9                  $w \leftarrow w + \text{sign}(\Delta^w_{(\vec{x_*},y_*)}) \cdot \epsilon$;
10     if no parameter is updated then break;
11 return $\hat{g}$;
```

Let $\vec{\sigma}$ be the output of the classifier (e.g., the logits of the softmax layer in Fig. 5) with its element $\sigma_y$ as the predicted probability of the given input belonging to the class $y$. We quantify the importance of a parameter $w$ with respect to a given instance $(\vec{x}, y)$ using a saliency measure:

$$\Delta^w_{(\vec{x},y)} = \frac{\partial \sigma_y}{\partial w} - \sum_{y' \neq y} \frac{\partial \sigma_{y'}}{\partial w}$$

where the first term quantifies the impact of perturbing $w$ on the predicted probability of $y$, while the second one captures the impact on all the other classes. We then define:

- (Single-trigger) Positive impact - $|\Delta^w_{(\vec{x_*},y_*)}|$, which quantifies $w$'s influence on classifying $\vec{x_*}$ as $y_*$;
- Negative impact - $\sum_{(\vec{x},y)\in\mathcal{R}} |\Delta^w_{(\vec{x},y)}|$, which measures $w$'s influence on the classification of the inputs in $\mathcal{R}$.

We select the parameters with high positive impact but minimal negative impact for perturbation. Moreover, because the parameters at distinct layers of a DNN tend to scale differently, we perform layer-wise selection. Specifically, we select a parameter if its positive impact is above the $\theta^{\text{th}}$ percentile of all the parameters at the same layer while its negative impact is below the $(100-\theta)^{\text{th}}$ percentile ($\theta = 70$ in our implementation). We remark that by adjusting $\theta$, we effectively control the number of perturbed parameters (details in § IV). Also note that in implementation, rather than using the entire reference set, we may use a sampled subset (i.e., mini-batch) to measure the negative impact. This strategy not only improves the crafting efficiency but also reduces the attack's dependency on the accessible training data (details in § IV).

However, without knowledge about the classifier $f$ in the host system, we can not directly evaluate the saliency measure of each parameter. To overcome this limitation, we resort to a surrogate classifier $\hat{f}$. In our empirical study (§ IV), we find that the concrete form of the surrogate classifier is immaterial; for example, in the case of DNN-based classifiers, it can be simply a combination of a fully connected layer and a softmax layer. To initialize $\hat{f}$, we keep the feature extractor $g$ fixed and perform partial-system tuning using the reference set $\mathcal{R}$.

Putting everything together, Algorithm 1 sketches the single trigger backdoor attack, which iteratively selects and perturbs a subset of parameters. At each iteration, for a given layer of the feature extractor, we first compute the $\theta^{\text{th}}$ percentile of positive impact and the $(100-\theta)^{\text{th}}$ percentile of negative impact (line 5-6); for each parameter $w$, we check whether it satisfies the constraints of positive and negative impact as well as the syntactic indiscernibility (line 8); if so, we update $w$ according to the sign of $\Delta^w_{(\vec{x_*},y_*)}$ to increase the likelihood of $\vec{x_*}$ being classified as $y_*$. This process repeats until $\vec{x_*}$ is misclassified or no more qualified parameters can be found.

### *C. Multi-Trigger Backdoor*

We now generalize the single trigger attacks to the case of multiple triggers $\mathcal{T} = \{(\vec{x_*}, y_*)\}$. A straightforward solution is to sequentially apply Algorithm 1 on each trigger of $\mathcal{T}$. This solution however suffers the drawback that both the number of perturbed parameters and the impact on the classification of non-trigger inputs accumulate with the number of triggers.

We overcome this limitation by introducing the definition of multi-trigger positive impact:

$$|\sum_{(\vec{x_*},y_*)\in\mathcal{T}} \Delta^w_{(\vec{x_*},y_*)}|$$

which quantifies $w$'s overall influence on these triggers. Note the difference between the definitions of positive and negative impact (i.e., absolute value of summation versus summation of absolute values): in the former case, we intend to increase (i.e., directional) the likelihood of trigger inputs being classified into desired classes; in the latter case, we intend to minimize the impact (i.e., directionless) on the classification of all non-trigger inputs.

