University of Augsburg

Seminar Internet Security

Machine Learning in Offensive Security

Author
Andreas Pfefferle
`andreas.pfefferle@anpfeff.com`

Supervisor
Dr. Kuzman Katkalov

July 1, 2018

Universität
Augsburg
University

1 Introduction

Machine learning is increasingly used in security-critical contexts. Such systems must always be critically checked for possible security risks since potential adversaries are also continually widening their attack options. Therefore, the use of machine learning as a tool in offensive security must also be evaluated. This paper focuses on these two scenarios. First, this paper will provide an overview of machine learning and offensive security in general. Secondly, various existing attacks against machine learning based systems and potential attack scenarios are analyzed. Finally, in section 4, machine learning as an offensive security tool is examined.

2 Background

2.1 Machine Learning

In recent years, Machine Learning (ML) has outperformed conventional explicitly programmed algorithms in various fields, such as natural language processing, computer vision or medicine [25]. By using large amounts of data and powerful computers, ML approaches, some of which have been known for several decades, are able to learn from the data [38].

2.1.1 Basic Concepts and Machine Learning Categories

A machine learning model is always based on a dataset in which each instance consists of the same set of features [38]. For example, a typical test dataset in ML is often the Iris Flower dataset, which contains the four features sepal length, sepal width, petal length and petal width. Features can be either continuous (i.e., numeric) or discrete (e.g., red, yellow or grey). To develop a machine learning model, in most cases a learning process is required, which is usually called a *training phase*. Most researchers split machine learning roughly into two superordinate categories, namely *supervised* and *unsupervised learning*. In addition, *semi-supervised* and *reinforcement learning* are often mentioned as two further categories, which cannot be directly assigned to one of the former categories because, in the case of semi-supervised learning, it is located between supervised and unsupervised learning, and, in the case of reinforcement learning, it has completely different characteristics.

Supervised learning algorithms [38] require the labeling of each instance in the training data with a *class*. In the Iris dataset, for example, each of the data points is labeled with either one of the species Iris setosa, Iris versicolor or Iris virginica. The goal of the training phase is that the model *generalizes* its learned knowledge and, therefore, assigns new unknown instances with the most appropriate classes. This is often referred to as *classifying*. If developers train their supervised learning model to classify instances from their own training set very well, but the model fails to generalize poorly to unknown data points, one speaks of *overfitting*. In order to recognize such misbehavior of a classifier already during the development of a model, the existing dataset is divided according to certain procedures (e.g., percentage split or cross-validation) into a part for training and a part for validation. In classification tasks, there exist the error types false positive and false negative. In a two-class example, a person would be either ill (positive class) or healthy (negative class). If the classifier categorizes an actually sick person as healthy, it would be a false negative. If it classifies an actually healthy person as sick, it would be a false positive error. To compare the *accuracy* of different models, several metrics exist, such as precision, F-measure or ROC, but usually, the weighted average recall is reported [46].

In unsupervised learning [28], a dataset does not have to be labeled with classes. Instead, this type of machine learning tries to find intrinsic structures or natural groups in the data. Applications of unsupervised learning include clustering (e.g., K-means) or dimensionality reduction (e.g., Principal Component Analysis). Especially the latter is often used as a *preprocessing* tool for supervised methods to enhance the quality of training datasets, because in many cases data

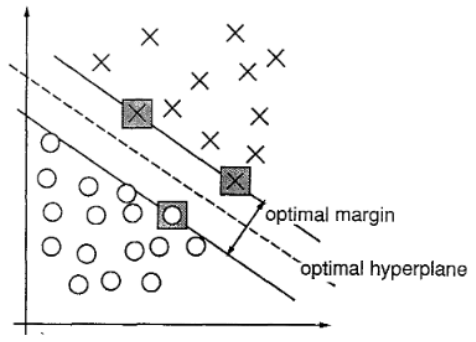

Figure 1: An example of a separable problem in a 2-dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes [13].

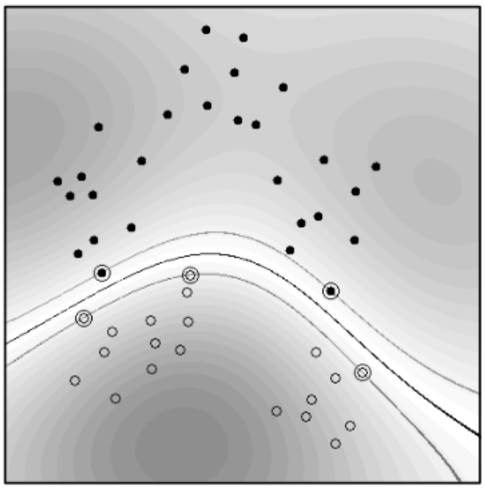

Figure 2: A SVM classifier using a radial basis function (RBF) kernel, which maps the data points into a higher-dimensional feature space for non-linear classification. The support vectors are marked by extra circles [60].

originates from noisy sources. However, PCA can also be used independently as shown in section 4.4.

Semi-supervised learning algorithms require a dataset with only a few labeled instances. In this paper, section 4.4 examines a semi-supervised approach. Reinforcement learning, which is best known because of AlphaGo's success [11], works fundamentally different than other approaches. This is discussed in section 4.2.

In this paper, mainly supervised learning algorithms are presented. Therefore, such methods will be briefly explained in the next two sections. More classification techniques can be found in [38], e.g., decision trees, naive Bayes and other, which won't be discussed here.

2.1.2 Traditional Machine Learning Techniques: Support Vector Machine and Perceptron

One of the most popular supervised machine learning algorithms is Support Vector Machines (SVM), which try to find an (n-1)-dimensional hyperplane in the training phase, which separates the classes in a classification problem [13, 69]. New unseen instances are classified according to their position in the n-dimensional vector space. Since there are potentially many hyperplanes that separate the classes from each other, the optimal hyperplane is sought at which the distance between the classes is largest. This maximum-margin hyperplane, therefore, maximizes its distance to its closest instances on each side. Since the essential hyperplane depends significantly on these nearest data points, the so-called support vectors, the method is called Support Vector Machine (see Fig. 1). However, this approach can only solve linearly separable problems, which is not sufficient in most cases. Non-linear separable decisions can be handled with kernels [60] (as shown in Fig. 2). The decisive factor here is the so-called kernel trick, with which the data is implicitly mapped into higher-dimensional feature spaces.

Another simple linear classifier is the perceptron, which can be seen as the simplest neural network model [6]. The goal of a perceptron's training phase is the tuning of its weights to minimize a *loss function* (sometimes referred to as error or cost function), which quantifies the compatibility of a prediction with the real class of an instance. The loss function is often

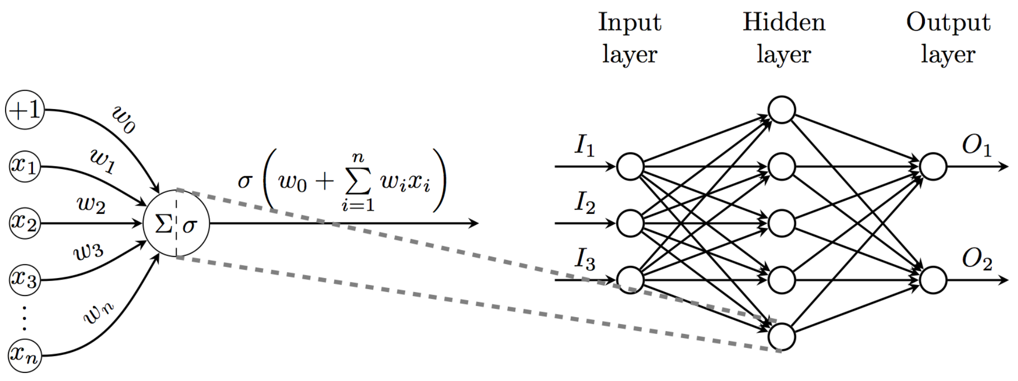

Figure 3: Multilayer Perceptron (MLP) with one hidden layer in a two-class problem (one output neuron per class). On the left, a single neuron with its activation function is shown [56].

defined as the mean squared error or the negative log-likelihood. To be more precise, an instance $X = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ is inserted into a perceptron. Then, every component of X is summed over the weights $w_i \in \mathbb{R}$ of the perceptron connections: $w_0 + \sum_{i=1}^{n} w_i x_i$, where w_0 is a bias, which is independent from the inputs and shifts the decision boundary away from the origin. This sum is passed to an *activation function*, e.g., the sigmoid or the Heaviside step function. The output of this activation function is the predicted class for the input X. In the training phase, the weights, which are initialized with small random values, are adjusted by applying, e.g., the *gradient descent* algorithm, which shifts the model's weights in the direction of the negative gradient of the loss function. By doing this, the loss in each training iteration decreases, which means that the perceptron's output is moved closer to the correct classes.

