Adversarial machine learning in Network Intrusion Detection Systems

Article *in* Expert Systems with Applications · August 2021

DOI: 10.1016/j.eswa.2021.115782

CITATIONS
163

READS
377

3 authors:

Elie Alhajjar
RAND Corporation
33 PUBLICATIONS **259** CITATIONS
SEE PROFILE

Paul Maxwell
United States Military Academy
29 PUBLICATIONS **321** CITATIONS
SEE PROFILE

Nathaniel Bastian
United States Military Academy
163 PUBLICATIONS **1,477** CITATIONS
SEE PROFILE

Expert Systems With Applications 186 (2021) 115782

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Adversarial machine learning in Network Intrusion Detection Systems

Elie Alhajjar *, Paul Maxwell, Nathaniel Bastian

Army Cyber Institute, United States Military Academy, West Point, NY 10996, United States of America

ARTICLE INFO

Keywords:
Network Intrusion Detection Systems
Adversarial machine learning
Evolutionary computation
Deep learning
Monte Carlo simulation

ABSTRACT

Adversarial examples are inputs to a machine learning system intentionally crafted by an attacker to fool the model into producing an incorrect output. These examples have achieved a great deal of success in several domains such as image recognition, speech recognition and spam detection. In this paper, we study the nature of the adversarial problem in Network Intrusion Detection Systems (NIDS). We focus on the attack perspective, which includes techniques to generate adversarial examples capable of evading a variety of machine learning models. More specifically, we explore the use of evolutionary computation (particle swarm optimization and genetic algorithm) and deep learning (generative adversarial networks) as tools for adversarial example generation. To assess the performance of these algorithms in evading a NIDS, we apply them to two publicly available data sets, namely the NSL-KDD and UNSW-NB15, and we contrast them to a baseline perturbation method: Monte Carlo simulation. The results show that our adversarial example generation techniques cause high misclassification rates in eleven different machine learning models, along with a voting classifier. Our work highlights the vulnerability of machine learning based NIDS in the face of adversarial perturbation.

1. Introduction

It is becoming evident each and every day that machine learning algorithms are achieving impressive results in domains in which it is hard to specify a set of rules for their procedures. Examples of this phenomenon include industries like finance (Bose & Mahapatra, 2001; Wong, Bodnovich, & Selvi, 1997), transportation (Pomerleau, 1991), education (Juola, 2006; Stańczyk & Cyran, 2007), health care (Kourou, Exarchos, Exarchos, Karamouzis, & Fotiadis, 2015) and tasks like image recognition (He, Zhang, Ren, & Sun, 2015, 2016; Simonyan & Zisserman, 2014), machine translation (Chen et al., 2018; Sutskever, Vinyals, & Le, 2014), and speech recognition (Lee, Park, Kim, & Lee, 2018; Vaswani et al., 2017; Xiong et al., 2016; Zhang, Geiger, Pohjalainen, Mousa, Jin, & Schuller, 2018).

Motivated by the ease of adoption and the increased availability of affordable computational power (especially cloud computing services), machine learning algorithms are being explored in almost every commercial application and are offering great promise for the future of automation. Facing such a vast adoption across multiple disciplines, some of their weaknesses are exposed and sometimes exploited by malicious actors. For example, a common challenge to these algorithms is "generalization" or "robustness", which is the ability of the algorithm to maintain performance whenever dealing with data coming from a different distribution with which it was trained.

For a long period of time, the sole focus of machine learning researchers was improving the performance of machine learning systems (true positive rate, accuracy, etc.). Nowadays, the robustness of these systems can no longer be ignored; many of them have been shown to be highly vulnerable to intentional adversarial attacks (Lin, Shi, & Xue, 2018; Papernot, McDaniel, Jha et al., 2016). This fact renders them inadequate for real-world applications, especially mission-critical ones. Adversarial machine learning is classified by that National Institute of Standards and Technology (NIST) into four categories of attacks: evasion, extraction, poisoning, and inference. In this work, we examine adversarial examples, which is a type of evasion attacks.

An adversarial example is an input to a machine learning model that an attacker has intentionally designed to cause the model to make a mistake. In general, the attacker may have no access to the architecture of the machine learning system being attacked, which is known as a black-box attack. Attackers can approximate a white-box attack by using the notion of "transferability", which means that an input designed to confuse a certain machine learning model is able to trigger a similar behavior within a different model. In this work, we model a white-box attack by evaluating our examples against a variety of machine learning models, thus showing performance over a wide range of possible systems.

Network Intrusion Detection Systems (NIDS) monitor network traffic to detect abnormal activities, such as attacks against hosts or servers. Historically, these systems operated using rule sets and code signatures created by expert human analysts. These are time consuming to create

* Corresponding author.
E-mail addresses: elie.alhajjar@westpoint.edu (E. Alhajjar), paul.maxwell@westpoint.edu (P. Maxwell), nathaniel.bastian@westpoint.edu (N. Bastian).

https://doi.org/10.1016/j.eswa.2021.115782
Received 14 May 2021; Received in revised form 11 August 2021; Accepted 15 August 2021
Available online 28 August 2021
0957-4174/Published by Elsevier Ltd.

and can only be created after the chosen attack method has been used at least once. Once these traditional systems identify a possible intrusion, an expert must adjudicate the instance to determine its final classification. Increasingly machine learning algorithms are being incorporated into NIDS to eliminate these issues. ML algorithms offer the benefit of detecting novel differences in network traffic through training on normal and attack traffic thereby eliminating the need for previous exposure to an attack. Machine learning systems also propose to lower the analyst's workload by reducing false positives that consume hours of time to process.

On the other hand, the adoption of machine learning algorithms in the network intrusion detection setting has raised a major security issue, namely the introduction of adversarial machine learning. Adversarial machine learning has the potential to deceive NIDS thereby making networks more vulnerable to hacking. In this work, we study adversarial machine learning through the lens of NIDS to show the sensitivity of machine learning models when faced with adversarial attacks generated by perturbation techniques based upon evolutionary computation, deep learning, and Monte Carlo methods. The results of this work can assist organizations with vulnerability analysis of their networks or enable cyber operations to achieve their objectives more effectively. Given our results, a cyber adversary with some insight on a target network NIDS could effectively attack that network with the most effective perturbation algorithm.

The contributions of this work are the creation and evaluation of three novel perturbation methods for the generation of adversarial examples: particle swarm optimization (PSO), genetic algorithm (GA), and generative adversarial network (GAN). Additionally, our methods contribute to the field through their unique treatment of the input data. These algorithms perturb the inputs without altering the malicious functionality of the network attack vectors, and they allow the perturbation of more than just binary fields as was previously studied. Thus, this work is among the first to consider the constrained nature of the feature space in the NIDS setting while ensuring functionality retention. We apply these supervisory-trained perturbation algorithms to two well-known data sets, NSL-KDD and UNSW-NB15, and show that the perturbations created by these techniques can fool eleven different machine learning models, plus an ensemble model. For our baseline adversarial example generation method, we use a Monte Carlo (MC) simulation for random generation of perturbations.

The paper is organized as follows. In Section 2, we give a brief overview of the field of adversarial machine learning. We set the notation and establish the state of the art taxonomy therein. In Section 3, we review the most relevant and up to date literature that deals with adversarial machine learning. Section 4 describes the methodology used to generate adversarial examples. We explain the intuition as well as the technical details of our chosen perturbation methods. In Section 5, we apply our evolutionary computation and deep learning methods to the two data sets mentioned above, and we record the results while contrasting them with the baseline perturbation method. We also discuss our findings and distinguish between the machine learning models that are highly sensitive and the ones that show some type of robustness. We conclude our work in Section 6 and pose some open questions and future research directions.

2. Prerequisites

Machine learning encompasses a vast field of techniques that extract patterns from data, as well as the theory and analysis relating to these algorithms. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E (Mitchell, 1997).

For the interested reader, we review the basic concepts of machine learning, broadly divided into three major areas: supervised learning, unsupervised learning, and reinforcement learning in Appendix A. In this section, we then introduce the notion of adversarial machine learning, common attack types, and categorize the types of attacks along three dimensions: timing, information, and goals. We adopt a simplistic approach for the sake of exposition, the reader interested in the technical definitions and details is referred to the books (Joseph, Nelson, Rubinstein, & Tygar, 2019; Mitchell, 1997; Vorobeychik, Kantarcioglu, Brachman, Stone, & Rossi, 2018).

Adversarial Machine Learning (AML) is a research field that lies at the intersection of machine learning and computer security. AML can take many forms. Evasion attacks attempt to deceive a ML system into misclassifying input data. For example, in an intrusion detection system, an evasion attack attempts to have malicious traffic classified as benign. A poisoning attack attempts to corrupt an ML model during its training phase through insertion of data points that will cause the resulting ML model to behave in a manner that is beneficial to an attacker. As an example of this type of attack is when an attacker inserts data into a system (e.g., anomaly detector) thereby training the model to accept the data as normal. Another AML attack is the model extraction attack. This type attempts to gather information about an ML system that allows the attack to recreate the model. The attacker can then use this model to learn about the target system and how to use it to their benefit. In this work, we focus on evasion attacks. Below we classify possible attacks into three main pillars.