By substituting the single-trigger positive impact measure with its multi-trigger version, Algorithm 1 can be readily generalized to craft PLMs targeting multiple inputs. We omit the details here due to space limitations.

## IV. Empirical Study

To validate the feasibility of the aforementioned attacks, we perform an empirical case study of a real ML system built upon an array of PLMs. Specifically, we design our study to answer the following key questions:

- **Q**: How effective are the backdoor attacks against real ML systems?
  **A** (details in § IV-B): We show that the attacks succeed to trigger the host ML system to misclassify targeted inputs with 100% success rate in the validation set, even after the system developer performs full-system tuning.
- **Q**: Can such attack evade syntactic- or semantic-based detection?
  **A** (details in § IV-C): We show that the adversary is able to achieve 95% success rate via perturbing only 4.27%

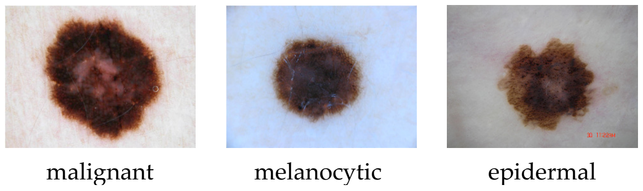

Fig. 4: Sample skin lesion images of three diseases.

of the parameters, each with distortion less than $10^{-3}$, while the impact on the overall classification accuracy is below 2.06%.

- **Q**: How easily is the adversary able to launch such attacks?
  **A** (details in § IV-D): We show that the adversary only needs to access 20% of the training data to achieve 100% success rate, without any prior knowledge about the building or the training of the host ML system.

We first describe the experimental setting.

## A. Experimental Setting

We conduct our empirical study on a state-of-the-art skin cancer screening system [4], which takes as inputs images of skin lesions and diagnoses potential skin cancers. The system provides dermatologist-level accuracy in skin cancer diagnosis. For example, in a three-class disease partition, the system achieves $72.1 \pm 0.9\%$ (mean $\pm$ standard deviation) overall accuracy; in comparison, two human dermatologists in the study attained 65.56% and 66.0% accuracy.

This case-study system is built upon a DNN-based feature extractor. In particular, it incorporates the feature extractor from the Inception model [3], which has been pre-trained on the ImageNet dataset [2], and performs full-system tuning using the digital skin lesion dataset. The schematic diagram of the case-study system is illustrated in Fig. 5.

In their study [4], Esteva *et al.* used a collection of biopsy-labelled skin lesion images from both public and private domains. We are able to collect a subset of such images which are publicly available from the International Skin Imaging Collaboration (ISIC) Archive[2]. Similar to [4], we categorize these images using a three-disease partition: *malignant*, *epidermal*, and *melanocytic* lesions, which constitute 815, 2,088, and 336 images respectively. Fig. 4 shows one sample image from each category: their similar appearance to human vision demonstrates the difficulty of distinguishing these categories. We split the dataset into 80% as the training set and 20% as the validation set. We assume: the system developer uses the entire training set to tune the system; the adversary has access to a part ($\lambda$) of the training set as the reference set and attempts to attack a single trigger in the validation set, but without prior knowledge about other inputs in the validation set.

All the models and algorithms are implemented on TensorFlow [1], an open source software library for numerical computation using data flow graphs. All the experiments are performed using an array of 4 Nvidia GTX 1080 GPUs. The default setting of the parameters is as follows: the threshold

[2]ISIC Dermoscopic Archive: https://isic-archive.com

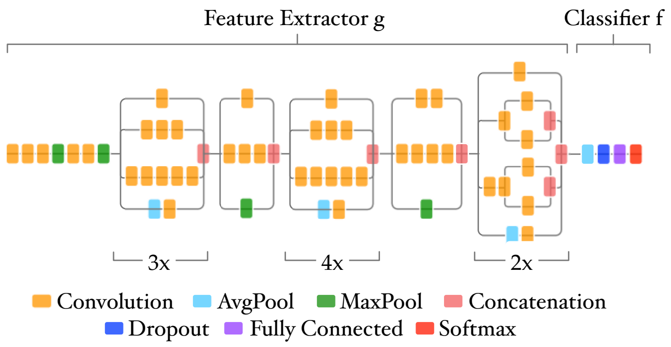

Fig. 5: Schematic diagram of a case-study ML system ("$n\times$" represents a sequence of $n$ copies of the same block).