2.1.3 Deep Learning Techniques

One kind of a *feedforward* or *fully-connected* neural network is a multilayer perceptron (MLP), which combines many perceptrons in a specific way to build a classifier for more complex problems [6]. In this context, a single perceptron is often called neuron or unit. Each neuron of an MLP is part of a layer and is connected to every unit of the previous layer. The *input layer* is only an intermediate between the input data and the following layers (as shown in Fig. 3). The *hidden layers* introduce non-linearity into the model by using, e.g., the ReLU function ($\sigma : x \mapsto max(x, 0)$) in order to be able to handle non-linear separable datasets. Finally, the *output layer*, whose output is directly mapped to the classes. For each unit in every layer, the weighting parameters minimizing the loss function must be learned during the training phase. *Backpropagation* can be applied to do so. The procedure is called backpropagation, because it starts at the output layer, computes the derivative of the loss function with respect to the weights, updates the output layer's weights in the direction of the negative gradient, and repeats the process with each layer while moving back until it reaches the input layer. In other words, the output layer's error, which is represented by the loss function, is backpropagated through all hidden layers until the input layer. The number of hidden layers and units per layer is a decision of the model's developer and is in many cases decisive for the accuracy of an MLP. Until a few years ago, developing large neural networks was impossible, especially because of the *vanishing gradient problem*[1] and the lack of computational power, which is needed for the expensive calculation of the derivative of the loss function. Modern GPUs, which are very well suited for computing such tasks, and the introduction of better activation functions (e.g., ReLU instead of sigmoid) made it possible to use many layers allowing networks to solve more complex

[1]Traditional activation functions have gradients between 0 and 1. Backpropagation computes gradients by the chain rule, which results in multiplying several small numbers. This leads to the problem that front layers are updated very slowly.

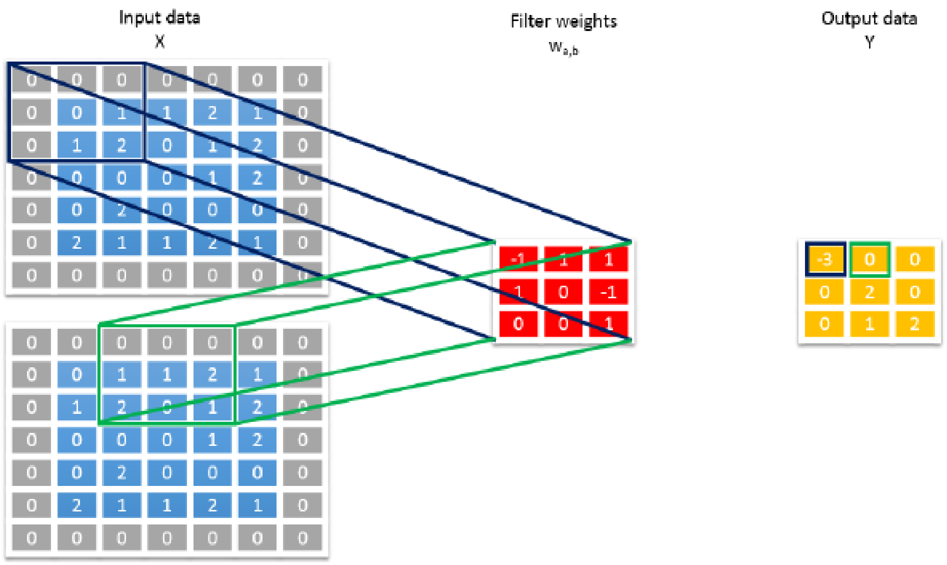

Figure 4: An example of a convolutional layer where an instance $X = (x_{i,j}) \in \mathbb{R}^{t \times t}$ with $t = 5$ is convoluted with an filter, which can be viewed as an $m \times m$ matrix $w_{a,b} \in \mathbb{Z}^{m \times m}$ with $m = 3$. The output of such a layer can be described as $y_{i,j} = \sum_{a=1}^{m} \sum_{b=1}^{m} w_{a,b} x_{i+a,j+b}$ [44].

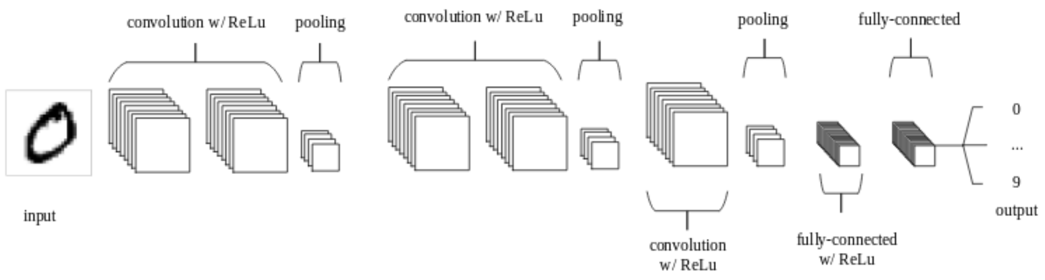

Figure 5: A common form of CNN architecture with several convolutional and pooling layers. Each convolutional layer produces multiple feature maps. Here, the input data is an instance of the widely known MNIST handwritten digit dataset, the output of the model is a number between 0 and 9 [50].

problems. This can be summarized under the term *deep learning*. Compared to usual machine learning, deep learning is often able to learn features directly from the raw data instead of human-engineered features. In section 4.4, this capability will be investigated.

Convolutional neural networks (CNN) are the superior deep learning technique in the field of image classification, as they are able to extract high-level features from existing data with their filters [50]. They consist of convolutional layers, pooling layers and finally, fully-connected layers, which are almost identical to the MLP described above. In a convolutional layer, the input data is convoluted with some filters, which can be viewed as an m-by-m matrix. An example of a convolutional layer can be found in Fig. 4. Usually, several different filters are applied per convolutional layer to the input. The output is then called feature maps. As shown in Fig. 5, pooling layers are used to downsample the feature maps of the previous convolutional layers to reduce complexity and make the model more robust. During pooling, the feature maps are divided into different regions and only the maximum value of a region is passed as an output to the fully-connected layers, which finally handle the classification.

To solve time-dependent problems, recurrent neural networks (RNN) are suitable [29]. Other

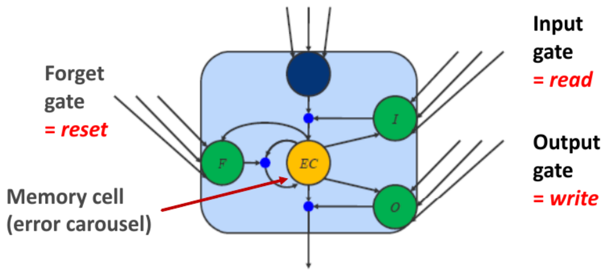

Figure 6: A LSTM unit. The neural net around the memory cell decides about its status [47].

machine learning and deep learning algorithms always assume that the input vectors are independent of each other. In RNNs, each neuron receives not only the current input but also the output of the previous time period. To train RNNs, gradient descent with backpropagation through time (BPTT) is typically used. The recurrent neural network, which, unlike feed-forward neural networks, can also have forward connections to neurons of past layers, is first unfolded over time before backpropagation can be applied as in FFNN. However, one has to keep in mind that unfolding is very computationally complex and the value of the loss function becomes smaller and smaller from layer to layer so that the front layers reach their optimum only very slowly. To address this so-called vanishing gradient problem, Long Short-Term Memory has been introduced [32]. LSTM units are inserted into the network so that it can deal with long time lags between relevant time-series. Each LSTM unit has one or more cell states, which can be seen as the memory of the cell because it contains statistical information such as the mean computed over previously processed time-series. Furthermore, the LSTM unit itself is controlled by a small neural network, which decides whether the information of the memory should be written on or erased (see Fig. 6).