First, the timing of the attack plays a crucial role in the process. The main distinction occurs between evasion attacks and poisoning attacks. The former ones are executed at decision time: they assume that the model has already been trained, so the attacker aims at changing its behavior to cause the model to make incorrect predictions. The latter ones, in contrast, take place before the model is trained. They aim at modifying a part of the data used for training purposes to corrupt the final model.

Second, we highlight the nature of information the attacker has about the learning algorithm. This allows us to compare white-box attacks and black-box attacks. Namely, white-box attacks assume that the model is known to the adversary, whereas in black-box attacks the adversary has limited or no information about the model, although may obtain some of the information indirectly, for example, through queries and captured responses.

Third, attackers may have different reasons for attacking, such as evading detection or reducing confidence in the algorithm. We differentiate between two broad classes of attack goals: targeted attacks and reliability attacks. In a targeted attack, the attacker's goal is to cause a misclassification into a specific label or target. In contrast, a reliability attack aims to degrade the perceived reliability of the machine learning system by maximizing prediction error without any particular target label.

3. Related work

3.1. Historical works

The first publication that hints at adversarial machine learning dates back to the early 2000's. In 2004, Dalvi, Domingos, Mausam, Sanghai, and Verma (2004) managed to fool classifiers for spam detection by adding intentional changes to the body of an email. In 2005, Lowd and Meek (2005) introduced the adversarial classifier reverse engineering (ACRE) learning problem, the task of learning sufficient information about a classifier to construct adversarial attacks. They presented efficient algorithms for reverse engineering linear classifiers with either continuous or Boolean features and demonstrated their effectiveness using real data from the domain of spam filtering.

A taxonomy for attacks and defenses in adversarial setting was first established by Barreno et al. in Barreno, Nelson, Sears, Joseph, and Tygar (2006) and later refined in Barreno, Nelson, Joseph, and Tygar (2010). This taxonomy clearly defines threat models and includes the concept of adversarial examples without any mentioning of the term.

The first time this term was coined was in 2013 when Szegedy et al. (2013) investigated properties of deep neural networks and found that by applying a certain imperceptible perturbation to an image, these networks misclassified it with high confidence. Moreover, they showed that the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the data set, to misclassify the same input.

The last decade has witnessed a large body of publications in the realm of adversarial machine learning in several domains and it is impossible to summarize it all in a paragraph or two. We point the curious reader to a couple of excellent recent articles that attempt to survey the field of AML. Liu et al. (2018) investigate security threats and give a systematic survey on them from two aspects, the training phase and the testing/inferring phase. They also categorize existing defensive techniques of machine learning into four groups: security assessment mechanisms, countermeasures in the training phase, those in the testing or inferring phase, data security, and privacy. Further, Akhtar and Mian (2018) focus on adversarial attacks on deep learning in computer vision. They emphasize that adversarial attacks are possible in practical conditions and review the contributions that evaluate such attacks in real-world scenarios.

A more detailed survey is provided by Serban et al. in Serban, Poll, and Visser (2018). The authors introduce an exhaustive list of attacks/defenses and thoroughly discuss the property of transferability. They provide a taxonomy of attacks and defenses that is specific to the adversarial examples field. This taxonomy is meant to properly structure the methods in AML and help future researchers to position their work in comparison to other publications.

3.2. Unconstrained domain works

Most of the adversarial machine learning research so far has focused on unconstrained domains (e.g., image and object recognition). This is mainly due to the following assumption: the adversary is always able to fully exploit each feature or pixel of a given image. Several attack algorithms currently exist in the literature and many variations to their models are widely implemented. These models include the Fast Gradient Sign Method (FGSM) (Goodfellow, Shlens et al., 2014), the Jacobian-based saliency map attack (JSMA) (Papernot, McDaniel, Jha et al., 2016), Deepfool (Moosavi-Dezfooli, Fawzi, & Frossard, 2016), and the Carlini Wagner attack (CW) (Carlini & Wagner, 2017).

Goodfellow et al. (Goodfellow, Shlens et al., 2014) proposed the FGSM to generate adversarial perturbations based on the gradient of the loss function relative to the input image and, thus, enable computational efficiency through backpropagation. Papernot, McDaniel, Jha et al. (2016) created adversarial saliency maps by computing forward derivatives, which are used to identify the input feature to be perturbed towards the target class. Moosavi-Dezfooli et al. (2016) proposed an approach to find the closest distance from original input to the decision boundary of adversarial examples. Carlini and Wagner (Carlini & Wagner, 2017) introduced three new gradient-based attack algorithms (l_2, l_∞, and l_0) that are more effective than all previously known methods in terms of the adversarial success rates achieved with minimal perturbation amounts.

Recent works on GANs have been published that use these tools in anomaly detection and data generation. The work of Zenati, Romain, Foo, Lecouat, and Chandrasekhar (2018) uses GANs along with an encoding network to develop a unique anomaly score. This work assists with creating real-time detectors but does not contribute to the field of adversarial machine learning. Andresini, Appice, De Rose, and Malerba (2021) created the MAGNETO system which uses GANs to create training data for intrusion detection systems. They treat netflow data as 2D images and then use GANs to create training data to resolve the issue of unbalanced training sets. Though their work uses GANs and is in a constrained domain, it does not treat data as being constrained and it is not used for generating adversarial examples.

3.3. Constrained domain works

Unlike unconstrained domains, the situation is quite different in constrained ones due to the following three characteristics:

1. The values within a single feature can be binary, continuous or categorical.
2. The values of different features in a data set can be correlated with one another.
3. Some features are fixed and cannot be controlled by a potential adversary.

For these reasons, it was not clear whether constrained domains are less vulnerable to adversarial example generation than unconstrained domains. In Sheatsley (2018), Sheatsley tested the above hypothesis by creating targeted universal perturbation vectors that encode feature saliency within the envelope of domain constraints. The experiment was able to generate misclassification rates greater than 95% by introducing two algorithms: the adaptive JSMA, which crafts adversarial examples that obey domain constraints, and the histogram sketch generation, which produces adversarial sketches.

In this work, we aim to provide more evidence to the same hypothesis. We adapt the algorithms used in GA, PSO, and GAN to generate adversarial examples in unconstrained domains. By doing so, we achieve high misclassification rates in the majority of commonly used machine learning models applied to the data sets NSL-KDD and UNSW-NB15. Our baseline MC simulation perturbation method generates more or less random perturbations in the data.

We end this section by mentioning that similar techniques appeared previously in the adversarial machine learning literature (Shah et al., 2019). Hu and Tan (2017) propose a GAN-based algorithm named MalGAN to generate adversarial malware examples which are able to bypass black box machine learning detection models. Nguyen, Yosinski, and Clune (2015) use evolutionary algorithms to generate images that are given high prediction scores by convolutional neural networks. Drawing ideas from genetic programming, Xu, Qi, and Evans (2016) propose a generic method to evaluate the robustness of classifiers under attack. Their key idea is to stochastically manipulate a malicious sample to find a variant that preserves the malicious behavior but is classified as benign by the classifier. More recently, Alzantot et al. (2019) introduced GenAttack, a gradient-free optimization technique that uses genetic algorithms for synthesizing adversarial examples in the black box setting. Mosli, Wright, Yuan, and Pan (2019) created AdversarialPSO, a black-box attack that uses fewer queries to create adversarial examples with high success rates. AdversarialPSO is based on particle swarm optimization, and is flexible in balancing the number of queries submitted to the target compared to the quality of imperceptible adversarial examples. Piplai et al. 9123023 use the Fast Sign Gradient Method to generate adversarial examples that defeat IDS systems. However, they do not consider the constraints on their data fields when crafting their examples and thus risk causing the attacks to fail. The authors in Han et al. 9448103 use a GAN to generate adversarial values for use in features input into their Particle Swarm Optimizer that generates adversarial examples in IDS environments. While they acknowledge that features have constraints in this domain, they only modify a small number of features and modify features such as the Transport level protocol field which will cause the traffic to fail. Additionally, their PSO heuristic attempts to minimize the distance between the original feature and the adversarial feature while our work does not attempt to do this. Finally, Devine and Bastian (Devine & Bastian, 2021) used a machine learning approach for robust malware classification that integrates a MC simulation for adversarial perturbation with meta-learning using a stacked ensemble-based methodology.

4. Methodology

In this section, we give a technical description of the features in the data sets used in our experiments. We then explain the details of the techniques employed for adversarial example generation. This leads to the layout of our computational setting.