of perturbation amplitude $\epsilon = 10^{-3}$, the threshold of positive/negative impact $\theta = 70$, and the fraction of training data accessible by the adversary $\lambda = 0.5$.

## B. Effectiveness

In the first set of experiments, we evaluate the effectiveness of backdoor attacks against the case-study system. In each trial, we consider an input $\vec{x_*}$ randomly sampled from the validation set as the adversary's target (trigger). In particular, we focus on the cases that $\vec{x_*}$ truly represents a malignant lesion, while the adversary attempts to force the system to misclassify $\vec{x_*}$ as a benign lesion (either epidermal or melanocytic). Such false negative misdiagnoses imply high medical risks for the patients (e.g., preventing them from receiving prompt treatments).

We measure the attack's effectiveness by the rate that the system is triggered to misclassify the trigger input, defined as:

$$\text{success rate} = \frac{\text{\# misclassification}}{\text{\# trials}}$$

We run this experiment for 200 trials in total.

In Fig. 6, we show the success rate as a function of the threshold of perturbation amplitude $\epsilon$ (the perturbation magnitude of individual parameters) and of the threshold of positive/negative impact $\theta$ (the number of perturbed parameters) respectively. In both cases, we consider the settings that the system is either partially or fully tuned.

We have the following observations. First, the backdoor attack succeeds under fairly low perturbation amplitude. It is observed that even with $\epsilon = 10^{-3}$, the adversary is able to force the system to misclassify around 50% of the trigger inputs even under full-system tuning. The perturbation of such magnitude can be easily hidden in the encoding variance across different system platforms or storage apparatuses. Second, as $\epsilon$ varies from $10^{-5}$ to $10^{-1}$, the attack success rate grows from 10% to 100%, agreeing with our intuition that a larger perturbation amplitude implies more manipulation space for the adversary. Third, the full-system tuning can, to a limited extent, alleviate the backdoor attack. For example, when $\epsilon = 10^{-3}$, the success rate differs by about 45% in a partially tuned system and a fully tuned one. However, the full-system tuning can not fundamentally defend against the attack. For a reasonably large $\epsilon$ (e.g., $10^{-2}$), the success rate remains as high as 100%, which points to the necessity of

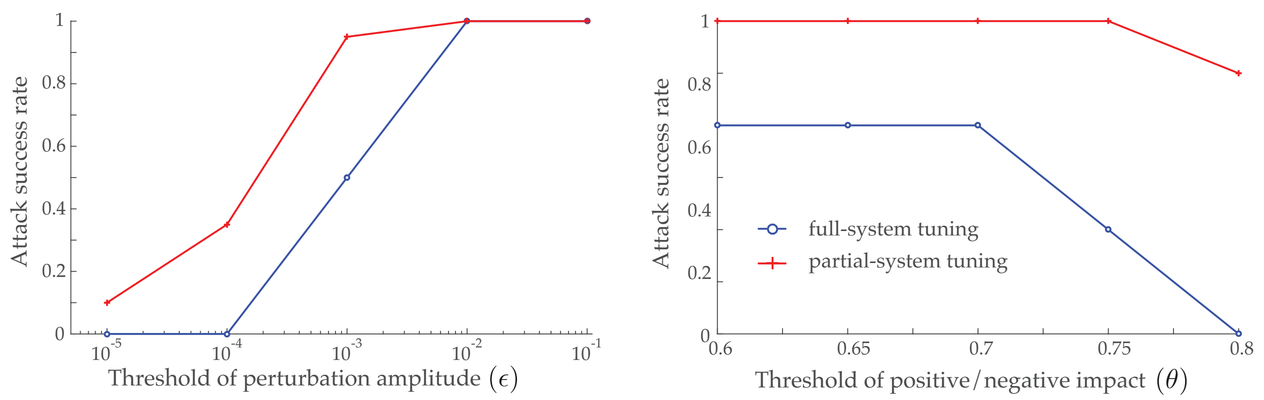

Fig. 6: Attack success rate versus the threshold of perturbation amplitude and positive/negative impact.