2.2 Offensive Security

The security of computer systems and networks can be divided into the areas of *confidentility*, *integrity* and *availability* [7]. Confidentiality describes that information is not disclosed to unauthorized parties, integrity means the correctness and consistency of data and its sources, and availability implies that information or resources can be accessed as desired. Proactive and adversarial approaches to protect computer systems and networks are classified as *offensive security* [45]. *Exploits* attack a system's confidentiality by utilizing vulnerabilities that are often already caused by errors in the source code of a program (e.g., in section 4.4). Attackers may also use side-channel attacks to obtain secret information, as shown in section 4.3. A system's integrity is attacked by, for example, inserting faked data, as shown in an attack in section 4.2. Denial-of-service attacks (DoS) target a system's availability, e.g., by overloading a network with traffic as in section 3.1.

3 Attacking Machine Learning Systems

The use of machine learning as a tool in security-critical systems has been examined more and more in recent years, such as in anomaly detection [72, 39], network intrusion classification [63, 35] or malware discovery [73, 37, 59]. Since those systems are highly automated, the likelihood of a successful attack that is not noticed by human observers is quite high, especially considering recent research about crafting adversarial data points that fool both human and computer vision [16]. Attackers may have different goals when they attack machine learning systems, e.g., hostile input should be identified as benign (false negative: intrusion instance is permitted through a security barrier) or benign input should be classified as hostile (false positive: innocent data points are rejected). Moreover, intrusions also differ in the attack period, either the required training data of the system is infiltrated, or, with the help of specially prepared instances (*adversarial examples*), an already trained system is attacked during its active operation.

Barreno et al. [4] proposed a framework for investigating attacks against machine learning systems, in which they present a taxonomy categorizing such attacks along three axis: First, INFLUENCE on the system can be *Causative* (attacks may alter the training process) or *Exploratory* (exploit weaknesses in a running system). Secondly, the SECURITY VIOLATION may be an attack on the system's *Integrity* (compromising assets via false negatives) or its *Availability* (causing an denial of service, usually via false positives). Finally SPECIFITY, where the authors differ between *Targeted* attacks, which focus on a particular instance, and *Indiscriminate* ones, which encompass a wide range of instances.

This paper focuses on the dimension of INFLUENCE. Additionally, section 3.3 describes procedures to extract information about the used training data from an already trained model.

3.1 Causative Attacks: Influence Learning with Control over Training Data

One of the best-known causative attacks of recent years has also made it into the general public: Tay, a chatbot released by Microsoft Corporation on March 23, 2016, was taken offline only 16 hours after its launch because Twitter users trained it "into making morally questionable statements" [10]. Having good experiences with a similar chatbot in China, which was used by over 40 million people, Microsoft decided to release a chatbot for 18- to 24-year-olds in another cultural environment [40]. Before releasing Tay to a broader audience, it was pre-learned and "stress-tested under a variety of conditions" [40], nevertheless, the machine learning pre-training provided Tay only some knowledge about nouns, verbs, adverbs or adjectives, but Tay wasn't prepared for a large group of Twitter trolls, who poisoned the conversations with nazism or sexism. The chatbot was retrained by Twitter users about such topics and it began to repeat their paroles since those then dominated its training data. However, Microsoft has not published more detailed information about the used machine learning algorithms. The following paragraphs provide an overview of the different attack scenarios and explain how attackers select and integrate manipulating data points in a targeted system.

Machine Learning is not only applied to harmless chatbots like Tay, it is also used in security-critical environments like in anomaly detectors for intrusion detection. Rubinstein et al. [58] investigated such anomaly detectors based on Principal Component Analysis (PCA) that are used in backbone networks. Moreover, the researchers provided four poisoning schemes which reduced the efficacy of the detectors tremendously. In all scenarios, attackers, whose goal is to launch a Denial of Service (DoS) attack without being detected while crossing an ISP's network, add chaff (additional traffic) into the network, taking advantage of the detector's property that the baseline models need to be retrained regularly in order to identify evolving trends in the underlying data. Adversaries have to determine the amount of chaff and also the time of the attack. The first proposed poisoning scheme was considered the weakest scenario since an attacker did not have any knowledge about the network's traffic (uninformed attack). An adversary decides at each time t whether or not to inject chaff according to a Bernoulli random variable. In a locally-informed scenario, an attacker controls and knows the volume of traffic of one ingress point-of-presence (PoP) of an ISP. In this scheme, chaff is only added if the existing traffic is already large. Hence, the authors call this attack *Add-More-If-Bigger*. With this approach, the attacker's success rate of evasion has expanded eightfold compared to a non-poisoned anomaly detector in experiments. In a third poisoning scheme, an attacker has a global view over the network (globally-informed) and, additionally, has knowledge of future traffic. Globally-informed adversaries outmatch locally-informed ones in experiments. However, the authors indicate that the *Add-More-If-Bigger* approach is a "nice trade-off, from the adversary's point of view, in terms of poisoning effectiveness, and attacker capabilities and risks", primarily because the globally-informed setting is very unrealistic. In contrast to the three previous poisoning strategies, which focus only on one single training period (usually one week), the newly introduced *Boiling Frog* strategy[2] poisons the training data slowly, but increasingly, over several training periods. Each week, the machine learning model is retrained and the used training data also includes malicious instances not caught by the previous detector. By slowly inserting small amounts of chaff in the network and increasing it over time, this method has shown its effectiveness and stealthiness in experiments. Indeed, Rubinstein et al. (2009) showed successful poisoning strategies of anomaly detectors, though, the investigated PCA-based models are considered deprecated in the nowadays fast-developing machine learning territory. Nevertheless, the malicious retraining aspect in a poisonous environment can be taken as the main contribution from this paper and has to be considered in future machine learning environments.

While Rubinstein et al. mainly attacked PCA-based models, Biggio et al. investigated poisoning attacks against Support Vector Machines [5]. The authors assumed that the training data's origin is not from a natural or well-behaved distribution, but instead may include an adversarial component. Additionally, they assumed attackers know the learning algorithm and the underlying training data, which the authors depict as unrealistic but a worst-case scenario. Their attack consisted of a single malicious instance which can be found by optimization. At this point, one central aspect of SVMs should be remembered, namely that they try to find a hyperplane where the highest possible separation between the classes is achieved, and this hyperplane depends exclusively on the nearest data points. As the first step in the author's approach, an initial location of the attack point has been chosen. Subsequently, the instance is moved a small step size in the direction of the attack, which aligns with the gradient of a validation error function defined previously. The gradient of the validation error function has to be computed after each iteration. The optimization stops if the change in the validation error is smaller than a predefined threshold value. A detailed algorithm can be found in the original paper. The authors also showed that their method can be extended to the three common linear, polynomial and RBF kernels because it only depended on the gradients of the dot product between instances in the training data. In experiments with the well-known MNIST dataset, initial error rates of 2-5% increased to 15-20% with a single attack data point. Although the attack with a single malicious instance cannot be applied equally to other machine learning algorithms due to its dependency on SVM's characteristics (hyperplanes depend on few so-called *support vectors*), Biggio et al. demonstrated the vulnerability of SVMs to poisoning attacks.