4.1. Data sets

There are a small number of publicly available, well-known labeled data sets of network traffic for cyber security research. Two of these data sets are: NSL-KDD (Tavallaee, Bagheri, Lu, & Ghorbani, 2009) and UNSW-NB15 (Moustafa & Slay, 2015). Both data sets have a mix of benign and malicious traffic with a variety of network traffic types and attack types. Both data sets have quality limitations due to factors like generation methods, prevalence of attack traffic, and size. Nevertheless, these data sets are commonly used to evaluate machine learning based NIDS.

In general, the data sets contain fields with raw and processed data derived from the underlying network traffic. Normally, this data is engineered in research work to detect intrusions in a network. In many works (Elsayed,, 2018; Grosse, Papernot, Manoharan, Backes, & McDaniel, 2016, 2017), effort is made to alter the data to create adversarial examples that can fool detection systems and classifiers. In domains such as image and speech recognition, perturbing the data to fool the systems can be done in a manner that does not affect the functionality or appearance of the original item. For example, changing pixel values in an image by a few bits can be undetectable to the human eye yet fool a trained machine learning classifier. In network security applications care must be taken when generating adversarial examples that alter the source data. There are data, such as the duration of a network packet flow, that if altered will not impact the functionality of the protocol. However, there are fields that if changed can cause the functionality of the transmission to fail, such as changing the protocol type from TCP to UDP.

The goal of an attacker (who may use adversarial examples) is to alter packets to achieve a desired effect while still maintaining the functionality of the traffic. Previous research (Grosse et al., 2016, 2017) acknowledge that certain fields in the data sets are immutable while others are open to perturbations. In this work, we subscribe to this position and place technical constraints on data fields that are mutable. For the data sets used, the mutable fields are described below.

4.1.1. NSL-KDD data set

This data set is an altered version of the KDD'99 data set with over 125,000 training samples and more than 22,000 test samples. As described in (Iglesias & Zseby, 2015), it contains 41 features based upon raw network traffic elements, flow traffic elements, and content. The data contains attacks from four attack types that were more prevalent at the time of the set's creation (Denial of Service, Probes, Remote-to-local, and User-to-Root). Attack traffic comprises a little over 50 percent of the data in the set and the normal data in the set is not representative of modern traffic. Engineering the features using common techniques such as one-hot encoding and Min-Max Scaling results in a final data set with 121 features. Of these features, some are immutable such as *protocol-type*, *service*, and *flag*. A change to these values would cause the underlying traffic to become non-functional. The remaining features are mutable, though we limit the perturbations to changes that increase the initial values. The rationale behind such decision is that these fields, such as src_bytes, can increase without altering the functionality of the traffic (e.g. null packets can be inserted into a particular communication flow, or delays can be added to the time between packets). Decreasing these fields without expert knowledge of the traffic content would not be certifiably feasible. Other constraints on perturbations to the data are fields that are binary (the value can only be flipped from 0 to 1 and vice versa) and fields that are linearly related such as $same\_srv\_rate$ and $diff\_srv\_rate$ must sum to 1. A change to one of these fields alters the content of the other. Overall, we consider the techniques to alter the mutable fields as feasible though we do not explore the actual implementation of these techniques. Our analysis of the engineered features results in 93 immutable features and 29 mutable ones.

4.1.2. UNSW-NB15 data set

The UNSW-NB15 data set was generated at the Cyber Range Lab of the Australian Centre for Cyber Security. It contains over 175,000 training samples and 82,000+ test samples that are labeled as benign or malicious (including attack type). The data contains nine types of more modern, low footprint attacks as described in (Moustafa & Slay, 2015). The benign traffic is more reflective of current network traffic as well. In this set, attacks compromise a much smaller percentage of the data at under 25 percent of the total. This data set was also purposely constructed to more closely match the content of the training set and test set as highlighted in (Moustafa & Slay, 2016). The data has entries with 49 fields of which some are generated by network flow analyzers and intrusion detection tools. Engineering these fields results in 196 features of which we identified 23 as mutable. Similar to the NSL-KDD data set, some fields are mutable but their values are constrained because they are: binary, increasing in value only, or have a linear relationship with another field. However, some fields in this set can have decreasing values such as $sttl$ (e.g., the time to live can arbitrarily be set by the sender), and $sload$ (e.g., the source node can increase/decrease packet sizes and transmission rates).

4.2. Adversarial example generation

Adversarial examples have been extensively used to evade machine learning systems. The methods of generation for these adversarial examples include mostly heuristic techniques (Grosse et al., 2017; Sheatsley, 2018; Yang, Wu, Li, & Chen, 2017) and GANs (Goodfellow, Pouget-Abadie et al., 2014; Hu & Tan, 2017; Lin et al., 2018). In this work, we introduce two novel techniques for adversarial example generation that use evolutionary computation, PSO and GA, along with a GAN that uniquely considers constrained feature sets to avoid the failure of the resulting traffic. Finally, we implement a MC simulation generator for baseline comparison.

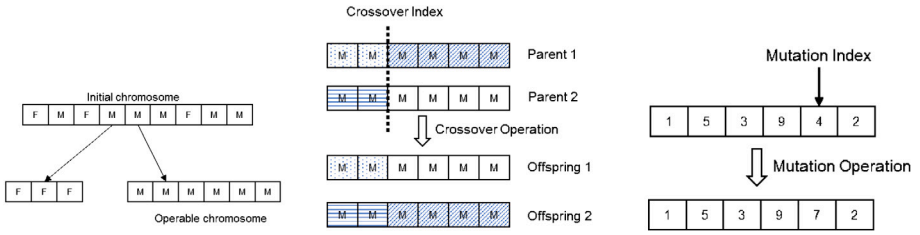
4.2.1. Genetic algorithm

Genetic algorithms are commonly used to find near-optimal solutions to NP-hard problems (Whitley, 1994). In the NIDS setting, searching the space with over 90 dimensions for a solution that best fools a machine learning classifier can benefit from this type of search heuristic. Fig. 1 illustrates a simplistic view of how a genetic algorithm works in general.

Genetic algorithms typically operate on data structures known as chromosomes; a chromosome is a representation of the problem's data. For this work, a chromosome is comprised of a data feature from the source data sets. The GA operates on each row entry in the data set creating a population of chromosomes. This initial population contains a seed chromosome based on the original row's values and randomly generated chromosomes based upon the seed. Each chromosome contains both mutable and immutable elements. As a result, prior to performing the algorithm's standard operations (cross-over and mutation), the chromosome is separated into two parts, the fixed and operable elements as shown in Fig. 1(a). Of note, each chromosome in the population has a common, fixed element. Once the chromosomes are sub-divided, the heuristic operations can be performed on the operable sub-part of the chromosome.

Based upon the chosen cross-over rate, the cross-over operation is performed. Two parent chromosomes are randomly selected from the population and a cross-over index is selected randomly. This index separates the operable chromosome into left and right portions in each parent. The right portions are then swapped between parents to form two new operable chromosomes. These child chromosomes are then inserted into the population. The cross-over operation is illustrated in Fig. 1(b).

The mutation operation perturbs values of randomly selected cells based on a chosen mutation rate. The operation first randomly selects chromosomes to operate upon based on the mutation rate. Then, a



(a) Initial Chromosome transformation into operable (mutable) and fixed (immutable) parts. Cells marked with an 'F' are immutable and those with an 'M' are mutable.

(b) Example cross-over operation on two parent operable chromosomes and the resulting child operable chromosomes.

(c) Example mutation operation performed on an operable chromosome.

Fig. 1. Genetic Algorithm Operators.

mutation index is randomly selected. Using this index, the targeted chromosome is modified as shown in the example of Fig. 1(c). The resulting chromosome is left in the population in its modified form.