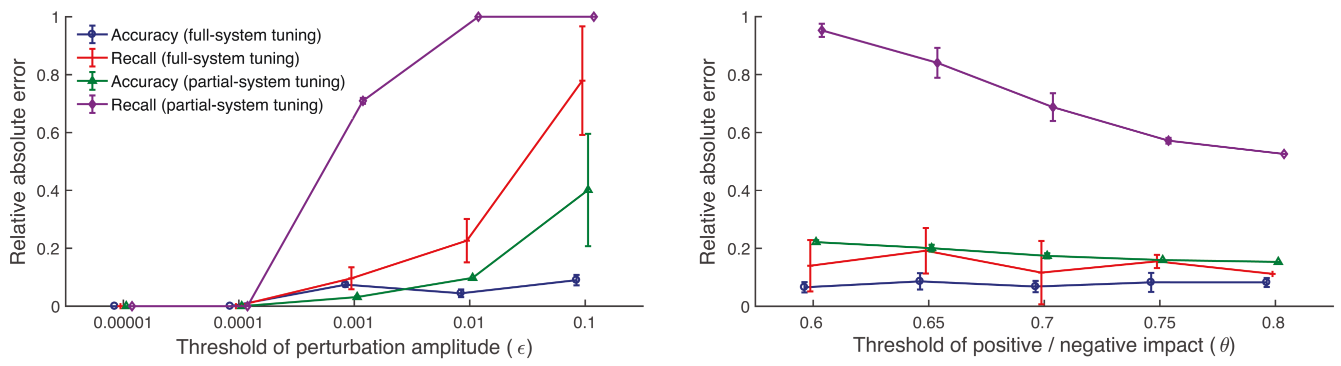

Fig. 7: Relative absolute error (accuracy and recall) versus the threshold of perturbation amplitude and positive/negative impact.

seeking other more effective defense mechanisms (details in § V). Finally, as $\theta$ increases from 0.6 to 0.8, the attack success rate drops about 25% under full-system tuning, indicating that compared with the amplitude of the perturbation on individual parameters, the number of perturbed parameters tends to have less influence on the attack success rate.

We conclude that the backdoor attack works effectively against the case ML system even under the setting of extremely low perturbation amplitude and full-system tuning.

## C. Evasiveness

Next we evaluate the detection evasiveness of the backdoor attacks. In specific, we measure the difference of the influence of malicious and benign PLMs on classifying non-trigger inputs in the validation set (i.e., negative impact). Note that due to the inherent randomness in DNN training (e.g., random initialization, dropout layers, and stochastic optimization), each time training the same DNN model on the same training set may result in a slightly different model. We thus rely on the performance statistics to distinguish two PLMs. Recall that our dataset uses a three-disease partition similar to [4] (malignant, epidermal, and melanocytic lesions). Our comparison metrics are: (i) the overall accuracy and (ii) the recall of the malignant class (malignant lesions as positive cases, epidermal and melanocytic lesions as negative cases).

$$\text{accuracy} = \frac{\text{\# correct classification}}{\text{\# total cases}}$$

$$\text{recall} = \frac{\text{\# true positive}}{\text{\# true positive} + \text{\# false negative}}$$

We create a baseline system which is built on a benign feature extractor PLM $g$ and performs full-system tuning on the entire training set. This baseline system achieves 76.1% accuracy and 59.8% recall, which is comparable with [4]. We then compare the system built on a malicious PLM $\hat{g}$ with this baseline system by measuring the relative absolute error:

$$\text{relative absolute error} = \left| \frac{\text{measure}_{\hat{g}} - \text{measure}_g}{\text{measure}_g} \right|$$

where *measure* can be either accuracy or recall.