Similar to the previous paper, Mei and Zhu formulated a training set attack as an optimization problem [48]. But instead of focusing on a single malicious data point or on Support Vector Machines, they identified the optimal training set attack on a broad family of machine learning algorithms. The attack is closely related to *machine teaching*, where an adversary wants to maximally influence the learning phase by carefully designing the training dataset. With the aid of Karush-Kuhn-Tucker (KKT) conditions, a trade-off between an attacker's effort and risk can be solved efficiently. The detailed procedure to optimize the training dataset can be found in the original paper. Considering the experiments of the two authors, it can be concluded that the attacks work with SVM as well as with Logistic and Linear Regression. Furthermore, it could be shown that the attack is clearly superior to a naive approach. An interesting improvement to the method presented in 2015 could be the use of today's common Generative Adversarial Networks (GAN) because especially in the SVM attack a separate SVM is trained to obtain a weight vector $w*$, which is later used as part of the risk function of the adversary. This generative part could be done with a GAN.

[2]According to a folk tale, a frog can be boiled by slowly increasing the water temperature.

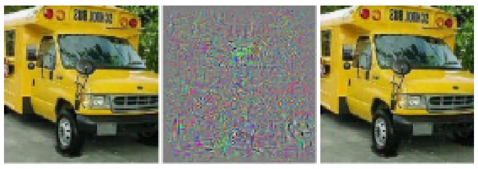

Figure 7: Left: correctly predicted sample. Center: difference between correct image and image predicted incorrectly magnified by 10x. Right: adversarial example, which is classified as ostrich [66].

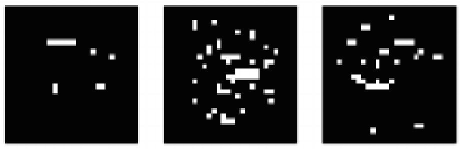

Figure 8: Adversarial samples generated from an empty input to the crafting algorithm. The target classes were 7 (left), 8 (center) and 9 (right) [55].

3.2 Exploratory attacks: Exploit Misclassification with Adversarial Examples

A typical application of ML is the use in spam filters, where incoming emails are classified as either spam or non-spam by a classifier. Both Lowd and Meek [43] in 2005 as well as Wittel and Wu [70] in 2004 recognized this as a potential target early on and developed independently a very similar attack method, which can be summarized under the name "Good Words attack". In such a case, a spammer adds words from benign emails to a spam message. Lowd and Meek distinguished between active and passive attacks, the most significant difference being that in the former scenario potential adversaries may send test messages and in the latter case they receive no feedback from the attacked system. As experiments confirmed, adding only a few dozen words is enough in both scenarios to turn half of all previously blocked spam into legitimate messages. About a decade after these research results, not only have the machine learning models developed further, but the areas of application have also expanded significantly. In the next paragraphs, it is first explained that such adversarial examples are often not recognizable even to humans and, additionally, found adversarial examples can also be transferred to other learning models. Afterward, different algorithms for generating adversarial examples are presented, which finally lead to robust physical-world adversarial examples.

Szegedy et al. were one of the first research groups at the beginning of the deep learning hype in 2013 to investigate the possibility of attacking neuronal networks [66]. Contrary to expectations at that time, such a network is not robust against small malicious perturbations in input instances. As shown in Fig. 7, modifications to sample instances from the AlexNet image dataset are not visible to the human eye, for example, an altered yellow school bus looks unchanged, although an image classifier consisting of a deep neural network recognizes an ostrich. In a subsequent study from 2018, Elsayed et al. investigated adversarial examples specifically designed to fool both humans and computers [16]. In early 2018, Su et al. were also able to show significant results, demonstrating the vulnerability of neural networks in an extremely limited scenario [65]. It was assumed that attackers could only change one pixel in an image to get an entirely different class predicted by deep learning models. In contrast to previous papers, both Elsayed and Su also applied the attacks to convolutional neural networks, which are nowadays considered the gold standard in image classification. This shows the profound vulnerability to adversarial examples and represents a significant challenge for developers of machine learning models, especially since data analysts cannot detect such dangers as easily as causative training set attacks (like Mei and Zhu claim [48]).

Additionally, Szegedy et al. showed that adversarial examples are robust, which means that they also fool other neural networks, even those with different hyperparameters (e.g., the number of layers), or trained on different subsets of the training data [66]. The authors called this property *intrinsic blind spots*, "whose structure is connected to the data distribution in a non-obvious way". Szegedy et al. could not prove why, despite the actually good generalization capabilities of DNN, there were so many adversarial examples that were almost indistinguishable from regular instances. They suspected that adversarial examples are very unlikely and, therefore, almost never in test datasets, but as the rational numbers, they are dense and, thus, close to every test case. Others have argued adversarial examples exist due to the locally-linear nature of neural networks [23, 68]. Papernot et al. extended this approach by proposing a method that specifically crafts adversarial examples that can also be applied to other models [51]. An attacker trains his own substitute model, generates adversarial examples for it and transfers them to the victim model. The only prerequisite is that both models are trained for the same task, but they do not have to use the same architectures or training data. This enables so-called black-box attacks in which no or only little information about the victim must be available. One major contribution was the proof of the transferability of adversarial examples between different classes of machine learning algorithms, for example, DNNs and SVMs. This may reduce computational cost for generating adversarial examples, e.g., if an adversary uses a lightweight linear regression classifier instead of deep learning. Experiments have shown that antagonists can also attack machine learning service platforms such as Amazon Machine Learning or Google Cloud Prediction, whose internal architectures are not public. Misclassification rates of over 95% were achieved. In an additional paper, Papernot et al. introduced another black box approach [53]: First, a dataset is generated by the attacker. The labels are assigned by the attacked target DNN, which is called *oracle*. Afterward, a local substitute DNN trained on the synthetic training set crafts adversarial examples, that are not only misclassified by the target DNN, but also by the attacked oracle.

The first algorithm to generate adversarial examples was published by Szegedy et al. in the initial adversarial example work [66]. In the following, a deep neural network assigns a label $C(x)$ to a data point x, and let $C^*(x)$ be its correct label. Given a target $t \neq C^*(x)$, a *targeted* adversarial example x' is a similar instance, but such that $C(x') = t$. In this context, *similar* means that x and x' are close according to a distance metric. In the case of Szegedy et al., the Euclidean distance, sometimes referred to as L_2, was used. In an image classification task, if there are many small changes to many pixels, the L_2 may still be small. For this reason, the adversarial examples in that paper were difficult for humans to distinguish from the original. The algorithm in this case is pretty straightforward: Minimize $L_2(x - x')$ such that $C(x') = t$. This minimization is computational expensive to solve, thus, the authors solved the following approximation, which is called *L-BFGS*: Minimize $c \cdot L_2(x - x') + loss_{F,l}(x')$, where $loss_{F,l}$ simply mapped an image to positive real number, for example, the cross-entropy function. Note that in *L-BFGS* the constraint $C(x') = t$ was dropped. Instead, we need to find the constant $c > 0$ that yields an adversarial example of minimum distance. Considering that, this optimization problem was repeatedly solved for different values of c. At a high level, this method used output variations to find corresponding input perturbations. A different approach was introduced by Papernot et al., who computed a direct mapping from input perturbations to output variations in order to find adversarial samples [55]. This attack is also known as the Jacobian-based Saliency Map Attack (JSMA). The authors used L_0 distance measures, which simply measure the number of coordinates i such that $x_i \neq x'_i$. In an image classification task, the L_0 distance corresponds to the number of pixels altered in this image. Furthermore, JSMA is a greedy algorithm that picks pixels one at a time, increasing the target classification on each iteration. Saliency maps are computed via the DNN's gradient function to model the impact of each pixel on the resulting classification. Large values in the saliency map show that altering this pixel will significantly enhance the likelihood of the model classify the image as the target class t. Thus, with the saliency map at hand, an adversary can modify the most important pixels of the image to force the model's misclassification. The procedure was repeated until either the misclassification succeeds, or more than a predefined threshold of pixels were altered which would make an attack perceptible. Papernot et al. also applied their approach to the MNIST dataset. The first strategy to alter pixels was based on increasing the intensity of the pixels