Upon completion of the cross-over and mutation operators, the chromosomes' fitness functions are evaluated. In this domain, the fitness metric is the output probability that the chromosome is classified as benign by a machine learning classifier. For the remainder of the paper, we pick the fitness function to be an ensemble of classifiers as described in the next section. To evaluate the chromosome, the operable chromosome must be recombined with the immutable part to form a valid chromosome. The best chromosome for each generation is maintained in subsequent generations. The chromosomes in the next generation, limited by the chosen population size variable, are selected using a roulette wheel technique. The process of selection, cross-over, and mutation is repeated until a chosen number of iterations is reached or the improvement in the best fitness is below a chosen threshold. A pseudo-code for the GA is shown in Algorithm 1

Algorithm 1 Genetic Algorithm

```
Choose cross-over rate, mutation rate, number of chromosomes, maximum iterations
Separate seed vector into immutable and mutable sub-vectors
Generate initial population of mutable vectors
    Mutable seed vector created as a chromosome
    Randomly generate remaining chromosomes
    Determine population best fitness using fixed portion of seed vector
while (iterations < maximumiterations and improvement > improvementminimum) do
    for number of cross-over operations do
        Randomly select two parent chromosomes
        Randomly select cross-over index
        Perform cross-over at the index
        Add two offspring chromosomes to the population
    for each chromosome in the population do
        if randomnumber ≤ mutationrate then
            Randomly select mutation index
            Update chromosome at mutation index with new value
    Calculate fitness of each chromosome
    Update population best fitness
    Select next generation using roulette wheel method
```

4.2.2. Particle swarm optimization

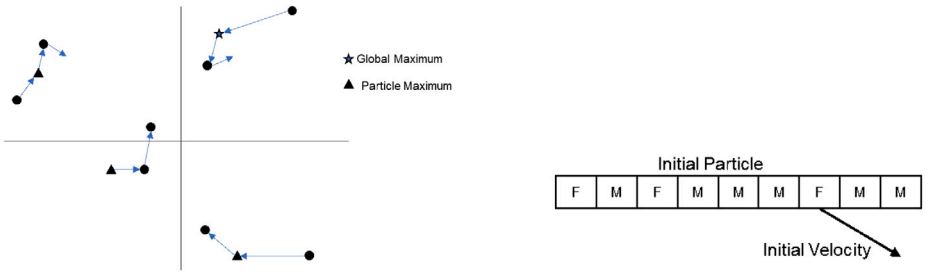
Particle swarm optimization is a bio-inspired heuristic based on behaviors of animals that flock or swarm (Clerc, 2010). The idea is that each member or particle of the swarm explores the search space with a calculated velocity. The velocity is updated based upon the particle's best solution and the swarm's optimal solution. The probability that the velocity is influenced by the best solutions is determined by selected weighting coefficients. We used the PySwarms code base (James V. Miranda, 2018) to develop the heuristic used in our work. Fig. 2(a) illustrates how the PSO algorithm works.

The heuristic begins with the creation of the swarm. The swarm is seeded with a row from the evaluated data set. The remaining particles are then randomly created using the seed particle as a baseline. Each particle is then randomly assigned an initial velocity. Similar to the GA, the particle is composed of mutable and immutable cells. An example particle is shown in Fig. 2(b).

In each iteration of the PSO algorithm, the particles' fitness function is evaluated using the same metric as the GA. The values of the particle's best location and the global best location are updated as required. Next, the distances between each particle's location and the particle's best location and the global best location is determined. These values are then used to update the particle's velocity and location. The heuristic is repeated until either a selected number of iterations is reached or the global best fitness improvement is below a chosen threshold. Upon completion, the global best fitness location is the desired output. The PSO pseudo-code is shown in Algorithm 2

Algorithm 2 Particle Swarm Optimization

```
Choose initial weights, number of particles, maximum iterations
Initialize swarm
    Seed vector created as particle
    Randomly generate remaining particles
    Randomly generate initial velocities
while (iterations < maximumiterations) and (improvement > improvementminimum) do
    Calculate each particle's fitness
    Update global best fitness and each particle's best fitness
    for each particle do
        Calculate distance between location and particle best fitness
        Calculate distance between location and global best fitness
        Update particle velocity
        Update particle location
Output global best fitness location
```



(a) Example Particle Swarm with particles moving in the search space.

(b) Example Particle after creation of the particle swarm.

Fig. 2. Particle Swarm Optimization Scheme.

4.2.3. Generative adversarial network

Generative adversarial networks are a deep learning technique that pits two neural networks against each other in a game-like setting. The GAN is generally composed of two deep neural network models: a generator and a discriminator. The generator's task is to learn from the discriminator's output and thus train so that its output may deceive the discriminator. The discriminator attempts to discern if its inputs are from the genuine data set or from the adversarial data set. As described in the original work (Goodfellow, Pouget-Abadie et al., 2014), the competition between the two components continues until neither can improve their abilities. In our scenario, it is desired that the generator, once trained, can modify malicious input vectors such that the output is classified as benign by the target classifier (i.e., the discriminator). The pseudo-code of our GAN is depicted in Algorithm 3

Algorithm 3 Generative Adversarial Network

```
for number of iterations do
    for number of generatortrainingsteps do
        Randomly select rows from benign and malicious traffic vectors
        Break malicious vector into mutable and immutable sub-vector
        Add noise randomly to malicious mutable sub-vector
        Combine malicious noisy mutable and original immutable sub-vectors
        Input to generator neural network
        Combine mutable output of generator with original immutable sub-vector
        Input to discriminator to for classification
            If classification = 'benign', feedback label = mutable output vector of generator
            If classification = 'malicious', feedback label = original benign vector
        Train generator using resulting feedback labels
    for number of discriminator training steps do
        Randomly select rows from benign traffic vectors
        Input benign traffic and generator created noisy traffic to discriminator and voting classifier to predict labels
            If discriminator label = voting classifier label, feedback = 'malicious'
            Else, feedback = 'benign'
        Train discriminator using resulting feedback labels
Input test vectors to generator to create adversarial vectors
Test adversarial vectors using voting classifier
```

The data input to the generator is divided into mutable and immutable parts like the GA and PSO. Noise is then added to the mutable portion of the data by randomly selecting features and perturbing their values. One contribution in the field of GANs that we make in this heuristic is the addition of non-binary features in the GAN's operations and the consideration of data sets that have immutable and constrained mutable elements. Other work in this area either does not consider data sets of this type or only use binary features in their operation thus limiting their application. Additionally, due to the nature of this domain, there is no need to search for a minimum perturbation in the data like in the image domain. Large changes in the data are not subject to easy recognition by human observers.

The noisy vector, containing the noisy mutable and original immutable portions, is then fed to the generator neural network whose output is a vector of the same size as the mutable portion of the original input. This adversarial vector output is then combined with the initial immutable portion of the input vector to form a new adversarial input. This adversarial vector is then fed to the discriminator to generate classification labels (benign or malicious). These labels are used to improve the neural networks' outputs. To accomplish our goal, the generator and discriminator are alternately trained using batches of training data. A depiction of the generator's training is shown in Fig. 3.

The discriminator is trained using the complete adversarial vectors from the generator along with benign vectors from the data set. These vectors are then classified by the discriminator and a separate machine learning classifier (in this case a voting classifier) as benign or malicious. The outputs of the two classifiers are compared in the following way. If both classify an input as malicious, then the feedback to the discriminator is labeled malicious. Otherwise, the classification is benign. A depiction of the discriminator training method is shown in Fig. 4.

4.2.4. Monte Carlo Simulation

To create a baseline comparison for the evolutionary computation (GA and PSO) and deep learning (GAN) adversarial example generation techniques, we implemented a MC simulation that randomly perturbs features. The heuristic operates by first randomly selecting the features to perturb from the list of mutable ones. Then, these features are increased in the range $[x_i, 1]$. Note that we abide by the same limitations on how the values are modified as in the other three methods. This process is repeated for a fixed number of iterations and the best fitness score is recorded.

4.3. Computational experiment

The perturbation methods based on evolutionary computation, deep learning, and MC simulation were run on both data sets: NSL-KDD (Tavallaee et al., 2009) and UNSW-NB15 (Moustafa & Slay, 2015). The evolutionary algorithms were run on standard laptops with 64 GB of RAM using Ubuntu linux virtual machines while the GAN was run on a Jupyter notebook hosted by servers located at the U.S. Army Engineer Research and Development Center. All algorithms used Python 3.6 along with key packages such as keras, numpy, scikit-learn (Pedregosa et al., 2021), and pandas. Only the vectors labeled as malicious were used in the evaluation of the methods because the

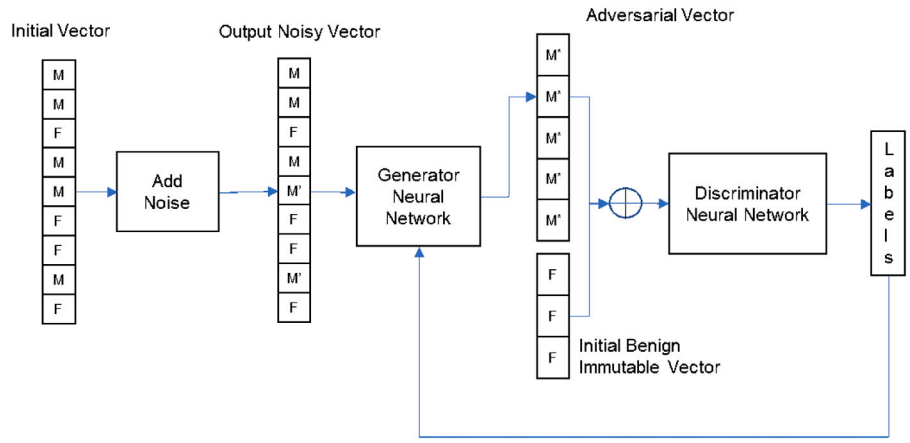


Fig. 3. GAN generator training operations. M' represents perturbed inputs and M* represents generator modified values. Output from the Discriminator is used to update the Generator model.