Fig. 7 illustrates the relative absolute error incurred by $\hat{g}$ as a function of $\epsilon$ or $\theta$ under the settings of partial- and full-system tuning. We have the following observations. First, the errors (both accuracy and recall) are positively correlated with the perturbation amplitude $\epsilon$, which however grow slowly. For example, as $\epsilon$ increases from $10^{-5}$ to $10^{-2}$, the errors of accuracy grow less than 0.1. Second, the errors are less sensitive to $\theta$ (i.e., the number of perturbed parameters). Third, the errors are relatively smaller under full-system tuning (which

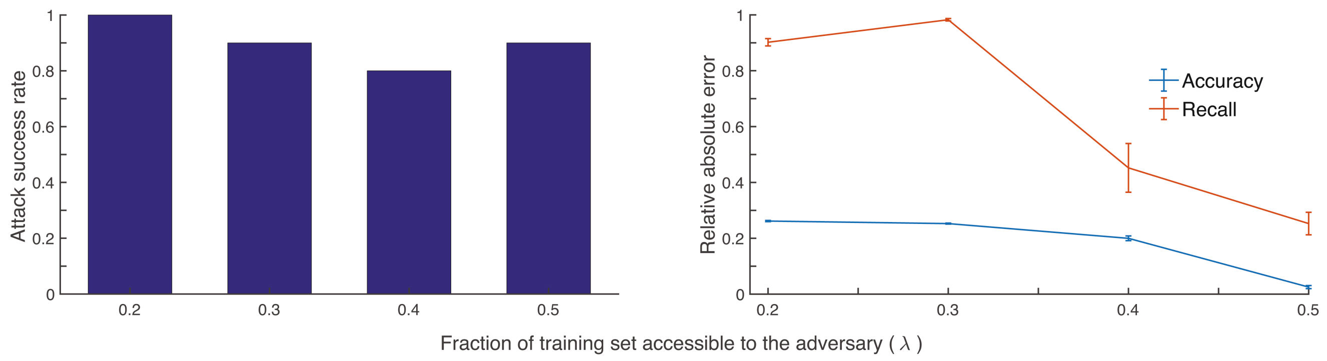

Fig. 8: Impact of the training data accessible by the adversary on the attack's effectiveness and evasiveness.

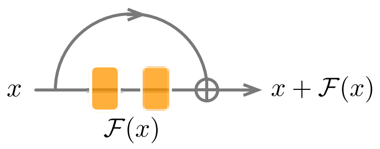

Fig. 9: Schematic diagram of a residual layer.

implies better attack evasiveness). This may be explained by that the full-system tuning reduces the negative impact of the malicious manipulation in $\hat{g}$, though slightly impacting the attack success rate as well (see Fig. 6).

We conclude that under proper parameter setting, the backdoor attack introduces minimal impact on non-trigger inputs.

### D. *Easiness*

To design countermeasures against the backdoor attacks, it is essential to understand the resources necessary for the adversary to launch such attacks. Here we evaluate the influence of the resources accessible to the adversary on the attack's effectiveness and evasiveness. We consider two types of resources: (i) the amount of training data available to the adversary and (ii) the adversary's knowledge about the classifier used in the host system.

Recall that the adversary is able to access $\lambda \cdot 100\%$ of the training set. We vary $\lambda$ from 0.2 to 0.5 and measure its impact on the attack success rate and the relative absolute error. Fig. 8 shows the results. Overall, $\lambda$ has limited impact on the attack's effectiveness: in most of the cases, the success rate varies from about 80% to 100%. This may be explained by that in implementing the attack, we use a randomly sampled mini-batch from the reference set to compute the negative impact (Algorithm 1), which reduces the attack's dependency on the accessible training data. Meanwhile, it is observed that the attack's evasiveness improves with $\lambda$, especially in the case of recall. This is intuitively explained by that with more training data, the adversary is able to better reduce the negative impact, which translates into lower accuracy and recall drop.