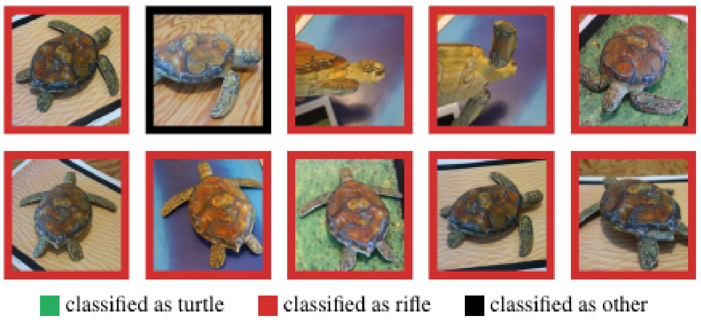

Figure 9: Randomly sampled poses of a single 3D-printed turtle adversarially perturbed to classify as a rifle at every viewpoint by an ImageNet classifier. Unpertubed 3D-printed turtles were correctly classified 100% of the time. A video of the attack can be found under `https://youtu.be/YXy6oX1iNoA` [2].

highlighted by the saliency maps. The authors proved that instances from all classes (0-9) could be reshaped to be classified as every other class. They also verified the validity of their algorithm by running the crafting algorithm on empty image (see Fig. 8) in which all pixels were initially set to an intensity of 0. Once again, each of the classes could be created as required. In another experiment, the intensities of the pixels in the maps were reduced instead of increased. Also, in this case, all source instances could be changed into each of the other classes. The authors also observed that decreasing pixel intensities seems harder to detect by the human eye.

At this point, other methods of attack should also be mentioned. First, the Fast Gradient Sign proposed by Goodfellow et al. [23], which resembles the *L-BFGS* algorithms. However, it has two fundamental differences: it is optimized for L_∞ distance metric and it is designed to be fast instead of producing optimal adversarial samples. Second, Deepfool [49] was constructed by imagining that neural networks are totally linear. With this simplification in mind, an adversarial example optimized for the L_2 distance metric within the simplified problem is created. Since neural networks are not actually linear, the process is repeated with different hyperparameters until a true adversarial example is found. Third, Carlini and Wagner [9] showed a new set of attacks that outperform all previously mentioned attack algorithms. The newly introduced attacks are tailored to the L_0, L_2 and L_∞ distance metrics. While previous attacks could be mitigated or even avoided with a technique called *Defensive Distillation* [52], the attacks by Carlini and Wagner can evade such measures. Finally, in early 2018, Tramer et al. introduce Ensemble Adversarial Training, in which training data is extended with perturbations from other models [67]. A library called `cleverhans` for crafting adversarial samples, building defenses, and benchmarking both can be found on GitHub [12]. The associated technical report currently includes eleven different attack methods [54].

In 2017, methods for creating robust physical-world adversarial examples were presented for the first time by Evtimov et al. [17]. These should be seen primarily in the context of assisted and autonomous driving, in which human lives are directly involved. The two proposed classes of physical attacks were, firstly, a poster printing attack, where an adversary prints actual-sized road signs on paper and overlays it on existing signs, and, secondly, a sticker attack, where attackers print only the perturbations and sticks them to an existing road sign. Experiments have proven, for example, the successful misclassification of stop signs as a speed limit sign. However, this attack required the preprocessing of large amounts of photos and road signs to produce a single adversarial examples, which may be unattainable for attackers. Moreover, the approach was strictly limited to 2D objects, such as road signs, and there was no clear transformation to the 3D case. In contrast, Athalye et al. provided a method for creating real-world 3D objects that deceive DNNs from multiple angles [2]. The authors introduced a novel algorithm called *Expectation over Transformation (EOT)*, which models multiple perturbations during the optimization procedure. To be precise, instead of optimizing a single instance, EOT uses a chosen distribution of transformation functions taking a input generated by the adversary to the "true" input perceived by the targeted classifier. This approach could be successfully applied to 2D and 3D rendered as well as robust 3D printed instances. For the robust real-world examples, a commercially-available full-color 3D color was used. As described in the paper, they had to consider light effects, camera disturbances and possible printing inaccuracies. The study included a 3D printed turtle whose EOT target class was "rifle" (see Fig. 9), and a 3D printed baseball whose target class was "espresso". For comparison, objects that were not modified with EOT were also printed. Such unmodified objects were correctly classified with 100% accuracy over a large number of samples. The modified 3D printed turtle was classified as rifle in over 82% in a quantitative analysis with over 100 photos from a wide range of viewpoints. Only in 2% the turtle was classified correctly as turtle. 16% were misclassified as neither of the two. However, the authors found an interesting semantic relationship in the misclassification: the turtle's second most popular class after rifle was "revolver", followed by "holster" and "assault rifle". The baseball, which was designed to be classified as espresso, was also often classified as "coffee" or "bakery". It should be emphasized once again that even at low cost - a commercially available 3D printer costs just a few hundred euros - there is an enormous potential for damage here, especially if one considers the increasing applications of image recognition in everyday life.

Recent research has also shown that neural networks are not only susceptible to adversarial samples, but that adversarial attacks can also be used for reprogramming a target model to perform tasks chosen by the attacker. [15]. According to Elsayed et al., such an attack could possibly be used to mine crypto currencies, or, by reprogramming digital assistants, also to obtain private data.

3.3 Data Extraction: Acquire Information from the Learning System

In recent years, machine learning based services - so-called ML-as-a-service cloud systems - have been made available to the public via APIs. These include functions for speech recognition, face recognition, medical purposes and more. Frederikson et al. retrieved genomic information about patients given some of their demographic information and access to a black box ML model [20]. However, this data extraction attack was limited to a setting in which the model was trained with only a small training set. This is due to the fact that their algorithm was a maximum a posteriori (MAP) estimator, which requested the black box oracle with different values for an unknown feature and picked the value that maximized the probability of having observed other known features. This requires computing the oracle's output of every possible value of each unknown feature. In another study, Frederikson et al. developed a further model inversion attacks which enabled potential adversaries to recover recognizable images of people's faces given only their name and access to a facial recognition system [19]. In order to do so, attackers needed to solve an optimization problem, where they had to find an input that maximizes the returned confidence of the oracle, provided that the classification also corresponds to the target. Although the reconstructed images were similar to the person being searched for, it was not possible to obtain a particular instance from the model. In the scenario described in the paper, an attacker was only able to reproduce an average of all images of a person in the dataset. Thus, a human postprocessing step was required to compare such an average image with a real image of the person. Nonetheless, the recovered images were good enough that test subjects could find the right target with an accuracy of approximately 80-95% out of a line-up.

Shokri et al. showed that the existence of an instance in a dataset can be proven [62]. This is particularly critical when it comes to very private data, such as medical, financial or political information. In *membership inference attacks*, as they are called by Shokri et al., an attacking ML model was trained, which should have been able to distinguish whether or not an instance was in the training set of a targeted model. Furthermore, the algorithm included the creation of *shadow models* which imitate the targeted model. The attack models were then trained on the labeled inputs and outputs of the shadow models. The authors showed three different approaches for getting training datasets for shadow models, e.g., by synthesizing the data from the targeted model. The idea behind shadow training was that similar models trained on similar data behave similarly. Therefore, the attack model was trained to classify if a data point was used in the training set of a shadow model and to be able to do the same task with the target model. Shokri et al. explained the success of membership inference attacks with a poor generalizability of the target model and a small diversity of trainings data. This type of attack, in which machine learning is not only attacked itself but is actively used as a tool, is examined in detail in the next chapter.

4 Using Machine Learning as a Tool in Offensive Security

In a recent report, the malicious use of artificial intelligence was investigated [8]. In addition to recommended measures for regulators, developers and researchers of AI, dangers in the domains of digital, physical and political security were also pointed out. The presented scenarios included automatic social engineering attacks, automation of cybercrime or vulnerability discovery, as well as extended monitoring options and much more. In the following sections, some concrete attacks are shown, and, furthermore, the question of how to penetrate or damage computer systems and networks with the help of machine learning is explored.