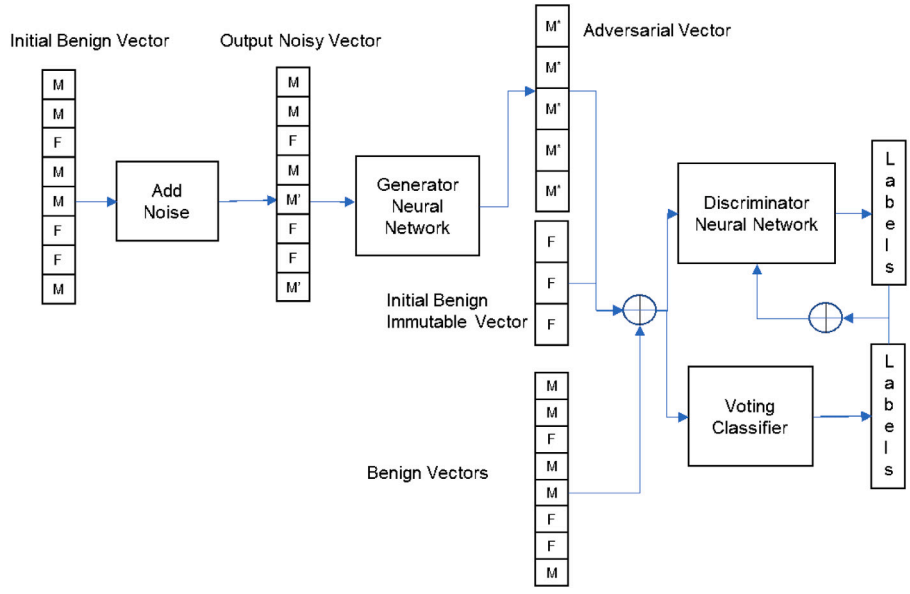


Fig. 4. GAN discriminator training. M' represents perturbed inputs and M* represents generator modified values. Output from the Discriminator and an independent classifier are compared to update the Discriminator model.

ultimate goal is deceiving the classifiers with modified malicious inputs. The NSL-KDD testing set contains 12,828 malicious vectors while the UNSW-NB15 test set contains 45,328 malicious vectors. A baseline classification is developed on the unmodified test set using classifiers similar to those described in the work of (Iglesias & Zseby, 2015). The fitness function used for GA and PSO is a Voting Ensemble Classifier formed by soft voting and the following sub-models: Support Vector Machine (SVM), Decision Tree (DT), Naive Bayes (NB), and K-Nearest Neighbors (KNN). The SVM settings are: $gamma = 0.01$, $C = 220.0$, $tol = 0.01$, $probability = True$. The Naive Bayes model has the default settings, while the Decision Tree algorithm uses the settings: $criterion = \}\}entropy\varepsilon$, $min\_samples\_split = 4$, $min\_samples\_leaf = 2$, $max\_depth = 20$, $min\_impurity\_decrease = 0.1$. Finally, the KNN algorithm settings are: $n\_neighbors = 3$, $algorithm = \}\}auto\varepsilon$.

The parameters of the GA are a population size of 100 chromosomes (the number of simultaneous solutions in the working set), 1000 generations maximum (limits the search length), a cross-over rate of 0.25 (percentage of chromosomes selected for the cross-over operation per generation), a mutation rate of 0.2 (percentage of chromosomes selected for the mutation operation per generation), and an early termination criterion of less than 0.01% improvement in 5 generations. The parameters used in the PSO heuristic are 200 particles, $c_1 = 0.5$ (coefficient for the influence of the particle local best solution on the output solution), $c_2 = 0.4$ (coefficient for the influence of the swarm global best solution on the output solution), $w = 0.7$ (weight for the particle velocity when calculating position updates), a maximum of 100 iterations, and an early termination criterion of less than 0.001% improvement. The MC simulation is run with a maximum of 15 perturbed features per vector and a total of 250 iterations per vector. The best settings on all the algorithms are determined experimentally.

The GAN generator is created using a Keras Sequential model with a dense input layer of size 122 for the NSL-KDD data set and 196 for the UNSW-NB15 data set. The generator has five additional dense layers of size 29 for the NSL-KDD data set and 23 for the UNSW-NB15 data set, and uses a *relu* activation function for all layers. The model employs a stochastic gradient descent (SGD) optimizer with an *lr* of 0.01, a *clipvalue* of 0.01 with a loss function of *mean_squared_error*. The GAN discriminator is a three-layer dense model with similar input sizes to the generator but with a single output. The initial two layers use a *relu* activation function while the output layer uses a *sigmoid* activation function. The model uses a RMSprop optimizer with an *lr* of 0.01, a *clipvalue* of 0.01, and a *binary_crossentropy* loss function. The models

Table 1
NSL-KDD Data Set. Evasion Rate of Malicious Vectors (numbers represent the percentages of malicious vectors classified as benign by the corresponding ML classifier).

ML model	Original vectors	PSO	GA	GAN	MC
Support vector machines	35.78	94.37	**99.95**	85.85	46.41
Decision trees	46.89	91.75	92.37	**92.60**	91.13
Naive Bayes	40.87	83.00	**88.27**	48.06	80.39
K-nearest neighbor	34.54	54.59	68.19	48.34	**69.32**
Random forest	37.81	34.01	46.41	**85.90**	45.53
Multi-layer perceptron	26.95	44.43	77.87	**86.18**	38.87
Gradient boosting	32.03	1.02	6.46	**66.01**	33.49
Linear regression	**37.57**	14.19	24.46	33.94	36.63
Linear discriminant analysis	39.66	**99.99**	97.18	55.19	96.19
Quadratic discriminant analysis	44.21	56.26	**67.78**	66.74	61.86
Bagging	28.78	54.81	56.15	**61.61**	41.64
Average	36.83	57.13	65.92	**66.40**	58.31

are trained for 300 iterations of 10 steps each and vector batch size of 50.

Each of the methods produces a set of perturbed malicious vectors. These vectors are then input into 11 separate classification models to evaluate their performance. Four of the evaluation classifiers are the sub-models used in the GA/PSO voting ensemble. The remaining seven are: Random Forest (RF), Multi-layer Perceptron (MLP), Gradient Boosting (GB), Logistic Regression (LR), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), and Bagging (BAG). The models are chosen to cover a broad range of classifier categories.

5. Results and discussion

5.1. General results

The differences in the two data sets used in this work do not allow for comparison between them in terms of commonalities. The attack types used in the sets only overlap in denial-of-service and probing attacks and the techniques in the attacks differ greatly due to the fifteen plus years between their creation. Additionally, the weakness of the NSL-KDD data set with respect to the composition of its training and test sets creates classification differences that are likely less prevalent in the UNSW-NB15 data set. Finally, the features of the two data sets only overlap in five collected fields. Therefore, the features used to identify malicious traffic are mostly incomparable. The discussion of the results by data set below indicates where useful similarities are found between them.

It should also be noted that no attempt is made to tune the classifiers for the sake of optimizing their performance under any metric. The classifiers are created to form a basis of comparison across a wide range of classifier types and the salient feature is the similarity of the models in their construction.

5.2. Results from the NSL-KDD data set

Before analyzing the effectiveness of the perturbation methods for adversarial example generation, the 11 machine learning models are first evaluated for performance (accuracy) using the full NSL-KDD data set. The MLP performed the best with an accuracy of 83.27% with the BAG (80.20%) and LDA (79.68%) models serving second and third best, respectively.

Recall that the PSO, GA, GAN and MC perturbation methods are used to generate adversarial malicious vectors to emulate an evasion attack against the NIDS. To compare the effectiveness of these four adversarial generation techniques, Table 1 shows the evasion rate of malicious vectors in terms of percentage of attack traffic classified as normal using the 11 classification models and compared against the unmodified, original vectors.

Compared to the baseline Monte Carlo perturbation technique, the PSO, GA and GAN performed strictly better for the SVM, DT, MLP, and BAG machine learning models. For the other classification models, there were varying results across the four perturbation methods. When comparing these adversarial generation techniques against the original vectors, the NIDS evasion rate was strictly better for all classifiers except for RF, GB, and LR. This also holds true for the average evasion rate across all classifiers for each adversarial example generation algorithm. The ensemble method classifiers were more resilient on average to adversarial attacks though the GAN had more success against them than the other methods. The evolutionary computation methods (PSO and GA) performed strictly better than the deep learning method (GAN) for the SVM, NB, KNN, and LDA classifiers, whereas the GAN performed better than both PSO and GA for the DT, RF, MLP, GB, LR and BAG classifiers. The former's performance can be attributed partially to the use of most of those models in the evaluation function for the heuristics. The GAN's superior performance against the remaining classifiers is partially attributed to the learning of its discriminator using the voting classifier and the generator. Additionally, this suggests that when the classifier under-the-hood of the NIDS uses a tree-based classification model (DT, RF, GB and BAG), then the GAN-based perturbation method is capable of achieving high misclassification rates.