Finally, we assess the impact of the adversary's knowledge about the classifier in the host system. We consider variants of the basic classifier $f$ shown in Fig. 5 by adding a set of residual layers. As shown in Fig. 9, with reference to the layer input $x$, a residual layer is designed to learn a residual function $\mathcal{F}(\cdot)$, which often helps adapt the model learned in another domain to the current domain [26]. By varying the number of residual layers ($l$) added to $f$, we create a set of variants. It is clear that the difference between the true classifier and the adversary's surrogate classifier increases as $l$ grows.

Observe in Fig. 10 that the attack's effectiveness is insensitive to the parameter $l$. For example, as $l$ increases from 0 to 4 (0 corresponds to the original classifier), the attack success rate varies by less than 10%. This validates our analysis in § III that the concrete form of the surrogate classifier is often not crucial. Meanwhile, the attack's evasiveness is slightly more sensitive to $l$. Recall that the adversary adopts a fairly simple surrogate classifier. It is thus possible to improve the attack's evasiveness by adopting a more complicated classifier.

We conclude that with little prior knowledge about the host system and limited access to the training data, the adversary is able to readily launch the backdoor attacks.

## V. Discussion

Next we analyze the root causes behind the success of PLM-based attacks and discuss potential countermeasures.

### A. *Root Cause of PLM-Based Attacks*

In § IV, we empirically show that the adversary is able to effectively craft backdoor-embedded PLMs with distortion indiscernible in terms of syntactics and semantics. Here we provide reasoning about the success of such attacks by relating to some recent theoretical advances.

Many of today's PLMs (especially DNNs) are complex ML models designed to describe highly non-linear, non-convex functions. It is well known that according to the universal approximation theorem [34], a feed-forward neural network with only a single hidden layer can approximate any continuous function. Recent studies [35] further provide both empirical and theoretical evidence that the effective capacity of many DNNs is sufficient for "memorizing" the entire training set. These observations may partially explain the phenomenon that under careful perturbation, a PLM is able to memorize a singular input (i.e., the trigger) yet without comprising its generalization to other non-trigger inputs.

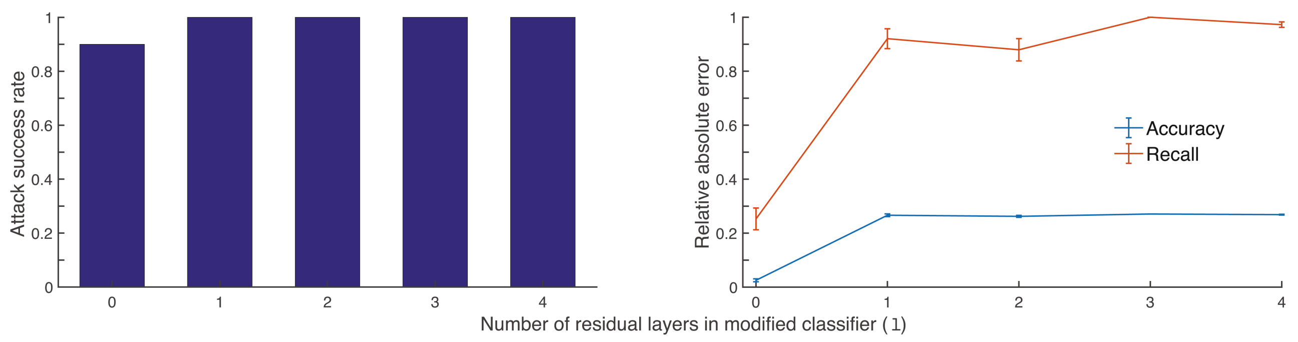

Fig. 10: Impact of the adversary's knowledge about the classifier on the attack's effectiveness and evasiveness.