4.1 Automation of Cybercrime Tasks

Several methods have already been invented to identify oneself as a human to web services. Such verification methods often serve to make the automatic creation of user accounts more difficult, as they can be used in large-scale social engineering attacks, for example. A project developed by Microsoft called ASIRRA CAPTCHA[3] tried to decide whether it was a human or a bot using a challenge consisting of 12 pictures of dogs or cats. The user had to select all cat images and none of the dog images. Less than a year after the introduction of the system in 2007, Golle was able to demonstrate a machine learning based attack on the system [22]. The classifier that was used consisted of a combination of two support vector machines, one of which was responsible for color properties and the other for texture features. Even before the era of neural networks began, an accuracy of over 80 percent was achieved in the classification of a single image. However, for the solution of an entire ASIRRA challenge, which consisted of 12 images, the success rate was just over 10 percent. A possibly significantly better performance rate could be achieved with today's techniques of convolutional neural networks, which are superior to other machine learning methods in image classification tasks.

ASIRRA was discontinued in 2014 because other verification methods such as Google's reCAPTCHA were able to establish themselves. These consist of a distorted word and a word scanned from a book. Stark et al. described an approach to automatically recognize these words with CNNs without having to have an extensive training dataset as an attacker [64]. Previous work has already shown that CNNs are an excellent solution for recognizing text in images [24][34]. However, a large training dataset was necessary in order to attain excellent performing convolutional neural networks. Additionally, collecting millions of hand-labeled CAPTCHAs is unattainable. The major contribution by Stark et al. was the introduction of a new learning approach which was able to train a CNN with only a small initial training dataset. In this so-called *Active Learning*, a classifier was trained on this data and was applied to new instances, resulting in an uncertainty measure and a label prediction. The uncertainty estimation decided whether an instance is informative enough to be included in the training set. Afterward, the model was retrained with the extended training set. The idea behind Active Learning is that

[3]ASIRRA is an acronym for Animal Species Image Recognition for Restricting Access.

the most informative samples influence the classifier most. The effectiveness of the approach could be demonstrated in experiments, where accuracies of almost 90% could be achieved.

Most web services use passwords as an authentication method, but numerous password database leaks have shown that criminals can rely on lousy user passwords. Common password guessing tools can expand existing password dictionaries by transforming (e.g., mix letter case or leet speak) or concatenating words or numbers. Hitaj et al. introduced a new approach called PassGAN, a deep learning approach for password guessing based on Generative Adversarial Networks [31]. Such a GAN consisted of a generative deep neural network G, which aimed to imitate the underlying distribution of the samples, and a discriminative deep neural network D, which tried to distinguish whether a sample was from the original data distribution or was crafted from G. In other words, G wanted to mock real instances, while D wanted to identify the fake samples. With this procedure, G could learn from D what is necessary be effective at mimicking the original data distribution. For PassGAN, this technique was used to generate new password guesses. The discriminator D was trained with a list of public passwords, e.g., from a password database leak. Password instances produced by G were then examined by D. With D's feedback, G could then adjust its network so that its fake samples were becoming more and more similar to the original dataset. It should be noted that G did not have access to the original password list. Experiments have shown that PassGAN effectively complements traditional rule-based password guessing tools such as HashCat. The combination of PassGAN and HashCat was able to guess between 51% and 73% additional passwords compared to HashCat alone.

4.2 Attacking Networks

In section 3.1, attacks on networks were already examined. However, in these attacks, it was assumed that machine learning is part of the defending side. In the following case studies, machine learning serves as a tool for the attackers.

Feng et al. demonstrated stealthy attacks on Industrial Control Systems [18], which were designed to react to changes in sensor values in a network and make decisions accordingly. In the presented attack procedure, a GAN was used to generate realistic malicious sensor data, which would successfully evade a black box anomaly detector on the defending side. In a gas pipeline case study, the authors were able to inject malicious pressure measurements which were not noticed by the defenders. This particular example illustrates the danger of successful attacks, as people might also be in danger here.

Networks can also be attacked by infiltrating a specific node of the system with malware. Although these often use machine learning based anti-malware software, which can also detect new types of malware due to its ability to generalize, they can be tricked with adversarial samples, as shown in section 3.2. To generate such instances in a black box attack, it was also possible to use reinforcement learning [1]. The proposed RL framework consisted of an agent and the environment, which was, in this case, an initial malware sample and the targeted anti-malware program. In each step, a sample was sent to the anti-malware program, and the agent received feedback: a reward that indicated whether the instance has been classified as malware, and a feature vector that represented the current status of the environment. With this information, the agent could modify the malware instance and repeat the procedure. The authors were able to show that the newly created malware samples can also be used to retrain the anti-malware models so that subsequent attacks were 33% less successful.

4.3 Side-Channel Attacks

In side-channel attacks, adversaries use information from, for example, power consumption, electromagnetic emissions or sounds to retrieve underlying information. An example from the mid-1980s is the reconstruction of screen contents with a monitor's electromagnetic emissions. In 2010, Backes et al. demonstrated an SVM based side-channel attack on dot matrix printers, which are still used today in the majority of all medical offices or in many bank branches in Germany [3]. The underlying idea was that such a printer becomes louder as more needles strike the paper. Therefore, the printer's sound was recorded and analyzed by several machine learning models. Experiments have shown that over 72% of the printed text could be recovered, and if an attacker used contextual knowledge about the text, results increased up to 95% accuracy. This could undoubtedly be surpassed with today's techniques of deep learning and the achievements in language processing.

Side-channel attacks are also often used to exploit weaknesses in the physical implementation of cryptographic devices, for example, to read cryptographic keys or other secrets. The leakage of such devices often depends on internally used secret keys. Thus, an adversary may perform key-recovery attacks. In order to protect cryptographic implementations, several side-channel countermeasures have been investigated. A distinction can be made between masking and hiding countermeasures: Masking involves randomly dividing a sensitive value into several parts so that an attacker has to find out all the so-called shares correctly. Hiding countermeasures try to make the physical leakage of the device as constant as possible, which is supposed to complicate the attack. First machine learning based side-channel attacks on cryptographic implementations mostly used Support Vector Machines and examined the power consumption of cryptographic devices, e.g. [33][30][41]. Maghrebi et al. extended these attacks by applying several deep learning techniques [44]. As in the previous papers, the authors tried to create a profiling model for each possible value of the target variable during the training phase. In the attack phase, the model should assign the most likely key to the current power consumption traces. In the experiments, it was assumed that an attacker would have access to a training device and could measure the power consumption during the execution of a cryptographic algorithm. Both unprotected and masked AES implementations were examined. In the unprotected setting, convolutional neural networks and autoencoders[4] performed best. CNNs needed around 200 power consumption traces in order to recover keys with a success rate of 100%. The authors explained this with the feature extraction techniques based on a CNN's filters, which can also extract hidden information from the traces. It has also been discovered that an LSTM performs significantly worse in this task. The reason could be that the leakage does not depend on time and LSTMs are specialized for such time-dependent tasks. The deep learning based attacks also achieved good results against masked implementations. While autoencoders needed around 500 power consumption traces to fully reconstruct keys, the CNN and MLP needed around 1000. In conclusion, the application of deep learning techniques surpasses both other machine learning approaches and traditional side-channel attacks.

4.4 Vulnerability Discovery

One of the primary tasks in the field of offensive security is finding security vulnerabilities in the program code. Such flaws are often caused by a lack of security awareness during the software development process and are the result of cost pressure in the business environment. Existing tools for finding vulnerabilities can be divided into static, dynamic and hybrid[5] analysis. A static analysis examines a program without its execution, on the one hand using graphs (e.g., control-flow graphs, data-flow graphs or program-dependence graphs), on the other hand with data modeling (e.g., to make assumptions about targets of an indirect jump statement). Dynamic analysis deals with a program at runtime in the context of an environment. Typical dynamic analysis methods are dynamic taint analysis or fuzzing, which will be discussed later. The hybrid approach tries to combine static and dynamic analysis. In the following, these methods are complemented and improved with the help of machine learning. At first, approaches are pursued which extract the features from program code by using manual preprocessing. Because

[4]Autoencoders are based on neural networks and are designed for the purpose of dimensionality reduction (unsupervised learning)

[5]hybrid analysis is sometimes also called *mixed* analysis, such as in [36]

it has been shown that it is difficult to assign labels to the features, a semi-supervised approach is then also considered. Afterward, more advanced methods are examined that automatically extract features from the source code using deep learning. Finally, machine learning based fuzzing methods will be discussed in more detail.