Table 2
NSL-KDD data set. Best evasion rate (%) by adversarial generation method.

Monte Carlo	Particle Swarm Opt.	Genetic algorithms	GAN
96.19	**99.99**	99.95	92.60

Upon examination of Fig. 5, it is evident that the DT classifier has the least variability across the four perturbation methods, with an evasion rate above 90%, which is more than double the performance of the original vectors. This holds true for the UNSW-NB15 data set as well. In this experiment, the DT classifier was extremely vulnerable to adversarial examples in general. It is also evident that for QDA and BAG classifiers, the four perturbation methods had limited performance in terms of evasion rate when compared to the original vectors. For the LR classifier, all four perturbation methods performed strictly worse than the unmodified vectors; the results were somewhat mixed for the GB classifier. The best evasion rates by adversarial generation method are depicted in Table 2.

5.3. Results from the UNSW-NB15 data set

For the UNSW-NB15 data set, Table 3 highlights the comparative analysis of the four perturbation methods for adversarial generation against the 11 machine learning models representing the NIDS classifier. Again, the performance is represented by the evasion rate of malicious vectors in terms of percentage classified as normal using the classifiers and compared against the original vectors. On average, the GA performed strictly better than the PSO, GAN and MC techniques and greatly fooled the NIDS compared to the original vectors (73.71% compared to 5.00% accuracy).

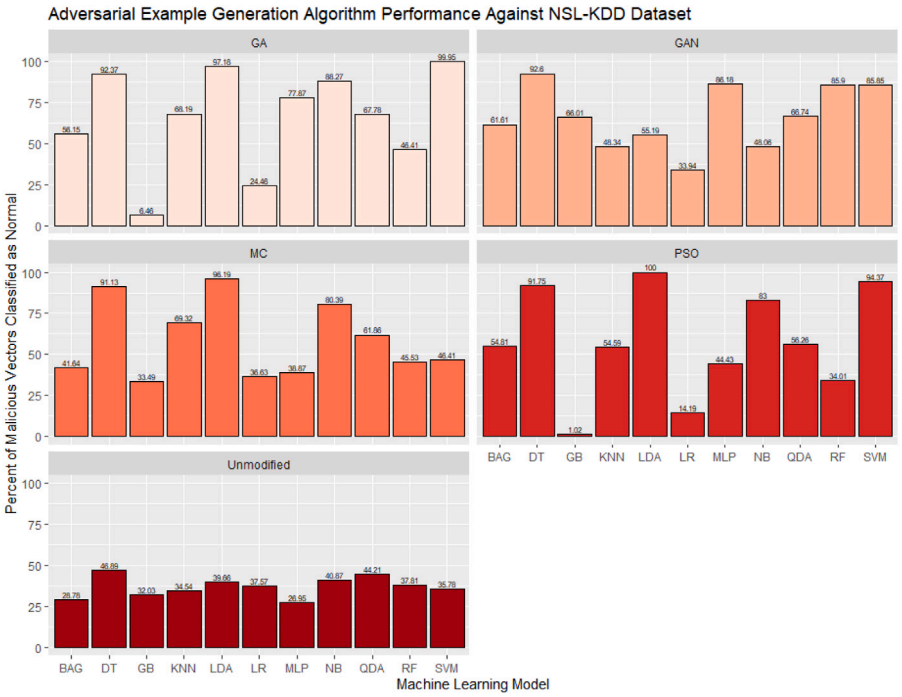


Fig. 5. Adversarial Example Generation Algorithm Evasion Performance Against NSL-KDD Data Set.

Table 3
UNSW-NB15 Data Set. Evasion Rate of Malicious Vectors (numbers represent the percentages of malicious vectors classified as benign by the corresponding ML classifier).

ML model	Original vectors	PSO	GA	GAN	MC
Support vector machines	0.37	96.85	**98.88**	22.04	77.67
Decision trees	0.14	96.85	**100.00**	95.43	99.53
Naive Bayes	**33.06**	**83.66**	75.56	32.98	32.90
K-nearest neighbor	3.82	23.56	**65.34**	4.19	10.64
Random forest	0.64	9.55	31.28	30.61	**46.04**
Multi-layer perceptron	0.63	11.74	39.36	53.90	**55.10**
Gradient boosting	0.55	98.06	**99.74**	99.42	99.39
Linear regression	2.51	86.96	**91.25**	71.54	88.77
Linear discriminant analysis	0.21	**85.68**	84.89	7.21	55.09
Quadratic discriminant analysis	10.18	4.77	24.82	**31.37**	30.42
Bagging	2.91	97.83	**99.65**	99.61	99.30
Average	5.00	63.23	**73.71**	49.85	63.17

Compared to the MC technique as the baseline perturbation method, PSO, GA and GAN performed strictly better for the NB machine learning models, and the results were greatly mixed for the other classifiers representing the NIDS. When comparing the four adversarial generation techniques against the original vectors, the NIDS evasion rate was strictly better for all classifiers except NB and QDA suggesting that modern traffic features are more susceptible to adversarial manipulation than the older traffic. The evolutionary computation methods (PSO and GA) performed strictly better than the deep learning method (GAN) for the SVM, DT, NB, KNN, LR, and LDA classifiers, whereas the GAN performed strictly better than both PSO and GA for the MLP and QDA. This seems to suggest that when the classifier under-the-hood of the NIDS uses a neural network based classification model (MLP), then the GAN-based perturbation method can more easily fool the NIDS; although, the MC technique did an overall better job of evading for this machine learning model.

Upon examination of Fig. 6, it is evident that the BAG, DT and GB classifiers had the least variability across the four perturbation methods, with an evasion rate above 95%. This suggests that each of the adversarial example generation techniques are nearly equally good at fooling NIDS that use tree-based classifiers; this also suggests that these types of machine learning models are easily fooled. This is similar to the RF classifier, which is also tree-based, but the overall performance is much less for the four perturbation methods. For the NB classifier, the GAN and MC performed strictly worse than the unmodified vectors, which was not the case for the evolutionary computation methods (PSO and GA) that performed strictly better (over double). The best evasion rates by adversarial generation method are depicted in Table 4.

Table 4
UNSW-NB15 data set. Best evasion rate (%) by adversarial generation method.

Monte Carlo	Particle Swarm Opt.	Genetic algorithms	GAN
99.53	98.06	**100.00**	99.61

6. Conclusion

It is widely believed nowadays that no domain is immune to adversarial machine learning. With the purpose of validating the hypothesis that constrained domains are at least as vulnerable to adversarial attacks as unconstrained ones, this paper investigated the effects of creating perturbations using techniques from evolutionary computation (PSO and GA) and deep learning (GAN). The main goal of our computational experimentation was to tweak malicious traffic in order to evade

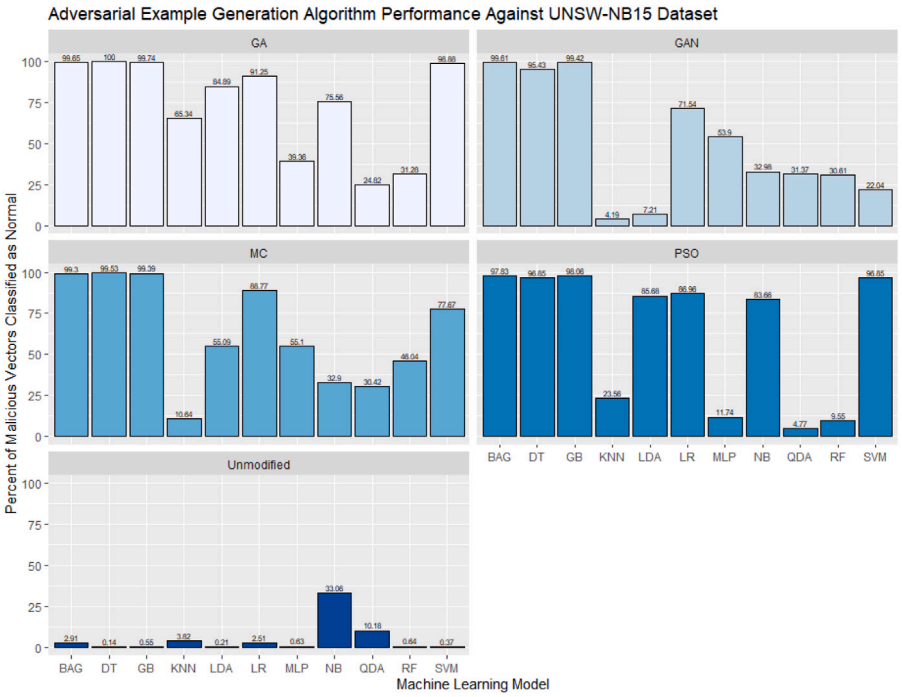


Fig. 6. Adversarial Example Generation Algorithm Evasion Performance Against UNSW-NB15 Data Set.

detection by NIDS, i.e. to "trick" the ML systems to classify such traffic as normal.