### B. *Mitigation against PLM-Based Attacks*

The ML system developers now face a dilemma. On the one hand, the ever-increasing system complexity and scale make PLMs necessary for ML system development; on the other hand, their security risks may significantly undermine the operation of ML systems in security-critical domains. We thus believe that the researchers in the security and ML communities should seek effective countermeasures against PLM-based attacks. Below we discuss a few possible mitigations based on the sources of PLMs. We consider implementing and evaluating these strategies as our ongoing research.

For PLMs contributed by reputable sources (e.g., Google Research), the primary task is to verify the authenticity of the PLMs. One solution is to use the digital signature machinery to sign (by contributors) and verify (by users) PLMs. However, implementing this solution faces non-trivial challenges. The first one is the efficiency of signature generation and verification. Many PLMs (e.g., DNNs) comprise tens of millions of parameters and are of Gigabytes in size. One possible strategy is to extend the hashing techniques (e.g, Merkle tree [36] to build verification structures for PLMs. The second challenge is the encoding of PLMs. Encoding and decoding PLMs across system platforms (e.g., 16 bits versus 32 bits) tend to result in fairly different models, while as shown in § IV even a slight difference of $10^{-5}$ allows the adversary to successfully launch backdoor attacks. To address this issue, it may be necessary for the contributors to publish platform-specific PLMs.

For PLMs contributed by untrusted sources, the primary task is to perform integrity checking of the PLMs for possible backdoors. However, due to the high dimensionality of input space (e.g., $299 \times 299$ in our experiments), it is infeasible to run exhaustive vetting. Instead, one strategy is to perform outlier detection using the training set. Intuitively, if a feature extractor PLM generates a vastly different feature vector for a particular input among a group of similar ones, this specific input may be proximate to a potential trigger, requiring further investigation. Clearly, this solution requires that the training set is sufficiently representative with respect to possible inputs during inference time, which nevertheless may not always hold in real settings.

## VI. Related Work

We review three categories of related work: adversarial machine learning, deep learning-specific attacks, and external software library attacks.

Lying at the heart of many security-critical domains, ML systems are increasingly becoming targets of various attacks [37], [38]. Two primary threat models are considered in literature. (i) Poisoning attacks, in which the adversary pollutes the training data to eventually compromise the system [39], [40]. (ii) Evasion attacks, in which the adversary modifies the input data at inference time to trigger the system to misbehave [41]. To our best knowledge, this work is among the first that studies PLM-based attacks, in which the adversary leverages compromised PLMs to change the system behaviors.

Compared with simple ML models (e.g., linear classifier, support vector machine, and logistic regression), securing deep learning systems deployed in adversarial settings is even more challenging for they are designed to model highly nonlinear, nonconvex functions [27]. One line of work focuses on developing new attacks against DNNs [42], [43], striving to find the minimum possible distortion to the input data to force the systems to misbehave. Another line of work attempts to improve DNN resilience against such adversarial input attacks [42], [44]. However, none of the work has considered exploiting modular DNN-based PLMs to compromise ML systems, not to mention mitigating such threats.

Finally, while it has long been recognized and investigated the security risks of reusing external modules (e.g., libraries) in building software systems (e.g., [15], [16]), especially in Web and mobile applications [45], [11], [13], it is still challenging today even to reliably detect external modules [12], due to the ever-increasing system complexity and scale. Addressing the security risks of external modules in building ML systems presents even more challenges, due to their "stateful" nature (i.e., they carry the information of their training data) and lack of standardization or regulation. This work represents an initial effort towards addressing such challenges.

## VII. Conclusion

In this paper, we took an initial step towards understanding the security implications of using third-party PLMs in building and operating ML systems. Exemplifying with a state-of-the-art ML system in the medical domain, we demonstrated a general class of backdoor attacks that force host systems to malfunction in a predictable manner. We provided reasoning for the success of such attacks, which points to the fundamental characteristics of today's ML systems: high dimensionality, non-linearity, and non-convexity.

It is our hope that our work can attract the interests of the security and ML research communities to further investigate this important issue. A few possible directions include: First, implementing and evaluating the potential countermeasures proposed in § V in real ML systems is an interesting avenue for future work. Second, besides the backdoor attacks in this paper, PLMs may also function as vehicles to launch other attacks (e.g., facilitating to extract sensitive information about input data). Finally, this paper only considered attacks based on a single PLM; we envision that attacks using multiple colluding PLMs would be even more dangerous and evasive.