In 2011, Yamaguchi et al. presented a method based on unsupervised learning to help narrow down the range of possible security vulnerabilities when manually auditing code [71]. As the pattern of an already known vulnerability is the starting point of the process to find other potentially vulnerable code with similar properties, the process is called vulnerability extrapolation. Vulnerability extrapolation enabled the authors to reduce the search range for vulnerable code in the library *FFmpeg* from several thousand functions to 20 functions. Among others, a zero-day vulnerability could be found. The idea behind the approach was that vulnerable code locations often share specific characteristics, for example, there are often errors related to string operations and memory functions. The presented vulnerability extrapolation consisted of 4 steps: First, the source code was tokenized and divided into the individual functions. The authors called these extracted symbols *API symbols*. As a second step, each function was inserted into a vector space. Each of the dimensions represented the frequency of a particular API symbol, for example, one dimension displayed the frequency of the expression `char`, another `mutex_lock`, and so on. Third, the unsupervised learning method Principal Component Analysis was used to find dominant API usage patterns. At this point, it should be remembered that unsupervised learning does not require a labeled dataset. Instead, it simply groups similar instances of a dataset. In the fourth step, code fragments similar to an already known vulnerability were identified, which should then be examined in more detail.

A similar approach to extract features from source code was developed by Harer et al. [27]. However, the code elements were additionally sorted into different categories after tokenizing. Categories included string literals, keywords (e.g., for and if), system function calls, types or operators. As in the work of Yamaguchi et al., the tokens were then transferred into a vectorial representation. In addition to source-based feature extraction, Harer et al. have also shown a build-based approach. During the build process, a control flow graph (CFG) and an opcode-vector were extracted. A CFG describes different paths a program could take during execution. An opcode-vector is a representation of different categories of instructions, for example, conditional, bit binary or conversion instructions. Since Harer et al. used convolutional neural networks and random forests[6] in its machine learning models, the dataset had to be labeled. Labeling the functions as "benign" and "malicious" was a very challenging task for the authors. With some workarounds, the labels were finally assigned by the Clang static analyzer. Yamaguchi et al. used unsupervised learning, so this step was not necessary there.

Between supervised learning, in which each individual instance of a dataset must be labeled with a class, and unsupervised learning, where no labels are required, there is also semi-supervised learning. In semi-supervised methods, only a small part of the data has to be labeled for training. The first study of semi-supervised machine learning in the context of vulnerability prediction was published in 2015 by Shar et al. [61]. The focus was on security flaws in web applications, which often have to struggle with SQL injection, cross-site scripting, remote code execution or file inclusion. In contrast to the methods above, not only static characteristics of programs were investigated, but also hybrid code attributes. Such hybrid features were obtained, for example, by first statically analyzing a code fragment and then testing it under certain conditions by execution with different test input. In this framework, a total of 33 different attributes were derived, with which the machine learning models were then trained. The authors have compared supervised and semi-supervised methods and have concluded that if enough training instances were labeled, supervised machine learning was superior. However, if, as in reality, only a few samples were labeled, semi-supervised approaches performed best.

In the ML-based vulnerability prediction methods previously presented, features were more

[6]Random forests are combinations of multiple decision trees [38]. Their output is usually the majority vote.

or less manually selected by the researchers (e.g., Shar et al. had 33 fixed features, in the case of Yamaguchi et al. they depended on the different words used in the specific source code). Dam et al. presented a way to use deep learning to extract the features automatically [14] from the source code. Long Short-Term Memory (LSTM), a special kind of recurrent neural networks, outperform other machine learning models in learning long-term dependencies in sequential data such as text and speech. This ability was used to capture long context relationships in source code, such as `try` and `catch` or `lock` and `unlock`. As shown, the extracted features included both the semantics and syntax of the code. To be more specific, each function was first tokenized and inserted into the LSTM model, which then output a vector representation of the method. These vectors could then be used to train a classifier similar to those above, which could then identify potentially vulnerable code fragments. Similar recently published deep learning based vulnerability detectors are, for example, VulDeePecker [42] or DeepBugs [57].

After considering applications with static and hybrid analysis, fuzzing with machine learning support as part of a dynamic analysis is now being investigated. Fuzzing is a process in which intentionally invalid data is sent to a program at runtime in order to bring it into an error state, which ultimately discloses a potential security flaw. Grieco et al. have already shown that such input data can be generated effectively with machine learning techniques [26]. Furthermore, they published a tool called *VDiscover* to predict vulnerabilities in such test cases. An advanced approach was presented by Godefroid et al., who used recurrent neural networks to generate new inputs that underlie the probability distribution of the learned model [21]. With the introduced algorithm SampleFuzz, first, a particular format such as PDF or HTML was learned by the RNN, but to generate good fuzzing samples, anomalies had to be inserted. In a case study, the PDF parser of Microsoft Edge was examined. Here, SampleFuzz proved to be superior to traditional approaches of fuzzing.

An even more detailed study of automated software vulnerability detection was recently published by Ji et al. in May 2018 [36].

5 Conclusion

In the first part of this paper, attacks on machine learning systems were discussed and the framework of Barreno et al. was presented which can be used to check the security of such systems. The focus here was on the dimension of influence, i.e., whether the system was attacked during the training phase or during runtime. It was shown, among other things, that Support Vector Machines, whose classification depends only on a few so-called support vectors, are vulnerable even to attacks where only one instance in the training dataset has been modified. Since new deep learning approaches often require enormous amounts of data to train the weights of their neurons, causative attacks are not yet widespread for these algorithms. However, the vulnerability of neural networks to adversarial examples is remarkable, since they generally have good generalization capabilities. As already described in section 3.2, there are several attempts to explain this phenomenon.

In the second part of the thesis, the possibility of using machine learning as a tool in offensive security was investigated. It has been shown that whenever large amounts of data are available, be it large password databases, physical measurement results in the context of side-channel attacks, or available open source code, machine learning can very successfully support the work of information security researchers. Through semi-supervised learning or domain-specific developments such as the active learning demonstrated earlier, it is possible to achieve good results even with smaller datasets.

References

[1] Hyrum S Anderson et al. “Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning”. In: *arXiv preprint arXiv:1801.08917* (2018).

[2] Anish Athalye and Ilya Sutskever. “Synthesizing robust adversarial examples”. In: *arXiv preprint arXiv:1707.07397* (2017).

[3] Michael Backes et al. “Acoustic Side-Channel Attacks on Printers.” In: *USENIX Security symposium.* 2010, pp. 307–322.

[4] Marco Barreno et al. “The security of machine learning”. In: *Machine Learning* 81.2 (2010), pp. 121–148.

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. “Poisoning attacks against support vector machines”. In: *arXiv preprint arXiv:1206.6389* (2012).

[6] Chris Bishop, Christopher M Bishop, et al. *Neural networks for pattern recognition.* Oxford university press, 1995.

[7] Matt Bishop. *Computer security: art and science.* Addison-Wesley Professional, 2003.

[8] Miles Brundage et al. “The malicious use of artificial intelligence: Forecasting, prevention, and mitigation”. In: *arXiv preprint arXiv:1802.07228* (2018).

[9] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *Security and Privacy (SP), 2017 IEEE Symposium on.* IEEE. 2017, pp. 39–57.

[10] Vicky Charisi et al. “Towards moral autonomous systems”. In: *arXiv preprint arXiv:1703.04741* (2017).

[11] Patricia S Churchland and Terrence J Sejnowski. *The computational brain.* MIT press, 2016.

[12] *cleverhans.* 2016. URL: `https://github.com/tensorflow/cleverhans`.