To the best of our knowledge, this work is the first to incorporate evolutionary algorithms in the realm of adversarial machine learning in the network intrusion detection setting. Throughout our results, it was evident that the data vectors created via the PSO, GA, and GAN methods achieved high misclassification rates against a handful of machine learning classifiers. We highlight some key takeaways below.

First, the SVM and DT classifiers were the most vulnerable against adversarial examples created by evolutionary algorithms and generative adversarial networks (> 90% evasion rate). This leads us to recommend refraining from using them in NIDS and in mission-critical tasks in general. Second, in the network traffic data, it is not quite realistic to assume that a potential adversary has the capability to alter each and every feature of the traffic. This is a crucial assumption that distinguishes between crafting adversarial examples in unconstrained domains versus ones in constrained domains. In our work, we rely on subject matter expertise to filter the features that can be modified by an attacker without breaking the functionality of the network traffic. It is worth mentioning that our strategy differs from that adopted in (Sheatsley, 2018), where the choice of modifiable features does not seem to preserve the network functionality. Third, our results show that the same set of adversarial examples that managed to fool one machine learning classifier also succeeded at fooling several others. A particular instance of such observation was the performance of the adversarial examples generated by the PSO algorithm in the UNSW-NB15 data set (98.06% against GB, 85.68% against LDA, and 97.83% against BAG). This observation can be considered as additional evidence to the transferability phenomenon first alluded to in (Papernot, McDaniel, & Goodfellow, 2016) within the image recognition setting and in (Sheatsley, 2018) within the network intrusion detection setting.

In the future, we aim to analyze the internal operations of the ML models used in this paper, to include conducting a feature space analysis. It is clear that all of them are vulnerable to adversarial perturbations, but it not clear why some of them are more robust than the others in the NIDS setting. We hope that a systematic study of their mechanism would shed some light on their robustness and explain their sensitivity to "small" input alterations.

Disclaimer

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Military Academy, the United States Army, the Department of Defense, or the United States Government.

CRediT authorship contribution statement

Elie Alhajjar: Conceptualization, Methodology, Writing – original draft. **Paul Maxwell:** Data curation, Software, Coding, Reviewing and editing. **Nathaniel Bastian:** Visualization, Validation, Reviewing and editing.

Declaration of competing interest

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

Acknowledgments

This work was partially funded by the U.S. Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL) Faculty and Cadet Collaborative Research Program through the Mathematical Sciences Center, Department of Mathematical Sciences, U.S. Military Academy, West Point, NY. This work was also supported in part by the National Security Agency Laboratory for Advanced Cybersecurity Research under Interagency Agreement No. USMA21035, the U.S. Army DEVCOM C5ISR Center under Interagency Agreement No. USMA21056, and the U.S. Army DEVCOM ARL under Interagency Agreement No. USMA21050. The authors would like to thank the U.S. Army Engineer Research and Development Center for use of their compute resources in the course of this work.

Table 5
Mutable Input Fields in Network Dataset. Field labels from Netflow data for both datasets that are considered mutable for creation of adversarial examples.

	Field name
1	src bytes
2	dst bytes
3	land
4	wrong fragment
5	urgent
6	hot
7	num failed logins
8	logged in
9	num comprom
10	root shell
11	su attempted
12	num root
13	num file creations
14	num shells
15	num access files
16	is hot login
17	is guest login
18	count
19	srv count
20	same srv rate
21	diff srv rate
22	srv diff host rate
23	dst host count
24	dst host srv count
25	dst host same srv rate
26	dst host diff srv rate
27	dst host same src port rate
28	dst host diff src port rate
29	duration

Appendix A. Overview of machine learning techniques

A.1. Supervised learning

Consider a data set $D = \{(x_i, y_i) : i = 1, \dots, n\}$ which lies in a product space $Z = X \times Y$, where $\times$ denotes the cartesian product. We call X the input space or feature space and Y the label space or the output space. In general, D is assumed to be drawn from an unknown distribution.

For a model class F of functions mapping the input space X to the output space Y, the ultimate goal is to find a model $f \in F$ which is a good approximation of the labels actually observed in the data. Given a labeled training data set D and a cost function l, the process goes as follows: the cost function assigns a numerical value to each combination of data point, true label, and classifier label. The classifier then tries to "learn" the function that minimizes the cost incurred in predicting the label y_i of a new data point (x_i, y_i) when only the predictor variable x_i is revealed to the classifier.

In simple terms, if there is some "true" function h we are trying to learn, the goal is to find a function $f \in F$ which is as "close" to h as possible, given the constraints that the model class imposes on us. When the labels are real numbers, the paradigm is called regression learning, and when Y is a finite set of labels it is called classification learning.

In regression learning, since the label values are real numbers, a loss function is used to penalize predictions that are made far from the observed labels. An appropriate set of functions is the set of l_p-norms defined as:

$$\|f(x) - y\|_p := \left(\sum_{i=1}^{n} |f(x_i) - y_i|^p \right)^{\frac{1}{p}}, \tag{1}$$

where $x \in X$ is the input value, $y \in Y$ is the true label and $f(x)$ is the value predicted by the model f. The most commonly used norms are the l_1-norm and the l_2-norm, also known as the Manhattan norm and the Euclidean norm, respectively. A well-known example here is linear regression, in which the model class F is the set of all linear functions and the standard method is the least squares method that minimizes the l_2-norm.

In classification learning, the labels form a finite set of values. For our purposes, we consider a binary set of labels $Y = \{0, 1\}$ where }}0ε indicates a benign network traffic and }}1ε indicates a malicious network traffic. The most natural loss function in this setting is a function that takes the value }}1ε if the predicted label matches the real label and }}0ε otherwise. Due to its non-convexity, this function is sometimes replaced by the hinge loss function or the logistic loss function.

A.2. Unsupervised learning

In unsupervised learning, a typical data set consists only of features without labels, that is,

$D = \{(x_i) : i = 1, \dots, n\}$. Hence, problems in this paradigm are concerned with identifying aspects of the joint distribution of observed features rather than predicting a target label. Common techniques include clustering, principal component analysis (PCA), and matrix completion.

In clustering, the data set D is partitioned into a collection S of subsets such that each subset of S contains feature vectors that are close to each other based on a chosen metric or distance. Examples of clustering are the k-means clustering and the Gaussian mixture model. Challenges with this method is determining the numbers of clusters and which clusters are indicative of malicious behavior.

In principal component analysis, the feature vectors x_i of the data set D form the rows of a matrix A of dimension say m. The task becomes to find a collection of orthonormal basis vectors of A of size $k < m$. These vectors are the eigenvectors of the matrix A and they correspond to the columns of a basis matrix B. Then any feature vector x_i can be reconstructed as $x_i' = BB^T x_i$, where T denotes the transpose of a matrix. The error can be calculated:

$$Error_i = x_i - x_i' = (I - BB^T)x_i, \tag{2}$$

where I is the $m \times m$ identity matrix, and the goal is eventually to minimize such error in order to get a good approximation of the original data.

In matrix completion, the general goal is to recover a complete matrix from a few given observations. It is usually impossible to complete an arbitrary matrix with only partial observations, so additional assumptions are needed to proceed. A crucial assumption for example is that the matrix has rank much smaller than the number of its rows and columns. In this context, the Frobenius norm is the most commonly used norm to minimize the error incurred by the matrix completion method.

A.3. Reinforcement learning

In reinforcement learning, the goal is to learn via interaction and feedback, or in other words learning to solve a task by trial-and-error. The mathematical foundation for reinforcement learning is based on Markov Decision Processes (MDPs), which we define below.

A discrete-time MDP is a tuple (S, A, T, r, δ) where S is a finite set of states, A is a finite set of actions, T is the set of transition probabilities, r is a reward function, and $\delta \in (0, 1)$ is a discount factor. Loosely speaking, reinforcement learning aims at maximizing the reward function for a state $s \in S$ and an action $a \in A$. Two techniques are widely used in this framework: Q-learning and policy learning.

In Q-learning, a function Q is defined that takes as input one state and one action and returns the expected reward of that action and all subsequent actions at that state. Before learning begins, Q is initialized

to a random fixed value and it gets updated as the learning proceeds until it converges to an optimal value.

In policy learning, a mapping $\pi : S \rightarrow A$ is introduced from the set of states to the set of actions. We refer to this map as policy: $a = \pi(s)$ means that when state s is observed, the best thing to do is to take action a. The task then becomes to find a policy with maximum expected return.

Appendix B. Mutable data features

See Table 5.

References

Akhtar, N., & Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, *6*, 14410–14430. http://dx.doi.org/10.1109/access.2018.2807385.

Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C.-J., & Srivastava, M. B. (2019). Genattack: Practical black-box attacks with gradient-free optimization. In *Proceedings of the genetic and evolutionary computation conference* (pp. 1111–1119). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3321707.3321749.

Andresini, G., Appice, A., De Rose, L., & Malerba, D. (2021). GAN augmentation to deal with imbalance in imaging-based intrusion detection. *Future Generation Computer Systems*, *123*, 108–127. http://dx.doi.org/10.1016/j.future.2021.04.017.

Barreno, M., Nelson, B., Joseph, A. D., & Tygar, J. D. (2010). The security of machine learning. *Machine Learning*, *81*(2), 121–148. http://dx.doi.org/10.1007/s10994-010-5188-5.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. D. (2006). Can machine learning be secure? In *Proceedings of the 2006 ACM symposium on information, computer and communications security*. ACM Press, http://dx.doi.org/10.1145/1128817.1128824.

Bose, I., & Mahapatra, R. K. (2001). Business data mining — a machine learning perspective. *Information & Management*, *39*(3), 211–225. http://dx.doi.org/10.1016/S0378-7206(01)00091-X.

Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 IEEE symposium on security and privacy*. IEEE, http://dx.doi.org/10.1109/sp.2017.49.

Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., et al. (2018). The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th annual meeting of the association for computational linguistics (Volume 1: Long papers)*. Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/p18-1008.

Clerc, M. (2010). *Particle swarm optimization (Vol. 93)*. John Wiley & Sons.

Dalvi, N., Domingos, P., Mausam, Sanghai, S., & Verma, D. (2004). Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 99–108). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/1014052.1014066.

Devine, S. M., & Bastian, N. D. (2021). An adversarial training based machine learning approach to malware classification under adversarial conditions. In *Proceedings of the 54th Hawaii international conference on system sciences* (pp. 827–836). ScholarSpace.

Elsayed, G., Shankar, S., Cheung, B., Papernot, N., Kurakin, A., Goodfellow, I., et al. (2018). Adversarial examples that fool both computer vision and time-limited humans. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), *Advances in neural information processing systems (Vol. 31)* (pp. 3910–3920). Curran Associates, Inc..

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv:1412.6572.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., & McDaniel, P. (2016). Adversarial perturbations against deep neural networks for malware classification. http://arxiv.org/abs/1606.04435 arXiv:1606.04435 [Cs],

Grosse, K., Papernot, N., Manoharan, P., Backes, M., & McDaniel, P. (2017). Adversarial examples for malware detection. In S. N. Foley, D. Gollmann, & E. Snekkenes (Eds.), *Computer security – ESORICS 2017 (Vol. 10493)* (pp. 62–79). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-66399-9_4, http://link.springer.com/10.1007/978-3-319-66399-9_4.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *2015 IEEE international conference on computer vision*, http://dx.doi.org/10.1109/iccv.2015.123.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE conference on computer vision and pattern recognition*, http://dx.doi.org/10.1109/cvpr.2016.90.

Hu, W., & Tan, Y. (2017). Generating adversarial malware examples for black-box attacks based on GAN. arXiv:1702.05983.

Iglesias, F., & Zseby, T. (2015). Analysis of network traffic features for anomaly detection. *Machine Learning*, *101*(1–3), 59–84. http://dx.doi.org/10.1007/s10994-014-5473-9.

James V. Miranda, L. (2018). Pyswarms: a research toolkit for particle swarm optimization in python. *The Journal of Open Source Software*, *3*(21), 433. http://dx.doi.org/10.21105/joss.00433.

Joseph, A., Nelson, B., Rubinstein, B., & Tygar, J. (2019). *Adversarial machine learning*. Cambridge University Press.

Juola, P. (2006). Authorship attribution. *Foundation and Trends in Information Retrieval*, *1*(3), 233–334. http://dx.doi.org/10.1561/1500000005.

Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V., & Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, *13*, 8–17. http://dx.doi.org/10.1016/j.csbj.2014.11.005.

Lee, K., Park, C., Kim, N., & Lee, J. (2018). Accelerating recurrent neural network language model based online speech recognition system. *2018 IEEE international conference on acoustics, speech and signal processing*, http://dx.doi.org/10.1109/icassp.2018.8462334.

Lin, Z., Shi, Y., & Xue, Z. (2018). IDSGAN: Generative adversarial networks for attack generation against intrusion detection. arXiv:1809.02077 [Cs].

Liu, Q., Li, P., Zhao, W., Cai, W., Yu, S., & Leung, V. C. M. (2018). A survey on security threats and defensive techniques of machine learning: A data driven view. *IEEE Access*, *6*, 12103–12117. http://dx.doi.org/10.1109/access.2018.2805680.

Lowd, D., & Meek, C. (2005). Adversarial learning. In *Proceeding of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining*. ACM Press, http://dx.doi.org/10.1145/1081870.1081950.

Mitchell, T. M. (1997). *Machine learning* (1st ed.). USA: McGraw-Hill, Inc.

Moosavi-Dezfooli, S.-M., Fawzi, A., & Frossard, P. (2016). DeepFool: A simple and accurate method to fool deep neural networks. In *2016 IEEE conference on computer vision and pattern recognition*. IEEE, http://dx.doi.org/10.1109/cvpr.2016.282.

Mosli, R., Wright, M., Yuan, B., & Pan, Y. (2019). They might NOT be giants: Crafting black-box adversarial examples with fewer queries using particle swarm optimization. arXiv:1909.07490.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 military communications and information systems conference* (pp. 1–6).

Moustafa, N., & Slay, J. (2016). The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective*, *25*(1–3), 18–31. http://dx.doi.org/10.1080/19393555.2015.1125974.

Nguyen, A., Yosinski, J., & Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE conference on computer vision and pattern recognition*. IEEE, http://dx.doi.org/10.1109/cvpr.2015.7298640.

Papernot, N., McDaniel, P., & Goodfellow, I. (2016). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. arXiv preprint arXiv:1605.07277.

Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. (2016). The limitations of deep learning in adversarial settings. *2016 IEEE European symposium on security and privacy*, http://dx.doi.org/10.1109/eurosp.2016.36.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2021). Scikit-learn: Machine learning in python. *Machine Learning in Python*, 6.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, *3*(1), 88–97. http://dx.doi.org/10.1162/neco.1991.3.1.88.

Serban, A. C., Poll, E., & Visser, J. (2018). Adversarial examples - A complete characterisation of the phenomenon.

Shah, R., Gaston, J., Harvey, M., McNamara, M., Ramos, O., You, Y., et al. (2019). Evaluating evasion attack methods on binary network traffic classifiers. In *Proceedings of the conference on information systems applied research ISSN (Vol. 2167)* (p. 1508).

Sheatsley, R. (2018). *Adversarial examples in constrained domains*. Pennsylvania State University, https://books.google.com/books?id=SV25wgEACAAJ.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Stańczyk, U., & Cyran, K. A. (2007). Machine learning approach to authorship attribution of literary texts. *International Journal of Applied Mathematics and Informatics*, *1*(4), 151–158.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., et al. (2013). Intriguing properties of neural networks. arXiv:1312.6199.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD cup 99 data set. In *Proceedings of the second IEEE international conference on computational intelligence for security and defense applications* (pp. 53–58). IEEE Press.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. arXiv:1706.03762.

Vorobeychik, Y., Kantarcioglu, M., Brachman, R., Stone, P., & Rossi, F. (2018). Adversarial machine learning. In *Synthesis lectures on artificial intelligence and machine learning*. Morgan & Claypool Publishers, https://books.google.com/books?id=Lw5pDwAAQBAJ.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, *4*(2), 65–85.

Wong, B. K., Bodnovich, T. A., & Selvi, Y. (1997). Neural network applications in business: A review and analysis of the literature (1988–1995). *Decision Support Systems*, *19*(4), 301–320. http://dx.doi.org/10.1016/S0167-9236(96)00070-X.

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., et al. (2016). Achieving human parity in conversational speech recognition. arXiv:1610.05256.

Xu, W., Qi, Y., & Evans, D. (2016). Automatically evading classifiers. In *Proceedings of the 2016 network and distributed systems symposium (Vol. 10)*.

Yang, C., Wu, Q., Li, H., & Chen, Y. (2017). Generative poisoning attack method against neural networks. arXiv:1703.01340 [Cs, Stat].

Zenati, H., Romain, M., Foo, C.-S., Lecouat, B., & Chandrasekhar, V. (2018). Adversarially learned anomaly detection. In *2018 IEEE international conference on data mining* (pp. 727–736). http://dx.doi.org/10.1109/ICDM.2018.00088.

Zhang, Z., Geiger, J., Pohjalainen, J., Mousa, A. E.-D., Jin, W., & Schuller, B. (2018). Deep learning for environmentally robust speech recognition. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *9*(5), 1–28. http://dx.doi.org/10.1145/3178115.