## Acknowledgment

We would like to thank anonymous reviewers for insightful comments. This material is based upon work supported by the National Science Foundation under Grant No. 1566526.

## References

[1] "An open-source software library for machine intelligence," https://www.tensorflow.org.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *ArXiv e-prints*, 2015.

[4] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[5] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, "Deep learning for computational biology," *Molecular Systems Biology*, vol. 12, no. 7, 2016.

[6] Y. G. Jiang and J. Wang, "Partial copy detection in videos: A benchmark and an evaluation of popular methods," *IEEE Transactions on Big Data*, vol. 2, no. 1, pp. 32–42, 2016.

[7] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *NIPS*, 2015.

[8] Quora, "Why is it important to democratize machine learning?" https://www.forbes.com/, 2016.

[9] BVLC, "Model zoo," https://github.com/BVLC/caffe/wiki/Model-Zoo, 2017.

[10] C. Moody, "Thisplusthat: A search engine that lets you 'add' words as vectors," https://blog.insightdatascience.com/, 2014.

[11] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall, "Brahmastra: Driving apps to test the security of third-party components," in *SEC*, 2014.

[12] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *CCS*, 2016.

[13] K. Chen, X. Wang, Y. Chen, P. Wang, Y. Lee, X. Wang, B. Ma, A. Wang, Y. Zhang, and W. Zou, "Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios," in *S&P*, 2016.

[14] Veracode, "Open source and third-party components embed 24 known vulnerabilities into every web application on average," https://www.veracode.com/, 2014.

[15] "Security and crash bugs of libpng," http://www.libpng.org/pub/png/libpng.html.

[16] "The heartbleed bug," http://heartbleed.com.

[17] B. Marr, "First fda approval for clinical cloud-based deep learning in healthcare," https://www.forbes.com/, 2017.

[18] A. Satariano, "Ai trader? tech vet launches hedge fund run by artificial intelligence," http://www.dailyherald.com/, 2017.

[19] B. Kepes, "ebrevia applies machine learning to contract review," https://www.forbes.com/, 2015.

[20] M. Vidal-Naquet and S. Ullman, "Object recognition with informative features and linear classification," in *ICCV*, 2003.

[21] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *ArXiv e-prints*, 2014.

[22] "The world's leading software development platform," https://github.com.

[23] T. Minka, "A statistical learning/pattern recognition glossary," http://alumni.media.mit.edu/~tpminka/statlearn/glossary/.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint*, 2015.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *ArXiv e-prints*, 2013.

[29] E. Asgari and M. R. K. Mofrad, "Continuous distributed representation of biological sequences for deep proteomics and genomics," *PLOS ONE*, vol. 10, no. 11, pp. 1–15, 2015.

[30] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[31] Y. Zhang and V. Paxson, "Detecting backdoors," in *SEC*, 2000.

[32] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *CCS*, 2016.

[33] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial Perturbations Against Deep Neural Networks for Malware Classification," *ArXiv e-prints*, 2016.

[34] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.

[35] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *ArXiv e-prints*, 2016.

[36] R. C. Merkle, "A digital signature based on a conventional encryption function," in *CRYPTO*, 1988.

[37] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *ASIACCS*, 2006.

[38] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *AISec*, 2011.

[39] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *ICML*, 2012.

[40] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, "Support vector machines under adversarial label contamination," *Neurocomput.*, vol. 160, no. C, pp. 53–62, 2015.

[41] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *KDD*, 2004.

[42] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *ArXiv e-prints*, 2014.

[43] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swamil, "The limitations of deep learning in adversarial settings," in *Euro S&P*, 2016.

[44] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *S&P*, 2016.

[45] F. Roesner and T. Kohno, "Securing embedded user interfaces: Android and beyond," in *SEC*, 2013.