[13] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[14] Hoa Khanh Dam et al. “Automatic feature learning for vulnerability prediction”. In: *arXiv preprint arXiv:1708.02368* (2017).

[15] Gamaleldin F Elsayed, Ian Goodfellow, and Jascha Sohl-Dickstein. “Adversarial Reprogramming of Neural Networks”. In: *arXiv preprint arXiv:1806.11146* (2018).

[16] Gamaleldin F Elsayed et al. “Adversarial Examples that Fool both Human and Computer Vision”. In: *arXiv preprint arXiv:1802.08195* (2018).

[17] Ivan Evtimov et al. “Robust Physical-World Attacks on Deep Learning Models”. In: *arXiv preprint arXiv:1707.08945* 1 (2017).

[18] Cheng Feng et al. “A Deep Learning-based Framework for Conducting Stealthy Attacks in Industrial Control Systems”. In: *arXiv preprint arXiv:1709.06397* (2017).

[19] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. “Model inversion attacks that exploit confidence information and basic countermeasures”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.* ACM. 2015, pp. 1322–1333.

[20] Matthew Fredrikson et al. “Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing.” In: *USENIX Security Symposium.* 2014, pp. 17–32.

[21] Patrice Godefroid, Hila Peleg, and Rishabh Singh. “Learn&fuzz: Machine learning for input fuzzing”. In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering.* IEEE Press. 2017, pp. 50–59.

[22] Philippe Golle. “Machine learning attacks against the Asirra CAPTCHA”. In: *Proceedings of the 15th ACM conference on Computer and communications security*. ACM. 2008, pp. 535–542.

[23] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[24] Ian J Goodfellow et al. “Multi-digit number recognition from street view imagery using deep convolutional neural networks”. In: *arXiv preprint arXiv:1312.6082* (2013).

[25] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[26] Gustavo Grieco et al. “Toward large-scale vulnerability discovery using Machine Learning”. In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM. 2016, pp. 85–96.

[27] Jacob A Harer et al. “Automated software vulnerability detection with machine learning”. In: *arXiv preprint arXiv:1803.04497* (2018).

[28] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. “Unsupervised learning”. In: *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[29] Michiel Hermans and Benjamin Schrauwen. “Training and analysing deep recurrent neural networks”. In: *Advances in neural information processing systems*. 2013, pp. 190–198.

[30] Annelie Heuser and Michael Zohner. “Intelligent machine homicide”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2012, pp. 249–264.

[31] Briland Hitaj et al. “Passgan: A deep learning approach for password guessing”. In: *arXiv preprint arXiv:1709.00440* (2017).

[32] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[33] Gabriel Hospodar et al. “Machine learning in side-channel analysis: a first study”. In: *Journal of Cryptographic Engineering* 1.4 (2011), p. 293.

[34] Max Jaderberg et al. “Reading text in the wild with convolutional neural networks”. In: *International Journal of Computer Vision* 116.1 (2016), pp. 1–20.

[35] Ahmad Javaid et al. “A deep learning approach for network intrusion detection system”. In: *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2016, pp. 21–26.

[36] Tiantian Ji et al. “The Coming Era of AlphaHacking?” In: *arXiv preprint arXiv:1805.11001* (2018).

[37] Bojan Kolosnjaji et al. “Deep learning for classification of malware system call sequences”. In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2016, pp. 137–149.

[38] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. “Supervised machine learning: A review of classification techniques”. In: *Emerging artificial intelligence applications in computer engineering* 160 (2007), pp. 3–24.

[39] Terran D Lane. *Machine learning techniques for the computer security domain of anomaly detection*. 2000.

[40] Peter Lee. *Learning from Tay’s introduction*. `https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction`. accessed 2018-04-27.

[41] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. “Power analysis attack: an approach based on machine learning”. In: *International Journal of Applied Cryptography* 3.2 (2014), pp. 97–115.

[42] Zhen Li et al. "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection". In: *arXiv preprint arXiv:1801.01681* (2018).

[43] Daniel Lowd and Christopher Meek. "Good Word Attacks on Statistical Spam Filters." In: *CEAS*. Vol. 2005. 2005.

[44] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. "Breaking cryptographic implementations using deep learning techniques". In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2016, pp. 3–26.

[45] Margaret Rouse. *Offensive security*. [Online; accessed June 11, 2018]. 2012. URL: `https://whatis.techtarget.com/definition/offensive-security`.

[46] Maximilian Schmitt. *Evaluation*. [Lecture slides - Praktikum Computational Intelligence (University of Augburg, WS2017/2018)]. 2017.

[47] Maximilian Schmitt. *Long Short Term Memory (LSTM)*. [Lecture slides - Praktikum Computational Intelligence (University of Augburg, WS2017/2018)]. 2018.

[48] Shike Mei and Xiaojin Zhu. "Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners." In: *AAAI*. 2015, pp. 2871–2877.

[49] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2574–2582.

[50] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[51] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples". In: *arXiv preprint arXiv:1605.07277* (2016).

[52] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 582–597.

[53] Nicolas Papernot et al. "Practical black-box attacks against machine learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM. 2017, pp. 506–519.

[54] Nicolas Papernot et al. "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library". In: *arXiv preprint arXiv:1610.00768* (2018).

[55] Nicolas Papernot et al. "The limitations of deep learning in adversarial settings". In: *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE. 2016, pp. 372–387.

[56] Petar Veličković. *Multilayer perceptron (MLP)*. [Online; accessed May 30, 2018]. 2016. URL: `https://github.com/PetarV-/TikZ/tree/master/Multilayer%20perceptron`.

[57] Michael Pradel and Koushik Sen. "DeepBugs: A Learning Approach to Name-based Bug Detection". In: *arXiv preprint arXiv:1805.11683* (2018).

[58] Benjamin IP Rubinstein et al. "Antidote: understanding and defending against poisoning of anomaly detectors". In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. ACM. 2009, pp. 1–14.

[59] Joshua Saxe and Konstantin Berlin. "Deep neural network based malware detection using two dimensional binary program features". In: *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE. 2015, pp. 11–20.

[60] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[61] Lwin Khin Shar, Lionel C Briand, and Hee Beng Kuan Tan. “Web application vulnerability prediction using hybrid program analysis and machine learning”. In: *IEEE Transactions on Dependable and Secure Computing* 12.6 (2015), pp. 688–707.

[62] Reza Shokri et al. “Membership inference attacks against machine learning models”. In: *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE. 2017, pp. 3–18.

[63] Robin Sommer and Vern Paxson. “Outside the closed world: On using machine learning for network intrusion detection”. In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, pp. 305–316.

[64] Fabian Stark et al. “Captcha recognition with active deep learning”. In: *Workshop New Challenges in Neural Computation 2015*. Citeseer. 2015, p. 94.

[65] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. “One pixel attack for fooling deep neural networks”. In: *arXiv preprint arXiv:1710.08864* (2017).

[66] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[67] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses”. In: *arXiv preprint arXiv:1705.07204* (2017).

[68] David Warde-Farley and Ian Goodfellow. “11 Adversarial Perturbations of Deep Neural Networks”. In: *Perturbations, Optimization, and Statistics* (2016), p. 311.

[69] Jason Weston and Chris Watkins. *Multi-class support vector machines*. Tech. rep. Citeseer, 1998.

[70] Gregory L Wittel and Shyhtsun Felix Wu. “On Attacking Statistical Spam Filters.” In: *CEAS*. 2004.

[71] Fabian Yamaguchi, Felix Lindner, and Konrad Rieck. “Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning”. In: *Proceedings of the 5th USENIX conference on Offensive technologies*. USENIX Association. 2011, pp. 13–13.

[72] Weizhong Yan and Lijie Yu. “On accurate and reliable anomaly detection for gas turbine combustors: A deep learning approach”. In: *Proceedings of the annual conference of the prognostics and health management society*. 2015.

[73] Zhenlong Yuan et al. “Droid-sec: deep learning in android malware detection”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 44. 4. ACM. 2014, pp. 371–372.